**Project team 11**

**A database system for OLA - Transportation Network Company**

Team Members: Riddhi Jagdish Patil

Riddhi Sanjay Panchal

Rohit Reddy Mullangi

Raj Hitendra Jani

***PROJECT PROPOSAL***

**Objective**

The objective is to create a database application for a transportation network company. This system will provide hassle-free transportation facilities to customers by allowing them to book a cab using Ola mobile application. Transport facility provided on this platform can range from AC luxury cars to micro range cars as well as localized offerings like the ubiquitous Auto-rickshaws to Shuttle buses for the daily commute. It will connect across multiple cities in the country providing over 100 vehicles for transportation.

This system will empower hundreds of thousands of driver-partners as entrepreneurs, by building an ecosystem encompassing financing institutions, car manufacturers, service providers etc. for drivers to grow professionally and personally as well as a consistent earning opportunity for them on the Ola platform.

Also, it will allow car owners to register their cars to provide Ola service to customers and gain a steady source of income.

**Context**

The main idea of this project is to build a database system for Ola transportation Network Company. This system will store all the information such as drivers of the system, the car information, owner information etc. which is required by the application to allow the customer to book a cab. The database will also store customer related information like ride history, wallet information, payment modes, most frequent routes of customers, feedback received from customer etc. This entire information will allow Ola service to serve the customer efficiently. In addition to this database can also store information like car insurance, promotional codes for ride booking, statistics of profit, rides cancelled, damaged cars etc. This additional information can be helpful for Ola promotion and statistical analysis.

**Scope**

Following is the scope of the project:

* The database system will store and process following:-
* Driver information along with driver's schedule.
* Details of available cars.
* Information about owners of the cars.
* Customer information along with details of his payment modes
* Ride history of the driver, customers, and car as a whole for tracking purpose.
* Promotional codes available for booking of cars.
* Feedback information of customers provided for Ola service and drivers.
* Billing information of each customer. Calculation of the drivers working hours and

earnings.

* Storing various statistics like percentage of rides that cancel, payments per driver per

month, overall profit done by the owner, company profit/loss etc.

* We will also store car damage or accidents reported information in the later stage of

software development.

* Storing of wallet information for customer.
* Insurance information of the car.

### ***ENVIRONMENT SETUP***

Database client: MySQL Workbench version 8.0.12

Database server: Amazon Web Services RDS server

Web - Application Server: Apache Tomcat Server version 7.0.91

Integrated Development Environment: Eclipse version Mars

JDK: JDK version 1.8.0\_181

Database - application connection: JDBC - ODBC Bridge driver

Application Web browser: Google Chrome, Mozilla Firefox

Programming languages: Core Java, HTML, CSS, JavaScript, JSP, Servlet

## 

## ***HIGH LEVEL REQUIREMENTS***

## **Initial user roles**

|  |  |
| --- | --- |
| USER ROLE | DESCRIPTION |
| Car Owner | They can register their to Ola Service |
| Driver | They drive registered cars in Ola system |
| Customer | They use the Ola Service to book a ride |
| System Admin | She/he has admin access to modify, insert, delete entries in database |

### **Initial user story descriptions**

|  |  |
| --- | --- |
| Story ID | Story Description |
| US 1 | As a Customer, I want to store my payment information |
| US 2 | As a Customer, I want to register to Ola cab service |
| US 3 | As a Customer, I want to login to the app |
| US 4 | As a Customer, I want to book for a ride |
| US 5 | As a Customer, I want to see the available car options |
| US 6 | As a Car owner, I want to register my car to Ola service |
| US 7 | As a Car owner, I want to check my Ola profit |
| US 8 | As a system admin, I want to add driver information so that they can be registered with Ola |
| US 9 | As an system admin, I want to add owner information so that they can rent their car |
| US 10 | As a driver I can view the car allotted to me. |
| US 11 | As a driver I can view my salary |
| US 12 | As a driver I can view number of hours worked |
| US 13 | As a driver I can view customer information and pick up and drop off locations |
| US 14 | As a customer I want to give feedback |

### 

## ***HIGH LEVEL CONCEPTUAL DESIGN***

**Entities**

Car

Driver

CarOwner

Customer

Ride

PaymentDetail

CustomerFeedback

**Relationships**

* CarOwner registers Car
* Customer stores PaymentDetail
* Customer books Car
* Driver accesses Car
* Driver views Customer
* Driver views Ride
* Driver views PaymentDetail
* Customer gives CustomerFeedback

# **Sprint 1**

***DETAILED CONCEPTUAL DESIGN***

Entity: **Customer**

Attributes:

customer\_id

name [ composite ]

first\_name

last\_name

password

email\_id

phone\_number

date\_of\_birth

age [ derived ]

card\_number [ multi-valued ]

gender

Entity: **Driver**

Attributes:

driver\_id

name [ composite ]

first\_name

last\_name

password

email\_id

address [ composite ]

address\_line1

address\_line2

city

state

zip\_code

phone\_number

license\_number

date\_of\_birth

age [ derived ]

gender

Entity: **CarOwner**

Attributes:

owner\_id

name [ composite ]

first\_name

last\_name

address [ composite ]

address\_line1

address\_line2

city

state

zip\_code

password

gender

Entity: **Car**

Attributes:

car\_number

car\_make

car\_model

car\_color

car\_type

car\_capacity

Entity: **Ride**

Attributes:

ride\_id

start\_time

end\_time

duration [ derived ]

pickup\_location

drop\_location

distance [ derived ]

fare [ derived ]

Relationship: **Customer** searches **Car**

Cardinality: One to Many

Participation:

Customer has a partial participation

Car has a partial participation

Note: No need to maintain this relation in database

Relationship: **Customer** books **Ride**

Cardinality: One to Many

Participation:

Customer has partial participation

Ride has total participation

Relationship: **Driver** facilitates a **Ride**

Cardinality: One to Many

Participation:

Driver has partial participation

Ride has total participation

Relationship: **CarOwner** registers **Car**

Cardinality: One to Many

Participation:

CarOwner has total participation

Car has total participation

Justification: CarOwner has total participation because he/she cannot register into system without registering car into system.

Relationship: **Driver** drives **Car**

Cardinality: One to one

Participation:

Driver has partial participation

Car has total participation

Justification: Driver has partial participation because every registered driver does not get a car assigned

## ***LOGICAL DESIGN***

Table: **Customer**

Columns:

customer\_id

first\_name

last\_name

password

email\_id

phone\_number

date\_of\_birth

gender

Note:Password will be stored in encrypted format.

Table: **CustomerCard**

Columns:

card\_number

customer\_id[foreign key; references **customer\_id** of **Customer**]

card\_type

card\_name

card\_expiry

card\_cvv

Note:card\_number will be stored in encrypted format.

Table: **Driver**

Columns:

driver\_id

first\_name

last\_name

password

email\_id

address\_line1

address\_line2

city

state

zip\_code

phone\_number

license\_number

date\_of\_birth

gender

Note:Password will be stored in encrypted format.

Table: **Car**

Columns:

car\_number

car\_model

car\_color

car\_type[foreign key; references **car\_type** of **RateChart**]

driver\_id[foreign key; references **driver\_id** of **Driver**]

owner\_id[foreign key; references **owner\_id** of **CarOwner**]

Table: **RateChart**

Columns:

car\_type

car\_capacity

rate\_per\_mile

Table: **CarOwner**

Columns:

owner\_id

first\_name

last\_name

password

address\_line1

address\_line2

city

state

zip\_code

gender

Table: **Ride**

Columns:

ride\_id

customer\_id[foreign key; references **customer\_id** of **Customer**]

car\_number[foreign key; references **car\_number** of **Car**]

driver\_id[foreign key; references **driver\_id** of **Driver**]

start\_time

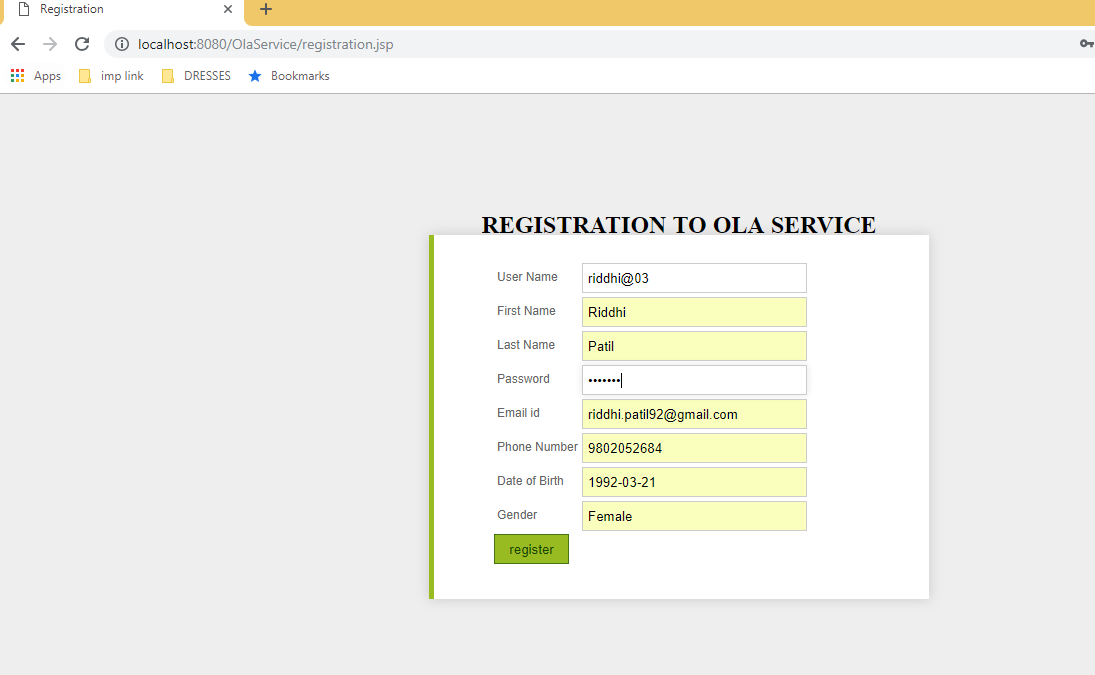
end\_time

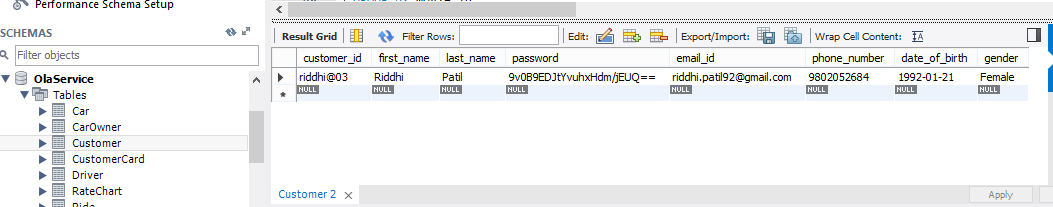
pickup\_location

drop\_location

***SQL QUERIES DEMO***

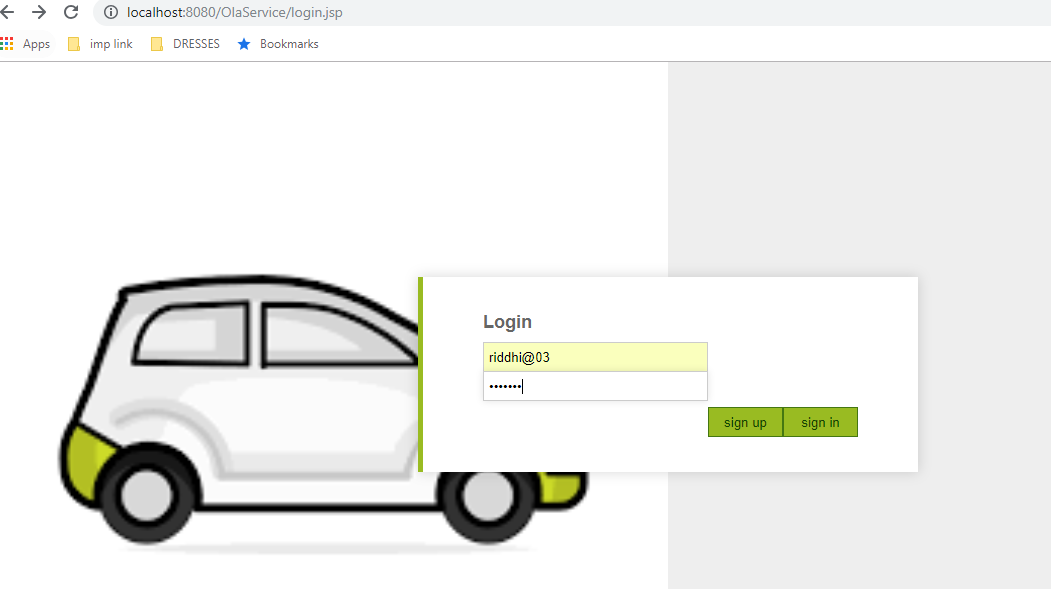
**Query:-** insert into Customer values('riddhi@03', 'Riddhi', 'Patil', '9v0B9EDJtYvuhxHdm/jEUQ==', 'riddhi.patil92@gmail.com', '9802052684', '1992-01-21', 'Female');





**Query: -** select password from Customer where customer\_id=’riddhi@03’;

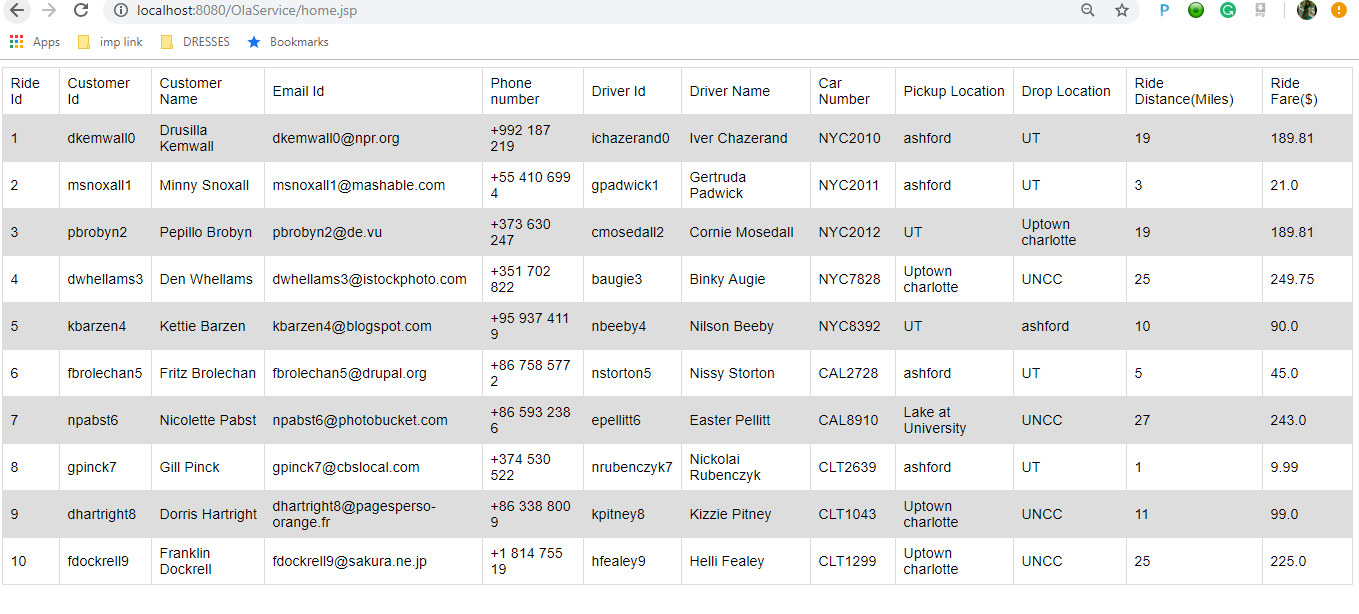
(Match the selected password with the one entered into UI, if matched display next screen)



**Query:-** select ride\_id, r.customer\_id, concat\_ws(' ',c.first\_name,c.last\_name) CustomerName, c.email\_id, c.phone\_number, r.driver\_id, concat\_ws(' ',d.first\_name,d.last\_name) DriverName, car\_number, pickup\_location, drop\_location,FLOOR(RAND(2)\*30) as Distance from Ride r inner join Customer c on r.customer\_id = c.customer\_id inner join Driver d on r.driver\_id = d.driver\_id;

This query will be executed in a loop for different car\_numbers.

select rate\_per\_mile from RateChart where car\_type = (select car\_type from Car where car\_number = ?);



**Sprint 2**

### **Initial user story descriptions**

|  |  |
| --- | --- |
| Story ID | Story Description |
| US 1 | As a Customer, I want to store my payment information |
| US 2 | As a Customer, I want to register to Ola cab service |
| US 3 | As a Customer, I want to login to the app |
| US 4 | As a Customer, I want to book for a ride |
| US 5 | As a Customer, I want to see the available car options |
| US 6 | As a Car owner, I want to register my car to Ola service |
| US 7 | As a Car owner, I want to check my Ola profit |
| US 8 | As a system admin, I want to add driver information so that they can be registered with Ola |
| US 9 | As an system admin, I want to add owner information so that they can rent their car |
| US 10 | As a driver I can view the car allotted to me. |
| US 11 | As a driver I can view my salary |
| US 12 | As a driver I can view number of hours worked |
| US 13 | As a driver I can view customer information and pick up and drop off locations |
| US 14 | As a customer I want to give feedback to driver |

***DETAILED CONCEPTUAL DESIGN***

Entity: **Customer**

Attributes:

customer\_id

name [ composite ]

first\_name

last\_name

password

email\_id

phone\_number

date\_of\_birth

age [ derived ]

card\_number [ multi-valued ]

gender

Entity: **Driver**

Attributes:

driver\_id

name [ composite ]

first\_name

last\_name

password

email\_id

address [ composite ]

address\_line1

address\_line2

city

state

zip\_code

phone\_number

license\_number

date\_of\_birth

age [ derived ]

gender

Entity: **CarOwner**

Attributes:

owner\_id

name [ composite ]

first\_name

last\_name

address [ composite ]

address\_line1

address\_line2

city

state

zip\_code

password

gender

Entity: **Car**

Attributes:

car\_number

car\_make

car\_model

car\_color

car\_type

car\_capacity

Entity: **Ride**

Attributes:

ride\_id

start\_time

end\_time

duration [ derived ]

pickup\_location

drop\_location

distance [ derived ]

fare [ derived ]

Relationship: **Customer** searches **Car**

Cardinality: One to Many

Participation:

Customer has a partial participation

Car has a partial participation

Note: No need to maintain this relation in database

Relationship: **Customer** books **Ride**

Cardinality: One to Many

Participation:

Customer has partial participation

Ride has total participation

Relationship: **Driver** facilitates a **Ride**

Cardinality: One to Many

Participation:

Driver has partial participation

Ride has total participation

Relationship: **CarOwner** registers **Car**

Cardinality: One to Many

Participation:

CarOwner has total participation

Car has total participation

Justification: CarOwner has total participation because he/she cannot register into system without registering car into system.

Relationship: **Driver** drives **Car**

Cardinality: One to one

Participation:

Driver has partial participation

Car has total participation

Justification: Driver has partial participation because every registered driver does not get a car assigned

Relationship: **Customer** gives feedback to **Driver**

Cardinality: Many to Many

Participation:

Customer has partial participation

Driver has total participation

## **LOGICAL DESIGN WITH NORMAL FORM IDENTIFICATION**

Table: **Customer**

Columns:

customer\_id

first\_name

last\_name

password

email\_id

phone\_number

date\_of\_birth

gender

Note:Password will be stored in encrypted format.

Highest normalization level: 4NF

Table: **CustomerCard**

Columns:

card\_number

customer\_id[foreign key; references **customer\_id** of **Customer**]

card\_type

card\_name

card\_expiry

card\_cvv

Note:card\_number will be stored in encrypted format.

Highest normalization level: 4NF

Table: **Driver**

Columns:

driver\_id

first\_name

last\_name

password

email\_id

address\_line1

address\_line2

city

state

zip\_code

phone\_number

license\_number

date\_of\_birth

gender

Note:Password will be stored in encrypted format.

Highest normalization level: 2NF

Justification (if below 4NF): We are leaving the table in 2NF because in practice that may not be useful because real address data is inconsistent and parts may be missing. Hence, we are not enforcing 4nf in table

Table: **Car**

Columns:

car\_number

car\_model

car\_color

car\_type[foreign key; references **car\_type** of **RateChart**]

driver\_id[foreign key; references **driver\_id** of **Driver**]

owner\_id[foreign key; references **owner\_id** of **CarOwner**]

Highest normalization level: 4NF

Table: **RateChart**

Columns:

car\_type

car\_capacity

rate\_per\_mile

Highest normalization level: 4NF

Table: **CarOwner**

Columns:

owner\_id

first\_name

last\_name

password

address\_line1

address\_line2

city

state

zip\_code

gender

Highest normalization level: 2NF

Justification (if below 4NF): We are leaving the table in 2NF because in practice that may not be useful because real address data is inconsistent and parts may be missing.hence we are not enforcing 4nf in table

Table: **Ride**

Columns:

ride\_id

customer\_id[foreign key; references **customer\_id** of **Customer**]

car\_number[foreign key; references **car\_number** of **Car**]

driver\_id[foreign key; references **driver\_id** of **Driver**]

start\_time

end\_time

pickup\_location

drop\_location

Highest normalization level: 4NF

Table:**Feedback**

Columns:

feedback\_id

customer\_id[foreign key; references **customer\_id** of **Customer**]

driver\_id[foreign key; references **driver\_id** of **Driver**]

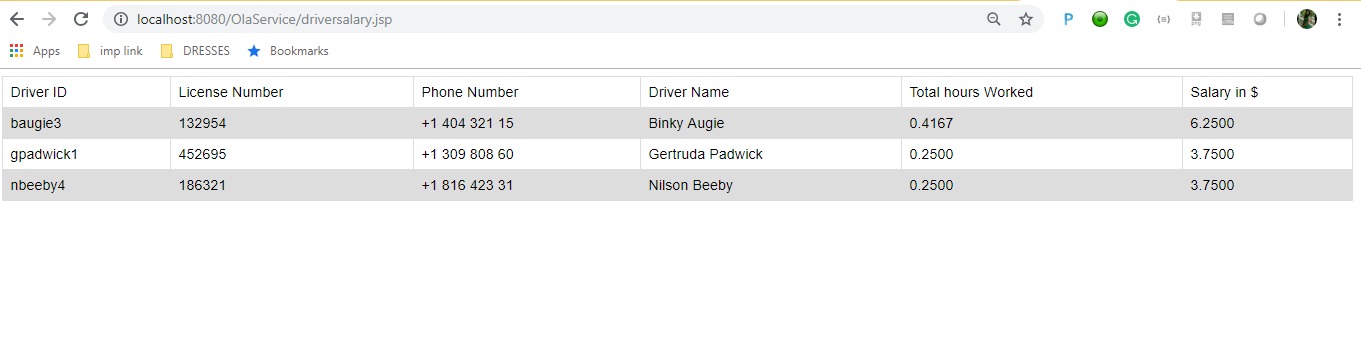
number\_of\_stars

Comment

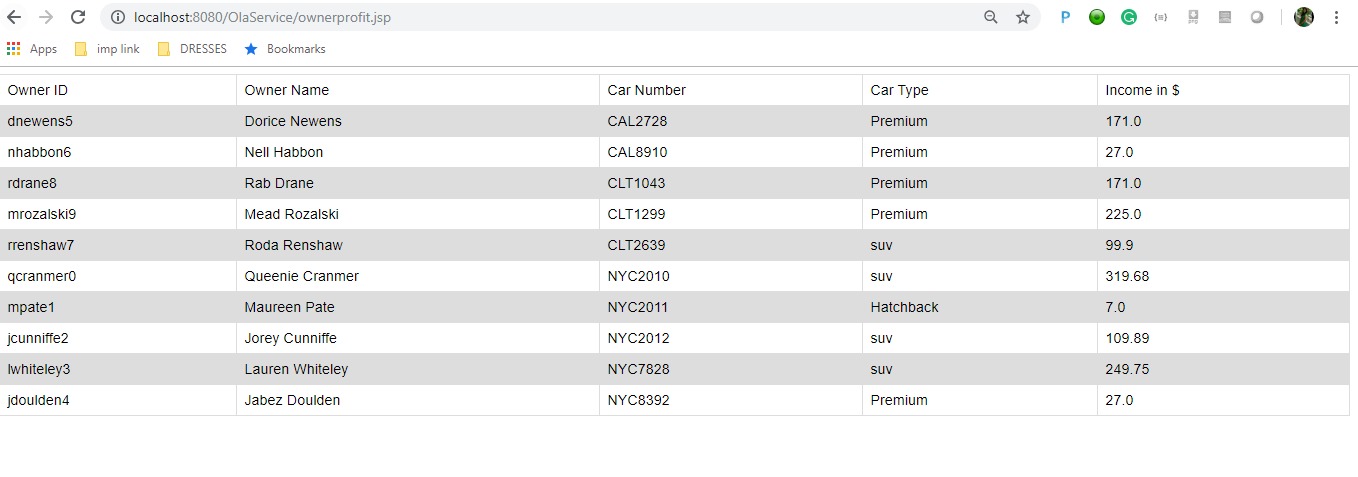
Highest normalization level: 4NF

**DEMO**

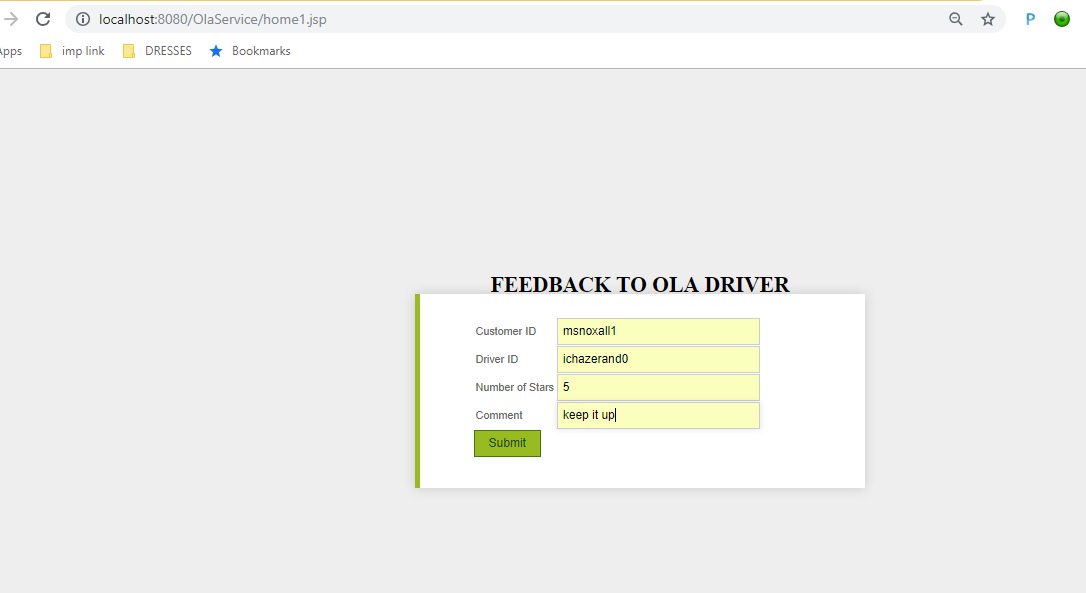
CREATE ALGORITHM=UNDEFINED DEFINER=`Riddhi`@`%` SQL SECURITY DEFINER VIEW `driver\_salary` AS select `d`.`driver\_id` AS `driver\_id`,`d`.`license\_number` AS `license\_number`,`d`.`phone\_number` AS `phone\_number`,concat\_ws(' ',`d`.`first\_name`,`d`.`last\_name`) AS `CONCAT\_WS(' ',d.first\_name, d.last\_name)`,(sum(timestampdiff(SECOND,`r`.`start\_time`,`r`.`end\_time`)) / 3600) AS `duration`,(sum(timestampdiff(SECOND,`r`.`start\_time`,`r`.`end\_time`)) \* (15 / 3600)) AS `salary` from (`Driver` `d` join `Ride` `r` on(((`d`.`driver\_id` = `r`.`driver\_id`) and (`r`.`start\_time` > '2018-11-01 00:00:00') and (`r`.`end\_time` < '2018-11-31 00:00:00')))) group by `d`.`driver\_id`

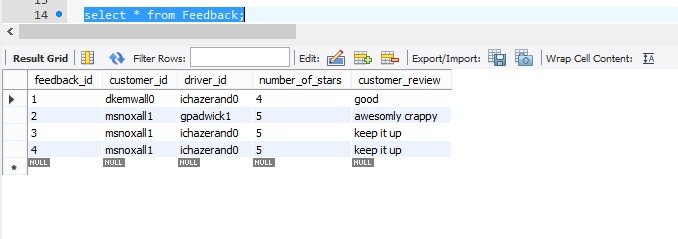


CREATE ALGORITHM=UNDEFINED DEFINER=`Riddhi`@`%` SQL SECURITY DEFINER VIEW `owner\_profit` AS select `c`.`owner\_id` AS `owner\_id`,concat\_ws(' ',`co`.`first\_name`,`co`.`last\_name`) AS `CONCAT\_WS(' ',co.first\_name,co.last\_name)`,`r`.`car\_number` AS `car\_number`,`c`.`car\_type` AS `car\_type`,sum((floor((rand(2) \* 30)) \* (select `rc`.`rate\_per\_mile` from `RateChart` `rc` where (`rc`.`car\_type` = `c`.`car\_type`)))) AS `Income` from ((`Car` `c` join `Ride` `r` on((`c`.`car\_number` = `r`.`car\_number`))) join `CarOwner` `co` on((`co`.`owner\_id` = `c`.`owner\_id`))) group by `r`.`car\_number`



insert into Feedback(customer\_id,driver\_id,number\_of\_stars,customer\_review) values('msnoxall1','ichazerand0',5,'Keep it up');





**Sprint 3**

**Initial user story descriptions**

|  |  |
| --- | --- |
| Story ID | Story Description |
| US 1 | As a Customer, I want to store my payment information |
| US 2 | As a Customer, I want to register to Ola cab service |
| US 3 | As a Customer, I want to login to the app |
| US 4 | As a Customer, I want to book for a ride |
| US 5 | As a Customer, I want to see the available car options |
| US 6 | As a Car owner, I want to register my car to Ola service |
| US 7 | As a Car owner, I want to check my Ola profit |
| US 8 | As a system admin, I want to add driver information so that they can be registered with Ola |
| US 9 | As a system admin, I want to add owner information so that they can rent their car |
| US 10 | As a driver I can view the car allotted to me. |
| US 11 | As a driver I can view my salary |
| US 12 | As a driver I can view number of hours worked |
| US 13 | As a driver I can view customer information and pick up and drop off locations |
| US 14 | As a customer I want to give feedback to Driver |
| US 15 | As a customer I want to take a subscription |
| US 16 | As a customer I want to use promotion codes on my rides |
| US 17 | As a Admin, I want to view ride history of drivers |
| US 18 | As a customer I want to view premium owners associated to my system |

## CONCEPTUAL DESIGN

Entity: **Customer**

Attributes:

customer\_id

name [ composite]

first\_name

last\_name

password

email\_id

phone\_number

date\_of\_birth

age [ derived]

card\_number [ multi-valued]

gender

Entity: **Driver**

Attributes:

driver\_id

name [ composite]

first\_name

last\_name

password

email\_id

address [ composite]

address\_line1

address\_line2

city

state

zip\_code

phone\_number

license\_number

date\_of\_birth

age [ derived]

gender

rating

Entity: **CarOwner**

Attributes:

owner\_id

name [ composite]

first\_name

last\_name

address [ composite]

address\_line1

address\_line2

city

state

zip\_code

password

gender

Entity: **Car**

Attributes:

car\_number

car\_make

car\_model

car\_color

car\_type

car\_capacity

Entity: **Ride**

Attributes:

ride\_id

start\_time

end\_time

duration [ derived]

pickup\_location

drop\_location

distance [ derived]

fare [ derived]

Entity: **Promotion**

Attributes:

voucher\_code

validity\_start\_date

validity\_end\_date

discount

max\_discount

Entity: **Subscriptions**

Attributes:

subscription\_id

period

subscription\_cost

availability

Relationship: **Customer** searches **Car**

Cardinality: One to Many

Participation:

Customer has a partial participation

Car has a partial participation

Note: No need to maintain this relation in database

Relationship: **Customer** books **Ride**

Cardinality: One to Many

Participation:

Customer has partial participation

Ride has total participation

Relationship: **Driver** facilitates a **Ride**

Cardinality: One to Many

Participation:

Driver has partial participation

Ride has total participation

Relationship: **CarOwner** registers **Car**

Cardinality: One to Many

Participation:

CarOwner has total participation

Car has total participation

Justification: CarOwner has total participation because he/she cannot register into system without registering car into system.

Relationship: **Driver** drives **Car**

Cardinality: One to one

Participation:

Driver has partial participation

Car has total participation

Justification: Driver has partial participation because every registered driver does not get a car assigned

Relationship: **Customer** gives feedback to **Driver**

Cardinality: Many to Many

Participation:

Customer has partial participation

Driver has partial participation

Relationship: **Customer** uses **Promotion**

Cardinality: Many to Many

Participation:

Customer has partial participation

Promotion has partial participation

Relationship: **Customer** takes **Subscription**

Cardinality: Many to Many

Participation:

Customer has partial participation

Subscription has partial participation

## LOGICAL DESIGN WITH HIGHEST NORMAL FORMS AND INDEXES

Table: **Customer**

Columns:

customer\_id

first\_name

last\_name

password

email\_id

phone\_number

date\_of\_birth

gender

Note: Password will be stored in encrypted format.

Highest normalization level: 4NF

Indexes:

Index 1: clustered

Columns: customer\_id

Justification: It is the default index since it is the primary key of the table.

Index 2: non-clustered

Columns: first\_name, last\_name

Justification: The Customername index has been created on columns first\_name and last\_name as these columns has been used in a lot of queries so it will improve performance of queries and increase the efficiency by providing the result quicker than it would have been without the Customername index.

Index 3: non-clustered

Columns: date\_of\_birth

Justification: The dob index has been created on column date\_of\_birth as it will be used frequently for searching customers and providing them with vouchers, this will reduce the time required to execute the queries and increase the efficiency by providing quicker results.

Table: **CustomerCard**

Columns:

card\_number

customer\_id[foreign key; references **customer\_id** of **Customer**]

card\_type

card\_name

card\_expiry

card\_cvv

Note: card\_number will be stored in encrypted format.

Highest normalization level: 4NF

Indexes:

Index 1: clustered

Columns: card\_number

Justification: It is the default index since it is the primary key of the table.

Index 2: non-clustered

Columns: customer\_id

Justification: It is the default foreign-key index since it is the foreign key of the table.

Table: **Driver**

Columns:

driver\_id

first\_name

last\_name

password

email\_id

address\_line1

address\_line2

city

state

zip\_code

phone\_number

license\_number

date\_of\_birth

gender

Note: Password will be stored in encrypted format.

Highest normalization level: 2NF

Justification (if below 4NF): We are leaving the table in 2NF because in practice that may not be useful because real address data is inconsistent, and parts may be missing. Hence, we are not enforcing 4nf in table.

Indexes:

Index 1: clustered

Columns: driver\_id

Justification: It is the default index since it is the primary key of the table.

Index 2: non-clustered

Columns: zip\_code

Justification: The zipcode1 index has been created on column zip\_code as it will be used frequently for searching drivers based on their location this will reduce the time required to execute the queries and increase the efficiency by providing quicker results.

Table: **Car**

Columns:

car\_number

car\_model

car\_color

car\_type[foreign key; references **car\_type** of **RateChart**]

driver\_id[foreign key; references **driver\_id** of **Driver**]

owner\_id[foreign key; references **owner\_id** of **CarOwner**]

Highest normalization level: 4NF

Indexes:

Index 1: clustered

Columns: car\_number

Justification: It is the default index since it is the primary key of the table.

Index 2: non-clustered

Columns: car\_type

Justification: It is the default foreign-key index since it is the foreign key of the table.

Index 3: non-clustered

Columns: driver\_id

Justification: It is the default foreign-key index since it is the foreign key of the table.

Index 4: non-clustered

Columns: owner\_id\_idx

Justification: It is the default foreign-key index since it is the foreign key of the table.

Table: **RateChart**

Columns:

car\_type

car\_capacity

rate\_per\_mile

Highest normalization level: 4NF

Index 1: clustered

Columns: car\_type

Justification: It is the default index since it is the primary key of the table.

Table: **CarOwner**

Columns:

owner\_id

first\_name

last\_name

password

address\_line1

address\_line2

city

state

zip\_code

gender

Highest normalization level: 2NF

Justification (if below 4NF): We are leaving the table in 2NF because in practice that may not be useful because real address data is inconsistent, and parts may be missing. Hence, we are not enforcing 4nf in table.

Indexes:

Index 1: clustered

Columns: owner\_id

Justification: It is the default index since it is the primary key of the table.

Index 2: non-clustered

Columns: first\_name, last\_name

Justification: The Ownername index has been created on columns first\_name and last\_name as these columns has been used in a lot of queries so it will improve performance of queries and increase the efficiency by providing the result quicker than it would have been without the Ownername index.

Table: **Ride**

Columns:

ride\_id

customer\_id[foreign key; references **customer\_id** of **Customer**]

car\_number[foreign key; references **car\_number** of **Car**]

driver\_id[foreign key; references **driver\_id** of **Driver**]

start\_time

end\_time

pickup\_location

drop\_location

voucher\_code[foreign key; references **voucher\_code** of **Promotion**]

Highest normalization level: 4NF

Indexes:

Index 1: clustered

Columns: ride\_id

Justification: It is the default index since it is the primary key of the table.

Index 2: non-clustered

Columns: customer\_id

Justification: It is the default foreign-key index since it is the foreign key of the table.

Index 3: non-clustered

Columns: car\_number

Justification: It is the default foreign-key index since it is the foreign key of the table.

Index 4: non-clustered

Columns: driver\_id

Justification: It is the default foreign-key index since it is the foreign key of the table.

Index 5: non-clustered

Columns: voucher\_code

Justification: It is the default foreign-key index since it is the foreign key of the table.

Table:**Feedback**

Columns:

feedback\_id

customer\_id[foreign key; references **customer\_id** of **Customer**]

driver\_id[foreign key; references **driver\_id** of **Driver**]

number\_of\_stars

Comment

Highest normalization level: 4NF

Indexes:

Index 1: clustered

Columns: feedback\_id

Justification: It is the default index since it is the primary key of the table.

Index 2: non-clustered

Columns: Customer\_id

Justification: It is the default foreign-key index since it is the foreign key of the table.

Index 3: non-clustered

Columns: driver\_id

Justification: It is the default foreign-key index since it is the foreign key of the table.

Table: **Promotion**

Columns:

voucher\_code

validity\_start\_date

validity\_end\_date

discount

max\_discount

Highest normalization level: 4NF

Indexes:

Index 1: clustered

Columns: voucher\_code

Justification: It is the default index since it is the primary key of the table.

Index 2: non-clustered

Columns: discount

Justification: The promotion\_discount index is created as it is used in the view which calculates total fare of the ride history and also bill generation by applying discount.

Table: **Subscribers**

Columns:

customer\_id [foreign key; references customer\_id of **Customer**]

subscription\_id [foreign key; references subscription\_id of **Subscription**]

start\_date

expiry\_date

**Justification for Primary Key**: This is a table which is made using cross reference approach hence we are combinedly using the primary keys of both the tables as the primary key in this table.

Highest normalization level: 4NF

Indexes:

Index 1: clustered

Columns: customer\_id, subscription\_id

Justification: It is the default index since it is the primary key of the table.

Table: **Subscription**

Columns:

subscription\_id

period

subscription\_cost

availability

Highest normalization level: 4NF

Indexes:

Index 1: clustered

Columns: subscription\_id

Justification: It is the default index since it is the primary key of the table.

## VIEWS AND STORED PROGRAMS

List the views relevant to your application here. Use the format specified below.

**View 1**: ride\_history

Goal: We are creating the ride\_history view that will contain the information such as full name of the customers and drivers as well as derived attributes such as total distance of ride, fare before discount and after discount. This will be used by the ola company to check the ride history of all the customers along with key details.

**View 2**: premium\_owners

Goal: Here we have a view where all the car owners are listed who have registered more than 2 cars with OLA. This will help ola to identify all the premium owners who are affiliated with OLA.

**Stored procedure 1**: driver\_rating

Parameters: IN customerID varchar (255), IN driverID varchar(255), IN userrating int(11),IN review varchar(255)

Goal: Whenever customer gives feedback to a driver, we insert the feedback into the feedback table and also calculate the driver rating based on consolidated list of the particular driver’s rides.

**Stored procedure 2**: create\_subscription

Parameters: IN customerId varchar (255), IN subscriptionId varchar (250)

Goal: By taking inputs as customer\_id and subscription\_id, based on the period of subscription taken by the customer we are updating the subscription expiry date in the system for the subscriber and also reducing the availability of total number subscriptions.

**Stored procedure 3**: top10\_drivers\_of\_the\_month

Parameters: IN startTime datetime, IN endTime datetime

Goal: We are using this procedure to calculate the top 10 drivers with maximum number of ride hours driven.

**Event 1**: Recurring Event

**Event Name**: delete\_past\_records

Goal:  It will work every 24 hours and delete all the records from the ride table that are older than 1 year.

**Event 2**: Recurring Event

**Event Name**: birthday\_voucher

Goal:  It will find out the details of the customer who have taken more than 3 rides and its their birthday on that day.

**Trigger**:

Note: We did not add any triggers. Whatever functionality that trigger can do we handled that functionality by using stored procedure and thus eliminated extra overhead caused by execution of triggers.

**DEMO**

1. View Ride History:-

CREATE

ALGORITHM = UNDEFINED

DEFINER = `Riddhi`@`%`

SQL SECURITY DEFINER

VIEW `ride\_history` AS

SELECT

CONCAT\_WS(' ', `c`.`first\_name`, `c`.`last\_name`) AS `Customer Name`,

CONCAT\_WS(' ', `d`.`first\_name`, `d`.`last\_name`) AS `Driver Name`,

`r`.`car\_number` AS `car\_number`,

`r`.`pickup\_location` AS `pickup\_location`,

`r`.`drop\_location` AS `drop\_location`,

FLOOR((RAND(2) \* 30)) AS `Trip distance`,

IFNULL(((FLOOR((RAND(2) \* 30)) \* `rc`.`rate\_per\_mile`) - LEAST(((FLOOR((RAND(2) \* 30)) \* `rc`.`rate\_per\_mile`) \* (`p`.`discount` / 100)),

`p`.`max\_discount`)),

(FLOOR((RAND(2) \* 30)) \* `rc`.`rate\_per\_mile`)) AS `Total Fare`

FROM

(((((`Customer` `c`

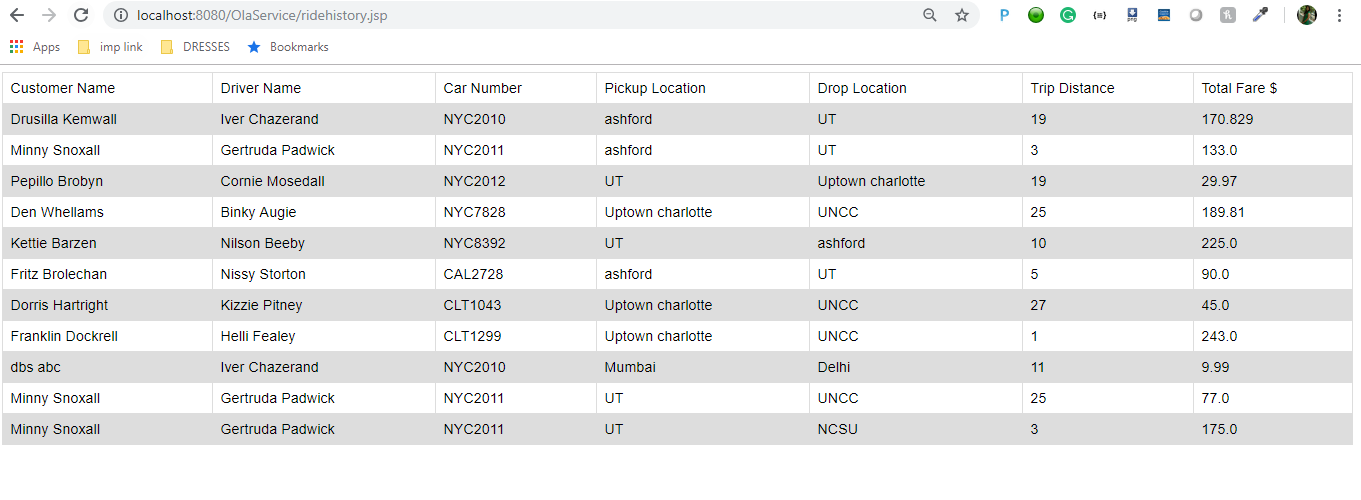
JOIN `Ride` `r` ON ((`r`.`customer\_id` = `c`.`customer\_id`)))

JOIN `Driver` `d` ON ((`d`.`driver\_id` = `r`.`driver\_id`)))

JOIN `Car` `ca` ON ((`r`.`car\_number` = `ca`.`car\_number`)))

JOIN `RateChart` `rc` ON ((`ca`.`car\_type` = `rc`.`car\_type`)))

LEFT JOIN `Promotion` `p` ON ((`p`.`voucher\_code` = `r`.`voucher\_code`)))



1. View Premium Owners:-

CREATE

ALGORITHM = UNDEFINED

DEFINER = `Riddhi`@`%`

SQL SECURITY DEFINER

VIEW `premium\_owners` AS

SELECT

`c`.`owner\_id` AS `owner\_id`,

CONCAT\_WS(' ', `c`.`first\_name`, `c`.`last\_name`) AS `Owner name`,

GROUP\_CONCAT(CONCAT\_WS(' : ',

`ca`.`car\_number`,

`ca`.`car\_model`)

SEPARATOR ' , ') AS `Car Details`

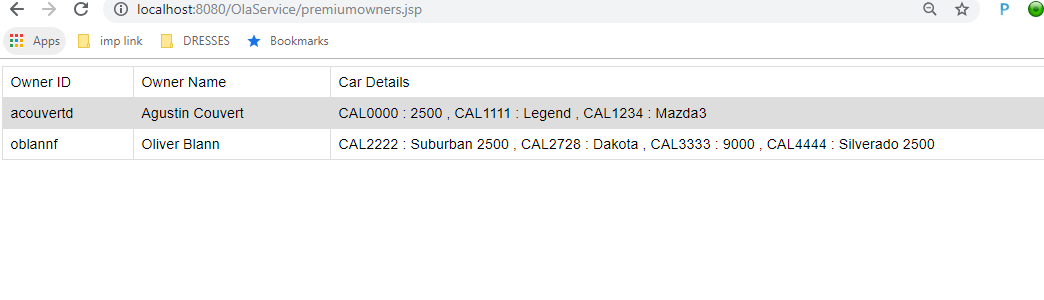
FROM

(`CarOwner` `c`

JOIN `Car` `ca` ON ((`ca`.`owner\_id` = `c`.`owner\_id`)))

GROUP BY `c`.`owner\_id`

HAVING (COUNT(`ca`.`owner\_id`) >= 2)



1. Add Feedback:-

CREATE DEFINER=`Riddhi`@`%` PROCEDURE `driver\_rating`(IN customerID varchar(255),IN driverID varchar(255), IN userrating int(11),IN review varchar(255))

BEGIN

DECLARE given\_rating int(11);

Insert into Feedback(customer\_id,driver\_id,number\_of\_stars,customer\_review) values(customerID,driverID,userrating,review);

SELECT truncate(AVG(number\_of\_stars),2) INTO given\_rating

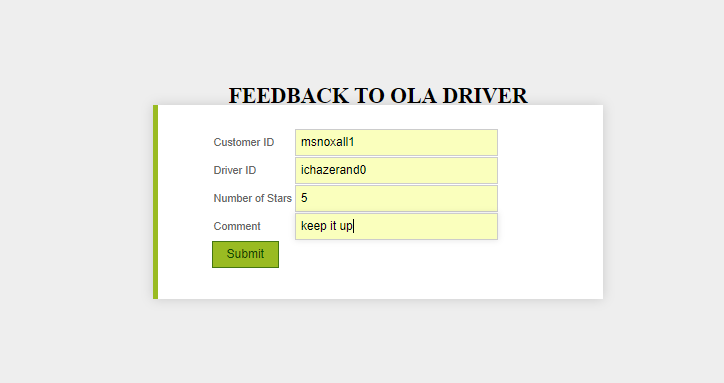
FROM Feedback f

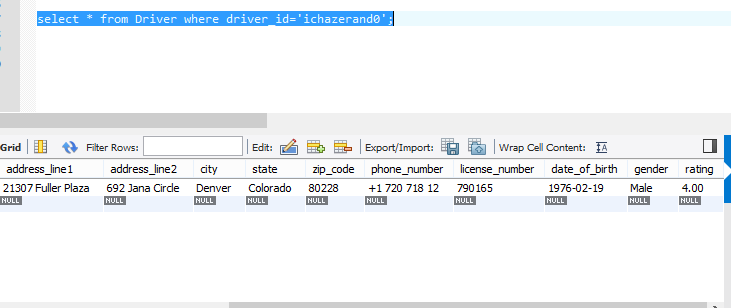
Where driver\_id = driverID;

UPDATE Driver SET rating = given\_rating

where driver\_id = driverID;

END





1. Create Subscription:-

CREATE DEFINER=`Riddhi`@`%` PROCEDURE `create\_subscription`(IN customerId varchar(255), IN subscriptionId varchar(250))

BEGIN

DECLARE avail INT default 0;

DECLARE max\_period varchar(255);

DECLARE expiryDate date;

SELECT availability,period into avail, max\_period from Subscription where subscription\_id = subscriptionId;

IF avail > 0 THEN

IF max\_period = 'monthly' THEN

set expiryDate = DATE\_ADD(NOW(), Interval 30 day);

ELSEIF max\_period ='quarterly' THEN

set expiryDate = DATE\_ADD(NOW(), Interval 90 day);

ELSEIF max\_period ='annual' THEN

set expiryDate = DATE\_ADD(NOW(), Interval 365 day);

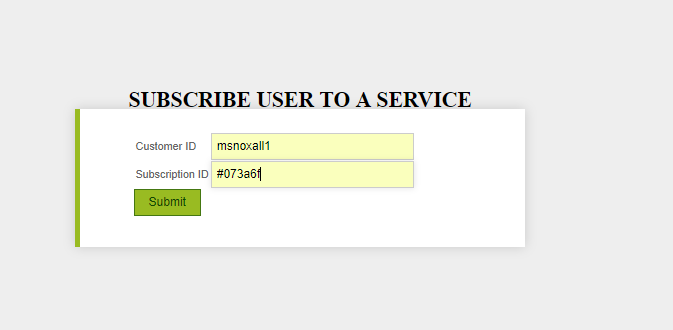
END IF;

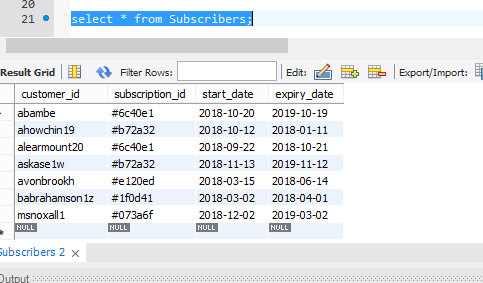
insert into Subscribers values(customerId,subscriptionId,NOW(),expiryDate);

update Subscription set availability = (availability-1) WHERE subscription\_id = subscriptionId;

END IF;

END





1. Top 10 drivers of the month:-

CREATE DEFINER=`Riddhi`@`%` PROCEDURE `top10\_drivers\_of\_the\_month`(IN startTime datetime, IN endTime datetime)

Begin

select d.driver\_id, concat\_ws(' ',d.first\_name, d.last\_name), d.email\_id, d.phone\_number, count(r.ride\_id) as `Total Rides` from Driver d

inner join Ride r on

r.driver\_id = d.driver\_id

where r.start\_time < startTime and

r.end\_time > endTime

group by r.driver\_id

order by `Total Rides` desc

limit 10;

End

