Day 1 Exercises

List of all assignments for first week together. Please find all DDL and all other needed queries for the exercises in following file on GitHub: `Day1/day1_exercises.sql`

03 Data Ingestion workflow

Let's establish all needed objects and integrations needed for data ingestion into Snowflake. Please use your trial AWS account, where we are going to use S3 as our external stage. During this exercise we are going to create following objects:

- Storage integration to link Snowflake and AWS together in secure way
- Stage object as file location description
- File Format with format definition for our source file

Let's start with creation S3 bucket in your AWS account. You can use whatever name. My bucket has name `oreilly-trainings`.
Now we need to configure the secure access to cloud Storage. It requires multiple configuration steps on AWS side, like creation of IAM policy and role, defining the access control requirements, etc.
Snowflake provide step by step guide for this setup. It is available here:
https://docs.snowflake.com/en/user-guide/data-load-s3-config-storage-integration
Once we have configured the integration between AWS and Snowflake we can proceed with FILE FORMAT creation. File format should have following parameters:

- Csv file as an type
- Comma as an delimiter
- It should skip header
- And fields are enclosed by double quotes (")

Next, we need to create a stage object:

- It will use our storage integration
- It will use our file format
- It will use our s3 bucket

Finally we have all the objects in place so we can proceed with target table creation.
Please create a table called `GPT_TWEETS`. DDL script is available on Github in following file:
`Day1/day1_exercises.sql`
Once we have landing table create, let's upload the `chatgpt-tweets-data.csv` into our S3 bucket and we are ready to go with data import. You

can download the file from the GitHub repository. It is under following path: `day1/source_files/chatgpt-tweets-data.csv.zip`
Once you download the file, please uncompress it before uploading into S3. Let's write an copy command to import the data. As source file contains some errors we need to use `ON_ERROR = CONTINUE` option.
As another example let's try to do some transformations as part of COPY command. We are going to select only a few columns from source file and also do simple casting for timestamp values. Please create a new landing table called `GPT_TWEETS_PROCCESSED`.
Now we can write a new COPY command command which will be selecting only following source columns:
`ID, DATE, USERNAME, PROCESSED_TWEET`

- `Let's still use ON_ERROR = CONTINUE option`
- Let's do a casting to TIMESTAMP for DATE value

As a last part of this exercise, let's try to explore a load metadata. Let's start with querying the metadata directly from the staged file along with some columns. We can get following values during loading data into the table:

- `METADATA$FILENAME,`
- `METADATA$FILE_ROW_NUMBER,`
- `METADATA$FILE_CONTENT_KEY,`
- `METADATA$FILE_LAST_MODIFIED,`
- `METADATA$START_SCAN_TIME`

Let's write an select query which will select those attributes along with first 3 columns from the staged file.

As an next example let's query the load metadata for given landing table from `COPY_HISTORY` table function in `INFORMATION_SCHEMA`. Firstly let's try it for the `GPT_TWEETS_PROCCESSED` table with START_DATE in last hour.

As a last example let's try the same thing but this time for `GPT_TWEETS` table. We should get also an info about error rows.

04 Semi structured data processing

We are going to practice working with semi structured data in this exercise along with using functions INFER_SCHEMA, GENERATE_COLUMN_DESCRIPTION and CREATE TABLE USING TEMPLATE which significantly help to speed up the data ingestion from Parquet or AVRO files.
1. Please download the parquet data from GITHUB. There are two files:

- `Day1/gpt_tweets.parwuet_0_0_0.snappy.parquet`
- `Day1/gpt_tweets.parwuet_0_0_1.snappy.parquet`

2. Upload the parquet files into your S3 bucket
3. We need to create a landing table for our data. Let's call it GPT_TWEETS_PARQUET
    - You can find all the DDLs and DMLs for this exercise in following file on GitHub: `day1/day1_exercises.sql`
4. For the INFER_SCHEMA function we will need a named file format for the parquet files. Please create `my_parquet_format`.
5. Now it is time to manually ingest data from Parquet format. We need to write a COPY command and specify the data type for each column. Please notice that all the columns are under pseudo column `$1` in the file. You need to use following syntax for each column: `$1:column_name::data_type`
6. Now let's try to automate it and avoid writing the list of attributes manually, same like creating the landing table manually. First, let's try to use `GENERATE_COLUMN_DESCRIPTION` function which will generate a list of columns together with the right data type. It will be based on source parquet file which is used as parameter.
7. Now we can copy the output and paste it into `CREATE` table statement or we can automate it further and create also table automatically by using `CREATE TABLE USING TEMPLATE` function. Let's create table called `GPT_TWEETS_PARQUET_TEMPLATE` by using this TEMPLATE feature.
8. Now we have another landing table in place so let's ingest the data there again. This time we are not going to manually write the body of the `COPY` statement and define each column together with the attribute but we are going to use `INFER_SCHEMA` function which will generate it for us. Then we can copy the output and paste into the `COPY` command.

05 Into into SnowSQL - CLI interface

Let's start with installation of the CLI client. Please go to: [SnowSQL - Snowflake Developers]

This page also contains link into documentation with detailed guide for installation or configuration of the CLI client.
Based on your OS and preferred way of installation download the needed files.

06 Views

This part is dedicated to practising working with views and their differences. We will try to create standard view, secure view and materializied view. Thanks to this exercise we will be able to describe their limitations. Please follow the code available in `day1/day1_exercises.sql`

07 Continuous data pipelines

We are going to practise the continuous data ingestion into Snowflake in this exercise. We will use snowpipe feature for this. It again requires configuration on the Cloud provider which is similar to settings which we have done for the batch data ingestion. There are required the bucket access permission and IAM role in order to access Snowflake. Now we also need some kind of notification to be sent to snowpipe when a new file will be landed in cloud storage. Let's deep dive into the setup:

1. Secure access to cloud storage configuration

This we already have in place. There are needed the following S3 bucket permissions

- s3:GetBucketLocation
- s3:GetObject
- s3:GetObjectVersion
- s3:ListBucket

We have bundled those into IAM policy and assign that policy to IAM role. Let's check in AWS console.

Once we have the IAM policy and IAM role we need to have storage integration on Snowflake side referencing this IAM role. This we have already done as well. Let's check our storage integration:

```
desc integration my_s3_access;
```

There we have the values `STORAGE_AWS_IAM_USER_ARN` and `STORAGE_AWS_EXTERNAL_ID` which we need for defining the trust relationship between Snowflake and AWS -> this we have done already.

2. Now let's move into defining the S3 Event Notification configuration

We will use the Amazon SQS for that. On Snowflake side we also need a stage. Let's reuse what we have defined for the batch processing - `my_s3_stage`.

We also need a landing table where we are going to ingest the data through snowpipe. You can find DDL script for a table in the file with all other queries for this exercise: `07_snowpipe.sql`

Now we can create a snowpipe which will be referencing our stage and copying data into our landing table:

```
create or replace pipe mypipe
    auto_ingest=true as
        copy into
data_engineering.public.snowpipe_landing
```

```
        from @data_engineering.public.my_s3_stage/
snowpipe/
        file_format = (type = 'CSV'
                       SKIP_HEADER = 1
                       FIELD_OPTIONALLY_ENCLOSED_BY='"');
```

Let's check the status of the pipe. If it is running. We can do that with following command:

```
select system$pipe_status('mypipe');
```

You can notice additional subdirectory after the stage name. Let's place the files for the snowpipe into separated directory to keep them separated from others.

Now, when we have the snowpipe object defined we can proceed with event notification configuration. We have to find out the SQS queue which Snowflake has assigned to our pipe.

Let's run the `show pipes` command and make a note of the ARN of the SQS queue. You can find it in `notification_channel`.

Now we need to configure that channel on S3 side:

- Let's go into the bucket properties, find an event notifications section and create event notification.
- We select all object create events in OBJECT creation section
- Scroll down to destination section where we select SQS queue and specify the SQS queue ARN which we got from show pipes command.

And that's it. We should have all the needed configuration in place. Let's try some data ingestion. Please download the sample data from GitHub. You can find them here:

- `Day1/source_files/snowpipe/MOCK_DATA.csv`
- `Day1/source_files/snowpipe/MOCK_DATA-2.csv`
- `Day1/source_files/snowpipe/MOCK_DATA-3.csv`

Let's upload the first file `MOCK_DATA.csv` into our S3 bucket, under the snowpipe subdirectory which we have defined in the Snowpipe definition.

We need to wait a little bit and then we can try to query the landing table. Soon there should be ingested the data. There might be up to 1 min latency so please keep trying to query the table for a while.

08 Streams and Task

Let’s follow-up on previous session dedicated to snowpipes. Now we try to automatically process the data which have been ingested by snowpipe. We need two additional database objects for it - streams and task.

1. We will start by creating a stream on top of the landing table.
2. We can check if stream already have some data by querying one of the system functions
3. Let’s upload the second file MOCK_DATA-2.csv into S3 bucket. This action should trigger following scenario:
    - SQS event notification will be sent
    - Snowpipe will fetch the file and ingest the data
    - Our newly created stream should have some data - exactly those from this file.
4. When done, we can check the count in landing table if it has now 2000 records.
5. It means our newly created stream has data now as well - check it by querying the same function again
6. Let’s query the stream itself - same like table to check the stream content. We can find the stream metadata at the end of the column list.

That’s it for streams. Now we have an object which holds only freshly loaded data we need to have an ability to process such data somehow (apply some transformations or aggregrations). And for that we need a task.

7. Let’s say we want to calculate how many entries for each gender are loaded for each ingested file. We will create a target table called `gender_summary` to save the result.
8. Next we need to grant a privilege to execute a task to our sysadmin role. Let’s run the grant command under the account admin role. EXECUTE TASK is account level privilege which is by default granted only to account admin
9. Now we can create a task `t_gender_cnt` which will be running each minute and it will have a condition to start only when our `str_landing` stream has some data. Task will insert aggregated data into the `gender_summary` table together with actual timestamp so we know when to load was done.
10. Let’s check if the state of the task with show tasks command. It is suspended. We need to resume it.
11. Once task is resumed it should be triggered within the minute because we still have data in our stream. Let’s wait for a while before checking the stream status and content of the `gender_summary` table.
12. We can also check the task history by querying the table function `task_history()` to see the loads. We can see many skipped instances - the condition was not fulfilled + there should be also one entry with SUCCEEDED status.

13. Let’s upload the latest file into external stage to test out the complete integration. File will be ingested by snowpipe, data will be available in stream which triggers our task which will process those data.
14. We can check the gender_summary table and task_history() table function again to see that also the latest file has been loaded and processed.
15. As a last step we can again suspend our task not to be started every minute.