Experiment:Working with SnowSQL

Overview

This experiment uses the Snowflake command line client, SnowSQL, to introduce key concepts and tasks, including:

- Creating required Snowflake objects (databases, tables, etc.) for storing and querying data.
- Loading a small amount of sample data from CSV files into a table.
- Querying the table.

Note

This experiment requires a virtual warehouse to load the sample data and execute queries. A running virtual warehouse consumes Snowflake credits; however, the number of credits consumed in this experiment will be minimal because the entire experiment can be completed in an hour or so.

In addition, your account will be billed a minimal amount for the on-disk storage required for the sample data in this experiment, unless you choose to drop the table at the end of the experiment.

This experiment requires a database, table, and virtual warehouse to load and query data. Creating these Snowflake *objects* requires a Snowflake *user* with a *role* with the necessary access control *permissions*. In addition, SnowSQL is required to execute the SQL statements in the experiment. Lastly, the experiment requires CSV files containing sample data to load.

Note: You can complete the experiment using an existing Snowflake warehouse, database and table, and your own local data files; however, for simplicity, we recommend using the Snowflake objects and set of data files we've noted in the experiment.

Required User and Permissions

To create the database, table, and virtual warehouse used in this experiment, your Snowflake user must have a role that has been granted the necessary permissions to create these objects.

Note: If you did not have a Snowflake user yet, or if your user did not have an appropriate role, you would contact one of your account or security administrators (users with the ACCOUNTADMIN or SECURITYADMIN role).

SnowSQL Installation

The SnowSQL installer is available for download from the Snowflake Client Repository. No authentication is required. This version of the SnowSQL installer enables auto-upgrade for patches.

For more detailed instructions, see Installing SnowSQL.

To install SnowSQL:

1. Open a terminal window.
2. Run `curl` to download the SnowSQL installer.

   For increased flexibility, Snowflake provides both Amazon Web Services (AWS) and Azure endpoints for downloading the SnowSQL installer. Accounts hosted on any supported cloud platform can download the installer from ***either*** endpoint.

   Linux

   AWS endpoint

```
$ curl -O https://sfc-repo.snowflakecomputing.com/snowsql/bootstrap/1.2/linux_x86_64/snowsql-1.2.21-linux_x86_64.bash
```

Microsoft Azure endpoint

```
$ curl -O https://sfc-
repo.azure.snowflakecomputing.com/snowsql/bootstrap/1.2/linux_x86_64/sn
owsql-1.2.21-linux_x86_64.bash
```

macOS

AWS endpoint

```
$ curl -O https://sfc-
repo.snowflakecomputing.com/snowsql/bootstrap/1.2/darwin_x86_64/snowsql
-1.2.21-darwin_x86_64.pkg
```

Microsoft Azure endpoint

```
$ curl -O https://sfc-
repo.azure.snowflakecomputing.com/snowsql/bootstrap/1.2/darwin_x86_64/s
nowsql-1.2.21-darwin_x86_64.pkg
```

Windows

AWS endpoint

```
$ curl -O https://sfc-
repo.snowflakecomputing.com/snowsql/bootstrap/1.2/windows_x86_64/snowsq
l-1.2.21-windows_x86_64.msi
```

Microsoft Azure endpoint

```
$ curl -O https://sfc-
repo.azure.snowflakecomputing.com/snowsql/bootstrap/1.2/windows_x86_64/
snowsql-1.2.21-windows_x86_64.msi
```

3. Run the installer:

Linux

1. Open a terminal window.
2. Run the Bash script installer from the download location:

```
3. $ bash snowsql-linux_x86_64.bash
```

4. Follow the instructions provided by the installer.

macOS

5. Double-click `snowsql-darwin_x86_64.pkg` in the download location to run the installer PKG file.
6. Follow the instructions provided by the installer.

Windows

7. Double-click `snowsql-windows_x86_64.msi` in the download location to run the installer MSI file.
8. Follow the instructions provided by the installer.

Configuring the Z Shell Alias (macOS Only)

If Z shell (also known as zsh) is your default terminal shell, set an alias to the SnowSQL executable so that you can run SnowSQL on the command line in Terminal. The SnowSQL installer installs the executable in `/Applications/SnowSQL.app/Contents/MacOS/snowsql` and appends this path to the PATH or alias entry in `~/.profile`. Because zsh does not normally read this file, add an alias to this path in `~/.zshrc`, which zsh ***does*** read.

To add an alias to the SnowSQL executable:

1. Open (or create, if missing) the `~/.zshrc` file.
2. Add the following line:

```
alias snowsql=/Applications/SnowSQL.app/Contents/MacOS/snowsql
```

4. Save the file.

Sample Data Files for Loading

Download the set of sample datafiles located at https://docs.snowflake.com/en/_downloads/34f4a66f56d00340f8f7a92acaccd977/getting-started.zip, and save the file to your local file system.

You can unpack the sample files in any location; however, we recommend using the directories referenced in the experiment examples:

Linux/macOS

`/tmp`

Windows

`C:\temp`

The sample files include dummy employee data in CSV format with five records each. The field delimiter is the comma (`,`) character.

Example record:

```
Althea,Featherstone,afeatherstona@sf_tuts.com,"8172 Browning Street, Apt B",Calatrava,7/12/2017
```

Note

- There are ***no*** blank spaces before or after the commas separating the fields in each record. This is the default that Snowflake expects when loading CSV data.
- Because the field delimiter is the comma character, any commas within a field string ***must*** be escaped, or the entire field must be enclosed in double quotes (`" "`).

Log into SnowSQL

After SnowSQL is installed, log in:

1. Open a terminal window.
2. Start SnowSQL at the command prompt:

```
$ snowsql -a <account_identifier> -u <user_name>
```

Where:

- `<account_identifier>`

  Unique identifier for your Snowflake account.

  The preferred format of the account identifier is as follows:

  `organization_name``-`*`account_name`*

  Names of your Snowflake organization and account. For details, see Option 1: Account Name in Your Organization.

  Alternatively, specify your *account locator*, along with the region and cloud platform where the account is hosted, if required. For details, see Option 2: Account Locator in a Region.

- `<user_name>` is the login name for your Snowflake user.

3. When prompted by SnowSQL, enter the password for your Snowflake user.

For more details, see Connecting Through SnowSQL.

Create Snowflake Objects

A database and table are required before you can load data. This experiment loads data into a table in a database named `sf_tuts`.

In addition, loading and querying data requires a virtual warehouse, which provides the necessary compute resources to perform these tasks. You can use your own warehouse, if one is available; otherwise, this step in the experiment includes a SQL command that creates an X-Small warehouse.

When you have completed the experiment, you can drop these objects to remove them from your account.

Creating a Database

Create the `sf_tuts` database using the CREATE DATABASE command:

```
create or replace database sf_tuts;
```

Note that you do not need to create a schema in the database because each database created in Snowflake contains a default schema named `public`.

Also, note that the database and schema you just created are now in use for your current session. This information is displayed in your SnowSQL command prompt, but can also be viewed using the following context functions:

```
select current_database(), current_schema();

+--------------------+------------------+
| CURRENT_DATABASE() | CURRENT_SCHEMA() |
|--------------------+------------------|
| SF_TUTS            | PUBLIC           |
+--------------------+------------------+
```

Creating a Table

Create a table named `emp_basic` in `sf_tuts.public` using the CREATE TABLE command:

```
create or replace table emp_basic (
  first_name string ,
  last_name string ,
  email string ,
  streetaddress string ,
  city string ,
  start_date date
  );
```

Note that the number of columns in the table, their positions, and their data types correspond to the fields in the sample CSV data files that you will be staging in the next step in this experiment.

Creating a Virtual Warehouse

Create an X-Small warehouse named `sf_tuts_wh` using the CREATE WAREHOUSE command:

```
create or replace warehouse sf_tuts_wh with
  warehouse_size='X-SMALL'
  auto_suspend = 180
  auto_resume = true
  initially_suspended=true;
```

Note that the warehouse is not started initially, but it is set to auto-resume, so it will automatically start running when you execute your first SQL command that requires compute resources.

Also, note that the warehouse is now in use for your current session. This information is displayed in your SnowSQL command prompt, but can also be viewed using the following context function:

```
select current_warehouse();

+----------------------+
| CURRENT_WAREHOUSE()  |
|----------------------|
| SF_TUTS_WH           |
+----------------------+
```

Stage the Data Files

Snowflake supports loading data from files that have been staged in either an internal (Snowflake) stage or external (Amazon S3, Google Cloud Storage, or Microsoft Azure) stage. Loading from an external stage is convenient if you already store data files in these cloud storage services.

In this experiment, we will upload (stage) the sample data files, downloaded earlier in the experiment, to an internal table stage. The command used to stage files is PUT.

Staging the Files

Execute PUT to upload local data files to the table stage provided for the `emp_basic` table you created. Note that the command is OS-specific because it references files in your local environment:

- Linux or macOS
- `put file:///tmp/employees0*.csv @sf_tuts.public.%emp_basic;`
- Windows
- `put file://c:\temp\employees0*.csv @sf_tuts.public.%emp_basic;`

Let's take a closer look at the command:

- `file:` specifies the full directory path and names of the files on your local machine to stage. Note that file system wildcards are allowed.
- `@<namespace>.%<table_name>` indicates to use the stage for the specified table, in this case the `emp_basic` table.

The command returns the following response, showing the files that were staged:

```
+-----------------+--------------------+-------------+-------------+---------
----------+--------------------+----------+---------+
| source          | target             | source_size | target_size |
source_compression | target_compression | status   | message |
|-----------------+--------------------+-------------+-------------+---------
----------+--------------------+----------+---------|
| employees01.csv | employees01.csv.gz |         360 |         287 | NONE
| GZIP               | UPLOADED |         |
| employees02.csv | employees02.csv.gz |         355 |         274 | NONE
| GZIP               | UPLOADED |         |
| employees03.csv | employees03.csv.gz |         397 |         295 | NONE
| GZIP               | UPLOADED |         |
| employees04.csv | employees04.csv.gz |         366 |         288 | NONE
| GZIP               | UPLOADED |         |
| employees05.csv | employees05.csv.gz |         394 |         299 | NONE
| GZIP               | UPLOADED |         |
+-----------------+--------------------+-------------+-------------+---------
----------+--------------------+----------+---------+
```

Note that the PUT command compresses files by default using `gzip`, as indicated in the TARGET_COMPRESSION column.

Listing the Staged Files (Optional)

You can see the list of the files you successfully staged by executing a LIST command:

```
list @sf_tuts.public.%emp_basic;

+--------------------+------+----------------------------------+-------------
-----------------+
| name               | size | md5                              |
last_modified                |
|--------------------+------+----------------------------------+-------------
-----------------|
| employees01.csv.gz |  288 | a851f2cc56138b0cd16cb603a97e74b1 | Tue, 9 Jan
2018 15:31:44 GMT |
| employees02.csv.gz |  288 | 125f5645ea500b0fde0cdd5f54029db9 | Tue, 9 Jan
2018 15:31:44 GMT |
| employees03.csv.gz |  304 | eafee33d3e62f079a054260503ddb921 | Tue, 9 Jan
2018 15:31:45 GMT |
| employees04.csv.gz |  304 | 9984ab077684fbcec93ae37479fa2f4d | Tue, 9 Jan
2018 15:31:44 GMT |
| employees05.csv.gz |  304 | 8ad4dc63a095332e158786cb6e8532d0 | Tue, 9 Jan
2018 15:31:44 GMT |
+--------------------+------+----------------------------------+-------------
-----------------+
```

Copy Data into the Target Table

Execute COPY INTO <table> to load your staged data into the target table.

Note that this command requires an active, running warehouse, which you created as a prerequisite for this experiment. If you don't have access to a warehouse, you will need to create one now.

```
copy into emp_basic
  from @%emp_basic
  file_format = (type = csv field_optionally_enclosed_by='"')
  pattern = '.*employees0[1-5].csv.gz'
  on_error = 'skip_file';
```

Let's look more closely at this command:

- The `FROM` clause identifies the internal stage location.

- `FILE_FORMAT` specifies the file type as CSV, and specifies the double-quote character (`"`) as the character used to enclose strings. Snowflake supports diverse file types and options. These are described in CREATE FILE FORMAT. The example COPY statement accepts all other default file format options.
- `PATTERN` applies pattern matching to load data from all files that match the regular expression `.*employees0[1-5].csv.gz`.
- `ON_ERROR` specifies what to do when the COPY command encounters errors in the files. By default, the command stops loading data when the first error is encountered; however, we've instructed it to skip any file containing an error and move on to loading the next file. Note that this is just for illustration purposes; none of the files in this experiment contain errors.

The COPY command also provides an option for validating files before you load them. You can review the COPY INTO <table> Snowflake documentation for additional error checking and validation instructions.

Snowflake returns the following results:

```
+--------------------+--------+-------------+-------------+-------------+----
--------+-------------+------------------+-----------------------+----------
---------------+
| file               | status | rows_parsed | rows_loaded | error_limit |
errors_seen | first_error | first_error_line | first_error_character |
first_error_column_name |
|--------------------+--------+-------------+-------------+-------------+----
--------+-------------+------------------+-----------------------+----------
---------------|
| employees02.csv.gz | LOADED |           5 |           5 |           1 |
0 | NULL        |             NULL |                  NULL | NULL
|
| employees04.csv.gz | LOADED |           5 |           5 |           1 |
0 | NULL        |             NULL |                  NULL | NULL
|
| employees05.csv.gz | LOADED |           5 |           5 |           1 |
0 | NULL        |             NULL |                  NULL | NULL
|
| employees03.csv.gz | LOADED |           5 |           5 |           1 |
0 | NULL        |             NULL |                  NULL | NULL
|
```

```
| employees01.csv.gz | LOADED |           5 |           5 |           1 |
0 | NULL        |             NULL |                   NULL | NULL
|
+--------------------+--------+-------------+-------------+-------------+----
---------+-------------+--------
```

Query the Loaded Data

You can query the data loaded in the `emp_basic` table using standard SQL and any supported functions and operators .

You can also manipulate the data, such as updating the loaded data or inserting more data, using standard DML commands.

Query All Data

Return all rows and columns from the table:

```
select * from emp_basic;

-- Partial results shown

+------------+--------------+---------------------------+--------------------
---------+--------------------+------------+
| FIRST_NAME | LAST_NAME    | EMAIL                     | STREETADDRESS
| CITY               | START_DATE |
|------------+--------------+---------------------------+--------------------
---------+--------------------+------------|
| Arlene     | Davidovits   | adavidovitsk@sf_tuts.com  | 7571 New Castle
Circle      | Meniko             | 2017-05-03 |
| Violette   | Shermore     | vshermorel@sf_tuts.com    | 899 Merchant Center
| Troitsk            | 2017-01-19 |
| Ron        | Mattys       | rmattysm@sf_tuts.com      | 423 Lien Pass
| Bayaguana          | 2017-11-15 |
...
...
...
| Carson     | Bedder       | cbedderh@sf_tuts.co.au    | 71 Clyde Gallagher
Place    | Leninskoye         | 2017-03-29 |
| Dana       | Avory        | davoryi@sf_tuts.com       | 2 Holy Cross Pass
| Wenlin             | 2017-05-11 |
| Ronny      | Talmadge     | rtalmadgej@sf_tuts.co.uk  | 588 Chinook Street
| Yawata             | 2017-06-02 |
+------------+--------------+---------------------------+--------------------
---------+--------------------+------------+
```

Insert Additional Rows of Data

In addition to loading data from staged files into a table, you can insert rows directly into a table using the INSERT DML command.

For example, to insert two additional rows into the table:

```
insert into emp_basic values
  ('Clementine','Adamou','cadamou@sf_tuts.com','10510 Sachs
Road','Klenak','2017-9-22') ,
  ('Marlowe','De Anesy','madamouc@sf_tuts.co.uk','36768 Northfield
Plaza','Fangshan','2017-1-26');
```

Query Rows Based on Email Address

Return a list of email addresses with United Kingdom domain names using the LIKE function:

```
select email from emp_basic where email like '%.uk';

+--------------------------+
| EMAIL                    |
|--------------------------|
| gbassfordo@sf_tuts.co.uk |
| rtalmadgej@sf_tuts.co.uk |
| madamouc@sf_tuts.co.uk   |
+--------------------------+
```

Query Rows Based on Start Date

Add 90 days to employee start dates using the DATEADD function to calculate when certain employee benefits might start. Filter the list by employees whose start date occurred earlier than January 1, 2017:

```
select first_name, last_name, dateadd('day',90,start_date) from emp_basic
where start_date <= '2017-01-01';

+------------+-----------+-------------------------------+
| FIRST_NAME | LAST_NAME | DATEADD('DAY',90,START_DATE)  |
|------------+-----------+-------------------------------|
```

```
| Granger    | Bassford  | 2017-03-30                    |
| Catherin   | Devereu   | 2017-03-17                    |
| Cesar      | Hovie     | 2017-03-21                    |
| Wallis     | Sizey     | 2017-03-30                    |
+------------+-----------+-------------------------------+
```

Summary and Clean Up

YEHHHH!!!!! You have successfully completed this SnowSQL experiment.

Please take a few minutes to review a short summary and the key points covered in the experiment. You might also want to consider cleaning up by dropping any objects you created in the experiment.

Experiment Summary and Key Points

In summary, data loading is performed in 2 steps:

Step 1

> Stage the files containing the data to be loaded. The files can be staged internally (in Snowflake) or in an external location.

Step 2

> Copy data from the staged files into a target table. A running, active warehouse is required for this step. Also, to perform this step, you must have an existing table into which the data from the files will be loaded.

Some key points to remember about loading CSV files:

- A CSV file consists of 1 or more records, with 1 or more fields in each record, and sometimes a header record.
- Records and fields in each file are separated by delimiters. The default delimiters are:

Records

newline characters

Fields

commas

In other words, Snowflake expects each record in a CSV file to be separated by newlines and the fields (i.e. individual values) in each record to be separated by commas. If ***different*** characters are used as record and field delimiters, you must ***explicitly*** specify this as part of the file format when loading.

- There is a ***direct*** correlation between the fields in the files and the columns in the table you will be loading, in terms of:
    - Number of fields (in the file) and columns (in the target table).
    - Positions of the fields and columns within their respective file/table.
    - Data types, such as string, number, or date, for fields and columns.

  If the numbers, positions, and data types don't all align, the records will not be loaded.

Note

Snowflake supports loading files in which the fields don't exactly align with the columns in the target table; however, this is a more advanced data loading path.

Experiment Clean Up (Optional)

Execute the following DROP <object> statements to return your system to its state before you began the experiment:

```
drop database if exists sf_tuts;

drop warehouse if exists sf_tuts_wh;
```

Tip

Before you drop the warehouse, consider whether you want to continue with your own testing or custom experiments, all of which would require a warehouse. You might be able to save some time and credits by reusing this warehouse for your own experimentation.

Exit the Connection

To exit a connection, use the `!exit` command for SnowSQL (or its alias, `!disconnect`).