НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

(повна назва інституту/факультету)

КАФЕДРА інформатики та програмної інженерії

(повна назва кафедри)

**КУРСОВА РОБОТА**

з дисципліни «Бази даних»

(назва дисципліни)

на тему: База даних служби таксі

Студента \_\_2\_\_ курсу \_ІП-23\_\_\_\_\_ групи

спеціальності 121 «Інженерія програмного забезпечення»

\_Гончаренка Івана Юрійовича\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник \_\_Марченко Олена Іванівна\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_ Оцінка ECTS \_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ – 2023 рік

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

Факультет Інформатики та обчислювальної техніки

(повна назва)

Кафедра Інформатики та програмної інженерії

(повна назва)

Дисципліна Бази даних

Курс \_\_2\_\_\_ Група \_\_ІП-23\_\_ Семестр \_\_\_3\_\_

**З А В Д А Н Н Я**

**НА КУРСОВУ РОБОТУ СТУДЕНТУ**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Гончаренку Івану Юрійовичу\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові)

1. Тема роботи \_\_\_\_\_База даних служби таксі\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

керівник роботи \_\_Марченко Олена Іванівна\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом роботи \_\_\_\_\_23.12.2023\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Вихідні дані до роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Розробити базу даних для служби таксі \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_1) Аналіз предметного середовища

2) Побудова ER-моделі

3) Побудова реляційної схеми з ER-моделі

4) Створення бази даних, у форматі обраної системи управління базою даних

5) Створення користувачів бази даних

6) Імпорт даних з використанням засобів СУБД в створену базу даних

7) Створення мовою SQL запитів

8) Оптимізація роботи запитів

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Дата видачі завдання\_\_\_\_08.11.2023**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів виконання курсового проекту | Строк виконання етапів проекту | Примітка |
| 1 | Аналіз предметного середовища | 12.11.2023 |  |
| 2 | Побудова ER-моделі | 3.12.2023 |  |
| 3 | Побудова реляційної схеми з ER-моделі | 7.12.2023 |  |
| 4 | Створення бази даних, у форматі обраної системи управління базою даних | 10.12.2023 |  |
| 5 | Створення користувачів бази даних | 12.12.2023 |  |
| 6 | Імпорт даних з використанням засобів СУБД в створену базу даних | 15.12.2023 |  |
| 7 | Створення мовою SQL запитів | 19.12.2023 |  |
| 8 | Оптимізація роботи запитів | 22.12.2023 |  |
| 9 | Оформлення пояснювальної записки | 25.12.2023 |  |
| 10 | Захист курсової роботи | 29.12.2023 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Студент \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_**Гончаренко І.Ю.**\_\_\_\_\_\_\_**

(підпис ) (прізвище та ініціали)

**Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_**\_Марченко О.І.\_**\_\_\_\_\_\_\_**

(підпис ) (прізвище та ініціали)

Зміст

[Вступ 5](#__RefHeading___Toc8225_3874632622)

[1 Опис предметного середовища 6](#__RefHeading___Toc20588_3372768573)

[2 Постановка завдання 9](#__RefHeading___Toc484_658664086)

[3 Побудова er-моделі 10](#__RefHeading___Toc484_6586640861)

[4 Вибір системи управління базами даних 17](#__RefHeading___Toc484_65866408611)

[5 Реалізація бази даних 18](#__RefHeading___Toc484_658664086111)

[6 Робота з базою даних 19](#__RefHeading___Toc484_6586640861111)

[Висновок 61](#__RefHeading___Toc484_65866408611112)

[Перелік посилань 62](#__RefHeading___Toc26265_658664086)

Вступ

Бази даних є досить важливими в багатьох галузях людської діяльності, адже можуть ефективно зберігати велику кількість структурованої інформації, яку відносно легко можна редагувати, видаляти, додавати нову та знаходити потрібну інформацію серед наявної. Бази даних широко використовуються в онлайн-торгівлі, фінансах, освіті, науці, державних і приватних установах.

Для виконання курсової роботи я обрав MySQL як систему керування базами даних, адже вона має високу продуктивність, відкритий програмний код та високий рівень безпеки.

1 Опис предметного середовища

Під час реєстрації кожен клієнт вказує своє прізвище, ім’я, а також номер телефону або адресу електронної пошти для підтвердження, для подальшого зв’язку та відновлення облікового запису. Ці дані зберігаються в базі даних у сутності «**Клієнт».** Незважаючи на це, номер телефону та адресу електронної пошти можна змінити, адже кожному клієнту присвоюється ідентифікаційний номер (далі ID).

Для поїздки клієнт обирає відповідний розділ та вказує дані поїздки (місто, куди, звідки, а також бажану ціну поїздки). Після цього, у базі даних служби таксі створюється сутність «**Замовлення»**, яка містить ці дані, а також автоматично додані дату та час, ID клієнта та статус замовлення.

Після цього, якщо в обраному клієнтом населеному пункті є вільні водії, вони можуть прийняти це замовлення. Якщо водій приймає замовлення, його ID додається до сутності «**Замовлення»**, а також змінюється статус замовлення на «Очікування водія».

Під час очікування на поїздку клієнт може відправити повідомлення водію чи іншому працівнику, а також отримати повідомлення від нього. Ці дані (ID працівника, ID клієнта, відправник (клієнт/працівник), та текст повідомлення) зберігаються в сутності «**Повідомлення»**.

Клієнт повинен оплатити замовлення або перед поїздкою або після її закінчення. Дані про оплату зберігаються в сутності «**Оплата»**, яка має такі атрибути: ID оплати (первинний ключ), ID замовлення (зовнішній ключ), дата й час оплати, статус оплати (оплачено або ні), початкова ціна поїздки, кінцева ціна поїздки, знижка (якщо вона є) та заробіток водія із поїздки.

Після того, як водій таксі приїде до пункту призначення, а замовлення матиме статус «Оплачено», статус замовлення зміниться на «Виконано», а клієнту буде запропоновано оцінити поїздку та водія за 5-бальною шкалою, а також написати додаткову інформацію, що буде збережено в сутності «**Відгук».**

Дані про працівників служби таксі такі як ID працівника, посада, ПІБ, номер телефону, адреса електронної пошти, а також номер картки (для отримання зарплати) зберігаються в сутності «**Працівник таксі»**. А в сутності «**Зарплата за місяць»** зберігаються дані про те, скільки годин певний працівник відпрацював із початку місяця, скільки він мав годин очікування (для водіїв), а також яку суму він заробив із початку місяця і яку частину із цього отримає собі. Також працівник може забронювати час роботи на певні години певного дня, що буде зберігатися в сутності «**Заброньований час роботи».**

Окремо серед працівників є сутність «**Водій»** для збереження інформації про водіїв таксі, такої як країна перебування, місто перебування, поточне розташування (координати), поточний стан можливості прийняття замовлень (може / не може), а також «**Автомобіль»** для інформації про автомобілі, якими користуються водії служби. Автомобіль може бути як власністю водія, так і власністю служби таксі, а відповідна сутність має такі атрибути як номер автомобіля (наприклад, AA1234BB), тип автомобіля (стандарт, комфорт або електро), рік випуску, кількість місць та ID водія, який використовує цей автомобіль.

Для використання бази даних передбачено три типи користувачів — клієнт таксі, водій таксі та адміністратор.

Клієнт служби таксі має права на перегляд інформації про водіїв та автомобілі, а також додавання

* відгуків
* повідомлень
* замовлень

Водій служби таксі має права на перегляд інформації про

* клієнтів
* водіїв
* працівників
* відгуки
* замовлення
* оплати
* зарплати
* автомобілі

та додавати зарезервований час, повідомлення або автомобіль.

Адміністратор таксі має повні права на керування базою даних таксі, включаючи операції перегляду, додавання, видалення, модифікації і тд.

Також передбачено можливість перевірки даних при додаванні, а саме

* невід’ємні значення для чисел
* створення перелічуваних типів даних
* обмеження оцінки водія номером від 1 до 5

2 Постановка завдання

Метою курсової роботи є розробка бази даних для зберігання інформації служби таксі на основі ER-моделі сутностей-зв’язків та описаного предметного середовища. Створені таблиці бази даних необхідно заповнити випадковими даними, створеними генератором, також необхідно створити користувачів бази даних на основі проаналізованого предметного середовища. Для заданих потреб користувачів потрібно написати запити мовою SQL, при виконанні яких будуть обрані дані з таблиць, створені представлення, збережені процедури, функції та тригери.

3 Побудова er-моделі

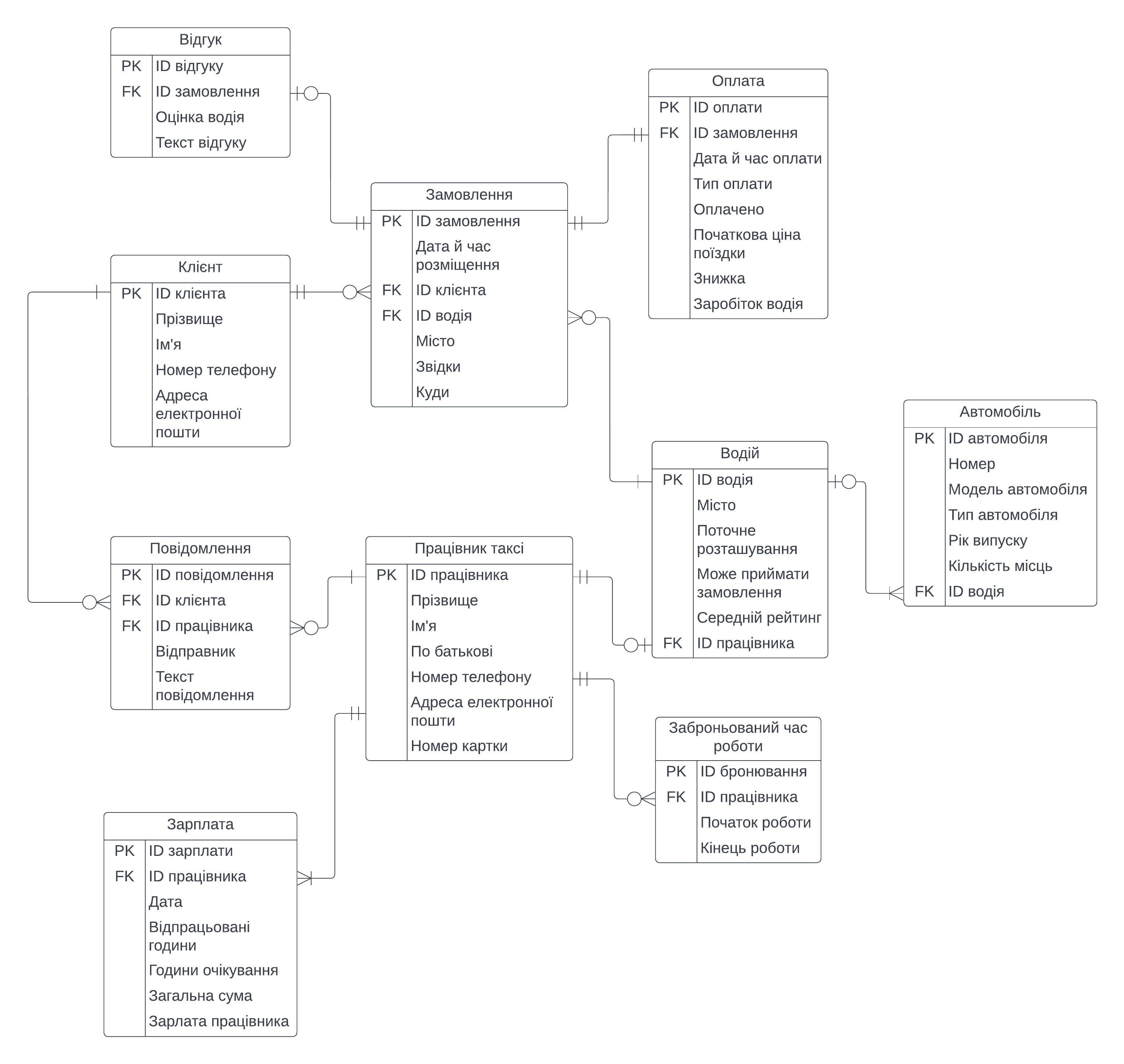
На основі аналізу предметного середовища можна виокремити такі сутності:

* Клієнт: дані про клієнта такі як прізвище, ім’я, номер телефону, адреса електронної пошти та ID клієнта
* Замовлення: дані про замовлення такі як дата й час розміщення, ID клієнта та водія, місто, початкова, кінцева точки призначення та ID замовлення
* Оплата: дані про оплату такі як дата й час оплати, ID замовлення, тип оплати, статус, початкова ціна, знижка, заробіток водія та ID оплати
* Працівник таксі: дані про працівника такі як прізвище, ім’я, номер телефону, адреса електронної пошти, номер картки та ID працівника
* Водій: дані про водія такі як місто, поточне розташування, статус можливості прийняття замовлень, середній рейтинг, ID працівника та водія
* Автомобіль: дані про автомобіль такі як номер, модель, тип, рік випуску, кількість місць та ID
* Повідомлення: дані про повідомлення такі як ID клієнта та працівника, відправник (клієнт чи працівник), текст та ID
* Відгук: дані про повідомлення такі як ID замовлення, оцінка водія, текст та ID
* Зарплата за місяць: дані про зарплату такі як ID працівника, дата, відпрацьовані години, години очікування, загальна сума, зарплата працівника та ID
* Заброньований час роботи: дані про час роботи такі як ID працівника, початок та кінець роботи, а також ID

Опишемо зв’язки між сутностями за допомогою таблиці:

|  |  |  |
| --- | --- | --- |
| Сутність №1 | Сутність №2 | Опис |
| Замовлення | Відгук | Один-до-одного |
| Оплата | Один-до-одного |
| Клієнт | Багато-до-одного |
| Водій | Багато-до-одного |
| Водій | Автомобіль | Один-до-багатьох |
| Працівник таксі | Один-до-одного |
| Працівник таксі | Заброньований час | Один-до-багатьох |
| Зарплата | Один-до-багатьох |
| Повідомлення | Один-до-багатьох |
| Клієнт | Повідомлення | Багато-до-одного |

**Отримана ER-модель**



Опишемо назви атрибутів бази даних та їх типу даних за допомогою таблиць:

Клієнт (clients)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Атрибут | | Тип даних | | Обмеження |
| ID клієнта | client\_id | | INT | PK |
| Прізвище | surname | | VARCHAR(50) | NOT NULL |
| Ім’я | name | | VARCHAR(50) | NOT NULL |
| Номер телефону | phone\_number | | BIGINT | UNSIGNED |
| Адреса ел. пошти | email | | VARCHAR(50) |  |

Працівник таксі (employees)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Атрибут | | Тип даних | | Обмеження |
| ID працівника | employee\_id | | INT | PK |
| Прізвище | surname | | VARCHAR(50) | NOT NULL |
| Ім’я | name | | VARCHAR(50) | NOT NULL |
| Номер телефону | phone\_number | | BIGINT |  |
| Адреса ел. пошти | email | | VARCHAR(50) |  |
| Номер картки | card\_number | | BIGINT |  |

Водій (drivers)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Атрибут | | Тип даних | | Обмеження |
| ID водія | driver\_id | | INT | PK |
| Місто | city | | VARCHAR(25) | NOT NULL |
| Поточне розташування | current\_location | | POINT |  |
| Може приймати замовлення | can\_take\_order | | BOOLEAN | NOT NULL |
| Середній рейтинг | average\_rating | | FLOAT |  |
| ID працівника | employee\_id | | INT | FK |

Замовлення (orders)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Атрибут | | Тип даних | | Обмеження |
| ID замовлення | order\_id | | INT | PK |
| ID клієнта | client\_id | | INT | FK, NOT NULL |
| ID водія | driver\_id | | INT | FK |
| Дата й час розміщення | datetime | | DATETIME | NOT NULL |
| Місто | city | | VARCHAR(25) | NOT NULL |
| Звідки | starting\_location | | POINT | NOT NULL |
| Куди | ending\_location | | POINT | NOT NULL |

Оплата (payments)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Атрибут | | Тип даних | | Обмеження |
| ID оплвти | payment\_id | | INT | PK |
| ID замовлення | order\_id | | INT | FK, NOT NULL UNIQUE |
| Дата й час оплати | datetime | | DATETIME |  |
| Тип оплати | type | | ENUM(‘Card’, ‘Cash’) |  |
| Оплачено | is\_paid | | BOOLEAN |  |
| Початкова ціна | starting\_price | | DECIMAL(7, 2) |  |
| Знижка | discount | | DECIMAL(7, 2) |  |
| Заробіток водія | driver\_income | | DECIMAL(7, 2) |  |
| Атрибут | | Тип даних | | Обмеження |
| ID відгуку | feedback\_id | | INT | PK |
| ID замовлення | order\_id | | INT | FK, NOT NULL UNIQUE |
| Оцінка водія | driver\_score | | TINYINT | UNSIGNED NOT NULL,  CHECK (driver\_score >= 1 AND driver\_score <= 5) |
| Текст відгуку | text | | VARCHAR(500) |  |

Повідомлення (messages)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Атрибут | | Тип даних | | Обмеження |
| ID повідомлення | message\_id | | INT | PK |
| ID клієнта | client\_id | | INT | FK, NOT NULL |
| ID працівника | employee\_id | | INT | FK, NOT NULL |
| Відправник | sender | | ENUM(‘Employee’, ‘Client’) | NOT NULL |
| Текст повідомлення | text | | VARCHAR(500) | NOT NULL |

Зарплата (salaries)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Атрибут | | Тип даних | | Обмеження |
| ID зарплати | salary\_id | | INT | PK |
| ID працівника | employee\_id | | INT | FK, NOT NULL |
| Дата | working\_hours | | SMALLINT | UNSIGNED NOT NULL |
| Відпрацьовані години | waiting\_hours | | SMALLINT | UNSIGNED |
| Години очікування | total\_amount | | DECIMAL(8, 2) |  |
| Загальна сума | employee\_salary | | DECIMAL(8, 2) |  |
| Зарплата працівн. | date | | DATE | NOT NULL |
| Атрибут | | Тип даних | | Обмеження |
| ID бронювання | reservation\_id | | INT | PK |
| ID працівника | employee\_id | | INT | FK, NOT NULL |
| Початок роботи | start | | DATETIME | NOT NULL |
| Кінець роботи | end | | DATETIME | NOT NULL |

Автомобіль (vehicles)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Атрибут | | Тип даних | | Обмеження |
| Номер автомобіля | vehicle\_number | | VARCHAR(10) | PK |
| Модель | model | | VARCHAR(50) | NOT NULL |
| Рік випуску | production\_year | | YEAR | NOT NULL |
| Тип | type | | ENUM(‘Standart’, ‘Comfort’, ‘Electric’) | NOT NULL |
| Кількість місць | seats\_number | | TINYINT | NOT NULL |
| ID водія | driver\_id | | INT | FK, NOT NULL |

4 Вибір системи управління базами даних

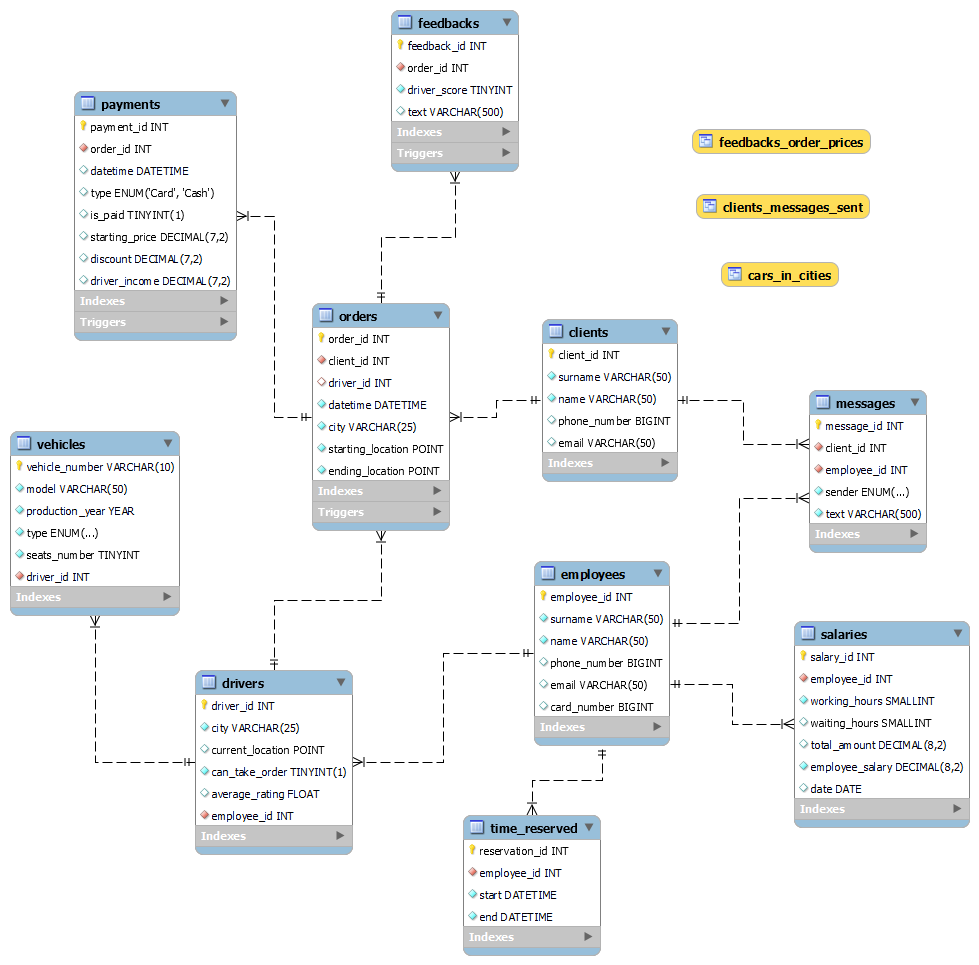
Для написання курсової роботи я обрав СУБД MySQL Цей вибір обґрунтовується його зручністю, надійністю, розширюваністю та широким спектром функціоналу.

MySQL є продуктом з відкритим кодом, що дозволяє безкоштовно використовувати та модифікувати його за власними потребами. Це особливо важливо для студентів, які можуть вільно експериментувати та навчатися без обмежень. Також MySQL володіє простим та інтуїтивно зрозумілим інтерфейсом, який полегшує розробку та адміністрування баз даних. Крім того, для MySQL існує велика кількість документації та активне спільнотове співтовариство, що сприяє швидкому вирішенню проблем та підтримці.

MySQL володіє високою надійністю та швидкодією, що робить його ідеальним вибором для великих та навіть критичних застосунків. Вбудовані механізми кешування та оптимізації запитів забезпечують ефективну роботу бази даних. MySQL дотримується стандартів SQL, що робить його сумісним із загальноприйнятими стандартами та забезпечує переносимість даних між різними системами.

5 Реалізація бази даних

Після створення ER-моделі, опису предметного середовища створимо базу даних за допомогою SQL-скриптів (Додаток А) і згенеруємо модель даних за допомогою самої СУБД MySQL:

Рис. 4.1 Схема даних, згенерована СУБД

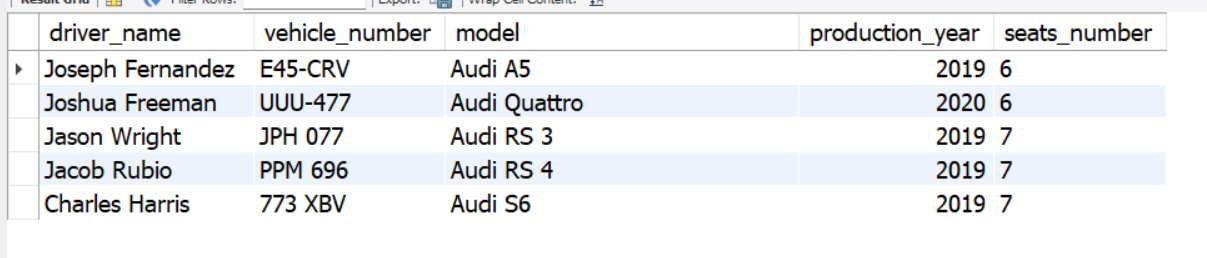
6 Робота з базою даних

Після створення фізичної БД, реалізуємо завдання, які були визначені на етапі постановки завдання.

1. Згенеруємо випадкові дані за допомогою написаною мною програми мовою Python (Додаток В)
2. Запишемо скрипти для вибірки даних:
   1. Дані про автомобілі із заданими параметрами:

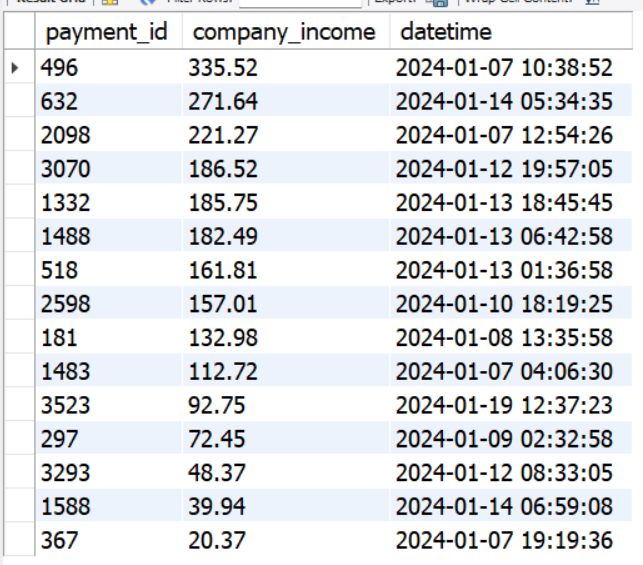
SELECT CONCAT(employees.name, " ", employees.surname) AS driver\_name,  
 vehicles.vehicle\_number,  
 vehicles.model,  
 vehicles.production\_year,  
 vehicles.seats\_number  
FROM vehicles  
JOIN drivers ON vehicles.driver\_id = drivers.driver\_id  
JOIN employees ON drivers.driver\_id = employees.employee\_id  
WHERE vehicles.seats\_number > 5  
 AND vehicles.model LIKE 'Audi%'  
 AND vehicles.production\_year >= 2019;

Результат:



* 1. SELECT   
      payment\_id,   
      (starting\_price - discount - driver\_income) AS company\_income,  
      p.datetime  
     FROM payments p  
     JOIN orders o ON p.order\_id = o.order\_id  
     WHERE   
      is\_paid = True AND city = "Boston"  
     ORDER BY company\_income DESC;

Результат:



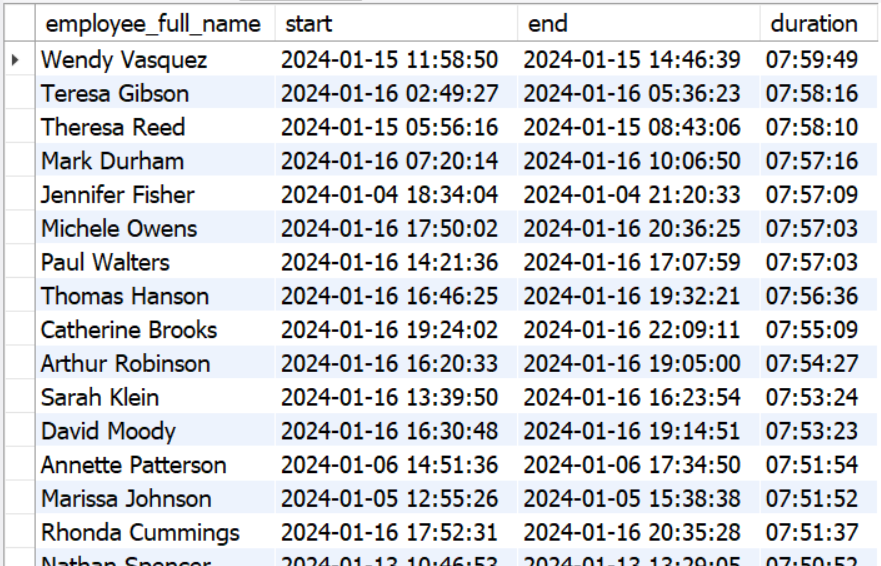
* 1. SELECT   
      (  
      SELECT CONCAT(name, " ", surname)   
      FROM employees   
      WHERE employees.employee\_id = salaries.employee\_id  
      ) AS employee\_full\_name,   
      working\_hours,   
      employee\_salary  
     FROM salaries  
     WHERE   
      employee\_salary > (SELECT AVG(employee\_salary) FROM salaries)  
     ORDER BY employee\_salary DESC;

Результат:

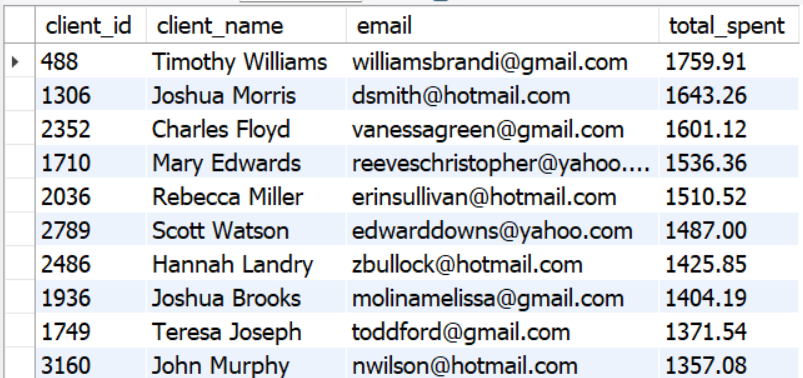


* 1. SELECT   
      (  
      SELECT CONCAT(name, " ", surname)  
      FROM employees WHERE employees.employee\_id = time\_reserved.employee\_id  
      ) AS employee\_full\_name,   
      start, end, SEC\_TO\_TIME(end - start) AS duration  
     FROM time\_reserved  
     WHERE SEC\_TO\_TIME(end - start) <= "08:00:00"  
     ORDER BY duration DESC;

Результат:

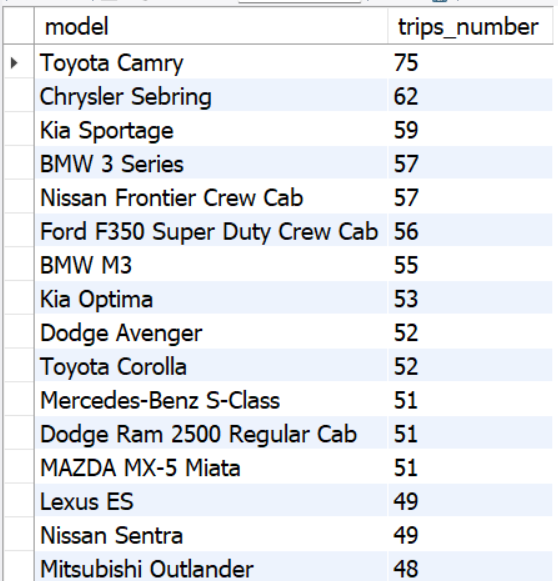


* 1. SELECT   
      c.client\_id,   
      CONCAT(c.name, " ", c.surname) AS client\_name,   
      c.email,  
      SUM(starting\_price - discount) AS total\_spent  
     FROM payments p  
     JOIN orders o ON o.order\_id = p.order\_id  
     JOIN clients c ON c.client\_id = o.client\_id  
     GROUP BY c.client\_id  
     ORDER BY total\_spent DESC  
     LIMIT 10;

Результат:

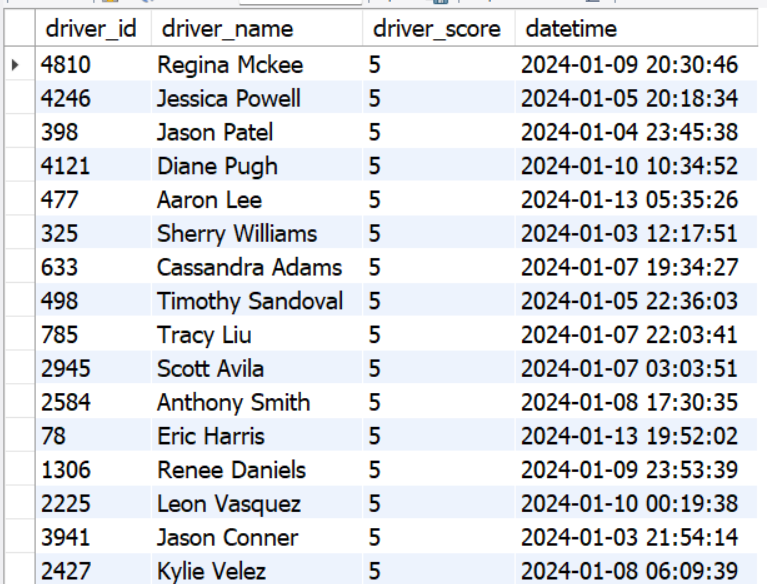
* 1. SELECT v.model, COUNT(v.model) AS trips\_number  
     FROM orders o  
     LEFT JOIN vehicles v ON o.driver\_id = v.driver\_id  
     GROUP BY v.model  
     ORDER BY trips\_number DESC;

Результат:



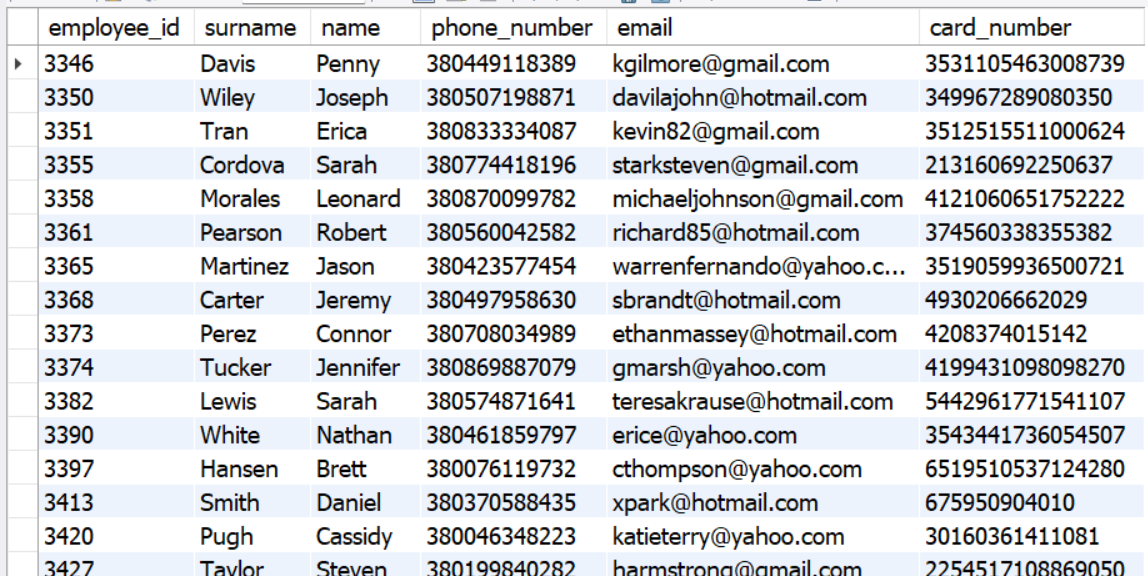
* 1. SELECT o.driver\_id, CONCAT(e.name, " ", e.surname) AS driver\_name,  
      driver\_score, o.datetime  
     FROM feedbacks f  
     JOIN orders o ON o.order\_id = f.order\_id  
     JOIN drivers d ON o.driver\_id = d.driver\_id  
     JOIN employees e ON e.employee\_id = d.employee\_id  
     ORDER BY driver\_score DESC;

Результат:



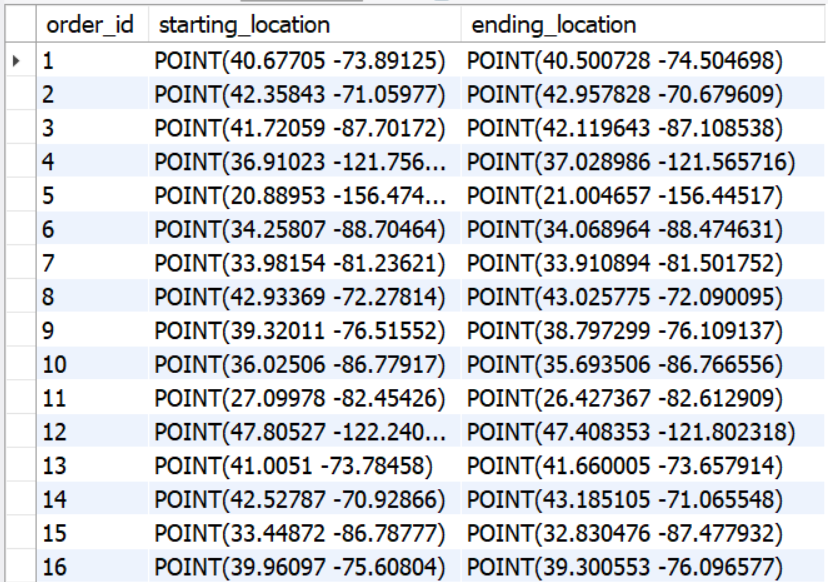
* 1. SELECT \* FROM employees  
     WHERE employee\_id   
     NOT IN (SELECT employee\_id FROM drivers);

Результат:



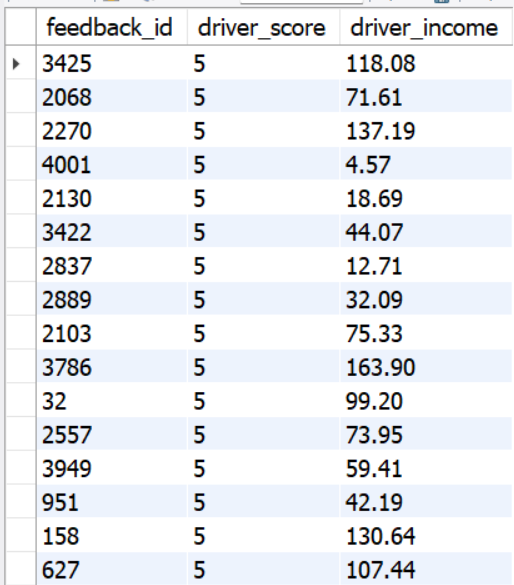
* 1. SELECT o.order\_id,   
      ST\_AsText(starting\_location) AS starting\_location,   
      ST\_AsText(ending\_location) AS ending\_location  
     FROM orders o  
     LEFT JOIN clients c ON c.client\_id = o.client\_id  
     LEFT JOIN drivers d ON d.driver\_id = o.driver\_id  
     LEFT JOIN employees e ON e.employee\_id = d.employee\_id;

Результат:



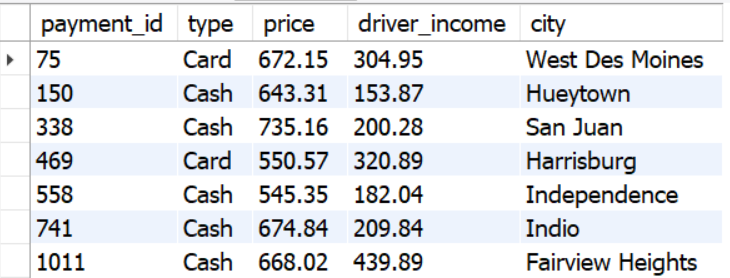
* 1. SELECT feedback\_id, driver\_score, driver\_income  
     FROM feedbacks f  
     JOIN payments p ON p.order\_id = f.order\_id  
     ORDER BY driver\_score DESC;

Результат:



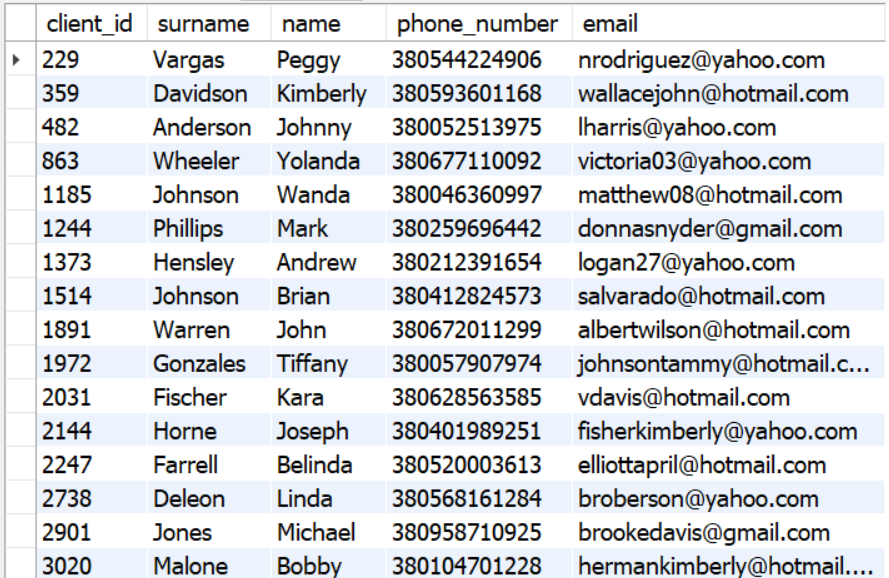
* 1. SELECT payment\_id, type, (starting\_price - discount) AS price, driver\_income, city  
     FROM payments  
     JOIN orders ON orders.order\_id = payments.order\_id  
     WHERE   
      starting\_price >= (SELECT AVG(starting\_price) FROM payments) \* 2.5;

Результат:



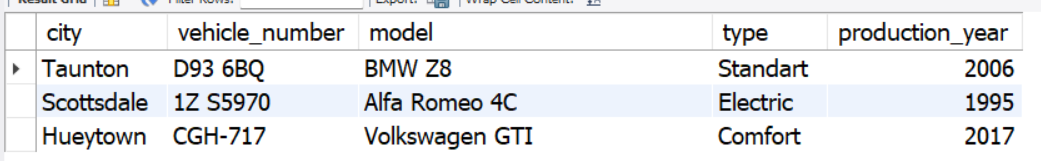
* 1. SELECT \* FROM clients  
     WHERE client\_id NOT IN (SELECT client\_id FROM orders);

Результат:



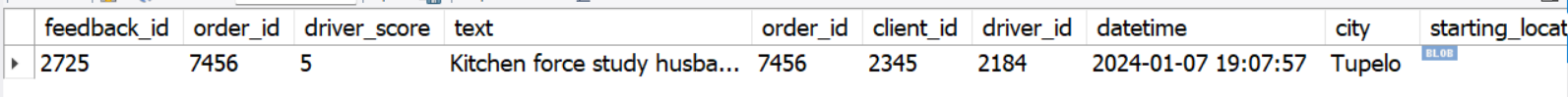
* 1. SELECT   
      city,   
      vehicle\_number, model, type, production\_year   
     FROM clients  
     JOIN orders ON orders.client\_id = clients.client\_id  
     JOIN vehicles ON vehicles.driver\_id = orders.driver\_id  
     WHERE clients.client\_id = 1234;

Результат:



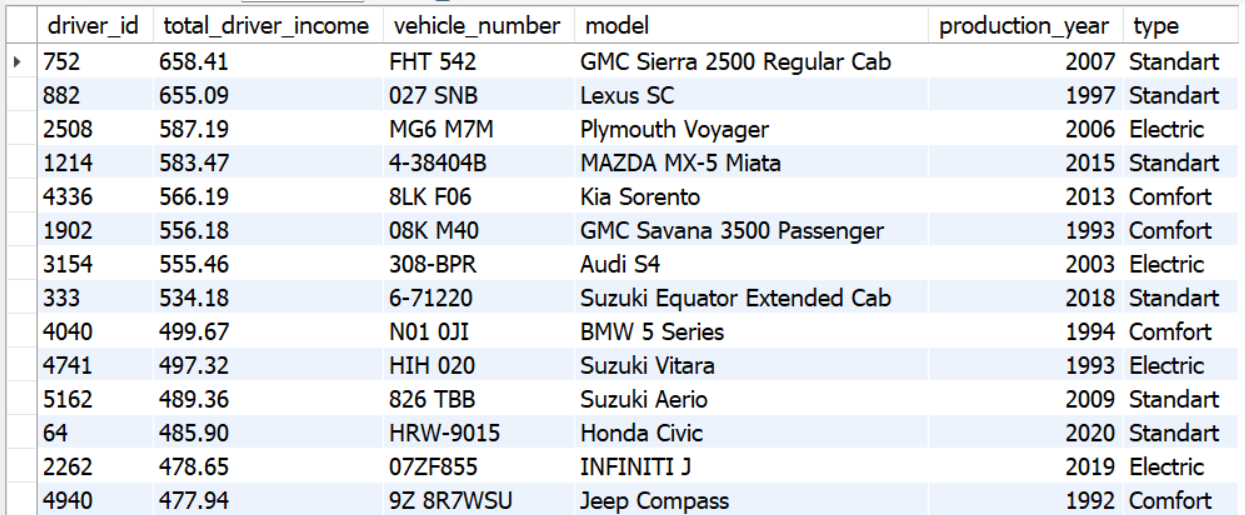
* 1. SELECT \* FROM feedbacks  
     JOIN orders ON feedbacks.order\_id = orders.order\_id  
     JOIN clients ON orders.client\_id = clients.client\_id  
     WHERE clients.client\_id = 2345;

Результат:



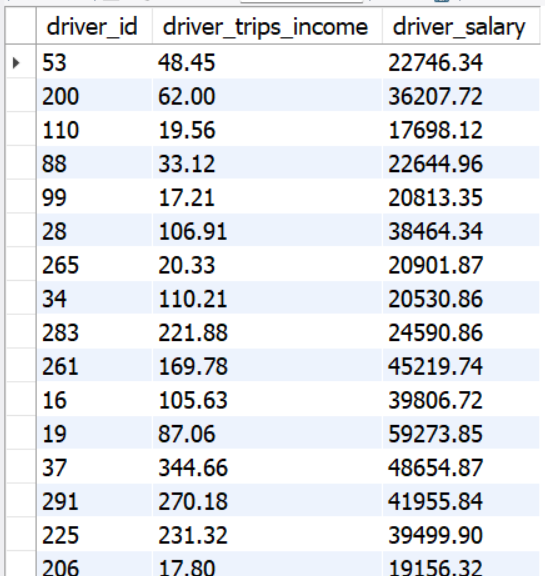
* 1. SELECT   
      vehicles.driver\_id,   
      SUM(driver\_income) AS total\_driver\_income,   
      vehicle\_number, model, production\_year, vehicles.type   
     FROM payments  
     JOIN orders ON orders.order\_id = payments.order\_id  
     JOIN vehicles ON vehicles.driver\_id = orders.driver\_id  
     GROUP BY vehicle\_number  
     ORDER BY total\_driver\_income DESC  
     LIMIT 100;

Результат:



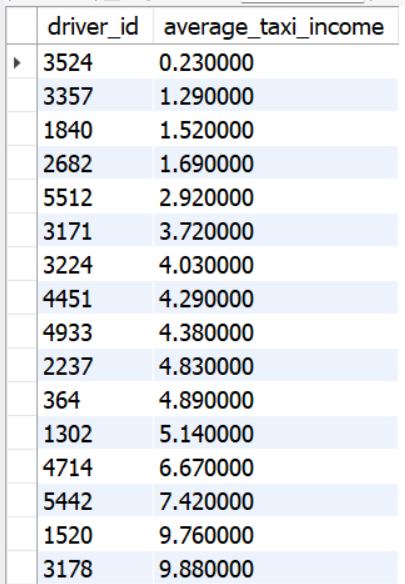
* 1. SELECT   
      orders.driver\_id,   
      SUM(driver\_income) AS driver\_trips\_income,   
      SUM(employee\_salary) AS driver\_salary  
     FROM payments  
     INNER JOIN orders ON orders.order\_id = payments.order\_id  
     INNER JOIN drivers ON drivers.driver\_id = orders.driver\_id  
     INNER JOIN salaries ON drivers.employee\_id = salaries.employee\_id  
     GROUP BY orders.driver\_id  
     HAVING orders.driver\_id <= 300;

Результат:



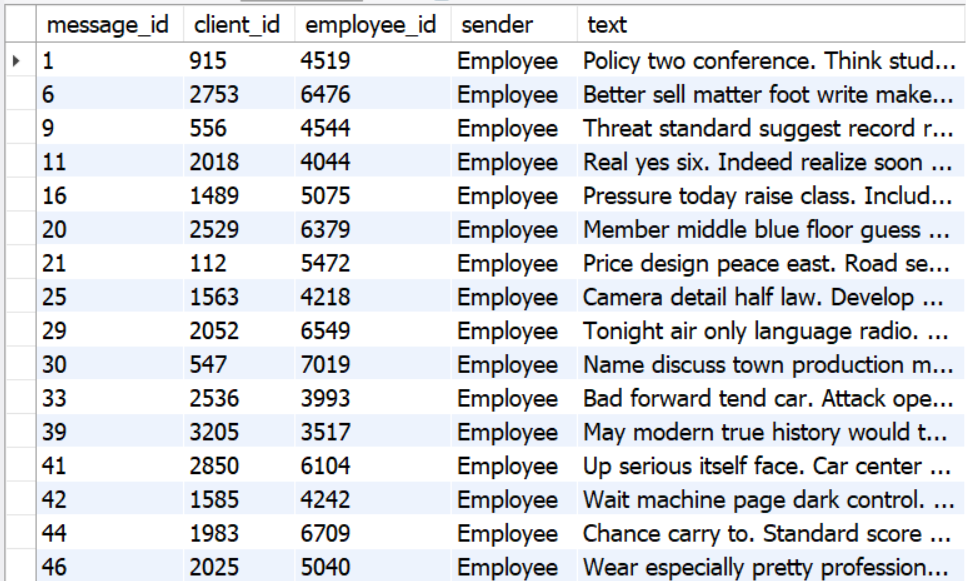
* 1. SELECT   
      drivers.driver\_id,   
      AVG(starting\_price - discount) AS average\_taxi\_income   
     FROM payments  
     INNER JOIN orders ON orders.order\_id = payments.order\_id  
     INNER JOIN drivers ON drivers.driver\_id = orders.driver\_id  
     GROUP BY drivers.driver\_id  
     HAVING average\_taxi\_income < 50  
     ORDER BY average\_taxi\_income ASC;

Результат:



* 1. SELECT messages.\*  
     FROM messages  
     JOIN clients ON messages.client\_id = clients.client\_id  
     WHERE sender = "Employee";

Результат:

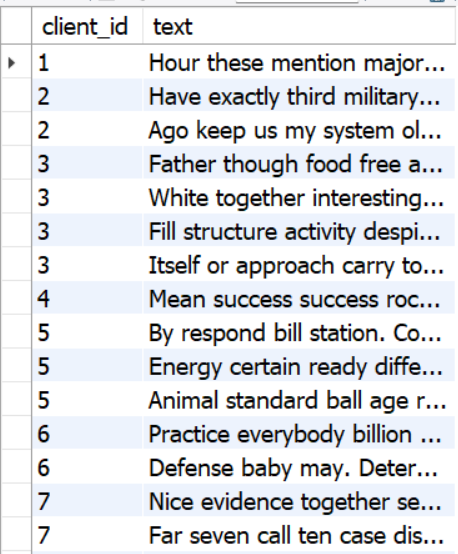


* 1. SELECT   
      message\_id,  
      CONCAT(c.name, " ", c.surname) AS client\_name,  
      CONCAT(e.name, " ", e.surname) AS driver\_name,  
      sender,  
      text  
     FROM clients c  
     JOIN messages m ON m.client\_id = c.client\_id  
     JOIN employees e ON e.employee\_id = m.employee\_id  
     JOIN drivers d ON d.employee\_id = m.employee\_id;

Результат:

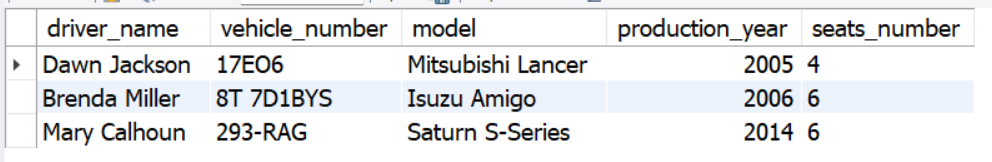


* 1. SELECT client\_id, text   
     FROM messages  
     WHERE sender = "Client"  
     UNION  
     SELECT orders.client\_id, text  
     FROM feedbacks  
     JOIN orders ON feedbacks.order\_id = orders.order\_id  
     ORDER BY client\_id;

Результат:

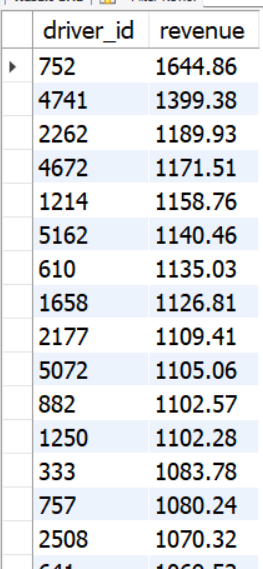
1. Запишемо скрипти для створення збережених процедур та функцій:
   1. DELIMITER $$  
      CREATE PROCEDURE get\_car\_by\_properties\_table(  
       IN car\_city VARCHAR(25), car\_type enum('Standart','Comfort','Electric'), min\_seats INT, min\_year INT)  
      BEGIN  
      DROP TABLE IF EXISTS car\_by\_properties\_table;  
      CREATE TEMPORARY TABLE IF NOT EXISTS car\_by\_properties\_table AS  
       SELECT CONCAT(employees.name, " ", employees.surname) AS driver\_name,  
       vehicles.vehicle\_number,  
       vehicles.model,  
       vehicles.production\_year,  
       vehicles.seats\_number  
       FROM vehicles  
       JOIN drivers ON vehicles.driver\_id = drivers.driver\_id  
       JOIN employees ON drivers.driver\_id = employees.employee\_id  
       WHERE   
       city = car\_city  
       AND vehicles.seats\_number >= min\_seats  
       AND type = car\_type  
       AND vehicles.production\_year >= min\_year;  
        
      END $$  
      DELIMITER ;  
        
      CALL get\_car\_by\_properties\_table("Boston", "Standart", 2, 2002);  
      SELECT \* FROM car\_by\_properties\_table;

Результат:



* 1. DELIMITER $$  
     CREATE PROCEDURE get\_top\_drivers\_by\_revenue(  
      IN n INT)  
     BEGIN  
     SELECT   
      orders.driver\_id,   
      SUM(starting\_price - discount) AS revenue  
     FROM payments  
     JOIN orders ON orders.order\_id = payments.order\_id  
     GROUP BY orders.driver\_id  
     ORDER BY revenue DESC  
     LIMIT n;  
     END $$  
     DELIMITER ;  
       
     CALL get\_top\_drivers\_by\_revenue(1000);

Результат:

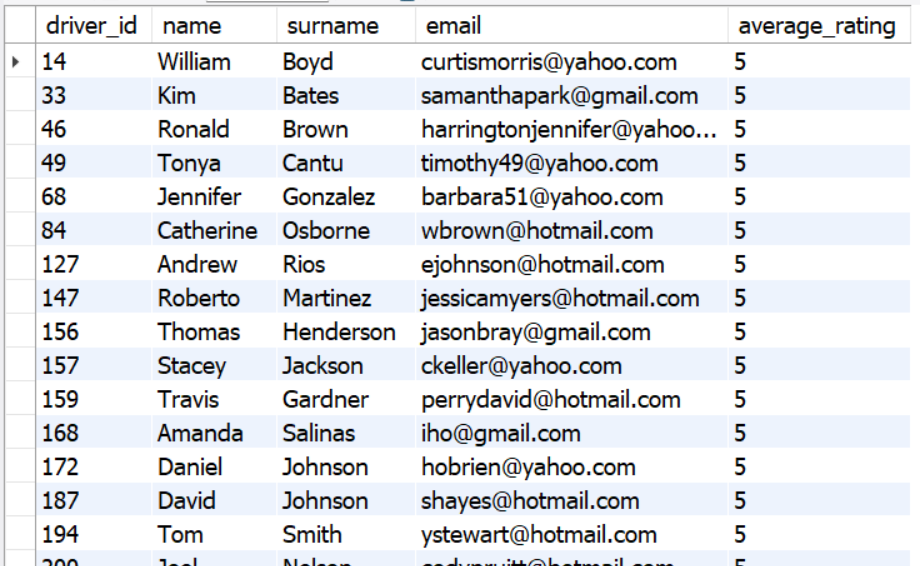


* 1. DELIMITER $$  
     CREATE PROCEDURE update\_average\_rating(  
      IN driverID INT  
     )  
     BEGIN  
      UPDATE drivers  
      SET average\_rating =   
      (  
      SELECT AVG(driver\_score) FROM feedbacks  
      JOIN orders ON orders.order\_id = feedbacks.order\_id  
      WHERE driver\_id = driverID  
      )  
      WHERE driver\_id = driverID;  
     END $$  
     DELIMITER ;  
       
     CALL update\_average\_rating(5);
  2. DELIMITER $$  
     CREATE PROCEDURE update\_all\_average\_ratings()  
     BEGIN  
      DECLARE drivers\_number INT DEFAULT 0;  
      DECLARE i INT DEFAULT 1;  
      SELECT COUNT(driver\_id)  
      INTO drivers\_number  
      FROM drivers;  
      HILE i <= drivers\_number DO  
      CALL update\_average\_rating(i);  
      SET i = i + 1;  
      END WHILE;  
      D $$  
     DELIMITER ;  
       
     CALL update\_all\_average\_ratings();

Результат:

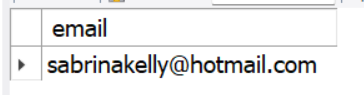
* 1. DELIMITER $$  
     CREATE PROCEDURE get\_highest\_ranked\_drivers()  
     BEGIN  
     SELECT driver\_id, name, surname, email, average\_rating   
     FROM drivers  
     JOIN employees ON drivers.employee\_id = employees.employee\_id  
     WHERE average\_rating = (SELECT MAX(average\_rating) FROM drivers)  
     ORDER BY average\_rating DESC;  
     END $$  
     DELIMITER ;  
       
     CALL get\_highest\_ranked\_drivers();

Результат:



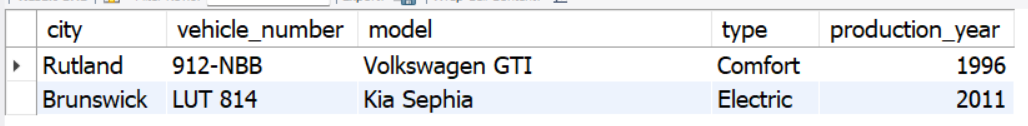
* 1. DELIMITER $$  
     CREATE PROCEDURE get\_client\_contact\_info(  
      IN id INT, IN contact\_type ENUM("email", "phone\_number")  
     )  
     BEGIN  
      IF (contact\_type = "email") THEN  
      SELECT email FROM clients WHERE client\_id = id;  
      ELSE  
      SELECT phone\_number FROM clients WHERE client\_id = id;  
      END IF;  
     END $$  
       
     DELIMITER ;  
       
     CALL get\_client\_contact\_info(3, "email");

Результат:



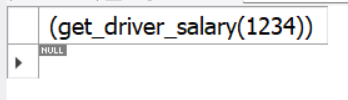
* 1. DELIMITER $$  
     CREATE PROCEDURE get\_used\_cars\_by\_client(  
      IN id INT  
     )  
     BEGIN  
      SELECT   
      city,   
      vehicle\_number, model, type, production\_year   
      FROM clients  
      JOIN orders ON orders.client\_id = clients.client\_id  
      JOIN vehicles ON vehicles.driver\_id = orders.driver\_id  
      WHERE clients.client\_id = id;  
     END $$  
     DELIMITER ;  
       
     CALL get\_used\_cars\_by\_client(228);

Результат:



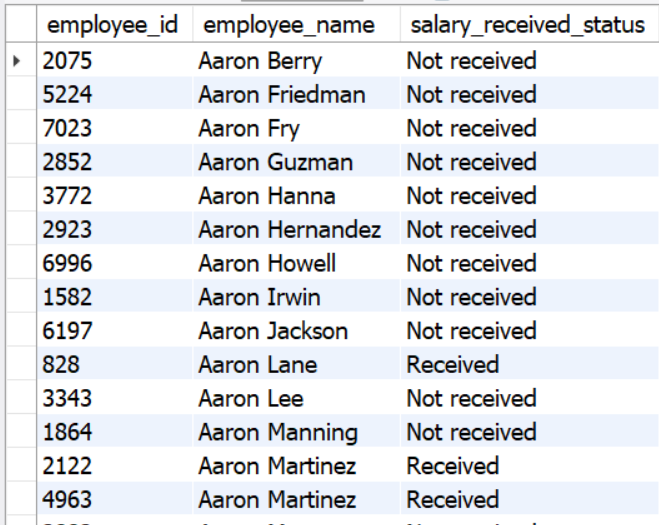
* 1. DELIMITER $$  
     CREATE FUNCTION get\_driver\_salary(  
      driverID INT  
     )  
     RETURNS DECIMAL(8, 2)  
     DETERMINISTIC  
     BEGIN  
      DECLARE salary DECIMAL(8, 2);  
      SELECT SUM(employee\_salary)   
      INTO salary  
      FROM salaries  
      JOIN drivers ON drivers.employee\_id = salaries.employee\_id  
      WHERE drivers.driver\_id = driverID;  
      RETURN salary;  
     END $$  
     DELIMITER ;  
       
     SELECT (get\_driver\_salary(1234));

Результат:



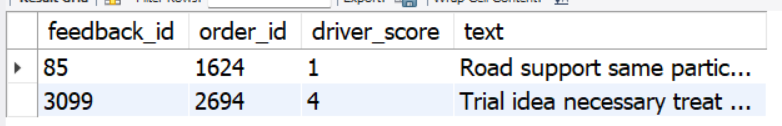
* 1. DELIMITER $$  
     CREATE FUNCTION salary\_received\_status(  
      employeeID INT  
     )  
     RETURNS ENUM("Received", "Not received")  
     DETERMINISTIC  
     BEGIN  
      DECLARE salaries\_number INT;  
      SELECT COUNT(date)   
      INTO salaries\_number  
      FROM salaries  
      WHERE employee\_id = employeeID;  
      IF salaries\_number = 0 THEN  
      RETURN "Not received";  
      ELSE   
      RETURN "Received";  
      END IF;END $$  
     DELIMITER ;

Результат:



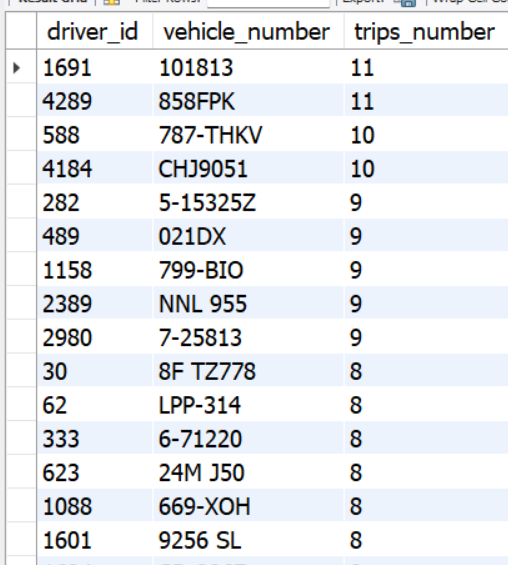
* 1. SELECT   
      employee\_id,   
      CONCAT(name, " ", surname) AS employee\_name,   
      salary\_received\_status(employee\_id) AS salary\_received\_status  
     FROM employees;  
       
     DROP PROCEDURE IF EXISTS get\_drivers\_feedback;  
     DELIMITER $$  
     CREATE PROCEDURE get\_drivers\_feedbacks (ID INT)  
       
     BEGIN  
      SELECT \* FROM feedbacks  
      WHERE order\_id IN (SELECT order\_id FROM orders WHERE orders.driver\_id = ID);  
     END $$  
       
     DELIMITER ;  
       
     CALL get\_drivers\_feedbacks(3456);

Результат:

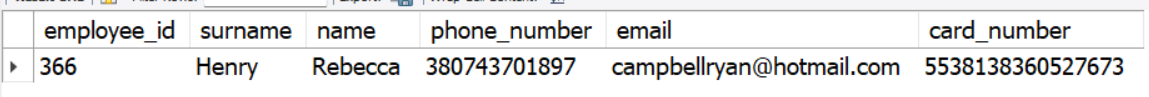


* 1. DELIMITER $$  
     CREATE PROCEDURE get\_top\_drivers\_by\_trips\_number()  
     BEGIN  
     SELECT vehicles.driver\_id, vehicle\_number, COUNT(vehicle\_number) AS trips\_number FROM vehicles  
     JOIN orders ON vehicles.driver\_id = orders.driver\_id  
     GROUP BY vehicles.driver\_id, vehicle\_number  
     ORDER BY trips\_number DESC;  
     END $$  
     DELIMITER ;  
       
     CALL get\_top\_drivers\_by\_trips\_number();

Результат:



* 1. DELIMITER $$  
     CREATE PROCEDURE get\_employee\_info (IN first\_name VARCHAR(50), IN last\_name VARCHAR(50))  
       
     BEGIN  
      SELECT \* FROM employees  
      WHERE name = first\_name AND surname = last\_name;  
     END $$  
       
       
     CALL get\_employee\_info("Rebecca", "Henry");  
     Результат:



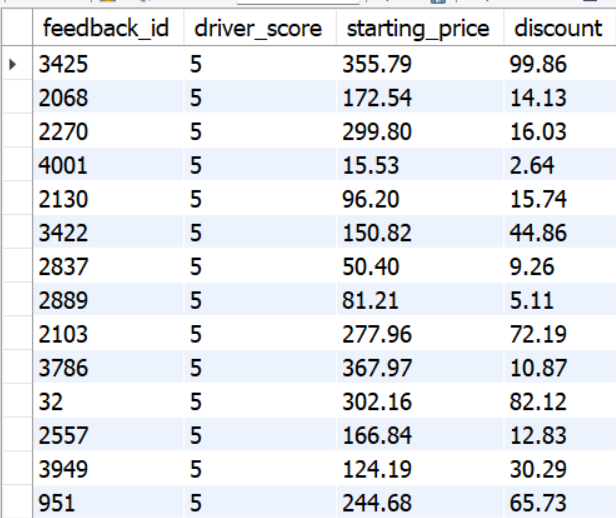
1. Запишемо скрипти для створення представлень:
   1. CREATE VIEW clients\_messages\_sent AS   
      SELECT   
       messages.client\_id,   
       CONCAT(clients.name, " ", clients.surname) AS client\_name,   
       COUNT(messages.client\_id) AS messages\_sent  
      FROM messages  
      JOIN clients ON messages.client\_id = clients.client\_id  
      GROUP BY messages.client\_id;

Результат:



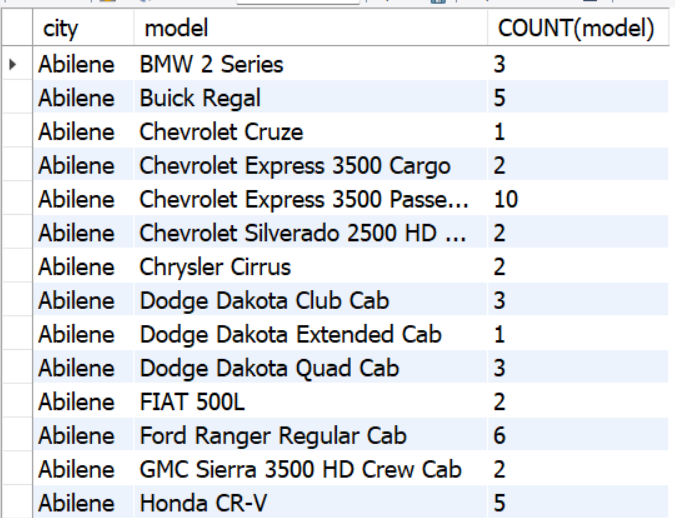
* 1. CREATE VIEW feedbacks\_order\_prices AS   
     SELECT feedback\_id, driver\_score, starting\_price, discount   
     FROM feedbacks f  
     JOIN payments p ON p.order\_id = f.order\_id  
     ORDER BY driver\_score DESC;

Результат:



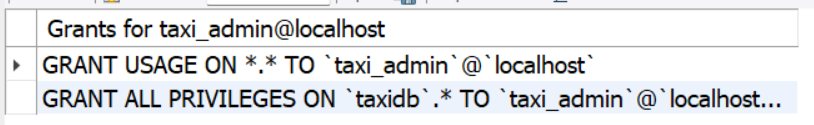
* 1. CREATE VIEW cars\_in\_cities AS   
     SELECT   
      city, model, COUNT(model) FROM clients  
     JOIN orders ON orders.client\_id = clients.client\_id  
     JOIN vehicles ON vehicles.driver\_id = orders.driver\_id  
     GROUP BY city, model  
     ORDER BY city;

Результат:



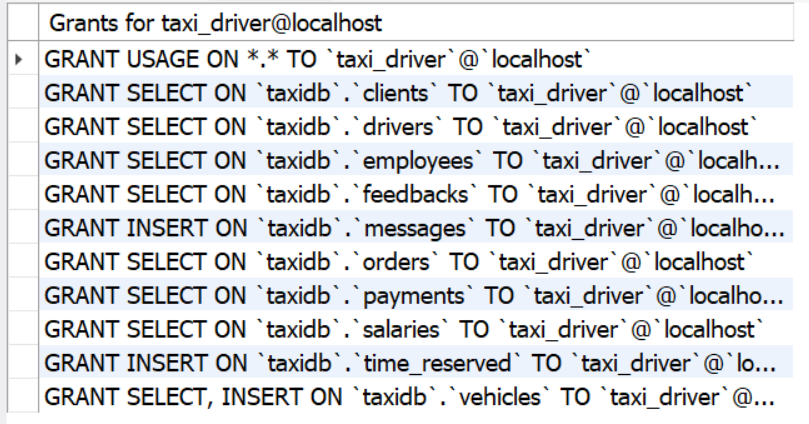
1. Запишемо скрипти для створення тригерів:
   1. DELIMITER $$  
      CREATE TRIGGER after\_driver\_score\_insert  
      AFTER INSERT ON feedbacks  
      FOR EACH ROW  
      BEGIN  
       DECLARE ID INT;  
       SELECT driver\_id INTO ID FROM orders WHERE order\_id = NEW.order\_id;  
       CALL update\_average\_rating(ID);  
      END $$  
      DELIMITER ;
   2. CREATE TRIGGER after\_order\_city\_insert  
      AFTER INSERT ON orders  
      FOR EACH ROW  
      UPDATE drivers  
      SET drivers.city = NEW.city  
      WHERE drivers.driver\_id = NEW.driver\_id;  
        
      DESC payments;
   3. DELIMITER $$  
      CREATE TRIGGER after\_order\_insert  
      AFTER INSERT ON orders  
      FOR EACH ROW  
      BEGIN  
       INSERT INTO payments (order\_id)  
       VALUES (NEW.order\_id);  
      END $$  
      DELIMITER ;
   4. DROP TRIGGER before\_payment\_insert  
      DELIMITER $$  
      CREATE TRIGGER after\_payment\_update  
      AFTER UPDATE ON payments  
      FOR EACH ROW  
      BEGIN  
       DECLARE orders\_number INT;  
       SELECT COUNT(order\_id)  
       INTO orders\_number  
       FROM orders  
       WHERE client\_id IN (SELECT client\_id FROM orders WHERE order\_id = NEW.order\_id);   
       IF orders\_number = 1 THEN  
       UPDATE payments  
       SET discount = starting\_price \* 0.5  
       WHERE payment\_id = NEW.payment\_id;  
       END IF;  
      END $$  
      DELIMITER ;
   5. DELIMITER $$  
      DROP TRIGGER before\_payment\_datetime\_insert;  
      CREATE TRIGGER before\_payment\_datetime\_update  
      BEFORE UPDATE ON payments  
      FOR EACH ROW  
      BEGIN  
       IF NEW.datetime IS NOT NULL THEN  
       SET NEW.is\_paid = TRUE;  
       END IF;  
      END $$  
      DELIMITER ;
2. Запишемо скрипти для створення користувачів і надання їм прав:
   1. CREATE USER 'taxi\_admin'@'localhost' IDENTIFIED BY 'admin228';  
      GRANT ALL PRIVILEGES ON taxiDB.\* TO taxi\_admin@localhost WITH GRANT OPTION;  
      SHOW GRANTS FOR [taxi\_admin@localhost](mailto:taxi_admin@localhost);

Результат:

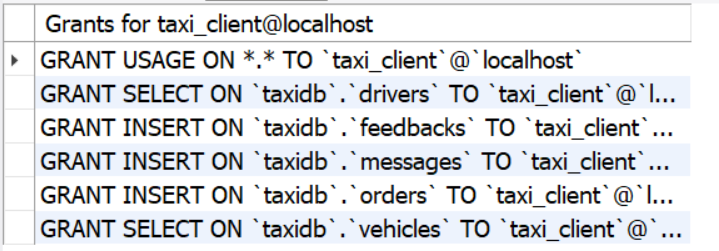


* 1. CREATE USER 'taxi\_driver'@'localhost' IDENTIFIED BY 'taxi228';  
       
     GRANT SELECT ON taxiDB.clients TO taxi\_driver@localhost;  
     GRANT SELECT ON taxiDB.drivers TO taxi\_driver@localhost;  
     GRANT SELECT ON taxiDB.employees TO taxi\_driver@localhost;  
     GRANT SELECT ON taxiDB.feedbacks TO taxi\_driver@localhost;  
     GRANT SELECT ON taxiDB.orders TO taxi\_driver@localhost;  
     GRANT SELECT ON taxiDB.payments TO taxi\_driver@localhost;  
     GRANT SELECT ON taxiDB.salaries TO taxi\_driver@localhost;  
       
     GRANT INSERT ON taxiDB.time\_reserved TO taxi\_driver@localhost;  
     GRANT INSERT ON taxiDB.messages TO taxi\_driver@localhost;  
       
     GRANT SELECT, INSERT ON taxiDB.vehicles TO taxi\_driver@localhost;  
     SHOW GRANTS FOR [taxi\_driver@localhost](mailto:taxi_driver@localhost);

Результат (2):



* 1. CREATE USER 'taxi\_client'@'localhost' IDENTIFIED BY 'taxi228';  
     SELECT user, host from mysql.user;  
       
     GRANT SELECT ON taxiDB.drivers TO taxi\_client@localhost;  
     GRANT INSERT ON taxiDB.feedbacks TO taxi\_client@localhost;  
     GRANT INSERT ON taxiDB.messages TO taxi\_client@localhost;  
     GRANT INSERT ON taxiDB.orders TO taxi\_client@localhost;  
     GRANT SELECT ON taxiDB.vehicles TO taxi\_client@localhost;  
     SHOW GRANTS FOR [taxi\_client@localhost](mailto:taxi_client@localhost);

Результат:

1. Запишемо скрипти для створення індексів (оптимізації):
   1. CREATE INDEX name\_surname\_idx

ON clients (name, surname);

* 1. CREATE INDEX name\_surname\_idx

ON employees (name, surname);

* 1. CREATE INDEX model\_year\_idx

ON vehicles (model, production\_year);

* 1. CREATE INDEX type\_seats\_number\_idx

ON vehicles (type, seats\_number);

* 1. CREATE INDEX city\_idx

ON drivers (city);

1. Запишемо скрипти для модифікування таблиць:
   1. ALTER TABLE vehicles  
      MODIFY COLUMN vehicle\_number VARCHAR(10);
   2. ALTER TABLE salaries  
      ADD COLUMN date DATE;
   3. ALTER TABLE payments  
      MODIFY datetime DATETIME;
   4. ALTER TABLE payments  
      MODIFY starting\_price DECIMAL(7, 2);
   5. ALTER TABLE payments  
      MODIFY driver\_income DECIMAL(7, 2);

Висновок

Під час виконання курсової роботи була виконана реалізація бази даних служби таксі. Спочатку було проаналізовано предметне середовище служби таксі. На основі предметного середовища було сформовано постановку задачі. Згідно неї було визначено необхідні сутності, зв’язки та типи користувачів. Також були розроблені тригери, обмеження, функції та процедури, які необхідні для правильного функціонування системи згідно предметної області. Для розробки використовувалася СУБД MySQL.

Перелік посилань

MySQL Tutorial - Learn MySQL Fast, Easy and Fun. MySQL Tutorial. URL: https://www.mysqltutorial.org/ (дата звернення: 22.01.2024).

Додаток А

CREATE DATABASE IF NOT EXISTS taxiDB;  
USE taxiDB;  
  
CREATE TABLE IF NOT EXISTS clients  
(  
 client\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 surname VARCHAR(50) NOT NULL,  
 name VARCHAR(50) NOT NULL,  
 phone\_number BIGINT UNSIGNED,  
 email VARCHAR(50)  
);  
  
CREATE TABLE IF NOT EXISTS employees  
(  
 employee\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 surname VARCHAR(50) NOT NULL,  
 name VARCHAR(50) NOT NULL,  
 patronymic VARCHAR(50),  
 phone\_number BIGINT UNSIGNED,  
 email VARCHAR(50),  
 card\_number BIGINT UNSIGNED  
);  
  
CREATE TABLE IF NOT EXISTS drivers  
(  
 driver\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 city VARCHAR(25) NOT NULL,  
 current\_location POINT,  
 can\_take\_order BOOLEAN NOT NULL,  
 average\_rating FLOAT,  
 employee\_id INT NOT NULL UNIQUE,  
 FOREIGN KEY (employee\_id) REFERENCES employees (employee\_id)  
);  
  
CREATE TABLE IF NOT EXISTS vehicles  
(  
 vehicle\_number VARCHAR(8) PRIMARY KEY,  
 model VARCHAR(50) NOT NULL,  
 production\_year YEAR NOT NULL,  
 type ENUM("Standart", "Comfort", "Electric") NOT NULL, -- типи авто  
 seats\_number TINYINT UNSIGNED NOT NULL,  
 driver\_id INT NOT NULL,  
 FOREIGN KEY (driver\_id) REFERENCES drivers (driver\_id)  
);  
  
CREATE TABLE IF NOT EXISTS salaries  
(  
 salary\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 employee\_id INT NOT NULL,  
 FOREIGN KEY (employee\_id) REFERENCES employees (employee\_id),  
 working\_hours SMALLINT UNSIGNED NOT NULL,  
 waiting\_hours SMALLINT UNSIGNED DEFAULT 0,  
 total\_amount DECIMAL(8, 2),  
 employee\_salary DECIMAL(8, 2) NOT NULL  
);  
  
CREATE TABLE IF NOT EXISTS time\_reserved  
(  
 reservation\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 employee\_id INT NOT NULL,  
 FOREIGN KEY (employee\_id) REFERENCES employees (employee\_id),  
 start DATETIME NOT NULL,  
 end DATETIME NOT NULL  
);  
  
CREATE TABLE IF NOT EXISTS messages  
(  
 message\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 client\_id INT NOT NULL,  
 FOREIGN KEY (client\_id) REFERENCES clients (client\_id),  
 employee\_id INT NOT NULL,  
 FOREIGN KEY (employee\_id) REFERENCES employees (employee\_id),  
 sender ENUM("Employee", "Client") NOT NULL,  
 text VARCHAR(500) NOT NULL  
);  
  
CREATE TABLE IF NOT EXISTS orders  
(  
 order\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 client\_id INT NOT NULL,  
 FOREIGN KEY (client\_id) REFERENCES clients (client\_id),  
 driver\_id INT,  
 FOREIGN KEY (driver\_id) REFERENCES drivers (driver\_id),  
 datetime DATETIME NOT NULL,  
 city VARCHAR(25) NOT NULL,  
 starting\_location POINT NOT NULL,  
 ending\_location POINT NOT NULL  
);  
  
CREATE TABLE IF NOT EXISTS payments  
(  
 payment\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 order\_id INT NOT NULL UNIQUE,  
 FOREIGN KEY (order\_id) REFERENCES orders (order\_id),  
 datetime DATETIME NOT NULL,  
 type ENUM("Card", "Cash") NOT NULL,  
 is\_paid BOOL DEFAULT FALSE,  
 starting\_price DECIMAL(7, 2) NOT NULL,  
 discount DECIMAL(7,2) DEFAULT 0,  
 driver\_income DECIMAL(7,2) NOT NULL  
);  
  
CREATE TABLE IF NOT EXISTS feedbacks  
(  
 feedback\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 order\_id INT NOT NULL UNIQUE,  
 FOREIGN KEY (order\_id) REFERENCES orders(order\_id),  
 driver\_score TINYINT UNSIGNED NOT NULL  
 CHECK (driver\_score >= 1 AND driver\_score <= 5), -- від 1 до 5--   
 text VARCHAR(500)  
);

Додаток B

import random  
from decimal import Decimal  
  
import mysql.connector  
from faker import Faker  
from faker\_vehicle import VehicleProvider  
  
db = mysql.connector.connect(  
 host="localhost",  
 user="taxi\_admin",  
 password="admin228",  
 database="taxiDB"  
)  
  
cursor = db.cursor()  
  
fake = Faker("en\_US")  
fake.add\_provider(VehicleProvider)  
  
  
def get\_location\_string(x, y):  
 location = f'ST\_GeomFromText("POINT({x} {y})")'  
 return location  
  
  
def get\_used\_ids(id\_name, table\_name):  
 cursor.execute(f"SELECT {id\_name} FROM {table\_name}")  
 used\_id = cursor.fetchall()  
 for i in range(0, len(used\_id)):  
 used\_id[i] = int(used\_id[i][0])  
 return used\_id  
  
  
def get\_not\_used\_ids(id\_name, table\_name):  
 used\_ids = get\_used\_ids(id\_name, table\_name)  
 cursor.execute(f"SELECT MAX({id\_name}) FROM {table\_name}")  
 max\_id = int(cursor.fetchone()[0])  
 all\_ids = list(range(1, max\_id + 1))  
 not\_used\_ids = [item for item in all\_ids if item not in used\_ids]  
 return max\_id, not\_used\_ids  
  
  
def get\_not\_drivers\_ids():  
 cursor.execute("SELECT employee\_id FROM employees WHERE employee\_id NOT IN (SELECT employee\_id FROM drivers);")  
 employee\_not\_driver\_id = cursor.fetchall()  
 for i in range(0, len(employee\_not\_driver\_id)):  
 employee\_not\_driver\_id[i] = int(employee\_not\_driver\_id[i][0])  
 return employee\_not\_driver\_id  
  
  
def get\_array\_from\_select(query):  
 cursor.execute(query)  
 array = cursor.fetchall()  
 for i in range(0, len(array)):  
 array[i] = int(array[i][0])  
 return array  
  
def get\_rows\_number(column\_name, table\_name):  
 cursor.execute(f"SELECT COUNT({column\_name}) FROM {table\_name}")  
 rows\_number = int(cursor.fetchone()[0])  
 return rows\_number  
  
  
def random\_phone\_number():  
 phone\_number = 3800  
  
 for i in range(9):  
 phone\_number += random.randint(0, 9)  
 phone\_number \*= 10  
  
 phone\_number = int(phone\_number / 10)  
 return phone\_number  
  
  
def generate\_random\_clients(rows\_number):  
 for i in range(rows\_number):  
 sql\_query = f'INSERT INTO clients (surname, name, phone\_number, email) VALUES ("{fake.last\_name()}", "{fake.first\_name()}", {random\_phone\_number()}, "{fake.ascii\_free\_email()}")'  
 # print(sql\_query)  
 cursor.execute(sql\_query)  
  
 db.commit()  
  
 print(f"{rows\_number} rows have been exported successfully to clients table")  
  
  
def generate\_random\_employees(rows\_number):  
 for i in range(rows\_number):  
 sql\_query = f'INSERT INTO employees (surname, name, phone\_number, email, card\_number) VALUES ("{fake.last\_name()}", "{fake.first\_name()}", {random\_phone\_number()}, "{fake.ascii\_free\_email()}", {fake.credit\_card\_number()})'  
 # print(sql\_query)  
 cursor.execute(sql\_query)  
  
 db.commit()  
  
 print(f"{rows\_number} rows have been exported successfully to employees table")  
  
  
def generate\_random\_drivers(rows\_number):  
 cursor.execute("SELECT COUNT(employee\_id) FROM employees")  
 n = int(cursor.fetchone()[0])  
  
 used\_id = get\_used\_ids("employee\_id", "drivers")  
 # indexes = list(range(444, 589))  
 # used\_driver\_ids = get\_used\_ids("driver\_id", "drivers")  
 # cursor.execute("SELECT MAX(driver\_id) FROM drivers")  
 # max\_id = int(cursor.fetchone()[0])  
 # all\_ids = list(range(1, max\_id + 1))  
 # not\_used\_ids = [item for item in all\_ids if item not in used\_driver\_ids]  
 # print(used\_id)  
 # print(not\_used\_ids)  
 # print(len(not\_used\_ids))  
 # for i in not\_used\_ids:  
 for i in range(rows\_number):  
 random\_location = fake.local\_latlng()  
 random\_id = random.choice([e for e in range(n) if e not in used\_id])  
 used\_id.append(random\_id)  
 # print(random\_id)  
  
 sql\_query = f'INSERT INTO drivers (city, current\_location, can\_take\_order, employee\_id) VALUES ("{random\_location[2]}", ST\_GeomFromText("POINT({float(random\_location[0])} {float(random\_location[1])})"), {bool(random.getrandbits(1))}, {random\_id})'  
 # sql\_query = f'INSERT INTO drivers (driver\_id, city, current\_location, can\_take\_order, employee\_id) VALUES ({i}, "{random\_location[2]}", ST\_GeomFromText("POINT({float(random\_location[0])} {float(random\_location[1])})"), {bool(random.getrandbits(1))}, {random\_id})'  
 print(sql\_query)  
 cursor.execute(sql\_query)  
 db.commit()  
  
 print(f"{rows\_number} rows have been exported successfully to drivers table")  
  
  
def generate\_random\_vehicles(rows\_number):  
 vehicle\_types = ["Standart", "Comfort", "Electric"]  
 cursor.execute("SELECT COUNT(driver\_id) FROM drivers")  
 n = int(cursor.fetchone()[0])  
 used\_id = get\_used\_ids("driver\_id", "vehicles")  
 # print(used\_id)  
 for i in range(n - 1):  
 random\_id = random.choice([e for e in range(n) if e not in used\_id])  
 used\_id.append(random\_id)  
 sql\_query = f'INSERT INTO vehicles (vehicle\_number, model, production\_year, type, seats\_number, driver\_id) VALUES ("{fake.license\_plate()}", "{fake.vehicle\_make\_model()}", {fake.vehicle\_year()}, "{random.choice(vehicle\_types)}", {random.randint(4, 7)}, {random\_id})'  
 print(sql\_query)  
 cursor.execute(sql\_query)  
 db.commit()  
  
 print(f"{rows\_number} rows have been exported successfully to employees table")  
  
  
def generate\_random\_orders(rows\_number):  
 cursor.execute("SELECT COUNT(driver\_id) FROM drivers")  
 drivers\_number = int(cursor.fetchone()[0])  
 cursor.execute("SELECT COUNT(client\_id) FROM clients")  
 clients\_number = int(cursor.fetchone()[0])  
  
 max\_id, not\_used\_ids = get\_not\_used\_ids("order\_id", "orders")  
 length = len(not\_used\_ids)  
  
 if not\_used\_ids and length > rows\_number:  
 this\_range = not\_used\_ids[0:rows\_number]  
 elif not\_used\_ids and length <= rows\_number:  
 this\_range = not\_used\_ids.extend(range(max\_id, max\_id + rows\_number - length))  
 else:  
 this\_range = range(rows\_number)  
  
 for i in this\_range:  
 random\_driver\_id = random.randint(1, drivers\_number)  
 random\_client\_id = random.randint(1, clients\_number)  
 random\_datetime = fake.date\_time\_this\_year()  
  
 cursor.execute(f"SELECT city FROM drivers WHERE driver\_id = {random\_driver\_id}")  
 city = str(cursor.fetchone()[0])  
  
 cursor.execute(  
 f"SELECT ST\_X(current\_location), ST\_Y(current\_location) FROM drivers WHERE driver\_id = {random\_driver\_id}")  
 start = cursor.fetchone()  
 start\_x = float(start[0])  
 start\_y = float(start[1])  
 starting\_location = get\_location\_string(start\_x, start\_y)  
  
 end\_x = fake.coordinate(start\_x, 0.7)  
 end\_y = fake.coordinate(start\_y, 0.7)  
 random\_ending\_location = get\_location\_string(end\_x, end\_y)  
 if not\_used\_ids:  
 sql\_query = f'INSERT INTO orders VALUES ({i}, {random\_client\_id}, {random\_driver\_id}, "{random\_datetime}", "{city}", {starting\_location}, {random\_ending\_location})'  
 else:  
 sql\_query = f'INSERT INTO orders (client\_id, driver\_id, datetime, city, starting\_location, ending\_location) VALUES ({random\_client\_id}, {random\_driver\_id}, "{random\_datetime}", "{city}", {starting\_location}, {random\_ending\_location})'  
 print(sql\_query)  
 cursor.execute(sql\_query)  
 db.commit()  
  
 print(f"{rows\_number} rows have been exported successfully to orders table")  
  
  
def generate\_random\_feedbacks(rows\_number):  
 cursor.execute("SELECT COUNT(order\_id) FROM orders")  
 n = int(cursor.fetchone()[0])  
 used\_order\_id = get\_used\_ids("order\_id", "feedbacks")  
 print(used\_order\_id)  
  
 for i in range(rows\_number):  
 random\_id = random.choice([e for e in range(n) if e not in used\_order\_id])  
 used\_order\_id.append(random\_id)  
  
 random\_driver\_score = random.randint(1, 5)  
 random\_text = fake.text(200)  
  
 sql\_query = f'INSERT INTO feedbacks (order\_id, driver\_score, text) VALUES ({random\_id}, {random\_driver\_score}, "{random\_text}");'  
 # print(sql\_query)  
 cursor.execute(sql\_query)  
 db.commit()  
  
 print(f"{rows\_number} rows have been exported successfully to feedbacks table")  
  
  
def generate\_random\_payments(rows\_number):  
 orders\_number = get\_rows\_number("order\_id", "orders")  
 used\_order\_id = get\_used\_ids("order\_id", "payments")  
 print(used\_order\_id)  
  
 type\_enum = ["Card", "Cash"]  
  
 for i in range(rows\_number):  
 random\_order\_id = random.choice([e for e in range(orders\_number) if e not in used\_order\_id])  
 used\_order\_id.append(random\_order\_id)  
  
 random\_type = random.choice(type\_enum)  
 cursor.execute(f"SELECT datetime FROM orders WHERE order\_id = {random\_order\_id}")  
 order\_datetime = cursor.fetchone()[0]  
 payment\_datetime = fake.date\_time\_between(order\_datetime)  
 # print(order\_datetime, payment\_datetime)  
  
 random\_paid\_status = bool(random.getrandbits(1))  
  
 random\_price = 100 + round(abs(random.normalvariate(200, 100)), 2)  
 random\_discount = round(random.uniform(0, 0.3) \* random\_price, 2)  
 random\_driver\_income = round(random.uniform(0.2, 0.7) \* (random\_price - random\_discount), 2)  
  
 sql\_query = f'INSERT INTO payments (order\_id, datetime, type, is\_paid, starting\_price, discount, driver\_income) VALUES ({random\_order\_id}, "{payment\_datetime}", "{random\_type}", {random\_paid\_status}, {random\_price}, {random\_discount}, {random\_driver\_income});'  
 print(sql\_query)  
 cursor.execute(sql\_query)  
 db.commit()  
  
 print(f"{rows\_number} rows have been exported successfully to feedbacks table")  
  
  
def generate\_random\_messages(rows\_number):  
 clients\_number = get\_rows\_number("client\_id", "clients")  
  
 not\_driver\_ids = get\_not\_drivers\_ids()  
  
 sender\_enum = ["Employee", "Client"]  
  
 max\_id, not\_used\_ids = get\_not\_used\_ids("message\_id", "messages")  
 length = len(not\_used\_ids)  
 #  
 print(not\_used\_ids)  
  
 if not\_used\_ids and length > rows\_number:  
 this\_range = not\_used\_ids[0:rows\_number]  
 elif not\_used\_ids and length <= rows\_number:  
 not\_used\_ids.extend(range(max\_id, max\_id + rows\_number - length))  
 this\_range = not\_used\_ids  
 else:  
 this\_range = range(rows\_number)  
 print(type(not\_used\_ids))  
 print(type(this\_range))  
  
 for i in this\_range:  
 random\_client\_id = random.randint(1, clients\_number + 1)  
  
 clients\_driver\_ids = get\_array\_from\_select(f"SELECT driver\_id FROM orders WHERE client\_id = {random\_client\_id}")  
  
 random\_employee\_id = random.choice(clients\_driver\_ids + not\_driver\_ids)  
  
 random\_sender = random.choice(sender\_enum)  
  
 random\_text = fake.text(100)  
 if not\_used\_ids:  
 sql\_query = f'INSERT INTO messages (message\_id, client\_id, employee\_id, sender, text) VALUES ({i}, {random\_client\_id}, {random\_employee\_id}, "{random\_sender}", "{random\_text}");'  
 else:  
 sql\_query = f'INSERT INTO messages (client\_id, employee\_id, sender, text) VALUES ({random\_client\_id}, {random\_employee\_id}, "{random\_sender}", "{random\_text}");'  
 print(sql\_query)  
 cursor.execute(sql\_query)  
 db.commit()  
  
 print(f"{rows\_number} rows have been exported successfully to feedbacks table")  
  
  
def generate\_random\_salaries(rows\_number):  
 employees\_number = get\_rows\_number("employee\_id", "employees")  
  
 sender\_enum = ["Employee", "Client"]  
  
 for i in range(rows\_number):  
 random\_employee\_id = random.randint(1, employees\_number)  
  
 random\_working\_hours = int(abs(random.normalvariate(40, 5)))  
 random\_waiting\_hours = int(abs(random.normalvariate(5, 1)))  
  
 cursor.execute(  
 f"SELECT (starting\_price - discount), driver\_income FROM payments WHERE order\_id IN (SELECT order\_id FROM orders WHERE driver\_id IN (SELECT employee\_id FROM drivers WHERE employee\_id = {random\_employee\_id}))")  
  
 driver\_income = 0  
 total\_income = 0  
 incomes = cursor.fetchall()  
 for income in incomes:  
 total\_income += income[0]  
 driver\_income += income[1]  
  
 random\_income = Decimal(abs(random.normalvariate(50000, 7000)))  
 total\_income = round(total\_income + random\_income, 2)  
 employee\_salary = round((driver\_income + random\_income \* Decimal(0.4)), 2)  
  
 random\_date = fake.date\_this\_year()  
  
 sql\_query = f'INSERT INTO salaries (employee\_id, working\_hours, waiting\_hours, total\_amount, employee\_salary, date) VALUES ({random\_employee\_id}, {random\_working\_hours}, {random\_waiting\_hours}, {total\_income}, {employee\_salary}, "{random\_date}");'  
 print(sql\_query)  
 cursor.execute(sql\_query)  
 db.commit()  
  
 print(f"{rows\_number} rows have been exported successfully to salaries table")  
  
  
def generate\_random\_time\_reserved(rows\_number):  
 employees\_number = get\_rows\_number("employee\_id", "employees")  
  
 for i in range(rows\_number):  
 random\_employee\_id = random.randint(1, employees\_number)  
  
 random\_start\_datetime = fake.date\_time\_this\_year()  
 random\_end\_datetime = fake.date\_time\_between(random\_start\_datetime)  
  
 sql\_query = f'INSERT INTO time\_reserved (employee\_id, start, end) VALUES ({random\_employee\_id}, "{random\_start\_datetime}", "{random\_end\_datetime}");'  
 print(sql\_query)  
 cursor.execute(sql\_query)  
 db.commit()  
  
 print(f"{rows\_number} rows have been exported successfully to time\_reserved table")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # print(get\_rows\_number("client\_id", "clients"))  
  
 # generate\_random\_time\_reserved(10000)  
 # print(fake.local\_latlng())  
 # print(get\_not\_drivers\_ids())  
 # print(get\_array\_from\_select("SELECT driver\_id FROM orders WHERE client\_id = 3"))  
 # generate\_random\_salaries(1000)  
 generate\_random\_payments(1000)  
 # generate\_random\_feedbacks(1000)  
 # generate\_random\_messages(2000)  
 # generate\_random\_orders(1000)  
 # generate\_random\_employees(1000)  
 # generate\_random\_drivers(1000)