Car Rental Management System

Project Report

Jaykishan Satikuvar (18BCP046)
Neev Shah (18BCP067)
Kalpit Lakhadhariya (18BCP047)
Kevin Patel (18BCP053)
Dhruv Soni (18BCP154D)

INDEX

Experiment – 1

Advantages of DBMS over File system

- ## Data Redundancy and Inconsistency.

Since different programmers create the files and application programs over a long period, the various files are likely to have different structures and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files). For example, if a car owner has a more than one car (say SUV and sedan) the location and mobile number of that car owner may appear in a file that consists of car owner's records of car owners having SUV cars and in a file that consists of car owners records of car owners having Sedan car. This redundancy leads to higher storage and access cost. In addition, it may lead to data inconsistency that is, the various copies of the same data may no longer agree. For example, a changed Car owner's mobile number may be reflected in the car owners having SUV car department records but not elsewhere in the system.

- ## Difficulty in accessing data

Suppose that car owner wants to find out his owned cars in any particular location. The car owner asks the data-processing department to generate such a list. Because the designers of the original system did not anticipate there is no application program on hand to meet it. DBMS is application program to generate the list of all location. Owner has now two choices: either obtain the list of all cars with its location needed information manually or ask a programmer to application program. Both alternatives are obviously unsatisfactory. After some day if a car owner wanted the

details of a driver driven his car a month ago, Then there is retirement of another program.

The point here is that conventional file-processing do not allow needed data to be retrieved in a convenient and efficient manner.

- Data Isolation

Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

- Integrity Problems

The data values stored in database must satisfy certain types of consistency constrains. The Car rental system maintains balance of customers. We need that the balance of any customers should not be less than zero. We also need that customers can't book it for less than 18 hours. However, when new constraints are needed, it is difficult to change the programs to enforce them. The problem is compounded when constrains involve several data items from different files.

- Atomicity Problem

A computer system, like any other device, is subject to failure In many applications, It is crucial that, if a failures occurs, the data be restored to the consistent state that existed prior to the failure. Suppose Customer has paid the payment but it has not reached at the driver's account, resulting in an inconsistence database state. The funds transfer must be atomic-it must happen in its entirety or not at all. It is difficult to ensure atomicity in conventional file-processing systems.

- Concurrent-access anomalies

For the sake of improved overall performance of the system and faster response, our system will allow multiple users to book a car simultaneously which is possible easily by using Database Management System. If a customer A and customer B tries to book a car for rent at a same time both the customers will get a car that they have booked whereas this type situations will create difficulties. That is the drawback of File system.

- Security Problems

In the file system every user may be able to access all the data. Any customer should not be given access to all the data like transaction detail of company and driver . If a customer given access to all the data it can be a threat for the developer’s business or any other person involved.

Experiment – 2

LIST OF TABLES

- Customer(customer_id, customer_name, license_number,customer_mail)
- Review_of_vehicle(trip_id,vehical_id,points_out_of_5,customer_id)
- Review_of_driver(trip_id,driver_id,points_out_of_5,customer_id)
- Offer(offer_code,offer_amount)
- Pays_bill(customer_id,bill_id)
- If_driver(driver_id,customer_id)
- Makes_trip(customer_id,trip_id)
- Trip(trip_id,start_date,no_of_days)
- Takes_vehicle(customer_id, vehical_id)
- Bil_details(bill_id,base_amount,fine,discount,deposite_paid,driver_charge, final_amount)
- vehicle(vehicle_id,vehicle_name,vehicle_type'expected_avg,price_per_day)
- Owned_by(vehicle_id,owner_id)
- Owner(owner_id,owner_name,owner_address,owner_mob_no)
- Pickup_centres(centre_id,center_address,contact_no)
- Goes_to(centre_id,vehicle_id)
- Final_bill(bill_id,offer_code,final_amount)
- Driver(driver_id,driver_name,gender,licence_no,mob_no,driver_address)

RELATIONAL MODEL

customer:
- candidate key: customer_id,licence_number
- primarykey: customer_id

vehicle:
- c.key: vehicle_id
- p.key: vehicle_id

driver:
- c.key: driver_id,licence_no,
- p.key: driver_id

makes_trip:
- superkey : customer_id , trip_id
- foreignKey : customer_id , trip_id
- candidatekey : trip_id and driver_id
- p.key : trip_id and driver_id

trip:
- s.key : trip_id
- c.key : trip_id
- p.key : trip_id

review_of_driver :
- f.key : trip_id,driver_id
- s.key : trip_id,driver_id
- c.key : trip_id , trip_id and driver_id
- p.key : trip_id

bill_details :
- s.key : bill_id
- c.key : bill_id
- p.key : bill_id

offer
- s.key: offer_code
- c.key : offer_code
- p.key : offer_code

final_bill:
- f.key : bill_id,offer_code
- s.key : bill_id,offer_code
- c.key : bill_id
- p.key : bill_id

pays_bill :
- f.key : customer_id,bill_id
- s_key : customer_id , bill_id
- c_key : customer_id and bill_id
- p.key : customer_id and bill_id

if_driver:
- f.key : driver_id , customer_id
- s.key : driver_id , customer_id
- c.key : driver_id and customer_id
- p.key : driver_id and customer_id

owner :
- s.key : owner_id
- c.key : owner_id
- p.key : owner_id

takes_vehicle :
- f.key : customer_id , vehicle_id
- s.key : customer_id , vehicle_id
- c.key : customer_id and vehicle_id
- p.key : customer_id and vehicle_id

owned_by:

f.key : vehicle_id , owner_id
s.key : vehicle_id , owner_id
c.key : vehicle_id and owner_id
p.key : vehicle_id and owner_id

pickup_centres:

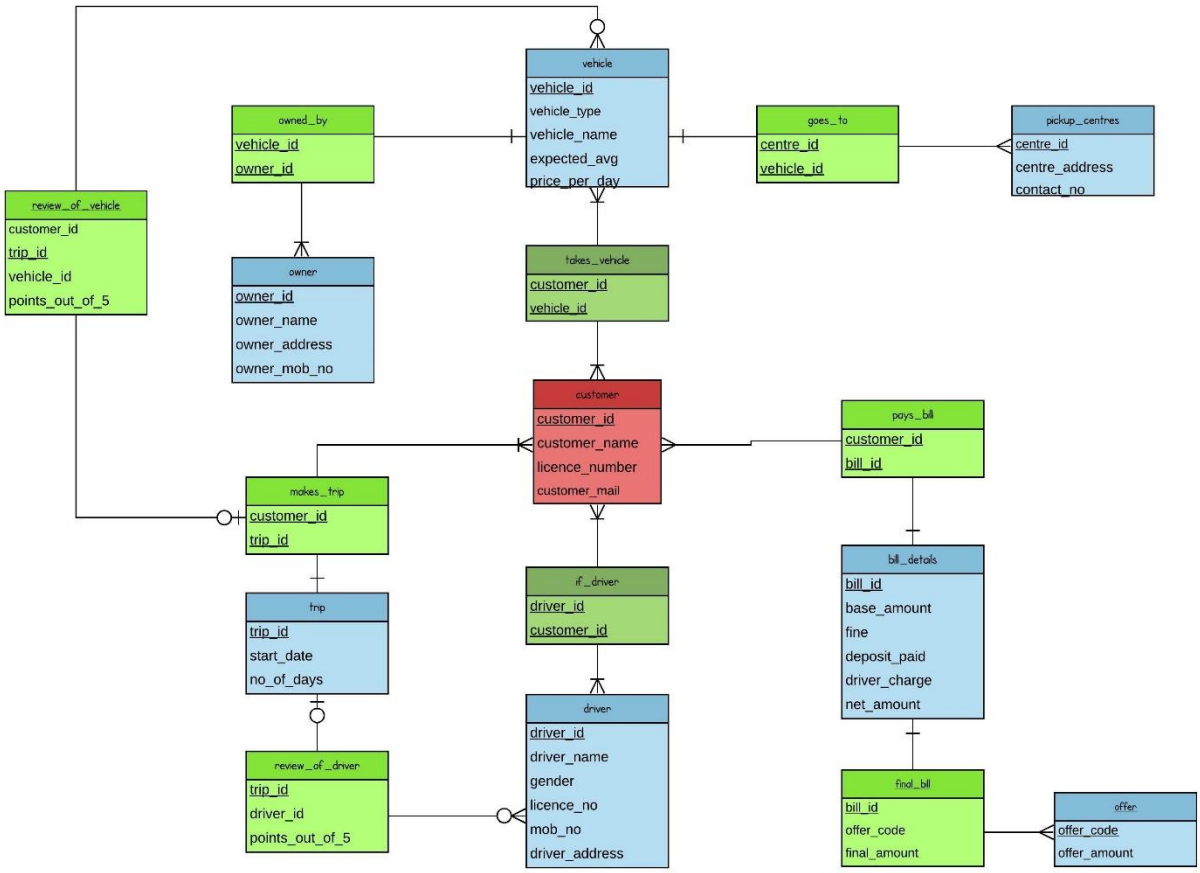
s.key : centre_id
c.key : centre_id
p.key : centre_id

review_of_vehicle :

f.key : trip_id,vehicle_id,customer_id
s.key : trip_id, trip_id and vehicle_id, trip_id and customer_id
c.key : trip_id , trip_id and vehicle_id, trip_id and customer_id
p.key : trip_id

goes_to :

f.key : vehicle_id , centre_id
s.key : vehicle_id , centre_id
c.key : vehicle_id and centre_id
p.key : vehicle_id and centre_id


vehicle
vehicle_id
vehicle_type
vehicle_name
expected_avg
price_per_day
owned_by
vehicle_id
owner_id
goes_to
centre_id
vehicle_id
pickup_centres
centre_id
centre_address
contact_no
review_of_vehicle
customer_id
trip_id
vehicle_id
points_out_of_5
owner
owner_id
owner_name
owner_address
owner_mob_no
takes_vehicle
customer_id
vehicle_id
customer
customer_id
customer_name
licence_number
customer_mail
pays_bill
customer_id
bill_id
makes_trip
customer_id
trip_id
trip
trip_id
start_date
no_of_days
if_driver
driver_id
customer_id
bill_details
bill_id
base_amount
fine
deposit_paid
driver_charge
net_amount
driver
driver_id
driver_name
gender
licence_no
mob_no
driver_address
review_of_driver
trip_id
driver_id
points_out_of_5
final_bill
bill_id
offer_code
final_amount
offer
offer_code
offer_amount

Experiment – 3

Relational Algebra

1.Selection:

The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. Sigma(σ)Symbol denotes it. It is used as an expression to choose tuples which meet the selection condition. Select operation selects tuples that satisfy a given predicate.

2.Projection:

The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation.

This helps to extract the values of specified attributes to eliminates duplicate values. (pi) The symbol used to choose attributes from a relation. This operation helps you to keep specific columns from a relation and discards the other columns.

3.Cartesian Product:

This type of operation is helpful to merge columns from two relations. Generally, a Cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations.

4.Union:

UNION is symbolized by ∪ symbol. It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples. So, set A UNION set B would be expressed as:

The result <- A ∪ B

For a union operation to be valid, the following conditions must hold -

- R and S must be the same number of attributes.
- Attribute domains need to be compatible.
- Duplicate tuples should be automatically removed.

5.Set difference:

- Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.

- The attribute name of A has to match with the attribute name in B.
- The two-operand relations A and B should be either compatible or Union compatible.
- It should be defined relation consisting of the tuples that are in relation A, but not in B.

6.Natural Join:

Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.

Example:

1. Selection:

```
σ customer_name = "Raju" (customer)
σ driver_name = "Madan" (driver)
σ owner_name = "Raman" (owner)
σ customer_name = "Raj" (customer)
σ owner_name = "Ram" (owner)
```

2. Projection:

```
Π Customer_id, customer_name (Customer)
Π driver_id, gender (driver)
Π owner_id, owner_name (owner)
Π Customer_id (Customer)
Π driver_id, driver_name (driver)
```

3. Cartesian product:

```
σ Customer_id = 'c011' (customer X if_driver)
σ Customer_id = 'c011' (driver X if_driver)
σ vehicle_id = 'v007' (owner X owned_by)
σ Customer_id = 'c001' (customer X if_driver)
σ vehicle_id = 'v008' (owner X owned_by)
```

4. Union:

```
Customer U if_driver
Driver U if_driver
Owner U owned_by
Customer U pays_bill
Bill_details U pays_bill
```

5. Set difference:

```
customer-if_driver
Driver-if_driver
Owner-owned_by
Customer - pays_bill
Bill_details - pays_bill
```

6. Natural join:

```
customer ⋈ if_driver
driver ⋈ if_driver
owner ⋈ owned_by
Bill_details ⋈ pays_bill
Customer ⋈ pays_bill
```

7. Composition of any two from (1-6) operators:

```
σ driver_name = "Raju" (customer ⋈ if_driver)
σ customer_name = "Madan" (customer ⋈ if_driver)
σ owner_name = "Rajesh" (owner ⋈ owned_by)
σ driver_name = "Rajm" (customer ⋈ if_driver ⋈ driver)
σ owner_id = "o011" (owner ⋈ owned_by)
```

8. Composition of any three of above (1-6) operators:

```
Π Customer_id, customer_name (σ driver_id = "d004" (customer ⋈ if_driver))
Π driver_id, driver_name (σ customer_id = "c044" (driver ⋈ if_driver))
Π vehicle_id, owner_name (σ owner_id = "o0088" (owner ⋈ owned_by))
Π Customer_id, customer_name (σ driver_name = "nishant" (customer ⋈ if_driver
⋈ driver))
Π vehicle_id, owner_name (σ owner_id = "o008" (owner ⋈ owned_by))
```

Experiment – 4

ENTITY RELATIONAL MODEL

customer:

candidatekey : customer_id,licence_number
primarykey : customer_id

vehicle:

c.key : vehicle_id
p.key : vehicle_id

driver:

c.key : driver_id,licence_no,
p.key : driver_id

makes_trip :

superkey : customer_id , trip_id
foreignKey : customer_id , trip_id
candidatekey : trip_id and driver_id
p.key : trip_id and driver_id

trip:

s.key : trip_id
c.key : trip_id
p.key : trip_id

review_of_driver :

f.key : trip_id,driver_id
s.key : trip_id,driver_id
c.key : trip_id , trip_id and driver_id
p.key : trip_id

bill_details :

s.key : bill_id
c.key : bill_id
p.key : bill_id

offer
s.key : offer_code
c.key : offer_code
p.key : offer_code

final_bill:
f.key : bill_id,offer_code
s.key : bill_id,offer_code
c.key : bill_id
p.key : bill_id

pays_bill :
f.key : customer_id,bill_id
s_key : customer_id , bill_id
c_key : customer_id and bill_id
p.key : customer_id and bill_id

if_driver:
f.key : driver_id , customer_id
s.key : driver_id , customer_id
c.key : driver_id and customer_id
p.key : driver_id and customer_id

owner :
s.key : owner_id
c.key : owner_id
p.key : owner_id

takes_vehicle :
f.key : customer_id , vehicle_id
s.key : customer_id , vehicle_id

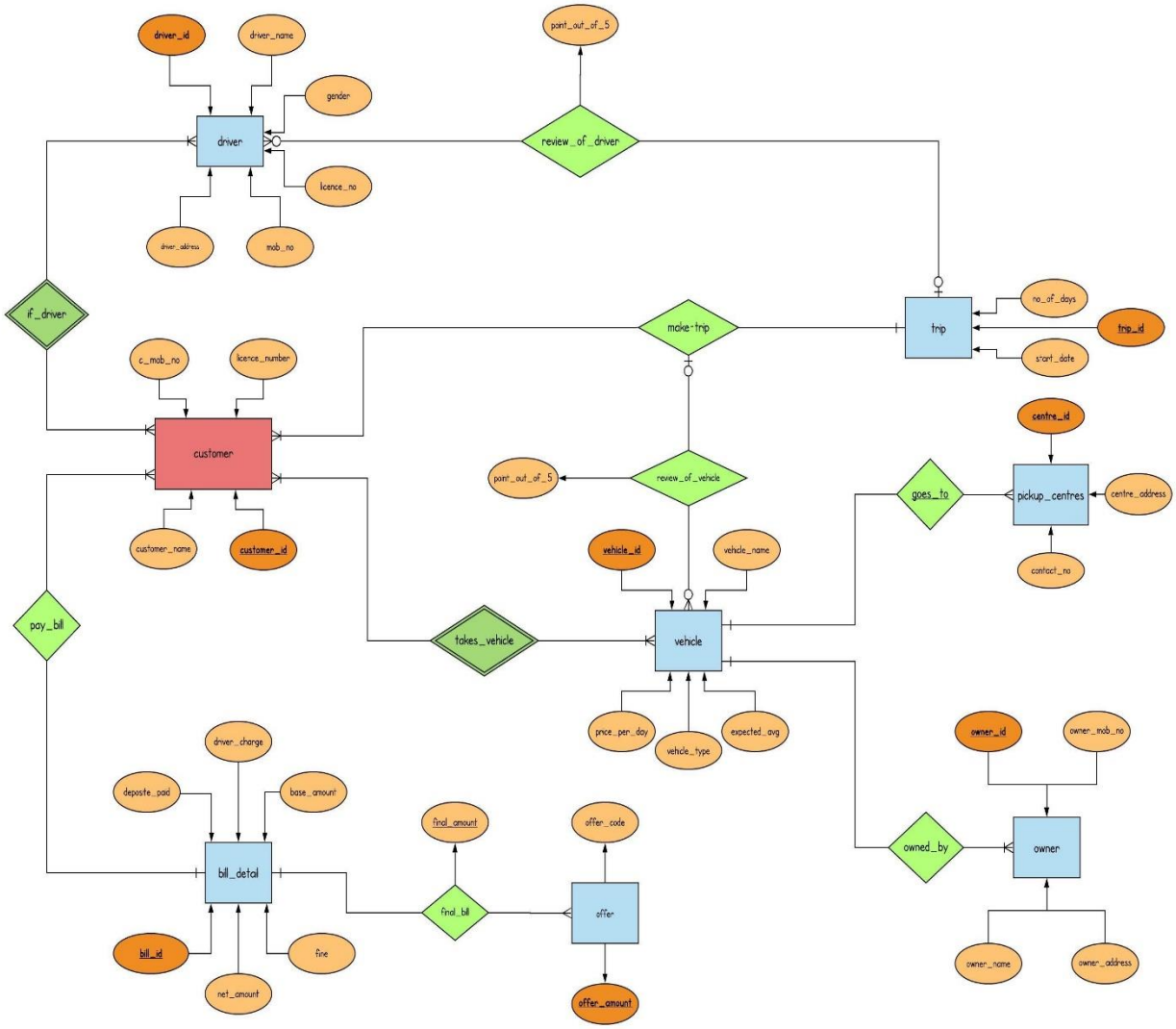
c.key : customer_id and vehicle_id
p.key : customer_id and vehicle_id

owned_by:
f.key : vehicle_id , owner_id
s.key : vehicle_id , owner_id
c.key : vehicle_id and owner_id
p.key : vehicle_id and owner_id

pickup_centres:
s.key : centre_id
c.key : centre_id
p.key : centre_id

review_of_vehicle :
f.key : trip_id,vehicle_id,customer_id
s.key : trip_id, trip_id and vehicle_id, trip_id and customer_id
c.key : trip_id , trip_id and vehicle_id, trip_id and customer_id
p.key : trip_id

goes_to :
f.key : vehicle_id , centre_id
s.key : vehicle_id , centre_id
c.key : vehicle_id and centre_id
p.key : vehicle_id and centre_id


driver_id
driver_name
gender
driver
licence_no
driver_address
mob_no
point_out_of_5
review_of_driver
if_driver
make_trip
trip
no_of_days
trip_id
start_date
c_mob_no
licence_number
customer
customer_name
customer_id
point_out_of_5
review_of_vehicle
goes_to
pickup_centres
centre_id
centre_address
contact_no
vehicle_id
vehicle_name
pay_bill
takes_vehicle
vehicle
price_per_day
vehicle_type
expected_avg
owner_id
owner_mob_no
driver_charge
deposite_paid
base_amount
bill_detail
final_amount
offer_code
owned_by
owner
final_bill
offer
bill_id
fine
net_amount
offer_amount
owner_name
owner_address

Experiment – 5 to 10

Exp – 6 Create tables

```
create table customer(
    customer_id     varchar(6),
    customer_name   varchar(20),
    licence_number  varchar(27),
    primary key(customer_id)
);

create table vehicle(
    vehicle_id      varchar(6),
    vehicle_type    varchar(8),
    vehicle_nam     varchar(8),
    expected_avg    numeric(2,2),
    price_per_day   numeric(4,0),
    primary key(vehicle_id)
);

create table takes_vehicle(
    vehicle_id      varchar(6),
    customer_id     varchar(6),
    foreign key(vehicle_id) references vehicle,
    foreign key(customer_id) references customer
```

```
);
create table owner(
        owner_id            varchar(6),
        owner_name          varchar(20),
        owner_address       varchar(100),
        owner_mob_no        numeric(10,0),
        primary key(owner_id)
);

create table driver(
    driver_id       varchar(6),
    driver_name     varchar(20),
    gender          varchar(6),
    licenece_no     varchar(28),
    mob_no          numeric(10,0),
    driver_address  varchar(100),
    primary key(driver_id)
    );

create table if_driver(
    driver_id       varchar(6),
    customer_id     varchar(6),
    foreign key(customer_id) references customer,
    foreign key(driver_id) references driver
    );
```

```
create table trip(
    trip_id       varchar(6),
    start_date    varchar(10),
    no_of_days    numeric(2,0),
    primary key(trip_id)
    );

create table makes_trip(
    trip_id       varchar(6),
    customer_id   varchar(6),
    foreign key(trip_id) references trip,
    foreign key(customer_id) references customer
    );

create table bill_details(
    bill_id       varchar(6),
    base_amount   numeric(5,1),
    fine          numeric(6,0),
    deposite_paid numeric(4,0),
    driver_charge numeric(4,1),
    net_amount    numeric(7,1),
    primary key(bill_id)
    );
```

```
create table pays_bill(
    bill_id        varchar(6),
    customer_id    varchar(6),
    foreign key(customer_id) references customer,
    foreign key(bill_id) references bill_details
    );

create table offer(
    offer_code    varchar(6),
    offer_amount    numeric(3,0),
    primary key(offer_id)
    );

create table final_bill(
    bill_id        varchar(6),
    offer_code     varchar(6),
    final_amount   numeric(7,1),
    foreign key(bill_id) references bill_details,
    foreign key(offer_code) references offer
    );

create table owned_by(
        vehicle_id varchar(6),
        owner_id varchar(6),
        foreign key(vehicle_id) references vehicle,
        foreign key(owner_id) references owner
```

```
);

create table goes_to(
        centre_id varchar(6),
        vehicle_id varchar(6),
        foreign key (centre_id) references pickup_centres,
        foreign key (vehicle_id) references vehicle
         );
create table review_of_driver(
        customer_id varchar(6),
        trip_id varchar(6),
        driver_id varchar(6),
        points_out_of_5 int(1),
        foreign key (customer_id) references customer,
        foreign key (trip_id) references trip,
        foreign key (driver_id) references driver
 );
create table review_of_vehicle(
         customer_id varchar(6),
         trip_id varchar(6),
         vehicle_id varchar(6),
         points_out_of_5 int(1),
         foreign key (customer_id) references customer,
         foreign key (trip_id) references trip,
         foreign key (vehicle_id) references driver
 );
```

Inserted Data (Sample Code)

Customer:-

```
insert into customer values('c01','yash','GJ56842','yash@mail.com');
insert into customer values('c02','raj','GJ45142','raj@mail.com');
insert into customer values('c03','parth','GJ12456','parth@mail.com');
insert into customer values('c04','pratik','GJ98562','pratik@mail.com');
insert into customer values('c05','vivek','GJ01245','vivek@mail.com');
```

vehicle:-

```
insert into  vehicle values('v01','two','passion pro',60,300);
insert into  vehicle values('v02','four','swift',18,800);
insert into  vehicle values('v03','four','verna',14,1200);
insert into  vehicle values('v04','four','innova',10,1800);
insert into  vehicle values('v05','two','hornet',40,500);
```

takes_vehicle:-

```
insert into takes_vehicle values('c01','v03');
insert into takes_vehicle values('c02','v05');
insert into takes_vehicle values('c03','v01');
insert into takes_vehicle values('c04','v04');
```

owner:-

```
insert into owner values('ow1','mohan','ahmedabad',9856231245);
insert into owner values('ow2','kiran','vadodara',9791234523);
insert into owner values('ow3','ramesh','surat',7856478523);
insert into owner values('ow4','mahesh','rajkot',8596475236);
insert into owner values('ow5','suresh','ahmedabad',8541236589);
```

driver:-

```
insert into driver values('d01','dhaval','male','GJ45612',7895647123,'surat');
insert into driver values('d02','meet','male','GJ85479',8945236153,'nadiad');
insert into driver values('d03','ruchit','male','GJ12547',9325687452,'ahmedabad');
insert into driver values('d01','riya','female','GJ4452 2',8859961230,'rajkot');
insert into driver values('d01','tiya','female','GJ25694',7884576236,'navsari');
```

if_driver:-

```
insert into if_driver values('d01','c02');
insert into if_driver values('d02','c01');
insert into if_driver values('d03','c05');
```

trip:-

```
insert into trip values('t01','2-jan-2019',2);
insert into trip values('t02','25-oct-2019',5);
insert into trip values('t03','17-aug-2019',1);
```

makes_trip:-

```
insert into makes_trip values('t01','c02');
insert into makes_trip values('t02','c01');
insert into makes_trip values('t03','c03');
```

bill_details:-

```
insert into bill_details values('b01',2000,0,4000,800,2800);
insert into bill_details values('b02',1000,200,2000,0,1200);
insert into bill_details values('b03',5000,0,4000,1500,6500);
```

pays_bill:-

insert into pays_bill values('b01','c02');

insert into pays_bill values('b02','c01');

insert into pays_bill values('b03','c04');

offer:-

insert into offer values('offer1',500);

insert into offer values('offer2',1000);

insert into offer values('offer3',1200);

final_bill:-

insert into final_bill values('b01','offer1',2300);

insert into final_bill values('b02','offer2',200);

insert into final_bill values('b03','offer3',5300);

owned_by:-

insert into owned_by values('v04','ow1');

insert into owned_by values('v05','ow2');

goes_to:-

insert into goes_to values('c01','v01');

insert into goes_to values('c02','v02');

insert into goes_to values('c03','v03');

insert into goes_to values('c04','v05');

review_of_driver:-

insert into review_of_driver values('c01','t01','d01',4);

insert into review_of_driver values('c02','t02','d02',3);

insert into review_of_driver values('c03','t03','d03',4);

review_of_vehicle:-

insert into review_of_vehicle values('c01','t01','v01',5);

insert into review_of_vehicle values('c02','t02','v02',4);

insert into review_of_vehicle values('c03','t03','v03',3);

Exp – 7 to 10

- Select max(base_amount)
  From bill_details

- Select customer_name
  From customer union if_driver
  Where driver_id='d003'

  Select customer_name
  From customer
  Where customer_id in(select customer_id
  From if_driver
  Where driver_id = 'd001');

  Select customer_name
  From customer
  Order by customer_name;

  select *owner_name*, *ID*, **avg** (*age*)
  from *owner*
  group by *owner_name*;

  select *bill_id*, **avg** (*base_amount*)
  from *bill_details*
  group by *final_amount*
  having avg (*base_amount*) > 6900

Functional Dependencies & Normalization:

Sr. No	Table Name	Functional Dependencies	Candidate key	1 NF	2 NF	3 NF	BCNF
1	Customer (customer_id, customer_name, c_mob_no, customer_mail, license_number)	customer_id -> R licence_no -> R c_mob_no -> R	customer_id licence_no c_mob_no	YES	YES	YES	YES
2	lf_driver (driver_id, customer_id)	R->R	driver_id, customer_id	YES	YES	YES	YES
3	Driver (driver_id, driver_name, gender, licence_no, mob_no, driver_addres)	driver_id -> R licence_no -> R mob_no -> R	driver_id licence_no mob_no	YES	YES	YES	YES
4	bill_details (bill_id, base_amount, fine,discount, deposite_paid, driver_charge, final_amount)	bill_id -> R	bill_id	YES	YES	YES	YES
5	Owner (owner_id, owner_mob_no, owner_address, owner_name)	owner_id -> R owner_mob_no -> R	owner_id owner_mob_no	YES	YES	YES	YES
6	owned_by (vehicle_id, owner_id)	R -> R	vehicle_id, owner_id	YES	YES	YES	YES

7	pickup_centre (centre_id, center_address, contact_no)	centre_id -> R contact_no -> R	centre_id contact_no	YES	YES	YES	YES
8	goes_to (centre_id, vehicle_id)	R -> R	centre_id, vehicle_id	YES	YES	YES	YES
9	pays_bil (customer_id, bill_id)	R -> R	customer_id, bill_id	YES	YES	YES	YES
10	review_of_driver (trip_id, driver_id, points_out_of_5, customer_id)	trip_id -> R	trip_id	YES	YES	YES	YES
11	takes_vehicle (customer_id, vehical_id)	R -> R	Customer_id, vehical_id	YES	YES	YES	YES
12	makes_trip (customer_id, trip_id)	R -> R	Customer_id, Trip_id	YES	YES	YES	YES
13	Trip (trip_id, start_date, no_of_days)	trip_id -> R	trip_id	YES	YES	YES	YES
14	review_of_vehicle (trip_id, vehical_id, points_out_of_5, customer_id)	trip_id -> R	trip_id	YES	YES	YES	YES
15	final_bill (bill_id, offer_code, final_amount)	bill_id -> R	bill_id	YES	YES	YES	YES

16	Offer (offer_code, offer_amount)	offer_code -> R	offer_code	YES	YES	YES	YES
17	Vehicle (vehicle_id, vehicle_name, vehicle_type, expected_avg, price_per_day)	vehicle_id -> R vehicle_name -> vehicle_type	vehicle_id vehicle_name	YES	YES	YES	YES

Uniqueness of project as to existing work:

- It's somewhat similar to that of Zoomcar as per as the service is concerned but as of now we are working on the database model of it which is the main aspect of it.

- If you will go through the reviews given by the users of Zoomcar you will get to know that they lack coordination which in turn means that the database is not proper and its not used up to its optimum level.

- You can read the reviews at https://www.mouthshut.com/product-reviews/Zoom-Cars-reviews-925725994

- Also, our customers will have option of opting for driver instead of self-driving.

- Our major concern is cars but we also provide two-wheeler services.

Stored Function

Definition: -

The CREATE FUNCTION statement is used for creating a stored function and user-defined functions. A stored function is a set of SQL statements that perform some operation and return a single value.

Advantage: -

- They allow modular programming.
- They allow faster execution.

Disadvantage: -

- function have limited error handling
- functions cannot use temporary tables

Examples: -

1.

```
Create function net_amount(base_amount numeric(5,1) ,fine numeric(6,0)
,driver_charge numeric(4,1) ,driver_charge numeric(4,1))

Returns numeric(7,1) as

Deterministic

Begin

    Return base_amount + fine + driver_charge – deposit_paid;

End;
```

2.

```
Create function amount(base_amount numeric(5,1) ,fine numeric(6,0) ,driver_charge
numeric(4,1))

Returns numeric(7,1) as

Deterministic

Begin

        Return base_amount + fine + driver_charge;

End;
```

3.

```
Create function FinalAmount(net_amount numeric(7,1) , offer_amount
numeric(3,0))

Returns numeric(7,1) as

deterministic

Begin

        Return net_amount – offer_amount;

End;
```

4.

```
DELIMITER $$
CREATE FUNCTION car_Level(
    base_amount DECIMAL(10,2)
)
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
    DECLARE car_level VARCHAR(20);

    IF credit base_amount > 5000  THEN
        SET car_level = 'EXCELLENT';
    ELSEIF ( base_amount<=5000 AND
            base_amount >= 1000) THEN
        SET car_level = 'GOOD';
    ELSEIF base_amount < 1000 THEN
        SET car_level = 'POOR';
    END IF;
    -- return the car_level
    RETURN (car_level);
END$$
DELIMITER ;
```

5.

```
CREATE FUNCTION find_driver_class
( driver_id VARCHAR(20))
RETURNS
@driverclass table (
driver_id VARCHAR(20),
avg(points_out_of_5 as rating )  ??????,
class_of_driver VARCHAR(20)
)
AS
BEGIN
INSERT INTO @driverclass
SELECT  driver_id,avg(points_out_of_5) as rating,class_of_driver
FROM review_of_driver
 GROUP BY driver_id
IF @@rating >5 THAN
BEGIN
INSERT INTO @AuthorsByState
VALUES (' ','Best')
END
IF @@rating >4 THAN
BEGIN
INSERT INTO @AuthorsByState
```

```
VALUES (' ','Good')
END
IF @@rating >3 THAN
BEGIN
INSERT INTO @AuthorsByState
VALUES (' ','Medium')
END
IF @@rating >2 THAN
BEGIN
INSERT INTO @AuthorsByState
VALUES (' ','Poor')
END
IF @@rating >1 THAN
BEGIN
INSERT INTO @AuthorsByState
VALUES (' ','Bed')
END
RETURN
END
GO
```

Conclusion, future work and references

Each time we have learnt something new, we have gone through many corrections after which we arrived at the above database model. Though, we do not claim that it is the optimum model as per as the practical use is concerned but we do conclude that after certain modifications it can match the standards of the present similar working database model which does require more exposure to this subject and a good experience of the practically working database models which we can analyze.