



TIBCO® Data Virtualization

Snowflake Adapter Guide

Version 8.7.0 | October 2023

Contents

Contents	2
Snowflake Adapter	4
Getting Started	4
Basic Tab	5
Logging	11
Creating a Custom OAuth App	13
Changelog	13
Advanced Features	19
User Defined Views	20
SSL Configuration	23
Firewall and Proxy	23
Query Processing	24
Logging	25
SQL Compliance	28
SELECT Statements	29
SELECT INTO Statements	197
INSERT Statements	197
UPDATE Statements	198
DELETE Statements	199
EXECUTE Statements	199
PIVOT and UNPIVOT	200
Data Model	201
Stored Procedures	202
Connection String Options	207
Authentication	214
Connection	220
Azure Authentication	221

SSO	222
KeyPairAuth	224
OAuth	229
SSL	239
Firewall	241
Proxy	244
Logging	251
Schema	251
Miscellaneous	254
Snowflake Adapter Limitations	269
TIBCO Product Documentation and Support Services	271
How to Access TIBCO Documentation	271
How to Contact TIBCO Support	272
Release Version Support	272
How to Join TIBCO Community	273
Legal and Third-Party Notices	274

Snowflake Adapter

Snowflake Version Support

The adapter enables standards-based access to all Snowflake editions. You can authenticate with a Snowflake user, SSO, or SSL client authentication. After authenticating, you can execute standard SQL queries to Snowflake tables or set [QueryPassthrough](#) to use any of the available Snowflake SQL syntax. For example, you can use extended projection to project columns over semistructured data.

SQL Compliance

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

Getting Started

Connecting to Snowflake

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

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

1. Unzip the `tdv.snowflake.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.snowflake/tdv.snowflake.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 Snowflake
```

Basic Tab

Connecting to Snowflake

In addition to providing authentication (see below) set the following properties to connect to a Snowflake database:

- **Url:** Both AWS and Azure instances are supported. For example:
 - AWS: `https://myaccount.region.snowflakecomputing.com`
 - Azure: `https://myaccount.region.azure.snowflakecomputing.com`

Account is only required if your Url does not conform to the usual syntax containing the account name at the beginning. Snowflake provides the Account name needed in this case.

Optionally, you can set Database and Schema to restrict the tables and views returned by the adapter.

Authenticating to Snowflake

The adapter supports Snowflake user authentication, federated authentication, and SSL client authentication. To authenticate, set User and Password, and select the authentication method in the AuthScheme property.

Passwords

Set User and Password to a Snowflake user and set AuthScheme to `PASSWORD`.

Key Pairs

The adapter allows you to authenticate using key pair authentication by creating a secure token with the private key defined for your user account. To connect with this method, set AuthScheme to PRIVATEKEY and set the following values:

- **User**: The user account to authenticate as.
- **PrivateKey**: The private key used for the user such as the path to the .pem file containing the private key.
- **PrivateKeyType**: The type of key store containing the private key such as PEMKEY_FILE, PFXFILE, etc.
- **PrivateKeyPassword**: The password for the specified private key.

Okta

Set the AuthScheme to **Okta**. The following connection properties are used to connect to Okta:

- **User**: Set this to the Okta user.
- **Password**: Set this to Okta password for the user.
- **MFAPasscode** (optional): Set this to the OTP code that was sent to your device. This property should be used only when the MFA is required for OKTA sign on.

The following SSOProperties are needed to authenticate to Okta:

- **Domain**: Set this to the OKTA org domain name.
- **MFAType** (optional): Set this to the multi-factor type. This property should be used only when the MFA is required for OKTA sign on. This property accepts one of the following values:
 - OKTAVerify
 - Email
 - SMS
- **APIToken** (optional): Set this to the API Token that the customer created from the Okta org. It should be used when authenticating a user via a trusted application or proxy that overrides OKTA client request context. In most contexts, it is not needed.

The following is an example connection string:

```
AuthScheme=OKTA;User=username;Password=password;Url='https://myaccount.region.snowflakecomputing.com';Warehouse=My_warehouse;SSO
Properties='Domain=https://cdata-okta.okta.com';
```

The following is an example connection string for OKTA MFA:

```
AuthScheme=OKTA;User=username;Password=password;MFAPasscode=8111461;Url='https://myaccount.region.snowflakecomputing.com';Warehouse=My_warehouse;SSO
Properties='Domain=https://cdata-okta.okta.com;MFAType=OktaVerify;';
```

AzureAD

Set the AuthScheme to **AzureAD**. The following connection properties are used to connect to AzureAD:

- User: Set this to your AD user. When connecting, your browser will open allowing you to login to Azure AD to complete the authentication.

The following is an example connection string for AzureAD:

```
AuthScheme=AzureAD;Url=https://myaccount.region.snowflakecomputing.com;User=user@domain.onmicrosoft.com;
```

PingFederate

Set the AuthScheme to **PingFederate**. The following connection properties are used to connect to PingFederate:

- User: Set this to your PingFederate user, the user should be added to PingFederate Data Stores. When connecting, your browser will open allowing you to login to PingFederate to complete the authentication.

The following is an example connection string for PingFederate(Assuming that Active Directory is used as a Data Store):

```
AuthScheme=PingFederate;Url=https://myaccount.region.snowflakecomputing.com;User=myuser@mydomain;Account=myaccount;Warehouse=mywarehouse;
```

OAuth

To authenticate with OAuth, set the AuthScheme to OAuth. You can authenticate by [Creating a Custom OAuth App](#) to obtain the OAuthClientId, OAuthClientSecret, and

CallbackURL connection properties.

Desktop Apps

This section describes desktop authentication using the credentials for your custom OAuth app. See [Creating a Custom OAuth App](#) for more information.

Get an OAuth Access Token

After setting the following, you are ready to connect:

- OAuthClientId: Set to the Client ID in your OAuth Integration settings.
- OAuthClientSecret: Set to the Client Secret in your OAuth Integration settings.
- CallbackURL: Set to the Redirect URL in your OAuth Integration settings.
- InitiateOAuth: Set to **GETANDREFRESH**. You can use InitiateOAuth to avoid repeating the OAuth exchange and manually setting the OAuthAccessToken.

When you connect, the adapter opens the OAuth endpoint in your default browser. Log in and grant permissions to the application. The adapter then completes the following OAuth process:

1. Extracts the access token from the callback URL and authenticates requests.
2. Obtains a new access token when the old one expires.
3. Saves OAuth values in OAuthSettingsLocation to be persisted across connections.

Manually Get an OAuth Access Token

Set the following connection properties to obtain the OAuthAccessToken:

- InitiateOAuth: Set to **OFF**.
- OAuthClientId: Set to the Client ID in your OAuth Integration settings.
- OAuthClientSecret: Set to the Client Secret in your OAuth Integration settings.

You can then call stored procedures to complete the OAuth exchange:

1. Call the [GetOAuthAuthorizationUrl](#) stored procedure. Set the CallbackURL input to the Redirect URI you specified in your app settings. The stored procedure returns the URL to the OAuth endpoint and the PKCEVerifier.
2. Open the URL, log in, and authorize the application. You are redirected back to the callback URL.

3. Call the [GetOAuthAccessToken](#) stored procedure. Set the CallbackURL input to the Redirect URI you specified in your app settings. Set the PKCEVerifier input to the value of the PKCEVerifier retrieved from the first step.

Headless Machines

To configure the driver to 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 these two options:
 - Option 1: Obtain the OAuthVerifier value as described in "Obtain and Exchange a Verifier Code" below.
 - Option 2: Install the adapter on another machine and transfer the OAuth authentication values after you authenticate through the usual browser-based flow, as described in "Transfer OAuth Settings" below.
2. Then configure the adapter to automatically refresh the access token from the headless machine.

Option 1: Obtain and Exchange a Verifier Code

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

See [Creating a Custom OAuth App](#) for a procedure. This section describes the procedure to authenticate and connect to data.

To obtain the verifier code, set the following properties on the headless machine:

- InitiateOAuth: Set to **OFF**.
- OAuthClientId: Set to the Client ID in your OAuth Integration settings.
- OAuthClientSecret: Set to the Client Secret in your OAuth Integration settings.

Next, authenticate from another machine and obtain the OAuthVerifier connection property:

1. Call the [GetOAuthAuthorizationUrl](#) stored procedure. Set the CallbackURL input to the Redirect URI you specified in your app settings. The stored procedure returns the

URL to the OAuth endpoint and the PKCEVerifier.

2. Open the returned URL in a browser. 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 and the value of the PKCEVerifier. You need to set the value of the Verifier in the OAuthVerifier connection property and set the value of the PKCEVerifier in the PKCEVerifier connection property.

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

- OAuthClientId: Set to the Client ID in your OAuth Integration settings.
- OAuthClientSecret: Set to the Client Secret in your OAuth Integration settings.
- OAuthVerifier: Set to the verifier code.
- PKCEVerifier: Set to the PKCE verifier code.
- OAuthSettingsLocation: Set to persist the encrypted OAuth authentication values to the specified file.
- InitiateOAuth: Set to **REFRESH**.

Connect to Data

After the OAuth settings file is generated, set the following properties to connect to data:

- OAuthSettingsLocation: Set to the file containing the encrypted OAuth authentication values. Make sure this file gives read and write permissions to the provider to enable the automatic refreshing of the access token.
- InitiateOAuth: Set to **REFRESH**.

Option 2: Transfer OAuth Settings

To install the adapter on another machine, authenticate, and then transfer the resulting OAuth values:

1. On a second machine, install the adapter and connect with the following properties set:
 - OAuthSettingsLocation: Set to a writable text file.
 - InitiateOAuth: Set to **GETANDREFRESH**.
 - OAuthClientId: Set to the Client ID in your app settings.

- OAuthClientSecret: Set to the Client Secret in your app settings.
 - CallbackURL: Set to the Callback URL in your app settings.
2. Test the connection to authenticate. The resulting authentication values are written, encrypted, to the path specified by OAuthSettingsLocation. 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 to **REFRESH**.
 - OAuthSettingsLocation: Set 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.

Configuring Access Control

If the authenticating user maps to a system-defined role, specify it in the RoleName property.

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 Snowflake 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

If you do not have access to the user name and password or do not wish to require them, you can use OAuth authentication. Snowflake uses the OAuth, which requires the authenticating user to interact with Snowflake via the browser. The adapter facilitates the OAuth exchange in various ways, as described in this section.

Create an OAuth Integration

To register your client, create an integration. An integration is a Snowflake object that provides an interface between Snowflake and third-party services, such as a client that supports OAuth.

Note: Only account administrators (users with the ACCOUNTADMIN role) or a role with the global CREATE INTEGRATION privilege can execute this SQL command

Create an integration using the CREATE SECURITY INTEGRATION command. For example:

```
create security integration MYINT
  type = oauth
  enabled = true
  oauth_client = custom
  oauth_client_type = 'CONFIDENTIAL'
  oauth_redirect_uri = 'http://localhost.com'
  oauth_issue_refresh_tokens = true
  oauth_refresh_token_validity = 86400
  blocked_roles_list = ('SYSADMIN')
  oauth_client_rsa_public_key = '
MIIBI
::
';
```

Changelog

General Changes

Date	Build Number	Change Type	Description
12/14/2022	8383	General	Changed <ul style="list-style-type: none"> Added the Default column to the sys_procedureparameters table.
09/30/2022	8308	General	Changed <ul style="list-style-type: none"> Added the IsPath column to the sys_procedureparameters table.
08/17/2022	8264	General	Changed <ul style="list-style-type: none"> We now support handling the keyword "COLLATE" as standard function name as well.
06/29/2022	8215	Snowflake	Changed <ul style="list-style-type: none"> Switched to using the LOCAL TEMPORARY table to support the MERGE statement. This saves cost against the Snowflake server.
03/31/2022	8125	Snowflake	Added <ul style="list-style-type: none"> Added support for using the GET and PUT commands in the SQL statement. This can be

			used to upload/download files for Azure/S3/GCP.
09/02/2021	7915	General	Added <ul style="list-style-type: none"> Added support for the STRING_SPLIT table-valued function in the CROSS APPLY clause.
08/07/2021	7889	General	Changed <ul style="list-style-type: none"> Added the KeySeq column to the sys_foreignkeys table.
08/06/2021	7888	General	Changed <ul style="list-style-type: none"> Added the new sys_primarykeys system table.
07/23/2021	7874	General	Changed <ul style="list-style-type: none"> Updated the Literal Function Names for relative date/datetime functions. Previously relative date/datetime functions resolved to a different value when used in the projection vs the predicate. I.e: SELECT LAST_MONTH() AS lm, Col FROM Table WHERE Col > LAST_MONTH(). Formerly the two LAST_MONTH() methods would resolve to different datetimes. Now they will match.

			<ul style="list-style-type: none"> As a replacement for the previous behavior, the relative date/datetime functions in the criteria may have an 'L' appended to them. ie: WHERE col > L_LAST_MONTH(). This will continue to resolve to the same values that previously were calculated in the criteria. Note that the "L_" prefix will only work in the predicate - it not available for the projection.
07/14/2021	7865	Snowflake	Changed <ul style="list-style-type: none"> Updated the protocol version to 3.13.5, synced with the latest official JDBC driver.
07/12/2021	7863	Snowflake	Added <ul style="list-style-type: none"> Added support for prepared queries via the new connection property, AllowPreparedStatement. Set this property to true to support prepared queries.
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.
04/23/2021	7785	General	<p>Added</p> <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	<p>Changed</p> <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	<p>Added</p> <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 <p>Changed</p> <ul style="list-style-type: none"> Reduced the length to 255 for varchar primary key and foreign key

			<p>columns.</p> <ul style="list-style-type: none"> 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	<p>Changed</p> <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
04/12/2021	7772	Snowflake	<p>Deprecated</p>

			<ul style="list-style-type: none"> Deprecated OKTAMfaProvider. The property is not useful as we only support OKTA providers currently.
03/19/2021	7749	Snowflake	<p>Added</p> <ul style="list-style-type: none"> Added support for Google Cloud Platform for Snowflake. <p>Deprecated</p> <ul style="list-style-type: none"> Deprecated SSOldp and SSOldpDomain. SSOldp is taken from the AuthScheme now. SSOldpDomain is replaced with the Domain, which may be passed in via the SSOProperties.
03/16/2021	7745	Snowflake	<p>Changed</p> <ul style="list-style-type: none"> Changed to use PKCE as default for OAuth flow.

Advanced Features

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

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

Logging

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

User Defined Views

The Snowflake 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 [DemoDB].[PUBLIC].Products WHERE MyColumn =
'value'"
  },
  "MyView2": {
    "query": "SELECT * FROM MyTable WHERE Id IN (1,2,3)"
  }
}
```

Use the [UserDefinedViews](#) connection property to specify the location of your JSON configuration file. For example:

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

Defining Views Using DDL Statements

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

Create a View

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

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

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

Alter a View

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

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

The view is then updated in the JSON configuration file.

Drop a View

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

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

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

Schema for User Defined Views

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

Working with User Defined Views

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

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

An example of a query to the driver:

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

Resulting in the effective query to the source:

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

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

SSL Configuration

Customizing the SSL Configuration

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

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

Firewall and Proxy

Connecting Through a Firewall or Proxy

HTTP Proxies

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

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

Other Proxies

Set the following properties:

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

Query Processing

Query Processing

CData has a client-side SQL engine built into the adapter library. This enables support for the full capabilities that SQL-92 offers, including filters, aggregations, functions, etc.

For sources that do not support SQL-92, the adapter offloads as much of SQL statement processing as possible to Snowflake 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 Snowflake Query Evaluation component examines SQL queries and returns information indicating what parts of the query the adapter is not capable of executing natively.

The Snowflake 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](#):
TIBCO® Data Virtualization Snowflake Adapter Guide

Query Execution.

Logging

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

Basic Logging

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

- Logfile: A filepath which designates the name and location of the log file.
- Verbosity: This is a numerical value (1-5) that determines the amount of detail in the log. See the page in the Connection Properties section for an explanation of the five levels.
- MaxLogFileSize: When the limit is hit, a new log is created in the same folder with the date and time appended to the end. The default limit is 100 MB. Values lower than 100 kB will use 100 kB as the value instead.
- MaxLogFileCount: A string specifying the maximum file count of log files. When the limit is hit, a new log is created in the same folder with the date and time appended to the end and the oldest log file will be deleted. Minimum supported value is 2. A value of 0 or a negative value indicates no limit on the count.

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

Log Verbosity

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

-
- | | |
|---|---|
| 1 | Setting <u>Verbosity</u> to 1 will log the query, the number of rows returned by it, the start of execution and the time taken, and any errors. |
|---|---|
-
- | | |
|---|--|
| 2 | Setting <u>Verbosity</u> to 2 will log everything included in <u>Verbosity</u> 1 and additional information about the request. |
|---|--|
-

-
- | | |
|---|--|
| 3 | Setting <u>Verbosity</u> to 3 will additionally log HTTP headers, as well as the body of the request and the response. |
|---|--|
-
- | | |
|---|--|
| 4 | Setting <u>Verbosity</u> to 4 will additionally log transport-level communication with the data source. This includes SSL negotiation. |
|---|--|
-
- | | |
|---|--|
| 5 | Setting <u>Verbosity</u> to 5 will additionally log communication with the data source and additional details that may be helpful in troubleshooting problems. This includes interface commands. |
|---|--|
-

The Verbosity should not be set to greater than 1 for normal operation. Substantial amounts of data can be logged at higher verbosity levels, 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 Snowflake 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 Snowflake 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').

Transactions and Batching

Transactions are not currently supported.

Additionally, the adapter does not support batching of SQL statements. To execute multiple commands, you can create multiple instances and execute each separately. Or, use [Batch Processing](#).

SELECT Statements

A SELECT statement can consist of the following basic clauses.

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

SELECT Syntax

The following syntax diagram outlines the syntax supported by the Snowflake 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> ]
} [ , ... ]
[
    JOIN <table_reference> [ ON <search_condition> ] [ [ AS ]
<identifier> ]
] [ ... ]
[ WHERE <search_condition> ]
[ GROUP BY <column_reference> [ , ... ]
[ HAVING <search_condition> ]
[ UNION [ ALL ] <select_statement> ]
[
    ORDER BY
    <column_reference> [ ASC | DESC ] [ NULLS FIRST | NULLS LAST ]
]
[
    LIMIT <expression>
    [
        { OFFSET | , }
        <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> { = | > | < | >= | <= | <> | != | AND | OR | LIKE |
NOT LIKE | IN | NOT IN | IS NULL | IS NOT NULL | ANY | ALL | EXISTS |
NOT EXISTS | + | - | * | / | % | || | -(negate) } [ <expression> ]
    } [ { AND | OR } ... ]

```

Examples

1. Return all columns:

```
SELECT * FROM [DemoDB].[PUBLIC].Products
```

2. Rename a column:

```
SELECT "ProductName" AS MY_ProductName FROM [DemoDB].
[PUBLIC].Products
```

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

```
SELECT CAST(Price AS VARCHAR) AS Str_Price FROM [DemoDB].
[PUBLIC].Products
```

4. Search data:

```
SELECT * FROM [DemoDB].[PUBLIC].Products WHERE ProductName =
'Konbu'
```

5. The Snowflake APIs support the following operators in the WHERE clause: =, >, <, >=, <=, <>, !=, AND, OR, LIKE, NOT LIKE, IN, NOT IN, IS NULL, IS NOT NULL, ANY, ALL, EXISTS, NOT EXISTS, +, -, *, /, %, ||, -(negate).

```
SELECT * FROM [DemoDB].[PUBLIC].Products WHERE ProductName =
'Konbu';
```

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

```
SELECT COUNT(*) AS MyCount FROM [DemoDB].[PUBLIC].Products
```

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

```
SELECT COUNT(DISTINCT ProductName) FROM [DemoDB].[PUBLIC].Products
```

8. Return the unique items matching the query criteria:

```
SELECT DISTINCT ProductName FROM [DemoDB].[PUBLIC].Products
```

9. Summarize data:

```
SELECT ProductName, MAX(Price) FROM [DemoDB].[PUBLIC].Products
GROUP BY ProductName
```

See [Aggregate Functions](#) for details.

10. Retrieve data from multiple tables.

```
SELECT Customers.ContactName, Orders.OrderDate FROM Customers JOIN
Orders ON Customers.CustomerID=Orders.CustomerID
```

See [JOIN Queries](#) for details.

11. Sort a result set in ascending order:

```
SELECT Id, ProductName FROM [DemoDB].[PUBLIC].Products ORDER BY
ProductName ASC
```

JOIN Queries

The adapter supports the complete join syntax in Snowflake. Snowflake 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
  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 Snowflake:

- Cross joins must not contain an ON clause.

Projection Functions

BITAND(expr1, expr2)

Bitwise AND of two numeric expressions (a and b).

- **expr1:** This expression must evaluate to a data type that can be cast to INTEGER.
- **expr2:** This expression must evaluate to a data type that can be cast to INTEGER.

BITNOT(expr)

Bitwise negation of a numeric expression.

- **expr:** This expression must evaluate to a data type that can be cast to INTEGER.

BITOR(expr1, expr2)

Bitwise OR of two numeric expressions (a and b).

- **expr1:** This expression must evaluate to a data type that can be cast to INTEGER.
- **expr2:** This expression must evaluate to a data type that can be cast to INTEGER.

BITSHIFTLEFT(expr1, n)

Shift the bits for a numeric expression n positions to the left.

- **expr1:**
- **n:**

BITSHIFTRIGHT(expr1, n)

Shift the bits for a numeric expression n positions to the right, with sign extension.

- **expr1:** This expression must evaluate to a data type that can be cast to INTEGER.
- **n:** The number of bits to shift by.

BITXOR(expr1, expr2)

Bitwise XOR of two numeric expressions (a and b).

- **expr1:**
- **expr2:**

BOOLAND(expr1, expr2)

Computes the Boolean AND of two numeric expressions. In accordance with Boolean semantics. Non-zero values (including negative numbers) are regarded as True. Zero values are regarded as False.

- **expr1:**
- **expr2:**

BOOLNOT(expr1)

Computes the Boolean NOT of a single numeric expression. In accordance with Boolean semantics: Non-zero values (including negative numbers) are regarded as True. Zero values are regarded as False.

- **expr1:**

BOOLOR(expr1, expr2)

Computes the Boolean OR of two numeric expressions. In accordance with Boolean semantics: Non-zero values (including negative numbers) are regarded as True. Zero values are regarded as False.

- **expr1:**
- **expr2:**

BOOLXOR(expr1, expr2)

Computes the Boolean XOR of two numeric expressions (i.e. one of the expressions, but not both expressions, is TRUE). In accordance with Boolean semantics: Non-zero values (including negative numbers) are regarded as True. Zero values are regarded as False.

- **expr1:**
- **expr2:**

COALESCE(expr1 [, ...])

Returns the first non-NULL expression among its arguments, or NULL if all its arguments are NULL.

- **expr1:** Returns the first non-NULL expression among its arguments, or NULL if all its arguments are NULL.

DECODE(expr, search1, result1 [, search2, result2, ...] [, default])

Compares the select expression to each search expression in order. As soon as a search expression matches the selection expression, the corresponding result expression is returned.

- **expr:** This is the "select expression". The "search expressions" are compared to this select expression, and if there is a match then DECODE returns the result that corresponds to that search expression. The select expression is typically a column, but can be a subquery, literal, or other expression.
- **search1:** The search expressions indicate the values to compare to the select expression. If one of these search expressions matches, the function returns the corresponding result. If more than one search expression would match, only the first match's result is returned.
- **result1:** The results are the values that will be returned if one of the search expressions matches the select expression.
- **search2:** The search expressions indicate the values to compare to the select expression. If one of these search expressions matches, the function returns the corresponding result. If more than one search expression would match, only the first match's result is returned.

- **result2:** The results are the values that will be returned if one of the search expressions matches the select expression.
- **default:** If an optional default is specified, and if none of the search expressions match the select expression, then DECODE returns this default value.

EQUAL_NULL(expr1, expr2)

Compares whether two expressions are equal. The function is NULL-safe, meaning it treats NULLs as known values for comparing equality. Note that this is different from the EQUAL comparison operator (=), which treats NULLs as unknown values.

- **expr1:**
- **expr2:**

GREATEST(expr [, ...])

Returns the largest value from a list of expressions. GREATEST supports all types, including VARIANT. The first argument determines the return type. If the first type is numeric, then the return type will be 'widened' according to the numeric types in the list of all arguments. If the first type is not numeric, then all other arguments must be convertible to the first type. If any of the argument values is NULL, the result will be NULL.

- **expr:**

IFF(condition, expr1, expr2)

Single-level if-then-else expression. Similar to CASE, but only allows a single condition. If condition evaluates to TRUE, returns expr1, otherwise returns expr2.

- **condition:** The condition is an expression that should evaluate to a BOOLEAN value (True, False, or NULL).
- **expr1:** A general expression. This value is returned if the condition is true.
- **expr2:** A general expression. This value is returned if the condition is false.

IFNULL(expr1, expr2)

If expr1 is NULL, returns expr2, otherwise returns expr1.

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

LEAST(expr [, ...])

Returns the smallest value from a list of expressions. LEAST supports all data types, including VARIANT.

- **expr**: The arguments must include at least one expression. All the expressions should be of the same type or compatible types.

NULLIF(1, 2)

Returns NULL if expr1 is equal to expr2, otherwise returns expr1.

- **expr1**: any expression
- **expr2**: any expression

NVL(expr1, expr2)

If expr1 is NULL, returns expr2, otherwise returns expr1.

- **expr1**: The expression to be checked to see whether it's NULL.
- **expr2**: If expr1 is NULL, this expression will be evaluated and its value will be returned.

NVL2(expr1, expr2, expr3)

Returns values depending on the nullness of the first argument: If expr1 is not null, then NVL2 returns expr2. If expr1 is null, then NVL2 returns expr3.

- **expr1**: The expression to be checked to see whether it's NULL.
- **expr2**: If expr1 is not NULL, this expression will be evaluated and its value will be returned.
- **expr3**: If expr1 is NULL, this expression will be evaluated and its value will be returned.

REGR_VALX(expr1, expr2)

If the first argument is NULL, returns NULL. Otherwise, returns the second argument. Contrast REGR_VALX and REGR_VALY with NVL: NVL is a NULL-replacing function. The less commonly used REGR_VALX and REGR_VALY are NULL-preserving functions.

- **expr1:**
- **expr2:**

REGR_VALY(expr1, expr2)

If the second argument is NULL, returns NULL; otherwise, returns the first argument. Contrast REGR_VALX and REGR_VALY with NVL: NVL is a NULL-replacing function. The less commonly used REGR_VALX and REGR_VALY are NULL-preserving functions.

- **expr1:**
- **expr2:**

ZEROIFNULL(expr)

Returns 0 if its argument is null; otherwise, returns its argument.

- **expr:**

CURRENT_CLIENT()

Returns the version of the client from which the function was called. If called from an application using the JDBC or ODBC driver to connect to Snowflake, returns the version of the driver.

CURRENT_DATE()

Returns the current date of the system.

CURRENT_TIME([fract_sec_precision])

Returns the current time for the system.

- **fract_sec_precision>**: This optional argument indicates the precision with which to report the time. For example, a value of 3 says to use 3 digits after the decimal point - i.e. to specify the time with a precision of milliseconds.

CURRENT_TIMESTAMP([fract_sec_precision])

Returns the current timestamp for the system.

- **fract_sec_precision**: This optional argument indicates the precision with which to report the time. For example, a value of 3 says to use 3 digits after the decimal point - i.e. to specify the time with a precision of milliseconds.

CURRENT_VERSION()

Returns the current Snowflake version.

LOCALTIME()

Returns the current time for the system. ANSI-compliant alias for CURRENT_TIME.

LOCALTIMESTAMP()

Returns the current timestamp for the system. ANSI-compliant alias for CURRENT_TIMESTAMP.

CURRENT_ROLE()

Returns the name of the role in use for the current session. To specify a different role for the session, execute the USE ROLE command.

CURRENT_SESSION()

Returns a unique system identifier for the Snowflake session corresponding to the present connection. This will generally be a system-generated alphanumeric string. It is NOT derived from the user name or user account.

CURRENT_STATEMENT()

Returns the SQL text of the statement that is currently executing.

CURRENT_TRANSACTION()

Returns the transaction id of an open transaction in the current session.

CURRENT_USER()

Returns the name of the user currently logged into the system.

LAST_QUERY_ID([num])

Returns the ID of a specified query in the current session. If no query is specified, the most recently-executed query is returned.

- **num:** Specifies the query to return, based on the position of the query (within the session).

LAST_TRANSACTION()

Returns the transaction ID of the last transaction that was either committed or rolled back in the current session.

CURRENT_DATABASE()

Returns the name of the database in use for the current session. To specify a different database for the session, execute the `USE DATABASE` command.

CURRENT_SCHEMA()

Returns the name of the schema in use by the current session. To specify a different schema for the session, execute the `USE SCHEMA` command.

CURRENT_SCHEMAS()

Returns active search path schemas. For more information about search path, see Object Name Resolution.

CURRENT_WAREHOUSE()

Returns the name of the warehouse in use for the current session. To specify a different warehouse for the session, execute the USE WAREHOUSE command.

CAST(source_expr AS target_data_type)

Converts a value of one data type into another data type. The semantics of CAST are the same as the semantics of the corresponding TO_ datatype conversion functions. If the cast is not possible, an error is raised. For more details, see the individual TO_ datatype conversion functions.

- **source_expr:** Expression of any supported data type to be converted into a different data type.
- **target_data_type:** The data type to which to convert the expression. If the data type supports additional properties, such as scale and precision (for numbers/decimals), the properties can be included.

TRY_CAST(1 AS 2)

A special version of CAST , :: that is available for a subset of data type conversions. It performs the same operation (i.e converts a value of one data type into another data type), but returns a NULL value instead of raising an error when the conversion can not be performed.

- **source_expr:** Expression of any supported data type to be converted into a different data type.
- **target_data_type:** The data type to which to convert the expression. If the data type supports additional properties, such as scale and precision (for numbers/decimals), the properties can be included.

TO_CHAR(expr [, format])

Converts the input expression to a string. For NULL input, the output is NULL.

- **expr:** An expression of any data type.
- **format:** The format of the output string

TO_VARCHAR(expr [, format])

Converts the input expression to a string. For NULL input, the output is NULL.

- **expr:** An expression of any data type.
- **format:** The format of the output string

TO_BINARY(string_expr [, format])

Converts the input expression to a binary value. For NULL input, the output is NULL.

- **string_expr:** A string expression.
- **format:** The binary format for conversion: HEX, BASE64, or UTF-8 (see Binary Input and Output). The default is the value of the BINARY_INPUT_FORMAT session parameter. If this parameter is not set, the default is HEX

TRY_TO_BINARY(string_expr [, format])

A special version of TO_BINARY that performs the same operation (i.e. converts an input expression to a binary value), but with error handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **string_expr:** A string expression.
- **format:** The binary format for conversion: HEX, BASE64, or UTF-8 (see Binary Input and Output). The default is the value of the BINARY_INPUT_FORMAT session parameter. If this parameter is not set, the default is HEX

TO_DECIMAL(expr [, format [, precision [, scale]]])

Converts an input expression to a fixed-point number. For NULL input, the output is NULL.

- **expr:** An expression of a numeric, character, or variant type.
- **format:** The SQL format model used to parse the input expr and return. For more information, see SQL Format Models.
- **precision:** The maximal number of decimal digits in the resulting number; from 1 to 38. In Snowflake, precision is not used for determination of the number of bytes needed to store the number and does not have any effect on efficiency, so the default is the maximum (38).
- **scale:** The number of fractional decimal digits (from 0 to precision - 1). 0 indicates no fractional digits (i.e. an integer number).

TRY_TO_DECIMAL(expr [, format [, precision [, scale]])

A special version of TO_DECIMAL, TO_NUMBER, TO_NUMERIC that performs the same operation (i.e. converts an input expression to a fixed-point number), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **expr:** An expression of a numeric, character, or variant type.
- **format:** The SQL format model used to parse the input expr and return. For more information, see SQL Format Models.
- **precision:** The maximal number of decimal digits in the resulting number; from 1 to 38. In Snowflake, precision is not used for determination of the number of bytes needed to store the number and does not have any effect on efficiency, so the default is the maximum (38).
- **scale:** The number of fractional decimal digits (from 0 to precision - 1). 0 indicates no fractional digits (i.e. an integer number).

TO_DOUBLE(expr [, format])

Converts an expression to a double-precision floating-point number.

- **expr:** An expression of a numeric, character, or variant type.
- **format:** If the expression evaluates to a string, then the function accepts an optional format model. Format models are described at SQL Format Models. The format model specifies the format of the input string, not the format of the output value.

TRY_TO_DOUBLE(expr [, format])

A special version of TO_DOUBLE that performs the same operation (i.e. converts an input expression to a double-precision floating-point number), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **expr:** An expression of a numeric, character, or variant type.
- **format:** If the expression evaluates to a string, then the function accepts an optional format model. Format models are described at SQL Format Models. The format model specifies the format of the input string, not the format of the output value.

TO_BOOLEAN(text_or_numeric_expr)

Converts the input text or numeric expression to a Boolean value. For NULL input, the output is NULL.

- **text_or_numeric_expr:** A text or numeric expression

TRY_TO_BOOLEAN(text_or_numeric_expr)

A special version of TO_BOOLEAN that performs the same operation (i.e. converts an input expression to a Boolean value), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **text_or_numeric_expr:** A text or numeric expression

TO_DATE(expr [, format])

Converts an input expression to a date

- **expr:** Expression to be converted into a date.
- **format:** Date format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TRY_TO_DATE(expr [, format])

A special version of TO_DATE that performs the same operation (i.e. converts an input expression to a Boolean value), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **expr:** Expression to be converted into a date.
- **format:** Date format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TO_TIME(expr [, format])

Converts an input expression into a time. If input is NULL, returns NULL.

- **expr:** Expression to be converted into a time
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TRY_TO_TIME(expr [, format])

A special version of TO_TIME that performs the same operation (i.e. converts an input expression to a Boolean value), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **expr:** Expression to be converted into a time
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TO_TIMESTAMP(expr [, format])

Converts an input expression into the corresponding timestamp

- **expr:** Expression to be converted into a timestamp
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TRY_TO_TIMESTAMP(expr [, format])

A special version of TO_TIMESTAMP that performs the same operation (i.e. converts an input expression to a Boolean value), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **expr:** Expression to be converted into a timestamp
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TO_TIMESTAMP_NTZ(expr [, format])

Converts an input expression into the corresponding timestamp

- **expr:** Expression to be converted into a timestamp
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TRY_TO_TIMESTAMP_NTZ(expr [, format])

A special version of TO_TIMESTAMP_NTZ that performs the same operation (i.e. converts an input expression to a Boolean value), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **expr:** Expression to be converted into a timestamp
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TO_TIMESTAMP_TZ(expr [, format])

Converts an input expression into the corresponding timestamp

- **expr:** Expression to be converted into a timestamp
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TRY_TO_TIMESTAMP_TZ(expr [, format])

A special version of TO_TIMESTAMP_TZ that performs the same operation (i.e. converts an input expression to a Boolean value), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **expr:** Expression to be converted into a timestamp
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TO_TIMESTAMP_LTZ(expr [, format])

Converts an input expression into the corresponding timestamp

- **expr:** Expression to be converted into a timestamp
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TRY_TO_TIMESTAMP_LTZ(expr [, format])

A special version of TO_TIMESTAMP_LTZ that performs the same operation (i.e. converts an input expression to a Boolean value), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **expr:** Expression to be converted into a timestamp
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

RANDOM(seed)

Each call returns a pseudo-random 64-bit integer.

- **seed:** Seed is an integer. Different seeds will cause RANDOM to produce different output values.

RANDSTR(length, gen)

Returns a random string of specified length. Individual characters are chosen uniformly at random from the following pool of characters: 0-9, a-z, A-Z.

- **length:** Length of the string to generate
- **gen:** The value for the generator expression, gen, is used as the seed for this uniform random distribution

UUID_STRING(uuid, name)

Generates either a version 4 (random) or version 5 (named) RFC 4122-compliant UUID as a formatted string.

- **uuid:** The string (known as the namespace)
- **name:** The name of the UUID

NORMAL(mean, stddev, gen)

Returns a normal-distributed floating point number, with specified mean and stddev (standard deviation).

- **mean:** This is the value that you would like the output values centered around
- **stddev:** This specifies the width of one standard deviation.
- **gen:** This specifies the generator expression for the function.

UNIFORM(min, max, gen)

Returns a uniformly random number, in the inclusive range [min, max].

- **min:** The Minimum Number
- **max:** The Maximum Number
- **gen:** The generator expression for the function

ZIPF(s, N, gen)

Returns a Zipf-distributed integer, for N elements and characteristic exponent s

- **s**: the characteristic exponent
- **N**: the number of elements
- **gen**: The generator expression for the function

SEQ1([sign])

Returns a sequence of monotonically increasing integers, with wrap-around. Wrap-around occurs after the largest representable integer of the integer width (1 byte).

- **sign**: The optional sign argument. If the optional sign argument is 1, the sequence continues at the smallest representable number based on the given integer width. The default sign argument is 0.

SEQ2([sign])

Returns a sequence of monotonically increasing integers, with wrap-around. Wrap-around occurs after the largest representable integer of the integer width (2 byte).

- **sign**: The optional sign argument. If the optional sign argument is 1, the sequence continues at the smallest representable number based on the given integer width. The default sign argument is 0.

SEQ4([sign])

Returns a sequence of monotonically increasing integers, with wrap-around. Wrap-around occurs after the largest representable integer of the integer width (4 byte).

- **sign**: The optional sign argument. If the optional sign argument is 1, the sequence continues at the smallest representable number based on the given integer width. The default sign argument is 0.

SEQ8([sign])

Returns a sequence of monotonically increasing integers, with wrap-around. Wrap-around occurs after the largest representable integer of the integer width (8 byte).

- **sign:** The optional sign argument. If the optional sign argument is 1, the sequence continues at the smallest representable number based on the given integer width. The default sign argument is 0.

DATE_FROM_PARTS(year, month, day)

Creates a date from individual numeric components that represent the year, month, and day of the month

- **year:** The integer expression to use as a year for building a date.
- **month:** The integer expression to use as a month for building a date, with January represented as 1, and December as 12.
- **day:** The integer expression to use as a day for building a date, usually in the 1-31 range.

TIME_FROM_PARTS(hour, minute, second, nanoseconds)

Creates a time from individual numeric components.

- **hour:** An integer expression to use as an hour for building a time, usually in the 0-23 range.
- **minute:** An integer expression to use as a minute for building a time, usually in the 0-59 range.
- **second:** An integer expression to use as a second for building a time, usually in the 0-59 range.
- **nanoseconds:** A 9-digit integer expression to use as a nanosecond for building a time

TIMESTAMP_FROM_PARTS(date_expr, time_expr)

Creates a timestamp from individual numeric components. If no time zone is in effect, the function can be used to create a timestamp from a date expression and a time expression.

- **date_expr**: provides the year, month, and day for the timestamp
- **time_expr**: provides the hour, minute, second, and nanoseconds within the day

TIMESTAMP_FROM_PARTS(year, month, day, hour, minute, second [, nanoseconds [, time_zone]])

Creates a timestamp from individual numeric components. If no time zone is in effect, the function can be used to create a timestamp from a date expression and a time expression.

- **year**: An integer expression to use as a year for building a timestamp.
- **month**: An integer expression to use as a month for building a timestamp, with January represented as 1, and December as 12.
- **day**: An integer expression to use as a day for building a timestamp, usually in the 1-31 range.
- **hour**: An integer expression to use as an hour for building a timestamp, usually in the 0-23 range.
- **minute**: An integer expression to use as a minute for building a timestamp, usually in the 0-59 range.
- **second**: An integer expression to use as a second for building a timestamp, usually in the 0-59 range.
- **nanoseconds**: An integer expression to use as a nanosecond for building a timestamp, usually in the 0-999999999 range.
- **time_zone**: A string expression to use as a time zone for building a timestamp (e.g. America/Los_Angeles)

TIMESTAMP_NTZ_FROM_PARTS(date_expr, time_expr)

Creates a timestamp from individual numeric components. If no time zone is in effect, the function can be used to create a timestamp from a date expression and a time expression.

- **date_expr**: provides the year, month, and day for the timestamp
- **time_expr**: provides the hour, minute, second, and nanoseconds within the day

TIMESTAMP_NTZ_FROM_PARTS(year, month, day, hour, minute, second [, nanoseconds])

Creates a timestamp from individual numeric components. If no time zone is in effect, the function can be used to create a timestamp from a date expression and a time expression.

- **year:** An integer expression to use as a year for building a timestamp.
- **month:** An integer expression to use as a month for building a timestamp, with January represented as 1, and December as 12.
- **day:** An integer expression to use as a day for building a timestamp, usually in the 1-31 range.
- **hour:** An integer expression to use as an hour for building a timestamp, usually in the 0-23 range.
- **minute:** An integer expression to use as a minute for building a timestamp, usually in the 0-59 range.
- **second:** An integer expression to use as a second for building a timestamp, usually in the 0-59 range.
- **nanoseconds:** An integer expression to use as a nanosecond for building a timestamp, usually in the 0-999999999 range.

TIMESTAMP_LTZ_FROM_PARTS(year, month, day, hour, minute, second [, nanoseconds])

Creates a timestamp from individual numeric components. If no time zone is in effect, the function can be used to create a timestamp from a date expression and a time expression.

- **year:** An integer expression to use as a year for building a timestamp.
- **month:** An integer expression to use as a month for building a timestamp, with January represented as 1, and December as 12.
- **day:** An integer expression to use as a day for building a timestamp, usually in the 1-31 range.
- **hour:** An integer expression to use as an hour for building a timestamp, usually in the 0-23 range.
- **minute:** An integer expression to use as a minute for building a timestamp, usually

in the 0-59 range.

- **second:** An integer expression to use as a second for building a timestamp, usually in the 0-59 range.
- **nanoseconds:** An integer expression to use as a nanosecond for building a timestamp, usually in the 0-999999999 range.

TIMESTAMP_TZ_FROM_PARTS(year, month, day, hour, minute, second [, nanoseconds [, time_zone]])

Creates a timestamp from individual numeric components. If no time zone is in effect, the function can be used to create a timestamp from a date expression and a time expression.

- **year:** An integer expression to use as a year for building a timestamp.
- **month:** An integer expression to use as a month for building a timestamp, with January represented as 1, and December as 12.
- **day:** An integer expression to use as a day for building a timestamp, usually in the 1-31 range.
- **hour:** An integer expression to use as an hour for building a timestamp, usually in the 0-23 range.
- **minute:** An integer expression to use as a minute for building a timestamp, usually in the 0-59 range.
- **second:** An integer expression to use as a second for building a timestamp, usually in the 0-59 range.
- **nanoseconds:** An integer expression to use as a nanosecond for building a timestamp, usually in the 0-999999999 range.
- **time_zone:** A string expression to use as a time zone for building a timestamp (e.g. America/Los_Angeles)

DATE_PART(date_or_time_part, date_or_time_expr)

Extracts the specified date or time part from a date, time, or timestamp.

- **date_or_time_part:** The date or time part to extract
- **date_or_time_expr:** The date or time expression to extract from

DAYNAME(date_or_timestamp_expr)

Extracts the three-letter day-of-week name from the specified date or timestamp.

- **date_or_timestamp_expr:** The date or time expression to extract from

HOUR(time_or_timestamp_expr)

Extracts the corresponding time part from a time or timestamp value.

- **time_or_timestamp_expr:**

MINUTE(time_or_timestamp_expr)

Extracts the corresponding time part from a time or timestamp value.

- **time_or_timestamp_expr:**

SECOND(time_or_timestamp_expr)

Extracts the corresponding time part from a time or timestamp value.

- **time_or_timestamp_expr:**

YEAR(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

YEAROFWEEK(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

YEAROFWEEKISO(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

DAY(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

DAYOFMONTH(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

DAYOFWEEK(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

DAYOFWEEKISO(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

DAYOFYEAR(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

WEEK(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

WEEKOFYEAR(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

WEEKISO(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

MONTH(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

QUARTER(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

LAST_DAY(date_or_time_expr [, date_part])

Returns the last day of the specified date part for a date or timestamp. Commonly used to return the last day of the month for a date or timestamp.

- **date_or_time_expr:**
- **date_part:**

MONTHNAME(date_or_timestamp_expr)

Extracts the three-letter month name from the specified date or timestamp.

- **date_or_timestamp_expr:**

NEXT_DAY(date_or_time_expr, dow_string)

Returns the date of the first specified DOW (day of week) that occurs after the input date.

- **date_or_time_expr:** Specifies the input date; can be a date or timestamp.
- **dow_string:** Specifies the day of week used to calculate the date for the previous day. The value can be a string literal or an expression that returns a string. The string must start with the first two characters (case-insensitive) of the day name

PREVIOUS_DAY(date_or_time_expr, dow)

Returns the date of the first specified DOW (day of week) that occurs before the input date.

- **date_or_time_expr:** Specifies the input date; can be a date or timestamp
- **dow:** Specifies the day of week used to calculate the date for the previous day. The value can be a string literal or an expression that returns a string. The string must start with the first two characters (case-insensitive) of the day name

ADD_MONTHS(date_or_timestamp_expr, num_months_expr)

Adds or subtracts a specified number of months to a date or timestamp, preserving the end-of-month information.

- **date_or_timestamp_expr:** This is the date or timestamp expression to which you want to add a specified number of months.
- **num_months_expr:** This is the number of months you want to add. This should be an integer. It may be positive or negative. If the value is a non-integer numeric value (for example, FLOAT) the value will be rounded to the nearest integer.

EXTRACT(date_or_time_part FROM date_or_time_expr)

Extracts the specified date or time part from a date, time, or timestamp.

- **date_or_time_part:**
- **date_or_time_expr:**

DATEADD(date_or_time_part, value, date_or_time_expr)

Adds the specified value for the specified date or time part to a date, time, or timestamp.

- **date_or_time_part:** This indicates the units of time that you want to add. For example if you want to add 2 days, then this will be DAY.
- **value:** This is the number of units of time that you want to add. For example, if you want to add 2 days, this will be 2.
- **date_or_time_expr:** must evaluate to a date, time, or timestamp. This is the date, time, or timestamp to which you want to add. For example, if you want to add 2 days to August 1, 2018, then this will be '2018-08-01'::DATE

TIMEADD(date_or_time_part, value, date_or_time_expr)

Adds the specified value for the specified date or time part to a date, time, or timestamp.

- **date_or_time_part:** This indicates the units of time that you want to add. For example if you want to add 2 days, then this will be DAY
- **value:** This is the number of units of time that you want to add. For example, if you want to add 2 days, this will be 2.
- **date_or_time_expr:** must evaluate to a date, time, or timestamp. This is the date, time, or timestamp to which you want to add. For example, if you want to add 2 days to August 1, 2018, then this will be '2018-08-01'::DATE.

TIMESTAMPADD(date_or_time_part, time_value, date_or_time_expr)

Adds the specified value for the specified date or time part to a date, time, or timestamp.

- **date_or_time_part:**
- **time_value:**
- **date_or_time_expr:**

DATEDIFF(date_or_time_part, date_or_time_expr1, date_or_time_expr2)

Calculates the difference between two date, time, or timestamp expressions based on the date or time part requested. The function returns the result of subtracting the second argument from the third argument.

- **date_or_time_part:**
- **date_or_time_expr1:** must be a date, a time, a timestamp, or an expression that can be evaluated to a date, a time, or a timestamp. The first value is subtracted from the second value.
- **date_or_time_expr2:** must be a date, a time, a timestamp, or an expression that can be evaluated to a date, a time, or a timestamp. The first value is subtracted from the second value.

TIMEDIFF(date_or_time_part, date_or_time_expr1, time_expr2)

Calculates the difference between two date, time, or timestamp expressions based on the specified date or time part.

- **date_or_time_part:**
- **date_or_time_expr1:** must be a date, a time, a timestamp, or an expression that can be evaluated to one of those. The first value is subtracted from the second value
- **time_expr2:** must be a date, a time, a timestamp, or an expression that can be evaluated to one of those.

TIMESTAMPDIFF(date_or_time_part, date_or_time_expr1, date_or_time_expr2)

Calculates the difference between two date, time, or timestamp expressions based on the specified date or time part.

- **date_or_time_part:**
- **date_or_time_expr1:**
- **date_or_time_expr2:**

DATE_TRUNC(date_or_time_part, date_or_time_expr)

Truncates a date, time, or timestamp to the specified part. Note that truncation is not the same as extraction. For example: Truncating a timestamp down to the quarter returns the timestamp corresponding to midnight of the first day of the quarter for the input timestamp. Extracting the quarter date part from a timestamp returns the quarter number of the year in the timestamp.

- **date_or_time_part:**
- **date_or_time_expr:**

CONVERT_TIMEZONE(source_tz, target_tz, source_timestamp_ntz)

Converts a timestamp to another time zone

- **source_tz:** String specifying the time zone for the input timestamp. Required for timestamps with no time zone (i.e. TIMEZONE_NTZ)
- **target_tz:** String specifying the time zone to which the input timestamp should be converted.
- **source_timestamp_ntz:** For the 3-argument version, string specifying the timestamp to convert (must be NTZ).

CONVERT_TIMEZONE(target_tz, source_timestamp)

Converts a timestamp to another time zone

- **target_tz:** String specifying the time zone to which the input timestamp should be converted.
- **source_timestamp:** For the 2-argument version, string specifying the timestamp to convert (can be any timestamp variant, including NTZ).

HASH(expr [, ...])

Returns a signed 64-bit hash value. Note that HASH never returns NULL, even for NULL inputs.

- **expr:**

HASH_AGG([DISTINCT] expr [, ...])

Returns an aggregate signed 64-bit hash value over the (unordered) set of input rows. HASH_AGG never returns NULL, even if no input is provided. Empty input "hashes" to 0.

- **expr:**

GET_DDL(object_type, namespace_object_name)

Returns a DDL statement that can be used to recreate the specified object. For databases and schemas, GET_DDL is recursive, i.e. it returns the DDL statements for recreating all supported objects within the specified database/schema.

- **object_type:** Specifies the type of object for which the DDL is returned.
- **namespace_object_name:** Specifies the fully-qualified name of the object for which the DDL is returned.

ABS(num_expr)

Returns the absolute value of a numeric expression.

- **num_expr:**

CEIL(input_expr [, scale_expr])

Returns values from input_expr rounded to the nearest equal or larger integer, or to the nearest equal or larger value with the specified number of places after the decimal point

- **input_expr:** The input_expr is the value to be rounded up, and should evaluate to a numeric data type, such as FLOAT or NUMBER.
- **scale_expr:** The scale_expr indicates the number of places after the decimal to which to round upward. This should evaluate to an integer.

FLOOR(input_expr [, scale_expr])

Returns values from input_expr rounded to the nearest equal or smaller integer, or to the nearest equal or smaller value with the specified number of places after the decimal point

- **input_expr:** The input_expr is the value to be rounded up, and should evaluate to a numeric data type, such as FLOAT or NUMBER.
- **scale_expr:** The scale_expr indicates the number of places after the decimal to which to round upward. This should evaluate to an integer.

MOD(expr1, expr2)

Returns the remainder of input expr1 divided by input expr2.

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

ROUND(input_expr, scale_expr)

Returns rounded values for input_expr

- **input_expr:** The expression to round
- **scale_expr:** If the optional scale_expr argument is specified, rounding is performed to the specified number of digits (negative digits round to factors-of-10)

SIGN(expr)

Returns the sign of its argument

- **expr**: an expression

TRUNCATE(input_expr, scale_expr)

Rounds the input expression down to the nearest (or equal) integer towards zero.

- **input_expr**: the expression to truncate
- **scale_expr**: If the optional scale_expr argument is provided, rounding is performed to the specified number of digits (negative digits round to factors-of-10)

CBRT(expr)

Returns the cubic root of a numeric expression.

- **expr**: a numeric expression

EXP(real_expr)

Computes Euler's number e raised to a floating-point value.

- **real_expr**: a real expression

FACTORIAL(integer_expr)

Computes the factorial of its input. The input argument must be an integer expression in the range of 0 to 33.

- **integer_expr**: an integer expression

POWER(x, y)

Returns a number (x) raised to the specified power (y).

- **x**:

- **y:**

SQRT(expr)

Returns the square-root of a non-negative numeric expression.

- **expr:** a non-negative numeric expression.

SQUARE(expr)

Returns the square of a numeric expression, i.e. a numeric expression multiplied by itself.

- **expr:** a numeric expression

LN(expr)

Returns the natural logarithm of a numeric expression.

- **expr:** a numeric expression

LOG(base, expr)

Returns the logarithm of a numeric expression.

- **base:** The "base" to use (e.g. 10 for base 10 arithmetic). This can be of any numeric data type (INTEGER, fixed-point, or floating point).
- **expr:** The value for which you want to know the log.

ACOS(real_expr)

Computes the inverse cosine (arc cosine) of its input; the result is a number in the interval $[-\pi, \pi]$.

- **real_expr:** This expression should evaluate to a real number greater than or equal to -1.0 and less than or equal to +1.0.

ACOSH(real_expr)

Computes the inverse (arc) hyperbolic cosine of its input.

- **real_expr:** This expression should evaluate to a FLOAT number greater than or equal to 1.0.

ASIN(real_expr)

Computes the inverse sine (arc sine) of its argument; the result is a number in the interval $[-\pi, \pi]$.

- **real_expr:** This expression should evaluate to a real number greater than or equal to -1.0 and less than or equal to +1.0.

ASINH(real_expr)

Computes the inverse (arc) hyperbolic sine of its argument.

- **real_expr:** This expression should evaluate to a real number.

ATAN(real_expr)

Computes the inverse tangent (arc tangent) of its argument; the result is a number in the interval $[-\pi, \pi]$.

- **real_expr:** This expression should evaluate to a real number.

ATANH(real_expr)

Computes the inverse (arc) hyperbolic tangent of its argument.

- **real_expr:**

COS(real_expr)

Computes the cosine of its argument; the argument should be expressed in radians.

- **real_expr:** This expression should evaluate to a real number. The value should be in radians, not degrees.

COSH(real_expr)

Computes the hyperbolic cosine of its argument.

- **real_expr:** This expression should evaluate to a real number.

COT(real_expr)

Computes the cotangent of its argument; the argument should be expressed in radians.

- **real_expr:** This expression should evaluate to a real number.

DEGREES(real_expr)

Converts radians to degrees.

- **real_expr:** An expression representing the number of radians.

RADIANS(real_expr)

Converts degrees to radians.

- **real_expr:**

SIN(real_expr)

Computes the sine of its argument; the argument should be expressed in radians.

- **real_expr:** This expression should evaluate to a real number. The value should be in radians, not degrees.

SINH(real_expr)

Computes the hyperbolic sine of its argument.

- **real_expr:** This expression should evaluate to a real number.

TAN(real_expr)

Computes the tangent of its argument; the argument should be expressed in radians.

- **real_expr:** This expression should evaluate to a real number. The value should be in radians, not degrees.

TANH(real_expr)

Computes the hyperbolic tangent of its argument.

- **real_expr:** This expression should evaluate to a real number.

ATAN2(real_expr, real_expr)

Computes the inverse tangent (arc tangent) of the ratio of its two arguments (i.e. $ATAN2(x,y) = ATAN(x/y)$). The result is a number in the interval $[-\pi, \pi]$.

- **real_expr:**
- **real_expr:**

PI()

Returns the value of pi as a floating-point value.

HAVERSINE(lat1, lon1, lat2, lon2)

Calculates the great circle distance in kilometers between two points on the Earth's surface, using the Haversine formula. The two points are specified by their latitude and longitude in degrees.

- **lat1:** Latitude of Point 1
- **lon1:** Longitude of Point 1
- **lat2:** Latitude of Point 2

- **lon2**: Longitude of Point 2

CHECK_JSON(string_or_variant_expr)

Checks the validity of a JSON document. If the input string is a valid JSON document or a NULL, the output is NULL (i.e. no error). If the input cannot be translated to a valid JSON value, the output string contains the error message.

- **string_or_variant_expr**: A VARIANT or string value (or expression) to check. If the expression is of type VARIANT, it should contain a string.

CHECK_XML(string_or_variant_expr)

Checks the validity of an XML document. If the input string is NULL or a valid XML document, the output is NULL. In case of an XML parsing error, the output string contains the error message.

- **string_or_variant_expr**: A VARIANT or string value (or expression) to check. If the expression is of type VARIANT, it should contain a string.

PARSE_JSON(expr)

Interprets an input string as a JSON document, producing a VARIANT value.

- **expr**: An expression of string type (e.g. VARCHAR) that holds valid JSON information.

PARSE_XML(expr)

Interprets an input string as an XML document, producing an OBJECT value. If the input is NULL, the output is NULL.

- **expr**: An expression of string type (e.g. VARCHAR) that holds valid JSON information.

STRIP_NULL_VALUE(expr)

Converts a JSON "null" value to a SQL NULL value. All other variant values are passed unchanged.

- **expr:** an expression to strip data from

ARRAY_APPEND(array, new_element)

Returns an array containing all elements from the source array as well as the new element. The new element is located at end of the array.

- **array:** The source array.
- **new_element:** The element to be appended. The element may be of almost any data type. The data type does not need to match the data type(s) of the existing elements in the array.

ARRAY_PREPEND(array, new_element)

Returns an array containing the new element as well as all elements from the source array. The new element is positioned at the beginning of the array.

- **array:** The source array.
- **new_element:** The element to be prepended.

ARRAY_CAT(array1, array2)

Returns a concatenation of two arrays.

- **array1:** The source array.
- **array2:** The array to be appended to array1.

ARRAY_INSERT(array, pos, new_element)

Returns an array containing all elements from the source array as well as the new element.

- **array:** The source array.
- **pos:** A (zero-based) position in the source array. The new element is inserted at this position. The original element from this position (if any) and all subsequent elements (if any) are shifted by one position to the right in the resulting array (i.e. inserting at position 0 has the same effect as using ARRAY_PREPEND). A negative position is

interpreted as an index from the back of the array (e.g. -1 results in insertion before the last element in the array).]

- **new_element:** The element to be inserted. The new element is located at position pos. The relative order of the other elements from the source array is preserved.

ARRAY_COMPACT(array1)

Returns a compacted array with missing and null values removed, effectively converting sparse arrays into dense arrays.

- **array1:** The source array.

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

Returns an array constructed from zero, one, or more inputs.

- **expr1:**
- **expr2:**

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

Returns an array constructed from zero, one, or more inputs.

- **expr1:**
- **expr2:**

ARRAY_CONTAINS(variant, array)

Takes a VARIANT and an ARRAY value as inputs and returns True if the VARIANT is contained in the ARRAY.

- **variant:**
- **array:**

ARRAY_POSITION(variant_expr, array)

Returns the index of the first occurrence of an element in an array.

- **variant_expr:** This expression should evaluate to a VARIANT value. The function searches for the first occurrence of this value in the array.
- **array:** The array to be searched.

ARRAY_SIZE(array_or_variant)

Returns the size of the input array. A variation of ARRAY_SIZE takes a VARIANT value as input. If the VARIANT value contains an array, the size of the array is returned; otherwise, NULL is returned if the value is not an array.

- **array_or_variant:**

ARRAY_SLICE(array, from, to)

Returns an array constructed from a specified subset of elements of the input array.

- **array:** The source array of which a subset of the elements are used to construct the resulting array.
- **from:** A position in the source array. The position of the first element is 0. Elements from positions less than from are not included in the resulting array.
- **to:** A position in the source array. Elements from positions equal to or greater than to are not included in the resulting array.

ARRAY_TO_STRING(array, separator_string)

Returns an input array converted to a string by casting all values to strings (using TO_VARCHAR) and concatenating them (using the string from the second argument to separate the elements).

- **array:** The array of elements to convert to a string.
- **separator_string:** The string to put between each element, typically a space, comma, or other human-readable separator.

ARRAYS_OVERLAP(array1, array2)

Compares whether two arrays have at least one element in common. Returns TRUE if there is at least one element in common; otherwise returns FALSE. The function is NULL-safe,

meaning it treats NULLs as known values for comparing equality.

- **array1:** an array
- **array2:** an array

OBJECT_AGG(key, value)

Returns one OBJECT per group. For each (key, value) input pair, where key must be a VARCHAR and value must be a VARIANT, the resulting OBJECT contains a key:value field.

- **key:**
- **value:**

OBJECT_CONSTRUCT([key1, value1 [, keyN, valueN, ...]])

Returns an object constructed from the arguments.

- **key1:**
- **value1:**
- **keyN:**
- **valueN:**

OBJECT_DELETE(object, key1 [, key2, ...])

Returns an object containing the contents of the input (i.e.source) object with one or more keys removed.

- **object:** The source object.
- **key1:** Key to be omitted from the returned object.
- **key2:** Key to be omitted from the returned object.

OBJECT_INSERT(object, key, value [, updateFlag])

Returns an object consisting of the input object with a new key-value pair inserted (or an existing key updated with a new value).

- **object:** The source object into which the new key-value pair is inserted.
- **key:** The new key to be inserted into the object. Must be different from all existing keys in the object, unless `updateFlag` is set to `TRUE`.
- **value:** The value associated with the key.
- **updateFlag:** Boolean flag that, when set to `TRUE`, specifies the input value is used to update/overwrite an existing key in the object, rather than inserting a new key-value pair.

XMLGET(type, tag_name [, instance_num])

Extracts an XML element object (often referred to as simply a "tag") from a content of outer XML element object by the name of the tag and its instance number (counting from 0)

- **type:**
- **tag_name:**
- **instance_num:** can be omitted, in which case the default value 0 is used.

GET(expr1, expr2)

Extracts a value from an object or array; returns `NULL` if either of the arguments is `NULL`.

- **expr1:** An Object, Variant, or Array
- **expr2:** A string value or an integer, which can be a constant or an expression

AS_ARRAY(variant_expr)

Casts a `VARIANT` value to an array.

- **variant_expr:** An expression that evaluates to a value of type `VARIANT`.

AS_BINARY(variant_expr)

Casts a `VARIANT` value to a binary string.

- **variant_expr:** An expression that evaluates to a value of type `VARIANT`.

AS_CHAR(variant_expr)

Casts a VARIANT value to a string. Does not convert values of other types into string.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_VARCHAR(variant_expr)

Casts a VARIANT value to a string. Does not convert values of other types into string.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_DATE(variant_expr)

Casts a VARIANT value to a date. Does not convert from timestamps.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_DOUBLE(variant_expr)

Casts a VARIANT value to a floating-point value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_REAL(variant_expr)

Casts a VARIANT value to a floating-point value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_INTEGER(variant_expr)

Casts a VARIANT value to an integer. Does not match non-integer values.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_OBJECT(variant_expr)

Casts a VARIANT value to an object.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_TIME(variant_expr)

Casts a VARIANT value to a time value. Does not convert from timestamps.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_TIMESTAMP_LTZ(variant_expr)

Casts a VARIANT value to the respective TIMESTAMP value with local time zone.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_TIMESTAMP_NTZ(variant_expr)

Casts a VARIANT value to the respective TIMESTAMP value with no time zone.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_TIMESTAMP_TZ(variant_expr)

Casts a VARIANT value to the respective TIMESTAMP value with time zone.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_DECIMAL(variant_expr [, precision [, scale]])

Casts a VARIANT value to a fixed-point decimal (does not match floating-point values), with optional precision and scale.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.
- **precision**: The number of significant digits of the decimal number to store.
- **scale**: The number of significant digits after the decimal point.

AS_NUMBER(variant_expr [, precision [, scale]])

Casts a VARIANT value to a fixed-point decimal (does not match floating-point values), with optional precision and scale.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.
- **precision**: The number of significant digits of the decimal number to store.
- **scale**: The number of significant digits after the decimal point.

STRTOK_TO_ARRAY(string [, delimiter])

Tokenizes the given string using the given set of delimiters and returns the tokens as an array. If either parameter is a NULL, a NULL is returned. An empty array is returned in case tokenization produces no tokens.

- **string**: Text to be tokenized.
- **delimiter**: Set of delimiters. Optional. Default value is a single space character

STRTOK_TO_SPLIT_TO_TABLE(string [, delimiter])

Tokenizes a string with the given set of delimiters and flattens the results into rows.

- **string**: Text to be tokenized.
- **delimiter**: Set of delimiters. Optional. Default value is a single space character

TO_ARRAY(expr)

Converts the input expression into an array: If the input is an ARRAY, or VARIANT containing an array value, the result is unchanged. For NULL or a JSON null input, returns NULL. For any other value, the result is a single-element array containing this value.

- **expr**: An expression of any data type.

TO_JSON(expr)

Converts any VARIANT value to a string containing the JSON representation of the value. If the input is NULL, the result is also NULL.

- **expr:** An expression of type VARIANT that holds valid JSON information.

TO_OBJECT(expr)

Converts the input value to an object

- **expr:** An expression of any data type.

TO_VARIANT(expr)

Converts any value to VARIANT value or NULL (if input is NULL).

- **expr:** An expression of any data type.

TO_XML(expr1)

Converts any VARIANT value to a string containing the XML representation of the value. If the input is NULL, the result is also NULL.

- **expr1:** The Variant to convert

IS_ARRAY(variant_expr)

Returns TRUE if its VARIANT argument contains an ARRAY value.

- **variant_expr:** An expression that evaluates to a value of type VARIANT.

IS_BOOLEAN(variant_expr)

Returns TRUE if its VARIANT argument contains an Boolean value.

- **variant_expr:** An expression that evaluates to a value of type VARIANT.

IS_BINARY(variant_expr)

Returns TRUE if its VARIANT argument contains an binary value.

- **variant_expr:** An expression that evaluates to a value of type VARIANT.

IS_CHAR(variant_expr)

Returns TRUE if its VARIANT argument contains an string value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_VARCHAR(variant_expr)

Returns TRUE if its VARIANT argument contains an string value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_DATE(variant_expr)

Returns TRUE if its VARIANT argument contains an DATE value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_DATE_VALUE(variant_expr)

Returns TRUE if its VARIANT argument contains an DATE value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_DECIMAL(variant_expr)

Returns TRUE if its VARIANT argument contains an fixed-point decimal value or integer value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_DOUBLE(variant_expr)

Returns TRUE if its VARIANT argument contains an a floating-point value, fixed-point decimal, or integer value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_REAL(variant_expr)

Returns TRUE if its VARIANT argument contains an a floating-point value, fixed-point decimal, or integer value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_INTEGER(variant_expr)

Returns TRUE if its VARIANT argument contains an integer value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_NULL_VALUE(variant_expr)

Returns TRUE if its VARIANT argument contains an NULL value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_OBJECT(variant_expr)

Returns TRUE if its VARIANT argument contains an OBJECT value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_TIME(variant_expr)

Returns TRUE if its VARIANT argument contains an TIME value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_TIMESTAMP_LTZ(variant_expr)

Verifies whether a VARIANT value contains the respective TIMESTAMP value

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_TIMESTAMP_NTZ(variant_expr)

Verifies whether a VARIANT value contains the respective TIMESTAMP value

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_TIMESTAMP_TZ(variant_expr)

Verifies whether a VARIANT value contains the respective TIMESTAMP value

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

TYPEOF(column)

Reports the type of a value stored in a VARIANT column. The type is returned as a string.

- **column**: The column to detect the type of

REGEXP_COUNT(subject, pattern [, position, parameters])

Returns the number of times that a pattern occurs in a string.

- **subject**: Subject to match.
- **pattern**: Pattern to match.
- **position**: Number of characters from the beginning of the string where the function starts searching for matches. Default: 1 (the search for a match starts at the first character on the left)
- **parameters**: String of one or more characters that specifies the parameters used for searching for matches

REGEXP_INSTR(subject, pattern [, position, occurrence, option, parameters])

Returns the position of the specified occurrence of the regular expression pattern in the string subject. If no match is found, returns 0.

- **subject**: Subject to match.

- **pattern:** Pattern to match.
- **position:** Number of characters from the beginning of the string where the function starts searching for matches. Default: 1
- **occurrence:** Specifies which occurrence of the pattern to match. The function skips the first occurrence -1 matches. Default: 1
- **option:** Specifies whether to return the offset of the first character of the match (0) or the offset of the first character following the end of the match (1). Default: 0
- **parameters:** String of one or more characters that specifies the parameters used for searching for matches. Supported values: c, i, m, e, s

REGEXP_LIKE(subject, pattern [, parameters])

Returns true if the subject matches the pattern. Both expressions must be text expressions.

- **subject:** Subject to match.
- **pattern:** Pattern to match.
- **parameters:** String of one or more characters that specifies the parameters used for searching for matches. Supported values: c, i, m, e, s

REGEXP_REPLACE(subject, pattern [, replacement [, position [, occurrence [, parameters]]]])

Returns the subject with the specified pattern (or all occurrences of the pattern) either removed or replaced by a replacement string. If no matches are found, returns the original subject.

- **subject:** Subject to match.
- **pattern:** Pattern to match.
- **replacement:** String the replaces the substrings matched by the pattern. If an empty string is specified, the function removes all matched patterns and returns the resulting string. Default: '' (empty string).
- **position:** Number of characters from the beginning of the string where the function starts searching for matches. Default: 1 (the search for a match starts at the first character on the left)

- **occurrence:** Specifies which occurrence of the pattern to replace. If 0 is specified, all occurrences are replaced. Default: 0 (all occurrences)
- **parameters:** String of one or more characters that specifies the parameters used for searching for matches. Supported values: c, i, m, e, s

REGEXP_SUBSTR(subject, pattern [, position, occurrence, parameters])

Returns the position of the specified occurrence of the regular expression pattern in the string subject. If no match is found, returns 0.

- **subject:** Subject to match.
- **pattern:** Pattern to match.
- **position:** Number of characters from the beginning of the string where the function starts searching for matches. Default: 1
- **occurrence:** Specifies which occurrence of the pattern to match. The function skips the first occurrence -1 matches. Default: 1
- **parameters:** String of one or more characters that specifies the parameters used for searching for matches. Supported values: c, i, m, e, s

RLIKE(subject, pattern, parameters)

Returns true if the subject matches the specified pattern. Both inputs must be text expressions.

- **subject:** Subject to match
- **pattern:** Pattern to match.
- **parameters:** String of one or more characters that specifies the parameters used for searching for matches. Supported values: c, i, m, e, s

ASCII(input)

Returns the ASCII code for the first character of a string. If the string is empty, a value of 0 is returned.

- **input:** The string for which the ASCII code for the first character in the string is

returned.

BIT_LENGTH(string_or_binary)

Returns the length of a string or binary value in bits. Snowflake doesn't use fractional bytes so length is always calculated as 8 * OCTET_LENGTH.

- **string_or_binary:** The string or binary value for which the length is returned.

CHARINDEX(expr1, expr2 [, start_pos])

Searches for the first occurrence of the first argument in the second argument and, if successful, returns the position (1-based) of the first argument in the second argument.

- **expr1:** A string or binary expression representing the value we are looking for.
- **expr2:** A string or binary expression representing the value in which we are searching.
- **start_pos:** A number indicating the position from where to start the search (with 1 representing the start of expr1). Default: 1

CHAR(input)

Converts a Unicode code point (including 7-bit ASCII) into the character that matches the input Unicode. If an invalid code point is specified, an error is returned.

- **input:** The Unicode code point for which the character is returned.

CHR(input)

Converts a Unicode code point (including 7-bit ASCII) into the character that matches the input Unicode. If an invalid code point is specified, an error is returned.

- **input:** The Unicode code point for which the character is returned.

CONCAT(expr1, expr2)

Concatenates two strings or two binary values. If one of the them is null, the result is also null.

- **expr1**: A string to concatenate
- **expr2**: A string to concatenate

CONTAINS(expr1, expr2)

Returns true if expr1 contains expr2. Both expressions must be text or binary expressions.

- **expr1**: A string or binary expression representing the value we are looking fo
- **expr2**: A string or binary expression representing the value in which we are searching.

EDITDISTANCE(expr1, expr2)

Computes the Levenshtein distance between two input strings. It is the number of single-character insertions, deletions or substitutions needed to convert one string to another.

- **expr1**: A string
- **expr2**: A string

ENDSWITH(expr1, expr2)

Returns TRUE if the first expression ends with second expression. Both expressions must be text or binary expressions.

- **expr1**: A string or binary expression representing the value we are looking fo
- **expr2**: A string or binary expression representing the value in which we are searching.

ILIKE(subject, pattern [, escape])

Allows matching of strings based on comparison with a pattern. Unlike the LIKE function, string matching is case-insensitive. LIKE, ILIKE, and RLIKE all perform similar operations; however, RLIKE uses POSIX EXE (Extended Regular Expression) syntax instead of the SQL pattern syntax used by LIKE and ILIKE.

- **subject**: Subject to match.

- **pattern:** Pattern to match.
- **escape:** Character(s) inserted in front of a wildcard character to indicate that the wildcard should be interpreted as a regular character and not as a wildcard.

INITCAP(expr [, delimiters])

Returns the input string (expr) with the first letter of each word in uppercase and the subsequent letters in lowercase.

- **expr:** The input string
- **delimiters:** an optional argument specifying a string of one or more characters that INITCAP uses as separators for words in the input expression:

INSERT(base_expr, pos, len, insert_expr)

Replaces a substring of the specified length, starting at the specified position, with a new string or binary value. This function should not be confused with the INSERT DML command.

- **base_expr:** The string or BINARY expression for which you want to insert/replace characters.
- **pos:** The offset at which to start inserting characters. This is 1-based, not 0-based.
- **len:** The number of characters (starting at pos) that you want to replace. Valid values range from 0 to the number of characters between pos and the end of the string.
- **insert_expr:** The string to insert into the base_expr. If this string is empty, and if len is greater than zero, then effectively the operation becomes a delete (some characters are deleted, and none are added).

LEFT(string_expr, length_expr)

Returns a leftmost substring of its input. LEFT(STR,N) is equivalent to SUBSTR(STR,1,N).

- **string_expr:** The string expression
- **length_expr:** The length to extract

LENGTH(expr)

Returns the length of a input string or binary value. For strings, the length is the number of characters, and UTF-8 characters are counted as a single character. For binary, the length is the number of bytes.

- **expr**: The expression to measure

LIKE(subject, pattern [, escape])

Allows case-sensitive matching of strings based on comparison with a pattern. For case-insensitive matching, use ILIKE instead.

- **subject**: Subject to match.
- **pattern**: Pattern to match.
- **escape**: Character(s) inserted in front of a wildcard character to indicate that the wildcard should be interpreted as a regular character and not as a wildcard.

LOWER(expr)

Returns the input string (expr) with all characters converted to lowercase.

- **expr**: The input string

LPAD(base, n [, pad])

Left-pads a string with characters from another string, or left-pads a binary value with bytes from another binary value.

- **base**: The base string to pad
- **n**: The number of characters to pad
- **pad**: The character to pad

LTRIM(expr [, characters])

Removes leading characters, including whitespace, from a string.

- **expr:** The string expression to be trimmed.
- **characters:** One or more characters to remove from the left side of expr. The default value is ' ' (a single blank space character),

OCTET_LENGTH(string_or_binary)

Returns the length of a string or binary value in bytes. This will be the same as LENGTH for ASCII strings and greater than LENGTH for strings using Unicode code points. For binary, this is always the same as LENGTH.

- **string_or_binary:** The string or binary value for which the length is returned.

PARSE_IP(expr, type [, permissive])

Returns a JSON object consisting of all the components from a valid INET (Internet Protocol) or CIDR (Classless Internet Domain Routing) IPv4 or IPv6 string.

- **expr:** A string expression.
- **type:** Identifies the type of IP address. Supports either INET or CIDR; case-insensitive.
- **permissive:** Flag that determines how parse errors are handled:

PARSE_URL(string [, permissive])

Returns a JSON object consisting of all the components (fragment, host, path, port, query, scheme) in a valid input URL/URI.

- **string:** String to parse.
- **permissive:** Flag that determines how parse errors are handled

POSITION(expr1, expr2 [, start_pos])

Searches for the first occurrence of the first argument in the second argument and, if successful, returns the position (1-based) of the first argument in the second argument.

- **expr1:** A string or binary expression representing the value we are looking for.

- **expr2:** A string or binary expression representing the value in which we are searching.
- **start_pos:** A number indicating the position from where to start the search (with 1 representing the start of expr2).

POSITION(expr1 IN expr2)

Searches for the first occurrence of the first argument in the second argument and, if successful, returns the position (1-based) of the first argument in the second argument.

- **expr1:** A string or binary expression representing the value we are looking for.
- **expr2:** A string or binary expression representing the value in which we are searching.

REPEAT(input, n)

Builds a string by repeating the input for the specified number of times.

- **input:** The input string from which the output string is built.
- **n:** The number of times the input string should be repeated. The minimum valid number is 0 (which results in an empty string).

REPLACE(subject, pattern [, replacement])

Removes all occurrences of a specified substring, and optionally replaces them with another string

- **subject:** The subject is the string in which to do the replacements. Typically, this is a column, but it can be a literal.
- **pattern:** This is the substring that you want to replace. Typically, this is a literal, but it can be a column or expression. Note that this is not a "regular expression"; if you want to use regular expressions to search for a pattern, use the REGEXP_REPLACE function.
- **replacement:** This is the value used as a replacement for the pattern. If this is omitted, or is an empty string, then the REPLACE function simply deletes all occurrences of the pattern.

REVERSE(subject)

Reverses the order of characters in a string, or of bytes in a binary value.

- **subject:** The string to reverse

RIGHT(string_expr, length_expr)

Returns a rightmost substring of its input.

- **string_expr:** The string expression
- **length_expr:** The length to extract

RPAD(base, n [, pad])

Right-pads a string or binary value with characters from another string.

- **base:** The base string to pad
- **n:** The number of times to pad
- **pad:** The string to pad with

RTRIM(expr [, characters])

Removes trailing characters, including whitespace, from a string.

- **expr:** The string expression to be trimmed.
- **characters:** One or more characters to remove from the right side of expr:

RTRIMMED_LENGTH(string_expr)

Returns the length of its argument, minus trailing whitespace, but including leading whitespace.

- **string_expr:** The string expression to measure

SPACE(n)

Builds a string consisting of the specified number of blank spaces.

- **n**: The number of blank spaces used to build the string.

SPLIT(string, separator)

Splits a given string with a given separator and returns the result in an array of strings. Contiguous split strings in the source string, or the presence of a split string at the beginning or end of the source string, results in an empty string in the output. An empty separator string results in an array containing only the source string. If either parameter is a NULL, a NULL is returned.

- **string**: Text to be split into parts.
- **separator**: Text to split string by.

SPLIT_TO_TABLE(string, delimiter)

Splits a string with the given delimiter and flattens the results into rows.

- **string**: Text to be split into parts.
- **delimiter**: Text to split string by.

STRTOK(string [, delimiter] [, partNr])

Tokenizes a given string and returns the requested part.

- **string**: Text to be tokenized.
- **delimiter**: Text representing the set of delimiters to tokenize on. Each character in the delimiter string is a delimiter. If the delimiter is empty, and the string is empty, then the function returns NULL. If the delimiter is empty, and the string is non empty, then the whole string will be treated as one token. The default value of the delimiter is a single space character.
- **partNr**: Requested token (1-based, i.e. the first token is token number 1, not token number 0). If the token number is out of range, then NULL is returned. The default value is 1.

SPLIT_PART(string, delimiter, partNr)

Splits a given string and returns the requested part. If a part does not exist, an empty string is returned. If any parameter is NULL, NULL is returned.

- **string:** Text to be split into parts.
- **delimiter:** Text representing the delimiter to split by.
- **partNr:** Requested part of the split (1-based). 0 is treated as 1. If the value is negative, the parts are counted from the right side of the string.

STARTSWITH(expr1, expr2)

Returns true if expr1 starts with expr2. Both expressions must be text or binary expressions.

- **expr1:** A string or binary expression representing the value we are looking for
- **expr2:** A string or binary expression representing the value in which we are searching.

SUBSTRING(base_expr, start_expr [, length_expr])

Returns the portion of the string or binary value from base_expr, starting from the character/byte specified by start_expr, with optionally limited length.

- **base_expr:** The base string
- **start_expr:** The position to start from
- **length_expr:** Up to length_expr characters/bytes are returned, otherwise all the characters until the end of the string or binary value are returned.

COMPRESS(input, method)

Returns the portion of the string or binary value from base_expr, starting from the character/byte specified by start_expr, with optionally limited length.

- **input:** A BINARY or string value (or expression) to be compressed.
- **method:** A string with compression method and optional compression level. Supported methods are: SNAPPY. ZLIB. ZSTD. BZ2. The compression level is specified

in parentheses, for example: `zlib(1)`. The compression level is a non-negative integer. 0 means default level (same as omitting the compression level). The compression level is ignored if the method doesn't support compression levels.

DECOMPRESS_BINARY(input, method)

Returns the portion of the string or binary value from `base_expr`, starting from the character/byte specified by `start_expr`, with optionally limited length.

- **input:** A BINARY value (or expression) with data that was compressed using one of the compression methods specified in COMPRESS. If you attempt to decompress a compressed string, rather than a compressed BINARY value, you will not get an error; the function will return a BINARY value. See the Usage Notes below for details.
- **method:** The compression method originally used to compress the input.

DECOMPRESS_STRING(input, method)

Decompresses the compressed BINARY input parameter to a string.

- **input:** A BINARY value (or expression) with data that was compressed using one of the compression methods specified in COMPRESS.
- **method:** The compression method originally used to compress the input.

SUBSTR(base_expr, start_expr [, length_expr])

Returns the portion of the string or binary value from `base_expr`, starting from the character/byte specified by `start_expr`, with optionally limited length.

- **base_expr:** The base string
- **start_expr:** The position to start from
- **length_expr:** Up to `length_expr` characters/bytes are returned, otherwise all the characters until the end of the string or binary value are returned.

TRANSLATE(subject, sourceAlphabet, targetAlphabet)

Translates `subject` from the characters in `sourceAlphabet` to the characters in `targetAlphabet`.

- **subject:** A string expression that is translated. If a character in subject is not contained in sourceAlphabet, the character is added to the result without any translation.
- **sourceAlphabet:** A string with all characters that are modified by this function. Each character is either translated to the corresponding character in the targetAlphabet or omitted in the result if the targetAlphabet has no corresponding character (i.e. has less characters than the sourceAlphabet).
- **targetAlphabet:** A string with all characters that are used to replace characters from the sourceAlphabet.

TRIM(expr [, characters])

Removes leading and trailing characters from a string.

- **expr:** A string expression to be trimmed.
- **characters:** One or more characters to remove from the left and right side of expr: The default value is ' ' (a single blank space character), i.e. if no characters are specified, all leading and trailing blank spaces are removed.

UNICODE(input)

Returns the Unicode code point for the first Unicode character in a string. If the string is empty, a value of 0 is returned.

- **input:** The string for which the Unicode code point for the first character in the string is returned.

UPPER(expr)

Returns the input string expr with all characters converted to uppercase.

- **expr:** The input string

BASE64_DECODE_BINARY(input [, alphabet])

Decodes a Base64-encoded string to a binary.

- **input:** A Base64-encoded string expression.
- **alphabet:** A string consisting of up to three ASCII characters

BASE64_DECODE_STRING(input [, alphabet])

Decodes a Base64-encoded string to a binary.

- **input:** A Base64-encoded string expression.
- **alphabet:** A string consisting of up to three ASCII characters

BASE64_ENCODE(input [, max_line_length] [, alphabet])

Encodes the input (string or binary) using Base64 encoding.

- **input:** A string or binary expression to be encoded.
- **max_line_length:** A positive integer that specifies the maximum number of characters in a single line of the output.
- **alphabet:** A string consisting of up to three ASCII characters

HEX_DECODE_BINARY(input)

Decodes a hex-encoded string to a binary.

- **input:** A string expression containing only hexadecimal digits. Typically, this input string is generated by calling the function HEX_ENCODE.

HEX_DECODE_STRING(input)

Decodes a hex-encoded string to a string.

- **input:** A hex-encoded string expression. Typically the input was created by a call to HEX_ENCODE.

HEX_ENCODE(input [, case])

Encodes the input using hexadecimal (also 'hex' or 'base16') encoding. The result is comprised of 16 different symbols: The numbers '0' to '9' as well as the letters 'A' to 'F'

- **input:** A binary or string expression to be encoded.
- **case:** This optional boolean argument controls the case of the letters ('A', 'B', 'C', 'D', 'E' and 'F') used in the encoding. The default value is 1 and indicates that uppercase letters are used. The value 0 indicates that lowercase letters are used. All other values are illegal and result in an error.

TRY_BASE64_DECODE_BINARY(input [, alphabet])

A special version of BASE64_DECODE_BINARY that returns a NULL value if an error occurs during decoding.

- **input:** A Base64-encoded string expression.
- **alphabet:** A string consisting of up to three ASCII characters

TRY_BASE64_DECODE_STRING(input [, alphabet])

A special version of BASE64_DECODE_BINARY that returns a NULL value if an error occurs during decoding

- **input:** A Base64-encoded string expression.
- **alphabet:** A string consisting of up to three ASCII characters

TRY_HEX_DECODE_BINARY(input)

A special version of HEX_DECODE_BINARY that returns a NULL value if an error occurs during decoding.

- **input:** A string expression containing only hexadecimal digits. Typically, this input string is generated by calling the function HEX_ENCODE.

TRY_HEX_DECODE_STRING(input)

A special version of HEX_DECODE_BINARY that returns a NULL value if an error occurs during decoding.

- **input:** A hex-encoded string expression. Typically the input was created by a call to

HEX_ENCODE.

MD5(msg)

Returns a 32-character hex-encoded string containing the 128-bit MD5 message digest.

- **msg**: A string expression, the message to be hashed.

MD5_HEX(msg)

Returns a 32-character hex-encoded string containing the 128-bit MD5 message digest.

- **msg**: A string expression, the message to be hashed

MD5_BINARY(msg)

Returns a 16-byte BINARY value containing the 128-bit MD5 message digest.

- **msg**: A string expression, the message to be hashed.

MD5_NUMBER(msg)

Returns the 128-bit MD5 message digest interpreted as a signed 128-bit big endian number. This representation is useful for maximally efficient storage and comparison of MD5 digests.

- **msg**: A string expression, the message to be hashed.

SHA1(msg)

Returns a 40-character hex-encoded string containing the 160-bit SHA-1 message digest.

- **msg**: A string expression, the message to be hashed.

SHA1_HEX(msg)

Returns a 40-character hex-encoded string containing the 160-bit SHA-1 message digest.

- **msg:** A string expression, the message to be hashed.

SHA1_BINARY(msg)

Returns a 20-byte binary containing the 160-bit SHA-1 message digest.

- **msg:** A string expression, the message to be hashed.

SHA2(msg [, digest_size])

Returns a hex-encoded string containing the N-bit SHA-2 message digest, where N is the specified output digest size.

- **msg:** A string expression, the message to be hashed
- **digest_size:** Size (in bits) of the output, corresponding to the specific SHA-2 function used to encrypt the string

SHA2_HEX(msg [, digest_size])

Returns a hex-encoded string containing the N-bit SHA-2 message digest, where N is the specified output digest size.

- **msg:** A string expression, the message to be hashed
- **digest_size:** Size (in bits) of the output, corresponding to the specific SHA-2 function used to encrypt the string

SHA2_BINARY(msg [, digest_size])

Returns a binary containing the N-bit SHA-2 message digest, where N is the specified output digest size.

- **msg:** A string expression, the message to be hashed
- **digest_size:** Size (in bits) of the output, corresponding to the specific SHA-2 function used to encrypt the string

ANY_VALUE([DISTINCT] expr)

Returns some value of the expression from the group. The result is non-deterministic.

- **expr:** A group of values to choose from

AVG([DISTINCT] expr)

Returns the average of non-NULL records. If all records inside a group are NULL, NULL is returned.

- **expr:** Expression is an expression that evaluates to a numeric data type (INTEGER, FLOAT, DECIMAL, etc.).

CORR(y, c)

Returns the correlation coefficient for non-null pairs in a group. It is computed for non-null pairs using the following formula: $\text{COVAR_POP}(y, x) / (\text{STDDEV_POP}(x) * \text{STDDEV_POP}(y))$

- **y:**
- **c:**

COUNT([DISTINCT] expr1 [, expr2])

Returns either the number of non-NULL records for the specified columns, or a total number of records.

- **expr1:** A column name.
- **expr2:** You may include additional column name(s) if you wish. For example, you could list the number of distinct combinations of last name and first name.

COVAR_POP(x, y)

Returns the population covariance for non-null pairs in a group. It is computed for non-null pairs using the following formula: $(\text{SUM}(x*y) - \text{SUM}(x) * \text{SUM}(y) / \text{COUNT}(*)) / \text{COUNT}(*)$
Where x is the independent variable and y is the dependent variable.

- **x:**

- **y:**

COVAR_SAMP(y, c)

Returns the sample covariance for non-null pairs in a group. It is computed for non-null pairs using the following formula: $(\text{SUM}(x*y) - \text{SUM}(x) * \text{SUM}(y) / \text{COUNT}(*)) / (\text{COUNT}(*) - 1)$ Where x is the independent variable and y is the dependent variable.

- **y:**
- **x:**

LISTAGG([DISTINCT] expr, delimiter)

Returns the concatenated input values, separated by the delimiter string.

- **expr:** The expression (typically a column name) that determines the values to be put into the list. The expression should evaluate to a string, or to a data type that can be cast to string.
- **delimiter:** A string, or an expression that evaluates to a string. In practice, this is usually a single-character string. The string should be surrounded by single quotes, as shown in the examples below.

MAX(expr)

Returns the minimum or maximum value for the records within expr. NULL values are ignored unless all the records are NULL, in which case a NULL value is returned.

- **expr:**

MIN(expr)

Returns the minimum or maximum value for the records within expr. NULL values are ignored unless all the records are NULL, in which case a NULL value is returned.

- **expr:**

MEDIAN(expr)

Determines the median of a set of values.

- **expr:** The expression must evaluate to a numeric data type (INTEGER, FLOAT, DECIMAL, or equivalent).

PERCENTILE_CONT(percentile)

Return a percentile value based on a continuous distribution of the input column (specified in `order_by_expr`). If no input row lies exactly at the desired percentile, the result is calculated using linear interpolation of the two nearest input values. NULL values are ignored in the calculation.

- **percentile:** The percentile of the value that you want to find. The percentile must be a constant between 0.0 and 1.0. For example, if you want to find the value at the 90th percentile, specify 0.9.

PERCENTILE_DISC(percentile)

Returns a percentile value based on a discrete distribution of the input column (specified in `order_by_expr`). The returned value is that whose row has the smallest CUME_DIST value that is greater than or equal to the given percentile. NULL values are ignored in the calculation.

- **percentile:** The percentile of the value that you want to find. The percentile must be a constant between 0.0 and 1.0. For example, if you want to find the value at the 90th percentile, specify 0.9.

STDDEV(expr)

Returns the sample standard deviation (square root of sample variance) of non-NULL values. If all records inside a group are NULL, returns NULL.

- **expr:** An expression that evaluates to a numeric value.

STDDEV_SAMP([DISTINCT] expr)

Returns the sample standard deviation (square root of sample variance) of non-NULL values. If all records inside a group are NULL, returns NULL.

- **expr:** An expression that evaluates to a numeric value (integer, floating point, or fixed point).

STDDEV_POP([DISTINCT] x)

Returns the population standard deviation (square root of variance) of non-NULL values. If all records inside a group are NULL, returns NULL.

- **x:**

SUM([DISTINCT] expr)

Returns the sum of non-NULL records for expr. If the DISTINCT keyword is used, the sum of unique non-null values is computed. If all records inside a group are NULL, a value of NULL is returned.

- **expr:** The expression to sum

VAR_POP([DISTINCT] x)

Returns the population variance of non-NULL records in a group. If all records inside a group are NULL, a NULL is returned.

- **x:**

VARIANCE_POP([DISTINCT] x)

Returns the population variance of non-NULL records in a group. If all records inside a group are NULL, a NULL is returned.

- **x:**

VAR_SAMP([DISTINCT] x)

Returns the sample variance of non-NULL records in a group. If all records inside a group are NULL, a NULL is returned.

- **x:**

VARIANCE([DISTINCT] x)

Returns the sample variance of non-NULL records in a group. If all records inside a group are NULL, a NULL is returned.

- **x:**

VARIANCE_SAMP([DISTINCT] x)

Returns the sample variance of non-NULL records in a group. If all records inside a group are NULL, a NULL is returned.

- **x:**

CUME_DIST()

Finds the cumulative distribution of a value with regard to other values within the same window partition.

DENSE_RANK()

Returns the rank of a value within a group of values, without gaps in the ranks. The rank value starts at 1 and continues up sequentially. If two values are the same, they will have the same rank.

FIRST_VALUE(expr)

Returns the first value within an ordered group of values.

- **expr:**

LAG(expr [, offset, default])

Accesses data in a previous row in the same result set without having to join the table to itself.

- **expr:** The string expression to be returned.
- **offset:** The number of rows backward from the current row from which to obtain a value; e.g., an offset of 2 returns the expr value with an interval of 2 rows.
- **default:** The expression to return when the offset goes out of the bounds of the window. Supports any expression whose type is compatible with expr

LAST_VALUE(expr)

Returns the last value within an ordered group of values.

- **expr:**

LEAD(expr, offset, default)

Accesses data in a subsequent row in the same result set without having to join the table to itself.

- **expr:** The string expression to be returned.
- **offset:** The number of rows forward from the current row from which to obtain a value; e.g., an offset of 2 returns the expr value with an interval of 2 rows.
- **default:** The expression to return when the offset goes out of the bounds of the window. Supports any expression whose type is compatible with expr.

NTH_VALUE(expr, n)

Returns the nth value (up to 1000) within an ordered group of values.

- **expr:**
- **n:** Input value n cannot be greater than 1000.

NTILE(constant_value)

Divides an ordered data set equally into the number of buckets specified by constant_value. Buckets are sequentially numbered 1 through constant_value.

- **constant_value:** The desired number of buckets; must be a positive integer value.

PERCENT_RANK()

Returns the relative rank of a value within a group of values.

RANK()

Returns the rank of a value within an ordered group of values.

ROW_NUMBER()

Returns a unique row number for each row within a window partition.

WIDTH_BUCKET(expr, min_value, max_value, num_buckets)

Constructs equi-width histograms, in which the histogram range is divided into intervals of identical size, and returns the bucket number into which the value of an expression falls, after it has been evaluated. The function returns an integer value or null (if any input is null).

- **expr:** The expression for which the histogram is created. This expression must evaluate to a numeric value or to a value that can be implicitly converted to a numeric value.
- **min_value:** The low and high end points of the acceptable range for the expression. The end points must also evaluate to numeric values and not be equal.
- **max_value:** The low and high end points of the acceptable range for the expression. The end points must also evaluate to numeric values and not be equal.
- **num_buckets:** The desired number of buckets; must be a positive integer value. A value from the expression is assigned to each bucket, and the function then returns the corresponding bucket number.

BITAND_AGG(expr)

Returns the bitwise AND value of all non-NULL numeric records in a group. If all records inside the group are NULL, or if the group is empty, the function returns NULL.

- **expr:**

BITOR_AGG(expr)

Returns the bitwise OR value of all non-NULL numeric records in a group. If all records inside the group are NULL, or if the group is empty, the function returns NULL.

- **expr:**

BITXOR_AGG([DISTINCT] expr)

Returns the bitwise XOR value of all non-NULL numeric records in a group. If all records inside the group are NULL, or if the group is empty, the function returns NULL.

- **expr:**

ARRAY_AGG([DISTINCT] expr)

Returns the input values, pivoted into an ARRAY. If the input is empty, an empty ARRAY is returned.

- **expr:** The expression (typically a column name) that determines the values to be put into the list

REGR_AVGX(y, x)

Returns the average of the independent variable for non-null pairs in a group, where x is the independent variable and y is the dependent variable.

- **y:**
- **x:**

REGR_COUNT(y, x)

Returns the number of non-null number pairs in a group.

- **y:**
- **x:**

REGR_INTERCEPT(y, x)

Returns the intercept of the univariate linear regression line for non-null pairs in a group. It is computed for non-null pairs using the following formula: $AVG(y) - REGR_SLOPE(y, x) * AVG(x)$
Where x is the independent variable and y is the dependent variable

- **y:**
- **x:**

REGR_R2(y, x)

Returns the coefficient of determination for non-null pairs in a group.

- **y:**
- **x:**

REGR_SLOPE(y, x)

Returns the slope of the linear regression line for non-null pairs in a group

- **y:**
- **x:**

REGR_SXX(y, x)

Returns $REGR_COUNT(y, x) * VAR_POP(x)$ for non-null pairs.

- **y:**
- **x:**

REGR_SXY(y, x)

Returns $\text{REGR_COUNT}(\text{expr1}, \text{expr2}) * \text{COVAR_POP}(\text{expr1}, \text{expr2})$ for non-null pairs.

- **y:**
- **x:**

REGR_SYY(y, x)

Returns $\text{REGR_COUNT}(y, x) * \text{VAR_POP}(y)$ for non-null pairs.

- **y:**
- **x:**

REGR_AVGY(y, x)

Returns the average of the dependent variable for non-null pairs in a group, where x is the independent variable and y is the dependent variable.

- **y:**
- **x:**

APPROX_COUNT_DISTINCT([DISTINCT] expr [, ...])

Uses HyperLogLog to return an approximation of the distinct cardinality of the input.

- **expr:**

HLL([DISTINCT] expr [, ...])

Uses HyperLogLog to return an approximation of the distinct cardinality of the input.

- **expr:**

HLL_COMBINE([DISTINCT] state)

Combines (merges) input states into a single output state. This allows scenarios where HLL_ACCUMULATE is run over horizontal partitions of the same table, producing an algorithm state for each table partition. These states can later be combined using HLL_COMBINE, producing the same output state as a single run of HLL_ACCUMULATE over the entire table.

- **state:** An expression that contains state information generated by a call to HLL_ACCUMULATE.

HLL_ESTIMATE(state)

Returns the cardinality estimate for the given HyperLogLog state. A HyperLogLog state produced by HLL_ACCUMULATE and HLL_COMBINE can be used to compute a cardinality estimate using the HLL_ESTIMATE function.

- **state:** An expression that contains state information generated by a call to HLL_ACCUMULATE or HLL_COMBINE.

HLL_ACCUMULATE([DISTINCT] expr)

Returns the HyperLogLog state at the end of aggregation.

- **expr:**

HLL_EXPORT(binary_expr)

Converts input in BINARY format to OBJECT format. The HyperLogLog states operated on by HLL_ACCUMULATE, HLL_COMBINE, and HLL_ESTIMATE are in a proprietary binary format that may change in future versions of Snowflake. For long-term storage of HyperLogLog states, and for integration with external tools, Snowflake supports converting states from the BINARY format to an OBJECT (which can be printed and exported as JSON), and vice versa.

- **binary_expr:**

HLL_IMPORT(obj)

Converts input in OBJECT format to BINARY format. The HyperLogLog states operated on by HLL_ACCUMULATE, HLL_COMBINE, and HLL_ESTIMATE are in a proprietary binary format that may change in future versions of Snowflake. For long-term storage of HyperLogLog states, and for integration with external tools, Snowflake supports using HLL_IMPORT to convert states from an OBJECT format to BINARY, and vice versa.

- **obj:**

APPROXIMATE_JACCARD_INDEX([DISTINCT] expr [, ...])

Returns an estimation of the similarity (Jaccard index) of inputs based on their MinHash states. For more information about Jaccard indexes and the related function MINHASH, see Estimating Similarity of Two or More Sets.

- **expr:** The expression(s) should be one or more MinHash states returned by calls to the MINHASH function. In other words, the expressions must be MinHash state information, not the column or expression for which you want the approximate similarity. (The example below helps make this clear.)

APPROXIMATE_SIMILARITY([DISTINCT] expr [, ...])

Returns an estimation of the similarity (Jaccard index) of inputs based on their MinHash states. For more information about Jaccard indexes and the related function MINHASH, see Estimating Similarity of Two or More Sets.

- **expr:** The expression(s) should be one or more MinHash states returned by calls to the MINHASH function. In other words, the expressions must be MinHash state information, not the column or expression for which you want the approximate similarity. (The example below helps make this clear.)

MINHASH(k, [DISTINCT] expr [, ...])

Returns a MinHash state containing an array of size k constructed by applying k number of different hash functions to the input rows and keeping the minimum of each hash function. This MinHash state can then be input to the APPROXIMATE_SIMILARITY function to estimate the similarity with one or more other MinHash states.

- **k**: specifies the number of hash functions to be created. The larger the value, the better the approximation; however, this value has a linear impact on the computation time for estimating similarity using APPROXIMATE_SIMILARITY. The suggested value is 100.
- **expr**:

MINHASH_COMBINE([DISTINCT] state)

Combines input MinHash states into a single MinHash output state. This Minhash state can then be input to the APPROXIMATE_SIMILARITY function to estimate the similarity with other MinHash states.

- **state**: Input MinHash state must have MinHash arrays of equal length.

APPROX_TOP_K(expr, k, counters)

Uses Space-Saving to return an approximation of the most frequent values in the input, along with their approximate frequencies. The output is a JSON array of arrays. In the inner arrays, the first entry is a value in the input, and the second entry corresponds to its estimated frequency in the input. The outer array contains k items, sorted by descending frequency.

- **expr**: The expression (e.g. column name) for which you want to find the most common values.
- **k**: The number of values whose counts you want approximated. For example, if you want to see the top 10 most common values, then set k to 10.
- **counters**: This is the maximum number of distinct values that can be tracked at a time during the estimation process. For example, if counters is set to 100000, then the algorithm tracks 100,000 distinct values, attempting to keep the 100,000 most frequent values.

APPROX_TOP_K_ACCUMULATE(expr, counters)

Returns the Space-Saving summary at the end of aggregation. (For more information about the Space-Saving summary, see Estimating Frequent Values.) The function APPROX_TOP_K discards its internal, intermediate state when the final cardinality estimate is returned. However, in certain advanced use cases, such as estimating incremental frequent values

during bulk loading, you might want to keep the intermediate state, in which case you would use `APPROX_TOP_K_ACCUMULATE` instead of `APPROX_TOP_K`.

- **expr:** The expression (e.g. column name) for which you want to find the most common values.
- **counters:** This is the maximum number of distinct values that can be tracked at a time during the estimation process. For example, if counters is set to 100000, then the algorithm tracks 100,000 distinct values, attempting to keep the 100,000 most frequent values

APPROX_TOP_K_COMBINE(state [, counters])

Combines (merges) input states into a single output state. This allows scenarios where `APPROX_TOP_K_ACCUMULATE` is run over horizontal partitions of the same table, producing an algorithm state for each table partition. These states can later be combined using `APPROX_TOP_K_COMBINE`, producing the same output state as a single run of `APPROX_TOP_K_ACCUMULATE` over the entire table.

- **state:** An expression that contains state information generated by a call to `APPROX_TOP_K_ACCUMULATE`.
- **counters:** This is the maximum number of distinct values that can be tracked at a time during the estimation process. For example, if counters is set to 100000, then the algorithm tracks 100,000 distinct values, attempting to keep the 100,000 most frequent values.

APPROX_TOP_K_ESTIMATE(state [, k])

Returns the approximate most frequent values and their estimated frequency for the given Space-Saving state. (For more information about the Space-Saving summary, see [Estimating Frequent Values](#).)

- **state:** An expression that contains state information generated by a call to `APPROX_TOP_K_ACCUMULATE` or `APPROX_TOP_K_COMBINE`.
- **k:** The number of values whose counts you want approximated. For example, if you want to see the top 10 most common values, then set k to 10.

APPROX_PERCENTILE(expr, percentile)

Returns an approximated value for the desired percentile (i.e. if column *c* has *n* numbers, then APPROX_PERCENTILE(*c*, *p*) returns a number such that approximately $n * p$ of the numbers in *c* are smaller than the returned number).

- **expr:** A valid expression, such as a column name, that evaluates to a numeric value.
- **percentile:** A constant real value greater than or equal to 0.0 and less than 1.0. This indicates the percentile (from 0 to 99.999...). E.g. The value 0.65 indicates the 65th percentile.

APPROX_PERCENTILE_ACCUMULATE(expr)

Returns the internal representation of the t-Digest state (as a JSON object) at the end of aggregation. (For more information about t-Digest, see: Estimating Percentile Values.)

- **expr:** A valid expression, such as a column name, that evaluates to a numeric value.

APPROX_PERCENTILE_COMBINE(state)

Combines (merges) percentile input states into a single output state.

- **state:** An expression that contains state information generated by a call to APPROX_PERCENTILE_ACCUMULATE.

APPROX_PERCENTILE_ESTIMATE(state, percentile)

Returns the desired approximated percentile value for the specified t-Digest state.

- **state:** An expression that contains state information generated by a call to APPROX_PERCENTILE_ACCUMULATE or APPROX_PERCENTILE_COMBINE.
- **percentile:** A constant real value greater than or equal to 0.0 and less than 1.0. This indicates the percentile (from 0 to 99.999...). E.g. The value 0.65 indicates the 65th percentile.

GROUPING(expr1 [, ...])

Describes which of a list of expressions are grouped in a row produced by a GROUP BY query.

- **expr1:**

GROUPING_ID(expr1 [, ...])

Describes which of a list of expressions are grouped in a row produced by a GROUP BY query.

- **expr1:**

SYSTEMABORT_SESSION(session_id)

Aborts the specified session.

- **session_id:** Identifier for the session to abort.

SYSTEMABORT_TRANSACTION(transaction_id)

Aborts the specified transaction, if it is running. If the transaction has already been committed or rolled back, then the state of the transaction is not altered.

- **transaction_id:** Identifier for the transaction to abort. To obtain transaction IDs, you can use the SHOW TRANSACTIONS or SHOW LOCKS commands.

SYSTEMCANCEL_ALL_QUERIES(session_id)

Cancels all active/running queries in the specified session.

- **session_id:** Identifier for the session for which to cancel all queries.

SYSTEMCANCEL_QUERY(query_id)

Cancels the specified query (or statement) if it is currently active/running.

- **query_id:** Identifier for the query to cancel.

SYSTEMPIPE_FORCE_RESUME(pipe_name)

Forces a pipe paused using ALTER PIPE to resume. This is necessary if the pipe owner transfers ownership of the pipe to another role while the pipe is paused.

- **pipe_name:** Pipe to resume running.

SYSTEMWAIT(amount [, time_unit])

Waits for a specified amount of time before proceeding.

- **amount:** Number specifying the amount of time to wait as determined by time_unit.
- **time_unit:** Time unit for amount. Accepted values are DAYS, HOURS, MINUTES, SECONDS, MILLISECONDS, MICROSECONDS, NANOSECONDS. The unit should be in single quotes

SYSTEMLAST_CHANGE_COMMIT_TIME(object_name)

Returns the commit time of the last DML change performed on a table or a view. In case of a view, the function returns the latest commit time of all the objects referenced in the view.

- **object_name:** ecifies the table or view for which the commit time for the last DML is returned

SYSTEMPIPE_STATUS(pipe_name)

Retrieves a JSON representation of the current status of a pipe.

- **pipe_name:** Pipe for which you want to retrieve the current status.

SYSTEMTYPEOF(expr)

Returns a string representing the SQL data type associated with an expression.

- **expr:**

Predicate Functions

BITAND(expr1, expr2)

Bitwise AND of two numeric expressions (a and b).

- **expr1:** This expression must evaluate to a data type that can be cast to INTEGER.
- **expr2:** This expression must evaluate to a data type that can be cast to INTEGER.

BITNOT(expr)

Bitwise negation of a numeric expression.

- **expr:** This expression must evaluate to a data type that can be cast to INTEGER.

BITOR(expr1, expr2)

Bitwise OR of two numeric expressions (a and b).

- **expr1:** This expression must evaluate to a data type that can be cast to INTEGER.
- **expr2:** This expression must evaluate to a data type that can be cast to INTEGER.

BITSHIFTLEFT(expr1, n)

Shift the bits for a numeric expression n positions to the left.

- **expr1:**
- **n:**

BITSHIFTRIGHT(expr1, n)

Shift the bits for a numeric expression n positions to the right, with sign extension.

- **expr1:** This expression must evaluate to a data type that can be cast to INTEGER.
- **n:** The number of bits to shift by.

BITXOR(expr1, expr2)

Bitwise XOR of two numeric expressions (a and b).

- **expr1:**
- **expr2:**

BOOLAND(expr1, expr2)

Computes the Boolean AND of two numeric expressions. In accordance with Boolean semantics. Non-zero values (including negative numbers) are regarded as True. Zero values are regarded as False.

- **expr1:**
- **expr2:**

BOOLNOT(expr1)

Computes the Boolean NOT of a single numeric expression. In accordance with Boolean semantics: Non-zero values (including negative numbers) are regarded as True. Zero values are regarded as False.

- **expr1:**

BOOLOR(expr1, expr2)

Computes the Boolean OR of two numeric expressions. In accordance with Boolean semantics: Non-zero values (including negative numbers) are regarded as True. Zero values are regarded as False.

- **expr1:**
- **expr2:**

BOOLXOR(expr1, expr2)

Computes the Boolean XOR of two numeric expressions (i.e. one of the expressions, but not both expressions, is TRUE). In accordance with Boolean semantics: Non-zero values (including negative numbers) are regarded as True. Zero values are regarded as False.

- **expr1:**
- **expr2:**

COALESCE(expr1 [, ...])

Returns the first non-NULL expression among its arguments, or NULL if all its arguments are NULL.

- **expr1:** Returns the first non-NULL expression among its arguments, or NULL if all its arguments are NULL.

DECODE(expr, search1, result1 [, search2, result2, ...] [, default])

Compares the select expression to each search expression in order. As soon as a search expression matches the selection expression, the corresponding result expression is returned.

- **expr:** This is the "select expression". The "search expressions" are compared to this select expression, and if there is a match then DECODE returns the result that corresponds to that search expression. The select expression is typically a column, but can be a subquery, literal, or other expression.
- **search1:** The search expressions indicate the values to compare to the select expression. If one of these search expressions matches, the function returns the corresponding result. If more than one search expression would match, only the first match's result is returned.
- **result1:** The results are the values that will be returned if one of the search expressions matches the select expression.
- **search2:** The search expressions indicate the values to compare to the select expression. If one of these search expressions matches, the function returns the corresponding result. If more than one search expression would match, only the first match's result is returned.
- **result2:** The results are the values that will be returned if one of the search expressions matches the select expression.
- **default:** If an optional default is specified, and if none of the search expressions match the select expression, then DECODE returns this default value.

EQUAL_NULL(expr1, expr2)

Compares whether two expressions are equal. The function is NULL-safe, meaning it treats NULLs as known values for comparing equality. Note that this is different from the EQUAL comparison operator (=), which treats NULLs as unknown values.

- **expr1:**
- **expr2:**

GREATEST(expr [, ...])

Returns the largest value from a list of expressions. GREATEST supports all types, including VARIANT. The first argument determines the return type. If the first type is numeric, then the return type will be 'widened' according to the numeric types in the list of all arguments. If the first type is not numeric, then all other arguments must be convertible to the first type. If any of the argument values is NULL, the result will be NULL.

- **expr:**

IFF(condition, expr1, expr2)

Single-level if-then-else expression. Similar to CASE, but only allows a single condition. If condition evaluates to TRUE, returns expr1, otherwise returns expr2.

- **condition:** The condition is an expression that should evaluate to a BOOLEAN value (True, False, or NULL).
- **expr1:** A general expression. This value is returned if the condition is true.
- **expr2:** A general expression. This value is returned if the condition is false.

IFNULL(expr1, expr2)

If expr1 is NULL, returns expr2, otherwise returns expr1.

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

LEAST(expr [, ...])

Returns the smallest value from a list of expressions. LEAST supports all data types, including VARIANT.

- **expr**: The arguments must include at least one expression. All the expressions should be of the same type or compatible types.

NULLIF(1, 2)

Returns NULL if expr1 is equal to expr2, otherwise returns expr1.

- **expr1**: any expression
- **expr2**: any expression

NVL(expr1, expr2)

If expr1 is NULL, returns expr2, otherwise returns expr1.

- **expr1**: The expression to be checked to see whether it's NULL.
- **expr2**: If expr1 is NULL, this expression will be evaluated and its value will be returned.

NVL2(expr1, expr2, expr3)

Returns values depending on the nullness of the first argument: If expr1 is not null, then NVL2 returns expr2. If expr1 is null, then NVL2 returns expr3.

- **expr1**: The expression to be checked to see whether it's NULL.
- **expr2**: If expr1 is not NULL, this expression will be evaluated and its value will be returned.
- **expr3**: If expr1 is NULL, this expression will be evaluated and its value will be returned.

REGR_VALX(expr1, expr2)

If the first argument is NULL, returns NULL. Otherwise, returns the second argument. Contrast REGR_VALX and REGR_VALY with NVL: NVL is a NULL-replacing function. The less commonly used REGR_VALX and REGR_VALY are NULL-preserving functions.

- **expr1:**
- **expr2:**

REGR_VALY(expr1, expr2)

If the second argument is NULL, returns NULL; otherwise, returns the first argument. Contrast REGR_VALX and REGR_VALY with NVL: NVL is a NULL-replacing function. The less commonly used REGR_VALX and REGR_VALY are NULL-preserving functions.

- **expr1:**
- **expr2:**

ZEROIFNULL(expr)

Returns 0 if its argument is null; otherwise, returns its argument.

- **expr:**

CURRENT_CLIENT()

Returns the version of the client from which the function was called. If called from an application using the JDBC or ODBC driver to connect to Snowflake, returns the version of the driver.

CURRENT_DATE()

Returns the current date of the system.

CURRENT_TIME([fract_sec_precision])

Returns the current time for the system.

- **fract_sec_precision>:** This optional argument indicates the precision with which to report the time. For example, a value of 3 says to use 3 digits after the decimal point - i.e. to specify the time with a precision of milliseconds.

CURRENT_TIMESTAMP([fract_sec_precision])

Returns the current timestamp for the system.

- **fract_sec_precision:** This optional argument indicates the precision with which to report the time. For example, a value of 3 says to use 3 digits after the decimal point - i.e. to specify the time with a precision of milliseconds.

CURRENT_VERSION()

Returns the current Snowflake version.

LOCALTIME()

Returns the current time for the system. ANSI-compliant alias for CURRENT_TIME.

LOCALTIMESTAMP()

Returns the current timestamp for the system. ANSI-compliant alias for CURRENT_TIMESTAMP.

CURRENT_ROLE()

Returns the name of the role in use for the current session. To specify a different role for the session, execute the USE ROLE command.

CURRENT_SESSION()

Returns a unique system identifier for the Snowflake session corresponding to the present connection. This will generally be a system-generated alphanumeric string. It is NOT derived from the user name or user account.

CURRENT_STATEMENT()

Returns the SQL text of the statement that is currently executing.

CURRENT_TRANSACTION()

Returns the transaction id of an open transaction in the current session.

CURRENT_USER()

Returns the name of the user currently logged into the system.

LAST_QUERY_ID([num])

Returns the ID of a specified query in the current session. If no query is specified, the most recently-executed query is returned.

- **num:** Specifies the query to return, based on the position of the query (within the session).

LAST_TRANSACTION()

Returns the transaction ID of the last transaction that was either committed or rolled back in the current session.

CURRENT_DATABASE()

Returns the name of the database in use for the current session. To specify a different database for the session, execute the USE DATABASE command.

CURRENT_SCHEMA()

Returns the name of the schema in use by the current session. To specify a different schema for the session, execute the USE SCHEMA command.

CURRENT_SCHEMAS()

Returns active search path schemas. For more information about search path, see Object Name Resolution.

CURRENT_WAREHOUSE()

Returns the name of the warehouse in use for the current session. To specify a different warehouse for the session, execute the USE WAREHOUSE command.

CAST(source_expr AS target_data_type)

Converts a value of one data type into another data type. The semantics of CAST are the same as the semantics of the corresponding TO_ datatype conversion functions. If the cast is not possible, an error is raised. For more details, see the individual TO_ datatype conversion functions.

- **source_expr:** Expression of any supported data type to be converted into a different data type.
- **target_data_type:** The data type to which to convert the expression. If the data type supports additional properties, such as scale and precision (for numbers/decimals), the properties can be included.

TRY_CAST(1 AS 2)

A special version of CAST , :: that is available for a subset of data type conversions. It performs the same operation (i.e converts a value of one data type into another data type), but returns a NULL value instead of raising an error when the conversion can not be performed.

- **source_expr:** Expression of any supported data type to be converted into a different data type.
- **target_data_type:** The data type to which to convert the expression. If the data type supports additional properties, such as scale and precision (for numbers/decimals), the properties can be included.

TO_CHAR(expr [, format])

Converts the input expression to a string. For NULL input, the output is NULL.

- **expr:** An expression of any data type.
- **format:** The format of the output string

TO_VARCHAR(expr [, format])

Converts the input expression to a string. For NULL input, the output is NULL.

- **expr:** An expression of any data type.
- **format:** The format of the output string

TO_BINARY(string_expr [, format])

Converts the input expression to a binary value. For NULL input, the output is NULL.

- **string_expr:** A string expression.
- **format:** The binary format for conversion: HEX, BASE64, or UTF-8 (see Binary Input and Output). The default is the value of the BINARY_INPUT_FORMAT session parameter. If this parameter is not set, the default is HEX

TRY_TO_BINARY(string_expr [, format])

A special version of TO_BINARY that performs the same operation (i.e. converts an input expression to a binary value), but with error handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **string_expr:** A string expression.
- **format:** The binary format for conversion: HEX, BASE64, or UTF-8 (see Binary Input and Output). The default is the value of the BINARY_INPUT_FORMAT session parameter. If this parameter is not set, the default is HEX

TO_DECIMAL(expr [, format [, precision [, scale]]])

Converts an input expression to a fixed-point number. For NULL input, the output is NULL.

- **expr:** An expression of a numeric, character, or variant type.
- **format:** The SQL format model used to parse the input expr and return. For more information, see SQL Format Models.
- **precision:** The maximal number of decimal digits in the resulting number; from 1 to 38. In Snowflake, precision is not used for determination of the number of bytes needed to store the number and does not have any effect on efficiency, so the default is the maximum (38).
- **scale:** The number of fractional decimal digits (from 0 to precision - 1). 0 indicates no fractional digits (i.e. an integer number).

TRY_TO_DECIMAL(expr [, format [, precision [, scale]])

A special version of TO_DECIMAL, TO_NUMBER, TO_NUMERIC that performs the same operation (i.e. converts an input expression to a fixed-point number), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **expr:** An expression of a numeric, character, or variant type.
- **format:** The SQL format model used to parse the input expr and return. For more information, see SQL Format Models.
- **precision:** The maximal number of decimal digits in the resulting number; from 1 to 38. In Snowflake, precision is not used for determination of the number of bytes needed to store the number and does not have any effect on efficiency, so the default is the maximum (38).
- **scale:** The number of fractional decimal digits (from 0 to precision - 1). 0 indicates no fractional digits (i.e. an integer number).

TO_DOUBLE(expr [, format])

Converts an expression to a double-precision floating-point number.

- **expr:** An expression of a numeric, character, or variant type.
- **format:** If the expression evaluates to a string, then the function accepts an optional format model. Format models are described at SQL Format Models. The format model specifies the format of the input string, not the format of the output value.

TRY_TO_DOUBLE(expr [, format])

A special version of TO_DOUBLE that performs the same operation (i.e. converts an input expression to a double-precision floating-point number), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **expr:** An expression of a numeric, character, or variant type.
- **format:** If the expression evaluates to a string, then the function accepts an optional format model. Format models are described at SQL Format Models. The format model specifies the format of the input string, not the format of the output value.

TO_BOOLEAN(text_or_numeric_expr)

Converts the input text or numeric expression to a Boolean value. For NULL input, the output is NULL.

- **text_or_numeric_expr:** A text or numeric expression

TRY_TO_BOOLEAN(text_or_numeric_expr)

A special version of TO_BOOLEAN that performs the same operation (i.e. converts an input expression to a Boolean value), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **text_or_numeric_expr:** A text or numeric expression

TO_DATE(expr [, format])

Converts an input expression to a date

- **expr:** Expression to be converted into a date.
- **format:** Date format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TRY_TO_DATE(expr [, format])

A special version of TO_DATE that performs the same operation (i.e. converts an input expression to a Boolean value), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **expr:** Expression to be converted into a date.
- **format:** Date format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TO_TIME(expr [, format])

Converts an input expression into a time. If input is NULL, returns NULL.

- **expr:** Expression to be converted into a time
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TRY_TO_TIME(expr [, format])

A special version of TO_TIME that performs the same operation (i.e. converts an input expression to a Boolean value), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **expr:** Expression to be converted into a time
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TO_TIMESTAMP(expr [, format])

Converts an input expression into the corresponding timestamp

- **expr:** Expression to be converted into a timestamp
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TRY_TO_TIMESTAMP(expr [, format])

A special version of TO_TIMESTAMP that performs the same operation (i.e. converts an input expression to a Boolean value), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **expr:** Expression to be converted into a timestamp
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TO_TIMESTAMP_NTZ(expr [, format])

Converts an input expression into the corresponding timestamp

- **expr:** Expression to be converted into a timestamp
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TRY_TO_TIMESTAMP_NTZ(expr [, format])

A special version of TO_TIMESTAMP_NTZ that performs the same operation (i.e. converts an input expression to a Boolean value), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **expr:** Expression to be converted into a timestamp
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TO_TIMESTAMP_TZ(expr [, format])

Converts an input expression into the corresponding timestamp

- **expr:** Expression to be converted into a timestamp
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TRY_TO_TIMESTAMP_TZ(expr [, format])

A special version of TO_TIMESTAMP_TZ that performs the same operation (i.e. converts an input expression to a Boolean value), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **expr:** Expression to be converted into a timestamp
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TO_TIMESTAMP_LTZ(expr [, format])

Converts an input expression into the corresponding timestamp

- **expr:** Expression to be converted into a timestamp
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

TRY_TO_TIMESTAMP_LTZ(expr [, format])

A special version of TO_TIMESTAMP_LTZ that performs the same operation (i.e. converts an input expression to a Boolean value), but with error-handling support (i.e. if the conversion cannot be performed, it returns a NULL value instead of raising an error).

- **expr:** Expression to be converted into a timestamp
- **format:** Time format specifier for string_expr or AUTO, which specifies for Snowflake to interpret the format.

RANDOM(seed)

Each call returns a pseudo-random 64-bit integer.

- **seed:** Seed is an integer. Different seeds will cause RANDOM to produce different output values.

RANDSTR(length, gen)

Returns a random string of specified length. Individual characters are chosen uniformly at random from the following pool of characters: 0-9, a-z, A-Z.

- **length:** Length of the string to generate
- **gen:** The value for the generator expression, gen, is used as the seed for this uniform random distribution

UUID_STRING(uuid, name)

Generates either a version 4 (random) or version 5 (named) RFC 4122-compliant UUID as a formatted string.

- **uuid:** The string (known as the namespace)
- **name:** The name of the UUID

NORMAL(mean, stddev, gen)

Returns a normal-distributed floating point number, with specified mean and stddev (standard deviation).

- **mean:** This is the value that you would like the output values centered around
- **stddev:** This specifies the width of one standard deviation.
- **gen:** This specifies the generator expression for the function.

UNIFORM(min, max, gen)

Returns a uniformly random number, in the inclusive range [min, max].

- **min:** The Minimum Number
- **max:** The Maximum Number
- **gen:** The generator expression for the function

ZIPF(s, N, gen)

Returns a Zipf-distributed integer, for N elements and characteristic exponent s

- **s**: the characteristic exponent
- **N**: the number of elements
- **gen**: The generator expression for the function

SEQ1([sign])

Returns a sequence of monotonically increasing integers, with wrap-around. Wrap-around occurs after the largest representable integer of the integer width (1 byte).

- **sign**: The optional sign argument. If the optional sign argument is 1, the sequence continues at the smallest representable number based on the given integer width. The default sign argument is 0.

SEQ2([sign])

Returns a sequence of monotonically increasing integers, with wrap-around. Wrap-around occurs after the largest representable integer of the integer width (2 byte).

- **sign**: The optional sign argument. If the optional sign argument is 1, the sequence continues at the smallest representable number based on the given integer width. The default sign argument is 0.

SEQ4([sign])

Returns a sequence of monotonically increasing integers, with wrap-around. Wrap-around occurs after the largest representable integer of the integer width (4 byte).

- **sign**: The optional sign argument. If the optional sign argument is 1, the sequence continues at the smallest representable number based on the given integer width. The default sign argument is 0.

SEQ8([sign])

Returns a sequence of monotonically increasing integers, with wrap-around. Wrap-around occurs after the largest representable integer of the integer width (8 byte).

- **sign:** The optional sign argument. If the optional sign argument is 1, the sequence continues at the smallest representable number based on the given integer width. The default sign argument is 0.

DATE_FROM_PARTS(year, month, day)

Creates a date from individual numeric components that represent the year, month, and day of the month

- **year:** The integer expression to use as a year for building a date.
- **month:** The integer expression to use as a month for building a date, with January represented as 1, and December as 12.
- **day:** The integer expression to use as a day for building a date, usually in the 1-31 range.

TIME_FROM_PARTS(hour, minute, second, nanoseconds)

Creates a time from individual numeric components.

- **hour:** An integer expression to use as an hour for building a time, usually in the 0-23 range.
- **minute:** An integer expression to use as a minute for building a time, usually in the 0-59 range.
- **second:** An integer expression to use as a second for building a time, usually in the 0-59 range.
- **nanoseconds:** A 9-digit integer expression to use as a nanosecond for building a time

TIMESTAMP_FROM_PARTS(date_expr, time_expr)

Creates a timestamp from individual numeric components. If no time zone is in effect, the function can be used to create a timestamp from a date expression and a time expression.

- **date_expr:** provides the year, month, and day for the timestamp
- **time_expr:** provides the hour, minute, second, and nanoseconds within the day

TIMESTAMP_FROM_PARTS(year, month, day, hour, minute, second [, nanoseconds [, time_zone]])

Creates a timestamp from individual numeric components. If no time zone is in effect, the function can be used to create a timestamp from a date expression and a time expression.

- **year:** An integer expression to use as a year for building a timestamp.
- **month:** An integer expression to use as a month for building a timestamp, with January represented as 1, and December as 12.
- **day:** An integer expression to use as a day for building a timestamp, usually in the 1-31 range.
- **hour:** An integer expression to use as an hour for building a timestamp, usually in the 0-23 range.
- **minute:** An integer expression to use as a minute for building a timestamp, usually in the 0-59 range.
- **second:** An integer expression to use as a second for building a timestamp, usually in the 0-59 range.
- **nanoseconds:** An integer expression to use as a nanosecond for building a timestamp, usually in the 0-999999999 range.
- **time_zone:** A string expression to use as a time zone for building a timestamp (e.g. America/Los_Angeles)

TIMESTAMP_NTZ_FROM_PARTS(date_expr, time_expr)

Creates a timestamp from individual numeric components. If no time zone is in effect, the function can be used to create a timestamp from a date expression and a time expression.

- **date_expr:** provides the year, month, and day for the timestamp
- **time_expr:** provides the hour, minute, second, and nanoseconds within the day

TIMESTAMP_NTZ_FROM_PARTS(year, month, day, hour, minute, second [, nanoseconds])

Creates a timestamp from individual numeric components. If no time zone is in effect, the function can be used to create a timestamp from a date expression and a time expression.

- **year:** An integer expression to use as a year for building a timestamp.
- **month:** An integer expression to use as a month for building a timestamp, with January represented as 1, and December as 12.
- **day:** An integer expression to use as a day for building a timestamp, usually in the 1-31 range.
- **hour:** An integer expression to use as an hour for building a timestamp, usually in the 0-23 range.
- **minute:** An integer expression to use as a minute for building a timestamp, usually in the 0-59 range.
- **second:** An integer expression to use as a second for building a timestamp, usually in the 0-59 range.
- **nanoseconds:** An integer expression to use as a nanosecond for building a timestamp, usually in the 0-999999999 range.

TIMESTAMP_LTZ_FROM_PARTS(year, month, day, hour, minute, second [, nanoseconds])

Creates a timestamp from individual numeric components. If no time zone is in effect, the function can be used to create a timestamp from a date expression and a time expression.

- **year:** An integer expression to use as a year for building a timestamp.
- **month:** An integer expression to use as a month for building a timestamp, with January represented as 1, and December as 12.
- **day:** An integer expression to use as a day for building a timestamp, usually in the 1-31 range.
- **hour:** An integer expression to use as an hour for building a timestamp, usually in the 0-23 range.
- **minute:** An integer expression to use as a minute for building a timestamp, usually

in the 0-59 range.

- **second:** An integer expression to use as a second for building a timestamp, usually in the 0-59 range.
- **nanoseconds:** An integer expression to use as a nanosecond for building a timestamp, usually in the 0-999999999 range.

TIMESTAMP_TZ_FROM_PARTS(year, month, day, hour, minute, second [, nanoseconds [, time_zone]])

Creates a timestamp from individual numeric components. If no time zone is in effect, the function can be used to create a timestamp from a date expression and a time expression.

- **year:** An integer expression to use as a year for building a timestamp.
- **month:** An integer expression to use as a month for building a timestamp, with January represented as 1, and December as 12.
- **day:** An integer expression to use as a day for building a timestamp, usually in the 1-31 range.
- **hour:** An integer expression to use as an hour for building a timestamp, usually in the 0-23 range.
- **minute:** An integer expression to use as a minute for building a timestamp, usually in the 0-59 range.
- **second:** An integer expression to use as a second for building a timestamp, usually in the 0-59 range.
- **nanoseconds:** An integer expression to use as a nanosecond for building a timestamp, usually in the 0-999999999 range.
- **time_zone:** A string expression to use as a time zone for building a timestamp (e.g. America/Los_Angeles)

DATE_PART(date_or_time_part, date_or_time_expr)

Extracts the specified date or time part from a date, time, or timestamp.

- **date_or_time_part:** The date or time part to extract
- **date_or_time_expr:** The date or time expression to extract from

DAYNAME(date_or_timestamp_expr)

Extracts the three-letter day-of-week name from the specified date or timestamp.

- **date_or_timestamp_expr:** The date or time expression to extract from

HOUR(time_or_timestamp_expr)

Extracts the corresponding time part from a time or timestamp value.

- **time_or_timestamp_expr:**

MINUTE(time_or_timestamp_expr)

Extracts the corresponding time part from a time or timestamp value.

- **time_or_timestamp_expr:**

SECOND(time_or_timestamp_expr)

Extracts the corresponding time part from a time or timestamp value.

- **time_or_timestamp_expr:**

YEAR(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

YEAROFWEEK(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

YEAROFWEEKISO(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

DAY(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

DAYOFMONTH(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

DAYOFWEEK(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

DAYOFWEEKISO(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

DAYOFYEAR(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

WEEK(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

WEEKOFYEAR(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

WEEKISO(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

MONTH(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

QUARTER(date_or_timestamp_expr)

Extracts the corresponding date part from a date or timestamp. These functions are alternatives to using the DATE_PART (or EXTRACT) function with the equivalent date part

- **date_or_timestamp_expr:**

LAST_DAY(date_or_time_expr [, date_part])

Returns the last day of the specified date part for a date or timestamp. Commonly used to return the last day of the month for a date or timestamp.

- **date_or_time_expr:**
- **date_part:**

MONTHNAME(date_or_timestamp_expr)

Extracts the three-letter month name from the specified date or timestamp.

- **date_or_timestamp_expr:**

NEXT_DAY(date_or_time_expr, dow_string)

Returns the date of the first specified DOW (day of week) that occurs after the input date.

- **date_or_time_expr:** Specifies the input date; can be a date or timestamp.
- **dow_string:** Specifies the day of week used to calculate the date for the previous day. The value can be a string literal or an expression that returns a string. The string must start with the first two characters (case-insensitive) of the day name

PREVIOUS_DAY(date_or_time_expr, dow)

Returns the date of the first specified DOW (day of week) that occurs before the input date.

- **date_or_time_expr:** Specifies the input date; can be a date or timestamp
- **dow:** Specifies the day of week used to calculate the date for the previous day. The value can be a string literal or an expression that returns a string. The string must start with the first two characters (case-insensitive) of the day name

ADD_MONTHS(date_or_timestamp_expr, num_months_expr)

Adds or subtracts a specified number of months to a date or timestamp, preserving the end-of-month information.

- **date_or_timestamp_expr:** This is the date or timestamp expression to which you want to add a specified number of months.
- **num_months_expr:** This is the number of months you want to add. This should be an integer. It may be positive or negative. If the value is a non-integer numeric value (for example, FLOAT) the value will be rounded to the nearest integer.

EXTRACT(date_or_time_part FROM date_or_time_expr)

Extracts the specified date or time part from a date, time, or timestamp.

- **date_or_time_part:**
- **date_or_time_expr:**

DATEADD(date_or_time_part, value, date_or_time_expr)

Adds the specified value for the specified date or time part to a date, time, or timestamp.

- **date_or_time_part:** This indicates the units of time that you want to add. For example if you want to add 2 days, then this will be DAY.
- **value:** This is the number of units of time that you want to add. For example, if you want to add 2 days, this will be 2.
- **date_or_time_expr:** must evaluate to a date, time, or timestamp. This is the date, time, or timestamp to which you want to add. For example, if you want to add 2 days to August 1, 2018, then this will be '2018-08-01'::DATE

TIMEADD(date_or_time_part, value, date_or_time_expr)

Adds the specified value for the specified date or time part to a date, time, or timestamp.

- **date_or_time_part:** This indicates the units of time that you want to add. For example if you want to add 2 days, then this will be DAY
- **value:** This is the number of units of time that you want to add. For example, if you want to add 2 days, this will be 2.
- **date_or_time_expr:** must evaluate to a date, time, or timestamp. This is the date, time, or timestamp to which you want to add. For example, if you want to add 2 days to August 1, 2018, then this will be '2018-08-01'::DATE.

TIMESTAMPADD(date_or_time_part, time_value, date_or_time_expr)

Adds the specified value for the specified date or time part to a date, time, or timestamp.

- **date_or_time_part:**
- **time_value:**
- **date_or_time_expr:**

DATEDIFF(date_or_time_part, date_or_time_expr1, date_or_time_expr2)

Calculates the difference between two date, time, or timestamp expressions based on the date or time part requested. The function returns the result of subtracting the second argument from the third argument.

- **date_or_time_part:**
- **date_or_time_expr1:** must be a date, a time, a timestamp, or an expression that can be evaluated to a date, a time, or a timestamp. The first value is subtracted from the second value.
- **date_or_time_expr2:** must be a date, a time, a timestamp, or an expression that can be evaluated to a date, a time, or a timestamp. The first value is subtracted from the second value.

TIMEDIFF(date_or_time_part, date_or_time_expr1, time_expr2)

Calculates the difference between two date, time, or timestamp expressions based on the specified date or time part.

- **date_or_time_part:**
- **date_or_time_expr1:** must be a date, a time, a timestamp, or an expression that can be evaluated to one of those. The first value is subtracted from the second value
- **time_expr2:** must be a date, a time, a timestamp, or an expression that can be evaluated to one of those.

TIMESTAMPDIFF(date_or_time_part, date_or_time_expr1, date_or_time_expr2)

Calculates the difference between two date, time, or timestamp expressions based on the specified date or time part.

- **date_or_time_part:**
- **date_or_time_expr1:**
- **date_or_time_expr2:**

DATE_TRUNC(date_or_time_part, date_or_time_expr)

Truncates a date, time, or timestamp to the specified part. Note that truncation is not the same as extraction. For example: Truncating a timestamp down to the quarter returns the timestamp corresponding to midnight of the first day of the quarter for the input timestamp. Extracting the quarter date part from a timestamp returns the quarter number of the year in the timestamp.

- **date_or_time_part:**
- **date_or_time_expr:**

CONVERT_TIMEZONE(source_tz, target_tz, source_timestamp_ntz)

Converts a timestamp to another time zone

- **source_tz:** String specifying the time zone for the input timestamp. Required for timestamps with no time zone (i.e. TIMEZONE_NTZ)
- **target_tz:** String specifying the time zone to which the input timestamp should be converted.
- **source_timestamp_ntz:** For the 3-argument version, string specifying the timestamp to convert (must be NTZ).

CONVERT_TIMEZONE(target_tz, source_timestamp)

Converts a timestamp to another time zone

- **target_tz:** String specifying the time zone to which the input timestamp should be converted.
- **source_timestamp:** For the 2-argument version, string specifying the timestamp to convert (can be any timestamp variant, including NTZ).

HASH(expr [, ...])

Returns a signed 64-bit hash value. Note that HASH never returns NULL, even for NULL inputs.

- **expr:**

HASH_AGG([DISTINCT] expr [, ...])

Returns an aggregate signed 64-bit hash value over the (unordered) set of input rows. HASH_AGG never returns NULL, even if no input is provided. Empty input "hashes" to 0.

- **expr:**

GET_DDL(object_type, namespace_object_name)

Returns a DDL statement that can be used to recreate the specified object. For databases and schemas, GET_DDL is recursive, i.e. it returns the DDL statements for recreating all supported objects within the specified database/schema.

- **object_type:** Specifies the type of object for which the DDL is returned.
- **namespace_object_name:** Specifies the fully-qualified name of the object for which the DDL is returned.

ABS(num_expr)

Returns the absolute value of a numeric expression.

- **num_expr:**

CEIL(input_expr [, scale_expr])

Returns values from input_expr rounded to the nearest equal or larger integer, or to the nearest equal or larger value with the specified number of places after the decimal point

- **input_expr:** The input_expr is the value to be rounded up, and should evaluate to a numeric data type, such as FLOAT or NUMBER.
- **scale_expr:** The scale_expr indicates the number of places after the decimal to which to round upward. This should evaluate to an integer.

FLOOR(input_expr [, scale_expr])

Returns values from input_expr rounded to the nearest equal or smaller integer, or to the nearest equal or smaller value with the specified number of places after the decimal point

- **input_expr:** The input_expr is the value to be rounded up, and should evaluate to a numeric data type, such as FLOAT or NUMBER.
- **scale_expr:** The scale_expr indicates the number of places after the decimal to which to round upward. This should evaluate to an integer.

MOD(expr1, expr2)

Returns the remainder of input expr1 divided by input expr2.

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

ROUND(input_expr, scale_expr)

Returns rounded values for input_expr

- **input_expr:** The expression to round
- **scale_expr:** If the optional scale_expr argument is specified, rounding is performed to the specified number of digits (negative digits round to factors-of-10)

SIGN(expr)

Returns the sign of its argument

- **expr**: an expression

TRUNCATE(input_expr, scale_expr)

Rounds the input expression down to the nearest (or equal) integer towards zero.

- **input_expr**: the expression to truncate
- **scale_expr**: If the optional scale_expr argument is provided, rounding is performed to the specified number of digits (negative digits round to factors-of-10)

CBRT(expr)

Returns the cubic root of a numeric expression.

- **expr**: a numeric expression

EXP(real_expr)

Computes Euler's number e raised to a floating-point value.

- **real_expr**: a real expression

FACTORIAL(integer_expr)

Computes the factorial of its input. The input argument must be an integer expression in the range of 0 to 33.

- **integer_expr**: an integer expression

POWER(x, y)

Returns a number (x) raised to the specified power (y).

- **x**:

- **y:**

SQRT(expr)

Returns the square-root of a non-negative numeric expression.

- **expr:** a non-negative numeric expression.

SQUARE(expr)

Returns the square of a numeric expression, i.e. a numeric expression multiplied by itself.

- **expr:** a numeric expression

LN(expr)

Returns the natural logarithm of a numeric expression.

- **expr:** a numeric expression

LOG(base, expr)

Returns the logarithm of a numeric expression.

- **base:** The "base" to use (e.g. 10 for base 10 arithmetic). This can be of any numeric data type (INTEGER, fixed-point, or floating point).
- **expr:** The value for which you want to know the log.

ACOS(real_expr)

Computes the inverse cosine (arc cosine) of its input; the result is a number in the interval $[-\pi, \pi]$.

- **real_expr:** This expression should evaluate to a real number greater than or equal to -1.0 and less than or equal to +1.0.

ACOSH(real_expr)

Computes the inverse (arc) hyperbolic cosine of its input.

- **real_expr:** This expression should evaluate to a FLOAT number greater than or equal to 1.0.

ASIN(real_expr)

Computes the inverse sine (arc sine) of its argument; the result is a number in the interval $[-\pi, \pi]$.

- **real_expr:** This expression should evaluate to a real number greater than or equal to -1.0 and less than or equal to +1.0.

ASINH(real_expr)

Computes the inverse (arc) hyperbolic sine of its argument.

- **real_expr:** This expression should evaluate to a real number.

ATAN(real_expr)

Computes the inverse tangent (arc tangent) of its argument; the result is a number in the interval $[-\pi, \pi]$.

- **real_expr:** This expression should evaluate to a real number.

ATANH(real_expr)

Computes the inverse (arc) hyperbolic tangent of its argument.

- **real_expr:**

COS(real_expr)

Computes the cosine of its argument; the argument should be expressed in radians.

- **real_expr:** This expression should evaluate to a real number. The value should be in radians, not degrees.

COSH(real_expr)

Computes the hyperbolic cosine of its argument.

- **real_expr:** This expression should evaluate to a real number.

COT(real_expr)

Computes the cotangent of its argument; the argument should be expressed in radians.

- **real_expr:** This expression should evaluate to a real number.

DEGREES(real_expr)

Converts radians to degrees.

- **real_expr:** An expression representing the number of radians.

RADIANS(real_expr)

Converts degrees to radians.

- **real_expr:**

SIN(real_expr)

Computes the sine of its argument; the argument should be expressed in radians.

- **real_expr:** This expression should evaluate to a real number. The value should be in radians, not degrees.

SINH(real_expr)

Computes the hyperbolic sine of its argument.

- **real_expr:** This expression should evaluate to a real number.

TAN(real_expr)

Computes the tangent of its argument; the argument should be expressed in radians.

- **real_expr:** This expression should evaluate to a real number. The value should be in radians, not degrees.

TANH(real_expr)

Computes the hyperbolic tangent of its argument.

- **real_expr:** This expression should evaluate to a real number.

ATAN2(real_expr, real_expr)

Computes the inverse tangent (arc tangent) of the ratio of its two arguments (i.e. $ATAN2(x,y) = ATAN(x/y)$). The result is a number in the interval $[-\pi, \pi]$.

- **real_expr:**
- **real_expr:**

PI()

Returns the value of pi as a floating-point value.

HAVERSINE(lat1, lon1, lat2, lon2)

Calculates the great circle distance in kilometers between two points on the Earth's surface, using the Haversine formula. The two points are specified by their latitude and longitude in degrees.

- **lat1:** Latitude of Point 1
- **lon1:** Longitude of Point 1
- **lat2:** Latitude of Point 2

- **lon2**: Longitude of Point 2

CHECK_JSON(string_or_variant_expr)

Checks the validity of a JSON document. If the input string is a valid JSON document or a NULL, the output is NULL (i.e. no error). If the input cannot be translated to a valid JSON value, the output string contains the error message.

- **string_or_variant_expr**: A VARIANT or string value (or expression) to check. If the expression is of type VARIANT, it should contain a string.

CHECK_XML(string_or_variant_expr)

Checks the validity of an XML document. If the input string is NULL or a valid XML document, the output is NULL. In case of an XML parsing error, the output string contains the error message.

- **string_or_variant_expr**: A VARIANT or string value (or expression) to check. If the expression is of type VARIANT, it should contain a string.

PARSE_JSON(expr)

Interprets an input string as a JSON document, producing a VARIANT value.

- **expr**: An expression of string type (e.g. VARCHAR) that holds valid JSON information.

PARSE_XML(expr)

Interprets an input string as an XML document, producing an OBJECT value. If the input is NULL, the output is NULL.

- **expr**: An expression of string type (e.g. VARCHAR) that holds valid JSON information.

STRIP_NULL_VALUE(expr)

Converts a JSON "null" value to a SQL NULL value. All other variant values are passed unchanged.

- **expr:** an expression to strip data from

ARRAY_APPEND(array, new_element)

Returns an array containing all elements from the source array as well as the new element. The new element is located at end of the array.

- **array:** The source array.
- **new_element:** The element to be appended. The element may be of almost any data type. The data type does not need to match the data type(s) of the existing elements in the array.

ARRAY_PREPEND(array, new_element)

Returns an array containing the new element as well as all elements from the source array. The new element is positioned at the beginning of the array.

- **array:** The source array.
- **new_element:** The element to be prepended.

ARRAY_CAT(array1, array2)

Returns a concatenation of two arrays.

- **array1:** The source array.
- **array2:** The array to be appended to array1.

ARRAY_INSERT(array, pos, new_element)

Returns an array containing all elements from the source array as well as the new element.

- **array:** The source array.
- **pos:** A (zero-based) position in the source array. The new element is inserted at this position. The original element from this position (if any) and all subsequent elements (if any) are shifted by one position to the right in the resulting array (i.e. inserting at position 0 has the same effect as using ARRAY_PREPEND). A negative position is

interpreted as an index from the back of the array (e.g. -1 results in insertion before the last element in the array).]

- **new_element:** The element to be inserted. The new element is located at position pos. The relative order of the other elements from the source array is preserved.

ARRAY_COMPACT(array1)

Returns a compacted array with missing and null values removed, effectively converting sparse arrays into dense arrays.

- **array1:** The source array.

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

Returns an array constructed from zero, one, or more inputs.

- **expr1:**
- **expr2:**

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

Returns an array constructed from zero, one, or more inputs.

- **expr1:**
- **expr2:**

ARRAY_CONTAINS(variant, array)

Takes a VARIANT and an ARRAY value as inputs and returns True if the VARIANT is contained in the ARRAY.

- **variant:**
- **array:**

ARRAY_POSITION(variant_expr, array)

Returns the index of the first occurrence of an element in an array.

- **variant_expr:** This expression should evaluate to a VARIANT value. The function searches for the first occurrence of this value in the array.
- **array:** The array to be searched.

ARRAY_SIZE(array_or_variant)

Returns the size of the input array. A variation of ARRAY_SIZE takes a VARIANT value as input. If the VARIANT value contains an array, the size of the array is returned; otherwise, NULL is returned if the value is not an array.

- **array_or_variant:**

ARRAY_SLICE(array, from, to)

Returns an array constructed from a specified subset of elements of the input array.

- **array:** The source array of which a subset of the elements are used to construct the resulting array.
- **from:** A position in the source array. The position of the first element is 0. Elements from positions less than from are not included in the resulting array.
- **to:** A position in the source array. Elements from positions equal to or greater than to are not included in the resulting array.

ARRAY_TO_STRING(array, separator_string)

Returns an input array converted to a string by casting all values to strings (using TO_VARCHAR) and concatenating them (using the string from the second argument to separate the elements).

- **array:** The array of elements to convert to a string.
- **separator_string:** The string to put between each element, typically a space, comma, or other human-readable separator.

ARRAYS_OVERLAP(array1, array2)

Compares whether two arrays have at least one element in common. Returns TRUE if there is at least one element in common; otherwise returns FALSE. The function is NULL-safe,

meaning it treats NULLs as known values for comparing equality.

- **array1:** an array
- **array2:** an array

OBJECT_AGG(key, value)

Returns one OBJECT per group. For each (key, value) input pair, where key must be a VARCHAR and value must be a VARIANT, the resulting OBJECT contains a key:value field.

- **key:**
- **value:**

OBJECT_CONSTRUCT([key1, value1 [, keyN, valueN, ...]])

Returns an object constructed from the arguments.

- **key1:**
- **value1:**
- **keyN:**
- **valueN:**

OBJECT_DELETE(object, key1 [, key2, ...])

Returns an object containing the contents of the input (i.e.source) object with one or more keys removed.

- **object:** The source object.
- **key1:** Key to be omitted from the returned object.
- **key2:** Key to be omitted from the returned object.

OBJECT_INSERT(object, key, value [, updateFlag])

Returns an object consisting of the input object with a new key-value pair inserted (or an existing key updated with a new value).

- **object:** The source object into which the new key-value pair is inserted.
- **key:** The new key to be inserted into the object. Must be different from all existing keys in the object, unless `updateFlag` is set to `TRUE`.
- **value:** The value associated with the key.
- **updateFlag:** Boolean flag that, when set to `TRUE`, specifies the input value is used to update/overwrite an existing key in the object, rather than inserting a new key-value pair.

XMLGET(type, tag_name [, instance_num])

Extracts an XML element object (often referred to as simply a "tag") from a content of outer XML element object by the name of the tag and its instance number (counting from 0)

- **type:**
- **tag_name:**
- **instance_num:** can be omitted, in which case the default value 0 is used.

GET(expr1, expr2)

Extracts a value from an object or array; returns `NULL` if either of the arguments is `NULL`.

- **expr1:** An Object, Variant, or Array
- **expr2:** A string value or an integer, which can be a constant or an expression

AS_ARRAY(variant_expr)

Casts a `VARIANT` value to an array.

- **variant_expr:** An expression that evaluates to a value of type `VARIANT`.

AS_BINARY(variant_expr)

Casts a `VARIANT` value to a binary string.

- **variant_expr:** An expression that evaluates to a value of type `VARIANT`.

AS_CHAR(variant_expr)

Casts a VARIANT value to a string. Does not convert values of other types into string.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_VARCHAR(variant_expr)

Casts a VARIANT value to a string. Does not convert values of other types into string.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_DATE(variant_expr)

Casts a VARIANT value to a date. Does not convert from timestamps.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_DOUBLE(variant_expr)

Casts a VARIANT value to a floating-point value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_REAL(variant_expr)

Casts a VARIANT value to a floating-point value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_INTEGER(variant_expr)

Casts a VARIANT value to an integer. Does not match non-integer values.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_OBJECT(variant_expr)

Casts a VARIANT value to an object.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_TIME(variant_expr)

Casts a VARIANT value to a time value. Does not convert from timestamps.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_TIMESTAMP_LTZ(variant_expr)

Casts a VARIANT value to the respective TIMESTAMP value with local time zone.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_TIMESTAMP_NTZ(variant_expr)

Casts a VARIANT value to the respective TIMESTAMP value with no time zone.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_TIMESTAMP_TZ(variant_expr)

Casts a VARIANT value to the respective TIMESTAMP value with time zone.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

AS_DECIMAL(variant_expr [, precision [, scale]])

Casts a VARIANT value to a fixed-point decimal (does not match floating-point values), with optional precision and scale.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.
- **precision**: The number of significant digits of the decimal number to store.
- **scale**: The number of significant digits after the decimal point.

AS_NUMBER(variant_expr [, precision [, scale]])

Casts a VARIANT value to a fixed-point decimal (does not match floating-point values), with optional precision and scale.

- **variant_expr:** An expression that evaluates to a value of type VARIANT.
- **precision:** The number of significant digits of the decimal number to store.
- **scale:** The number of significant digits after the decimal point.

STRTOK_TO_ARRAY(string [, delimiter])

Tokenizes the given string using the given set of delimiters and returns the tokens as an array. If either parameter is a NULL, a NULL is returned. An empty array is returned in case tokenization produces no tokens.

- **string:** Text to be tokenized.
- **delimiter:** Set of delimiters. Optional. Default value is a single space character

STRTOK_TO_SPLIT_TO_TABLE(string [, delimiter])

Tokenizes a string with the given set of delimiters and flattens the results into rows.

- **string:** Text to be tokenized.
- **delimiter:** Set of delimiters. Optional. Default value is a single space character

TO_ARRAY(expr)

Converts the input expression into an array: If the input is an ARRAY, or VARIANT containing an array value, the result is unchanged. For NULL or a JSON null input, returns NULL. For any other value, the result is a single-element array containing this value.

- **expr:** An expression of any data type.

TO_JSON(expr)

Converts any VARIANT value to a string containing the JSON representation of the value. If the input is NULL, the result is also NULL.

- **expr:** An expression of type VARIANT that holds valid JSON information.

TO_OBJECT(expr)

Converts the input value to an object

- **expr:** An expression of any data type.

TO_VARIANT(expr)

Converts any value to VARIANT value or NULL (if input is NULL).

- **expr:** An expression of any data type.

TO_XML(expr1)

Converts any VARIANT value to a string containing the XML representation of the value. If the input is NULL, the result is also NULL.

- **expr1:** The Variant to convert

IS_ARRAY(variant_expr)

Returns TRUE if its VARIANT argument contains an ARRAY value.

- **variant_expr:** An expression that evaluates to a value of type VARIANT.

IS_BOOLEAN(variant_expr)

Returns TRUE if its VARIANT argument contains an Boolean value.

- **variant_expr:** An expression that evaluates to a value of type VARIANT.

IS_BINARY(variant_expr)

Returns TRUE if its VARIANT argument contains an binary value.

- **variant_expr:** An expression that evaluates to a value of type VARIANT.

IS_CHAR(variant_expr)

Returns TRUE if its VARIANT argument contains an string value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_VARCHAR(variant_expr)

Returns TRUE if its VARIANT argument contains an string value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_DATE(variant_expr)

Returns TRUE if its VARIANT argument contains an DATE value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_DATE_VALUE(variant_expr)

Returns TRUE if its VARIANT argument contains an DATE value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_DECIMAL(variant_expr)

Returns TRUE if its VARIANT argument contains an fixed-point decimal value or integer value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_DOUBLE(variant_expr)

Returns TRUE if its VARIANT argument contains an a floating-point value, fixed-point decimal, or integer value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_REAL(variant_expr)

Returns TRUE if its VARIANT argument contains an a floating-point value, fixed-point decimal, or integer value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_INTEGER(variant_expr)

Returns TRUE if its VARIANT argument contains an integer value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_NULL_VALUE(variant_expr)

Returns TRUE if its VARIANT argument contains an NULL value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_OBJECT(variant_expr)

Returns TRUE if its VARIANT argument contains an OBJECT value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_TIME(variant_expr)

Returns TRUE if its VARIANT argument contains an TIME value.

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_TIMESTAMP_LTZ(variant_expr)

Verifies whether a VARIANT value contains the respective TIMESTAMP value

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_TIMESTAMP_NTZ(variant_expr)

Verifies whether a VARIANT value contains the respective TIMESTAMP value

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

IS_TIMESTAMP_TZ(variant_expr)

Verifies whether a VARIANT value contains the respective TIMESTAMP value

- **variant_expr**: An expression that evaluates to a value of type VARIANT.

TYPEOF(column)

Reports the type of a value stored in a VARIANT column. The type is returned as a string.

- **column**: The column to detect the type of

REGEXP_COUNT(subject, pattern [, position, parameters])

Returns the number of times that a pattern occurs in a string.

- **subject**: Subject to match.
- **pattern**: Pattern to match.
- **position**: Number of characters from the beginning of the string where the function starts searching for matches. Default: 1 (the search for a match starts at the first character on the left)
- **parameters**: String of one or more characters that specifies the parameters used for searching for matches

REGEXP_INSTR(subject, pattern [, position, occurrence, option, parameters])

Returns the position of the specified occurrence of the regular expression pattern in the string subject. If no match is found, returns 0.

- **subject**: Subject to match.

- **pattern:** Pattern to match.
- **position:** Number of characters from the beginning of the string where the function starts searching for matches. Default: 1
- **occurrence:** Specifies which occurrence of the pattern to match. The function skips the first occurrence -1 matches. Default: 1
- **option:** Specifies whether to return the offset of the first character of the match (0) or the offset of the first character following the end of the match (1). Default: 0
- **parameters:** String of one or more characters that specifies the parameters used for searching for matches. Supported values: c, i, m, e, s

REGEXP_LIKE(subject, pattern [, parameters])

Returns true if the subject matches the pattern. Both expressions must be text expressions.

- **subject:** Subject to match.
- **pattern:** Pattern to match.
- **parameters:** String of one or more characters that specifies the parameters used for searching for matches. Supported values: c, i, m, e, s

REGEXP_REPLACE(subject, pattern [, replacement [, position [, occurrence [, parameters]]]])

Returns the subject with the specified pattern (or all occurrences of the pattern) either removed or replaced by a replacement string. If no matches are found, returns the original subject.

- **subject:** Subject to match.
- **pattern:** Pattern to match.
- **replacement:** String the replaces the substrings matched by the pattern. If an empty string is specified, the function removes all matched patterns and returns the resulting string. Default: '' (empty string).
- **position:** Number of characters from the beginning of the string where the function starts searching for matches. Default: 1 (the search for a match starts at the first character on the left)

- **occurrence:** Specifies which occurrence of the pattern to replace. If 0 is specified, all occurrences are replaced. Default: 0 (all occurrences)
- **parameters:** String of one or more characters that specifies the parameters used for searching for matches. Supported values: c, i, m, e, s

REGEXP_SUBSTR(subject, pattern [, position, occurrence, parameters])

Returns the position of the specified occurrence of the regular expression pattern in the string subject. If no match is found, returns 0.

- **subject:** Subject to match.
- **pattern:** Pattern to match.
- **position:** Number of characters from the beginning of the string where the function starts searching for matches. Default: 1
- **occurrence:** Specifies which occurrence of the pattern to match. The function skips the first occurrence -1 matches. Default: 1
- **parameters:** String of one or more characters that specifies the parameters used for searching for matches. Supported values: c, i, m, e, s

RLIKE(subject, pattern, parameters)

Returns true if the subject matches the specified pattern. Both inputs must be text expressions.

- **subject:** Subject to match
- **pattern:** Pattern to match.
- **parameters:** String of one or more characters that specifies the parameters used for searching for matches. Supported values: c, i, m, e, s

ASCII(input)

Returns the ASCII code for the first character of a string. If the string is empty, a value of 0 is returned.

- **input:** The string for which the ASCII code for the first character in the string is

returned.

BIT_LENGTH(string_or_binary)

Returns the length of a string or binary value in bits. Snowflake doesn't use fractional bytes so length is always calculated as $8 * \text{OCTET_LENGTH}$.

- **string_or_binary**: The string or binary value for which the length is returned.

CHARINDEX(expr1, expr2 [, start_pos])

Searches for the first occurrence of the first argument in the second argument and, if successful, returns the position (1-based) of the first argument in the second argument.

- **expr1**: A string or binary expression representing the value we are looking for.
- **expr2**: A string or binary expression representing the value in which we are searching.
- **start_pos**: A number indicating the position from where to start the search (with 1 representing the start of expr1). Default: 1

CHAR(input)

Converts a Unicode code point (including 7-bit ASCII) into the character that matches the input Unicode. If an invalid code point is specified, an error is returned.

- **input**: The Unicode code point for which the character is returned.

CHR(input)

Converts a Unicode code point (including 7-bit ASCII) into the character that matches the input Unicode. If an invalid code point is specified, an error is returned.

- **input**: The Unicode code point for which the character is returned.

CONCAT(expr1, expr2)

Concatenates two strings or two binary values. If one of the them is null, the result is also null.

- **expr1**: A string to concatenate
- **expr2**: A string to concatenate

CONTAINS(expr1, expr2)

Returns true if expr1 contains expr2. Both expressions must be text or binary expressions.

- **expr1**: A string or binary expression representing the value we are looking fo
- **expr2**: A string or binary expression representing the value in which we are searching.

EDITDISTANCE(expr1, expr2)

Computes the Levenshtein distance between two input strings. It is the number of single-character insertions, deletions or substitutions needed to convert one string to another.

- **expr1**: A string
- **expr2**: A string

ENDSWITH(expr1, expr2)

Returns TRUE if the first expression ends with second expression. Both expressions must be text or binary expressions.

- **expr1**: A string or binary expression representing the value we are looking fo
- **expr2**: A string or binary expression representing the value in which we are searching.

ILIKE(subject, pattern [, escape])

Allows matching of strings based on comparison with a pattern. Unlike the LIKE function, string matching is case-insensitive. LIKE, ILIKE, and RLIKE all perform similar operations; however, RLIKE uses POSIX EXE (Extended Regular Expression) syntax instead of the SQL pattern syntax used by LIKE and ILIKE.

- **subject**: Subject to match.

- **pattern:** Pattern to match.
- **escape:** Character(s) inserted in front of a wildcard character to indicate that the wildcard should be interpreted as a regular character and not as a wildcard.

INITCAP(expr [, delimiters])

Returns the input string (expr) with the first letter of each word in uppercase and the subsequent letters in lowercase.

- **expr:** The input string
- **delimiters:** an optional argument specifying a string of one or more characters that INITCAP uses as separators for words in the input expression:

INSERT(base_expr, pos, len, insert_expr)

Replaces a substring of the specified length, starting at the specified position, with a new string or binary value. This function should not be confused with the INSERT DML command.

- **base_expr:** The string or BINARY expression for which you want to insert/replace characters.
- **pos:** The offset at which to start inserting characters. This is 1-based, not 0-based.
- **len:** The number of characters (starting at pos) that you want to replace. Valid values range from 0 to the number of characters between pos and the end of the string.
- **insert_expr:** The string to insert into the base_expr. If this string is empty, and if len is greater than zero, then effectively the operation becomes a delete (some characters are deleted, and none are added).

LEFT(string_expr, length_expr)

Returns a leftmost substring of its input. LEFT(STR,N) is equivalent to SUBSTR(STR,1,N).

- **string_expr:** The string expression
- **length_expr:** The length to extract

LENGTH(expr)

Returns the length of a input string or binary value. For strings, the length is the number of characters, and UTF-8 characters are counted as a single character. For binary, the length is the number of bytes.

- **expr:** The expression to measure

LIKE(subject, pattern [, escape])

Allows case-sensitive matching of strings based on comparison with a pattern. For case-insensitive matching, use ILIKE instead.

- **subject:** Subject to match.
- **pattern:** Pattern to match.
- **escape:** Character(s) inserted in front of a wildcard character to indicate that the wildcard should be interpreted as a regular character and not as a wildcard.

LOWER(expr)

Returns the input string (expr) with all characters converted to lowercase.

- **expr:** The input string

LPAD(base, n [, pad])

Left-pads a string with characters from another string, or left-pads a binary value with bytes from another binary value.

- **base:** The base string to pad
- **n:** The number of characters to pad
- **pad:** The character to pad

LTRIM(expr [, characters])

Removes leading characters, including whitespace, from a string.

- **expr:** The string expression to be trimmed.
- **characters:** One or more characters to remove from the left side of expr. The default value is ' ' (a single blank space character),

OCTET_LENGTH(string_or_binary)

Returns the length of a string or binary value in bytes. This will be the same as LENGTH for ASCII strings and greater than LENGTH for strings using Unicode code points. For binary, this is always the same as LENGTH.

- **string_or_binary:** The string or binary value for which the length is returned.

PARSE_IP(expr, type [, permissive])

Returns a JSON object consisting of all the components from a valid INET (Internet Protocol) or CIDR (Classless Internet Domain Routing) IPv4 or IPv6 string.

- **expr:** A string expression.
- **type:** Identifies the type of IP address. Supports either INET or CIDR; case-insensitive.
- **permissive:** Flag that determines how parse errors are handled:

PARSE_URL(string [, permissive])

Returns a JSON object consisting of all the components (fragment, host, path, port, query, scheme) in a valid input URL/URI.

- **string:** String to parse.
- **permissive:** Flag that determines how parse errors are handled

POSITION(expr1, expr2 [, start_pos])

Searches for the first occurrence of the first argument in the second argument and, if successful, returns the position (1-based) of the first argument in the second argument.

- **expr1:** A string or binary expression representing the value we are looking for.

- **expr2:** A string or binary expression representing the value in which we are searching.
- **start_pos:** A number indicating the position from where to start the search (with 1 representing the start of expr2).

POSITION(expr1 IN expr2)

Searches for the first occurrence of the first argument in the second argument and, if successful, returns the position (1-based) of the first argument in the second argument.

- **expr1:** A string or binary expression representing the value we are looking for.
- **expr2:** A string or binary expression representing the value in which we are searching.

REPEAT(input, n)

Builds a string by repeating the input for the specified number of times.

- **input:** The input string from which the output string is built.
- **n:** The number of times the input string should be repeated. The minimum valid number is 0 (which results in an empty string).

REPLACE(subject, pattern [, replacement])

Removes all occurrences of a specified substring, and optionally replaces them with another string

- **subject:** The subject is the string in which to do the replacements. Typically, this is a column, but it can be a literal.
- **pattern:** This is the substring that you want to replace. Typically, this is a literal, but it can be a column or expression. Note that this is not a "regular expression"; if you want to use regular expressions to search for a pattern, use the REGEXP_REPLACE function.
- **replacement:** This is the value used as a replacement for the pattern. If this is omitted, or is an empty string, then the REPLACE function simply deletes all occurrences of the pattern.

REVERSE(subject)

Reverses the order of characters in a string, or of bytes in a binary value.

- **subject:** The string to reverse

RIGHT(string_expr, length_expr)

Returns a rightmost substring of its input.

- **string_expr:** The string expression
- **length_expr:** The length to extract

RPAD(base, n [, pad])

Right-pads a string or binary value with characters from another string.

- **base:** The base string to pad
- **n:** The number of times to pad
- **pad:** The string to pad with

RTRIM(expr [, characters])

Removes trailing characters, including whitespace, from a string.

- **expr:** The string expression to be trimmed.
- **characters:** One or more characters to remove from the right side of expr:

RTRIMMED_LENGTH(string_expr)

Returns the length of its argument, minus trailing whitespace, but including leading whitespace.

- **string_expr:** The string expression to measure

SPACE(n)

Builds a string consisting of the specified number of blank spaces.

- **n**: The number of blank spaces used to build the string.

SPLIT(string, separator)

Splits a given string with a given separator and returns the result in an array of strings. Contiguous split strings in the source string, or the presence of a split string at the beginning or end of the source string, results in an empty string in the output. An empty separator string results in an array containing only the source string. If either parameter is a NULL, a NULL is returned.

- **string**: Text to be split into parts.
- **separator**: Text to split string by.

SPLIT_TO_TABLE(string, delimiter)

Splits a string with the given delimiter and flattens the results into rows.

- **string**: Text to be split into parts.
- **delimiter**: Text to split string by.

STRTOK(string [, delimiter] [, partNr])

Tokenizes a given string and returns the requested part.

- **string**: Text to be tokenized.
- **delimiter**: Text representing the set of delimiters to tokenize on. Each character in the delimiter string is a delimiter. If the delimiter is empty, and the string is empty, then the function returns NULL. If the delimiter is empty, and the string is non empty, then the whole string will be treated as one token. The default value of the delimiter is a single space character.
- **partNr**: Requested token (1-based, i.e. the first token is token number 1, not token number 0). If the token number is out of range, then NULL is returned. The default value is 1.

SPLIT_PART(string, delimiter, partNr)

Splits a given string and returns the requested part. If a part does not exist, an empty string is returned. If any parameter is NULL, NULL is returned.

- **string:** Text to be split into parts.
- **delimiter:** Text representing the delimiter to split by.
- **partNr:** Requested part of the split (1-based). 0 is treated as 1. If the value is negative, the parts are counted from the right side of the string.

STARTSWITH(expr1, expr2)

Returns true if expr1 starts with expr2. Both expressions must be text or binary expressions.

- **expr1:** A string or binary expression representing the value we are looking for
- **expr2:** A string or binary expression representing the value in which we are searching.

SUBSTRING(base_expr, start_expr [, length_expr])

Returns the portion of the string or binary value from base_expr, starting from the character/byte specified by start_expr, with optionally limited length.

- **base_expr:** The base string
- **start_expr:** The position to start from
- **length_expr:** Up to length_expr characters/bytes are returned, otherwise all the characters until the end of the string or binary value are returned.

COMPRESS(input, method)

Returns the portion of the string or binary value from base_expr, starting from the character/byte specified by start_expr, with optionally limited length.

- **input:** A BINARY or string value (or expression) to be compressed.
- **method:** A string with compression method and optional compression level. Supported methods are: SNAPPY. ZLIB. ZSTD. BZ2. The compression level is specified

in parentheses, for example: `zlib(1)`. The compression level is a non-negative integer. 0 means default level (same as omitting the compression level). The compression level is ignored if the method doesn't support compression levels.

DECOMPRESS_BINARY(input, method)

Returns the portion of the string or binary value from `base_expr`, starting from the character/byte specified by `start_expr`, with optionally limited length.

- **input:** A BINARY value (or expression) with data that was compressed using one of the compression methods specified in COMPRESS. If you attempt to decompress a compressed string, rather than a compressed BINARY value, you will not get an error; the function will return a BINARY value. See the Usage Notes below for details.
- **method:** The compression method originally used to compress the input.

DECOMPRESS_STRING(input, method)

Decompresses the compressed BINARY input parameter to a string.

- **input:** A BINARY value (or expression) with data that was compressed using one of the compression methods specified in COMPRESS.
- **method:** The compression method originally used to compress the input.

SUBSTR(base_expr, start_expr [, length_expr])

Returns the portion of the string or binary value from `base_expr`, starting from the character/byte specified by `start_expr`, with optionally limited length.

- **base_expr:** The base string
- **start_expr:** The position to start from
- **length_expr:** Up to `length_expr` characters/bytes are returned, otherwise all the characters until the end of the string or binary value are returned.

TRANSLATE(subject, sourceAlphabet, targetAlphabet)

Translates `subject` from the characters in `sourceAlphabet` to the characters in `targetAlphabet`.

- **subject:** A string expression that is translated. If a character in subject is not contained in sourceAlphabet, the character is added to the result without any translation.
- **sourceAlphabet:** A string with all characters that are modified by this function. Each character is either translated to the corresponding character in the targetAlphabet or omitted in the result if the targetAlphabet has no corresponding character (i.e. has less characters than the sourceAlphabet).
- **targetAlphabet:** A string with all characters that are used to replace characters from the sourceAlphabet.

TRIM(expr [, characters])

Removes leading and trailing characters from a string.

- **expr:** A string expression to be trimmed.
- **characters:** One or more characters to remove from the left and right side of expr: The default value is ' ' (a single blank space character), i.e. if no characters are specified, all leading and trailing blank spaces are removed.

UNICODE(input)

Returns the Unicode code point for the first Unicode character in a string. If the string is empty, a value of 0 is returned.

- **input:** The string for which the Unicode code point for the first character in the string is returned.

UPPER(expr)

Returns the input string expr with all characters converted to uppercase.

- **expr:** The input string

BASE64_DECODE_BINARY(input [, alphabet])

Decodes a Base64-encoded string to a binary.

- **input:** A Base64-encoded string expression.
- **alphabet:** A string consisting of up to three ASCII characters

BASE64_DECODE_STRING(input [, alphabet])

Decodes a Base64-encoded string to a binary.

- **input:** A Base64-encoded string expression.
- **alphabet:** A string consisting of up to three ASCII characters

BASE64_ENCODE(input [, max_line_length] [, alphabet])

Encodes the input (string or binary) using Base64 encoding.

- **input:** A string or binary expression to be encoded.
- **max_line_length:** A positive integer that specifies the maximum number of characters in a single line of the output.
- **alphabet:** A string consisting of up to three ASCII characters

HEX_DECODE_BINARY(input)

Decodes a hex-encoded string to a binary.

- **input:** A string expression containing only hexadecimal digits. Typically, this input string is generated by calling the function HEX_ENCODE.

HEX_DECODE_STRING(input)

Decodes a hex-encoded string to a string.

- **input:** A hex-encoded string expression. Typically the input was created by a call to HEX_ENCODE.

HEX_ENCODE(input [, case])

Encodes the input using hexadecimal (also 'hex' or 'base16') encoding. The result is comprised of 16 different symbols: The numbers '0' to '9' as well as the letters 'A' to 'F'

- **input:** A binary or string expression to be encoded.
- **case:** This optional boolean argument controls the case of the letters ('A', 'B', 'C', 'D', 'E' and 'F') used in the encoding. The default value is 1 and indicates that uppercase letters are used. The value 0 indicates that lowercase letters are used. All other values are illegal and result in an error.

TRY_BASE64_DECODE_BINARY(input [, alphabet])

A special version of BASE64_DECODE_BINARY that returns a NULL value if an error occurs during decoding.

- **input:** A Base64-encoded string expression.
- **alphabet:** A string consisting of up to three ASCII characters

TRY_BASE64_DECODE_STRING(input [, alphabet])

A special version of BASE64_DECODE_BINARY that returns a NULL value if an error occurs during decoding

- **input:** A Base64-encoded string expression.
- **alphabet:** A string consisting of up to three ASCII characters

TRY_HEX_DECODE_BINARY(input)

A special version of HEX_DECODE_BINARY that returns a NULL value if an error occurs during decoding.

- **input:** A string expression containing only hexadecimal digits. Typically, this input string is generated by calling the function HEX_ENCODE.

TRY_HEX_DECODE_STRING(input)

A special version of HEX_DECODE_BINARY that returns a NULL value if an error occurs during decoding.

- **input:** A hex-encoded string expression. Typically the input was created by a call to

HEX_ENCODE.

MD5(msg)

Returns a 32-character hex-encoded string containing the 128-bit MD5 message digest.

- **msg**: A string expression, the message to be hashed.

MD5_HEX(msg)

Returns a 32-character hex-encoded string containing the 128-bit MD5 message digest.

- **msg**: A string expression, the message to be hashed

MD5_BINARY(msg)

Returns a 16-byte BINARY value containing the 128-bit MD5 message digest.

- **msg**: A string expression, the message to be hashed.

MD5_NUMBER(msg)

Returns the 128-bit MD5 message digest interpreted as a signed 128-bit big endian number. This representation is useful for maximally efficient storage and comparison of MD5 digests.

- **msg**: A string expression, the message to be hashed.

SHA1(msg)

Returns a 40-character hex-encoded string containing the 160-bit SHA-1 message digest.

- **msg**: A string expression, the message to be hashed.

SHA1_HEX(msg)

Returns a 40-character hex-encoded string containing the 160-bit SHA-1 message digest.

- **msg:** A string expression, the message to be hashed.

SHA1_BINARY(msg)

Returns a 20-byte binary containing the 160-bit SHA-1 message digest.

- **msg:** A string expression, the message to be hashed.

SHA2(msg [, digest_size])

Returns a hex-encoded string containing the N-bit SHA-2 message digest, where N is the specified output digest size.

- **msg:** A string expression, the message to be hashed
- **digest_size:** Size (in bits) of the output, corresponding to the specific SHA-2 function used to encrypt the string

SHA2_HEX(msg [, digest_size])

Returns a hex-encoded string containing the N-bit SHA-2 message digest, where N is the specified output digest size.

- **msg:** A string expression, the message to be hashed
- **digest_size:** Size (in bits) of the output, corresponding to the specific SHA-2 function used to encrypt the string

SHA2_BINARY(msg [, digest_size])

Returns a binary containing the N-bit SHA-2 message digest, where N is the specified output digest size.

- **msg:** A string expression, the message to be hashed
- **digest_size:** Size (in bits) of the output, corresponding to the specific SHA-2 function used to encrypt the string

ANY_VALUE([DISTINCT] expr)

Returns some value of the expression from the group. The result is non-deterministic.

- **expr:** A group of values to choose from

AVG([DISTINCT] expr)

Returns the average of non-NULL records. If all records inside a group are NULL, NULL is returned.

- **expr:** Expression is an expression that evaluates to a numeric data type (INTEGER, FLOAT, DECIMAL, etc.).

CORR(y, c)

Returns the correlation coefficient for non-null pairs in a group. It is computed for non-null pairs using the following formula: $\text{COVAR_POP}(y, x) / (\text{STDDEV_POP}(x) * \text{STDDEV_POP}(y))$

- **y:**
- **c:**

COUNT([DISTINCT] expr1 [, expr2])

Returns either the number of non-NULL records for the specified columns, or a total number of records.

- **expr1:** A column name.
- **expr2:** You may include additional column name(s) if you wish. For example, you could list the number of distinct combinations of last name and first name.

COVAR_POP(x, y)

Returns the population covariance for non-null pairs in a group. It is computed for non-null pairs using the following formula: $(\text{SUM}(x*y) - \text{SUM}(x) * \text{SUM}(y) / \text{COUNT}(*)) / \text{COUNT}(*)$
Where x is the independent variable and y is the dependent variable.

- **x:**

- **y:**

COVAR_SAMP(y, c)

Returns the sample covariance for non-null pairs in a group. It is computed for non-null pairs using the following formula: $(\text{SUM}(x*y) - \text{SUM}(x) * \text{SUM}(y) / \text{COUNT}(*)) / (\text{COUNT}(*) - 1)$. Where x is the independent variable and y is the dependent variable.

- **y:**
- **x:**

LISTAGG([DISTINCT] expr, delimiter)

Returns the concatenated input values, separated by the delimiter string.

- **expr:** The expression (typically a column name) that determines the values to be put into the list. The expression should evaluate to a string, or to a data type that can be cast to string.
- **delimiter:** A string, or an expression that evaluates to a string. In practice, this is usually a single-character string. The string should be surrounded by single quotes, as shown in the examples below.

MAX(expr)

Returns the minimum or maximum value for the records within expr. NULL values are ignored unless all the records are NULL, in which case a NULL value is returned.

- **expr:**

MIN(expr)

Returns the minimum or maximum value for the records within expr. NULL values are ignored unless all the records are NULL, in which case a NULL value is returned.

- **expr:**

MEDIAN(expr)

Determines the median of a set of values.

- **expr:** The expression must evaluate to a numeric data type (INTEGER, FLOAT, DECIMAL, or equivalent).

PERCENTILE_CONT(percentile)

Return a percentile value based on a continuous distribution of the input column (specified in `order_by_expr`). If no input row lies exactly at the desired percentile, the result is calculated using linear interpolation of the two nearest input values. NULL values are ignored in the calculation.

- **percentile:** The percentile of the value that you want to find. The percentile must be a constant between 0.0 and 1.0. For example, if you want to find the value at the 90th percentile, specify 0.9.

PERCENTILE_DISC(percentile)

Returns a percentile value based on a discrete distribution of the input column (specified in `order_by_expr`). The returned value is that whose row has the smallest CUME_DIST value that is greater than or equal to the given percentile. NULL values are ignored in the calculation.

- **percentile:** The percentile of the value that you want to find. The percentile must be a constant between 0.0 and 1.0. For example, if you want to find the value at the 90th percentile, specify 0.9.

STDDEV(expr)

Returns the sample standard deviation (square root of sample variance) of non-NULL values. If all records inside a group are NULL, returns NULL.

- **expr:** An expression that evaluates to a numeric value.

STDDEV_SAMP([DISTINCT] expr)

Returns the sample standard deviation (square root of sample variance) of non-NULL values. If all records inside a group are NULL, returns NULL.

- **expr:** An expression that evaluates to a numeric value (integer, floating point, or fixed point).

STDDEV_POP([DISTINCT] x)

Returns the population standard deviation (square root of variance) of non-NULL values. If all records inside a group are NULL, returns NULL.

- **x:**

SUM([DISTINCT] expr)

Returns the sum of non-NULL records for expr. If the DISTINCT keyword is used, the sum of unique non-null values is computed. If all records inside a group are NULL, a value of NULL is returned.

- **expr:** The expression to sum

VAR_POP([DISTINCT] x)

Returns the population variance of non-NULL records in a group. If all records inside a group are NULL, a NULL is returned.

- **x:**

VARIANCE_POP([DISTINCT] x)

Returns the population variance of non-NULL records in a group. If all records inside a group are NULL, a NULL is returned.

- **x:**

VAR_SAMP([DISTINCT] x)

Returns the sample variance of non-NULL records in a group. If all records inside a group are NULL, a NULL is returned.

- **x:**

VARIANCE([DISTINCT] x)

Returns the sample variance of non-NULL records in a group. If all records inside a group are NULL, a NULL is returned.

- **x:**

VARIANCE_SAMP([DISTINCT] x)

Returns the sample variance of non-NULL records in a group. If all records inside a group are NULL, a NULL is returned.

- **x:**

CUME_DIST()

Finds the cumulative distribution of a value with regard to other values within the same window partition.

DENSE_RANK()

Returns the rank of a value within a group of values, without gaps in the ranks. The rank value starts at 1 and continues up sequentially. If two values are the same, they will have the same rank.

FIRST_VALUE(expr)

Returns the first value within an ordered group of values.

- **expr:**

LAG(expr [, offset, default])

Accesses data in a previous row in the same result set without having to join the table to itself.

- **expr:** The string expression to be returned.
- **offset:** The number of rows backward from the current row from which to obtain a value; e.g., an offset of 2 returns the expr value with an interval of 2 rows.
- **default:** The expression to return when the offset goes out of the bounds of the window. Supports any expression whose type is compatible with expr

LAST_VALUE(expr)

Returns the last value within an ordered group of values.

- **expr:**

LEAD(expr, offset, default)

Accesses data in a subsequent row in the same result set without having to join the table to itself.

- **expr:** The string expression to be returned.
- **offset:** The number of rows forward from the current row from which to obtain a value; e.g., an offset of 2 returns the expr value with an interval of 2 rows.
- **default:** The expression to return when the offset goes out of the bounds of the window. Supports any expression whose type is compatible with expr.

NTH_VALUE(expr, n)

Returns the nth value (up to 1000) within an ordered group of values.

- **expr:**
- **n:** Input value n cannot be greater than 1000.

NTILE(constant_value)

Divides an ordered data set equally into the number of buckets specified by constant_value. Buckets are sequentially numbered 1 through constant_value.

- **constant_value:** The desired number of buckets; must be a positive integer value.

PERCENT_RANK()

Returns the relative rank of a value within a group of values.

RANK()

Returns the rank of a value within an ordered group of values.

ROW_NUMBER()

Returns a unique row number for each row within a window partition.

WIDTH_BUCKET(expr, min_value, max_value, num_buckets)

Constructs equi-width histograms, in which the histogram range is divided into intervals of identical size, and returns the bucket number into which the value of an expression falls, after it has been evaluated. The function returns an integer value or null (if any input is null).

- **expr:** The expression for which the histogram is created. This expression must evaluate to a numeric value or to a value that can be implicitly converted to a numeric value.
- **min_value:** The low and high end points of the acceptable range for the expression. The end points must also evaluate to numeric values and not be equal.
- **max_value:** The low and high end points of the acceptable range for the expression. The end points must also evaluate to numeric values and not be equal.
- **num_buckets:** The desired number of buckets; must be a positive integer value. A value from the expression is assigned to each bucket, and the function then returns the corresponding bucket number.

BITAND_AGG(expr)

Returns the bitwise AND value of all non-NULL numeric records in a group. If all records inside the group are NULL, or if the group is empty, the function returns NULL.

- **expr:**

BITOR_AGG(expr)

Returns the bitwise OR value of all non-NULL numeric records in a group. If all records inside the group are NULL, or if the group is empty, the function returns NULL.

- **expr:**

BITXOR_AGG([DISTINCT] expr)

Returns the bitwise XOR value of all non-NULL numeric records in a group. If all records inside the group are NULL, or if the group is empty, the function returns NULL.

- **expr:**

ARRAY_AGG([DISTINCT] expr)

Returns the input values, pivoted into an ARRAY. If the input is empty, an empty ARRAY is returned.

- **expr:** The expression (typically a column name) that determines the values to be put into the list

REGR_AVGX(y, x)

Returns the average of the independent variable for non-null pairs in a group, where x is the independent variable and y is the dependent variable.

- **y:**
- **x:**

REGR_COUNT(y, x)

Returns the number of non-null number pairs in a group.

- **y:**
- **x:**

REGR_INTERCEPT(y, x)

Returns the intercept of the univariate linear regression line for non-null pairs in a group. It is computed for non-null pairs using the following formula: $AVG(y) - REGR_SLOPE(y, x) * AVG(x)$
Where x is the independent variable and y is the dependent variable

- **y:**
- **x:**

REGR_R2(y, x)

Returns the coefficient of determination for non-null pairs in a group.

- **y:**
- **x:**

REGR_SLOPE(y, x)

Returns the slope of the linear regression line for non-null pairs in a group

- **y:**
- **x:**

REGR_SXX(y, x)

Returns $REGR_COUNT(y, x) * VAR_POP(x)$ for non-null pairs.

- **y:**
- **x:**

REGR_SXY(y, x)

Returns $\text{REGR_COUNT}(\text{expr1}, \text{expr2}) * \text{COVAR_POP}(\text{expr1}, \text{expr2})$ for non-null pairs.

- **y:**
- **x:**

REGR_SYY(y, x)

Returns $\text{REGR_COUNT}(y, x) * \text{VAR_POP}(y)$ for non-null pairs.

- **y:**
- **x:**

REGR_AVGY(y, x)

Returns the average of the dependent variable for non-null pairs in a group, where x is the independent variable and y is the dependent variable.

- **y:**
- **x:**

APPROX_COUNT_DISTINCT([DISTINCT] expr [, ...])

Uses HyperLogLog to return an approximation of the distinct cardinality of the input.

- **expr:**

HLL([DISTINCT] expr [, ...])

Uses HyperLogLog to return an approximation of the distinct cardinality of the input.

- **expr:**

HLL_COMBINE([DISTINCT] state)

Combines (merges) input states into a single output state. This allows scenarios where HLL_ACCUMULATE is run over horizontal partitions of the same table, producing an algorithm state for each table partition. These states can later be combined using HLL_COMBINE, producing the same output state as a single run of HLL_ACCUMULATE over the entire table.

- **state:** An expression that contains state information generated by a call to HLL_ACCUMULATE.

HLL_ESTIMATE(state)

Returns the cardinality estimate for the given HyperLogLog state. A HyperLogLog state produced by HLL_ACCUMULATE and HLL_COMBINE can be used to compute a cardinality estimate using the HLL_ESTIMATE function.

- **state:** An expression that contains state information generated by a call to HLL_ACCUMULATE or HLL_COMBINE.

HLL_ACCUMULATE([DISTINCT] expr)

Returns the HyperLogLog state at the end of aggregation.

- **expr:**

HLL_EXPORT(binary_expr)

Converts input in BINARY format to OBJECT format. The HyperLogLog states operated on by HLL_ACCUMULATE, HLL_COMBINE, and HLL_ESTIMATE are in a proprietary binary format that may change in future versions of Snowflake. For long-term storage of HyperLogLog states, and for integration with external tools, Snowflake supports converting states from the BINARY format to an OBJECT (which can be printed and exported as JSON), and vice versa.

- **binary_expr:**

HLL_IMPORT(obj)

Converts input in OBJECT format to BINARY format. The HyperLogLog states operated on by HLL_ACCUMULATE, HLL_COMBINE, and HLL_ESTIMATE are in a proprietary binary format that may change in future versions of Snowflake. For long-term storage of HyperLogLog states, and for integration with external tools, Snowflake supports using HLL_IMPORT to convert states from an OBJECT format to BINARY, and vice versa.

- **obj:**

APPROXIMATE_JACCARD_INDEX([DISTINCT] expr [, ...])

Returns an estimation of the similarity (Jaccard index) of inputs based on their MinHash states. For more information about Jaccard indexes and the related function MINHASH, see Estimating Similarity of Two or More Sets.

- **expr:** The expression(s) should be one or more MinHash states returned by calls to the MINHASH function. In other words, the expressions must be MinHash state information, not the column or expression for which you want the approximate similarity. (The example below helps make this clear.)

APPROXIMATE_SIMILARITY([DISTINCT] expr [, ...])

Returns an estimation of the similarity (Jaccard index) of inputs based on their MinHash states. For more information about Jaccard indexes and the related function MINHASH, see Estimating Similarity of Two or More Sets.

- **expr:** The expression(s) should be one or more MinHash states returned by calls to the MINHASH function. In other words, the expressions must be MinHash state information, not the column or expression for which you want the approximate similarity. (The example below helps make this clear.)

MINHASH(k, [DISTINCT] expr [, ...])

Returns a MinHash state containing an array of size k constructed by applying k number of different hash functions to the input rows and keeping the minimum of each hash function. This MinHash state can then be input to the APPROXIMATE_SIMILARITY function to estimate the similarity with one or more other MinHash states.

- **k**: specifies the number of hash functions to be created. The larger the value, the better the approximation; however, this value has a linear impact on the computation time for estimating similarity using APPROXIMATE_SIMILARITY. The suggested value is 100.
- **expr**:

MINHASH_COMBINE([DISTINCT] state)

Combines input MinHash states into a single MinHash output state. This Minhash state can then be input to the APPROXIMATE_SIMILARITY function to estimate the similarity with other MinHash states.

- **state**: Input MinHash state must have MinHash arrays of equal length.

APPROX_TOP_K(expr, k, counters)

Uses Space-Saving to return an approximation of the most frequent values in the input, along with their approximate frequencies. The output is a JSON array of arrays. In the inner arrays, the first entry is a value in the input, and the second entry corresponds to its estimated frequency in the input. The outer array contains k items, sorted by descending frequency.

- **expr**: The expression (e.g. column name) for which you want to find the most common values.
- **k**: The number of values whose counts you want approximated. For example, if you want to see the top 10 most common values, then set k to 10.
- **counters**: This is the maximum number of distinct values that can be tracked at a time during the estimation process. For example, if counters is set to 100000, then the algorithm tracks 100,000 distinct values, attempting to keep the 100,000 most frequent values.

APPROX_TOP_K_ACCUMULATE(expr, counters)

Returns the Space-Saving summary at the end of aggregation. (For more information about the Space-Saving summary, see Estimating Frequent Values.) The function APPROX_TOP_K discards its internal, intermediate state when the final cardinality estimate is returned. However, in certain advanced use cases, such as estimating incremental frequent values

during bulk loading, you might want to keep the intermediate state, in which case you would use `APPROX_TOP_K_ACCUMULATE` instead of `APPROX_TOP_K`.

- **expr:** The expression (e.g. column name) for which you want to find the most common values.
- **counters:** This is the maximum number of distinct values that can be tracked at a time during the estimation process. For example, if counters is set to 100000, then the algorithm tracks 100,000 distinct values, attempting to keep the 100,000 most frequent values

APPROX_TOP_K_COMBINE(state [, counters])

Combines (merges) input states into a single output state. This allows scenarios where `APPROX_TOP_K_ACCUMULATE` is run over horizontal partitions of the same table, producing an algorithm state for each table partition. These states can later be combined using `APPROX_TOP_K_COMBINE`, producing the same output state as a single run of `APPROX_TOP_K_ACCUMULATE` over the entire table.

- **state:** An expression that contains state information generated by a call to `APPROX_TOP_K_ACCUMULATE`.
- **counters:** This is the maximum number of distinct values that can be tracked at a time during the estimation process. For example, if counters is set to 100000, then the algorithm tracks 100,000 distinct values, attempting to keep the 100,000 most frequent values.

APPROX_TOP_K_ESTIMATE(state [, k])

Returns the approximate most frequent values and their estimated frequency for the given Space-Saving state. (For more information about the Space-Saving summary, see [Estimating Frequent Values](#).)

- **state:** An expression that contains state information generated by a call to `APPROX_TOP_K_ACCUMULATE` or `APPROX_TOP_K_COMBINE`.
- **k:** The number of values whose counts you want approximated. For example, if you want to see the top 10 most common values, then set k to 10.

APPROX_PERCENTILE(expr, percentile)

Returns an approximated value for the desired percentile (i.e. if column *c* has *n* numbers, then APPROX_PERCENTILE(*c*, *p*) returns a number such that approximately $n * p$ of the numbers in *c* are smaller than the returned number).

- **expr:** A valid expression, such as a column name, that evaluates to a numeric value.
- **percentile:** A constant real value greater than or equal to 0.0 and less than 1.0. This indicates the percentile (from 0 to 99.999...). E.g. The value 0.65 indicates the 65th percentile.

APPROX_PERCENTILE_ACCUMULATE(expr)

Returns the internal representation of the t-Digest state (as a JSON object) at the end of aggregation. (For more information about t-Digest, see: Estimating Percentile Values.)

- **expr:** A valid expression, such as a column name, that evaluates to a numeric value.

APPROX_PERCENTILE_COMBINE(state)

Combines (merges) percentile input states into a single output state.

- **state:** An expression that contains state information generated by a call to APPROX_PERCENTILE_ACCUMULATE.

APPROX_PERCENTILE_ESTIMATE(state, percentile)

Returns the desired approximated percentile value for the specified t-Digest state.

- **state:** An expression that contains state information generated by a call to APPROX_PERCENTILE_ACCUMULATE or APPROX_PERCENTILE_COMBINE.
- **percentile:** A constant real value greater than or equal to 0.0 and less than 1.0. This indicates the percentile (from 0 to 99.999...). E.g. The value 0.65 indicates the 65th percentile.

GROUPING(expr1 [, ...])

Describes which of a list of expressions are grouped in a row produced by a GROUP BY query.

- **expr1:**

GROUPING_ID(expr1 [, ...])

Describes which of a list of expressions are grouped in a row produced by a GROUP BY query.

- **expr1:**

SYSTEMABORT_SESSION(session_id)

Aborts the specified session.

- **session_id:** Identifier for the session to abort.

SYSTEMABORT_TRANSACTION(transaction_id)

Aborts the specified transaction, if it is running. If the transaction has already been committed or rolled back, then the state of the transaction is not altered.

- **transaction_id:** Identifier for the transaction to abort. To obtain transaction IDs, you can use the SHOW TRANSACTIONS or SHOW LOCKS commands.

SYSTEMCANCEL_ALL_QUERIES(session_id)

Cancels all active/running queries in the specified session.

- **session_id:** Identifier for the session for which to cancel all queries.

SYSTEMCANCEL_QUERY(query_id)

Cancels the specified query (or statement) if it is currently active/running.

- **query_id**: Identifier for the query to cancel.

SYSTEMPIPE_FORCE_RESUME(pipe_name)

Forces a pipe paused using ALTER PIPE to resume. This is necessary if the pipe owner transfers ownership of the pipe to another role while the pipe is paused.

- **pipe_name**: Pipe to resume running.

SYSTEMWAIT(amount [, time_unit])

Waits for a specified amount of time before proceeding.

- **amount**: Number specifying the amount of time to wait as determined by time_unit.
- **time_unit**: Time unit for amount. Accepted values are DAYS, HOURS, MINUTES, SECONDS, MILLISECONDS, MICROSECONDS, NANOSECONDS. The unit should be in single quotes

SYSTEMLAST_CHANGE_COMMIT_TIME(object_name)

Returns the commit time of the last DML change performed on a table or a view. In case of a view, the function returns the latest commit time of all the objects referenced in the view.

- **object_name**: ecifies the table or view for which the commit time for the last DML is returned

SYSTEMPIPE_STATUS(pipe_name)

Retrieves a JSON representation of the current status of a pipe.

- **pipe_name**: Pipe for which you want to retrieve the current status.

SYSTEMTYPEOF(expr)

Returns a string representing the SQL data type associated with an expression.

- **expr**:

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 Id, ProductName INTO 'csv://c:/[DemoDB].[PUBLIC].Products.txt' FROM '[DemoDB].[PUBLIC].Products' WHERE ProductName = 'Konbu'");
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 '[DemoDB].[PUBLIC].Products' IN 'csv://filename=c:/[DemoDB].[PUBLIC].Products.csv;delimiter=tab' FROM '[DemoDB].[PUBLIC].Products' WHERE ProductName = 'Konbu'");
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 [DemoDB].[PUBLIC].Products (ProductName)
VALUES (?)";
PreparedStatement pstmt = connection.prepareStatement
(cmd,Statement.RETURN_GENERATED_KEYS);
pstmt.setString(1, "Konbu");
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 [DemoDB].[PUBLIC].Products SET ProductName='Konbu'
WHERE Id = ?";
PreparedStatement pstmt = connection.prepareStatement(cmd);
pstmt.setString(1, "22");
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:snowflake:url=https://myaccount.region.snowflakecomputing.com;use
r=Admin;password=test123;Database=Northwind;Warehouse=TestWarehouse;Acco
unt=Tester1;");
String cmd = "DELETE FROM [DemoDB].[PUBLIC].Products WHERE Id = ?";
PreparedStatement pstmt = connection.prepareStatement(cmd);
pstmt.setString(1, "22");
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 adapter leverages the Snowflake API to enable bidirectional SQL access to Snowflake data.

Discovering Schemas

The Snowflake Adapter dynamically obtains the metadata as defined within Snowflake for the Warehouse, Database, and Schema specified. Database and Schema are both optional and will restrict the tables and views to only the values you specify in each property.

Stored Procedures

[Stored Procedures](#) are functions for OAuth Authentication.

Stored Procedures

Stored procedures are function-like interfaces that extend the functionality of the adapter beyond simple SELECT/INSERT/UPDATE/DELETE operations with Snowflake.

Stored procedures accept a list of parameters, perform their intended function, and then return, if applicable, any relevant response data from Snowflake, along with an indication of whether the procedure succeeded or failed.

Snowflake Adapter Stored Procedures

Name	Description
GetOAuthAccessToken	If using a Windows application, set Authmode to App. If using a Web app, set Authmode to Web and specify the Verifier obtained by GetOAuthAuthorizationUrl.
GetOAuthAuthorizationUrl	Gets the authorization URL that must be opened separately by the user to grant access to your application. Only needed when developing Web apps.
GetSSOAuthAuthorizationURL	Gets Browser-based SSO authorization URL. Access the URL returned in the output in a Web browser. This requests the access token that can be used as part of the connection string to Snowflake.
RefreshOAuthAccessToken	Refreshes the OAuth access token used for authentication with Snowflake.

GetOAuthAccessToken

If using a Windows application, set Authmode to App. If using a Web app, set Authmode to Web and specify the Verifier obtained by GetOAuthAuthorizationUrl.

Input

Name	Type	Required	Description
AuthMode	String	False	The type of authentication mode to use. The allowed values are APP, WEB.
CallbackUrl	String	False	The page to return the user after authorization is complete.
Verifier	String	False	The verifier code returned by Snowflake after permissions have been granted for the app to connect. WEB Authmode only.
PKCEVerifier	String	False	The PKCEVerifier returned by GetOAuthAuthorizationURL.
Prompt	String	False	Defaults to 'select_account' which prompts the user to select account while authenticating. Set to 'None', for no prompt, 'login' to force user to enter their credentials or 'consent' to trigger the OAuth consent dialog after the user signs in, asking the user to grant permissions to the app.

Result Set Columns

Name	Type	Description
OAuthRefreshToken	<i>String</i>	A token that may be used to obtain a new access token.
OAuthAccessToken	<i>String</i>	The OAuth access token.
ExpiresIn	<i>String</i>	The remaining lifetime on the access token. A -1 denotes that it will not expire.

GetOAuthAuthorizationUrl

Gets the authorization URL that must be opened separately by the user to grant access to your application. Only needed when developing Web apps.

Input

Name	Type	Required	Description
CallbackUrl	<i>String</i>	<i>False</i>	The page to return the user after authorization is complete.
Scope	<i>String</i>	<i>False</i>	The scope of access to Snowflake. The scope parameters in the initial authorization request

			optionally limit the operations and role permitted by the access token, the default scope was refresh_token.
State	<i>String</i>	<i>False</i>	Any value that you wish to be sent with the callback.
Prompt	<i>String</i>	<i>False</i>	Defaults to 'select_account' which prompts the user to select account while authenticating. Set to 'None', for no prompt, 'login' to force user to enter their credentials or 'consent' to trigger the OAuth consent dialog after the user signs in, asking the user to grant permissions to the app.

Result Set Columns

Name	Type	Description
Url	<i>String</i>	The authorization url.
PKCEVerifier	<i>String</i>	A random value used as input for GetOAuthAccessToken in the PKCE flow.

GetSSOAuthAuthorizationURL

Gets Browser-based SSO authorization URL. Access the URL returned in the output in a Web browser. This requests the access token that can be used as part of the connection string to Snowflake.

Input

Name	Type	Required	Description
Port	<i>String</i>	<i>False</i>	The listening port of the callback url. The default value is 80.

Result Set Columns

Name	Type	Description
ProofKey	<i>String</i>	
SSOURL	<i>String</i>	
TokenURL	<i>String</i>	

RefreshOAuthAccessToken

Refreshes the OAuth access token used for authentication with Snowflake.

Input

Name	Type	Required	Description
OAuthRefreshToken	String	True	Set this to the token value that expired.

Result Set Columns

Name	Type	Description
OAuthAccessToken	String	The authentication token returned from AzureDataCatalog. This can be used in subsequent calls to other operations for this particular service.
OAuthRefreshToken	String	This is the same as the access token.
ExpiresIn	String	The remaining lifetime on the access token.

Connection String Options

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

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

Authentication

Property	Description
AuthScheme	The authentication scheme used. Accepted entries are Password, OKTA, PrivateKey, AzureAD, OAuth, or PingFederate.
Account	The Account provided for authentication with Snowflake database. This is usually derived from the URL automatically.
Warehouse	The name of the Snowflake warehouse.
User	The username provided for authentication with the Snowflake database.
Password	The user's password.
URL	The URL of Snowflake database.
RoleName	The role of the Snowflake user: PUBLIC, SYSADMIN, or ACCOUNTADMIN.
CredentialsLocation	The location of the settings file where credentials are saved.

Connection

Property	Description
UseVirtualHosting	If true (default), buckets will be referenced in the request using the hosted-style request: http://yourbucket.s3.amazonaws.com/yourobject. If set to false, the bean will use the path-style request: http://s3.amazonaws.com/yourbucket/yourobject. Note that this property will be set to false, in case of an S3 based custom service when the CustomURL is specified.

Azure Authentication

Property	Description
AzureTenant	The Microsoft Online tenant being used to access data. If not specified, your default tenant will be used.

SSO

Property	Description
ProofKey	The ProofKey for authentication with Snowflake database. This is usually derived from GetSSOAuthorizationURL call.
ExternalToken	The External Token for authentication with the Snowflake database. This is usually derived from the external handler. For example, handle the callback URL from procedure GetSSOAuthorizationURL will get this token.
SSOProperties	Additional properties required to connect to the identity provider in a semicolon-separated list.

KeyPairAuth

Property	Description
PrivateKey	The private key provided for key pair authentication with Snowflake.
PrivateKeyPassword	The password for the private key specified in the PrivateKey property, if required.
PrivateKeyType	The type of key store containing the private key to use with key pair authentication.

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.
CallbackURL	The OAuth callback URL to return to when authenticating. This value must match the callback URL you specify in your Add-In settings.
State	An optional value that has meaning for your OAuth App.
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://.
OAuthAuthenticator	This determines the authenticator that the OAuth application requests from Snowflake.
Scope	This determines the scopes that the OAuth application requests from Snowflake.
OAuthAuthorizationURL	The authorization URL for the OAuth service.
OAuthAccessTokenURL	The URL to retrieve the OAuth access token from.
OAuthVerifier	The verifier code returned from the OAuth authorization URL.
PKCEVerifier	A random value used as input for calling GetOAuthAccessToken in the PKCE flow.
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.

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.
Database	The name of the Snowflake database.
Schema	The schema of the Snowflake database.

Miscellaneous

Property	Description
AllowPreparedStatement	Prepare a query statement before its execution.
AsyncQueryTimeout	The timeout for asynchronous requests issued by the provider to download large result sets.
CustomStage	The name of a custom stage to use during bulk write operations.
EnableArrow	Whether to support Apache Arrow.
ExternalStageAWSAccessKey	Your AWS account access key. Only used when defining a CustomStage for bulk write operations.
ExternalStageAWSSecretKey	Your AWS account secret key. Only used when defining a CustomStage for bulk write operations.
ExternalStageAzureSASToken	The string value of the Azure Blob shared access signature.
IgnoreCase	Whether to ignore case in identifiers. Default: false.
IncludeTableTypes	If set to true, the provider will report the types of individual tables and views.
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.
MaxThreads	Specifies the number of concurrent requests.
MergeDelete	A boolean indicating whether batch DELETE statements should be converted to MERGE statements automatically. Only used when the DELETE statement's where clause contains a table's primary key field only and they are combined with AND logical operator.

MergeInsert	A boolean indicating whether INSERT statements should be converted to MERGE statements automatically. Only used when the INSERT contains a table's primary key field.
MergeUpdate	A boolean indicating whether batch UPDATE statements should be converted to MERGE statements automatically. Only used when the UPDATE statement's where clause contains a table's primary key field only and they are combined with AND logical operator.
Other	These hidden properties are used only in specific use cases.
Pagesize	The maximum number of results to return per page from Snowflake.
Readonly	You can use this property to enforce read-only access to Snowflake from the provider.
ReplaceInvalidUTF8Chars	Specifies whether to replace invalid UTF8 characters with a '?'.
RetryOnS3Timeout	Whether or not to retry when network issues occur at during chunk downloading.
SessionParameters	The session parameters for Snowflake. For example: SessionParameters='QUERY_TAG=MyTag;QUOTED_IDENTIFIERS_IGNORE_CASE=True;';
Timeout	The value in seconds until the timeout error is thrown, canceling the operation.
UseAsyncQuery	This field sets whether async query is enabled.
UserDefinedViews	A filepath pointing to the JSON configuration file containing your custom views.

Authentication

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

Property	Description
AuthScheme	The authentication scheme used. Accepted entries are Password, OKTA, PrivateKey, AzureAD, OAuth, or PingFederate.
Account	The Account provided for authentication with Snowflake database. This is usually derived from the URL automatically.
Warehouse	The name of the Snowflake warehouse.
User	The username provided for authentication with the Snowflake database.
Password	The user's password.
URL	The URL of Snowflake database.
RoleName	The role of the Snowflake user: PUBLIC, SYSADMIN, or ACCOUNTADMIN.
CredentialsLocation	The location of the settings file where credentials are saved.

AuthScheme

The authentication scheme used. Accepted entries are Password, OKTA, PrivateKey, AzureAD, OAuth, or PingFederate.

Possible Values

Password, OKTA, PrivateKey, AzureAD, OAuth, PingFederate

Data Type

string

Default Value

"Password"

Remarks

The adapter supports the following authentication mechanisms. See the Getting Started chapter for authentication guides.

- Password: Set this to authenticate with a Snowflake user.
- OKTA: Set this to use the OKTA SSO identity provider. Set [SSOProperties](#) in addition to the [User](#) and [Password](#) you use to authenticate to OKTA.
- AzureAD: Set this along with [User](#) to use the Azure Active Directory identity provider. When connecting, your browser will open allowing you to login to Azure AD to complete the authentication.
- PingFederate: Set this along with [User](#) to use the PingFederate SSO identity provider. When connecting, your browser will open allowing you to login to PingFederate to complete the authentication.
- PrivateKey: Set this to use key pair authentication. Set [PrivateKey](#), [PrivateKeyPassword](#) and [PrivateKeyType](#) in addition to authenticate with key pair authentication.
- OAuth: Set this to use oauth authentication. Set [OAuthClientId](#), [OAuthClientSecret](#) to the Snowflake OAuth credentials. Additionally, set [InitiateOAuth](#) to GETANDREFRESH. Note that the CData driver always uses PKCE with OAuth for extra security.

Account

The Account provided for authentication with Snowflake database. This is usually derived from the URL automatically.

Data Type

string

Default Value

""

Remarks

The Account provided for authentication with Snowflake database. This is usually derived from the URL automatically and will not need to be set manually. A notable exception is

Snowflake VPS if your Account name doesn't follow the usual URL syntax
<https://myaccount.region.snowflakecomputing.com>. Snowflake provides the Account name in this case.

Warehouse

The name of the Snowflake warehouse.

Data Type

string

Default Value

""

Remarks

The name of the Snowflake warehouse.

User

The username provided for authentication with the Snowflake database.

Data Type

string

Default Value

""

Remarks

The username provided for authentication with the Snowflake database.

Password

The user's password.

Data Type

string

Default Value

""

Remarks

The password provided for authentication with Snowflake.

URL

The URL of Snowflake database.

Data Type

string

Default Value

""

Remarks

Set this property to the URL of the Snowflake database instance.

AWS format:

```
https://myaccount.region.snowflakecomputing.com
```

Azure format:

```
https://myaccount.region.azure.snowflakecomputing.com
```

GCP format:

```
https://myaccount.gcp.snowflakecomputing.com
```

RoleName

The role of the Snowflake user: PUBLIC, SYSADMIN, or ACCOUNTADMIN.

Data Type

string

Default Value

""

Remarks

The role of the Snowflake user using the specified database. The defaults in Snowflake are: PUBLIC, SYSADMIN, or ACCOUNTADMIN. A custom role may also be specified.

CredentialsLocation

The location of the settings file where credentials are saved.

Data Type

string

Default Value

"%APPDATA%\CDData\Snowflake Data Provider\CredentialsFile.txt"

Remarks

If left unspecified, the default location is "%APPDATA%\CDData\Snowflake Data Provider\CredentialsFile.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

Connection

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

Property	Description
UseVirtualHosting	If true (default), buckets will be referenced in the request using the hosted-style request: <code>http://yourbucket.s3.amazonaws.com/yourobject</code> . If set to false, the bean will use the path-style request: <code>http://s3.amazonaws.com/yourbucket/yourobject</code> . Note that this property will be set to false, in case of an S3 based custom service when the CustomURL is specified.

UseVirtualHosting

If true (default), buckets will be referenced in the request using the hosted-style request: `http://yourbucket.s3.amazonaws.com/yourobject`. If set to false, the bean will use the path-style request: `http://s3.amazonaws.com/yourbucket/yourobject`. Note that this property will be set to false, in case of an S3 based custom service when the CustomURL is specified.

Data Type

bool

Default Value

true

Remarks

If true (default), buckets will be referenced in the request using the hosted-style request: `http://yourbucket.s3.amazonaws.com/yourobject`. If set to false, the bean will use the path-style request: `http://s3.amazonaws.com/yourbucket/yourobject`. Note that this property will be set to false, in case of an S3 based custom service when the CustomURL is specified.

Azure Authentication

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

Property	Description
AzureTenant	The Microsoft Online tenant being used to access data. If not specified, your default tenant will be used.

AzureTenant

The Microsoft Online tenant being used to access data. If not specified, your default tenant will be used.

Data Type

string

Default Value

""

Remarks

The Microsoft Online tenant being used to access data. For instance, contoso.onmicrosoft.com. Alternatively, specify the tenant Id. This value is the directory Id in the Azure Portal > Azure Active Directory > Properties.

Typically it is not necessary to specify the Tenant. This can be automatically determined by Microsoft when using the [OAuthGrantType](#) set to CODE (default). However, it may fail in the case that the user belongs to multiple tenants. For instance, if an Admin of domain A invites a user of domain B to be a guest user. The user will now belong to both tenants. It is a good practice to specify the Tenant, although in general things should normally work without having to specify it.

The [AzureTenant](#) is required when setting [OAuthGrantType](#) to CLIENT. When using client credentials, there is no user context. The credentials are taken from the context of the app itself. While Microsoft still allows client credentials to be obtained without specifying which Tenant, it has a much lower probability of picking the specific tenant you want to work with. For this reason, we require [AzureTenant](#) to be explicitly stated for all client credentials connections to ensure you get credentials that are applicable for the domain you intend to connect to.

SSO

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

Property	Description
ProofKey	The ProofKey for authentication with Snowflake database. This is usually derived from GetSSOAuthorizationURL call.
ExternalToken	The External Token for authentication with the Snowflake database. This is usually derived from the external handler. For example, handle the callback URL from procedure GetSSOAuthorizationURL will get this token.
SSOProperties	Additional properties required to connect to the identity provider in a semicolon-separated list.

ProofKey

The ProofKey for authentication with Snowflake database. This is usually derived from GetSSOAuthorizationURL call.

Data Type

string

Default Value

""

Remarks

ExternalToken

The External Token for authentication with the Snowflake database. This is usually derived from the external handler. For example, handle the callback URL from procedure GetSSOAuthorizationURL will get this token.

Data Type

string

Default Value

""

Remarks

SSOProperties

Additional properties required to connect to the identity provider in a semicolon-separated list.

Data Type

string

Default Value

""

Remarks

Additional properties required to connect to the identity provider in a semicolon-separated list. The following sections provide examples using the Okta provider.

OKTA

- **Domain** is the Okta domain you are signing in with, for example: myorg.okta.com.
- **APIToken** is your Okta API token. In most cases it is unnecessary but can be provided if needed.

KeyPairAuth

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

Property	Description
PrivateKey	The private key provided for key pair authentication with Snowflake.
PrivateKeyPassword	The password for the private key specified in the PrivateKey property, if required.
PrivateKeyType	The type of key store containing the private key to use with key pair authentication.

PrivateKey

The private key provided for key pair authentication with Snowflake.

Data Type

string

Default Value

""

Remarks

The path to the file containing the private key or the name of the certificate store for the client certificate. The [PrivateKeyType](#) field specifies the type of the certificate store specified by [PrivateKey](#). If the store is password protected, specify the password in [PrivateKeyPassword](#).

When the certificate store type is PEMKEY_FILE, PFXFILE, etc., this property must be set to the path to the file. When the type is PEMKEY_BLOB, PFXBLOB, etc., the property must be set to the binary contents of the file.

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.

PrivateKeyPassword

The password for the private key specified in the [PrivateKey](#) property, if required.

Data Type

string

Default Value

""

Remarks

The password for the private key specified in the [PrivateKey](#) property, if required.

PrivateKeyType

The type of key store containing the private key to use with key pair authentication.

Possible Values

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

Data Type

string

Default Value

"USER"

Remarks

This property can take one of the following values:

USER -
default

For Windows, this specifies that the certificate store is a certificate store owned by the current user.

	Note that this store type is not available in Java.
MACHINE	For Windows, this specifies that the certificate store is a machine store. Note that this store type is not available in Java.
PFXFILE	The certificate store is the name of a PFX (PKCS12) file containing certificates.
PFXBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in PFX (PKCS12) format.
JKSFILE	The certificate store is the name of a Java key store (JKS) file containing certificates. Note that this store type is only available in Java.
JKSBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in JKS format. Note that this store type is only available in Java.
PEMKEY_FILE	The certificate store is the name of a PEM-encoded file that

		contains a private key and an optional certificate.
PEMKEY_BLOB		The certificate store is a string (base64-encoded) that contains a private key and an optional certificate.
	PUBLIC_KEY_FILE	The certificate store is the name of a file that contains a PEM- or DER-encoded public key certificate.
	PUBLIC_KEY_BLOB	The certificate store is a string (base-64-encoded) that contains a PEM- or DER-encoded public key certificate.
	SSHPUBLIC_KEY_FILE	The certificate store is the name of a file that contains an SSH-style public key.
	SSHPUBLIC_KEY_BLOB	The certificate store is a string (base-64-encoded) that contains an SSH-style public key.
P7BFILE		The certificate store is the name of a PKCS7 file containing certificates.
PPKFILE		The certificate store is the name of a file that contains a PuTTY Private Key (PPK).

XMLFILE	The certificate store is the name of a file that contains a certificate in XML format.
XMLBLOB	The certificate store is a string that contains a certificate in XML format.

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.
CallbackURL	The OAuth callback URL to return to when authenticating. This value must match the callback URL you specify in your Add-In settings.
State	An optional value that has meaning for your OAuth App.
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://.
OAuthAuthenticator	This determines the authenticator that the OAuth application requests from Snowflake.
Scope	This determines the scopes that the OAuth application requests from Snowflake.
OAuthAuthorizationURL	The authorization URL for the OAuth service.
OAuthAccessTokenURL	The URL to retrieve the OAuth access token from.
OAuthVerifier	The verifier code returned from the OAuth authorization URL.
PKCEVerifier	A random value used as input for calling GetOAuthAccessToken in the PKCE flow.
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.

CallbackURL

The OAuth callback URL to return to when authenticating. This value must match the callback URL you specify in your Add-In settings.

Data Type

string

Default Value

""

Remarks

During the authentication process, the OAuth authorization server redirects the user to this URL. This value must match the callback URL you specify in your Add-In settings.

State

An optional value that has meaning for your OAuth App.

Data Type

string

Default Value

""

Remarks

Used in OAuth authentication: This is an optional value that has meaning for your OAuth App.

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%\CDData\Snowflake 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%\CDData\Snowflake 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

OAuthAuthenticator

This determines the authenticator that the OAuth application requests from Snowflake.

Possible Values

None, Azure, OKTA

Data Type

string

Default Value

"None"

Remarks

This determines the authenticator that the OAuth application requests from Snowflake.

Scope

This determines the scopes that the OAuth application requests from Snowflake.

Data Type

string

Default Value

""

Remarks

By default the adapter will request that the user authorize all available scopes. If you want to override this, you can set this property to a space-separated list of OAuth scopes.

OAuthAuthorizationURL

The authorization URL for the OAuth service.

Data Type

string

Default Value

""

Remarks

The authorization URL for the OAuth service. At this URL, the user logs into the server and grants permissions to the application. In OAuth 1.0, if permissions are granted, the request token is authorized.

OAuthAccessTokenURL

The URL to retrieve the OAuth access token from.

Data Type

string

Default Value

""

Remarks

The URL to retrieve the OAuth access token from. In OAuth 1.0, the authorized request token is exchanged for the access token at this URL.

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.

PKCEVerifier

A random value used as input for calling GetOAuthAccessToken in the PKCE flow.

Data Type

string

Default Value

""

Remarks

This is usually derived from [GetOAuthAuthorizationUrl](#) call.

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.

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----- MIICHTCCAE4CAQAwDQYJKoZIhvd.....Qw== -----END CERTIFICATE-----
A path to a local file containing the certificate	C:\cert.cer
The public key (example shortened for brevity)	-----BEGIN RSA PUBLIC KEY----- MIGfMA0GCSq.....AQAB -----END RSA PUBLIC KEY--- --
The MD5 Thumbprint (hex values can also be either space or colon separated)	ecadbdda5a1529c58a1e9e09828d70e4
The SHA1 Thumbprint (hex values can also be either space or colon separated)	34a929226ae0819f2ec14b4a3d904f801cbb150d

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

Certificates are validated as trusted by the machine based on the System's trust store. The trust store used is the 'javax.net.ssl.trustStore' value specified for the system. If no value is specified for this property, Java's default trust store is used (for example, JAVA_HOME\lib\security\cacerts).

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

Firewall

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

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

FirewallType

The protocol used by a proxy-based firewall.

Possible Values

NONE, TUNNEL, SOCKS4, SOCKS5

Data Type

string

Default Value

"NONE"

Remarks

This property specifies the protocol that the adapter will use to tunnel traffic through the [FirewallServer](#) proxy. Note that by default, the adapter connects to the system proxy; to

disable this behavior and connect to one of the following proxy types, set [ProxyAutoDetect](#) to false.

Type	Default Port	Description
TUNNEL	80	When this is set, the adapter opens a connection to Snowflake 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.
Database	The name of the Snowflake database.
Schema	The schema of the Snowflake database.

Location

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

Data Type

string

Default Value

"%APPDATA%\CData\Snowflake 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\Snowflake 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

Database

The name of the Snowflake database.

Data Type

string

Default Value

""

Remarks

The name of the Snowflake database.

Schema

The schema of the Snowflake database.

Data Type

string

Default Value

""

Remarks

The schema of the Snowflake database.

Miscellaneous

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

Property	Description
AllowPreparedStatement	Prepare a query statement before its execution.
AsyncQueryTimeout	The timeout for asynchronous requests issued by the provider to download large result sets.
CustomStage	The name of a custom stage to use during bulk write operations.
EnableArrow	Whether to support Apache Arrow.
ExternalStageAWSAccessKey	Your AWS account access key. Only used when defining a CustomStage for bulk write operations.
ExternalStageAWSSecretKey	Your AWS account secret key. Only used when defining a CustomStage for bulk write operations.
ExternalStageAzureSASToken	The string value of the Azure Blob shared access signature.
IgnoreCase	Whether to ignore case in identifiers. Default: false.
IncludeTableTypes	If set to true, the provider will report the types of individual tables and views.
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.
MaxThreads	Specifies the number of concurrent requests.
MergeDelete	A boolean indicating whether batch DELETE statements should be converted to MERGE statements automatically.

	Only used when the DELETE statement's where clause contains a table's primary key field only and they are combined with AND logical operator.
MergeInsert	A boolean indicating whether INSERT statements should be converted to MERGE statements automatically. Only used when the INSERT contains a table's primary key field.
MergeUpdate	A boolean indicating whether batch UPDATE statements should be converted to MERGE statements automatically. Only used when the UPDATE statement's where clause contains a table's primary key field only and they are combined with AND logical operator.
Other	These hidden properties are used only in specific use cases.
Pagesize	The maximum number of results to return per page from Snowflake.
Readonly	You can use this property to enforce read-only access to Snowflake from the provider.
ReplaceInvalidUTF8Chars	Specifies whether to replace invalid UTF8 characters with a '?'.
RetryOnS3Timeout	Whether or not to retry when network issues occur at during chunk downloading.
SessionParameters	The session parameters for Snowflake. For example: SessionParameters='QUERY_TAG=MyTag;QUOTED_IDENTIFIERS_IGNORE_CASE=True;';
Timeout	The value in seconds until the timeout error is thrown, canceling the operation.
UseAsyncQuery	This field sets whether async query is enabled.
UserDefinedViews	A filepath pointing to the JSON configuration file containing your custom views.

AllowPreparedStatement

Prepare a query statement before its execution.

Data Type

bool

Default Value

false

Remarks

If the AllowPreparedStatement property is set to false, statements are parsed each time they are executed. Setting this property to false can be useful if you are executing many different queries only once.

If you are executing the same query repeatedly, you will generally see better performance by leaving this property at the default, true. Preparing the query avoids recompiling the same query over and over. However, prepared statements also require the adapter to keep the connection active and open while the statement is prepared.

AsyncQueryTimeout

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

Data Type

int

Default Value

300

Remarks

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

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

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

CustomStage

The name of a custom stage to use during bulk write operations.

Data Type

string

Default Value

""

Remarks

The name of a custom stage to use during bulk write operations. This can be an internal or external stage. If the stage is external, the AWS or Azure credentials must be provided as well via the `ExternalStageAWSAccessKey/ExternalStageAWSSecretKey` or `ExternalStageAzureAccessKey` properties.

When the `CustomStage` property is left unspecified, the adapter will generate a temporary stage automatically during the upload process and delete it after the upload is complete.

To avoid parsing errors with the generated CSV, you should include the `FIELD_OPTIONALLY_ENCLOSED_BY` parameter on the stage definition and set it to the double quote character. Otherwise, you may face parsing issues if you have string values that contain special characters in CSV (commas, double quotes, etc.). For example:

```
CREATE STAGE "TEST_STAGE_CDATA" FILE_FORMAT = ( FIELD_OPTIONALLY_
ENCLOSED_BY='\"' )
```

EnableArrow

Whether to support Apache Arrow.

Data Type

bool

Default Value

true

Remarks

Whether to support Apache Arrow.

ExternalStageAWSAccessKey

Your AWS account access key. Only used when defining a CustomStage for bulk write operations.

Data Type

string

Default Value

""

Remarks

Your AWS account access key. This value is accessible from your AWS security credentials page:

1. Sign into the AWS Management console with the credentials for your root account.
2. Select your account name or number and select My Security Credentials in the menu that is displayed.
3. Click Continue to Security Credentials and expand the Access Keys section to manage or create root account access keys.

ExternalStageAWSSecretKey

Your AWS account secret key. Only used when defining a CustomStage for bulk write operations.

Data Type

string

Default Value

""

Remarks

Your AWS account secret key. This value is accessible from your AWS security credentials page:

1. Sign into the AWS Management console with the credentials for your root account.
2. Select your account name or number and select My Security Credentials in the menu that is displayed.
3. Click Continue to Security Credentials and expand the Access Keys section to manage or create root account access keys.

ExternalStageAzureSASToken

The string value of the Azure Blob shared access signature.

Data Type

string

Default Value

""

Remarks

The string value of the Azure Blob shared access signature.

You can go to "Shared access signature" in "Settings" section for your Azure Blob container through Azure Portal, then click "Generate SAS token and URL" and copy the value from "Blob SAS token" textbox. Please be cautious to select the proper permission (Create, Write, Delete) in "Permissions" dropdown list and validity of Start and Expiry time before you generate SAS token.

IgnoreCase

Whether to ignore case in identifiers. Default: false.

Data Type

bool

Default Value

false

Remarks

A session parameter that specifies whether Snowflake will treat identifiers as case sensitive. Default: false(case is sensitive).

IncludeTableTypes

If set to true, the provider will report the types of individual tables and views.

Data Type

bool

Default Value

false

Remarks

If set to true, the adapter will report the types of individual tables and views.

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.

MaxThreads

Specifies the number of concurrent requests.

Data Type

string

Default Value

"5"

Remarks

This property allows you to issue multiple requests simultaneously, thereby improving performance.

MergeDelete

A boolean indicating whether batch DELETE statements should be converted to MERGE statements automatically. Only used when the DELETE statement's where clause contains a table's primary key field only and they are combined with AND logical operator.

Data Type

bool

Default Value

false

Remarks

A boolean indicating whether DELETE statements should be converted to MERGE statements automatically to allow for upsert functionality. This property is primarily intended for use with tools where you have no direct control over the queries being executed. Otherwise, as long as Query Passthrough is True, you could execute the MERGE command directly.

When this property is False, DELETE bulk statements won't be executed against the server. When it is set to True and the DELETE query contains the primary key field, the Snowflake will send a MERGE query that will execute a DELETE if a match is found in Snowflake. For example, this query:

```
DELETE FROM "Table" WHERE "ID" = 1 AND "NAME" = 'Jerry'
```

Will be sent to Snowflake as the following MERGE request:

```
MERGE INTO "Table" AS "Target" USING "RTABLE1_TMP_20eca05b-c050-47dd-89bc-81c7f617f877" AS "Source" ON ("Target"."ID" = "Source"."ID" AND "Target"."NAME" = "Source"."NAME")  
WHEN MATCHED THEN DELETE
```

MergeInsert

A boolean indicating whether INSERT statements should be converted to MERGE statements automatically. Only used when the INSERT contains a table's primary key field.

Data Type

bool

Default Value

false

Remarks

A boolean indicating whether INSERT statements should be converted to MERGE statements automatically to allow for upsert functionality. This property is primarily intended for use with tools where you have no direct control over the queries being executed. Otherwise, as long as Query Passthrough is True, you could execute the MERGE command directly.

When this property is False, INSERT statements are executed directly against the server. When it is set to True and the INSERT query contains the primary key field, the Snowflake will send a MERGE query that will execute an INSERT if no match is found in Snowflake or an UPDATE if it is. For example this query:

```
INSERT INTO "Table" ("ID", "NAME", "AGE") VALUES (1, 'NewName', 10)
```

Will be sent to Snowflake as the following MERGE request:

```
MERGE INTO "Table" AS "Target" USING (SELECT 1 AS "ID") AS [Source] ON  
("Target"."ID" = "Source"."ID")  
WHEN NOT MATCHED THEN INSERT ("ID", "NAME", "AGE") VALUES (1, 'NewName',  
10)  
WHEN MATCHED THEN UPDATE SET "NAME" = 'NewName', "AGE" = 10
```

MergeUpdate

A boolean indicating whether batch UPDATE statements should be converted to MERGE statements automatically. Only used when the UPDATE statement's where clause contains a table's primary key field only and they are combined with AND logical operator.

Data Type

bool

Default Value

false

Remarks

A boolean indicating whether UPDATE statements should be converted to MERGE statements automatically to allow for upsert functionality. This property is primarily intended for use with tools where you have no direct control over the queries being executed. Otherwise, as long as Query Passthrough is True, you could execute the MERGE command directly.

When this property is False, UPDATE statements are executed directly against the server. When it is set to True and the UPDATE query contains the primary key field, the Snowflake will send a MERGE query that will execute an INSERT if no match is found in Snowflake or an UPDATE if it is. For example this query:

```
UPDATE "Table" SET "NAME" = 'NewName', "AGE" = 10 WHERE "ID" = 1
```

Will be sent to Snowflake as the following MERGE request:

```
MERGE INTO "Table" AS "Target" USING "RTABLE1_TMP_20eca05b-c050-47dd-89bc-81c7f617f877" AS "Source" ON ("Target"."ID" = "Source"."ID")  
WHEN MATCHED THEN UPDATE SET "Target"."NAME" = "Source"."NAME",  
"Target"."AGE" = "Source"."AGE"
```

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.

Pagesize

The maximum number of results to return per page from Snowflake.

Data Type

int

Default Value

5000

Remarks

The Pagesize property affects the maximum number of results to return per page from Snowflake. Setting a higher value may result in better performance at the cost of additional memory allocated per page consumed.

Readonly

You can use this property to enforce read-only access to Snowflake 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.

ReplaceInvalidUTF8Chars

Specifies whether to replace invalid UTF8 characters with a '?'.

Data Type

bool

Default Value

false

Remarks

Specifies whether to replace invalid UTF8 characters with a '?'

RetryOnS3Timeout

Whether or not to retry when network issues occur at during chunk downloading.

Data Type

bool

Default Value

false

Remarks

Typically if a network issue such as a timeout occurs during chunk downloading of data, the Snowflake Adapter will throw an exception. Set this property to true to cause the Snowflake Adapter to attempt retrying the request before failing.

SessionParameters

The session parameters for Snowflake. For example: SessionParameters='QUERY_TAG=MyTag;QUOTED_IDENTIFIERS_IGNORE_CASE=True;';.

Data Type

string

Default Value

""

Remarks

The session parameters for Snowflake. For example: SessionParameters='QUERY_TAG=MyTag;QUOTED_IDENTIFIERS_IGNORE_CASE=True;';

Timeout

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

Data Type

int

Default Value

120

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.

UseAsyncQuery

This field sets whether async query is enabled.

Data Type

bool

Default Value

false

Remarks

This field sets whether async query is enabled.

UserDefinedViews

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

Data Type

string

Default Value

""

Remarks

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

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

This User Defined View configuration file is formatted as follows:

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

For example:

```
{
  "MyView": {
    "query": "SELECT * FROM [DemoDB].[PUBLIC].Products WHERE MyColumn =
'value'"
  },
  "MyView2": {
    "query": "SELECT * FROM MyTable WHERE Id IN (1,2,3)"
  }
}
```

Use the [UserDefinedViews](#) connection property to specify the location of your JSON configuration file. For example:

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

Snowflake Adapter Limitations

Following are some of the limitations of the Snowflake adapter:

1. The following datatypes are not supported by Snowflake. Tables with columns of these datatypes cannot be cached:

- - BLOB
- - CLOB

2. When using `cache_status/cache_tracking` and caching is done in the One-Table-Per-Snapshot mode, it is recommended to use a different datasource as the container.

3. Week start is configurable in Snowflake through user session parameter - `WEEK_START`. If `WEEK_START` is set to 1, `DayOfWeek` function returns the same value as TDV.

4. For non-analytic functions (`COUNT`, `MIN` / `MAX`, `SUM`), the default is the following cumulative window frame (in accordance with the ANSI standard):

`RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`

For analytic functions (`FIRST_VALUE`, `LAST_VALUE`, `NTH_VALUE`), the default is the entire window.

Note that this deviates from the ANSI standard.

TIBCO Product Documentation and Support Services

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

How to Access TIBCO Documentation

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

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

Product-Specific Documentation

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

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

TDV Adapter Guides

TDV Data Source Toolkit Guide (Formerly Extensibility Guide)

- **References**

TDV Reference Guide

TDV Application Programming Interface Guide

- **Other**

TDV Business Directory Guide

TDV Discovery Guide

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

How to Contact TIBCO Support

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

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

Release Version Support

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

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

How to Join TIBCO Community

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

Legal and Third-Party Notices

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

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

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

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

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

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

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

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

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

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

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

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

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