



Car Dealership System

AbdelRahman Ammar 202102787

Abdelrhman Fouda 202101892

Hassan Ghataty 202103095

Introduction

- **Project Problem Statement**

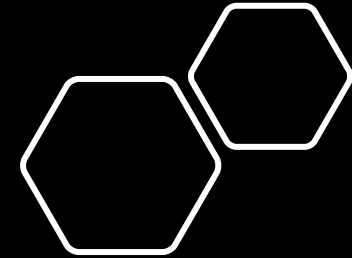
- The current car dealership operations lack an efficient, digital system to manage customer interactions, inventory tracking, and user role-specific functionalities. Manual processes are error-prone, time-consuming, and limit scalability.

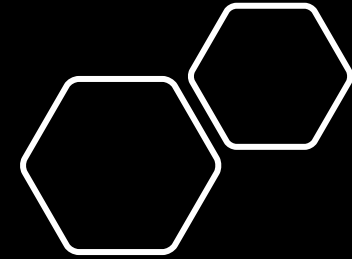
- **Summary**

- The Car Dealership System is a web-based solution designed to streamline operations for car dealerships. It provides:
- Role-based access for Admin, Salesperson, and Customer.
- Inventory management with real-time updates.
- Test drive scheduling, car bookings, and inquiry tracking.
- Comprehensive sales reporting and database-driven functionality.

Functional Requirements

REQ_ID	Requirement Description	Category
REQ_001	The system must allow adding, updating, and deleting car details (e.g., make, model, year, price).	M
REQ_002	The system must manage and display car inventory (available, sold, under maintenance).	M
REQ_003	The system must allow customers to schedule test drives for available cars.	M
REQ_004	The system must support customer profile creation and management.	M
REQ_005	The system must record and track customer inquiries	M
REQ_006	The system must enable filter by price range.	M
REQ_007	The system must allow vehicle maintenance scheduling and tracking.	M
REQ_008	The system must include a dashboard summarizing total sales, inventory, and bookings.	M
REQ_009	The system must include user roles (e.g., Admin, Salesperson, Customer) with proper access levels.	M
REQ_010	The system should provide sales reports	S
REQ_011	The system should allow exporting reports	S
REQ_012	The system could provide an AI-based recommendation system for similar car models.	C
REQ_013	The system could send automated notifications for scheduled test drives or offers.	C
REQ_014	The system could provide real-time updates on car availability.	C
REQ_015	The system could integrate a chatbot for customer inquiries.	C
REQ_016	The system will handle car insurance services.	W





Non-Functional Requirements

REQ_ID	Requirement Description	Category
REQ_017	The system must be compatible with all major browsers (e.g., Chrome, Firefox, Edge).	Usability
REQ_018	The system must have a user-friendly and intuitive interface for non-technical users.	Usability
REQ_019	The system should respond within 2 seconds for user actions (e.g., filter).	Performance
REQ_020	The system should handle up to 100 concurrent users without performance degradation.	Scalability



Use Case 1

Use Case 1: Signing Up

Use Case ID: UC_001

Use Case Name: Customer Signup

Actors:

- Customer

Preconditions:

- The customer is on the login page.
- The system displays a link to the signup page.

Basic Flow:

1. The customer clicks the "Sign up here" link on the login page.
2. The system displays the signup form.
3. The customer enters their Name, Email, Password, and confirms their Role as "Customer."
4. The customer clicks the "Sign Up" button.
5. The system validates the input (e.g., checks if the email is unique and the password is valid).
6. If validation passes, the system creates a new customer account and redirects the user to the login page.
7. The system displays a message: "Signup successful. Please log in."

Postconditions:

- A new customer record is created in the database.
- The customer can log in using their newly created account.

Use Case 2

Use Case 2: Booking a Car

Use Case ID: UC_002

Use Case Name: Book a Car

Actors:

- Customer

Preconditions:

- The customer must be logged in.
- At least one car must be available for booking.

Basic Flow:

1. The customer logs into the system and navigates to the "Book a Car" page.
2. The system displays a list of available cars, including details such as Make, Model, and Price.
3. The customer selects a car they want to book.
4. The customer clicks the "Book" button.
5. The system records the booking with a status of "Pending" and displays a confirmation message: "Car booked successfully."

Postconditions:

- A booking record is created in the database with a Pending status.
- The car remains in the inventory until the booking is approved.

Use Case 3

Use Case 3: Approving a Pending Booking

Use Case ID: UC_003

Use Case Name: Approve a Pending Booking

Actors:

- Salesperson

Preconditions:

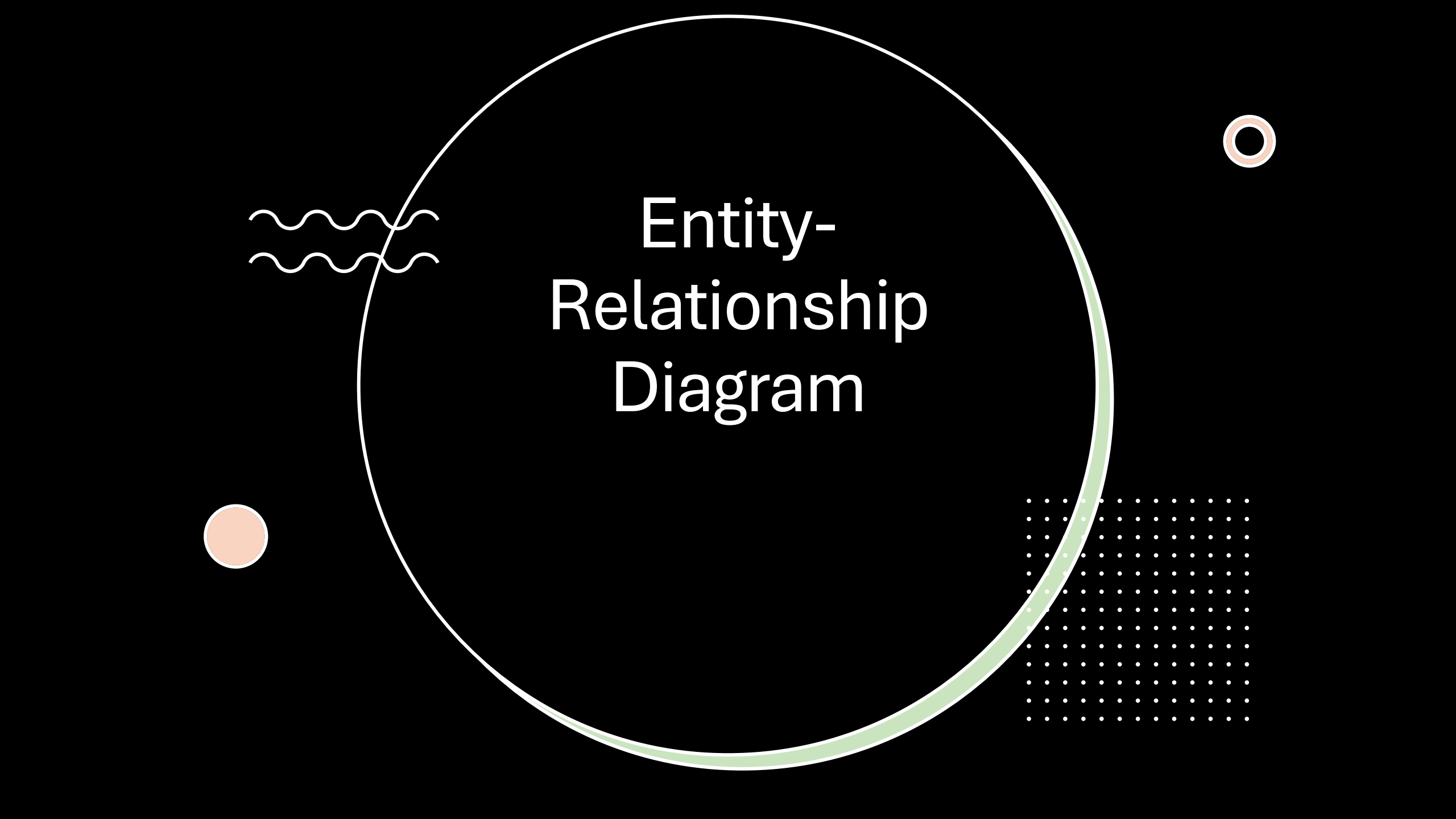
- The salesperson must be logged in.
- At least one booking with a Pending status exists.

Basic Flow:

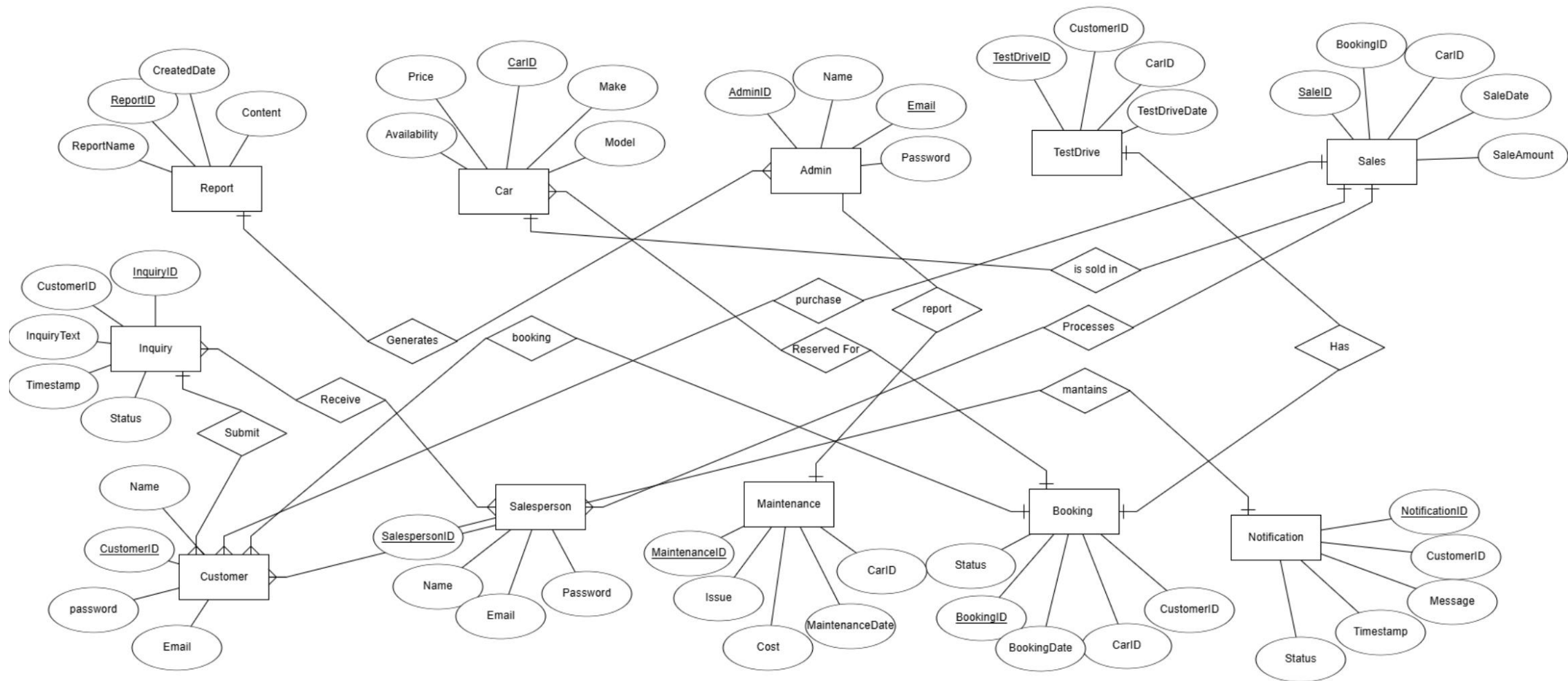
1. The salesperson logs into the system and navigates to the "View All Bookings" page.
2. The system displays a list of bookings with their details (Customer Name, Car, Status).
3. The salesperson identifies a Pending booking.
4. The salesperson clicks the "Approve" button for the selected booking.
5. The system updates the booking status to "Approved".
6. The system updates the car's availability to Unavailable.
7. The system sends a notification to the customer: "Your booking has been approved."
8. The system displays a success message: "Booking approved successfully."

Postconditions:

- The booking status is updated to Approved in the database.
- The car's availability is updated to Unavailable in the inventory.
- A notification is sent to the customer.



Entity- Relationship Diagram



Relational Database Schema

Salesperson

<u>SalesPersonID</u>	Name	Email	Password
----------------------	------	-------	----------

Customer

<u>CustomerID</u>	Name	Email	Password
-------------------	------	-------	----------

Admin

<u>AdminID</u>	Name	Email	Password
----------------	------	-------	----------

Car

<u>CarID</u>	Model	Make	Price	Availability
--------------	-------	------	-------	--------------

Sales

<u>SaleID</u>	CarID	BookingID	SaleDate	SaleAmount
---------------	-------	-----------	----------	------------

Booking

<u>BookingID</u>	BookingDate	Status	CustomerID	CarID
------------------	-------------	--------	------------	-------

Maintenance

<u>MaintenanceID</u>	CarID	Issue	MaintenanceDate	Cost
----------------------	-------	-------	-----------------	------

TestDrive

<u>TestDriveID</u>	CustomerID	CarID	TestDriveDate
--------------------	------------	-------	---------------

Report

<u>ReportID</u>	ReportName	CreatedDate	Content
-----------------	------------	-------------	---------

Inquiry

<u>InquiryID</u>	CustomerID	InquiryText	Timestamp	Status
------------------	------------	-------------	-----------	--------

Notification

<u>NotificationID</u>	CustomerID	Message	Timestamp	Status
-----------------------	------------	---------	-----------	--------

SQL Queries: CREATE

```
-- Customer Table
CREATE TABLE Customer (
    CustomerID NUMBER PRIMARY KEY,
    Name VARCHAR2(100) NOT NULL,
    Address VARCHAR2(255),
    PhoneNumber VARCHAR2(15),
    Email VARCHAR2(100) NOT NULL UNIQUE,
    Password VARCHAR2(100) NOT NULL
);

-- SalesPerson Table
CREATE TABLE SalesPerson (
    SalesPersonID NUMBER PRIMARY KEY,
    Name VARCHAR2(100) NOT NULL,
    Department VARCHAR2(100),
    PhoneNumber VARCHAR2(15),
    Email VARCHAR2(100) NOT NULL UNIQUE,
    Password VARCHAR2(100) NOT NULL
);

-- Admin Table
CREATE TABLE Admin (
    AdminID NUMBER PRIMARY KEY,
    Name VARCHAR2(100) NOT NULL,
    PhoneNumber VARCHAR2(15),
    Email VARCHAR2(100) NOT NULL UNIQUE,
    Password VARCHAR2(100) NOT NULL
);
```

```
-- Car Table
CREATE TABLE Car (
    CarID NUMBER PRIMARY KEY,
    Make VARCHAR2(100) NOT NULL,
    Model VARCHAR2(100) NOT NULL,
    Year NUMBER NOT NULL,
    Price NUMBER NOT NULL,
    Specifications VARCHAR2(255),
    Status VARCHAR2(50),
    ImageURL VARCHAR2(255)
);

-- Report Table
CREATE TABLE Report (
    ReportID NUMBER PRIMARY KEY,
    AdminID NUMBER REFERENCES Admin(AdminID),
    ReportType VARCHAR2(50),
    DateGenerated DATE,
    FileURL VARCHAR2(255)
);

-- Maintenance Table
CREATE TABLE Maintenance (
    MaintenanceID NUMBER PRIMARY KEY,
    CarID NUMBER REFERENCES Car(CarID),
    MaintenanceDate DATE NOT NULL,
    Description VARCHAR2(255),
    MaintenanceStatus VARCHAR2(50)
);
```

```
-- Booking Table
CREATE TABLE Booking (
    BookingID NUMBER PRIMARY KEY,
    BookingType VARCHAR2(50),
    BookingDate DATE,
    ExpirationDate DATE,
    BookingStatus VARCHAR2(50),
    CustomerID NUMBER REFERENCES Customer(CustomerID),
    CarID NUMBER REFERENCES Car(CarID)
);

-- TestDrive Table
CREATE TABLE TestDrive (
    TestDriveID NUMBER PRIMARY KEY,
    BookingID NUMBER REFERENCES Booking(BookingID),
    TestDriveDate DATE,
    Feedback VARCHAR2(255)
);

-- Sales Table
CREATE TABLE Sales (
    SaleID NUMBER PRIMARY KEY,
    CarID NUMBER REFERENCES Car(CarID),
    CustomerID NUMBER REFERENCES Customer(CustomerID),
    SaleDate DATE,
    TotalPrice NUMBER NOT NULL,
    InvoiceURL VARCHAR2(255)
);
```

```
-- Notification Table
CREATE TABLE Notification (
    NotificationID NUMBER PRIMARY KEY,
    CustomerID NUMBER REFERENCES Customer(CustomerID),
    Message VARCHAR2(255),
    NotificationType VARCHAR2(50),
    NotificationDate DATE,
    ReadStatus VARCHAR2(50)
);
```

SQL Queries

REQ_002: Manage and display car inventory.

```
SELECT * FROM Car WHERE Availability = 1;
```

```
SELECT * FROM Car WHERE Availability = 0;
```

REQ_006: Enable filtering by price range.

```
SELECT * FROM Car WHERE Price BETWEEN 20000 AND 50000;
```

Approve a Booking

```
UPDATE Booking SET Status = 'Approved' WHERE BookingID = 5;
```

```
UPDATE Car SET Availability = 0 WHERE CarID = (SELECT CarID FROM Booking WHERE  
BookingID = 5);
```

View Car Inventory

```
SELECT CarID, Model, Make, Price, Availability FROM Car WHERE Availability = 1;
```

```
SELECT CarID, Model, Make, Price, Availability FROM Car WHERE Availability = 0;
```

Initial Prompt:

Project Title: Functional Website for a Car Dealership System

Overview:

I want to build a fully functional website for a car dealership system that integrates with a database managed in Microsoft SQL Server Management Studio. The website should have distinct user roles: Admin, Salesperson, and Customer, each with specific functionalities.

Core Requirements:

Login/Signup Page:

Allow users to register and log in securely.
Differentiate between Admin, Salesperson, and Customer roles upon login.
Role-Based Functionalities:

Customer:

Book a car.
Book a test drive.

Salesperson:

Book a car or test drive on behalf of a customer.
View all bookings and test drives.

Admin:

Manage car maintenance (add, update, or delete records).
View sales reports (generated from the database).

Database Schema:

I've already designed the database schema with the following tables:

Customer, SalesPerson, Admin, Car, Report, Maintenance, Booking, TestDrive, Sales, Notification.
(See detailed schema below.)

Technical Details:

Frontend: Open to suggestions for the framework (e.g., React, Angular, or plain HTML/CSS/JavaScript).

Backend: Could use a server-side language like Python (Django/Flask), Node.js, or ASP.NET.

Database: Microsoft SQL Server.

Additional Features:

Responsive design for mobile and desktop.

Secure password storage and authentication (e.g., hashing passwords).

Notification system for customers (e.g., test drive reminders or booking confirmations).

Database Schema:

The schema includes tables for managing users, cars, bookings, test drives, maintenance, sales, reports, and notifications.

-- Customer Table

```
CREATE TABLE Customer (  
    CustomerID NUMBER PRIMARY KEY,  
    Name VARCHAR2(100) NOT NULL,  
    Address VARCHAR2(255),  
    PhoneNumber VARCHAR2(15),  
    Email VARCHAR2(100) NOT NULL UNIQUE,  
    Password VARCHAR2(100) NOT NULL  
);
```

-- SalesPerson Table

```
CREATE TABLE SalesPerson (  
    SalesPersonID NUMBER PRIMARY KEY,  
    Name VARCHAR2(100) NOT NULL,  
    Department VARCHAR2(100),  
    PhoneNumber VARCHAR2(15),  
    Email VARCHAR2(100) NOT NULL UNIQUE,  
    Password VARCHAR2(100) NOT NULL  
);
```

-- Admin Table

```
CREATE TABLE Admin (  
    AdminID NUMBER PRIMARY KEY,  
    Name VARCHAR2(100) NOT NULL,  
    PhoneNumber VARCHAR2(15),  
    Email VARCHAR2(100) NOT NULL UNIQUE,  
    Password VARCHAR2(100) NOT NULL  
);
```

Test Cases

Test Case ID	Description	Preconditions	Test Data	Test Steps	Expected Result	Actual Result	Status
TC_006_01	Search for cars within a price range.	Customer is logged in.	Min Price: 15000, Max Price: 30000.	1. Navigate to "Search Cars". 2. Enter a valid price range. 3. Submit the form.	Cars within the specified price range are displayed.	Cars within the specified price range are displayed.	Pass
TC_006_02	Search for cars with no matching results.	Customer is logged in.	Min Price: 80000, Max Price: 90000.	1. Navigate to "Search Cars". 2. Enter a price range with no matching cars. 3. Submit the form.	A message is displayed stating "No cars found in the specified price range."	A message is displayed stating "No cars found in the specified price range."	Pass
TC_006_03	Attempt to search with invalid input.	Customer is logged in.	Min Price: -5000, Max Price: 10000.	1. Navigate to "Search Cars". 2. Enter invalid input in the price range fields. 3. Submit the form.	A message is displayed stating "Invalid input."	A message is displayed stating "No cars found in the specified price range."	Fail

Test Case ID	Description	Preconditions	Test Data	Test Steps	Expected Result	Actual Result	Status
TC_002_01	Display available cars in inventory.	Admin is logged in.	None.	1. Navigate to "Manage Cars".	All available cars are displayed with details like make, model, and price.	Car listed as available	Pass
TC_002_02	Display sold cars in inventory.	Admin is logged in.	Car ID: 2, Status: Sold.	1. Navigate to "Manage Cars". 2. Change status of car to "Sold". 3. Refresh the inventory page.	Car is listed under the "Sold" status.	Car listed as unavailable	Pass
TC_002_03	Display cars under maintenance.	Admin is logged in.	Car ID: 3, Status: Under Maintenance.	1. Navigate to "Manage Maintenance". 2. Add a maintenance record for a car. 3. Check car status in inventory.	Car is listed as "Under Maintenance" in the inventory.	Car listed as available	Fail

Test Case ID	Description	Preconditions	Test Data	Test Steps	Expected Result	Actual Result	Status
TC_011_01	Salesperson approves booking.	Salesperson is logged in, and booking waiting to be approved.	None.	1. Log in as Salesperson. 2. Navigate to the "View Bookings" page. 3. Click "Approve" on booking.	The booking is approved, and status of car changed to 'Unavailable'	The booking is approved, and status of car changed to 'Unavailable'	Pass
TC_011_02	Salesperson views all bookings and their statuses successfully.	Salesperson is logged in. Bookings exist in the database for multiple customers.	None.	1. Log in as a Salesperson. 2. Navigate to the "View All Bookings" page. 3. Verify that the list displays bookings with details (Customer Name, Car Model, Status).	All bookings are displayed with accurate information.	Bookings are displayed correctly.	Pass

Automated Test Cases

Sign Up As Customer

```
[Documentation]    Test signing up as a new customer.
Open Browser      ${BASE_URL}    ${BROWSER}
Click Link        xpath=//a[contains(text(), 'Sign up here')]    # Navigate to the signup page
Input Text        name=name       ${CUSTOMER_NAME}
Input Text        name=email      ${CUSTOMER_EMAIL}
Input Text        name=password   ${CUSTOMER_PASSWORD}
Select From List By Value    name=role    Customer
Click Button      xpath=//button[@type='submit']    # Submit the form
Wait Until Page Contains    Login    # Ensure the login page is displayed
Close Browser
```

Approve A Pending Booking

```
[Documentation]    Test approving a pending booking as a salesperson.
Open Browser      ${BASE_URL}    ${BROWSER}

# Step 1: Login as Salesperson
Input Text        name=email      ${SALESPERSON_EMAIL}
Input Text        name=password   ${SALESPERSON_PASSWORD}
Select From List By Value    name=role    Salesperson
Click Button      xpath=//button[@type='submit']
Wait Until Page Contains    Welcome, Salesperson

# Step 2: Navigate to "View Bookings" page
Click Link        xpath=//a[contains(text(), 'View All Bookings')]
Wait Until Page Contains    Booking ID

# Step 3: Approve the first pending booking
Click Button      xpath=//form[1]//button[@type='submit']    # Approve the first pending booking
Wait Until Page Contains    Approved    # Wait for the status to update to "Approved"

# Step 4: Verify the status change in the row
Element Text Should Be    xpath=//table//tr[1]//td[last()]    Approved

# Close the browser after the test
Close Browser
```

Book A Car

```
[Documentation]    Test booking a car as a customer.
Open Browser      ${BASE_URL}    ${BROWSER}

# Step 1: Login as Customer
Input Text        name=email      ${CUSTOMER_EMAIL}
Input Text        name=password   ${CUSTOMER_PASSWORD}
Select From List By Value    name=role    Customer
Click Button      xpath=//button[@type='submit']
Wait Until Page Contains    Welcome, Customer

# Step 2: Navigate to "Book a Car" page
Click Link        xpath=//a[contains(text(), 'Book a Car')]
Wait Until Page Contains    Available Cars

# Step 3: Book the first available car
Click Button      xpath=//form[1]//button[@type='submit']    # Select the first car
Wait Until Page Contains    Car booked successfully!    # Verify booking success

# Step 4: Return to Customer Dashboard
Click Link        xpath=//a[contains(text(), 'Back to Dashboard')]

# Step 5: Verify dashboard still accessible
Wait Until Page Contains    Welcome, Customer

# Close the browser after the test
Close Browser
```


Results

```
=====
Test Cases
=====
```

```
Sign Up As Customer :: Test signing up as a new customer.
```

```
DevTools listening on ws://127.0.0.1:51621/devtools/browser/6b713c92-a5c3-449f-8d47-3d703484658f
```

```
Sign Up As Customer :: Test signing up as a new customer. | PASS |
```

```
-----
Book A Car :: Test booking a car as a customer.
```

```
DevTools listening on ws://127.0.0.1:51652/devtools/browser/8d240107-291b-4be2-904d-72a6b4756d32
```

```
Book A Car :: Test booking a car as a customer. | PASS |
```

```
-----
Approve A Pending Booking :: Test approving a pending booking as a...
```


```
DevTools listening on ws://127.0.0.1:51696/devtools/browser/7dace377-fa3a-4223-8888-8114be2e312b
```

```
Approve A Pending Booking :: Test approving a pending booking as a... | PASS |
```

```
-----
Test Cases | PASS |
```

```
3 tests, 3 passed, 0 failed
```

```
=====
```



Challenges Faced and Solutions



What Would You Do Differently?



Adopt a Frontend Framework

Change: Use a modern frontend framework like React or Angular to improve user experience and make the design more dynamic.

Reason: The current HTML/CSS implementation is functional but lacks interactivity and scalability for larger applications.



Enhance Security

Change: Implement password hashing (e.g., bcrypt) and HTTPS to further secure user data.

Reason: Plain-text password storage in the database is a security risk and should be avoided.



Optimize Testing:

Change: Increase test automation coverage and integrate a CI/CD pipeline for continuous testing and deployment.

Reason: Automating more workflows would save time and reduce human errors during testing.

What Would You Do the Same?



Flask for Backend:

Keep: Use Flask for its simplicity and flexibility.

Reason: It allowed for rapid development and easy integration with the database.



Role-Based Access Control:

Keep: The current role-based access control implementation.

Reason: It effectively ensured users only accessed functionalities relevant to their roles.



Relational Database Design:

Keep: The normalized database schema.

Reason: It provided clear relationships between entities and supported efficient querying.