31Rentals Database Design Document

Group 13: Kelly Ng, Siem Russom, Peter Lemanski, and Omar Mohamed

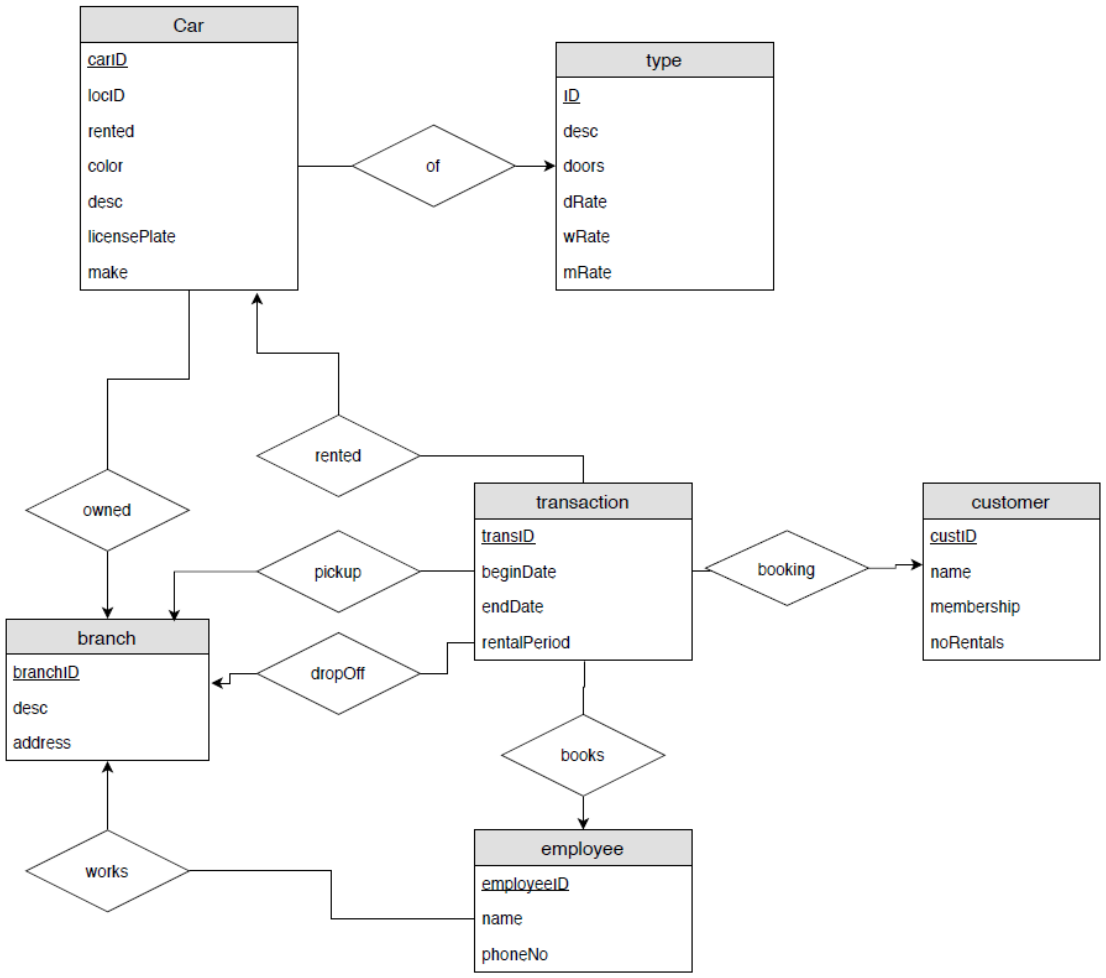
Computer 291: AS01

Instructor: Dr. Mohamad El-Hajj

Winter 2020

MacEwan University

Entity Relationship (ER) Diagram



Tables

- Cars (carID, locID, rented, color, licensePlate, make, FK typeID, FK branchID)
- Type (typeID, desc, door, dRate, wRate, mRate)
- Branch(branchID, desc, address)
- Transactions (transID, beginDate, endDate, rentalPeriod, FK carID, FK branchID, FK customerID, FK employeeID)
- Customer (customerID, name, membership, noRentals)
- Employee(employeeID, name, pNumber, FK branchID)

Foreign Keys and References

There are **7** foreign keys across our relations. Each foreign key in one relation references a primary in another relation.

Foreign Key	References
Car.typeID	Type.typeID
Car.branchID	Branch.branchID
Transaction.carID	Car.carID
Transaction.branchID	Branch.branchID
Transaction.customerID	Customer.customerID
Transaction.employeeID	Employees.employeeID
Employee.branchID	Branch.branchID

Relation Design Description

Cars Table: This relation will track of the cars inventory. Every car, regardless of which branch it belongs to will be stored in this table. The table has the following columns:

- carID (int) : A unique identifier for each entry. This is also the primary key for the Cars relation.
- locID (int): This is the current location or pickup location of the car. This a **foreign key** that references Branch.branchID.
- rented (int): The rental status of the car. 1 if rented and 0 if not.
- color (varchar(20)): color of car.

- liscencePlate (varchar(50)): license plate of car
- make (varchar(50)): Car model or make
- typeID (int): The type of car. This is a **foreign key** that references Type.typeID

The **Cars** table has the following relationships.

Relation	**Relationship**	**Relation**
Cars	**Car is of type**	Type
Cars	**Car is at/owned**	Branch
Cars	**Car is rented**	Transaction

Type Table: These table stores the types of cars available and their daily/weekly/monthly rental rates. The four types are Compact, Sedan, Crossover, SUV with their corresponding ID 1,2,3,4 respectively. The columns are:

- typeID (int): The primary key for the Types relation. The ID number corresponds with the car type as explained above.
- description(varchar(50)): Car type description in string form. Example, typeID 1 will have Compact as description
- dRate (int): Daily rate
- wRate (int): Weekly rate
- mRate (int): Monthly rate

Type table has the following relationships

Relation	Relationship	Relation
Type	**Car is of type**	Car

Branch Table: The branch table has a list of branches. Using foreign key that references Cars.carID, we can see which cars are in which branch. The columns for this table are:

- branchID (int): Primary key for the Branch relation. This is also referenced by the **Cars** and **Transactions** tables.
- Description (varchar(50)): Stores the description of the branch, mainly its name. Example, Edmonton North.
- Address (varchar(50)): the physical address of the branch.

The table has the following relationships.

Relation	Relationship	Relation
Branch	**Has cars/inventory**	Cars
Branch	**Works in this branch**	Employees
Branch	**Car rented from or dropped of at**	Transaction

Transactions Table: Keeps a record of all transactions across all branches. Rentals initiated or returns are recorded on this table. The relation has the following columns.

- transID(int): Primary key. Each transaction has a unique identifier, its transID.
- beginDate (datetime): Beginning of rental period.
- endDate (datetime): End of rental period.
- rentalPeriod (int): Difference between endDate and beginDate. Each day counts as one. This is used to calculate late fees and rates.
- carID: **Foreign key** that references Cars.carID.
- branchID: **Foreign key** that references Branch.branchID
- customerID: **Foreign key** that references Customers.customerID
- employeeID: **Foreign key** that references Employees.employeeID

the transaction table has the following relationships.

Relation	**Relationship**	**Relation**
Transactions	**Car is rented/returned**	Cars
Transactions	**Rented from/returned to**	Branch
Transactions	**Rented by**	Customers
Transactions	**Rented/Returned by**	Employees

Customers Table: The customers relation keeps a record of existing and new customers.

- customerID (int): Primary key. Each customer has a unique identifier.

- name (varchar(50)): name of customer
- membership (int): customer is a Gold member (1) or not (0).
- noRentals (int): the number of rentals. Customers with 3+ rentals are given Gold member status.

The table has the following relationship.

Relation	**Relationship**	**Relation**
Customer	**Rented**	Transaction

Employee Table: Keeps a track of employees and what branch they work in. The columns are:

- employeeID (int): Primary key. This acts as both a unique identifier and a login password.
- Name (varchar(50)): Employee name.
- branchID (int): **Foreign key** that references Branch.branchID.

The table has the following relationships.

Relation	**Relationship**	**Relation**
Employees	**Rented/Returned by**	Transactions

Integrity Constraints

Not null constraint: All 6 of our tables contain at least one column whose value cannot be Null. For each table, their corresponding Primary ID as specified in the descriptions

above cannot have Null values. Our Primary keys are used as unique identifiers for our transactions and so every row entry must have them.

We've designed our tables with a starting Primary Key value and those values are auto incremented with every entry. This is might not be ideal for enterprise level software but for the purposes of this project it serves its purpose.

Referential Integrity: As shown above, there are 7 Foreign keys across our relations. **All** our Foreign keys reference columns that are specified as **Primary key** in the table being referenced. Our Foreign Keys are not new values but rather references to values that exist as Primary keys in other tables. Our system does not update any primary keys. However, since our system allows for entries to be deleted, we've enabled the Delete on cascade option for our foreign keys. This will ensure that when a primary key being refenced by a foreign key is deleted, the foreign key will not reference to it and thus be deleted itself. We did this to avoid having foreign keys that reference NULL primary key values.