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

**Experiment No. 13**

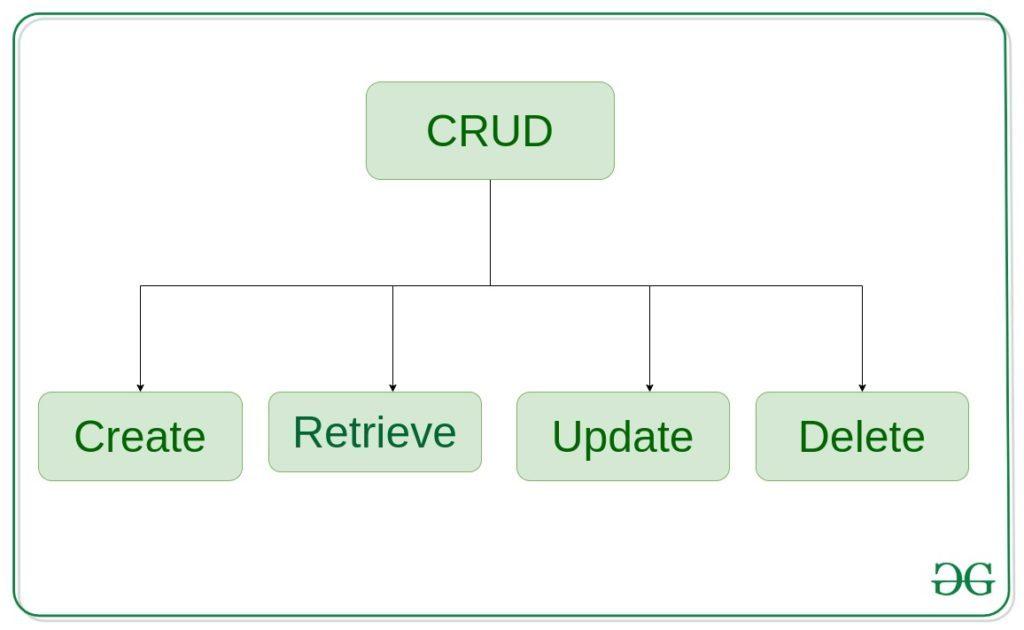
**Title:** Program to demonstrate CRUD (create, read, update and delete) operations on database (SQLite/ MySQL) using python

**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:")

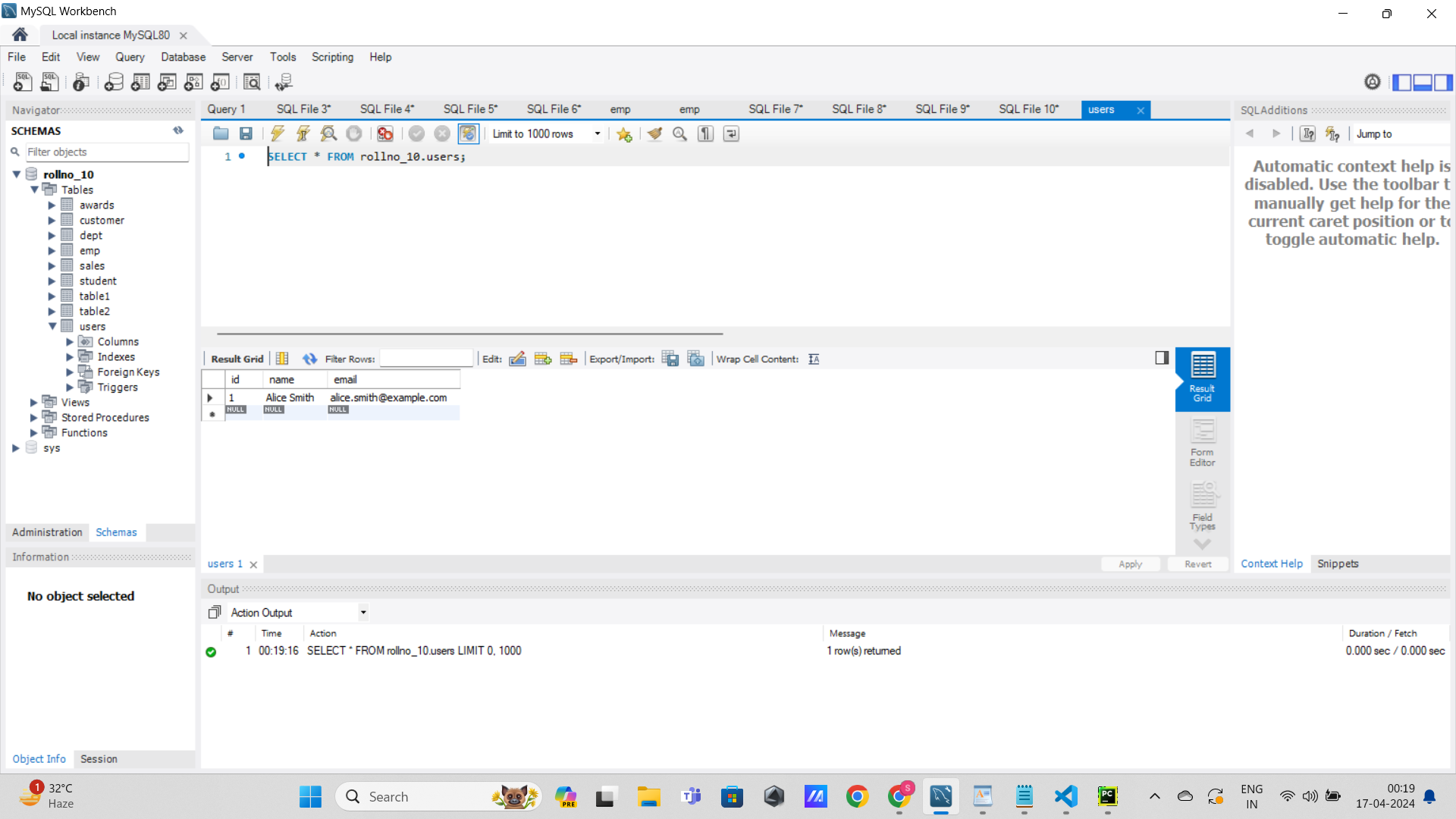
read\_users()

# Close the connection

conn.close()

**Output :**

****

****

**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.