|  |  |
| --- | --- |
| Car Rental BigData & Database Systems | May 20  2024 |

**Group Member Name : Syed Najaf Ali 27662**

**Shoaib Hassan 23647**

Introduction

The car rental industry is growing rapidly, driven by increased awareness of environmental issues and the adoption of clean energy vehicles. To manage this efficiently, a well-structured database is crucial. This project involves designing a database system for an electric car rental business, focusing on entity relationships, functional dependencies, and normalization to ensure minimal data redundancy and data consistency.

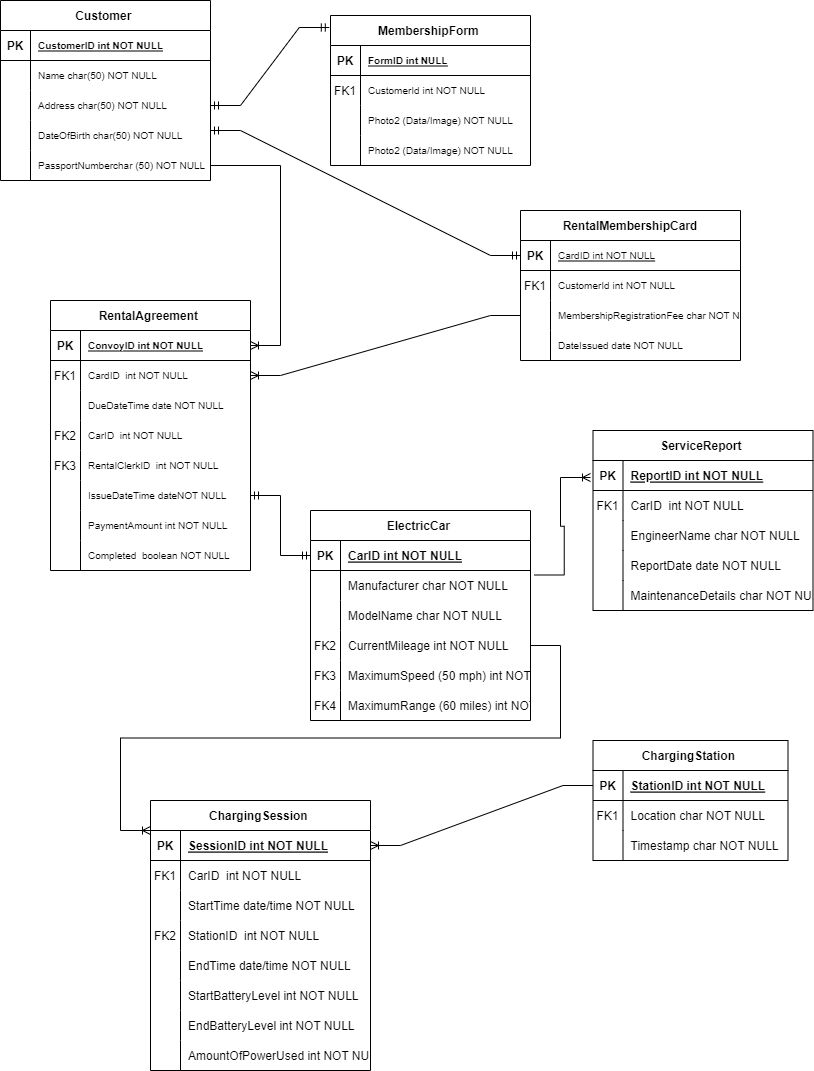
**Existing Models/Literature Review**

Several models and literature sources discuss database design principles and best practices, especially in the context of rental and inventory management systems. Common themes include the importance of normalization to reduce data redundancy, the use of primary and foreign keys to maintain data integrity, and the implementation of efficient querying mechanisms to enhance performance. Studies also emphasize the balance between normalization and practical considerations such as query performance and ease of data retrieval.

**GitHub link Below :**

<https://github.com/Najaf555/Car-Rental-Bigdata-DBS/tree/main/BigData%26DatabaseSystem>

**ERD Model Diagram**



**Working**

1. **Functional Dependency Statement**

**Customer Entity:**

1. CustomerID → Name, Address, DateOfBirth, PassportNumber

* The customer’s identity card uniquely identifies their name, address, date of birth and passport number.
* This represented a one-to-one relationship without partial dependencies.

**MembershipForm Entity:**

1. FormID → CustomerID, Photo1, Photo2, IDCopy

* Each membership card ID is associated with a specific customer and includes two photos and a copy of the ID.
* This is a one-to-one relationship, ensuring no partial dependencies.

**RentalMembershipCard Entity:**

1. CardID → CustomerID, MembershipRegistrationFee, DateIssued

* Each rental membership card ID uniquely matches the customer, the registration amount, and the date of issue.
* This represented a one-to-one relationship without partial dependencies.

**RentalAgreement Entity:**

1. AgreementID → CardID, CarID, RentalClerkID, IssueDateTime, DueDateTime, PaymentAmount, Completed

* Each rental contract associates the ID with the specific card (and thus the customer), the rented e-vehicle, the payment clerk, the checkout date/time, the due date/time, the amount payment.
* This represented a one-to-one relationship without partial dependencies.

**ElectricCar Entity:**

1. CarID → Manufacturer, ModelName, CurrentMileage, MaximumSpeed, MaximumRange

* The e-car ID determines the make, model name, current mileage, top speed and maximum range.
* This represented a one-to-one relationship without partial dependencies.

**ChargingSession Entity:**

1. SessionID → CarID, StationID, StartTime, EndTime, StartBatteryLevel, EndBatteryLevel, AmountOfPowerUsed

* Each charging session ID (SessionID) is associated with a specific e-car (CarID), charging station (StationID), start time (StartTime), end time (EndTime), battery level (StartBatteryLevel, EndBatteryLevel), and power the amount of power consumed is related (AmountOfPower Consumed).
* This represented a one-to-one relationship without partial dependencies.

**ChargingStation Entity:**

1. StationID → Location, Timestamp

* Each charging station ID is associated with a specific location and time stamp.
* This represented a one-to-one relationship without partial dependencies.

**ServiceReport Entity:**

1. ReportID → CarID, EngineerName, ReportDate, MaintenanceDetails

* Each Service Report ID (ReportID) is uniquely generated for a specific e-Vehicle (CarID), engineer name (EngineerName), report date (ReportDate), and maintenance details (MaintenanceDetails).
* This represented a one-to-one relationship without partial dependencies.

**Maximum range data changed, maximum of 60 Miles.**

To update the 'MaximumRange' data in the 'ElectricCar' table to a maximum of 60 miles based on the scenario requirements, you can use a SQL UPDATE statement.

USE ElectricCarRentalDB;

UPDATE ElectricCar

SET MaximumRange = 60

WHERE MaximumRange > 60 OR MaximumRange IS NULL;

* This SQL statement updates the 'ElectricCar' table.
* The SET MaximumRange = 60 section specifies the new value (60 miles) that you want to set for the 'MaximumRange' attribute.
* The WHERE MaximumRange > 60 OR MaximumRange IS NULL condition filters rows where 'MaximumRange' is currently greater than 60 or where 'MaximumRange' is NULL (ensures that the governments maximum distance value is 60 miles).
* Using this SQL statement, I will update the ‘MaximumRange’ attribute for all electric vehicles in the database to ensure that the condition specified range requirement of 60 miles is met.

**Normalization**

In building our database for the electric car rental project, I aimed to organize our tables using normalization principles that provided minimal data redundancy and ensured data consistency if I will obey it.

* I organized the tables to have a well-defined primary key and reduced the use of multi-valued dependencies or dependencies operating on non-key assets.
* The attributes were organized logically, and each table was designed to represent a specific company or relationship in a business context.

**What we didn’t do and why we didn’t**

Although we aimed to build a general database, achieving a complete BCNF in the real world can sometimes be difficult or unnecessary depending on specific business needs and data complexity.

* I would not decompose tables beyond 3NF or BCNF into a more generalized approach if it provided significant benefits in terms of data management or query performance without complications.
* In some cases, a balance between normalization principles and practical considerations may be chosen for denormalization to optimize query performance or facilitate data retrieval.

**Issues**

Before designing our database, we considered potential generalization problems and aimed to solve them easily by considering table-feature design.

* The tables were created using explicit primary key and foreign key relationships to reduce redundancy and ensure data consistency.
* By organizing the tables based on entity relationships and functional dependencies, we aimed to minimize normalization inconsistencies before they arose during use.
* In summary, although efforts to normalize to 3NF or BCNF are important in database design, practical considerations and business needs may influence the extent to which full normalization can be achieved

**Partial Dependencies in Database Normalization**

**Why Remove Partial Dependencies?**

* Data redundancy: Partial dependencies can cause the same information to be stored in multiple locations in the database, resulting in data redundancy. This redundancy wastes storage space and updating the same data in different locations can lead to inconsistent results.
* Inconsistent updates: If there is a dependent component, updating a primary key (or part of it) may require you to update several rows to keep the data consistent This creates a risk of update anomalies is significant, where data changes cause inconsistencies or errors.

**What we did to address partial dependencies:**

* In building our database for the electric car rental project, we aimed to eliminate partial dependencies by creating tables based on functional reliability and normalization principles on the various types.
* Primary identification: We carefully identified the primary key of each table, ensuring that it identifies each row uniquely and minimally (there are no redundant properties in the primary key).
* Normalized Tables: By normalizing our tables to Third Normal Form (3NF) or Boyce-Codd Normal Form (BCNF), our goal was to partially eliminate dependencies and ensure that non-prime properties are key dependent especially so completely in use.
* Designed relationships: We used foreign keys to create valid relationships between tables, ensuring data integrity and avoiding partial dependencies between related entities.

**Potential problems if partial dependencies are not removed:**

* Data inconsistencies: If it does not have partial dependencies, modifying the database can cause inconsistencies where the associated data does not accurately reflect the changes
* Query problems: When data is spread across multiple tables due to partial dependencies, querying the database can become more complex, requiring more complex joins and queries function to obtain the desired information
* Increased storage and maintenance costs: Redundant data resulting from partial dependencies can increase storage requirements and maintenance efforts, and aka database performance and scalability

**Conclusion.**

Each entity's primary key uniquely determines its attributes, and there aren't any partial dependencies or violations of normalization ideas found based totally at the given statistics.

In our electric car rental Db, having both an 'Employee' table and a 'Payment' table is really important for keeping everything running smoothly.

**Employee Table Explanation:**

The ‘Employees’ table helps us manage employees more efficiently. We have a variety of roles such as mortgage clerks, maintenance technicians, and administrative staff, each with specific roles that are important to our day-to-day operations

For example, invoicing clerks interact directly with customers during rentals, so they need good customer service skills and knowledge of our leasing policies. I makesure and maintain our electric vehicles are working safely for customers. Using the ‘Employees’ table we can view important information such as employee identity, name, role, contact details, when they started working with us and so on. This makes it easier to manage our team, keep track of who gets what, keep track of things like paychecks and how well everyone is doing.

**Payment Table Explanation:**

The ‘Payment’ table is important for managing cash side rental business. In our scenario, customers pay for rental cars, memberships, and sometimes repairs or extras needed for their rental.

For example, when a customer takes an electric vehicle, we create a payment record that shows the amount paid, including any taxes or fees, and how it was paid (such as by credit card or cash) This payment information is intended for use will keep track of our finances in order to, do our audits, etc. It is also important to make sure we know how much money we making from rent. Using the ‘Payment’ table we record information such as payment ID, amount, date of payment, method of payment, and associated customer or rental agreement. This allows to manage finances. This helps us track of our finances, reports , and make sure we're managing our revenue properly.

1. **SetUp Tables**

* **Caption:** This screenshot displays that I create two tables in the ElectricCarRentalDB database: chargingstation and servicereport.
* **Explanation:**

I created two tables:

chargingstation with columns StationID (primary key), Location, and Timestamp.

servicereport with columns ReportID (primary key), CarID (foreign key referencing electriccar table), EngineerName, ReportDate, MaintenanceDetails, and RepairCost.

* **Thought Process:**

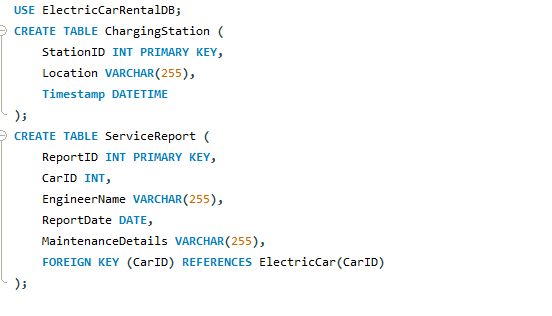
The chargingstation table is designed to store information about electric vehicle charging stations, including their unique ID (StationID), location (Location), and the timestamp (Timestamp) of data entry.

The servicereport table is for recording service reports related to electric cars .

* **Problem Resolution:**

Tables were created without having any issues.

The foreign key constraint (CarID) was correctly defined in the servicereport table, referencing the CarID primary key in the electriccar table .

****

* **Caption:** This screenshot show creation of rentalmembershipcard and electriccar tables in the ElectricCarRentalDB database.
* **Explanation:**

I added two tables:

rentalmembershipcard for managing rental membership cards, including fields for card ID (CardID), customer ID (CustomerID), registration fee (MembershipRegistrationFee), and issuance date (DateIssued).

electriccar to store details of electric vehicles, with columns for car ID (CarID), manufacturer (Manufacturer), model name (ModelName), current mileage (CurrentMileage), maximum speed (MaximumSpeed), and maximum range (MaximumRange).

* **Thought Process:**

Maked rentalmembershipcard to link membership cards to customers and capture related information.

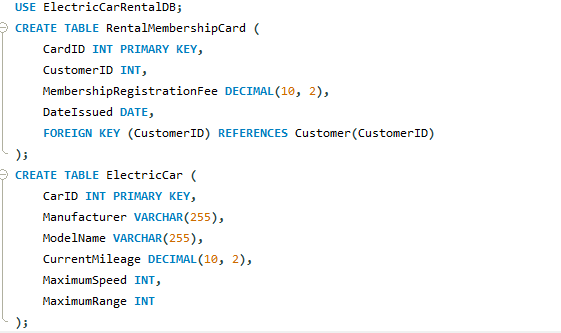
Defined electriccar to maintain details of electric vehicles for rental management.

* **Problem Resolution:**

Tables were created without having any issues.

Foreign key (CustomerID) in rentalmembershipcard references CustomerID in the customer table.

Primary keys (CardID for rentalmembershipcard and CarID for electriccar) set for uniqueness and data integrity.

****

* **Caption:** This screenshot show the creation of customer and membershipform tables in the ElectricCarRentalDB database.
* **Explanation:**

I added two tables:

customer to manage customer details, including CustomerID, Name, Address, DateOfBirth, and PassportNumber.

membershipform for storing membership application forms with FormID, CustomerID (linked to customer table), and blob fields for photos and ID copies.

* **Thought Process:**

I make customer table to store essential customer data for the rental service.

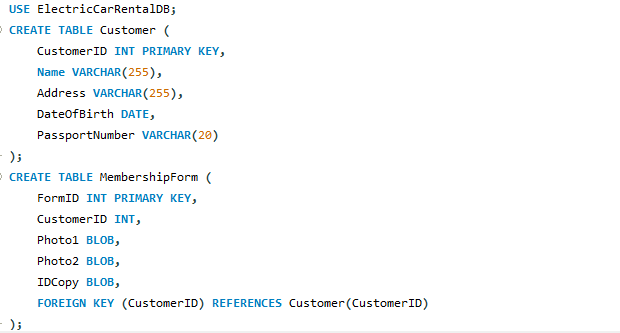
Created membershipform table to handle membership application forms associated with customers.

* **Problem Resolution:**

Tables were created without having any issues.

Foreign key (CustomerID) in membershipform references CustomerID in the customer table.

Primary keys (CustomerID for customer and FormID for membershipform) were defined for data integrity and uniqueness.

****