Car Rental Database Assignment

Ayush Byju, Goncalo Morais, Chetas Patel
CP363
Haytham Qushtom
April 7, 2023

Assignment 1: Phase I: Logical Database Design

By: Ayush Byju, Goncalo Morais, Chetas Patel
Course: CP-363-B
Haytham Qushtom
January 22, 2023

Application Database:

The database system we chose as our model example was a car rental DBMS which primarily manages a system that holds information related to entities such as Cars, Customers, Transactions, etc. This system needs to be organized and structured in a way that allows quick access to its held information and its attributes. Each entity its structure needs to hold information in a way that is concise and allows for its data to be retrieved in an organized manner. As such car rentals DBMS must have a unique form of withholding its information to be able to retrieve its data structured and efficient manner different from a simple linear database.

The primary users of this DBMS would be the personnel and staff working at the selected car rental company or in some cases could be management companies that monitor their software and DB systems. A secondary user of the system would be the customers themselves as there would be separate web pages that the customers would have access to which would allow them access information that is related to their specific purchase/rental.

The primary goal of our database system would be to develop a system that allows the rental company to organize and structure its data in a concise, efficient way. Specific entities such as Customer, Department, Employees, etc. which will have specific attributes that will allow them to be concise and retrieve information efficiently when needed. One example would be the entity Customers which would allow the rental company to retrieve information on every customer easily and concisely as it would have attributes like name, phone, address, and order history. The DBMS would allow every user to easily retrieve a large amount of information that's stored on server end.

Data Types And Relations:

Entities

Employees:

Information:

Contains data related to the employees of the company, including their name, contact information, job role, and schedules.

Use:

This is used to: manage, track and evaluate the performance of employees and schedule their work.

Departments:

Information:

The department entity in the car rental database stores information on the different departments within the company, including the type of cars they handle and the employees that work in those departments.

Use:

This is used to: manage, track and evaluate the performance of each department.

Customers:

Information:

Holds data on the customers of the company, including their names, contact information, and rental history.

Use:

This is used to: match customer needs with available cars, track their rental history, and monitor customer satisfaction.

Cars:

Information:

Holds information on the cars available for rent, including make, model, year, location, and maintenance history.

Use:

This is used to: match customer needs with available cars, track the usage and maintenance of cars and monitor their performance.

Locations:

Information:

Holds information on the different pick-up and drop-off locations for the cars, including addresses and contact information. Along with the store locations.

Use:

This is used to: track the popularity of each location and monitor their performance.

Bookings:

Information:

Holds information on the reservations made by customers, including the dates and times of the reservation, the car rented, and the customer who made the reservation.

Use:

This is used to: match customer needs with available cars, track their rental history, and monitor customer satisfaction.

Transactions:

Information:

Holds information on the billing process, including the cost of the rental, any additional charges, and payment methods.

Use:

This is used to:calculate the cost of the rental, track payments, and monitor the performance of the billing process.

Transactions:

Information:

Holds various reports on the car rental system, including customer rental history, car usage, and revenue.

Use:

This is used to: track the performance of the business, identify any issues that may arise, and make decisions based on data.

Relations:

One of the key components of this database is the relationships between the different parts of the system. These relationships are critical to the efficient and effective management of the car rental business, and they are what allows the company to track customer information, reservations, and car availability, etc. These relationships and how they interact is essential for effectively managing a car rental business.

Cars and department:

The car information in the database is linked to the various departments of the car rental that are responsible for its care. Ranging from cleaning to maintenance records for those cars, so the company can easily track the instances where operations on the car occurred.

Employee and department:

The relationship between employee and department allows the company to track which employees work in which department and manage their schedules and roles accordingly.

Location and bookings:

The relationship between location and bookings allows the company to track which locations are popular and which cars are frequently rented from those locations. Furthermore, it allows you to see the pickup and drop off of cars.

Customers and cars:

The relationship between customers and cars allows the company to track the rental history of each customer and match them with the appropriate car for their next rental. In addition, it allows the customer to view and be given information about the car they have rented.

Customers and employees:

The relationship between customers and employees allows the company to track which employees are handling which customers and identify any issues that may arise.

Customers and transaction:

The relationship between customer and transaction allows the company to track the rental history of each customer and monitor their payments, invoices, and any additional charges.

Databases Assignment 2

Total and partial participation for all ER elements :

Total	**Partial**
Employes (M) works for Departments (1) (Total)	Departments (M) work on Cars (M)
Booking (1) (Total) manages Location (1)	Customer (1) rents Car (1)
Transication (M) bills Customer (M) (Total)	Employes (1) services Customer (1)

Cardinalities have been addressed in the diagram (6-8)

Cardinalities
1) Employes (M) works for Departments (1)
2) Departments (M) work on Cars (M)
3) Booking (1) manages Location (1)
4) Customer (1) rents Car (1)
5) Employes (1) services Customer (1)
6) Transication (M) bills Customer (M)

Diagram captures all necessary attributes for DBMS

Multivalued attributes:
- Employee Info
- Customer Info
- Car Info
- Department Info

Composite attributes:
- Customer Address
- Location Address
- Vehicle Info
- Transaction Receipt

Diagram captures all keys (primary or foreign)

Primary	Foreign
Booking	Reports
Customer	Cars

Diagram captures all entities necessary to satisfy DBMS solution

1) Employees
2) Customers
3) Cars
4) Locations

DBMS has more than 5 entities (so 6)

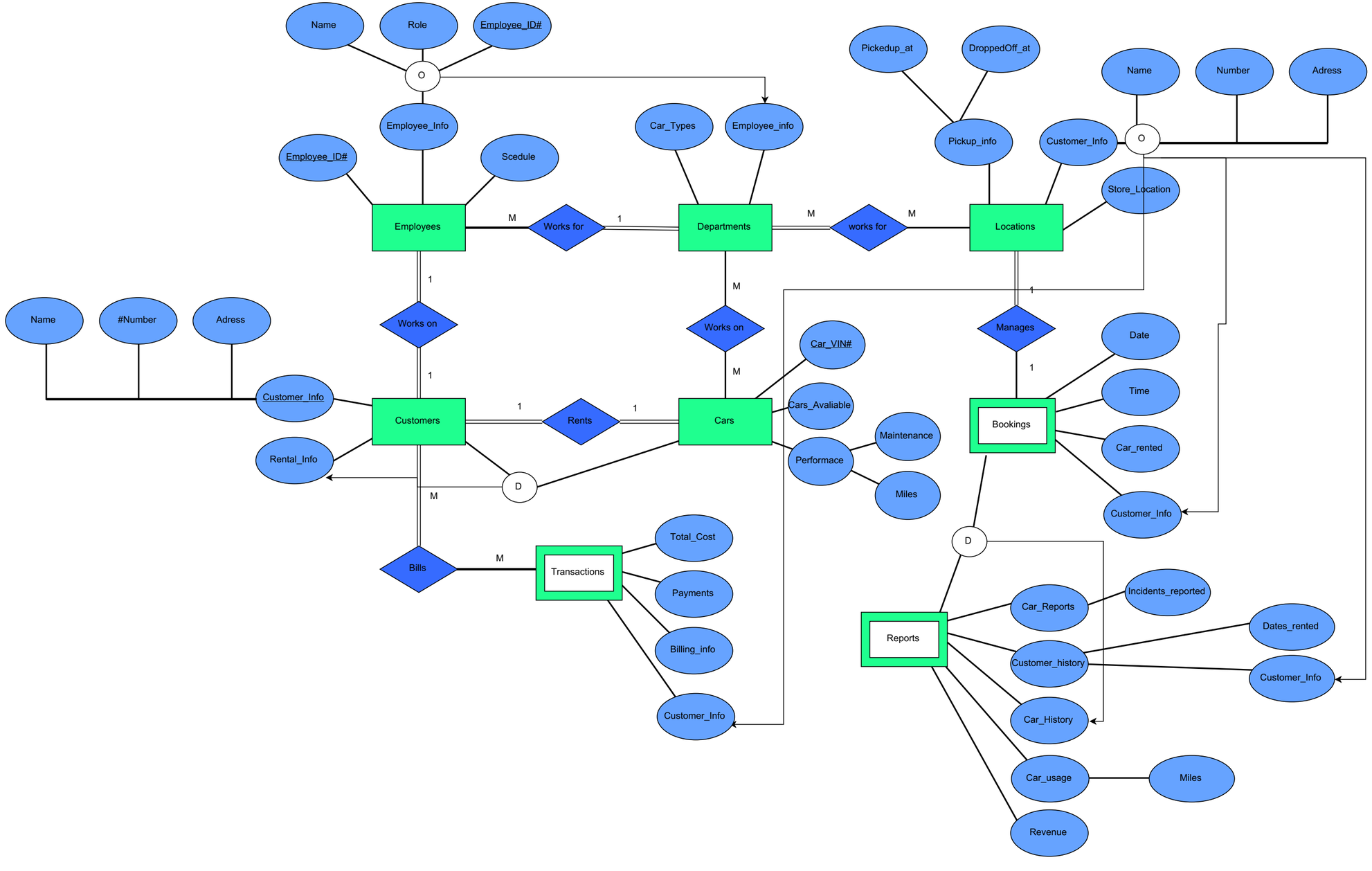
Entities

1) Employees
2) Departments
3) Customers
4) Cars
5) Locations
6) Bookings
7) Transactions
8) Reports

Has a minimum of 5 relationships and all of its necessary information

Relationships

1) Cars and department
2) Employee and department
3) Location and bookings
4) Customers and Cars
5) Customers and employees
6) Customers and transaction


Name
Role
Employee_ID#
O
Employee_Info
Employee_ID#
Scedule
Car_Types
Employee_info
Pickedup_at
DroppedOff_at
Pickup_info
Name
Number
Adress
Customer_Info
O
Store_Location
Employees
M
Works for
1
Departments
M
works for
M
Locations
1
M
1
Works on
Works on
Manages
1
M
1
Name
#Number
Adress
Customer_Info
Customers
1
Rents
1
Cars
Car_VIN#
Cars_Avaliable
Maintenance
Performace
Miles
Bookings
Date
Time
Car_rented
Customer_Info
M
D
D
Bills
M
Transactions
Total_Cost
Payments
Billing_info
Customer_Info
Reports
Car_Reports
Incidents_reported
Customer_history
Dates_rented
Customer_Info
Car_History
Car_usage
Miles
Revenue

Assignment 6

Written by: Ayush Byju,Goncalo Morais,

Course: CP-363-B Haytham Qushtom

Tables normalized in 3NF and or BCNF.

Functional dependencies for the initial tables are as follows:

Departments table:

Department_Id → Department_Name, Cars_Types

Cars_Types → Department_Name

Cars table:

Car_VIN → CarsAvailable, Performance, Departments_Car_Types

Departments_Car_Types → CarsAvailable, Performance

Customers table:

Customer_Info → Rental_Info, Cars_Car_VIN, Transaction_Total_Cost

Cars_Car_VIN → Rental_Info, Transaction_Total_Cost

Transactions table:

Customer_Info → TotalCost, Payments, BillingInfo

Locations table:

StoreLocation → PickUpInfo, DropOffInfo, Customer_Info

Bookings table:

VehicleRented → BookingDate, BookingTime, Customer_Info

Customer_Info → BookingDate, BookingTime

Reports table:

VehicleVIN → IncidentReports, DatesReturned, ReturnedTime, NewMileage, Revenue, Customer_Info

Customer_Info → IncidentReports, DatesReturned, ReturnedTime, NewMileage, Revenue

Employees table:

Employee_ID → Employee_Info, Schedule, Departments_Car_Types, Customer_Info Customer_Info → Employee_Info, Schedule

Departments_Car_Types → Employee_Info, Schedule

After normalizing the schema for the db using 3NF AND BNCF

Departments table:

Already in BCNF, no further optimization needed

Cars table:

Move the CarsAvailable attribute to a separate Availability table with Car_VIN and Availability_Date as primary keys.

Move the Performance attribute to a separate Performance table with Car_VIN and Performance_Date as primary keys.

Keep the Departments_Car_Types attribute in the Cars table as a foreign key to the Departments table.

Customers table:

Move the Rental_Info attribute to a separate Rentals table with Customer_Info and Rental_Date as primary keys.

Keep the Cars_Car_VIN attribute in the Customers table as a foreign key to the Cars table.

Total cost can be calculated from the Rentals table, so it is not necessary to have it as a separate attribute in the Customers table.

Transactions table:

Move the Payments attribute to a separate Payments table with BillingInfo and Payment_Date as primary keys.

Keep the Customer_Info attribute in the Transactions table as a foreign key to the Customers table. Total cost can be calculated from the Rentals table, so it is not necessary to have it as a separate attribute in the Transactions table.

Locations table:

Move the PickUpInfo and DropOffInfo attributes to a separate Reservations table with Customer_Info, Car_VIN, PickUpDate, and DropOffDate as primary keys.

Keep the StoreLocation attribute in the Locations table as a primary key. Keep the Customer_Info attribute in the Locations table as a foreign key to the Customers table.

Bookings table:

Already in BCNF, no further optimization needed.

Reports table:

Move the IncidentReports attribute to a separate Incidents table with VehicleVIN and Incident_Date as primary keys.

Keep the VehicleVIN attribute in the Reports table as a foreign key to the Cars table. Keep the Customer_Info attribute in the Reports table as a foreign key to the Customers table.

Employees table:

Keep the Employee_ID attribute as the primary key.

Move the Employee_Info attribute to a separate EmployeeDetails table with Employee_ID and Detail_Date as primary keys.

Keep the Schedule attribute in the Employees table as a separate attribute. Keep the Departments_Car_Types attribute in the Employees table as a foreign key to the Departments table. Keep the Customer_Info attribute in the Employees table as a foreign key to the Customers table.

Functional Dependencies with all the primary keys for all tables in proper format (represented in a table)

Table_Name (Determinants)	**Primary Key Functional Dependencies (Dependents)**
Departments	Primary Key: {Department_Id} Functional Dependencies: Department_Id -> Department_Name, Department_Id -> Cars_Types
Cars	Primary Key: {Car_VIN} Functional Dependencies: Car_VIN -> CarsAvailable, Car_VIN -> Performance, Car_VIN -> Departments_Car_Types
Customers	Primary Key: {Customer_Info} Functional Dependencies: Customer_Info -> Rental_Info, Customer_Info -> Cars_Car_VIN, Customer_Info -> Transaction_Total_Cost
Transactions	Primary Key: {Transaction_ID} Functional Dependencies: Customer_Info -> TotalCost, Customer_Info -> Payments, Customer_Info -> BillingInfo
Locations	Primary Key: {StoreLocation} Functional Dependencies: StoreLocation -> PickUpInfo, StoreLocation -> DropOffInfo, StoreLocation -> Customer_Info
Bookings	Primary Key: {VehicleRented} Functional Dependencies: VehicleRented -> BookingDate, VehicleRented -> BookingTime, VehicleRented -> Customer_Info
Reports	Primary Key: {VehicleVIN} Functional Dependencies: VehicleVIN -> IncidentReports, VehicleVIN -> DatesReturned, VehicleVIN -> ReturnedTime, VehicleVIN -> NewMileage, VehicleVIN -> Revenue, VehicleVIN -> Customer_Info

Employees	Primary Key: {Employee_ID} Functional Dependencies: Employee_ID -> Employee_Info, Employee_ID -> Schedule, Employee_ID -> Departments_Car_Types, Employee_ID -> Customer_Info

Assignment 7

Written by: Ayush Byju,Goncalo Morais

Course: CP-363-B Haytham Qushtom

All tables normalized in table format below:

Table_Name (Determinants)	**Primary Key Functional Dependencies (Dependents)**
Departments	Primary Key: {Department_Id} Functional Dependencies: Department_Id -> Department_Name, Department_Id -> Cars_Types
Cars	Primary Key: {Car_VIN} Functional Dependencies: Car_VIN -> CarsAvailable, Car_VIN -> Performance, Car_VIN -> Departments_Car_Types
Customers	Primary Key: {Customer_Info} Functional Dependencies: Customer_Info -> Rental_Info, Customer_Info -> Cars_Car_VIN, Customer_Info -> Transaction_Total_Cost
Transactions	Primary Key: {Transaction_ID} Functional Dependencies: Customer_Info -> TotalCost, Customer_Info -> Payments, Customer_Info -> BillingInfo
Locations	Primary Key: {StoreLocation} Functional Dependencies: StoreLocation -> PickUpInfo, StoreLocation -> DropOffInfo, StoreLocation -> Customer_Info
Bookings	Primary Key: {VehicleRented} Functional Dependencies: VehicleRented -> BookingDate, VehicleRented -> BookingTime, VehicleRented -> Customer_Info
Reports	Primary Key: {VehicleVIN} Functional Dependencies: VehicleVIN -> IncidentReports, VehicleVIN -> DatesReturned, VehicleVIN -> ReturnedTime, VehicleVIN -> NewMileage, VehicleVIN -> Revenue, VehicleVIN -> Customer_Info
Employees	Primary Key: {Employee_ID} Functional Dependencies: Employee_ID -> Employee_Info, Employee_ID -> Schedule, Employee_ID -> Departments_Car_Types, Employee_ID -> Customer_Info

Reasoning:

Departments table:

Already in BCNF, no further optimization needed

Cars table:

Move the CarsAvailable attribute to a separate Availability table with Car_VIN and Availability_Date as primary keys.

Move the Performance attribute to a separate Performance table with Car_VIN and Performance_Date as primary keys.

Keep the Departments_Car_Types attribute in the Cars table as a foreign key to the Departments table.

Customers table:

Move the Rental_Info attribute to a separate Rentals table with Customer_Info and Rental_Date as primary keys.

Keep the Cars_Car_VIN attribute in the Customers table as a foreign key to the Cars table.

Total cost can be calculated from the Rentals table, so it is not necessary to have it as a separate attribute in the Customers table.

Transactions table:

Move the Payments attribute to a separate Payments table with BillingInfo and Payment_Date as primary keys.

Keep the Customer_Info attribute in the Transactions table as a foreign key to the Customers table. Total cost can be calculated from the Rentals table, so it is not necessary to have it as a separate attribute in the Transactions table.

Locations table:

Move the PickUpInfo and DropOffInfo attributes to a separate Reservations table with Customer_Info, Car_VIN, PickUpDate, and DropOffDate as primary keys.

Keep the StoreLocation attribute in the Locations table as a primary key. Keep the Customer_Info attribute in the Locations table as a foreign key to the Customers table.

Bookings table:

Already in BCNF, no further optimization needed.

Reports table:

Move the IncidentReports attribute to a separate Incidents table with VehicleVIN and Incident_Date as primary keys.

Keep the VehicleVIN attribute in the Reports table as a foreign key to the Cars table. Keep the Customer_Info attribute in the Reports table as a foreign key to the Customers table.

Employees table:

Keep the Employee_ID attribute as the primary key.

Move the Employee_Info attribute to a separate EmployeeDetails table with Employee_ID and Detail_Date as primary keys.

Keep the Schedule attribute in the Employees table as a separate attribute. Keep the Departments_Car_Types attribute in the Employees table as a foreign key to the Departments table. Keep the Customer_Info attribute in the Employees table as a foreign key to the Customers table.

Transitive function dependency and compound PK

Consider the Customers table:

```
CREATE TABLE Customers (
  Customer_Info  VARCHAR(255) NOT NULL,
  Rental_Info  MEDIUMBLOB  NOT  NULL,
  Cars_Car_VIN  VARCHAR(17) NOT  NULL,
  Transaction_Total_Cost  INT  NOT NULL,
  PRIMARY KEY (Customer_Info(255)) ,
  CONSTRAINT fk_customers_cars  FOREIGN KEY (Cars_Car_VIN)  REFERENCES  Cars(Car_VIN)
);
```

We create a transitive functional dependency where the Transaction_Total_Cost attribute depends on the Performance attribute of the car rented by the customer, which in turn depends on the Departments_Car_Types attribute of the department that owns the car.

To create this transitive functional dependency, we can modify the Customers table

Modified customers' table:

```
CREATE TABLE Customers (
  Customer_Info  VARCHAR(255) NOT NULL,
  Rental_Info  MEDIUMBLOB  NOT  NULL,
  Cars_Car_VIN  VARCHAR(17) NOT  NULL,
  Transaction_Total_Cost  INT  NOT NULL,
  Performance  VARCHAR(45) NOT NULL,
  Departments_Car_Types  VARCHAR(45) NOT NULL,
  PRIMARY KEY (Customer_Info(255)) ,
  CONSTRAINT fk_customers_cars  FOREIGN KEY (Cars_Car_VIN)  REFERENCES  Cars(Car_VIN),
  FOREIGN KEY (Departments_Car_Types,  Cars_Car_VIN)  REFERENCES  Cars(Departments_Car_Types,  Car_VIN)
);
```

we add the Performance and Departments_Car_Types attributes to the Customers table and also add a foreign key constraint that references the Cars table using both the Departments_Car_Types and Car_VIN attributes.

By doing this, we have create a transitive functional dependency where the Transaction_Total_Cost attribute depends on the composite key of Departments_Car_Types and Car_VIN. This ensures that the

transaction cost is dependent on the car's performance, which in turn is dependent on the department that owns the car.

Table Decomposition

Consider the Departments table:

```
CREATE TABLE Departments (
  Department_Id INT PRIMARY KEY,
  Department_Name VARCHAR(45) NOT NULL,
  Cars_Types VARCHAR(45) UNIQUE
);
```

Here, the Cars_Types attribute is functionally dependent on the Department_Id attribute, which is part of the primary key. However, it is also functionally dependent on the Department_Name attribute, which is not part of the primary key. This is a compound functional dependency, which violates the 2NF.

To decompose this table into 2NF, we can split it into two separate tables:

```
CREATE TABLE Departments (
  Department_Id INT PRIMARY KEY,
  Department_Name VARCHAR(45) NOT NULL
);
CREATE TABLE Department_Cars (
  Department_Id INT,
  Cars_Types VARCHAR(45),
  PRIMARY KEY (Department_Id, Cars_Types),
  FOREIGN KEY (Department_Id) REFERENCES Departments(Department_Id)
);
```

Here, we have removed the Cars_Types attribute from the Departments table and created a new table called Department_Cars. The new table has a composite primary key consisting of Department_Id and Cars_Types, and it has a foreign key reference to the Departments table.

This decomposition ensures that each table has a single purpose or topic, and there are no partial dependencies or compound functional dependencies in the schema.

Assignment 8

Written by: Ayush Byju,Goncalo Morais

Course: CP-363-B Haytham Qushtom

Partial dependencies and Functional Dependencies

List of all FDs:

Departments:

Department_Id -> Department_Name

Department_Id -> Cars_Types

Cars:

Car_VIN -> CarsAvailable

Car_VIN -> Performance

(Cars_Types, Car_VIN) -> Departments_Car_Types

Customers:

Customer_Info -> Rental_Info

Customer_Info -> Cars_Car_VIN

(Cars_Car_VIN) -> Transaction_Total_Cost

Transactions:

Customer_Info -> TotalCost

Customer_Info -> Payments

Customer_Info -> BillingInfo

Locations:

StoreLocation -> PickUpInfo

StoreLocation -> DropOffInfo

StoreLocation -> Customer_Info

Bookings:

VehicleRented -> BookingDate

VehicleRented -> BookingTime

VehicleRented -> Customer_Info

Reports:

VehicleVIN -> IncidentReports

VehicleVIN -> DatesReturned

VehicleVIN -> ReturnedTime

VehicleVIN -> NewMileage

VehicleVIN -> Revenue

VehicleVIN -> Customer_Info

Employees:

Employee_ID -> Employee_Info

Employee_ID -> Schedule

Employee_ID -> Departments_Car_Types

Employee_ID -> Customer_Info

We applied the Bernstein and BCNF algorithms on the “Cars” table. I made the “CarsAvailable” attribute is dependent on the “Department_Name” attribute. This creates a transitive dependency through the Departments table, since “Department_Name” is not a candidate key of Departments:

Cars:

Car_VIN -> CarsAvailable

Car_VIN -> Performance

(Cars_Types, Car_VIN) -> Departments_Car_Types

Departments_Car_Types -> Department_Id

Department_Id -> Department_Name

Department_Id -> Cars_Types

(Cars_Types, Department_Name) -> CarsAvailable

To show a partial dependency in the Customers table, let's say that the Rental_Info attribute is dependent only on the Cars_Car_VIN attribute, and not on the entire Customer_Info attribute. This creates a partial dependency:

Customers:

Customer_Info -> Cars_Car_VIN, Transaction_Total_Cost

Cars_Car_VIN -> Rental_Info

Algorithms:

Departments table:

The Departments table is already in 3NF since it has only one non-prime attribute and all functional dependencies are already fully functional.

Cars table:

The Cars table has the functional dependency Departments_Car_Types -> Department_Name, Cars_Types, which is a partial dependency since Departments_Car_Types is not a candidate key. To normalize the table, we can split it into two tables:

Cars (Car_VIN, CarsAvailable, Performance, Departments_Car_Types)

Departments (Departments_Car_Types, Department_Name, Cars_Types)

This satisfies 3NF since all functional dependencies are now fully functional.

Customers table:

The Customers table has the functional dependency Customer_Info -> Rental_Info, Cars_Car_VIN, Transaction_Total_Cost, and the transitive dependency Cars_Car_VIN -> Car_VIN. To normalize the table, we can split it into three tables:

Customers (Customer_Info, Transaction_Total_Cost)

Rentals (Customer_Info, Rental_Info)

Cars_Rented (Customer_Info, Car_VIN)

This satisfies 3NF since all functional dependencies are now fully functional.

Transactions table:

The Transactions table is already in BCNF since there is only one candidate key (Customer_Info) and all functional dependencies are fully functional.

Locations table:

The Locations table has the functional dependency Customer_Info -> Customer_Info and the transitive dependency Customer_Info -> PickUpInfo, DropOffInfo, StoreLocation. To normalize the table, we can split it into two tables:

Customers (Customer_Info)

Locations (StoreLocation, PickUpInfo, DropOffInfo, Customer_Info)

This satisfies 3NF since all functional dependencies are now fully functional.

Bookings table:

The Bookings table has the functional dependency VehicleRented -> BookingDate, BookingTime, Customer_Info and the transitive dependency Customer_Info -> Customer_Info. To normalize the table, we can split it into two tables:

Customers (Customer_Info)

Bookings (VehicleRented, BookingDate, BookingTime, Customer_Info)

This satisfies 3NF since all functional dependencies are now fully functional.

Reports table:

The Reports table has the functional dependency VehicleVIN -> IncidentReports, DatesReturned, ReturnedTime, NewMileage, Revenue, Customer_Info and the transitive dependency Customer_Info -> Customer_Info. To normalize the table, we can split it into two tables:

Customers (Customer_Info)

Reports (VehicleVIN, IncidentReports, DatesReturned, ReturnedTime, NewMileage, Revenue, Customer_Info)

This satisfies 3NF since all functional dependencies are now fully functional.

Employees table:

The Employees table has the functional dependencies Customer_Info -> Customer_Info, Departments_Car_Types -> Department_Name, Cars_Types. Since there are no partial or transitive dependencies between attributes that are not candidate keys, this table is already in BCNF.

Assignment 10 Ras for all queries:

-- Query 1:

```
SELECT Customer_Info
FROM Customers c
WHERE EXISTS (
  SELECT 1
  FROM Bookings b
  WHERE b.Customer_Info = c.Customer_Info
);
```

--RA for query 1:

π Customer_Info (σ EXISTS (π 1 (Bookings ⋈ Customer_Info = Customer_Info Customers c)))

-- Query 2:

```
SELECT Car_VIN, Performance
FROM Cars
UNION
SELECT VehicleRented, BookingDate
FROM Bookings;
```

--RA for query 2:

π Car_VIN, Performance ((π Car_VIN, Performance Cars) ⋃ (π VehicleRented, BookingDate Bookings))

-- Query 3:

```
SELECT Department_Name, Cars_Types
FROM Departments d
WHERE NOT EXISTS (
  SELECT 1
  FROM Cars c
  WHERE c.Departments_Car_Types = d.Cars_Types
  AND c.CarsAvailable = 'Yes'
```

);

--RA for query 3:

π Department_Name, Cars_Types (σ NOT EXISTS (π 1 (σ Departments_Car_Types = Cars_Types AND CarsAvailable = 'Yes' Cars c) Departments d))

-- Query 4:

SELECT Departments_Car_Types, COUNT(*) AS NumBookings

FROM Bookings

JOIN Cars ON Bookings.VehicleRented = Cars.Car_VIN

GROUP BY Departments_Car_Types;

--RA for query 4:

π Departments_Car_Types, COUNT(*) AS NumBookings (Bookings ⋈ VehicleRented = Car_VIN Cars) (σ Departments_Car_Types = Departments_Car_Types (γ Departments_Car_Types, COUNT(*) Bookings))

-- Query 5:

SELECT Departments_Car_Types, COUNT(*) AS NumBookings

FROM Bookings

JOIN Cars ON Bookings.VehicleRented = Cars.Car_VIN

GROUP BY Departments_Car_Types

HAVING COUNT(*) > 10;

--RA for query 5:

π Departments_Car_Types, COUNT(*) AS NumBookings (Bookings ⋈ VehicleRented = Car_VIN Cars) (σ Departments_Car_Types = Departments_Car_Types (γ Departments_Car_Types, COUNT(*) Bookings) (σ COUNT(*) > 10))

-- Query 6:

SELECT PickUpInfo, DropOffInfo, StoreLocation, c.Customer_Info

FROM Locations l

JOIN Customers c ON l.Customer_Info = c.Customer_Info

WHERE c.Rental_Info IS NOT NULL

```
UNION

SELECT PickUpInfo, DropOffInfo, StoreLocation, c.Customer_Info

FROM Locations l

JOIN Transactions t ON l.Customer_Info = t.Customer_Info

JOIN Customers c ON t.Customer_Info = c.Customer_Info

WHERE t.TotalCost > 1000;
```

--RA for query 6:

π PickUpInfo, DropOffInfo, StoreLocation, Customer_Info ((Locations ⋈ Customer_Info = Customer_Info Customers c) ⋃ (Locations ⋈ Customer_Info = Customer_Info Transactions t ⋈ TotalCost > 1000 Customers c))

```
-- Query 7:

SELECT d.Department_Name, d.Cars_Types, COUNT(e.Employee_ID) AS NumEmployees

FROM Departments d

LEFT JOIN Employees e ON d.Cars_Types = e.Departments_Car_Types

GROUP BY d.Department_Name, d.Cars_Types

HAVING COUNT(e.Employee_ID) > 0;
```

--RA for query 7:

π Department_Name, Cars_Types, COUNT(Employee_ID) AS NumEmployees ((γ Department_Name, Cars_Types, COUNT(Employee_ID) Employees e) ⋈ Cars_Types = Departments_Car_Types (Departments d)) (σ COUNT(Employee_ID) > 0)

```
-- Query 8:

SELECT Department_Name

FROM Departments;
```

--RA for query 8:

π Department_Name Departments d

```
-- Query 9:

UPDATE Cars
```

SET Performance = 'High'

WHERE Car_VIN = '#F3E4NVE';

--RA for query 9:

Cars ← Cars ⋈ Car_VIN = '#F3E4NVE'

Cars ← Cars {_ Performance ← 'High'}

-- Query 10:

DELETE FROM Customers

WHERE Customer_Info = 'John Doe: 433 Wish Wash street';

--RA for query 10:

Customers ← Customers - Customer_Info = 'John Doe: 433 Wish Wash street'

-- Query 11:

SELECT AVG(TotalCost)

FROM Transactions;

--RA for query 11:

π AVG(TotalCost) (Transactions)

-- Query 12:

SELECT PickUpInfo, DropOffInfo

FROM Locations

WHERE Customer_Info = 'customer123';

--RA for query 12:

π PickUpInfo, DropOffInfo (σ Customer_Info = 'customer123' Locations)

-- Query 13:

SELECT DISTINCT Departments_Car_Types

FROM Employees

WHERE Employee_ID IN (SELECT DISTINCT Employee_ID FROM Transactions);

--RA for query 13:

π DISTINCT Departments_Car_Types (σ Employee_ID IN (π DISTINCT Employee_ID (Transactions)) Employees)

-- Query 14:

SELECT BookingDate, BookingTime, VehicleRented

FROM Bookings

WHERE BookingDate >= '2022-01-01'

ORDER BY BookingDate DESC, BookingTime DESC;

--RA for query 14:

π BookingDate, BookingTime, VehicleRented (σ BookingDate >= '2022-01-01' (γ BookingDate, BookingTime, VehicleRented (Bookings)))

-- Query 15:

SELECT Customer_Info, SUM(Revenue) AS TotalRevenue

FROM Reports

GROUP BY Customer_Info

HAVING TotalRevenue > 1000;

--RA for query 15:

π Customer_Info, SUM(Revenue) AS TotalRevenue (γ Customer_Info, SUM(Revenue) Reports) (σ SUM(Revenue) > 1000)