Role: Data Engineer

Accessing Databases using Python Script

Using databases is an important and useful method of sharing information. To preserve repeated storage of the same files containing the required data, it is a good practice to save the said data on a database on a server and access the required subset of information using database management systems.

In this lab, you'll learn how to create a database, load data from a CSV file as a table, and then run queries on the data using Python.

Objectives

In this lab you'll learn how to:

1. Create a database using Python
2. Load the data from a CSV file as a table to the database
3. Run basic "queries" on the database to access the information

Scenario

Consider a dataset of employee records that is available with an HR team in a CSV file. **As a Data Engineer**, you are required to create the database called STAFF and load the contents of the CSV file as a table called INSTRUCTORS. The headers of the available data are :

Header	Description
ID	Employee ID
FNAME	First Name
LNAME	Last Name
CITY	City of residence
CCODE	Country code (2 letters)

Setting Up

Usually, the database for storing data would be created on a server to which the other team members would have access. For the purpose of this lab, we are going to create the database on a dummy server using SQLite3 library.

Note: SQLite3 is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed

SQL database engine in the world. SQLite3 comes bundled with Python and does not require installation.

Initial steps

For this lab, you will need a Python file in the `project` folder. You can name it `db_code.py`.
Run the following command in the terminal. Make sure the current directory in the terminal window is `/home/project/`

```
wget
https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMSkillsNetwork-PY0221EN-Coursera/labs/v2/INSTRUCTOR.csv
```

Further, to read the CSV and interact with the database, you'll need the `pandas` library. This library will first have to be installed in the Cloud IDE framework. For this, run the below mentioned statement in a terminal window.

```
python3.11 -m pip install pandas
```

Python Scripting: Database initiation

Let us first create a database using Python.

Open `db_code.py` and import the `sqlite3` library using the below mentioned command.

```
import sqlite3
```

Import the `pandas` library in `db_code.py` using the following code.

```
import pandas as pd
```

Now, you can use SQLite3 to create and connect your process to a new database `STAFF` using the following statements.

```
conn = sqlite3.connect('STAFF.db')
```

Python Scripting: Create and Load the table

To create a table in the database, you first need to have the attributes of the required table. Attributes are columns of the table. Along with their names, the knowledge of their data types are also required. The attributes for the required tables in this lab were shared in the Lab Scenario.

Add the following statements to `db_code.py` to feed the required table name and attribute details for the table.

```
table_name = 'INSTRUCTOR'
attribute_list = ['ID', 'FNAME', 'LNAME', 'CITY', 'CCODE']
```

Note: This information can be updated for the case of any other kind of table.

Save the file using `Ctrl+S`.

Reading the CSV file

Now, to read the CSV using Pandas, you use the `read_csv()` function. Since this CSV does not contain headers, you can use the keys of the `attribute_dict` dictionary as a list to assign headers to the data. For this, add the commands below to `db_code.py`.

```
file_path = '/home/project/INSTRUCTOR.csv'
df = pd.read_csv(file_path, names = attribute_list)
```

Loading the data to a table

The `pandas` library provides easy loading of its dataframes directly to the database. For this, you may use the `to_sql()` method of the `dataframe` object.

However, while you load the data for creating the table, you need to be careful if a table with the same name already exists in the database. If so, and it isn't required anymore, the tables should be replaced with the one you are loading here. You may also need to append some information to an existing table. For this purpose, `to_sql()` function uses the argument `if_exists`. The possible usage of `if_exists` is tabulated below.

Argument usage	**Description**
`if_exists = 'fail'`	Default. The command doesn't work if a table with the same name exists in the database.
`if_exists = 'replace'`	The command replaces the existing table in the database with the same name.
`if_exists = 'append'`	The command appends the new data to the existing table with the same name.

As you need to create a fresh table upon execution, add the following commands to the code. The print command is optional, but helps identify the completion of the steps of code until this point.

```
df.to_sql(table_name, conn, if_exists = 'replace', index =False)
print('Table is ready')
```

Python Scripting: Running basic queries on data

Now that the data is uploaded to the table in the database, anyone with access to the database can retrieve this data by executing SQL queries.

Some basic SQL queries to test this data are `SELECT` queries for viewing data, and `COUNT` query to count the number of entries.

SQL queries can be executed on the data using the `read_sql` function in `pandas`.

Now, run the following tasks for data retrieval on the created database.

1. Viewing all the data in the table.

Add the following lines of code to `db_code.py`

```
query_statement = f"SELECT * FROM {table_name}"
query_output = pd.read_sql(query_statement, conn)
print(query_statement)
print(query_output)
```

2. Viewing only FNAME column of data.

Add the following lines of code to `db_code.py`

```
query_statement = f"SELECT FNAME FROM {table_name}"
query_output = pd.read_sql(query_statement, conn)
print(query_statement)
print(query_output)
```

3. Viewing the total number of entries in the table.

Add the following lines of code to `db_code.py`

```
query_statement = f"SELECT COUNT(*) FROM {table_name}"
query_output = pd.read_sql(query_statement, conn)
print(query_statement)
print(query_output)
```

Now try appending some data to the table. Consider the following.

a. Assume the `ID` is `100`.

b. Assume the first name, `FNAME`, is `John`.

c. Assume the last name as `LNAME`, `Doe`.

d. Assume the city of residence, `CITY` is `Paris`.

e. Assume the country code, `CCODE` is `FR`.

Use the following statements to create the dataframe of the new data.

```
data_dict = {'ID' : [100],
        'FNAME' : ['John'],
        'LNAME' : ['Doe'],
        'CITY' : ['Paris'],
        'CCODE' : ['FR']}
data_append = pd.DataFrame(data_dict)
```

Now use the following statement to append the data to the `INSTRUCTOR` table.

```
data_append.to_sql(table_name, conn, if_exists = 'append', index =False)
print('Data appended successfully')
```

Now, repeat the `COUNT` query. You will observe an increase by 1 in the output of the first `COUNT` query and the second one.

Before proceeding with the final execution, you need to add the command to close the connection to the database after all the queries are executed.

Add the following line at the end of `db_code.py` to close the connection to the database.

```
conn.close()
```

Code Execution

Execute `db_code.py` from the terminal window using the following command.

```
python3.11 db_code.py
```