Getting Started With Snowflake SQL API

Overview

Welcome! The Snowflake SQL API is a REST API that you can use to access and update data in a Snowflake database. You can use this API to execute [standard queries] and most [DDL] and [DML] statements.

This getting started guide will walk you through executing a SQL statement with the API and retrieving the results.

Prerequisites

- A familiarity with Snowflake
- A familiarity with SQL

What You'll Learn

- Perform simple queries
- Manage your deployment (e.g., provision users and roles, create tables, etc.)
- Submit one SQL statement for execution per API call.
- Check the status of the execution of a statement.
- Cancel the execution of a statement.

What You'll Need

- A Snowflake Account with an accessible warehouse, database, schema, and role
- SnowSQL 1.2.17 or higher
- Working Key-Pair authentication

What You'll Build

- An execution of a statement using the Snowflake SQL API

Introducing the API

Head to the SQL API by navigating to your version of the following URL, replacing `*account_locator*` with the account locator for your own Snowflake account:

```
https://*account_locator*.snowflakecomputing.com/api/v2
```

Negative : Note that the account locator might include additional segments for your region and cloud provider. See [Specifying Region Information in Your Account Hostname] for details.

Now let's break down the parts of the API before we begin using it. The API consists of the `/api/v2/statements/` resource and provides the following endpoints:

- `/api/v2/statements` : You'll use this endpoint to [submit a SQL statement for execution].
- `/api/v2/statements/*statementHandle*` : You'll use this endpoint to [check the status of the execution of a statement].
- `/api/v2/statements/*statementHandle*/cancel` : You'll use this endpoint to [cancel the execution of a statement].

In the steps to come, you shall use all these endpoints to familiarize yourself with the API.

Positive : You can use development tools and libraries for REST APIs (e.g., Postman) to send requests and handle responses.

Limitations of the SQL API

It's important to be aware of the [limitations that the SQL API] currently has. In particular noting that `GET` and `PUT` are not supported.

Assigning a Unique Request ID for Resubmitting Requests

In some cases, it might not be clear if Snowflake executed the SQL statement in an API request (e.g., due to a network error or a timeout). You might choose to resubmit the same request to Snowflake again in case Snowflake did not execute the statement.

If Snowflake already executed the statement in the initial request and you resubmit the request again, the statement is executed twice. For some types of requests, repeatedly executing the same statement can have unintended consequences (e.g., inserting duplicate data into a table).

To prevent Snowflake from executing the same statement twice when you resubmit a request, you can use a request ID to distinguish your request from other requests. Suppose you specify the same request ID in the initial request and in the resubmitted request. In that case, Snowflake does not execute the statement again if the statement has already executed successfully.

To specify a request ID, generate a universally unique identifier (UUID) and include this identifier in the `requestId` query parameter:

```
POST /api/v2/statements?requestId=<UUID> HTTP/1.1
```

If Snowflake fails to process a request, you can submit the same request again with the same request ID. Using the same request ID indicates to the server that you are submitting the same request again.

Now let's move on to additional information you need to include in requests: authentication parameters.

Authenticating to the Server

When you send a request, the request must include authentication information. There are two options for providing authentication: OAuth and JWT key pair authentication. You can use whichever one you have previously implemented or whichever one you are most comfortable with. This example will be detailing authentication with JWT.

If you haven't done so already, make sure you have [key pair authentication] working with Snowflake already.

You can test to make sure you can successfully connect to Snowflake Key Pairs using the following command:

```
$ snowsql -a <account> -u <user> --private-key-path <path to private key>
```

After you've verified you can connect to Snowflake using key-pair authentication, you'll need to generate a JWT token. This JWT token is time limited token which has been signed with your key and Snowflake will know that you authorized this token to be used to authenticate as you for the SQL API.

```
$ snowsql -a <account> -u <user> --private-key-path <path to private key> --generate-
jwt
<returns JWT token>
```

You'll need the JWT token generated to be used for using the SQL API. The following headers need be set in each API request that you send within your application code:

- `Authorization: Bearer *jwt_token*` where `*jwt_token*` is the generated JWT token from SnowSQL

- `X-Snowflake-Authorization-Token-Type: KEYPAIR_JWT`

Altogether, your request query and header will take the following form:

```
POST /api/v2/statements?requestId=<UUID> HTTP/1.1
Authorization: Bearer <jwt_token>
Content-Type: application/json
Accept: application/json
User-Agent: myApplication/1.0
X-Snowflake-Authorization-Token-Type: KEYPAIR_JWT
```

Now that you have been introduced to authentication and unique request IDs, you can now move to actually making a request to execute a SQL statement.

Submitting a Request to Execute a SQL Statement

To submit a SQL statement for execution, send a [POST request to the /api/v2/statements/ endpoint]:

```
POST /api/v2/statements?requestId=<UUID> HTTP/1.1
Authorization: Bearer <jwt_token>
Content-Type: application/json
Accept: application/json
User-Agent: myApplication/1.0
X-Snowflake-Authorization-Token-Type: KEYPAIR_JWT

(request body)
```

In the request URL, you can also set query parameters to:

- Execute the statement asynchronously: `async=true`

For the [body of the request], set the following fields:

- Set the `statement` field to the SQL statement that you want to execute.
- To specify the warehouse, database, schema, and role to use, set the `warehouse`, `database`, `schema`, and `role` fields.

Negative : Note: the values in these fields are case-sensitive.

- To set a timeout for the statement execution, set the `timeout` field to the maximum number of seconds to wait. If the `timeout` field is not set, the timeout specified by the [STATEMENT_TIMEOUT_IN_SECONDS] parameter is used.

```
POST /api/v2/statements HTTP/1.1
Authorization: Bearer <jwt_token>
Content-Type: application/json
Accept: application/json
User-Agent: myApplication/1.0
X-Snowflake-Authorization-Token-Type: KEYPAIR_JWT

{
"statement": "select * from T",
"timeout": 60,
"database": "<your_database>",
"schema": "<your_schema>",
"warehouse": "<your_warehouse>",
"role": "<your_role>"
}
```

Let's go over some specific fields in this request:

- The `statement` field specifies the SQL statement to execute.
- The `timeout` field specifies that the server allows 60 seconds for the statement to be executed.

If the statement was executed successfully, Snowflake returns the HTTP response code 200 and the first results in a [ResultSet] object. We'll go over how to check the status and retrieve the results after we look at including bind variables.

Now we'll look at how you can include bind variables (`?` placeholders) in the statement and set the `bindings` field to an object that specifies the corresponding Snowflake data types and values for each variable.

Using Bind Variables in a Statement

If you want to use bind variables (`?` placeholders) in the statement, use the `bindings` field to specify the values that should be inserted.

Set this field to a JSON object that specifies the [Snowflake data type] and value for each bind variable.

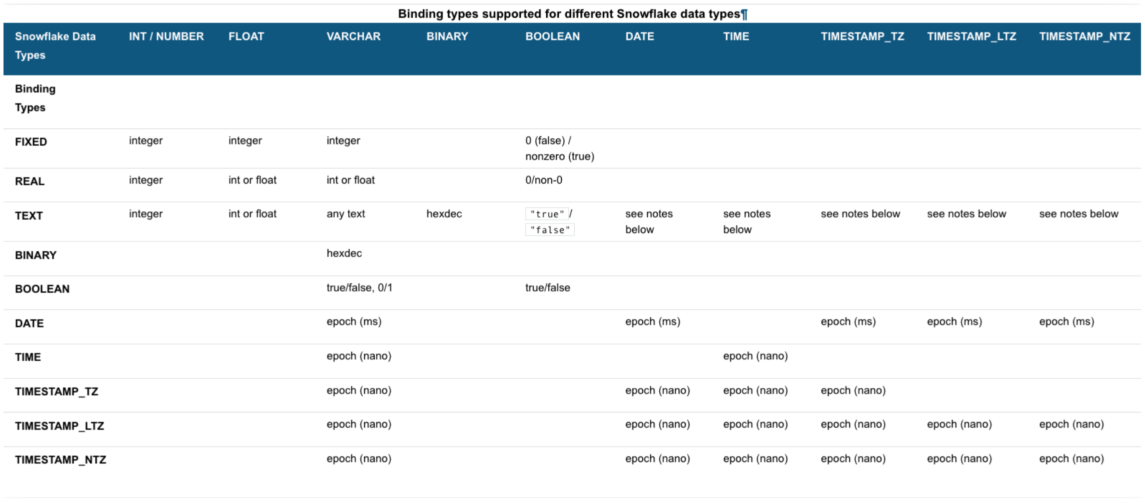
```
...
"statement": "select * from T where c1=?",
...
"bindings": {
"1": {
"type": "FIXED",
"value": "123"
}
},
...
```

Choose the binding type that corresponds to the type of the value that you are binding. For example, if the value is a string representing a date (e.g. `2021-04-15`) and you want to insert the value into a DATE column, use the `TEXT` binding type.

The following table specifies the values of the `type` field that you can use to bind to different [Snowflake data types] for this preview release.

- The first column on the left specifies the binding types that you can use.
- The rest of the columns specify the Snowflake data type of the column where you plan to insert the data.
- Each cell specifies the type of value that you can use with a binding type to insert data into a column of a particular Snowflake data type.

Negative : If the cell for a binding type and Snowflake data type is empty, you cannot use the specified binding type to insert data into a column of that Snowflake data type.



Binding types supported for different Snowflake data types¶

Snowflake Data Types	INT / NUMBER	FLOAT	VARCHAR	BINARY	BOOLEAN	DATE	TIME	TIMESTAMP_TZ	TIMESTAMP_LTZ	TIMESTAMP_NTZ
Binding Types										
FIXED	integer	integer	integer		0 (false) / nonzero (true)					
REAL	integer	int or float	int or float		0/non-0					
TEXT	integer	int or float	any text	hexdec	`"true"` / `"false"`	see notes below	see notes below	see notes below	see notes below	see notes below
BINARY			hexdec							
BOOLEAN			true/false, 0/1		true/false					
DATE			epoch (ms)			epoch (ms)		epoch (ms)	epoch (ms)	epoch (ms)
TIME			epoch (nano)				epoch (nano)			
TIMESTAMP_TZ			epoch (nano)			epoch (nano)	epoch (nano)	epoch (nano)		
TIMESTAMP_LTZ			epoch (nano)			epoch (nano)	epoch (nano)	epoch (nano)	epoch (nano)	epoch (nano)
TIMESTAMP_NTZ			epoch (nano)			epoch (nano)	epoch (nano)	epoch (nano)	epoch (nano)	epoch (nano)

For additional information on binding specific data types, see the documentation's section on [Using Bind Variables in a Statement]

If the value is in a format not supported by Snowflake, the API returns an error:

```
{
code: "100037",
message: "<bind type> value '<value>' is not recognized",
sqlState: "22018",
statementHandle: "<ID>"
}
```

Whether you use bind variables or not, you'll want to check the status of your statements. Let's look at that next.

Checking the Status of the Execution of the Statement

When you submit a SQL statement for execution, Snowflake returns a 202 response code if the execution of the statement has not yet been completed or if you submitted an asynchronous query.

In the body of the response, Snowflake includes a [QueryStatus] object. The `statementStatusUrl` field in this object specifies the URL to the [/api/v2/statements/ endpoint] that you can use to check the execution status:

```
{
"code": "090001",
"sqlState": "00000",
"message": "successfully executed",
"statementHandle": "e4ce975e-f7ff-4b5e-b15e-bf25f59371ae",
"statementStatusUrl": "/api/v2/statements/e4ce975e-f7ff-4b5e-b15e-bf25f59371ae"
}
```

As illustrated by the URL above, in requests to check the status of a statement and cancel the execution of a statement, you specify the statement handle (a unique identifier for the statement) as a path parameter in order to identify the statement to use.

Note that the `QueryStatus` object also provides the statement handle as a separate value in the `statementHandle` field.

To check the status of the execution of the statement, send a GET request using this URL:

```
GET /api/v2/statements/{statementHandle}
```

For example, the following request checks the execution status of the statement with the handle `e4ce975e-f7ff-4b5e-b15e-bf25f59371ae`:

```
GET /api/v2/statements/e4ce975e-f7ff-4b5e-b15e-bf25f59371ae HTTP/1.1
Authorization: Bearer <jwt_token>
Content-Type: application/json
Accept: application/json
User-Agent: myApplication/1.0
X-Snowflake-Authorization-Token-Type: KEYPAIR_JWT
```

If the statement has finished executing successfully, Snowflake returns the HTTP response code 200 and the first results in a [ResultSet] object. However, if an error occurred when executing the statement, Snowflake returns the HTTP response code 422 with a [QueryFailureStatus] object.

Once the statement has executed successfully, you can then retrieve the results, detailed in the next step.

Cancelling the Execution of a SQL Statement

To cancel the execution of a statement, send a POST request to the [cancel endpoint].

```
POST /api/v2/statements/{statementHandle}/cancel
```

Retrieving the Results

If you [submit a SQL statement for execution] or [check the status of statement execution], Snowflake returns a [ResultSet] object in the body of the response if the statement was executed successfully.

The following is an example of a `ResultSet` object that is returned for a query, truncated for brevity.

```
{
  "code" : "090001",
  "statementStatusUrl" : "/api/v2/statements/01a288b9-0603-af68-0000-328502422e7e?
requestId=f8ccd534-7cd5-4c06-b673-f25361e96d7f",
  "requestId" : "f8ccd534-7cd5-4c06-b673-f25361e96d7f",
  "sqlState" : "00000",
  "statementHandle" : "01a288b9-0603-af68-0000-328502422e7e",
  "message" : "Statement executed successfully.",
  "createdOn" : 1645742998434,
  "resultSetMetaData" : {
    "rowType" : [ {
      "name" : "HIGH_NDV_COLUMN",
      "database" : "",
      "schema" : "",
      "table" : "",
      "type" : "fixed",
      "scale" : 0,
      "precision" : 19,
      "byteLength" : null,
      "nullable" : false,
      "collation" : null,
      "length" : null
    }, {
      "name" : "LOW_NDV_COLUMN",
      "database" : "",
      "schema" : "",
      "table" : "",
      "type" : "fixed",
      "scale" : 0,
      "precision" : 2,
      "byteLength" : null,
      "nullable" : false,
      "collation" : null,
      "length" : null
    }, {
      "name" : "CONSTANT_COLUM",
      "database" : "",
      "schema" : "",
      "table" : "",
      "type" : "fixed",
      "scale" : 0,
      "precision" : 1,
      "byteLength" : null,
      "nullable" : false,
      "collation" : null,
      "length" : null
    } ],
    "numRows" : 100000000,
    "format" : "jsonv2",
    "partitionInfo" : [ {
      "rowCount" : 8192,
      "uncompressedSize" : 152879,
      "compressedSize" : 22412
    }, {
      "rowCount" : 53248,
      "uncompressedSize" : 1048161,
      "compressedSize" : 151251
    }, {
      "rowCount" : 86016,
      "uncompressedSize" : 1720329,
      "compressedSize" : 249447
    }, {
```

Notice that there is an `ARRAY` of `partitionInfo` objects. These partitions objects give you some information, such as rowCount and size, about the partitions that are available for retrieval. The API returns the data data inline in JSON of the first partition, or partition `0` with the response and we'll cover how to retrieve subsequent partitions in a later section.

Getting Metadata About the Result

It also includes, in the response the `rowType` which gives additional metadata about the datatypes and names of data returned from the query. This metadata is only included in the initial response, and no metadata is returned when retrieving subsequent partitions.

In the `ResultSet` object returned in the response, the `resultSetMetaData` field contains a [`ResultSet_resultSetMetaData`] object that describes the result set (for example, the format of the results, etc.).

In this object, the `rowType` field contains an array of [ResultSet_resultSetMetaData_rowType] objects. Each object describes a column in the results. The `type` field specifies the Snowflake data type of the column.

```
{
"resultSetMetaData": {
"rowType": [
{
"name":"ROWNUM",
"type":"FIXED",
"length":0,
"precision":38,
"scale":0,
"nullable":false
}, {
"name":"ACCOUNT_NAME",
"type":"TEXT",
"length":1024,
"precision":0,
"scale":0,
"nullable":false
}, {
"name":"ADDRESS",
"type":"TEXT",
"length":16777216,
"precision":0,
"scale":0,
"nullable":true
}, {
"name":"ZIP",
"type":"TEXT",
"length":100,
"precision":0,
"scale":0,
"nullable":true
}, {
"name":"CREATED_ON",
"type":"TIMESTAMP_NTZ",
"length":0,
"precision":0,
"scale":3,
"nullable":false
}
]
},
}
```

Retrieving Result Partitions

Partitions are [retrieved] using the `partition=n` query parameter at `/api/v2/statements/<handle>?partition=<partition_number>` endpoint. This can be used to iterate, or even retrieve in parallel, the results from the SQL API call.

```
GET /api/v2/statements/e4ce975e-f7ff-4b5e-b15e-bf25f59371ae?partition=1 HTTP/1.1
Authorization: Bearer <jwt_token>
Content-Type: application/json
Accept: application/json
User-Agent: myApplication/1.0
X-Snowflake-Authorization-Token-Type: KEYPAIR_JWT
```

The above query call, with `partition=1` returns just the data below. Notice that the `data` object does not contain any of the additional metadata about the result. Only the data, in gzipped format that needs to be uncompressed.

```
{"data": [
["32768","3","5"],
["32772","4","5"],
["32776","3","5"],
["32780","3","5"],
["32784","2","5"],
....
]}
```

Each array within the array contains the data for a row:

- The first element in each array is a JSON string containing a sequence ID that starts from 0.
- The rest of the elements in each array represent the data in a row.

The data in the result set is encoded in JSON v1.0, which means that all data is expressed as strings, regardless of the Snowflake data type of the column.

For example, the value `1.0` in a `NUMBER` column is returned as the string `" 1.0"`. As another example, timestamps are returned as the number of nanoseconds since the epoch. For example, the timestamp for Thursday, January 28, 2021 10:09:37.123456789 PM is returned as `"1611871777123456789"`.

You are responsible for converting the strings to the appropriate data types.

Helpful URLs to iterate and retrieve partitions

Since your results may contain significant number of partitions, the Snowflake SQL API provides a special header, `Link`, which assists in helping clients traverse and retrieve those results.

To get the next parition of results or other partitions, use the URLs provided in the [Link header] in the HTTP response. The `Link` header specifies the URLs for retrieving the first, next, previous, and last partitions of results:

```
HTTP/1.1 200 OK
Link: </api/v2/statements/01a288b9-0603-af68-0000-328502422e7e?requestId=918e2211-
d1d6-4c53-bec3-457d047651f7&partition=0>; rel="first"
,</api/v2/statements/01a288b9-0603-af68-0000-328502422e7e?requestId=51ad0c0a-e514-
4d8a-9cf2-cf537d439e39&partition=1>; rel="next"
,</api/v2/statements/01a288b9-0603-af68-0000-328502422e7e?requestId=c6a17bb3-7593-
489d-bbac-5e3b268bc6da&partition=47>; rel="last"
...
```