SYMBIOSIS INSTITUTE OF TECHNOLOGY (SIT)
Constituent of Symbiosis International (Deemed University), Pune
(Established under Section 3 of the UGC Act of 1956 vide notification number F-9-12/2001-U-3 of the Government of india)
Re-Accredited by NAAC with 'A++' Grade
॥वसुधैव कुटुम्बकम्॥

TDM

CA-1

Car Rental Service Database

Arya Dhole	22070123027
Hardik Kalsule	22070123045
Abhishek Verma	22070123004
Mohd Areeb Idrees	22070123069
Yaduraj Pawar	22070123134

Department of Electronics and Telecommunication Engineering
Symbiosis Institute of Technology, Pune

SYMBIOSIS INSTITUTE OF TECHNOLOGY (SIT)
Constituent of Symbiosis International (Deemed University), Pune
(Established under Section 3 of the UGC Act of 1956 vide notification number F-9-12/2001-U-3 of the Government of india)
Re-Accredited by NAAC with 'A++' Grade
॥वसुधैव कुटुम्बकम्॥

Index

Sr No	Topic/Content	Page No
1	Introduction	3
2	ER and EER Diagram Design	6
3	Codd's 12 Rules Justification	12
4	Normalization Process	15
5	Indexing	23
6	Views	25
7	Procedure and Triggers	28
11	Conclusion	34

Department of Electronics and Telecommunication Engineering
Symbiosis Institute of Technology, Pune

Introduction

Car rental services provide a convenient and flexible solution for customers who require temporary access to vehicles for travel, business, or leisure. Managing a car rental service involves multiple aspects, including customer registrations, vehicle availability, rental bookings, billing, and maintenance tracking. To ensure seamless operations, a well-structured **relational database management system (RDBMS)** is necessary.

This project focuses on designing a **Car Rental Service Database** that efficiently stores, organizes, and manages data related to customers, vehicles, rental bookings, payments, and maintenance schedules. The database adheres to **Codd's 12 Rules for relational databases** and is **normalized up to Third Normal Form (3NF)** to ensure data integrity and reduce redundancy.

Core Functionalities

1. Customer Management

The system stores and maintains all essential customer-related data, allowing for quick and efficient rental processes.

- **Customer Information Storage:**
    - **Personal Details:** Name, date of birth, gender, and nationality.
    - **Contact Information:** Email, phone number, and emergency contact.
    - **Address:** Permanent and temporary addresses.
    - **Driving License Details:** License number, issuing state, and expiration date.
    - **Unique Identifiers:** Each customer is assigned a **unique ID** (primary key) to ensure proper referencing in the database.
- **Customer Record Updates:** Customers can update personal details, change contact information, or modify their rental preferences.
- **Data Validation:** The system uses constraints and triggers to validate customer data before insertion (e.g., no duplicate license numbers).

2. Vehicle Management

A well-organized vehicle management system ensures smooth operations and optimized fleet utilization.

- **Vehicle Information Storage:**
    - **Vehicle Details:** Model, make, year, registration number, and color.
    - **Category:** Sedan, SUV, hatchback, luxury, or economy.
    - **Rental Pricing:** Hourly, daily, or weekly rental rates.
    - **Availability Status:** Whether the vehicle is currently rented or available.

    - **Mileage & Maintenance Records:** Tracks mileage history and maintenance schedules.

- **Vehicle Maintenance Tracking:** The system records vehicle servicing details, including **last service date**, **next due service**, and **repair history**.
- **Data Integrity Measures:** Foreign key constraints link each vehicle to rental records to prevent **double bookings** or improper status updates.

3. Rental Booking Management

This functionality is the core of the system, ensuring efficient and error-free booking processes.

- **Booking Creation:** Customers can rent vehicles by specifying rental start and end dates.
- **Availability Checking:** The system ensures that the requested vehicle is available for the selected rental period before confirming the booking.
- **Rental Contract Storage:** Captures essential details such as rental duration, total cost, security deposit, and pickup/drop-off location.
- **Modifying & Cancelling Rentals:** Customers can modify or cancel bookings based on system rules and availability.
- **Automatic Status Updates:** The system updates the vehicle's **availability status** once a rental is confirmed or completed.

4. Payment & Billing Management

A robust billing system ensures seamless financial transactions and revenue tracking.

- **Payment Processing:** Customers can make payments using different payment methods, such as credit/debit cards, online banking, or cash.
- **Invoice Generation:** Automatically generates **detailed invoices**, including rental charges, additional fees (if any), and applicable discounts.
- **Security Deposit Handling:** The system tracks and refunds deposits upon vehicle return.
- **Late Fee Calculation:** If a vehicle is returned late, the system automatically calculates and applies late fees.
- **Discount & Loyalty Programs:** Customers can avail of special discounts based on their rental history or promotional offers.

5. Data Integrity & Security

Maintaining accurate and reliable data is crucial for the system's efficiency.

- **Referential Integrity:** Uses **foreign keys** to link rental bookings to customers and vehicles, ensuring valid relationships.
- **Preventing Duplicate Bookings:** Triggers enforce the rule that prevents a vehicle from being rented to multiple customers simultaneously.
- **Data Validation:** Ensures valid license numbers, proper email formats, and non-null essential fields before inserting or updating data.

6. Reporting & Data Analytics

The system provides valuable insights for administrators and managers.

- **Booking Reports:** Displays total rentals per vehicle, per customer, or per time period.
- **Revenue Reports:** Summarizes rental earnings, late fees, and outstanding payments.
- **Fleet Utilization Reports:** Shows which vehicles are rented most frequently and identifies idle vehicles.
- **Customer Insights:** Tracks repeat customers and their rental preferences for marketing purposes.
- **Real-Time Data Access:** Reports are updated dynamically as new bookings, payments, and cancellations occur.

7. Automation & Optimization

Automation enhances system efficiency by reducing manual intervention.

- **Stored Procedures:**
    - A procedure to **calculate rental costs** based on duration and vehicle type.
    - A procedure to **fetch available vehicles** for a given time period.
- **Triggers for Business Rules:**
    - A trigger to automatically **update vehicle availability** when a rental is confirmed or completed.
    - A trigger to **apply discounts** based on customer rental history.
- **Indexing for Performance:**
    - Frequently queried fields like **CustomerID**, **VehicleID**, and **RentalDate** are indexed using **B-Tree indexing** for **faster retrieval**.

ER and EER Diagram Design

The **Entity-Relationship (ER) model** is a high-level conceptual design that visually represents data, showing how entities (objects) relate to each other within a system. The **Enhanced Entity-Relationship (EER) model** builds upon the ER model by introducing advanced features such as subclasses, superclasses, and constraints, making it suitable for more complex real-world applications. These models help in structuring databases efficiently, ensuring data integrity and reducing redundancy.

2.1. ER Diagram

An ER diagram is a graphical representation of a database that illustrates entities, their attributes, and relationships. It helps in designing the structure of a database before implementation.

Entities and Attributes with Data Types

1. Customer (Primary Entity)

- CustomerID (PK, INT, AUTO_INCREMENT)
- FName (VARCHAR(50), NOT NULL)
- MName (VARCHAR(50), NULLABLE)
- LName (VARCHAR(50), NOT NULL)
- PhoneNumber (VARCHAR(15), UNIQUE, NOT NULL)
- DrivingLicense (VARCHAR(20), UNIQUE, NOT NULL)
- EmailID (VARCHAR(100), UNIQUE, NOT NULL)
- Address (TEXT, NULLABLE)
- IsPremium (BOOLEAN, NOT NULL)
- MemberID (INT, NULLABLE)
- CreatedAt (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)
- UpdatedAt (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP)

2. Car (Primary Entity)

- RegistrationNumber (PK, VARCHAR(20))
- AvailabilityFlag (BOOLEAN, NOT NULL)
- ModelYear (INT, NOT NULL)
- Model (VARCHAR(50), NOT NULL)
- MadeBy (VARCHAR(50), NOT NULL)
- Mileage (DECIMAL(5,2), NOT NULL)

3. Location (Primary Entity)

- LocationID (PK, INT, AUTO_INCREMENT)
- Name (VARCHAR(100), NOT NULL)
- State (VARCHAR(50), NOT NULL)
- City (VARCHAR(50), NOT NULL)
- ZipCode (VARCHAR(10), NOT NULL)
- Street (VARCHAR(100), NOT NULL)

4. Car Category (Primary Entity)

- CategoryID (PK, INT, AUTO_INCREMENT)
- Name (VARCHAR(50), NOT NULL)
- NumberOfLuggage (INT, NOT NULL)
- NumberOfPerson (INT, NOT NULL)
- CostPerDay (DECIMAL(8,2), NOT NULL)
- LateFeePerHour (DECIMAL(8,2), NOT NULL)

5. Booking (Relationship Entity)

- BookingID (PK, INT, AUTO_INCREMENT)
- CustomerID (FK → Customer, INT, NOT NULL)
- CarID (FK → Car, VARCHAR(20), NOT NULL)
- PickUpLocation (FK → Location, INT, NOT NULL)
- DropOffLocation (FK → Location, INT, NOT NULL)
- StartDate (DATE, NOT NULL)
- EndDate (DATE, NOT NULL)

6. Billing (Relationship Entity)

- BillingID (PK, INT, AUTO_INCREMENT)
- BookingID (FK → Booking, INT, NOT NULL)
- LateFee (DECIMAL(8,2), NULLABLE)
- AmountDiscounted (DECIMAL(8,2), NULLABLE)
- TotalAmount (DECIMAL(10,2), NOT NULL)
- TaxAmount (DECIMAL(8,2), NOT NULL)
- Status (VARCHAR(50), NOT NULL)
- BillingDate (DATE, NOT NULL)

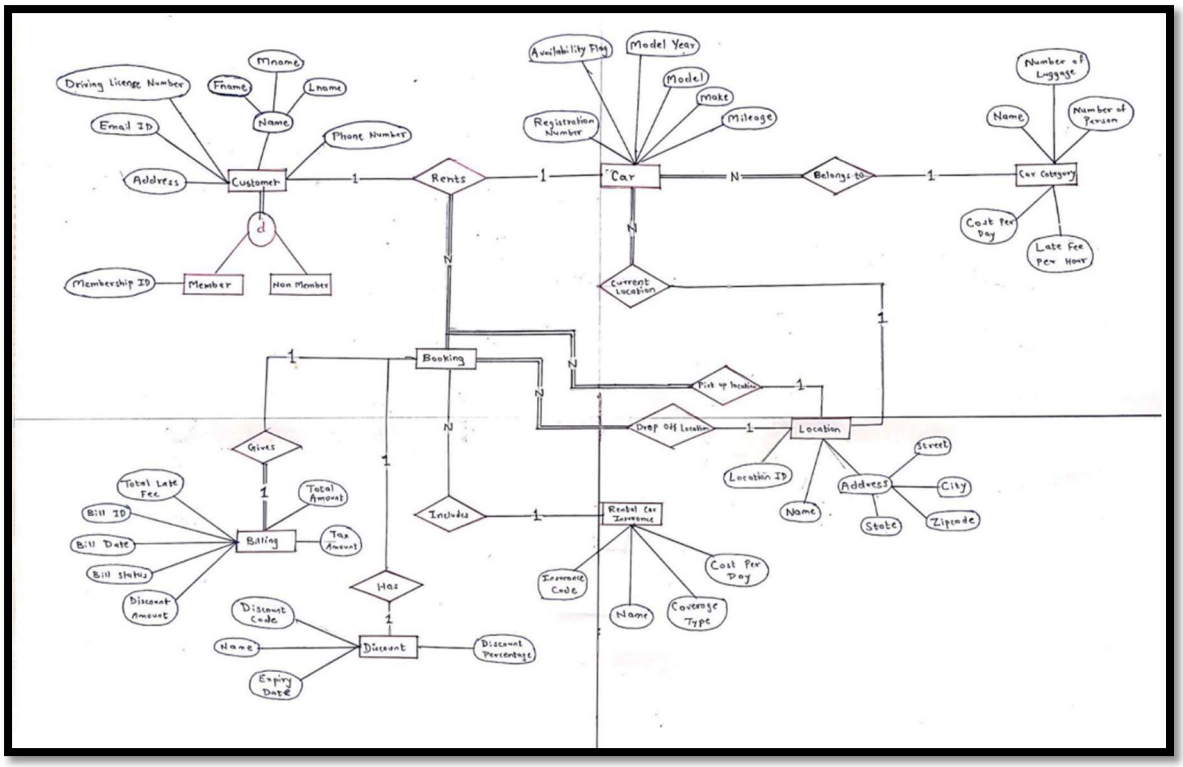
7. Discount (Primary Entity)

- DiscountID (PK, INT, AUTO_INCREMENT)
- Code (VARCHAR(20), UNIQUE, NOT NULL)
- Name (VARCHAR(50), NOT NULL)
- ExpiryDate (DATE, NOT NULL)
- DiscountPercentage (DECIMAL(5,2), NOT NULL)

8. Rental Car Insurance (Primary Entity)

- InsuranceID (PK, INT, AUTO_INCREMENT)
- Code (VARCHAR(20), UNIQUE, NOT NULL)
- Name (VARCHAR(50), NOT NULL)
- CoverageType (VARCHAR(50), NOT NULL)
- CostPerDay (DECIMAL(8,2), NOT NULL)

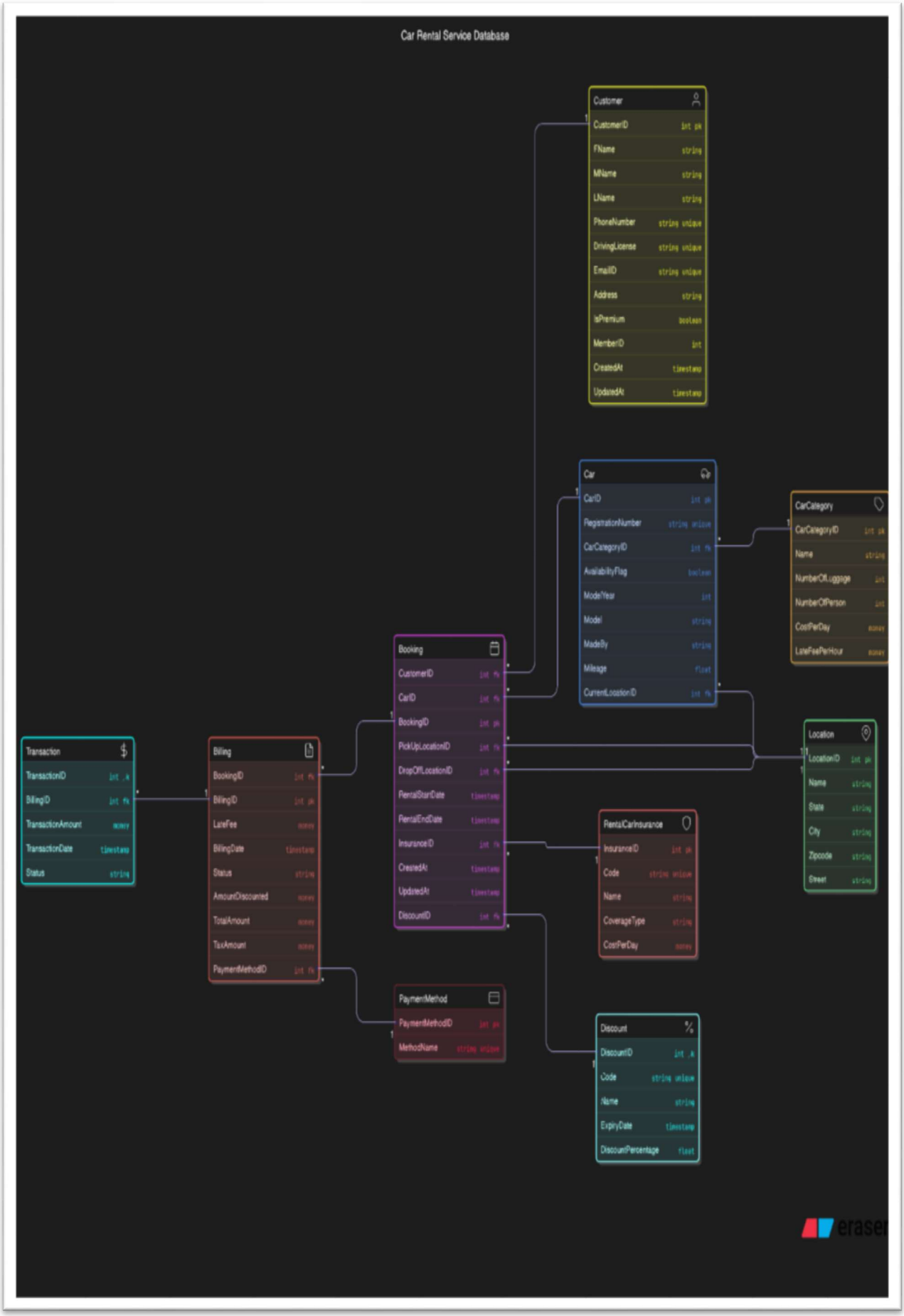
ER Diagram :



EER Diagram :

The **EER diagram** extends the ER model by adding **specialization, generalization, and inheritance** concepts. Here are some potential EER modifications:

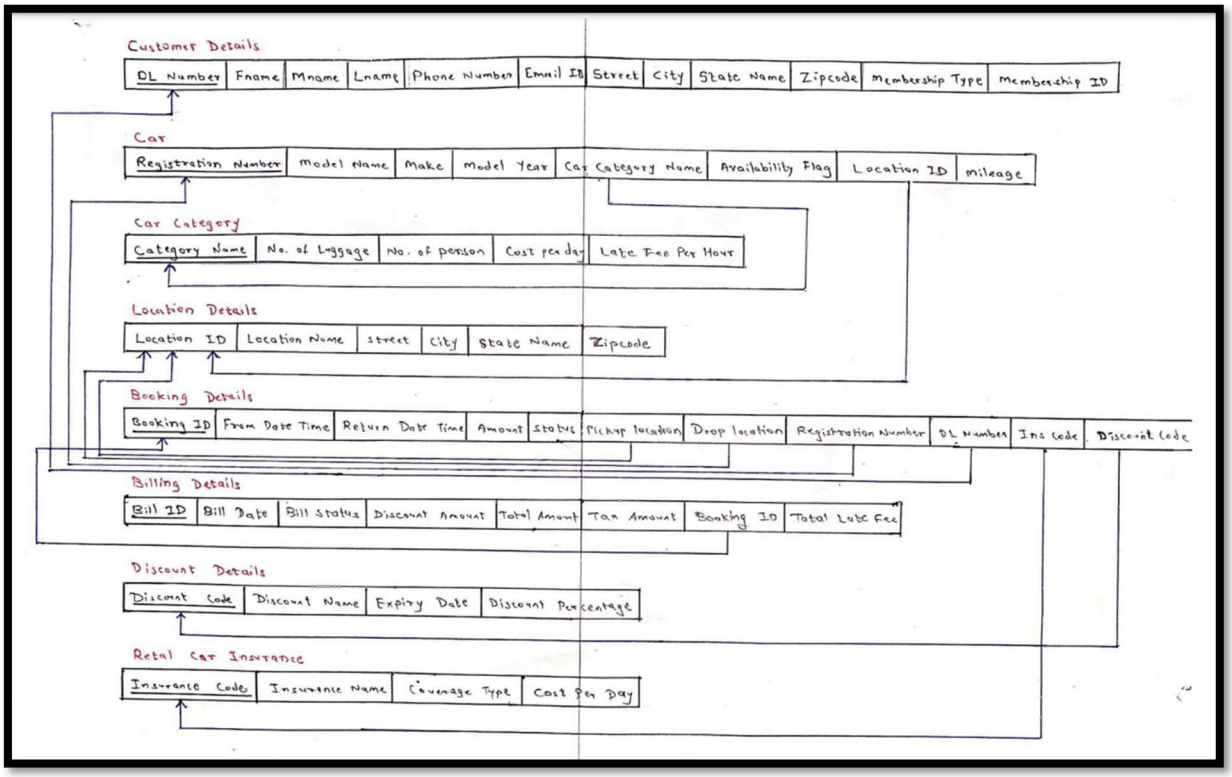
1. **Specialization in Customers**
    - **Premium Customers (Subset of Customer)**
        - Special benefits or loyalty program details.
    - **Regular Customers (Subset of Customer)**
        - Standard rental agreements.
2. **Generalization for Vehicles**
    - Car can be generalized into different vehicle types:
        - **Economy, SUV, Luxury, Convertible, Minivan** (inherits from CarCategory).
3. **Relationship Enhancements**
    - Billing is associated with Discount through a **"Has"** relationship.
    - CarCategory and Car are related via a **"Belongs to"** relationship.


Car Rental Service Database
Customer
CustomerID int pk
FName string
MName string
LName string
PhoneNumber string unique
DrivingLicense string unique
EmailID string unique
Address string
isPremium boolean
MemberID int
CreatedAt timestamp
UpdatedAt timestamp
Car
CarID int pk
RegistrationNumber string unique
CarCategoryID int fk
AvailabilityFlag boolean
ModelYear int
Model string
MadeBy string
Mileage float
CurrentLocationID int fk
CarCategory
CarCategoryID int pk
Name string
NumberOfLuggage int
NumberOfPerson int
CostPerDay money
LateFeePerHour money
Booking
CustomerID int fk
CarID int fk
BookingID int pk
PickUpLocationID int fk
DropOffLocationID int fk
RentalStartDate timestamp
RentalEndDate timestamp
InsuranceID int fk
CreatedAt timestamp
UpdatedAt timestamp
DiscountID int fk
Location
LocationID int pk
Name string
State string
City string
Zipcode string
Street string
Transaction
TransactionID int pk
BillingID int fk
TransactionAmount money
TransactionDate timestamp
Status string
Billing
BookingID int fk
BillingID int pk
LateFee money
BillingDate timestamp
Status string
AmountDiscounted money
TotalAmount money
TaxAmount money
PaymentMethodID int fk
RentalCarInsurance
InsuranceID int pk
Code string unique
Name string
CoverageType string
CostPerDay money
PaymentMethod
PaymentMethodID int pk
MethodName string unique
Discount
DiscountID int pk
Code string unique
Name string
ExpiryDate timestamp
DiscountPercentage float
eraser

2.3. Relational Model

The **Relational Model** is widely used due to its **simplicity, flexibility, and efficiency** in handling large datasets. It ensures **data consistency and integrity** through well-defined constraints such as **primary keys, foreign keys, and normalization rules**. The model also supports **logical and physical data independence**, meaning users can interact with the database without needing to know how the data is stored internally.

Additionally, **query languages like SQL** provide a powerful mechanism for retrieving and manipulating data efficiently. The relational model's **tabular structure** makes it intuitive and easy to understand, while its mathematical foundation ensures reliability. Due to these advantages, **most modern database management systems (DBMS) follow the relational model**, making it a cornerstone of database technology.



Codd's 12 Rules Justification

Codd's 12 Rules define the principles of a **Relational Database Management System (RDBMS)** to ensure data integrity, consistency, and independence. These rules mandate that data be stored in tables, accessed using primary keys, and manipulated through a comprehensive language like SQL. They enforce systematic handling of **NULL values**, data independence, and view updatability, ensuring flexibility and security. Integrity constraints must be enforced at the database level, and low-level access should not bypass these rules. Additionally, the system should support distributed databases without affecting queries or functionality.

3.1. How the Car Rental Database Adheres to Codd's 12 Rules

Below is an analysis of how the car rental database follows these rules:

1. Information Rule

All data must be stored in tables, represented explicitly as values.

- **Implementation:** The system organizes data into well-defined tables such as Customers, Cars, Bookings, and Payments. Each table contains structured columns (e.g., CustomerName, CarModel, BookingDate), and rows store individual records.
- **Compliance:** This structured format ensures that all information is stored in a tabular form, making retrieval and management efficient.

2. Guaranteed Access Rule

Every data item must be uniquely accessible using a combination of the table name, primary key, and column name.

- **Implementation:** Each table uses **primary keys** (e.g., CustomerID for Customers, BookingID for Bookings) to uniquely identify records.
- **Compliance:** The use of primary keys ensures **direct access to any data element**, preventing redundancy and ensuring efficient retrieval.

3. Systematic Treatment of NULL Values

The database must support NULL values to represent missing or inapplicable data.

- **Implementation:** Certain columns, such as ReturnDate in Bookings, allow NULL values if a car has not yet been returned.
- **Compliance:** This flexibility ensures **data completeness** by distinguishing between known values and missing or undefined data.

4. Dynamic Online Catalog (Data Dictionary)

The system should store metadata in a structured and accessible way.

- **Implementation:** The database maintains system metadata in a schema, storing details about tables, columns, and constraints.
- **Compliance:** Using the INFORMATION_SCHEMA and SHOW TABLES commands, users can **query metadata dynamically**, ensuring efficient database management.

5. Comprehensive Data Sub-Language Rule

A relational system must support at least one language that can define, manipulate, and query data.

- **Implementation:** The database supports **SQL**, which enables table creation, data insertion, updates, and retrieval.
- **Compliance:** SQL allows both **DDL (Data Definition Language) and DML (Data Manipulation Language)** operations, meeting the requirement for a comprehensive query language.

6. View Updatability Rule

Views should function as virtual tables and be updateable when logically possible.

- **Implementation:** The system allows views such as ActiveRentals (which filters ongoing bookings). If logically feasible, updates to views propagate to the base tables.
- **Compliance:** This enables **data abstraction and controlled access** while maintaining integrity.

7. High-Level Insert, Update, and Delete

Operations must be supported for sets of data, not just single rows.

- **Implementation:** The system supports batch updates and deletions, such as removing expired bookings.
- **Compliance:** Supporting operations on multiple records improves **efficiency and database consistency**.

8. Physical Data Independence

Changes in storage structures should not affect how data is accessed.

- **Implementation:** Users interact with data via queries, regardless of how it is physically stored (e.g., indexing, partitioning).
- **Compliance:** This abstraction ensures **robustness**, as queries remain valid even if storage structures change.

9. Logical Data Independence

Changes in table structure should not affect existing applications.

- **Implementation:** If columns are added to Cars, existing applications and queries should continue to function without modification, as long as they do not rely on the new attributes.
- **Compliance:** Applications remain **unaffected by modifications**, ensuring flexibility in database evolution.

10. Integrity Independence

Integrity constraints should be stored in the database and not at the application level.

- **Implementation:** Constraints such as NOT NULL, UNIQUE, and FOREIGN KEY are enforced at the database level.
- **Compliance:** This ensures **consistent data validation** without relying on external application logic.

11. Distribution Independence

The system should work the same way regardless of whether data is distributed across multiple locations.

- **Implementation:** The database can operate in a distributed setup, ensuring seamless access whether deployed locally or across cloud environments.
- **Compliance:** Queries and transactions function **independently of data distribution**, maintaining performance and consistency.

12. Nonsubversion Rule

If low-level access is possible, it must not bypass integrity rules defined in the database.

- **Implementation:** Even with direct access, constraints prevent unauthorized modifications (e.g., foreign key restrictions).
- **Compliance:** This ensures **data security and integrity**, preventing unauthorized or inconsistent changes.

Normalization Process

Normalization is a process in database design that eliminates redundancy and organizes data efficiently by breaking it into smaller related tables. It follows a series of normal forms (1NF, 2NF, 3NF, etc.), each reducing data anomalies and improving consistency. **First Normal Form (1NF)** removes duplicate columns and ensures atomicity by keeping values in separate rows. **Second Normal Form (2NF)** eliminates partial dependencies by ensuring all non-key attributes depend on the entire primary key. **Third Normal Form (3NF)** removes transitive dependencies, ensuring that non-key attributes depend only on the primary key, leading to a well-structured and scalable database.

4.1. 1NF (First Normal Form)

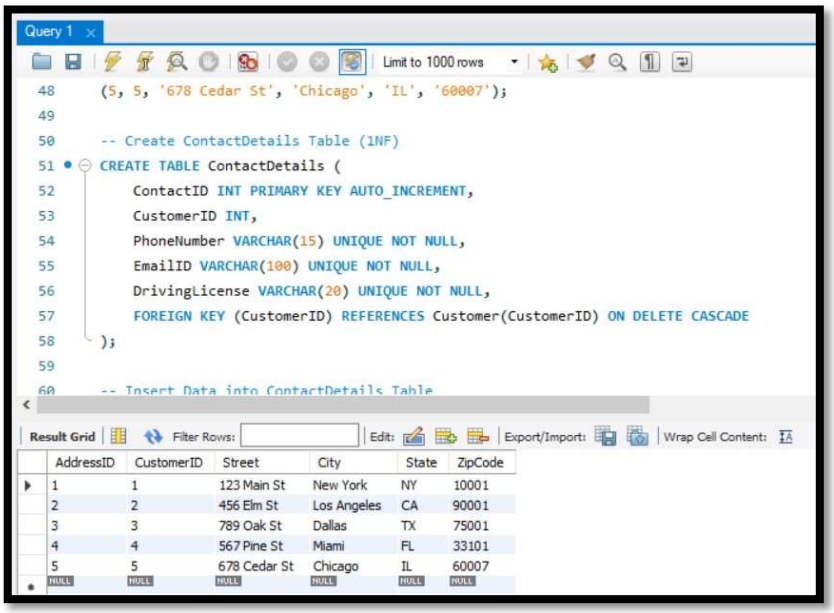
Problem in Unnormalized Form (UNF):

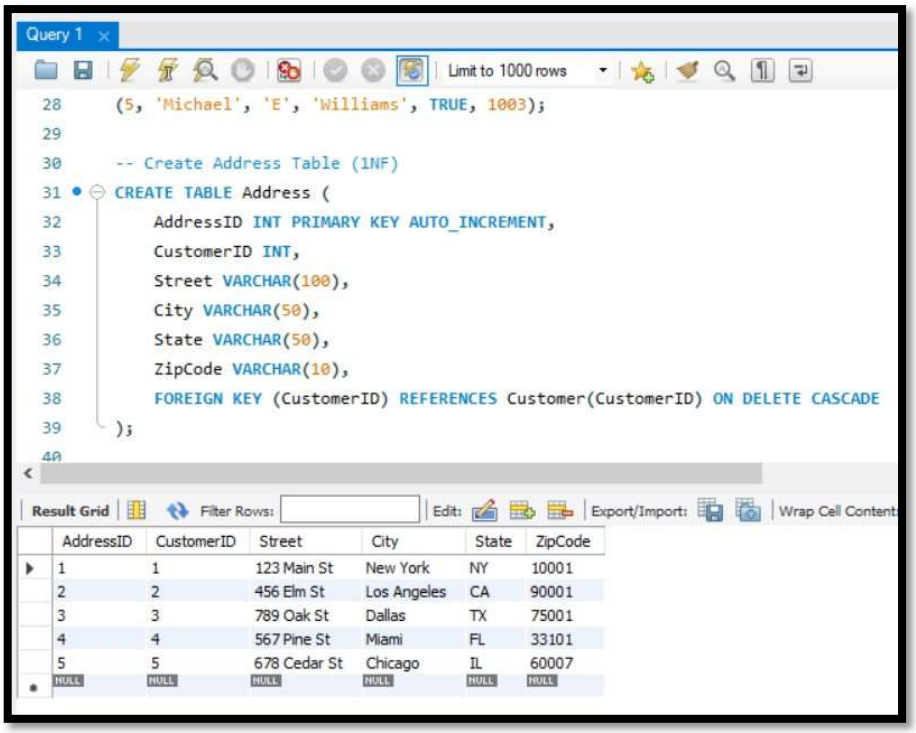
- Customers can have multiple phone numbers stored in a single field.
- Rentals might store multiple CarIDs in a single row.

Solution:

- Create a separate table **CustomerPhones** for storing multiple phone numbers.
- Ensure that each car rental transaction is stored separately for each car rented.

Code and Outputs :

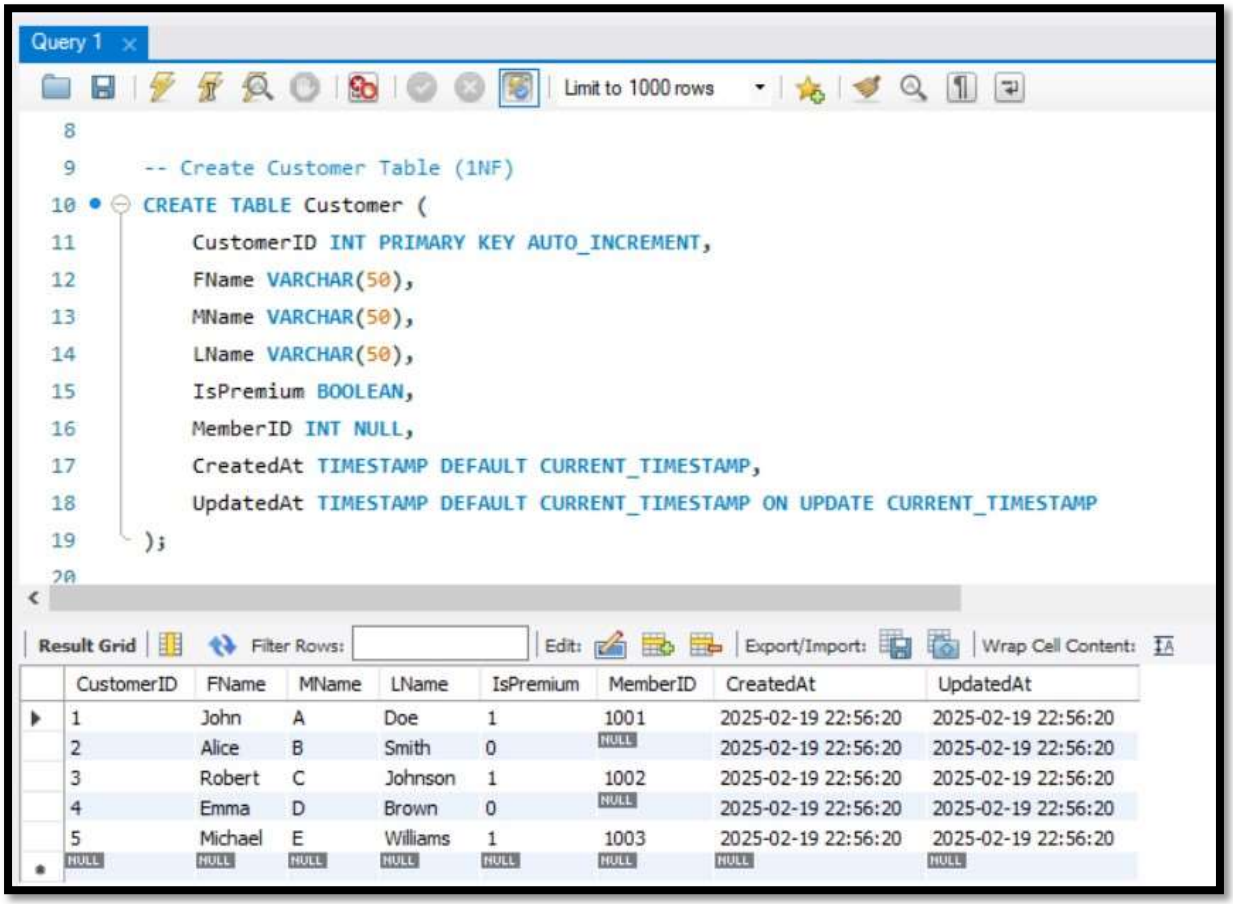




```
Query 1
Limit to 1000 rows
28      (5, 'Michael', 'E', 'Williams', TRUE, 1003);
29
30      -- Create Address Table (1NF)
31  •   CREATE TABLE Address (
32          AddressID INT PRIMARY KEY AUTO_INCREMENT,
33          CustomerID INT,
34          Street VARCHAR(100),
35          City VARCHAR(50),
36          State VARCHAR(50),
37          ZipCode VARCHAR(10),
38          FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID) ON DELETE CASCADE
39      );
40
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

AddressID	CustomerID	Street	City	State	ZipCode
1	1	123 Main St	New York	NY	10001
2	2	456 Elm St	Los Angeles	CA	90001
3	3	789 Oak St	Dallas	TX	75001
4	4	567 Pine St	Miami	FL	33101
5	5	678 Cedar St	Chicago	IL	60007
NULL	NULL	NULL	NULL	NULL	NULL



```
Query 1
Limit to 1000 rows
 8
 9      -- Create Customer Table (1NF)
10  •   CREATE TABLE Customer (
11          CustomerID INT PRIMARY KEY AUTO_INCREMENT,
12          FName VARCHAR(50),
13          MName VARCHAR(50),
14          LName VARCHAR(50),
15          IsPremium BOOLEAN,
16          MemberID INT NULL,
17          CreatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
18          UpdatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
19      );
20
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

CustomerID	FName	MName	LName	IsPremium	MemberID	CreatedAt	UpdatedAt
1	John	A	Doe	1	1001	2025-02-19 22:56:20	2025-02-19 22:56:20
2	Alice	B	Smith	0	NULL	2025-02-19 22:56:20	2025-02-19 22:56:20
3	Robert	C	Johnson	1	1002	2025-02-19 22:56:20	2025-02-19 22:56:20
4	Emma	D	Brown	0	NULL	2025-02-19 22:56:20	2025-02-19 22:56:20
5	Michael	E	Williams	1	1003	2025-02-19 22:56:20	2025-02-19 22:56:20
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

4.2. 2NF (Second Normal Form)

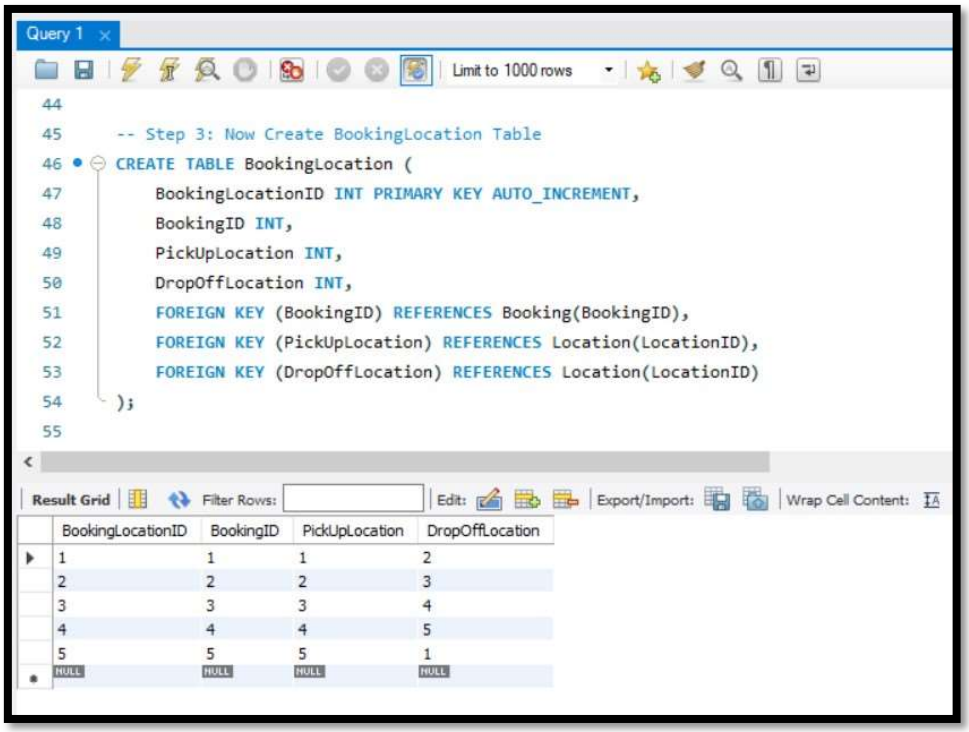
Problem in 1NF:

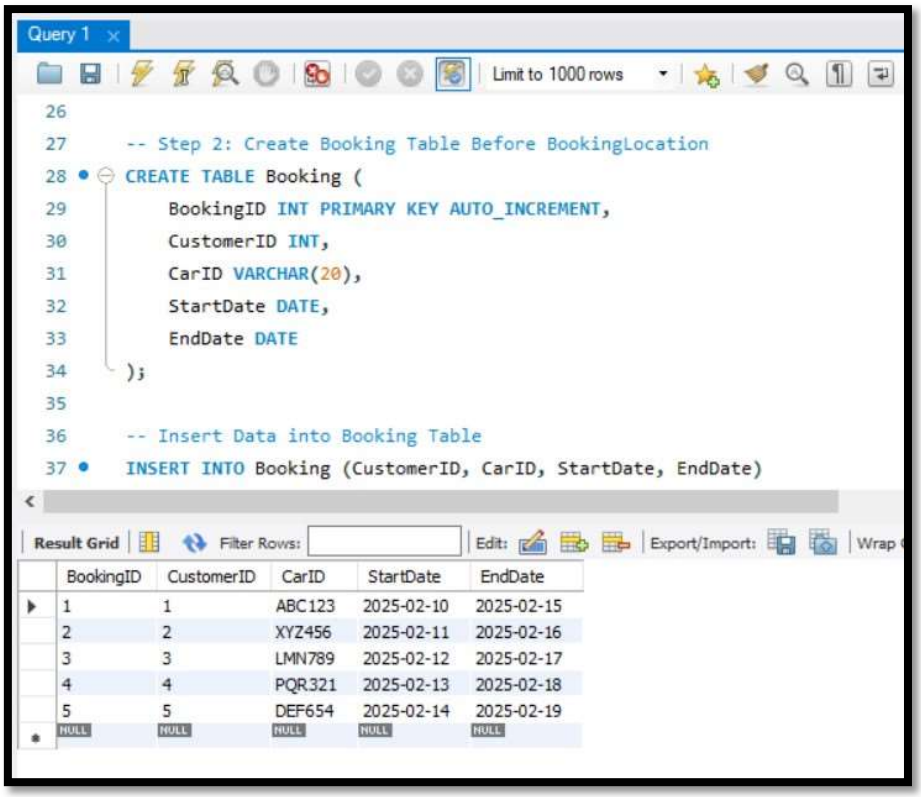
- **Rentals** table has **Car Model, Brand, and Year**, but these depend only on **CarID** rather than the full composite key **(RentalID, CarID)**.

Solution:

- Move **Car details** to a separate **Cars** table.
- Keep only **CarID** in the Rentals table to reference Cars.

Code and Output :

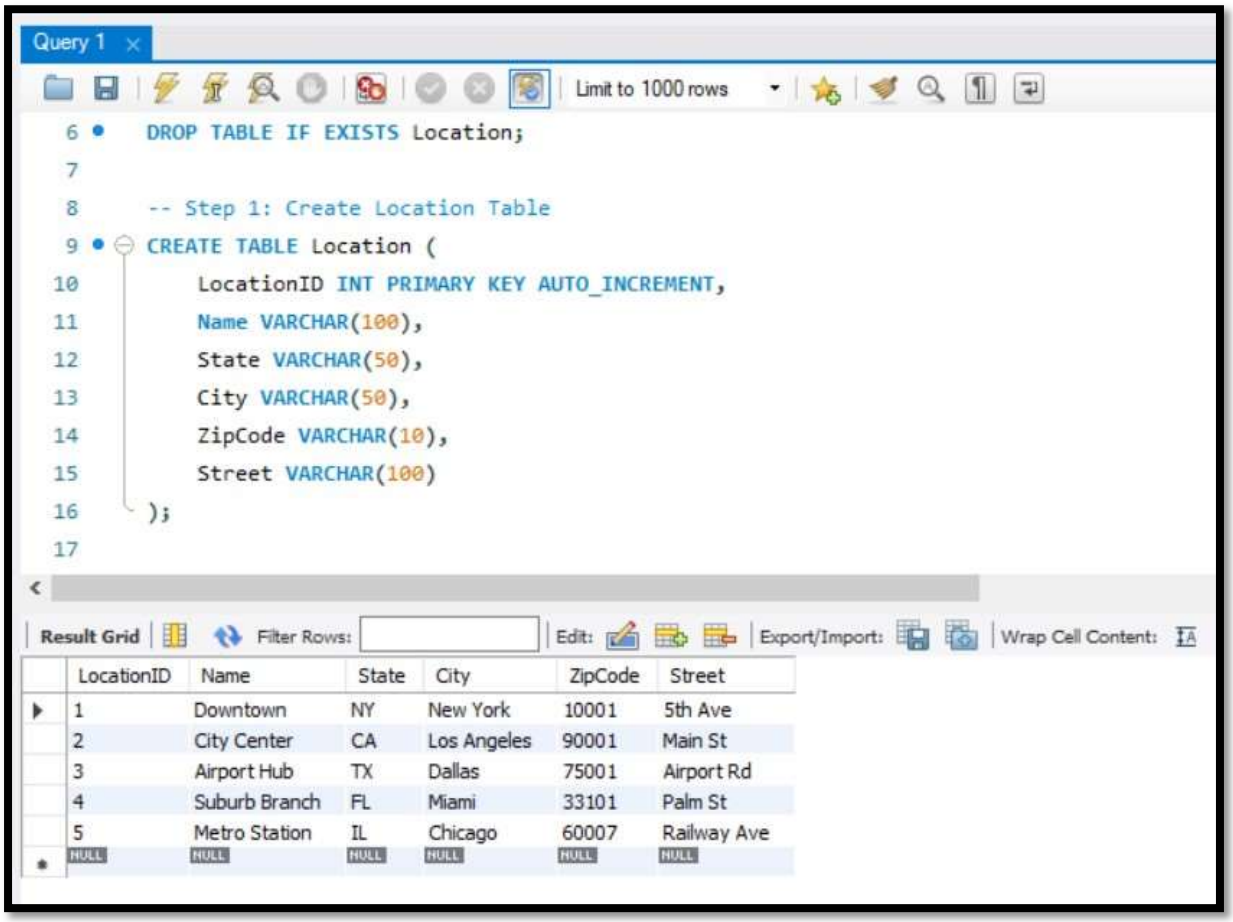




```
Query 1
Limit to 1000 rows

26
27      -- Step 2: Create Booking Table Before BookingLocation
28  •   CREATE TABLE Booking (
29          BookingID INT PRIMARY KEY AUTO_INCREMENT,
30          CustomerID INT,
31          CarID VARCHAR(20),
32          StartDate DATE,
33          EndDate DATE
34      );
35
36      -- Insert Data into Booking Table
37  •   INSERT INTO Booking (CustomerID, CarID, StartDate, EndDate)
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

BookingID	CustomerID	CarID	StartDate	EndDate
1	1	ABC123	2025-02-10	2025-02-15
2	2	XYZ456	2025-02-11	2025-02-16
3	3	LMN789	2025-02-12	2025-02-17
4	4	PQR321	2025-02-13	2025-02-18
5	5	DEF654	2025-02-14	2025-02-19
NULL	NULL	NULL	NULL	NULL



```
Query 1
Limit to 1000 rows

6   •   DROP TABLE IF EXISTS Location;
7
8       -- Step 1: Create Location Table
9   •   CREATE TABLE Location (
10          LocationID INT PRIMARY KEY AUTO_INCREMENT,
11          Name VARCHAR(100),
12          State VARCHAR(50),
13          City VARCHAR(50),
14          ZipCode VARCHAR(10),
15          Street VARCHAR(100)
16      );
17
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

LocationID	Name	State	City	ZipCode	Street
1	Downtown	NY	New York	10001	5th Ave
2	City Center	CA	Los Angeles	90001	Main St
3	Airport Hub	TX	Dallas	75001	Airport Rd
4	Suburb Branch	FL	Miami	33101	Palm St
5	Metro Station	IL	Chicago	60007	Railway Ave
NULL	NULL	NULL	NULL	NULL	NULL

4.3. 3NF (Third Normal Form)

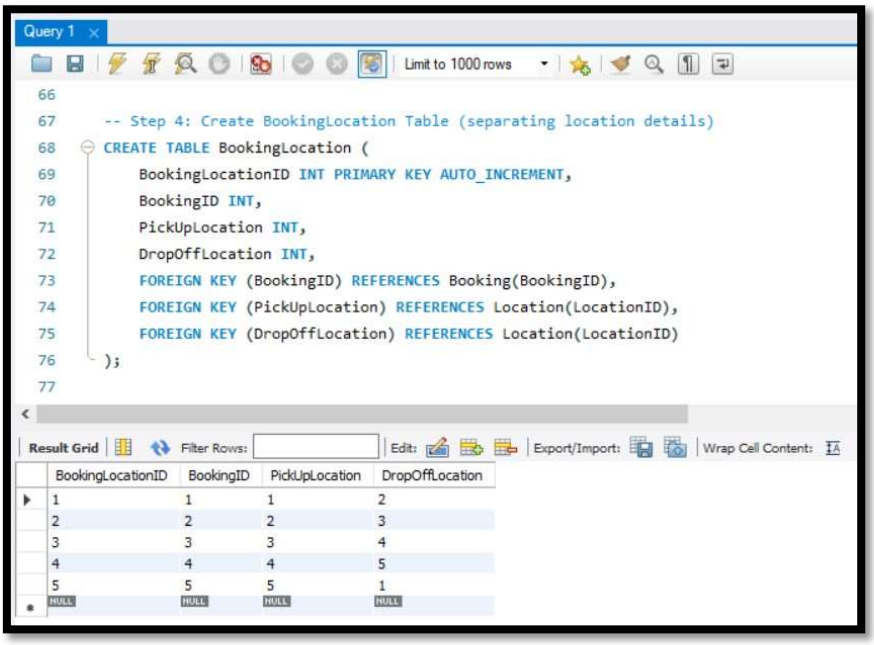
Problem in 2NF:

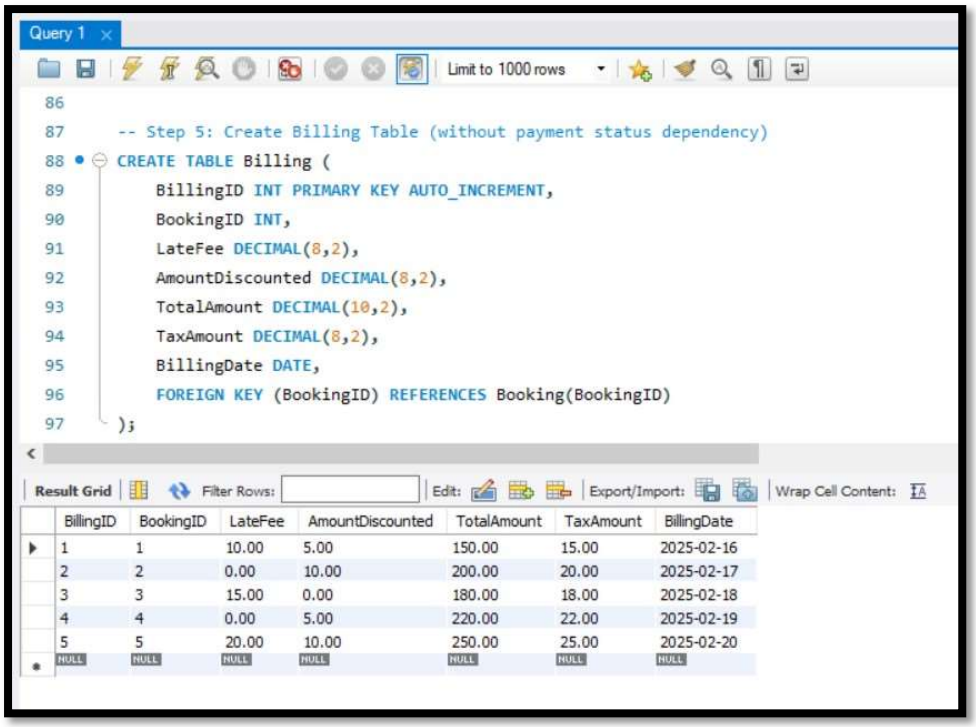
- **BrandName** in **Cars** is dependent on **BrandID**, not CarID.
- **City and State** in **Customers** have a transitive dependency.

Solution:

- Move **Brands** to a separate **Brands table**.
- Move **City and State** to a **Cities table**.

Code and Output :

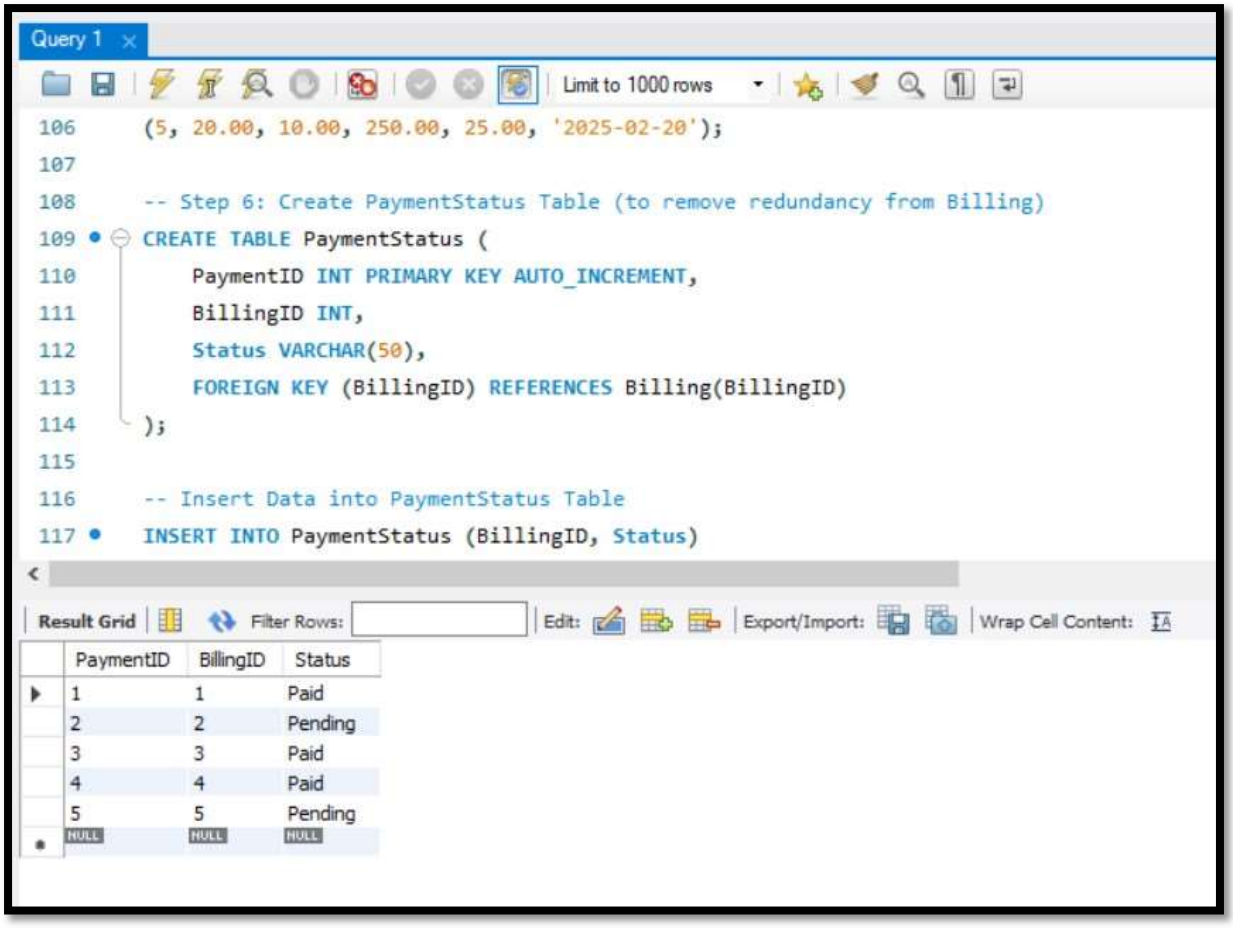




```
Query 1
Limit to 1000 rows

86
87      -- Step 5: Create Billing Table (without payment status dependency)
88  •   CREATE TABLE Billing (
89          BillingID INT PRIMARY KEY AUTO_INCREMENT,
90          BookingID INT,
91          LateFee DECIMAL(8,2),
92          AmountDiscounted DECIMAL(8,2),
93          TotalAmount DECIMAL(10,2),
94          TaxAmount DECIMAL(8,2),
95          BillingDate DATE,
96          FOREIGN KEY (BookingID) REFERENCES Booking(BookingID)
97      );
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

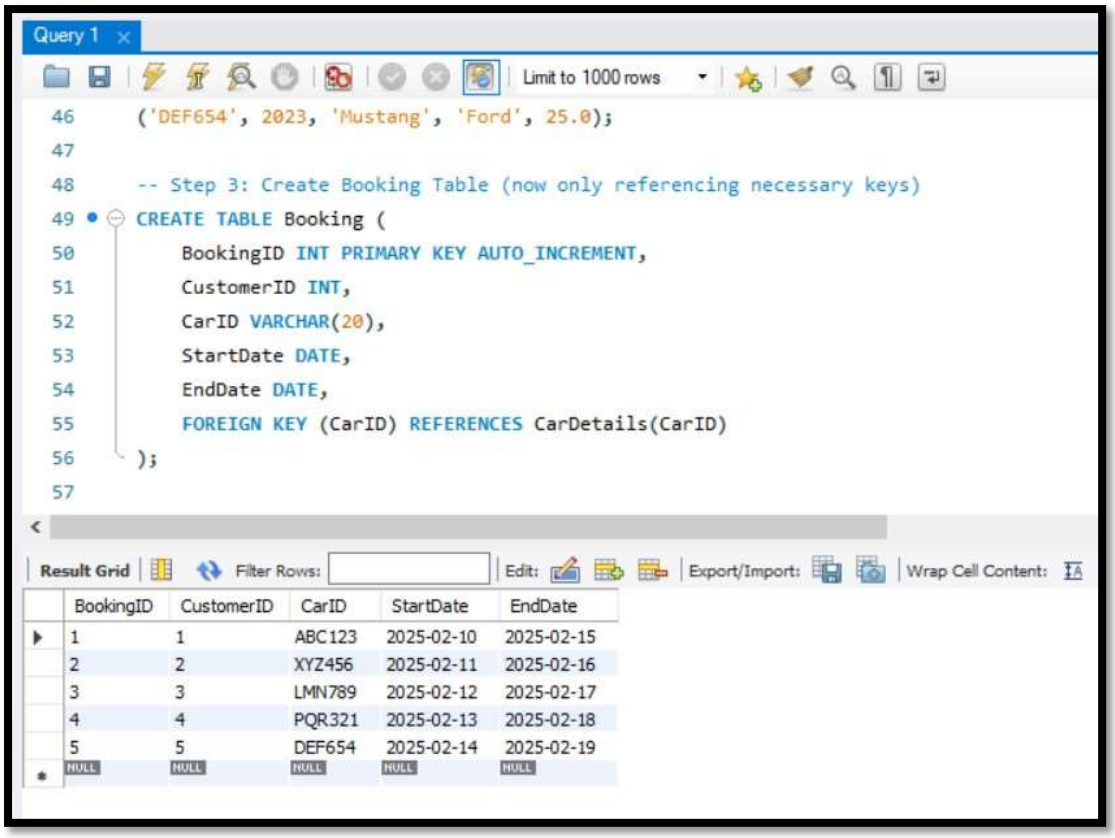
BillingID	BookingID	LateFee	AmountDiscounted	TotalAmount	TaxAmount	BillingDate
1	1	10.00	5.00	150.00	15.00	2025-02-16
2	2	0.00	10.00	200.00	20.00	2025-02-17
3	3	15.00	0.00	180.00	18.00	2025-02-18
4	4	0.00	5.00	220.00	22.00	2025-02-19
5	5	20.00	10.00	250.00	25.00	2025-02-20
NULL	NULL	NULL	NULL	NULL	NULL	NULL

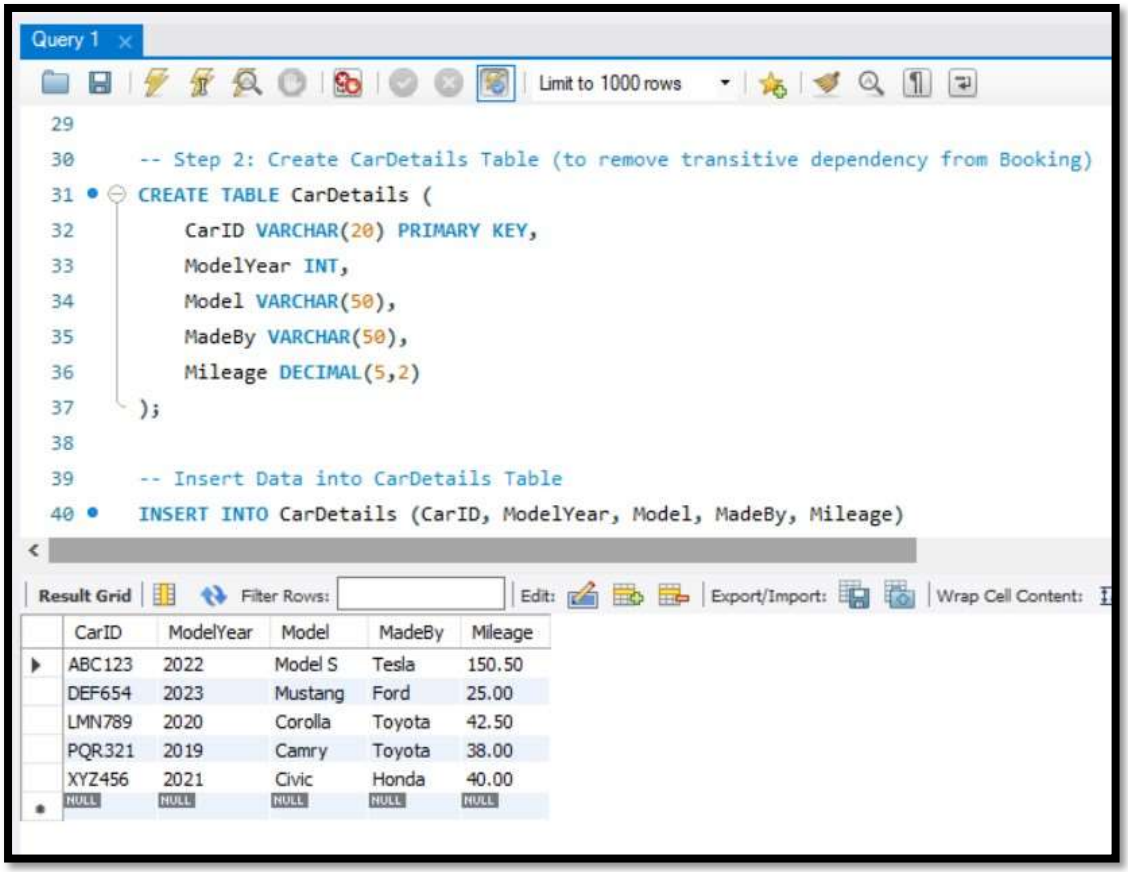


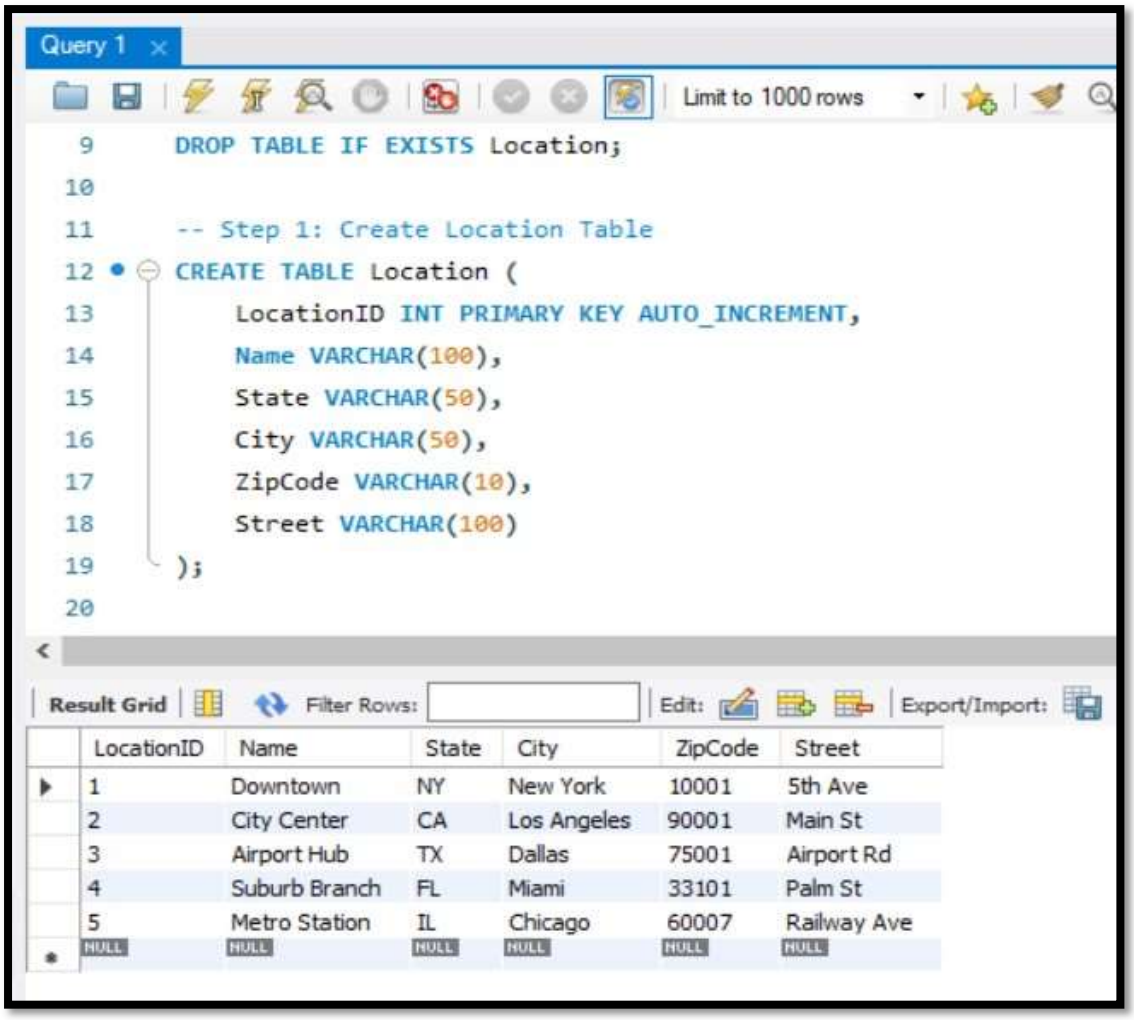
```
Query 1
Limit to 1000 rows

106         (5, 20.00, 10.00, 250.00, 25.00, '2025-02-20');
107
108     -- Step 6: Create PaymentStatus Table (to remove redundancy from Billing)
109 •   CREATE TABLE PaymentStatus (
110         PaymentID INT PRIMARY KEY AUTO_INCREMENT,
111         BillingID INT,
112         Status VARCHAR(50),
113         FOREIGN KEY (BillingID) REFERENCES Billing(BillingID)
114     );
115
116     -- Insert Data into PaymentStatus Table
117 •   INSERT INTO PaymentStatus (BillingID, Status)
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

PaymentID	BillingID	Status
1	1	Paid
2	2	Pending
3	3	Paid
4	4	Paid
5	5	Pending
NULL	NULL	NULL


Query 1
Limit to 1000 rows
('DEF654', 2023, 'Mustang', 'Ford', 25.0);
-- Step 3: Create Booking Table (now only referencing necessary keys)
CREATE TABLE Booking (
BookingID INT PRIMARY KEY AUTO_INCREMENT,
CustomerID INT,
CarID VARCHAR(20),
StartDate DATE,
EndDate DATE,
FOREIGN KEY (CarID) REFERENCES CarDetails(CarID)
);
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
BookingID CustomerID CarID StartDate EndDate
1 1 ABC123 2025-02-10 2025-02-15
2 2 XYZ456 2025-02-11 2025-02-16
3 3 LMN789 2025-02-12 2025-02-17
4 4 PQR321 2025-02-13 2025-02-18
5 5 DEF654 2025-02-14 2025-02-19
NULL NULL NULL NULL NULL


Query 1
Limit to 1000 rows
-- Step 2: Create CarDetails Table (to remove transitive dependency from Booking)
CREATE TABLE CarDetails (
CarID VARCHAR(20) PRIMARY KEY,
ModelYear INT,
Model VARCHAR(50),
MadeBy VARCHAR(50),
Mileage DECIMAL(5,2)
);
-- Insert Data into CarDetails Table
INSERT INTO CarDetails (CarID, ModelYear, Model, MadeBy, Mileage)
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
CarID ModelYear Model MadeBy Mileage
ABC123 2022 Model S Tesla 150.50
DEF654 2023 Mustang Ford 25.00
LMN789 2020 Corolla Toyota 42.50
PQR321 2019 Camry Toyota 38.00
XYZ456 2021 Civic Honda 40.00
NULL NULL NULL NULL NULL

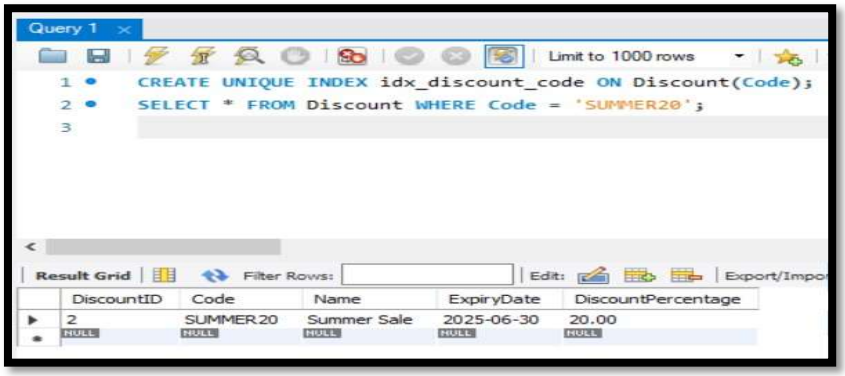

Query 1
Limit to 1000 rows
9 DROP TABLE IF EXISTS Location;
10
11 -- Step 1: Create Location Table
12 CREATE TABLE Location (
13 LocationID INT PRIMARY KEY AUTO_INCREMENT,
14 Name VARCHAR(100),
15 State VARCHAR(50),
16 City VARCHAR(50),
17 ZipCode VARCHAR(10),
18 Street VARCHAR(100)
19);
20
Result Grid
Filter Rows:
Edit:
Export/Import:
LocationID Name State City ZipCode Street
1 Downtown NY New York 10001 5th Ave
2 City Center CA Los Angeles 90001 Main St
3 Airport Hub TX Dallas 75001 Airport Rd
4 Suburb Branch FL Miami 33101 Palm St
5 Metro Station IL Chicago 60007 Railway Ave
NULL NULL NULL NULL NULL NULL

Indexing

5.1. Index on Code in the Discount Table

- **Reasoning:**
  A unique index on Code ensures that each discount code remains distinct, preventing duplicate entries and maintaining data integrity in the discount system.

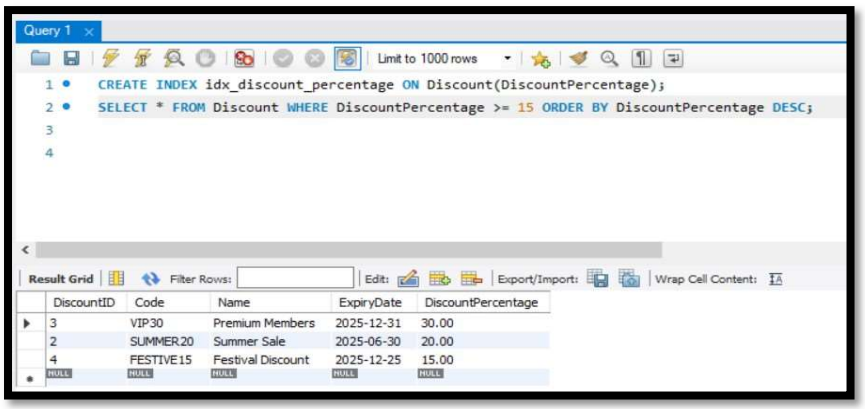
- **Benefits:**
  This improves the efficiency of searching for discount codes when applying promotional offers, ensuring fast lookups for queries.



5.2. Index on DiscountPercentage in the Discount Table

- **Reasoning:**
  Indexing the DiscountPercentage column optimizes queries that filter or sort discounts based on their percentage value, such as identifying the best available discounts for a customer.

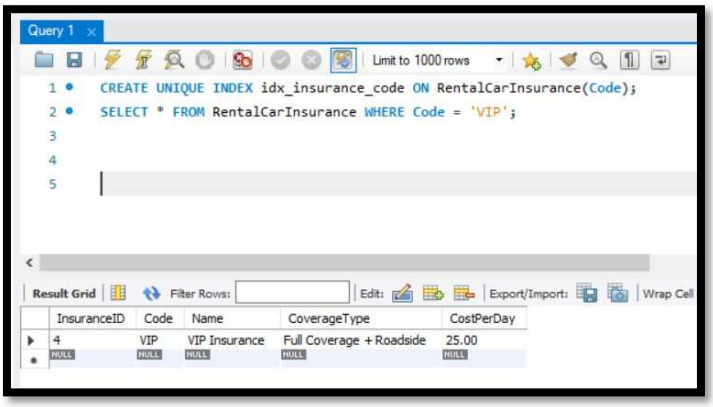
- **Benefits:**
  This enhances the speed of queries like which helps in listing active discount offers efficiently.



5.3. Index on Code in the RentalCarInsurance Table

- **Reasoning:**
  The Code column acts as a unique identifier for insurance plans, and indexing it ensures quick retrieval of insurance details when customers select a coverage plan.

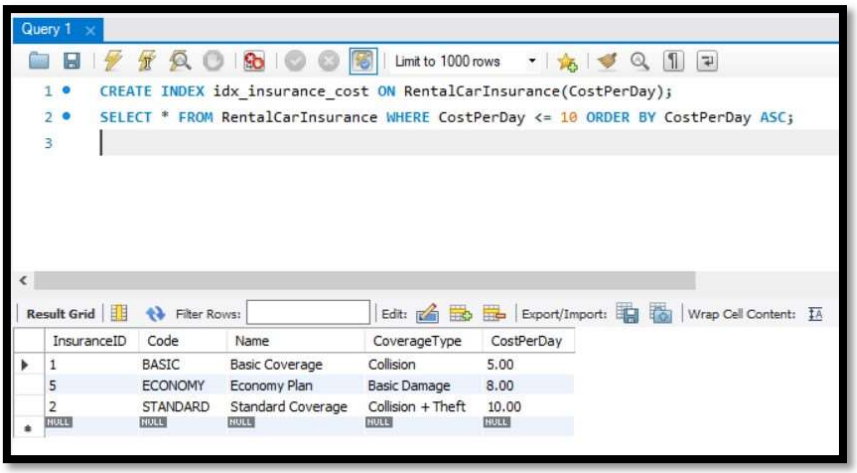
- **Benefits:**
  Queries will execute faster, reducing response time during the booking process.



5.4. Index on CostPerDay in the RentalCarInsurance Table

- **Reasoning:**
  This index allows for optimized filtering and sorting of insurance plans based on cost, enabling quick access to budget-friendly or premium insurance options.

- **Benefits:**
  Queries will run faster, improving the efficiency of cost-based searches in the system.



Views

A **view** in SQL is a **virtual table** that is derived from the result of a SQL query. Unlike physical tables, a view does not store data on disk; instead, it dynamically presents data from one or more underlying tables whenever it is queried. Views provide a way to simplify complex queries, enhance security, and organize data efficiently without duplicating storage.

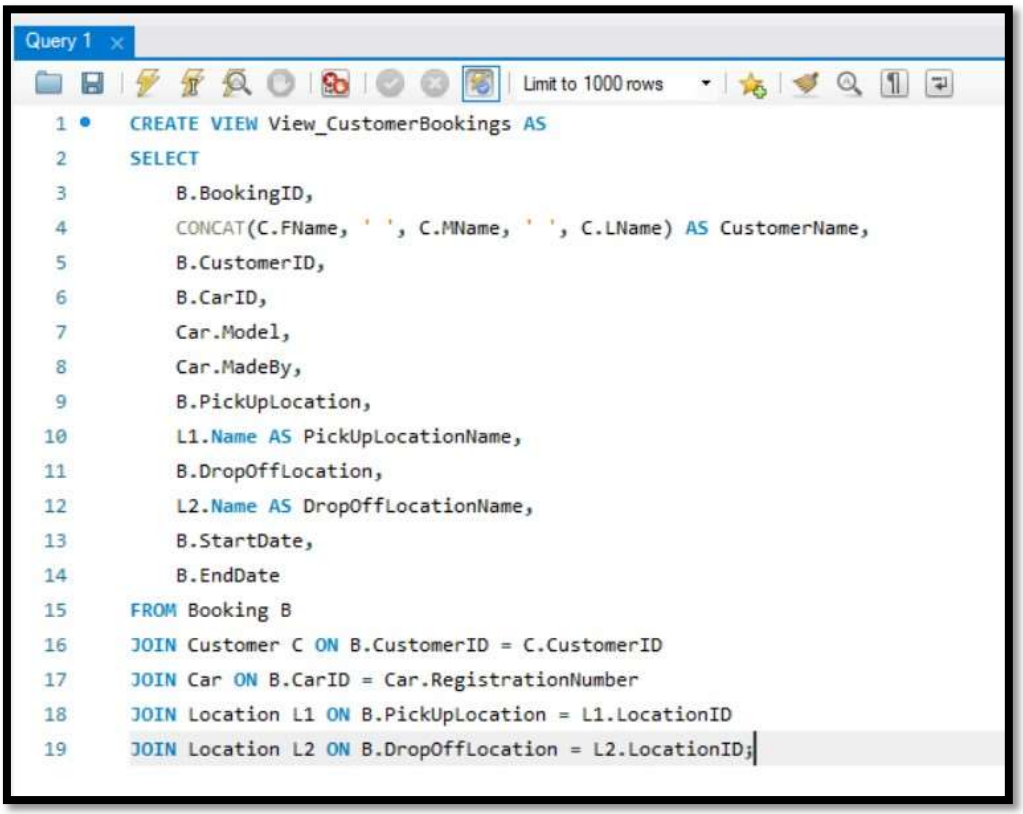
6.1. Creation and Explanation of View_CustomerBookings

The View_CustomerBookings is created to provide a simplified and structured representation of customer booking details by combining data from multiple tables (Booking, Customer, Car, and Location).

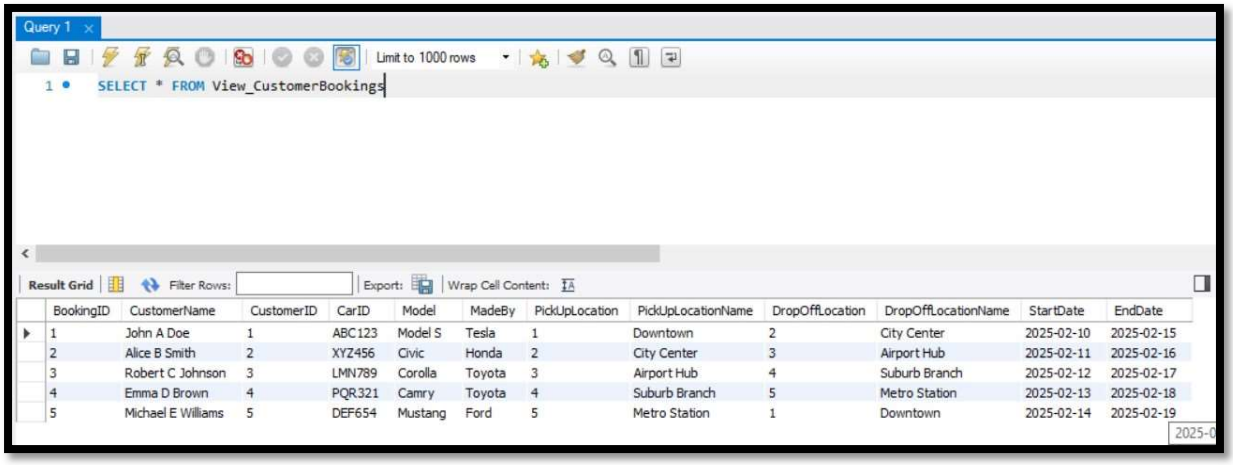
This view retrieves key details, including:

- Booking Information (Booking ID, Start Date, End Date)
- Customer Information (Concatenated Full Name)
- Car Details (Car Model, Manufacturer)
- Pickup and Drop-off Locations (Including location names)

Code :



Output :



BookingID	CustomerName	CustomerID	CarID	Model	MadeBy	PickUpLocation	PickUpLocationName	DropOffLocation	DropOffLocationName	StartDate	EndDate
1	John A Doe	1	ABC123	Model S	Tesla	1	Downtown	2	City Center	2025-02-10	2025-02-15
2	Alice B Smith	2	XYZ456	Civic	Honda	2	City Center	3	Airport Hub	2025-02-11	2025-02-16
3	Robert C Johnson	3	LMN789	Corolla	Toyota	3	Airport Hub	4	Suburb Branch	2025-02-12	2025-02-17
4	Emma D Brown	4	PQR321	Camry	Toyota	4	Suburb Branch	5	Metro Station	2025-02-13	2025-02-18
5	Michael E Williams	5	DEF654	Mustang	Ford	5	Metro Station	1	Downtown	2025-02-14	2025-02-19

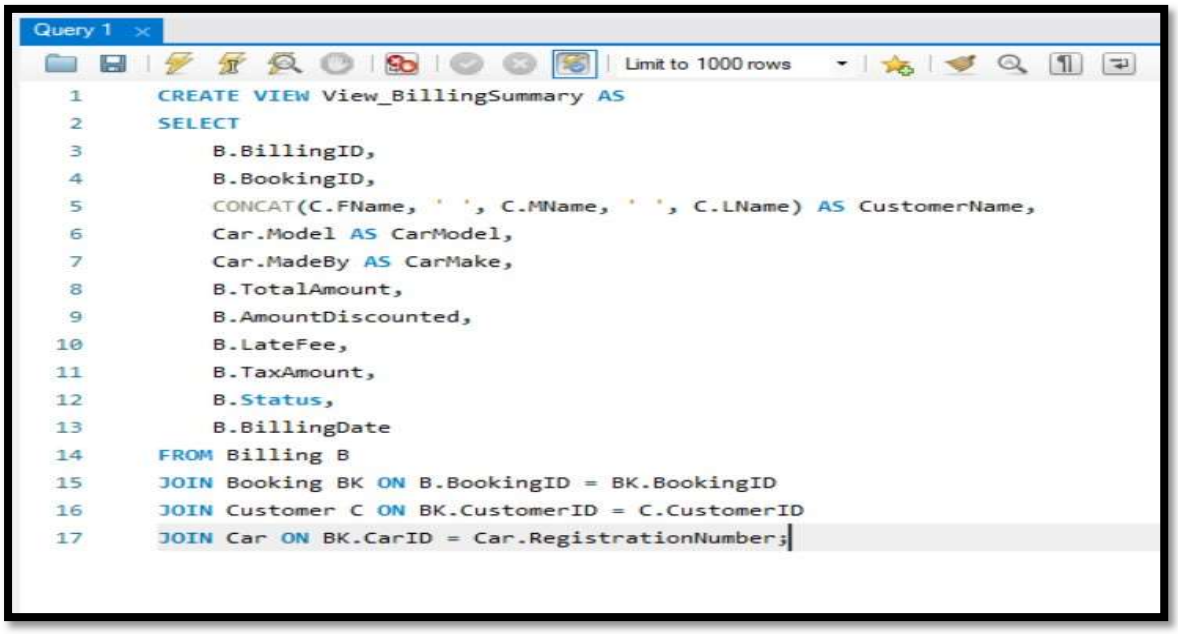
6.2. Creation and Explanation of View_BillingSummary

The View_BillingSummary is designed to present a concise and structured summary of billing transactions in the car rental system. This view extracts and combines essential billing-related details from multiple tables (Billing, Booking, Customer, and Car) to provide a comprehensive financial overview.

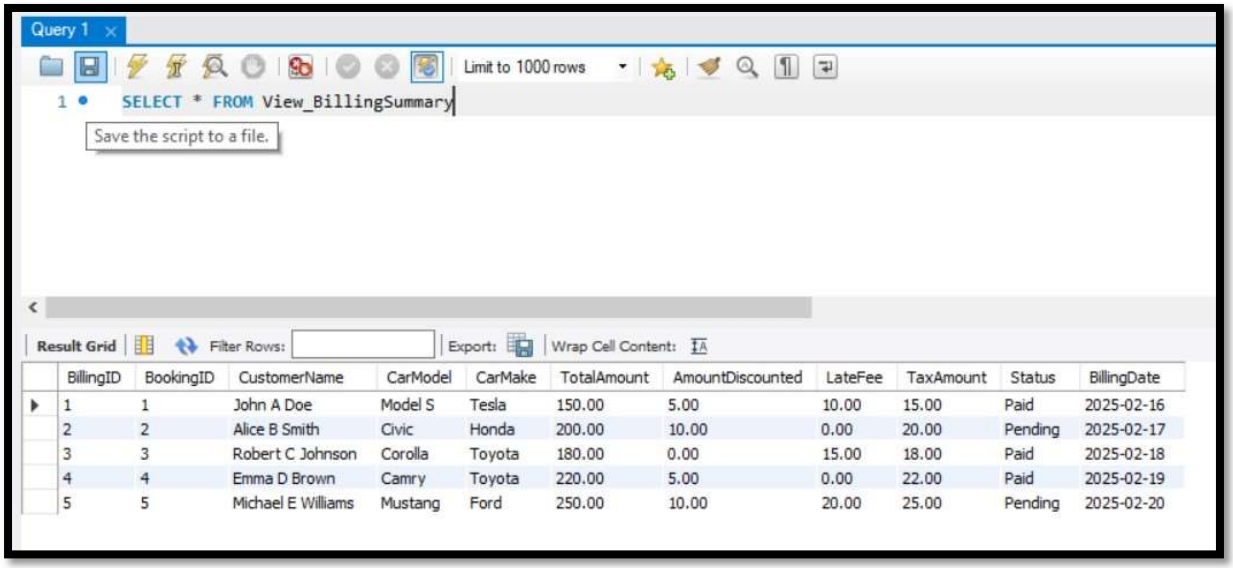
This view retrieves key billing details, including:

- Billing and Booking Information (Billing ID, Booking ID, Billing Date, Status)
- Customer Information (Concatenated Full Name)
- Car Details (Car Model and Manufacturer)
- Financial Details (Total Amount, Discount Applied, Late Fees, Taxes)

Code :



Output :



BillingID	BookingID	CustomerName	CarModel	CarMake	TotalAmount	AmountDiscounted	LateFee	TaxAmount	Status	BillingDate
1	1	John A Doe	Model S	Tesla	150.00	5.00	10.00	15.00	Paid	2025-02-16
2	2	Alice B Smith	Civic	Honda	200.00	10.00	0.00	20.00	Pending	2025-02-17
3	3	Robert C Johnson	Corolla	Toyota	180.00	0.00	15.00	18.00	Paid	2025-02-18
4	4	Emma D Brown	Camry	Toyota	220.00	5.00	0.00	22.00	Paid	2025-02-19
5	5	Michael E Williams	Mustang	Ford	250.00	10.00	20.00	25.00	Pending	2025-02-20

Procedures and Triggers

6.3. Definition of Procedures in SQL

A **Stored Procedure** in SQL is a precompiled collection of one or more SQL statements that are stored in the database and can be executed as a single unit. It allows users to encapsulate logic, making database operations more efficient and reusable.

Key Features of Stored Procedures:

1. **Encapsulation:** Procedures group multiple SQL queries into a single unit.
2. **Reusability:** Can be called multiple times, reducing code duplication.
3. **Improved Performance:** Since procedures are precompiled, execution is faster than running individual queries separately.
4. **Security:** Can be granted execution privileges without exposing table structures.
5. **Parameterization:** Accepts input and output parameters, making queries dynamic.

6.3.1. Purpose of GetCustomerBookings Procedure

The procedure is created to **fetch all booking details** for a given customer from the Booking table. This helps in quickly retrieving customer-specific booking history without writing repetitive queries.

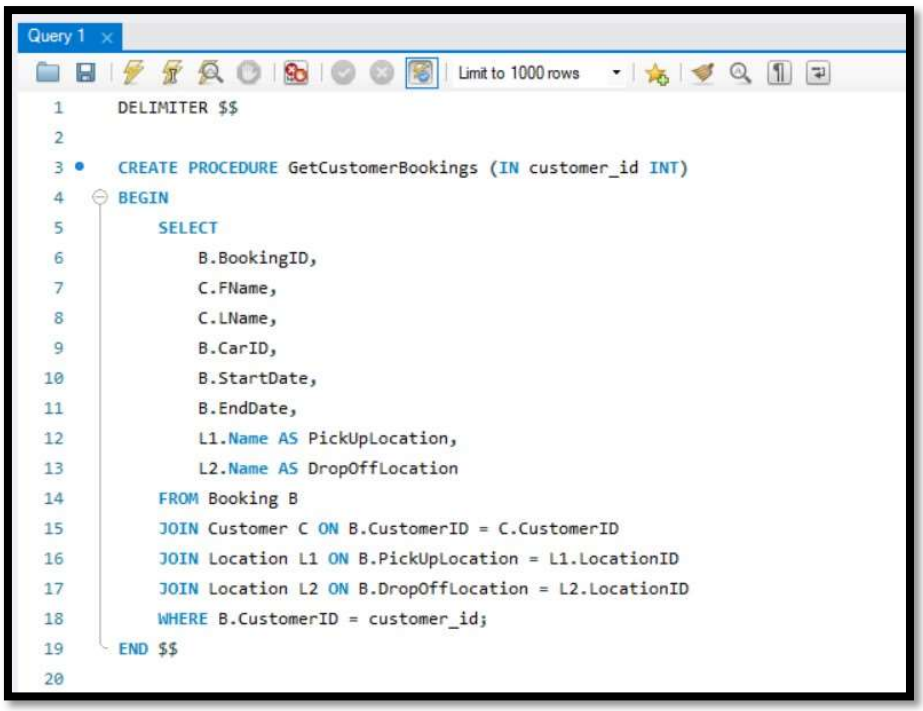
Explanation of Each Component:

1. **Input Parameter:**
    - IN customer_id INT: Takes a customer's ID as an input to filter bookings for that specific individual.
2. **Selected Columns:**
    - B.BookingID: Retrieves the booking ID.
    - C.FName, C.LName: Fetches the first and last name of the customer.
    - B.CarID: Shows the car associated with the booking.
    - B.StartDate, B.EndDate: Displays the start and end dates of the rental period.
    - L1.Name AS PickUpLocation: Retrieves the name of the pickup location.
    - L2.Name AS DropOffLocation: Retrieves the name of the drop-off location.
3. **Joins Used:**
    - JOIN Customer C ON B.CustomerID = C.CustomerID → To get customer details.
    - JOIN Location L1 ON B.PickUpLocation = L1.LocationID → To get pickup location name.

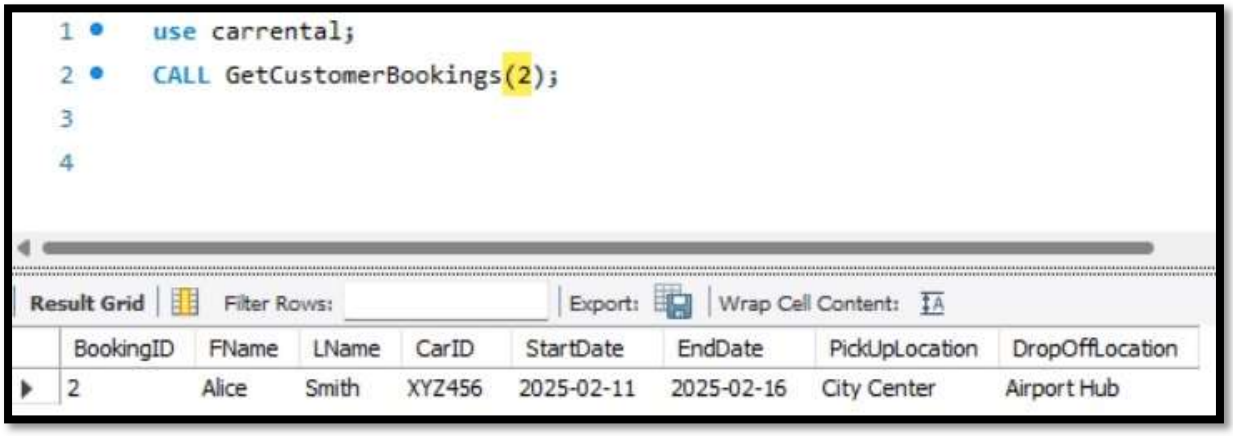
- JOIN Location L2 ON B.DropOffLocation = L2.LocationID → To get drop-off location name.

4. **Filtering Condition:**
   - WHERE B.CustomerID = customer_id → Ensures that only bookings belonging to the specified customer are retrieved.

Code :



Output :



6.3.2. Purpose of ApplyDiscount Procedure

This procedure applies a discount to a booking by retrieving the discount percentage based on a provided discount code. If the discount is valid (i.e., it has not expired), the procedure calculates the discount amount and updates the Billing table accordingly.

Explanation of Each Component:

1. **Retrieve the Discount Percentage:**
    - It checks the Discount table for the provided discount_code and ensures that the discount is still valid (ExpiryDate >= CURDATE()).
    - The discount percentage is stored in the variable discount_value.
2. **Fetch the Original Total Amount:**
    - The procedure retrieves the TotalAmount from the Billing table for the given booking_id.
3. **Update the Billing Table:**
    - The AmountDiscounted column is updated with the calculated discount.
    - The TotalAmount is updated by subtracting the discount.

Code :

```
CREATE PROCEDURE ApplyDiscount (IN booking_id INT, IN discount_code VARCHAR(20))
BEGIN
    DECLARE discount_value DECIMAL(5,2);
    DECLARE original_total DECIMAL(10,2);
    DECLARE discount_amount DECIMAL(10,2);

    -- Get discount percentage
    SELECT DiscountPercentage INTO discount_value
    FROM Discount
    WHERE Code = discount_code AND ExpiryDate >= CURDATE()
    LIMIT 1;

    -- If discount exists, apply it
    IF discount_value IS NOT NULL THEN
        -- Fetch the original total amount
        SELECT TotalAmount INTO original_total
        FROM Billing
        WHERE BookingID = booking_id;

        -- Calculate the discount amount
        SET discount_amount = (original_total * discount_value) / 100;

        -- Update the Billing Table
        UPDATE Billing
        SET AmountDiscounted = discount_amount,
            TotalAmount = original_total - discount_amount
        WHERE BookingID = booking_id;
    END IF;
```

Output :

```
use carrental;
CALL ApplyDiscount(1,'WELCOME10');

SELECT * FROM Billing WHERE BookingID = 1;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell

BillingID	BookingID	LateFee	AmountDiscounted	TotalAmount	TaxAmount	Status	BillingDate
1	1	10.00	13.50	121.50	15.00	Paid	2025-02-16
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

6.4. Definition of Triggers in SQL

A **trigger** is a special type of stored procedure that is automatically executed in response to specific events occurring in a table. Triggers are used to enforce business rules, maintain data integrity, and automate processes without manual intervention.

Types of Triggers:

1. **BEFORE Triggers** – Executed before an INSERT, UPDATE, or DELETE operation occurs.
2. **AFTER Triggers** – Executed after an INSERT, UPDATE, or DELETE operation is completed.
3. **INSTEAD OF Triggers** – Used mainly in views, replacing an INSERT, UPDATE, or DELETE operation with a custom operation.

6.4.1. Explanation of the Trigger before_billing_insert

This trigger is designed to **automatically update** the TotalAmount field **before inserting** a new record into the Billing table.

Functionality:

- It ensures that the TotalAmount value is correctly calculated by applying the tax and discount before the record is inserted.
- The NEW keyword refers to the new row being inserted into the Billing table.

Purpose & Benefits:

1. **Ensures Data Consistency** – Prevents errors where the final amount does not include tax or discount.
2. **Automates Calculation** – Reduces manual effort in computing the total amount for each transaction.

3. **Enhances Data Integrity** – Ensures every inserted billing record follows a uniform calculation rule.

Code :

```
DELIMITER $$

CREATE TRIGGER before_billing_insert
BEFORE INSERT ON Billing
FOR EACH ROW
BEGIN
    -- Calculate the final total amount before inserting the record
    SET NEW.TotalAmount = (NEW.TotalAmount + NEW.TaxAmount - NEW.AmountDiscounted);
END $$

DELIMITER ;
```

Output :

```
use carrental;
INSERT INTO Billing (BookingID, LateFee, AmountDiscounted, TotalAmount, TaxAmount, Status, BillingDate)
VALUES (5, 10.00, 5.00, 150.00, 15.00, 'Pending', '2025-02-21');
SELECT * FROM Billing WHERE BookingID = 5;
```

BillingID	BookingID	LateFee	AmountDiscounted	TotalAmount	TaxAmount	Status	BillingDate
5	5	20.00	10.00	250.00	25.00	Pending	2025-02-20
7	5	10.00	5.00	160.00	15.00	Pending	2025-02-21
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

6.4.2. Explanation of the Trigger after_booking_insert

This trigger is designed to **automatically update the availability status** of a car **after a new booking is inserted** into the Booking table.

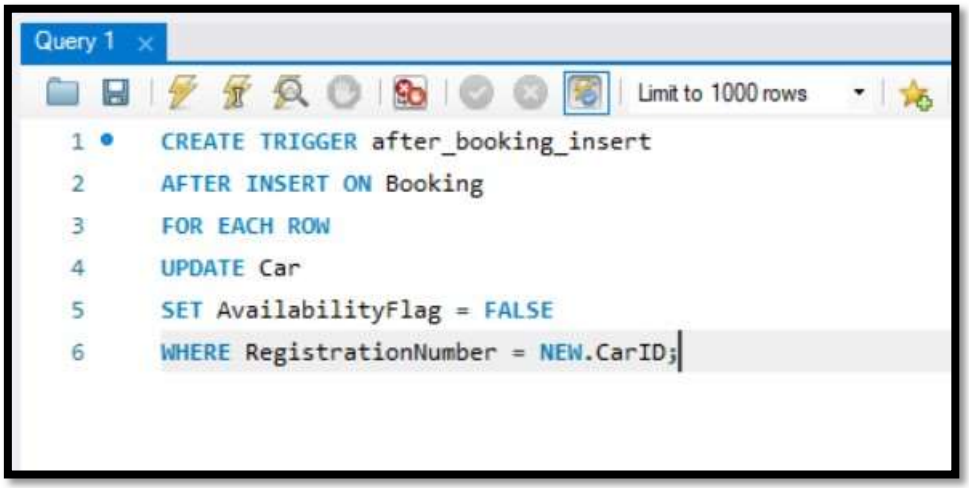
Functionality:

- The trigger **executes after a new booking is added** to the Booking table.
- It updates the **AvailabilityFlag** of the car associated with the new booking by setting it to FALSE, indicating that the car is no longer available for other bookings.
- The NEW keyword refers to the **newly inserted row** in the Booking table.
- The update operation targets the Car table, where it finds the car **using its RegistrationNumber** and updates its status.

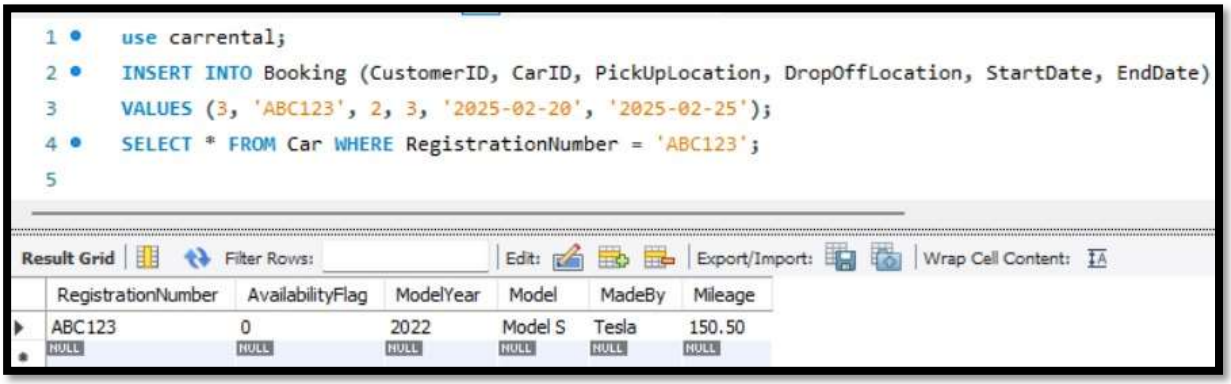
Purpose & Benefits:

- **Maintains Car Availability Status** – Ensures that once a car is booked, it is marked as unavailable, preventing double bookings.
- **Automates Inventory Management** – No manual intervention is needed to update car availability, reducing errors.
- **Enhances System Reliability** – Guarantees data consistency by enforcing business rules on car availability.

Code :



Output :



Conclusion

The development of the Car Rental Service database ensures an efficient, reliable, and scalable system for managing customers, vehicles, bookings, and financial transactions. By leveraging relational database principles, we have designed a system that maintains data integrity, enforces consistency, and optimizes query performance.

Through the process of normalization, we have refined the database schema to eliminate redundancy and ensure efficient data storage. Additionally, by adhering to Codd's 12 Rules, the system is fully relational and supports structured data access, logical independence, and robust security measures.

To further enhance performance, we implemented **indexing**, which improves query efficiency, particularly for frequent searches on primary keys and foreign keys. Additionally, **stored procedures** have been utilized to automate complex operations such as booking management and billing calculations, ensuring consistency and reducing redundancy. **Triggers** have been introduced to enforce business rules, such as automatically updating vehicle availability upon booking confirmation or applying late fees when a car is returned past the due date.

With these enhancements, the Car Rental Service database is optimized for high performance, seamless operations, and robust data integrity. Future improvements may include integrating machine learning for demand forecasting, dynamic pricing strategies, and a more advanced reporting system for business insights. This structured and well-optimized database serves as a strong foundation for the continuous growth and efficiency of the car rental service.