PROJECT REPORT

PROJECT TITLE – UBER
COURSE NO. AND SECTION – CS 6360.004
TEAM NUMBER – UBER-TEAM 5
TEAM MEMBERS – 1. Anish Joshi (axj200101)
2. Sirisha Satish (sxs210095)

DATA REQUIREMENTS –

This project is based on creating an Uber Database which fulfills the basic requirements at par similar with the real world system.
Following are the data requirements:
Starting with the user/passenger/customer, we have the following requirements to be fulfilled:

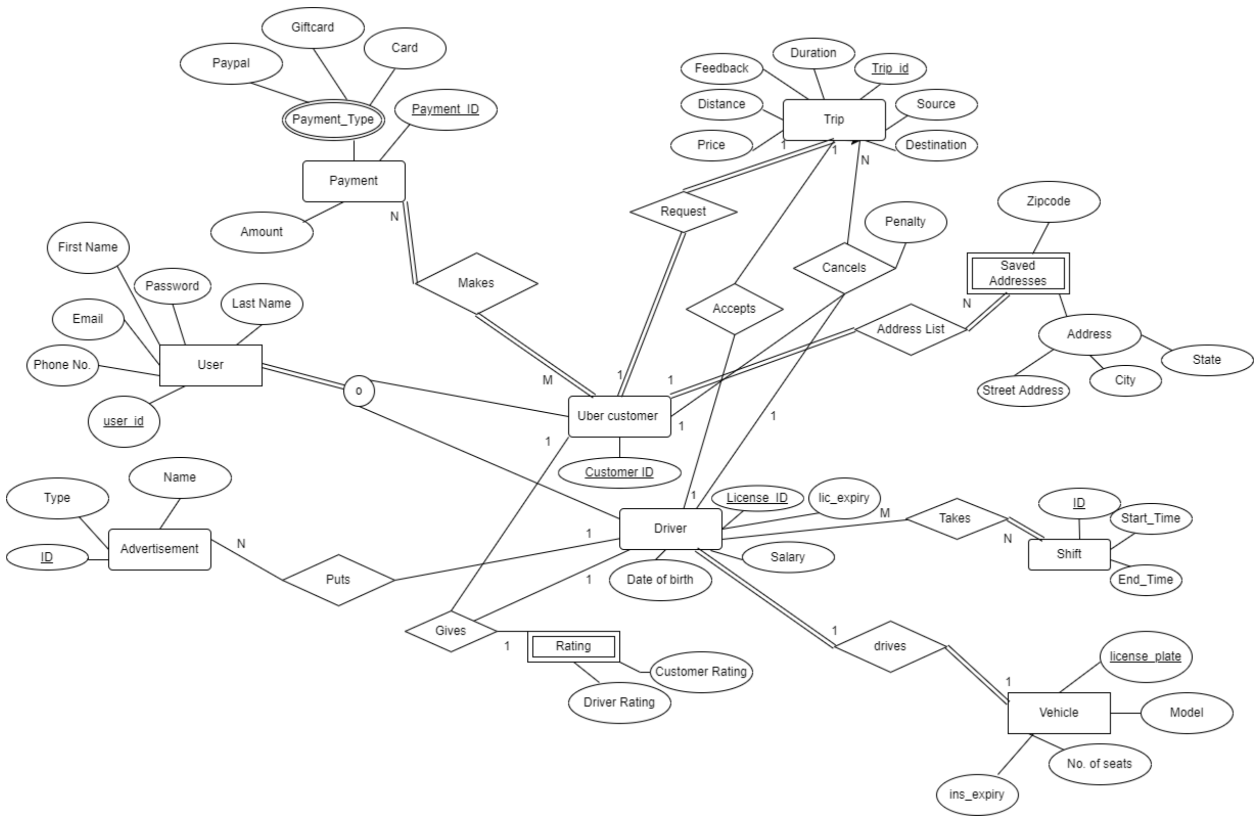
1. **Registration** – A user can be either a passenger or a driver. If it's a driver, then he/she has to enter vehicle details as well.
2. **Payment methods** – A user can save multiple payment methods such as Card, Giftcard, and PayPal.
3. **Saved places** – The user has the option to add a home, work and multiple addresses under "other".
4. A customer can cancel his/her trip and a penalty fee is charged.
   Continuing onto the trip, we have the following requirements to be fulfilled:
5. **Trip rating** – The passenger/customer should be able to rate the trip and give rating points to the driver on the basis of two or three parameters.
6. **Trip requests –** A trip request can be made by only one customer at a time. Continuing onto the driver, we have the following requirements to be fulfilled:
7. Driver has the freedom to either accept a trip request or decline a trip request. He should also be able to cancel a trip after accepting it on the basis of some ground rules.
8. The personal details of the driver along with its shift details.
9. A driver may also put advertisements in his cars as per his/her wish.
10. A driver drives a car..

RELATIONSHIPS –

1. Uber customer–gives–rating : 1:1 cardinality ratio.
2. Driver–gives–rating: 1:1 cardinality ratio.
3. Driver–puts–advertisement: 1:N cardinality ratio.
4. Driver–drives–vehicle: 1:1 total participation cardinality ratio.
5. Uber customer–Address List–Saved addresses: 1:N total participation cardinality ratio.
6. Uber customer–makes–Payments: M:N total participation cardinality ratio.
7. Uber customer–requests–Trip: 1:1 cardinality ratio.
8. Driver–accepts–Trip: 1:1 cardinality ratio.
9. Driver–cancels–Trip: 1:N cardinality ratio.
10. Uber customer–cancels–Trip: 1:N cardinality ratio.
11. Driver–takes–shift: M:N cardinality ratio.

Total 1:1 relationships = 5.
Total 1:N relationships = 4.
Total M:N relationships = 2.
Total relationships = 11.

The following is the ER diagram:



The following is the Entity Relationship diagram.
Google Drive Link :
https://drive.google.com/file/d/1pvNnptonEgkl7y429GuMazri2WqYuRgi/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 the Trip table, lic_id is a foreign key.
- In the Trip table, cust_ID is a foreign key.
- In the Takes_shift table, lic_id is a foreign key.
- In the Gives_Rating table, cust_ID is the foreign key.
- In the Trip_Cancel, cust_ID is a foreign key.
- In the Cust_Address table, cust_ID is a foreign key.
- In the Rating_Driver, cust_ID is a foreign key.
- In the Cust_Payment table, cust_ID is a foreign key.
- In the Vehicle_Driver table, lic_id is a foreign key.
- In the Advertisement table, lic_id is a foreign key.
- In the Rating_Cust table, lic_id is a foreign key.
- In the Gives_Rating table, lic_id is a foreign key.
- In the Takes_Shift table, shift_ID is a foreign key.
- In the Cust_Payment table, payment_ID is a foreign key.
- In the Vehicle_Driver table, lic_plate is a foreign key.

RELATIONAL MAPPING –

We have two M:N relationships which will have their own separate relations as follows:

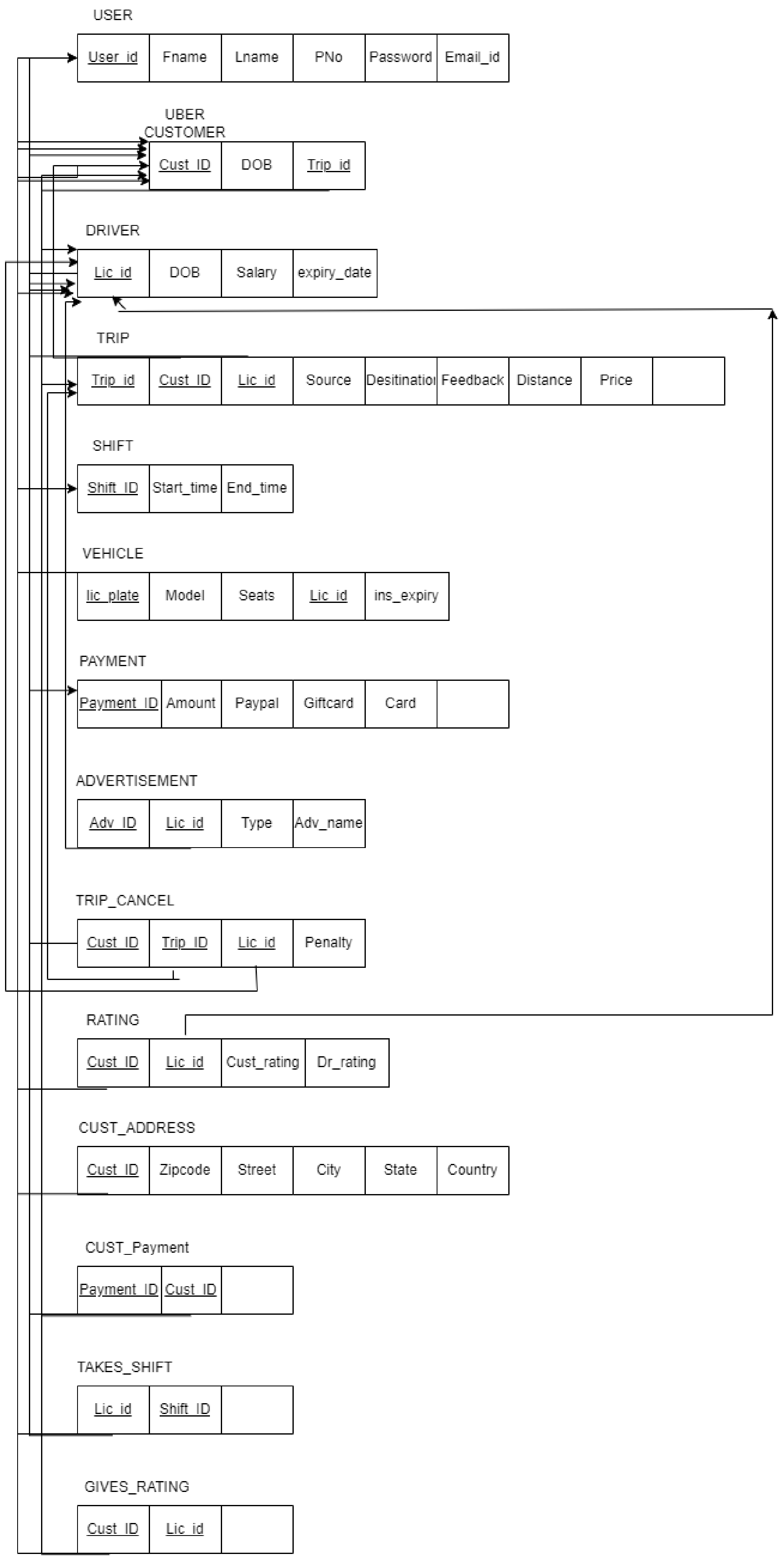
1. Trip_taken(Shift_ID,Lic_id)
2. Payment_made(Payment_ID,cust_ID)

We have two ternary relationships which will also have their own separate relations as follows:

1. trip_cancellation(Trip_ID,cust_ID,Lic_id,penalty)
2. gives_rating(cust_ID,lic_id)

The following is the Relational mapping of the ER diagram:
Google Drive Link:
https://drive.google.com/file/d/1CwNSw9fr5EfY9sWiTN_ad5BeQA0LzvvL/view?usp=sharing

Diagram on the next page.


USER
User_id | Fname | Lname | PNo | Password | Email_id
UBER CUSTOMER
Cust_ID | DOB | Trip_id
DRIVER
Lic_id | DOB | Salary | expiry_date
TRIP
Trip_id | Cust_ID | Lic_id | Source | Desitinatioi | Feedback | Distance | Price
SHIFT
Shift_ID | Start_time | End_time
VEHICLE
lic_plate | Model | Seats | Lic_id | ins_expiry
PAYMENT
Payment_ID | Amount | Paypal | Giftcard | Card
ADVERTISEMENT
Adv_ID | Lic_id | Type | Adv_name
TRIP_CANCEL
Cust_ID | Trip_ID | Lic_id | Penalty
RATING
Cust_ID | Lic_id | Cust_rating | Dr_rating
CUST_ADDRESS
Cust_ID | Zipcode | Street | City | State | Country
CUST_Payment
Payment_ID | Cust_ID
TAKES_SHIFT
Lic_id | Shift_ID
GIVES_RATING
Cust_ID | Lic_id

NORMALIZATION –

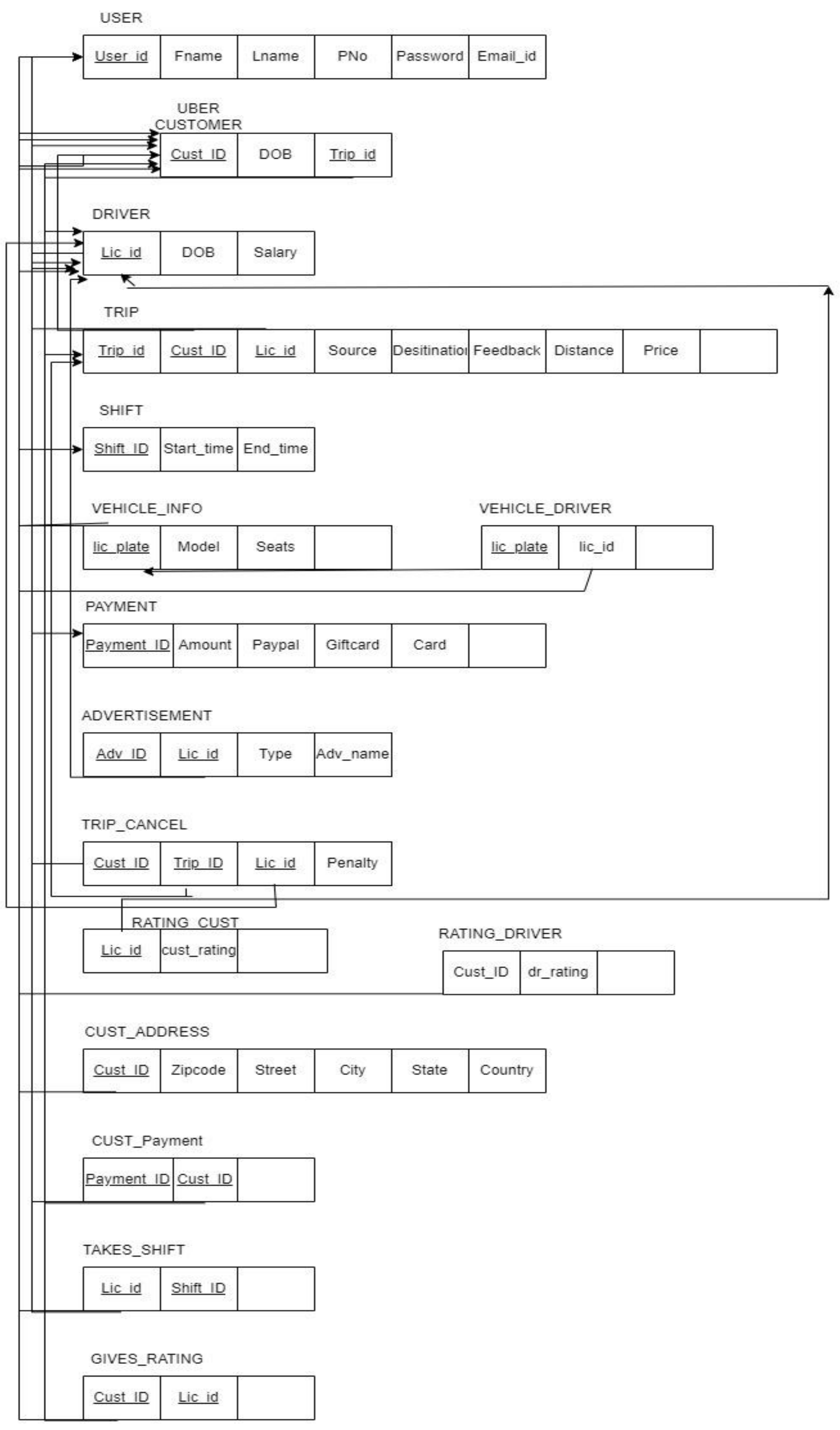
1NF: All the relations are already in 1NF.

2NF: The following are the 2NF violations and their solutions:

1. In the rating table, cust_rating is dependent only on lic_id and dr_rating is dependent only on the cus_id. So we will create two separate relations namely rating_customer and rating_driver.
   rating_cust(lic_id,cust_rating)
   rating_driver(cust_ID,dr_rating)
2. In the vehicle table, model and seats are dependent only on the lic_plate and not on the lic_id. So we will divide the table into two separate tables.
   vehicle_info(lic_plate,model,seats)
   vehicle_driver(lic_plate,lic_id)

The following is the relational schema after normalization.
Google Drive Link:
https://drive.google.com/file/d/1kv7zo8T9fUwOwdmAO6scnHCK9x7tle8p/view

Diagram on the next page.



USER
User_id | Fname | Lname | PNo | Password | Email_id
UBER CUSTOMER
Cust_ID | DOB | Trip_id
DRIVER
Lic_id | DOB | Salary
TRIP
Trip_id | Cust_ID | Lic_id | Source | Desitinatioı | Feedback | Distance | Price
SHIFT
Shift_ID | Start_time | End_time
VEHICLE_INFO
lic_plate | Model | Seats
VEHICLE_DRIVER
lic_plate | lic_id
PAYMENT
Payment_ID | Amount | Paypal | Giftcard | Card
ADVERTISEMENT
Adv_ID | Lic_id | Type | Adv_name
TRIP_CANCEL
Cust_ID | Trip_ID | Lic_id | Penalty
RATING_CUST
Lic_id | cust_rating
RATING_DRIVER
Cust_ID | dr_rating
CUST_ADDRESS
Cust_ID | Zipcode | Street | City | State | Country
CUST_Payment
Payment_ID | Cust_ID
TAKES_SHIFT
Lic_id | Shift_ID
GIVES_RATING
Cust_ID | Lic_id

CREATION OF TABLES –

```
/* CREATION OF TABLES FOR UBER SYSTEM */
/* USER TABLE */
CREATE TABLE USER (
    user_id integer,
    fname varchar(20) NOT NULL,
    lname varchar(20) NOT NULL,
    phone_no varchar(20),
    pass_word varchar(30) NOT NULL,
    email_id varchar(30) NOT NULL,
    primary key(user_id)
);
/* UBER CUSTOMER TABLE */
CREATE TABLE UBER_CUSTOMER (
    cust_ID integer,
    cust_DOB date NOT NULL,
    trip_id integer,
    primary key(cust_ID)
);
/* DRIVER TABLE */
CREATE TABLE DRIVER (
    lic_id integer,
    driver_DOB date NOT NULL,
    Salary integer NOT NULL,
    lic_expiry date NOT NULL,
    primary key(lic_id)
);
/* TRIP TABLE */
CREATE TABLE TRIP (
    trip_ID integer,
    cust_ID integer NOT NULL,
    lic_id integer NOT NULL,
    source varchar(50) NOT NULL,
    destination varchar(60) NOT NULL,
    feedback varchar(60),
    distance integer NOT NULL,
    trip_price integer NOT NULL,
    primary key(trip_ID)
);
/* SHIFT TABLE */
CREATE TABLE SHIFT (
    shift_ID integer,
    shift_start_time integer NOT NULL,
    shift_end_time integer NOT NULL,
    primary key(shift_ID)
);
```

```
/* VEHICLE_INFO TABLE */
CREATE TABLE VEHICLE_INFO (
    lic_plate varchar(30),
    model varchar(20) NOT NULL,
    seats integer NOT NULL,
    ins_expiry DATE,
    primary key(lic_plate)
);

/* VEHICLE_DRIVER TABLE */
CREATE TABLE VEHICLE_DRIVER (
    lic_plate varchar(30),
    lic_id integer NOT NULL,
    primary key(lic_plate)
);
/* PAYMENT TABLE */
CREATE TABLE PAYMENT (
    payment_ID integer,
    pay_amount integer NOT NULL,
    paypal boolean default FALSE,
    giftcard boolean default FALSE,
    card boolean default FALSE,
    primary key(payment_ID)
);
/* ADVERTISEMENT TABLE */
CREATE TABLE ADVERTISEMENT (
    adv_ID integer,
    lic_id integer NOT NULL,
    adv_type varchar(40) NOT NULL,
    adv_name varchar(40) NOT NULL,
    primary key(adv_ID)
);
/* TRIP_CANCELLATION TABLE */
CREATE TABLE TRIP_CANCELLATION (
    cust_ID integer,
    trip_ID integer,
    lic_id integer,
    penalty decimal(10,2)
);
/* RATING FOR THE CUSTOMER BY THE DRIVER TABLE */
CREATE TABLE RATING_CUST (
    lic_id integer,
    cust_rating decimal default 2.0
);
/* RATING FOR THE DRIVER BY THE CUSTOMER TABLE */
CREATE TABLE RATING_DRIVER (
    cust_ID integer,
    dr_rating decimal default 2.0
);
/* SAVED ADDRESSES OF THE CUSTOMER TABLE */
CREATE TABLE CUST_ADDRESS (
    cust_ID integer,
    zipcode varchar(10) NOT NULL,
    street varchar(70) NOT NULL,
    city varchar(30) NOT NULL,
    state varchar(30) NOT NULL,
    country varchar(20) NOT NULL
);
/* PAYMENT MADE BY CUSTOMER TABLE */
CREATE TABLE CUST_PAYMENT (
    payment_ID integer NOT NULL,
    cust_ID integer NOT NULL
);
/* SHIFT TAKEN UP BY THE DRIVER TABLE */
CREATE TABLE TAKES_SHIFT (
    lic_id integer NOT NULL,
    shift_ID integer NOT NULL
);
/* RATING TABLE */
CREATE TABLE GIVES_RATING (
    cust_ID integer NOT NULL,
    lic_id integer NOT NULL
);
```

Alter table commands

```
alter table
trip add constraint trip_driver_id_fk foreign key(lic_id)
references Driver(lic_id)
ON DELETE CASCADE;

alter table
Trip add constraint trip_cust_id_fk foreign key(cust_ID)
references Uber_Customer(cust_ID)
ON DELETE CASCADE;

alter table
Takes_Shift add constraint tshift_Lic_id_fk foreign key(lic_id)
references Driver(lic_id)
ON DELETE CASCADE;

alter table
Gives_Rating add constraint grate_cust_id_fk foreign key(cust_ID)
references Uber_Customer(cust_ID)
```

```
ON DELETE CASCADE;

alter table
Trip_Cancel add constraint tcancel_cust_id_fk foreign key(cust_ID)
references Uber_Customer(cust_ID)
ON DELETE CASCADE;

alter table
Trip_Cancel add constraint tcancel_trip_id_fk foreign key(trip_ID)
references Uber_Customer(trip_ID)
ON DELETE CASCADE;

alter table
Trip_Cancel add constraint tcancel_lic_id_fk foreign key(lic_ID)
references Uber_Customer(lic_ID)
ON DELETE CASCADE;

alter table
Cust_Address add constraint add_cust_id_fk foreign key(cust_ID)
references Uber_Customer(cust_ID)
ON DELETE CASCADE;

alter table
Rating_driver add constraint driverRate_cust_id_fk foreign key(cust_ID)
references Uber_Customer(cust_ID)
ON DELETE CASCADE;

alter table
Cust_Payment add constraint pay_cust_id_fk foreign key(cust_ID)
references Uber_Customer(cust_ID)
ON DELETE CASCADE;

alter table
Vehicle_Driver add constraint vdriver_lic_id_fk foreign key(lic_id)
references Driver(lic_id)
ON DELETE CASCADE;

alter table
Advertisement add constraint adv_lic_id_fk foreign key(lic_id)
references Driver(lic_id)
ON DELETE CASCADE;

alter table
Rating_Cust add constraint rc_lic_id_fk foreign key(lic_id)
references Driver(lic_id)
ON DELETE CASCADE;
```

```
alter table
Gives_Rating add constraint gr_lic_id_fk foreign key(lic_id)
references Driver(lic_id)
ON DELETE CASCADE;

alter table
Takes_Shift add constraint takes_shift_id_fk foreign key(shift_ID)
references Shift(shift_ID)
ON DELETE CASCADE;

alter table
Vehicle_Driver add constraint vd_Lic_fk foreign key(Lic_Plate)
references Vehicle_Info(Lic_Plate)
ON DELETE CASCADE;
```

PROCEDURES AND TRIGGERS –

1. **Procedure to book a trip:**
   We enter the trip details here by entering data into trip_ID, cust_ID, source, destination,etc.

```
/* PROCEDURE TO BOOK A TRIP */
CREATE OR REPLACE PROCEDURE trip_booking {
    trip_ID IN integer,
    lic_id IN integer,
    cust_ID IN integer,
    source IN varchar,
    destination IN varchar,
    feedback IN varchar,
    distance IN integer,
    trip_price IN integer
} AS
BEGIN
    INSERT INTO TRIP VALUES (
        trip_ID,
        lic_id,
        cust_ID,
        source,
        destination,
        feedback,
        distance,
        trip_price
    );
END;
```

2. **Procedure to register a driver**
   Firstly, we enter data into the user table such as user_ID, email, password, fname, name, phone_no.
   Finally, we enter data into the driver table such as lic_id, salary, DOB,license expiry date.

```
/* PROCEDURE TO REGISTER A DRIVER */
CREATE OR REPLACE PROCEDURE driver_registration (
    user_ID IN integer,
    lic_id IN integer,
    driver_DOB IN date,
    salary IN integer,
    email_id IN varchar,
    pass_word IN varchar,
    fname IN varchar,
    lname IN varchar,
    phone_no IN varchar,
    lic_expiry IN date
)
BEGIN
    INSERT INTO user VALUES(
        user_ID,
        fname,
        lname,
        email_id,
        pass_word,
        phone_no
    );
    INSERT INTO driver VALUES(
        lic_id,
        driver_DOB,
        salary
    );
END;
```

3. **Procedure to check if the user exists**

   This procedure checks if the user exists on the basis of the email address. This allows us to keep the system free of duplicate values.

```
/* PROCEDURE TO CHECK IF THE USER ALREADY EXISTS */
CREATE OR REPLACE PROCEDURE user_exists (
    email_id IN varchar
) AS
DECLARE user_flag INT;
BEGIN
    IF EXISTS(SELECT 1 FROM USER WHERE email_id=email_id)
    BEGIN
        SET user_flag=1;
    END
    ELSE
    BEGIN
        SET user_flag=0;
    END
    IF (user_flag=1)
    BEGIN
        RAISERROR ("This user already exists",16,1);
        ROLLBACK;
    END
END;
```

4. **Procedure to save the address**

   This procedure is used to enter the address of the customer. Values such as street,city,state,zipcode,country, etc are inserted.

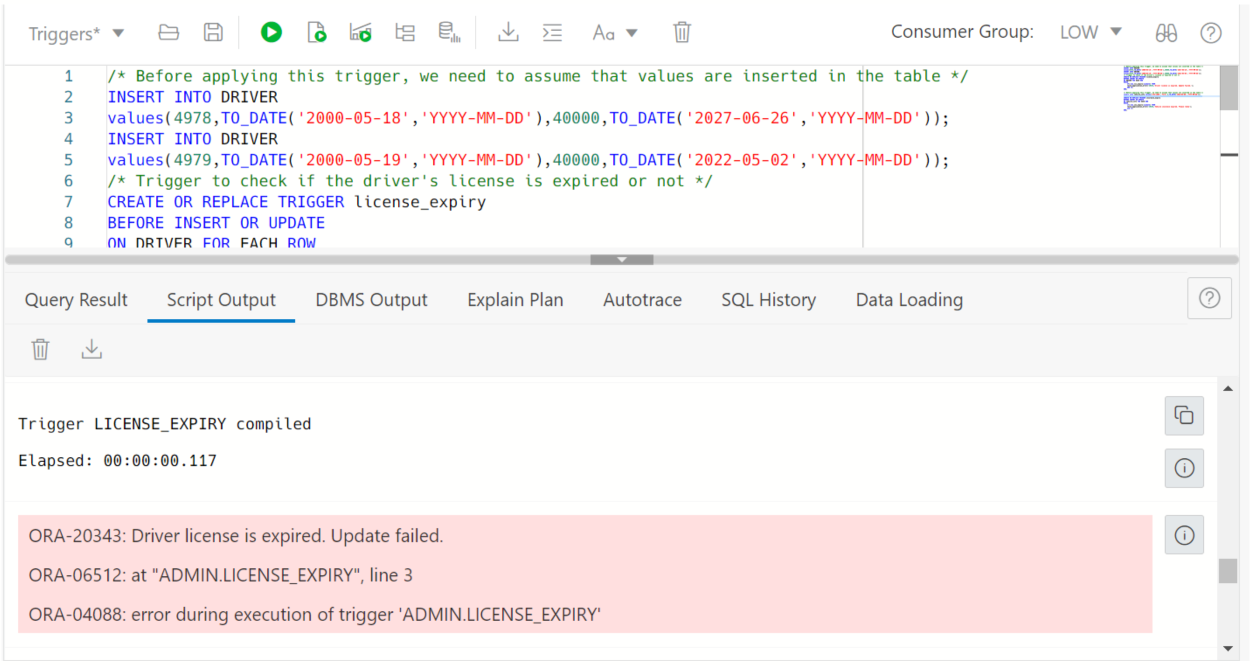
```
/* PROCEDURE TO SAVE ADDRESS */
CREATE OR REPLACE PROCEDURE save_address (
    cust_ID IN integer,
    zipcode IN varchar,
    street IN varchar,
    city IN varchar,
    state IN varchar,
    country IN varchar
) AS
BEGIN
    INSERT INTO cust_address (
        cust_ID,
        zipcode,
        street,
        city,
        state,
        country
    );
END;
```

5. **Trigger to check that the driver's license shouldn't have expired.**

   This trigger is used to validate the driver's license.

```
/* Before applying this trigger, we need to assume that values are inserted in
the table */
INSERT INTO DRIVER
values(4978,TO_DATE('2000-05-18','YYYY-MM-DD'),40000,TO_DATE('2027-06-26','YYYY
-MM-DD'));
INSERT INTO DRIVER
values(4979,TO_DATE('2000-05-19','YYYY-MM-DD'),40000,TO_DATE('2022-05-02','YYYY
-MM-DD'));
/* Trigger to check if the driver's license is expired or not */
CREATE OR REPLACE TRIGGER license_expiry
BEFORE INSERT OR UPDATE
ON DRIVER FOR EACH ROW
BEGIN
    IF(:new.lic_expiry<sysdate) THEN
         raise_application_error(-20343,'Driver  license  is  expired.  Update
failed.');
    END IF;
END;
```

The following is the output of the above trigger.



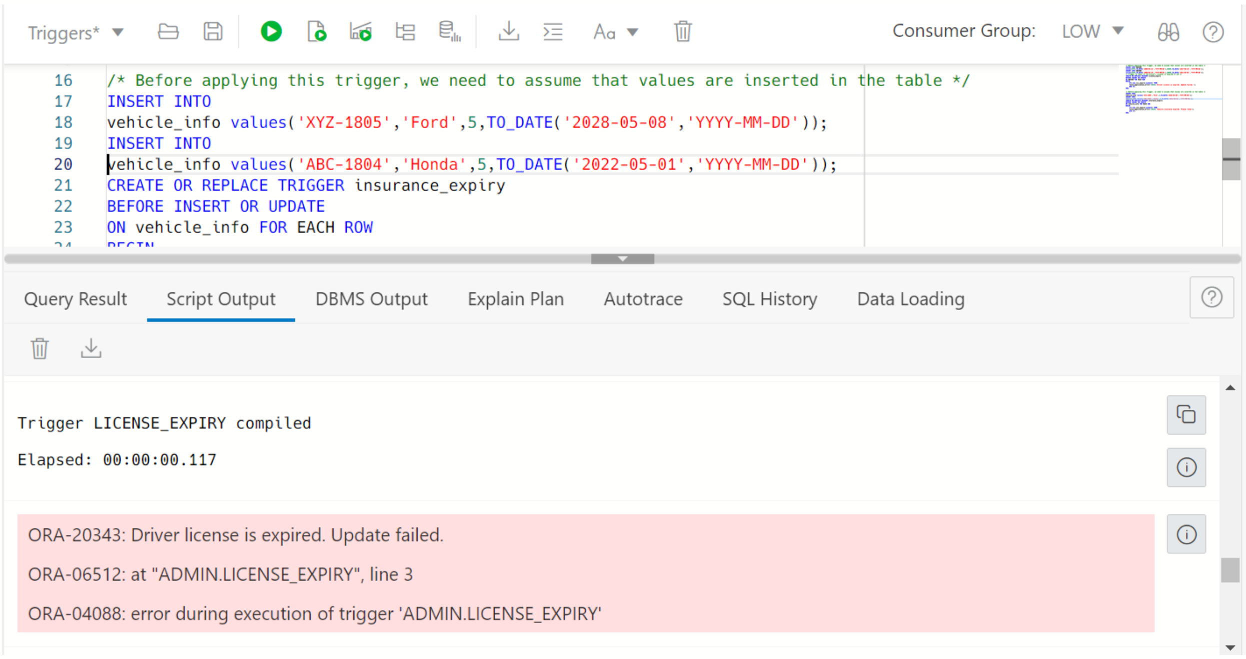
```
/* Before applying this trigger, we need to assume that values are inserted in the table */
INSERT INTO DRIVER
values(4978,TO_DATE('2000-05-18','YYYY-MM-DD'),40000,TO_DATE('2027-06-26','YYYY-MM-DD'));
INSERT INTO DRIVER
values(4979,TO_DATE('2000-05-19','YYYY-MM-DD'),40000,TO_DATE('2022-05-02','YYYY-MM-DD'));
/* Trigger to check if the driver's license is expired or not */
CREATE OR REPLACE TRIGGER license_expiry
BEFORE INSERT OR UPDATE
ON DRIVER FOR EACH ROW
```

```
Trigger LICENSE_EXPIRY compiled

Elapsed: 00:00:00.117

ORA-20343: Driver license is expired. Update failed.

ORA-06512: at "ADMIN.LICENSE_EXPIRY", line 3

ORA-04088: error during execution of trigger 'ADMIN.LICENSE_EXPIRY'
```

6. Trigger to check the validity of the insurance of the vehicle.

```
/* Before applying this trigger, we need to assume that values are inserted in
the table */
INSERT INTO
vehicle_info values('XYZ-1805','Ford',5,TO_DATE('2028-05-08','YYYY-MM-DD'));
INSERT INTO
vehicle_info values('ABC-1804','Honda',5,TO_DATE('2022-05-01','YYYY-MM-DD'));
CREATE OR REPLACE TRIGGER insurance_expiry
BEFORE INSERT OR UPDATE
ON vehicle_info FOR EACH ROW
BEGIN
    if(:new.ins_expiry<sysdate) THEN
    raise_application_error(-20346,'Vehicle insurance expired. Please renew');
    end if;
END;
```

The following is the output of the trigger.



```
/* Before applying this trigger, we need to assume that values are inserted in the table */
INSERT INTO
vehicle_info values('XYZ-1805','Ford',5,TO_DATE('2028-05-08','YYYY-MM-DD'));
INSERT INTO
vehicle_info values('ABC-1804','Honda',5,TO_DATE('2022-05-01','YYYY-MM-DD'));
CREATE OR REPLACE TRIGGER insurance_expiry
BEFORE INSERT OR UPDATE
ON vehicle_info FOR EACH ROW
```

```
Trigger LICENSE_EXPIRY compiled

Elapsed: 00:00:00.117
```

```
ORA-20343: Driver license is expired. Update failed.

ORA-06512: at "ADMIN.LICENSE_EXPIRY", line 3

ORA-04088: error during execution of trigger 'ADMIN.LICENSE_EXPIRY'
```