Technical University of Denmark

02170 - Mandatory Assignment

Vehicle Rental Mangement Database

Group 07
Adrian Ursu - s240160
Antonijs Bolsakovs - s225124
Andrej Kitanovski - s235246
Niclas Søe Irsbøl - s236136
Viktor Manuel Guijarro - s215205

March 31, 2025

Contents

1 Statement of Requirements

Our database models a **Vehicle Rental Mangement System**, that keeps track of Customers, Vehicles that are being rented, categories, Bookings, Payments, Insurance, Reviews & Maintenance.

The database is designed to manage rentals efficiently by tracking which vehicles are being rented to customers, when they are available, and ensuring easy transactions of payments between customers and the rental service.

The database provides the customer with an easy overview of what vehicles are being rented out, and the customers can search for specific vehicles based on the preferences, such as make, model and price per day.

Customers can rent a vehicle from different vehicle categories, such as economy or premium. Each booking is associated with a customer, a selected vehicle, and a specified rental period. The system also ensures that customers complete their bookings by processing payments.

Each vehicle in the system has detailed records, including its make, model, registration number, rental status, and maintenance history. The vehicles being rented also have to undergo maintenance at a given dates to ensure safety and reliability. Additionally, vehicles are linked to an insurance policy that provides coverage in case of accidents or damages during the rental period.

Customers will also be able to leave a review after they have been using a vehicle, writing valuable feedback for future customers. These reviews can include rating and comments.

2 Conceptual Design

To illustrate the conceptual design of our database, Figure 1 presents an Entity-Relationship diagram. In this diagram, entity sets are represented as blue rectangles containing their respective attributes, while relationship sets are shown as diamonds.

In the Statement of Requirements we have colored the entities, relations and attributes in red, blue and violet, respectively.

The entity set Customer represents individuals who use the vehicle booking service. Each customer has attributes such as first_name, last_name, email, phone, address, and created_at. A customer can make multiple bookings, resulting in a one-to-many relationship between Customer and Booking. Similarly, a customer can write reviews for vehicles, establishing a one-to-many relationship with the Review entity.

The entity set Booking records each vehicle reservation made by a customer. It includes the following attributes: booking_id, customer_id, vehicle_id, start_date, end_date, status, created_at. Each booking is associated with one customer and one vehicle, forming a many-to-one relationship with both. In addition, each booking has a single associated payment, leading to a one-to-one relationship between Booking and Payment.

The entity set Payment contains information about completed transactions. Its attributes include: payment_id, booking_id, amount, payment_method, status, payment_date. Since each booking can only have one payment, the relationship between Booking and Payment is one-to-one, with total participation on the payment side.

The entity set Vehicle represents the cars available in the system and the information about them. Each vehicle has the next properties: vehicle_id, make, model, year, registration_number, price, status, and a foreign key category_id linking it to its category. Vehicles can be reserved in multiple bookings, hence forming a one-to-many relationship with Booking. In addition, vehicles may undergo maintenance, resulting in a one-to-many relationship with the Maintenance entity. They are also covered by an insurance policy, forming a one-to-one relationship with the Insurance entity, assuming one vehicle has one current policy at a time.

The entity set Review captures feedback left by customers regarding vehicles. Each review is linked to both a customer_id and a vehicle_id, with attributes such as rating, comment, review_date. The relationship is many-to-one with both Customer and Vehicle, indicating that a customer can write multiple reviews and a vehicle can receive multiple reviews.

The entity set Maintenance records service events for vehicles, containing details such as: service_date, details, cost, status. Each maintenance entry is linked to a single vehicle, forming a one-to-many relationship.

The entity set Insurance tracks insurance policies for vehicles. Each policy has the following attributes: insurance_id, vehicle_id, policy_number, provider, coverage_details, start_date and end_date. The assumption here is a one-to-one relationship between Vehicle and Insurance, with each vehicle having on active policy at a given time.

Lastly, the entity set Category classifies vehicles by type, such as "Economy" or "Premium", using the attributes type and description. Each vehicle belongs to one category, establishing a many-to-one relationship from Vehicle to Category.

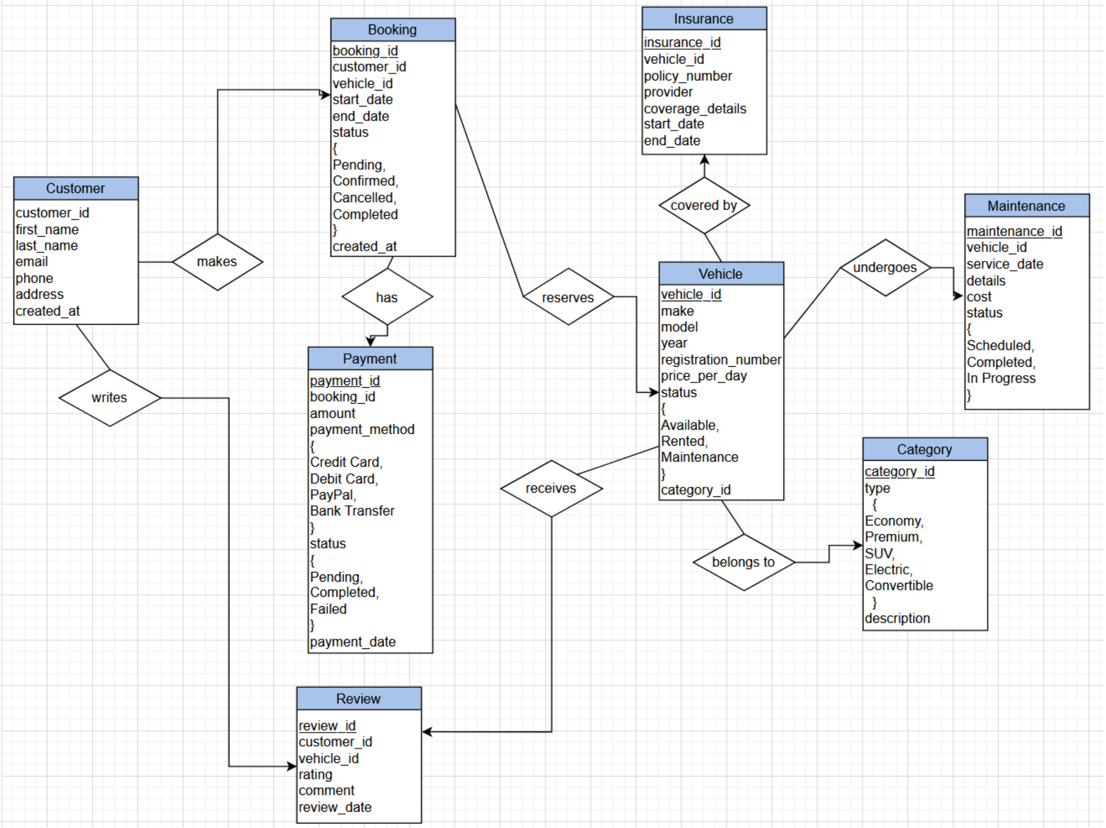


Figure 1: Entity-Relationship Diagram

3 Logical Design

Based on conceptual design above, we have converted the diagram into a logical design, which can be seen below.

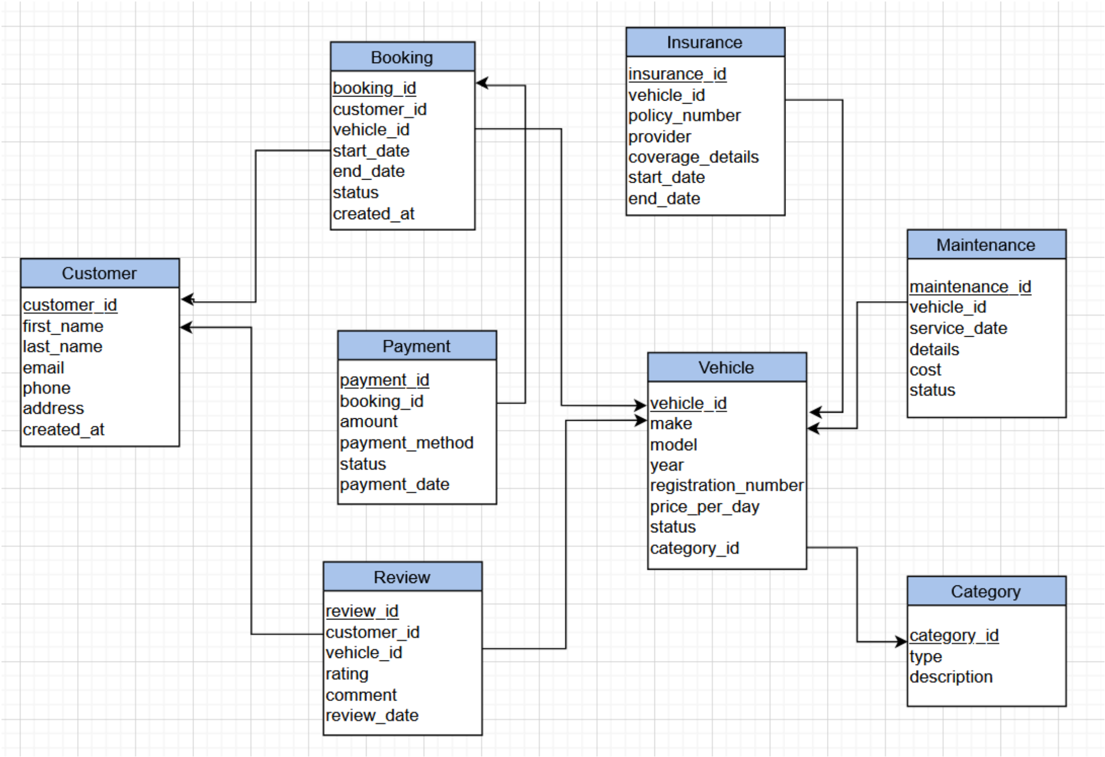


Figure 2: Relation Schema Diagram

The design of our database schema was structured to ensure data integrity, efficiency, and scalability. To ensure referential integrity, we use *ON DELETE CASCADE* where necessary, particularly for dependent records like *Payment* and *Booking*. Additionally, weak entities such as *Insurance* and *Maintenance* were designed with foreign keys, ensuring they cannot exist without their parent *Vehicle*.

4 Implementation

Making the tables proved to be quite easy since we already had made the diagrams for the conceptual design, and the logical design, which gave us a good overview of what tables the database should include and the relationships between them.

In each table we make use of constraints that will ensure that some values remain unique to avoid duplications. In combination with this, we use the constraint *AUTO_INCREMENT* to keep all the *id's* such as *customer_id*, category_id, vehicle_id and so on unique. This also keeps a level of consistency in the database. To make sure that an attribute always has a value we make use of the *NOT NULL constraint* and *DEFAULT*, in this way an attribute will always be associated with a value, this can for example be seen in the table below:

```
CREATE TABLE customer (
    customer_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(20) UNIQUE NOT NULL,
    address TEXT NOT NULL,
    created_at DATE NOT NULL
);
```

We also make use of *Enum*'s for attribute with predefined possible values, such as the **Category** table as seen below:

```
CREATE TABLE category (
    category_id INT AUTO_INCREMENT PRIMARY KEY,
    name ENUM('Economy', 'Premium', 'SUV', 'Electric', 'Convertible') NOT NULL,
    description TEXT
);
```

To maintain relationships between tables, we make use of *FOREIGN KEYS*, where attributes in different tables refers to each other.

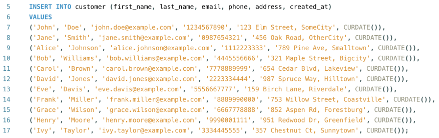
Additionally, to ensure consistency and maintain referential integrity across table, we make use of the foreign key constraints *ON DELETE SET NULL* and *ON DELETE CASCADE*, this ensure that when a referenced record in a table is deleted, all associated records are also automatically deleted. Take for example the *Customer* table, if we deleted a referenced record in that table, all associated records in the *Review* table are also being deleted automatically.

Another example is in our *Vehicle* table, if an instance of a Vehicle is being deleted, the *category_id* of that vehicle is being set to null, since that given vehicle doesn't exists anymore.

5 Database Instance

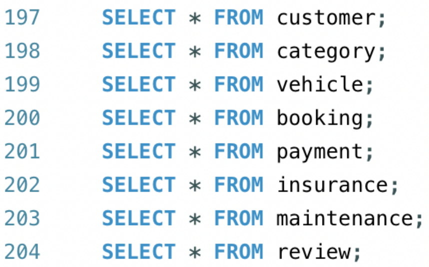
In this part we create the database instances. We set the starting date of all rentals to the current date in which we create the tables.

You will underneath an example of an **"INSERT"** for customer and the **"SELECT * FROM"**.



```
INSERT INTO customer (first_name, last_name, email, phone, address, created_at)
VALUES
('John', 'Doe', 'john.doe@example.com', '1234567890', '123 Elm Street, SomeCity', CURDATE()),
('Jane', 'Smith', 'jane.smith@example.com', '0987654321', '456 Oak Road, OtherCity', CURDATE()),
('Alice', 'Johnson', 'alice.johnson@example.com', '1112223333', '789 Pine Ave, Smalltown', CURDATE()),
('Bob', 'Williams', 'bob.williams@example.com', '4445556666', '321 Maple Street, Bigcity', CURDATE()),
('Carol', 'Brown', 'carol.brown@example.com', '7778889999', '654 Cedar Blvd, Lakeview', CURDATE()),
('David', 'Jones', 'david.jones@example.com', '2223334444', '987 Spruce Way, Hilltown', CURDATE()),
('Eve', 'Davis', 'eve.davis@example.com', '5556667777', '159 Birch Lane, Riverdale', CURDATE()),
('Frank', 'Miller', 'frank.miller@example.com', '8889990000', '753 Willow Street, Coastville', CURDATE()),
('Grace', 'Wilson', 'grace.wilson@example.com', '6667778888', '852 Aspen Rd, Forestburg', CURDATE()),
('Henry', 'Moore', 'henry.moore@example.com', '9990001111', '951 Redwood Dr, Greenfield', CURDATE()),
('Ivy', 'Taylor', 'ivy.taylor@example.com', '3334445555', '357 Chestnut Ct, Sunnytown', CURDATE());
```

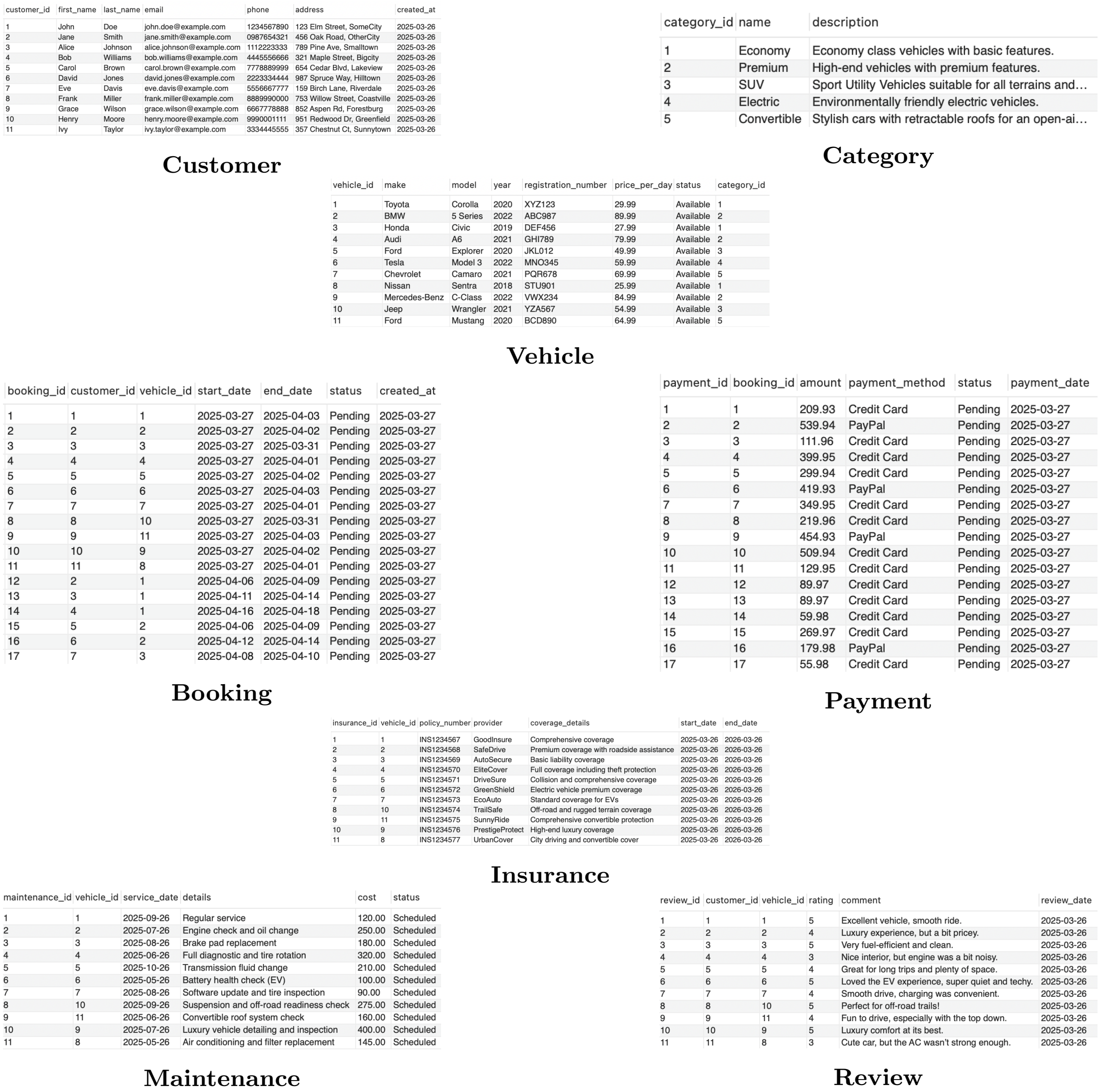
INSERT



```
SELECT * FROM customer;
SELECT * FROM category;
SELECT * FROM vehicle;
SELECT * FROM booking;
SELECT * FROM payment;
SELECT * FROM insurance;
SELECT * FROM maintenance;
SELECT * FROM review;
```

SELECT * FROM

Figure 3



customer_id	first_name	last_name	email	phone	address	created_at
1	John	Doe	john.doe@example.com	1234567890	123 Elm Street, SomeCity	2025-03-26
2	Jane	Smith	jane.smith@example.com	0987654321	456 Oak Road, OtherCity	2025-03-26
3	Alice	Johnson	alice.johnson@example.com	1112223333	789 Pine Ave, Smalltown	2025-03-26
4	Bob	Williams	bob.williams@example.com	4445556666	321 Maple Street, Bigcity	2025-03-26
5	Carol	Brown	carol.brown@example.com	7778889999	654 Cedar Blvd, Lakeview	2025-03-26
6	David	Jones	david.jones@example.com	2223334444	987 Spruce Way, Hilltown	2025-03-26
7	Eve	Davis	eve.davis@example.com	5556667777	159 Birch Lane, Riverdale	2025-03-26
8	Frank	Miller	frank.miller@example.com	8889990000	753 Willow Street, Coastville	2025-03-26
9	Grace	Wilson	grace.wilson@example.com	6667778888	852 Aspen Rd, Forestburg	2025-03-26
10	Henry	Moore	henry.moore@example.com	9990001111	951 Redwood Dr, Greenfield	2025-03-26
11	Ivy	Taylor	ivy.taylor@example.com	3334445555	357 Chestnut Ct, Sunnytown	2025-03-26

Customer

category_id	name	description
1	Economy	Economy class vehicles with basic features.
2	Premium	High-end vehicles with premium features.
3	SUV	Sport Utility Vehicles suitable for all terrains and…
4	Electric	Environmentally friendly electric vehicles.
5	Convertible	Stylish cars with retractable roofs for an open-ai…

Category

vehicle_id	make	model	year	registration_number	price_per_day	status	category_id
1	Toyota	Corolla	2020	XYZ123	29.99	Available	1
2	BMW	5 Series	2022	ABC987	89.99	Available	2
3	Honda	Civic	2019	DEF456	27.99	Available	1
4	Audi	A6	2021	GHI789	79.99	Available	2
5	Ford	Explorer	2020	JKL012	49.99	Available	3
6	Tesla	Model 3	2022	MNO345	59.99	Available	4
7	Chevrolet	Camaro	2021	PQR678	69.99	Available	5
8	Nissan	Sentra	2018	STU901	25.99	Available	1
9	Mercedes-Benz	C-Class	2022	VWX234	84.99	Available	2
10	Jeep	Wrangler	2021	YZA567	54.99	Available	3
11	Ford	Mustang	2020	BCD890	64.99	Available	5

Vehicle

booking_id	customer_id	vehicle_id	start_date	end_date	status	created_at
1	1	1	2025-03-27	2025-04-03	Pending	2025-03-27
2	2	2	2025-03-27	2025-04-02	Pending	2025-03-27
3	3	3	2025-03-27	2025-03-31	Pending	2025-03-27
4	4	4	2025-03-27	2025-04-01	Pending	2025-03-27
5	5	5	2025-03-27	2025-04-02	Pending	2025-03-27
6	6	6	2025-03-27	2025-04-03	Pending	2025-03-27
7	7	7	2025-03-27	2025-04-01	Pending	2025-03-27
8	8	10	2025-03-27	2025-03-31	Pending	2025-03-27
9	9	11	2025-03-27	2025-04-03	Pending	2025-03-27
10	10	9	2025-03-27	2025-04-02	Pending	2025-03-27
11	11	8	2025-03-27	2025-04-01	Pending	2025-03-27
12	2	1	2025-04-06	2025-04-09	Pending	2025-03-27
13	3	1	2025-04-11	2025-04-14	Pending	2025-03-27
14	4	1	2025-04-16	2025-04-18	Pending	2025-03-27
15	5	2	2025-04-06	2025-04-09	Pending	2025-03-27
16	6	2	2025-04-12	2025-04-14	Pending	2025-03-27
17	7	3	2025-04-08	2025-04-10	Pending	2025-03-27

Booking

payment_id	booking_id	amount	payment_method	status	payment_date
1	1	209.93	Credit Card	Pending	2025-03-27
2	2	539.94	PayPal	Pending	2025-03-27
3	3	111.96	Credit Card	Pending	2025-03-27
4	4	399.95	Credit Card	Pending	2025-03-27
5	5	299.94	Credit Card	Pending	2025-03-27
6	6	419.93	PayPal	Pending	2025-03-27
7	7	349.95	Credit Card	Pending	2025-03-27
8	8	219.96	Credit Card	Pending	2025-03-27
9	9	454.93	PayPal	Pending	2025-03-27
10	10	509.94	Credit Card	Pending	2025-03-27
11	11	129.95	Credit Card	Pending	2025-03-27
12	12	89.97	Credit Card	Pending	2025-03-27
13	13	89.97	Credit Card	Pending	2025-03-27
14	14	59.98	Credit Card	Pending	2025-03-27
15	15	269.97	Credit Card	Pending	2025-03-27
16	16	179.98	PayPal	Pending	2025-03-27
17	17	55.98	Credit Card	Pending	2025-03-27

Payment

insurance_id	vehicle_id	policy_number	provider	coverage_details	start_date	end_date
1	1	INS1234567	GoodInsure	Comprehensive coverage	2025-03-26	2026-03-26
2	2	INS1234568	SafeDrive	Premium coverage with roadside assistance	2025-03-26	2026-03-26
3	3	INS1234569	AutoSecure	Basic liability coverage	2025-03-26	2026-03-26
4	4	INS1234570	EliteCover	Full coverage including theft protection	2025-03-26	2026-03-26
5	5	INS1234571	DriveSure	Collision and comprehensive coverage	2025-03-26	2026-03-26
6	6	INS1234572	GreenShield	Electric vehicle premium coverage	2025-03-26	2026-03-26
7	7	INS1234573	EcoAuto	Standard coverage for EVs	2025-03-26	2026-03-26
8	10	INS1234574	TrailSafe	Off-road and rugged terrain coverage	2025-03-26	2026-03-26
9	11	INS1234575	SunnyRide	Comprehensive convertible protection	2025-03-26	2026-03-26
10	9	INS1234576	PrestigeProtect	High-end luxury coverage	2025-03-26	2026-03-26
11	8	INS1234577	UrbanCover	City driving and convertible cover	2025-03-26	2026-03-26

Insurance

maintenance_id	vehicle_id	service_date	details	cost	status
1	1	2025-09-26	Regular service	120.00	Scheduled
2	2	2025-07-26	Engine check and oil change	250.00	Scheduled
3	3	2025-08-26	Brake pad replacement	180.00	Scheduled
4	4	2025-06-26	Full diagnostic and tire rotation	320.00	Scheduled
5	5	2025-10-26	Transmission fluid change	210.00	Scheduled
6	6	2025-05-26	Battery health check (EV)	100.00	Scheduled
7	7	2025-08-26	Software update and tire inspection	90.00	Scheduled
8	10	2025-09-26	Suspension and off-road readiness check	275.00	Scheduled
9	11	2025-06-26	Convertible roof system check	160.00	Scheduled
10	9	2025-07-26	Luxury vehicle detailing and inspection	400.00	Scheduled
11	8	2025-05-26	Air conditioning and filter replacement	145.00	Scheduled

Maintenance

review_id	customer_id	vehicle_id	rating	comment	review_date
1	1	1	5	Excellent vehicle, smooth ride.	2025-03-26
2	2	2	4	Luxury experience, but a bit pricey.	2025-03-26
3	3	3	5	Very fuel-efficient and clean.	2025-03-26
4	4	4	3	Nice interior, but engine was a bit noisy.	2025-03-26
5	5	5	4	Great for long trips and plenty of space.	2025-03-26
6	6	6	5	Loved the EV experience, super quiet and techy.	2025-03-26
7	7	7	4	Smooth drive, charging was convenient.	2025-03-26
8	8	10	5	Perfect for off-road trails!	2025-03-26
9	9	11	4	Fun to drive, especially with the top down.	2025-03-26
10	10	9	5	Luxury comfort at its best.	2025-03-26
11	11	8	3	Cute car, but the AC wasn't strong enough.	2025-03-26

Review

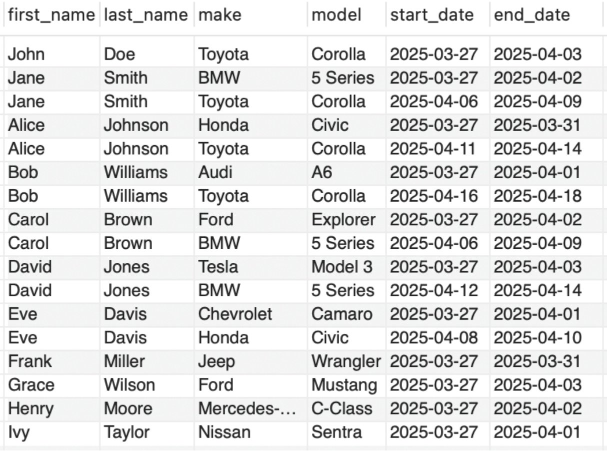
Figure 4: All data instances

6 SQL Data Queries

In this section, we created and tested three SQL queries on our VehicleRentalDB database using JOIN, GROUP BY, and a set operation (IN). Each query was designed to demonstrate core SQL functionalities in the context of our vehicle rental system.

Query 1: JOIN

```
SELECT
  c.first_name,
  c.last_name,
  v.make,
  v.model,
  b.start_date,
  b.end_date
FROM booking b
JOIN customer c ON b.customer_id = c.customer_id
JOIN vehicle v ON b.vehicle_id = v.vehicle_id;
```

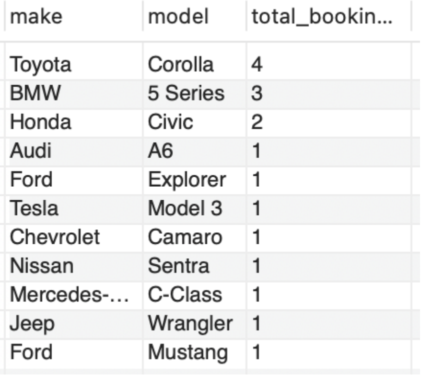


first_name	last_name	make	model	start_date	end_date
John	Doe	Toyota	Corolla	2025-03-27	2025-04-03
Jane	Smith	BMW	5 Series	2025-03-27	2025-04-02
Jane	Smith	Toyota	Corolla	2025-04-06	2025-04-09
Alice	Johnson	Honda	Civic	2025-03-27	2025-03-31
Alice	Johnson	Toyota	Corolla	2025-04-11	2025-04-14
Bob	Williams	Audi	A6	2025-03-27	2025-04-01
Bob	Williams	Toyota	Corolla	2025-04-16	2025-04-18
Carol	Brown	Ford	Explorer	2025-03-27	2025-04-02
Carol	Brown	BMW	5 Series	2025-04-06	2025-04-09
David	Jones	Tesla	Model 3	2025-03-27	2025-04-03
David	Jones	BMW	5 Series	2025-04-12	2025-04-14
Eve	Davis	Chevrolet	Camaro	2025-03-27	2025-04-01
Eve	Davis	Honda	Civic	2025-04-08	2025-04-10
Frank	Miller	Jeep	Wrangler	2025-03-27	2025-03-31
Grace	Wilson	Ford	Mustang	2025-03-27	2025-04-03
Henry	Moore	Mercedes-...	C-Class	2025-03-27	2025-04-02
Ivy	Taylor	Nissan	Sentra	2025-03-27	2025-04-01

Explanation: This query joins the `booking`, `customer`, and `vehicle` tables to show which customer booked which vehicle and for what dates.

Query 2: GROUP BY

```
SELECT
  v.make,
  v.model,
  COUNT(b.booking_id) AS total_bookings
FROM vehicle v
JOIN booking b ON v.vehicle_id = b.vehicle_id
GROUP BY v.vehicle_id;
```

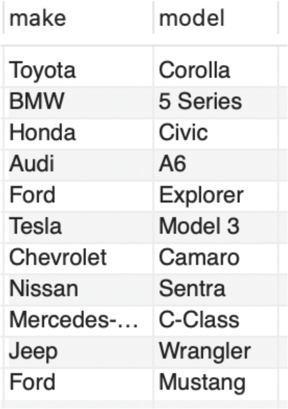


make	model	total_bookin...
Toyota	Corolla	4
BMW	5 Series	3
Honda	Civic	2
Audi	A6	1
Ford	Explorer	1
Tesla	Model 3	1
Chevrolet	Camaro	1
Nissan	Sentra	1
Mercedes-...	C-Class	1
Jeep	Wrangler	1
Ford	Mustang	1

Explanation: This query groups all bookings by vehicle and counts how many times each vehicle was booked.

Query 3: IN (Set Operation)

```
SELECT make, model
FROM vehicle
WHERE vehicle_id IN (
    SELECT vehicle_id FROM review
);
```



make	model
Toyota	Corolla
BMW	5 Series
Honda	Civic
Audi	A6
Ford	Explorer
Tesla	Model 3
Chevrolet	Camaro
Nissan	Sentra
Mercedes-...	C-Class
Jeep	Wrangler
Ford	Mustang

Explanation: This query finds all vehicles that have been reviewed by customers by selecting only those whose IDs appear in the `review` table.

Dummy Data Setup:

To test the queries, we inserted realistic dummy data into the database, including:

- 11 customers
- 11 vehicles from different categories
- 17 bookings across several customers and vehicles
- 17 payments linked to existing bookings
- 11 reviews added by different customers

The queries were run and tested using MySQL Workbench.

7 SQL Programming

In this section, we demonstrate examples of two SQL functions, one stored procedure, and one trigger implemented within our `VehicleRentalDB`. These SQL components encapsulate logic for reuse, automate actions, and enforce data integrity.

7.1 Function

7.1.1 Function 1 - `calculateTotalRentalCost`

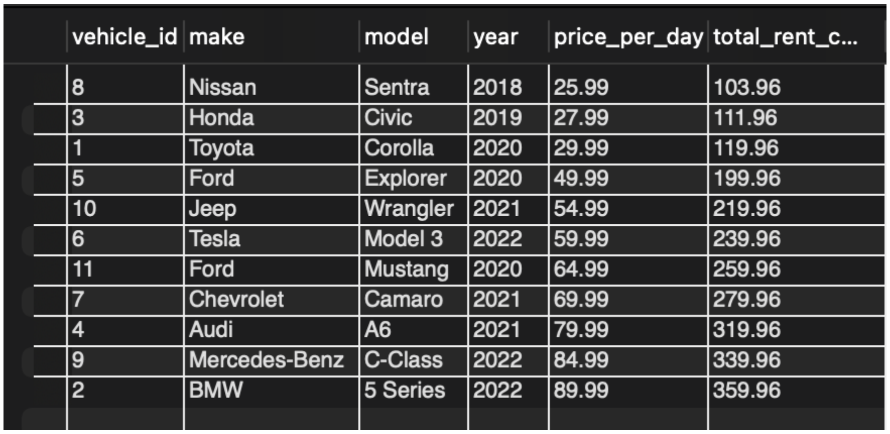
The function `calculateTotalRentalCost` takes a `vehicle_id`, a `start_date`, and an `end_date`, and returns the total cost of the rental by multiplying the number of days by the vehicle's daily rental price.

```
DELIMITER //
CREATE FUNCTION calculateTotalRentalCost(
    p_vehicle_id INT,
    p_start_date DATE,
    p_end_date DATE
)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE rental_days INT;
    DECLARE daily_rate DECIMAL(10,2);
    DECLARE total_cost DECIMAL(10,2);

    SET rental_days = DATEDIFF(p_end_date, p_start_date);

    SELECT price_per_day INTO daily_rate
    FROM vehicle
    WHERE vehicle_id = p_vehicle_id;

    SET total_cost = rental_days * daily_rate;

    RETURN total_cost;
END //

DELIMITER ;
```

Usage Example:

```
SELECT
    vehicle_id,
    make,
    model,
    year,
    price_per_day,
    calculateTotalRentalCost(vehicle_id, '2025-04-01', '2025-04-05') AS total_rent_cost
FROM vehicle
ORDER BY price_per_day;
```

This query applies the function to all vehicles in the database, calculating how much each one would cost to rent for a 4-day period. The result is useful for comparing rental prices across different vehicle models and types.



vehicle_id	make	model	year	price_per_day	total_rent_c...
8	Nissan	Sentra	2018	25.99	103.96
3	Honda	Civic	2019	27.99	111.96
1	Toyota	Corolla	2020	29.99	119.96
5	Ford	Explorer	2020	49.99	199.96
10	Jeep	Wrangler	2021	54.99	219.96
6	Tesla	Model 3	2022	59.99	239.96
11	Ford	Mustang	2020	64.99	259.96
7	Chevrolet	Camaro	2021	69.99	279.96
4	Audi	A6	2021	79.99	319.96
9	Mercedes-Benz	C-Class	2022	84.99	339.96
2	BMW	5 Series	2022	89.99	359.96

Figure 5: Total Rental Cost Calculation using Function

7.1.2 Function 2 - `getBookingDuration`

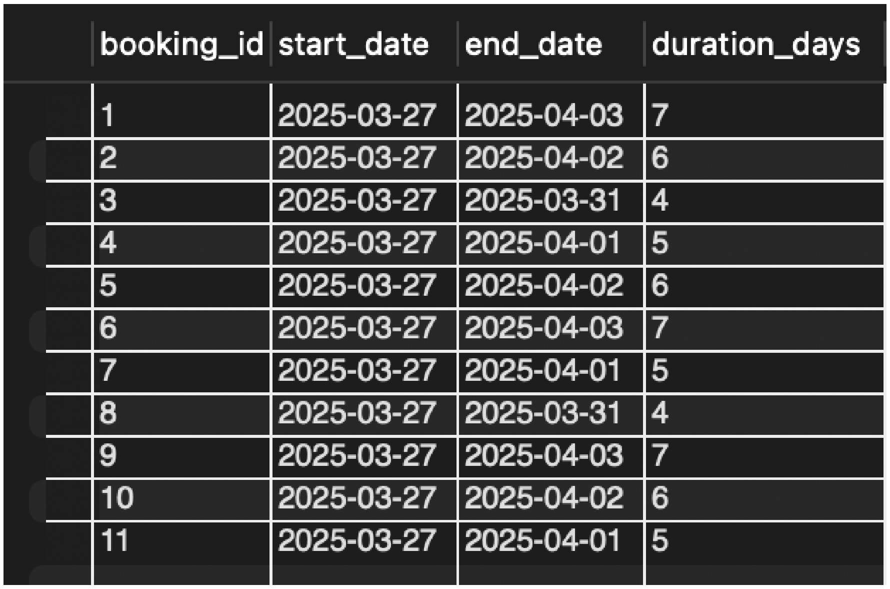
This function returns the duration of a booking in days using the start and end dates.

```
DELIMITER //
CREATE FUNCTION getBookingDuration(startDate DATE, endDate DATE)
RETURNS INT
DETERMINISTIC
BEGIN
    RETURN DATEDIFF(endDate, startDate);
END //

DELIMITER ;
```

Usage Example:

```
SELECT
  booking_id,
  start_date,
  end_date,
  getBookingDuration(start_date, end_date) AS duration_days
FROM booking;
```

This query shows a list of bookings and uses the function to calculate how many days each booking spans. It's especially useful for customer billing, internal analytics, or usage reports. By embedding this logic in a function, the calculation is reusable and consistent across queries.



booking_id	start_date	end_date	duration_days
1	2025-03-27	2025-04-03	7
2	2025-03-27	2025-04-02	6
3	2025-03-27	2025-03-31	4
4	2025-03-27	2025-04-01	5
5	2025-03-27	2025-04-02	6
6	2025-03-27	2025-04-03	7
7	2025-03-27	2025-04-01	5
8	2025-03-27	2025-03-31	4
9	2025-03-27	2025-04-03	7
10	2025-03-27	2025-04-02	6
11	2025-03-27	2025-04-01	5

Figure 6: Booking Duration Using Function

7.2 Procedure

createBookingWithPayment is a stored procedure that inserts a new booking and immediately calculates and records the related payment.

```
Delimiter //

CREATE PROCEDURE createBookingWithPayment(
    IN cust_id INT,
    IN veh_id INT,
    IN start_date DATE,
    IN end_date DATE,
    IN payment_method VARCHAR(20)
)
BEGIN
    DECLARE booking_days INT;
    DECLARE price DECIMAL(10,2);
    DECLARE booking_id INT;

    SET booking_days = DATEDIFF(end_date, start_date);
    SELECT price_per_day INTO price FROM vehicle WHERE vehicle_id = veh_id;

    INSERT INTO booking (customer_id, vehicle_id, start_date, end_date, created_at)
    VALUES (cust_id, veh_id, start_date, end_date, CURDATE());

    SET booking_id = LAST_INSERT_ID();

    INSERT INTO payment (booking_id, amount, payment_method, status, payment_date)
    VALUES (booking_id, price * booking_days, payment_method, 'Completed', CURDATE());
END //
Delimiter ;
```

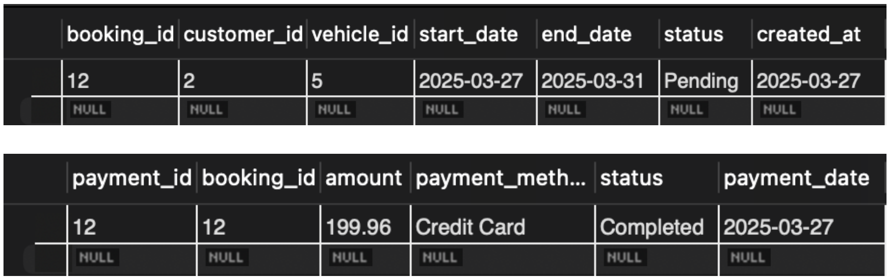
Usage Example:

```
CALL createBookingWithPayment(2, 5, CURDATE(), DATE_ADD(CURDATE(), INTERVAL 4 DAY), 'Credit Card')
```

Querying the created data:

```
SELECT * FROM booking WHERE customer_id = 2 ORDER BY booking_id DESC LIMIT 1;
SELECT * FROM payment ORDER BY payment_id DESC LIMIT 1;
```

Explanation: The procedure creates a new booking for customer 2 and vehicle 5, then calculates and inserts the associated payment. The follow-up queries confirm that both the booking and the payment were successfully recorded.



booking_id	customer_id	vehicle_id	start_date	end_date	status	created_at
12	2	5	2025-03-27	2025-03-31	Pending	2025-03-27
NULL	NULL	NULL	NULL	NULL	NULL	NULL

payment_id	booking_id	amount	payment_meth...	status	payment_date
12	12	199.96	Credit Card	Completed	2025-03-27
NULL	NULL	NULL	NULL	NULL	NULL

Figure 7: Booking and Payment Inserted via Procedure

7.3 Trigger

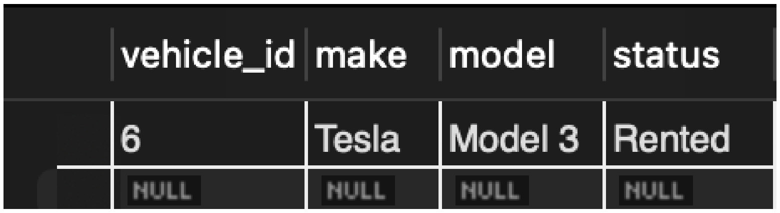
The trigger `updateVehicleStatusAfterBooking` automatically updates the vehicle's status to `'Rented'` once a booking has been created.

```
Delimiter //

CREATE TRIGGER updateVehicleStatusAfterBooking
AFTER INSERT ON booking
FOR EACH ROW
BEGIN
    UPDATE vehicle
    SET status = 'Rented'
    WHERE vehicle_id = NEW.vehicle_id;
END //
Delimiter ;
```

Usage Example:

```
INSERT INTO booking (customer_id, vehicle_id, start_date, end_date, created_at)
VALUES (3, 6, CURDATE(), DATE_ADD(CURDATE(), INTERVAL 3 DAY), CURDATE());

SELECT vehicle_id, make, model, status
FROM vehicle
WHERE vehicle_id = 6;
```

After manually inserting a booking for vehicle 6, the follow-up query confirms that its status has been updated to 'Rented', showing the trigger works as intended to maintain availability status automatically.



vehicle_id	make	model	status
6	Tesla	Model 3	Rented
NULL	NULL	NULL	NULL

Figure 8: Vehicle Status Updated Automatically via Trigger

8 SQL Table Modifications

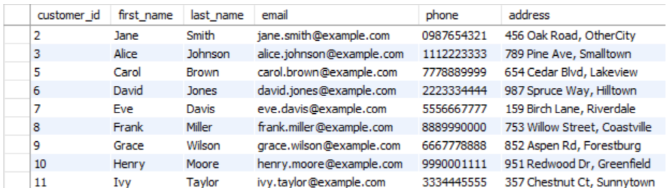
In this section, we will show we will show the execution of a *DELETE* statement and a *UPDATE* statement.

8.1 DELETE statement

The SQL script below deletes the users 'John' and 'Bob' from the Customers table.

```
DELETE FROM customer
WHERE first_name IN ('John', 'Bob');
```

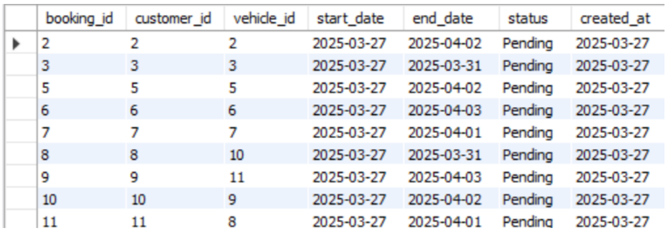
And as a result of these we see that if we query all listed Customers in the Customers table, the users 'John' and 'Bob' will be gone along with all of their attributes.



customer_id	first_name	last_name	email	phone	address
2	Jane	Smith	jane.smith@example.com	0987654321	456 Oak Road, OtherCity
3	Alice	Johnson	alice.johnson@example.com	1112223333	789 Pine Ave, Smalltown
5	Carol	Brown	carol.brown@example.com	7778889999	654 Cedar Blvd, Lakeview
6	David	Jones	david.jones@example.com	2223334444	987 Spruce Way, Hilltown
7	Eve	Davis	eve.davis@example.com	5556667777	159 Birch Lane, Riverdale
8	Frank	Miller	frank.miller@example.com	8889990000	753 Willow Street, Coastville
9	Grace	Wilson	grace.wilson@example.com	6667778888	852 Aspen Rd, Forestburg
10	Henry	Moore	henry.moore@example.com	9990001111	951 Redwood Dr, Greenfield
11	Ivy	Taylor	ivy.taylor@example.com	3334445555	357 Chestnut Ct, Sunnytown

Figure 9: Customer deletion

This will also result in all referenced records that are related to 'John' and 'Bob' through foreign key constraints being set to null or deleted, which is ensured by the referential integrity in our design. As an example if we query for all data inserted in the *Booking* table, we will see that the related record associated with customers 'John' and 'Bob', which have customer_id's 1 and 3, respectively, will be deleted.



booking_id	customer_id	vehicle_id	start_date	end_date	status	created_at
2	2	2	2025-03-27	2025-04-02	Pending	2025-03-27
3	3	3	2025-03-27	2025-03-31	Pending	2025-03-27
5	5	5	2025-03-27	2025-04-02	Pending	2025-03-27
6	6	6	2025-03-27	2025-04-03	Pending	2025-03-27
7	7	7	2025-03-27	2025-04-01	Pending	2025-03-27
8	8	10	2025-03-27	2025-03-31	Pending	2025-03-27
9	9	11	2025-03-27	2025-04-03	Pending	2025-03-27
10	10	9	2025-03-27	2025-04-02	Pending	2025-03-27
11	11	8	2025-03-27	2025-04-01	Pending	2025-03-27

Figure 10: Customer deletion - Referential integrity

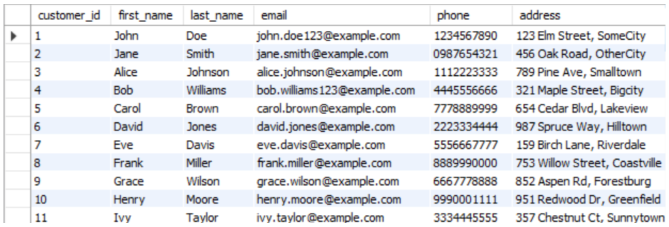
Since we have the referential integrity constraint ON DELETE CASCADE referring to the *customer_id* attribute, deleting a row from the customer table will automatically delete all related rows in the referencing table. This ensure that the data in consistent across the database.

8.2 UPDATE statement

In a case where the users 'John' and 'Bob' want to have their email-addresses updated, we can achieve this using the UPDATE statement below.

```
UPDATE customer
SET email =
    CASE
        WHEN first_name = 'John' THEN 'john.doe123@example.com'
        WHEN first_name = 'Bob' THEN 'bob.williams123@example.com'
    END
WHERE first_name IN ('John', 'Bob');
```

Now, when we query for all instances of customer, we see that the email addresses of these two customers has been updated.



customer_id	first_name	last_name	email	phone	address
1	John	Doe	john.doe123@example.com	1234567890	123 Elm Street, SomeCity
2	Jane	Smith	jane.smith@example.com	0987654321	456 Oak Road, OtherCity
3	Alice	Johnson	alice.johnson@example.com	1112223333	789 Pine Ave, Smalltown
4	Bob	Williams	bob.williams123@example.com	4445556666	321 Maple Street, Bigcity
5	Carol	Brown	carol.brown@example.com	7778889999	654 Cedar Blvd, Lakeview
6	David	Jones	david.jones@example.com	2223334444	987 Spruce Way, Hilltown
7	Eve	Davis	eve.davis@example.com	5556667777	159 Birch Lane, Riverdale
8	Frank	Miller	frank.miller@example.com	8889990000	753 Willow Street, Coastville
9	Grace	Wilson	grace.wilson@example.com	6667778888	852 Aspen Rd, Forestburg
10	Henry	Moore	henry.moore@example.com	9990001111	951 Redwood Dr, Greenfield
11	Ivy	Taylor	ivy.taylor@example.com	3334445555	357 Chestnut Ct, Sunnytown

Figure 11: Customer update

As we see above, the email addresses of the customers 'John' and 'Bob' have been updated.