Oliver J. Molenschot

Analyzing Road Safety Using SQL

Documentation

Outline

Oliver J. Molenschot

Introduction

I completed this project to develop my skills in the language SQL (using MySQL). Below you will find a detailed explanation of the process during the project. I mention specific challenges that occurred and the solutions I developed.

Uploading and Cleaning the Data

I downloaded the data on the UK Road Safety Accidents from this government page. I analyzed it in Python using the Pandas library and wrote SQL code to create new tables. I created the column names and datatypes manually to learn in detail about the different data types available on SQL.

While trying to load the data (CSV file) onto the newly created table, the following error occurred:

```
The MySQL server is running with the --secure-file-priv option so it cannot execute this statement windows
```

For debugging purposes, I ran the following code:

```
SHOW VARIABLES LIKE 'secure_file_priv';
```

Which displayed a directory from which MySQL had permissions to read and write. I added the CSV files in that directory, which solved the problem. Some of the dates in the excel file were marked as text (string). I changed the whole column into the date format. Rows containing empty cells were removed using the dataframe.dropna(inplace=True) function in pandas (python). Afterward, the data was loaded (see Appendix) The same process mentioned above was performed for the vehicles-dataset and vehicle-types CSV files (except for finding the directory wherein MySQL has permissions, as that directory remains the same).

Problem 1: Evaluate Median Severity Value of Accidents Caused by Motorcycles

For the first task, I began by determining the median of all accident severity cells through the following code:

```
SET @rowindex := -1;

SELECT

        AVG(a.severity) as Median
```

```
FROM
        (SELECT @rowindex:=@rowindex +1 AS rowindex,
    accident.Accident_Severity AS severity
FROM accident
    ORDER BY accident.Accident_Severity) AS a
WHERE
a.rowindex IN (FLOOR(@rowindex / 2), CEIL(@rowindex / 2));
```

I confirmed that the result was current using Python, through Pandas, using the function dataframe[column_name].median()

The second part of the first question specifies a condition for the median, namely that it must be only for motorcycle vehicles. To achieve this, one must first (A) find the multiple motorcycle indices under the columns vehicle_type_index in the table vehicle_types. Then, they (B) must be matched with the vehicle_type column under the table vehicles. Later, the indices (C) under the column accident_index (filtered by the vehicle_type column) must be matched with the column accident_index from the accident key. Finally, the median (D) must be calculated only using the remaining rows filtered by the column accident_index. Thus, two nested queries are required to achieve the above.

Sub-Problem A

```
SELECT vehicle_types.Vehicle_type_index
FROM vehicle_types
WHERE label LIKE '%otorcycle%'
);
```

Note that “otorcycle" was used instead of "Motorcycle" or "motorcycle" to avoid the concern of the first letter being either in uppercase or lowercase.

Sub-Problem B

```
SELECT vehicles.Accident_Index, vehicles.Vehicle_Type
FROM vehicles
```

```
WHERE Vehicle_Type IN (
   SELECT vehicle_types.Vehicle_type_index
   FROM vehicle_types
   WHERE label LIKE '%otorcycle%'
);
```

Sub-Problem C

```
WHERE accident.Accident_Index IN (
        SELECT vehicles.Accident_Index
        FROM vehicles
        WHERE Vehicle_Type IN (
            SELECT vehicle_types.Vehicle_type_index
            FROM vehicle_types
            WHERE label LIKE '%otorcycle%'
            )
        )
```

Note that this is part of the code is present in sub-problem D.

Sub-Problem D

This final part contains the whole code (all of the aforementioned summed) to find the median accident severity based on motorcycle vehicles:

```
SET @rowindex := -1;
SELECT AVG(a.severity) as Median
FROM (
    SELECT @rowindex:=@rowindex +1 AS rowindex,
    accident.Accident_Severity AS severity
```

```
    FROM accident
    WHERE accident.Accident_Index IN (
        SELECT vehicles.Accident_Index
        FROM vehicles
        WHERE Vehicle_Type IN (
            SELECT vehicle_types.Vehicle_type_index
            FROM vehicle_types
            WHERE label LIKE '%otorcycle%')
)
ORDER BY accident.Accident_Severity) AS a
WHERE
a.rowindex IN (FLOOR(@rowindex / 2), CEIL(@rowindex / 2));
```

Problem 2: Evaluate Accident Severity per Vehicle Type

To achieve question 2, namely, "Evaluate Accident Severity per Vehicle Type", proper foreign keys are required. It is problematic because the accident_index has both repeating values (which impedes it from being a primary key) and some of its values in the accident table are missing from the vehicles table. To fix this, I first (A) delete rows wherein the accident_index is duplicated within the vehicles table. Secondly, the (B) accident_index values present in the accident table but not in the vehicles table must trigger the deletion of the row.

Sub-Problem A

```
DELETE FROM vehicles
WHERE key_index NOT IN (
    SELECT max_index
    FROM  (
        SELECT MAX(key_index)  as max_index
        FROM vehicles
```

```
        GROUP BY vehicles.Accident_Index
        ) as t
);
```

Note that the second inner loop has no natural necessity. It is present only because MySQL doesn't allow updating the table you are already using in an inner select as the update criteria. The second inner loop (middle loop) thus creates a temporary table to work around this constraint.

Sub-Problem B

```
DELETE FROM accident
WHERE accident.Accident_Index NOT IN (
    SELECT vehicles.Accident_Index
    FROM vehicles
);
```

The foreign key was then created:

```
ALTER TABLE accident
ADD FOREIGN KEY (Accident_Index)
REFERENCES vehicles(Accident_Index)
ON DELETE SET NULL
ON UPDATE SET NULL;
```

Now that the primary key of the vehicles table is changed and the foreign key of the accident table is present, the database can answer the second question. The following code provides an answer by displaying the accident_severity values while displaying their respective vehicle_type values. The join command simplifies the process.

```
SELECT acc_severity, veh_type
FROM
        (
   SELECT accident.Accident_Severity AS acc_severity,
   vehicles.Vehicle_Type AS veh_type
```

```
    FROM accident
    JOIN vehicles ON accident.Accident_Index=vehicles.Accident_Index
    ) AS T
ORDER BY veh_type
;
```

Problem 3: Calculate the Average Severity by Vehicle Type

The following code provides a solution to the third question ("Calculate the Average Severity by vehicle type"). It contains the code above (which answers question two) and contains both the command AVG and the command GROUP BY.

```
SELECT AVG(acc_severity), veh_type
FROM (
    SELECT accident.Accident_Severity AS acc_severity,
    vehicles.Vehicle_Type AS veh_type
    FROM accident
    JOIN vehicles ON accident.Accident_Index=vehicles.Accident_Index
    ) AS T
GROUP BY veh_type
ORDER BY veh_type
;
```

Problem 4: Calculate the Average Severity by Motorcycles

Below is the answer to question 4. It is similar to the one above; the only difference being the "WHERE" command which includes a nested query:

```
SELECT AVG(acc_severity), veh_type
FROM
        (
```

```
    SELECT accident.Accident_Severity AS acc_severity,
    vehicles.Vehicle_Type AS veh_type
    FROM accident
    JOIN vehicles ON accident.Accident_Index=vehicles.Accident_Index
    ) AS T
WHERE veh_type IN (
     SELECT vehicle_types.Vehicle_type_index
     FROM vehicle_types
     WHERE label LIKE '%otorcycle%')
GROUP BY veh_type
ORDER BY veh_type
;
```

Appendix

Appendix A: Creating accident table and loading data

```
CREATE TABLE accident (
        Event_number INT(6) PRIMARY KEY,
        Accident_Index VARCHAR(20),
        Location_Easting_OSGR DOUBLE(10,2),
        Location_Northing_OSGR DOUBLE(10,2),
        Longitude DOUBLE(11,8),
        Latitude DOUBLE(11,8),
        Police_Force INT (3),
        Accident_Severity INT(1),
        Number_of_Vehicles INT(2),
        Number_of_Casualties INT(2),
        Date_column DATE,
        Day_of_Week INT(1),
        time_column TIME(2),
        Local_Authority_District INT(3),
        Local_Authority_Highway VARCHAR(20),
        first_Road_Class INT(1),
        first_Road_Number INT(4),
        Road_Type INT(1),
        Speed_limit INT(2),
        Junction_Detail INT(1),
        Junction_Control INT(1),
        second_Road_Class INT(1),
        second_Road_Number INT(4),
```

```
    Pedestrian_Crossing_Human_Control INT(1),
    Pedestrian_Crossing_Physical_Facilities INT(1),
    Light_Conditions INT(1),
    Weather_Conditions INT(1),
    Road_Surface_Conditions INT(1),
    Special_Conditions_at_Site int(1),
    Carriageway_Hazards INT(1),
    Urban_or_Rural_Area INT(1),
    Did_Police_Officer_Attend_Scene_of_Accident INT(1),
    LSOA_of_Accident_Location VARCHAR(20)
);
```

```
LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/accidents-dataset-uk-updated.csv"
    INTO TABLE accident
  FIELDS TERMINATED BY ','
  LINES TERMINATED BY '\n'
    IGNORE 1 ROWS;
```

Appendix B: Creating vehicles table and loading data

```
CREATE TABLE vehicles (
    key_index INT(8) PRIMARY KEY,
    Accident_Index VARCHAR(25),
    Vehicle_Reference INT(2),
    Vehicle_Type INT(2),
    Towing_and_Articulation INT(1),
    Vehicle_Manoeuvre INT(2),
```

```
    Vehicle_Location_Restricted_Lane INT(1),
    Junction_Location INT(1),
    Skidding_and_Overturning INT(1),
    Hit_Object_in_Carriageway INT(2),
    Vehicle_Leaving_Carriageway INT(1),
    Hit_Object_off_Carriageway INT(2),
    first_Point_of_Impact INT(1),
    Was_Vehicle_Left_Hand_Drive INT(1),
    Journey_Purpose_of_Driver INT(1),
    Sex_of_Driver INT(1),
    Age_of_Driver INT(2),
    Age_Band_of_Driver INT(2),
    Engine_Capacity_CC INT(5),
    Propulsion_Code INT(2),
    Age_of_Vehicle INT(3),
    Driver_IMD_Decile INT,
    Driver_Home_Area_Type INT(1),
    Vehicle_IMD_Decile INT
);
```

```
LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/vehicles-dataset-uk.csv"
    INTO TABLE vehicles
  FIELDS TERMINATED BY ','
  LINES TERMINATED BY '\n'
    IGNORE 1 ROWS;
```

Appendix C: Creating vehicles types table and loading data

```
CREATE TABLE vehicle_types (
        Vehicle_type_index INT(2) PRIMARY KEY,
        code INT,
        label VARCHAR(50)
);
```

```
LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/vehicle-types-dataset-uk.csv"
        INTO TABLE vehicle_types
   FIELDS TERMINATED BY ','
   LINES TERMINATED BY '\n'
        IGNORE 1 ROWS;
```