Project Report

Team Members:

- *Vasepalli Dinnupratyusha(Dxv200013@utdallas.edu)*
- *Naveena Kavya Somepalli (nxs200003@utdallas.edu)*
- *Yasaswini Muthineni (yxm200020@utdallas.edu)*

The project topic is based on creating Uber database which fulfill basic functional requirements. The following are the date requirements.

Data Requirements:

1. **User Can Register -** User can be a driver or a Passenger.
2. **User can save places –** User can add home, work, and multiple other addresses.
3. **Passenger places a request** – Each request has one user only.
4. **Driver accepts or cancels a Request –** Driver can accept or cancel multiple requests.
5. **Driver manages cab –** Driver can drive multiple cabs.
6. **Driver's Shift –** Details of driver's shift.
7. **User can have multiple Payment methods -** user can have multiple cards saved, gift cards.
8. **User rates a Trip** - Both passenger and driver can give ratings.

Relationships:

1. **Address List:** User can save multiple addresses, but an address can belong to one user. Thus, cardinality ratio is 1:N.
2. **Driver's Shift:** Driver can have multiple shifts and a shift can also have multiple drivers at the same time. Thus, cardinality ratio is M:N.
3. **Driver drives Cab:** A driver can drive multiple cabs, but a cab can only have one driver. Thus, the cardinality. 1:N

4. **Rating :** A driver can give single rating for Passenger in a trip and a passenger can give single rating for a driver in the trip. Thus, the cardinality. 1:1
5. **Passenger's payment method :** Passenger can have multiple payment methods, but Payment is linked to single passenger. Thus, the cardinality. 1:N
6. **Passenger – Request :** Passenger can place one request and request belongs to a single passenger. Thus, the cardinality. 1:1
7. **Passenger – Request Cancellation :** Passenger can cancel a request and a request cancellation belongs to a single passenger. Thus, the cardinality. 1:1
8. **Driver – Request :** Driver can have multiple requests and a request goes to multiple Drivers. Thus, the cardinality. M:N
9. **Driver – Request Cancellation :** Driver can cancel multiple requests and a request can be cancelled by multiple Drivers. Thus, the cardinality. M:N
10. **Request – Trip :** A request can belong to a single trip and a trip can belong to single request. Thus, the cardinality. 1:1

Number of 1:1 Relationships : 4

Number of 1:N Relationships : 3

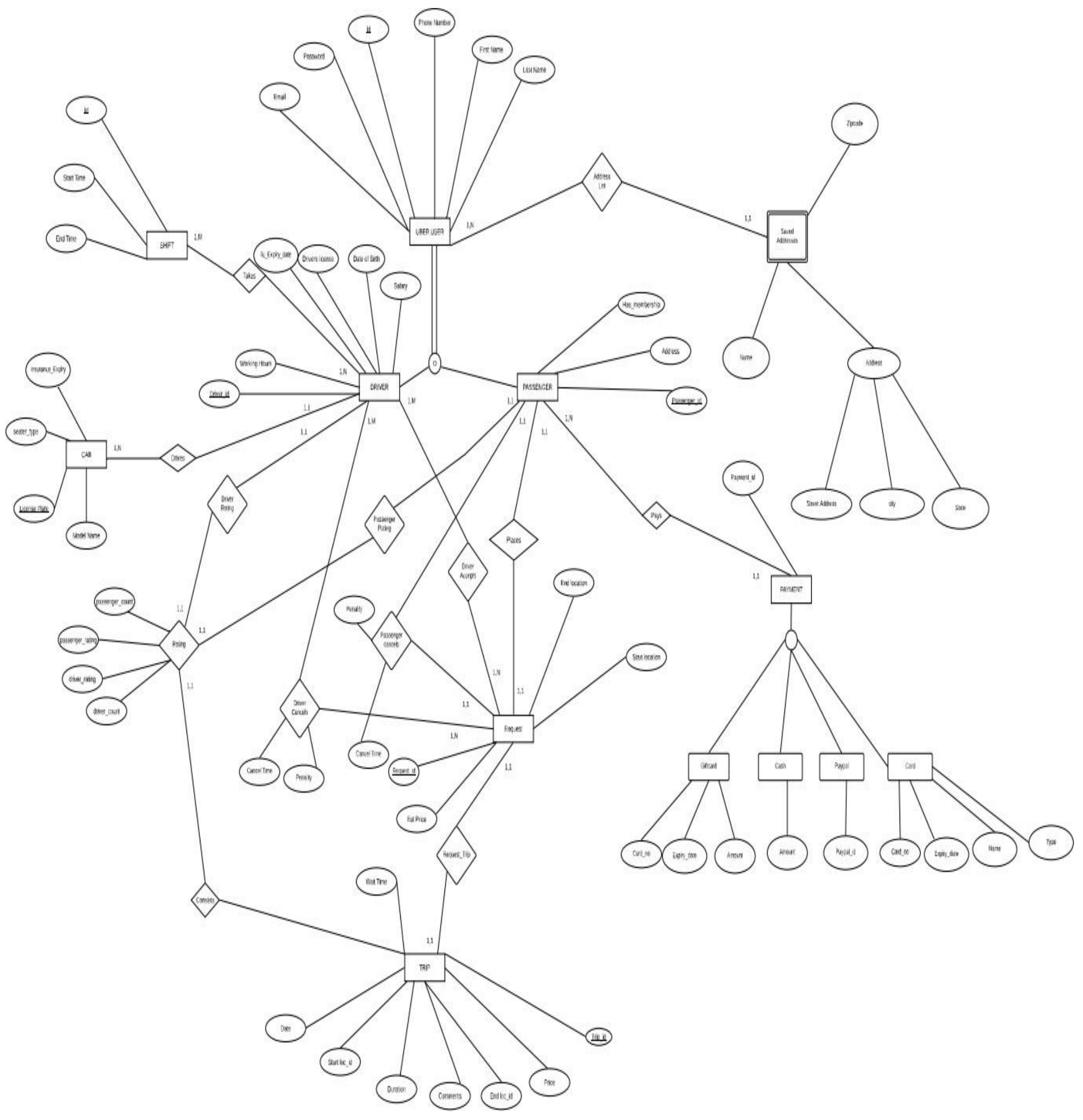
Number of M:N Relationships : 3

Total Number of Relationships : 10

ENTITY RELATIONAL DIAGRAM:

Google drive link: Please download for image clarity

Link: https://drive.google.com/file/d/14fCAK8Fv6LL720SNUq4NIPORRbIhlYvL/view?usp=sharing



RELATIONAL SCHEMA:

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

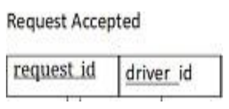
1. For every 1:1 binary relationship, in the total participation entity add the primary key of the other entity as the foreign key.
2. For every 1:N binary relationship, add to the entity on the N side the primary key of the other entity as the foreign key.
3. For M: N binary relationship, make a new entity with foreign key as the primary key of the two participating entities. Their combination forms the new primary key.

The following are the foreign keys of the tables:

- In Shift table, driver_id is foreign key.
- In Cash table , Passenger_id is foreign key.
- In Paypal table , Passenger_id is foreign key.
- In Card table , Passenger_id is foreign key.
- In GiftCard table , Passenger_id is foreign key.
- In Request table, Passenger_id is foreign key.
- In Trip table Request_id is foreign key.
- In Cab table, Driver_id is foreign key.
- We create a table Request Accepted with foreign keys as Driver_id , Request_id.
- We create a table Request Cancellation with foreign keys as Driver_id , Request_id.
- We create a table Driver_Shift with foreign keys as Driver_id ,id.

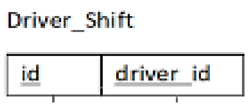
Following is the relational Schema before Normalization:

1. We have a 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.
2. 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.
3. We include primary key(user_id) in subclasses as passenger_id and driver_id.
4. According to the mapping rules discussed above, we have 3 M:N hence 3 new tables have been created as below :

1. 

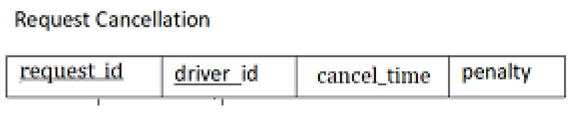
Request Accepted

request_id	driver_id

2. 

Driver_Shift

id	driver_id

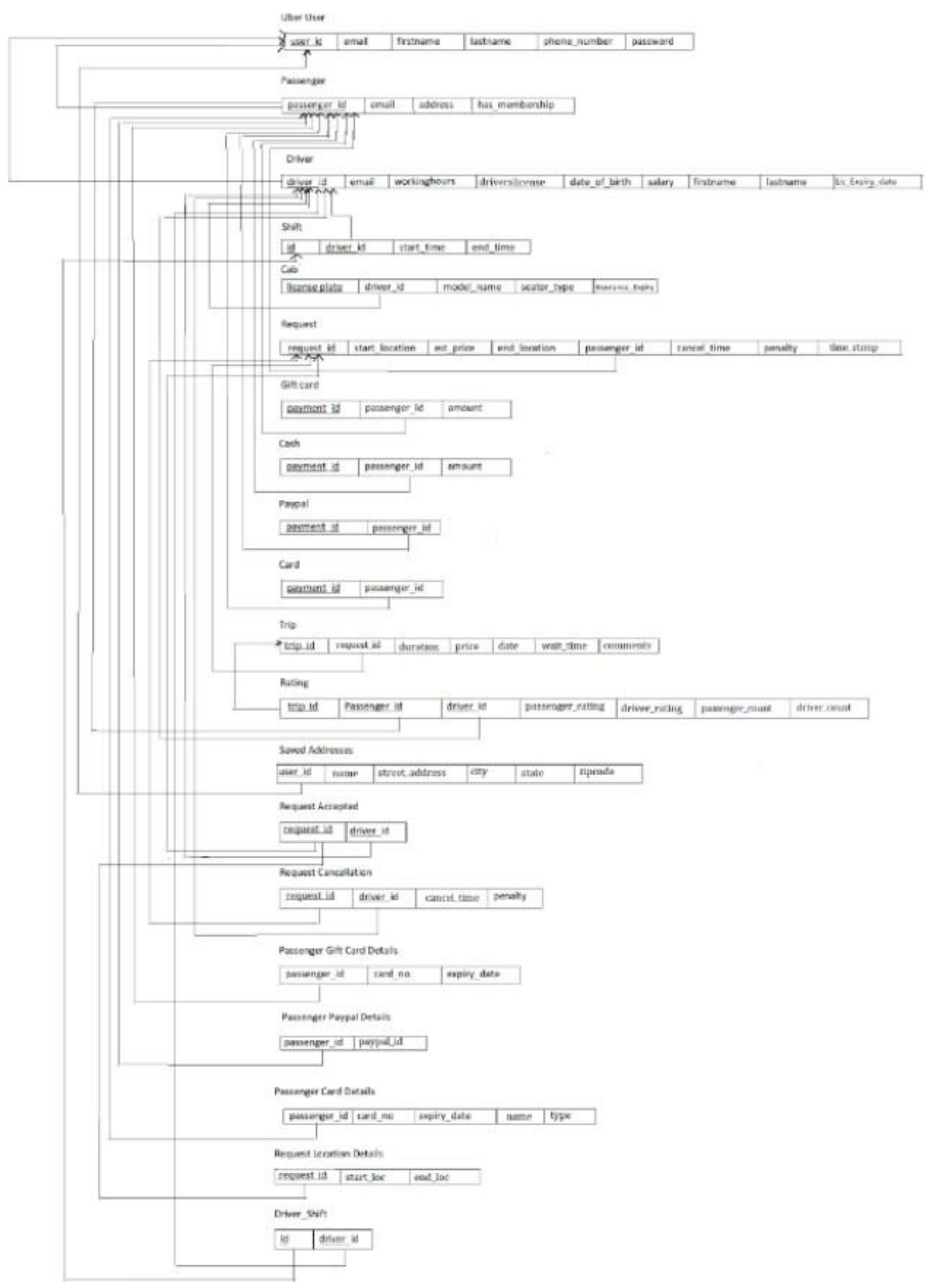
3. 

Request Cancellation

request_id	driver_id	cancel_time	penalty

Google Drive link :
https://drive.google.com/file/d/1NN3Qgg_PbrNyyWEP6RYAPDMtRrf8OEsB/view?usp=sharing

Database Design



Normalization :

1NF : All the relations are in 1NF

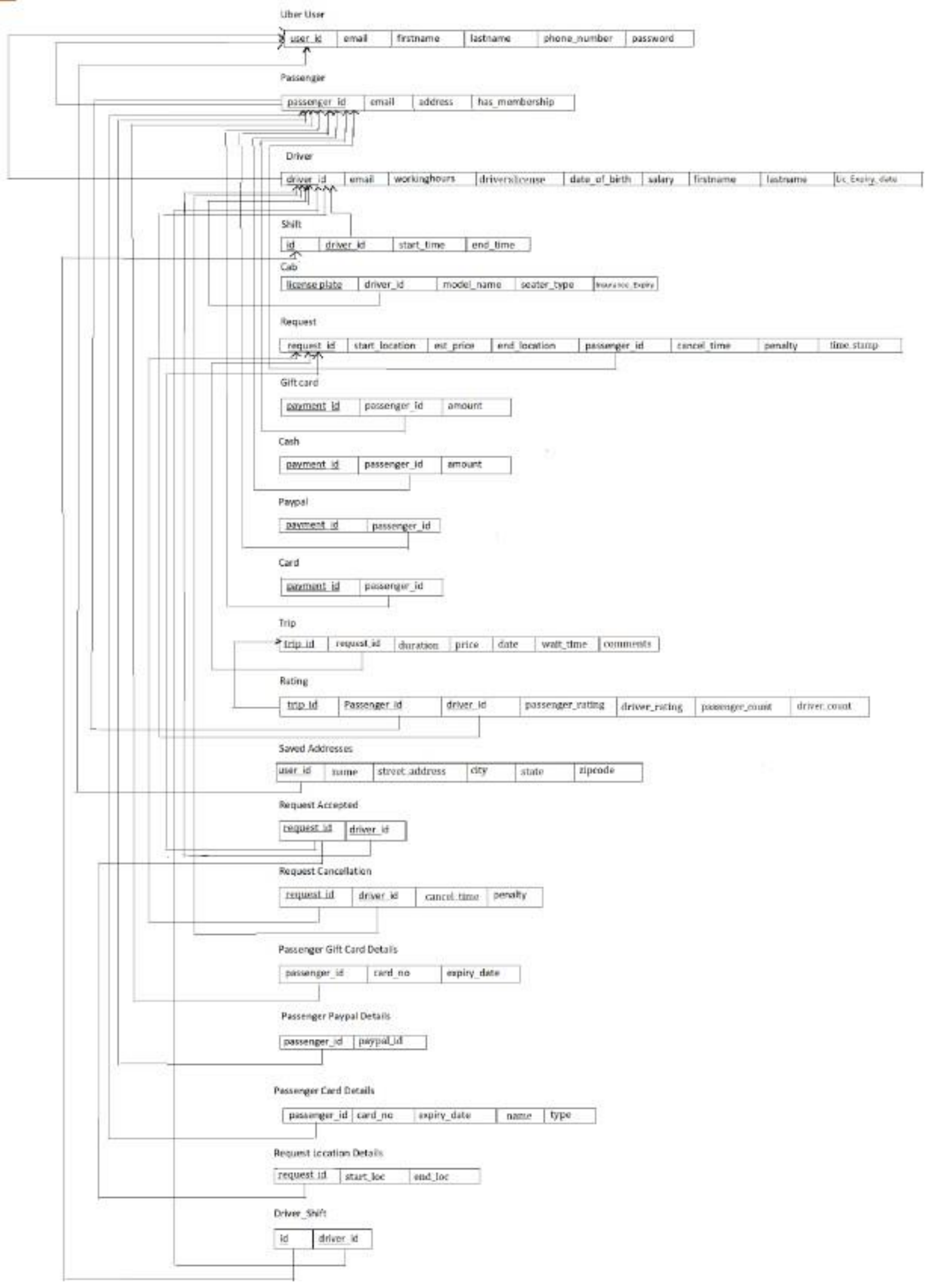
2NF : The following relations violate 2NF

a. In Gift Card, Expiry_date and Card no are dependent only on Passenger_ID . Hence, a new Table - Passenger Gift card details is created.
b. In Paypal, Paypal_id is dependent only on Passenger_ID. Hence, new table - Passenger Paypal details is created
c. In Card, Expiry_date, Card no, Name , Type are dependent only on Passenger_ID . Hence, a new Table - Passenger card details is created.
d. In trip, start_loc and end_doc are dependent only on Request_id. Hence, a new table Request Location details is created.

3NF : All the relations are in 3NF

<u>Following the relational Schema After Normalization:</u>

Google Drive Link : https://drive.google.com/file/d/1jO1y-c_LYpJeng3dHzaVFuNmXe9HxsOl/view?usp=sharing


Uber User
user_id
email
firstname
lastname
phone_number
password
Passenger
passenger_id
email
address
has_membership
Driver
driver_id
email
workinghours
driverslicense
date_of_birth
salary
firstname
lastname
Lic_Expiry_date
Shift
id
driver_id
start_time
end_time
Cab
license plate
driver_id
model_name
seater_type
Insurance_Expiry
Request
request_id
start_location
est_price
end_location
passenger_id
cancel_time
penalty
time stamp
Gift card
payment_id
passenger_id
amount
Cash
payment_id
passenger_id
amount
Paypal
payment_id
passenger_id
Card
payment_id
passenger_id
Trip
trip_id
request_id
duration
price
date
wait_time
comments
Rating
trip_id
Passenger_id
driver_id
passenger_rating
driver_rating
passenger_count
driver_count
Saved Addresses
user_id
name
street address
city
state
zipcode
Request Accepted
request_id
driver_id
Request Cancellation
request_id
driver_id
cancel_time
penalty
Passenger Gift Card Details
passenger_id
card_no
expiry_date
Passenger Paypal Details
passenger_id
paypal_id
Passenger Card Details
passenger_id
card_no
expiry_date
name
type
Request Location Details
request_id
start_loc
end_loc
Driver_Shift
id
driver_id

PROCEDURES:

1. **Drivers average rating:** This procedure is to calculate the average rating of each driver.

2. **Cancel Uber membership:** This procedure is to cancel the uber membership for the passenger.

3. **Register Passenger**: Invoked by Passenger responsible for

   a) registering user given email, fname, lname, phone number and password.

   b) registering the Passenger itself setting its uber membership as false by default.

4. **Register Driver**: has 2 responsibilities

   a) registering user given email, fname, lname , phone number and password.

   b) registering the driver itself given working hours, salary, Drivers license, date of birth and setting its average rating as 2.5 default.

5. **To check if user already exists:**
   a) Procedure validates user details and doesn't allow duplicate details.

6. **To save multiple addresses:**
   a) User can save multiple user addresses.
   b) Adds information like Street address, city , state, zipcode

7. **Add card information:**
   a) add cards information like card number, expiry_date , name, card type.

8. **Book a Trip :**
   a) To add information regarding trip details like trip_id , request_id, payment_id , price , trip_daye, wait_time , comments.

TRIGGERS :

1. **Trigger to check that the Driver's license should not have expired:**

   This is a trigger which is used for validating the driver's license.

2. **Trigger to check that the Insurance for the vehicle should not have expired:**

   This is a trigger which is used to check if the cabs insurance had expired.

CODE FOR UBER SYSTEM:

TABLES :

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

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

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

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

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

create table Paypal(Payment_id integer primary key,
Passenger_id integer not null
```

```
);
```

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

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

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

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

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

create table 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 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 Request_Accepted(Request_id integer,
Driver_id integer
);

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

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

create table Saved_Address(User_id integer primary key,
Name varchar(40) not null,
Street_address varchar(40),
City varchar(40),
State varchar(40),
Zipcode integer
);
```

APPLYING CONSTRAINTS:

```
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;
```

CODE FOR STORED PROCEDURES AND TRIGGERS:

PROCEDURES:

1. **To calculate the average rating of all the drivers:**

```
create or replace PROCEDURE Average_Rating AS

 CURSOR DrivRating IS SELECT AVG(driver_rating) as AvgRating, driver_Id
FROM

 Rating GROUP BY driver_Id;

thisRating DrivRating%ROWTYPE;

BEGIN

OPEN DrivRating;

 LOOP

 FETCH DrivRating INTO thisRating;
EXIT WHEN (DrivRating%NOTFOUND);

 dbms_output.put_line(thisRating.AvgRating || ' is the Average rating for
the driver ID:' || thisRating.driver_id);

END LOOP;

 CLOSE DrivRating;
```

```
END;
```

2. Cancel Uber membership:

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

3. Procedure to Register a Passenger

```
CREATE OR REPLACE PROCEDURE register_passenger (
Id IN INTEGER,
email  IN VARCHAR,
firstname IN VARCHAR,
lastname IN VARCHAR,
phone_number IN VARCHAR,
password IN VARCHAR)
AS BEGIN

INSERT INTO uber_user VALUES ( Id, email,
firstname, lastname, phone_number,password);


INSERT INTO Passenger VALUES (Id,email,NULL,0);

END register_passenger;
```

4. Procedure to Register 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;
```

5. **Check if user already exists**

```
CREATE OR REPLACE PROCEDURE check_user (Email IN VARCHAR) AS
DECLARE UserExists INT;
BEGIN
IF EXISTS(SELECT 1 FROM UBER_USER WHERE Email = Email)
   BEGIN

      SET UserExists = 1;
   END
ELSE
      BEGIN
         SET UserExists= 0;
      END
IF (UserExists = 1)
   BEGIN
        RAISERROR('User exists already',16,1);
         ROLLBACK;
   END
```

6. **Procedure to Save multiple addresses:**

```
CREATE OR REPLACE PROCEDURE add_saved_addresses (
   email IN VARCHAR,
   Name IN VARCHAR,

   Street_address IN VARCHAR,
   city IN VARCHAR,
   state IN VARCHAR,
   zipcode IN NUMBER
   ) AS

   BEGIN

   INSERT INTO saved_address VALUES (email, name,
   street_address,city, state, zipcode );
   END;
```

7. **Procedure to add card info:**

```
CREATE OR REPLACE PROCEDURE add_card_info (
  passenger_id IN INTEGER,
  card_number IN NUMBER,
  expiry_date IN DATE,
  name IN VARCHAR,
  type IN VARCHAR

  ) AS

  BEGIN
  INSERT INTO Passenger_Card_Details VALUES (Passenger_id,

  card_number,expiry_date, name,type );

  END ;
```

8. **Procedure to book a trip:**

```
CREATE OR REPLACE PROCEDURE book_trip (

  Trip_id IN INTEGER,
  Request_id IN INTEGER,
  Payment_id IN INTEGER,
  Duration IN NUMBER,
  Price IN DECIMAL,
  date in Date,
  Wait_time IN INTEGER,
  comments IN VARCHAR
) AS


BEGIN
INSERT INTO Trip VALUES (

  Trip_id, Request_id, Payment_id,Duration, Price, trip_date, Wait_time,
  Comments);

  END ;
```

9. **procedure to accept the trip request:CREATE OR** REPLACE PROCEDURE accept_request (Request_id IN INTEGER,

```
   Driver_id IN INTEGER,

   ) AS
   BEGIN
   INSERT INTO Request_Accepted
      VALUES (Request_id, Driver_id);
   END;
```

Triggers:

Assuming the following values of driver table are inserted in the table before the trigger " Expiry"

```
insert into driver values(79456, 'yash@gmail.com',6, 'ya6789',TO_DATE('1989-12-09','YYYY-MM-DD'),70000,'yashu','vinny',TO_DATE('1997-11-17','YYYY-MM-DD'));
```

1. **Trigger to check if the driver's license is expire.**

```
create or replace TRIGGER Expiry
before insert or update
on DRIVER for each row
```

```
Begin
if (:new.lic_Expiry_date < sysdate) then
raise_application_error( -
20098, 'Update cannot happen as the driver license is expired');
end if;
End;
```

Assuming the following values of cab table are inserted in the table before the trigger " Insurance_Expirydate"

```
insert into cab values('ab123', 79456, 'audi',4,TO_DATE('2020-12-09','YYYY-MM-DD'));
```

2. **Trigger to check that the Insurance for the cab should not have expired:**

```
create or replace TRIGGER Insurance_Expirydate
before insert or update
on cab for each row
Begin
if (:new.Insurance_Expiry < sysdate) then
raise_application_error( -
20099, 'This is a custom error for Insurance : Insurance has expired');
end if;
End;
```

RESULTS AND OUPUT:

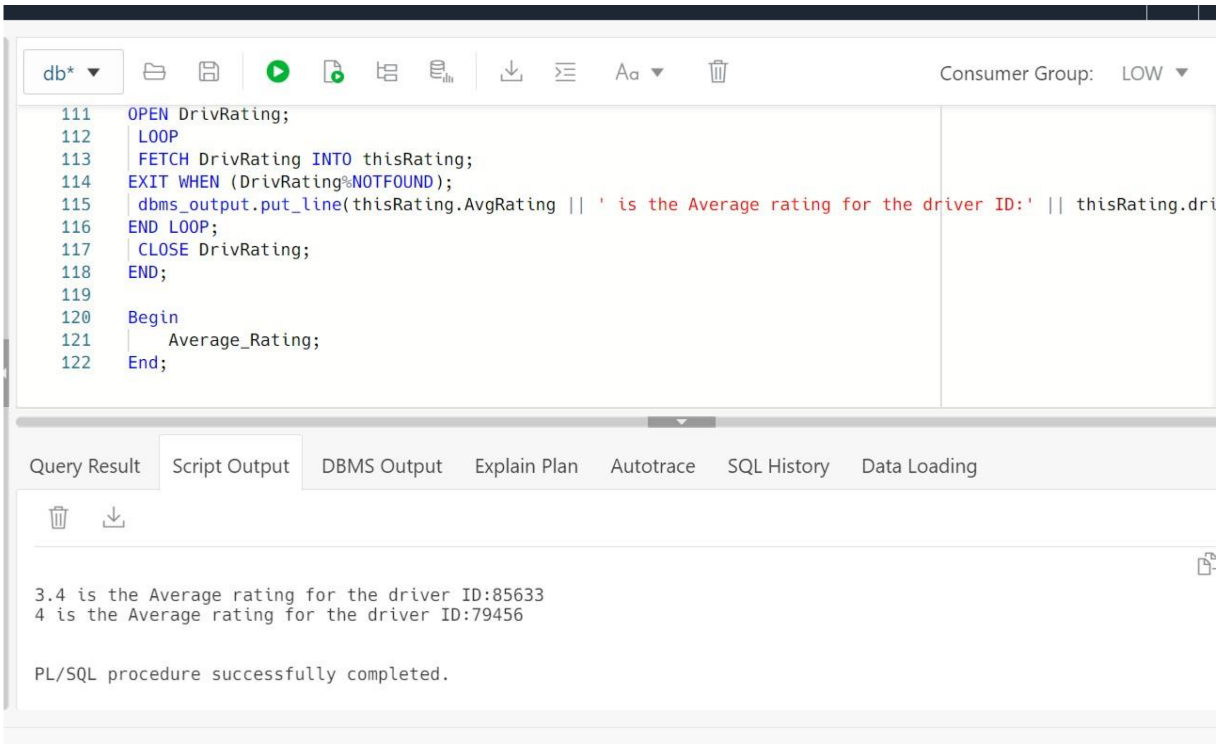
1. **Procedure for rating:**

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

```
insert into rating values(8,123, 79456, 3.5, 4 );
insert into rating values(1,456, 85633, 4,3.4);
insert into rating values(6,125, 79456, 5.5, 4 );
```

Procedure call:

```
Begin
Average_Rating;
End;
```



```
OPEN DrivRating;
 LOOP
 FETCH DrivRating INTO thisRating;
EXIT WHEN (DrivRating%NOTFOUND);
 dbms_output.put_line(thisRating.AvgRating || ' is the Average rating for the driver ID:' || thisRating.dri
END LOOP;
 CLOSE DrivRating;
END;

Begin
    Average_Rating;
End;
```

```
3.4 is the Average rating for the driver ID:85633
4 is the Average rating for the driver ID:79456

PL/SQL procedure successfully completed.
```

Output screenshot:

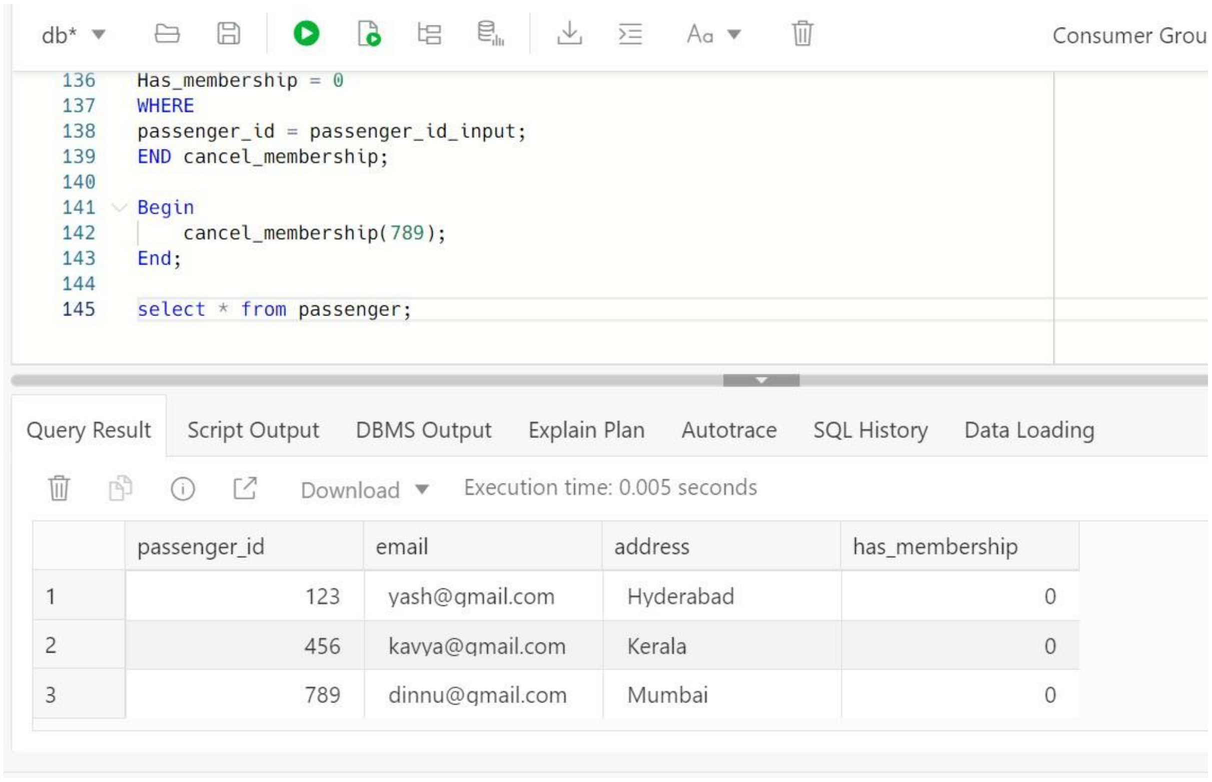
2. Assuming the following values are already in the table passenger:

```
insert into passenger values(123, 'yash@gmail.com', 'Hyderabad', 0);

insert into passenger values(456, 'kavya@gmail.com', 'Kerala', 0);

insert into passenger values(789, 'dinnu@gmail.com', 'Mumbai', 1);
```

Procedure call:

```
        Begin
       cancel_membership(789);
End;
```

Select * from passenger;



```
Has_membership = 0
WHERE
passenger_id = passenger_id_input;
END cancel_membership;

Begin
    cancel_membership(789);
End;

select * from passenger;
```

Query Result | Script Output | DBMS Output | Explain Plan | Autotrace | SQL History | Data Loading

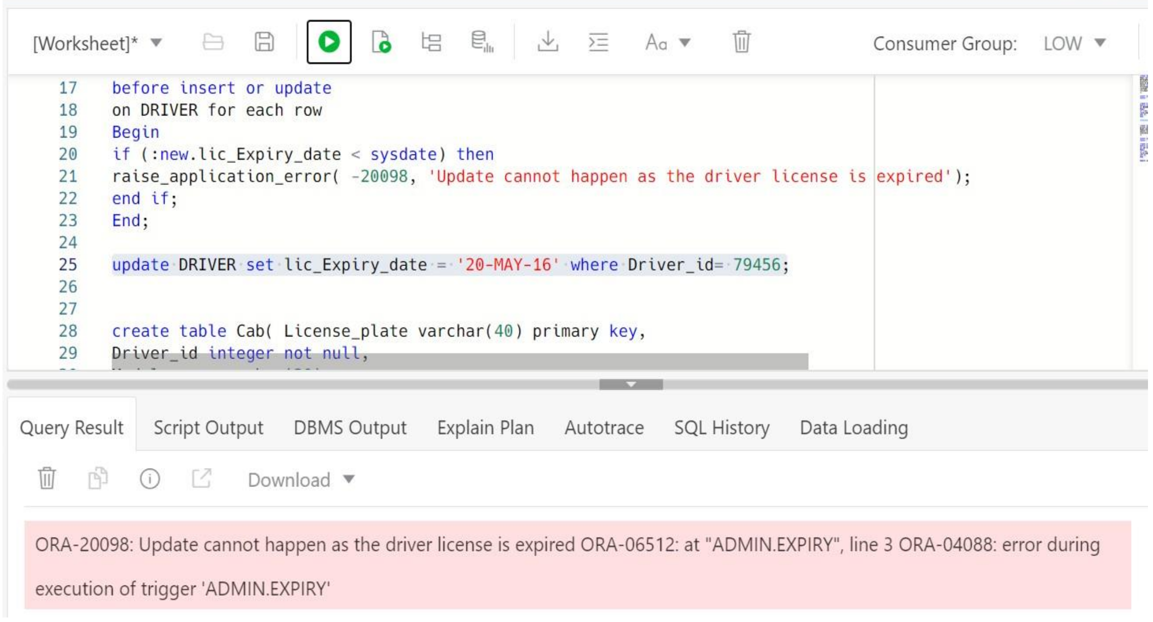
Execution time: 0.005 seconds

	passenger_id	email	address	has_membership
1	123	yash@gmail.com	Hyderabad	0
2	456	kavya@gmail.com	Kerala	0
3	789	dinnu@gmail.com	Mumbai	0

TRIGGERS:

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

OUTPUT SCREENSHOT:

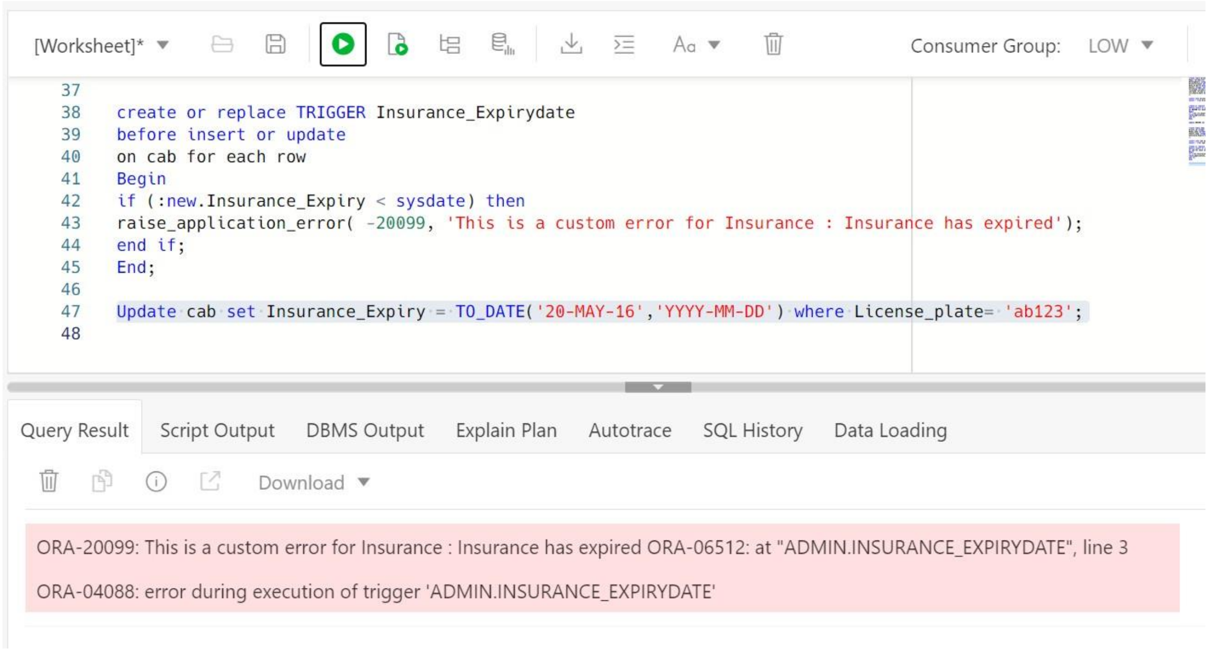


```
before insert or update
on DRIVER for each row
Begin
if (:new.lic_Expiry_date < sysdate) then
raise_application_error( -20098, 'Update cannot happen as the driver license is expired');
end if;
End;

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


create table Cab( License_plate varchar(40) primary key,
Driver_id integer not null,
```

```
ORA-20098: Update cannot happen as the driver license is expired ORA-06512: at "ADMIN.EXPIRY", line 3 ORA-04088: error during execution of trigger 'ADMIN.EXPIRY'
```

2. Update cab set Insurance_Expiry = TO_DATE('20-MAY-16','YYYY-MM-DD') where License_plate= 'ab123';

OUTPUT SCREENSHOT:



```

create or replace TRIGGER Insurance_Expirydate
before insert or update
on cab for each row
Begin
if (:new.Insurance_Expiry < sysdate) then
raise_application_error( -20099, 'This is a custom error for Insurance : Insurance has expired');
end if;
End;

Update cab set Insurance_Expiry = TO_DATE('20-MAY-16','YYYY-MM-DD') where License_plate= 'ab123';

```

```
ORA-20099: This is a custom error for Insurance : Insurance has expired ORA-06512: at "ADMIN.INSURANCE_EXPIRYDATE", line 3
ORA-04088: error during execution of trigger 'ADMIN.INSURANCE_EXPIRYDATE'
```