Q.1) Car Rental Management System Description: Develop a database to manage car rentals, customer details, car availability, and billing.

-> The **Car Rental Management System** database includes tables for storing customer details, car inventory, rentals, billing, and additional services. It supports processes to check car availability, register rentals, and handle returns. Triggers automatically update car availability, and queries analyze usage, popular services, and revenue by car type.

### 1. Database Tables Design

**Table: Customers**

**Customers**: This table stores detailed customer information, including name, contact details, and address, allowing the rental agency to manage and retrieve customer profiles efficiently. It supports customer identification during rental transactions, billing, and customer service interactions.

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| customer\_id | INT (PK) | Unique identifier for each customer |
| first\_name | VARCHAR(50) | Customer's first name |
| last\_name | VARCHAR(50) | Customer's last name |
| phone\_number | VARCHAR(15) | Customer's contact number |
| email | VARCHAR(100) | Customer's email address |
| address | VARCHAR(255) | Customer's address |

**Table: Cars**

**Cars**: The Cars table maintains a record of all available vehicles, including attributes like make, model, type, daily rental rate, and availability status. This table helps manage the car inventory, track car types, and set rental rates, ensuring accurate availability for rentals.

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| car\_id | INT (PK) | Unique identifier for each car |
| make | VARCHAR(50) | Car manufacturer |
| model | VARCHAR(50) | Car model |
| year | INT | Year of manufacture |
| car\_type | VARCHAR(50) | Type (e.g., sedan, SUV) |
| available | BOOLEAN | Availability status |
| rate\_per\_day | DECIMAL(10,2) | Daily rental rate |

**Table: Rentals**

**Rentals**: This table logs all rental transactions, connecting customers with rented cars and tracking rental details such as pickup and drop-off dates. It allows the agency to monitor active rentals, calculate rental durations, and prepare billing records based on the rental period.

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| rental\_id | INT (PK) | Unique identifier for each rental |
| customer\_id | INT (FK) | References Customers |
| car\_id | INT (FK) | References Cars |
| pickup\_date | DATE | Rental start date |
| dropoff\_date | DATE | Rental end date |
| total\_amount | DECIMAL(10,2) | Total rental cost |

**Table: Billing**

**Billing**: The Billing table manages payment records for each rental, storing information about the billing date, amount, and payment status (paid or unpaid). It supports financial tracking, enabling the agency to manage outstanding balances and streamline payment processing for each customer.

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| billing\_id | INT (PK) | Unique billing record |
| rental\_id | INT (FK) | References Rentals |
| billing\_date | DATE | Date of billing |
| amount | DECIMAL(10,2) | Amount billed |
| payment\_status | VARCHAR(20) | Status of payment (e.g., paid/unpaid) |

**Table: Additional\_Services**

**Additional\_Services**: This table lists extra services (e.g., GPS, insurance) that customers can add to their rentals. It includes service names, descriptions, and rates, allowing the agency to offer and bill for add-ons, enhancing customer service options.

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| service\_id | INT (PK) | Unique identifier for each service |
| service\_name | VARCHAR(50) | Name of service (e.g., GPS) |
| service\_rate | DECIMAL(10,2) | Rate for the service |
| description | VARCHAR(255) | Service description |

**Table: Rental\_Services**

**Rental\_Services**: This table links Rentals to Additional\_Services, enabling the tracking of specific services added to each rental. It allows for detailed billing and reporting on service usage, making it easier to identify popular add-on services.

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| rental\_id | INT (FK) | References Rentals |
| service\_id | INT (FK) | References Additional\_Services |

### 2. Constraints for Referential Integrity

* **Foreign Keys**: Enforce relationships:
  + customer\_id in Rentals references Customers(customer\_id)
  + car\_id in Rentals references Cars(car\_id)
  + rental\_id in Billing references Rentals(rental\_id)
  + rental\_id in Rental\_Services references Rentals(rental\_id)
  + service\_id in Rental\_Services references Additional\_Services(service\_id)
* **Primary Keys**: Each table has a primary key for unique identification.
* **Check Constraints**: Ensure data consistency, such as non-negative rates and valid dates.

### 3. Stored Procedures

#### a. Check Car Availability

#### **Check Car Availability**: A process to confirm if a specific car is available for the desired rental dates, factoring in existing rentals. This helps prevent booking conflicts and ensures that cars are rented only when available.

sql

Copy code

CREATE PROCEDURE CheckCarAvailability(IN carId INT, IN checkDate DATE)

BEGIN

DECLARE isAvailable BOOLEAN;

SELECT available INTO isAvailable FROM Cars WHERE car\_id = carId;

IF isAvailable THEN

SELECT 'Car is available';

ELSE

SELECT 'Car is not available';

END IF;

END;

#### b. Make a Rental

#### **Make Rental**: This process registers a new rental, creating records in the Rentals and Billing tables. It also updates the car’s availability status, marking it as unavailable until the rental is complete.

sql

Copy code

CREATE PROCEDURE MakeRental(IN custId INT, IN carId INT, IN pickup DATE, IN dropoff DATE)

BEGIN

DECLARE total DECIMAL(10,2);

DECLARE dailyRate DECIMAL(10,2);

SELECT rate\_per\_day INTO dailyRate FROM Cars WHERE car\_id = carId;

SET total = DATEDIFF(dropoff, pickup) \* dailyRate;

INSERT INTO Rentals (customer\_id, car\_id, pickup\_date, dropoff\_date, total\_amount)

VALUES (custId, carId, pickup, dropoff, total);

UPDATE Cars SET available = FALSE WHERE car\_id = carId;

END;

#### c. Process Return

#### **Process Return**: Finalizes a rental by calculating the total amount due, updating the billing status, and setting the car back to available. This ensures accurate billing and car availability management.

sql

Copy code

CREATE PROCEDURE ProcessReturn(IN rentalId INT)

BEGIN

DECLARE carId INT;

SELECT car\_id INTO carId FROM Rentals WHERE rental\_id = rentalId;

UPDATE Cars SET available = TRUE WHERE car\_id = carId;

END;

### 4. Triggers

#### a. Trigger on Rental Pickup (Update Car Availability)

#### **Triggers**: Automatic triggers adjust car availability when rentals are picked up or returned, ensuring real-time updates on car status.

sql

Copy code

CREATE TRIGGER BeforeRentalPickup

BEFORE INSERT ON Rentals

FOR EACH ROW

BEGIN

UPDATE Cars SET available = FALSE WHERE car\_id = NEW.car\_id;

END;

#### b. Trigger on Rental Drop-off (Update Car Availability)

**Queries**: SQL queries generate reports on car usage, most popular services, and revenue by car type, supporting data-driven decision-making and trend analysis.

sql

Copy code

CREATE TRIGGER AfterRentalDropoff

AFTER UPDATE ON Rentals

FOR EACH ROW

WHEN NEW.dropoff\_date IS NOT NULL

BEGIN

UPDATE Cars SET available = TRUE WHERE car\_id = NEW.car\_id;

END;

### 5. SQL Queries for Reports

#### a. Generate Report on Car Usage

sql

Copy code

SELECT car\_id, COUNT(rental\_id) AS usage\_count

FROM Rentals

GROUP BY car\_id

ORDER BY usage\_count DESC;

#### b. Generate Report on Popular Services

sql

Copy code

SELECT service\_name, COUNT(rental\_id) AS usage\_count

FROM Rental\_Services rs

JOIN Additional\_Services s ON rs.service\_id = s.service\_id

GROUP BY service\_name

ORDER BY usage\_count DESC;

#### c. Generate Revenue by Car Type

sql

Copy code

SELECT car\_type, SUM(total\_amount) AS total\_revenue

FROM Rentals r

JOIN Cars c ON r.car\_id = c.car\_id

GROUP BY car\_type

ORDER BY total\_revenue DESC;