Taxi Application Database

University of Niagara Falls Canada
Master of Data Analytics
CPSC-500-5: SQL Databases

Student: Shristi Duwadi (NF1000995)

Professor: Omid Isfahanialamdari

December 9, 2024

Chapter 1. Introduction

The development of the taxi application database reflects a structured approach to data management and operational efficiency. Using an SQL-based architecture with 11 interconnected tables, the system integrates key data entities, including customers, drivers, vehicles, payments, and trip history. The database follows Third Normal Form (3NF) principles, reducing redundancy and maintaining data integrity for accurate and efficient data retrieval.

Data manipulation plays a vital role in maintaining clean and usable data for analysis. Techniques such as removing prefixes (e.g., "Mr." and "Mrs.") from names and correcting inaccurate driver and customer scores enhance data quality. Foreign key constraints enforce referential integrity, while "LOCK TABLES" and "UNLOCK TABLES" commands prevent data corruption during insertion processes.

To streamline analysis, VIEWS are implemented, allowing reusable queries that provide insights into trip activity, payment trends, and driver performance. This enables data-driven decision-making, supports business forecasting, and enhances reporting efficiency. Additionally, Python-generated synthetic data is used to test the system's functionality and ensure its reliability.

Overall, the database not only supports the operational needs of the taxi service but also enables real-time decision-making and strategic planning. By facilitating customer insights, driver performance analysis, and financial tracking, the system provides a robust platform for future development and advanced analytics.

Chapter 2. Database creation

Initially, we established a database named **"final_project"** utilizing the following SQL command:

CREATE DATABASE IF NOT EXISTS `final_project`;

To ensure the SQL table was created without errors, we adhered to a structured and systematic approach in the database creation process. Table 2.1 shows the table creation order and description of each tables created.

Table 2.1 Table orders and it's description

Order	Table name	Description
1	vehicle	Information about each vehicle
2	status_person	Status of a user: active, inactive or suspended
3	dni_type	Identity document type: passport, ID, driver's license
4	account_user	Account user information
5	score	Drivers' and customers' score of each trip
6	payment_method	Payment method choices
7	location	Route of each trip
8	payment	Payment information of each trip
9	driver	Driver information
10	customer	Customer information
11	trip_history	Trip history

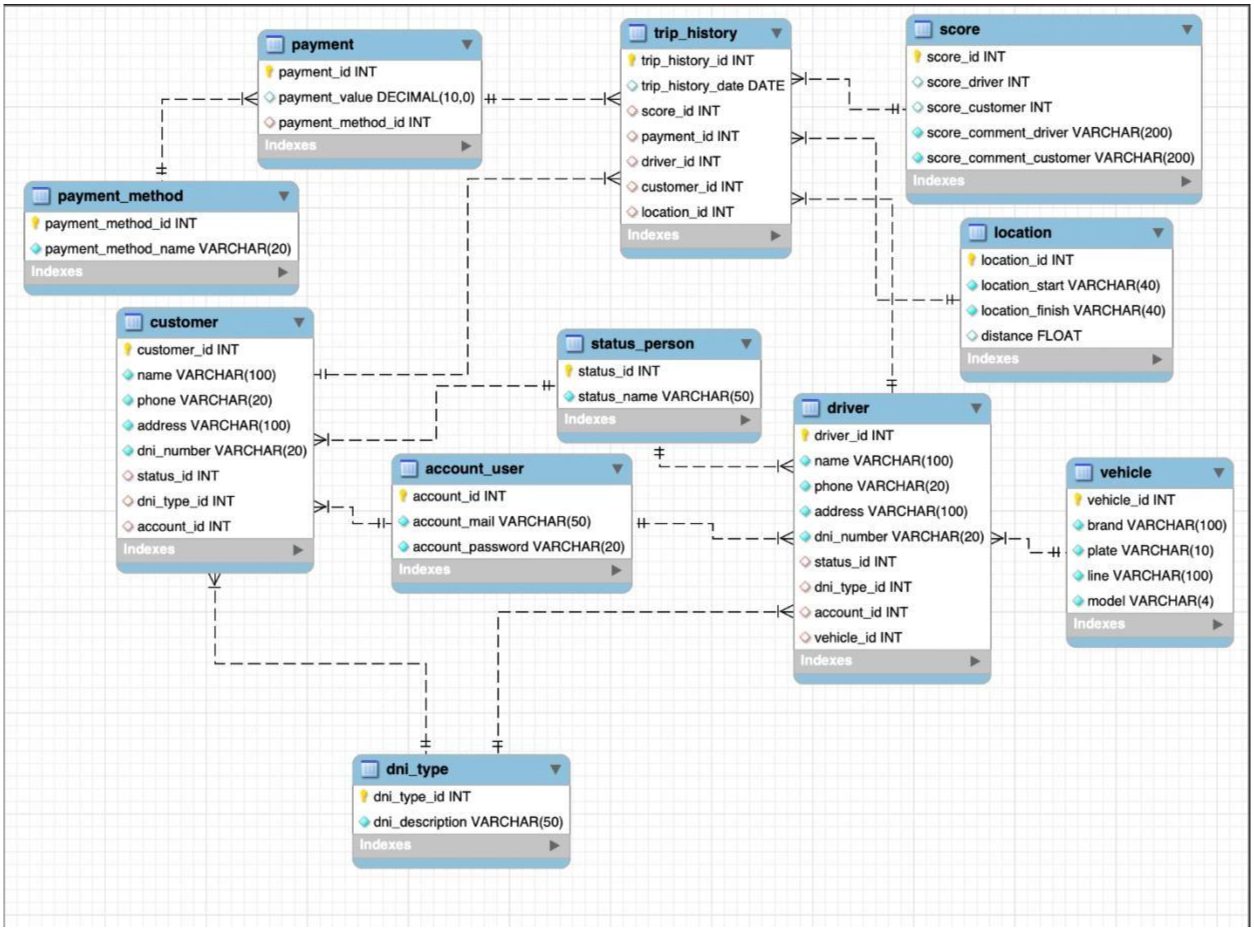
We ensured that attribute constraints were properly identified and relationships between tables were clearly defined.

To make sure the database is in Third Normal Form (3NF), we made sure that we followed the following steps:

1. Removed repeating groups from individual tables
2. Established separate tables for each related data
3. Used primary keys for each separate tables
4. Used foreign keys to connect the tables
5. Made sure that any field that does not depend on the key is removed

Figure 2.1 illustrates the Entity-Relationship (ER) diagram of the database following its creation.

Figure 2.1 Database ER diagram



Chapter 3. Data Description

For the creation of the data, different python scripts have been created, through which, by calling the faker library, which is a package that allows the creation of false or test data, it has been possible to obtain the data necessary for the development.

In the development it has been contemplated the necessity of being able to enter the data programmatically or by means of a csv file, this, by means of the script it is possible to carry out its obtaining as it is shown in the Appendix B.

Table 3.1 Quantity of data registers

Attribute	Number of Registers
customers	600
drivers	400
vehicles	400
location	2000
payment	2000
score	2000
Trip_history	2000
user	1000

To make sure the primary key of the table matches the foreign keys of other tables, attribute range and types were controlled. At the end, 11 separate tables of random values in different types of data were generated.

“LOCK TABLES” and “UNLOCK TABLES” commands are used for each data insertion to protect the data to be overwriting during the process.

Table 3.1 to Table 3.11 below describes the attributes within each table.

Table 3.2 Description of Attributes in table “vehicle”

Attribute	TYPE	Description
vehicle_id	Integer	Primary key of the table
brand	Varchar	
Plate	Varchar	
Line	Varchar	Car type

model	Varchar	4 digit number that identifies car model year

Table 3.3 Description of Attributes in table “status_person”

Attribute	TYPE	Description
status_id	Integer	Primary key of the table
status_name	Varchar	User status for both driver and customer

Table 3.4 Description of Attributes in table “account_user”

Attribute	TYPE	Description
account_id	Integer	Primary key of the table
account_mail	Varchar	User account login email address for both driver and customer
account_password	Varchar	User account login password for both driver and customer

Table 3.5 Description of Attributes in table “score”

Attribute	TYPE	Description
score_id	Integer	Primary key of the table
score_driver	Integer	The score that the customer gave to the driver for each trip

score_customer	Integer	The score that the driver gave to the customer for each trip
score_comment_driver	Varchar	The comment about a driver from customer
score_comment_customer	Varchar	The comment about a customer from a driver

Table 3.6 Description of Attributes in table “payment_method”

Attribute	TYPE	Description
payment_method_id	Integer	Primary key of the table
payment_method_name	Varchar	Payment method choices

Table 3.7 Description of Attributes in table “location”

Attribute	TYPE	Description
location_id	Integer	Primary key of the table
location_start	Varchar	Longitude and latitude information of starting point
location_finish	Varchar	Longitude and latitude information of finish point
distance	Float	Distance between starting and finishing points

Table 3.8 Description of Attributes in table “payment”

Attribute	TYPE	Description
payment_id	Integer	Primary key of the table
payment_value	Integer	Amounts that have been paid by each trip
payment_method_id	Integer	Method of payment by each payment

Table 3.9 Description of Attributes in table “driver”

Attribute	TYPE	Description
driver_id	Integer	Primary key of the table
name	Varchar	Driver’s name
phone	Varchar	Driver’s phone number
address	Varchar	Driver’s address
dni_number	Varchar	Driver’s identification number
status_id	Integer	Driver’s status, as some driver might be active, while some stopped driving taxi and account turned inactive
account_id	Integer	User account id
dni_type_id	Integer	DNI type that is used for identification
vehicle_id	Integer	Vehicle id that connect with vehicle information table

Table 3.10 Description of Attributes in table “customer”

Attribute	TYPE	Description
customer_id	Integer	Primary key of the table
name	Varchar	

phone	Varchar	
address	Varchar	
dni_number	Varchar	
status_id	Integer	
account_id	Integer	
dni_type_id	Integer	

Table 3.11 Description of Attributes in table “trip_history”

Attribute	TYPE	Description
trip_history_id	Integer	Primary key of the table
trip_history_date	Date	
score_id	Integer	
payment_id	Integer	
location_id	Integer	
customer_id	Integer	
driver_id	Integer	

Table 3.12 Description of Attributes in table “dni_type”

Attribute	TYPE	Description
dni_type_id	Integer	Primary key of the table
dni_description	Varchar	

Chapter 4. Data Manipulation

It is crucial to keep the data clean to make use of the data in the future. Thus, after the creation of the database, we checked if there was a missing value or a value in other formats. For example, attribute “name” in entity “customer” should not include “Mr.” or “Mrs.”, thus we checked if there’s a value including above by using the following formula:

```
SELECT
    c.name
    FROM
    final_project.customer AS c


WHERE
    c.name LIKE '%Mr%';
```

As an output, there were 7 rows that include “Mr.” or “Mrs.” in the customer name attribute. Thus, we used following query to update the attribute.

```
UPDATE final_project.customer AS c
SET c.name = TRIM(
     REPLACE(REPLACE(c.name, 'Mr. ', ''), 'Mrs. ', '')
)
WHERE
    c.name LIKE 'Mr. %' OR c.name LIKE 'Mrs. %';
```

As the entity “driver” also has “name” attribute, we did above steps to the “driver” table as well. As a result, the 5 rows were updated in the attribute.

Due to a misunderstanding during one trip, it was found that a customer and a driver were given each other a score of 1. But later the misunderstanding was resolved and both sides contacted the company to remove the score information, as it was greatly affecting the average score of both sides. According to customer ID and driver ID we found that the score_id that we need to delete is “84” as in Figure 4.1.

Figure 4.1 Trip history of driver_id=122 and customer_id=49

trip_history_date	score_id	driver_id	customer_id
2023-03-08	84	122	49

Thus, using the following query we deleted the score information.

without making any changes to other tables. Figure 4.2 shows the result of a deletion process.

DELETE FROM final_project.score

WHERE score_id=84;

Since we created the table “trip_history” with the following query, deletion will not affect the rows of trip_history, it will only show NULL value.

FOREIGN KEY (score_id) REFERENCES score(score_id) ON DELETE SET NULL,

Figure 4.2 Result of deleting

trip_history_date	score_id	driver_id	customer_id
2023-03-08	NULL	122	49

Chapter 5. Data Retrieval

Analysis on Drivers data

To analyze the performance of drivers and to keep the performance up, it is crucial to find the top contributors to the company’s revenue. Thus, we found out the top 10 drivers with highest amount of payment, their number of trips and average score compared to highest number of trips, average number of trips, highest average score and average score in general. For this purpose, multiple CTE's were used in a query.

Figure 5.1 CTE-1 Highest number of trips

```
WITH trips AS (
    SELECT
        th.driver_id,
        COUNT(th.driver_id) AS highest_number_of_trip
    FROM final_project.trip_history th
    GROUP BY th.driver_id
    ORDER BY highest_number_of_trip DESC
    LIMIT 1
),
```

Figure 5.2 CTE-2 Average number of trips

```
average_trips AS(
SELECT
    AVG(trip_numbers.number_of_trips) AS average_number_of_trips
    FROM (
        SELECT
        th.driver_id,
        COUNT(th.driver_id) AS number_of_trips
        FROM final_project.trip_history th
        GROUP BY th.driver_id) trip_numbers
),
```

Figure 5.3 CTE-3 Highest average score

```
highest_average_score AS(
SELECT
    th.driver_id,
    AVG(s.score_driver) AS highest_score
    FROM final_project.trip_history th
    JOIN final_project.score s ON s.score_id=th.score_id
GROUP BY th.driver_id
ORDER BY AVG(s.score_driver) DESC
LIMIT 1),
```

Figure 5.4 CTE-4 Average score

```
average_score AS(
SELECT
    AVG(s.score_driver) AS average_score
FROM final_project.score s
)
```

The main query is as follows:

Figure 5.5 Main query

```
SELECT
    d.name,
    th.driver_id,
    SUM(p.payment_value) AS total_payment,
    COUNT(th.driver_id) AS total_trips,
    AVG(s.score_driver) AS average_driver_score,
    trips.highest_number_of_trip,
    average_trips.average_number_of_trips,
    highest_average_score.highest_score,
    average_score.average_score
FROM final_project.trip_history th
JOIN final_project.driver d ON th.driver_id = d.driver_id
JOIN final_project.payment p ON th.payment_id = p.payment_id
JOIN final_project.score s ON s.score_id = th.score_id
JOIN trips
JOIN average_trips
JOIN highest_average_score
JOIN average_score
GROUP BY d.name, th.driver_id, trips.highest_number_of_trip,  average_trips.average_number_of_trips,
    highest_average_score.highest_score,
    average_score.average_score
ORDER BY SUM(p.payment_value) DESC
LIMIT 10;
```

Fig 5.6 Result of Query in Figure 5.1-Figure 5.5

name	driver_id	total_payment	total_trips	average_drive	highest_r	average_number	highest_avera	average_score
Christine Park	381	510	11	2.3636	11	5.1151	5.0000	2.9890
Danny Hubbard	173	503	9	3.1111	11	5.1151	5.0000	2.9890
Lisa Green	155	461	11	2.1818	11	5.1151	5.0000	2.9890
Kyle Gonzales	361	460	10	3.0000	11	5.1151	5.0000	2.9890
William Fox	140	447	8	1.7500	11	5.1151	5.0000	2.9890
Michelle Dillon	18	431	10	3.1000	11	5.1151	5.0000	2.9890
Cynthia Harper	310	431	10	3.1000	11	5.1151	5.0000	2.9890
Justin Bird	141	428	11	2.9091	11	5.1151	5.0000	2.9890
Ronnie Reyes	204	421	8	2.2500	11	5.1151	5.0000	2.9890
Monica Hamilton	179	421	8	3.0000	11	5.1151	5.0000	2.9890

Analysis on customer data

In the business, keeping the loyal customers' engagement and keeping their satisfaction high is important. To find out the loyal customers, we retrieved the top 10 customers with the highest number of trips, and we included the average score given to drivers and received from the drivers as well.

Figure 5.7 Query to retrieve the top 10 customers with the highest number of trips with average scores

```
SELECT
    th.customer_id,
    c.name AS customer_name,
    COUNT(th.trip_history_id) AS number_of_trips,
    AVG(s.score_customer) AS customer_score,
    AVG(s.score_driver) AS driver_score
FROM final_project.trip_history th
JOIN final_project.customer c ON c.customer_id=th.customer_id
JOIN final_project.score s ON s.score_id=th.score_id
GROUP BY th.customer_id, c.name
ORDER BY COUNT(th.trip_history_id) DESC
LIMIT 10;
```

Figure 5.8 Result of Query 5.7

customer_id	customer_name	number_of_trips	customer_score	driver_score
284	Timothy Hunt	9	2.4444	2.6667
488	Thomas Dunn	9	1.8889	3.5556
588	Sydney Lee DVM	9	3.2222	2.4444
458	Jeanne White	9	2.6667	3.3333
114	Timothy Guerrero	8	2.6250	3.1250
546	Jane Gutierrez DDS	8	3.3750	3.2500
158	Victor Richards	8	2.6250	3.3750
24	Jessica Foster	8	2.6250	3.5000
432	Kayla Moss	8	3.3750	3.0000
269	Jennifer Freeman	8	3.1250	2.7500

Analysis on trips dates

For the taxi service business, seeing the pattern of sales by month and analyzing if there's a correlation between a month and the number of calls is useful for predicting the future business plan. Thus, we retrieved the number of trips and the total amount of payments made by month to make it ready for the next step of analysis.

Figure 5.9 Query to retrieve number of trips and the total amount of payments made by month

```sql
SELECT
    YEAR(th.trip_history_date) AS year,
    MONTH(th.trip_history_date) AS month,
    COUNT(th.driver_id) AS number_of_trips,
    SUM(p.payment_value) AS total_amount_of_payment
FROM
    final_project.trip_history th
JOIN
    final_project.payment p ON th.payment_id=p.payment_id
GROUP BY
    YEAR(th.trip_history_date),
    MONTH(th.trip_history_date)
ORDER BY
    YEAR(th.trip_history_date) ASC,
    MONTH(th.trip_history_date) ASC;
```

As the query will be used every month to review the performances by month, we used VIEW, so that we don't have to write a query next time.

Figure 5.10 Query to create VIEW

```
CREATE VIEW Trip_Summary AS
SELECT
    YEAR(th.trip_history_date) AS year,
    MONTH(th.trip_history_date) AS month,
    COUNT(th.driver_id) AS number_of_trips,
    SUM(p.payment_value) AS total_amount_of_payment
FROM
    final_project.trip_history th
JOIN
    final_project.payment p ON th.payment_id = p.payment_id
GROUP BY
    YEAR(th.trip_history_date),
    MONTH(th.trip_history_date)
ORDER BY
    YEAR(th.trip_history_date) ASC,
    MONTH(th.trip_history_date) ASC;
```

Conclusion

The development of the taxi application database reflects a comprehensive and well-structured approach to data management and operational efficiency. The system employs an optimized SQL-based architecture with 11 interconnected tables, ensuring seamless integration and accurate relationships between key data entities such as customers, drivers, vehicles, payments, and trip history. By adhering to the principles of the Third Normal Form (3NF), the database reduces redundancy, maintains consistency, and strengthens data integrity, enabling accurate and efficient data retrieval.

The implementation of views enhances the system's analytical capabilities, allowing for the creation of reusable queries that provide critical business insights, such as tracking monthly trip activity, payment trends, and driver performance. These features empower the company to make data-driven decisions, streamline reporting, and support business forecasting. Additionally, the use of Python-generated synthetic data for testing purposes ensures system reliability and allows for comprehensive testing of database functionality.

The database system not only supports the operational needs of the taxi service but also provides valuable insights into customer preferences, driver performance, and financial metrics. This data-driven approach facilitates real-time decision-making, promotes efficiency, and enhances customer satisfaction. Ultimately, the system serves as a vital tool for supporting the long-term growth and strategic planning of the taxi service, providing a robust platform for future development and advanced analytics.

For a better experience in the data obtained regarding the trips made, it would be necessary to increase the number of these trips, so that each user and driver would have more data assigned to their record.

References

- Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, *13*(6), 377–387. https://doi.org/10.1145/362384.362685
- Date, C. J. (2009). *Database design and relational theory: Normal forms and relational database systems*. O'Reilly Media.
- Harrington, J. L. (2009). *Relational database design and implementation* (3rd ed.). Morgan Kaufmann/Elsevier.
- Open Text BC. (2024). *Chapter 12: Normalization – Database Design*. OpenTextBC. Retrieved from https://opentextbc.ca/dbdesign01/chapter/chapter-12-normalization/

Appendix A

Database creation SQL script

```
CREATE DATABASE  IF NOT EXISTS `final_project`;
use final_project;


CREATE TABLE final_project.vehicle (
   vehicle_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
   brand VARCHAR(100) NOT NULL DEFAULT "",
   plate VARCHAR(10) NOT NULL DEFAULT "",
   line VARCHAR(100) NOT NULL DEFAULT "",
   model VARCHAR(4) NOT NULL DEFAULT ""
);


CREATE TABLE final_project.status_person (
   status_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
   status_name VARCHAR(50) NOT NULL DEFAULT ""
);


CREATE TABLE final_project.dni_type (
   dni_type_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
   dni_description VARCHAR(50) NOT NULL DEFAULT ""
);
```

```
CREATE TABLE final_project.account_user (
    account_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    account_mail VARCHAR(50) NOT NULL DEFAULT "",
    account_password VARCHAR(20) NOT NULL DEFAULT ""
);

CREATE TABLE final_project.score (
    score_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    score_driver INT DEFAULT 3,
    score_customer INT DEFAULT 3,
    score_comment_driver VARCHAR(200) NOT NULL DEFAULT "",
    score_comment_customer VARCHAR(200) NOT NULL DEFAULT ""
);

CREATE TABLE final_project.payment_method (
    payment_method_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    payment_method_name VARCHAR(20) NOT NULL DEFAULT ""
);

CREATE TABLE final_project.location (
    location_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    location_start VARCHAR(40) NOT NULL DEFAULT "",
    location_finish VARCHAR(40) NOT NULL DEFAULT "",
```

```
    distance FLOAT
);

CREATE TABLE final_project.payment (
    payment_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    payment_value DECIMAL,
    payment_method_id INT,
    FOREIGN KEY (payment_method_id) REFERENCES
payment_method(payment_method_id) ON DELETE CASCADE
);

CREATE TABLE final_project.driver (
    driver_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL DEFAULT "",
    phone VARCHAR(20) NOT NULL DEFAULT "",
    address VARCHAR(100) NOT NULL DEFAULT "",
    dni_number VARCHAR(20) NOT NULL DEFAULT "",
    status_id INT,
    dni_type_id INT,
    account_id INT,
    vehicle_id INT,
    FOREIGN KEY (status_id) REFERENCES status_person(status_id) ON DELETE
CASCADE,
```

FOREIGN KEY (dni_type_id) REFERENCES dni_type(dni_type_id) ON DELETE CASCADE,

FOREIGN KEY (account_id) REFERENCES account_user(account_id) ON DELETE CASCADE,

FOREIGN KEY (vehicle_id) REFERENCES vehicle(vehicle_id) ON DELETE CASCADE

);

CREATE TABLE final_project.customer (

customer_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL DEFAULT "",

phone VARCHAR(20) NOT NULL DEFAULT "",

address VARCHAR(100) NOT NULL DEFAULT "",

dni_number VARCHAR(20) NOT NULL DEFAULT "",

status_id INT,

dni_type_id INT,

account_id INT,

FOREIGN KEY (status_id) REFERENCES status_person(status_id) ON DELETE CASCADE,

FOREIGN KEY (dni_type_id) REFERENCES dni_type(dni_type_id) ON DELETE CASCADE,

FOREIGN KEY (account_id) REFERENCES account_user(account_id) ON DELETE CASCADE

```
);

CREATE TABLE final_project.trip_history (
    trip_history_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    trip_history_date DATE,
    score_id INT,
    payment_id INT,
    driver_id INT,
    customer_id INT,
    location_id INT,
    FOREIGN KEY (score_id) REFERENCES score(score_id) ON DELETE SET NULL,
    FOREIGN KEY (payment_id) REFERENCES payment(payment_id) ON SET NULL,
    FOREIGN KEY (driver_id) REFERENCES driver(driver_id) ON DELETE
CASCADE,
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id) ON DELETE
CASCADE,
    FOREIGN KEY (location_id) REFERENCES location(location_id) ON DELETE SET
NULL
);
```

Appendix B

Scripts Python Fake Data

- **Creation of the .txt file**

```
from faker import Faker

import random



faker = Faker('en_CA')

num_records = 500 # Number of rows

# Name Of the file

output_file = 'taxi_customers.txt'

textOutput = 'INSERT INTO `customer` VALUES '

# Creatin of random data

for i in range(1, num_records + 1):

    name = faker.name()

    email = faker.email()

    phone = faker.phone_number()

    registration_date = faker.date_between(start_date='-3y', end_date='today')

    address = faker.city()
```

```
        payment_method = random.randint(1,4)

        trip_history = random.randint(1001, 5000)

        status = random.randint(1, 3)

        textOutput = "{} {}".format(textOutput, (i, name, email, phone, registration_date,
address, payment_method, trip_history, status))

with open(output_file, mode='w', newline='', encoding='utf-8') as file:

    file.write(textOutput)

print(f"Succesfull Data created...")
```

- Creation of the .csv file

```
import csv

from faker import Faker

import random

faker = Faker('en_CA')

num_records = 600 # Number of rows


# Name Of the file

output_file = 'taxi_customers.csv'
```

```
with open(output_file, mode='w', newline='', encoding='utf-8') as file:
    writer = csv.writer(file)
    # Write headers
    writer.writerow([
        "customer_id", "name", "mail", "phone",
        "registration_date", "address", "payment_method",
        "trip_history", "status"
    ])

    # Create fake data
    for i in range(1, num_records + 1):
        name = faker.name()
        email = faker.email()
        phone = faker.phone_number()
        registration_date = faker.date_between(start_date='-3y', end_date='today')
        address = faker.city()
```

```
        payment_method = random.randint(1,4)

        trip_history = random.randint(1001, 5000)

        status = random.randint(1, 3)



        writer.writerow([

            i, name, email, phone, registration_date,

            address, payment_method, trip_history, status

        ])

print(f"Succesfull Data created in csv...")
```