

## **Experiment 10**

Write python programs to understand GUI database connectivity to perform CRUD operations in python (Use any one database like SQLite, MySQL, Oracle, PostgreSQL etc.).

Roll No.	61
Name	V Krishnasubramaniam
Class	D10-A
Subject	Python Lab
LO Mapped	LO1: Understand the structure, syntax, and semantics of the Python language LO5: Gain proficiency in evaluating database operations in Python.

**Aim:**

Write python programs to understand GUI database connectivity to perform CRUD operations in python (Use any one database like SQLite, MySQL, Oracle, PostgreSQL etc.).

**Introduction:****Database**

A database is a file that is organized for storing data. Like a dictionary, database software is designed to keep the inserting and accessing of data very fast, even for large amounts of data. Database software maintains its performance by building indexes as data is added to the database to allow the computer to jump quickly to a particular entry. SQL stands for Structured Query Language and is a widely used programming language for managing relational databases. There are many different flavours of database system language which are used for a wide variety of purposes including:

1. Oracle
2. MySQL
3. Microsoft SQL Server
4. PostgreSQL
5. SQLite

MySQL is one of the most popular database management systems (DBMSs) on the market today. As most software applications need to interact with data in some form, programming languages like Python provide tools for storing and accessing these data sources.

**MySQL Connector Python**

A database driver is a piece of software that allows an application to connect and interact with a database system. Programming languages like Python need a special driver before they can speak to a database from a specific vendor.

These drivers are typically obtained as third-party modules. The Python Database API (DB-API) defines the standard interface with which all Python database drivers must comply. These details are documented in PEP 249. All Python database drivers, such as sqlite3 for SQLite, pycopg for PostgreSQL, and MySQL Connector/Python for MySQL, follow these implementation rules.

**Installation**

in Python you need to install a Python MySQL connector to interact with a MySQL database. Many packages follow the DB-API standards, but the most popular among them is MySQL Connector/Python. You can get it with pip:  
pip install mysql-connector-python

pip installs the connector as a third-party module in the currently active virtual environment. It's recommended that you set up an isolated virtual environment for the project along with all the dependencies.

To test if the installation was successful, type the following command on your Python terminal:  
`import mysql.connector`

If the above code executes with no errors, then `mysql.connector` is installed and ready to use. If you encounter any errors, then make sure you're in the correct virtual environment and you're using the right Python interpreter.

Make sure that you're installing the correct `mysql-connector-python` package, which is a pure-Python implementation. Beware of similarly named but now deprecated connectors like `mysql-connector`.

MySQL is a server-based database management system. One server might contain multiple databases. To interact with a database, you must first establish a connection with the server. The general workflow of a Python program that interacts with a MySQL-based database is as follows:

1. Connect to the MySQL server.
2. Create a new database.
3. Connect to the newly created or an existing database.
4. Execute a SQL query and fetch results.
5. Inform the database if any changes are made to a table.
6. Close the connection to the MySQL server.

### Establishing a connection

The first step in interacting with a MySQL server is to establish a connection. To do this, you need `connect()` from the `mysql.connector` module. This function takes in parameters like host, user, and password and returns a `MySQLConnection` object. You can receive these credentials as input from the user and pass them to `connect()`.

Argument	Description
Username	The username that you use to work with MySQL Server. The default username for the MySQL database is root.
Password	Password is given by the user at the time of installing the MySQL server. If you are using root then you won't need the password.
Host name	The server name or IP address on which MySQL is running. If you are running on localhost, then you can use localhost or its IP 127.0.0.0
Database name	The name of the database to which you want to connect and perform the operations.

Use the `connect()` method of the `MySQL Connector` class with the required arguments to connect MySQL. It would return a `MySQLConnection` object if the connection established successfully.

```
Connection-Object = mysql.connector.connect(host = <host-name>, user = <username>, passwd = <password>)
```

The connect() method can throw a Database error exception if one of the required parameters is wrong. For example, if you provide a database name that is not present in MySQL. The is\_connected() is the method of the MySQLConnection class through which we can verify if our Python application is connected to MySQL. At last, we are closing the MySQL database connection using a close() method of MySQLConnection class.

Example:

```
from getpass import getpass
from mysql.connector import connect, Error
try:
    with connect(
        host="localhost",
        user=input("Enter username: "),
        password=getpass("Enter password: "),
    ) as connection:
        print(connection)
except Error as e:
    print(e)
```

The code above uses the entered login credentials to establish a connection with your MySQL server. In return, you get a MySQLConnection object, which is stored in the connection variable. From now on, you'll use this variable to access your MySQL server.

There are several important things to notice in the code above:

1. You should always deal with the exceptions that might be raised while establishing a connection to the MySQL server. This is why you use a try ... except block to catch and print any exceptions that you might encounter.
2. You should always close the connection after you're done accessing the database. Leaving unused open connections can lead to several unexpected errors and performance issues. The above code takes advantage of a context manager using with, which abstracts away the connection cleanup process.
3. You should never hard-code your login credentials, that is, your username and password, directly in a Python script. This is a bad practice for deployment and poses a serious security threat. The code above prompts the user for login credentials. It uses the built-in getpass module to hide the password. While this is better than hard-coding, there are other, more secure ways to store sensitive information, like using environment variables.

You've now established a connection between your program and your MySQL server, but you still need to either create a new database or connect to an existing database inside the server.

### **Creating database**

To execute a SQL query in Python, you'll need to use a cursor, which abstracts away the access to database records. MySQL Connector/Python provides you with the MySQLCursor class,

which instantiates objects that can execute MySQL queries in Python. An instance of the MySQLCursor class is also called a cursor.

The cursor object can be defined as an abstraction specified in the Python DB-API 2.0. It facilitates us to have multiple separate working environments through the same connection to the database. We can create the cursor object by calling the 'cursor' function of the connection object. The cursor object is an important aspect of executing queries to the databases.

Use the cursor() method of a MySQLConnection object to create a cursor object to perform various SQL operations.

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

The execute() methods run the SQL query and return the result. Use cursor.fetchall() or fetchone() or fetchmany() to read query results. Use cursor.close() and connection.close() method to close open connections after your work completes.

## CRUD Operations on Database Tables

### Insert a single row into MySQL table from Python:

Define a SQL Insert query. Next, prepare a SQL INSERT query to insert a row into a table. In the insert query, we mention column names and their values to insert in a table.

```
INSERT INTO mysql_table (col1, col2, ...) VALUES (val1, val2, ...);
```

Get Cursor Object from Connection by using a connection.cursor() method to create a cursor object. This method creates a new MySQLCursor object.

Execute the insert query using the cursor.execute() method. This method executes the operation stored in the Insert query.

Commit your changes after the successful execution of a query, changes persist into a database using the commit() of a connection class.

Get the number of rows affected after a successful insert operation, use a cursor.rowcount() method to get the number of rows affected. The count depends on how many rows you are inserting.

Verify the result using the SQL SELECT query by executing a MySQL select query from Python to see the new changes.

Close the cursor object and database connection object using cursor.close() and connection.close() method to close open connections after your work completes.

### Example:

```
import mysql.connector
conn=mysql.connector.connect(host='localhost',database='world',user='root',password='student')
if conn.is_connected():
    print('Connected to MySQL database')
cursor=conn.cursor()
```

```
str="insert into emptab(eno,ename,sal) values(9999,'srinivas',9999.99)"
try:
    cursor.execute(str)
    conn.commit()
    print('1 row inserted....')
except:
    conn.rollback()
cursor.close()
conn.close()
```

**Output:**

Connected to MySQL database  
1 row inserted....

**Select rows from MySQL table from Python:**

Define a SQL SELECT Query. Next, prepare a SQL SELECT query to fetch rows from a table. You can select all or limited rows based on your requirement. If the where condition is used, then it decides the number of rows to fetch.

```
SELECT col1, col2,...colnN FROM MySQL_table WHERE id = 10;
```

Get Cursor Object from Connection using a connection.cursor() method to create a cursor object. This method creates a new MySQLCursor object.

Execute the select query using the cursor.execute() method.

Extract all rows from a result after successfully executing a Select operation, use the fetchall() method of a cursor object to get all rows from a query result. It returns a list of rows.

Iterate a row list using a for loop and access each row individually (Access each row's column data using a column name or index number).

Close the cursor object and database connection object using cursor.close() and connection.close() method to close open connections after your work completes.

**Example:**

```
import mysql.connector
conn=mysql.connector.connect(host='localhost',database='world',user='root',password='student')
if conn.is_connected():
    print('Connected to MySQL database')
cursor=conn.cursor()
cursor.execute("select * from emptab")
rows=cursor.fetchall()
print('Total number of rows=',cursor.rowcount)
for row in rows:
    eno=row[0]
    ename=row[1]
    sal=row[2]
    print('%-6d %-15s %10.2f' % (eno,ename,sal))
cursor.close()
```

```
conn.close()
```

**Output:**

```
Connected to MySQL database
Total number of rows=2
1001  Nagesh7800.00
1002  Ganesh8800.00
```

**Update a row from MySQL table from Python:**

Prepare a SQL Update Query. Prepare an update statement query with data to update.

```
UPDATE table_name SET col1 = val1, col2 = val2 WHERE condition;
```

Execute the UPDATE query using cursor.execute() method. This method executes the operation stored in the UPDATE query.

Commit your changes and make modification persistent into a database using the commit() of a connection class.

Extract the number of rows affected after a successful update operation, use a cursor.rowcount() method to get the number of rows affected. The count depends on how many rows you are updating.

Verify the result using the SQL SELECT query by executing a MySQL select query from Python to see the new changes.

Close the cursor object and database connection object using cursor.close() and connection.close() method to close open connections after your work completes.

**Example:**

```
import mysql.connector
conn=mysql.connector.connect(host='localhost',database='world',user='root',password='student')
cursor=conn.cursor()
def update_rows(eno):
    str="update emptab set sal=sal+1000 where
    eno='%d'"
    args=(eno)
    try:
        cursor.execute(str%args)
        conn.commit()
        print('1 row updated....')
    except:
        conn.rollback()
    finally:
        cursor.close()
        conn.close()
x=int(input('Enter eno:'))
update_rows(x)
```

**Output:**

Enter eno: 1002  
1 row updated....

### **Delete a single row from MySQL table from Python:**

Define a SQL Delete Query. Next, prepare a SQL delete query to delete a row from a table. Delete query contains the row to be deleted based on a condition placed in where clause of a query.

```
DELETE FROM MySQL_table WHERE id=10;
```

Get Cursor Object from Connection using a connection.cursor() method to create a cursor object. This method creates a new MySQLCursor object.

Execute the delete query using the cursor.execute() method. This method executes the operation stored in the delete query. After a successful delete operation, the execute() method returns us the number of rows affected.

Commit your changes after successfully executing a delete operation, make changes persistent into a database using the commit() of a connection class.

Get the number of rows affected by using a cursor.rowcount() method to get the number of rows affected. The count depends on how many rows you are deleting. You can also execute a MySQL select query from Python to verify the result.

Close the cursor object and database connection object using cursor.close() and connection.close() method to close open connections after your work completes.

#### Example:

```
import mysql.connector;
conn=mysql.connector.connect(host='localhost',database='world',user='root',password='student')
cursor=conn.cursor()
def delete_rows(eno):
    str="delete from emptab where eno='%d'"
    args=(eno)
    try:
        cursor.execute(str%args)
        conn.commit()
        print('1 row deleted....')
    except:
        conn.rollback()
    finally:
        cursor.close()
        conn.close()
x=int(input('Enter eno:'))
delete_rows(x)
```

#### Output:

Enter eno:1001  
1 row deleted....



**Results:**

Program in gui with mysql database.py:

```
from tkinter.ttk import *
from tkinter import *
import mysql.connector
from tkinter import messagebox

mydb=mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="school"
)

mycursor=mydb.cursor()
root=Tk()
root.title("Student Data")
root.geometry("440x410")

label1 = Label(root, text="Roll No.", width=13, anchor='w', font=("Sylfaen", 12)).grid(row=1,
column=0, padx=(10,0))
label2 = Label(root, text="First Name", width=13, anchor='w', font=("Sylfaen", 12)).grid(row=2,
column=0, padx=(10,0))
label3 = Label(root, text="Last Name", width=13, anchor='w', font=("Sylfaen", 12)).grid(row=3,
column=0, padx=(10,0))
label4 = Label(root, text="Phone Number", width=13, anchor='w', font=("Sylfaen",
12)).grid(row=4, column=0, padx=(10,0))
label5 = Label(root, text="City", width=13, anchor='w', font=("Sylfaen", 12)).grid(row=5,
column=0, padx=(10,0))
label6 = Label(root, text="State", width=13, anchor='w', font=("Sylfaen", 12)).grid(row=6,
column=0, padx=(10,0))
label7 = Label(root, text="Age", width=13, anchor='w', font=("Sylfaen", 12)).grid(row=7,
column=0, padx=(10,0))

e1 = Entry(root, width=30)
e1.grid(row=1, column=1, padx=(0,20), pady = 20)
e2 = Entry(root, width=30)
e2.grid(row=2, column=1, padx=(0,20), pady = 20)
e3 = Entry(root, width=30)
e3.grid(row=3, column=1, padx=(0,20), pady = 20)
e4 = Entry(root, width=30)
```

```
e4.grid(row=4, column=1, padx=(0,20), pady = 20)
e5 = Entry(root, width=30)
e5.grid(row=5, column=1, padx=(0,20), pady = 20)
e6 = Entry(root, width=30)
e6.grid(row=6, column=1, padx=(0,20), pady = 20)
e7 = Entry(root, width=30)
e7.grid(row=7, column=1, padx=(0,20), pady = 20)
```

```
def Register():
```

```
    RollNo=int(e1.get())
    dbRollNo=""
    Select="select RollNo from student where RollNo=%s" %(RollNo)
    mycursor.execute(Select)
    result=mycursor.fetchall()
    for i in result:
        dbRollNo=i[0]
    if(RollNo == dbRollNo):
        messagebox.askokcancel("Information", "Record Already exists")
    else:
        Insert="Insert into student(RollNo,First_Name,Last_Name,Phone_Number,City,State,Age)
values(%s,%s,%s,%s,%s,%s,%s)"
        First_Name=e2.get()
        Last_Name=e3.get()
        Phone_Number=e4.get()
        City=e5.get()
        State=e6.get()
        Age=e7.get()
        if(First_Name !="" and Last_Name !="" and Phone_Number !="" and City !="" and State
!="" and Age != ""):
            Value=(RollNo,First_Name,Last_Name,Phone_Number,City,State,Age)
            mycursor.execute(Insert,Value)
            mydb.commit()
            messagebox.askokcancel("Information", "Record inserted")
            e1.delete(0, END)
            e2.delete(0, END)
            e3.delete(0, END)
            e4.delete(0, END)
            e5.delete(0, END)
            e6.delete(0, END)
            e7.delete(0, END)
        else:
            if (First_Name == "" and Last_Name == "" and Phone_Number == "" and City == "" and
State == "" and Age == ""):
                messagebox.askokcancel("Information", "New Entry Fill All Details")
```

```
    else:
        messagebox.askokcancel("Information", "Some fields left blank")

def ShowRecord():
    RollNo=int(e1.get())
    dbRollNo=""
    Select="select RollNo from student where RollNo=%s" %(RollNo)
    mycursor.execute(Select)
    result1=mycursor.fetchall()
    for i in result1:
        dbRollNo=i[0]
    Select1="select First_Name,Last_Name,Phone_Number,City,State,Age from student where
RollNo='%s'" %(RollNo)
    mycursor.execute(Select1)
    result2=mycursor.fetchall()
    First_Name=""
    Last_Name=""
    Phone_Number=""
    City=""
    State=""
    Age=""
    if(RollNo == dbRollNo):
        for i in result2:
            First_Name=i[0]
            Last_Name=i[1]
            Phone_Number=i[2]
            City=i[3]
            State=i[4]
            Age=i[5]
            e2.insert(0,First_Name)
            e3.insert(0, Last_Name)
            e4.insert(0, Phone_Number)
            e5.insert(0, City)
            e6.insert(0, State)
            e7.insert(0, Age)
    else:
        messagebox.askokcancel("Information","No Record exists")

def Delete():
    RollNo=int(e1.get())
    Delete="delete from student where RollNo=%s" %(RollNo)
    mycursor.execute(Delete)
    mydb.commit()
```

```
messagebox.showinfo("Information","Record Deleted")
e1.delete(0,END)
e2.delete(0, END)
e3.delete(0, END)
e4.delete(0, END)
e5.delete(0, END)
e6.delete(0, END)
e7.delete(0,END)
```

```
def Update():
```

```
    RollNo=int(e1.get())
```

```
    First_Name=e2.get()
```

```
    Last_Name=e3.get()
```

```
    Phone_Number=e4.get()
```

```
    City=e5.get()
```

```
    State=e6.get()
```

```
    Age=e7.get()
```

```
    Update="Update student set First_Name='%s', Last_Name='%s', Phone_Number='%s',  
City='%s', State='%s', Age='%s' where RollNo='%s'"
```

```
%(First_Name,Last_Name,Phone_Number,City,State,Age,RollNo)
```

```
    mycursor.execute(Update)
```

```
    mydb.commit()
```

```
    messagebox.showinfo("Info","Record Update")
```

```
def Showall():
```

```
    class A(Frame):
```

```
        def __init__(self, parent):
```

```
            Frame.__init__(self, parent)
```

```
            self.CreateUI()
```

```
            self.LoadTable()
```

```
            self.grid(sticky=(N, S, W, E))
```

```
            parent.grid_rowconfigure(0, weight=1)
```

```
            parent.grid_columnconfigure(0, weight=1)
```

```
    def CreateUI(self):
```

```
        tv=Treeview(self)
```

```
        tv['columns']=('1', '2', '3', '4', '5', '6', '7')
```

```
        tv['show']='headings'
```

```
        tv.column("1", width = 90, anchor ='c')
```

```
        tv.column("2", width = 90, anchor ='c')
```

```
        tv.column("3", width = 90, anchor ='c')
```

```
        tv.column("4", width = 90, anchor ='c')
```

```
tv.column("5", width = 90, anchor ='c')
tv.column("6", width = 90, anchor ='c')
tv.column("7", width = 90, anchor ='c')

tv.heading('1', text='Roll No.')
tv.heading('2', text='First Name')
tv.heading('3', text='Last Name')
tv.heading('4', text='Phone Number')
tv.heading('5', text='City')
tv.heading('6', text='State')
tv.heading('7', text='Age')

tv.grid(sticky=(N,S,W,E))
self.tview = tv
self.grid_rowconfigure(0, weight=1)
self.grid_columnconfigure(0, weight=1)

def LoadTable(self):
    Select="Select * from student"
    mycursor.execute(Select)
    result=mycursor.fetchall()
    RollNo=""
    First_Name=""
    Last_Name=""
    Phone_Number=""
    City=""
    State=""
    Age=""
    for i in result:
        RollNo=i[0]
        First_Name=i[1]
        Last_Name=i[2]
        Phone_Number=i[3]
        City=i[4]
        State=i[5]
        Age=i[6]

self.tview.insert("",'end',text="L1",values=(RollNo,First_Name,Last_Name,Phone_Number,City
,State,Age))
root=Tk()
root.title("Overview Page")
root.resizable(width = 1, height = 1)
A(root)

def Clear():
```

```
e1.delete(0, END)
e2.delete(0, END)
e3.delete(0, END)
e4.delete(0, END)
e5.delete(0, END)
e6.delete(0, END)
e7.delete(0, END)
```

```
button1 = Button(root, text="Register", width=10, bg="LightSkyBlue1", command=Register)
button1.grid(row=1,column=3)
button2 = Button(root, text="Delete", width=10, bg="LightSkyBlue2", command=Delete)
button2.grid(row=2,column=3)
button3 = Button(root, text="Update", width=10, bg="LightSkyBlue3", command=Update)
button3.grid(row=3,column=3)
button4 = Button(root, text="Show record", width=10, bg="SkyBlue1", command=ShowRecord)
button4.grid(row=4,column=3)
button5 = Button(root, text="Show All", width=10, bg="SkyBlue2", command=Showall)
button5.grid(row=5,column=3)
button6 = Button(root, text="Clear", width=10, bg="SkyBlue3", command=Clear)
button6.grid(row=6,column=3)
```

```
root.mainloop()
```

### Output:

Inserting:

Student Data

Roll No.

First Name

Last Name

Phone Number

City

State

Age

Show all records:

Overview Page						
Roll No.	First Name	Last Name	Phone Number	City	State	Age
1	Aamir	Ansari	1234567890	Mumbai	Maharashtra	20
15	Isha	Gawde	1234567890	Mumbai	Maharashtra	19
24	Sreekesh	Iyer	1234567890	Chennai	Tamil Nadu	19
30	Kanaiya	Kanabar	1234567890	Rajkot	Gujarat	19
53	Ninad	Rao	1234567890	Mangalore	Karnataka	19
61	Krishnasubramani	V	1234567890	Pallakad	Kerala	20

Show record/Reading data:

Student Data


Roll No.	<input type="text" value="1"/>	Register
First Name	<input type="text" value="Aamir"/>	Delete
Last Name	<input type="text" value="Ansari"/>	Update
Phone Number	<input type="text" value="1234567890"/>	Show record
City	<input type="text" value="Mumbai"/>	Show All
State	<input type="text" value="Maharashtra"/>	Clear
Age	<input type="text" value="20"/>	

Updating data:

Student Data

Roll No.	<input type="text" value="1"/>	Register
First Name	<input type="text" value="Aamir"/>	Delete
Last Name	<input type="text" value="Ansari"/>	Update
Phone Number	<input type="text" value="1234567890"/>	Show record
City	<input type="text" value="Navi Mumbai"/>	Show All
State	<input type="text" value="Maharashtra"/>	Clear
Age	<input type="text" value="20"/>	

Info

 Record Update

OK



Overview Page						
Roll No.	First Name	Last Name	Phone Number	City	State	Age
1	Aamir	Ansari	1234567890	Navi Mumbai	Maharashtra	20
15	Isha	Gawde	1234567890	Mumbai	Maharashtra	19
24	Sreekesh	Iyer	1234567890	Chennai	Tamil Nadu	19
30	Kanaiya	Kanabar	1234567890	Rajkot	Gujarat	19
53	Ninad	Rao	1234567890	Mangalore	Karnataka	19
61	Krishnasubramani	V	1234567890	Pallakad	Kerala	20

Deleting data:

Student Data

Roll No.

27

Register

First Name

Delete

Last Name

Update

Phone Number

Show record

City

Show All

State

Clear

Age

Information

i

Record Deleted

OK

Overview Page						
Roll No.	First Name	Last Name	Phone Number	City	State	Age
1	Aamir	Ansari	1234567890	Navi Mumbai	Maharashtra	20
15	Isha	Gawde	1234567890	Mumbai	Maharashtra	19
24	Sreekesh	Iyer	1234567890	Chennai	Tamil Nadu	19
30	Kanaiya	Kanabar	1234567890	Rajkot	Gujarat	19
53	Ninad	Rao	1234567890	Mangalore	Karnataka	19
61	Krishnasubramani	V	1234567890	Pallakad	Kerala	20

### Program in gui with mysqlite database.py:

```

from tkinter import *
from tkinter import messagebox
from tkinter import ttk
import sqlite3

class Database:
    def __init__(self, db):
        self.conn = sqlite3.connect(db)
        self.cur = self.conn.cursor()
        self.cur.execute("CREATE TABLE IF NOT EXISTS user (user_id INTEGER PRIMARY KEY, fname text, lname text, city text, gender text, age INTEGER)")
        self.conn.commit()
    def fetch(self):
        self.cur.execute("SELECT * FROM user")
        rows = self.cur.fetchall()
        return rows
    def insert(self, user_id, fname, lname, city, gender, age):
        self.cur.execute("INSERT INTO user VALUES (?, ?, ?, ?, ?, ?)",
            (user_id, fname, lname, city, gender, age))
        self.conn.commit()
    def remove(self, user_id):
        self.cur.execute("DELETE FROM user WHERE user_id=?", (user_id,))
        self.conn.commit()
    def update(self, user_id, fname, lname, city, gender, age):
        self.cur.execute("UPDATE user SET fname = ?, lname = ?, city = ?, gender = ?, age = ? WHERE user_id = ?",
            (fname, lname, city, gender, age, user_id))
        self.conn.commit()
    def __del__(self):

```

```
self.conn.close()

def add_item():
    if rollVar.get() == " or fnameVar.get() == " or lnameVar.get() == " or cityVar.get() == " or
genderVar.get() == " or ageVar.get() == 0:
        messagebox.showerror('Required Fields', 'Please include all fields')
        return
    db.insert(rollVar.get(), fnameVar.get(),lnameVar.get(),cityVar.get(),genderVar.get(),
ageVar.get())
    listbox.delete(0, END)
    listbox.insert(END, (rollVar.get(),
fnameVar.get(),lnameVar.get(),cityVar.get(),genderVar.get(), ageVar.get()))
    clear_text()
    listbox.delete(0, END)
    for row in db.fetch():
        listbox.insert(END, row)

def select_item(event):
    try:
        global selected_item
        index = listbox.curselection()[0]
        selected_item = listbox.get(index)
        rollVar.set(selected_item[0])
        fnameVar.set(selected_item[1])
        lnameVar.set(selected_item[2])
        cityVar.set(selected_item[3])
        genderVar.set(selected_item[4])
        ageVar.set(selected_item[5])
    except IndexError:
        pass

def remove_item():
    db.remove(selected_item[0])
    clear_text()
    listbox.delete(0, END)
    for row in db.fetch():
        listbox.insert(END, row)

def update_item():
    db.update(selected_item[0], fnameVar.get(),lnameVar.get(),cityVar.get(),genderVar.get(),
ageVar.get())
```

```
listbox.delete(0, END)
for row in db.fetch():
    listbox.insert(END, row)
```

```
def clear_text():
    rollEntry.delete(0, END)
    fnameEntry.delete(0, END)
    lnameEntry.delete(0, END)
    cityMenu.delete(0, END)
    ageSpinbox.delete(0,END)
```

```
db = Database('store.db')
```

```
app = Tk()
app.configure(bg='light cyan')
app.title('CRUD Application')
app.geometry('446x500')
```

```
rollVar = IntVar()
rollLabel = Label(app ,text = "User ID",bg="light cyan",width=30, anchor='w').grid(row =
0,column = 0)
rollEntry = Entry(app,textvariable=rollVar)
rollEntry.grid(row = 0,column = 1,pady=10)
```

```
fnameVar = StringVar()
fnameLabel = Label(app ,text = "First Name",bg="light cyan",width=30, anchor='w').grid(row =
1,column = 0)
fnameEntry = Entry(app,textvariable=fnameVar)
fnameEntry.grid(row = 1,column = 1,pady=10)
```

```
lnameVar = StringVar()
lnameLabel = Label(app ,text = "Last Name",width=30, anchor='w',bg="light cyan").grid(row =
2,column = 0)
lnameEntry = Entry(app,textvariable=lnameVar)
lnameEntry.grid(row = 2,column = 1,pady=10)
```

```
cityVar = StringVar()
```

```
cityLabel = Label(app ,text = "Select City",width=30, anchor='w',bg="light cyan").grid(row =
3,column = 0)
cityMenu = ttk.Combobox(app,width=17,textvariable=cityVar)
cityMenu['values']=('Mumbai','Chennai','Delhi','Thrissur','Kochi','Palakkad','Mangalore','Rajkot','
Surat')
cityMenu.grid(row = 3,column = 1,pady=10)
```

```
genderLabel = Label(app ,text = "Gender",width=30, anchor='w',bg="light cyan").grid(row =
4,column = 0)
frame = Frame(app,bg="light cyan")
frame.grid(row=4,column=1,pady=10)
genderVar = StringVar()
dr1=Radiobutton(frame,text='Male',variable = genderVar,value='Male',
tristatevalue="x",bg="light cyan")
dr1.pack(side=LEFT)
dr2=Radiobutton(frame,text='Female',variable = genderVar,value='Female',
tristatevalue="x",bg="light cyan")
dr2.pack(side=LEFT)
```

```
ageVar = IntVar()
ageLabel = Label(app ,text = "Age",width=30, anchor='w',bg="light cyan").grid(row = 5,column
= 0)
ageSpinbox = Spinbox(app, from_=1, to=130, width=19,textvariable=ageVar)
ageSpinbox.grid(row=5,column=1,pady=10)
```

```
listbox = Listbox(app, height=11, width=70)
listbox.grid(row=7, column=0, columnspan=3, rowspan=6, padx=10)
listbox.bind('<<ListboxSelect>>', select_item)
```

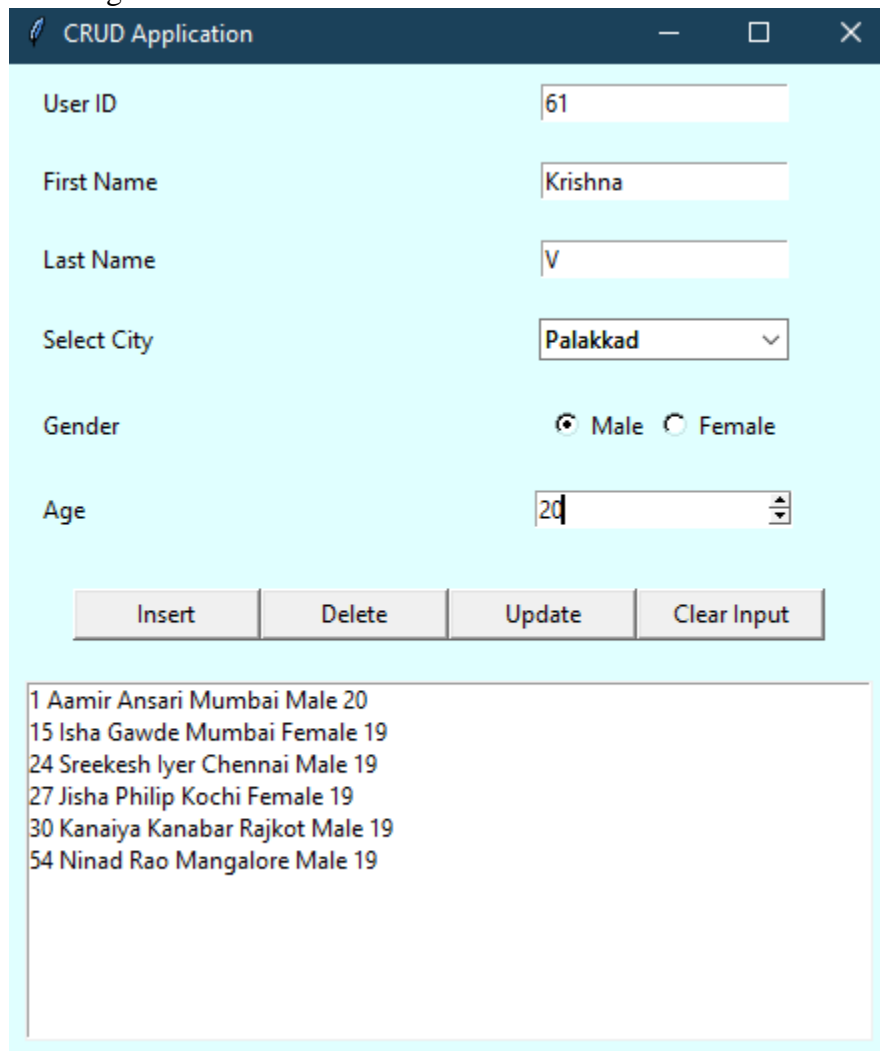
```
frame1 = Frame(app, bg="light cyan")
frame1.grid(row=6,column=0,columnspan=4,pady=20)
add_btn = Button(frame1, text='Insert', width=12, command=add_item)
add_btn.pack(side=LEFT)
remove_btn = Button(frame1, text='Delete', width=12, command=remove_item)
remove_btn.pack(side=LEFT)
update_btn = Button(frame1, text='Update', width=12, command=update_item)
update_btn.pack(side=LEFT)
clear_btn = Button(frame1, text='Clear Input', width=12, command=clear_text)
clear_btn.pack(side=LEFT)
```

```
listbox.delete(0, END)
for row in db.fetch():
    listbox.insert(END, row)
```

```
app.mainloop()
```

### Output:

Inserting:



User ID	First Name	Last Name	City	Gender	Age
1	Aamir	Ansari	Mumbai	Male	20
15	Isha	Gawde	Mumbai	Female	19
24	Sreekesh	Iyer	Chennai	Male	19
27	Jisha	Philip	Kochi	Female	19
30	Kanaiya	Kanabar	Rajkot	Male	19
54	Ninad	Rao	Mangalore	Male	19

Reading and Viewing:

CRUD Application

User ID

27

First Name

Jisha

Last Name

Philip

Select City

Kochi

Gender

☐ Male ☒ Female

Age

19

Insert

Delete

Update

Clear Input

1 Aamir Ansari Mumbai Male 20

15 Isha Gawde Mumbai Female 19

24 Sreelesh Iyer Chennai Male 19

27 Jisha Philip Kochi Female 19

30 Kanaiya Kanabar Rajkot Male 19

54 Ninad Rao Mangalore Male 19

61 Krishna V Palakkad Male 20

Updating:

CRUD Application

User ID

27

First Name

Jisha

Last Name

Philip

Select City

Thrissur

Gender

☐ Male ☒ Female

Age

19

Insert

Delete

Update

Clear Input

1 Aamir Ansari Mumbai Male 20  
15 Isha Gawde Mumbai Female 19  
24 Sreekesh Iyer Chennai Male 19  
27 Jisha Philip Kochi Female 19  
30 Kanaiya Kanabar Rajkot Male 19  
54 Ninad Rao Mangalore Male 19  
61 Krishna V Palakkad Male 20



CRUD Application

User ID

27

First Name

Jisha

Last Name

Philip

Select City

Thrissur

Gender

☐ Male ☒ Female

Age

19

Insert

Delete

Update

Clear Input

1 Aamir Ansari Mumbai Male 20

15 Isha Gawde Mumbai Female 19

24 Sreekesh Iyer Chennai Male 19

27 Jisha Philip Thrissur Female 19

30 Kanaiya Kanabar Rajkot Male 19

54 Ninad Rao Mangalore Male 19

61 Krishna V Palakkad Male 20

Deleting:

CRUD Application

User ID

1

First Name

Aamir

Last Name

Ansari

Select City

Mumbai

Gender

☒ Male ☐ Female

Age

20

Insert

Delete

Update

Clear Input

1 Aamir Ansari Mumbai Male 20

15 Isha Gawde Mumbai Female 19

24 Sreekesh Iyer Chennai Male 19

27 Jisha Philip Thrissur Female 19

30 Kanaiya Kanabar Rajkot Male 19

54 Ninad Rao Mangalore Male 19

61 Krishna V Palakkad Male 20

CRUD Application

User ID

First Name

Last Name

Select City

Gender

☒ Male ☐ Female

Age

Insert

Delete

Update

Clear Input

15 Isha Gawde Mumbai Female 19

24 Sreekesh Iyer Chennai Male 19

27 Jisha Philip Thrissur Female 19

30 Kanaiya Kanabar Rajkot Male 19

54 Ninad Rao Mangalore Male 19

61 Krishna V Palakkad Male 20

### **Conclusion:**

Thus, we have understood the basics of the concepts of database connectivity in Python by performing hands-on practical programs on creating, reading, updating and deleting data in database tables.