UBER
Project Report
Database Design

Team:

Sandhya Rayala (sxr210059)
Sreekar Kumar Jasti (sxj210039)
Abhilash Vadla (axv200018)

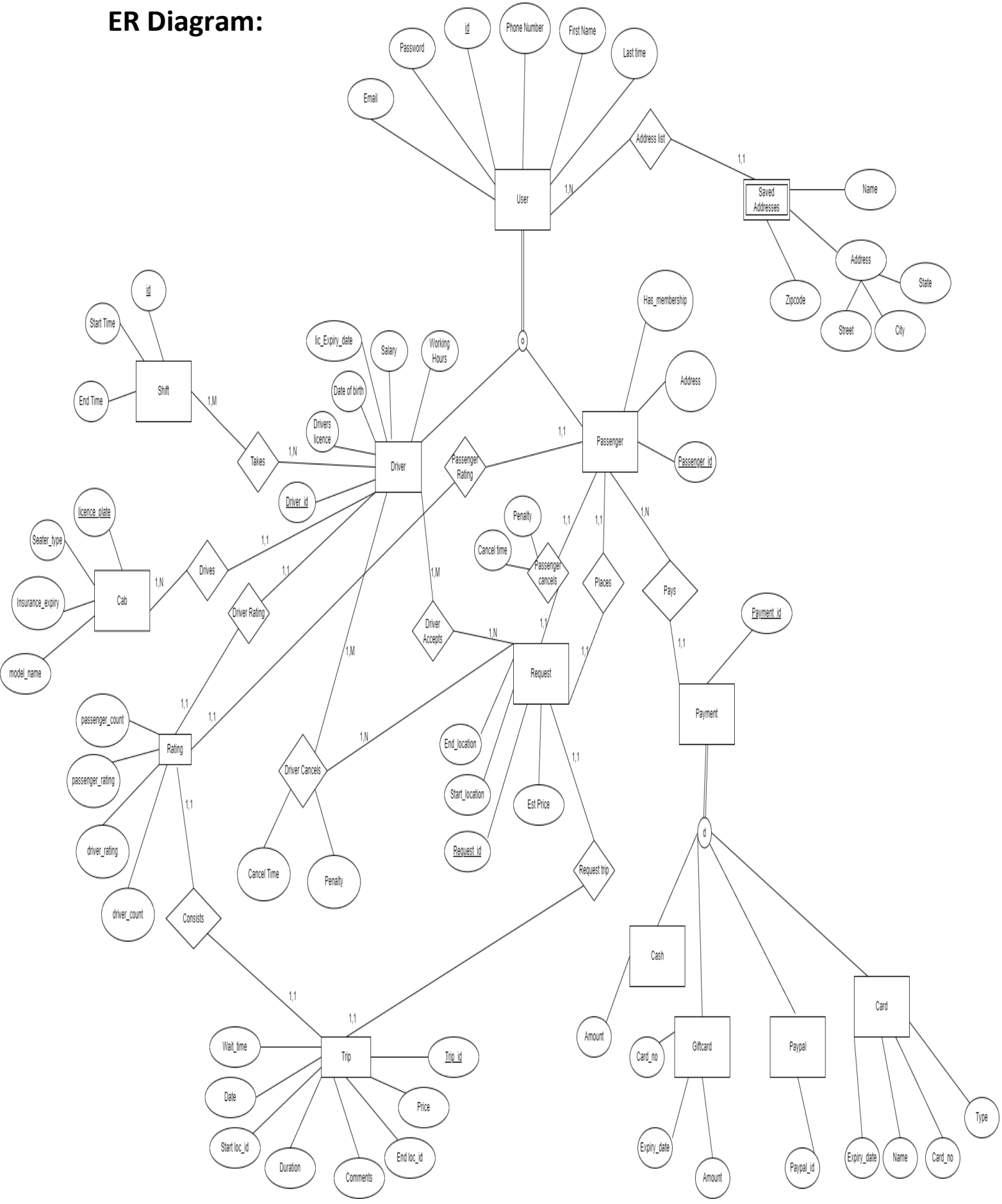
The project deals with the creation of uber database fulfilling basic functional requirements. Following are the few required components -

- The user (passenger and driver) needs to **register** with the franchise.
- The users can have multiple **marked spots** which they travel frequently (home, work, cafeterias).
- Passenger can **request to book a ride** altogether once at an instance.
- Driver's choice of accepting or denying the ride.
- Driver's choice of selecting **multiple cars** for multiple shifts at different locations.
- Passenger's choice of **payment methods**.
- **Rating** one to one regarding the ride.

Relationships:

- 1:1
  - A driver can rate the passenger and vice-versa once for each ride.
  - A passenger can request for a ride or can cancel the same altogether one at time.
- 1:N
  - Either the passenger or the driver can save multiple locations as per their connivence.
  - A driver can drive multiple cabs.
  - A passenger can have multiple payment methods or saved cards.
- M:N
  - A driver can have multiple shifts and in the context of same in a shift there can be n number of drivers.
  - A driver could receive multiple requests at any instance of time and a common request can be sent to multiple drivers.
  - A driver can cancel multiple requests and a request can be canceled by multiple drivers.

ER Diagram:


id
Password
Phone Number
First Name
Last time
Email
User
Address list
1,1
1,N
Saved Addresses
Name
Address
State
Zipcode
Street
City
o
Has_membership
Address
Passenger id
Passenger
1,1
Passenger Rating
Driver
lic_Expiry_date
Salary
Working Hours
Date of birth
Drivers licence
Driver_id
Takes
1,N
Shift
id
Start Time
End Time
1,M
licence_plate
Seater_type
Insurance_expiry
model_name
Cab
1,N
Drives
1,1
Driver Rating
1,1
Penalty
Cancel time
Passenger cancels
1,1
1,1
Places
1,N
Pays
Payment_id
1,1
Payment
Driver Accepts
1,M
1,N
Request
1,1
1,1
1,M
passenger_count
passenger_rating
driver_rating
driver_count
Rating
1,1
1,1
1,1
Driver Cancels
1,N
Cancel Time
Penalty
End_location
Start_location
Request_id
Est Price
1,1
Request trip
d
Cash
Amount
Card_no
Giftcard
Expiry_date
Amount
Paypal
Paypal_id
Card
Expiry_date
Name
Card_no
Type
Consists
1,1
1,1
Trip
Wait_time
Date
Start loc_id
Duration
Comments
End loc_id
Price
Trip_id

Relational Schema:

The following are the mapping rules to draw relational schema from ER Diagram.

- In the entire participation entity, add the primary key of the other entity as the foreign key for every 1:1 binary relationship.
- Add the primary key of the other entity as the foreign key to the entity on the N side of every 1: N binary connection.
- Create a new entity with the foreign key as the main key of the two participating entities for a M: N binary connection. The new primary key is formed by their combination.

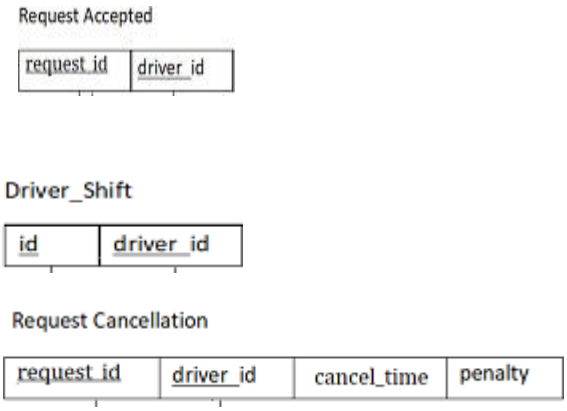
The foreign keys of the tables are as follows:

- driver_id is a foreign key in the Shift table.
- Passenger_id is a foreign key in the Cash table.
- Passenger_id is a foreign key in the Paypal table.
- Passenger_id is a foreign key in the Card table.
- Passenger_id is a foreign key in the GiftCard table.
- Passenger_id is a foreign key in the Request table.
- Request_id is a foreign key in the Trip table.
- Driver_id is a foreign key in the Cab table.
- We make a Request Accepted table with the foreign keys Driver_id and Request_id.
- Request Cancellation is a table containing foreign keys Driver_id and Request_id.
- We make a table called Driver Shift with the foreign keys Driver_id and id.

Relational Schema before Normalization

- The generalization in the Payment Entity which can have Gift-Card , Cash , PayPal , Card, and we represented each payment method as a separate relation in the relational schema diagram.
- We have overlapping between Uber_User and (Passenger, Driver). The User_id in Uber_User is referenced as Passenger_id in Passenger and driver_id in Driver.
- So, we include primary key(user_id) in subclasses as passenger_id and driver_id.

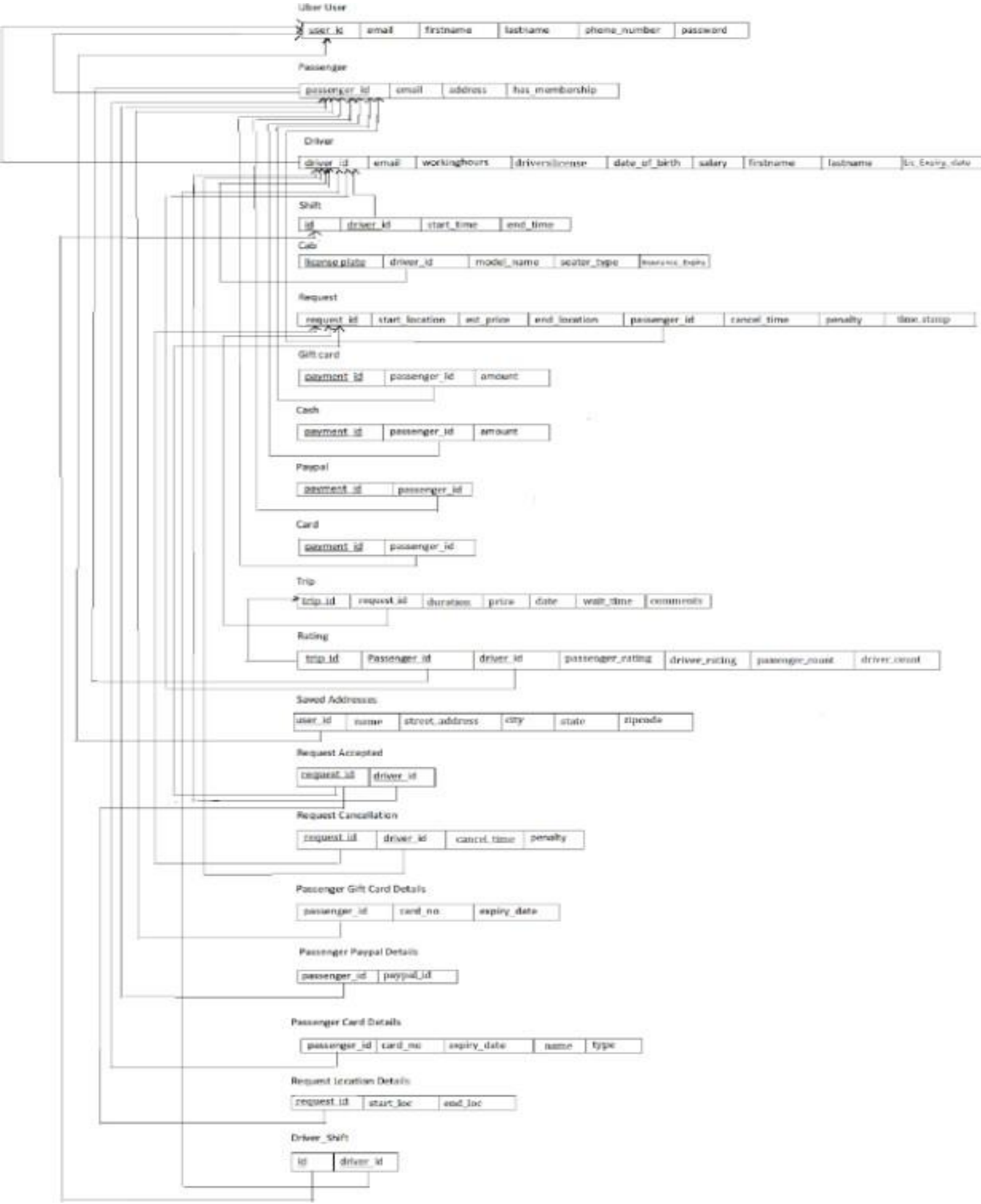
- According to the mapping rules discussed above, we have 3 M:N, hence 3 new tables have been created as below :



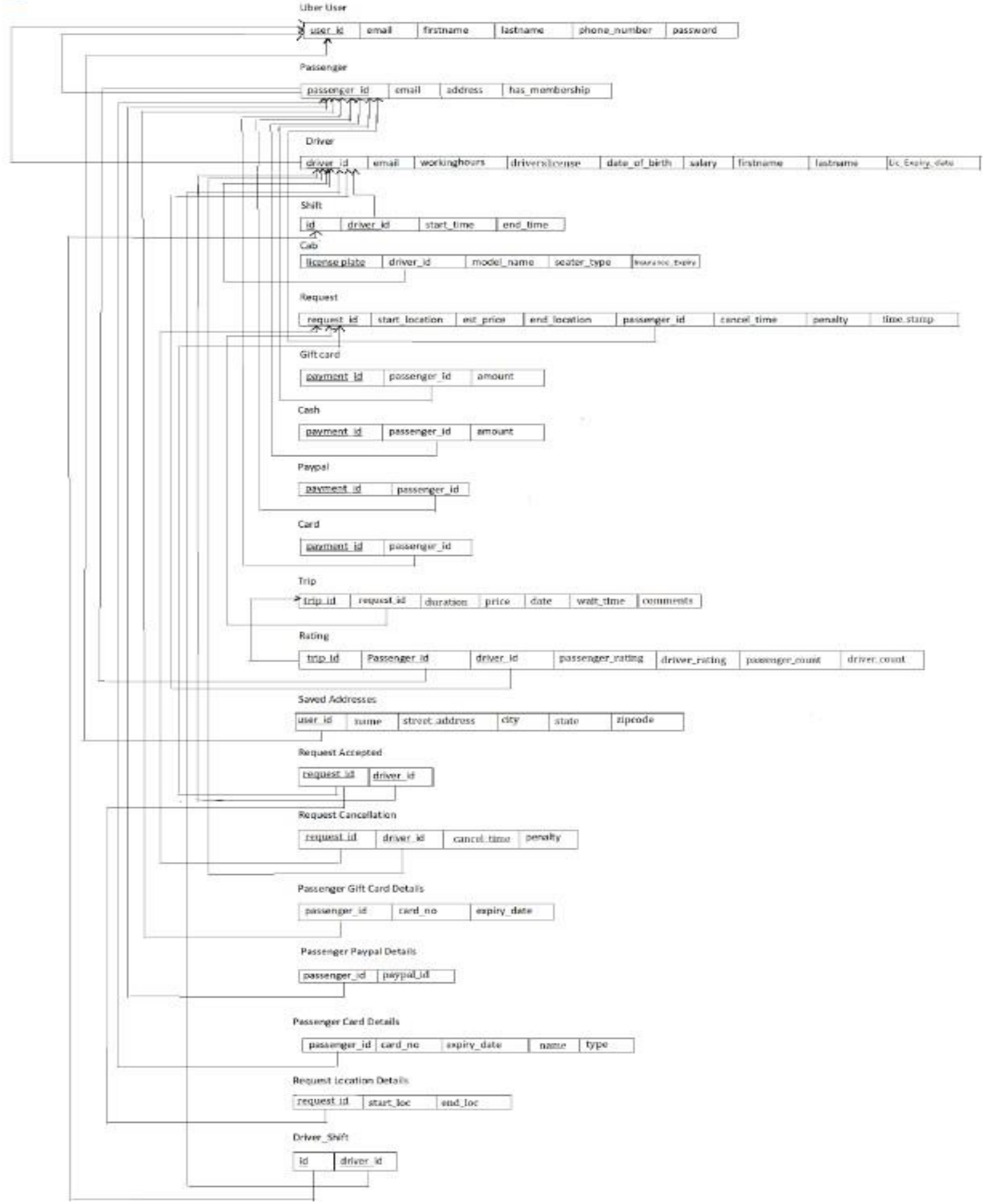
Normalization :

- 1NF : All the relations are in 1NF
- 2NF : The following relations violate 2NF
    - In Gift Card, Expiry_date and Card no are dependent only on Passenger_ID . Hence, a new Table - Passenger Gift card details is created.
    - In Paypal, Paypal_id is dependent only on Passenger_ID. Hence, new table - Passenger Paypal details is created
    - In Card, Expiry_date, Card no, Name , Type are dependent only on Passenger_ID . Hence, a new Table - Passenger card details is created.
    - In trip, start_loc and end_doc are dependent only on Request_id. Hence, a new table Request Location details is created

Relational Schema:



Relational Schema After Normalization:



Tables:

```
create table uber.Uber_user(
User_id integer primary key,
Email Varchar(40) NOT NULL,
Firstname Varchar(40) NOT NULL,
Lastname Varchar(40),
Phone_number varchar(10),
Password varchar(40)
);

create table uber.Passenger(
Passenger_id integer primary key,
Email varchar(40) not null,
Address varchar(40),
Has_membership boolean default false
);

create table uber.Driver(Driver_id integer primary key,
Email varchar(40) not null,
Workinghours integer,
Driverslicense varchar(40) not null,
Date_of_Birth date,
Salary integer not null,
firstname varchar(40) not null,
lastname varchar(40) not null,
lic_expiry_date date
);

create table uber.Shift(Id integer primary key,
driver_id integer,
Start_time integer not null,
End_time integer
);

create table uber.Cab( License_plate varchar(40) primary key,
Driver_id integer not null,
Model_name varchar(20),
Seater_type integer,
Insurance_expiry date
);

create table uber.Request(Request_id integer primary key,
Start_location varchar(40),
```

```
Est_time integer,
End_location varchar(40),
Passenger_id integer not null,
Cancel_time integer,
Penalty integer
);

create table uber.Gift_card(Payment_id integer primary key,
Passenger_id integer not null,
Amount integer
);

create table uber.Cash(Payment_id integer primary key,
Passenger_id integer not null,
Amount integer
);

create table uber.Paypal(Payment_id integer primary key,
Passenger_id integer not null
);

create table uber.Passenger_Paypal_Details(Passenger_id integer primary key,
Paypal_id integer not null
);

create table uber.Card(Payment_id integer primary key,
Passenger_id integer not null
);

create table uber.Passenger_Card_Details(Passenger_id integer primary key,
Card_no integer not null,
Expiry_date date,
Name varchar(40),
Type varchar(40)
);

create table uber.Passenger_gift_Card_Details(Passenger_id integer primary key,
Card_no integer not null,
Expiry_date date
);

create table uber.Trip(Trip_id integer primary key,
Request_id integer not null,
Payment_id integer,
```

```
Duration integer,
Price decimal not null,
Date date,
Wait_time integer,
Comment varchar(40)
);

create table uber.Request_Location_Details(Request_id integer primary key,
Start_loc varchar(20),
End_loc varchar(20)
);

create table uber.Rating(Trip_id integer primary key,
Passenger_id integer,
Driver_id integer,
Passenger_rating decimal default 2.5,
Driver_rating decimal default 2.5
);

create table uber.Saved_Address(Email varchar(40) primary key,
Name varchar(40) not null,
Street_address varchar(40),
City varchar(40),
State varchar(40),
Zipcode integer
);

create table uber.Request_Accepted(Request_id integer,
Driver_id integer
);

create table uber.Request_Cancellation(Request_id integer,
Driver_id integer not null,
Cancel_time integer,
Penalty decimal(10,2)
);

create table uber.Driver_Shift(Id integer primary key,
Driver_id integer not null
);
```

Applying Constraints for Tables:

alter table gift_card add constraint Passgift_id_fk foreign key(Passenger_id) references Passenger(Passenger_id) ON DELETE CASCADE;

alter table request add constraint Passenger_id_fk foreign key(Passenger_id) references Passenger(Passenger_id) ON DELETE CASCADE;

alter table cash add constraint Pass_cash_id_fk foreign key(Passenger_id) references Passenger(Passenger_id) ON DELETE CASCADE;

alter table paypal add constraint Paypal_id_fk foreign key(Passenger_id) references Passenger(Passenger_id) ON DELETE CASCADE;

alter table card add constraint Pass_card_id_fk foreign key(Passenger_id) references Passenger(Passenger_id) ON DELETE CASCADE;

alter table request add constraint Passenger_id_fk foreign key(Passenger_id) references Passenger(Passenger_id) ON DELETE CASCADE;

alter table shift add constraint driver_id_fk foreign key(driver_id) references Driver(driver_id) ON DELETE CASCADE;

alter table cab add constraint driver_cab_id_fk foreign key(driver_id) references Driver(driver_id) ON DELETE CASCADE;

alter table rating add constraint driver_rate_id_fk foreign key(driver_id) references Driver(driver_id) ON DELETE CASCADE;

alter table rating add constraint pass_rate_id_fk foreign key(passenger_id) references Passenger(passenger_id) ON DELETE CASCADE;

alter table rating add constraint trip_id_fk foreign key(trip_id) references Trip(trip_id) ON DELETE CASCADE;

alter table request_accepted add constraint driver_accept_id_fk foreign key(driver_id) references Driver(driver_id) ON DELETE CASCADE;

alter table request_cancellation add constraint driver_cancel_id_fk foreign key(driver_id) references Driver(driver_id) ON DELETE CASCADE;

alter table trip add constraint request_id_fk foreign key(request_id) references Request(request_id) ON DELETE CASCADE;

```
alter table request_accepted add constraint request_accept_id_fk foreign
key(request_id) references Request(request_id) ON DELETE CASCADE;

alter table request_cancellation add constraint request_cancel_id_fk foreign
key(request_id) references Request(request_id) ON DELETE CASCADE;
```

Procedures:

A stored procedure is a prepared SQL statement that you can save, so the code can be reused multiple times.
We pass parameters to a stored procedure, so that the stored procedure can act based on the values that are passed.

1)Average rating of all drivers:

```
CREATE or REPLACE PROCEDURE Avg_rating AS
CURSOR DRating IS SELECT AVG(driver_rating) AS AvgRating,driver_Id
FROM Rating
GROUP BY driver_Id;
thisRating DRating%ROWTYPE;
BEGIN
OPEN DRating;
LOOP
FETCH DRating INTO thisRating;
EXIT WHEN (DRating%NOTFOUND);
Dbms_output.put_line('Average rating of driver with
ID:'||thisRating.driver_id||is||thisRating.AvgRating);
END LOOP;
CLOSE DRating;
END;
```

2)Cancel Membership:

```
CREATE or REPLACE PROCEDURE cancel_membership(pass_id IN VARCHAR) AS
BEGIN
UPDATE passenger
SET
Has_membership = 0
WHERE passenger_id = pass_id;
END cancel_membership;
```

3) Register as a Passenger

```
CREATE OR REPLACE PROCEDURE registerpassenger ( Id
IN INTEGER,
email IN VARCHAR,
first_name IN VARCHAR,
last_name IN VARCHAR,
phone_number IN VARCHAR,
password IN VARCHAR) AS
BEGIN
INSERT INTO uber_user VALUES (Id,email,
first_name,
last_name,
phone_number,
password
);
INSERT INTO Passenger VALUES (
Id,
email,
NULL,
0
);
END register_passenger;
```

4) Register as a driver

```
CREATE OR REPLACE PROCEDURE register_driver (
    Id IN INTEGER,
```

```
   Email IN VARCHAR,
firstname IN VARCHAR, lastname IN VARCHAR, phone_number IN VARCHAR,
working_hours IN INTEGER, drivers_license IN VARCHAR, date_of_birth IN INTEGER,
   Salary IN INTEGER) AS

BEGIN

INSERT INTO uber_user VALUES (

Id,

Email,

firstname,

lastname,

phone_number,

NULL

);

INSERT INTO Driver VALUES (

   Id,

   Email,

   working_hours,

   drivers_license,

   date_of_birth,

   NULL

   );

END;
```

Triggers:

1. Validate the driver’s license expiry date

   Each driver ought to have a substantial and non-lapsed driver's permit thus we add a trigger to approve. In the event that on the off chance that any driver's permit is terminated, then consequently a message invokes stating the same.

```
create or replace TRIGGER LicenseExpiry
before insert or update on DRIVER for each row
Begin
if (:new.lic_Expiry_date < sysdate) then
raise_application_error ( -20098, 'The driver’s license is expired and so couldn’t accept the request. Please update or renew the license');
end if;
End;
```

2. Validate the vehicle insurance expiry date

   Every vehicle/cab should have a substantial and non-lapsed insurance thus we add a trigger to approve. If in any case the vehicle insurance is expired, then automatically invokes a message stating the same.

```
create or replace TRIGGER InsuranceExpiry
before insert or update on cab for each row
Begin
if (:new.Insurance_Expiry < sysdate) then
raise_application_error ( -20099,’The Insurance for the vehicle is expired. Please renew and update the new information.’);
end if;
End;
```

RESULTS AND OUTPUT:

1. Average Rating Procedure:

Assuming the following values are in the table rating before the execution of procedure.

insert into rating values(18,1023, 791456, 3.5, 4);
insert into rating values(10,4156, 859633, 4,3.4);
insert into rating values(60,1925, 794056, 5.5, 4);

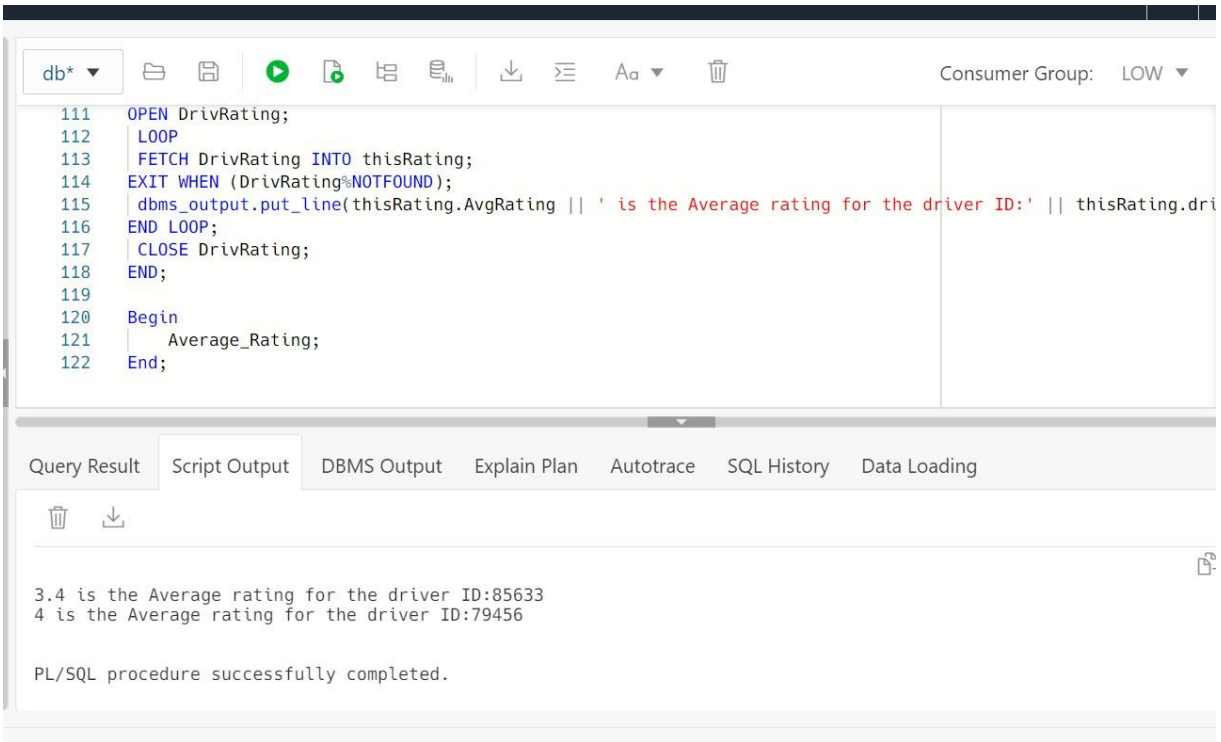
Procedure call:

Begin

Average_Rating;

End;

Output screenshot:



Triggers:

update DRIVER set lic_Expiry_date = '20-MAY- 16' where Driver_id= 79456;

Output screenshot: