# RENTAL CAR MANAGEMENT SYSTEM

MINOR PROJECT REPORT

By

**RAAHIL HASSAN (RA2311047010010)**

**JESWIN M (RA2311047010043)**

*Under the Guidance of*

**DR.M.FERNI UKRIT**

,Professor,Computational Intelligence

*in partial fulfillment of the requirements for the degree of*

**BACHELOR OF TECHNOLOGY**

*in*

**ARTIFICIAL INTELLIGENCE**

**DEPARTMENT OF COMPUTATIONAL INTELLIGENCE COLLEGE OF ENGINEERING AND TECHNOLOGY**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR– 603 203**

**SRM INSTITUTE OF SCIENNCE AND TECHNOLOGY**

**KATTANKULTHAR-603 203**

**BONAFIDE CERTIFICATE**

This is to certify that the project work titled *'RENTAL CAR MANAGEMENT SYSTEM '*has been completed out by **RAAHIL HASSAN (RA2311047010010) ,JESWIN M (RA2311047010043),** and students of the **3rd Year** / **5th Semester B.Tech ,**as part of the course *21CSC205P - Project course* during the academic year **2025-2026** at **SRM Institute of science and Technology,** Kattankulathur.The work presented is their original contribution and has been duly completed under the supervision of the course instructor.

**Date:**

| **Faculty in charge** | **HEAD OF THE DEPARTMENT** |
|---|---|
| Dr.M.Ferni Ukrit | Dr.R.Annie Uthra |
| Professor | Professor and Head |
| Department of Computational Intelligence | Department of computational Intelligence |
| SRM-KTR | SRM-KTR |

## ABSTRACT

The **Car Rental Management System** is designed to automate and streamline the operations of vehicle rental businesses. It simplifies processes such as car reservations, fleet management, payment processing, and customer handling by integrating them into a single centralized platform. The system eliminates manual paperwork, minimizes human error, and enhances the efficiency of both employees and customers.

Developed using **HTML, CSS, MySQL, and Flask**, the system provides a user-friendly interface for managing customers, vehicles, reservations, rentals, and payments. Administrators can efficiently monitor vehicle availability, track rentals, and generate reports, while customers can easily browse cars, make bookings, and complete transactions online.

The database component ensures data consistency, integrity, and security through normalization and the use of SQL concepts such as **joins, views, triggers, and cursors**. Additionally, data control (DCL) and transaction control (TCL) commands are implemented to ensure secure and reliable operations.

By integrating automation and database technologies, this system enhances customer satisfaction, reduces operational costs, and provides real-time insights for better decision-making. It represents a robust and scalable solution for managing modern car rental businesses.

**Keywords:** Car Rental, Database Management, Reservation System, SQL, Fleet Management, Payment Processing, Flask, MySQL.

# TABLE OF CONTENTS

# Problem Statement:(CHAPTER-1)

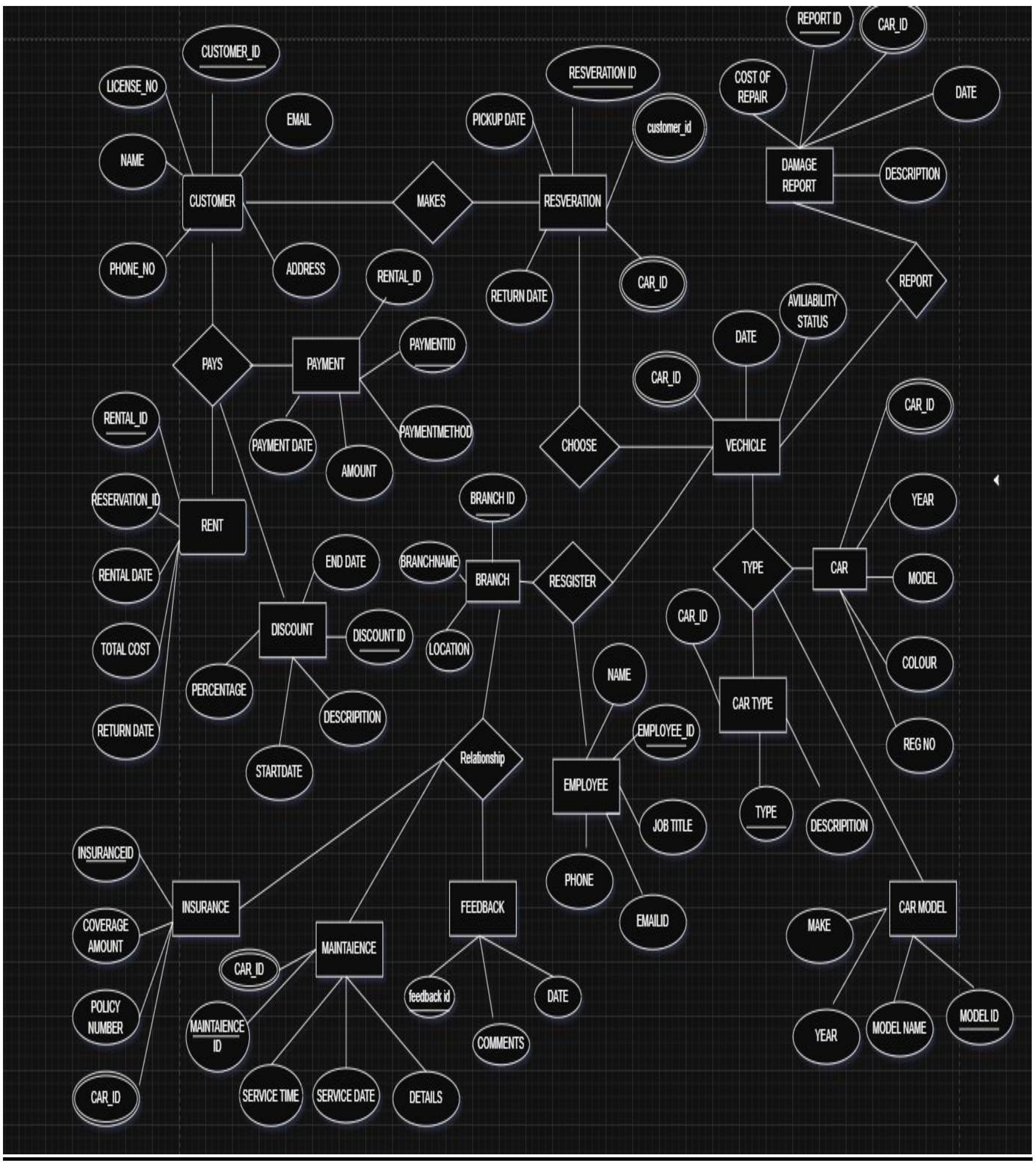
The existing car rental process in many organizations is largely manual and inefficient, involving handwritten records, phone-based bookings, and disorganized tracking of vehicles, customers, and payments. This traditional method leads to **booking errors, data redundancy, poor tracking, and lack of real-time availability**.

The **Car Rental Management System** is developed to overcome these issues by providing a **centralized, automated database system** that integrates all business operations such as **vehicle management, customer registration, reservation tracking, rental processing, and payment handling**

## Problem Description:

1. **Manual Record Handling:**
   Traditional car rental agencies maintain information manually, which increases the risk of data inconsistency and loss.
2. **Limited Real-Time Availability:**
   Vehicle availability status is not updated dynamically, leading to overlapping reservations or unutilized vehicles.
3. **Delayed Processing:**
   Manual verification and billing cause unnecessary delays in confirming bookings and generating invoices.
4. **Lack of Integration:**
   Customer details, vehicle data, and payment records are stored separately, making it difficult to retrieve complete information**.**
5. **Error-Prone Transactions:**
   Human errors during entry of rental duration, payment amount, or car return dates often cause financial and operational issues**.**
6. **Poor Security:**
   Sensitive information like driver's license numbers and payment data lack proper security and access control**.**
7. **Limited Reporting:**
   Generating analytical reports for vehicle usage, income, or customer activity requires extensive manual effort.

## ER DIAGRAM:


CUSTOMER_ID
LICENSE_NO
EMAIL
NAME
CUSTOMER
PHONE_NO
ADDRESS
MAKES
RESVERATION ID
PICKUP DATE
customer_id
RESVERATION
RETURN DATE
CAR_ID
REPORT ID
CAR_ID
COST OF REPAIR
DATE
DAMAGE REPORT
DESCRIPTION
REPORT
RENTAL_ID
PAYS
PAYMENT
PAYMENTID
PAYMENT DATE
AMOUNT
PAYMENTMETHOD
CHOOSE
VECHICLE
DATE
AVILIABILITY STATUS
CAR_ID
RENTAL_ID
RESERVATION_ID
RENT
RENTAL DATE
TOTAL COST
RETURN DATE
BRANCH ID
BRANCHNAME
BRANCH
RESGISTER
LOCATION
END DATE
DISCOUNT
DISCOUNT ID
PERCENTAGE
DESCRIPITION
STARTDATE
TYPE
CAR
CAR_ID
YEAR
MODEL
COLOUR
REG NO
CAR_ID
CAR TYPE
TYPE
DESCRIPITION
NAME
EMPLOYEE_ID
EMPLOYEE
JOB TITLE
PHONE
EMAILID
Relationship
INSURANCEID
COVERAGE AMOUNT
POLICY NUMBER
CAR_ID
INSURANCE
MAINTAIENCE
CAR_ID
MAINTAIENCE ID
SERVICE TIME
SERVICE DATE
DETAILS
FEEDBACK
feedback id
COMMENTS
DATE
CAR MODEL
MAKE
YEAR
MODEL NAME
MODEL ID

# <u>ER DIAGRAM FOR RENTAL CAR MANAMENT SYSTEM</u>

An Entity Relationship Diagram (ERD) visually represents the relationships between different entities in a system, illustrating how data is structured and interconnected.

## <u>Entities:</u>

- Customer
- Car
- Reservation
- Rental
- Payment
- Employee
- Branch
- Car Type
- Car Model
- Insurance
- Damage Report
- Maintenance
- Discount
- Feedback

**Relationships Overview:**

In this Car Rental Management System, the relationship between **Customer** and **Reservation** is **One-to-Many**, meaning one customer can make multiple reservations, but each reservation belongs to one specific customer.

The **Car** to **Reservation** relationship is also **One-to-Many**, as one car can be reserved multiple times, but each reservation corresponds to one particular car.

The **Reservation** to **Rental** relationship is **One-to-One**, since each reservation results in one unique rental record once the car is handed over.

The **Rental** to **Payment** relationship follows a **One-to-Many** pattern — one rental can have multiple payments (for example, partial or installment payments), but each payment belongs to one rental transaction.

The **Employee** to **Rental** relationship is **One-to-Many**, as one employee can handle several rental processes, but each rental is managed by only one staff member.

The **Branch** to **Employee** relationship is **One-to-Many**, representing that one branch can have multiple employees, but each employee belongs to a single branch.

The **Car Type** to **Car** relationship is **One-to-Many**, indicating that one type of car (e.g., Sedan, SUV, Hatchback) can include many specific car models.

The **Car** to **Insurance** relationship is **One-to-One**, where each car has exactly one insurance policy linked to it.

The **Car** to **Damage Report** relationship is **One-to-Many**, since a single car may have multiple damage records over time.

The **Car** to **Maintenance** relationship is **One-to-Many**, as one vehicle may undergo multiple maintenance or servicing operations.

The **Discount** to **Reservation** relationship is **One-to-Many**, meaning a single discount offer can apply to multiple reservations.

The **Customer** to **Feedback** relationship is **One-to-Many**, allowing a customer to submit feedback for multiple rentals.

Finally, the **Car** to **Feedback** relationship is also **One-to-Many**, where one car can receive multiple feedback entries from different customers.

---

## Entity Descriptions and Attributes:

### 1. Customer

**Attributes:**

- CustomerID: Unique identifier for each customer.
- Name: Full name of the customer.
- Address: Residential address.
- Phone: Contact number.
- Email: Email address.
- LicenseNumber: Driver’s license number.

**Relationships:**
Connected to **Reservation** through the *Makes* relationship and to **Feedback** through the *Provides* relationship.

---

### 2. Car

**Attributes:**

- CarID: Unique identifier for each car.
- Make: Car manufacturer.
- Model: Model name of the car.
- Year: Year of manufacture.
- Color: Color of the car.
- RegistrationNo: Vehicle registration number.
- DailyRate: Daily rental rate.
- Status: Availability status (Available / Rented / Maintenance).

**Relationships:**
Linked to **Reservation**, **Rental**, **Insurance**, **Maintenance**, **Damage Report**, and **Feedback** through their respective relationships.

---

### 3. Reservation

**Attributes:**

- ReservationID: Unique identifier for each reservation.
- CustomerID: Foreign key referencing Customer.
- CarID: Foreign key referencing Car.
- ReservationDate: Date when the reservation was made.
- PickupDate: Date the car is scheduled for pickup.
- ReturnDate: Expected return date.
- Status: Current reservation status (Pending / Confirmed / Completed).

**Relationships:**
Connected to **Customer** through *Makes*, to **Car** through *Reserves*, and to **Rental** through *Generates* relationship.

---

### 4. Rental

**Attributes:**

- RentalID: Unique identifier for each rental.
- ReservationID: Foreign key referencing Reservation.
- CustomerID: Foreign key referencing Customer.
- CarID: Foreign key referencing Car.
- EmployeeID: Foreign key referencing Employee.
- RentalDate: Date when the car was issued.
- ReturnDate: Actual return date.
- TotalCost: Final rental cost.

**Relationships:**
Linked to **Reservation** through *Generated From* and to **Payment** through *Has* relationship.

---

### 5. Payment

**Attributes:**

- PaymentID: Unique identifier for each payment.
- RentalID: Foreign key referencing Rental.
- PaymentDate: Date of payment.
- Amount: Amount paid.
- PaymentMethod: Type of payment (Cash, Card, UPI, etc.).

**Relationships:**
Linked to **Rental** through the *Belongs To* relationship.

---

### 6. Employee

**Attributes:**

- EmployeeID: Unique identifier for each employee.
- Name: Employee name.
- JobTitle: Role in the company.
- Phone: Contact number.
- Email: Email address.
- BranchID: Foreign key referencing Branch.

**Relationships:**
Connected to **Rental** through *Handles* and to **Branch** through *Works At* relationship.

---

**7. Branch**

**Attributes:**

- BranchID: Unique branch identifier.
- BranchName: Name of the branch.
- Address: Full address.
- City: City location.
- State: State name.
- ContactNumber: Branch phone number.

**Relationships:**
Connected to **Employee** through *Employs* relationship.

---

**8. Car Type**

**Attributes:**

- CarTypeID: Unique type identifier.
- Type: Category name (e.g., Sedan, SUV, Hatchback).
- Description: Details about the car type.

**Relationships:**
Connected to **Car** through *Categorizes* relationship.

---

**9. Car Model**

**Attributes:**

- ModelID: Unique model identifier.
- Make: Manufacturer name.
- ModelName: Name of the model.
- Year: Year of release.

**Relationships:**
Linked to **Car** through *Defines* relationship.

---

**10. Insurance**

**Attributes:**

- InsuranceID: Unique insurance identifier.
- CarID: Foreign key referencing Car.
- PolicyNumber: Policy number.
- InsuranceType: Type of insurance.
- CoverageAmount: Total coverage amount.

**Relationships:**
Connected to **Car** through *Covers* relationship.

---

**11. Damage Report**

**Attributes:**

- DamageReportID: Unique identifier for each damage record.
- CarID: Foreign key referencing Car.
- DateReported: Date the damage was reported.
- Description: Description of the issue.
- RepairCost: Estimated or actual cost of repair.

**Relationships:**
Linked to **Car** through *Reports Damage On* relationship.

---

**12. Maintenance**

**Attributes:**

- MaintenanceID: Unique service identifier.
- CarID: Foreign key referencing Car.
- ServiceDate: Date of maintenance.
- ServiceType: Type of maintenance.
- Details: Service description.
- Cost: Total service cost.

**Relationships:**
Linked to **Car** through *Maintains* relationship.

---

**13. Discount**

**Attributes:**

- DiscountID: Unique identifier.
- Description: Details about the discount offer.
- DiscountPercentage: Value of the discount.
- StartDate: Validity start date.
- EndDate: Validity end date.

**Relationships:**
Connected to **Reservation** through *Applied To* relationship.

---

**14. Feedback**

**Attributes:**

- FeedbackID: Unique feedback identifier.
- CustomerID: Foreign key referencing Customer.
- CarID: Foreign key referencing Car.
- Rating: Customer's rating (1–5).
- Comments: Feedback remarks.
- Date: Date feedback was submitted.

**Relationships:**
Linked to **Customer** through *Provided By* and to **Car** through *For* relationship.

---

The **Entity–Relationship Diagram (ERD)** for the **Car Rental Management System** illustrates how data is structured and connected across the main modules such as **customers, cars, reservations, rentals, payments, employees, and branches**.

The **Car** entity is central to the system, maintaining complete details about each vehicle including model, registration number, rate, and status. It links with **Reservation**, **Rental**, **Maintenance**, and **Insurance** to manage availability, usage, and safety.

The **Customer** entity handles user information like contact and license details and connects to **Reservation**, **Rental**, and **Feedback** for complete tracking of their rental history. The **Reservation** and **Rental** entities ensure smooth processing from booking to car return, while **Payment** records financial transactions securely.

**Employees** and **Branches** manage operations across locations, while **Maintenance**, **Damage Report**, and **Insurance** ensure the fleet's reliability. **Discount** and **Feedback** improve customer satisfaction and service quality.

# RELATION SCHEMA FOR CAR RENTAL MANAGEMENT SYSTEM

# CHAPTER 2 :

## CUSTOMER TABLE:

- Customer_ID
- Customer_Name
- Customer_Address
- Customer_Phone
- Customer_Email
- License_No

**Schema:**
Customer (Customer_ID, Customer_Name, Customer_Address, Customer_Phone, Customer_Email, License_No)

## CAR TABLE :

- Car_ID
- Car_Model
- Car_Type
- Car_Year
- Car_Color
- Registration_No
- Daily_Rate
- Status

**Schema:**
Car (Car_ID, Car_Model, Car_Type, Car_Year, Car_Color, Registration_No, Daily_Rate, Status)

## RESERVATION TABLE:

- Reservation_ID
- Customer_ID
- Car_ID
- Reservation_Date
- Pickup_Date
- Return_Date
- Status

**Schema:**
Reservation (Reservation_ID, Customer_ID, Car_ID, Reservation_Date, Pickup_Date, Return_Date, Status)

## RENTAL TABLE:

- Rental_ID
- Reservation_ID
- Customer_ID
- Car_ID
- Employee_ID
- Rental_Date
- Return_Date
- Total_Cost

  **Schema:**
  Rental (Rental_ID, Reservation_ID, Customer_ID, Car_ID, Employee_ID, Rental_Date, Return_Date, Total_Cost)

## PAYMENT TABLE:

- Payment_ID
- Rental_ID
- Payment_Date
- Amount
- Payment_Method

  **Schema:**
  Payment (Payment_ID, Rental_ID, Payment_Date, Amount, Payment_Method)

## EMPLOYEE TABLE:

- Employee_ID
- Employee_Name
- Job_Title
- Phone_No
- Email
- Branch_ID
-

  **Schema:**
  Employee (Employee_ID, Employee_Name, Job_Title, Phone_No, Email, Branch_ID)

## BRANCH TABLE:

- Branch_ID
- Branch_Name
- Branch_Address
- City
- State

**Schema:**
Branch (Branch_ID, Branch_Name, Branch_Address, City, State)

## CAR TYPE TABLE:

- CarType_ID
- Type
- Description

**Schema:**
CarType (CarType_ID, Type, Description)

## INSURANCE TABLE:

- Insurance_ID
- Car_ID
- Policy_Number
- Insurance_Type
- Coverage_Amount

**Schema:**
Insurance (Insurance_ID, Car_ID, Policy_Number, Insurance_Type, Coverage_Amount)

## MAINTENANCE TABLE:

- Maintenance_ID
- Car_ID
- Service_Date
- Service_Type
- Details
- Cost

**Schema:**
Maintenance (Maintenance_ID, Car_ID, Service_Date, Service_Type, Details, Cost)

## DISCOUNT TABLE:

- Discount_ID
- Description
- Discount_Percentage
- Start_Date
- End_Date

**Schema:**
Discount (Discount_ID, Description, Discount_Percentage, Start_Date, End_Date)

## FEEDBACK TABLE:

- Feedback_ID
- Customer_ID
- Car_ID
- Rating
- Comments
- Date

**Schema:**
Feedback (Feedback_ID, Customer_ID, Car_ID, Rating, Comments, Date)

## COMPLEX QUERIES BASED ON THE CONCEPTS OF CONSTRAINTS, SETS, JOINS, VIEWS, TRIGGERS AND CURSORS:

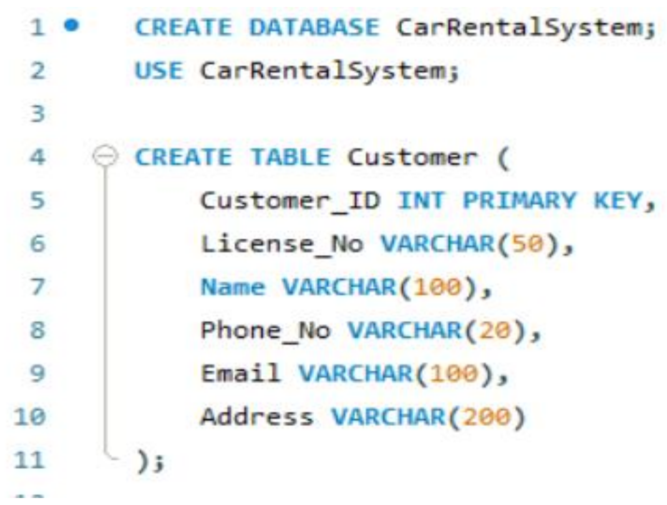
CREATION OF THE SQL DATABASE USING SQL COMMANDS

**Creating a Database:**

1. Customer
2. Car
3. Reservation
4. Rental
5. Payment
6. Employee
7. Branch
8. Car Type
9. Car Model
10. Insurance
11. Damage Report
12. Maintenance
13. Vehicle

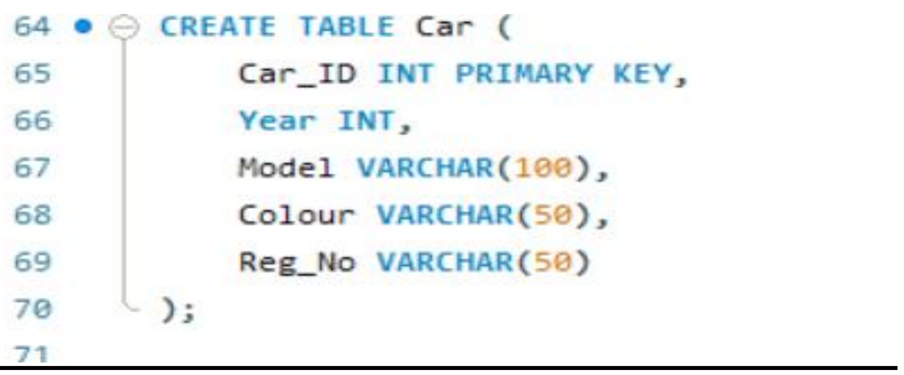
## CREATION OF THE SQL DATABASE USING SQL COMMANDS

## 1. CUSTOMER:



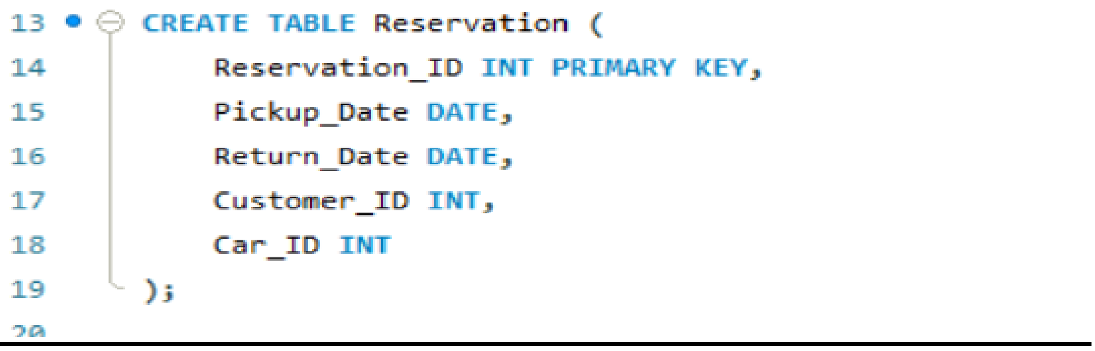
```sql
CREATE DATABASE CarRentalSystem;
USE CarRentalSystem;

CREATE TABLE Customer (
    Customer_ID INT PRIMARY KEY,
    License_No VARCHAR(50),
    Name VARCHAR(100),
    Phone_No VARCHAR(20),
    Email VARCHAR(100),
    Address VARCHAR(200)
);
```

## 2.CAR:



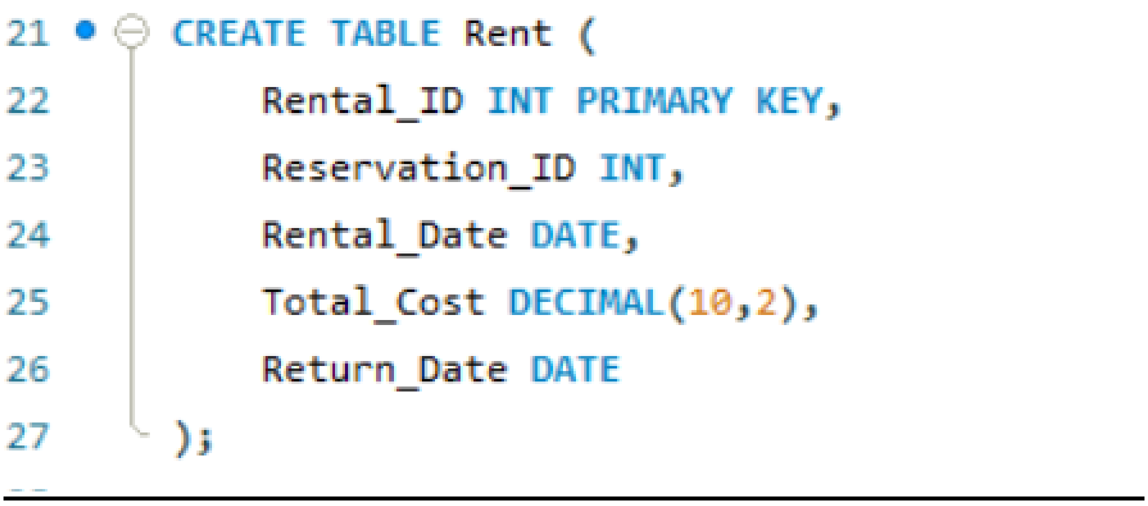
```
CREATE TABLE Car (
        Car_ID INT PRIMARY KEY,
        Year INT,
        Model VARCHAR(100),
        Colour VARCHAR(50),
        Reg_No VARCHAR(50)
);
```

**3. RESERVATION:**



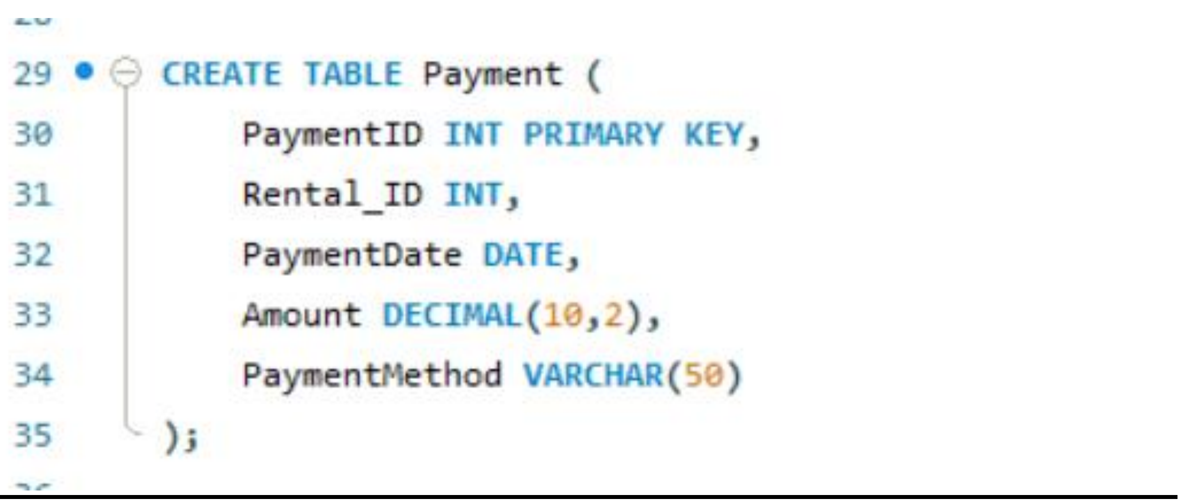
```
CREATE TABLE Reservation (
        Reservation_ID INT PRIMARY KEY,
        Pickup_Date DATE,
        Return_Date DATE,
        Customer_ID INT,
        Car_ID INT
);
```

**4. RENTAL**



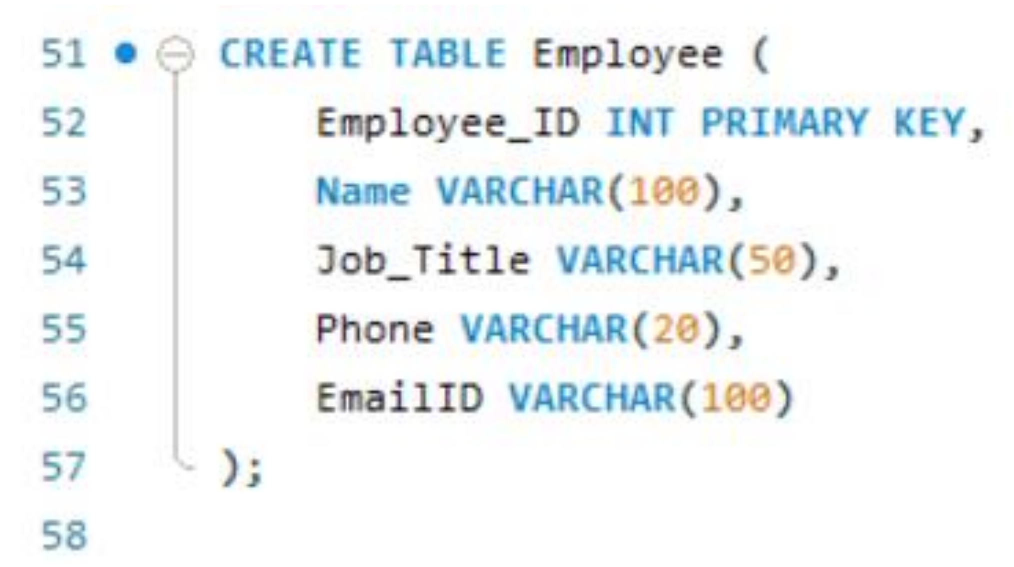
```
CREATE TABLE Rent (
        Rental_ID INT PRIMARY KEY,
        Reservation_ID INT,
        Rental_Date DATE,
        Total_Cost DECIMAL(10,2),
        Return_Date DATE
);
```

**5. PAYMENT**



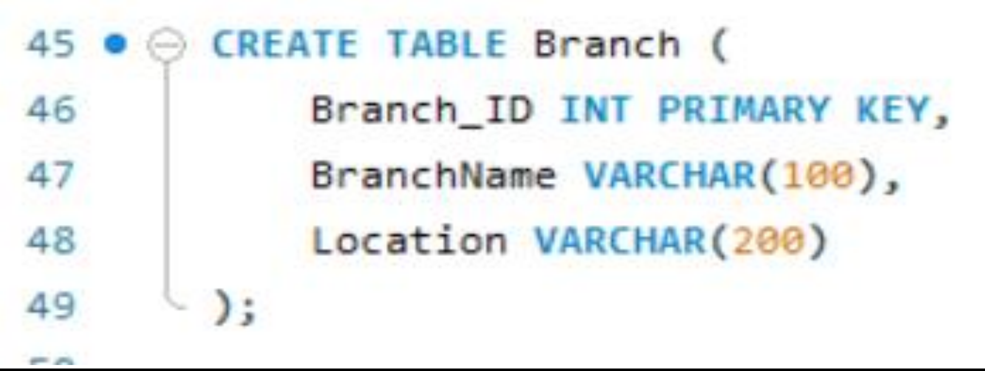
```
CREATE TABLE Payment (
        PaymentID INT PRIMARY KEY,
        Rental_ID INT,
        PaymentDate DATE,
        Amount DECIMAL(10,2),
        PaymentMethod VARCHAR(50)
);
```

**6. EMPLOYEE**



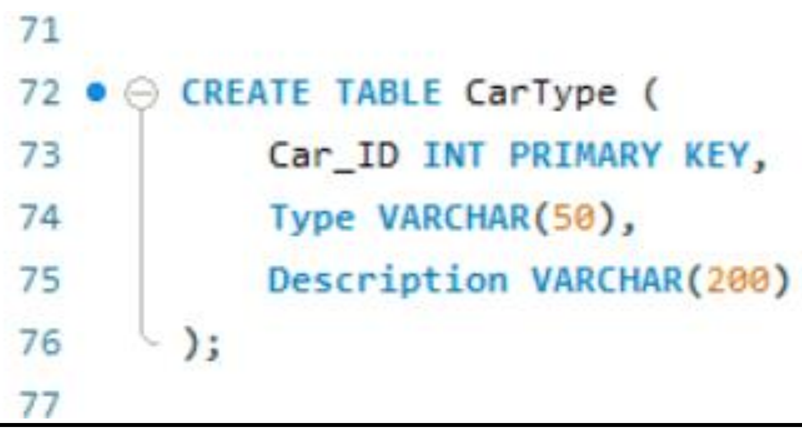
```sql
CREATE TABLE Employee (
    Employee_ID INT PRIMARY KEY,
    Name VARCHAR(100),
    Job_Title VARCHAR(50),
    Phone VARCHAR(20),
    EmailID VARCHAR(100)
);
```

**7. BRANCH**



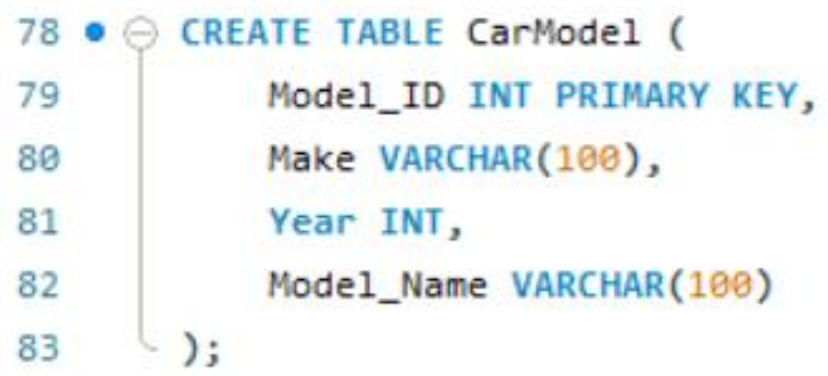
```sql
CREATE TABLE Branch (
    Branch_ID INT PRIMARY KEY,
    BranchName VARCHAR(100),
    Location VARCHAR(200)
);
```

**8. CAR TYPE**



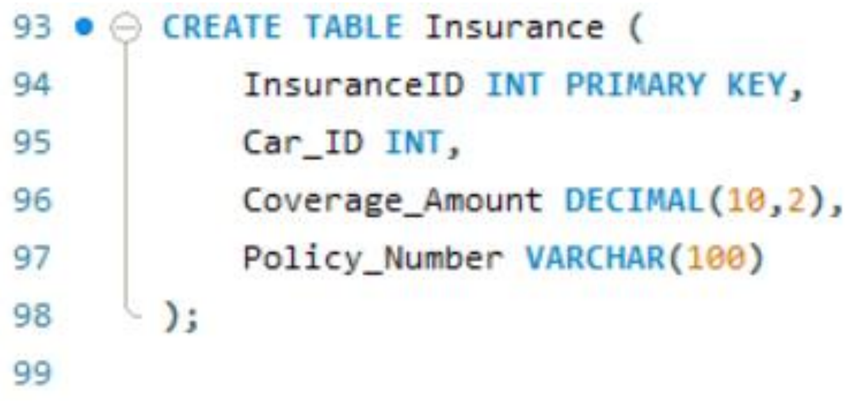
```
CREATE TABLE CarType (
    Car_ID INT PRIMARY KEY,
    Type VARCHAR(50),
    Description VARCHAR(200)
);
```

**9. CAR MODEL**



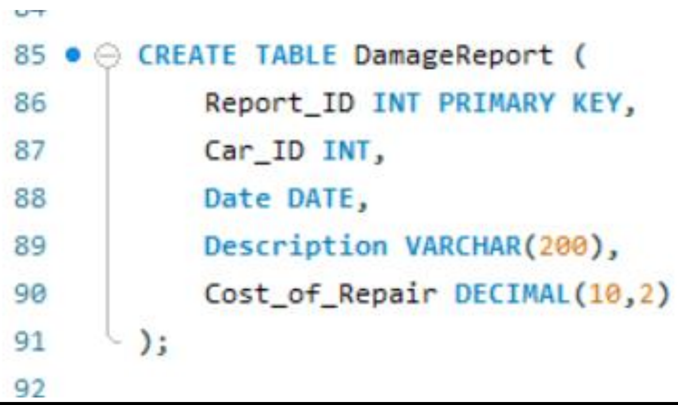
```
CREATE TABLE CarModel (
    Model_ID INT PRIMARY KEY,
    Make VARCHAR(100),
    Year INT,
    Model_Name VARCHAR(100)
);
```

**10. INSURANCE**



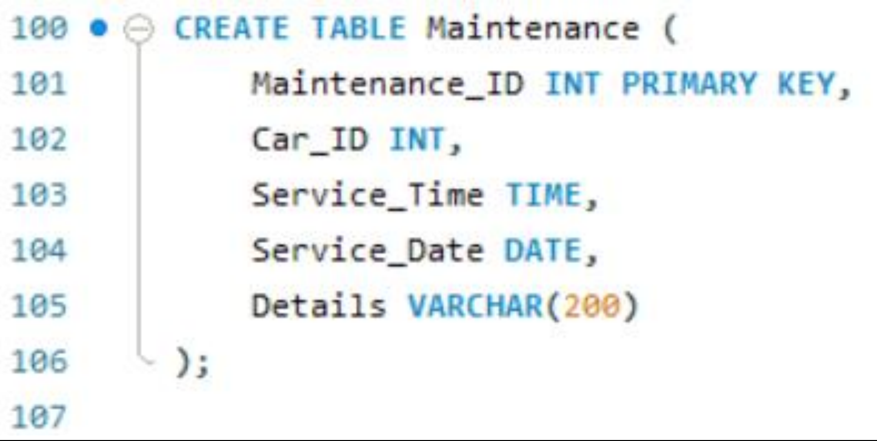
```sql
CREATE TABLE Insurance (
    InsuranceID INT PRIMARY KEY,
    Car_ID INT,
    Coverage_Amount DECIMAL(10,2),
    Policy_Number VARCHAR(100)
);
```

## 11. DAMAGE REPORT

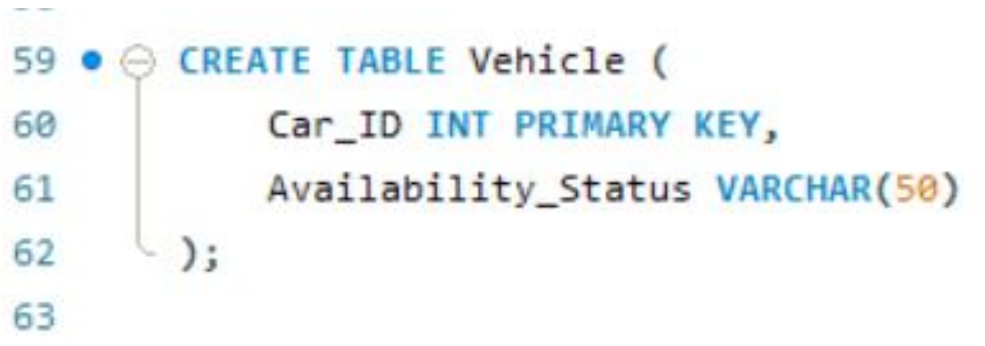


```sql
CREATE TABLE DamageReport (
    Report_ID INT PRIMARY KEY,
    Car_ID INT,
    Date DATE,
    Description VARCHAR(200),
    Cost_of_Repair DECIMAL(10,2)
);
```

## 12. MAINTENANCE


```sql
CREATE TABLE Maintenance (
    Maintenance_ID INT PRIMARY KEY,
    Car_ID INT,
    Service_Time TIME,
    Service_Date DATE,
    Details VARCHAR(200)
);
```

## 13. VEHICLE


```sql
CREATE TABLE Vehicle (
    Car_ID INT PRIMARY KEY,
    Availability_Status VARCHAR(50)
);
```

## COMMENTS ON SQL CONCEPTS USED (CHAPTER-3)

1. SET OPERATIONS:

Set operations are used to combine the results of two or more SELECT statements into a single result set. They help retrieve related data from multiple tables while removing or including duplicates. These operations simplify data comparison and integration across entities.

2. JOINS:

Joins are used to connect rows from two or more tables based on a common field. They enable retrieval of related information stored in different tables. This concept is essential for displaying combined data such as customer, order, and product details together.

3. COMPLEX QUERIES:

Complex queries involve multiple SQL clauses such as joins, subqueries, and aggregate functions. They are used to analyze, filter, and summarize data from multiple tables. These queries help generate meaningful business insights from large datasets.

4. CURSOR:

A cursor is a database object that allows row-by-row processing of query results. It is used in stored procedures to handle data sequentially. Cursors are useful when each record needs to be processed individually in complex transactions.

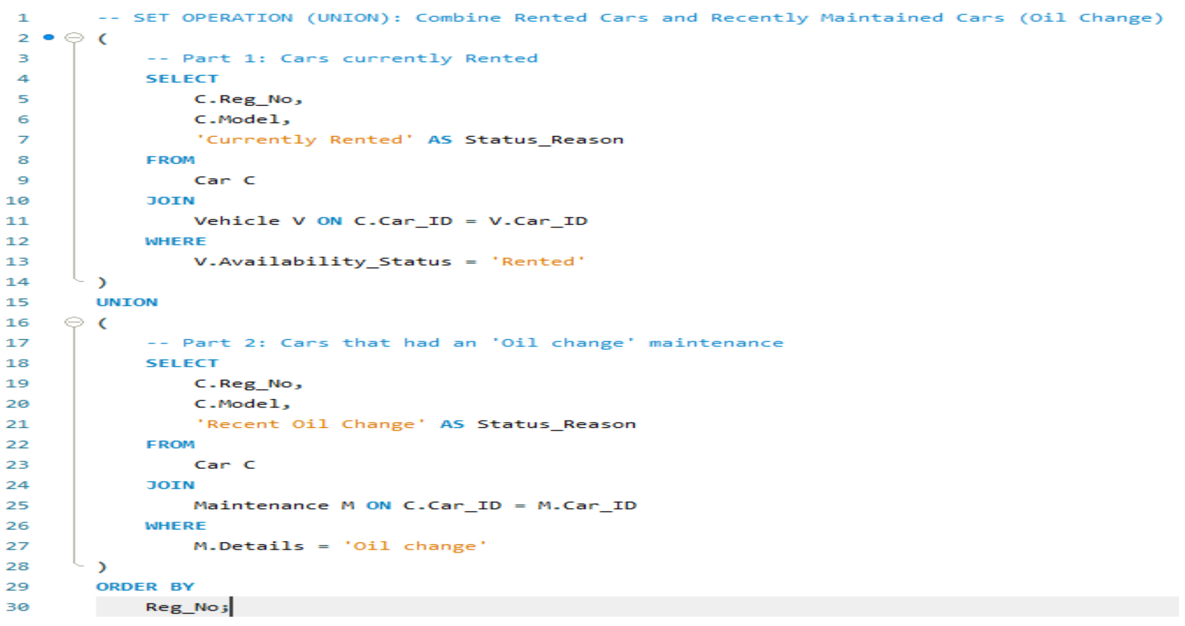
5. VIEW:

A view is a virtual table that displays data derived from one or more tables. It simplifies complex queries and enhances data security by restricting direct access to base tables. Views help users focus only on relevant information.

6. TRIGGER:

A trigger is an automated process that executes in response to specific database events such as INSERT, UPDATE, or DELETE. It maintains data integrity and ensures that dependent records are updated automatically. Triggers improve consistency and automation in databases.

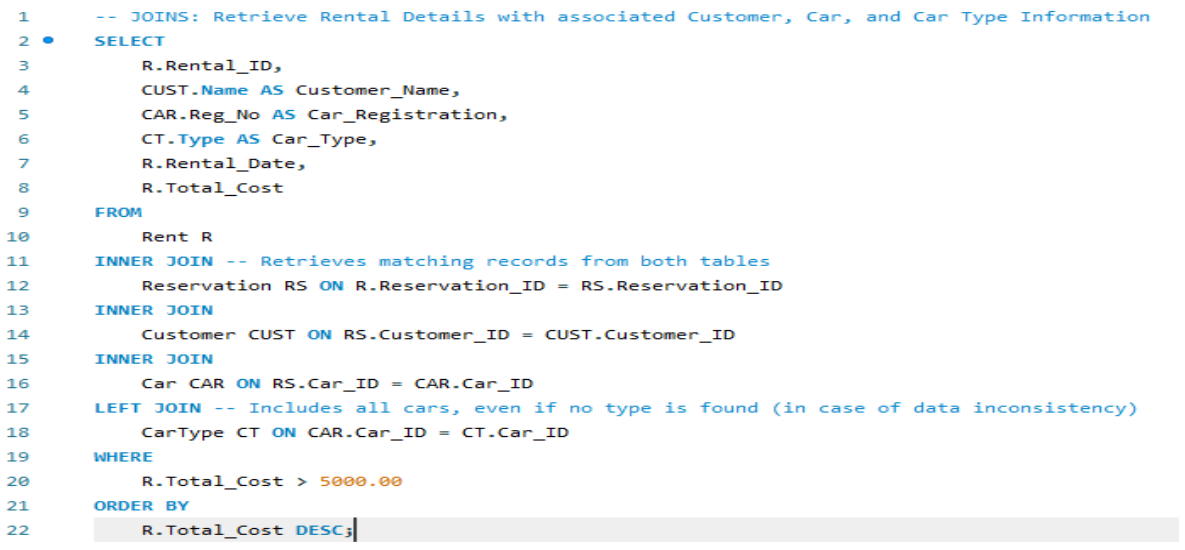
## **1.SET OPERATIONS:**



```
-- SET OPERATION (UNION): Combine Rented Cars and Recently Maintained Cars (Oil Change)
(
    -- Part 1: Cars currently Rented
    SELECT
        C.Reg_No,
        C.Model,
        'Currently Rented' AS Status_Reason
    FROM
        Car C
    JOIN
        Vehicle V ON C.Car_ID = V.Car_ID
    WHERE
        V.Availability_Status = 'Rented'
)
UNION
(
    -- Part 2: Cars that had an 'Oil change' maintenance
    SELECT
        C.Reg_No,
        C.Model,
        'Recent Oil Change' AS Status_Reason
    FROM
        Car C
    JOIN
        Maintenance M ON C.Car_ID = M.Car_ID
    WHERE
        M.Details = 'Oil change'
)
ORDER BY
    Reg_No;
```

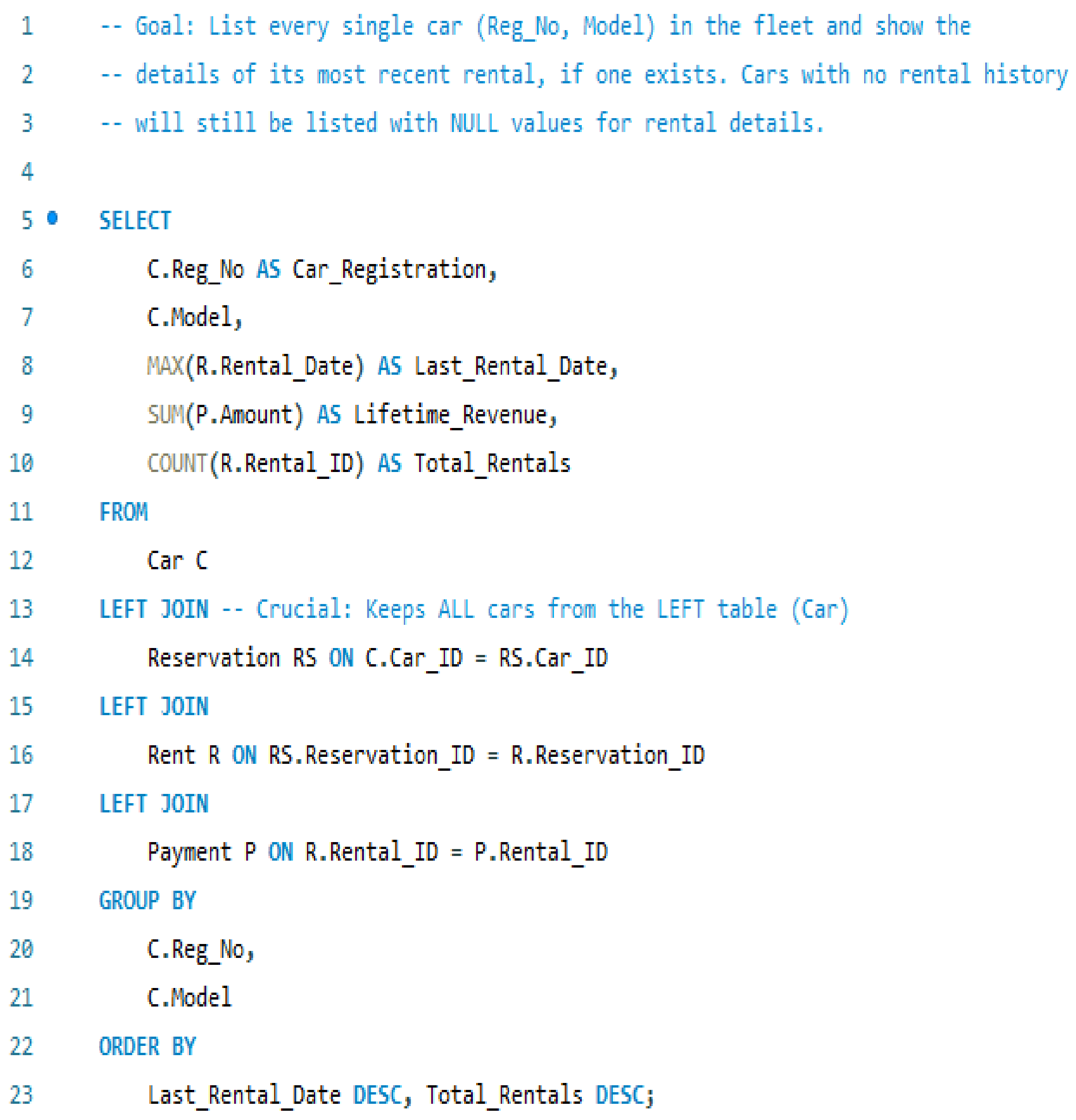
## **2.JOINS:**

\



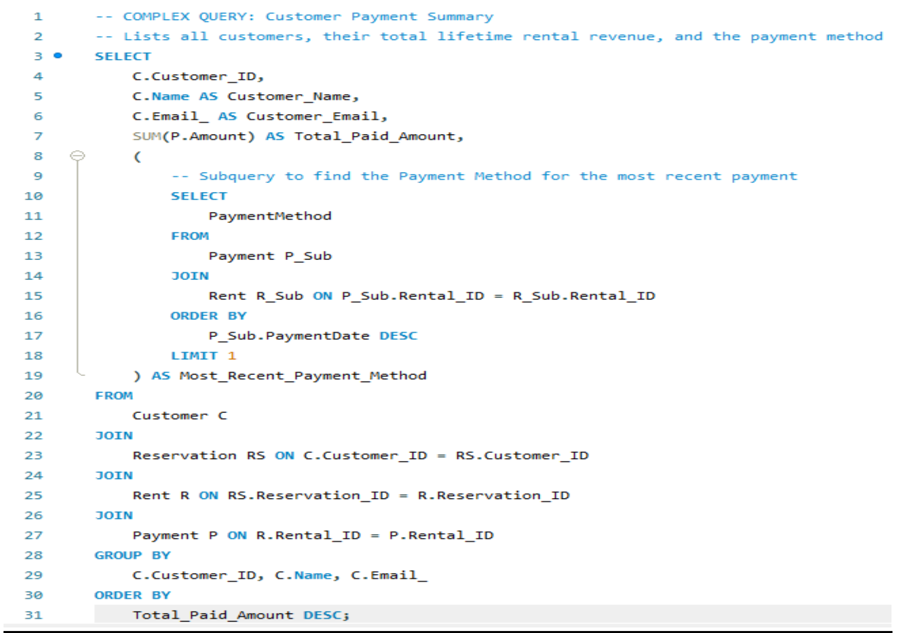
```
-- JOINS: Retrieve Rental Details with associated Customer, Car, and Car Type Information
SELECT
    R.Rental_ID,
    CUST.Name AS Customer_Name,
    CAR.Reg_No AS Car_Registration,
    CT.Type AS Car_Type,
    R.Rental_Date,
    R.Total_Cost
FROM
    Rent R
INNER JOIN -- Retrieves matching records from both tables
    Reservation RS ON R.Reservation_ID = RS.Reservation_ID
INNER JOIN
    Customer CUST ON RS.Customer_ID = CUST.Customer_ID
INNER JOIN
    Car CAR ON RS.Car_ID = CAR.Car_ID
LEFT JOIN -- Includes all cars, even if no type is found (in case of data inconsistency)
    CarType CT ON CAR.Car_ID = CT.Car_ID
WHERE
    R.Total_Cost > 5000.00
ORDER BY
    R.Total_Cost DESC;
```

## LEFT JOIN:

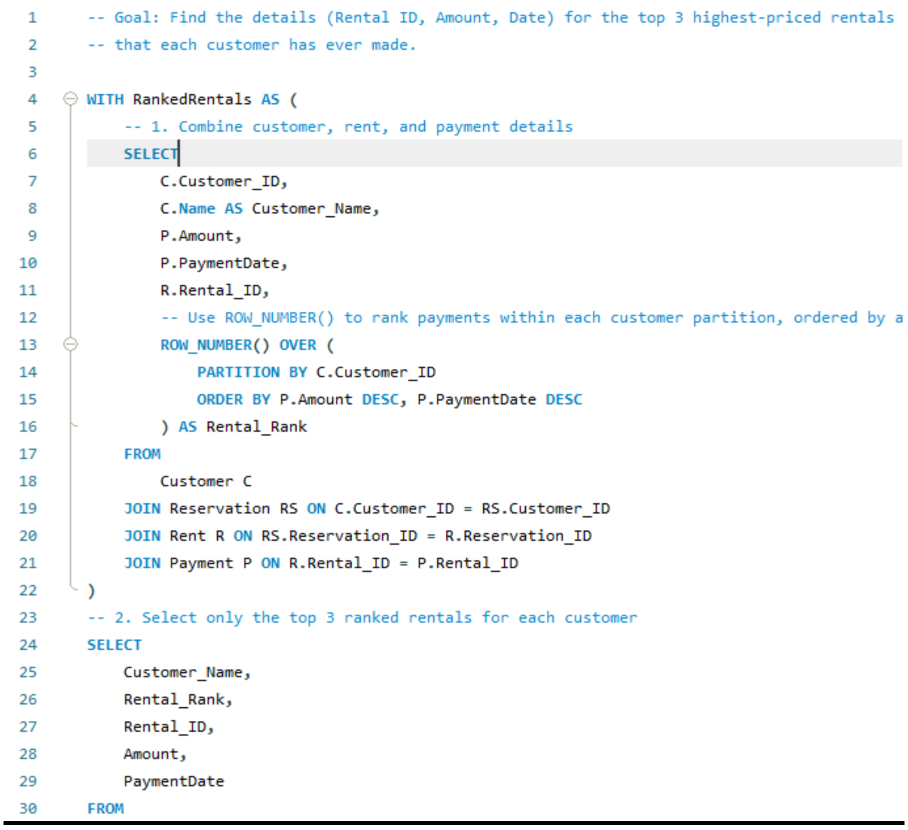


```sql
-- Goal: List every single car (Reg_No, Model) in the fleet and show the
-- details of its most recent rental, if one exists. Cars with no rental history
-- will still be listed with NULL values for rental details.

SELECT
    C.Reg_No AS Car_Registration,
    C.Model,
    MAX(R.Rental_Date) AS Last_Rental_Date,
    SUM(P.Amount) AS Lifetime_Revenue,
    COUNT(R.Rental_ID) AS Total_Rentals
FROM
    Car C
LEFT JOIN -- Crucial: Keeps ALL cars from the LEFT table (Car)
    Reservation RS ON C.Car_ID = RS.Car_ID
LEFT JOIN
    Rent R ON RS.Reservation_ID = R.Reservation_ID
LEFT JOIN
    Payment P ON R.Rental_ID = P.Rental_ID
GROUP BY
    C.Reg_No,
    C.Model
ORDER BY
    Last_Rental_Date DESC, Total_Rentals DESC;
```

## 3.COMPLEX QUERIES:

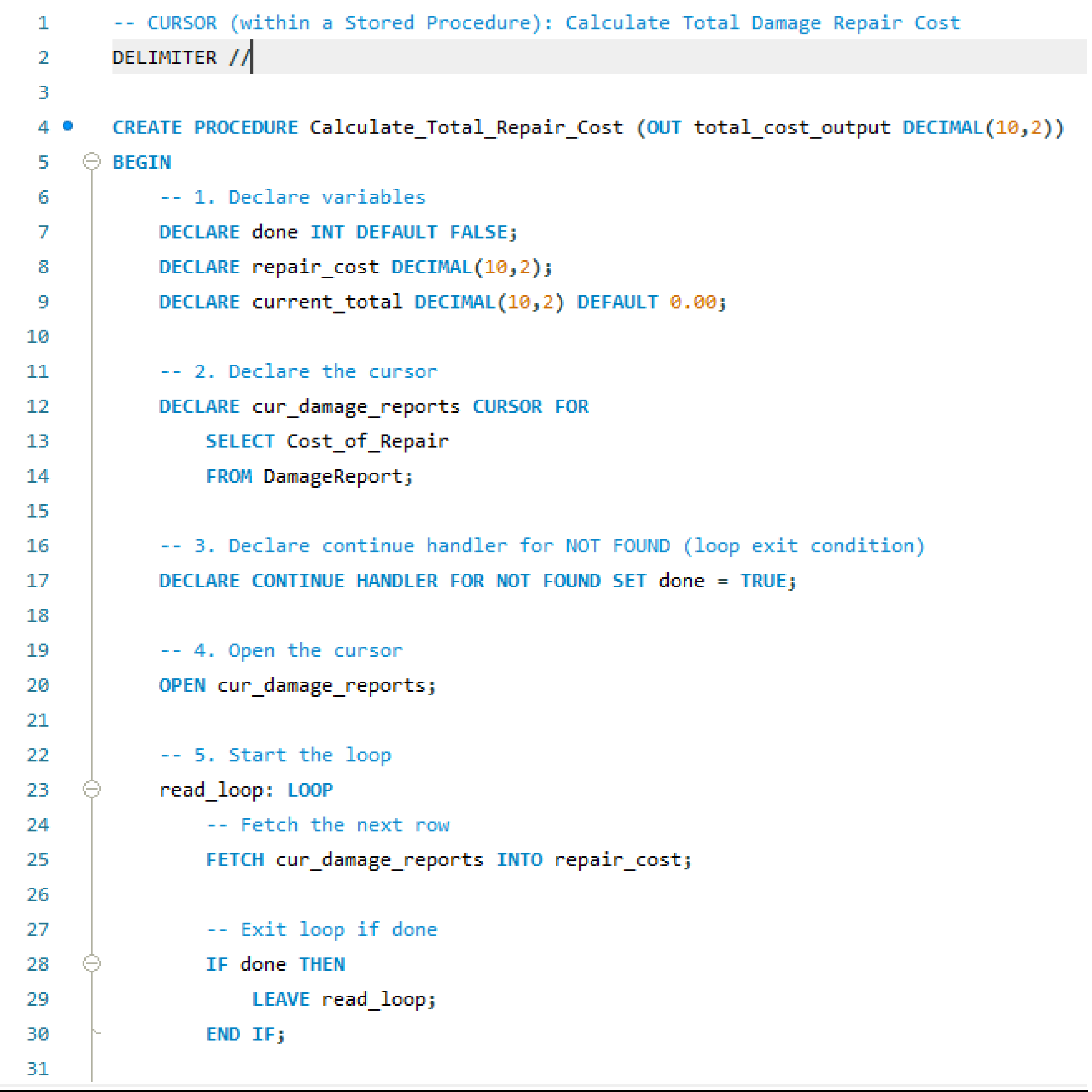


```sql
-- COMPLEX QUERY: Customer Payment Summary
-- Lists all customers, their total lifetime rental revenue, and the payment method
SELECT
    C.Customer_ID,
    C.Name AS Customer_Name,
    C.Email_ AS Customer_Email,
    SUM(P.Amount) AS Total_Paid_Amount,
    (
        -- Subquery to find the Payment Method for the most recent payment
        SELECT
            PaymentMethod
        FROM
            Payment P_Sub
        JOIN
            Rent R_Sub ON P_Sub.Rental_ID = R_Sub.Rental_ID
        ORDER BY
            P_Sub.PaymentDate DESC
        LIMIT 1
    ) AS Most_Recent_Payment_Method
FROM
    Customer C
JOIN
    Reservation RS ON C.Customer_ID = RS.Customer_ID
JOIN
    Rent R ON RS.Reservation_ID = R.Reservation_ID
JOIN
    Payment P ON R.Rental_ID = P.Rental_ID
GROUP BY
    C.Customer_ID, C.Name, C.Email_
ORDER BY
    Total_Paid_Amount DESC;
```



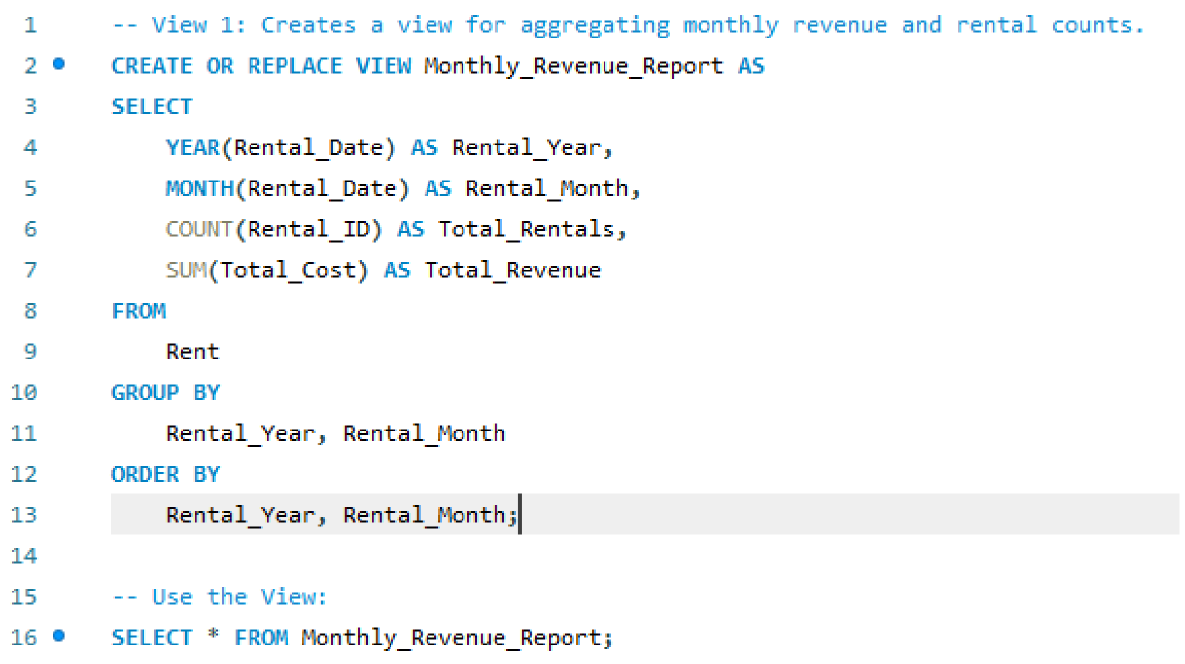
```sql
-- Goal: Find the details (Rental ID, Amount, Date) for the top 3 highest-priced rentals
-- that each customer has ever made.

WITH RankedRentals AS (
    -- 1. Combine customer, rent, and payment details
    SELECT
        C.Customer_ID,
        C.Name AS Customer_Name,
        P.Amount,
        P.PaymentDate,
        R.Rental_ID,
        -- Use ROW_NUMBER() to rank payments within each customer partition, ordered by a
        ROW_NUMBER() OVER (
            PARTITION BY C.Customer_ID
            ORDER BY P.Amount DESC, P.PaymentDate DESC
        ) AS Rental_Rank
    FROM
        Customer C
    JOIN Reservation RS ON C.Customer_ID = RS.Customer_ID
    JOIN Rent R ON RS.Reservation_ID = R.Reservation_ID
    JOIN Payment P ON R.Rental_ID = P.Rental_ID
)
-- 2. Select only the top 3 ranked rentals for each customer
SELECT
    Customer_Name,
    Rental_Rank,
    Rental_ID,
    Amount,
    PaymentDate
FROM
```

## 4.CURSOR:

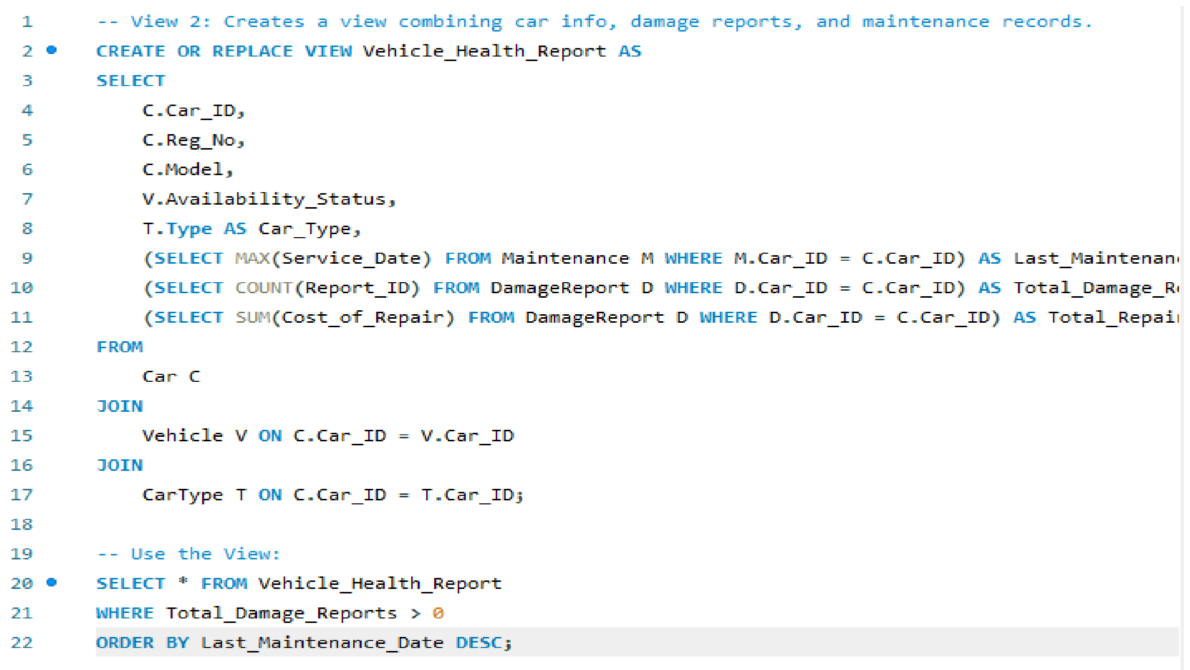


```sql
-- CURSOR (within a Stored Procedure): Calculate Total Damage Repair Cost
DELIMITER //

CREATE PROCEDURE Calculate_Total_Repair_Cost (OUT total_cost_output DECIMAL(10,2))
BEGIN
    -- 1. Declare variables
    DECLARE done INT DEFAULT FALSE;
    DECLARE repair_cost DECIMAL(10,2);
    DECLARE current_total DECIMAL(10,2) DEFAULT 0.00;

    -- 2. Declare the cursor
    DECLARE cur_damage_reports CURSOR FOR
        SELECT Cost_of_Repair
        FROM DamageReport;

    -- 3. Declare continue handler for NOT FOUND (loop exit condition)
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    -- 4. Open the cursor
    OPEN cur_damage_reports;

    -- 5. Start the loop
    read_loop: LOOP
        -- Fetch the next row
        FETCH cur_damage_reports INTO repair_cost;

        -- Exit loop if done
        IF done THEN
            LEAVE read_loop;
        END IF;

```

## 5.VIEW:

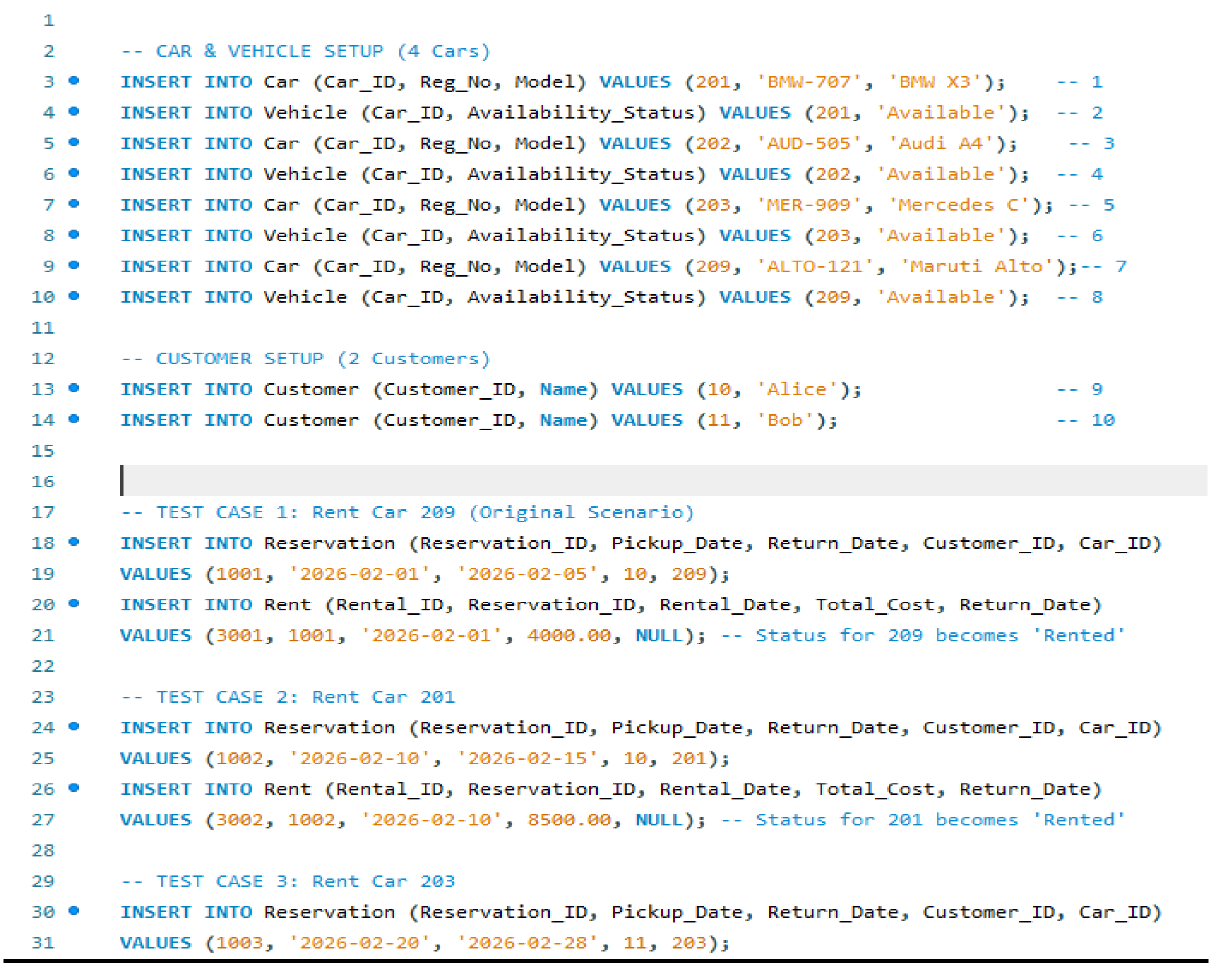


```sql
-- View 1: Creates a view for aggregating monthly revenue and rental counts.
CREATE OR REPLACE VIEW Monthly_Revenue_Report AS
SELECT
    YEAR(Rental_Date) AS Rental_Year,
    MONTH(Rental_Date) AS Rental_Month,
    COUNT(Rental_ID) AS Total_Rentals,
    SUM(Total_Cost) AS Total_Revenue
FROM
    Rent
GROUP BY
    Rental_Year, Rental_Month
ORDER BY
    Rental_Year, Rental_Month;

-- Use the View:
SELECT * FROM Monthly_Revenue_Report;
```

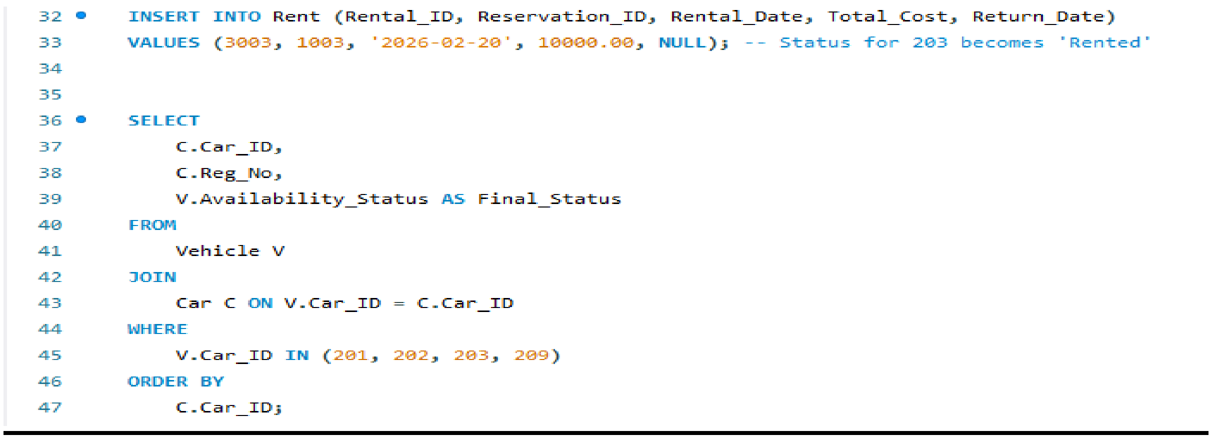


```sql
-- View 2: Creates a view combining car info, damage reports, and maintenance records.
CREATE OR REPLACE VIEW Vehicle_Health_Report AS
SELECT
    C.Car_ID,
    C.Reg_No,
    C.Model,
    V.Availability_Status,
    T.Type AS Car_Type,
    (SELECT MAX(Service_Date) FROM Maintenance M WHERE M.Car_ID = C.Car_ID) AS Last_Maintenan
    (SELECT COUNT(Report_ID) FROM DamageReport D WHERE D.Car_ID = C.Car_ID) AS Total_Damage_R
    (SELECT SUM(Cost_of_Repair) FROM DamageReport D WHERE D.Car_ID = C.Car_ID) AS Total_Repai
FROM
    Car C
JOIN
    Vehicle V ON C.Car_ID = V.Car_ID
JOIN
    CarType T ON C.Car_ID = T.Car_ID;

-- Use the View:
SELECT * FROM Vehicle_Health_Report
WHERE Total_Damage_Reports > 0
ORDER BY Last_Maintenance_Date DESC;
```

## 6.TRIGGER:



```sql

-- CAR & VEHICLE SETUP (4 Cars)
INSERT INTO Car (Car_ID, Reg_No, Model) VALUES (201, 'BMW-707', 'BMW X3');     -- 1
INSERT INTO Vehicle (Car_ID, Availability_Status) VALUES (201, 'Available');  -- 2
INSERT INTO Car (Car_ID, Reg_No, Model) VALUES (202, 'AUD-505', 'Audi A4');     -- 3
INSERT INTO Vehicle (Car_ID, Availability_Status) VALUES (202, 'Available');  -- 4
INSERT INTO Car (Car_ID, Reg_No, Model) VALUES (203, 'MER-909', 'Mercedes C'); -- 5
INSERT INTO Vehicle (Car_ID, Availability_Status) VALUES (203, 'Available');  -- 6
INSERT INTO Car (Car_ID, Reg_No, Model) VALUES (209, 'ALTO-121', 'Maruti Alto');-- 7
INSERT INTO Vehicle (Car_ID, Availability_Status) VALUES (209, 'Available');  -- 8

-- CUSTOMER SETUP (2 Customers)
INSERT INTO Customer (Customer_ID, Name) VALUES (10, 'Alice');                 -- 9
INSERT INTO Customer (Customer_ID, Name) VALUES (11, 'Bob');                   -- 10


-- TEST CASE 1: Rent Car 209 (Original Scenario)
INSERT INTO Reservation (Reservation_ID, Pickup_Date, Return_Date, Customer_ID, Car_ID)
VALUES (1001, '2026-02-01', '2026-02-05', 10, 209);
INSERT INTO Rent (Rental_ID, Reservation_ID, Rental_Date, Total_Cost, Return_Date)
VALUES (3001, 1001, '2026-02-01', 4000.00, NULL); -- Status for 209 becomes 'Rented'

-- TEST CASE 2: Rent Car 201
INSERT INTO Reservation (Reservation_ID, Pickup_Date, Return_Date, Customer_ID, Car_ID)
VALUES (1002, '2026-02-10', '2026-02-15', 10, 201);
INSERT INTO Rent (Rental_ID, Reservation_ID, Rental_Date, Total_Cost, Return_Date)
VALUES (3002, 1002, '2026-02-10', 8500.00, NULL); -- Status for 201 becomes 'Rented'

-- TEST CASE 3: Rent Car 203
INSERT INTO Reservation (Reservation_ID, Pickup_Date, Return_Date, Customer_ID, Car_ID)
VALUES (1003, '2026-02-20', '2026-02-28', 11, 203);
```



```sql
INSERT INTO Rent (Rental_ID, Reservation_ID, Rental_Date, Total_Cost, Return_Date)
VALUES (3003, 1003, '2026-02-20', 10000.00, NULL); -- Status for 203 becomes 'Rented'


SELECT
    C.Car_ID,
    C.Reg_No,
    V.Availability_Status AS Final_Status
FROM
    Vehicle V
JOIN
    Car C ON V.Car_ID = C.Car_ID
WHERE
    V.Car_ID IN (201, 202, 203, 209)
ORDER BY
    C.Car_ID;
```

# NORMALIZATION: (CHAPTER-4)

In database design, **Normalization** is the process of organizing data in a structured manner to **eliminate redundancy** and **maintain data integrity**. It divides large, complex tables into smaller, related ones while ensuring logical data relationships and reducing anomalies in insert, update, and delete operations.

Each **Normal Form (NF)**—from **First Normal Form (1NF)** to **Boyce–Codd Normal Form (BCNF)**—addresses specific dependency issues to ensure that the database is both efficient and consistent.

In this project, the **Reservation** and **Rental** tables required normalization up to **BCNF** due to partial and transitive dependencies in the initial design.

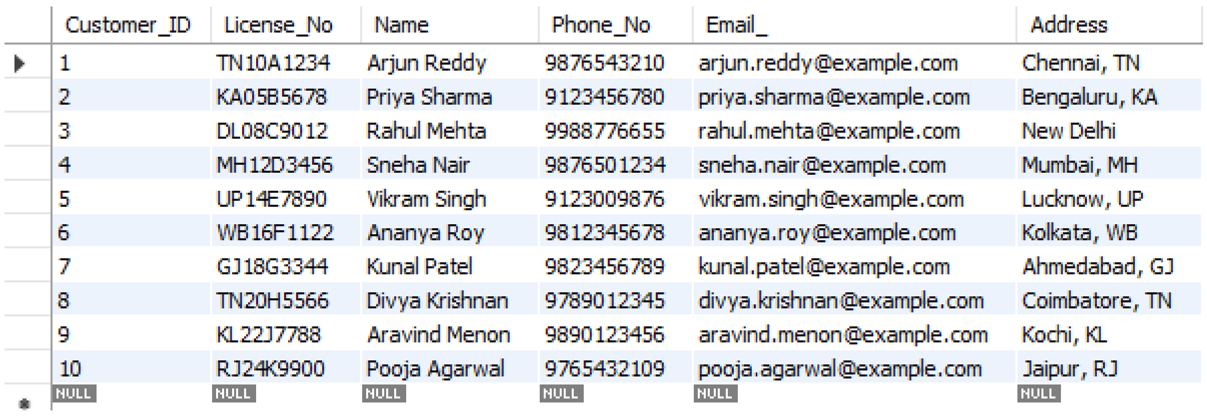
Originally, some attributes such as TotalCost and ReturnDate were partially dependent on non-primary fields like Car_ID or Customer_ID instead of the composite key (Reservation_ID, Car_ID).
To satisfy **BCNF**, these attributes were separated into distinct tables—**Reservation**, **Rental**, and **Payment**—ensuring that every determinant is a **candidate key** and all non-key attributes depend only on the primary key.

The remaining tables—**Customer**, **Car**, **Employee**, **Branch**, and **Insurance**—were already in **Third Normal Form (3NF)** because:

- All non-key attributes depend entirely on the primary key.
- No transitive dependencies exist.
- Each table stores unique, non-repetitive information.

## 1. FIRST NORMAL FORM (1NF):

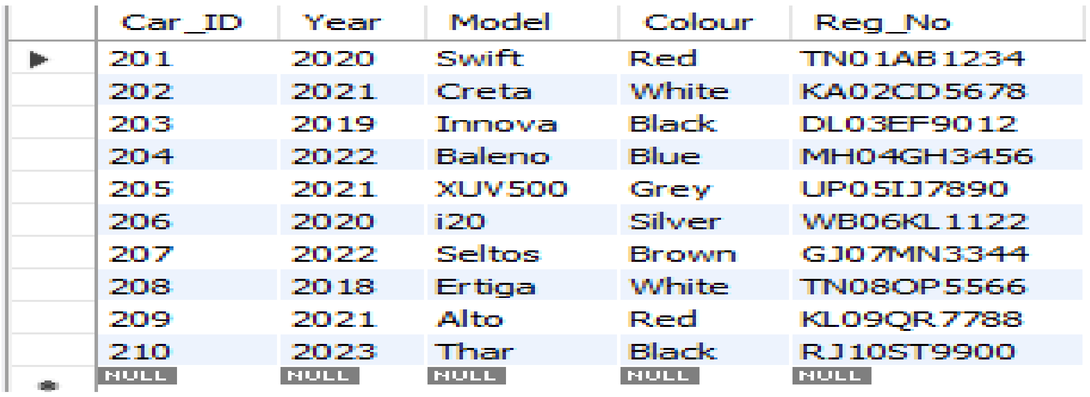
A table is in **First Normal Form (1NF)** when all its attributes contain **only atomic (indivisible) values**, and each record is unique. It eliminates repeating groups or arrays, ensuring that every field holds only one value. This step organizes data into a simple, tabular structure for easy retrieval.



| Customer_ID | License_No | Name | Phone_No | Email_ | Address |
|---|---|---|---|---|---|
| 1 | TN10A1234 | Arjun Reddy | 9876543210 | arjun.reddy@example.com | Chennai, TN |
| 2 | KA05B5678 | Priya Sharma | 9123456780 | priya.sharma@example.com | Bengaluru, KA |
| 3 | DL08C9012 | Rahul Mehta | 9988776655 | rahul.mehta@example.com | New Delhi |
| 4 | MH12D3456 | Sneha Nair | 9876501234 | sneha.nair@example.com | Mumbai, MH |
| 5 | UP14E7890 | Vikram Singh | 9123009876 | vikram.singh@example.com | Lucknow, UP |
| 6 | WB16F1122 | Ananya Roy | 9812345678 | ananya.roy@example.com | Kolkata, WB |
| 7 | GJ18G3344 | Kunal Patel | 9823456789 | kunal.patel@example.com | Ahmedabad, GJ |
| 8 | TN20H5566 | Divya Krishnan | 9789012345 | divya.krishnan@example.com | Coimbatore, TN |
| 9 | KL22J7788 | Aravind Menon | 9890123456 | aravind.menon@example.com | Kochi, KL |
| 10 | RJ24K9900 | Pooja Agarwal | 9765432109 | pooja.agarwal@example.com | Jaipur, RJ |
| NULL | NULL | NULL | NULL | NULL | NULL |

## 2. SECOND NORMAL FORM (2NF):

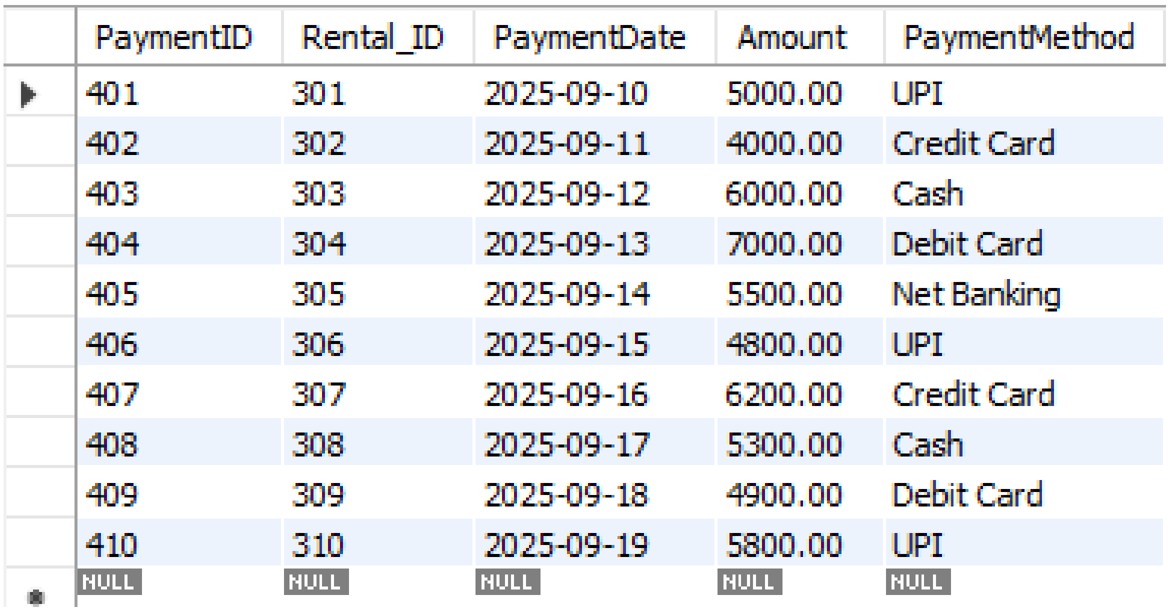
A table is in **Second Normal Form (2NF)** if it is already in 1NF and all **non-key attributes depend entirely on the primary key**. It removes **partial dependency**, meaning that no column should depend on only part of a composite key. This reduces redundancy and improves data consistency.



| Car_ID | Year | Model | Colour | Reg_No |
|---|---|---|---|---|
| 201 | 2020 | Swift | Red | TN01AB1234 |
| 202 | 2021 | Creta | White | KA02CD5678 |
| 203 | 2019 | Innova | Black | DL03EF9012 |
| 204 | 2022 | Baleno | Blue | MH04GH3456 |
| 205 | 2021 | XUV500 | Grey | UP05IJ7890 |
| 206 | 2020 | i20 | Silver | WB06KL1122 |
| 207 | 2022 | Seltos | Brown | GJ07MN3344 |
| 208 | 2018 | Ertiga | White | TN08OP5566 |
| 209 | 2021 | Alto | Red | KL09QR7788 |
| 210 | 2023 | Thar | Black | RJ10ST9900 |
| NULL | NULL | NULL | NULL | NULL |

## 3. THIRD NORMAL FORM (3NF):

A table is in **Third Normal Form (3NF)** when it is already in 2NF and **no non-key attribute depends on another non-key attribute**. This eliminates **transitive dependency**, ensuring that each non-key field depends only on the primary key. 3NF provides a more stable and efficient database design.



| PaymentID | Rental_ID | PaymentDate | Amount | PaymentMethod |
|---|---|---|---|---|
| 401 | 301 | 2025-09-10 | 5000.00 | UPI |
| 402 | 302 | 2025-09-11 | 4000.00 | Credit Card |
| 403 | 303 | 2025-09-12 | 6000.00 | Cash |
| 404 | 304 | 2025-09-13 | 7000.00 | Debit Card |
| 405 | 305 | 2025-09-14 | 5500.00 | Net Banking |
| 406 | 306 | 2025-09-15 | 4800.00 | UPI |
| 407 | 307 | 2025-09-16 | 6200.00 | Credit Card |
| 408 | 308 | 2025-09-17 | 5300.00 | Cash |
| 409 | 309 | 2025-09-18 | 4900.00 | Debit Card |
| 410 | 310 | 2025-09-19 | 5800.00 | UPI |
| NULL | NULL | NULL | NULL | NULL |

# Implementation of Data Control Language(DCL)

# (CHAPTER-5)

## DCL:

Data Control Language (DCL) is a crucial component of Structured Query Language (SQL) that deals with the rights, permissions, and access control of data within a database.

While Data Definition Language (DDL) defines the structure of the database and Data Manipulation Language (DML) handles data operations, DCL focuses on who can access and manipulate data.

It plays an essential role in maintaining the security, integrity, and privacy of the information stored in a database.

In multi-user database environments, where many users work with the same data, it becomes necessary to control access levels. DCL ensures that only authorized users can perform specific actions such as reading, modifying, or deleting data

### Main Objectives of DCL

To manage and control user access to the database system.

To assign or remove privileges for performing operations on database objects.

To safeguard sensitive or confidential data from unauthorized access.

To ensure data consistency and prevent malicious or accidental data loss.

To maintain accountability by defining different access levels for different users.

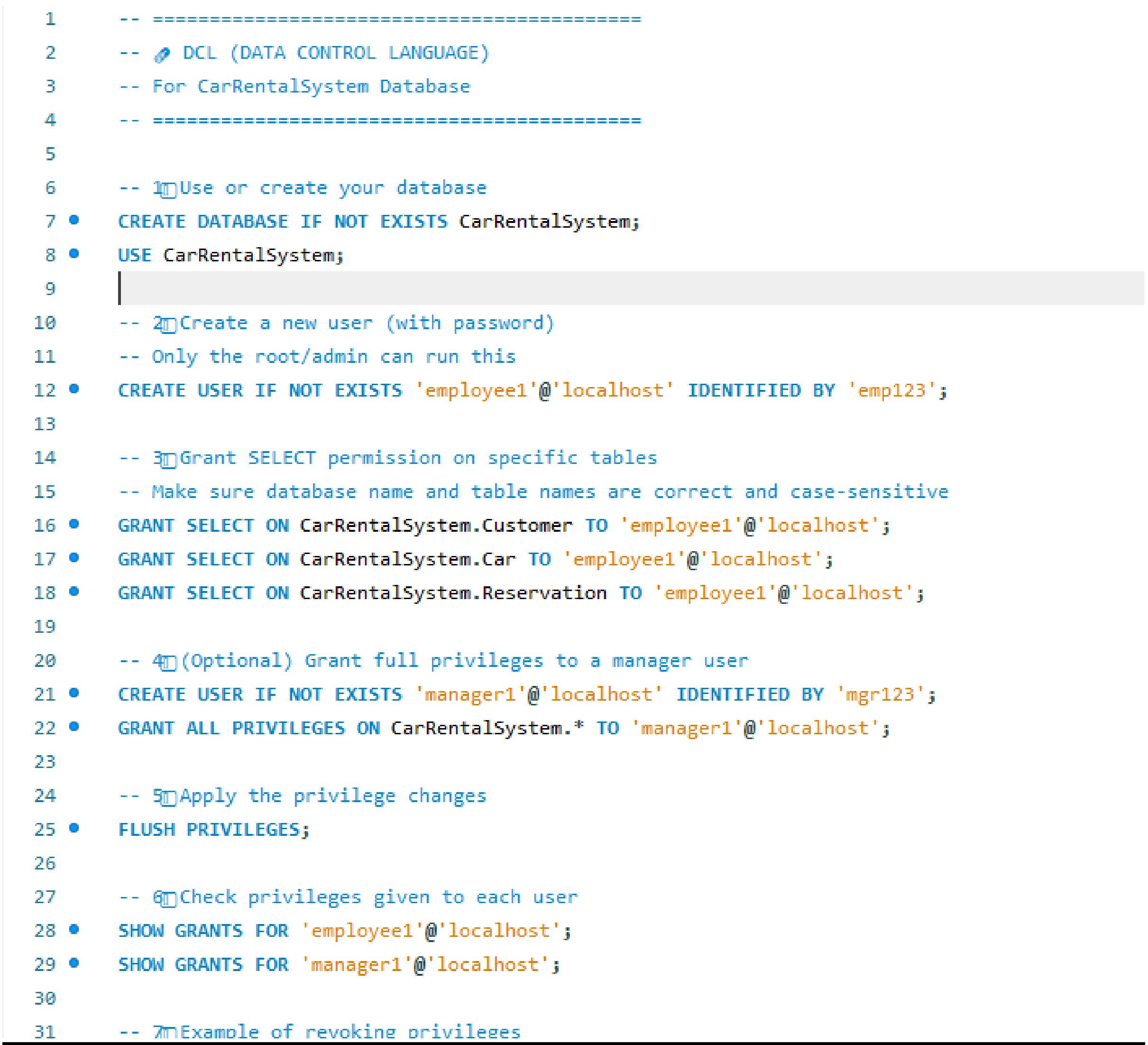
### Main Commands:

CREATE USER

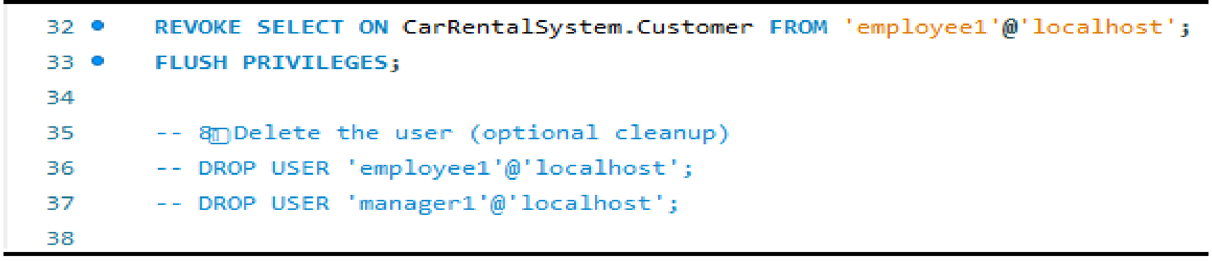
GRANT

REVOKE

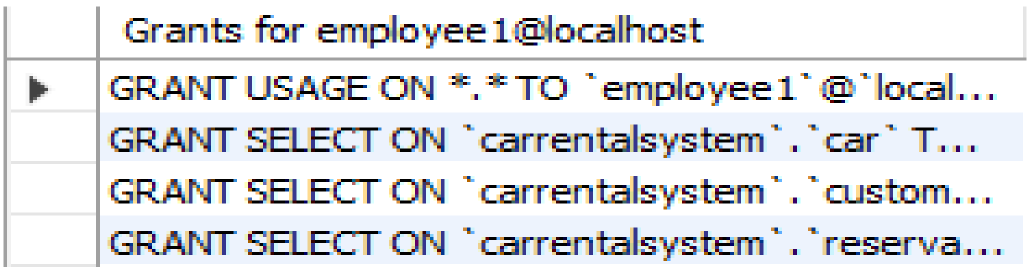
FLUSH PRIVILEGES



```sql
-- ============================================
-- ✏ DCL (DATA CONTROL LANGUAGE)
-- For CarRentalSystem Database
-- ============================================

-- 1️⃣Use or create your database
CREATE DATABASE IF NOT EXISTS CarRentalSystem;
USE CarRentalSystem;

-- 2️⃣Create a new user (with password)
-- Only the root/admin can run this
CREATE USER IF NOT EXISTS 'employee1'@'localhost' IDENTIFIED BY 'emp123';

-- 3️⃣Grant SELECT permission on specific tables
-- Make sure database name and table names are correct and case-sensitive
GRANT SELECT ON CarRentalSystem.Customer TO 'employee1'@'localhost';
GRANT SELECT ON CarRentalSystem.Car TO 'employee1'@'localhost';
GRANT SELECT ON CarRentalSystem.Reservation TO 'employee1'@'localhost';

-- 4️⃣(Optional) Grant full privileges to a manager user
CREATE USER IF NOT EXISTS 'manager1'@'localhost' IDENTIFIED BY 'mgr123';
GRANT ALL PRIVILEGES ON CarRentalSystem.* TO 'manager1'@'localhost';

-- 5️⃣Apply the privilege changes
FLUSH PRIVILEGES;

-- 6️⃣Check privileges given to each user
SHOW GRANTS FOR 'employee1'@'localhost';
SHOW GRANTS FOR 'manager1'@'localhost';

-- 7️⃣Example of revoking privileges
```



```sql
REVOKE SELECT ON CarRentalSystem.Customer FROM 'employee1'@'localhost';
FLUSH PRIVILEGES;

-- 8️⃣Delete the user (optional cleanup)
-- DROP USER 'employee1'@'localhost';
-- DROP USER 'manager1'@'localhost';
```



```
Grants for employee1@localhost
GRANT USAGE ON *.* TO `employee1`@`local...
GRANT SELECT ON `carrentalsystem`.`car` T...
GRANT SELECT ON `carrentalsystem`.`custom...
GRANT SELECT ON `carrentalsystem`.`reserva...
```

# Transaction Control Language(TCL):

## TCL:

Transaction Control Language (TCL) is a subset of SQL that deals with the management of transactions within a database.

A transaction is a logical unit of work that consists of one or more SQL statements executed as a single operation.TCL commands ensure that these operations are performed reliably and that the database remains consistent even in case of errors or system failures.

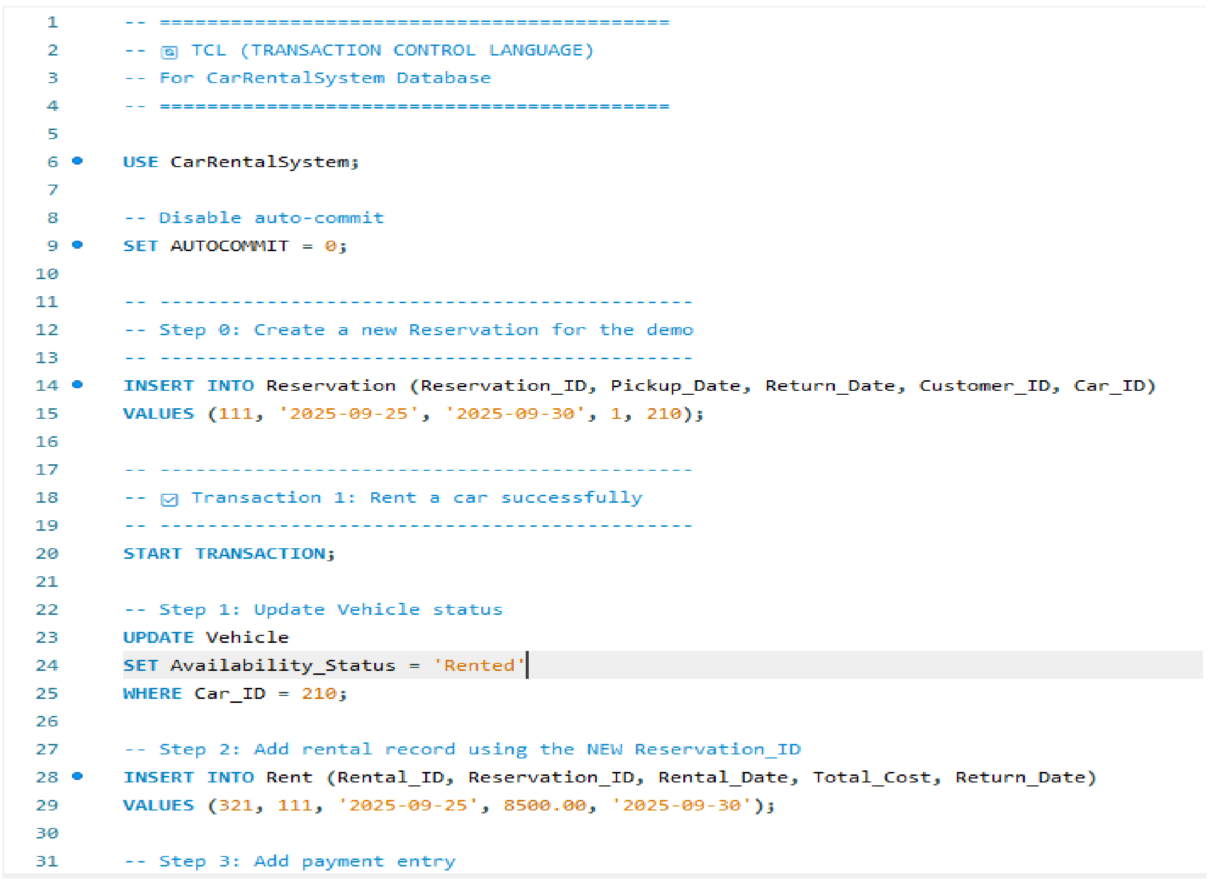
In database systems such as the Online Railway Reservation System, TCL commands are crucial for handling tasks like ticket booking, payment processing, or seat availability updates — where all related actions must either complete successfully together or not occur at all.

### Purpose of TCL

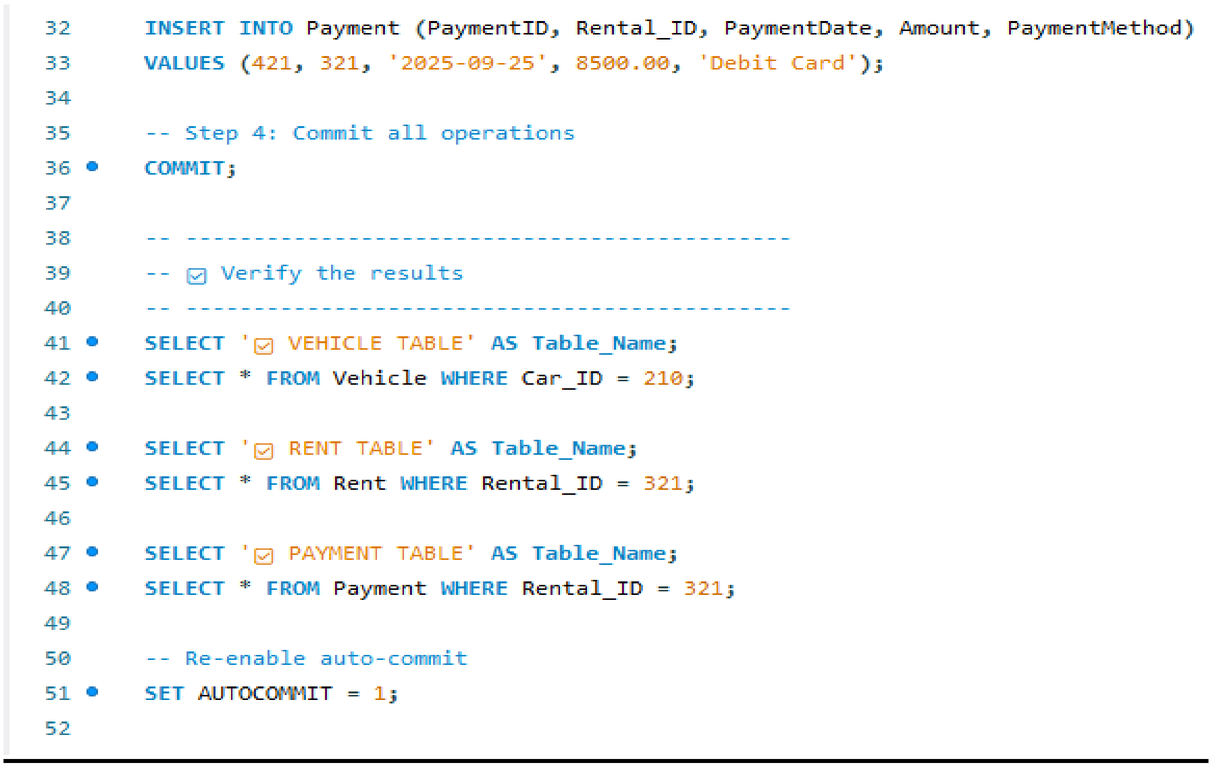
- To maintain **data consistency** and **integrity** during multiple database operations.
- To allow users to **confirm** or **undo** changes made during a transaction.
- To ensure that either **all** parts of a transaction succeed or **none** of them are applied.
- To manage and control changes until the user decides to make them permanent.

## Main Commands:

- COMMIT
- ROLLBACK
- SAVEPOINT
- SET AUTOCOMMIT



```sql
-- ===========================================
-- 🔄 TCL (TRANSACTION CONTROL LANGUAGE)
-- For CarRentalSystem Database
-- ===========================================

USE CarRentalSystem;

-- Disable auto-commit
SET AUTOCOMMIT = 0;

-- ---------------------------------------------
-- Step 0: Create a new Reservation for the demo
-- ---------------------------------------------
INSERT INTO Reservation (Reservation_ID, Pickup_Date, Return_Date, Customer_ID, Car_ID)
VALUES (111, '2025-09-25', '2025-09-30', 1, 210);

-- ---------------------------------------------
-- ✅ Transaction 1: Rent a car successfully
-- ---------------------------------------------
START TRANSACTION;

-- Step 1: Update Vehicle status
UPDATE Vehicle
SET Availability_Status = 'Rented'
WHERE Car_ID = 210;

-- Step 2: Add rental record using the NEW Reservation_ID
INSERT INTO Rent (Rental_ID, Reservation_ID, Rental_Date, Total_Cost, Return_Date)
VALUES (321, 111, '2025-09-25', 8500.00, '2025-09-30');

-- Step 3: Add payment entry
```



```sql
INSERT INTO Payment (PaymentID, Rental_ID, PaymentDate, Amount, PaymentMethod)
VALUES (421, 321, '2025-09-25', 8500.00, 'Debit Card');

-- Step 4: Commit all operations
COMMIT;

-- ---------------------------------------------
-- ✅ Verify the results
-- ---------------------------------------------
SELECT '✅ VEHICLE TABLE' AS Table_Name;
SELECT * FROM Vehicle WHERE Car_ID = 210;

SELECT '✅ RENT TABLE' AS Table_Name;
SELECT * FROM Rent WHERE Rental_ID = 321;

SELECT '✅ PAYMENT TABLE' AS Table_Name;
SELECT * FROM Payment WHERE Rental_ID = 321;

-- Re-enable auto-commit
SET AUTOCOMMIT = 1;
```

# CHAPTER 6:

## FRONT END:

The Car Rental Management System is designed to simplify and automate vehicle rental operations such as car booking, payment processing, customer management, and fleet tracking.
It integrates a user-friendly web interface with a relational SQL database to ensure seamless data flow between the frontend and backend.
Developed using HTML, CSS, and JavaScript for the frontend and Flask (Python) for the backend, the system allows both customers and administrators to perform operations efficiently and securely.

## System Architecture and Components:

The project is modular in design, with separate folders and files to organize frontend and backend operations clearly. Below is an overview of the system's key folders and their purposes:

| FOLDER/FILE | DESCRIPTION |
|---|---|
| `app.py` | The main Flask file that manages routing, connects to the MySQL database, and handles backend logic for car listings, reservations, rentals, and payments |
| `static/` | Contains static files such as CSS, JavaScript, and car images that define the website's appearance and interactivity. |
| `templates/` | Contains HTML files that serve as the user interface for customers and administrators, dynamically rendered using Flask's Jinja templating engine. |
| `database/` | Holds SQL scripts and schema definitions used to create and populate the database tables. |

The **app.py** file serves as the **controller**, managing routes for user login, car search, reservation handling, rental processing, and payment confirmation.
It interacts with the database to retrieve and update records, ensuring real-time synchronization between user actions and stored data.

The **static/** folder includes CSS files that maintain a consistent design and layout across all pages, and JavaScript files that enable dynamic features such as date validation, booking confirmation, and live updates without page reloads.

The **templates/** folder stores all the HTML pages that users interact with. Each template uses **Jinja2 syntax** to display data from the backend — for example, showing available cars, customer bookings, and payment summaries dynamically.

## Database Design:

The backend uses a **relational SQL database** that organizes retail data into multiple interconnected tables. Each table stores specific information essential for the system's functionality.

| TABLE | PURPOSE |
|---|---|
| **Customer** | Stores customer details such as name, contact, address, and driver's license number. |
| **Car** | Holds information about cars, including model, type, registration number, and availability. |
| **Reservation** | Tracks car booking details, including pickup and return dates. |
| **Rental** | Records confirmed rentals, including cost, customer, car, and employee details. |
| **payment** | Manages all financial transactions related to rentals. |
| **Employee** | Contains staff details for those handling rentals and customers |
| **Branch** | Stores information about rental branch locations. |
| **Insurance** | Tracks insurance policies associated with each car. |
| **Maintenance** | Records servicing and maintenance details of each vehicle. |
| **Feedback** | Captures customer feedback and ratings. |

This structure ensures data consistency, eliminates redundancy, and enables smooth transactions between users, products, and payments.

## HTML Template Pages Overview:

The templates in the **templates/** folder form the core of the user interface. They allow users to interact with the system through forms, dashboards, and product pages. Below is an overview of each HTML page and its functionality:

| TEMPLATE | DESCRIPTION |
|---|---|
| **index.html** | Serves as the homepage displaying featured products and navigation links to other sections. |
| **login.html** | Allows users to log in or register. Validates credentials using the backend database. |

| TEMPLATE | DESCRIPTION |
|---|---|
| **register_list.html** | Lets new users create accounts and stores their details in the Customer table. |
| **car.html** | Displays all available cars with filters for car type, price, and availability. |
| **rental_details.html** | Shows current and past rentals linked to the logged-in user. |
| **payment.html** | Processes online payments and records them in the Payment table. |
| **admin_dashboard.html** | Provides the admin interface for managing cars, customers, and payments. |

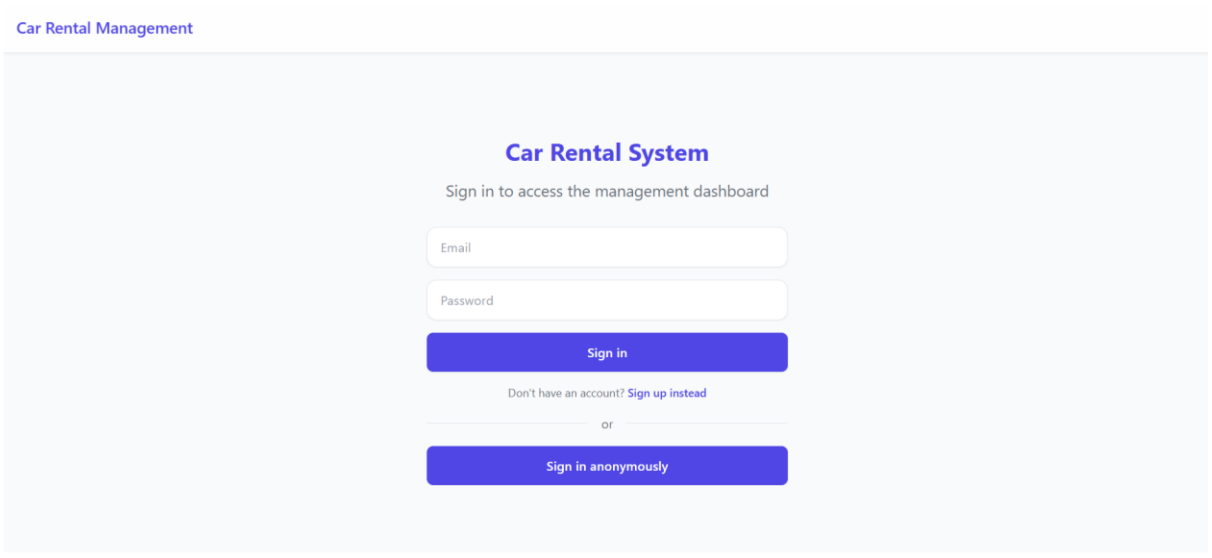
## Application Flow:

1. **User Authentication:**
   - Users access the system via login.html and verify credentials stored in the database.
   - Customers are directed to the booking dashboard, while admins access the management panel.
2. **Car Browsing and Reservation:**
   - Customers browse available cars on car_list.html.
   - They can select a car and proceed to book it using reservation.html.
3. **Rental Confirmation and Payment:**
   - Upon confirmation, reservation data is stored in the **Reservation** table.
   - The user completes payment on payment.html, and details are added to the **Payment** table.
4. **Admin and Management Features:**
   - Admins can view, update, or delete records via admin_dashboard.html.
   - They can manage fleet availability, check rentals, and monitor payments.

## SIGN IN :

```
"use client";
import { useAuthActions } from "@convex-dev/auth/react";
import { useState } from "react";
import { toast } from "sonner";

export function SignInForm() {
  const { signIn } = useAuthActions();
  const [flow, setFlow] = useState<"signIn" | "signUp">("signIn");
  const [submitting, setSubmitting] = useState(false);

  return (
<div className="w-full">
  <form
    className="flex flex-col gap-form-field"
    onSubmit={(e) => {
      e.preventDefault();
      setSubmitting(true);
      const formData = new FormData(e.target as HTMLFormElement);
      formData.set("flow", flow);
      void signIn("password", formData).catch((error) => {
        let toastTitle = "";
        if (error.message.includes("Invalid password")) {
          toastTitle = "Invalid password. Please try again.";
        } else {
          toastTitle =
            flow === "signIn"
              ? "Could not sign in, did you mean to sign up?"
              : "Could not sign up, did you mean to sign in?";
        }
        toast.error(toastTitle);
        setSubmitting(false);
      });
    }}
  >
    <input
      className="auth-input-field"
      type="email"
      name="email"
      placeholder="Email"
      required
    />
    <input
      className="auth-input-field"
      type="password"
      name="password"
      placeholder="Password"
      required
    />
    <button className="auth-button" type="submit" disabled={submitting}>
      {flow === "signIn" ? "Sign in" : "Sign up"}
    </button>
    <div className="text-center text-sm text-secondary">
      <span>
        {flow === "signIn"
          ? "Don't have an account? "
          : "Already have an account? "}
      </span>
      <button
        type="button"
        className="text-primary hover:text-primary-hover hover:underline font-
medium cursor-pointer"
```

```
          onClick={() => setFlow(flow === "signIn" ? "signUp" : "signIn")}
        >
          {flow === "signIn" ? "Sign up instead" : "Sign in instead"}
        </button>
      </div>
    </form>
    <div className="flex items-center justify-center my-3">
      <hr className="my-4 grow border-gray-200" />
      <span className="mx-4 text-secondary">or</span>
      <hr className="my-4 grow border-gray-200" />
    </div>
    <button className="auth-button" onClick={() => void signIn("anonymous")}>
      Sign in anonymously
    </button>
  </div>
 );
}
```



Car Rental Management
Car Rental System
Sign in to access the management dashboard
Email
Password
Sign in
Don't have an account? Sign up instead
or
Sign in anonymously

**app.tsx:**

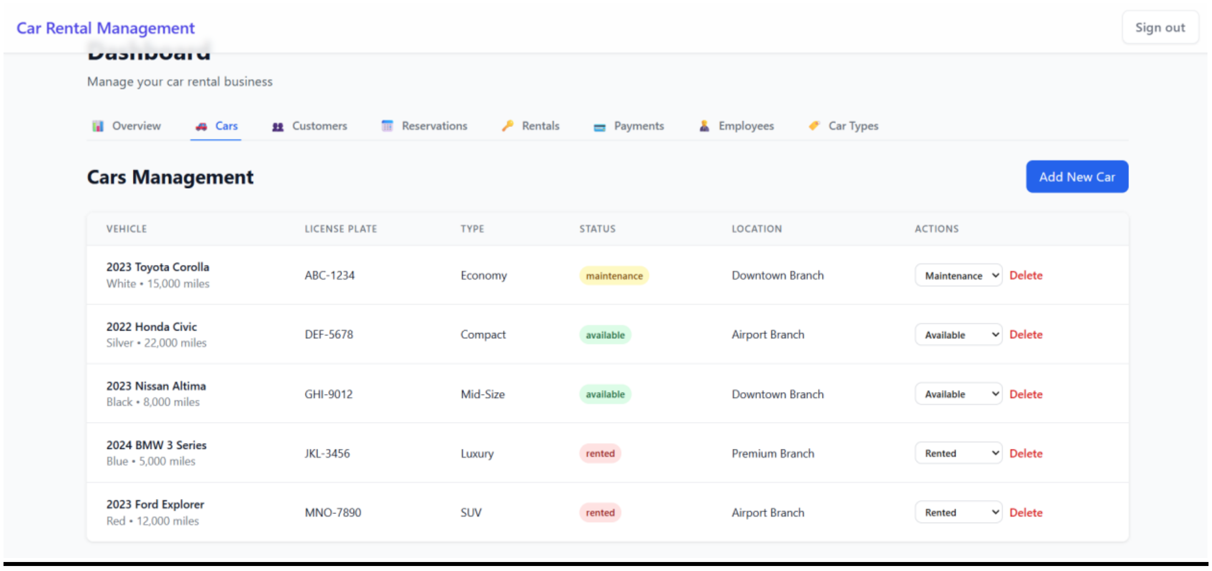
```
import { Authenticated, Unauthenticated, useQuery } from "convex/react";
import { api } from "../convex/_generated/api";
import { SignInForm } from "./SignInForm";
import { SignOutButton } from "./SignOutButton";
import { Toaster } from "sonner";
import { CarCatalog } from "./components/CarCatalog";
import { ReservationList } from "./components/ReservationList";
import { RentalHistory } from "./components/RentalHistory";
import { useState } from "react";
```

```
export default function App() {
  const [currentView, setCurrentView] = useState<"catalog" | "reservations" |
"history">("catalog");
  const reservations = useQuery(api.reservations.list) || [];
  const reservationCount = reservations.length;

  return (
    <div className="min-h-screen flex flex-col bg-gray-50">
      <header className="sticky top-0 bg-white border-b shadow-sm">
        <div className="max-w-7xl mx-auto px-4 h-16 flex justify-between items-
center">
          <div className="flex items-center gap-8">
            <h1 className="text-2xl font-bold text-gray-900">CarRentalHub</h1>
            <Authenticated>
              <nav className="flex gap-6">
                <button
                  onClick={() => setCurrentView("catalog")}
                  className={`px-3 py-2 rounded-md text-sm font-medium ${
                    currentView === "catalog" ? "bg-blue-100 text-blue-700" : "text-gray-
600 hover:text-gray-900"
                  }`}
                >
                  Cars
                </button>
                <button
                  onClick={() => setCurrentView("reservations")}
                  className={`px-3 py-2 rounded-md text-sm font-medium relative ${
                    currentView === "reservations" ? "bg-blue-100 text-blue-700" : "text-
gray-600 hover:text-gray-900"
                  }`}
                >
                  My Reservations
                  {reservationCount > 0 && (
                    <span className="absolute -top-1 -right-1 bg-red-500 text-white text-xs
rounded-full h-5 w-5 flex items-center justify-center">
                      {reservationCount}
                    </span>
                  )}
                </button>
                <button
                  onClick={() => setCurrentView("history")}
                  className={`px-3 py-2 rounded-md text-sm font-medium ${
                    currentView === "history" ? "bg-blue-100 text-blue-700" : "text-gray-
600 hover:text-gray-900"
                  }`}
                >
                  Rental History
                </button>
              </nav>
            </Authenticated>
          </div>
          <SignOutButton />
        </div>
      </header>

      <main className="flex-1">
        <Content currentView={currentView} />
      </main>

      <Toaster />
    </div>
  );
}

function Content({ currentView }: { currentView: "catalog" | "reservations" |
"history" }) {
  const user = useQuery(api.auth.loggedInUser);

  if (user === undefined) {
    return <div className="flex justify-center items-center min-h-96 animate-spin
border-b-2 border-blue-600 rounded-full h-8 w-8"></div>;
  }

  return (
    <div className="max-w-7xl mx-auto px-4 py-8">
      <Unauthenticated>
        <div className="max-w-md mx-auto text-center">
          <h2 className="text-3xl font-bold text-gray-900 mb-4">Welcome to
CarRentalHub</h2>
          <p className="text-gray-600 mb-6">Sign in to start booking your ride</p>
          <SignInForm />
        </div>
      </Unauthenticated>

      <Authenticated>
        {currentView === "catalog" && <CarCatalog />}
        {currentView === "reservations" && <ReservationList />}
        {currentView === "history" && <RentalHistory />}
      </Authenticated>
    </div>
  );
}
```

## **CarCatalog.tsx**

```
import { useQuery, useMutation } from "convex/react";
import { api } from "../../convex/_generated/api";
import { toast } from "sonner";
import { Id } from "../../convex/_generated/dataModel";
import { useState } from "react";

export function CarCatalog() {
  const [search, setSearch] = useState("");
  const cars = useQuery(api.cars.list, { search }) || [];
  const reserveCar = useMutation(api.reservations.create);

  const handleReserve = async (carId: Id<"cars">) => {
    try {
      await reserveCar({ carId });
      toast.success("Car reserved successfully!");
    } catch {
      toast.error("Reservation failed");
    }
  };

  return (
    <div className="space-y-8">
      <div className="flex gap-4">
        <input
          type="text"
          placeholder="Search cars..."
          value={search}
          onChange={(e) => setSearch(e.target.value)}
          className="flex-1 px-4 py-2 border border-gray-300 rounded-lg"
        />
      </div>

      {cars.length === 0 ? (
        <p className="text-center text-gray-600">No cars found.</p>
      ) : (
        <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-6">
          {cars.map((car) => (
            <div key={car._id} className="bg-white rounded-lg shadow-md p-4
hover:shadow-lg">
              <img
                src={car.imageUrl || "/car-placeholder.png"}
                alt={car.model}
                className="w-full h-48 object-cover rounded-md mb-3"
              />
              <h3 className="font-bold text-lg">{car.make} {car.model}</h3>
              <p className="text-gray-600">Year: {car.year}</p>
              <p className="font-semibold text-blue-600 mb-
3">₹{car.dailyRate}/day</p>
              <button
                onClick={() => handleReserve(car._id)}
                className="w-full bg-blue-600 text-white py-2 rounded-lg hover:bg-blue-
700"
              >
```

```
                Reserve Now
              </button>
            </div>
          ))}
        </div>
      )}
    </div>
  );
}
```



Manage your car rental business

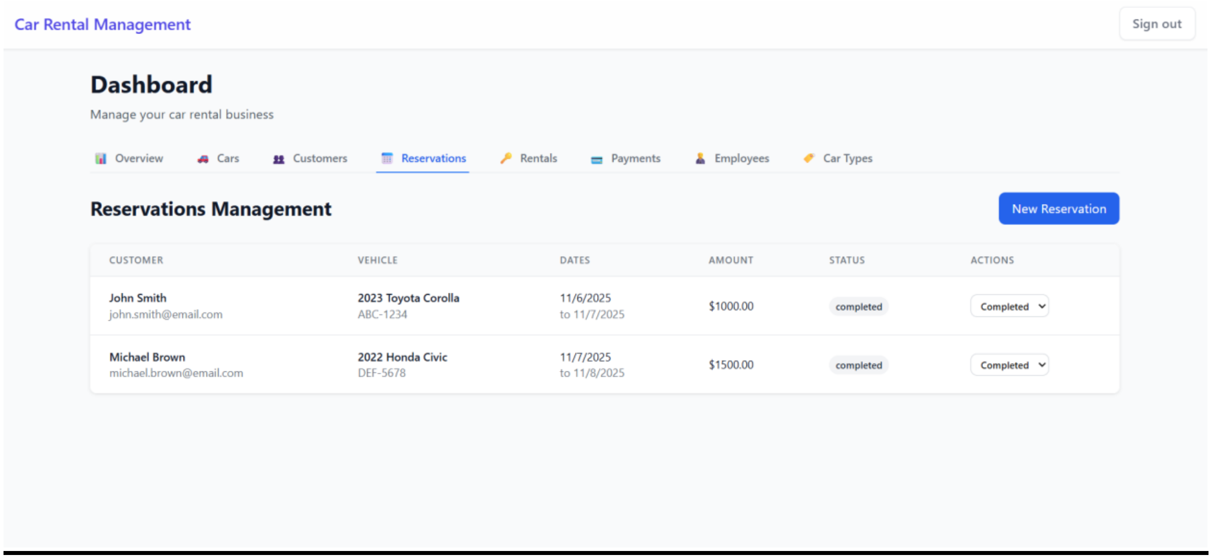
Cars Management

| VEHICLE | LICENSE PLATE | TYPE | STATUS | LOCATION | ACTIONS |
|---|---|---|---|---|---|
| 2023 Toyota Corolla<br>White • 15,000 miles | ABC-1234 | Economy | maintenance | Downtown Branch | Maintenance Delete |
| 2022 Honda Civic<br>Silver • 22,000 miles | DEF-5678 | Compact | available | Airport Branch | Available Delete |
| 2023 Nissan Altima<br>Black • 8,000 miles | GHI-9012 | Mid-Size | available | Downtown Branch | Available Delete |
| 2024 BMW 3 Series<br>Blue • 5,000 miles | JKL-3456 | Luxury | rented | Premium Branch | Rented Delete |
| 2023 Ford Explorer<br>Red • 12,000 miles | MNO-7890 | SUV | rented | Airport Branch | Rented Delete |

## ReservationList.tsx

```
import { useQuery, useMutation } from "convex/react";
import { api } from "../../convex/_generated/api";
import { toast } from "sonner";

export function ReservationList() {
  const reservations = useQuery(api.reservations.list) || [];
  const cancelReservation = useMutation(api.reservations.cancel);
  const confirmRental = useMutation(api.rentals.create);

  if (reservations.length === 0)
    return <p className="text-center py-12 text-gray-600">No active reservations
found.</p>;

  return (
    <div className="max-w-4xl mx-auto">
      <h1 className="text-3xl font-bold mb-6">My Reservations</h1>
      <div className="space-y-4">
        {reservations.map((r) => (
          <div key={r._id} className="bg-white shadow rounded-lg p-6 flex justify-
between items-center">
            <div>
```

```
             <h3 className="font-semibold">{r.car.make} {r.car.model}</h3>
             <p className="text-sm text-gray-600">
              From {r.pickupDate} to {r.returnDate}
             </p>
            </div>
            <div className="flex gap-3">
             <button
              onClick={() => confirmRental({ reservationId: r._id })}
              className="bg-green-600 text-white px-4 py-2 rounded-lg hover:bg-green-
      700"
             >
              Confirm Rental
             </button>
             <button
              onClick={() => cancelReservation({ id: r._id })}
              className="bg-red-500 text-white px-4 py-2 rounded-lg hover:bg-red-600"
             >
              Cancel
             </button>
            </div>
           </div>
          ))}
         </div>
        </div>
       );
}
```


Car Rental Management

Sign out

## Dashboard

Manage your car rental business

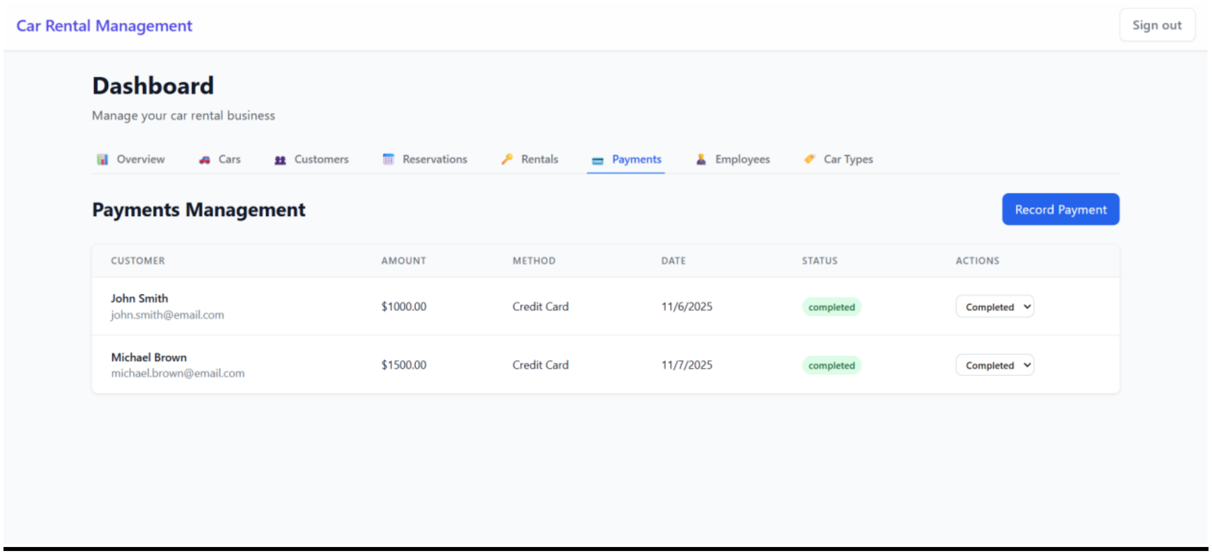
### Reservations Management

| CUSTOMER | VEHICLE | DATES | AMOUNT | STATUS | ACTIONS |
|---|---|---|---|---|---|
| John Smith<br>john.smith@email.com | 2023 Toyota Corolla<br>ABC-1234 | 11/6/2025<br>to 11/7/2025 | $1000.00 | completed | Completed |
| Michael Brown<br>michael.brown@email.com | 2022 Honda Civic<br>DEF-5678 | 11/7/2025<br>to 11/8/2025 | $1500.00 | completed | Completed |

## RentalHistory.tsx:

```
      import { useQuery } from "convex/react";
      import { api } from "../../convex/_generated/api";
```

```
export function RentalHistory() {
  const rentals = useQuery(api.rentals.list) || [];

  if (rentals.length === 0)
    return (
      <div className="text-center py-12">
        <div className="text-6xl mb-4">🚗</div>
        <h2 className="text-2xl font-bold">No completed rentals yet</h2>
        <p className="text-gray-600">Your rental history will appear here.</p>
      </div>
    );

  return (
    <div className="max-w-4xl mx-auto space-y-6">
      <h1 className="text-3xl font-bold text-gray-900 mb-6">Rental History</h1>
      {rentals.map((r) => (
        <div key={r._id} className="bg-white rounded-lg shadow-md p-6">
          <div className="flex justify-between">
            <div>
              <h3 className="font-semibold text-lg">
                {r.car.make} {r.car.model}
              </h3>
              <p className="text-sm text-gray-600">
                {r.rentalDate} – {r.returnDate}
              </p>
            </div>
            <div className="text-right">
              <p className="text-lg font-bold text-gray-900">₹{r.totalCost}</p>
              <span
                className={`px-3 py-1 rounded-full text-sm ${
                  r.status === "returned" ? "bg-green-100 text-green-700" : "bg-yellow-100
text-yellow-800"
                }`}
              >
                {r.status}
              </span>
            </div>
          </div>
        </div>
      ))}
    </div>
  );
}
```



| CUSTOMER | AMOUNT | METHOD | DATE | STATUS | ACTIONS |
|---|---|---|---|---|---|
| John Smith john.smith@email.com | $1000.00 | Credit Card | 11/6/2025 | completed | Completed |
| Michael Brown michael.brown@email.com | $1500.00 | Credit Card | 11/7/2025 | completed | Completed |

## SIGN OUT:

```
"use client";
import { useAuthActions } from "@convex-dev/auth/react";
import { useConvexAuth } from "convex/react";

export function SignOutButton() {
  const { isAuthenticated } = useConvexAuth();
  const { signOut } = useAuthActions();

  if (!isAuthenticated) {
    return null;
  }

  return (
    <button
      className="px-4 py-2 rounded bg-white text-secondary border border-gray-200
font-semibold hover:bg-gray-50 hover:text-secondary-hover transition-colors
shadow-sm hover:shadow"
      onClick={() => void signOut()}
    >
      Sign out
    </button>
  );
}
```

## Database Integration:

The **Rental Car Management System** integrates seamlessly with a **MySQL database**, ensuring efficient storage, retrieval, and management of rental, vehicle, and customer information. The backend, built using **Flask**, provides smooth interaction between the frontend and database, allowing real-time data updates across all modules such as car availability, booking status, and payment details. This integration improves scalability, reliability, and data consistency, making the system suitable for managing multiple concurrent car rentals securely and efficiently.

## Functionality Highlights

- **User Registration and Authentication:**
  Customers and admins can securely register and log in. Role-based access ensures that only authorized users can manage cars, rentals, and transactions.
- **Car Management:**
  Admins can add, update, or delete car details such as model, type, year, price per day, and availability. The system automatically updates availability when a car is rented or returned.
- **Booking and Rental Management:**
  Customers can view available cars, check pricing, and make reservations. The system prevents double booking and updates the status of vehicles in real time.
- **Payment Processing:**
  After booking, customers can make secure payments. All payment information is recorded in the database, linked to the respective rental and customer.
- **Customer Dashboard:**
  Users can view their booking history, payment receipts, and active rentals in one place.
- **Admin Dashboard:**
  The admin can monitor all rentals, view customer activity, manage fleet information, and generate performance reports.
- **Return and Billing Management:**
  The system calculates rental costs automatically based on duration, car type, and late-return penalties.

## Discussion:

The **Rental Car Management System** fulfills its goal of automating and digitizing the car rental process. It enhances customer convenience and operational efficiency by allowing seamless booking, payment, and return handling.
The use of Flask APIs ensures smooth communication between the frontend and MySQL database, while proper normalization and relationships maintain data accuracy.
Overall, the system provides an easy-to-use and reliable digital platform for managing car rental operations with minimal manual intervention.

## Security Measures:

- **Secure Login and Role Management:** Passwords are encrypted, and access permissions are assigned based on user roles (admin or customer).
- **Data Encryption:** Sensitive data such as passwords and payment details are securely encrypted.
- **Secure Communication:** HTTPS and SSL protocols ensure data security during transactions.
- **Validation and SQL Injection Prevention:** Input validation and prepared statements are used to protect against malicious queries.
- **Data Privacy Compliance:** The system aligns with data protection standards to safeguard user and payment information.

## Data Integrity:

- **Transaction Management:** Atomic transactions ensure data consistency during car booking and payment.
- **Input Validation:** All customer inputs are verified to prevent errors or invalid data entries.
- **Logging and Auditing:** Each booking, cancellation, and update is recorded for accountability.
- **Backup and Recovery:** Automated database backups protect against data loss due to system failures.

## User Interface Design:

- **Simple Navigation:** The interface allows easy access to car listings, bookings, payments, and dashboards.
- **Responsive Design:** Works across all devices—desktop, tablet, and mobile.
- **Clear Visuals:** Displays car images, pricing, and availability for quick decision-making.
- **Accessibility Features:** Uses readable fonts and color contrast for better usability.

## Challenges and Limitations:

- **Real-Time Vehicle Tracking:** Integration with GPS systems for live tracking is not yet implemented.
- **Scalability:** As the customer base and car fleet grow, system optimization may be required.
- **Payment Gateway Integration:** Secure and reliable payment API integration can be improved further.

- **Concurrent Bookings:** Managing simultaneous booking requests may require database-level locking mechanisms.

## Future Enhancements:

- **Mobile App Development:** Building Android and iOS versions for easier access.
- **AI-Based Car Recommendation:** Suggest cars based on user preferences and booking history.
- **Integration with GPS Tracking:** Real-time location and monitoring of rented vehicles.
- **Digital Contracts and E-Signatures:** Automate rental agreements digitally.
- **Analytics Dashboard:** Introduce predictive analytics for demand forecasting and fleet optimization.

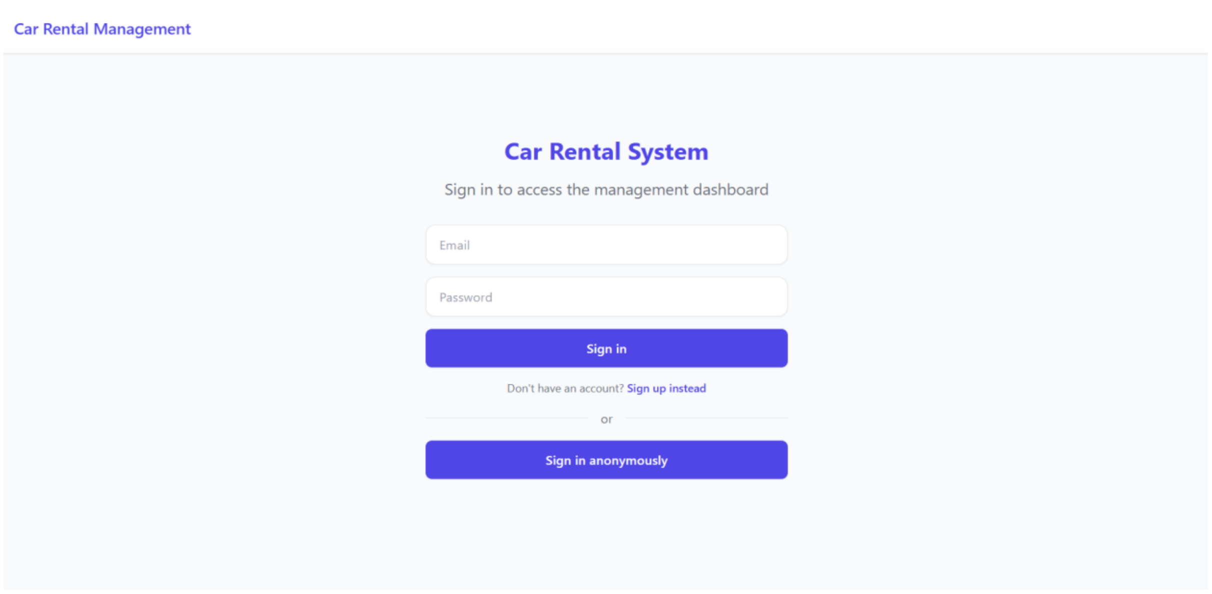
## Conclusion:

In conclusion, the **Rental Car Management System** provides a secure, efficient, and user-friendly platform for automating the car rental process. It minimizes manual work, ensures transparency, and enhances user satisfaction through real-time updates and secure transactions.
With future enhancements like GPS tracking, AI integration, and mobile app support, the system can evolve into a comprehensive digital solution for modern vehicle rental businesses.

sector.

## FRONT END PICTURES:

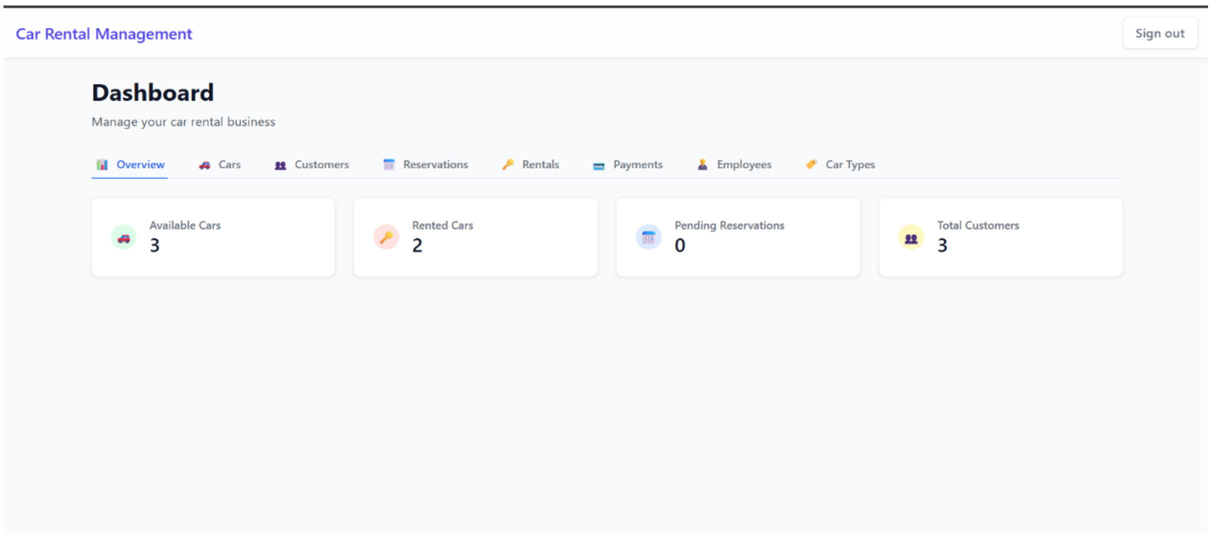
SIGN IN/SIGN UP:


Car Rental Management

Car Rental System

Sign in to access the management dashboard

**DASHBOARD**:


Car Rental Management

Dashboard

Manage your car rental business

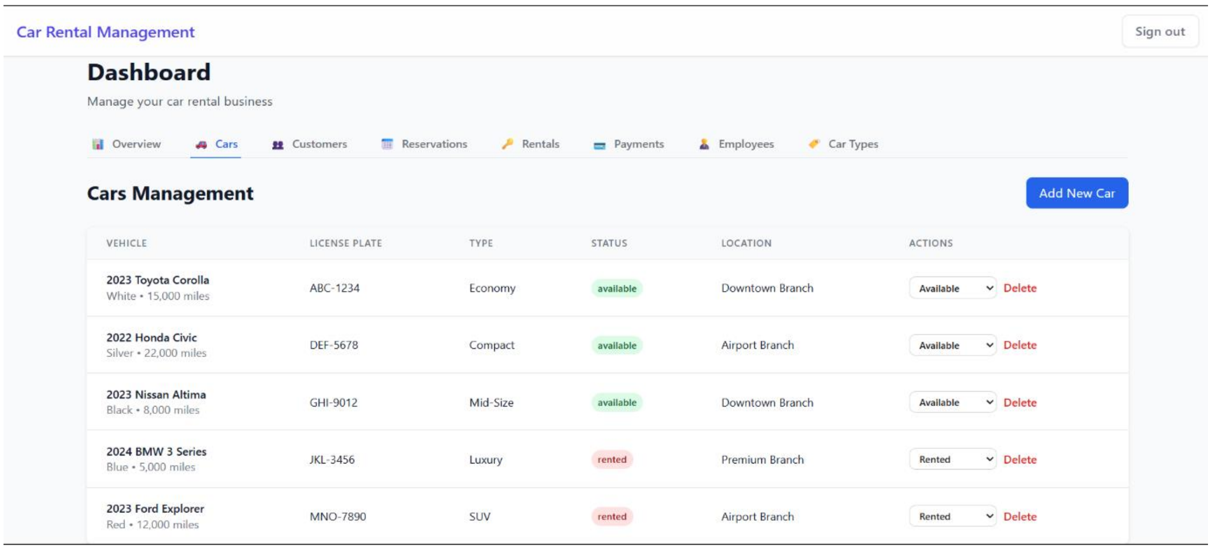
Available Cars 3

Rented Cars 2

Pending Reservations 0

Total Customers 3

## CARS:



Car Rental Management
Sign out

Dashboard

Manage your car rental business

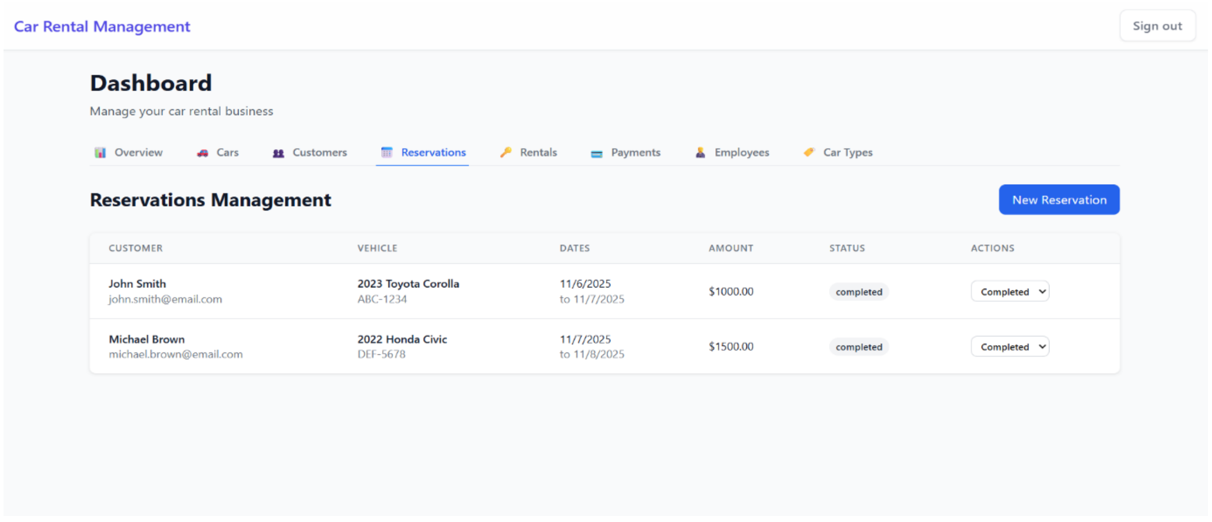
Overview | Cars | Customers | Reservations | Rentals | Payments | Employees | Car Types

Cars Management

Add New Car

| VEHICLE | LICENSE PLATE | TYPE | STATUS | LOCATION | ACTIONS |
|---|---|---|---|---|---|
| 2023 Toyota Corolla<br>White • 15,000 miles | ABC-1234 | Economy | available | Downtown Branch | Available Delete |
| 2022 Honda Civic<br>Silver • 22,000 miles | DEF-5678 | Compact | available | Airport Branch | Available Delete |
| 2023 Nissan Altima<br>Black • 8,000 miles | GHI-9012 | Mid-Size | available | Downtown Branch | Available Delete |
| 2024 BMW 3 Series<br>Blue • 5,000 miles | JKL-3456 | Luxury | rented | Premium Branch | Rented Delete |
| 2023 Ford Explorer<br>Red • 12,000 miles | MNO-7890 | SUV | rented | Airport Branch | Rented Delete |

## Reservations:



Car Rental Management
Sign out

Dashboard

Manage your car rental business

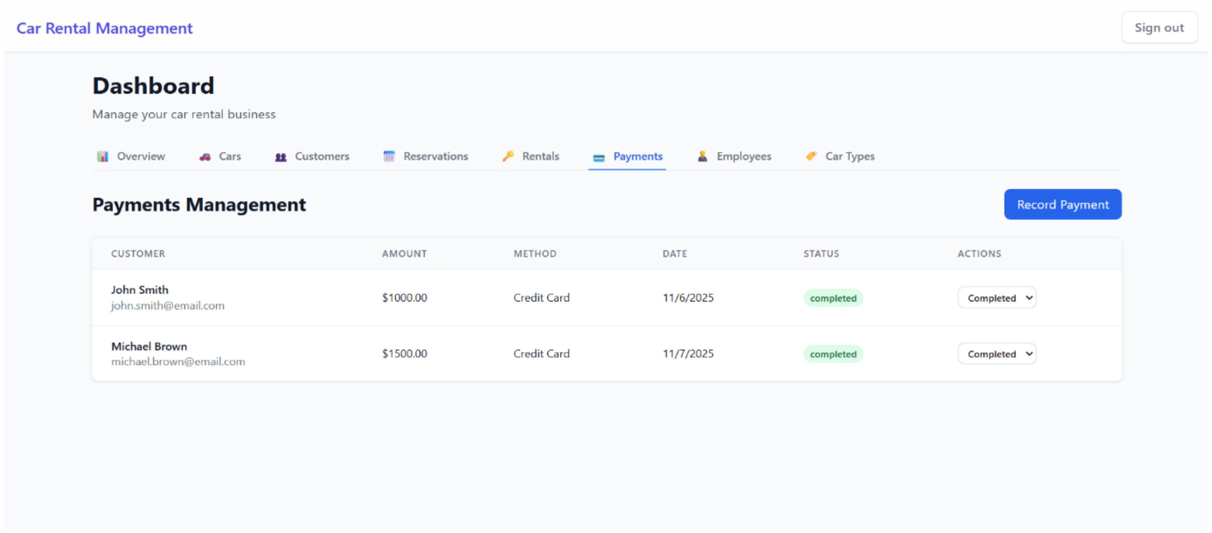
Overview | Cars | Customers | Reservations | Rentals | Payments | Employees | Car Types

Reservations Management

New Reservation

| CUSTOMER | VEHICLE | DATES | AMOUNT | STATUS | ACTIONS |
|---|---|---|---|---|---|
| John Smith<br>john.smith@email.com | 2023 Toyota Corolla<br>ABC-1234 | 11/6/2025<br>to 11/7/2025 | $1000.00 | completed | Completed |
| Michael Brown<br>michael.brown@email.com | 2022 Honda Civic<br>DEF-5678 | 11/7/2025<br>to 11/8/2025 | $1500.00 | completed | Completed |

## Payments:



Car Rental Management

Sign out

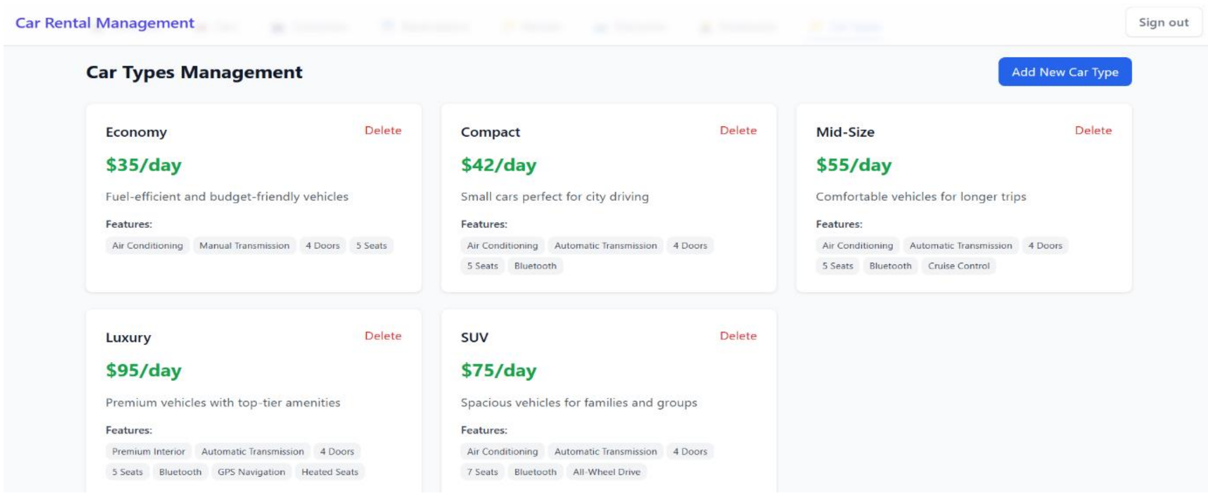
Dashboard

Manage your car rental business

Payments Management

| CUSTOMER | AMOUNT | METHOD | DATE | STATUS | ACTIONS |
|---|---|---|---|---|---|
| John Smith john.smith@email.com | $1000.00 | Credit Card | 11/6/2025 | completed | Completed |
| Michael Brown michael.brown@email.com | $1500.00 | Credit Card | 11/7/2025 | completed | Completed |

## Car Types:



Car Rental Management

Sign out

Car Types Management

Economy — Delete
$35/day
Fuel-efficient and budget-friendly vehicles
Features: Air Conditioning, Manual Transmission, 4 Doors, 5 Seats

Compact — Delete
$42/day
Small cars perfect for city driving
Features: Air Conditioning, Automatic Transmission, 4 Doors, 5 Seats, Bluetooth

Mid-Size — Delete
$55/day
Comfortable vehicles for longer trips
Features: Air Conditioning, Automatic Transmission, 4 Doors, 5 Seats, Bluetooth, Cruise Control

Luxury — Delete
$95/day
Premium vehicles with top-tier amenities
Features: Premium Interior, Automatic Transmission, 4 Doors, 5 Seats, Bluetooth, GPS Navigation, Heated Seats

SUV — Delete
$75/day
Spacious vehicles for families and groups
Features: Air Conditioning, Automatic Transmission, 4 Doors, 7 Seats, Bluetooth, All-Wheel Drive