Exercises

List of all assignments together.

03 Create basic database objects

1. Use role `SYSADMIN`
2. Create a database called `CITIBIKE` with **SQL command**
3. Create schema called `WORK`, this time use **Snowsight UI**
4. Create table `TRIPS` in `WORK` schema. Please use **SQL** or **UI** (do not forget to setup the context for your worksheet)

Here you can find columns and data types for the `TRIPS` table:

```
tripduration integer,
 starttime timestamp,
 stoptime timestamp,
 start_station_id integer,
 start_station_name string,
 start_station_latitude float,
 start_station_longitude float,
 end_station_id integer,
 end_station_name string,
 end_station_latitude float,
 end_station_longitude float,
 bikeid integer,
 membership_type string,
 usertype string,
 birth_year integer,
 gender integer
```

05 Create virtual warehouse

- Create your first virtual warehouse called `COMPUTE_WH`, with following parameters:
    - Extra small size
    - Enabled auto resume
    - Enabled auto suspend
    - Suspend after 1 minute
    - It will be suspended after creation

- Create a multi cluster warehouse called MULTI_COMPUTE_WH with following parameters:
    - Small size
    - Enabled auto resume
    - Enable atuo suspend
    - Suspend after 3 minutes
    - It will be suspended after creation
    - Min number of clusters: 1
    - Max number of clusters: 3
    - Economy scaling policy

- Use SQL for virtual warehouse creation
- Use `SYSADMIN` role for object creation
- Use Documentation for the right SQL syntax

06 Data Loading principles

1. In `PUBLIC` schema, Create a file format called `FF_CSV` with following parameters

- Format type: csv
- Column separator: comma
- No lines for skipping header
- Field optionally enclosed: Double Quote
- In case the column value has empty string ('') replace it by NULL

2. In `PUBLIC` schema, create a stage called `S_CITIBIKE_TRIPS`
   - URL: s3://snowflake-workshop-lab/citibike-trips-csv/
   - FILE_FORMAT - our file format created in previous step

3. Check the content of the stage with `LIST` command. Send me how many files you can see in stage.

Please note that we do not use any authentication because it is a public bucket. In real scenario we would have to use either access keys or better to create a STORAGE_INTEGRATION object.

07 COPY command

1. Using Copy command for loading the CSV files:

- Use `FILE_FORMAT` and `STAGE` objects created earlier
- First, scale up your `COMPUTE_WH` warehouse to use Large size
- Load data into table `TRIPS`
- In case of failure during the import skip the file
- Check the `COPY` command results - how many files were processed? Send the number into the chat
- Scale down the `COMPUTE_WH` to Extra-small size
- Query the table to check the table content
- Find out number of records loaded into `TRIPS` table and send it into the chat

Use Snowflake documentation for the right COPY command syntax

2. Working with PARQUET data

Now we are going to practise usage of Parquet data together with already learnt concepts for file formats, stages and copy command. We will be exporting Data into Parquet files which will be stored in our user stage this time.
Then we will import those data again into the table to see the difference between CSV (flat file) and PARQUET (semi-structured data format)

First we have to offload the data into the stage. Let's use the user stage this time and put the files under /`parquet` subdirectory. Please use `COPY` command option for header to include column headers into the export: `HEADER = true`

Make a note how many records have been exported.

Let's create a copy of our `TRIPS` table where we are going to load the data from PARQUET file. Here is the script:

```
/*CREATE copy of TRIPS table for loading from parquet */
create table trips_parquet like trips;

create or replace table trips_parquet
(tripduration integer,
  starttime timestamp,
  stoptime timestamp,
  start_station_id integer,
  start_station_name string,
  start_station_latitude float,
  start_station_longitude float,
  end_station_id integer,
  end_station_name string,
  end_station_latitude float,
  end_station_longitude float,
  bikeid integer,
  membership_type string,
  usertype string,
  birth_year integer,
  gender integer);
```

Now let's upload the data from Parquet files placed in user stage into the table `TRIPS_PARQUET`

When copying into the table do not forget to specify list of table columns similarly like you would do in normal `INSERT` query.

Use column names in uppercase as they are stored in uppercase in parquet files and referencing the elements in semi-structured files is case-sensitive!

COPY command will have following structure - pseudo code

```
COPY into trips_parquet (<<table columns>>)
FROM
(
    SELECT
        $1.column_name::column_data_type,
        ...
        ...
    FROM
       user stage/our subdirectory
```

```
  )
FILE_FORMAT definition
```

3. Using INFER_SCHEMA function

Let's suppose we are about to import a new parquet file where we do not know its structure.
Use `INFER_SCHEMA` function to find out how the file’s schema looks like.

Use the parquet files exported in previous steps. They are available in your user stage.

In order to use `INFER_SCHEMA` function, you need to have a file format with defined parquet type. This one will be then used in `INFER_SCHEMA` function call.

Please create following file format:

```
create file format my_parquet_format
type = parquet;
```

Pass this file format into the INFER_SCHEMA function together with the parquet files in our user stage.
Run the function and inspect the function output. Send into the chat number of returned rows.
You can see that functions provide the list of columns together with their data types and expressions which you have to use in `COPY` command definition.