Getting Started with Python

Overview

You can connect to Snowflake in many languages. If your language of choice is Python, you'll want to begin here to connect to Snowflake. We'll walk you through getting the Python Connector up and running, and then explore the basic operations you can do with it. You'll find the Python Connector to be quite robust, as it even supports integration with Pandas DataFrames.

Prerequisites

- Familiarity with Python

What You'll Learn

- how to set up the Python connector
- how to test your installation
- how to connect to Snowflake
- how to set session parameters
- how to create a warehouse
- how to create a database
- how to create a schema
- how to create a table
- how to insert data
- how to query data

What You'll Need

- A Snowflake Account
- A Text Editor

What You'll Build

- Examples of using the Snowflake Python Connector

Set up the Python Connector

For this guide, we'll be using Python 3. Let's check what version of Python you have on your system. Do this by opening a terminal and entering the following command:

```
python --version
```

If it outputs 3.5 or higher, you're good to go! If not, you'll need to install a newer version of Python. You can grab the latest release on the Python website.

Once you've got a recent version of Python, you can install the Snowflake Connector for Python. You'll do this via the Python package installer `pip` and by running the following command:

```
pip install --upgrade snowflake-connector-python
```

If you're on a Linux distribution, you'll also need to install a few packages from your distribution's repository. Specifically, you'll need the equivalent of:

- libm-devel
- openssl-devel

Once you have both Python and the Snowflake Connector installed, you're ready to go! Let's make sure that's the case.

Test Your Installation

Before we delve into *using* the Snowflake connector, let's ensure it's installed correctly. We can do this with the following script.

```
#!/usr/bin/env python
import snowflake.connector

# Gets the version
ctx = snowflake.connector.connect(
    user='<your_user_name>',
    password='<your_password>',
    account='<your_account_name>'
    )
cs = ctx.cursor()
try:
    cs.execute("SELECT current_version()")
    one_row = cs.fetchone()
    print(one_row[0])
finally:
    cs.close()
ctx.close()
```

Open up a text editor and copy that script in and save it as `validate.py`. You'll need to input your own user name, password, and account in the corresponding fields.

Open up a terminal in the location where the file is saved, and run the following command:

```
python validate.py
```

If everything is good to go, you'll see the installed Snowflake version. Otherwise, you'll get errors specific to your situation.

Connect to Snowflake

It's time to use the Snowflake Connector for Python. Open up your Python environment. The first thing you'll need to do is to import the Snowflake Connector module. Do this before using any Snowflake related commands.

```
import snowflake.connector
```

You may want to consider reading in your login information from external sources, such as environment variables. This will add some security to your scripts, and save you time in the long run. In this example, we're using `os.getenv` to pull the environment variable `SNOWSQL_PWD` for our variable `PASSWORD.`

```
PASSWORD = os.getenv('SNOWSQL_PWD')
```

You'll use those variables within the Python Connector like so:

```
conn = snowflake.connector.connect(
    user=USER,
```

```
    password=PASSWORD,
    account=ACCOUNT
    )
```

Nice! Now let's delve into the specifics.

Set Session Parameters

You can set session parameters to tweak your session and have it set up just the way you want it. There are two ways to accomplish this. First, you can set them when you initially connect, like so:

```
conn = snowflake.connector.connect(
    user='XXXX',
    password='XXXX',
    account='XXXX',
    session_parameters={
        'QUERY_TAG': 'EndOfMonthFinancials',
    }
)
```

Alternatively, you can set them after you connect by executing the SQL statement `ALTER SESSION SET`:

```
conn.cursor().execute("ALTER SESSION SET QUERY_TAG = 'EndOfMonthFinancials'")
```

You may have taken advantage of Snowflake's ability to use robust security methods. If so, you'll want to take extra steps to configure the Snowflake Python Connector.

- [Single Sign-on (SSO)]
- [Key Pair Authentication]
  - [Key Rotation]
- [OAuth]

At this point, you can begin manipulating objects within Snowflake.

Create a Warehouse

If you're familiar with the SQL commands to interact with Snowflake, then the commands within the Python connector will be familiar as well. Let's break down what a command looks like.

Remember that `conn` is the object that connects you to your Snowflake account. You can use that to execute SQL commands with the following format.

```
conn.cursor().execute("YOUR SQL COMMAND")
```

We first want to create a [virtual warehouse]. Virtual warehouses contain the servers that are required for you to perform queries and DML operations with Snowflake. Creating one can be done by incorporating the `CREATE WAREHOUSE` SQL command.

```
conn.cursor().execute("CREATE WAREHOUSE IF NOT EXISTS tiny_warehouse_mg")
```

The `CREATE WAREHOUSE` command creates a warehouse, as expected, but also implicitly sets that warehouse as the active warehouse for your session. If you've already created a warehouse, you can explicitly set it as the active warehouse with the `USE WAREHOUSE` command.

```
conn.cursor().execute("USE WAREHOUSE tiny_warehouse_mg")
```

Now that you have a warehouse, let's talk about databases.

Create a Database

The next step is to create a database. Databases contain your schemas, which contain your database objects. We can create one in a similar manner to creating a warehouse, but this time with the `CREATE DATABASE` command.

```
conn.cursor().execute("CREATE DATABASE IF NOT EXISTS testdb")
```

Again, creating a database also sets that database as the active one for the current session. If you need to set an already created database as the active database for the session, use the `USE DATABASE` command.

```
conn.cursor().execute("USE DATABASE testdb")
```

After that, you need to create a schema.

Create a Schema

Schemas are the grouping of your database objects. These include your tables, the data within them, and views. Schemas are found within databases. You can create a schema with the `CREATE SCHEMA` command.

```
conn.cursor().execute("CREATE SCHEMA IF NOT EXISTS testschema")
```

Since `CREATE SCHEMA` also sets it as the active schema for your session, you don't have to explicitly call `USE SCHEMA` as well. Only do so if you would like to use an already created schema.

```
conn.cursor().execute("USE SCHEMA testschema")
```

By default, the schema will be used in the current database. You can use it in another database by specifying that other database.

```
conn.cursor().execute("USE SCHEMA otherdb.testschema")
```

With a schema in use, it's time to move on to dealing with tables.

Create a Table

Alright, with a warehouse, database, and schema created you have everything you need to begin manipulating data in tables. First, you'll need to create the table. That's done with the `CREATE TABLE` command.

```
conn.cursor().execute(
    "CREATE OR REPLACE TABLE "
    "test_table(col1 integer, col2 string)")
```

Within `conn.cursor().execute` this example creates a table named `test_table` with two columns, one named `col1` which will contain integers and `col2` which will contain strings.

Insert Data

With the table `test_table` created, you can add data to it. You can do so with the command `INSERT`.

```
conn.cursor().execute(
    "INSERT INTO test_table(col1, col2) "
    "VALUES(123, 'test string1'),(456, 'test string2')")
```

This command inserts data into `test_table` row by row. The value found in the first column and first row is "123" and so on.

If you don't want to manually insert data row by row, however, you can load data instead. This is done with a combination of the `PUT` and `COPY INTO` commands.

```
conn.cursor().execute("PUT file:///tmp/data/file* @%test_table")
conn.cursor().execute("COPY INTO test_table")
```

The `PUT` command here is staging the file, and the `COPY INTO` command is copying that data from that file into the specified table. You can also use the `COPY INTO` command to copy data from an external location.

Query Data

Of course, you'll want to query your data at some point. It's easy to do that within the Python Connector too. To view values from the table, you can do so easily with the `print` command.

```
col1, col2 = conn.cursor().execute("SELECT col1, col2 FROM test_table").fetchone()
print('{0}, {1}'.format(col1, col2))
```

This code snippet is using the SQL command `SELECT col1, col2 FROM test_table` to select specific columns, and then is printing the first values in each, or in other words, the first row.

If you'd like to print entire columns, you can do so in a similar manner.

```
for (col1, col2) in conn.cursor().execute("SELECT col1, col2 FROM test_table"):
    print('{0}, {1}'.format(col1, col2))
```

To efficiently use your resources, you'll want to remember to explicitly close your connection to Snowflake after performing queries.

```
connection.close()
```

Look at that! You have now implemented all the steps needed to manipulate data within Snowflake.

Next Steps: Advanced Manipulation with Python

By now you have a grasp on the basics of using the Python Connector. As you may have noticed, it leans heavily on already-established Snowflake SQL commands. Of course, this isn't all you can do with the Connector. Here are some potential next steps that may be of interest to you depending on your use case.

- [Bind Data]
- [Use Pandas Dataframes]
- [Snowflake SQLAlchemy Toolkit]

This is just the surface of what you can do with Snowflake. To learn more, see the [Snowflake documentation].