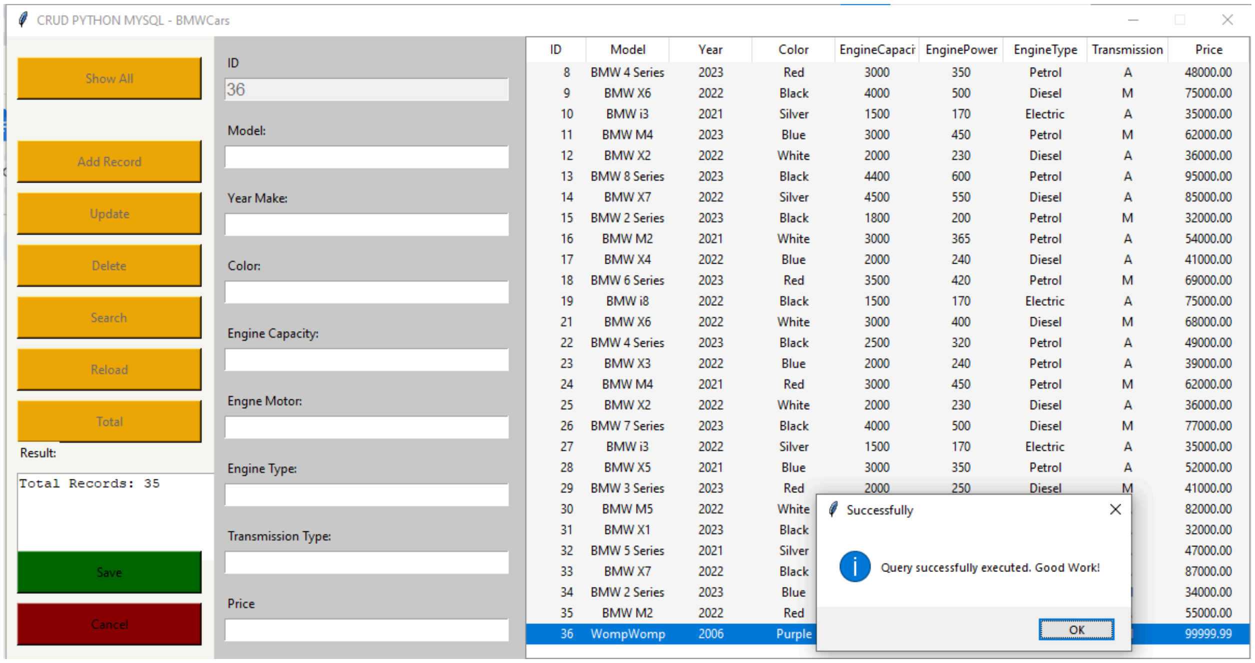
ADD



ID: 36

Model:

Year Make:

Color:

Engine Capacity:

Engne Motor:

Engine Type:

Transmission Type:

Price

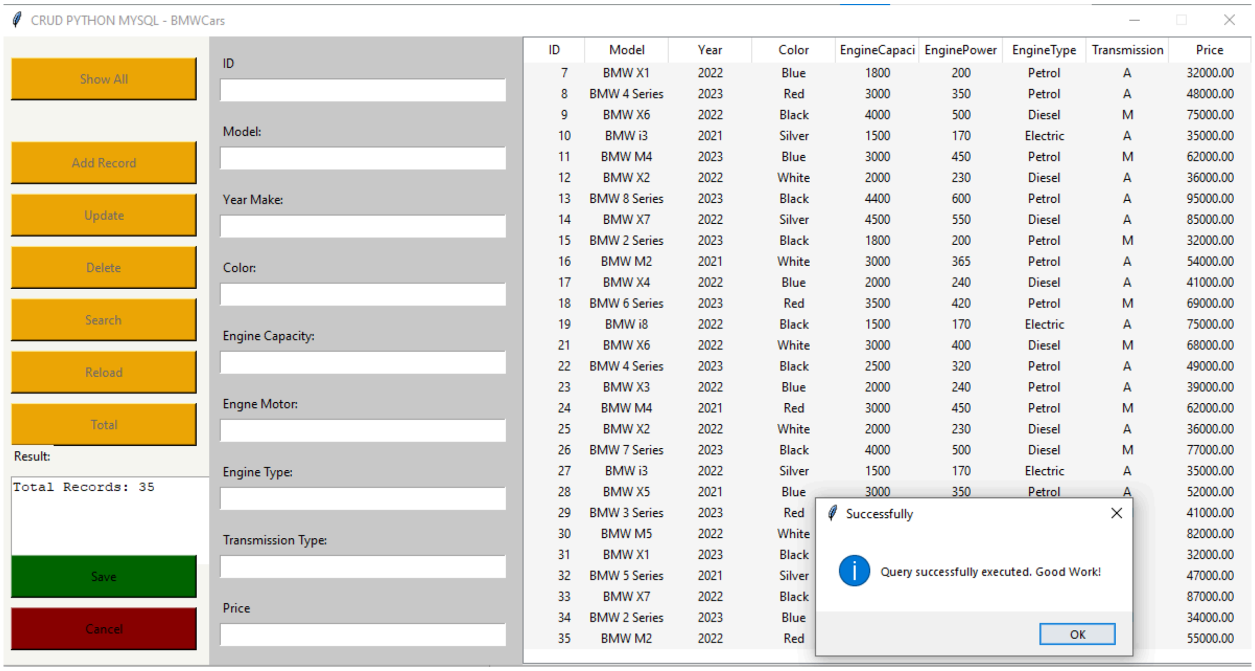
Result:

```
Total Records: 35
```

ID	Model	Year	Color	EngineCapaci	EnginePower	EngineType	Transmission	Price
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					82000.00
31	BMW X1	2023	Black					32000.00
32	BMW 5 Series	2021	Silver					47000.00
33	BMW X7	2022	Black					87000.00
34	BMW 2 Series	2023	Blue					34000.00
35	BMW M2	2022	Red					55000.00
36	WompWomp	2006	Purple					99999.99

Successfully

Query successfully executed. Good Work!



ID

Model:

Year Make:

Color:

Engine Capacity:

Engne Motor:

Engine Type:

Transmission Type:

Price

Result:

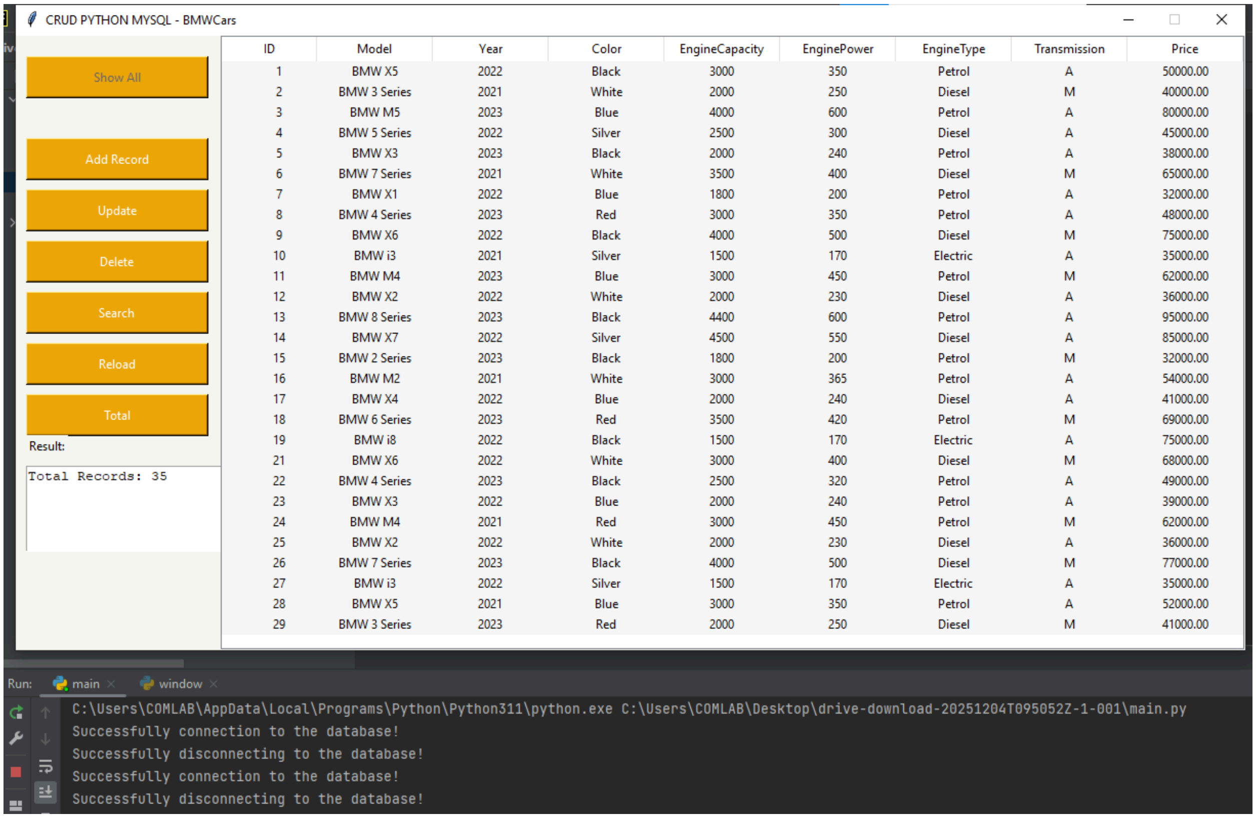
```
Total Records: 35
```

ID	Model	Year	Color	EngineCapaci	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					41000.00
30	BMW M5	2022	White					82000.00
31	BMW X1	2023	Black					32000.00
32	BMW 5 Series	2021	Silver					47000.00
33	BMW X7	2022	Black					87000.00
34	BMW 2 Series	2023	Blue					34000.00
35	BMW M2	2022	Red					55000.00

Successfully

Query successfully executed. Good Work!

UPDATE



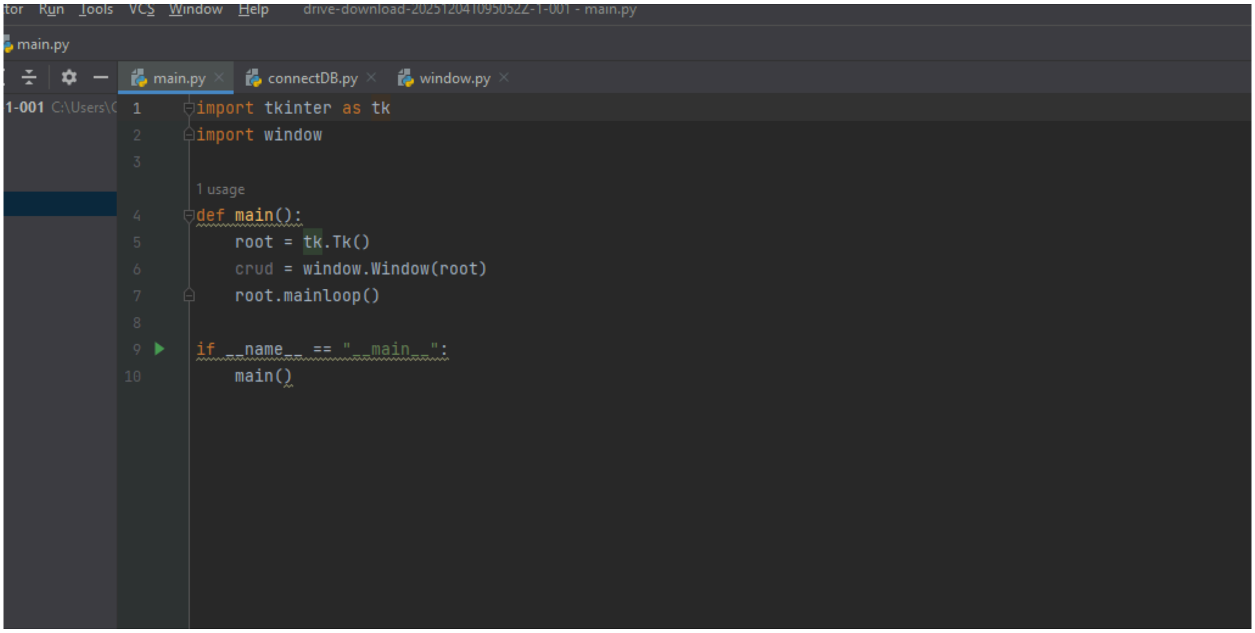
Result:

```
Total Records: 35
```

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	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

```
C:\Users\COMLAB\AppData\Local\Programs\Python\Python311\python.exe C:\Users\COMLAB\Desktop\drive-download-20251204T095052Z-1-001\main.py
Successfully connection to the database!
Successfully disconnecting to the database!
Successfully connection to the database!
Successfully disconnecting to the database!
```

Main



tor Run Tools VCS Window Help drive-download-20251204T095052Z-1-001 - main.py
main.py
main.py
connectDB.py
window.py
1-001 C:\Users\C
1 usage

```python
import tkinter as tk
import window

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

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

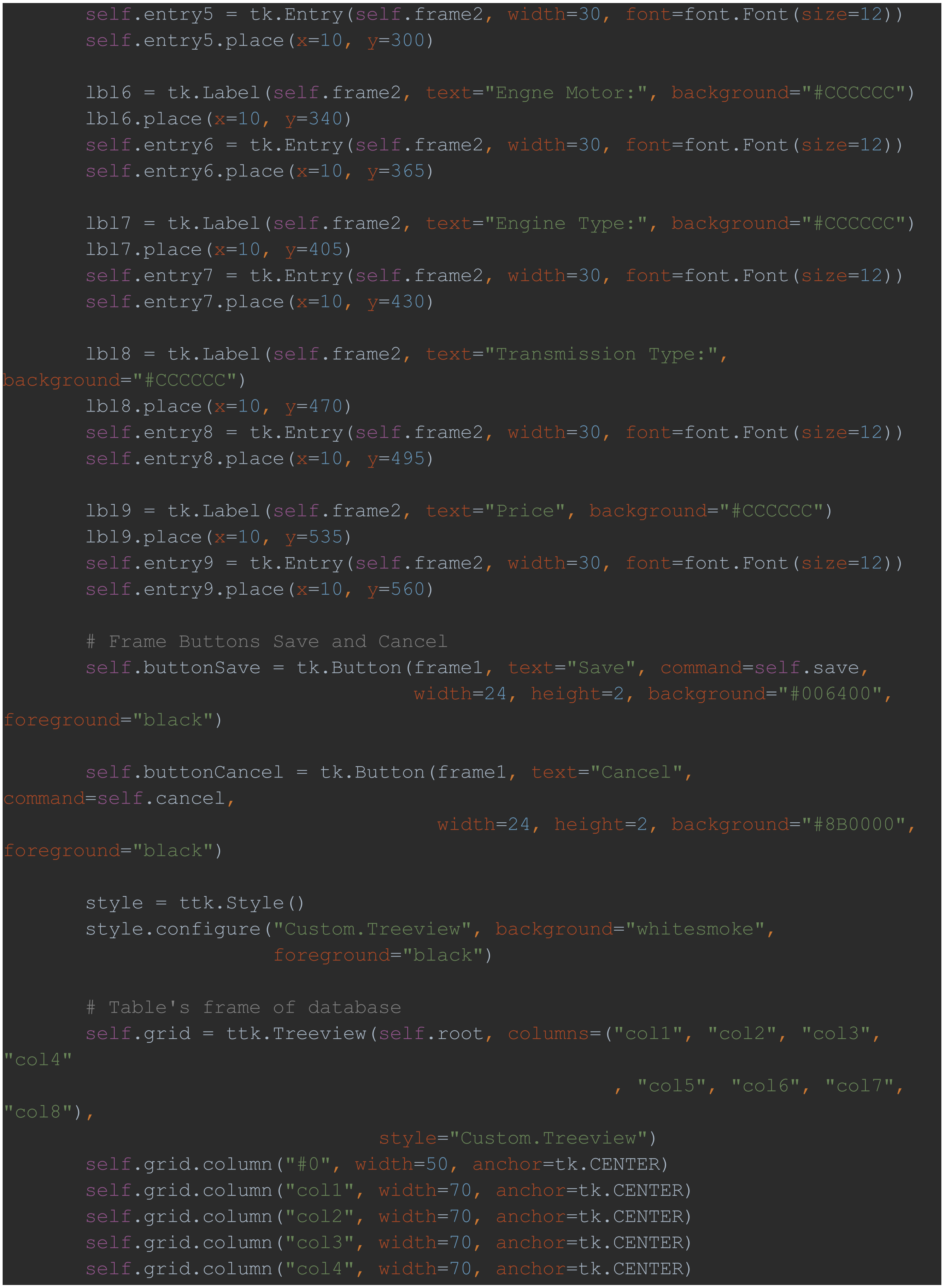
connectDB

```
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()
```

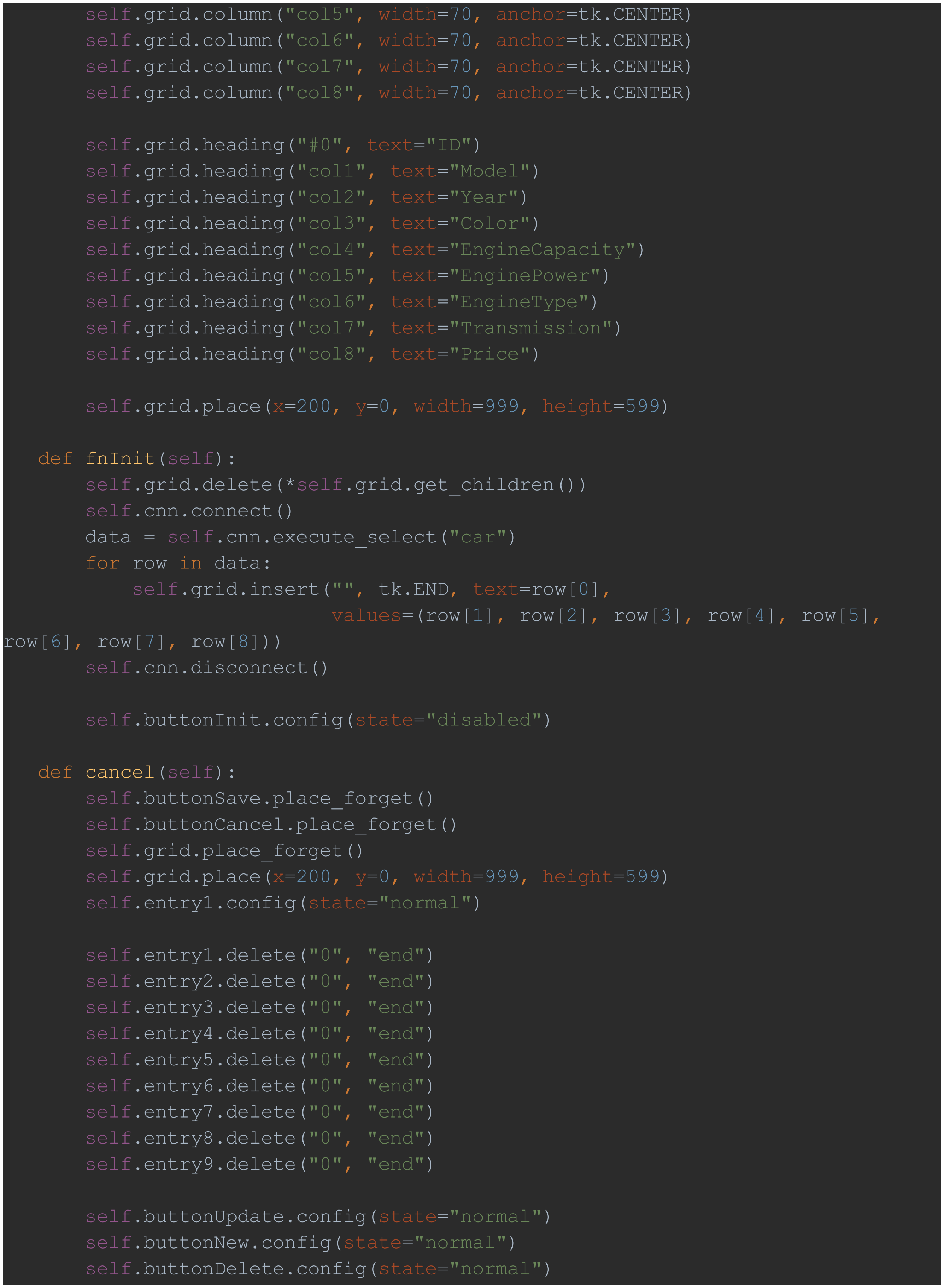
```
        self.commit_to_db(sql)

    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 au
```

window.py

```
import tkinter as tk
from tkinter import font
from tkinter import ttk
from connectDB import *
import os
from tkinter import messagebox

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

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

   def settings(self):
       self.root.title("CRUD PYTHON MYSQL - BMWCars")  # Title of the window
       self.root.resizable(0, 0)  # Disable to resizable the window

      # Size and position of the window
       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):
       # FRAME BUTTONS
       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, background="#eba607",
                                   foreground="white")
       self.buttonInit.place(x=10, y=20)

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

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

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

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

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

        self.buttonTotal = tk.Button(frame1, text="Total",
command=self.show_total_records,
                                         width=24, height=2, background="#eba607",
foreground="white")
        self.buttonTotal.place(x=10, y=350)

        self.lblResult = tk.Label(frame1, text="Result:", bg="#f7f5f0",
fg="black")
        self.lblResult.place(x=10, y=390)

        self.txtResult = tk.Text(frame1, width=24, height=5)
        self.txtResult.place(x=10, y=420)

        # FRAME INPUT 25 + 40
        self.frame2 = tk.Frame(self.root, width=300, height=600, bg="#CCCCCC")

        lbl1 = tk.Label(self.frame2, text="ID", background="#CCCCCC")
        lbl1.place(x=10, y=15)
        self.entry1 = tk.Entry(self.frame2, width=30, font=font.Font(size=12))
        self.entry1.place(x=10, y=40)

        lbl2 = tk.Label(self.frame2, text="Model:", background="#CCCCCC")
        lbl2.place(x=10, y=80)
        self.entry2 = tk.Entry(self.frame2, width=30, font=font.Font(size=12))
        self.entry2.place(x=10, y=105)

        lbl3 = tk.Label(self.frame2, text="Year Make:", background="#CCCCCC")
        lbl3.place(x=10, y=145)
        self.entry3 = tk.Entry(self.frame2, width=30, font=font.Font(size=12))
        self.entry3.place(x=10, y=170)

        lbl4 = tk.Label(self.frame2, text="Color:", background="#CCCCCC")
        lbl4.place(x=10, y=210)
        self.entry4 = tk.Entry(self.frame2, width=30, font=font.Font(size=12))
        self.entry4.place(x=10, y=235)

        lbl5 = tk.Label(self.frame2, text="Engine Capacity:",
background="#CCCCCC")
        lbl5.place(x=10, y=275)
```

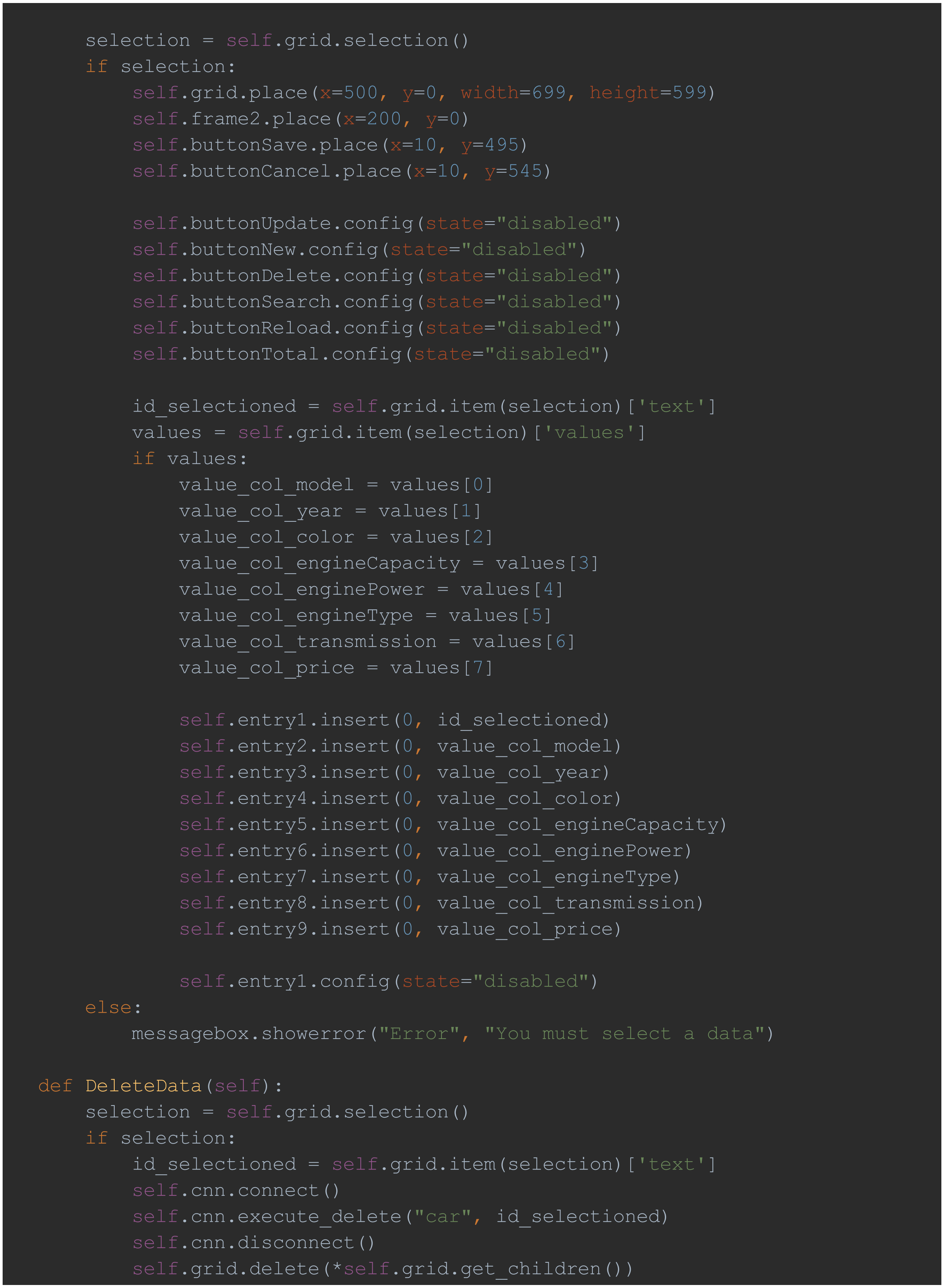


```
        self.entry5 = tk.Entry(self.frame2, width=30, font=font.Font(size=12))
        self.entry5.place(x=10, y=300)

        lbl6 = tk.Label(self.frame2, text="Engne Motor:", background="#CCCCCC")
        lbl6.place(x=10, y=340)
        self.entry6 = tk.Entry(self.frame2, width=30, font=font.Font(size=12))
        self.entry6.place(x=10, y=365)

        lbl7 = tk.Label(self.frame2, text="Engine Type:", background="#CCCCCC")
        lbl7.place(x=10, y=405)
        self.entry7 = tk.Entry(self.frame2, width=30, font=font.Font(size=12))
        self.entry7.place(x=10, y=430)

        lbl8 = tk.Label(self.frame2, text="Transmission Type:",
background="#CCCCCC")
        lbl8.place(x=10, y=470)
        self.entry8 = tk.Entry(self.frame2, width=30, font=font.Font(size=12))
        self.entry8.place(x=10, y=495)

        lbl9 = tk.Label(self.frame2, text="Price", background="#CCCCCC")
        lbl9.place(x=10, y=535)
        self.entry9 = tk.Entry(self.frame2, width=30, font=font.Font(size=12))
        self.entry9.place(x=10, y=560)

        # Frame Buttons Save and Cancel
        self.buttonSave = tk.Button(frame1, text="Save", command=self.save,
                                   width=24, height=2, background="#006400",
foreground="black")

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

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

        # Table's frame of database
        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)
        self.grid.column("col1", width=70, anchor=tk.CENTER)
        self.grid.column("col2", width=70, anchor=tk.CENTER)
        self.grid.column("col3", width=70, anchor=tk.CENTER)
        self.grid.column("col4", width=70, anchor=tk.CENTER)
```



```
       self.grid.column("col5", width=70, anchor=tk.CENTER)
       self.grid.column("col6", width=70, anchor=tk.CENTER)
       self.grid.column("col7", width=70, anchor=tk.CENTER)
       self.grid.column("col8", width=70, anchor=tk.CENTER)

       self.grid.heading("#0", text="ID")
       self.grid.heading("col1", text="Model")
       self.grid.heading("col2", text="Year")
       self.grid.heading("col3", text="Color")
       self.grid.heading("col4", text="EngineCapacity")
       self.grid.heading("col5", text="EnginePower")
       self.grid.heading("col6", text="EngineType")
       self.grid.heading("col7", text="Transmission")
       self.grid.heading("col8", text="Price")

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

   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], row[2], row[3], row[4], row[5],
row[6], row[7], row[8]))
       self.cnn.disconnect()

       self.buttonInit.config(state="disabled")

   def cancel(self):
       self.buttonSave.place_forget()
       self.buttonCancel.place_forget()
       self.grid.place_forget()
       self.grid.place(x=200, y=0, width=999, height=599)
       self.entry1.config(state="normal")

       self.entry1.delete("0", "end")
       self.entry2.delete("0", "end")
       self.entry3.delete("0", "end")
       self.entry4.delete("0", "end")
       self.entry5.delete("0", "end")
       self.entry6.delete("0", "end")
       self.entry7.delete("0", "end")
       self.entry8.delete("0", "end")
       self.entry9.delete("0", "end")

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

```
        self.buttonSearch.config(state="normal")
        self.buttonReload.config(state="normal")
        self.buttonTotal.config(state="normal")

    def save(self):

        txtid = 0
        txtmodel = ""
        txtyear = ""
        txtcolor = ""
        txtcapacity = 0
        txtpower = 0
        txttype = ""
        txttrans = ""
        txtprice = 0.0

        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 as e:
            print("All fields must be filled in")
        finally:
            self.entry1.delete("0", "end")
            self.entry2.delete("0", "end")
            self.entry3.delete("0", "end")
            self.entry4.delete("0", "end")
            self.entry5.delete("0", "end")
            self.entry6.delete("0", "end")
            self.entry7.delete("0", "end")
            self.entry8.delete("0", "end")
            self.entry9.delete("0", "end")

        self.cnn.connect()

        if txtid == "" or txtmodel == "" or txtyear == "" or txtcolor == "" or
txtcapacity == "" or txtpower == "" or txttype == "" or txttrans == "" or
txtprice == "":
            messagebox.showerror("Error", "All fields must be filled in.")
        else:
            if self.entry1.cget("state") == "normal":
                self.cnn.execute_insert("car", txtid, txtmodel, txtyear,
txtcolor,
                                         txtcapacity, txtpower, txttype,
txttrans, txtprice)
            elif self.entry1.cget("state") == "disabled":
                self.cnn.execute_update("car", txtid, txtmodel, txtyear,
txtcolor,
                                         txtcapacity, txtpower, txttype,
txttrans, txtprice)

        self.cnn.disconnect()

        self.grid.delete(*self.grid.get_children())
        self.fnInit()

        self.entry1.delete("0", "end")
        self.entry2.delete("0", "end")
        self.entry3.delete("0", "end")
        self.entry4.delete("0", "end")
        self.entry5.delete("0", "end")
        self.entry6.delete("0", "end")
        self.entry7.delete("0", "end")
        self.entry8.delete("0", "end")
        self.entry9.delete("0", "end")

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

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

    def InsertData(self):
        self.grid.place(x=500, y=0, width=699, height=599)
        self.frame2.place(x=200, y=0)
        self.buttonSave.place(x=10, y=495)
        self.buttonCancel.place(x=10, y=545)

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

    def UpdateData(self):
```



```
        selection = self.grid.selection()
        if selection:
            self.grid.place(x=500, y=0, width=699, height=599)
            self.frame2.place(x=200, y=0)
            self.buttonSave.place(x=10, y=495)
            self.buttonCancel.place(x=10, y=545)

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

            id_selectioned = self.grid.item(selection)['text']
            values = self.grid.item(selection)['values']
            if values:
                value_col_model = values[0]
                value_col_year = values[1]
                value_col_color = values[2]
                value_col_engineCapacity = values[3]
                value_col_enginePower = values[4]
                value_col_engineType = values[5]
                value_col_transmission = values[6]
                value_col_price = values[7]

                self.entry1.insert(0, id_selectioned)
                self.entry2.insert(0, value_col_model)
                self.entry3.insert(0, value_col_year)
                self.entry4.insert(0, value_col_color)
                self.entry5.insert(0, value_col_engineCapacity)
                self.entry6.insert(0, value_col_enginePower)
                self.entry7.insert(0, value_col_engineType)
                self.entry8.insert(0, value_col_transmission)
                self.entry9.insert(0, value_col_price)

                self.entry1.config(state="disabled")
        else:
            messagebox.showerror("Error", "You must select a data")

    def DeleteData(self):
        selection = self.grid.selection()
        if selection:
            id_selectioned = self.grid.item(selection)['text']
            self.cnn.connect()
            self.cnn.execute_delete("car", id_selectioned)
            self.cnn.disconnect()
            self.grid.delete(*self.grid.get_children())
            self.fnInit()

    #Total
    def show_total_records(self):
        self.cnn.connect()
        data = self.cnn.execute_select("car")
        total = len(data)
        self.cnn.disconnect()

        self.txtResult.delete("1.0", tk.END)
        self.txtResult.insert(tk.END, f"Total Records: {total+1}")
    def SearchData(self):
        new_window = tk.Toplevel(self.root)
        new_window.title("Search")
        new_window.resizable(0, 0)

        # Size and position of the window
        widthScreen = self.root.winfo_screenwidth()
        heightScreen = self.root.winfo_screenheight()
        widthWindow = 700
        heightWindow = 50
        pwidth = int(widthScreen / 2 - widthWindow / 2)
        pheight = int(heightScreen / 2 - heightWindow / 2)
        new_window.geometry(f"{widthWindow}x{heightWindow}+{pwidth}+{pheight -
60}")

        def show_search_data(i, search_text):
            found_items = []
            all_items_values = []

            self.cnn.connect()
            data = self.cnn.execute_select("car")
            for row in data:
                all_items_values.append(list(row))
            self.cnn.disconnect()

            for j in range(len(all_items_values)):
                if search_text.lower() == str(all_items_values[j][i]).lower():
                    found_items.append(all_items_values[j])

            print(all_items_values)
            self.grid.delete(*self.grid.get_children())
            print(found_items)
            for data in found_items:
                self.grid.insert('', tk.END, text=data[0], values=data[1:])

            new_window.destroy()

        def get_selected_option(search_text):
           selected_option = radio_var.get()

           if (selected_option == "opcion1"):
               show_search_data(0, search_text)
           elif (selected_option == "opcion2"):
               show_search_data(1, search_text)
           elif (selected_option == "opcion3"):
               show_search_data(2, search_text)
           elif (selected_option == "opcion4"):
               show_search_data(8, search_text)
           else:
               show_search_data(0, search_text)

       style = ttk.Style()
       style.configure("TRadiobutton", font=("Helvetica", 12))
       style.configure("NoFocus.TRadiobutton",
highlightbackground=new_window.cget("background"))

       radio_var = tk.StringVar()

       radio_button1 = ttk.Radiobutton(new_window, text="Id",
variable=radio_var,
                                       value="option1",
style="NoFocus.TRadiobutton")
       radio_button1.place(x=30, y=12)

       radio_button2 = ttk.Radiobutton(new_window, text="Model",
variable=radio_var,
                                       value="option2",
style="NoFocus.TRadiobutton")
       radio_button2.place(x=80, y=12)

       radio_button3 = ttk.Radiobutton(new_window, text="Year",
variable=radio_var,
                                       value="option3",
style="NoFocus.TRadiobutton")
       radio_button3.place(x=160, y=12)

       radio_button4 = ttk.Radiobutton(new_window, text="Price",
variable=radio_var,
                                       value="option4")
       radio_button4.place(x=240, y=12)

       entry_search = tk.Entry(new_window, width=30, font=font.Font(size=10))
       entry_search.place(x=320, y=14)

       button_get_selected = ttk.Button(new_window, text="Get Selected Option",
                                        command=lambda:
get_selected_option(entry_search.get()))
```

```
    button_get_selected.place(x=550, y=11)
```