**EXPERIMENT 9**

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

**THEORY:**

**Sqlite Database**

SQLite is a self-contained transactional relational database engine that doesn't require a server configuration, as in the case of Oracle, MySQL, etc. The entire SQLite database is contained in a single file, which can be put anywhere in the computer's file system. SQLite is widely used as an embedded database in mobile devices, web browsers and other stand-alone applications. In spite of being small in size, it is a fully ACID compliant database conforming to ANSI SQL standards.

**Python DB-API**

Python Database API is a set of standards recommended by a Special Interest Group for database module standardization. Python modules that provide database interfacing functionality with all major database products are required to adhere to this standard. DB-API standards.

Standard Python distribution has in-built support for SQLite database connectivity. It contains sqlite3 module which adheres to DB-API 2.0. Other RDBMS compliant modules to DB-API:

* MySQL: PyMySql module
* Oracle: Cx-Oracle module
* SQL Server: PyMsSql module
* PostGreSQL: psycopg2 module
* ODBC: pyodbc module

In order to establish a connection with a SQLite database, sqlite3 module needs to be imported and the **connect()** function needs to be executed.

>>> import sqlite3

>>> db=sqlite3.connect('test.db')

The following methods are defined in the connection class:

Graphical user interface, text, application, email

Description automatically generated

A **cursor** is a Python object that enables one to work with the database. It acts as a handle for a given SQL query; it allows the retrieval of one or more rows of the result. A cursor object is obtained from the connection to execute SQL queries using the following statement:

>>>cur = db.cursor()

The following methods of the cursor object are:

Graphical user interface, text, application, email

Description automatically generated

**CRUD Operations:**

**1) CREATE - a New Table**

A string enclosing the CREATE TABLE query is passed as parameter to the execute() method of the cursor object.

**2) Insert a New Record**

The execute() method of the cursor object should be called with a string argument representing the INSERT query syntax. For example in a student table having three fields: name, age and marks. The string holding the INSERT query is defined as:

>>> qry="INSERT INTO student (name, age, marks) VALUES ('Rajeev',20,50);"

**3) Insert multiple records**

executemany() method is used to add multiple records at once. Data to be added should be given in a list of tuples, with each tuple containing one record. The list object (containing tuples) is the parameter of the executemany() method, along with the query string. For example,

>>> qry="insert into student (name, age, marks) values(?,?,?);"

>>> students=[('Jill', 18, 70), ('Jane', 25, 87)]

……

>>> cur=db.cursor()

>>> cur.executemany(qry, students)

>>> db.commit()

**4) READ - Retrieve Records**

When the query string holds a SELECT query, the execute() method forms a result set object containing the records returned. Python DB-API defines two methods to fetch the records:

* fetchone(): Fetches the next available record from the result set. It is a tuple consisting of values of each column of the fetched record.

record = cur.fetchone()

* fetchall(): Fetches all remaining records in the form of a list of tuples. Each tuple corresponds to one record and contains values of each column in the table.

records = cur.fetchall()

**5) Update a Record**

The query string in the execute() method should contain an UPDATE query syntax. For example, to update the value of 'age' to 17 for 'John'

>>> qry="update student set age = 17 where name='John' ; "

**6) Delete a Record**

The query string should contain the DELETE query syntax. For example, the below code is used to delete 'Bill' from the student table.

>>> qry="DELETE from student where name='Bill';"

**CODE:**

import sqlite3

conn = sqlite3.connect("test.db")

crsr = conn.cursor()

print("Connected to database")

#Creating table

cmd = "CREATE TABLE Employee(empId INTEGER PRIMARY KEY AUTOINCREMENT, fname VARCHAR(20) NOT NULL, lname VARCHAR(20) NOT NULL, gender CHAR(1) DEFAULT 'M');"

crsr.execute(cmd)

#Inserting data

cmd = "INSERT INTO Employee(fname,lname,gender) VALUES ('Jack','Smith','M'), ('Jane','Doyle', 'F'), ('Tim','Brookes','M'), ('Lydia','Simson','F'), ('Penny','Hill','M');"

crsr.execute(cmd)

conn.commit()

#Reading data

print("All Employees:")

cmd = "SELECT \* FROM Employee;"

crsr.execute(cmd)

results = crsr.fetchall()

for emp in results :

print(emp)

#Updating data

crsr = conn.cursor()

cmd = "UPDATE Employee SET gender='F' WHERE fname='Penny' AND lname='Hill';"

crsr.execute(cmd)

conn.commit()

print("\nAfter updating employee gender: ")

cmd = "SELECT \* FROM Employee;"

crsr.execute(cmd)

results = crsr.fetchall()

for emp in results :

print(emp)

#Deleting data

crsr = conn.cursor()

cmd = "DELETE FROM Employee WHERE fname='Penny';"

crsr.execute(cmd)

conn.commit()

print("\nAfter deleting employee record:")

cmd = "SELECT \* FROM Employee;"

crsr.execute(cmd)

results = crsr.fetchall()

for emp in results :

print(emp)

conn.close()

**OUTPUT:**

Text

Description automatically generated

**CONCLUSION:** In this experiment, I have implemented the CRUD i.e. Create, Read, Update and Delete operations on SQLite database using python. **Python DB-API** provide database interfacing functionality with all major database products are required to adhere to this standard. DB-API is also compatible with MySQL: PyMySql module Oracle: Cx-Oracle module, SQL Server: PyMsSql module, PostGreSQL: psycopg2 module and ODBC: pyodbc module **Sqlite Database** is a self-contained transactional relational database engine that is used as an embedded database in python.