|  |
| --- |
| **BOSTON UNIVERSITY-MET** |
| **CS-669: Term Project (Iteration-5)** |
|  |
|  |
| ***Submitted by: Neha Bais*** |
| **11/20/2020** |

Contents

[Project Overview: 3](#_Toc57067905)

[Entities: 3](#_Toc57067906)

[Rent-A-Ride Structural Rules: 3](#_Toc57067907)

[Conceptual ER Diagram: 4](#_Toc57067908)

[Rent-A-Ride Attributes: 5](#_Toc57067909)

[Physical ERD: 10](#_Toc57067910)

[Normalization: 11](#_Toc57067911)

[Create Tables Scripts: 11](#_Toc57067912)

[Indexing: 11](#_Toc57067913)

[Insert Transactions for database: 13](#_Toc57067914)

[Maintaining History Tables with Triggers: 14](#_Toc57067915)

[Organization-Driven Queries: 14](#_Toc57067916)

[Summary and Reflection: 14](#_Toc57067917)

# Project Overview:

My project is basically to create a database for an application named ***“Rent-A-Ride”*** that allows users to rent a car based on car’s make and model. Generally, people look out for a temporary car when they do not own a car, or when visiting different cities/countries, or people who are waiting for their car to be repaired in order to get ease for their travel. This application allows a customer to select a car based on different car categories like SUV, Mini-Van, Sedan etc. Also, it provides a flexibility to customers to select the car based on different locations from where he can pick-up and return the car based on his ease and comfort.

Here is the basic functionality of my application. This application is used by customers who would like to rent a car. Customer first download the App or visits the website and registers himself. He will then be provided a list of cars available to rent based on his choices of car’s category and location. He then selects a car and reserves it providing the date and timeframe for which he would like to rent. Customers who don’t have a vehicle’s insurance are required to take the rental insurance plan or if they already have an insurance, they have the flexibility to buy the rental plan or not. Apart from these customers are charged a late-fee, if they return the car after the due-date.

If the customer takes the membership, he will be provided a default membership discount of 10% in addition to several discount offers offered by my system.

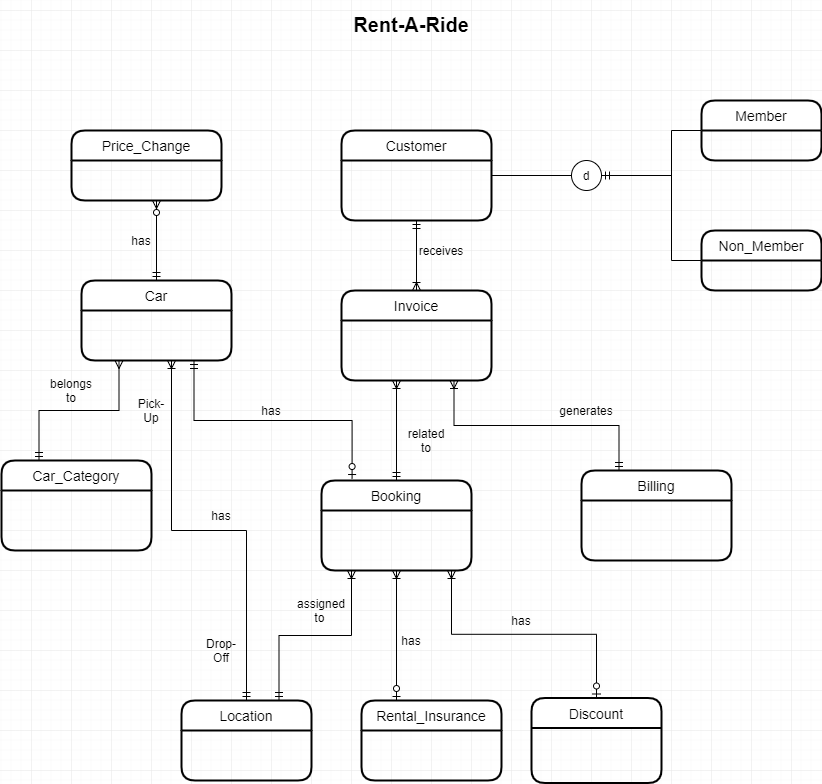
# Entities:

1. Customer
2. Member
3. Non-Member
4. Car
5. Car Category
6. Location
7. Booking
8. Discount
9. Rental Insurance
10. Invoice
11. Billing

# Rent-A-Ride Structural Rules:

1. Each car belongs to a particular category. Each category will have several cars.
2. A customer can be a member or a non-member of the application.
3. Each car is associated to a location. Each location can have single or multiple cars.
4. Each customer has one or multiple bookings. Each booking is associated to one customer.
5. A customer can have many invoices related to a booking. Each booking can have many invoices but all of them are related to a single customer only.
6. Each booking is related with only one car at a time. Each car may have none or at-most one booking (a single booking is allowed for a particular car and this car can’t be allotted to other customer unless and until the first customer cancels his booking).
7. Each booking has a pickup/drop-off location. Each location can have different bookings for pick-up and drop-off the cars.
8. Each booking can have at-most one discount. Each discount code can be applied to multiple bookings.
9. Each booking may have a rental insurance. A rental insurance may be associated to multiple bookings.
10. A single bill is generated for multiple invoices associated with a particular booking. A single bill is generated for a customer when he returns the car.

# Conceptual ER Diagram:



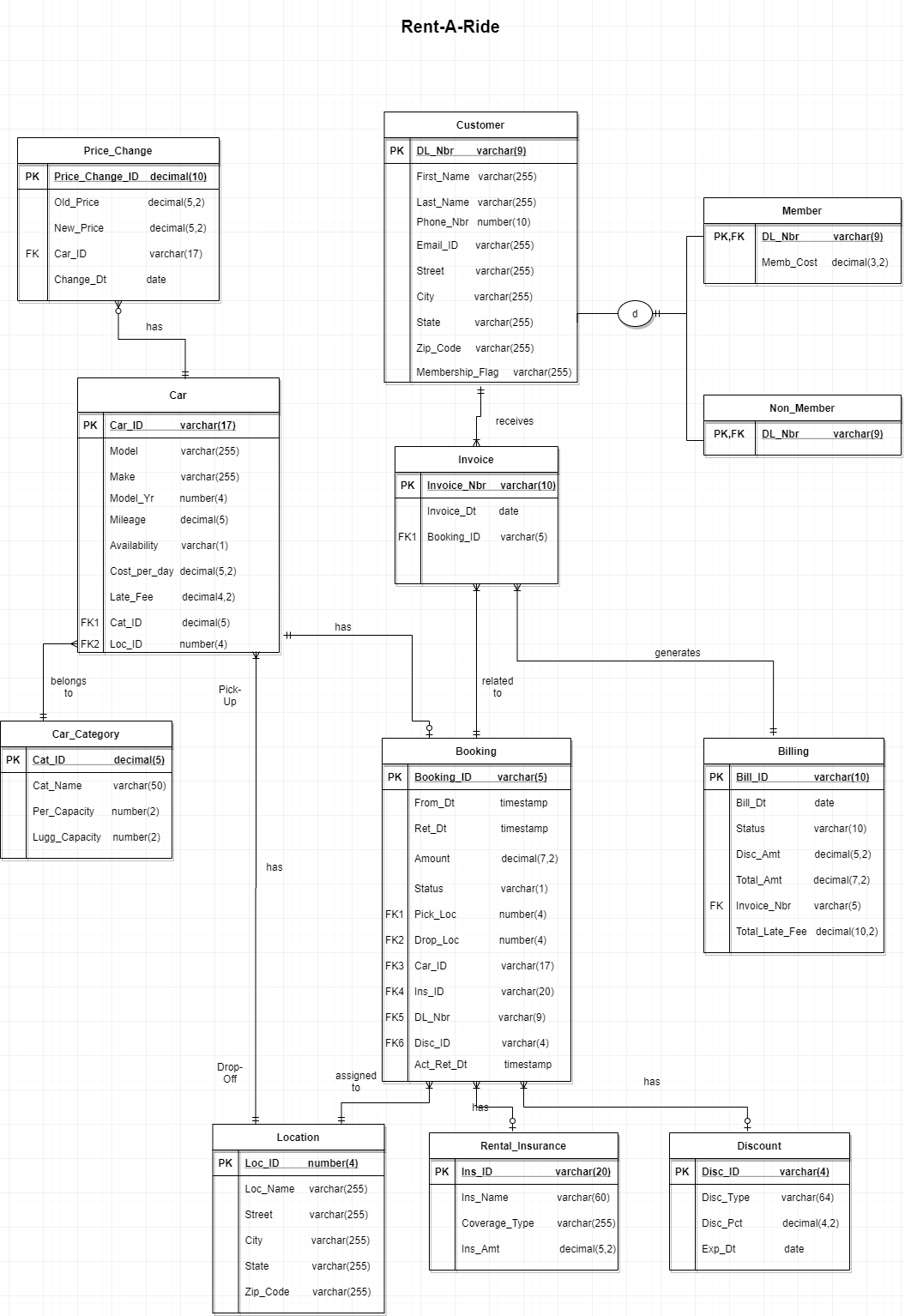
# Rent-A-Ride Attributes:

I will now add the attributes and their datatypes to each table. The choice and reasoning for each attribute is listed in below table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Table** | **Attribute** | **Datatype** | **Reasoning** |
| Customer | DL\_Nbr | varchar(9) | Every customer has a driving license which uniquely identifies him/her. DL is 9 characters, so I assigned 9. |
| First\_Name | varchar(255) | This is the first name of the customer, up to 255 characters of the  first name. |
| Last\_Name | varchar(255) | This is the last name of the customer, up to 255 characters of the  last name. |
| Phone\_Nbr | number (10) | Phone numbers are 10 digits long, so I assigned 10 here. |
| Email\_ID | varchar(255) | This is the email address of the customer. 255 characters  should be a safe upper bound. |
| Street | varchar(255) | Keep extra buffer in case of longer address |
| City | varchar(255) | City in which customer resides. Keeping an extra buffer for safer side. |
| State | varchar(255) | State in which customer resides. Keeping an extra buffer for safer side. |
| Zip\_Code | varchar(64) | Postal code of area in which customer resides. Keeping an extra buffer for safer side. |
| Membership\_Flg | varchar(1) | Indicator whether customer is a member or not. Assigned 1 character to indicate M/N. |
| Member | DL\_Nbr | varchar(9) | Every customer has a driving license which uniquely identifies him/her. DL is 9 characters, so I assigned 9. |
| Memb\_Cost | decimal(1,2) | In case customer wants to take membership, what will be the membership cost. Not charging a large membership fee, so assigned (1,2). By default it will be $5.00 |
| Non\_Member | DL\_Nbr | varchar(9) | Every customer has a driving license which uniquely identifies him/her. DL is 9 characters, so I assigned 9. |
| Car | Car\_ID | varchar(17) | Each car is uniquely identified by a VIN number which is 17 characters long including numeric and characters. |
| Model | varchar(255) | Name of the model of the car. Keeping it 255 as to keep some extra buffer. |
| Make | varchar(255) | Model make of the car. Keeping an extra buffer for name. |
| Model\_Yr | number(4) | Assigned make year of car as 4 as year will be of 4 digits. |
| Mileage | decimal(5) | On an average, a car give mileage of 12000-15000 per year. So assigning 5 digits for mileage |
| Availabilty | varchar(1) | Flag to indicate whether this car is available or not for booking. So, character 1 for this indicator. |
| Cat\_ID | decimal(5) | same as cat id for referencing a car category |
| Loc\_ID | number(4) | A number to uniquely identify a location. I assigned it as 4 as safer side, in case more locations are added to the database. |
| Cost\_per\_day | decimal(5,2) | Per day cost of the car. Can be maximum of $999 depending of which car customer choose. Assigned 3 as digits and 2 as decimal standard. |
| Late\_Fee | decimal(4,2) | How much late fee will be charged per hour. Mostly will be between $1 – $10. So, assigned (2,2) as decimal format. |
| Car\_Category | Cat\_ID | decimal(5) | To identify each car category. Keeping some buffer for safer end. |
| Cat\_Name | varchar(50) | Keeping an extra buffer for category name which can be SUV/Sedan etc… |
| Per\_Capacity | number(2) | Depending on category, a car can accommodate a maximum of 8-10 persons. So assigned 2 digits for this. |
| Lugg\_Capacity | number(2) | Luggage carrying capacity depending on category. SUV has more than sedan. Assigned 2 as a buffer. |
| Location | Loc\_ID | number(4) | A number to uniquely identify a location. I assigned it as 4 as safer side, in case more locations are added to the database. |
| Loc\_Name | varchar(255) | Keeping an extra buffer for safer side. |
| Street | varchar(255) | Keeping an extra buffer for safer side. |
| City | varchar(255) | Keeping an extra buffer for safer side. |
| State | varchar(255) | Keeping an extra buffer for safer side. |
| Zip\_Code | varchar(255) | Keeping an extra buffer for safer side. |
| Rental\_Insurance | Ins\_ID | varchar(20) | Insurance number can be a combination of characters and digits. Keeping some extra buffer for a safer bound. |
| Ins\_Name | varchar(60) | Keeping an extra buffer for safer side. |
| Coverage\_Type | varchar(255) | Keeping an extra buffer for safer side. |
| Ins\_Amt | decimal(5,2) | Mostly it is 2 digits, but I am keeping it as 3 digits to keep some extra buffer. |
| Discount | Disc\_ID | varchar(4) | Discount code, combination of digits and characters. Keeping it as varchar(4) |
| Disc\_Type | varchar(64) | Can be student discount, corporate discount etc… Keeping some extra buffer here. |
| Disc\_Pct | decimal(4,2) | Percentage can be between 1-2 digits. So kept as 2 as discount percentage. |
| Exp\_Dt | date | Date at which the discount code expires. |
| Billing | Bill\_Id | varchar(10) | Can be a combination of digits and characters. Keeping it 10 to keep some buffer. |
| Bill\_Dt | date | Date at which bill is generated |
| Status | varchar(10) | Paid/Pending. Keeping some buffer for safer bound. |
| Actual\_Ret\_Dt | date | Date at which customer actually returned the car. |
| Disc\_Amt | decimal(5,2) | Amount calculated based on discount %. Keeping some buffer to allow a discount amount of maximum 3 digits. |
| Tot\_Amt | decimal(7,2) | Total amount charged to customer. Keeping some buffer for safer end. |
| Invoice\_Nbr | varchar(10) | Allow a maximum of 5 characters as invoice number which can be a combination of characters or digits, so varchar(5) |
| Total\_Late\_Fee | decimal(10,2) | Total late fee calculated based on the actual return date which will then be added t total amount in the Bill. |
| Invoice | Invoice\_Nbr | varchar(10) | Allow a maximum of 5 characters as invoice number which can be a combination of characters or digits, so varchar(5) |
| Booking\_ID | varchar(5) | Unique id generated against each booking. Keeping it as varchar(5) as can be combination of characters and digits and will be a maximum of 5 characters. |
| Invoice\_Dt | date | Date at which invoice generated. |
| Booking | Booking\_ID | varchar(5) | Unique id generated against each booking. Keeping it as varchar(5) as can be combination of characters and digits and will be a maximum of 5 characters. |
| From\_Dt | timestamp | Date and time from which booking is confirmed. |
| Ret\_Dt | timestamp | Date and time till when booking is confirmed |
| Amount | decimal(7,2) | Amount calculated as per booking days and car type. Keeping some buffer to accommodate amount. |
| Status | varchar(20) | Confirmed/Cancelled… Keeping some buffer for safer bound. |
| Pick\_Loc | number(4) | Referring to location table, so same as Loc\_ID. |
| Drop\_Loc | number(4) | Referring to location table, so same as Loc\_ID. |
| Car\_ID | varchar(17) | Referring car entity, so same as Car\_ID |
| Insurance\_ID | varchar (20) | Same as Insurance\_ID from Rental\_Insurance table. |
| DL\_Nbr | varchar(9) | Foreign key referencing customer table |
| Disc\_ID | varchar(4) | Same as Disc\_ID from discount table. |
| Invoice\_Nbr | varchar(5) | Same as Invoice\_nbr from Invoice entity. |
| Act\_Ret\_Dt | Timestamp | To track the actual return date and time which will then be used to calculate late fee for customer |
| Price\_Change | Price\_Change\_ID | decimal(10) | Unique ID for each change |
| Old\_Price | decimal(5,2) | Old price of car before updating |
| New\_Price | decimal(5,2) | New price of car after updating |
| Car\_ID | varchar(17) | Car id for which price was changed |
| Change\_Dt | date | Date at which price changed |

# Physical ERD:

For entities, relationship and primary and foreign key constraints along with attributes.



# Normalization:

**First Normal Form**: Above ERD is in first normal form because

* Data is arranged in a regular table with similar data in the columns.
* There are no repeating groups; each cell contains one data item.
* All tables have a primary key.
* All non-key attributes depend on the primary key.

**Second Normal Form:** Above ERD is in second normal form because

* All tables are in 1NF.
* Each non-key attribute depends on the entire primary key; there are no partial dependencies.

**Third Normal Form:** Above ERD is in third normal form because

* All tables are in 2NF.
* The table contains no transitive dependencies; all dependencies are on the primary key.

# Create Tables Scripts:



# Indexing:

* Primary keys which are already indexed are mentioned below:

1. Customer.DL\_Nbr
2. Member.DL\_Nbr
3. Non\_Member.DL\_nbr
4. Location.Loc\_ID
5. Rental\_Insurance.Ins\_ID
6. Discount.Disc\_ID
7. Car\_Category.Cat\_ID
8. Car.Car\_ID
9. Booking.Booking\_ID
10. Invoice.Invoice\_Nbr
11. Billing.Bill\_ID

* All Foreign keys need an index. Below is the table identifying each foreign key column:

|  |  |  |
| --- | --- | --- |
| Column | Unique | Description |
| Car.Cat\_ID | Not Unique | There can be many cars for same car category. |
| Car.Loc\_ID | Not Unique | There can be many cars for same location. |
| Booking.Pick\_Loc | Not Unique | There can be many booking for same pick-up location. |
| Booking.Drop\_Loc | Not Unique | There can be many booking for same drop-off location. |
| Booking.Car\_ID | Unique | There will be a single booking for the same car |
| Booking.Ins\_ID | Not Unique | The same insurance can be applied to multiple bookings |
| Booking.Disc\_ID | Not Unique | The same discount can be applied to multiple bookings. |
| Invoice.Booking\_ID | Not Unique | There can be multiple invoices for a booking. |
| Invoice.DL\_Nbr | Not Unique | There can be multiple invoices for a customer |
| Billing.Invoice\_Nbr | Not Unique | There can be multiple invoices for a same bill. |

Apart from these, below are three query driven indexes:

1. It’s reasonable that there might be many queries that limit cars selection based on their availability. So, I select Car.Availabilty to be indexed. This would be non-unique because many cars could be available.
2. It’s also reasonable that many customers would want to limit their car selection based on the cost that the car is charged for one day. So, I select Car.Cost\_per\_day to be indexed. This would be non-unique because it might be possible that different cars have same price.
3. It is also reasonable if a customer wants to take a rental insurance but wants to select it based on the insurance amount, maybe he wants to take insurance with minimum insurance cost. So, I select Rental\_Insurance.Ins\_Amt to be indexed. This would also be non-unique as there might be different insurance for same cost.



# Insert Transactions for database:

The first 2 use-cases for Rent-A-Rideare listed below:

*Customer App/Website Registration*:

1. The person visits the website or download the application.
2. The application asks them to enter their details and get registered.
3. The user will be provided an option to take the membership or not while registering.
4. Once the customer entered all his details, he will get registered to the application.
5. *A customer can be a member or a non-member of the application.*

*Car category look up Use Case:*

1. Customer is able to see different car categories as per his requirement and comfort.
2. Also, can find the cost of renting a car per day belonging to a particular category.
3. Customer can also see how much late fee will be charged if he is not able to return the car at the specified time.

For these use-cases, I will implement a transaction that creates a Member and Non-Member, Customer and Car\_Category table transactions using the store procedure.

The screenshots for the procedure implementation are attached below as well as the script contains the Procedure and normal Insert statements for remaining tables.

Also, here I have attached the some more procedures and triggers scripts for tables which are dependent on each other.

1. **Create\_Invoice\_Trigger.txt:** Trigger to make an entry in Invoice table as soon as a booking is made for a car.
2. **Generate\_billing\_trigger.txt:** Contains two stored procedures. One is to calculate discount amount for customers and the other one is to calculate the Late fee in case the customer returns the car after the return date committed at time of booking.

This file also contains a trigger Gnerate\_Billing which will make an entry in Billing table with the discount amount and late fee calculated as per above procedures.

1. **Update\_Car\_details\_trigger.txt:** This file contains a trigger script for updating the Availability flag in Car table once a car is returned by customer or booking is cancelled by customer.





# Maintaining History Tables with Triggers:

For this task, I am thinking of tracking the price changes for a car. This will help the business to track what kind of cars have the most price changes and then can conclude like which cars are making them profit or loss.

Let’s say, a price drop will occur when the car is in system for a long time and customer are not choosing that one for their use, so in order to make business for that car, business decided to drop its price so as to attract a customer for lower price car. Later on, when generating the revenue report for business, if they want to know which cars prices were modified, they can easily fetch the data from this history table.

The new structural rule for this will be: Each car can have many price changes, and each price change is for one car.

The conceptual and physical ERD are also modified above for this implementation and the updated one can be seen in this document above.

The screenshot for trigger creation and price changes is attached below:

# Organization-Driven Queries:

I have implemented below 3 queries which may be asked by the organization:

1. My organization wants to send some additional offers to those customers who have booked a Luxury car. For this purpose, they want all these customers first name, last name and phone number.
2. The organization wants to know the number of bookings for each location as well as the location details.
3. The organization wants to know the details of those bookings made after the price change date for those cars whose price have been changed.

Below is the screenshot for these 3-query implementations along with results:



**Script execution order:**

1. Create\_Tables\_Script
2. Create\_Index\_Script
3. history\_table\_trigger
4. Create\_Invoice\_Trigger
5. Insert\_Data\_Scripts
6. update\_car\_details\_trigger
7. generate\_billing\_trigger

# Summary and Reflection:

My database is for an application named “***Rent-A-Ride***” which allows user to rent a car based on his choices of car’s category, pick-up and drop-off location. This app will also avail an additional discount of 10% when a customer takes its membership in addition to various discount offered by my application. It will provide invoice to the user when picking the car and will generate a total bill for customer when he returns the car.

The structural database rules and conceptual ERD contains the main entities of my design and the relationship between them. It also contains a hierarchy of customer to indicate whether he is a member or non-member of my application. The physical ERD is also created from the conceptual one, which contains the same entities and relationships in addition to the primary and foreign key constraint along with various attributes.

The SQL script that creates all tables and also import indexes have been created to help speed up access to my database are also attached in this document. Stored procedures have been implemented to create transactions for some of my tables of the database. Some triggers are also implemented in order to maintain the dependency between the different entities f my database. A history table for tracking the price change of a car for a specific month is also implemented.

As now the database is fully functional, I can say it has been a long journey but a satisfactory one. In the overall process, I found the implementation of triggers to maintain the tables dependency quite challenging one, but I am happy to see that it is working perfectly fine now. There can be some improvements for my database, but for now I feel that this is quite a solid foundation to move forward with.