Project Title: WanderWheel Car Rental System

Batch: B1

Group members:

UCE2023502: Advika Khorgade

UCE2023509: Anushka Bora

UCE2023514: Arya Davare

UCE2023520: Vrunda Deshpande

Software Requirements Specification (SRS)

1. Introduction

- **Project Title**: WanderWheel Car Rental System
- **Purpose**: This project aims to provide a car rental service where users can sign up, log in, view available cars, book a car, and manage their bookings. The system is database-driven, utilizing MySQL for data storage and management.
- **Scope**: The system covers basic user management, car availability, booking functionality, and booking history, with added automation through SQL functions, stored procedures, triggers, and events.

2. System Overview

Actors

- **User**: A registered user who can log in, view available cars, and make bookings.
- **System**: The database and application interface that handles user requests, car availability, and booking management.

Main Functions

1. **User Registration and Login**:

    - **Signup**: Allows users to create an account by providing details like username, password, and contact information.
    - **Login**: Enables existing users to log in using their username and password.
2. **Car Viewing**:
    - Displays a list of all cars that are currently available for booking.
3. **Car Booking**:
    - Allows users to book a car by selecting an available car, specifying the start and end dates for the booking, and viewing the total cost.
4. **Booking History**:
    - Moves completed bookings to a separate history table for reference and display completed bookings.
5. **Account Information**:
    - Users can view their ongoing bookings, past bookings, total spending, and high-cost bookings.

---

3. Functional Requirements

3.1 User Registration

- **Input**: Username, password, full name, phone number, and email.
- **Process**: Stores user details in the `Users` table.
- **Output**: Confirms successful signup.

3.2 User Login

- **Input**: Username and password.
- **Process**: Validates credentials against the `Users` table.
- **Output**: Confirms successful login or notifies of incorrect credentials.

3.3 Display Available Cars

- **Input**: None.

- **Process**: Queries the `Cars` table for cars with status `available`.
- **Output**: List of available cars with car ID, model, seating capacity, rate per day, and car number.

3.4 Car Booking

- **Input**: Car ID, start date, end date.
- **Process**:
    - Calculates the total cost based on the daily rental rate and duration ,using function CalculateTotalCost.
    - Stores booking details in the `Bookings` table.
    - Updates car status to `booked` in the `Cars` table.
- **Output**: Confirms booking and displays the total cost and assigned driver details.

Shows ongoing

3.5 Booking History and Status Management

- **Input**: User ID.
- **Process**:
    - Moves completed bookings to `BookingHistory` when the end date has passed.
    - Updates the car status to `available`.
- **Output**: completed bookings for the user.

3.6 Account Information

- **Total Spending**:
    - Retrieves the total amount spent by the user on bookings from `TotalSpendingView`.
- **High-Cost Bookings**:
    - Shows bookings where the cost is above the average booking cost

AverageBookingCost.

---

4. Non-Functional Requirements

- **Performance**: Ensures fast response times for login, booking, and data retrieval.
- **Security**: User passwords should be securely stored (e.g., hashed).
- **Reliability**: Ensures consistent data across tables with triggers and stored procedures for data integrity.
- **Usability**: Provides a simple, user-friendly console interface for easy interaction.

---

5. Database Requirements

5.1 Database Tables

- **Users**: Stores user account information.
- **Cars**: Contains details on each car and its status.
- **Bookings**: Stores ongoing booking information.
- **BookingHistory**: Archives completed bookings.
- **TotalSpending and highCostBooking View**: View that calculates the total spending for each user and spendings above average.

5.2 SQL Components

1. **Views**:
    - **TotalSpendingView**: Aggregates total spending by each user from `BookingHistory` to provide a summary of their expenditure.
    - **AverageBookingCost View**: Calculates the average booking cost across all bookings, allowing the system to identify high-cost bookings for a specific user.
2. **Functions**:
    - **CalculateTotalCost**: Calculates the total cost for a booking based on car rental rates and booking duration, returning the total cost to the application.

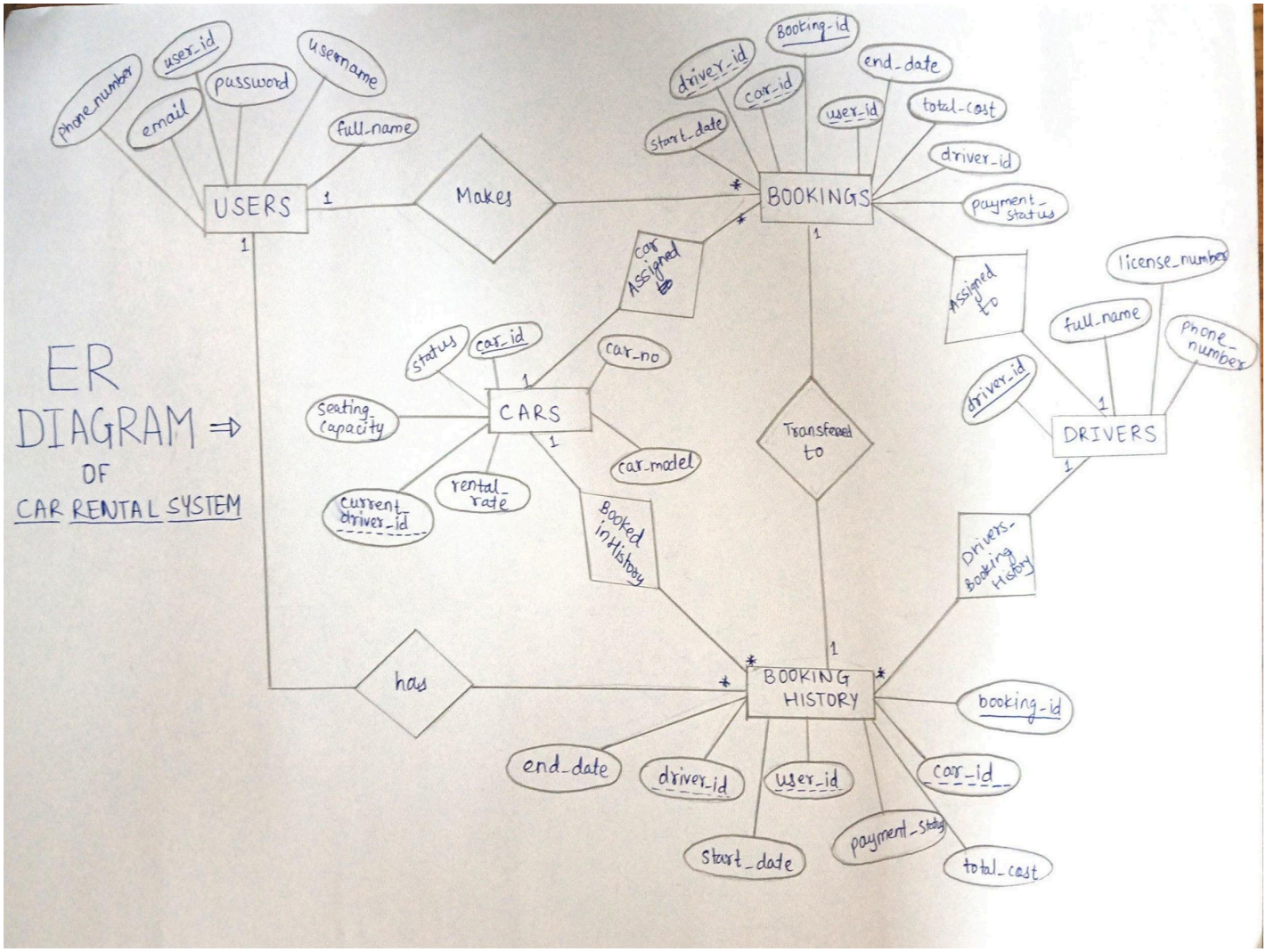
3. **Stored Procedures**:
    - **GetActiveBookings**: A stored procedure that retrieves ongoing bookings for a specific user, simplifying the retrieval of current bookings and keeping the SQL logic within the database.
4. **Triggers**:
    - **after_booking_move**: After a booking is transferred to `BookingHistory`, this trigger updates the car's status to `available` in the `Cars` table, ensuring that cars are automatically freed up once a booking ends.
5. **Events**:
    - **move_expired_bookings**: An event that runs daily to move expired bookings (where the end date is before today) from the `Bookings` table to `BookingHistory`. This keeps the system up-to-date with completed rentals.

---

6. System Requirements

- **Software**: MySQL database, Java (JDBC),java Swing.
- **Hardware**: Server with sufficient processing power for database operations.

ERD-


ER
DIAGRAM ⇒
OF
CAR RENTAL SYSTEM
USERS
BOOKINGS
CARS
DRIVERS
BOOKING HISTORY
Makes
Car Assigned to
Assigned to
Transferred to
Booked in History
Drivers Booking History
has
user_id
username
password
email
phone_number
full_name
Booking_id
driver_id
car_id
user_id
end_date
total_cost
start_date
driver_id
payment_status
status
car_id
car_no
seating_capacity
car_model
current_driver_id
rental_rate
license_number
full_name
phone_number
driver_id
booking_id
car_id
user_id
driver_id
end_date
start_date
payment_status
total_cost
1
*

TABLES

```
CREATE TABLE Users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    full_name VARCHAR(50),
    email VARCHAR(255) NOT NULL UNIQUE,
    phone_number VARCHAR(15) UNIQUE
);

CREATE TABLE Drivers (
    driver_id INT AUTO_INCREMENT PRIMARY KEY,
    full_name VARCHAR(255) NOT NULL,
    license_number VARCHAR(50) NOT NULL UNIQUE,
    phone_number VARCHAR(15) UNIQUE
);

CREATE TABLE Cars (
    car_id INT AUTO_INCREMENT PRIMARY KEY,
    car_model VARCHAR(50),
    seating_capacity INT,
    rental_rate DOUBLE,
    car_no VARCHAR(50) UNIQUE,  -- Ensuring unique car number
    status VARCHAR(20) DEFAULT 'available'  -- 'available' or 'booked'
);

CREATE TABLE BookingHistory (
    booking_id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT,
    car_id INT,
    driver_id INT,
    start_date DATE,
    end_date DATE,
    total_cost DECIMAL(10, 2),
    payment_status VARCHAR(20),
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (car_id) REFERENCES Cars(car_id),
    FOREIGN KEY (driver_id) REFERENCES Drivers(driver_id)
);

CREATE TABLE Bookings (
    booking_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
```

```
    car_id INT,
    driver_id INT,
    start_date DATE,
    end_date DATE,
    total_cost DOUBLE,
    payment_status VARCHAR(20),
    status VARCHAR(20) DEFAULT 'pending',  -- 'pending' or 'completed'
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (car_id) REFERENCES Cars(car_id),
    FOREIGN KEY (driver_id) REFERENCES Drivers(driver_id)
);
```

Normalisation

1. 1st Normal Form (1NF): Each table has a primary key, and all columns contain atomic values
   E.g-each phone_number column contains a single phone number per user or driver.

2. 2nd Normal Form (2NF): The tables have met 1NF and all non-primary key attributes are fully functionally dependent on the primary key. Each table has a primary key and there are no partial dependencies.

3. 3rd Normal Form (3NF): The tables have met 2NF, and there are no transitive dependencies. Each non-key attribute is only dependent on the primary key. For instance, the email and phone_number fields in the Users table depend solely on user_id.

So, all the above tables are optimized for relational integrity and efficiency, preventing redundancy of data.

SQL Queries

SQL

```
mysql> create database wanderwheel;
Query OK, 1 row affected (0.00 sec)

mysql> use wanderwheel;
Database changed
mysql> CREATE TABLE Users (
    -> user_id INT AUTO_INCREMENT PRIMARY KEY,
    -> username VARCHAR(255) NOT NULL,
    -> password VARCHAR(255) NOT NULL,
    -> full_name varchar(50),
    -> email VARCHAR(255) NOT NULL UNIQUE,
    -> phone_number VARCHAR(15) UNIQUE     );
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE Drivers (
    ->     driver_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     full_name VARCHAR(255) NOT NULL,
    ->     license_number VARCHAR(50) NOT NULL UNIQUE,
    ->     phone_number VARCHAR(15) UNIQUE);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO Drivers (full_name, license_number, phone_number)
    -> VALUES
    -> ('Amit Sharma', 'LIC123456', '9876543210' ),
    -> ('Priya Verma', 'LIC654321', '9876543201' ),
    -> ('Raj Kumar', 'LIC987654', '9876543192' ),
    -> ('Sita Patel', 'LIC112233', '9876543183'),
    -> ('Vikram Singh', 'LIC445566', '9876543174' ),
    -> ('Neha Yadav', 'LIC778899', '9876543165'),
```

```
   -> ('Ravi Gupta', 'LIC223344', '9876543156'),
   -> ('Rashmi Nair', 'LIC556677', '9876543147' ),
   -> ('Manoj Reddy', 'LIC889900', '9876543138'),
   -> ('Anjali Iyer', 'LIC334455', '9876543129' );
Query OK, 10 rows affected (0.00 sec)
Records: 10  Duplicates: 0  Warnings: 0



mysql> CREATE TABLE Cars (
   ->     car_id INT PRIMARY KEY AUTO_INCREMENT,
   ->     car_model VARCHAR(50),
   ->     seating_capacity INT,
   ->     rental_rate DOUBLE,
   ->     car_no VARCHAR(50),  -- car number
   ->     status VARCHAR(20) DEFAULT 'available',  -- 'available' or 'booked'
   ->     current_driver_id INT,
   ->     FOREIGN KEY (current_driver_id) REFERENCES Drivers(driver_id)
   -> );
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE Bookings (
   ->     booking_id INT PRIMARY KEY AUTO_INCREMENT,
   ->     user_id INT,
   ->     car_id INT,
   ->     driver_id INT,
   ->     start_date DATE,
   ->     end_date DATE,
   ->     total_cost DOUBLE,
   ->     payment_status VARCHAR(20),
   ->     status VARCHAR(20) DEFAULT 'pending',  -- 'pending' or 'completed'
   ->     FOREIGN KEY (user_id) REFERENCES Users(user_id),
   ->     FOREIGN KEY (car_id) REFERENCES Cars(car_id),
   ->     FOREIGN KEY (driver_id) REFERENCES Drivers(driver_id)
   -> );
Query OK, 0 rows affected (0.01 sec)



mysql> INSERT INTO Cars (car_model, seating_capacity, rental_rate, car_no, status)
   -> VALUES ('Sedan', 4, 500, 'KA01AB1234', 'available'),
   ->        ('Hatchback', 4, 450, 'KA02BC2345', 'available'),
   ->        ('Sedan', 4, 550, 'KA03CD3456', 'available'),
   ->        ('SUV', 4, 600, 'KA04DE4567', 'available');
Query OK, 4 rows affected (0.00 sec)
```

```
Records: 4  Duplicates: 0  Warnings: 0

mysql> INSERT INTO Cars (car_model, seating_capacity, rental_rate, car_no, status)
    -> VALUES ('MPV', 7, 900, 'KA05EF5678', 'available'),
    ->        ('MPV', 7, 950, 'KA06FG6789', 'available'),
    ->        ('SUV', 7, 1000, 'KA07GH7890', 'available'),
    ->        ('SUV', 7, 1050, 'KA08HI8901', 'available');
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0



mysql> INSERT INTO Cars (car_model, seating_capacity, rental_rate, car_no, status)
    -> VALUES ('Traveller', 17, 1500, 'KA09IJ9012', 'available'),
    ->        ('Traveller', 17, 1600, 'KA10JK0123', 'available');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select*from drivers;
+-----------+--------------+----------------+--------------+
| driver_id | full_name    | license_number | phone_number |
+-----------+--------------+----------------+--------------+
|         1 | Amit Sharma  | LIC123456      | 9876543210   |
|         2 | Priya Verma  | LIC654321      | 9876543201   |
|         3 | Raj Kumar    | LIC987654      | 9876543192   |
|         4 | Sita Patel   | LIC112233      | 9876543183   |
|         5 | Vikram Singh | LIC445566      | 9876543174   |
|         6 | Neha Yadav   | LIC778899      | 9876543165   |
|         7 | Ravi Gupta   | LIC223344      | 9876543156   |
|         8 | Rashmi Nair  | LIC556677      | 9876543147   |
|         9 | Manoj Reddy  | LIC889900      | 9876543138   |
|        10 | Anjali Iyer  | LIC334455      | 9876543129   |
+-----------+--------------+----------------+--------------+
10 rows in set (0.00 sec)


mysql> select*from cars;
+--------+-----------+-----------------+-------------+------------+-----------+-------------------+
| car_id | car_model | seating_capacity | rental_rate | car_no     | status    | current_driver_id |
+--------+-----------+-----------------+-------------+------------+-----------+-------------------+
|      1 | Sedan     |               4 |         500 | KA01AB1234 | available |              NULL |
|      2 | Hatchback |               4 |         450 | KA02BC2345 | available |              NULL |
|      3 | Sedan     |               4 |         550 | KA03CD3456 | available |              NULL |
|      4 | SUV       |               4 |         600 | KA04DE4567 | available |              NULL |
```

```
|      5 | MPV       |              7 |        900 | KA05EF5678 | available |             NULL |
|      6 | MPV       |              7 |        950 | KA06FG6789 | available |             NULL |
|      7 | SUV       |              7 |       1000 | KA07GH7890 | available |             NULL |
|      8 | SUV       |              7 |       1050 | KA08HI8901 | available |             NULL |
|      9 | Traveller |             17 |       1500 | KA09IJ9012 | available |             NULL |
|     10 | Traveller |             17 |       1600 | KA10JK0123 | available |             NULL |
+--------+-----------+----------------+------------+------------+-----------+------------------+
10 rows in set (0.00 sec)

mysql> update cars set current_driver_id = car_id;
Query OK, 10 rows affected (0.01 sec)
Rows matched: 10  Changed: 10  Warnings: 0

mysql> select*from cars;
+--------+-----------+----------------+------------+------------+-----------+------------------+
| car_id | car_model | seating_capacity | rental_rate | car_no     | status    | current_driver_id |
+--------+-----------+----------------+------------+------------+-----------+------------------+
|      1 | Sedan     |              4 |        500 | KA01AB1234 | available |                1 |
|      2 | Hatchback |              4 |        450 | KA02BC2345 | available |                2 |
|      3 | Sedan     |              4 |        550 | KA03CD3456 | available |                3 |
|      4 | SUV       |              4 |        600 | KA04DE4567 | available |                4 |
|      5 | MPV       |              7 |        900 | KA05EF5678 | available |                5 |
|      6 | MPV       |              7 |        950 | KA06FG6789 | available |                6 |
|      7 | SUV       |              7 |       1000 | KA07GH7890 | available |                7 |
|      8 | SUV       |              7 |       1050 | KA08HI8901 | available |                8 |
|      9 | Traveller |             17 |       1500 | KA09IJ9012 | available |                9 |
|     10 | Traveller |             17 |       1600 | KA10JK0123 | available |               10 |
+--------+-----------+----------------+------------+------------+-----------+------------------+
10 rows in set (0.01 sec)




mysql> ALTER TABLE Bookings
    -> DROP COLUMN status;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0



mysql> CREATE TABLE BookingHistory (
    ->     booking_id INT PRIMARY KEY,
    ->     user_id INT,
```

```
    ->     car_id INT,
    ->     driver_id INT,
    ->     start_date DATE,
    ->     end_date DATE,
    ->     total_cost DECIMAL(10, 2),
    ->     payment_status VARCHAR(20),
    ->     FOREIGN KEY (user_id) REFERENCES Users(user_id),
    ->     FOREIGN KEY (car_id) REFERENCES Cars(car_id),
    ->     FOREIGN KEY (driver_id) REFERENCES Drivers(driver_id)
    -> );
Query OK, 0 rows affected (0.02 sec)


(method makeBooking)
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION CalculateTotalCost(carId INT, startDate DATE, endDate DATE)
    -> RETURNS DECIMAL(10, 2)
    -> DETERMINISTIC
    -> BEGIN
    ->     DECLARE dailyRate DECIMAL(10, 2);
    ->     DECLARE totalCost DECIMAL(10, 2);
    ->
    ->     -- Get the rental rate for the selected car
    ->     SELECT rental_rate INTO dailyRate FROM Cars WHERE car_id = carId;

    ->
    ->     -- Calculate total cost based on the number of rental days
    ->     SET totalCost = DATEDIFF(endDate, startDate) * dailyRate;
    ->
    ->     RETURN totalCost;
    -> END //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //
mysql> CREATE EVENT IF NOT EXISTS move_expired_bookings
    -> ON SCHEDULE EVERY 1 DAY -- runs daily; adjust frequency as needed
    -> DO
    -> BEGIN
    ->     -- Move expired bookings to BookingHistory
    ->     INSERT INTO BookingHistory (booking_id, user_id, car_id, driver_id, start_date,
end_date, total_cost, payment_status)
    ->     SELECT booking_id, user_id, car_id, driver_id, start_date, end_date, total_cost,
payment_status
```

```
    ->     FROM Bookings
    ->     WHERE end_date < CURDATE();
    ->
    ->     -- Delete expired bookings from Bookings table
    ->     DELETE FROM Bookings WHERE end_date < CURDATE();
    ->    );
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)
```

(method `updateExpiredBookings`)

```
mysql> DELIMITER //
mysql> CREATE TRIGGER after_booking_move
    -> AFTER INSERT ON BookingHistory
    -> FOR EACH ROW
    -> BEGIN
    ->     UPDATE Cars
    ->     SET status = 'available'
    ->     WHERE car_id = NEW.car_id;
    -> END;
    -> //
Query OK, 0 rows affected (0.05 sec)

mysql> DELIMITER ;
```

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE GetActiveBookings(IN userId INT)
    -> BEGIN
    ->     SELECT booking_id, car_id, start_date, end_date
    ->     FROM Bookings
    ->     WHERE user_id = userId AND end_date >= CURDATE();
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
```

(method `displayTotalSpending`)

```
mysql> CREATE VIEW TotalSpendingView AS
    -> SELECT user_id, SUM(total_cost) AS total_spending
    -> FROM BookingHistory
```

```
    -> GROUP BY user_id;
    -> //
Query OK, 0 rows affected (0.01 sec)

(displayHighCostBookings)
mysql> delimiter //
mysql> delimiter ;
mysql> CREATE VIEW AverageBookingCost AS
    -> SELECT AVG(total_cost) AS avg_cost
    -> FROM Bookings;
Query OK, 0 rows affected (0.00 sec)
```

```
SELECT b.booking_id, u.full_name, b.start_date, b.end_date, b.total_cost
FROM Bookings b
JOIN Users u ON b.user_id = u.user_id
WHERE b.start_date BETWEEN '2024-01-01' AND '2024-12-31'
ORDER BY b.total_cost DESC;
```

```
SELECT u.user_id, u.full_name, SUM(bh.total_cost) AS total_spent
FROM Users u
INNER JOIN BookingHistory bh ON u.user_id = bh.user_id
GROUP BY u.user_id, u.full_name
ORDER BY total_spent DESC;
```

```
SELECT b.booking_id, u.full_name, b.start_date, b.end_date, b.total_cost
FROM Bookings b
INNER JOIN Users u ON b.user_id = u.user_id
WHERE b.total_cost > (SELECT AVG(total_cost) FROM Bookings)
ORDER BY b.total_cost DESC;
```

```
SELECT c.car_id, c.car_model, c.seating_capacity, c.rental_rate, c.car_no, d.full_name AS
driver_name
FROM Cars c
INNER JOIN Drivers d ON c.current_driver_id = d.driver_id
WHERE c.status = 'available'
ORDER BY c.rental_rate DESC;
```

```
SELECT bh.booking_id, u.full_name, bh.start_date, bh.end_date, bh.total_cost
FROM BookingHistory bh
INNER JOIN Users u ON bh.user_id = u.user_id
WHERE bh.total_cost BETWEEN 1000 AND 5000
ORDER BY bh.end_date DESC;
```

```
SELECT c.car_id, c.car_model, c.seating_capacity, c.rental_rate, c.car_no, d.full_name AS
driver_name
FROM Cars c
LEFT JOIN Drivers d ON c.current_driver_id = d.driver_id
ORDER BY c.car_model;
```

```
SELECT u.user_id, u.full_name, SUM(b.total_cost) AS total_booking_cost
FROM Users u
INNER JOIN Bookings b ON u.user_id = b.user_id
GROUP BY u.user_id, u.full_name
HAVING total_booking_cost > 5000
ORDER BY total_booking_cost DESC;
```