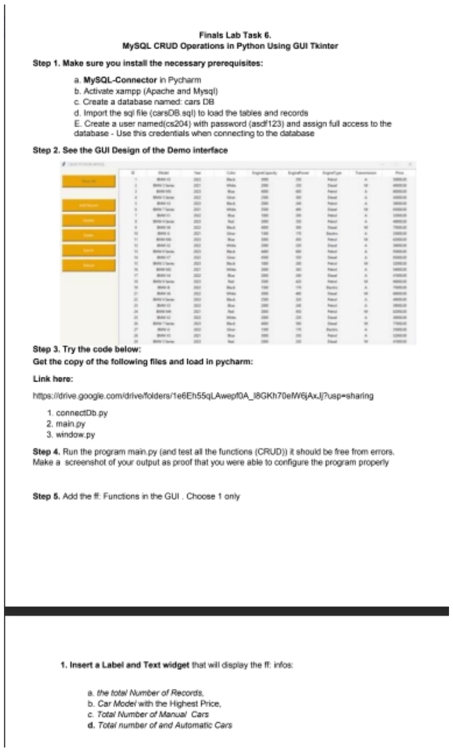
Calma, Michael Vincent, L.

BSCS – C204

Final Lab Task 6:

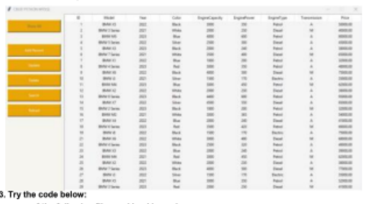


Finals Lab Task 6.
MySQL CRUD Operations in Python Using GUI Tkinter

Step 1. Make sure you install the necessary prerequisites:

a. **MySQL-Connector** in Pycharm
b. Activate xampp (Apache and Mysql)
c. Create a database named: cars DB
d. Import the sql file (carsDB.sql) to load the tables and records
E. Create a user named(cs204) with password (asdf123) and assign full access to the database - Use this credentials when connecting to the database

Step 2. See the GUI Design of the Demo interface



Step 3. Try the code below:
Get the copy of the following files and load in pycharm:

Link here:

https://drive.google.com/drive/folders/1e6Eh55qLAwepf0A_I8GKh70eMW6jAxJj?usp=sharing

1. connectDb.py
2. main.py
3. window.py

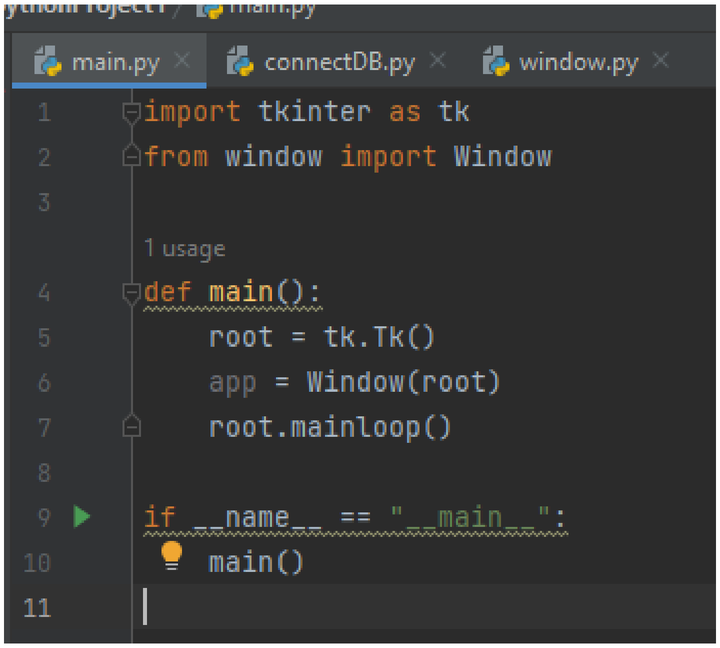
Step 4. Run the program main.py (and test all the functions (CRUD)) it should be free from errors. Make a screenshot of your output as proof that you were able to configure the program properly

Step 5. Add the ff: Functions in the GUI . Choose 1 only

1. **Insert a Label and Text widget** that will display the ff: infos:

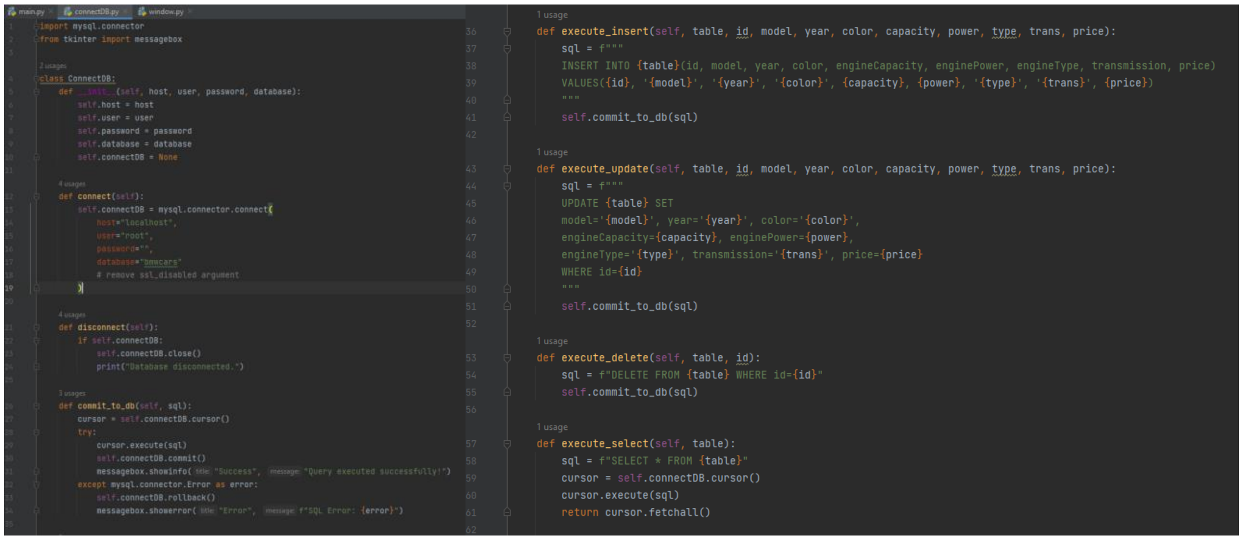
   a. *the total Number of Records,*
   b. *Car Model* with the Highest Price,
   c. *Total Number of Manual Cars*
   d. *Total number of and Automatic Cars*

Code:
main.py:



```python
import tkinter as tk
from window import Window


def main():
    root = tk.Tk()
    app = Window(root)
    root.mainloop()

if __name__ == "__main__":
    main()

```

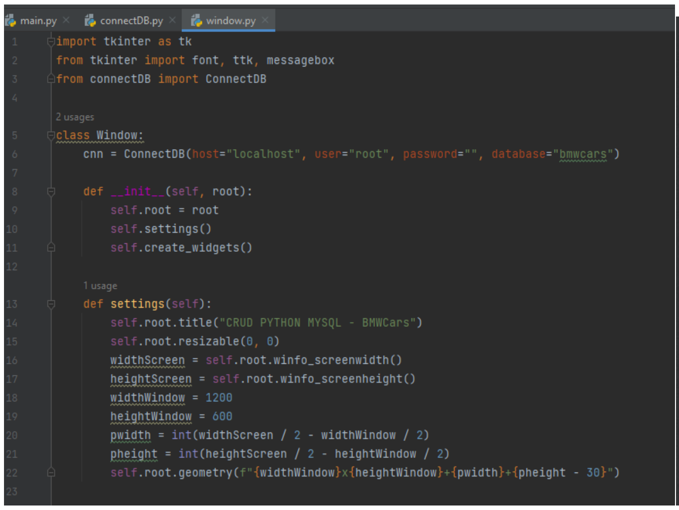
connectDB.py:



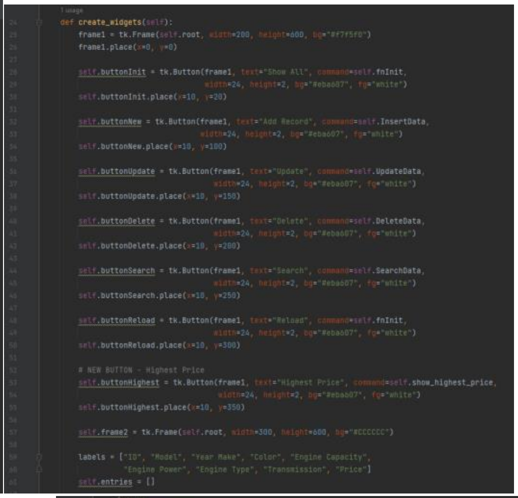
```
main.py  connectDB.py  window.py
import mysql.connector
from tkinter import messagebox

2 usages
class ConnectDB:
    def __init__(self, host, user, password, database):
        self.host = host
        self.user = user
        self.password = password
        self.database = database
        self.connectDB = None

    4 usages
    def connect(self):
        self.connectDB = mysql.connector.connect(
            host="localhost",
            user="root",
            password="",
            database="bmwcars"
            # remove ssl_disabled argument
        )

    4 usages
    def disconnect(self):
        if self.connectDB:
            self.connectDB.close()
            print("Database disconnected.")

    3 usages
    def commit_to_db(self, sql):
        cursor = self.connectDB.cursor()
        try:
            cursor.execute(sql)
            self.connectDB.commit()
            messagebox.showinfo( title: "Success",  message: "Query executed successfully!")
        except mysql.connector.Error as error:
            self.connectDB.rollback()
            messagebox.showerror( title: "Error",  message: f"SQL Error: {error}")
```

```
    1 usage
    def execute_insert(self, table, id, model, year, color, capacity, power, type, trans, price):
        sql = f"""
        INSERT INTO {table}(id, model, year, color, engineCapacity, enginePower, engineType, transmission, price)
        VALUES({id}, '{model}', '{year}', '{color}', {capacity}, {power}, '{type}', '{trans}', {price})
        """
        self.commit_to_db(sql)

    1 usage
    def execute_update(self, table, id, model, year, color, capacity, power, type, trans, price):
        sql = f"""
        UPDATE {table} SET
        model='{model}', year='{year}', color='{color}',
        engineCapacity={capacity}, enginePower={power},
        engineType='{type}', transmission='{trans}', price={price}
        WHERE id={id}
        """
        self.commit_to_db(sql)

    1 usage
    def execute_delete(self, table, id):
        sql = f"DELETE FROM {table} WHERE id={id}"
        self.commit_to_db(sql)

    1 usage
    def execute_select(self, table):
        sql = f"SELECT * FROM {table}"
        cursor = self.connectDB.cursor()
        cursor.execute(sql)
        return cursor.fetchall()
```

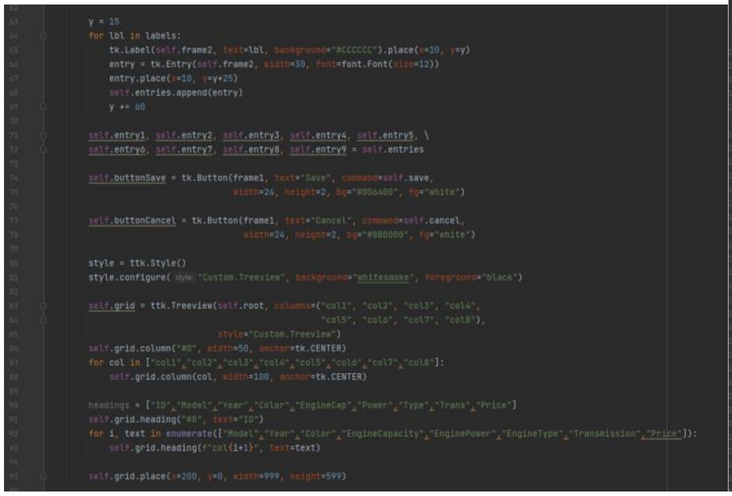
window.py:



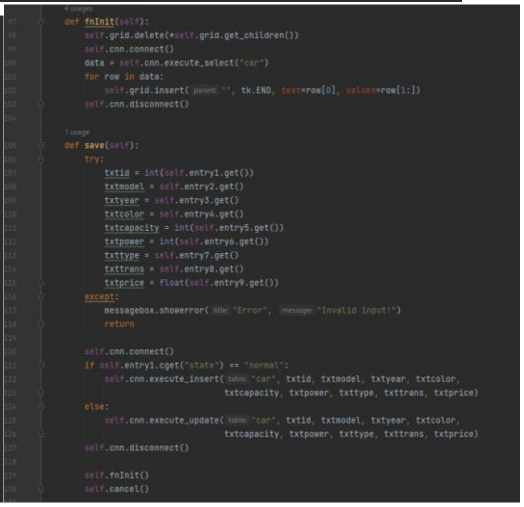
```
main.py  connectDB.py  window.py
import tkinter as tk
from tkinter import font, ttk, messagebox
from connectDB import ConnectDB

2 usages
class Window:
    cnn = ConnectDB(host="localhost", user="root", password="", database="bmwcars")

    def __init__(self, root):
        self.root = root
        self.settings()
        self.create_widgets()

    1 usage
    def settings(self):
        self.root.title("CRUD PYTHON MYSQL - BMWCars")
        self.root.resizable(0, 0)
        widthScreen = self.root.winfo_screenwidth()
        heightScreen = self.root.winfo_screenheight()
        widthWindow = 1200
        heightWindow = 600
        pwidth = int(widthScreen / 2 - widthWindow / 2)
        pheight = int(heightScreen / 2 - heightWindow / 2)
        self.root.geometry(f"{widthWindow}x{heightWindow}+{pwidth}+{pheight - 30}")
```



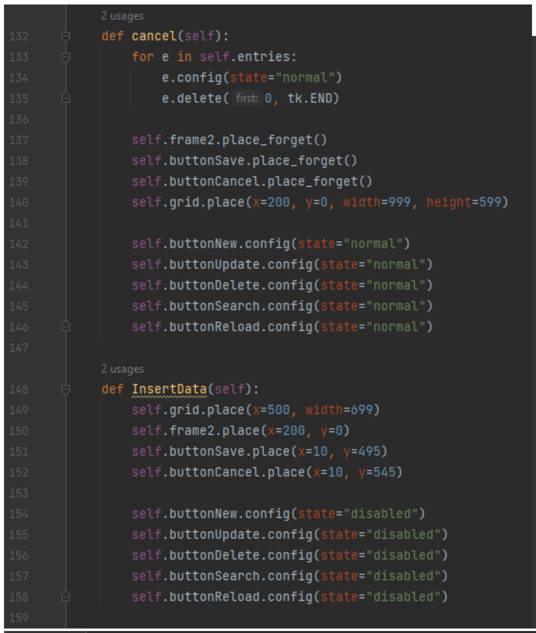
```
    1 usage
    def create_widgets(self):
        frame1 = tk.Frame(self.root, width=200, height=600, bg="#f7f5f0")
        frame1.place(x=0, y=0)

        self.buttonInit = tk.Button(frame1, text="Show All", command=self.fnInit,
                                    width=24, height=2, bg="#eba607", fg="white")
        self.buttonInit.place(x=10, y=20)

        self.buttonNew = tk.Button(frame1, text="Add Record", command=self.InsertData,
                                   width=24, height=2, bg="#eba607", fg="white")
        self.buttonNew.place(x=10, y=100)

        self.buttonUpdate = tk.Button(frame1, text="Update", command=self.UpdateData,
                                      width=24, height=2, bg="#eba607", fg="white")
        self.buttonUpdate.place(x=10, y=150)

        self.buttonDelete = tk.Button(frame1, text="Delete", command=self.DeleteData,
                                      width=24, height=2, bg="#eba607", fg="white")
        self.buttonDelete.place(x=10, y=200)

        self.buttonSearch = tk.Button(frame1, text="Search", command=self.SearchData,
                                      width=24, height=2, bg="#eba607", fg="white")
        self.buttonSearch.place(x=10, y=250)

        self.buttonReload = tk.Button(frame1, text="Reload", command=self.fnInit,
                                      width=24, height=2, bg="#eba607", fg="white")
        self.buttonReload.place(x=10, y=300)

        # NEW BUTTON - Highest Price
        self.buttonHighest = tk.Button(frame1, text="Highest Price", command=self.show_highest_price,
                                       width=24, height=2, bg="#eba607", fg="white")
        self.buttonHighest.place(x=10, y=350)

        self.frame2 = tk.Frame(self.root, width=300, height=600, bg="#CCCCCC")

        labels = ["ID", "Model", "Year Make", "Color", "Engine Capacity",
                  "Engine Power", "Engine Type", "Transmission", "Price"]
        self.entries = []
```



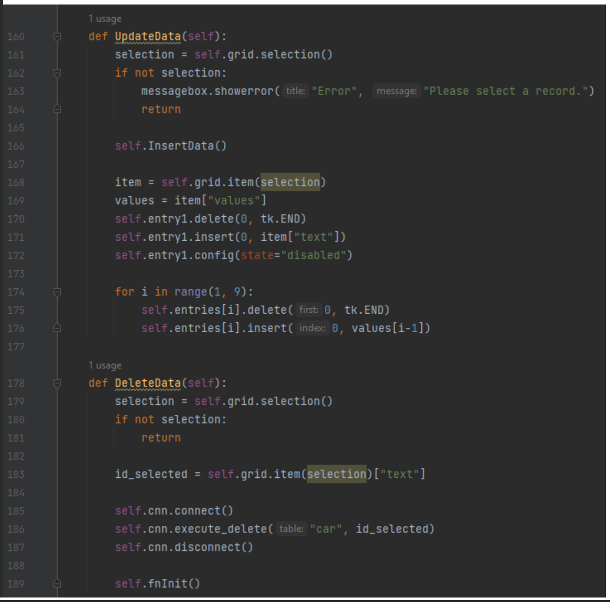
```
        y = 15
        for lbl in labels:
            tk.Label(self.frame2, text=lbl, background="#CCCCCC").place(x=10, y=y)
            entry = tk.Entry(self.frame2, width=30, font=font.Font(size=12))
            entry.place(x=10, y=y+25)
            self.entries.append(entry)
            y += 60

        self.entry1, self.entry2, self.entry3, self.entry4, self.entry5, \
        self.entry6, self.entry7, self.entry8, self.entry9 = self.entries

        self.buttonSave = tk.Button(frame1, text="Save", command=self.save,
                                    width=24, height=2, bg="#006400", fg="white")

        self.buttonCancel = tk.Button(frame1, text="Cancel", command=self.cancel,
                                      width=24, height=2, bg="#880000", fg="white")

        style = ttk.Style()
        style.configure( style: "Custom.Treeview", background="whitesmoke", foreground="black")

        self.grid = ttk.Treeview(self.root, columns=("col1", "col2", "col3", "col4",
                                                     "col5", "col6", "col7", "col8"),
                                 style="Custom.Treeview")
        self.grid.column("#0", width=50, anchor=tk.CENTER)
        for col in ["col1","col2","col3","col4","col5","col6","col7","col8"]:
            self.grid.column(col, width=100, anchor=tk.CENTER)

        headings = ["ID","Model","Year","Color","EngineCap","Power","Type","Trans","Price"]
        self.grid.heading("#0", text="ID")
        for i, text in enumerate(["Model","Year","Color","EngineCapacity","EnginePower","EngineType","Transmission","Price"]):
            self.grid.heading(f"col{i+1}", text=text)

        self.grid.place(x=200, y=0, width=999, height=599)
```



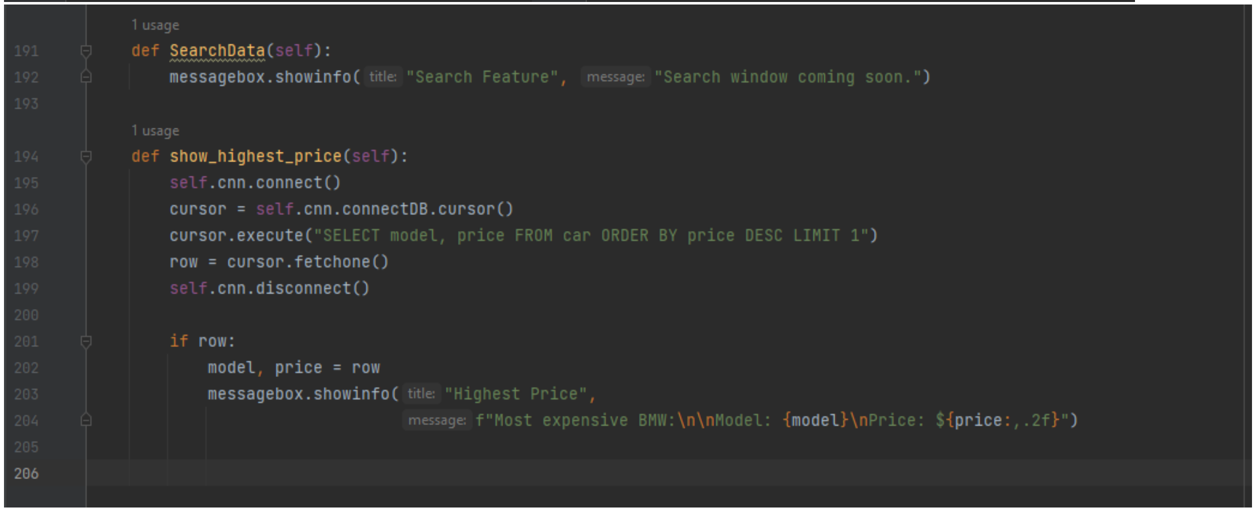
```
    4 usages
    def fnInit(self):
        self.grid.delete(*self.grid.get_children())
        self.cnn.connect()
        data = self.cnn.execute_select("car")
        for row in data:
            self.grid.insert( parent: "", tk.END, text=row[0], values=row[1:])
        self.cnn.disconnect()

    1 usage
    def save(self):
        try:
            txtid = int(self.entry1.get())
            txtmodel = self.entry2.get()
            txtyear = self.entry3.get()
            txtcolor = self.entry4.get()
            txtcapacity = int(self.entry5.get())
            txtpower = int(self.entry6.get())
            txttype = self.entry7.get()
            txttrans = self.entry8.get()
            txtprice = float(self.entry9.get())
        except:
            messagebox.showerror( title: "Error",  message: "Invalid input!")
            return

        self.cnn.connect()
        if self.entry1.cget("state") == "normal":
            self.cnn.execute_insert( table: "car", txtid, txtmodel, txtyear, txtcolor,
                                    txtcapacity, txtpower, txttype, txttrans, txtprice)
        else:
            self.cnn.execute_update( table: "car", txtid, txtmodel, txtyear, txtcolor,
                                    txtcapacity, txtpower, txttype, txttrans, txtprice)
        self.cnn.disconnect()

        self.fnInit()
        self.cancel()
```



```
2 usages
def cancel(self):
    for e in self.entries:
        e.config(state="normal")
        e.delete( first: 0, tk.END)

    self.frame2.place_forget()
    self.buttonSave.place_forget()
    self.buttonCancel.place_forget()
    self.grid.place(x=200, y=0, width=999, height=599)

    self.buttonNew.config(state="normal")
    self.buttonUpdate.config(state="normal")
    self.buttonDelete.config(state="normal")
    self.buttonSearch.config(state="normal")
    self.buttonReload.config(state="normal")

2 usages
def InsertData(self):
    self.grid.place(x=500, width=699)
    self.frame2.place(x=200, y=0)
    self.buttonSave.place(x=10, y=495)
    self.buttonCancel.place(x=10, y=545)

    self.buttonNew.config(state="disabled")
    self.buttonUpdate.config(state="disabled")
    self.buttonDelete.config(state="disabled")
    self.buttonSearch.config(state="disabled")
    self.buttonReload.config(state="disabled")
```

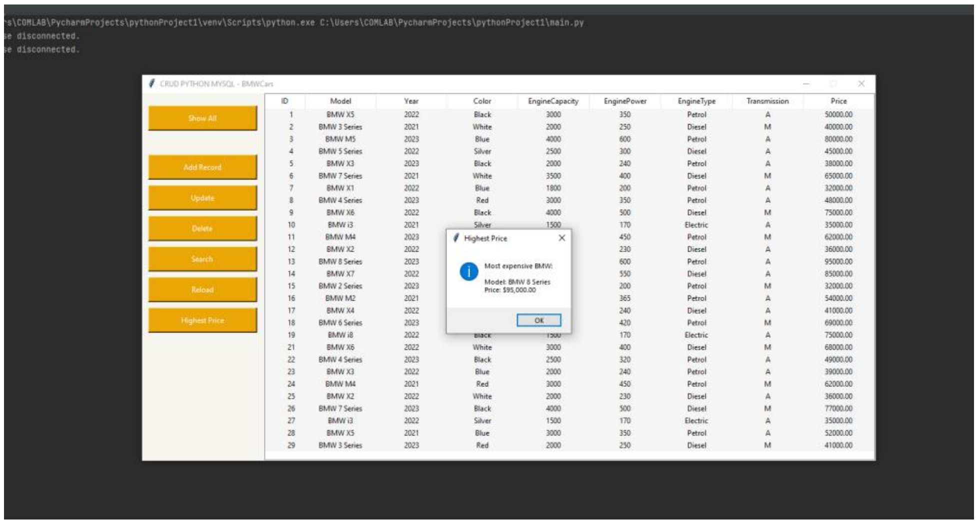


```
1 usage
def UpdateData(self):
    selection = self.grid.selection()
    if not selection:
        messagebox.showerror( title: "Error",  message: "Please select a record.")
        return

    self.InsertData()

    item = self.grid.item(selection)
    values = item["values"]
    self.entry1.delete(0, tk.END)
    self.entry1.insert(0, item["text"])
    self.entry1.config(state="disabled")

    for i in range(1, 9):
        self.entries[i].delete( first: 0, tk.END)
        self.entries[i].insert( index: 0, values[i-1])

1 usage
def DeleteData(self):
    selection = self.grid.selection()
    if not selection:
        return

    id_selected = self.grid.item(selection)["text"]

    self.cnn.connect()
    self.cnn.execute_delete( table: "car", id_selected)
    self.cnn.disconnect()

    self.fnInit()
```



```
1 usage
def SearchData(self):
    messagebox.showinfo( title: "Search Feature",  message: "Search window coming soon.")

1 usage
def show_highest_price(self):
    self.cnn.connect()
    cursor = self.cnn.connectDB.cursor()
    cursor.execute("SELECT model, price FROM car ORDER BY price DESC LIMIT 1")
    row = cursor.fetchone()
    self.cnn.disconnect()

    if row:
        model, price = row
        messagebox.showinfo( title: "Highest Price",
                             message: f"Most expensive BMW:\n\nModel: {model}\nPrice: ${price:,.2f}")

```

Output:



```
s\COMLAB\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:\Users\COMLAB\PycharmProjects\pythonProject1\main.py
e disconnected.
e disconnected.
```

CRUD PYTHON MYSQL - BMWCars

ID	Model	Year	Color	EngineCapacity	EnginePower	EngineType	Transmission	Price
1	BMW X5	2022	Black	3000	350	Petrol	A	50000.00
2	BMW 3 Series	2021	White	2000	250	Diesel	M	40000.00
3	BMW M5	2023	Blue	4000	600	Petrol	A	80000.00
4	BMW 5 Series	2022	Silver	2500	300	Diesel	A	45000.00
5	BMW X3	2023	Black	2000	240	Petrol	A	38000.00
6	BMW 7 Series	2021	White	3500	400	Diesel	M	65000.00
7	BMW X1	2022	Blue	1800	200	Petrol	A	32000.00
8	BMW 4 Series	2023	Red	3000	350	Petrol	A	48000.00
9	BMW X6	2022	Black	4000	500	Diesel	M	75000.00
10	BMW i3	2021	Silver	1500	170	Electric	A	35000.00
11	BMW M4	2023	[illegible]	[illegible]	450	Petrol	M	62000.00
12	BMW X2	2022	[illegible]	[illegible]	230	Diesel	A	36000.00
13	BMW 8 Series	2023	[illegible]	[illegible]	600	Petrol	A	95000.00
14	BMW X7	2022	[illegible]	[illegible]	550	Diesel	A	85000.00
15	BMW 2 Series	2023	[illegible]	[illegible]	200	Petrol	M	32000.00
16	BMW M2	2021	[illegible]	[illegible]	365	Petrol	A	54000.00
17	BMW X4	2022	[illegible]	[illegible]	240	Diesel	A	41000.00
18	BMW 6 Series	2023	[illegible]	[illegible]	420	Petrol	M	69000.00
19	BMW i8	2022	Black	1500	170	Electric	A	75000.00
21	BMW X6	2022	White	3000	400	Diesel	M	68000.00
22	BMW 4 Series	2023	Black	2500	320	Petrol	A	49000.00
23	BMW X3	2022	Blue	2000	240	Petrol	A	39000.00
24	BMW M4	2021	Red	3000	450	Petrol	M	62000.00
25	BMW X2	2022	White	2000	230	Diesel	A	36000.00
26	BMW 7 Series	2023	Black	4000	500	Diesel	M	77000.00
27	BMW i3	2022	Silver	1500	170	Electric	A	35000.00
28	BMW X5	2021	Blue	3000	350	Petrol	A	52000.00
29	BMW 3 Series	2023	Red	2000	250	Diesel	M	41000.00

Highest Price

Most expensive BMW:

Model: BMW 8 Series
Price: $95,000.00