Experiment No. 13
Program to demonstrate CRUD (create, read, update and delete) operations on database (SQLite/ MySQL) using python
Date of Performance: 02/04/2024
Date of Submission: 02/04/2024

Experiment No. 13

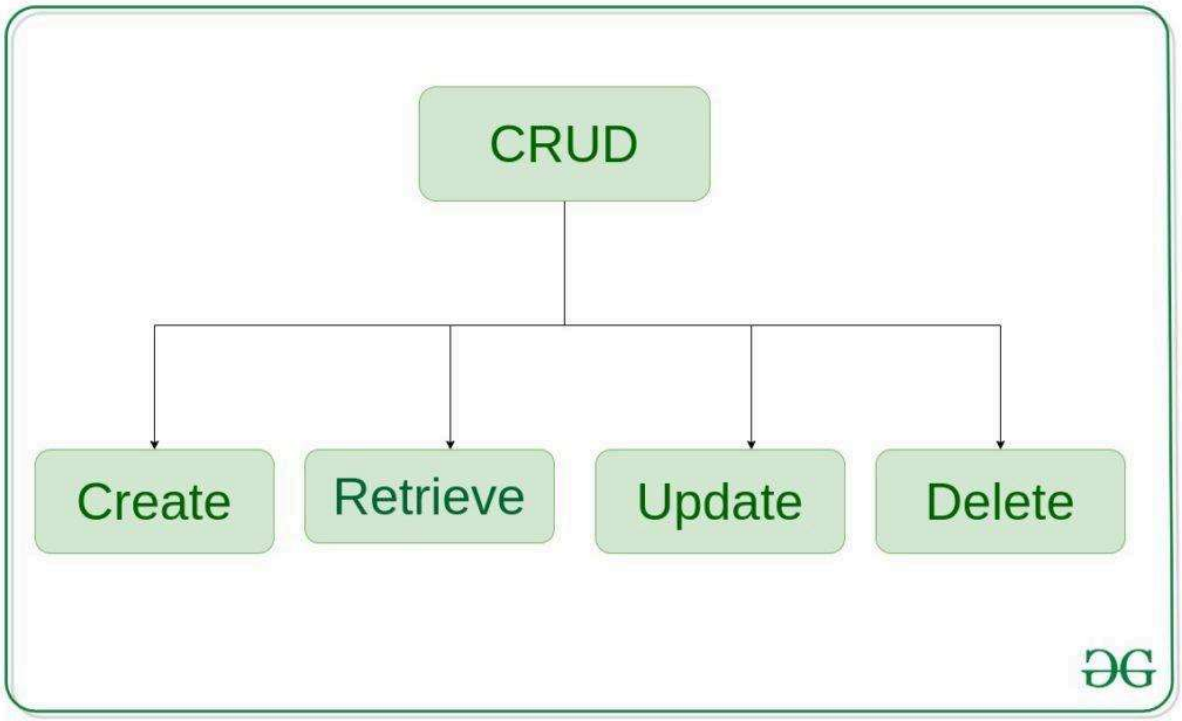
Title: Program to demonstrate CRUD (create, read, update and delete) operations on database (SQLite/ MySQL) using python

Aim: To study and implement CRUD (create, read, update and delete) operations on database (SQLite/ MySQL) using python

Objective: To introduce database connectivity with python

Theory:

In general CRUD means performing Create, Retrieve, Update and Delete operations on a table in a database. Let's discuss what actually CRUD means,



Create – create or add new entries in a table in the database.
Retrieve – read, retrieve, search, or view existing entries as a list(List View) or retrieve a particular entry in detail (Detail View)
Update – update or edit existing entries in a table in the database
Delete – delete, deactivate, or remove existing entries in a table in the database

Code :

```
import mysql.connector


# Establish connection to MySQL database
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="root",
    database="rollno_10"
)
cursor = conn.cursor()


# Create table
cursor.execute('''CREATE TABLE IF NOT EXISTS users
                  (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), email VARCHAR(255))''')


# Create (insert) operation
def create_user(name, email):
    sql = "INSERT INTO users (name, email) VALUES (%s, %s)"
    val = (name, email)
    cursor.execute(sql, val)
    conn.commit()
```

```
    print("User created successfully")


# Read operation
def read_users():
    cursor.execute("SELECT * FROM users")
    rows = cursor.fetchall()
    for row in rows:
        print(row)


# Update operation
def update_user(user_id, new_name, new_email):
    sql = "UPDATE users SET name = %s, email = %s WHERE id = %s"
    val = (new_name, new_email, user_id)
    cursor.execute(sql, val)
    conn.commit()
    print("User updated successfully")


# Delete operation
def delete_user(user_id):
    sql = "DELETE FROM users WHERE id = %s"
    val = (user_id,)
    cursor.execute(sql, val)
    conn.commit()
```

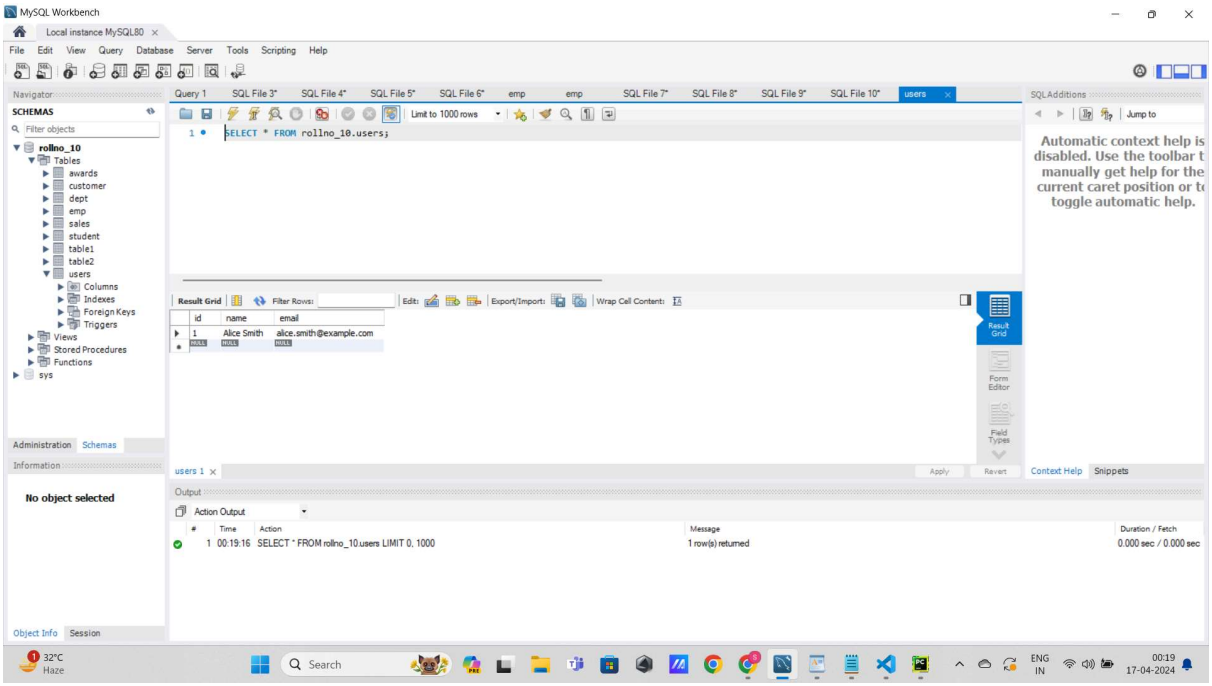
```
    print("User deleted successfully")


# Test the CRUD operations
create_user("Alice", "alice@example.com")
create_user("Bob", "bob@example.com")
print("Users before update:")
read_users()
update_user(1, "Alice Smith", "alice.smith@example.com")
print("Users after update:")
read_users()
delete_user(2)
print("Users after delete:")
read_users()


# Close the connection
conn.close()
```

Output :

```
User created successfully
User created successfully
Users before update:
(1, 'Alice', 'alice@example.com')
(2, 'Bob', 'bob@example.com')
User updated successfully
Users after update:
(1, 'Alice Smith', 'alice.smith@example.com')
(2, 'Bob', 'bob@example.com')
User deleted successfully
Users after delete:
(1, 'Alice Smith', 'alice.smith@example.com')
```



Conclusion:

Database connectivity in Python allows seamless interaction between Python applications and databases, enabling operations such as querying, inserting, updating, and deleting data. Libraries like `sqlite3` for SQLite or `mysql-connector-python` for MySQL provide functions to establish connections, execute SQL queries, and handle database transactions. This connectivity empowers developers to build robust, data-driven applications with ease, enhancing efficiency and scalability.