

SQL ASSIGNMENT

Report on Creating a Database Schema for Synthetic Car Data

Name : Dhavalkumar Sureshbhai Pithadiya

Student ID : 22003118

Introduction: The aim of this report is to document the process of generating synthetic car data using Python creating a relational database schema, and integrating the generated data into an SQLite database.

```
[83] pip install faker

Requirement already satisfied: faker in /usr/local/lib/python3.10/dist-packages (20.0.0)
Requirement already satisfied: python-dateutil>=2.4 in /usr/local/lib/python3.10/dist-packages (from faker) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.4->faker) (1.16.0)

[84] # importing the library functions
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
from faker import Faker
import sqlite3
from typing_extensions import TypedDict

fake = Faker()

# Number of samples
n = 1500

def random_letter():
    return chr(np.random.randint(65, 90)) # ASCII values for uppercase letters

def random_number():
    # Ensure the generated number is between 07 to 14 and 56 to 63 for the car
    return str(np.random.choice([str(i).zfill(2) for i in range(7, 15)] +
                               [str(i).zfill(2) for i in range(56, 64)]))

fake = Faker()

# Number of samples
n = 1500

def random_letter():
    return chr(np.random.randint(65, 90)) # ASCII values for uppercase letters

def random_number():
    # Ensure the generated number is between 07 to 14 and 56 to 63 for the car
    return str(np.random.choice([str(i).zfill(2) for i in range(7, 15)] +
                               [str(i).zfill(2) for i in range(56, 64)]))

1s completed at 3:16AM
```

Here, I have generated data for car which was sold out in 2010 to 2015 with the help of google colab. For initial base we have installed necessary library functions. In order to get fake data we have used **faker** library function. To make reliable data I have taken 1200 rows and 10 columns with different parameters.

```
def random_number():
    # Ensure the generated number is between 07 to 14 and 56 to 63 for the car
    return str(np.random.choice([str(i).zfill(2) for i in range(7, 15)] +
                               [str(i).zfill(2) for i in range(56, 64)]))

# Generate unique car numbers with the specified format of a UK car
unique_car_numbers = set()
while len(unique_car_numbers) < n:
    car_num = f'{random_letter()}{random_letter()}{random_number()} \
{random_letter()}{random_letter()}{random_number()}'
    unique_car_numbers.add(car_num)

numberplate = list(unique_car_numbers)

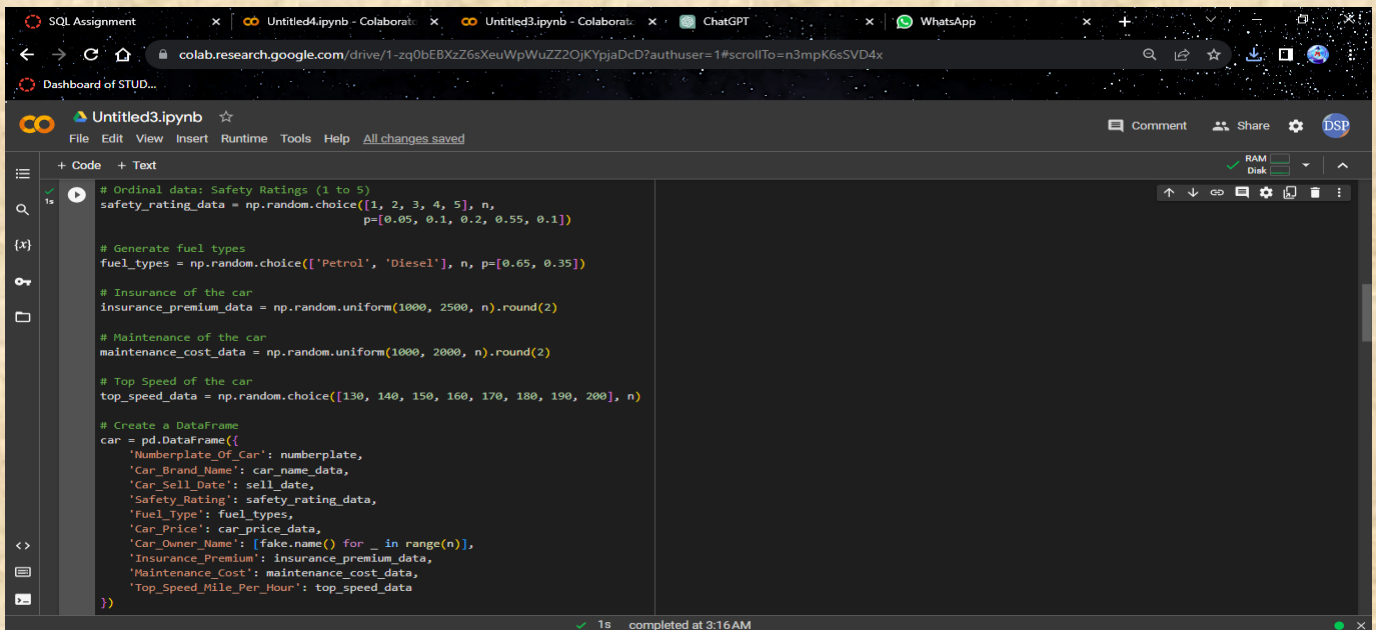
# Name of the car brand name
car_brands = ['Mercedes', 'BMW', 'Toyota', 'Ford', 'Peugeot', 'Audi', 'Nissan']
car_name_data = np.random.choice(car_brands, n)

# Interval data: date of selling
sell_year = np.random.randint(2010, 2015, n)
sell_month = np.random.randint(1, 13, n)
sell_day = np.random.randint(1, 29, n)
sell_date = [f'{sell_year[i]}-{sell_month[i].zfill(2)}-{sell_day[i].zfill(2)}' for i in range(n)]

# Ratio data: Price of the Car
car_price_data = np.random.randint(10000, 20001, n)

# Ordinal data: Safety Ratings (1 to 5)
```

In given data I have provided all four type data Nominal, Ordinal, Interval, Ratio(**NOIR**). This includes details such as car unique number plates(because it can not be repeated), car brand names, selling dates, prices, safety ratings, fuel types, insurance premiums, maintenance costs, and top speeds. We can see in the below image.

A screenshot of a Google Colab notebook interface. The browser tabs at the top include 'SQL Assignment', 'Untitled4.ipynb - Colaborat...', 'Untitled3.ipynb - Colaborat...', 'ChatGPT', and 'WhatsApp'. The address bar shows a Google Drive link. The notebook is titled 'Untitled3.ipynb' and has a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. On the left sidebar, there are icons for file explorer, search, and other notebook functions. The main code area contains the following Python code:

```
# Ordinal data: Safety Ratings (1 to 5)
safety_rating_data = np.random.choice([1, 2, 3, 4, 5], n,
p=[0.05, 0.1, 0.2, 0.55, 0.1])

# Generate fuel types
fuel_types = np.random.choice(['Petrol', 'Diesel'], n, p=[0.65, 0.35])

# Insurance of the car
insurance_premium_data = np.random.uniform(1000, 2500, n).round(2)

# Maintenance of the car
maintenance_cost_data = np.random.uniform(1000, 2000, n).round(2)

# Top Speed of the car
top_speed_data = np.random.choice([130, 140, 150, 160, 170, 180, 190, 200], n)

# Create a DataFrame
car = pd.DataFrame({
    'Numberplate_Of_Car': numberplate,
    'Car_Brand_Name': car_name_data,
    'Car_Sell_Date': sell_date,
    'Safety_Rating': safety_rating_data,
    'Fuel_Type': fuel_types,
    'Car_Price': car_price_data,
    'Car_Owner_Name': [fake.name() for _ in range(n)],
    'Insurance_Premium': insurance_premium_data,
    'Maintenance_Cost': maintenance_cost_data,
    'Top_Speed_Mile_Per_Hour': top_speed_data
})
```

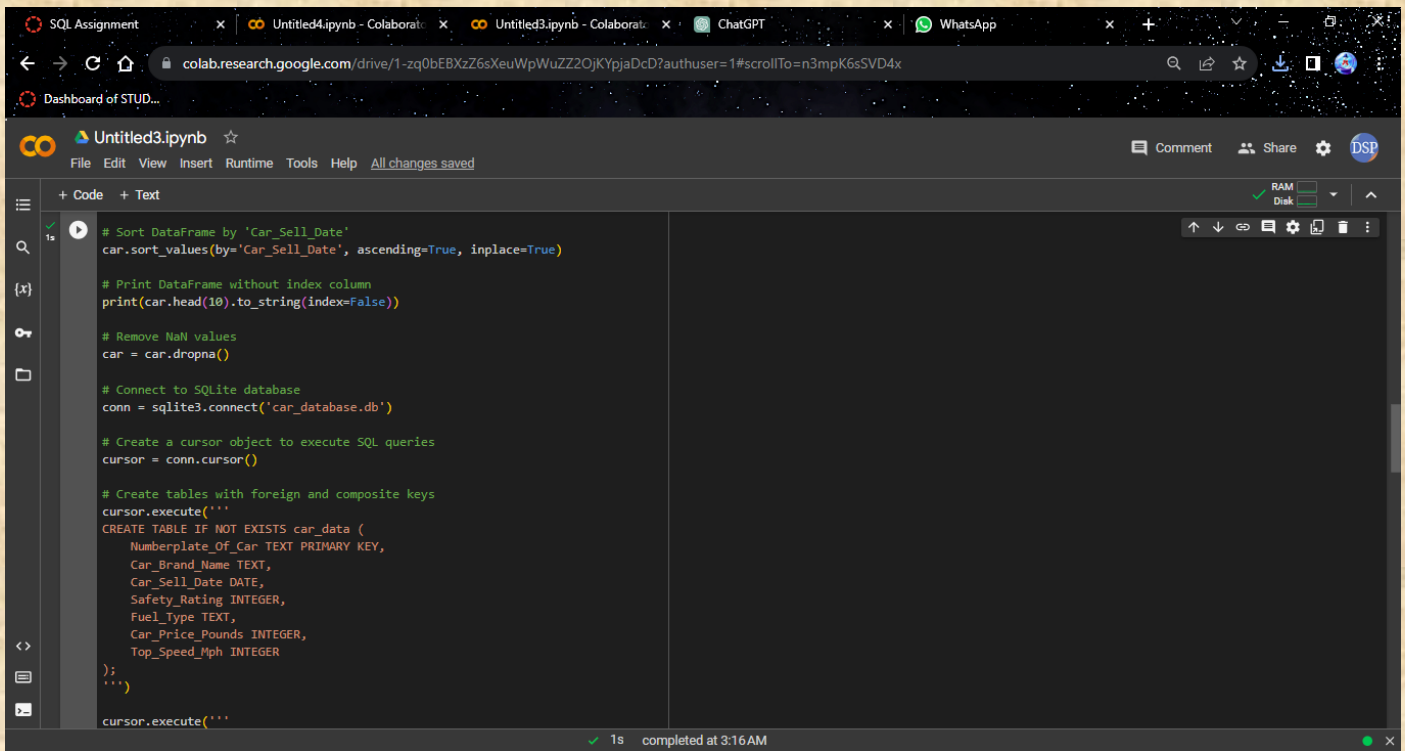
The status bar at the bottom indicates '1s completed at 3:16 AM'.

DataFrame Creation and Sorting: The generated data is organized into a Pandas DataFrame named 'car.' To enhance readability, the DataFrame is sorted based selling dates in ascending order.

Futhermore,

SQLite Database Connection and Schema Creation:

A connection to an SQLite database, named 'car_database.db,' is established. The code defines three tables - 'car_data,' 'car_expense,' and 'car_owner' - incorporating foreign key relationships. This schema is designed to capture the various aspects of car-related data and expenses.



The screenshot shows a Jupyter Notebook titled 'Untitled3.ipynb' in a web browser. The code in the notebook is as follows:

```
# Sort DataFrame by 'Car_Sell_Date'
car.sort_values(by='Car_Sell_Date', ascending=True, inplace=True)

# Print DataFrame without index column
print(car.head(10).to_string(index=False))

# Remove NaN values
car = car.dropna()

# Connect to SQLite database
conn = sqlite3.connect('car_database.db')

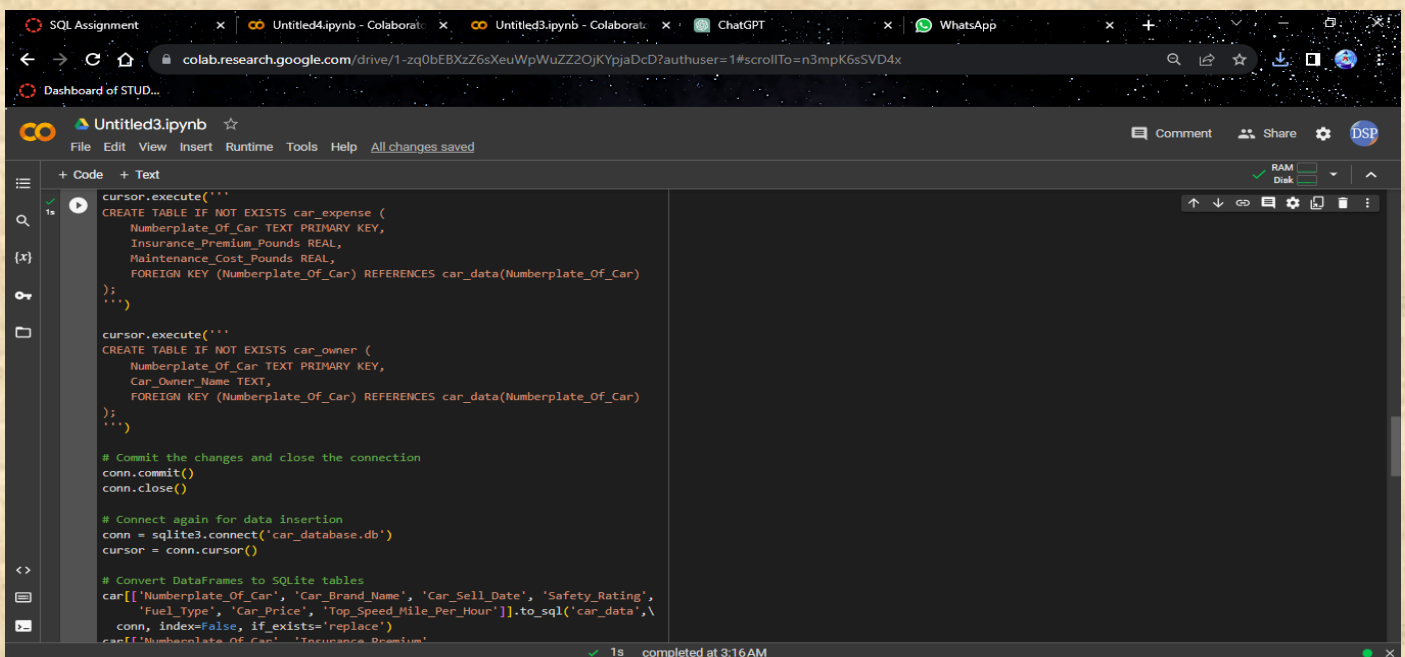
# Create a cursor object to execute SQL queries
cursor = conn.cursor()

# Create tables with foreign and composite keys
cursor.execute('''
CREATE TABLE IF NOT EXISTS car_data (
    Numberplate_Of_Car TEXT PRIMARY KEY,
    Car_Brand_Name TEXT,
    Car_Sell_Date DATE,
    Safety_Rating INTEGER,
    Fuel_Type TEXT,
    Car_Price_Pounds INTEGER,
    Top_Speed_Mph INTEGER
);
''')

cursor.execute('''
```

The notebook interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), a toolbar with icons for code, text, and search, and a status bar at the bottom indicating '1s completed at 3:16AM'.

Following this we have to make transformation, which is



The screenshot shows a Jupyter Notebook titled 'Untitled3.ipynb' in a web browser. The code in the notebook is as follows:

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS car_expense (
    Numberplate_Of_Car TEXT PRIMARY KEY,
    Insurance_Premium_Pounds REAL,
    Maintenance_Cost_Pounds REAL,
    FOREIGN KEY (Numberplate_Of_Car) REFERENCES car_data(Numberplate_Of_Car)
);
''')

cursor.execute('''
CREATE TABLE IF NOT EXISTS car_owner (
    Numberplate_Of_Car TEXT PRIMARY KEY,
    Car_Owner_Name TEXT,
    FOREIGN KEY (Numberplate_Of_Car) REFERENCES car_data(Numberplate_Of_Car)
);
''')

# Commit the changes and close the connection
conn.commit()
conn.close()

# Connect again for data insertion
conn = sqlite3.connect('car_database.db')
cursor = conn.cursor()

# Convert DataFrames to SQLite tables
car[['Numberplate_Of_Car', 'Car_Brand_Name', 'Car_Sell_Date', 'Safety_Rating',
    'Fuel_Type', 'Car_Price', 'Top_Speed_Mile_Per_Hour']].to_sql('car_data', \
    conn, index=False, if_exists='replace')
car[['Numberplate_Of_Car', 'Insurance_Premium'
```

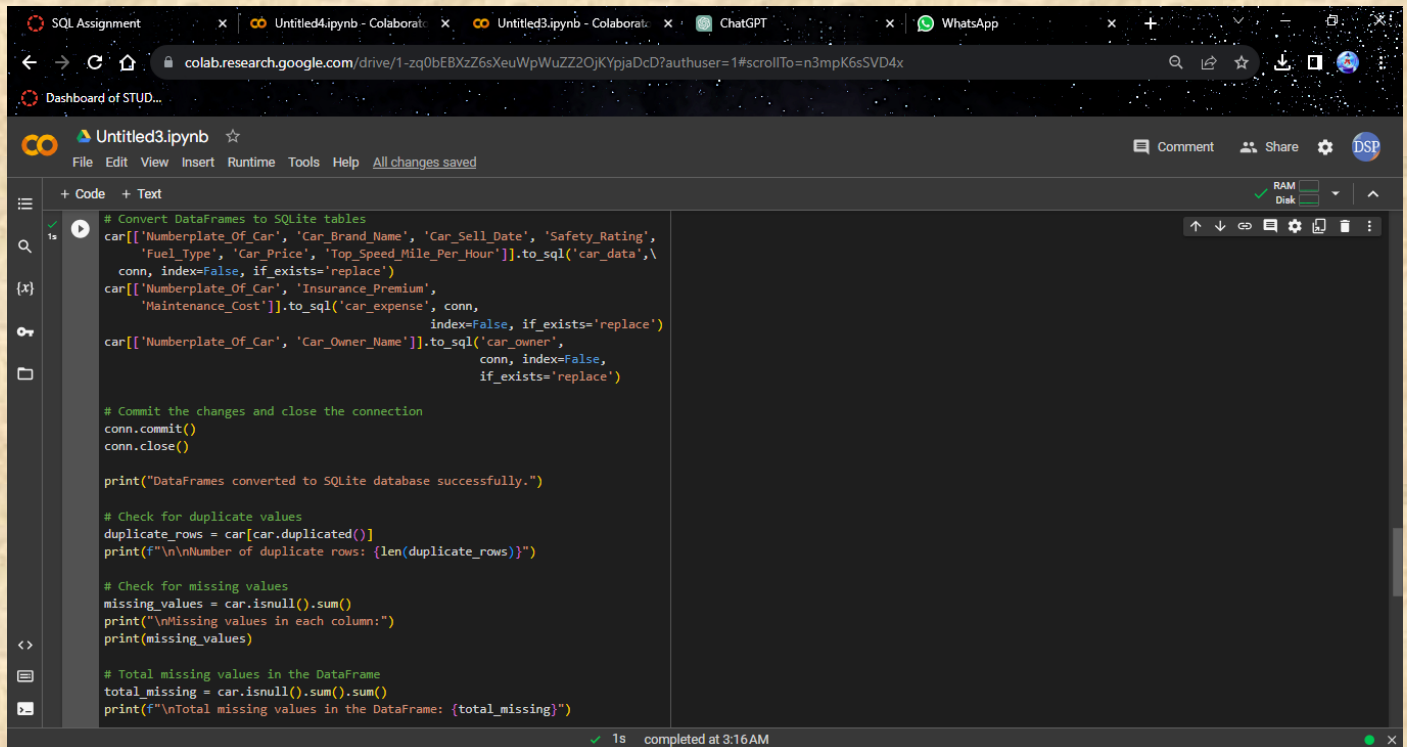
The notebook interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), a toolbar with icons for code, text, and search, and a status bar at the bottom indicating '1s completed at 3:16AM'.

Data Insertion into the Database:

Using the SQLite connection, the code converts the Pandas DataFrames into corresponding tables in the database with use of foreign keys and composite keys as well , ensuring seamless integration of the generated car data.

After completing that we just do ...

Data Validation: The code includes checks for data integrity, identifying and removing duplicate rows, as well as handling missing values within the DataFrame.



```
# Convert DataFrames to SQLite tables
car[['Numberplate_Of_Car', 'Car_Brand_Name', 'Car_Sell_Date', 'Safety_Rating',
      'Fuel_Type', 'Car_Price', 'Top_Speed_Mile_Per_Hour']].to_sql('car_data', \
conn, index=False, if_exists='replace')
car[['Numberplate_Of_Car', 'Insurance_Premium',
      'Maintenance_Cost']].to_sql('car_expense', conn,
                                index=False, if_exists='replace')
car[['Numberplate_Of_Car', 'Car_Owner_Name']].to_sql('car_owner',
conn, index=False,
if_exists='replace')

# Commit the changes and close the connection
conn.commit()
conn.close()

print("DataFrames converted to SQLite database successfully.")

# Check for duplicate values
duplicate_rows = car[car.duplicated()]
print(f"\n\nNumber of duplicate rows: {len(duplicate_rows)}")

# Check for missing values
missing_values = car.isnull().sum()
print("\n\nMissing values in each column:")
print(missing_values)

# Total missing values in the DataFrame
total_missing = car.isnull().sum().sum()
print(f"\n\nTotal missing values in the DataFrame: {total_missing}")
```

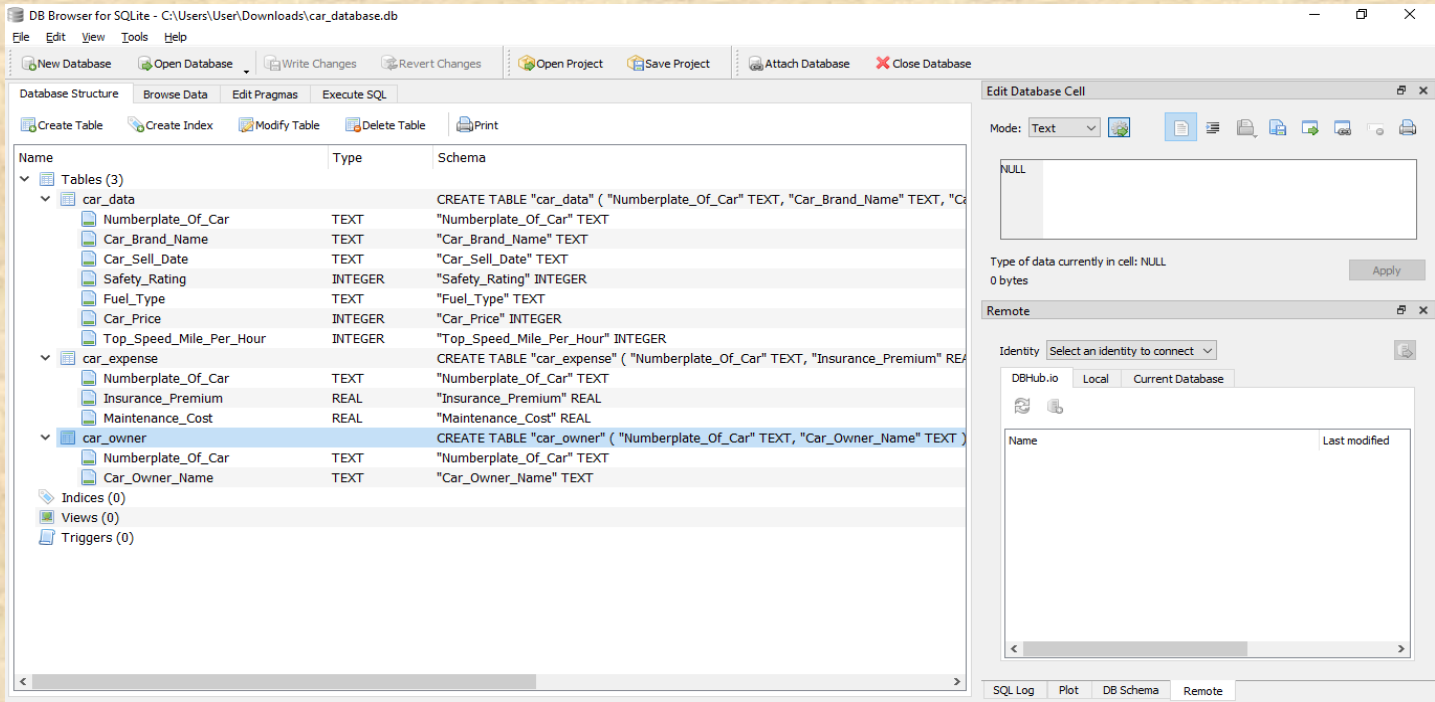
Here we got the final

Conclusion: The Python script successfully accomplishes the task of generating synthetic car data, creating a relational database schema, and populating the SQLite database. The integration of these components lays the groundwork for further analysis and querying of the car-related information.

The screenshot shows a Jupyter Notebook with a DataFrame containing car data. The DataFrame has columns: Numberplate_Of_Car, Car_Brand_Name, Car_Sell_Date, Safety_Rating, Fuel_Type, Car_Price, Car_Owner_Name, Insurance_Premium, Maintenance_Cost, and Top_Speed_Mile_Per_Hour. The data is displayed in a table format. Below the table, the text 'DataFrames converted to SQLite database successfully.' is shown. Further down, the output shows the number of duplicate rows (0) and missing values in each column (all 0). The dtype is 'int64'. The total missing values in the DataFrame are 0.

| Numberplate_Of_Car | Car_Brand_Name | Car_Sell_Date | Safety_Rating | Fuel_Type | Car_Price | Car_Owner_Name | Insurance_Premium | Maintenance_Cost | Top_Speed_Mile_Per_Hour |
|--------------------|----------------|---------------|---------------|-----------|-----------|-------------------|-------------------|------------------|-------------------------|
| EY60 QLN | Audi | 2010-01-01 | 4 | Petrol | 19941 | Brittany Thomas | 2475.76 | 1039.40 | 180 |
| ES63 MRP | Toyota | 2010-01-01 | 3 | Petrol | 19319 | Matthew Shields | 1694.95 | 1874.50 | 200 |
| CU08 EXG | Peugeot | 2010-01-03 | 4 | Petrol | 10452 | Amanda Patton | 1774.10 | 1703.32 | 160 |
| HW11 HQY | Audi | 2010-01-03 | 4 | Petrol | 12008 | Anthony Scott | 1844.78 | 1437.11 | 190 |
| FR59 OLO | Ford | 2010-01-05 | 4 | Petrol | 15624 | Richard Little | 2087.73 | 1919.25 | 190 |
| TJ61 TBL | BMW | 2010-01-06 | 4 | Diesel | 10549 | Shawn Huff | 2000.88 | 1679.88 | 130 |
| PV59 VCU | Audi | 2010-01-09 | 3 | Petrol | 11919 | Savannah Gonzales | 1802.45 | 1593.85 | 150 |
| RD14 MDR | Nissan | 2010-01-09 | 2 | Petrol | 11376 | Samuel Garcia | 2112.28 | 1106.50 | 190 |
| LP63 TXY | Ford | 2010-01-14 | 4 | Petrol | 13062 | Annette Larson | 2365.07 | 1674.46 | 180 |
| NL14 FWV | Mercedes | 2010-01-16 | 2 | Petrol | 18100 | Amanda Arnold | 1045.45 | 1893.04 | 130 |

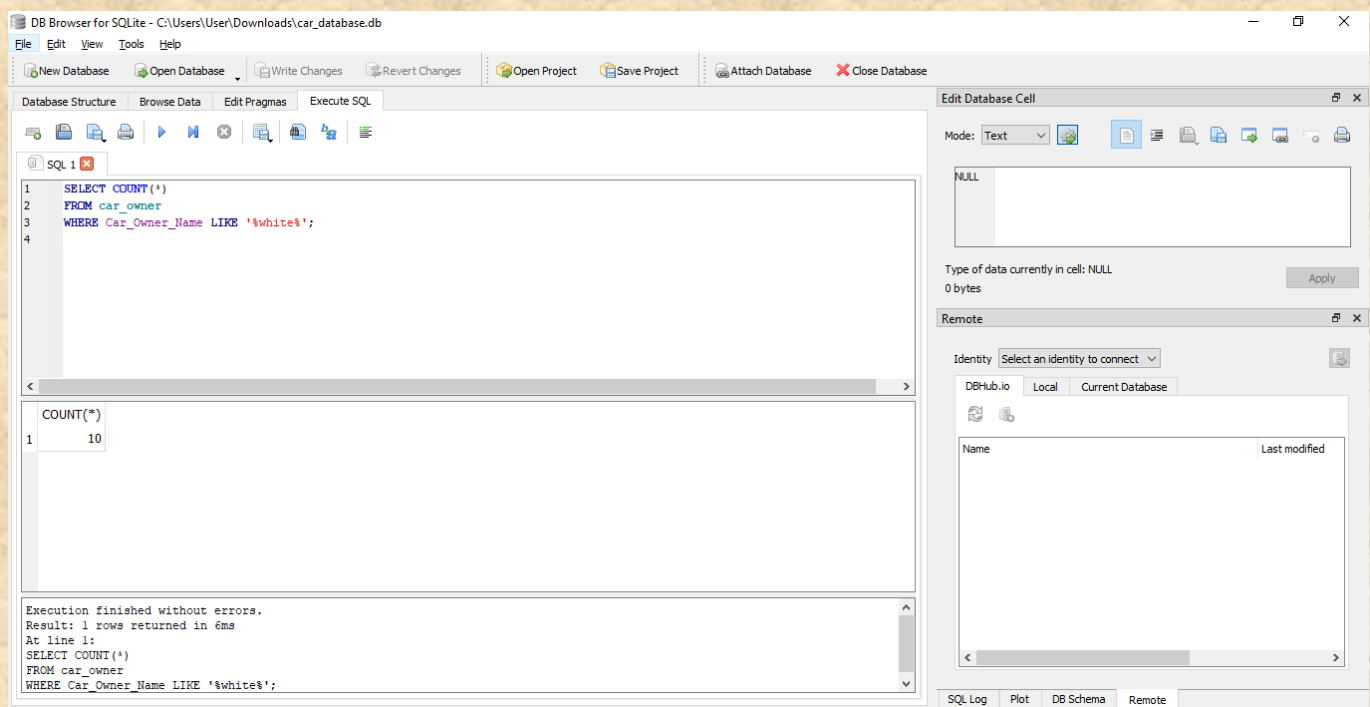
After running all over the code we generate fake car database in our programme so here database file look like as given in the below picture



Here we create 3 table for car data in that 1st table represents basic car details such as Numberplate, Car Brand, Selling price of car and date, Safety rating , fuel type and top speed , 2nd Table represent expense of insurance and maintenance and 3rd Table stands for car owner name.

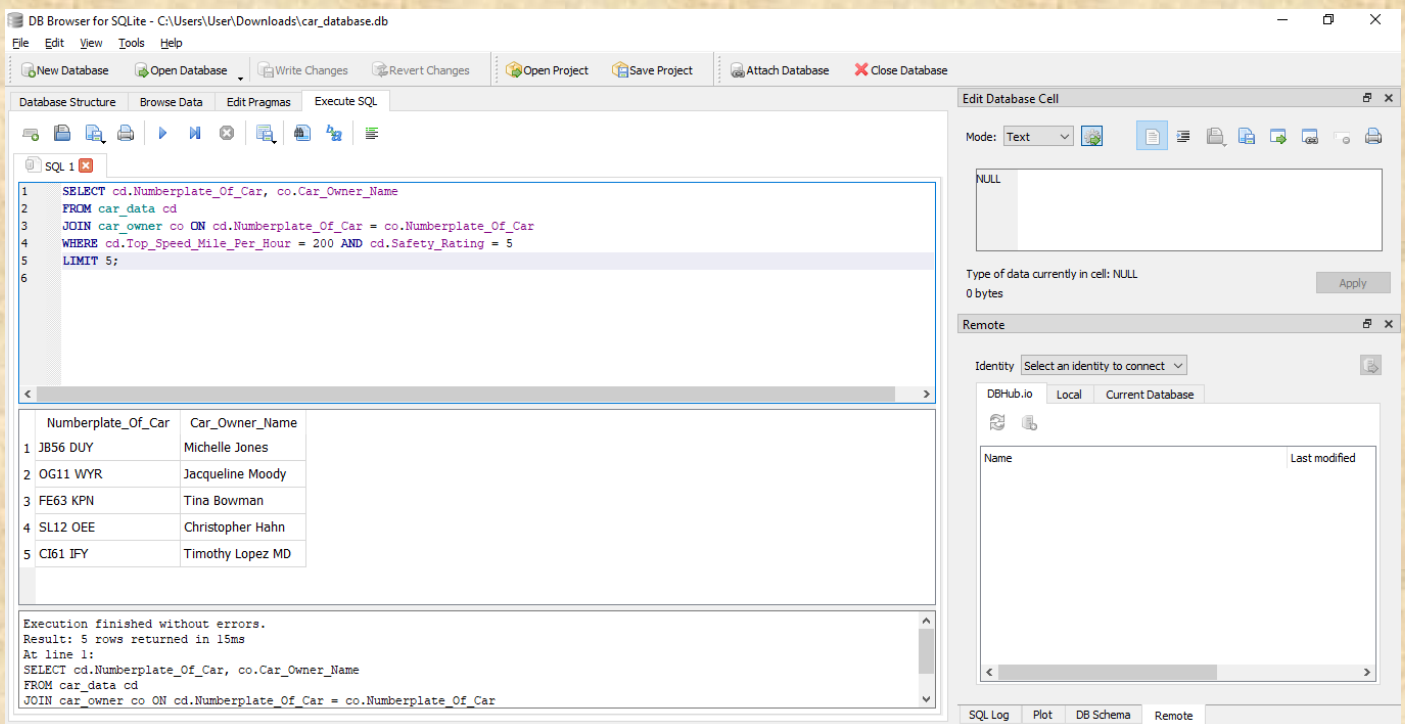
These 3 Table represent distinct values and parameters which is most useful for car information

In order to learn more about our car database schema, we can run just simple query to execute SQL. So here is small example is given in the image.

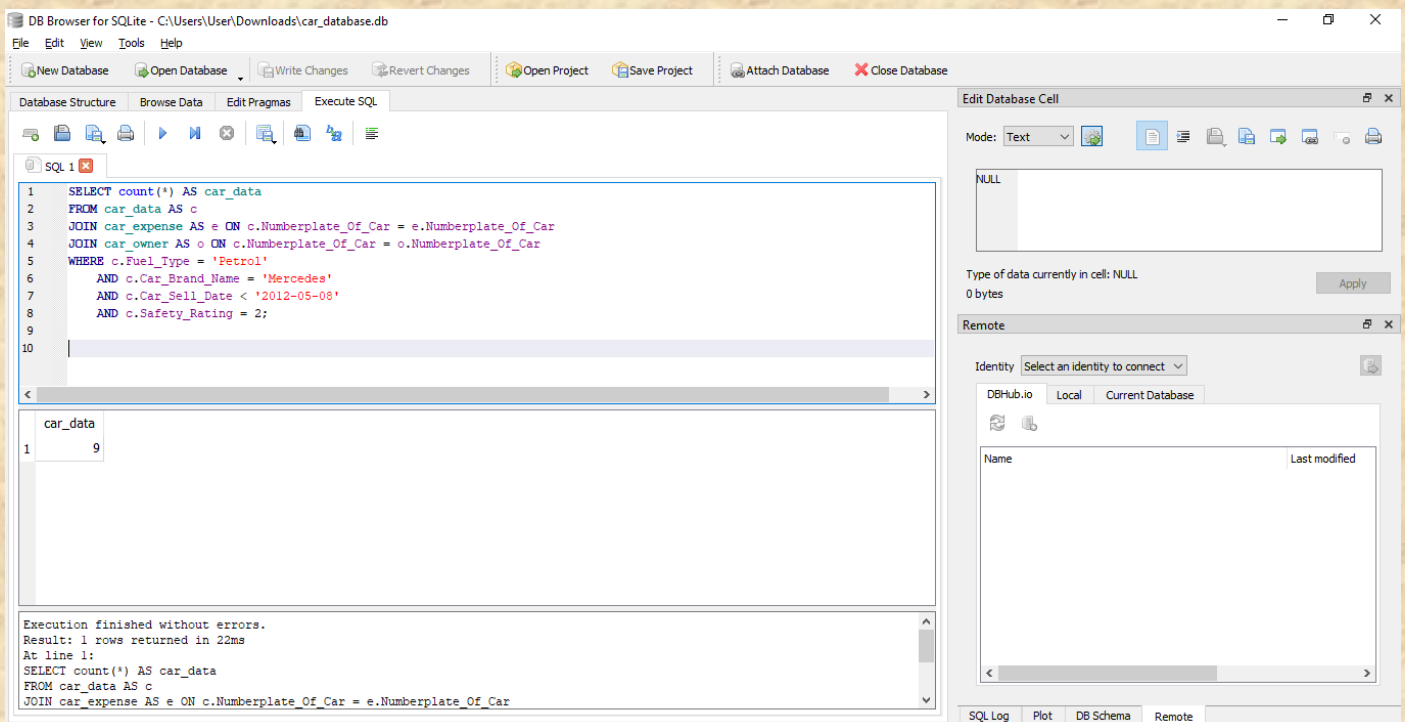


In this window we have just try to find name of the car owner with the last name white so we find in total 10 person with given surname.

Similarly, Consider another example for given car data with highest rating and top speed so if we would like to know car owner name than we can use join function to combine two table and output will be like as below:

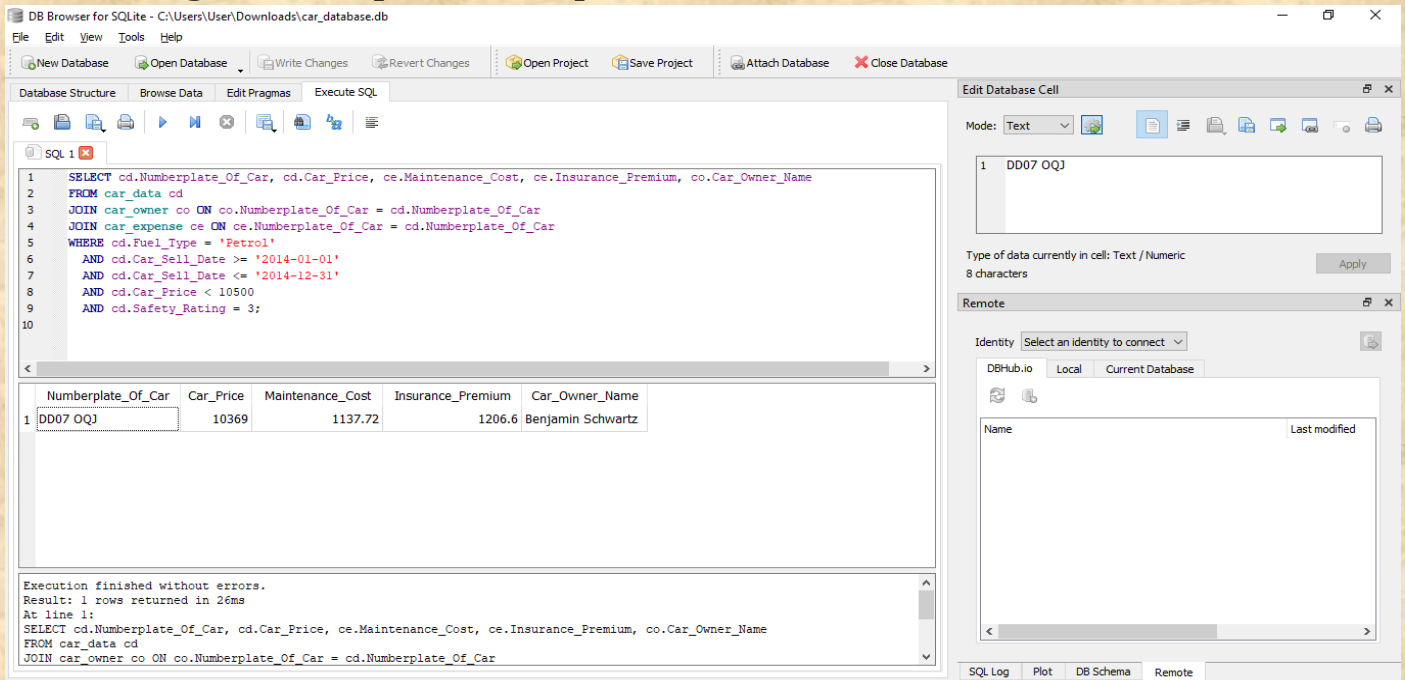


Another example over here where we can connect table with using our specified input code (like JOIN, ON, AND, WHERE) for the sql schema execute so we can get appropriate answer like below



In our given Database all of the three table has a single column which is same that is called foreign Key. In order to join table we are using nothing else but foreign key and This query selects common value information for a petrol car sold in 2014, with a price less

than £10500 and a safety rating of 3. We can Adjust the date range according to our specific requirements.



The screenshot shows the DB Browser for SQLite application. The main window displays a SQL query in the 'Execute SQL' tab. The query filters for cars with a fuel type of 'Petrol', a sell date between 2014-01-01 and 2014-12-31, a price under 10500, and a safety rating of 3. The results are shown in a table with one row for car 'DD07 OQJ'.

```
1 SELECT cd.Numberplate_Of_Car, cd.Car_Price, ce.Maintenance_Cost, ce.Insurance_Premium, co.Car_Owner_Name
2 FROM car_data cd
3 JOIN car_owner co ON co.Numberplate_Of_Car = cd.Numberplate_Of_Car
4 JOIN car_expense ce ON ce.Numberplate_Of_Car = cd.Numberplate_Of_Car
5 WHERE cd.Fuel_Type = 'Petrol'
6 AND cd.Car_Sell_Date >= '2014-01-01'
7 AND cd.Car_Sell_Date <= '2014-12-31'
8 AND cd.Car_Price < 10500
9 AND cd.Safety_Rating = 3;
10
```

| Numberplate_Of_Car | Car_Price | Maintenance_Cost | Insurance_Premium | Car_Owner_Name |
|--------------------|-----------|------------------|-------------------|-------------------|
| DD07 OQJ | 10369 | 1137.72 | 1206.6 | Benjamin Schwartz |

Execution finished without errors.
Result: 1 rows returned in 26ms
At line 1:
SELECT cd.Numberplate_Of_Car, cd.Car_Price, ce.Maintenance_Cost, ce.Insurance_Premium, co.Car_Owner_Name
FROM car_data cd
JOIN car_owner co ON co.Numberplate_Of_Car = cd.Numberplate_Of_Car

The right sidebar shows the 'Edit Database Cell' window with the value 'DD07 OQJ' and the 'Remote' connection settings.

For Coding and Database file checkout Github repository [Over Here.](#)