PP Lab 10

A.Y.: 2022-2023

Lab 10: Python and SQLite Database connectivity

==========

Name: Srihari Thyagarajan

Program: B Tech Artificial Intelligence

Div & Roll number: B3, I066

Date: 17/10/2022

Aim: EXPT 10: SQLite Database

1. Write a Python program to create a SQLite database and connect with the database and print the version of the SQLite database.
2. Write a Python program to create a table and insert some records in that table. Finally selects all rows from the table and display the records.
3. Write a Python program to insert values to a table from user input.
4. Write a Python program to delete a specific row from a given SQLite table.

Reference - https://www.w3resource.com/python-exercises/sqlite/index.php

In [1]:

```
import sqlite3
```

Create a new database

- The connect() function opens a connection to an SQLite database.
- It returns a Connection object that represents the database.
- By using the Connection object, you can perform various database operations.

In [26]:

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

print("Opened database successfully")
```

```
Opened database successfully
```

In [27]:

```
cursor = conn.cursor()
```

Create a Table

To create a new table in SQLite, you use CREATE TABLE statement using the following syntax:

```
CREATE TABLE table_name (
    column_1 data_type PRIMARY KEY,
    column_2 data_type NOT NULL,
    column_3 data_type DEFAULT 0,
    table_constraints
)
```

- Specify the name of the table that you want to create after the CREATE TABLE keywords.
- Use IF NOT EXISTS option to create a new table if it does not exist. Attempting to create a table that already exists without using the IF NOT EXISTS option will result in an error.
- Specify the column list of the table. Each column has a name, data type, and the column constraint. SQLite supports PRIMARY KEY, UNIQUE, NOT NULL, and CHECK column constraints.
- Specify the table constraints such as PRIMARY KEY, FOREIGN KEY, UNIQUE, and CHECK constraints.
- The primary key of a table is a column or a group of columns that uniquely identify each row in the table.

In [25]:

```
query = '''
CREATE TABLE COMPANY
(
ID INT PRIMARY KEY NOT NULL,
NAME TEXT NOT NULL,
AGE INT NTO NULL,
ADDRESS CHAR(50),
SALARY REAL)
'''

cursor.execute(query)

print("Table created successfully")
```

In [10]:

```
# Show the tables
cursor.execute(" SELECT name FROM sqlite_master WHERE type='table' ")
print(cursor.fetchall())
```

```
[('COMPANY',)]
```

Insert data in table

- To insert a single row into a table, you use the following form of the INSERT statement:

```
INSERT INTO table (column1,column2 ,..)
VALUES( value1, value2 ,...)
```

- Specify the name of the table to which you want to insert data after the INSERT INTO keywords.
- Add a comma-separated list of columns after the table name. The column list is optional. However, it is a good practice to include the column list after the table name.
- Add a comma-separated list of values after the VALUES keyword. If you omit the column list, you have to specify values for all columns in the value list. The number of values in the value list must be the same as the number of columns in the column list.

- To insert multiple rows into a table, you use the following form of the INSERT statement:

```
INSERT INTO table1 (column1,column2 ,..)
VALUES
   (value1,value2 ,...),
   (value1,value2 ,...),
    ...
   (value1,value2 ,...)
```

- Each value list following the VALUES clause is a row that will be inserted into the table.

In [11]:

```
query = '''
INSERT INTO COMPANY (ID, NAME, AGE, ADDRESS, SALARY)
VALUES
   (1, 'Paul', 32, 'California', 20000.00),
   (2, 'Allen', 25, 'Texas', 15000.00),
   (3, 'Teddy', 23, 'Norway', 20000.00),
   (4, 'Mark', 25, 'Richmond', 65000.00)

'''

cursor.execute(query)

print("Records created successfully")
```

```
Records created successfully
```

Select operation:

- The SELECT statement is one of the most commonly used statements in SQL.
- The SQLite SELECT statement provides all features of the SELECT statement in SQL standard.

```
SELECT column_list
FROM table
```

- Specify the table where you want to get data from in the FROM clause. Notice that you can have more than one table in the FROM clause.
- Second, specify a column or a list of comma-separated columns in the SELECT clause.

In [17]:

```
sel_query = '''
SELECT id, name, address, salary FROM COMPANY
'''

# Method 1
query_result = cursor.execute(sel_query)
for row in query_result:
  print(row)

# Method 2
# cursor.execute("SELECT * FROM COMPANY")
# result_query = cursor.fetchall()
# for row in result_query:
#   print(row)
```

```
(1, 'Paul', 'California', 20000.0)
(2, 'Allen', 'Texas', 15000.0)
(3, 'Teddy', 'Norway', 20000.0)
(4, 'Mark', 'Richmond', 65000.0)
[(1, 'Paul', 32, 'California', 20000.0), (2, 'Allen', 25, 'Texas', 15000.
0), (3, 'Teddy', 23, 'Norway', 20000.0), (4, 'Mark', 25, 'Richmond', 6500
0.0)]
```

Update operation

- To update existing data in a table, you use SQLite UPDATE statement. The following illustrates the syntax of the UPDATE statement:

```
UPDATE table
SET column_1 = new_value_1,
    column_2 = new_value_2
WHERE
    search_condition
```

- Specify the table where you want to update after the UPDATE clause.
- Set new value for each column of the table in the SET clause.
- Specify rows to update using a condition in the WHERE clause.
- The WHERE clause is optional. If you skip it, the UPDATE statement will update data in all rows of the table.

In [19]:

```
update_query = '''
Update Company set Salary = 25000 WHERE ID = 1
'''

cursor.execute(update_query)

sel_query = '''
SELECT ID, NAME, ADDRESS, SALARY FROM COMPANY
'''
query_result = cursor.execute(sel_query)
for row in query_result:
  print(row)
```

```
(1, 'Paul', 'California', 25000.0)
(2, 'Allen', 'Texas', 15000.0)
(3, 'Teddy', 'Norway', 20000.0)
(4, 'Mark', 'Richmond', 65000.0)
```

Delete operation

- The SQLite DELETE statement allows you to delete one row, multiple rows, and all rows in a table.
- The syntax of the SQLite DELETE statement is as follows:

  ```
  DELETE FROM table
  WHERE search_condition
  ```

- Specify the name of the table which you want to remove rows after the DELETE FROM keywords.
- Add a search condition in the WHERE clause to identify the rows to remove. The WHERE clause is an optional part of the DELETE statement. If you omit the WHERE clause, the DELETE statement will delete all rows in the table.

In [20]:

```
delete_query = '''
Delete from Company where ID = 2
'''
cursor.execute(delete_query)

sel_query = '''
SELECT * FROM COMPANY
'''
query_result = cursor.execute(sel_query)
for row in query_result:
  print(row)
```

```
(1, 'Paul', 32, 'California', 25000.0)
(3, 'Teddy', 23, 'Norway', 20000.0)
(4, 'Mark', 25, 'Richmond', 65000.0)
```

In [28]:

```
cursor.close()
conn.close()
```

Conclusion:
From the above experiment, I learnt the following:
Python – SQLite connection, running and executing various queries and fetching the respective data accordingly in the tabular format with the help of pandas.