Asuar, Blaine M

C204
Final Lab Task 6 : Mysql and Ttkinter GUI

Main.py

```
import tkinter as tk
import window

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

if __name__ == "__main__":
    main()
```

Window.py

```
import tkinter as tk
from tkinter import font, messagebox, ttk
from connectDB import ConnectDB

class Window:
    cnn = ConnectDB(host="localhost", user="cs2044", password="asdf1234",
database="CarsDb")

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

    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}")

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

        buttons = [
            ("Show All", self.fnInit, 20),
            ("Add Record", self.InsertData, 100),
            ("Update", self.UpdateData, 150),
```

```
            ("Delete", self.DeleteData, 200),
            ("Search", self.SearchData, 250),
            ("Reload", self.fnInit, 300),
            ("Total Cars", self.total_cars, 350),
            ("Automatic Cars", self.total_automatic, 400)
        ]

        for text, cmd, y in buttons:
            tk.Button(frame1, text=text, command=cmd, width=24, height=2,
bg="#eba607", fg="white").place(x=10, y=y)

        # Right Frame with entries
        self.frame2 = tk.Frame(self.root, width=300, height=600,
bg="#CCCCCC")
        labels = ["ID", "Model", "Year Make", "Color", "Engine Capacity",
"Engine Motor",
                  "Engine Type", "Transmission Type", "Price"]
        self.entries = []
        y = 15
        for text in labels:
            tk.Label(self.frame2, text=text, bg="#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 += 65

        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")
        self.buttonCancel = tk.Button(frame1, text="Cancel",
command=self.cancel, width=24, height=2, bg="#8B0000")

        # Treeview
        style = ttk.Style()
        style.configure("Custom.Treeview", background="whitesmoke",
foreground="black")
        self.grid = ttk.Treeview(self.root, columns=[f"col{i}" for i in
range(1, 9)], style="Custom.Treeview")
        self.grid.column("#0", width=50, anchor=tk.CENTER)
        for i in range(1, 9):
            self.grid.column(f"col{i}", width=100, anchor=tk.CENTER)

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

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

    # --- CRUD and helper methods ---
    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("", tk.END, text=row[0], values=row[1:])
        self.cnn.disconnect()

    def cancel(self):
        self.buttonSave.place_forget()
        self.buttonCancel.place_forget()
        self.grid.place(x=200, y=0, width=999, height=599)
        for e in self.entries:
            e.config(state="normal")
            e.delete(0, tk.END)
        for b in [self.buttonUpdate, self.buttonNew, self.buttonDelete,
self.buttonSearch, self.buttonReload]:
            b.config(state="normal")

    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 ValueError:
            messagebox.showerror("Error", "All fields must be valid and
filled.")
            return

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

    def InsertData(self):
        self.frame2.place(x=200, y=0)
        self.grid.place(x=500, y=0, width=699, height=599)
        self.buttonSave.place(x=10, y=500)
        self.buttonCancel.place(x=10, y=550)
        self.disable_actions()

    def UpdateData(self):
        selection = self.grid.selection()
        if not selection:
            messagebox.showerror("Error", "Select a row to update.")
```

```
            return
        item = self.grid.item(selection)
        id_sel = item['text']
        values = item['values']
        self.frame2.place(x=200, y=0)
        self.grid.place(x=500, y=0, width=699, height=599)
        self.buttonSave.place(x=10, y=500)
        self.buttonCancel.place(x=10, y=550)
        self.disable_actions()
        entries = [id_sel] + list(values)
        for entry, data in zip(self.entries, entries):
            entry.insert(0, data)
        self.entry1.config(state="disabled")

    def DeleteData(self):
        selection = self.grid.selection()
        if not selection:
            messagebox.showerror("Error", "Select a row to delete.")
            return
        id_sel = self.grid.item(selection)['text']
        self.cnn.connect()
        self.cnn.execute_delete("car", id_sel)
        self.cnn.disconnect()
        self.fnInit()

    def disable_actions(self):
        for b in [self.buttonUpdate, self.buttonNew, self.buttonDelete,
self.buttonSearch, self.buttonReload]:
            b.config(state="disabled")

    # Placeholder methods (implement in ConnectDB)
    def total_cars(self):
        self.cnna.connect()
        total = self.cnn.count_all("car")
        self.cnn.disconnect()
        messagebox.showinfo("Total Cars", f"Total cars: {total}")

    def total_automatic(self):
        self.cnn.connect()
        total = self.cnn.count_automatic("car")
        self.cnn.disconnect()
        messagebox.showinfo("Automatic Cars", f"Automatic cars: {total}")

    def SearchData(self):
        # Add your search implementation here
        pass
```

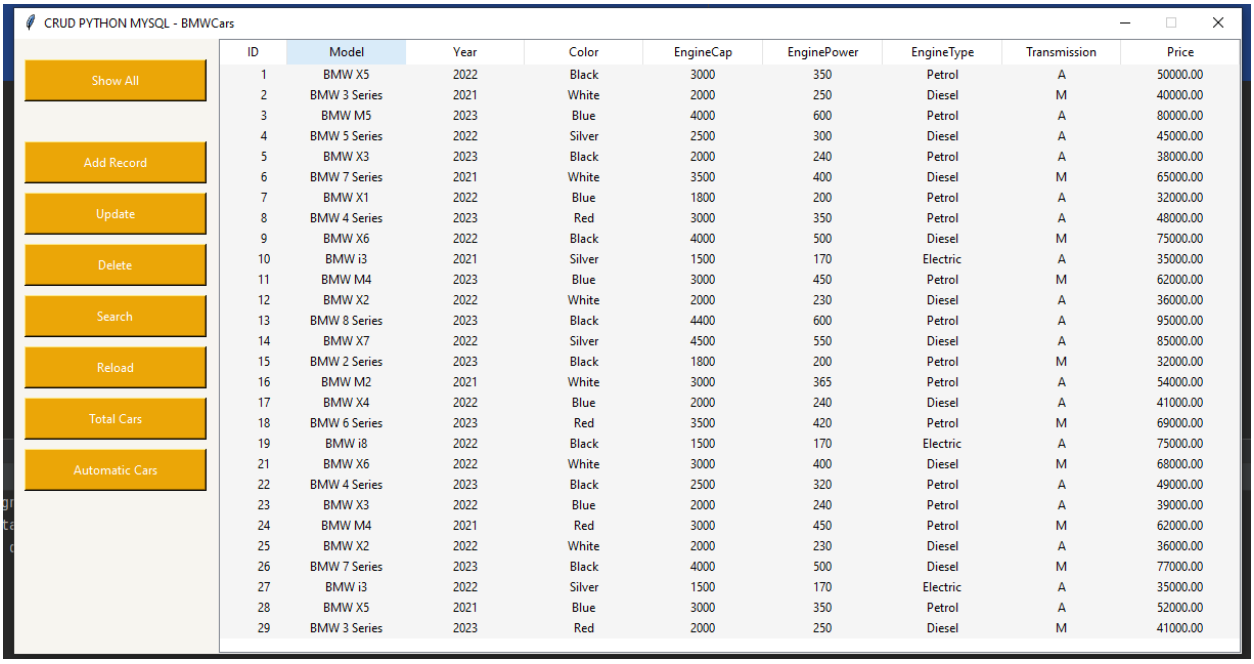
Connector

```
import mysql.connector
from tkinter import messagebox

class ConnectDB:
    def __init__(self, host, user, password, database):
        self.host = host
        self.user = user
```

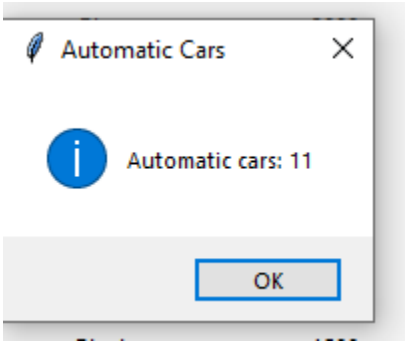
```
        self.password = password
        self.database = database
        self.connectDB = None

    def connect(self):
        try:
            self.connectDB = mysql.connector.connect(
                host=self.host,
                user=self.user,
                password=self.password,
                database=self.database,
                ssl_disabled=True
            )
            print("Successfully connection to the database!")
        except mysql.connector.Error as error:
            print("Something went wrong connecting to the database: ", error)

    def disconnect(self):
        if self.connectDB:
            self.connectDB.close()
            print("Successfully disconnecting to the database!")

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

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

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

    def commit_to_db(self, sql):
        cursor = self.connectDB.cursor()
        try:
            cursor.execute(sql)
            self.connectDB.commit()
            print("Query successfully executed")
            messagebox.showinfo("Successfully", "Query successfully executed.
Good Work!")
        except mysql.connector.Error as error:
            self.connectDB.rollback()
            print("Error executing the query:", error)
            messagebox.showerror("Error", "Duplicate ID entry, please try
again!")

    def execute_select(self, table):
```

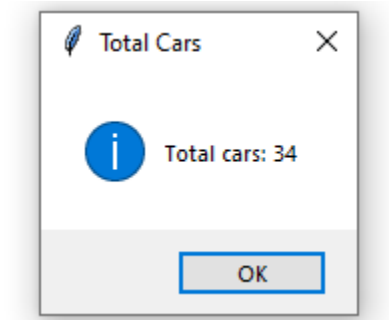
```
        sql = f"SELECT * FROM {table}"
        cursor = self.connectDB.cursor()
        try:
            cursor.execute(sql)
            rows = cursor.fetchall()
            return rows
        except mysql.connector.Error as error:
            print("Error executing the query:", error)
            return []

    def __str__(self):
        data = self.execute_select("car")
        aux = ""
        for row in data:
            aux += str(row) + "\n"
        return aux

    def get_highest_price_car(self):
        try:
            cursor = self.connectDB.cursor()
            cursor.execute("SELECT * FROM car ORDER BY price DESC LIMIT 1")
            return cursor.fetchone()
        except mysql.connector.Error as error:
            print("Error fetching highest price:", error)
            return None

    def count_all(self, table):
        cursor = self.connectDB.cursor()
        cursor.execute(f"SELECT COUNT(*) FROM {table}")
        return cursor.fetchone()[0]

    def count_automatic(self, table):
        cursor = self.connectDB.cursor()
        cursor.execute(f"SELECT COUNT(*) FROM {table} WHERE
transmission='M'")
        return cursor.fetchone()[0]
```

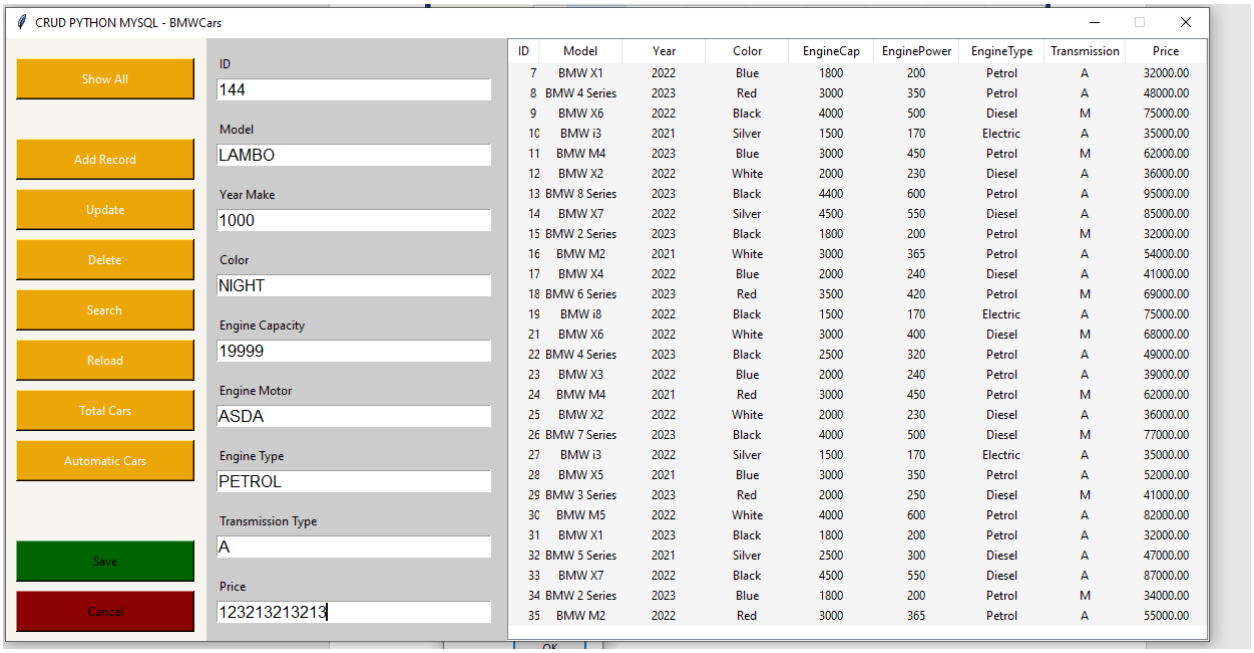
Sample Code



ID	Model	Year	Color	EngineCap	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	Blue	3000	450	Petrol	M	62000.00
12	BMW X2	2022	White	2000	230	Diesel	A	36000.00
13	BMW 8 Series	2023	Black	4400	600	Petrol	A	95000.00
14	BMW X7	2022	Silver	4500	550	Diesel	A	85000.00
15	BMW 2 Series	2023	Black	1800	200	Petrol	M	32000.00
16	BMW M2	2021	White	3000	365	Petrol	A	54000.00
17	BMW X4	2022	Blue	2000	240	Diesel	A	41000.00
18	BMW 6 Series	2023	Red	3500	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






CRUD PYTHON MYSQL - BMWCars
Show All
Add Record
Update
Delete
Search
Reload
Total Cars
Automatic Cars
Save
Cancel
ID
144
Model
LAMBO
Year Make
1000
Color
NIGHT
Engine Capacity
19999
Engine Motor
ASDA
Engine Type
PETROL
Transmission Type
A
Price
123213213213
ID Model Year Color EngineCap EnginePower EngineType Transmission Price
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 Blue 3000 450 Petrol M 62000.00
12 BMW X2 2022 White 2000 230 Diesel A 36000.00
13 BMW 8 Series 2023 Black 4400 600 Petrol A 95000.00
14 BMW X7 2022 Silver 4500 550 Diesel A 85000.00
15 BMW 2 Series 2023 Black 1800 200 Petrol M 32000.00
16 BMW M2 2021 White 3000 365 Petrol A 54000.00
17 BMW X4 2022 Blue 2000 240 Diesel A 41000.00
18 BMW 6 Series 2023 Red 3500 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
30 BMW M5 2022 White 4000 600 Petrol A 82000.00
31 BMW X1 2023 Black 1800 200 Petrol A 32000.00
32 BMW 5 Series 2021 Silver 2500 300 Diesel A 47000.00
33 BMW X7 2022 Black 4500 550 Diesel A 87000.00
34 BMW 2 Series 2023 Blue 1800 200 Petrol M 34000.00
35 BMW M2 2022 Red 3000 365 Petrol A 55000.00