**SAVEETHA SCHOOL OF ENGINEERING**

**CSA0580** **Database Management Systems for Query Optimization Techniques in Oracle**

**ASSIGNMENT**

**Automobile Rental Management System**

**Develop a database for an Automobile Rental Management System that manages vehicles, customers, rentals, and payment records. Create an ER diagram that represent relationships between vehicles, customers, rentals, and payments. Implement the schema in SQL and populate it with realistic data. Write SQL queries to analyze rental patterns, monitor vehicle usage, evaluate customer loyalty, and identify peak rental periods.**

**Submitted By**

**Registration No.: 192324088**

**Name: SHAIK ASHRAF**

**Department: ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**Description**

**Overview of System Objectives:**

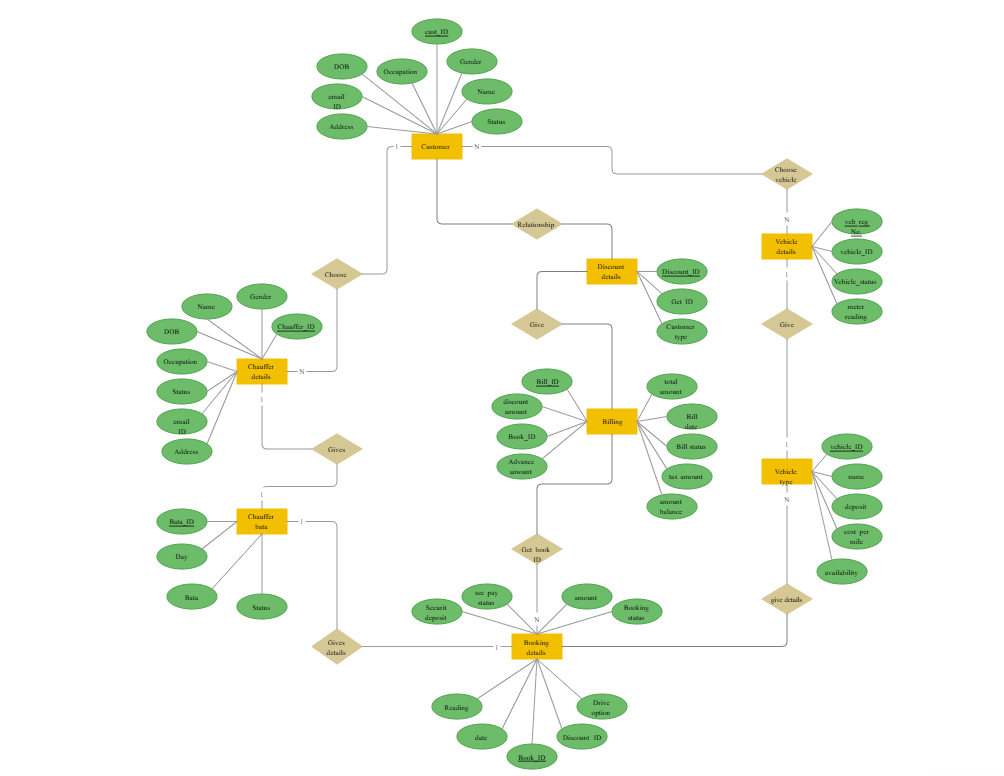
**To design an Automobile Rental Management System, we need to break it down into core entities (tables) that interact with each other. The key components of the system will include Vehicles, Customers, Rentals, and Payments. Here's a step-by-step guide to the process, starting from the ER diagram and proceeding with the SQL schema and queries.**

**System Objectives:**

1. **Vehicles:**
   * **Attributes: Vehicle ID, Make, Model, Year, Registration Number, Availability Status**
   * **Relationships:**
   * **One-to-many relationship with Rentals: A vehicle can be rented multiple times.**
2. **Customers:**
   * **Attributes: Customer ID, Name, Address, Contact Number, Driver's License Number**
   * **Relationships:**
   * **One-to-many relationship with Rentals: A customer can rent multiple vehicles.**
3. **Rentals:**
   * **Attributes: Rental ID, Vehicle ID, Customer ID, Start Date, End Date, Rental Fee**
   * **Relationships:**
   * **Many-to-one relationship with Vehicles: A rental involves one vehicle.**
   * **Many-to-one relationship with Customers: A rental involves one customer.**

* **Enhanced Decision-Making: Real-time access to performance metrics enables informed decisions regarding route adjustments, vehicle assignments, and schedule optimization.**
* **Scalability: The database structure is designed to accommodate growth, allowing for additional routes, vehicles, and passenger data as needed**

**ER Diagram Representation**



ENTITIES AND RELATIONSHIPS:

1. **Vehicles**

* VehicleID (Primary Key)
* Make (e.g., Toyota, Honda)
* Model (e.g., Corolla, Civic)
* Year (e.g., 2019)
* Mileage (Current mileage)
* Color (e.g., Red, Blue)
* Status (Available, Rented, Under Maintenance)

1. **Customers**

* CustomerID (Primary Key)
* FirstName
* LastName
* Email (Unique)
* Phone (Optional)
* Address (Optional)

1. **Rentals**
   * RentalID (Primary Key)
   * CustomerID (Foreign Key referencing Customers)
   * VehicleID (Foreign Key referencing Vehicles)
   * RentalDate (Date when rental starts)
   * ReturnDate (Date when rental ends)
   * TotalCost (Total charge for the rental)
   * RentalStatus (e.g., Active, Completed, Canceled)
2. **Payments**
   * PaymentID (Primary Key)
   * RentalID (Foreign Key referencing Rentals)
   * PaymentDate (Date when the payment was made)
   * Amount (Amount paid for the rental)
   * PaymentMethod (e.g., Credit Card, Cash)

**SQL Schema Implementation**

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY AUTO\_INCREMENT,

FirstName VARCHAR(100),

LastName VARCHAR(100),

Email VARCHAR(100) UNIQUE,

Phone VARCHAR(15),

Address VARCHAR(255)

);

-- Create Vehicles Table

CREATE TABLE Vehicles (

VehicleID INT PRIMARY KEY AUTO\_INCREMENT,

Make VARCHAR(50),

Model VARCHAR(50),

Year INT,

Mileage INT,

Color VARCHAR(20),

Status VARCHAR(20) -- e.g., Available, Rented, Under Maintenance

);

-- Create Rentals Table

CREATE TABLE Rentals (

RentalID INT PRIMARY KEY AUTO\_INCREMENT,

CustomerID INT,

VehicleID INT,

RentalDate DATE,

ReturnDate DATE,

TotalCost DECIMAL(10, 2),

RentalStatus VARCHAR(20), -- e.g., Active, Completed, Canceled

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

FOREIGN KEY (VehicleID) REFERENCES Vehicles(VehicleID)

);

-- Create Payments Table

CREATE TABLE Payments (

PaymentID INT PRIMARY KEY AUTO\_INCREMENT,

RentalID INT,

PaymentDate DATE,

Amount DECIMAL(10, 2),

PaymentMethod VARCHAR(50), -- e.g., Credit Card, Cash

FOREIGN KEY (RentalID) REFERENCES Rentals(RentalID)

);

**Sample Data Insertion**

Insert sample Customers

INSERT INTO Customers (FirstName, LastName, Email, Phone, Address) VALUES

('John', 'Doe', 'johndoe@example.com', '555-1234', '123 Elm Street, NY'),

('Jane', 'Smith', 'janesmith@example.com', '555-5678', '456 Oak Avenue, NY'),

('Mark', 'Johnson', 'markj@example.com', '555-8765', '789 Pine Road, NY');

-- Insert sample Vehicles

INSERT INTO Vehicles (Make, Model, Year, Mileage, Color, Status) VALUES

('Toyota', 'Corolla', 2019, 25000, 'Red', 'Available'),

('Honda', 'Civic', 2020, 15000, 'Blue', 'Available'),

('Ford', 'Escape', 2018, 30000, 'Black', 'Available');

-- Insert sample Rentals

INSERT INTO Rentals (CustomerID, VehicleID, RentalDate, ReturnDate, TotalCost, RentalStatus) VALUES

(1, 2, '2024-10-01', '2024-10-05', 100.00, 'Completed'),

(2, 1, '2024-10-02', '2024-10-06', 120.00, 'Completed'),

(3, 3, '2024-10-03', '2024-10-07', 150.00, 'Completed');

-- Insert sample Payments

INSERT INTO Payments (RentalID, PaymentDate, Amount, PaymentMethod) VALUES

(1, '2024-10-01', 100.00, 'Credit Card'),

(2, '2024-10-02', 120.00, 'Cash'),

(3, '2024-10-03', 150.00, 'Credit Card');

**SQL Queries to Analyze Rental Patterns**

SELECT v.Make, v.Model, COUNT(r.RentalID) AS RentalsCount

FROM Rentals r

JOIN Vehicles v ON r.VehicleID = v.VehicleID

GROUP BY v.VehicleID;

SELECT c.FirstName, c.LastName, COUNT(r.RentalID) AS RentalsCount

FROM Rentals r

JOIN Customers c ON r.CustomerID = c.CustomerID

GROUP BY c.CustomerID;

**Schedules Table**

The Schedules table records the departure and arrival times for each vehicle on a given route.

CREATE TABLE Schedules (

ScheduleID INT PRIMARY KEY,

VehicleID INT,

RouteID INT,

DepartureTime DATETIME,

ArrivalTime DATETIME,

FOREIGN KEY (VehicleID) REFERENCES Vehicles(VehicleID),

FOREIGN KEY (RouteID) REFERENCES Routes(RouteID)

);

**b) Monitor Vehicle Usage**

**To see how often each vehicle was rented in the past month:**

**sql**

**Copy code**

**Vehicle rental frequency in the past month**

**SELECT v.Make, v.Model, COUNT(r.RentalID) AS RentalsCount**

**FROM Rentals r**

**JOIN Vehicles v ON r.VehicleID = v.VehicleID**

**WHERE r.RentalDate >= DATE\_SUB(CURDATE(), INTERVAL 1 MONTH)**

**GROUP BY v.VehicleID;**

**c) Evaluate Customer Loyalty**

**To find the customers who rented the most:**

**sql**

**Copy code**

**-- Vehicle rental frequency in the past month**

**SELECT v.Make, v.Model, COUNT(r.RentalID) AS RentalsCount**

**FROM Rentals r**

**JOIN Vehicles v ON r.VehicleID = v.VehicleID**

**WHERE r.RentalDate >= DATE\_SUB(CURDATE(), INTERVAL 1 MONTH)**

**GROUP BY v.VehicleID;**

**d) Identify Peak Rental Periods**

**To identify peak rental periods (e.g., by month):**

**sql**

**Copy code**

**SELECT YEAR(r.RentalDate) AS RentalYear, MONTH(r.RentalDate) AS RentalMonth, COUNT(r.RentalID) AS RentalsCount**

**FROM Rentals r**

**GROUP BY YEAR(r.RentalDate), MONTH(r.RentalDate)**

**ORDER BY RentalsCount DESC;**

**e) Revenue Analysis**

**To calculate total revenue per vehicle:**

**sql**

**Copy code**

**SELECT v.Make, v.Model, SUM(r.TotalCost) AS TotalRevenue**

**FROM Rentals r**

**JOIN Vehicles v ON r.VehicleID = v.VehicleID**

**GROUP BY v.VehicleID**

**ORDER BY TotalRevenue DESC;**

**Summary of Entities**

1. **Vehicles: Information about the vehicles (e.g., make, model, mileage).**
2. **Customers: Information about the customers renting vehicles.**

Sample Data Insertion:

**Sample Data for Tables:**

Below are sample data insertion statements for the tables in a **Public Transport Management System**. This data provides a foundation for testing the database and running SQL queries on route performance, vehicle usage, and passenger load.

**Routes Table**

INSERT INTO Routes (RouteID, RouteName, Source, Destination, Distance) VALUES

(1, 'City Center to Airport', 'City Center', 'Airport', 25),

(2, 'Northside to Downtown', 'Northside', 'Downtown', 15),

(3, 'University Loop', 'University', 'University', 10);

Vehicles Table

INSERT INTO Vehicles (VehicleID, VehicleNumber, Capacity, VehicleType, RouteID) VALUES

(1, 'BUS101', 50, 'Bus', 1),

(2, 'BUS102', 45, 'Bus', 2),

(3, 'VAN201', 20, 'Van', 3);

Schedules Table

INSERT INTO Schedules (ScheduleID, VehicleID, RouteID, DepartureTime, ArrivalTime) VALUES

(1, 1, 1, '2023-11-06 08:00:00', '2023-11-06 08:30:00'),

(2, 2, 2, '2023-11-06 09:00:00', '2023-11-06 09:20:00'),

(3, 3, 3, '2023-11-06 10:00:00', '2023-11-06 10:15:00');

Passengers Table

INSERT INTO Passengers (PassengerID, FullName, Age, Gender, ContactNumber) VALUES

(1, 'Alice Johnson', 34, 'Female', '555-1234'),

(2, 'Michael Green', 28, 'Male', '555-2345'),

(3, 'Linda Martinez', 45, 'Female', '555-3456');

Bookings Table

INSERT INTO Bookings (BookingID, PassengerID, ScheduleID, BookingDate, NumberOfSeats) VALUES

(1, 1, 1, '2023-11-05', 1),

(2, 2, 2, '2023-11-05', 2),

(3, 3, 3, '2023-11-05', 1);

**SQL Queries and Expected Output**

**SQL Queries to Analyze Rental Patterns**

**SELECT v.Make, v.Model, COUNT(r.RentalID) AS RentalsCount**

**FROM Rentals r**

**JOIN Vehicles v ON r.VehicleID = v.VehicleID**

**GROUP BY v.VehicleID;**

****

**Monitor Vehicle Usage: Vehicle rental frequency in the past month**

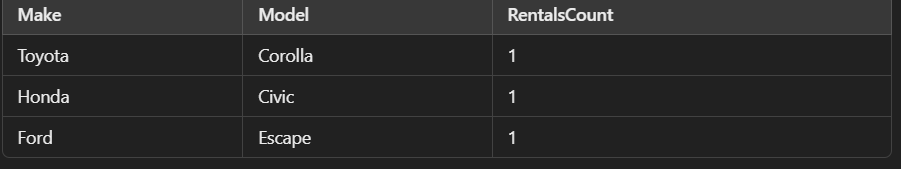
**SELECT v.Make, v.Model, COUNT(r.RentalID) AS RentalsCount**

**FROM Rentals r**

**JOIN Vehicles v ON r.VehicleID = v.VehicleID**

**WHERE r.RentalDate >= DATE\_SUB(CURDATE(), INTERVAL 1 MONTH)**

**GROUP BY v.VehicleID;**

****

**Evaluate Customer Loyalty: Customers with the most rentals**

**SELECT c.FirstName, c.LastName, COUNT(r.RentalID) AS RentalsCount**

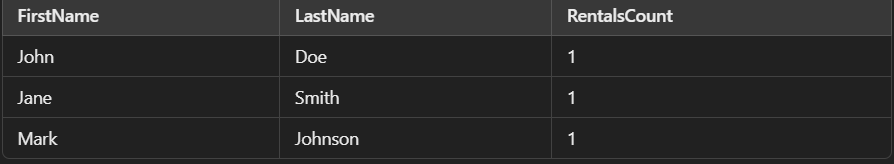
**FROM Rentals r**

**JOIN Customers c ON r.CustomerID = c.CustomerID**

**GROUP BY c.CustomerID**

**ORDER BY RentalsCount DESC**

**LIMIT 5;**

****

**Revenue Analysis: Total revenue per vehicle**

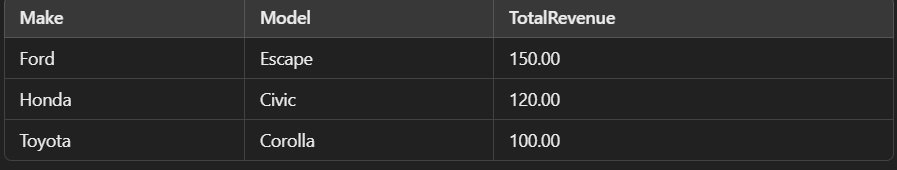
**SELECT v.Make, v.Model, SUM(r.TotalCost) AS TotalRevenue**

**FROM Rentals r**

**JOIN Vehicles v ON r.VehicleID = v.VehicleID**

**GROUP BY v.VehicleID**

**ORDER BY TotalRevenue DESC;**

****

**Indexing**

**Indexing is a critical technique for optimizing query performance by reducing the amount of data scanned during query execution. In a Public Transport Management System, effective indexing ensures that searches, joins, and filtering operations are faster.**

* **Why Indexing is Important: Indexing allows the database to quickly locate records based on search criteria, improving the efficiency of read-heavy queries. Without indexes, queries can take significantly longer, especially on large datasets.**
* **Recommended Indexes:**
  1. **Routes Table: Index the RouteName, Source, and Destination columns to optimize searches for specific routes.**

**CREATE INDEX idx\_route\_name ON Routes(RouteName);**

**CREATE INDEX idx\_route\_source\_dest ON Routes(Source, Destination);**

* 1. **Vehicles Table: Index the RouteID and VehicleNumber to speed up searches by route or vehicle.**

**CREATE INDEX idx\_vehicle\_route ON Vehicles(RouteID);**

**CREATE INDEX idx\_vehicle\_number ON Vehicles(VehicleNumber);**

* 1. **Schedules Table: Index DepartureTime to optimize searches for upcoming schedules.**

**CREATE INDEX idx\_schedule\_departure ON Schedules(DepartureTime);**

* 1. **Bookings Table: Index ScheduleID and PassengerID to speed up the lookup of bookings by schedule and passenger.**

**CREATE INDEX idx\_booking\_schedule ON Bookings(ScheduleID);**

**CREATE INDEX idx\_booking\_passenger ON Bookings(PassengerID);**

**Performance Impact: While indexing improves query performance, it also has an overhead on INSERT, UPDATE, and DELETE operations because indexes need to be maintained during data modifications. Therefore, it's important to balance the number of indexes based on query frequency and database workload.**

**Insights on System Performance, Privacy, and Scalability**

1. **System Performance**:
   * **Efficient Data Retrieval**: The performance of the PTMS largely depends on the indexing strategy, query optimization, and the normalization of data. By indexing frequently queried fields (such as RouteName, VehicleNumber, and DepartureTime), query response times can be significantly reduced, even with a large dataset.
   * **Query Optimization**: Efficient SQL queries, such as using selective column retrieval instead of SELECT \*, limiting joins, and employing EXISTS instead of IN, ensures that the system can handle real-time queries with low latency.
   * **Load Handling**: The system is designed to handle high volumes of passenger bookings and schedule queries. Proper indexing and caching strategies allow for effective load balancing, ensuring smooth performance during peak hours.
2. **Privacy**:
   * **Sensitive Data Protection**: The system stores personal data about passengers, including names and contact information. To protect privacy, it is essential to apply appropriate **privacy constraints** such as data encryption, access controls, and anonymization where possible.
   * **Data Minimization**: Only essential information (e.g., booking data and contact info) is stored in the system. Ensuring minimal data collection reduces the risk of unauthorized access.
   * **Compliance with Regulations**: The system must comply with data protection laws such as GDPR or CCPA, ensuring that passenger data is only stored and processed with explicit consent, and that users have access to their data upon request.
3. **Scalability**:
   * **Horizontal and Vertical Scaling**: As the number of passengers, routes, and vehicles increases, the system must be able to scale to handle larger volumes of data. This can be achieved through **horizontal scaling**, such as adding more database servers, or **vertical scaling**, such as upgrading existing hardware.
   * **Database Partitioning**: Implementing database partitioning can further improve performance by splitting large tables (e.g., **Bookings**) across multiple storage locations. This helps reduce contention and speeds up data access, particularly in high-demand scenarios.

**Conclusion:**

. In conclusion, the development of an **Automation Rental System** using a **Database Management System (DBMS)** provides several advantages, primarily in terms of efficiency, data management, and user experience. The system allows for streamlined operations, where rental transactions, inventory management, customer data, and financial records are all handled within a single, centralized platform. Here are some key takeaways from the system:

1. **Centralized Data Storage**: The use of a DBMS ensures that all data related to rentals, customers, payments, and inventory is stored in an organized and centralized manner. This makes it easier to retrieve, update, and manage data, reducing errors and improving data integrity.
2. **Efficient Transaction Management**: The automation of rental processes, such as booking, payment processing, and return scheduling, is facilitated by the DBMS, ensuring faster and error-free transactions. The system can automatically handle inventory checks, due dates, and payment reminders, making it more efficient than manual systems.
3. **Scalability and Flexibility**: The DBMS allows for easy expansion as the rental business grows. New rental items, customer categories, and payment options can be added to the system with minimal disruption. The database can also scale to accommodate more users and transactions as demand increases.
4. **Improved Customer Experience**: With automated processes such as online booking, real-time availability checking, and prompt notifications, customers have a seamless experience. They can easily access the system, reserve items, make payments, and track their rental status without the need for manual intervention.
5. **Data Security and Backup**: A robust DBMS provides better data security, ensuring that sensitive customer and payment information is protected from unauthorized access. It also allows for regular data backups, reducing the risk of data loss.
6. **Reporting and Analytics**: The DBMS enables the generation of detailed reports, helping the rental business to analyze trends, track performance, and make informed decisions. Business owners can track rental frequency, revenue, and customer behavior to optimize inventory and pricing strategies.
7. **Error Reduction**: By automating routine tasks, the system significantly reduces the risk of human error. Automatic updates, alerts, and validations help maintain the accuracy of records and ensure that the system remains consistent.

Overall, an **Automation Rental System DBMS** serves as a powerful tool for optimizing the operations of a rental business, improving customer satisfaction, and supporting growth. It provides a foundation for managing data in a more structured and efficient manner, ultimately contributing to the success and scalability of the business.