



TIBCO® Data Virtualization

Google Big Query Adapter Guide

Version 8.7.0 | October 2023

Contents

Contents	2
Google BigQuery Adapter	5
Google BigQuery Version Support	5
SQL Compliance	5
Getting Started	5
Connecting to Google BigQuery	5
Deploying the Google BigQuery Adapter	5
Basic Tab	6
Logging	12
Creating a Custom OAuth App	13
Advanced Integrations	15
Minimum Required Roles	17
Changelog	18
Advanced Features	26
User Defined Views	26
SSL Configuration	27
Firewall and Proxy	27
Query Processing	27
Logging	27
User Defined Views	27
SSL Configuration	30
Firewall and Proxy	30
Query Processing	31
Logging	32
SQL Compliance	35
SELECT Statements	35
INSERT Statements	36

UPDATE Statements	36
DELETE Statements	36
EXECUTE Statements	36
Names and Quoting	36
SELECT Statements	36
SELECT INTO Statements	118
INSERT Statements	118
UPDATE Statements	119
DELETE Statements	120
EXECUTE Statements	121
PIVOT and UNPIVOT	122
Data Model	123
Views	123
Stored Procedures	123
Additional Metadata	124
Tables	124
Views	125
Stored Procedures	128
Data Type Mapping	144
Connection String Options	147
Authentication	147
BigQuery	147
Storage API	148
Uploading	148
OAuth	149
JWT OAuth	150
SSL	150
Firewall	151
Proxy	151
Logging	152
Schema	152
Miscellaneous	152

Authentication	154
BigQuery	157
Storage API	163
Uploading	166
OAuth	170
JWT OAuth	177
SSL	184
Firewall	185
Proxy	189
Logging	195
Schema	196
Miscellaneous	202
Google BigQuery Adapter Limitations	214
TIBCO Product Documentation and Support Services	215
How to Access TIBCO Documentation	215
Release Version Support	216
How to Contact TIBCO Support	216
How to Join TIBCO Community	217
Legal and Third-Party Notices	218

Google BigQuery Adapter

Google BigQuery Version Support

The adapter enables read/write SQL-92 access to the BigQuery tables in your Google account or Google Apps domain. The complete aggregate and join syntax in BigQuery is supported. Additionally, statements in the BigQuery syntax can be passed through. The adapter uses version 2.0 of the BigQuery Web services API: You must enable this API by creating a project in the Google Developers Console. See [Connecting to Google](#) for a guide to creating a project and authenticating to this API.

SQL Compliance

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

Getting Started

Connecting to Google BigQuery

[Basic Tab](#) shows how to authenticate to Google BigQuery 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 Google BigQuery Adapter

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

1. Unzip the `tdv.googlebigquery.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.googlebigquery/tdv.googlebigquery.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 GoogleBigQuery
```

Basic Tab

Connecting to Google BigQuery

By default, the CData adapter connects to all available projects in your database. To limit the scope of your connection, set combinations of the following properties:

- ProjectId: specifies which projects the driver connects to
- BillingProjectId: specifies which projects are billed
- DatasetId: specifies which datasets the driver accesses

Authenticating to Google BigQuery

The adapter supports using user accounts, service accounts and GCP instance accounts for authentication.

The following sections discuss the available authentication schemes for Google BigQuery:

- User Accounts (OAuth)
- Service Account (OAuthJWT)
- GCP Instance Account

User Accounts (OAuth)

AuthScheme must be set to **OAuth** in all user account flows.

Web Applications

When connecting via a Web application, you need to create and register a custom OAuth application with Google BigQuery. You can then use the adapter to acquire and manage the OAuth token values. See [Creating a Custom OAuth App](#) for more information about custom applications.

Get an OAuth Access Token

Set the following connection properties to obtain the OAuthAccessToken:

- OAuthClientId: Set this to the Client Id in your application settings.
- OAuthClientSecret: Set this to the Client Secret in your application settings.

Then call stored procedures to complete the OAuth exchange:

1. Call the [GetOAuthAuthorizationURL](#) stored procedure. Set the CallbackURL input to the Callback URL you specified in your application settings. The stored procedure returns the URL to the OAuth endpoint.
2. Navigate to the URL that the stored procedure returned in Step 1. Log in to the custom OAuth application and authorize the web application. Once authenticated, the browser redirects you to the callback URL.
3. Call the [GetOAuthAccessToken](#) stored procedure. Set AuthMode to **WEB** and the Verifier input to the "code" parameter in the query string of the callback URL.

Once you have obtained the access and refresh tokens, you can connect to data and refresh the OAuth access token either automatically or manually.

Automatic Refresh of the OAuth Access Token

To have the driver automatically refresh the OAuth access token, set the following on the first data connection:

- InitiateOAuth: Set this to **REFRESH**.
- OAuthClientId: Set this to the Client Id in your application settings.

- OAuthClientSecret: Set this to the Client Secret in your application settings.
- OAuthAccessToken: Set this to the access token returned by [GetOAuthAccessToken](#).
- OAuthRefreshToken: Set this to the refresh token returned by [GetOAuthAccessToken](#).
- OAuthSettingsLocation: Set this to the path where the adapter saves the OAuth token values, which persist across connections.

On subsequent data connections, the values for OAuthAccessToken and OAuthRefreshToken are taken from OAuthSettingsLocation.

Manual Refresh of the OAuth Access Token

The only value needed to manually refresh the OAuth access token when connecting to data is the OAuth refresh token.

Use the [RefreshOAuthAccessToken](#) stored procedure to manually refresh the OAuthAccessToken after the ExpiresIn parameter value returned by [GetOAuthAccessToken](#) has elapsed, then set the following connection properties:

- OAuthClientId: Set this to the Client Id in your application settings.
- OAuthClientSecret: Set this to the Client Secret in your application settings.

Then call [RefreshOAuthAccessToken](#) with OAuthRefreshToken set to the OAuth refresh token returned by [GetOAuthAccessToken](#). After the new tokens have been retrieved, open a new connection by setting the OAuthAccessToken property to the value returned by [RefreshOAuthAccessToken](#).

Finally, store the OAuth refresh token so that you can use it to manually refresh the OAuth access token after it has expired.

Headless Machines

To configure the driver, use OAuth with a user account on a headless machine. You need to authenticate on another device that has an Internet browser.

1. Choose one of two options:

- Option 1: Obtain the OAuthVerifier value as described in "Obtain and Exchange a Verifier Code" below.
- Option 2: Install the adapter on a machine with an Internet browser and transfer the OAuth authentication values after you authenticate through the

usual browser-based flow.

2. Then configure the adapter to automatically refresh the access token on the headless machine.

Option 1: Obtain and Exchange a Verifier Code

To obtain a verifier code, you must authenticate at the OAuth authorization URL.

Follow the steps below to authenticate from the machine with an Internet browser and obtain the OAuthVerifier connection property.

1. Choose one of these options:

- If you are using the Embedded OAuth Application click [Google BigQuery OAuth endpoint](#) to open the endpoint in your browser.
- If you are using a custom OAuth application, create the Authorization URL by setting the following properties:
 - InitiateOAuth: Set to **OFF**.
 - OAuthClientId: Set to the client Id assigned when you registered your application.
 - OAuthClientSecret: Set to the client secret assigned when you registered your application.

Then call the [GetOAuthAuthorizationURL](#) stored procedure with the appropriate CallbackURL. Open the URL returned by the stored procedure in a browser.

2. Log in and grant permissions to the adapter. You are then redirected to the callback URL, which contains the verifier code.
3. Save the value of the verifier code. Later you will set this in the OAuthVerifier connection property.

Next, you need to exchange the OAuth verifier code for OAuth refresh and access tokens. Set the following properties:

On the headless machine, set the following connection properties to obtain the OAuth authentication values:

- InitiateOAuth: Set this to **REFRESH**.
- OAuthVerifier: Set this to the verifier code.
- OAuthClientId: (custom applications only) Set this to the Client Id in your custom

OAuth application settings.

- OAuthClientSecret: (custom applications only) Set this to the Client Secret in the custom OAuth application settings.
- OAuthSettingsLocation: Set this to persist the encrypted OAuth authentication values to the specified file.

After the OAuth settings file is generated, you need to re-set the following properties to connect:

- InitiateOAuth: Set this to **REFRESH**.
- OAuthClientId: (custom applications only) Set this to the client Id assigned when you registered your application.
- OAuthClientSecret: (custom applications only) Set this to the client secret assigned when you registered your application.
- OAuthSettingsLocation: Set this to the file containing the encrypted OAuth authentication values. Make sure this file gives read and write permissions to the adapter to enable the automatic refreshing of the access token.

Option 2: Transfer OAuth Settings

Prior to connecting on a headless machine, you need to create and install a connection with the driver on a device that supports an Internet browser. Set the connection properties as described in "Desktop Applications" above.

After completing the instructions in "Desktop Applications", the resulting authentication values are encrypted and written to the path specified by OAuthSettingsLocation. The default filename is *OAuthSettings.txt*.

Once you have successfully tested the connection, copy the OAuth settings file to your headless machine.

On the headless machine, set the following connection properties to connect to data:

- InitiateOAuth: Set this to **REFRESH**.
- OAuthClientId: (custom applications only) Set this to the client Id assigned when you registered your application.
- OAuthClientSecret: (custom applications only) Set this to the client secret assigned when you registered your application.
- OAuthSettingsLocation: Set this to the path to your OAuth settings file. Make sure this

file gives read and write permissions to the adapter to enable the automatic refreshing of the access token.

Service Accounts (OAuthJWT)

To authenticate using a service account, you must create a new service account and have a copy of the accounts certificate. If you do not already have a service account, you can create one by following the procedure in [Creating a Custom OAuth App](#).

For a JSON file, set these properties:

- AuthScheme: Set this to **OAuthJWT**.
- InitiateOAuth: Set this to **GETANDREFRESH**.
- OAuthJWTCertType: Set this to **GOOGLEJSON**.
- OAuthJWTCert: Set this to the path to the *.json* file provided by Google.
- OAuthJWTSubject: (optional) Only set this value if the service account is part of a GSuite domain and you want to enable delegation. The value of this property should be the email address of the user whose data you want to access.

For a PFX file, set these properties instead:

- AuthScheme: Set this to **OAuthJWT**.
- InitiateOAuth: Set this to **GETANDREFRESH**.
- OAuthJWTCertType: Set this to **PFXFILE**.
- OAuthJWTCert: Set this to the path to the *.pfx* file provided by Google.
- OAuthJWTCertPassword: (optional) Set this to the *.pfx* file password. In most cases you must provide this since Google encrypts PFX certificates.
- OAuthJWTCertSubject: (optional) Set this only if you are using a OAuthJWTCertType which stores multiple certificates. Should not be set for PFX certificates generated by Google.
- OAuthJWTIssuer: Set this to the email address of the service account. This address will usually include the domain **iam.gserviceaccount.com**.
- OAuthJWTSubject: (optional) Only set this value if the service account is part of a GSuite domain and you want to enable delegation. The value of this property should be the email address of the user whose data you want to access.

GCP Instance Accounts

When running on a GCP virtual machine, the adapter can authenticate using a service account tied to the virtual machine. To use this mode, set AuthScheme to **GCPInstanceAccount**.

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 Google BigQuery 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_dsrc.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**.

Creating a Custom OAuth App

You can use a custom OAuth app to authenticate a service account or a user account.

When to Create a Custom OAuth App

Web Applications

You need to create a custom OAuth app in the web flow.

Desktop Applications

Creating a custom OAuth app is optional as the driver is already registered with Instagram and you can connect with its embedded credentials. You might want to create a custom OAuth app to change the information displayed when users log into the Instagram OAuth endpoint to grant permissions to the driver.

Headless Machines

Creating a custom OAuth app is optional to authenticate a headless machine; the driver is already registered with Instagram and you can connect with its embedded credentials. In the headless OAuth flow, users need to authenticate via a browser on another machine. You might want to create a custom OAuth app to change the information displayed when users log into the Instagram OAuth endpoint to grant permissions to the driver.

Create an OAuth App for User Account Authentication

Follow the procedure below to register an app and obtain the OAuthClientId and OAuthClientSecret.

Create a Custom OAuth App: Desktop

1. Log into the Google API Console and open a project. Select the API Manager from the main menu.
2. In the user consent flow, click **Credentials > Create Credentials > OAuth Client Id**. Then click **Other**.
3. After creating the app, the OAuthClientId and OAuthClientSecret are displayed.
4. Click **Library > BigQuery API > Enable API**.

Create an OAuth App for Service Account Authentication

Follow the steps below to create an OAuth application and generate a private key.

1. Log into the Google API Console and open a project. Select the API Manager from the main menu.
2. Click **Create Credentials > Service Account Key**.
3. In the Service Account menu, select **New Service Account** or select an existing service account.
4. If you are creating a new service account, additionally select one or more roles. You can assign primitive roles at the project level in the IAM and Admin section; other roles enable you to further customize access to Google APIs.
5. In the **Key Type** section, select the JSON key type.
6. Create the app to download the key file.
7. Click **Library > BigQuery API > Enable API**.

Advanced Integrations

The following sections detail adapter settings that may be needed in advanced integrations.

Saving Result Sets

Large result sets must be saved in a temporary or permanent table. You can use the following properties to control table persistence:

Accessing Temporary Tables

You can use the following properties to manage temporary tables. These temporary tables are managed by Google BigQuery and automatically expires after 24 hours.

- AllowLargeResultSets: Store large result sets in temporary tables in a hidden dataset. If this is not set, an error is returned for result sets bigger than a certain size.

Accessing Persistent Tables

Persist result sets larger than 128MB in a permanent table. To do so, you can set `DestinationTable` to the qualified name of the table. For example, `"DestinationProjectId:DestinationDataSet.TableName"`. Subsequent queries replace the table with the new result set.

Limiting Billing

Set `MaximumBillingTier` to override your project limits on the maximum cost for any given query in a connection.

Bulk Modes

Google BigQuery provides several interfaces for operating on batches of rows. The adapter supports these methods through the `InsertMode` option, each of which are specialized to different use cases:

- The **Streaming** API is intended for use where the most important factor is being able to insert quickly. However, rows which are inserted via the API are queued and only appear in the table after a delay. Sometimes this delay can be as high as 20-30 minutes which makes this API incompatible with cases where you want to insert data and then run other operations on it immediately. You should avoid modifying the table while any rows are in the streaming queue: Google BigQuery prevents DML operations from running on the table while any rows are in the streaming queue, and changing the table's metadata (name, schema, etc.) may cause streamed rows that haven't been committed to be lost.
- The **DML** mode API uses Standard SQL INSERT queries to upload data. This is by the most robust method of uploading data because any errors in the uploaded rows will be reported immediately. The adapter also uses this API in a synchronous way so once the INSERT is processed, any rows can be used by other operations without waiting. However, it is by far the slowest insert method and should only be used for small data volumes.
- The **Upload** mode uses the multipart upload API for uploading data. This method is intended for performing low-cost medium to large data loads within a reasonable time. When using this mode the adapter will upload the inserted rows to public

temporary storage within BigQuery and then create a load job for them. This job will execute and the adapter can either wait for it (see [WaitForBatchResults](#)) or let it run asynchronously. Waiting for the job will report any errors that the job encounters but will take more time. Determining if the job failed without waiting for it requires manually checking the job status via the job stored procedures.

- The **GCSStaging** mode is the same as Upload except that it uses your Google Cloud Storage account to store staged data instead of BigQuery public storage. The adapter cannot act asynchronously in this mode because it must delete the file after the load is complete, which means that [WaitForBatchResults](#) has no effect. Because this depends on external data, you must set the [GCSBucket](#) to the name of your bucket and ensure that [Scope](#) (this is a space delimited set of scopes) contains at least the scopes <https://www.googleapis.com/auth/bigquery> and https://www.googleapis.com/auth/devstorage.read_write. The devstorage scope used for GCS also requires that you connect using a service account because Google BigQuery does not allow user accounts to use this scope.

In addition to bulk inserts, the adapter also supports performing bulk UPDATE and DELETE operations. This requires the adapter to upload the data containing the filters and rows to set into a new table in BigQuery, then perform a MERGE between the two tables and drop the temporary table. [InsertMode](#) determines how the rows are inserted into the temporary table but the Streaming and DML modes are not supported.

In order to perform bulk UPDATE specifically, you must define a primary key on the relevant tables. The simplest way to do this is via the [PrimaryKeyIdentifiers](#) property. Columns that are used for UPDATE filters must be marked as primary keys, while all other columns may be modified by bulk UPDATE.

Minimum Required Roles

Minimum Required Roles for Service Accounts

The following roles allow SELECT queries to work with a service account:

- BigQuery Data Viewer (*roles/bigquery.dataViewer*): read data and metadata
- BigQuery Filtered Data Viewer (*roles/bigquery.filteredDataViewer*): view filtered table data
- BigQuery Job User (*roles/bigquery.jobUser*): run jobs, including queries, within the project

Changelog

General Changes

Date	Build Number	Change Type	Description
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.
06/06/2022	8192	Google BigQuery	Added <ul style="list-style-type: none"> Added support for MERGE statements. The syntax is a subset of what BigQuery natively supports, including everything except the MATCHED BY SOURCE, DEFAULT and WHEN MATCHED AND clauses.
04/27/2022	8152	Google BigQuery	Added <ul style="list-style-type: none"> Added support for the SAFE_CAST function. Due to syntax limitations, it must be written this way when passthrough mode is disabled (the default): <div data-bbox="893 1503 1346 1535" data-label="Text"> <pre>SAFE_CAST(somecol, 'INTEGER')</pre> </div> . It supports SQL type names (VARCHAR, INT, etc.) as well as BigQuery type names (STRING, INT64, etc.)
04/11/2022	8136	Google BigQuery	Changed <ul style="list-style-type: none"> Changed what views are retrieved

			depending on the connection property UseLegacySQL. Connections using standard SQL will not report views that are only available in legacy SQL, and the same goes for standard views in legacy SQL connections.
03/01/2022	8095	Google BigQuery	Added <ul style="list-style-type: none"> Added support for CREATE VIEW AS statements.
02/18/2022	8084	Google BigQuery	Added <ul style="list-style-type: none"> Added support for ALTER TABLE. Due to BigQuery limitations, the only supported operations are ADD COLUMN, DROP COLUMN, ALTER COLUMN and (table) RENAME TO.
01/26/2022	8061	Google BigQuery	Added <ul style="list-style-type: none"> Added support for querying snapshot tables. Append it to the AllowedTableTypes connection property (along with other allowed table types) to make them show up.
01/13/2022	8048	Google BigQuery	Added <ul style="list-style-type: none"> Added support for the NATIVEQUERY table function. This function can be used after a FROM to execute a query using BigQuery-native SQL. For example, <div data-bbox="893 1528 1378 1627" data-label="Text"> <pre>SELECT * FROM NATIVEQUERY ('SELECT * FROM UNNEST([1,2,3]) AS a')</pre> </div> will execute the inner query in BigQuery directly and return the results. This will work even in tools which are not normally

compatible with QueryPassthrough=true.			
01/04/2022	8039	Google BigQuery	Added <ul style="list-style-type: none"> Added support for querying materialized views.
12/20/2021	8024	Google BigQuery	Added <ul style="list-style-type: none"> Added support for using the BigQuery TABLESAMPLE without enabling QueryPassthrough. If TableSamplePercent is set then the random row selection happens server-side for each table used in a query.
12/10/2021	8014	Google BigQuery	Changed <ul style="list-style-type: none"> Migrate to the latest version of the Storage API.
10/21/2021	7964	Google BigQuery	Added <ul style="list-style-type: none"> Added support for reading policy tags from the Data Catalog service. These are exposed using the PolicyTags column on sys_tablecolumns.
10/20/2021	7963	Google BigQuery	Added <ul style="list-style-type: none"> Added support for parameterized data types, both on the read side with and without UseStorageAPI as well as on the metadata side. The metadata is only reported on the table itself and not in queries, since BigQuery only enforces length/precision/scale when the data is written.
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.
09/02/2021	7915	Google BigQuery	Added <ul style="list-style-type: none"> Added support for bulk UPDATE and DELETE operations. These make better use of BigQuery's DML query limits because each batch is just one MERGE operation. They are also faster because they use the bulk upload methods to send the list of rows to be changed. Primary keys must be defined on the tables to use DELETE.
08/09/2021	7891	Google BigQuery	Added <ul style="list-style-type: none"> Support for staging data in all formats allowed by BigQuery. JSON and Avro were already supported, this adds options that allow uploading data stored in the CSV, Parquet and ORC formats.
08/07/2021	7889	General	Changed <ul style="list-style-type: none"> Add the KeySeq column to the sys_foreignkeys table.
08/06/2021	7888	General	Changed <ul style="list-style-type: none"> Add the new sys_primarykeys system table.
07/28/2021	7879	Google BigQuery	Added <ul style="list-style-type: none"> Added support for reading TIME, DATETIME and TIMESTAMP data at the full precision supported by BigQuery. We now report microsecond precision in .NET-based editions and millisecond precision in Java-based editions.
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: <code>SELECT LAST_MONTH() AS lm, Col FROM Table WHERE Col > LAST_MONTH()</code>. Formerly the two <code>LAST_MONTH()</code> 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: <code>WHERE col > L_LAST_MONTH()</code>. 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.
07/20/2021	7871	Google BigQuery	Added <ul style="list-style-type: none"> Added support for staging data with Google Cloud Storage when performing inserts. This is similar to the existing Upload insert mode, but transfers the row data into GCS and then creates a load job referencing it instead of performing the upload directly into BigQuery.
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.
06/18/2021	7839	Google	Added

		BigQuery	<ul style="list-style-type: none"> Added support for the GOOGLEJSONBLOB JWT certificate type. This works like the existing GOOGLEJSON certificate type except that the certificate is provided as JSON text instead of as a file path.
05/27/2021	7817	Google BigQuery	Added <ul style="list-style-type: none"> Added support for automatically reconnecting on Storage connections that have long idle times. This is useful most for ETL applications and other scenarios where the consumer reading data out of the driver is expected to be slower than the driver itself. In these cases the connection can be dropped by the other side (either by the Storage API itself or network appliances like firewalls and proxies) leading to timeouts. To avoid this the driver will restart the read from the same position using a new connection.
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. • 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
04/05/2021	7765	Google BigQuery	Added <ul style="list-style-type: none"> • Added support for data unnesting. This mode performs client-side expansion of array data across multiple rows in a way that emulates the compact preview in the BigQuery UI. Server-side support of some options is limited in this mode (LIMIT and OFFSET) but otherwise behaves like the nested mode with the exception that it always outputs flat data and never outputs aggregates.
Changed			

- Updated the default value for WaitForBatchResults to true since not checking the job status at smaller batch sizes can lead to lost jobs.

Deprecated

- Deprecated the UseStreamingInserts connection property in favor of InsertMode. The default for InsertMode is Streaming.

03/04/2021	7733	Google BigQuery	Removed <ul style="list-style-type: none"> • Removed the TempTableDataset and TempTableExpirationTime connection properties. These properties have been defunct since we moved to using the BigQuery-managed query cache for large resultsets.
01/29/2021	7699	Google BigQuery	Added <ul style="list-style-type: none"> • Added support for the BIGNUMERIC type, which is twice the size of the existing numeric type. JDBC supports this type at full precision while .NET requires using the connection property IgnoreTypes=decimal because the full precision of BIGNUMERIC (as with regular NUMERIC) is too high for System.Decimal.
01/08/2021	7678	Google BigQuery	Added <ul style="list-style-type: none"> • Added support for using the upload API for batch inserts. This is an API that is slower than the streaming API for large volumes of data, but is free to use and doesn't have the same buffering delays that streaming does. It is also async by default although the option to wait on the batch is available via the new

WaitForBatchResults connection property.			
12/23/2020	7663	Google BigQuery	Added <ul style="list-style-type: none"> Added support for unnesting metadata with the Storage API.
12/21/2020	7660	Google BigQuery	Added <ul style="list-style-type: none"> Added support for unnesting metadata with the REST API. By enabling UnnestArrays, metadata for fields within REPEATED RECORD types is returned as separate columns instead of generating JSON aggregates. These values can be selected within queries but are currently treated as NULL.
12/07/2020	7646	Google BigQuery	Added <ul style="list-style-type: none"> Added support for injecting partition filters into queries that require them but do not have them, if the user enables the InsertPartitionFilterOption. This modifies queries against partitioned tables that require a partition filter so that they always contain a valid partition filter, which allows queries like simple SELECT * FROM t to work. Currently all partitions are selected by the generated filter.

Advanced Features

This section details a selection of advanced features of the Google BigQuery 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 Google BigQuery and then processes the rest of the query in memory (client-side).

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 Google BigQuery 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 [publicdata].[samples].github_nested WHERE
MyColumn = 'value'"
  },
  "MyView2": {
    "query": "SELECT * FROM MyTable WHERE Id IN (1,2,3)"
  }
}
```

Use the UserDefinedViews connectio 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 Google BigQuery 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 Google BigQuery Query Evaluation component examines SQL queries and returns information indicating what parts of the query the adapter is not capable of executing natively.

The Google BigQuery 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 Verbosity 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 Verbosity to 2 will log everything included in Verbosity 1 and additional information about the request. |
|---|--|
-
- | | |
|---|---|
| 3 | Setting Verbosity to 3 will additionally log HTTP headers, as well as the body of the request and the response. |
|---|---|
-
- | | |
|---|---|
| 4 | Setting Verbosity to 4 will additionally log transport-level communication with the data source. This includes SSL negotiation. |
|---|---|
-
- | | |
|---|---|
| 5 | Setting Verbosity 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 verbositys, 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 Google BigQuery 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 Google BigQuery API.

INSERT Statements

See [INSERT Statements](#) for a syntax reference and examples, as well as retrieving the new records' Ids.

UPDATE Statements

The primary key Id is required to update a record. See [UPDATE Statements](#) for a syntax reference and examples.

DELETE Statements

The primary key Id is required to delete a record. See [DELETE 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

Google BigQuery API Syntax

The Google BigQuery API offers additional SQL operators and functions. A complete list of the available syntax is located at: <https://cloud.google.com/bigquery/query-reference>

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 Google BigQuery 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 | NOT LIKE |
IN | NOT IN | IS NULL | IS NOT NULL | AND | OR | + | - | * | / | % | ||
  } [ <expression> ]
  } [ { AND | OR } ... ]

```

Examples

1. Return all columns:

```
SELECT * FROM [publicdata].[samples].github_nested
```

2. Rename a column:

```
SELECT "repository.name" AS MY_repository.name FROM [publicdata].
[samples].github_nested
```

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

```
SELECT CAST(repository.watchers AS VARCHAR) AS Str_
repository.watchers FROM [publicdata].[samples].github_nested
```

4. Search data:

```
SELECT * FROM [publicdata].[samples].github_nested WHERE
repository.name = 'EntityFramework'
```

5. The Google BigQuery APIs support the following operators in the WHERE clause: =, >, <, >=, <=, <>, !=, LIKE, NOT LIKE, IN, NOT IN, IS NULL, IS NOT NULL, AND, OR, +, -, *, /, %, ||.

```
SELECT * FROM [publicdata].[samples].github_nested WHERE
repository.name = 'EntityFramework';
```

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

```
SELECT COUNT(*) AS MyCount FROM [publicdata].[samples].github_
nested
```

7. Return the unique items matching the query criteria:

```
SELECT DISTINCT repository.name FROM [publicdata].[samples].github_
nested
```

8. Summarize data:

```
SELECT repository.name, MAX(repository.watchers) FROM [publicdata].
[samples].github_nested GROUP BY repository.name
```

See [Aggregate Functions](#) for details.

9. Retrieve data from multiple tables.

```
SELECT * FROM CRMAccounts INNER JOIN ERPCustomers ON
CRMAccounts.BillingState = ERPCustomers.BillingState
```

See [JOIN Queries](#) for details.

- Sort a result set in ascending order:

```
SELECT actor.attributes.email, repository.name FROM [publicdata].
[samples].github_nested ORDER BY repository.name ASC
```

Aggregate Functions

Google BigQuery API Syntax

The Google BigQuery API offers additional SQL operators and functions. A complete list of the available syntax is located at: <https://cloud.google.com/bigquery/query-reference>

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 [publicdata].[samples].github_nested WHERE
repository.name = 'EntityFramework'
```

AVG

Returns the average of the column values.


```
SELECT repository.name, AVG(repository.watchers) FROM [publicdata].
[samples].github_nested WHERE repository.name = 'EntityFramework' GROUP
BY repository.name
```

MIN

Returns the minimum column value.

```
SELECT MIN(repository.watchers), repository.name FROM [publicdata].
[samples].github_nested WHERE repository.name = 'EntityFramework' GROUP
BY repository.name
```

MAX

Returns the maximum column value.

```
SELECT repository.name, MAX(repository.watchers) FROM [publicdata].
[samples].github_nested WHERE repository.name = 'EntityFramework' GROUP
BY repository.name
```

SUM

Returns the total sum of the column values.

```
SELECT SUM(repository.watchers) FROM [publicdata].[samples].github_
nested WHERE repository.name = 'EntityFramework'
```

CORR

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

```
SELECT repository.name, CORR(repository.watchers, repository.size) FROM
[publicdata].[samples].github_nested
```

COVAR_POP

Computes the population covariance of the values computed by a set of number pairs.

```
SELECT repository.name, COVAR_POP(repository.watchers, repository.size)
FROM [publicdata].[samples].github_nested
```

COVAR_SAMP

Computes the sample covariance of the values computed by a set of number pairs.

```
SELECT repository.name, COVAR_SAMP(repository.watchers, repository.size)
FROM [publicdata].[samples].github_nested
```

NTH

Returns the nth sequential value in the scope of the function, where n is a constant. The NTH function starts counting at 1, so there is no zeroth term. If the scope of the function has less than n values, the function returns NULL.

```
SELECT repository.name, NTH(n, actor.attributes.email) FROM
[publicdata].[samples].github_nested
```

STDDEV

Returns the standard deviation of the computed values. Rows with a NULL value are not included in the calculation.

```
SELECT repository.name, STDDEV(repository.watchers) FROM [publicdata].
[samples].github_nested
```

JOIN Queries

The adapter supports the complete join syntax in Google BigQuery. Google BigQuery supports inner joins, outer joins, and cross joins. The default is inner. Multiple join operations are supported.

```
SELECT field_1 [..., field_n] FROM
  table_1 [[AS] alias_1]
  [[INNER|FULL|RIGHT|LEFT] OUTER|CROSS] JOIN [EACH]
  table_2 [[AS] alias_2]
  [ON join_condition_1 [... AND join_condition_n]]
```

```
] +
```

Note that the default join is an inner join. The following limitations exist on joins in Google BigQuery:

- Cross joins must not contain an ON clause.
- Normal joins require that the right-side table must contain less than 8 MB of compressed data. If you are working with tables larger than 8 MB, use the EACH modifier. Note that EACH cannot be used in cross joins.

Projection Functions

ANY_VALUE(expression)

Returns any value from the input or NULL if there are zero input rows. The value returned is non-deterministic, which means you might receive a different result each time you use this function.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** The expression to retrieve a value from.

APPROX_COUNT_DISTINCT(expression)

Returns the approximate result for COUNT(DISTINCT expression). The value returned is a statistical estimate-not necessarily the actual value. This function is less accurate than COUNT(DISTINCT expression), but performs better on huge input.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** The expression to perform the approximate count distinct on.

APPROX_QUANTILES(expression, number)

Returns the approximate boundaries for a group of expression values, where number represents the number of quantiles to create. This function returns an array of number + 1 elements, where the first element is the approximate minimum and the last element is the

approximate maximum.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** The expression to perform the approximate quantiles on.
- **number:** The number of quantiles to create.

APPROX_TOP_COUNT(expression, number)

Returns the approximate top elements of expression. The number parameter specifies the number of elements returned.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** The expression to perform the approximate top count on.
- **number:** The number of elements to be returned.

APPROX_TOP_SUM(expression, weight, number)

Returns the approximate top elements of expression, based on the sum of an assigned weight. The number parameter specifies the number of elements returned. If the weight input is negative or NaN, this function returns an error.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** The expression to perform the approximate top sum on.
- **weight:** The assigned weight.
- **number:** The number of elements to be returned.

ARRAY(subquery)

The ARRAY function returns an ARRAY with one element for each row in a subquery.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **subquery:** The subquery to execute.

ARRAY_CONCAT(array_expr1 [, array_expr2 [, ...]])

Concatenates one or more arrays with the same element type into a single array.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **array_expr1:** The first array to concatenate.
- **array_expr2:** The second array to concatenate.

ARRAY_LENGTH(array_expr)

Returns the size of the array. Returns 0 for an empty array. Returns NULL if the array_expression is NULL.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **array_expr:** The array expression to get the size of.

ARRAY_TO_STRING(array_expr, delimiter [, null_text])

Returns a concatenation of the elements in array_expression as a STRING. The value for array_expression can either be an array of STRING or BYTES data types.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **array_expr:** The array expression to convert to string.
- **delimiter:** The delimiter string used to delimit the array values.
- **null_text:** If the null_text parameter is used, the function replaces any NULL values in the array with the value of null_text. If the null_text parameter is not used, the function omits the NULL value and its preceding delimiter.

GENERATE_ARRAY(start_expr, end_expr [, step_expr])

Returns an array of values. The start_expression and end_expression parameters determine the inclusive start and end of the array.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **start_expr:** The starting value.

- **end_expr**: The ending value.
- **step_expr**: The increment used to generate array values.

GENERATE_DATE_ARRAY(start_date, end_date [, INTERVAL int_expr date_part])

Returns an array of dates. The start_date and end_date parameters determine the inclusive start and end of the array.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **start_date**: The starting date.
- **end_date**: The ending date.
- **int_expr**: The increment used to generate dates.
- **date_part**: The date part used to increment the generated dates. Valid values are: DAY, WEEK, MONTH, QUARTER, and YEAR.

ARRAY_REVERSE(array_expr)

Returns the input ARRAY with elements in reverse order.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **array_expr**: The array to reverse.

ARRAY_AGG(expression)

Returns an ARRAY of expression values.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression**: The expression values to generate an array from.

ARRAY_CONCAT_AGG(expression1[, expression2 [,...]])

Concatenates elements from expression of type ARRAY, returning a single ARRAY as a result. This function ignores NULL input arrays, but respects the NULL elements in non-NULL input arrays (an error is raised, however, if an array in the final query result contains

a NULL element).

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression1:** The first expression to concatenate.
- **expression2:** The first expression to concatenate.

AVG([DISTINCT] expression)

Returns the average on non-null values. Each distinct value of expression is aggregated only once into the result.

- **expression:** The expression to use to compute the average.

BIT_AND(numeric_expression)

Returns the result of a bitwise AND operation between each instance of numeric_expr across all rows. NULL values are ignored. This function returns NULL if all instances of numeric_expr evaluate to NULL.

- **numeric_expression:** The numeric expression to perform the bitwise operation.

BIT_COUNT(expression)

The input, expression, must be an integer or BYTES. Returns the number of bits that are set in the input expression. For integers, this is the number of bits in two's complement form.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** The expression to perform the bit count operation on.

BIT_OR(numeric_expression)

Returns the result of a bitwise OR operation between each instance of numeric_expr across all rows. NULL values are ignored. This function returns NULL if all instances of numeric_expr evaluate to NULL.

- **numeric_expression:** The numeric expression to perform the bitwise operation.

BIT_XOR(numeric_expression)

Returns the result of a bitwise XOR operation between each instance of numeric_expr across all rows. NULL values are ignored. This function returns NULL if all instances of numeric_expr evaluate to NULL.

- **numeric_expression:** The numeric expression to perform the bitwise operation.

CORR(numeric_expression1, numeric_expression2)

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

- **numeric_expression1:** The first series.
- **numeric_expression2:** The second series.

COUNTIF(expression)

Returns the count of TRUE values for expression. Returns 0 if there are zero input rows or expression evaluates to FALSE for all rows.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** The expression to evaluate.

COVAR_POP(numeric_expression1, numeric_expression2)

Computes the population covariance of the values computed by numeric_expression1 and numeric_expression2.

- **numeric_expression:** The first series.
- **numeric_expression:** The second series.

COVAR_SAMP(numeric_expression1, numeric_expression2)

Computes the sample covariance of the values computed by numeric_expression1 and numeric_expression2.

- **numeric_expression:** The first series.
- **numeric_expression:** The second series.

FIRST(column)

Returns the first sequential value in the scope of the function.

Note: this function is only available when UseLegacySQL=True.

- **column:** Any column expression.

FIRST_VALUE(value_expression [(IGNORE/RESPECT) NULLS])

Returns the value of the value_expression for the first row in the current window frame.

Note: this function only supports [IGNORE NULLS] when using Standard SQL (UseLegacySQL=False).

- **value_expression:** Any value expression

GROUP_CONCAT(string_expression [, separator])

Concatenates multiple strings into a single string, where each value is separated by the optional separator parameter. If separator is omitted, returns a comma-separated string.

Note: this function is only available when UseLegacySQL=True.

- **string_expression:** The string expression to concat.
- **separator:** The separator.

GROUP_CONCAT_UNQUOTED(string_expression [, separator])

Concatenates multiple strings into a single string, where each value is separated by the optional separator parameter. If separator is omitted, BigQuery returns a comma-separated string. Unlike GROUP_CONCAT, this function will not add double quotes to returned values that include a double quote character. For example, the string a"b would return as a"b.

Note: this function is only available when UseLegacySQL=True.

- **string_expression**: The string expression to concat.
- **separator**: The separator.

LAST(column)

Returns the last sequential value in the scope of the function.

Note: this function is only available when UseLegacySQL=True.

- **column**: Any column expression

LAST_VALUE(value_expression [(IGNORE/RESPECT) NULLS])

Returns the value of the value_expression for the last row in the current window frame.

Note: this function only supports [IGNORE NULLS] when using Standard SQL (UseLegacySQL=False).

- **value_expression**: Any value expression

LOGICAL_AND(expression)

Returns the logical AND of all non-NULL expressions. Returns NULL if there are zero input rows or expression evaluates to NULL for all rows.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression**: The expression to perform the logical AND on.

LOGICAL_OR(expression)

Returns the logical OR of all non-NULL expressions. Returns NULL if there are zero input rows or expression evaluates to NULL for all rows.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression**: The expression to perform the logical OR on.

NEST(expression)

Aggregates all values in the current aggregation scope into a repeated field. For example, the query `SELECT x, NEST(y) FROM ... GROUP BY x` returns one output record for each distinct `x` value, and contains a repeated field for all `y` values paired with `x` in the query input. The `NEST` function requires a `GROUP BY` clause.

Note: this function is only available when `UseLegacySQL=True`.

- **expression:** Any expression.

NOW()

Returns the current UNIX timestamp in microseconds.

Note: this function is only available when `UseLegacySQL=True`.

NTH(n, field)

Returns the `n`th sequential value in the scope of the function, where `n` is a constant. The `NTH` function starts counting at 1, so there is no zeroth term. If the scope of the function has less than `n` values, the function returns `NULL`.

Note: this function is only available when `UseLegacySQL=True`.

- **n:** The `n`th sequential value.
- **field:** The column name.

NTH_VALUE(value_expression, constant_integer_expression)

Returns the value of `value_expression` at the `Nth` row of the current window frame, where `Nth` is defined by `constant_integer_expression`. Returns `NULL` if there is no such row.

Note: this function is only available when using Standard SQL (`UseLegacySQL=False`).

- **value_expression:** Any value expression.
- **constant_integer_expression:** The `n`th sequential value.

QUANTILES(expression [, buckets])

Computes approximate minimum, maximum, and quantiles for the input expression. NULL input values are ignored. Empty or exclusively-NULL input results in NULL output. The number of quantiles computed is controlled with the optional buckets parameter, which includes the minimum and maximum in the count.

Note: this function is only available when UseLegacySQL=True.

- **expression:** The numeric expression to compute quantiles on.
- **buckets:** The number of buckets.

STDDEV(numeric_expression)

Returns the standard deviation of the values computed by numeric_expr. Rows with a NULL value are not included in the calculation.

- **numeric_expression:** The series to calculate STDDEV on.

STDDEV_POP(numeric_expression)

Computes the population standard deviation of the value computed by numeric_expr.

- **numeric_expression:** The series to calculate STDDEV on.

STDDEV_SAMP([DISTINCT] numeric_expression)

Computes the sample standard deviation of the value computed by numeric_expr.

- **numeric_expression:** The series to calculate STDDEV on.

STRING_AGG(expression[, delimiter])

Returns a value (either STRING or BYTES) obtained by concatenating non-null values. If a delimiter is specified, concatenated values are separated by that delimiter; otherwise, a comma is used as a delimiter.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** The string expression to concatenate.
- **delimiter:** The delimiter to separate concatenated values.

SUM([DISTINCT] expression)

Returns the sum on non-null values. Each distinct value of expression is aggregated only once into the result.

- **expression:** The expression to use to compute the sum.

TOP(column [, max_values][, multiplier])

TOP is a function that is an alternative to the GROUP BY clause. It is used as simplified syntax for GROUP BY ... ORDER BY ... LIMIT Generally, the TOP function performs faster than the full ... GROUP BY ... ORDER BY ... LIMIT ... query, but may only return approximate results.

Note: this function is only available when UseLegacySQL=True.

- **numeric_expression:** The series to calculate STDDEV on.
- **max_values:** The maximum number of results to return. Default is 20.
- **multiplier:** A positive integer that increases the value(s) returned by COUNT(*) by the multiple specified.

UNIQUE(expression)

Returns the set of unique, non-NULL values in the scope of the function in an undefined order.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any expression.

VARIANCE(numeric_expression)

Computes the variance of the values computed by numeric_expr. Rows with a NULL value are not included in the calculation.

- **numeric_expression:** The series to calculate VARIANCE on.

VAR_POP(numeric_expression)

Computes the population variance of the values computed by numeric_expr.

- **numeric_expression:** The series to calculate VARIANCE on.

VAR_SAMP([DISTINCT] numeric_expression)

Computes the sample variance of the values computed by numeric_expr.

- **numeric_expression:** The series to calculate VARIANCE on.

RANK()

Returns the ordinal (1-based) rank of each row within the ordered partition. All peer rows receive the same rank value. The next row or set of peer rows receives a rank value which increments by the number of peers with the previous rank value, instead of a rank value which always increments by 1.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

PERCENT_RANK()

Return the percentile rank of a row defined as $(RK-1)/(NR-1)$, where RK is the RANK of the row and NR is the number of rows in the partition. Returns 0 if NR=1.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

NTILE(constant_integer_expression)

This function divides the rows into constant_integer_expression buckets based on row ordering and returns the 1-based bucket number that is assigned to each row. The number of rows in the buckets can differ by at most 1. The remainder values (the remainder of number of rows divided by buckets) are distributed one for each bucket, starting with bucket 1. If constant_integer_expression evaluates to NULL, 0 or negative, an error is

provided.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **constant_integer_expression:** The number of buckets to divide the rows into.

LEAD(value_expression [, offset [, default_expression]])

Returns the value of the value_expression on a subsequent row. Changing the offset value changes which subsequent row is returned; the default value is 1, indicating the next row in the window frame. An error occurs if offset is NULL or a negative value.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **value_expression:** The value expression.
- **offset:** The offset to use. Must be a non-negative integer.
- **default_expression:** The default expression. Must be compatible with the value_expression type.

LAG(value_expression [, offset [, default_expression]])

Returns the value of the value_expression on a subsequent row. Changing the offset value changes which subsequent row is returned; the default value is 1, indicating the next row in the window frame. An error occurs if offset is NULL or a negative value.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **value_expression:** The value expression.
- **offset:** The offset to use. Must be a non-negative integer.
- **default_expression:** The default expression. Must be compatible with the value_expression type.

PERCENTILE_CONT(value_expression [, percentile [{RESPECT | IGNORE} NULLS]])

Computes the specified percentile value for the value_expression, with linear interpolation.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **value_expression:** A numeric expression.
- **percentile:** A literal in the range [0, 1].

PERCENTILE_DISC(value_expression, percentile [{RESPECT | IGNORE} NULLS])

Computes the specified percentile value for a discrete value_expression. The returned value is the first sorted value of value_expression with cumulative distribution greater than or equal to the given percentile value.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **value_expression:** Any orderable type.
- **percentile:** A literal in the range [0, 1].

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

Returns the value of the first non-null expression. The remaining expressions are not evaluated. All input expressions must be implicitly coercible to a common supertype.

Note: this function currently accepts up to 9 expressions.

- **expr1:** Any expression
- **expr2:** Any expression

NULLIF(expression, expression_to_match)

Returns NULL if expression = expression_to_match is true, otherwise returns expression. expression and expression_to_match must be implicitly coercible to a common supertype; equality comparison is done on coerced values.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** Any expression
- **expression_to_match:** Any expression to be matched

CUME_DIST()

Return the relative rank of a row defined as NP/NR. NP is defined to be the number of rows that either precede or are peers with the current row. NR is the number of rows in the partition.

Note: this function returns a double when using Legacy SQL (UseLegacySQL=True).

DENSE_RANK()

Returns the ordinal (1-based) rank of each row within the window partition. All peer rows receive the same rank value, and the subsequent rank value is incremented by one.

ROW_NUMBER()

Does not require the ORDER BY clause. Returns the sequential row ordinal (1-based) of each row for each ordered partition. If the ORDER BY clause is unspecified then the result is non-deterministic.

IFNULL(expr, null_result)

If expr is NULL, return null_result. Otherwise, return expr. If expr is not NULL, null_result is not evaluated. expr and null_result must be implicitly coercible to a common supertype. Synonym for COALESCE(expr, null_result)

- **expr:** Any expression
- **null_result:** The result to return if expr is null

CAST(expression AS type)

Cast is used in a query to indicate that the result type of an expression should be converted to some other type.

- **expression:** The expression to cast.
- **type:** The type to cast the expression to.

SAFE_CAST(expression, type)

Cast is used in a query to indicate that the result type of an expression should be converted to some other type. SAFE_CAST is identical to CAST, except it returns NULL instead of raising an error.

- **expression:** The expression to cast.
- **type:** The type to cast the expression to.

CURRENT_DATE()

Returns a human-readable string of the current date in the format %Y-%m-%d.

DATE(timestamp [, timezone])

Converts a timestamp_expression to a DATE data type.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **timestamp:** The timestamp from which to return the date.
- **timezone:** The timezone to use when converting the timestamp. If not specified, the default timezone, UTC, is used.

DATEDIFF(timestamp1, timestamp2)

Returns the number of days between two TIMESTAMP data types. The result is positive if the first TIMESTAMP data type comes after the second TIMESTAMP data type, and otherwise the result is negative.

Note: this function is only available when UseLegacySQL=True.

- **timestamp1:** The first timestamp.
- **timestamp2:** The second timestamp.

DATE_DIFF(date1, date2, date_part)

Computes the number of specified date_part differences between two date expressions. This can be thought of as the number of date_part boundaries crossed between the two dates. If the first date occurs before the second date, then the result is negative.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **date1:** The first date.
- **date2:** The second date.
- **date_part:** The date part. Supported values are: DAY, MONTH, QUARTER, YEAR.

DATE_TRUNC(date, date_part)

Truncates the date to the specified granularity.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **date:** The date to truncate.
- **date_part:** The date part. Supported values are: DAY, WEEK, ISOWEEK, MONTH, QUARTER, YEAR, ISOYEAR.

FORMAT_DATE(format_string, date_expr)

Formats the date_expr according to the specified format_string.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **format_string:** The format string used to format the date_expr.
- **date_expr:** The date to format.

PARSE_DATE(format_string, date_string)

Uses a format_string and a string representation of a date to return a DATE object.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **format_string:** The format string used to parse the date_string.
- **date_string:** The date string to parse.

CURRENT_DATETIME([timezone])

Returns the current time as a DATETIME object.

- **timezone:** The timezone to use when retrieving the current datetime object.

DATETIME(timestamp [, timezone])

Constructs a DATETIME object using a TIMESTAMP object.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **timestamp:** The timestamp from which to return the datetime.
- **timezone:** The timezone to use when converting the timestamp. If not specified, the default timezone, UTC, is used.

DATETIME_DIFF(datetime1, datetime2, date_part)

Computes the number of specified date_part differences between two date expressions. This can be thought of as the number of date_part boundaries crossed between the two dates. If the first date occurs before the second date, then the result is negative.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **datetime1:** The first datetime.
- **datetime2:** The second datetime.
- **date_part:** The date part. Possible values include: MICROSECOND, MILLISECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, and YEAR.

DATETIME_TRUNC(datetime, part)

Truncates the datetime to the specified granularity.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **date:** The datetime to truncate.
- **part:** The date part. Possible values include: MICROSECOND, MILLISECOND, SECOND, MINUTE, HOUR, DAY, WEEK, ISOWEEK, MONTH, QUARTER, YEAR, and ISOYEAR.

FORMAT_DATETIME(format_string, datetime_expr)

Formats the date_expr according to the specified format_string.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **format_string:** The format string used to format the date_expr.
- **datetime_expr:** The datetime to format.

PARSE_DATETIME(format_string, datetime_string)

Uses a format_string and a string representation of a date to return a DATETIME object.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **format_string:** The format string used to parse the date_string.
- **datetime_string:** The datetime string to parse.

CURRENT_TIME()

Returns a human-readable string of the server's current time in the format %H:%M:%S.

TIME(timestamp [, timezone])

Constructs a DATETIME object using a TIMESTAMP object.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **timestamp:** The timestamp from which to return the datetime.
- **timezone:** The timezone to use when converting the timestamp. If not specified, the default timezone, UTC, is used.

TIME_DIFF(time1, time2, time_part)

Computes the number of specified time_part differences between two time expressions.

This can be thought of as the number of time_part boundaries crossed between the two times. If the first time occurs before the second time, then the result is negative.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **time1:** The first time.
- **time2:** The second time.
- **time_part:** The time part. Possible values include: MICROSECOND, MILLISECOND, SECOND, MINUTE, HOUR.

TIME_TRUNC(time, part)

Truncates the time to the specified granularity.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **time:** The time to truncate.
- **part:** The time part. Possible values include: MICROSECOND, MILLISECOND, SECOND, MINUTE, HOUR.

FORMAT_TIME(format_string, time_expr)

Formats the date_expr according to the specified format_string.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **format_string:** The format string used to format the date_expr.
- **time_expr:** The time to format.

PARSE_TIME(format_string, time_string)

Uses a format_string and a string representation of a time to return a TIME object.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **format_string:** The format string used to parse the time_string.
- **time_string:** The time string to parse.

CURRENT_TIMESTAMP()

Returns a TIMESTAMP data type of the server's current time in the format %Y-%m-%d %H:%M:%S.

TIMESTAMP_DIFF(timestamp1, timestamp2, time_part)

Computes the number of specified time_part differences between two timestamp expressions. This can be thought of as the number of time_part boundaries crossed between the two timestamp. If the first timestamp occurs before the second timestamp, then the result is negative.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **timestamp1:** The first timestamp.
- **timestamp2:** The second timestamp.
- **time_part:** The timestamp part. Possible values include: MICROSECOND, MILLISECOND, SECOND, MINUTE, HOUR.

FORMAT_TIMESTAMP(format_string, timestamp_expr)

Formats the date_expr according to the specified format_string.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **format_string:** The format string used to format the date_expr.
- **timestamp_expr:** The timestamp to format.

PARSE_TIMESTAMP(format_string, timestamp_string)

Uses a format_string and a string representation of a timestamp to return a TIMESTAMP object.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **format_string:** The format string used to parse the timestamp_string.
- **timestamp_string:** The timestamp string to parse.

DAY(timestamp)

Returns the day of the month of a TIMESTAMP data type as an integer between 1 and 31, inclusively.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the day of the month.

DAYOFWEEK(timestamp)

Returns the day of the week of a TIMESTAMP data type as an integer between 1 (Sunday) and 7 (Saturday), inclusively.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the day of the week.

DAYOFYEAR(timestamp)

Returns the day of the year of a TIMESTAMP data type as an integer between 1 and 366, inclusively. The integer 1 refers to January 1.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the day of the year.

FORMAT_UTC_USEC(unix_timestamp)

Returns a human-readable string representation of a UNIX timestamp in the format YYYY-MM-DD HH:MM:SS.ffffff.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The unix timestamp to format.

HOUR(timestamp)

Returns the hour of a TIMESTAMP data type as an integer between 0 and 23, inclusively.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the hour as an integer.

MINUTE(timestamp)

Returns the minutes of a TIMESTAMP data type as an integer between 0 and 59, inclusively.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the minutes as an integer.

MONTH(timestamp)

Returns the month of a TIMESTAMP data type as an integer between 1 and 12, inclusively.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the month as an integer.

MSEC_TO_TIMESTAMP(unix_timestamp)

Converts a UNIX timestamp in milliseconds to a TIMESTAMP data type.

Note: this function is only available when UseLegacySQL=True.

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

PARSE_UTC_USEC(date_string)

Converts a date string to a UNIX timestamp in microseconds. date_string must have the format YYYY-MM-DD HH:MM:SS[.uuuuuu]. The fractional part of the second can be up to 6 digits long or can be omitted.

Note: this function is only available when UseLegacySQL=True.

- **date_string:** The date string to convert.

QUARTER(timestamp)

Returns the quarter of the year of a `TIMESTAMP` data type as an integer between 1 and 4, inclusively.

Note: this function is only available when `UseLegacySQL=True`.

- **timestamp:** The timestamp from which to return the quarter as an integer.

SEC_TO_TIMESTAMP(unix_timestamp)

Converts a UNIX timestamp in seconds to a `TIMESTAMP` data type.

Note: this function is only available when `UseLegacySQL=True`.

- **unix_timestamp:** The unix timestamp to convert.

SECOND(timestamp)

Returns the seconds of a `TIMESTAMP` data type as an integer between 0 and 59, inclusively. During a leap second, the integer range is between 0 and 60, inclusively.

Note: this function is only available when `UseLegacySQL=True`.

- **timestamp:** The timestamp from which to return the second as an integer.

STRFTIME_UTC_USEC(unix_timestamp, date_format_str)

Returns a human-readable date string in the format `date_format_str`. `date_format_str` can include date-related punctuation characters (such as / and -) and special characters accepted by the `strftime` function in C++ (such as %d for day of month).

Note: this function is only available when `UseLegacySQL=True`.

- **unix_timestamp:** The unix timestamp to convert.
- **date_format_str:** The date format string.

TIMESTAMP_SECONDS(unix_timestamp)

Interprets INT64_expression as the number of seconds since 1970-01-01 00:00:00 UTC.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

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

TIMESTAMP_MILLIS(unix_timestamp)

Interprets INT64_expression as the number of milliseconds since 1970-01-01 00:00:00 UTC.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

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

TIMESTAMP_MICROS(unix_timestamp)

Interprets INT64_expression as the number of microseconds since 1970-01-01 00:00:00 UTC.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

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

TIMESTAMP_TO_MSEC(timestamp)

Converts a TIMESTAMP data type to a UNIX timestamp in milliseconds.

Note: this function is only available when UseLegacySQL=True.

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

TIMESTAMP_TO_SEC(timestamp)

Converts a TIMESTAMP data type to a UNIX timestamp in seconds.

Note: this function is only available when UseLegacySQL=True.

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

TIMESTAMP_TO_USEC(timestamp)

Converts a TIMESTAMP data type to a UNIX timestamp in microseconds.

Note: this function is only available when UseLegacySQL=True.

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

UNIX_DATE(date_string)

Returns the number of days since 1970-01-01.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **date_string:** The date string to convert.

UNIX_SECONDS(timestamp)

Returns the number of seconds since 1970-01-01 00:00:00 UTC. Truncates higher levels of precision.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

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

UNIX_MILLIS(timestamp)

Returns the number of milliseconds since 1970-01-01 00:00:00 UTC. Truncates higher levels of precision.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

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

UNIX_MICROS(timestamp)

Returns the number of microseconds since 1970-01-01 00:00:00 UTC. Truncates higher levels of precision.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

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

USEC_TO_TIMESTAMP(unix_timestamp)

Converts a UNIX timestamp in microseconds to a TIMESTAMP data type.

Note: this function is only available when UseLegacySQL=True.

- **unix_timestamp:** The unix timestamp to convert.

UTC_USEC_TO_DAY(unix_timestamp)

Shifts a UNIX timestamp in microseconds to the beginning of the day it occurs in.

Note: this function is only available when UseLegacySQL=True.

- **unix_timestamp:** The unix timestamp to shift.

UTC_USEC_TO_HOUR(unix_timestamp)

Shifts a UNIX timestamp in microseconds to the beginning of the hour it occurs in.

Note: this function is only available when UseLegacySQL=True.

- **unix_timestamp:** The unix timestamp to shift.

UTC_USEC_TO_MONTH(unix_timestamp)

Shifts a UNIX timestamp in microseconds to the beginning of the month it occurs in.

Note: this function is only available when UseLegacySQL=True.

- **unix_timestamp:** The unix timestamp to shift.

UTC_USEC_TO_WEEK(unix_timestamp, day_of_week)

Returns a UNIX timestamp in microseconds that represents a day in the week of the unix_timestamp argument.

Note: this function is only available when UseLegacySQL=True.

- **unix_timestamp:** The unix timestamp to shift.
- **day_of_week:** A day of the week from 0 (Sunday) to 6 (Saturday).

UTC_USEC_TO_YEAR(unix_timestamp)

Returns a UNIX timestamp in microseconds that represents the year of the `unix_timestamp` argument.

Note: this function is only available when UseLegacySQL=True.

- **unix_timestamp:** The unix timestamp to convert.

WEEK(timestamp)

Returns the week of a `TIMESTAMP` data type as an integer between 1 and 53, inclusively. Weeks begin on Sunday, so if January 1 is on a day other than Sunday, week 1 has fewer than 7 days and the first Sunday of the year is the first day of week 2.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the week as an integer.

YEAR(timestamp)

Returns the year of a `TIMESTAMP` data type.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the year as an integer.

ABS(expression)

Returns the absolute value of the argument.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any column or literal expression.

ACOS(expression)

Returns the arc cosine of the argument.

- **expression:** Any column or literal expression.

ACOSH(expression)

Returns the arc hyperbolic cosine of the argument.

- **expression:** Any column or literal expression.

ASIN(expression)

Returns arcsine in radians.

- **expression:** Any column or literal expression.

ASINH(expression)

Returns the arc hyperbolic sine of the argument.

- **expression:** Any column or literal expression.

ATAN(expression)

Returns arc tangent of the argument.

- **expression:** Any column or literal expression.

ATANH(expression)

Returns the arc hyperbolic tangent of the argument.

- **expression:** Any column or literal expression.

ATAN2(expression1, expression2)

Returns the arc tangent of the two arguments.

- **expression1**: Any column or literal expression.
- **expression2**: Any column or literal expression.

CEIL(expression)

Rounds the argument up to the nearest whole number and returns the rounded value.

- **expression**: Any column or literal expression.

CEILING(expression)

Synonym for CEIL function.

- **expression**: Any column or literal expression.

COS(expression)

Returns the cosine of the argument.

- **expression**: Any column or literal expression.

COSH(expression)

Returns the hyperbolic cosine of the argument.

- **expression**: Any column or literal expression.

DEGREES(expression)

Returns expression, converted from radians to degrees.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any column or literal expression.

EXP(expression)

Returns the result of raising the constant "e" - the base of the natural logarithm - to the power of expression.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any column or literal expression.

FLOOR(expression)

Rounds the argument down to the nearest whole number and returns the rounded value.

- **expression:** Any column or literal expression.

LN(expression)

Returns the natural logarithm of the argument.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any column or literal expression.

LOG(expression)

Returns the natural logarithm of the argument.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any column or literal expression.

LOG2(expression)

Returns the Base-2 logarithm of the argument.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any column or literal expression.

LOG10(expression)

Returns the Base-10 logarithm of the argument.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any column or literal expression.

PI()

Returns PI.

Note: this function is only available when UseLegacySQL=True.

POW(expression1, expression2)

Returns the result of raising expression1 to the power of expression2.

- **expression1:** Any column or literal expression.
- **expression2:** Any column or literal expression.

POWER(expression1, expression2)

Synonym of POW function.

- **expression1:** Any column or literal expression.
- **expression2:** Any column or literal expression.

RADIANS(expression)

Returns expression, converted from degrees to radians.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any column or literal expression.

RAND([expression])

Returns a random float value in the range $0.0 \leq \text{value} < 1.0$. Each `int32_seed` value always generates the same sequence of random numbers within a given query, as long as you don't use a `LIMIT` clause. If `int32_seed` is not specified, BigQuery uses the current timestamp as the seed value.

Note: this function is only available when `UseLegacySQL=True`.

- **expression:** Any column or literal expression.

ROUND(expression [, integer_digits])

Rounds the argument either up or down to the nearest whole number (or if specified, to the specified number of digits) and returns the rounded value.

- **expression:** Any column or literal expression.
- **integer_digits:** The number of digits to round to.

GREATEST(value1[, value2 [, ...]])

Returns `NULL` if any of the inputs is `NULL`. Otherwise, returns `NaN` if any of the inputs is `NaN`. Otherwise, returns the largest value among X_1, \dots, X_N according to the `<` comparison.

Note: this function is only available when using Standard SQL (`UseLegacySQL=False`).

- **value1:** The first value to compare.
- **value2:** The second value to compare.

LEAST(value1[, value2 [, ...]])

Returns `NULL` if any of the inputs is `NULL`. Otherwise, returns `NaN` if any of the inputs is `NaN`. Otherwise, returns the smallest value among X_1, \dots, X_N according to the `>` comparison.

Note: this function is only available when using Standard SQL (`UseLegacySQL=False`).

- **value1:** The first value to compare.
- **value2:** The second value to compare.

SIN(expression)

Returns the sine of the argument.

- **expression:** Any column or literal expression.

SINH(expression)

Returns the hyperbolic sine of the argument.

- **expression:** Any column or literal expression.

SQRT(expression)

Returns the square root of the expression.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any column or literal expression.

TAN(expression)

Returns the tangent of the argument.

- **expression:** Any column or literal expression.

TANH(expression)

Returns the hyperbolic tangent of the argument.

- **expression:** Any column or literal expression.

TRUNC(expression [, integer_digits])

Rounds X to the nearest integer whose absolute value is not greater than Xs. When the integer_digits parameter is specified this function is similar to ROUND(X, N) but always rounds towards zero. Unlike ROUND(X, N) it never overflows.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** Any column or literal expression.
- **integer_digits:** The number of digits to round to.

BYTE_LENGTH(str)

Returns the length of the value in bytes, regardless of whether the type of the value is STRING or BYTES.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **str:** The string to calculate the length on.

CHAR_LENGTH(str)

Returns the length of the STRING in characters.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **str:** The string to calculate the length on.

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

Returns the concatenation of two or more strings, or NULL if any of the values are NULL.

- **str1:** The first string to concatenate.
- **str2:** The second string to concatenate.
- **str3:** The third string to concatenate.

ENDS_WITH(str1, str2)

Takes two values. Returns TRUE if the second value is a suffix of the first.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **str1**: The string to search in.
- **str2**: The string to search for.

FROM_BASE64(string_expr)

Converts the base64-encoded input string_expr into BYTES format. To convert BYTES to a base64-encoded STRING, use TO_BASE64.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **string_expr**: The string to convert from base64 encoding.

FROM_HEX(string_expr)

Converts a hexadecimal-encoded STRING into BYTES format. Returns an error if the input STRING contains characters outside the range (0..9, A..F, a..f). The lettercase of the characters does not matter. To convert BYTES to a hexadecimal-encoded STRING, use TO_HEX.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **string_expr**: The string to convert from hexadecimal encoding.

INSTR(str1, str2)

Returns the one-based index of the first occurrence of str2 in str1, or returns 0 if str2 does not occur in str1.

Note: this function is only available when UseLegacySQL=True.

- **str1**: The string to search in.
- **str2**: The string to search for.

LEFT(str, numeric_expression)

Returns the leftmost numeric_expr characters of str. If the number is longer than str, the full string will be returned. Example: LEFT('seattle', 3) returns sea.

Note: this function is only available when UseLegacySQL=True.

- **str:** The string to perform the LEFT operation on.
- **numeric_expression:** The number of characters to return.

LENGTH(str)

Returns a numerical value for the length of the string. Example: if str is '123456', LENGTH returns 6.

- **str:** The string to calculate the length on.

LOWER(str)

Returns the original string with all characters in lower case.

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

LPAD(str1, numeric_expression[, str2])

Pads str1 on the left with str2, repeating str2 until the result string is exactly numeric_expr characters. Example: LPAD('1', 7, '?') returns ??????1.

- **str1:** The string to pad.
- **numeric_expression:** The number of str2 instances to pad.
- **str2:** The pad characters.

LTRIM(str1 [, str2])

Removes characters from the left side of str1. If str2 is omitted, LTRIM removes spaces from the left side of str1. Otherwise, LTRIM removes any characters in str2 from the left side of

str1 (case-sensitive).

- **str1:** The string to trim.
- **str2:** The characters to trim from str1.

REPEAT(str, repetitions)

Returns a value that consists of original_value, repeated. The repetitions parameter specifies the number of times to repeat original_value. Returns NULL if either original_value or repetitions are NULL. This function return an error if the repetitions value is negative.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **str:** The string to repeat.
- **str2:** The number of repetitions.

REPLACE(original_value, from_value, to_value)

Replaces all instances of from_value within original_value with to_value.

- **original_value:** The string to search in.
- **from_value:** The string to search for.
- **to_value:** The string to replace instances of from_value.

REVERSE(str)

Returns the reverse of the input STRING or BYTES.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

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

RIGHT(str, numeric_expression)

Returns the rightmost numeric_expr characters of str. If the number is longer than the string, it will return the whole string. Example: RIGHT('kirkland', 4) returns land.

Note: this function is only available when UseLegacySQL=True.

- **str:** The string to perform the RIGHT operation on.
- **numeric_expression:** The number of characters to return.

RPAD(str1, numeric_expression, str2)

Pads str1 on the right with str2, repeating str2 until the result string is exactly numeric_expr characters. Example: RPAD('1', 7, '?') returns 1?????.

- **str1:** The string to pad.
- **numeric_expression:** The number of str2 instances to pad.
- **str2:** The pad characters.

RTRIM(str1 [, str2])

Removes trailing characters from the right side of str1. If str2 is omitted, RTRIM removes trailing spaces from str1. Otherwise, RTRIM removes any characters in str2 from the right side of str1 (case-sensitive).

- **str1:** The string to trim.
- **str2:** The characters to trim from str1.

SPLIT(str [, delimiter])

Splits a string into repeated substrings. If delimiter is specified, the SPLIT function breaks str into substrings, using delimiter as the delimiter.

- **str:** The string to split.
- **delimiter:** The delimiter to split the string on. Default delimiter is a comma (,).

STARTS_WITH(str1, str2)

Takes two values. Returns TRUE if the second value is a prefix of the first.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **str1:** The string to search in.
- **str2:** The string to search for.

STRPOS(str1, str2)

Returns the 1-based index of the first occurrence of value2 inside value1. Returns 0 if value2 is not found.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **str1:** The string to search in.
- **str2:** The string to search for.

SUBSTR(str, index [, max_len])

Returns a substring of str, starting at index. If the optional max_len parameter is used, the returned string is a maximum of max_len characters long. Counting starts at 1, so the first character in the string is in position 1 (not zero). If index is 5, the substring begins with the 5th character from the left in str. If index is -4, the substring begins with the 4th character from the right in str. Example: SUBSTR('awesome', -4, 4) returns the substring some.

- **str:** The original string.
- **index:** The starting index.
- **max_len:** The maximum length of the return string.

TO_BASE64(string_expr)

Converts a sequence of BYTES into a base64-encoded STRING. To convert a base64-encoded STRING into BYTES, use FROM_BASE64.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **string_expr**: The string to convert to base64 encoding.

TO_HEX(string_expr)

Converts a sequence of BYTES into a hexadecimal STRING. Converts each byte in the STRING as two hexadecimal characters in the range (0..9, a..f). To convert a hexadecimal-encoded STRING to BYTES, use FROM_HEX.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **string_expr**: The string to convert to hexadecimal encoding.

TRIM(str1 [, str2])

Removes all leading and trailing characters that match value2. If value2 is not specified, all leading and trailing whitespace characters (as defined by the Unicode standard) are removed. If the first argument is of type BYTES, the second argument is required. If value2 contains more than one character or byte, the function removes all leading or trailing characters or bytes contained in value2.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **str1**: The string to trim.
- **str2**: The optional string characters to trim from str1.

UPPER(str)

Returns the original string with all characters in upper case.

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

JSON_EXTRACT(json, json_path)

Selects a value in json according to the JSONPath expression json_path. json_path must be a string constant. Returns the value in JSON string format.

- **json**: The JSON to select a value from.

- **json_path:** The JSON path of the value contained in json.

JSON_EXTRACT_SCALAR(json, json_path)

Selects a value in json according to the JSONPath expression json_path. json_path must be a string constant, and bracket notation is not supported. Returns a scalar JSON value.

- **json:** The JSON to select a value from.
- **json_path:** The JSON path of the value contained in json.

REGEXP_CONTAINS(str, reg_exp)

Returns TRUE if value is a partial match for the regular expression, regex. You can search for a full match by using ^ (beginning of text) and \$ (end of text). If the regex argument is invalid, the function returns an error.

Note: this function is only available when UseLegacySQL=True.

- **str:** The string to match in the regular expression.
- **reg_exp:** The regular expression to match.

REGEXP_EXTRACT(str, reg_exp)

Returns the portion of str that matches the capturing group within the regular expression.

- **str:** The string to match in the regular expression.
- **reg_exp:** The regular expression to match.

REGEXP_EXTRACT_ALL(str, reg_exp)

Returns an array of all substrings of value that match the regular expression, regex. The REGEXP_EXTRACT_ALL function only returns non-overlapping matches. For example, using this function to extract ana from banana returns only one substring, not two.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **str:** The string to match in the regular expression.

- **reg_exp**: The regular expression to match.

REGEXP_REPLACE(orig_str, reg_exp, replace_str)

Returns a string where any substring of orig_str that matches reg_exp is replaced with replace_str. For example, REGEXP_REPLACE ('Hello', 'lo', 'p') returns Help.

- **orig_str**: The original string to match in the regular expression.
- **reg_exp**: The regular expression to match.
- **replace_str**: The replacement for the matched orig_str in the regular expression.

FORMAT_IP(integer_value)

Converts 32 least significant bits of integer_value to human-readable IPv4 address string.

Note: this function is only available when UseLegacySQL=True.

- **integer_value**: The integer value to convert to an IPv4 address.

PARSE_IP(readable_ip)

Converts a string representing IPv4 address to unsigned integer value. For example, PARSE_IP('0.0.0.1') will return 1. If string is not a valid IPv4 address, PARSE_IP will return NULL.

Note: this function is only available when UseLegacySQL=True.

- **readable_ip**: The IPv4 address to convert to an integer.

NET.IPV4_FROM_INT64(integer_value)

Converts an IPv4 address from integer format to binary (BYTES) format in network byte order. In the integer input, the least significant bit of the IP address is stored in the least significant bit of the integer, regardless of host or client architecture. For example, 1 means 0.0.0.1, and 0x1FF means 0.0.1.255.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **integer_value:** The integer value to convert to an IPv4 address.

NET.IPV4_TO_INT64(readable_ip)

Converts an IPv4 address from binary (BYTES) format in network byte order to integer format. In the integer output, the least significant bit of the IP address is stored in the least significant bit of the integer, regardless of host or client architecture. For example, 1 means 0.0.0.1, and 0x1FF means 0.0.1.255. The output is in the range [0, 0xFFFFFFFF]. If the input length is not 4, this function throws an error. This function does not support IPv6.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **readable_ip:** The IPv4 address to convert to an integer.

FARM_FINGERPRINT(expression)

Computes the fingerprint of the STRING or BYTES input using the Fingerprint64 function from the open-source FarmHash library. The output of this function for a particular input will never change.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** The expression to use to compute the fingerprint.

MD5(expression)

Computes the hash of the input using the MD5 algorithm. The input can either be STRING or BYTES. The string version treats the input as an array of bytes. This function returns 16 bytes.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** The expression to use to compute the hash.

SHA1(expression)

Computes the hash of the input using the SHA-1 algorithm. The input can either be STRING or BYTES. The string version treats the input as an array of bytes. This function returns 20 bytes.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** The expression to use to compute the hash.

SHA256(expression)

Computes the hash of the input using the SHA-256 algorithm. The input can either be STRING or BYTES. The string version treats the input as an array of bytes. This function returns 32 bytes.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** The expression to use to compute the hash.

SHA512(expression)

Computes the hash of the input using the SHA-512 algorithm. The input can either be STRING or BYTES. The string version treats the input as an array of bytes. This function returns 64 bytes.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** The expression to use to compute the hash.

TIMESTAMP(datetime_expression[, timezone])

Convert a date, datetime, or string to a TIMESTAMP data type.

Note: this function does not support the timezone parameter and requires datetime_expression to be a string when using Legacy SQL (UseLegacySQL=True).

- **datetime_expression:** The expression to be converted to a timestamp
- **timezone:** The timezone to be used. If no timezone is specified, the default timezone, UTC, is used

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.

Predicate Functions

REGEXP_MATCH(str, reg_exp)

Returns true if str matches the regular expression. For string matching without regular expressions, use CONTAINS instead of REGEXP_MATCH.

Note: this function is only available when UseLegacySQL=True.

- **str:** The string to match in the regular expression.
- **reg_exp:** The regular expression to match.

CAST(expression AS type)

Cast is used in a query to indicate that the result type of an expression should be converted to some other type.

- **expression:** The expression to cast.
- **type:** The type to cast the expression to.

SAFE_CAST(expression, type)

Cast is used in a query to indicate that the result type of an expression should be converted to some other type. SAFE_CAST is identical to CAST, except it returns NULL instead of raising an error.

- **expression:** The expression to cast.
- **type:** The type to cast the expression to.

CURRENT_DATE()

Returns a human-readable string of the current date in the format %Y-%m-%d.

DATE(timestamp [, timezone])

Converts a timestamp_expression to a DATE data type.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **timestamp:** The timestamp from which to return the date.
- **timezone:** The timezone to use when converting the timestamp. If not specified, the default timezone, UTC, is used.

DATEDIFF(timestamp1, timestamp2)

Returns the number of days between two TIMESTAMP data types. The result is positive if the first TIMESTAMP data type comes after the second TIMESTAMP data type, and otherwise the result is negative.

Note: this function is only available when UseLegacySQL=True.

- **timestamp1:** The first timestamp.
- **timestamp2:** The second timestamp.

DATE_DIFF(date1, date2, date_part)

Computes the number of specified date_part differences between two date expressions. This can be thought of as the number of date_part boundaries crossed between the two dates. If the first date occurs before the second date, then the result is negative.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **date1:** The first date.
- **date2:** The second date.
- **date_part:** The date part. Supported values are: DAY, MONTH, QUARTER, YEAR.

DATE_TRUNC(date, date_part)

Truncates the date to the specified granularity.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **date:** The date to truncate.
- **date_part:** The date part. Supported values are: DAY, WEEK, ISOWEEK, MONTH, QUARTER, YEAR, ISOYEAR.

FORMAT_DATE(format_string, date_expr)

Formats the date_expr according to the specified format_string.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **format_string:** The format string used to format the date_expr.
- **date_expr:** The date to format.

PARSE_DATE(format_string, date_string)

Uses a format_string and a string representation of a date to return a DATE object.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **format_string:** The format string used to parse the date_string.
- **date_string:** The date string to parse.

CURRENT_DATETIME([timezone])

Returns the current time as a DATETIME object.

- **timezone:** The timezone to use when retrieving the current datetime object.

DATETIME(timestamp [, timezone])

Constructs a DATETIME object using a TIMESTAMP object.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **timestamp:** The timestamp from which to return the datetime.
- **timezone:** The timezone to use when converting the timestamp. If not specified, the default timezone, UTC, is used.

DATETIME_DIFF(datetime1, datetime2, date_part)

Computes the number of specified date_part differences between two date expressions. This can be thought of as the number of date_part boundaries crossed between the two dates. If the first date occurs before the second date, then the result is negative.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **datetime1:** The first datetime.
- **datetime2:** The second datetime.
- **date_part:** The date part. Possible values include: MICROSECOND, MILLISECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, and YEAR.

DATETIME_TRUNC(datetime, part)

Truncates the datetime to the specified granularity.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **date:** The datetime to truncate.
- **part:** The date part. Possible values include: MICROSECOND, MILLISECOND, SECOND, MINUTE, HOUR, DAY, WEEK, ISOWEEK, MONTH, QUARTER, YEAR, and ISOYEAR.

FORMAT_DATETIME(format_string, datetime_expr)

Formats the date_expr according to the specified format_string.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **format_string:** The format string used to format the date_expr.
- **datetime_expr:** The datetime to format.

PARSE_DATETIME(format_string, datetime_string)

Uses a format_string and a string representation of a date to return a DATETIME object.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **format_string:** The format string used to parse the date_string.
- **datetime_string:** The datetime string to parse.

CURRENT_TIME()

Returns a human-readable string of the server's current time in the format %H:%M:%S.

TIME(timestamp [, timezone])

Constructs a DATETIME object using a TIMESTAMP object.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **timestamp:** The timestamp from which to return the datetime.
- **timezone:** The timezone to use when converting the timestamp. If not specified, the default timezone, UTC, is used.

TIME_DIFF(time1, time2, time_part)

Computes the number of specified time_part differences between two time expressions. This can be thought of as the number of time_part boundaries crossed between the two times. If the first time occurs before the second time, then the result is negative.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **time1:** The first time.
- **time2:** The second time.
- **time_part:** The time part. Possible values include: MICROSECOND, MILLISECOND, SECOND, MINUTE, HOUR.

TIME_TRUNC(time, part)

Truncates the time to the specified granularity.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **time:** The time to truncate.
- **part:** The time part. Possible values include: MICROSECOND, MILLISECOND, SECOND, MINUTE, HOUR.

FORMAT_TIME(format_string, time_expr)

Formats the date_expr according to the specified format_string.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **format_string:** The format string used to format the date_expr.
- **time_expr:** The time to format.

PARSE_TIME(format_string, time_string)

Uses a format_string and a string representation of a time to return a TIME object.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **format_string:** The format string used to parse the time_string.
- **time_string:** The time string to parse.

CURRENT_TIMESTAMP()

Returns a TIMESTAMP data type of the server's current time in the format %Y-%m-%d %H:%M:%S.

TIMESTAMP_DIFF(timestamp1, timestamp2, time_part)

Computes the number of specified time_part differences between two timestamp expressions. This can be thought of as the number of time_part boundaries crossed between the two timestamp. If the first timestamp occurs before the second timestamp,

then the result is negative.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **timestamp1:** The first timestamp.
- **timestamp2:** The second timestamp.
- **time_part:** The timestamp part. Possible values include: MICROSECOND, MILLISECOND, SECOND, MINUTE, HOUR.

FORMAT_TIMESTAMP(format_string, timestamp_expr)

Formats the date_expr according to the specified format_string.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **format_string:** The format string used to format the date_expr.
- **timestamp_expr:** The timestamp to format.

PARSE_TIMESTAMP(format_string, timestamp_string)

Uses a format_string and a string representation of a timestamp to return a TIMESTAMP object.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **format_string:** The format string used to parse the timestamp_string.
- **timestamp_string:** The timestamp string to parse.

DAY(timestamp)

Returns the day of the month of a TIMESTAMP data type as an integer between 1 and 31, inclusively.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the day of the month.

DAYOFWEEK(timestamp)

Returns the day of the week of a TIMESTAMP data type as an integer between 1 (Sunday) and 7 (Saturday), inclusively.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the day of the week.

DAYOFYEAR(timestamp)

Returns the day of the year of a TIMESTAMP data type as an integer between 1 and 366, inclusively. The integer 1 refers to January 1.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the day of the year.

FORMAT_UTC_USEC(unix_timestamp)

Returns a human-readable string representation of a UNIX timestamp in the format YYYY-MM-DD HH:MM:SS.ffffff.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The unix timestamp to format.

HOUR(timestamp)

Returns the hour of a TIMESTAMP data type as an integer between 0 and 23, inclusively.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the hour as an integer.

MINUTE(timestamp)

Returns the minutes of a TIMESTAMP data type as an integer between 0 and 59, inclusively.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the minutes as an integer.

MONTH(timestamp)

Returns the month of a TIMESTAMP data type as an integer between 1 and 12, inclusively.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the month as an integer.

MSEC_TO_TIMESTAMP(unix_timestamp)

Converts a UNIX timestamp in milliseconds to a TIMESTAMP data type.

Note: this function is only available when UseLegacySQL=True.

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

PARSE_UTC_USEC(date_string)

Converts a date string to a UNIX timestamp in microseconds. date_string must have the format YYYY-MM-DD HH:MM:SS[.uuuuuu]. The fractional part of the second can be up to 6 digits long or can be omitted.

Note: this function is only available when UseLegacySQL=True.

- **date_string:** The date string to convert.

QUARTER(timestamp)

Returns the quarter of the year of a TIMESTAMP data type as an integer between 1 and 4, inclusively.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the quarter as an integer.

SEC_TO_TIMESTAMP(unix_timestamp)

Converts a UNIX timestamp in seconds to a TIMESTAMP data type.

Note: this function is only available when UseLegacySQL=True.

- **unix_timestamp:** The unix timestamp to convert.

SECOND(timestamp)

Returns the seconds of a TIMESTAMP data type as an integer between 0 and 59, inclusively. During a leap second, the integer range is between 0 and 60, inclusively.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the second as an integer.

STRFTIME_UTC_USEC(unix_timestamp, date_format_str)

Returns a human-readable date string in the format date_format_str.date_format_str can include date-related punctuation characters (such as / and -) and special characters accepted by the strftime function in C++ (such as %d for day of month).

Note: this function is only available when UseLegacySQL=True.

- **unix_timestamp:** The unix timestamp to convert.
- **date_format_str:** The date format string.

TIMESTAMP_SECONDS(unix_timestamp)

Interprets INT64_expression as the number of seconds since 1970-01-01 00:00:00 UTC.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

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

TIMESTAMP_MILLIS(unix_timestamp)

Interprets INT64_expression as the number of milliseconds since 1970-01-01 00:00:00 UTC.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

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

TIMESTAMP_MICROS(unix_timestamp)

Interprets INT64_expression as the number of microseconds since 1970-01-01 00:00:00 UTC.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

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

TIMESTAMP_TO_MSEC(timestamp)

Converts a TIMESTAMP data type to a UNIX timestamp in milliseconds.

Note: this function is only available when UseLegacySQL=True.

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

TIMESTAMP_TO_SEC(timestamp)

Converts a TIMESTAMP data type to a UNIX timestamp in seconds.

Note: this function is only available when UseLegacySQL=True.

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

TIMESTAMP_TO_USEC(timestamp)

Converts a TIMESTAMP data type to a UNIX timestamp in microseconds.

Note: this function is only available when UseLegacySQL=True.

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

UNIX_DATE(date_string)

Returns the number of days since 1970-01-01.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **date_string:** The date string to convert.

UNIX_SECONDS(timestamp)

Returns the number of seconds since 1970-01-01 00:00:00 UTC. Truncates higher levels of precision.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

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

UNIX_MILLIS(timestamp)

Returns the number of milliseconds since 1970-01-01 00:00:00 UTC. Truncates higher levels of precision.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

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

UNIX_MICROS(timestamp)

Returns the number of microseconds since 1970-01-01 00:00:00 UTC. Truncates higher levels of precision.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

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

USEC_TO_TIMESTAMP(unix_timestamp)

Converts a UNIX timestamp in microseconds to a TIMESTAMP data type.

Note: this function is only available when UseLegacySQL=True.

- **unix_timestamp:** The unix timestamp to convert.

UTC_USEC_TO_DAY(unix_timestamp)

Shifts a UNIX timestamp in microseconds to the beginning of the day it occurs in.

Note: this function is only available when UseLegacySQL=True.

- **unix_timestamp:** The unix timestamp to shift.

UTC_USEC_TO_HOUR(unix_timestamp)

Shifts a UNIX timestamp in microseconds to the beginning of the hour it occurs in.

Note: this function is only available when UseLegacySQL=True.

- **unix_timestamp:** The unix timestamp to shift.

UTC_USEC_TO_MONTH(unix_timestamp)

Shifts a UNIX timestamp in microseconds to the beginning of the month it occurs in.

Note: this function is only available when UseLegacySQL=True.

- **unix_timestamp:** The unix timestamp to shift.

UTC_USEC_TO_WEEK(unix_timestamp, day_of_week)

Returns a UNIX timestamp in microseconds that represents a day in the week of the unix_timestamp argument.

Note: this function is only available when UseLegacySQL=True.

- **unix_timestamp:** The unix timestamp to shift.
- **day_of_week:** A day of the week from 0 (Sunday) to 6 (Saturday).

UTC_USEC_TO_YEAR(unix_timestamp)

Returns a UNIX timestamp in microseconds that represents the year of the unix_timestamp argument.

Note: this function is only available when UseLegacySQL=True.

- **unix_timestamp:** The unix timestamp to convert.

WEEK(timestamp)

Returns the week of a TIMESTAMP data type as an integer between 1 and 53, inclusively. Weeks begin on Sunday, so if January 1 is on a day other than Sunday, week 1 has fewer than 7 days and the first Sunday of the year is the first day of week 2.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the week as an integer.

YEAR(timestamp)

Returns the year of a TIMESTAMP data type.

Note: this function is only available when UseLegacySQL=True.

- **timestamp:** The timestamp from which to return the year as an integer.

ABS(expression)

Returns the absolute value of the argument.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any column or literal expression.

ACOS(expression)

Returns the arc cosine of the argument.

- **expression:** Any column or literal expression.

ACOSH(expression)

Returns the arc hyperbolic cosine of the argument.

- **expression:** Any column or literal expression.

ASIN(expression)

Returns arcsine in radians.

- **expression:** Any column or literal expression.

ASINH(expression)

Returns the arc hyperbolic sine of the argument.

- **expression:** Any column or literal expression.

ATAN(expression)

Returns arc tangent of the argument.

- **expression:** Any column or literal expression.

ATANH(expression)

Returns the arc hyperbolic tangent of the argument.

- **expression:** Any column or literal expression.

ATAN2(expression1, expression2)

Returns the arc tangent of the two arguments.

- **expression1:** Any column or literal expression.
- **expression2:** Any column or literal expression.

CEIL(expression)

Rounds the argument up to the nearest whole number and returns the rounded value.

- **expression:** Any column or literal expression.

CEILING(expression)

Synonym for CEIL function.

- **expression:** Any column or literal expression.

COS(expression)

Returns the cosine of the argument.

- **expression:** Any column or literal expression.

COSH(expression)

Returns the hyperbolic cosine of the argument.

- **expression:** Any column or literal expression.

DEGREES(expression)

Returns expression, converted from radians to degrees.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any column or literal expression.

EXP(expression)

Returns the result of raising the constant "e" - the base of the natural logarithm - to the power of expression.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any column or literal expression.

FLOOR(expression)

Rounds the argument down to the nearest whole number and returns the rounded value.

- **expression:** Any column or literal expression.

LN(expression)

Returns the natural logarithm of the argument.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any column or literal expression.

LOG(expression)

Returns the natural logarithm of the argument.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any column or literal expression.

LOG2(expression)

Returns the Base-2 logarithm of the argument.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any column or literal expression.

LOG10(expression)

Returns the Base-10 logarithm of the argument.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any column or literal expression.

PI()

Returns PI.

Note: this function is only available when UseLegacySQL=True.

POW(expression1, expression2)

Returns the result of raising expression1 to the power of expression2.

- **expression1**: Any column or literal expression.
- **expression2**: Any column or literal expression.

POWER(expression1, expression2)

Synonym of POW function.

- **expression1**: Any column or literal expression.
- **expression2**: Any column or literal expression.

RADIANS(expression)

Returns expression, converted from degrees to radians.

Note: this function is only available when UseLegacySQL=True.

- **expression**: Any column or literal expression.

RAND([expression])

Returns a random float value in the range $0.0 \leq \text{value} < 1.0$. Each int32_seed value always generates the same sequence of random numbers within a given query, as long as you don't use a LIMIT clause. If int32_seed is not specified, BigQuery uses the current timestamp as the seed value.

Note: this function is only available when UseLegacySQL=True.

- **expression**: Any column or literal expression.

ROUND(expression [, integer_digits])

Rounds the argument either up or down to the nearest whole number (or if specified, to the specified number of digits) and returns the rounded value.

- **expression:** Any column or literal expression.
- **integer_digits:** The number of digits to round to.

GREATEST(value1[, value2 [, ...]])

Returns NULL if any of the inputs is NULL. Otherwise, returns NaN if any of the inputs is NaN. Otherwise, returns the largest value among X1,...,XN according to the < comparison.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **value1:** The first value to compare.
- **value2:** The second value to compare.

LEAST(value1[, value2 [, ...]])

Returns NULL if any of the inputs is NULL. Otherwise, returns NaN if any of the inputs is NaN. Otherwise, returns the smallest value among X1,...,XN according to the > comparison.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **value1:** The first value to compare.
- **value2:** The second value to compare.

SIN(expression)

Returns the sine of the argument.

- **expression:** Any column or literal expression.

SINH(expression)

Returns the hyperbolic sine of the argument.

- **expression:** Any column or literal expression.

SQRT(expression)

Returns the square root of the expression.

Note: this function is only available when UseLegacySQL=True.

- **expression:** Any column or literal expression.

TAN(expression)

Returns the tangent of the argument.

- **expression:** Any column or literal expression.

TANH(expression)

Returns the hyperbolic tangent of the argument.

- **expression:** Any column or literal expression.

TRUNC(expression [, integer_digits])

Rounds X to the nearest integer whose absolute value is not greater than Xs. When the integer_digits parameter is specified this function is similar to ROUND(X, N) but always rounds towards zero. Unlike ROUND(X, N) it never overflows.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** Any column or literal expression.
- **integer_digits:** The number of digits to round to.

BYTE_LENGTH(str)

Returns the length of the value in bytes, regardless of whether the type of the value is STRING or BYTES.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **str**: The string to calculate the length on.

CHAR_LENGTH(str)

Returns the length of the STRING in characters.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **str**: The string to calculate the length on.

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

Returns the concatenation of two or more strings, or NULL if any of the values are NULL.

- **str1**: The first string to concatenate.
- **str2**: The second string to concatenate.
- **str3**: The third string to concatenate.

ENDS_WITH(str1, str2)

Takes two values. Returns TRUE if the second value is a suffix of the first.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **str1**: The string to search in.
- **str2**: The string to search for.

FROM_BASE64(string_expr)

Converts the base64-encoded input string_expr into BYTES format. To convert BYTES to a base64-encoded STRING, use TO_BASE64.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **string_expr**: The string to convert from base64 encoding.

FROM_HEX(string_expr)

Converts a hexadecimal-encoded STRING into BYTES format. Returns an error if the input STRING contains characters outside the range (0..9, A..F, a..f). The lettercase of the characters does not matter. To convert BYTES to a hexadecimal-encoded STRING, use TO_HEX.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **string_expr**: The string to convert from hexadecimal encoding.

INSTR(str1, str2)

Returns the one-based index of the first occurrence of str2 in str1, or returns 0 if str2 does not occur in str1.

Note: this function is only available when UseLegacySQL=True.

- **str1**: The string to search in.
- **str2**: The string to search for.

LEFT(str, numeric_expression)

Returns the leftmost numeric_expr characters of str. If the number is longer than str, the full string will be returned. Example: LEFT('seattle', 3) returns sea.

Note: this function is only available when UseLegacySQL=True.

- **str**: The string to perform the LEFT operation on.
- **numeric_expression**: The number of characters to return.

LENGTH(str)

Returns a numerical value for the length of the string. Example: if str is '123456', LENGTH returns 6.

- **str**: The string to calculate the length on.

LOWER(str)

Returns the original string with all characters in lower case.

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

LPAD(str1, numeric_expression[, str2])

Pads str1 on the left with str2, repeating str2 until the result string is exactly numeric_expr characters. Example: LPAD('1', 7, '?') returns ??????1.

- **str1:** The string to pad.
- **numeric_expression:** The number of str2 instances to pad.
- **str2:** The pad characters.

LTRIM(str1 [, str2])

Removes characters from the left side of str1. If str2 is omitted, LTRIM removes spaces from the left side of str1. Otherwise, LTRIM removes any characters in str2 from the left side of str1 (case-sensitive).

- **str1:** The string to trim.
- **str2:** The characters to trim from str1.

REPEAT(str, repetitions)

Returns a value that consists of original_value, repeated. The repetitions parameter specifies the number of times to repeat original_value. Returns NULL if either original_value or repetitions are NULL. This function return an error if the repetitions value is negative.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **str:** The string to repeat.
- **str2:** The number of repetitions.

REPLACE(original_value, from_value, to_value)

Replaces all instances of from_value within original_value with to_value.

- **original_value**: The string to search in.
- **from_value**: The string to search for.
- **to_value**: The string to replace instances of from_value.

REVERSE(str)

Returns the reverse of the input STRING or BYTES.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

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

RIGHT(str, numeric_expression)

Returns the rightmost numeric_expr characters of str. If the number is longer than the string, it will return the whole string. Example: RIGHT('kirkland', 4) returns land.

Note: this function is only available when UseLegacySQL=True.

- **str**: The string to perform the RIGHT operation on.
- **numeric_expression**: The number of characters to return.

RPAD(str1, numeric_expression, str2)

Pads str1 on the right with str2, repeating str2 until the result string is exactly numeric_expr characters. Example: RPAD('1', 7, '?') returns 1?????.

- **str1**: The string to pad.
- **numeric_expression**: The number of str2 instances to pad.
- **str2**: The pad characters.

RTRIM(str1 [, str2])

Removes trailing characters from the right side of str1. If str2 is omitted, RTRIM removes trailing spaces from str1. Otherwise, RTRIM removes any characters in str2 from the right side of str1 (case-sensitive).

- **str1**: The string to trim.
- **str2**: The characters to trim from str1.

SPLIT(str [, delimiter])

Splits a string into repeated substrings. If delimiter is specified, the SPLIT function breaks str into substrings, using delimiter as the delimiter.

- **str**: The string to split.
- **delimiter**: The delimiter to split the string on. Default delimiter is a comma (,).

STARTS_WITH(str1, str2)

Takes two values. Returns TRUE if the second value is a prefix of the first.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **str1**: The string to search in.
- **str2**: The string to search for.

STRPOS(str1, str2)

Returns the 1-based index of the first occurrence of value2 inside value1. Returns 0 if value2 is not found.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **str1**: The string to search in.
- **str2**: The string to search for.

SUBSTR(str, index [, max_len])

Returns a substring of str, starting at index. If the optional max_len parameter is used, the returned string is a maximum of max_len characters long. Counting starts at 1, so the first character in the string is in position 1 (not zero). If index is 5, the substring begins with the 5th character from the left in str. If index is -4, the substring begins with the 4th character from the right in str. Example: SUBSTR('awesome', -4, 4) returns the substring some.

- **str**: The original string.
- **index**: The starting index.
- **max_len**: The maximum length of the return string.

TO_BASE64(string_expr)

Converts a sequence of BYTES into a base64-encoded STRING. To convert a base64-encoded STRING into BYTES, use FROM_BASE64.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **string_expr**: The string to convert to base64 encoding.

TO_HEX(string_expr)

Converts a sequence of BYTES into a hexadecimal STRING. Converts each byte in the STRING as two hexadecimal characters in the range (0..9, a..f). To convert a hexadecimal-encoded STRING to BYTES, use FROM_HEX.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **string_expr**: The string to convert to hexadecimal encoding.

TRIM(str1 [, str2])

Removes all leading and trailing characters that match value2. If value2 is not specified, all leading and trailing whitespace characters (as defined by the Unicode standard) are removed. If the first argument is of type BYTES, the second argument is required. If value2 contains more than one character or byte, the function removes all leading or trailing characters or bytes contained in value2.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **str1**: The string to trim.
- **str2**: The optional string characters to trim from str1.

UPPER(str)

Returns the original string with all characters in upper case.

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

JSON_EXTRACT(json, json_path)

Selects a value in json according to the JSONPath expression json_path. json_path must be a string constant. Returns the value in JSON string format.

- **json**: The JSON to select a value from.
- **json_path**: The JSON path of the value contained in json.

JSON_EXTRACT_SCALAR(json, json_path)

Selects a value in json according to the JSONPath expression json_path. json_path must be a string constant, and bracket notation is not supported. Returns a scalar JSON value.

- **json**: The JSON to select a value from.
- **json_path**: The JSON path of the value contained in json.

REGEXP_CONTAINS(str, reg_exp)

Returns TRUE if value is a partial match for the regular expression, regex. You can search for a full match by using ^ (beginning of text) and \$ (end of text). If the regex argument is invalid, the function returns an error.

Note: this function is only available when UseLegacySQL=True.

- **str**: The string to match in the regular expression.
- **reg_exp**: The regular expression to match.

REGEXP_EXTRACT(str, reg_exp)

Returns the portion of str that matches the capturing group within the regular expression.

- **str**: The string to match in the regular expression.
- **reg_exp**: The regular expression to match.

REGEXP_EXTRACT_ALL(str, reg_exp)

Returns an array of all substrings of value that match the regular expression, regex. The REGEXP_EXTRACT_ALL function only returns non-overlapping matches. For example, using this function to extract ana from banana returns only one substring, not two.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **str**: The string to match in the regular expression.
- **reg_exp**: The regular expression to match.

REGEXP_REPLACE(orig_str, reg_exp, replace_str)

Returns a string where any substring of orig_str that matches reg_exp is replaced with replace_str. For example, REGEXP_REPLACE ('Hello', 'lo', 'p') returns Help.

- **orig_str**: The original string to match in the regular expression.
- **reg_exp**: The regular expression to match.
- **replace_str**: The replacement for the matched orig_str in the regular expression.

FORMAT_IP(integer_value)

Converts 32 least significant bits of integer_value to human-readable IPv4 address string.

Note: this function is only available when UseLegacySQL=True.

- **integer_value**: The integer value to convert to an IPv4 address.

PARSE_IP(readable_ip)

Converts a string representing IPv4 address to unsigned integer value. For example, `PARSE_IP('0.0.0.1')` will return 1. If string is not a valid IPv4 address, `PARSE_IP` will return NULL.

Note: this function is only available when `UseLegacySQL=True`.

- **readable_ip:** The IPv4 address to convert to an integer.

NET.IPV4_FROM_INT64(integer_value)

Converts an IPv4 address from integer format to binary (BYTES) format in network byte order. In the integer input, the least significant bit of the IP address is stored in the least significant bit of the integer, regardless of host or client architecture. For example, 1 means 0.0.0.1, and 0x1FF means 0.0.1.255.

Note: this function is only available when using Standard SQL (`UseLegacySQL=False`).

- **integer_value:** The integer value to convert to an IPv4 address.

NET.IPV4_TO_INT64(readable_ip)

Converts an IPv4 address from binary (BYTES) format in network byte order to integer format. In the integer output, the least significant bit of the IP address is stored in the least significant bit of the integer, regardless of host or client architecture. For example, 1 means 0.0.0.1, and 0x1FF means 0.0.1.255. The output is in the range [0, 0xFFFFFFFF]. If the input length is not 4, this function throws an error. This function does not support IPv6.

Note: this function is only available when using Standard SQL (`UseLegacySQL=False`).

- **readable_ip:** The IPv4 address to convert to an integer.

FARM_FINGERPRINT(expression)

Computes the fingerprint of the STRING or BYTES input using the Fingerprint64 function from the open-source FarmHash library. The output of this function for a particular input will never change.

Note: this function is only available when using Standard SQL (`UseLegacySQL=False`).

- **expression:** The expression to use to compute the fingerprint.

MD5(expression)

Computes the hash of the input using the MD5 algorithm. The input can either be STRING or BYTES. The string version treats the input as an array of bytes. This function returns 16 bytes.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** The expression to use to compute the hash.

SHA1(expression)

Computes the hash of the input using the SHA-1 algorithm. The input can either be STRING or BYTES. The string version treats the input as an array of bytes. This function returns 20 bytes.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** The expression to use to compute the hash.

SHA256(expression)

Computes the hash of the input using the SHA-256 algorithm. The input can either be STRING or BYTES. The string version treats the input as an array of bytes. This function returns 32 bytes.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** The expression to use to compute the hash.

SHA512(expression)

Computes the hash of the input using the SHA-512 algorithm. The input can either be STRING or BYTES. The string version treats the input as an array of bytes. This function returns 64 bytes.

Note: this function is only available when using Standard SQL (UseLegacySQL=False).

- **expression:** The expression to use to compute the hash.

TIMESTAMP(datetime_expression[, timezone])

Convert a date, datetime, or string to a TIMESTAMP data type.

Note: this function does not support the timezone parameter and requires datetime_expression to be a string when using Legacy SQL (UseLegacySQL=True).

- **datetime_expression:** The expression to be converted to a timestamp
- **timezone:** The timezone to be used. If no timezone is specified, the default timezone, UTC, is used

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 actor.attributes.email,
repository.name INTO 'csv://c:/[publicdata].[samples].github_nested.txt'
FROM '[publicdata].[samples].github_nested' WHERE repository.name =
'EntityFramework'");
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 '[publicdata].
[samples].github_nested' IN 'csv://filename=c:/[publicdata].
[samples].github_nested.csv;delimiter=tab' FROM '[publicdata].
[samples].github_nested' WHERE repository.name = 'EntityFramework'");
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. To retrieve the Id of the last inserted record use `getGeneratedKeys`. Additionally, set the **RETURN_GENERATED_KEYS** flag of the `Statement` class when you call `prepareStatement`.

```
String cmd = "INSERT INTO [publicdata].[samples].github_nested
(repository.name) VALUES (?)";
PreparedStatement pstmt = connection.prepareStatement
(cmd, Statement.RETURN_GENERATED_KEYS);
pstmt.setString(1, "CoreCLR");
int count = pstmt.executeUpdate();
System.out.println(count+" rows were affected");
ResultSet rs = pstmt.getGeneratedKeys();
while(rs.next()){
    System.out.println(rs.getString("Id"));
}
connection.close();
```

UPDATE Statements

To modify existing records, use UPDATE statements.

Update Syntax

The UPDATE statement takes as input a comma-separated list of columns and new column values as name-value pairs in the SET clause, as shown in the following example:

```

UPDATE <table_name> SET { <column_reference> = <expression> } [ , ... ]
WHERE { Id = <expression> } [ { AND | OR } ... ]
<expression> ::=
    | @ <parameter>
    | ?
    | <literal>

```

You can use the `executeUpdate` method of the `Statement` or `PreparedStatement` classes to execute data manipulation commands and retrieve the rows affected, as shown in the following example:

```

String cmd = "UPDATE [publicdata].[samples].github_nested SET
repository.name='CoreCLR' WHERE Id = ?";
PreparedStatement pstmt = connection.prepareStatement(cmd);
pstmt.setString(1, "1");
int count = pstmt.executeUpdate();
System.out.println(count + " rows were affected");
connection.close();

```

DELETE Statements

To delete information from a table, use DELETE statements.

DELETE Syntax

The DELETE statement requires the table name in the FROM clause and the row's primary key in the WHERE clause, as shown in the following example:

```

<delete_statement> ::= DELETE FROM <table_name> WHERE { Id =
<expression> } [ { AND | OR } ... ]
<expression> ::=
    | @ <parameter>
    | ?
    | <literal>

```

You can use the `executeUpdate` method of the `Statement` or `PreparedStatement` classes to execute data manipulation commands and retrieve the number of affected rows, as shown in the following example:

```

Connection connection = DriverManager.getConnection

```



```
( "jdbc:googlebigquery:InitiateOAuth=GETANDREFRESH;ProjectId=NameOfProject;DatasetId=NameOfDataset;", );
String cmd = "DELETE FROM [publicdata].[samples].github_nested WHERE Id = ?";
PreparedStatement pstmt = connection.prepareStatement(cmd);
pstmt.setString(1, "1");
int count=pstmt.executeUpdate();
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 Google BigQuery Adapter models the data as defined within Google BigQuery for the ProjectId and DatasetId configured.

Views

[Views](#) are client-side tables that cannot be modified. The adapter uses these to report metadata about the Google BigQuery projects and datasets it is connected to.

In addition, the adapter supports server-side views defined within Google BigQuery. These views may be used in SELECT statements the same way as tables. However, view schemas can easily become out of date and require the adapter to refresh them. Please see [RefreshViewSchemas](#) for more details.

Stored Procedures

[Stored Procedures](#) are function-like interfaces to the data source. The adapter uses these to manage Google BigQuery tables and jobs and to perform OAuth operations.

In addition to the client-side stored procedures offered by the adapter, there is also support for server-side stored procedures defined in Google BigQuery. The adapter supports both CALL and EXEC using the procedure's parameter names. Note that adapter only supports IN parameters and resultset return values.

```
CALL `psychic-valve-137816`.Northwind.MostPopularProduct()
CALL `psychic-valve-137816`.Northwind.GetStockedValue(24, 0.75)
EXEC `psychic-valve-137816`.Northwind.MostPopularProduct
```

```
EXEC `psychic-valve-137816`.Northwind.GetSockedValue productId = 24,
discountRate = 0.75
```

Additional Metadata

Table Descriptions

Google BigQuery supports setting descriptions on tables but the adapter does not report these by default. [ShowTableDescriptions](#) can be used to report table descriptions.

Primary Keys

Google BigQuery does not support primary keys natively, but the adapter allows you to define them so they can be used in environments that require primary keys to modify data. Primary keys can be defined using the [PrimaryKeyIdentifiers](#) option.

Policy Tags

If policy tags from the Data Catalog service are defined on a table, they can be retrieved from the system tables using the PolicyTags column:

```
SELECT ColumnName, PolicyTags FROM sys_tablecolumns
WHERE CatalogName = 'psychic-valve-137816'
AND SchemaName = 'Northwind'
AND TableName = 'Customers'
```

Tables

Tables

Table definitions are dynamically generated based on the table definitions within Google BigQuery for the Project and Dataset specified in the connection string options.

Views

Views are composed of columns and pseudo columns. Views are similar to tables in the way that data is represented; however, views do not support updates. Entities that are represented as views are typically read-only entities. Often, a stored procedure is available to update the data if such functionality is applicable to the data source.

Queries can be executed against a view as if it were a normal table, and the data that comes back is similar in that regard.

Dynamic views, such as queries exposed as views, and views for looking up specific combinations of project_team work items are supported.

Google BigQuery Adapter Views

Name	Description
Datasets	Lists all the accessible datasets for a given project.
PartitionsList	Lists the partitioning definitions for tables
PartitionsValues	Lists the partitioning ranges for tables
Projects	Lists all the projects for the authorized user.

Datasets

Lists all the accessible datasets for a given project.

Columns

Name	Type	Description

Id [KEY]	<i>String</i>	The fully qualified, unique, opaque Id of the dataset.
Kind	<i>String</i>	The resource type.
FriendlyName	<i>String</i>	A descriptive name for the dataset
DatasetReference_ProjectId	<i>String</i>	A unique reference to the container project.
DatasetReference_DatasetId	<i>String</i>	A unique reference to the dataset, without the project name.

PartitionsList

Lists the partitioning definitions for tables

Columns

Name	Type	Description
Id [KEY]	<i>String</i>	A unique identifier for the partition.
ProjectId	<i>String</i>	The project that the table belongs to.
DatasetId	<i>String</i>	The dataset that the table belongs to.
TableName	<i>String</i>	The name of the table.

ColumnName	<i>String</i>	The name of the column used for partitioning.
ColumnType	<i>String</i>	The type of the partitioning column.
Kind	<i>String</i>	The type of partitioning used by the table. One of DATE, RANGE or INGESTION.
RequireFilter	<i>Boolean</i>	Whether a filter on the partition column is required to query the table.

PartitionsValues

Lists the partitioning ranges for tables

Columns

Name	Type	Description
Id	<i>String</i>	A unique identifier for the partition.
RangeLow	<i>String</i>	The lowest value of the partition column. Either an integer when Kind is RANGE, or a date otherwise.
RangeHigh	<i>String</i>	The highest value of the partition column. Either an integer when Kind is RANGE, or a date otherwise.
RangeInterval	<i>String</i>	The range of values which are included in each partition. Only valid when Kind is RANGE
DateResolution	<i>String</i>	How much of the date is significant to a TIME or INGESTION partition column. One of DAY, HOUR, MONTH or YEAR.

Projects

Lists all the projects for the authorized user.

Columns

Name	Type	Description
Id [KEY]	<i>String</i>	The unique identifier of the Project
Kind	<i>String</i>	The resource type.
FriendlyName	<i>String</i>	A descriptive name for the project.
NumericId	<i>String</i>	The numeric Id of the project.
ProjectReference_ProjectId	<i>String</i>	A unique reference to the project.

Stored Procedures

Stored procedures are function-like interfaces that extend the functionality of the adapter beyond simple SELECT/INSERT/UPDATE/DELETE operations with Google BigQuery.

Stored procedures accept a list of parameters, perform their intended function, and then return, if applicable, any relevant response data from Google BigQuery, along with an indication of whether the procedure succeeded or failed.

Google BigQuery Adapter Stored Procedures

Name	Description
CancelJob	Cancels a running BigQuery job.
CreateSchema	Creates a schema file for the specified table or view.
DeleteTable	Deletes the specified table from Google BigQuery.
GetJob	Retrieves the configuration information and execution state for an existing job
GetOAuthAccessToken	Obtains the OAuth access token to be used for authentication with various Google services.
GetOAuthAuthorizationURL	Obtains the OAuth authorization URL for authentication with various Google services.
InsertJob	Inserts a Google BigQuery job, which can then be selected later to retrieve the query results.
InsertLoadJob	Inserts a Google BigQuery load job, which adds data from Google Cloud Storage into an existing table.
RefreshOAuthAccessToken	Obtains the OAuth access token to be used for authentication with various Google services.

CancelJob

Cancels a running BigQuery job.

Input

Name	Type	Description
JobId	<i>String</i>	The Id of the job you wish to cancel.

Region	<i>String</i>	The region where the job is executing. Not required if the job is a US or EU region.
--------	---------------	--

Result Set Columns

Name	Type	Description
JobId	<i>String</i>	The JobId of the cancelled Job.
Region	<i>String</i>	The region where the job was executing.
Configuration_query_query	<i>String</i>	The query of the cancelled job.
Configuration_query_destinationTable_tableId	<i>String</i>	The destination table tableId of the cancelled job.
Configuration_query_destinationTable_projectId	<i>String</i>	The destination table projectId of the newly inserted job.
Configuration_query_destinationTable_datasetId	<i>String</i>	The destination table datasetId of the newly inserted job.
Status_State	<i>String</i>	Running state of the job.
Status_errorResult_reason	<i>String</i>	A short error code that summarizes the error.
Status_errorResult_message	<i>String</i>	A human-readable description of the error.

CreateSchema

Creates a schema file for the specified table or view.

CreateSchema

Creates a local schema file (.rsd) from an existing table or view in the data model.

The schema file is created in the directory set in the Location connection property when this procedure is executed. You can edit the file to include or exclude columns, rename columns, or adjust column datatypes.

The adapter checks the Location to determine if the names of any .rsd files match a table or view in the data model. If there is a duplicate, the schema file will take precedence over the default instance of this table in the data model. If a schema file is present in Location that does not match an existing table or view, a new table or view entry is added to the data model of the adapter.

Input

Name	Type	Accepts Output Streams	Description
TableName	String	False	The name of the table or view.
FileName	String	False	The full file path and name of the schema to generate. Ex: 'C:\\Users\\User\\Desktop\\GoogleBigQuery\\table.rsd'
FileStream	String	True	Stream to write the schema to. Only used if FileName is not provided.

Result Set Columns

Name	Type	Description
Result	<i>String</i>	Returns Success or Failure.
FileData	<i>String</i>	The content of the schema encoded as base64. Only returned if FileName and FileStream are not provided.

DeleteTable

Deletes the specified table from Google BigQuery.

Input

Name	Type	Description
TableId	<i>String</i>	TableId of the table you wish to delete. ProjectId and DatasetId can come from connection properties, or to override them, the format projectId:datasetId.TableId.

Result Set Columns

Name	Type	Description
Success	<i>String</i>	Returns true if operation is successful, else an exception is returned.

GetJob

Retrieves the configuration information and execution state for an existing job

Input

Name	Type	Description
JobId	<i>String</i>	The Id of the job you wish to return.
Region	<i>String</i>	The region where the job is executing. Not required if the job is a US or EU region.

Result Set Columns

Name	Type	Description
JobId	<i>String</i>	The JobId of the newly insert Job.
Region	<i>String</i>	The region where the job is executing.
Configuration_query_query	<i>String</i>	The query of the newly inserted Job.
Configuration_query_destinationTable_tableId	<i>String</i>	The destination table tableId of the newly inserted Job.
Configuration_query_destinationTable_projectId	<i>String</i>	The destination table projectId of the newly inserted Job.
Configuration_query_destinationTable_datasetId	<i>String</i>	The destination table

		datasetId of the newly inserted Job.
Status_State	String	Running state of the job.
Status_errorResult_reason	String	A short error code that summarizes the error.
Status_errorResult_message	String	A human-readable description of the error.

GetOAuthAccessToken

Obtains the OAuth access token to be used for authentication with various Google services.

Input

Name	Type	Description
AuthMode	String	The type of authentication mode to use. The allowed values are <i>APP</i> , <i>WEB</i> . The default value is <i>WEB</i> .
Verifier	String	The verifier code returned by Google after permissions have been granted for the app to connect. <i>WEB</i> Authmode only.
Scope	String	The scope of access to Google APIs. By default, access to all APIs used by this data provider will be specified.
CallbackURL	String	Determines where the response is sent. The value of this parameter must exactly match one of the values registered in the APIs Console (including the http or https schemes, case, and trailing '/').

Prompt	<i>String</i>	<p>This field indicates the prompt to present the user. It accepts one of the following values: NONE, CONSENT, SELECT ACCOUNT. The default is SELECT_ACCOUNT, so a given user will be prompted to select the account to connect to. If it is set to CONSENT, the user will see a consent page every time, even if they have previously given consent to the application for a given set of scopes. Lastly, if it is set to NONE, no authentication or consent screens will be displayed to the user.</p> <p>The default value is <i>SELECT_ACCOUNT</i>.</p>
AccessType	<i>String</i>	<p>Indicates if your application needs to access a Google API when the user is not present at the browser. This parameter defaults to offline. If your application needs to refresh access tokens when the user is not present at the browser, then use offline. This will result in your application obtaining a refresh token the first time your application exchanges an authorization code for a user.</p> <p>The allowed values are <i>ONLINE</i>, <i>OFFLINE</i>.</p> <p>The default value is <i>OFFLINE</i>.</p>
State	<i>String</i>	<p>Indicates any state which may be useful to your application upon receipt of the response. The Google Authorization Server roundtrips this parameter, so your application receives the same value it sent. Possible uses include redirecting the user to the correct resource in your site, nonces, and cross-site-request-forgery mitigations.</p>

Result Set Columns

Name	Type	Description
OAuthAccessToken	<i>String</i>	The authentication token returned from Google. This can be used in subsequent calls to other operations for this particular service.

OAuthRefreshToken	<i>String</i>	A token that may be used to obtain a new access token.
ExpiresIn	<i>String</i>	The remaining lifetime on the access token.

GetOAuthAuthorizationURL

Obtains the OAuth authorization URL for authentication with various Google services.

Input

Name	Type	Description
Scope	<i>String</i>	The scope of access to Google APIs. By default, access to all APIs used by this data provider will be specified.
CallbackURL	<i>String</i>	Determines where the response is sent. The value of this parameter must exactly match one of the values registered in the APIs Console (including the http or https schemes, case, and trailing '/').
Prompt	<i>String</i>	<p>This field indicates the prompt to present the user. It accepts one of the following values: NONE, CONSENT, SELECT_ACCOUNT. The default is SELECT_ACCOUNT, so a given user will be prompted to select the account to connect to. If it is set to CONSENT, the user will see a consent page every time, even if they have previously given consent to the application for a given set of scopes. Lastly, if it is set to NONE, no authentication or consent screens will be displayed to the user.</p> <p>The default value is <i>SELECT_ACCOUNT</i>.</p>
AccessType	<i>String</i>	Indicates if your application needs to access a Google API when the user is not present at the browser. This parameter defaults to offline. If your application needs to refresh access

tokens when the user is not present at the browser, then use offline. This will result in your application obtaining a refresh token the first time your application exchanges an authorization code for a user.

The allowed values are *ONLINE*, *OFFLINE*.

The default value is *OFFLINE*.

State	<i>String</i>	Indicates any state which may be useful to your application upon receipt of the response. The Google Authorization Server roundtrips this parameter, so your application receives the same value it sent. Possible uses include redirecting the user to the correct resource in your site, nonces, and cross-site-request-forgery mitigations.
-------	---------------	--

Result Set Columns

Name	Type	Description
URL	<i>String</i>	The URL to complete user authentication.

InsertJob

Inserts a Google BigQuery job, which can then be selected later to retrieve the query results.

Input

Name	Type	Description
Query	<i>String</i>	The query to submit to Google BigQuery.

IsDML	<i>String</i>	Should be true if the query is a DML statement and false otherwise. The default value is <i>false</i> .
DestinationTable	<i>String</i>	The destination table for the query, in the format DestProjectId:DestDatasetId.DestTable
WriteDisposition	<i>String</i>	How to write data to the destination table, such as truncate existing results, appending existing results, or writing only when the table is empty. The allowed values are <i>WRITE_TRUNCATE</i> , <i>WRITE_APPEND</i> , <i>WRITE_EMPTY</i> . The default value is <i>WRITE_TRUNCATE</i> .
DryRun	<i>String</i>	Whether or not this is a dry run of the job.
MaximumBytesBilled	<i>String</i>	If provided, BigQuery will cancel the job if it attempts to process more than this many bytes.
Region	<i>String</i>	The region to start executing the job in.

Result Set Columns

Name	Type	Description
JobId	<i>String</i>	The JobId of the newly insert Job.
Region	<i>String</i>	The region where the job is executing.
Configuration_query_query	<i>String</i>	The query of the newly inserted Job.

Configuration_query_destinationTable_tableId	<i>String</i>	The destination table tableId of the newly inserted Job.
Configuration_query_destinationTable_projectId	<i>String</i>	The destination table projectId of the newly inserted Job.
Configuration_query_destinationTable_datasetId	<i>String</i>	The destination table datasetId of the newly inserted Job.
Status_State	<i>String</i>	Running state of the job.
Status_errorResult_reason	<i>String</i>	A short error code that summarizes the error.
Status_errorResult_message	<i>String</i>	A human-readable description of the error.

InsertLoadJob

Inserts a Google BigQuery load job, which adds data from Google Cloud Storage into an existing table.

Input

Name	Type	Description
SourceURIs	<i>String</i>	A space-separated list of Google Cloud Storage URIs
SourceFormat	<i>String</i>	The source format that the files are formatted in. The allowed values are <i>AVRO</i> , <i>NEWLINE_</i>

		<i>DELIMITED_JSON, DATASTORE_BACKUP, PARQUET, ORC, CSV.</i>
DestinationTable	<i>String</i>	The destination table for the query, in the format DestProjectId.DestDatasetId.DestTable
DestinationTableProperties	<i>String</i>	A JSON object containing the table friendlyName, description and list of labels.
DestinationTableSchema	<i>String</i>	A JSON list containing the fields used to create the table.
DestinationEncryptionConfiguration	<i>String</i>	A JSON object giving the KMS encryption settings for the table.
SchemaUpdateOptions	<i>String</i>	A JSON list giving the options to apply when updating the destination table schema.
TimePartitioning	<i>String</i>	A JSON object giving the time partitioning type and field.
RangePartitioning	<i>String</i>	A JSON object giving the range partitioning field and buckets.
Clustering	<i>String</i>	A JSON object giving the fields to be used for clustering.
Autodetect	<i>String</i>	Whether options and schema should be automatically determined for JSON and CSV files.
CreateDisposition	<i>String</i>	Whether to create the destination table if it does not exist. The allowed values are <i>CREATE_IF_NEEDED</i> , <i>CREATE_NEVER</i> . The default value is <i>CREATE_IF_NEEDED</i> .

WriteDisposition	String	<p>How to write data to the destination table, such as truncate existing results, appending existing results, or writing only when the table is empty.</p> <p>The allowed values are <i>WRITE_TRUNCATE</i>, <i>WRITE_APPEND</i>, <i>WRITE_EMPTY</i>.</p> <p>The default value is <i>WRITE_APPEND</i>.</p>
Region	String	<p>The region to start executing the job in. Both the GCS resources and the BigQuery dataset must be in the same region.</p>
DryRun	String	<p>Whether or not this is a dry run of the job.</p> <p>The default value is <i>false</i>.</p>
MaximumBadRecords	String	<p>If provided, the number of records that can be invalid before the entire job is canceled. By default all records must be valid.</p> <p>The default value is <i>0</i>.</p>
IgnoreUnknownValues	String	<p>Whether to ignore unknown fields in the input file or treat them as errors. By default they are treated as errors.</p> <p>The default value is <i>false</i>.</p>
AvroUseLogicalTypes	String	<p>Whether to use Avro logical types when converting Avro data into BigQuery types.</p> <p>The default value is <i>true</i>.</p>
CSVSkipLeadingRows	String	<p>How many rows to skip at the start of CSV files. Usually used for skipping header rows.</p>

CSVEncoding	<i>String</i>	<p>The name of the encoding used for CSV files.</p> <p>The allowed values are <i>ISO-8859-1</i>, <i>UTF-8</i>.</p> <p>The default value is <i>UTF-8</i>.</p>
CSVNullMarker	<i>String</i>	<p>If provided, this string is used for NULL values within CSV files. By default CSV files cannot use NULL.</p>
CSVFieldDelimiter	<i>String</i>	<p>The character used to separate columns within CSV files.</p> <p>The default value is <i>,</i>.</p>
CSVQuote	<i>String</i>	<p>The character used for quoted fields in CSV files. May be set to empty to disable quoting.</p> <p>The default value is <i>"</i>.</p>
CSVAllowQuotedNewlines	<i>String</i>	<p>Whether CSV files can contain newlines within quoted fields.</p> <p>The default value is <i>false</i>.</p>
CSVAllowJaggedRows	<i>String</i>	<p>Whether lines in CSV files may contain missing fields. False by default</p> <p>The default value is <i>false</i>.</p>
DSBackupProjectionFields	<i>String</i>	<p>A JSON list of fields to load from a Cloud datastore backup.</p>
ParquetOptions	<i>String</i>	<p>A JSON object giving the Parquet-specific import options.</p>
DecimalTargetTypes	<i>String</i>	<p>A JSON list giving the preference order applied to numeric types.</p>
HivePartitioningOptions	<i>String</i>	<p>A JSON object giving the source-side partitioning options.</p>

Result Set Columns

Name	Type	Description
JobId	<i>String</i>	The JobId of the newly insert Job.
Region	<i>String</i>	The region where the job is executing.
Configuration_load_destinationTable_tableId	<i>String</i>	The destination table tableId of the newly inserted Job.
Configuration_load_destinationTable_projectId	<i>String</i>	The destination table projectId of the newly inserted Job.
Configuration_load_destinationTable_datasetId	<i>String</i>	The destination table datasetId of the newly inserted Job.
Status_State	<i>String</i>	Running state of the job.
Status_errorResult_reason	<i>String</i>	A short error code that summarizes the error.
Status_errorResult_message	<i>String</i>	A human-readable description of the error.

RefreshOAuthAccessToken

Obtains the OAuth access token to be used for authentication with various Google services.

Input

Name	Type	Description
OAuthRefreshToken	<i>String</i>	The refresh token returned from the original authorization code exchange.

Result Set Columns

Name	Type	Description
OAuthAuthToken	<i>String</i>	The authentication token returned from Google. This can be used in subsequent calls to other operations for this particular service.
OAuthRefreshToken	<i>String</i>	A token that may be used to obtain a new access token.
ExpiresIn	<i>String</i>	The remaining lifetime on the access token.

Data Type Mapping

Data Type Mappings

The adapter maps types from the data source to the corresponding data type available in the schema. The table below documents these mappings.

Google BigQuery	CData Schema
-----------------	--------------

STRING	string
BYTES	binary
INTEGER	long
FLOAT	double
NUMERIC	decimal
BIGNUMERIC	decimal
BOOLEAN	bool
DATE	date
TIME	time
DATETIME	datetime
TIMESTAMP	datetime
STRUCT	See below
ARRAY	See below
GEOGRAPHY	string

Note that the NUMERIC type supports 38 digits of precision and the BIGDECIMAL type supports 76 digits of precision. Most platforms do not have a decimal type that supports the full precision of these values (.NET decimal supports 28 digits, and Java BigDecimal supports 38 by default). If this is the case then these columns can be cast to a string when queried, or the connection can be configured to ignore them by setting **IgnoreTypes=decimal**.

STRUCT and ARRAY Types

Google BigQuery supports two kinds of types for storing compound values in a single row, **STRUCT** and **ARRAY**. In some places within Google BigQuery these are also known as

RECORD and **REPEATED** types.

A **STRUCT** is a fixed-size group of values which are accessed by name and can have different types. The adapter flattens structs so their individual fields can be accessed using dotted names. Note that these dotted names must be quoted.

```
-- trade_value STRUCT<currency STRING, value FLOAT>
SELECT CONCAT([trade_value.value], ' ', NULLIF([trade_value.currency],
'USD'))
FROM trades
```

An **ARRAY** is a group of values with the same type that can have any size. The adapter treats the array as a single compound value and reports it as a JSON aggregate.

These types may be combined such that a STRUCT type contains an ARRAY field, or an ARRAY field is a list of STRUCT values. The outer type takes precedence in how the field is processed:

```
/* Table contains fields:
  stocks STRUCT<symbol STRING, prices ARRAY<FLOAT>>
  offers: ARRAY<STRUCT<currency STRING, value FLOAT>>
*/
SELECT [stocks.symbol], /* ARRAY field can be read from STRUCT, but is
converted to JSON */
      [stocks.prices],
      [offers]           /* STRUCT fields in an ARRAY cannot be accessed
*/
FROM market
```

Type Parameters

The adapter exposes parameters on the following types. In each case the type parameters are optional, Google BigQuery has default values for types that are not parameterized.

- *STRING(length)*
- *BYTES(length)*
- *NUMERIC(precision)* or *NUMERIC(precision, scale)*
- *BIGNUMERIC(precision)* or *BIGNUMERIC(precision, scale)*

These parameters are primarily for restricting the data written to the table. They are included in the table metadata as the column size for STRING and BYTES, and the numeric precision and scale for NUMERIC and BIGNUMERIC.

Type parameters have no effect on queries and are not reported within query metadata. For example, here the output of CONCAT is a plain STRING even though its inputs are *a* `STRING(100)` and *b* `STRING(100)`.

```
SELECT CONCAT(a, b) FROM table_with_length_params
```

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 type of authentication to use when connecting to Google BigQuery.
ProjectId	The ProjectId used to resolve unqualified tables.
DatasetId	The DatasetId used to resolve unqualified tables.
BillingProjectId	The ProjectId of the billing project for executing jobs.

BigQuery

Property	Description
AllowLargeResultSets	Whether or not to allow large datasets to be stored in

	temporary tables for large datasets.
DestinationTable	This property determines where query results are stored in Google BigQuery.
UseQueryCache	Specifies whether to use Google BigQuery's built-in query cache.
PageSize	The number of results to return per page from Google BigQuery.
PollingInterval	This determines how long to wait in seconds, between checks to see if a job has completed.
AllowUpdatesWithoutKey	Whether or not to allow update without primary keys.
FilterColumns	Please set `AllowUpdatesWithoutKey` to true before you could use this property.
UseLegacySQL	Specifies whether to use BigQuery's legacy SQL dialect for this query. By default, Standard SQL will be used.

Storage API

Property	Description
UseStorageAPI	Specifies whether to use BigQuery's Storage API for bulk data reads.
UseArrowFormat	Specifies whether to use the Arrow format with BigQuery's Storage API.
StorageThreshold	The minimum number of rows a query must return to invoke the Storage API.
StoragePageSize	Specifies the page size to use for Storage API queries.

Uploading

Property	Description
InsertMode	Specifies what kind of method to use when inserting data. By default streaming inserts are used.
WaitForBatchResults	Whether to wait for the job to complete when using the bulk upload API. Only active when InsertMode is set to Upload.
GCSBucket	Specifies the name of a GCS bucket to upload bulk data for staging.
GCSBucketFolder	Specifies the name of the folder in GCSBucket to upload bulk data for staging.
TempTableDataset	The prefix of the dataset that will contain temporary tables when performing bulk UPDATE or DELETE operations.

OAuth

Property	Description
InitiateOAuth	Set this property to initiate the process to obtain or refresh the OAuth access token when you connect.
OAuthClientId	The client Id assigned when you register your application with an OAuth authorization server.
OAuthClientSecret	The client secret assigned when you register your application with an OAuth authorization server.
OAuthAccessToken	The access token for connecting using OAuth.
OAuthSettingsLocation	The location of the settings file where OAuth values are saved when InitiateOAuth is set to GETANDREFRESH or REFRESH. Alternatively, this can be held in memory by specifying a value starting with memory://.

Scope	Identifies the Google API access that your application is requesting. Scopes are space-delimited.
OAuthVerifier	The verifier code returned from the OAuth authorization URL.
OAuthRefreshToken	The OAuth refresh token for the corresponding OAuth access token.
OAuthExpiresIn	The lifetime in seconds of the OAuth AccessToken.
OAuthTokenTimestamp	The Unix epoch timestamp in milliseconds when the current Access Token was created.

JWT OAuth

Property	Description
OAuthJWTCert	The JWT Certificate store.
OAuthJWTCertType	The type of key store containing the JWT Certificate.
OAuthJWTCertPassword	The password for the OAuth JWT certificate.
OAuthJWTCertSubject	The subject of the OAuth JWT certificate.
OAuthJWTIssuer	The issuer of the Java Web Token.
OAuthJWTSubject	The user subject for which the application is requesting delegated access.

SSL

Property	Description

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.
RefreshViewSchemas	Allows the provider to determine up-to-date view schemas automatically.
ShowTableDescriptions	Controls whether table descriptions are returned via the platform metadata APIs and sys_tables / sys_views.
PrimaryKeyIdentifiers	Set this property to define primary keys.
AllowedTableTypes	Specifies what kinds of tables will be visible.
FlattenObjects	Determines whether the provider flattens STRUCT fields into top-level columns.

Miscellaneous

Property	Description
StorageTimeout	How long a Storage API connection must remain idle before the provider reconnects.
AllowAggregateParameters	Allows raw aggregates to be used in parameters when QueryPassthrough is enabled.
ApplicationName	An application name in the form application/version. For example, AcmeReporting/1.0.
AuditLimit	The maximum number of rows which will be stored within an audit table.
AuditMode	What provider actions should be recorded to audit tables.
BigQueryOptions	A comma separated list of Google BigQuery options.
GenerateSchemaFiles	Indicates the user preference as to when schemas should be generated and saved.
MaximumBillingTier	The MaximumBillingTier is a positive integer that serves as a multiplier of the basic price per TB. For example, if you set MaximumBillingTier to 2, the maximum cost for that query will be 2x basic price per TB.
MaximumBytesBilled	Limits how many bytes BigQuery will allow a job to consume before it is cancelled.
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 Google BigQuery from the provider.
TableSamplePercent	This determines what percent of a table is sampled with the

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

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 type of authentication to use when connecting to Google BigQuery.
ProjectId	The ProjectId used to resolve unqualified tables.
DatasetId	The DatasetId used to resolve unqualified tables.
BillingProjectId	The ProjectId of the billing project for executing jobs.

AuthScheme

The type of authentication to use when connecting to Google BigQuery.

Possible Values

Auto, OAuth, OAuthJWT, GCPIInstanceAccount

Data Type

string

Default Value

"Auto"

Remarks

- Auto: Lets the driver decide automatically based on the other connection properties you have set.
- OAuth: Set this to perform OAuth authentication using a standard user account.
- OAuthJWT: Set this to perform OAuth authentication using an OAuth service account.
- GCPIstanceAccount: Set this to get Access Token from Google Cloud Platform instance.

ProjectId

The ProjectId used to resolve unqualified tables.

Data Type

string

Default Value

""

Remarks

When a query refers to a table it can leave the project implicit, or qualify the project directly as the catalog portion of the table:

```
/* Implicit, resolved against connection string */
SELECT FirstName, LastName FROM `Northwind`.`customers`
/* Explicit, project specified as catalog */
SELECT FirstName, LastName FROM `psychic-valve-137816`.`Northwind`.`customers`
```

If the query contains unqualified table references then they are resolved this way:

1. If this property is set then the specified project is used.
2. Otherwise, if the [BillingProjectId](#) is set then that project is used.

3. Finally, if neither are set then the catalog from the first table in the query is used. In the above example the *psychic-valve-137816* project would be used to resolve any other unqualified tables that would appear in the query (for example, if a JOIN were added).

Note that the query is not consulted when QueryPassthrough is enabled. So you either must set the connection ProjectId and [DatasetId](#) or qualify each individual table; otherwise the SELECT query fails.

DatasetId

The DatasetId used to resolve unqualified tables.

Data Type

string

Default Value

""

Remarks

When a query refers to a table it can leave the dataset implicit, or qualify the dataset directly as the schema portion of the table:

```
/* Implicit, resolved against connection string */
SELECT FirstName, LastName FROM `customers`
/* Explicit, dataset specified as schema */
SELECT FirstName, LastName FROM `psychic-valve-137816`.`Northwind`.`customers`
```

If the query contains unqualified table references then they are resolved this way:

1. If this property is set then the specified dataset is used.
2. Otherwise the schema from the first table in the query is used. In the above example the *Northwind* dataset would be used to resolve any other unqualified tables that would appear in the query (for example, if a JOIN were added).

Note that the query is not consulted when QueryPassthrough is enabled. So you either must set the connection [ProjectId](#) and DatasetId or qualify each individual table; otherwise the SELECT query fails.

BillingProjectId

The ProjectId of the billing project for executing jobs.

Data Type

string

Default Value

""

Remarks

The ProjectId of the billing project for executing jobs. You can obtain the project Id in the Google APIs console: In the main menu, click API Project and copy the Id. For example: *psychic-valve-137816*. (Note that your domain name is not part of the Id.)

The billing project is determined based on the value of this property, the [ProjectId](#) and the query:

1. If this property is set then the specified project will be billed.
2. Otherwise, if the [ProjectId](#) is set then that project will be billed.
3. Finally, if neither are set then the catalog from the first table in the query will be used. For example, the below query would bill the *psychic-valve-137816* project.

```
SELECT FirstName, LastName FROM `psychic-valve-137816`.`Northwind`.`customers`
```

BigQuery

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

Property	Description
AllowLargeResultSets	Whether or not to allow large datasets to be stored in temporary tables for large datasets.
DestinationTable	This property determines where query results are stored in Google BigQuery.
UseQueryCache	Specifies whether to use Google BigQuery's built-in query cache.
PageSize	The number of results to return per page from Google BigQuery.
PollingInterval	This determines how long to wait in seconds, between checks to see if a job has completed.
AllowUpdatesWithoutKey	Whether or not to allow update without primary keys.
FilterColumns	Please set `AllowUpdatesWithoutKey` to true before you could use this property.
UseLegacySQL	Specifies whether to use BigQuery's legacy SQL dialect for this query. By default, Standard SQL will be used.

AllowLargeResultSets

Whether or not to allow large datasets to be stored in temporary tables for large datasets.

Data Type

bool

Default Value

true

Remarks

Whether or not to allow large datasets to be stored in temporary tables for large datasets.

DestinationTable

This property determines where query results are stored in Google BigQuery.

Data Type

string

Default Value

""

Remarks

Google BigQuery queries have a maximum amount of data they are allowed to return directly. If this limit is exceeded, then queries will fail with an error message like *Response too large to return*. When this option is enabled the response limit does not apply, because all query responses are stored in a Google BigQuery table before being returned.

This option is set differently depending upon whether your connection is using [UseLegacySQL](#) or not. By default this option is set using the standard SQL syntax:

DestinationTable=project-name.dataset-name.table-name

When [UseLegacySQL](#) is enabled, this option is set using the legacy table syntax:

DestinationTable=project-name:dataset-name.table-name

When using this option with multiple connections, make sure that each connection has its own destination table. Sharing a table between connections can lead to results getting lost because parallel queries can overwrite each others results.

UseQueryCache

Specifies whether to use Google BigQuery's built-in query cache.

Data Type

bool

Default Value

true

Remarks

Google BigQuery will cache the results of recent queries, and will use this cache for queries by default. Google BigQuery automatically updates the cache when a table is modified, so performance is generally better without any risk of queries returning stale data.

If this is set to false, the query is always run against the table directly.

PageSize

The number of results to return per page from Google BigQuery.

Data Type

string

Default Value

"100000"

Remarks

The pagesize can control the number of results returned per page from Google BigQuery. Setting a higher pagesize will cause more data to come back in a single HTTP request, but may take longer to execute. Setting a smaller pagesize will increase the number of HTTP

requests to get all the data, but is generally recommended to ensure timeout exceptions do not occur.

Note that this option does not have an effect if [UseStorageApi](#) is enabled and the queries being executed can be executed on the Storage API. See [StoragePageSize](#) for more information.

PollingInterval

This determines how long to wait in seconds, between checks to see if a job has completed.

Data Type

string

Default Value

"1"

Remarks

This only applies to queries which are stored to a table instead of streamed directly to the adapter. This applies in only three cases:

- [DestinationTable](#) is set.
- [AllowLargeResultSets](#) is true and the query takes longer than [Timeout](#) seconds.
- [UseStorageApi](#) is enabled and the query is complex.

This property determines how long to wait between checking whether or not the query's results are ready. Very large resultsets or complex queries may take longer to process, and a low polling interval may result in may unnecessary requests being made to check the query status.

AllowUpdatesWithoutKey

Whether or not to allow update without primary keys.

Data Type

bool

Default Value

false

Remarks

Whether or not to allow update without primary keys.

FilterColumns

Please set `AllowUpdatesWithoutKey` to true before you could use this property.

Data Type

string

Default Value

""

Remarks

Remember setting `AllowUpdatesWithoutKey` to true before you could use this property:

Set the property like this:

```
`filterColumns=col1[,col2[,col3]]`;
```

UseLegacySQL

Specifies whether to use BigQuery's legacy SQL dialect for this query. By default, Standard SQL will be used.

Data Type

bool

Default Value

false

Remarks

If set to true, the query will use BigQuery's Legacy SQL dialect to rebuild the query. If set to false, the query will use BigQuery's standard SQL: <https://cloud.google.com/bigquery/sql-reference/>.

When UseLegacySQL is set to false, the values of [AllowLargeResultSets](#) is ignored. The query will be run as if AllowLargeResultSets is true.

Storage API

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

Property	Description
UseStorageAPI	Specifies whether to use BigQuery's Storage API for bulk data reads.
UseArrowFormat	Specifies whether to use the Arrow format with BigQuery's Storage API.
StorageThreshold	The minimum number of rows a query must return to invoke the Storage API.
StoragePageSize	Specifies the page size to use for Storage API queries.

UseStorageAPI

Specifies whether to use BigQuery's Storage API for bulk data reads.

Data Type

bool

Default Value

true

Remarks

By default the adapter will use the Storage API instead of the default REST API. Depending upon the complexity of the query, the adapter may execute the query in one of two ways:

- Simple queries that read all columns from only one table, and have no extra clauses except LIMIT, are executed directly within the Storage API.
- All other queries are executed as a query job which writes to a temporary table. Once the query is complete, the results are read from the temporary table using the Storage API.

The BigQuery Storage API can read data faster and more efficiently than the REST API (accessible by setting this option to false), but is priced differently and requires extra OAuth permissions when using your own OAuth app. It also uses the separate [StoragePageSize](#) property instead of [PageSize](#).

The BigQuery REST API requires no extra permissions and uses standard pricing, but is slower than the Storage API.

UseArrowFormat

Specifies whether to use the Arrow format with BigQuery's Storage API.

Data Type

bool

Default Value

false

Remarks

This property only has an effect when [UseStorageApi](#) is enabled. When performing reads against the Storage API, the adapter can request data in different formats. By default it uses Avro but enabling this option makes it use Arrow.

This option should be enabled when working with time series data or other datasets that have many date, time, datetime or timestamp fields. For these datasets using Arrow can have noticeable improvements over using Avro. Otherwise Avro and Arrow read times are very close and switching between them is unlikely to make a significant difference.

StorageThreshold

The minimum number of rows a query must return to invoke the Storage API.

Data Type

string

Default Value

"100000"

Remarks

When the adapter receives a query too complex to be run directly in the Storage API, it creates a query job and uses the Storage API to read from the query results table. If the query job returns fewer than the number of rows provided in this option, then the results are returned directly and the Storage API is not used.

This value should be set between 1 and 100000. Higher values will use the Storage API only for large resultsets, but will be delayed by reading more results from the query job. Lower values will result in smaller delays but will use the Storage API for more queries.

Note that this option only has an effect if [UseStorageApi](#) is enabled and the queries being executed cannot be executed directly on the Storage API. Queries which run directly on Storage never create query jobs.

StoragePageSize

Specifies the page size to use for Storage API queries.

Data Type

string

Default Value

"10000"

Remarks

When [UseStorageApi](#) is enabled and the query being executed can be run on the Storage API, this option controls how many rows the adapter is allowed to buffer on the client.

A higher value will generally make queries faster at the expense of consuming more memory, while lower values will conserve memory but make queries slower.

Uploading

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

Property	Description
InsertMode	Specifies what kind of method to use when inserting data. By default streaming inserts are used.
WaitForBatchResults	Whether to wait for the job to complete when using the bulk upload API. Only active when InsertMode is set to Upload.
GCSBucket	Specifies the name of a GCS bucket to upload bulk data for staging.
GCSBucketFolder	Specifies the name of the folder in GCSBucket to upload bulk data for staging.
TempTableDataset	The prefix of the dataset that will contain temporary tables when performing bulk UPDATE or DELETE operations.

InsertMode

Specifies what kind of method to use when inserting data. By default streaming inserts are used.

Possible Values

Streaming, DML, Upload, GCSTaging

Data Type

string

Default Value

"Streaming"

Remarks

This section provides only a summary of the mechanisms that each of these modes use. Please see [Advanced Integrations](#) for more details on how to use each of these modes.

- **Streaming** uses the Google BigQuery streaming API (also called **insertAll**).
- **DML** uses the Google BigQuery query API to generate INSERT SQL statements which insert individual rows.
- **Upload** uses the Google BigQuery upload API to create a load job which copies the rows from temporary server-side storage.
- **GCSTaging** is similar to the Upload mode except that it uses your Google Cloud Storage account instead of public storage.

When [UseLegacySQL](#) is true only **Streaming** and **Upload** modes are allowed. The Legacy SQL dialect does not support **DML** statements.

WaitForBatchResults

Whether to wait for the job to complete when using the bulk upload API. Only active when InsertMode is set to Upload.

Data Type

bool

Default Value

true

Remarks

This property determines whether the adapter will wait for batch jobs to report their status. By default property is true and INSERT queries will complete only once Google BigQuery has finished executed them. When this property is false the INSERT query will complete as soon as a job is submitted for it.

The default mode is recommended for reliability:

1. Inserts will never fail silently. If the adapter does not wait for the job to finish, it will never receive an error if the job failed to execute.
2. If the insert batch size is small enough, the adapter may submit jobs quickly enough that it hits Google BigQuery's load job limits. This does not happen when waiting for batch results because the adapter will not allow more than one job to execute at the same time on the same connection.

You can disable this option to achieve lower delays when inserting, but you must also make sure to obey the Google BigQuery rate limits and check the status of each job to track their status and determine whether they have succeeded or failed.

GCSBucket

Specifies the name of a GCS bucket to upload bulk data for staging.

Data Type

string

Default Value

""

Remarks

Only applies when [InsertMode](#) is set to GCSStaging, and if that option is set to use staging then this option is required.

GCSBucketFolder

Specifies the name of the folder in GCSBucket to upload bulk data for staging.

Data Type

string

Default Value

""

Remarks

Only applies when [InsertMode](#) is set to GCSStaging. If not set the adapter defaults to writing to the root of the bucket.

TempTableDataset

The prefix of the dataset that will contain temporary tables when performing bulk UPDATE or DELETE operations.

Data Type

string

Default Value

"_CDataTempTableDataset"

Remarks

Internally bulk UPDATE and DELETE use Google BigQuery MERGE queries, which require creating a table to hold all the update operations. This option is used along with the target table's region to determine the name of the dataset where these temporary tables are created. Each region must have its own temporary dataset so that the temporary table and the MERGE table can be stored in the same project/dataset. This avoids unnecessary data transfer charges.

For example, the adapter would create a dataset called "_CDataTempTableDataset_US" for tables in the US region and a dataset called "_CDataTempTableDataset_asia_southeast_1" for tables in the Singapore region.

OAuth

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

Property	Description
InitiateOAuth	Set this property to initiate the process to obtain or refresh the OAuth access token when you connect.
OAuthClientId	The client Id assigned when you register your application with an OAuth authorization server.
OAuthClientSecret	The client secret assigned when you register your application with an OAuth authorization server.
OAuthAccessToken	The access token for connecting using OAuth.
OAuthSettingsLocation	The location of the settings file where OAuth values are saved when InitiateOAuth is set to GETANDREFRESH or REFRESH. Alternatively, this can be held in memory by specifying a value starting with memory://.
Scope	Identifies the Google API access that your application is requesting. Scopes are space-delimited.

OAuthVerifier	The verifier code returned from the OAuth authorization URL.
OAuthRefreshToken	The OAuth refresh token for the corresponding OAuth access token.
OAuthExpiresIn	The lifetime in seconds of the OAuth AccessToken.
OAuthTokenTimestamp	The Unix epoch timestamp in milliseconds when the current Access Token was created.

InitiateOAuth

Set this property to initiate the process to obtain or refresh the OAuth access token when you connect.

Possible Values

OFF, GETANDREFRESH, REFRESH

Data Type

string

Default Value

"OFF"

Remarks

The following options are available:

1. **OFF**: Indicates that the OAuth flow will be handled entirely by the user. An OAuthAccessToken will be required to authenticate.
2. **GETANDREFRESH**: Indicates that the entire OAuth Flow will be handled by the adapter. If no token currently exists, it will be obtained by prompting the user via the browser. If a token exists, it will be refreshed when applicable.
3. **REFRESH**: Indicates that the adapter will only handle refreshing the

OAuthAccessToken. The user will never be prompted by the adapter to authenticate via the browser. The user must handle obtaining the OAuthAccessToken and OAuthRefreshToken initially.

OAuthClientId

The client Id assigned when you register your application with an OAuth authorization server.

Data Type

string

Default Value

""

Remarks

As part of registering an OAuth application, you will receive the OAuthClientId value, sometimes also called a consumer key, and a client secret, the [OAuthClientSecret](#).

OAuthClientSecret

The client secret assigned when you register your application with an OAuth authorization server.

Data Type

string

Default Value

""

Remarks

As part of registering an OAuth application, you will receive the [OAuthClientId](#), also called a consumer key. You will also receive a client secret, also called a consumer secret. Set the client secret in the [OAuthClientSecret](#) property.

OAuthAccessToken

The access token for connecting using OAuth.

Data Type

string

Default Value

""

Remarks

The [OAuthAccessToken](#) property is used to connect using OAuth. The [OAuthAccessToken](#) is retrieved from the OAuth server as part of the authentication process. It has a server-dependent timeout and can be reused between requests.

The access token is used in place of your user name and password. The access token protects your credentials by keeping them on the server.

OAuthSettingsLocation

The location of the settings file where OAuth values are saved when InitiateOAuth is set to GETANDREFRESH or REFRESH. Alternatively, this can be held in memory by specifying a value starting with memory://.

Data Type

string

Default Value

"%APPDATA%\\CData\\GoogleBigQuery Data Provider\\OAuthSettings.txt"

Remarks

When [InitiateOAuth](#) is set to GETANDREFRESH or REFRESH, the adapter saves OAuth values to avoid requiring the user to manually enter OAuth connection properties and allowing the credentials to be shared across connections or processes.

Alternatively to specifying a file path, memory storage can be used instead. Memory locations are specified by using a value starting with 'memory:/' followed by a unique identifier for that set of credentials (ex: memory://user1). The identifier can be anything you choose but should be unique to the user. Unlike with the file based storage, you must manually store the credentials when closing the connection with memory storage to be able to set them in the connection when the process is started again. The OAuth property values can be retrieved with a query to the sys_connection_props system table. If there are multiple connections using the same credentials, the properties should be read from the last connection to be closed.

If left unspecified, the default location is "%APPDATA%\\CData\\GoogleBigQuery Data Provider\\OAuthSettings.txt" 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

Scope

Identifies the Google API access that your application is requesting. Scopes are space-delimited.

Data Type

string

Default Value

""

Remarks

By default the adapter requests only the scope <https://www.googleapis.com/auth/bigquery>, but in some cases more scopes are required. For example, when reading data using an external table connected to Google Drive, the additional scope <https://www.googleapis.com/auth/drive> is also required.

OAuthVerifier

The verifier code returned from the OAuth authorization URL.

Data Type

string

Default Value

""

Remarks

The verifier code returned from the OAuth authorization URL. This can be used on systems where a browser cannot be launched such as headless systems.

Authentication on Headless Machines

See to obtain the [OAuthVerifier](#) value.

Set [OAuthSettingsLocation](#) along with [OAuthVerifier](#). When you connect, the adapter exchanges the [OAuthVerifier](#) for the OAuth authentication tokens and saves them, encrypted, to the specified file. Set [InitiateOAuth](#) to GETANDREFRESH automate the exchange.

Once the OAuth settings file has been generated, you can remove [OAuthVerifier](#) from the connection properties and connect with [OAuthSettingsLocation](#) set.

To automatically refresh the OAuth token values, set [OAuthSettingsLocation](#) and additionally set [InitiateOAuth](#) to REFRESH.

OAuthRefreshToken

The OAuth refresh token for the corresponding OAuth access token.

Data Type

string

Default Value

""

Remarks

The [OAuthRefreshToken](#) property is used to refresh the [OAuthAccessToken](#) when using OAuth authentication.

OAuthExpiresIn

The lifetime in seconds of the OAuth AccessToken.

Data Type

string

Default Value

""

Remarks

Pair with OAuthTokenTimestamp to determine when the AccessToken will expire.

OAuthTokenTimestamp

The Unix epoch timestamp in milliseconds when the current Access Token was created.

Data Type

string

Default Value

""

Remarks

Pair with OAuthExpiresIn to determine when the AccessToken will expire.

JWT OAuth

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

Property	Description
OAuthJWTCert	The JWT Certificate store.
OAuthJWTCertType	The type of key store containing the JWT Certificate.

OAuthJWTCertPassword	The password for the OAuth JWT certificate.
OAuthJWTCertSubject	The subject of the OAuth JWT certificate.
OAuthJWTIssuer	The issuer of the Java Web Token.
OAuthJWTSubject	The user subject for which the application is requesting delegated access.

OAuthJWTCert

The JWT Certificate store.

Data Type

string

Default Value

""

Remarks

The name of the certificate store for the client certificate.

The [OAuthJWTCertType](#) field specifies the type of the certificate store specified by [OAuthJWTCert](#). If the store is password protected, specify the password in [OAuthJWTCertPassword](#).

[OAuthJWTCert](#) is used in conjunction with the [OAuthJWTCertSubject](#) field in order to specify client certificates. If [OAuthJWTCert](#) has a value, and [OAuthJWTCertSubject](#) is set, a search for a certificate is initiated. Please refer to the [OAuthJWTCertSubject](#) field for details.

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 (i.e. PKCS12 certificate store).

OAuthJWTCertType

The type of key store containing the JWT Certificate.

Possible Values

USER, MACHINE, PFXFILE, PFXBLOB, JKSFIL, JKSBLOB, PEMKEY_FILE, PEMKEY_BLOB, PUBLIC_KEY_FILE, PUBLIC_KEY_BLOB, SSHPUBLIC_KEY_FILE, SSHPUBLIC_KEY_BLOB, P7BFILE, PPKFILE, XMLFILE, XMLBLOB, GOOGLEJSON, GOOGLEJSONBLOB

Data Type

string

Default Value

"GOOGLEJSON"

Remarks

This property can take one of the following values:

USER	For Windows, this specifies that the certificate store is a certificate store owned by the current user. <i>Note:</i> This store type is not available in Java.
MACHINE	For Windows, this specifies that the certificate store is a machine store. <i>Note:</i> 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. <i>Note:</i> this store type is only available in Java.
JKSBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in Java key store (JKS) format. <i>Note:</i> 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 PPK (PuTTY Private Key).
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.
GOOGLEJSON	The certificate store is the name of a JSON file containing the service account information. Only valid when connecting to a Google service.
GOOGLEJSONBLOB	The certificate store is a string that contains the service account JSON. Only valid when connecting to a Google service.

OAuthJWTCertPassword

The password for the OAuth JWT 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 in order to open the certificate store.

This is not required when using the GOOGLEJSON [OAuthJWTCertType](#). Google JSON keys are not encrypted.

OAuthJWTCertSubject

The subject of the OAuth JWT 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 instance "CN=www.server.com, OU=test, C=US, E=support@cdata.com". Common fields and their meanings are displayed 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.

OAuthJWTIssuer

The issuer of the Java Web Token.

Data Type

string

Default Value

""

Remarks

The issuer of the Java Web Token. This is typically either the Client Id or Email Address of the OAuth Application.

This is not required when using the GOOGLEJSON [OAuthJWTCertType](#). Google JSON keys contain a copy of the issuer account.

OAuthJWTSubject

The user subject for which the application is requesting delegated access.

Data Type

string

Default Value

""

Remarks

The user subject for which the application is requesting delegated access. Typically, the user account name or email address.

SSL

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

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

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----- MIICHTCCAe4CAQAwDQYJKoZIhvw... ...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)	e0a0b5a1529c58a1e9e09828d 70e4
The SHA1 Thumbprint (hex values can also be either space or colon separated)	34a029926a081b2ec14b4a3d904f 801cbb150d

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 Google BigQuery 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.
RefreshViewSchemas	Allows the provider to determine up-to-date view schemas automatically.

ShowTableDescriptions	Controls whether table descriptions are returned via the platform metadata APIs and sys_tables / sys_views.
PrimaryKeyIdentifiers	Set this property to define primary keys.
AllowedTableTypes	Specifies what kinds of tables will be visible.
FlattenObjects	Determines whether the provider flattens STRUCT fields into top-level columns.

Location

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

Data Type

string

Default Value

"%APPDATA%\\CData\\GoogleBigQuery 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%\\CData\\GoogleBigQuery 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

RefreshViewSchemas

Allows the provider to determine up-to-date view schemas automatically.

Data Type

bool

Default Value

true

Remarks

When using BigQuery views, BigQuery stores a copy of the view schema with the view itself. However, these stored view schemas are not updated when the tables used by the view change. This means that the stored view schema can easily become out of date and cause queries using the view to fail.

By default, the adapter will not use the stored view schema and will instead query the view to determine the available columns. This guarantees that the schema will be up to date although it requires the adapter to start a query job.

You can disable this option to force the adapter to use the stored view schemas. This prevents the adapter from running any queries when getting a view schema, but also means that queries using the view will fail if the schema is out of date.

ShowTableDescriptions

Controls whether table descriptions are returned via the platform metadata APIs and sys_tables / sys_views.

Data Type

bool

Default Value

false

Remarks

By default table descriptions are not shown, since the Google BigQuery API requires an extra request beyond what is usually required for reading tables.

Enabling this option will show table descriptions, but will cost an extra API request for every table when a table list is fetched. This can slow down metadata operations on large datasets.

PrimaryKeyIdentifiers

Set this property to define primary keys.

Data Type

string

Default Value

""

Remarks

Google BigQuery does not natively support primary keys, but for certain DML operations or database tools you may need to define them. By default this option is disabled and no tables will have primary keys except for the ones defined in schema files (if you set [Location](#)).

Primary keys are defined using a list of rules which match tables and provide a list of key columns. For example, **PrimaryKeyIdentifiers="*=key;transactions=tx_date,tx_serial;user_comments="** has three rules separated by semicolons:

1. The first rule ***=key** means that every table without a more specific rule will have one primary key column called *key*. Tables that do not have a *key* column will not have any primary keys.
2. The second rule **transactions=tx_date,tx_serial** means that the *transactions* table will have the two primary key columns *tx_date* and *tx_serial*. If any of those columns are missing from the table then they will be ignored.
3. The third rule **user_comments=** means that the *user_comments* table will have no primary keys. The only use that empty key lists have is in overriding the default rule. If there is no default rule present then the only tables with primary keys would be the ones explicitly listed.

Note that the table names can include just the table, the table and dataset or the table, dataset and project. Both column and table names may be quoted using SQL quotes:

```
/* Rules with just table names use the connection ProjectId (or
DataProjectId) and DatasetId.
   All these rules refer to the same table with a connection where
ProjectId=someProject;DatasetId=someDataset */
someTable=a,b,c
someDataset.someTable=a,b,c
someProject.someDataset.someTable=a,b,c
/* Any table or column name may be quoted */
'someProject`.`someDataset`.`someTable`='a',[b],"c"
```

AllowedTableTypes

Specifies what kinds of tables will be visible.

Data Type

string

Default Value

"TABLE,EXTERNAL,VIEW,MATERIALIZED_VIEW"

Remarks

This option is a comma-separated list of the table type values that the adapter displays. Any table-like or view-like entity that doesn't have a matching type will not be reported when listing tables.

- **TABLE** Standard tables
- **EXTERNAL** Read-only table stored on another service (like GCS or Drive)
- **SNAPSHOT** A read-only table that preserves the data of another table at a specific point in time
- **VIEW** Standard views
- **MATERIALIZED_VIEW** A view that is recalculated and cached each time its base table changes

For example, to restrict the adapter to listing only simple tables and views, this option would be set to **TABLE,VIEW**

FlattenObjects

Determines whether the provider flattens STRUCT fields into top-level columns.

Data Type

bool

Default Value

true

Remarks

By default the adapter reports each field in a STRUCT column as its own column while the STRUCT column itself is hidden. This process is recursively applied to nested STRUCT values. For example, if the following table is defined in Google BigQuery then the adapter reports 3 columns: **location.coords.lat**, **location.coords.lon** and **location.country**:

```
CREATE TABLE t(location STRUCT<coords STRUCT<lat FLOAT64, lon FLOAT64>,
country STRING>);
```

If this property is disabled, then the top-level STRUCT is not expanded and is left as its own column. The value of this column is reported as a JSON aggregate. In the above example, the adapter reports only the **location** column when flattening is disabled.

Miscellaneous

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

Property	Description
StorageTimeout	How long a Storage API connection must remain idle before the provider reconnects.
AllowAggregateParameters	Allows raw aggregates to be used in parameters when QueryPassthrough is enabled.
ApplicationName	An application name in the form application/version. For example, AcmeReporting/1.0.
AuditLimit	The maximum number of rows which will be stored within an audit table.
AuditMode	What provider actions should be recorded to audit tables.
BigQueryOptions	A comma separated list of Google BigQuery options.
GenerateSchemaFiles	Indicates the user preference as to when schemas should be generated and saved.
MaximumBillingTier	The MaximumBillingTier is a positive integer that serves as a multiplier of the basic price per TB. For example, if you set MaximumBillingTier to 2, the maximum cost for that query will be 2x basic price per TB.
MaximumBytesBilled	Limits how many bytes BigQuery will allow a job to consume before it is cancelled.
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 Google BigQuery from the provider.
TableSamplePercent	This determines what percent of a table is sampled with the TABLESAMPLE operator.
Timeout	The value in seconds until the timeout error is thrown, canceling the operation.

StorageTimeout

How long a Storage API connection must remain idle before the provider reconnects.

Data Type

string

Default Value

"60"

Remarks

Google BigQuery and many proxies/firewalls restrict the amount of time that idle connections stay alive before they are forcibly closed. This can be a problem when using the Storage API because the adapter may stream data faster than it can be consumed. While the consumer is catching up, the adapter does not use its connection and it may be closed by the next time the adapter uses it.

To avoid this the adapter will automatically close and reopen the connection if it has been idle for too long. This property controls how many seconds the connection has to be idle for the adapter to reset it. To disable these resets this property can also set to 0 or a negative value.

AllowAggregateParameters

Allows raw aggregates to be used in parameters when QueryPassthrough is enabled.

Data Type

bool

Default Value

false

Remarks

This option affects how string parameters are handled when using direct queries through [QueryPassthrough](#). For example, consider this query:

```
INSERT INTO proj.data.tbl(x) VALUES (@x)
```

By default, this option is disabled and string parameters are quoted and escaped into SQL strings. That means that any value can be safely used as a string parameter, but it also means that parameters cannot be used as raw aggregate values:

```
/*
 * If @x is set to: test value ' contains quote
 *
 * Result is a valid query
 */
INSERT INTO proj.data.tbl(x) VALUES ('test value \' contains quote')
/*
 * If @x is set to: ['valid', ('aggregate', 'value')]
 *
 * Result contains string instead of aggregate:
 */
INSERT INTO proj.data.tbl(x) VALUES ('[\'valid\', (\'aggregate\',
\'value\')])')
```

When this option is enabled, string parameters are inserted directly into the query. This means that raw aggregates can be used as parameters, but it also means that all simple strings must be escaped:

```

/*
 * If @x is set to: test value ' contains quote
 *
 * Result is an invalid query
 */
INSERT INTO proj.data.tbl(x) VALUES (test value ' contains quote)
/*
 * If @x is set to: ['valid', ('aggregate', 'value')]
 *
 * Result is an aggregate
 */
INSERT INTO proj.data.tbl(x) VALUES (['valid', ('aggregate', 'value')])

```

ApplicationName

An application name in the form application/version. For example, AcmeReporting/1.0.

Data Type

string

Default Value

""

Remarks

The adapter identifies itself to BigQuery using a Google partner User-Agent header. The first part of the User-Agent is fixed and identifies the client as a specific build of the CData adapter. The last portion reports the specific application using the adapter.

AuditLimit

The maximum number of rows which will be stored within an audit table.

Data Type

string

Default Value

"1000"

Remarks

When auditing is enabled with the [AuditMode](#) option, this property is used to determine how many rows will be allowed in the audit table at once.

By default this property is 1000, meaning that only the 1000 most recent audit events will be available within the audit table.

This property can also be set to -1, which places no limits on the size of the audit table. In this mode, the audit table should be periodically cleared to prevent the adapter from using excessive memory.

```
DELETE FROM AuditJobs#TEMP
```

AuditMode

What provider actions should be recorded to audit tables.

Data Type

string

Default Value

""

Remarks

The adapter can record certain internal actions taken when it runs queries. For each of those actions listed in this option, the adapter will create a temporary audit table which logs when the action took place, what query caused the action and any other relevant information.

By default this option is set to 'none' and the adapter does not record any audit information. This option can also be set to a comma-separated list of the following actions:

Mode Name	Audit Table	Description	Columns
start-jobs	AuditJobs#TEMP	Records all jobs started by the adapter	Timestamp,Query,ProjectId,Location,Job Id

Refer to [AuditLimit](#) for more information on how to limit the size of these tables.

BigQueryOptions

A comma separated list of Google BigQuery options.

Data Type

string

Default Value

""

Remarks

A list of Google BigQuery options:

Option	Description
gbqoImplicitJoinAsUnion	This option will prevent the driver from converting an IMPLICIT JOIN into a CROSS JOIN as expected by SQL92. Instead, it will leave it as an IMPLICIT JOIN, which Google BigQuery will execute as a UNION ALL.

GenerateSchemaFiles

Indicates the user preference as to when schemas should be generated and saved.

Possible Values

Never, OnUse, OnStart, OnCreate

Data Type

string

Default Value

"Never"

Remarks

This property outputs schemas to .rsd files in the path specified by [Location](#).

Available settings are the following:

- Never: A schema file will never be generated.
- OnUse: A schema file will be generated the first time a table is referenced, provided the schema file for the table does not already exist.
- OnStart: A schema file will be generated at connection time for any tables that do not currently have a schema file.
- OnCreate: A schema file will be generated by when running a CREATE TABLE SQL query.

Note that if you want to regenerate a file, you will first need to delete it.

Generate Schemas with SQL

When you set [GenerateSchemaFiles](#) to **OnUse**, the adapter generates schemas as you execute SELECT queries. Schemas are generated for each table referenced in the query.

When you set [GenerateSchemaFiles](#) to **OnCreate**, schemas are only generated when a CREATE TABLE query is executed.

Generate Schemas on Connection

Another way to use this property is to obtain schemas for every table in your database when you connect. To do so, set [GenerateSchemaFiles](#) to **OnStart** and connect.

MaximumBillingTier

The MaximumBillingTier is a positive integer that serves as a multiplier of the basic price per TB. For example, if you set MaximumBillingTier to 2, the maximum cost for that query will be 2x basic price per TB.

Data Type

string

Default Value

""

Remarks

Limits the billing tier for this job. Queries that have resource usage beyond this tier will fail (without incurring a charge). If unspecified, this will be set to your project default. If your query is too compute intensive for BigQuery to complete at the standard per TB pricing tier, BigQuery returns a billingTierLimitExceeded error and an estimate of how much the query would cost. To run the query at a higher pricing tier, pass a new value for maximumBillingTier as part of the query request. The maximumBillingTier is a positive integer that serves as a multiplier of the basic price per TB. For example, if you set maximumBillingTier to 2, the maximum cost for that query will be 2x basic price per TB.

MaximumBytesBilled

Limits how many bytes BigQuery will allow a job to consume before it is cancelled.

Data Type

string

Default Value

""

Remarks

When this value is provided, all jobs will use this value as their default billing cap. If a job uses more than this many bytes, BigQuery will cancel it and it will not be billed. By default there is no cap and all jobs will be billed for however many bytes they consume.

This only has an effect when using [DestinationTable](#) or when using the InsertJob stored procedure. BigQuery does not allow standard query jobs to have byte limits.

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 Google BigQuery 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.

TableSamplePercent

This determines what percent of a table is sampled with the TABLESAMPLE operator.

Data Type

string

Default Value

""

Remarks

This option can be set to make the adapter use the TABLESAMPLE for each table referenced by a query. The value determines what percent is provided to the PERCENT clause. That clause will only be generated if this property's value is above zero.

```
-- Input SQL
SELECT * FROM `tbl`
-- Generated Google BigQuery SQL when TableSamplePercent=10
SELECT * FROM `tbl` TABLESAMPLE SYSTEM (10 PERCENT)
```

This option is subject to a few limitations:

- It is applied during query conversion and has no effect when [QueryPassthrough](#) is set.
- More rows may be returned than expected due to how the server implements TABLESAMPLE. Please see the Google BigQuery documentation for more information.
- TABLESAMPLE is not supported on views. If a view is queried in sampling mode, the adapter will omit the TABLESAMPLE clause for the view.

Timeout

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

Data Type

string

Default Value

"300"

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.

Third Party Copyrights

Google BigQuery Adapter Limitations

Following are some of the limitations of the Google BigQuery adapter:

1. For clearing cache, the following two conditions must be met:

- Rows can be removed from Google BigQuery tables after 90 minutes
- The `cache_status`/`cache_tracking` tables should be pointed to a different datasource, such as PostgreSQL. It is recommended to choose the “Use a different data source” option, while defining the Cache status and Tracking tables.

2. In the caching configuration, while choosing the table for multi-table caching, the fields “Table Schema” and “Table Catalog” are not optional for Google BigQuery data source.

3. CREATE/DROP indexes for tables is not supported. You must uncheck the option "Drop indexes before load and create indexes after load" in the caching configuration before initiating caching.

4. In order to use the Select/Insert native cache loading functions, choose the “Enable Native Data Loading” in the Overrides panel and uncheck the “Use Streaming Inserts” option in the datasource configuration.

5. Native cache loading with Insert/Select is not supported for synthesized views.

6. The following datatypes are not supported by Google BigQuery. Tables with columns of these datatypes cannot be cached:

- BLOB
- CLOB
- BT_INT_LONG

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.

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 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.

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, Two-Second Advantage, TIBCO Spotfire, TIBCO ActiveSpaces, TIBCO Spotfire Developer, TIBCO EMS, TIBCO Spotfire Automation Services, TIBCO Enterprise Runtime for R, TIBCO Spotfire Server, TIBCO Spotfire Web Player, TIBCO Spotfire Statistics Services, S-PLUS, and TIBCO Spotfire 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.