# Inserting data into Snowflake using SQLAlchemy

*SQLAlchemy* can be used to load the data in the Snowflake. I tried it from my laptop to Snowflake in AWS. It took *3min 13 seconds to load* *298,424 rows* and *29 seconds to load 42,632 rows* from my laptop to Snowflake in AWS. The libraries that we need for this purpose are –

* *pip install snowflake-connector-python*
* *pip install -r https://raw.githubusercontent.com/snowflakedb/snowflake-connector-python/v2.4.3/tested\_requirements/requirements\_36.reqs*
* *pip install sqlalchemy*
* *pip install snowflake-sqlalchemy*

You may face issue while executing the first *pip*. If you face any issue, please use the following –

* *pip install -r https://raw.githubusercontent.com/snowflakedb/snowflake-connector-python/v2.4.3/tested\_requirements/requirements\_36.reqs --user*

**For Snowflake setup** - This entry-level lab introduces you to the user interface and basic capabilities of Snowflake, and is designed specifically for use with the Snowflake, free 30-day trial at https://trial.snowflake.com. When done with the lab you should be ready to load your own data into Snowflake and learn its more advanced capabilities. The Snowflake edition (Standard, Enterprise, e.g.), cloud provider (AWS, Azure, e.g.), and Region (US East, EU, e.g.) do \*not\* matter for this lab. But I have chosen AWS and selected the region which is physically closest to me. And select the Enterprise edition. After registering, you will receive an email with an activation link and your Snowflake account URL. Bookmark this URL for easy, future access. After activation, you will create a user name and password. Please refer to the <https://s3.amazonaws.com/snowflake-workshop-lab/Snowflake_free_trial_LabGuide.pdf> for step-by-step setup instruction.

Step 1 – Install the pre-requisites for python and get the Snowflake Lab working.

Step 2 – Gather all the connection parameters for the Snowflake

*user = 'xxxxx',*

*password = 'xxxxxxx',*

*database = 'DEMO\_DB',*

*schema = 'public',*

*warehouse = 'compute\_wh',*

*role='SYSADMIN'*

Step 3 – Configure the engine and connection

from snowflake.sqlalchemy import URL

from sqlalchemy import create\_engine, exc

def snow\_connector(accoutName, userName, userPassword, databaseName, schemaName, warehouseName, roleName):

    try:

        engine = create\_engine(URL(

            account = accoutName, #'<account\_name>.<region\_id>.<cloud>' (e.g. 'xy789.east-us-2.aws')

            user = userName,

            password = userPassword,

            database = databaseName,

            schema = schemaName,

            warehouse = warehouseName,

            role= roleName

        ))

        connection = engine.connect()

        return connection

    except exc.SQLAlchemyError as e:

        print("Engine Creation Module")

        print(e)

        sys.exit(1)

    except exc.DBAPIError  as e:

        print("Engine Creation Module")

        print(e)

        sys.exit(1)

In the account name, we have to mention the Cloud provider name, like – aws or azure. For me it was *hg69887.us-east-2.aws*

The actual URL looks like the following

'snowflake://<user\_login\_name>:<password>@<account\_name>/<database\_name>/<schema\_name>?warehouse=<warehouse\_name>&role=<role\_name>'

Step 4 – Object name case handling - Snowflake stores all case-insensitive object names in uppercase text. In contrast, SQLAlchemy considers all lowercase object names to be case-insensitive. Snowflake SQLAlchemy converts the object name case during schema-level communication (i.e. during table and index reflection). If you use uppercase object names, SQLAlchemy assumes they are case-sensitive and encloses the names with quotes. This behavior will cause mismatches against data dictionary data received from Snowflake, so unless identifier names have been truly created as case sensitive using quotes (e.g. " Python\_Connect"), all lowercase names should be used on the SQLAlchemy side. Check the column names in the datafile. It should follow the allowed characters in Snowflake.

Step 5 – Put the data file into a Dataframe. We need the column name in the data file. Those will ultimately become the table columns, if you do a table replace.

df = pd.read\_csv(fileName,header=0)

Step 6 – Insert the data in the table using to\_sql

df.to\_sql( name= tableName , con = conn, if\_exists='replace', index = False, chunksize=16000)

Few parameters that we need to take care of -

1. *if\_exists - {‘fail’, ‘replace’, ‘append’}, default ‘fail’*
   * *fail: Raise a ValueError.*
   * *replace: Drop the table before inserting new values.*
   * *append: Insert new values to the existing table.*
2. *Index* – Keep it always “False”
3. Chunksize – Currently the snowflake connector does not allow to load more than 16,384 rows from a single connection. So for that reason it has been set to 16,000. If the number of records are more than 16,000, Python will start multiple instances. The log for those instances will look like the following

INFO:snowflake.connector.cursor:query: [INSERT INTO python\_connect (zip, type, decommissioned, primary\_city, acceptable\_...]

INFO:snowflake.connector.cursor:query execution done

INFO:snowflake.connector.cursor:query: [INSERT INTO python\_connect (zip, type, decommissioned, primary\_city, acceptable\_...]

INFO:snowflake.connector.cursor:query execution done

INFO:snowflake.connector.cursor:query: [INSERT INTO python\_connect (zip, type, decommissioned, primary\_city, acceptable\_...]

INFO:snowflake.connector.cursor:query execution done

INFO:snowflake.connector.cursor:query: [INSERT INTO python\_connect (zip, type, decommissioned, primary\_city, acceptable\_...]

INFO:snowflake.connector.cursor:query execution done

INFO:snowflake.connector.cursor:query: [INSERT INTO python\_connect (zip, type, decommissioned, primary\_city, acceptable\_...]

INFO:snowflake.connector.cursor:query execution done

INFO:snowflake.connector.cursor:query: [INSERT INTO python\_connect (zip, type, decommissioned, primary\_city, acceptable\_...]

INFO:snowflake.connector.cursor:query execution done

INFO:snowflake.connector.cursor:query: [COMMIT]

Once the command is fired with if\_exists = ‘replace’, the process will go through a number of steps

INFO:snowflake.connector.cursor:query: [DESC TABLE /\* sqlalchemy:\_has\_object \*/ python\_connect]

INFO:snowflake.connector.cursor:query: [SHOW /\* sqlalchemy:get\_table\_names \*/ TABLES IN public]

INFO:snowflake.connector.cursor:query: [SHOW /\* sqlalchemy:\_get\_schema\_primary\_keys \*/PRIMARY KEYS IN SCHEMA demo\_db.pub...]

INFO:snowflake.connector.cursor:query: [SELECT /\* sqlalchemy:\_get\_schema\_columns \*/ ic.table\_name, ic.column\_name, ic.da...]

INFO:snowflake.connector.cursor:query: [SHOW /\* sqlalchemy:\_get\_schema\_foreign\_keys \*/ IMPORTED KEYS IN SCHEMA demo\_db.p...]

INFO:snowflake.connector.cursor:query: [SHOW /\* sqlalchemy:\_get\_table\_comment \*/ TABLES LIKE 'python\_connect']

INFO:snowflake.connector.cursor:query: [DROP TABLE python\_connect]

INFO:snowflake.connector.cursor:query: [CREATE TABLE python\_connect ( zip BIGINT, type TEXT, decommissioned BIGINT, prim...]

INFO:snowflake.connector.cursor:query: [INSERT INTO python\_connect (zip, type, decommissioned, primary\_city, acceptable\_...]

INFO:snowflake.connector.cursor:query: [INSERT INTO python\_connect (zip, type, decommissioned, primary\_city, acceptable\_...]

INFO:snowflake.connector.cursor:query: [INSERT INTO python\_connect (zip, type, decommissioned, primary\_city, acceptable\_...]

INFO:snowflake.connector.cursor:query: [COMMIT]

INFO:snowflake.connector.cursor:query execution done

Step 7 – Close the connection and engine

    finally:

        conn.close()

        engine.dispose()

If we don’t want to use SAQLAlechemy then we can insert the data as following

from snowflake.connector.pandas\_tools import pd\_writer

df.to\_sql(tableName, engine, index=False, method=pd\_writer)