Taxi Service System

DATABASE DESIGN PROJECT

Priyank Shah (pks170030)
Dhwani Kaneria (drk170130)

CS6360.002

Requirements

A taxi service system is a network where the customers find an easier way to commute from one place to another. Here, we have provided with the database system which enables a smooth transition of this of taxi service model. This project is an attempt to bring together all the aspects which are included in the taxi management system. If you want to run a taxi business, you need to keep record for all the data of driver as well as customers. To ensure that all the records for this database are correct, we have created a database system which can help in tracking information of the cabs, their drivers and customers.

Following are the names of tables that we used in our schema along with brief description of each table.

- Person:
  This represents a human being in the model. It is further divided into three types. A person can be a customer, an agent or a driver. It has all the details of the person. The attributes are first name, last name, the person's SSN, etc. Generalization is used here.

- Customer:
  It is a type of a person. All the attributes of a person are included here. Other than these attributes, it has customer id.

- Driver:
  It is a type of a person. All the attributes of a person are included here. Other than these attributes, it has driver id, driver's license number, its rating.

- Agent:
  It is a type of a person. All the attributes of a person are included here. Other than these attributes, it has agent id and its start and end shift timings.

- Car:
  A driver drives a car. Car table has all the information related to car. Additionally, we have also shown that is car empty at any specific time. If it is empty, then only it will allow customers to book ride from that car.

- Search:
  A search table has a list of cars with their current locations. This will in finding cars which has their current location near to the customer's current location.

- Booking Ride:
  This has all the information about a specific booking. It includes information about the car, driver, customer, the source and the destination of the ride.

- Feedback:
  After the customer has completed the ride. It can give feedback to the driver via this table.

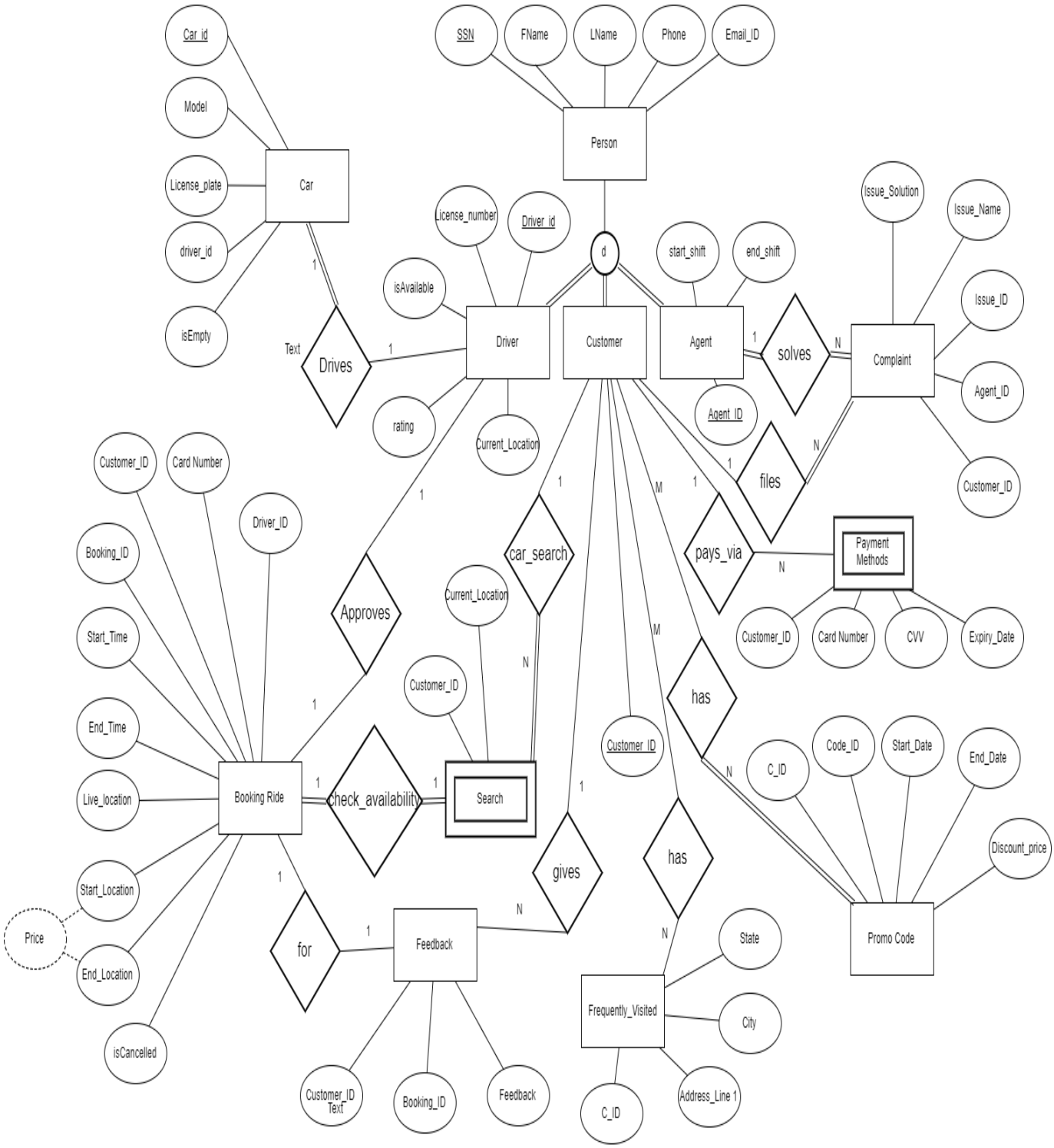
- Frequently visited:
  It has all the addresses of those places which a customer visits more frequently. It is easier to show this information to the customer as he doesn't have to add the addresses again and again.

- Promo Code:
  This has information about only those customers who have a promo code. Promo codes does not work for all the customers. Also, there is some expiry date set for each promo code. And amount of discount for that promo code.

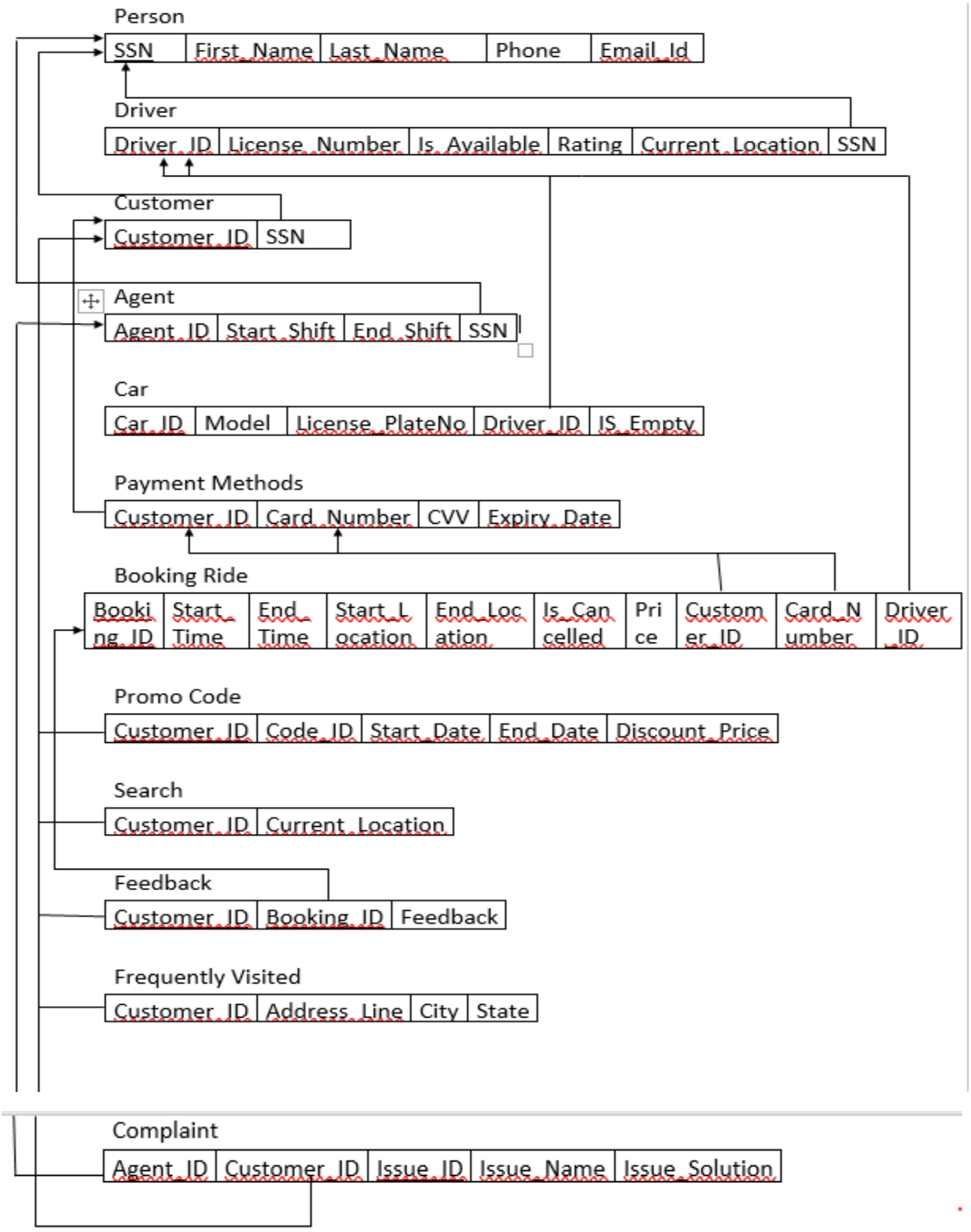
- Payment Methods:
  It includes information about a credit card or a debit card for a specific customer.

- Complaint:
  This table is necessary. If a customer finds any issue in taking a ride, they can file a complaint. Agents are there to help the customers in solving these complaints.

ER Diagram


Car_id
Model
License_plate
driver_id
isEmpty
Car
SSN
FName
LName
Phone
Email_ID
Person
License_number
Driver_id
d
isAvailable
start_shift
end_shift
Issue_Solution
Issue_Name
Issue_ID
Agent_ID
Customer_ID
Complaint
Text
Drives
Driver
Customer
Agent
solves
Agent_ID
rating
Current_Location
files
Customer_ID
Card Number
Driver_ID
Booking_ID
Approves
car_search
pays_via
Payment Methods
Customer_ID
Card Number
CVV
Expiry_Date
Start_Time
Current_Location
Customer_ID
has
End_Time
Customer_ID
C_ID
Code_ID
Start_Date
End_Date
Live_location
Booking Ride
check_availability
Search
gives
has
Discount_price
Start_Location
Price
for
Feedback
State
Promo Code
End_Location
Frequently_Visited
City
isCancelled
Customer_ID
Text
Booking_ID
Feedback
C_ID
Address_Line 1

Normalized Relational Model



Normalization Rules/ Functional Dependencies:

The tables that we have created and used are already in 3NF. All the dependencies are preserved here. We have already decomposed all the relationships into separate relations. Here, in all the relations the non-prime attributes depend only on the prime attributes i.e. attributes which are either primary key or candidate key.

SQL statements for database creation

```
CREATE TABLE agent (
agent_id varchar(25) NOT NULL,
end_shift TIMESTAMP,
start_shift TIMESTAMP,
ssn numeric(11) NOT NULL,
PRIMARY KEY(agent_id)
);

CREATE TABLE bookingride (
booking_id varchar(25) NOT NULL,
customer_id varchar(25),
driver_id varchar(25) ,
card_no numeric(11),
start_time timestamp ,
end_time timestamp ,
live_location VARCHAR(45) ,
start_location VARCHAR(45) ,
end_location VARCHAR(45) ,
is_cancelled char  check (  is_cancelled in (0,1)) ,
PRIMARY KEY (booking_id)
);
```

```
CREATE TABLE car (
car_id varchar(25) NOT NULL,
model VARCHAR(45) ,
licence_plate VARCHAR(45) ,
driver_id varchar(25),
is_empty number(1,0),
is_empty char  check (  is_empty in (0,1)) ,
PRIMARY KEY (car_id)
);

CREATE TABLE complaint (
issue_id varchar(25)  NOT NULL,
agent_id varchar(25),
customer_id varchar(25) ,
issue_name VARCHAR(45) ,
issue_solution VARCHAR(45) ,
PRIMARY KEY (issue_id)
);

CREATE TABLE customer (
customer_id varchar(25) NOT NULL,
ssn numeric(11),
PRIMARY KEY (customer_id)
);
```

```
CREATE TABLE driver (
driver_id varchar(25) NOT NULL,
licence_no numeric(11) ,
isAvailable char  check (  isAvailable in (0,1)) ,
rating numeric(11) ,
currentLocation VARCHAR(45),
ssn numeric(11),
PRIMARY KEY (driver_id)
);

CREATE TABLE feedback (
customer_id varchar(25) NOT NULL,
booking_id varchar(25),
feedback VARCHAR(45) ,
PRIMARY KEY (customer_id)
);

CREATE TABLE frequently_visited (
customer_id varchar(25) NOT NULL,
address VARCHAR(45) ,
city VARCHAR(45) ,
state VARCHAR(45) ,
PRIMARY KEY (customer_id)
);
```

```
CREATE TABLE payment_method (
customer_id varchar(25),
card_no numeric(11) ,
cvv numeric(11) ,
expiry_date timestamp
);

CREATE TABLE person (
ssn numeric(11) NOT NULL,
fName VARCHAR(45) ,
lName VARCHAR(45) ,
phone numeric(11) ,
email_id VARCHAR(45) ,
PRIMARY KEY (ssn)
);

CREATE TABLE promo_code (
customer_id varchar(25) NOT NULL,
code_no numeric(11) ,
start_date timestamp,
end_date timestamp,
discount_price numeric(11) ,
PRIMARY KEY (customer_id)
);

CREATE TABLE search (
customer_id varchar(25) NOT NULL,
current_location VARCHAR(45)
);
```

```
CREATE TABLE distance (
start_location VARCHAR(45) ,
end_location VARCHAR(45) ,
Miles number(11)
);
```

Inserting values:

Agent:

```
INSERT INTO AGENT (AGENT_ID, END_SHIFT, START_SHIFT, SSN) VALUES ('A1',to_timestamp('8:00', 'HH.MI.SSXFF AM'),to_timestamp('5:00', 'HH.MI.SSXFF PM'),1.234567895E9);

INSERT INTO AGENT (AGENT_ID, END_SHIFT, START_SHIFT, SSN) VALUES ('A2',to_timestamp('10:00', 'HH.MI.SSXFF AM'),to_timestamp('7:00', 'HH.MI.SSXFF PM'),1.234567895E9);

INSERT INTO AGENT (AGENT_ID, END_SHIFT, START_SHIFT, SSN) VALUES ('A3',to_timestamp('9:00', 'HH.MI.SSXFF AM'),to_timestamp('6:00', 'HH.MI.SSXFF AM'),1.234567895E9);

INSERT INTO AGENT (AGENT_ID, END_SHIFT, START_SHIFT, SSN) VALUES ('A4',to_timestamp('12:00', 'HH.MI.SSXFF PM'),to_timestamp('9:00', 'HH.MI.SSXFF PM'),1.234567895E9);

INSERT INTO AGENT (AGENT_ID, END_SHIFT, START_SHIFT, SSN) VALUES ('A5',to_timestamp('9:00', 'HH.MI.SSXFF PM'),to_timestamp('12:00', 'HH.MI.SSXFF PM'),1.234567895E9);

INSERT INTO AGENT (AGENT_ID, END_SHIFT, START_SHIFT, SSN) VALUES ('A6',to_timestamp('9:00', 'HH.MI.SSXFF PM'),to_timestamp('12:00', 'HH.MI.SSXFF PM'),1.234567895E9);
```

Booking:

```
INSERT INTO bookingride (booking_id, customer_id, driver_id, card_no,start_time,end_time,live_location,start_location,end_location,is_cancelled) VALUES ('B1','C4','D7','369258147',to_timestamp('04-12-17 8:00', 'YYYY-MM-DD HH24:MI:SS'),to_timestamp('04-12-17 8:30', 'YYYY-MM-DD HH24:MI:SS'),'Frankford Apt','Frankford Apt','DFW Union tower',0);

INSERT INTO bookingride (booking_id, customer_id, driver_id, card_no,start_time,end_time,live_location,start_location,end_location,is_cancelled) VALUES ('B2','C3','D2','147258369',to_timestamp('03-12-17 4:00', 'YYYY-MM-DD HH:MI:SS PM'),to_timestamp('03-12-17 4:30', 'YYYY-MM-DD HH:MI:SS'),'Macullum','Macullum','Frankford Apt',0);
```

```
INSERT INTO bookingride (booking_id, customer_id, driver_id,
card_no,start_time,end_time,live_location,start_location,end_location,is_cancelled) VALUES
('B3','C5','D4','258147369',to_timestamp('01-12-17 7:00', 'YYYY-MM-DD HH:MI:SS
PM'),to_timestamp('01-12-17 7:15', 'YYYY-MM-DD HH:MI:SS PM'),'Chatham Courts','Frankford
Apt','McCallum Blvd',0);

INSERT INTO bookingride (booking_id, customer_id, driver_id,
card_no,start_time,end_time,live_location,start_location,end_location,is_cancelled) VALUES
('B4','C1','D5','123456789',to_timestamp('27-11-17 6:30', 'YYYY-MM-DD HH:MI:SS
AM'),to_timestamp('27-11-17 6:30', 'YYYY-MM-DD HH:MI:SS AM'),'Marquis','Marquis','McCallum
Blvd',0);

INSERT INTO bookingride (booking_id, customer_id, driver_id,
card_no,start_time,end_time,live_location,start_location,end_location,is_cancelled) VALUES
('B5','C1','D8','894231842',to_timestamp('26-11-17 10:40', 'YYYY-MM-DD HH:MI:SS
PM'),to_timestamp('26-11-17 10:40', 'YYYY-MM-DD HH:MI:SS PM'),'Chatham Courts','Frankford
Apt','Chatham Courts',0);

INSERT INTO bookingride (booking_id, customer_id, driver_id,
card_no,start_time,end_time,live_location,start_location,end_location,is_cancelled) VALUES
('B6','C7','D9','564612375',to_timestamp('03-11-17 7:15', 'YYYY-MM-DD HH:MI:SS
AM'),to_timestamp('03-11-17 7:15', 'YYYY-MM-DD HH:MI:SS AM'),'McCallum Blvd','McCallum
Blvd','DFW Union tower',0);

INSERT INTO bookingride (booking_id, customer_id, driver_id,
card_no,start_time,end_time,live_location,start_location,end_location,is_cancelled) VALUES
('B7','C5','D4','258147369',to_timestamp('28-11-17 2:00', 'YYYY-MM-DD HH:MI:SS
PM'),to_timestamp('28-11-17 2:00', 'YYYY-MM-DD HH:MI:SS PM'),'Richardson','Frankford
Apt','Marquis',0);

INSERT INTO bookingride (booking_id, customer_id, driver_id,
card_no,start_time,end_time,live_location,start_location,end_location,is_cancelled) VALUES
('B8','C6','D2','654988412',to_timestamp('25-11-17 6:10', 'YYYY-MM-DD HH:MI:SS
AM'),to_timestamp('25-11-17 6:10', 'YYYY-MM-DD HH:MI:SS AM'),'Marquis','Marquis','McCallum
Blvd',0);
```

Updating Values:

```
alter table driver
add constraint FK1
foreign key (ssn)
references person (ssn)
on delete cascade;
```

```
alter table customer
add constraint FK2
foreign key (ssn)
references person (ssn)
on delete cascade;

alter table agent
add constraint FK3
foreign key (ssn)
references person (ssn)
on delete cascade;

alter table car
add constraint FK4
foreign key (driver_id)
references driver (driver_id)
on delete cascade;

alter table bookingride
add constraint FK5
foreign key (driver_id)
references driver (driver_id)
on delete cascade;

alter table payment_method
add constraint FK6
foreign key (customer_id)
references customer (customer_id)
on delete cascade;

alter table promo_code
add constraint FK7
foreign key (customer_id)
references customer (customer_id)
on delete cascade;

alter table search
add constraint FK8
foreign key (customer_id)
references customer (customer_id)
on delete cascade;

alter table feedback
add constraint FK9
foreign key (customer_id)
references customer (customer_id)
on delete cascade;
```

```
alter table frequently_visited
add constraint FK10
foreign key (customer_id)
references customer (customer_id)
on delete cascade;

alter table complaint
add constraint FK11
foreign key (customer_id)
references customer (customer_id)
on delete cascade;

alter table complaint
add constraint FK12
foreign key (agent_id)
references agent (agent_id)
on delete cascade;

alter table bookingride
add constraint FK13
foreign key (customer_id,card_no)
references payment_method (customer_id,card_no)
on delete cascade;

alter table feedback
add constraint FK14
foreign key (booking_id)
references bookingride (booking_id)
on delete cascade;
```

PL/ SQL:

Stored Procedures:

1) **Stored procedure to check if the end time of the ride is equal to current time. Updating the value to know whether the car is available or not.**

```
SET serveroutput ON


DECLARE
   thisCar car%ROWTYPE;
   CURSOR AvailableCar IS
   select c.CAR_ID,c.MODEL,c.LICENCE_PLATE,c.DRIVER_ID,c.IS_EMPTY from driver d, car c where
c.DRIVER_ID=d.DRIVER_ID and d.DRIVER_ID in (select DRIVER_ID
   from bookingride where to_char(END_TIME,'HH.MI.SSXFF AM')=to_char(sysdate,'HH.MI.SSXFF
AM'))
   FOR UPDATE;
BEGIN
   OPEN AvailableCar;
   LOOP
      FETCH AvailableCar INTO thisCar;
      EXIT WHEN (AvailableCar%NOTFOUND);
         IF (thisCar.is_empty = 0) THEN
         UPDATE car SET thisCar.is_empty = 1
         WHERE CURRENT OF AvailableCar;
          DBMS_OUTPUT.PUT_LINE('Now, Car is Available');
         END IF;
   END LOOP;
   CLOSE AvailableCar;
END;
```

2) Stored procedure to show the final prices for those customers who have the promo codes.

```
DECLARE
   bookingid bookingride.booking_id%TYPE;
   DISCOUNT_PRICE promo_code.DISCOUNT_PRICE%TYPE;
   Rate distance.MILES%TYPE;
   CURSOR getPrice IS
  select d.MILES/2,p.DISCOUNT_PRICE as Price , b.BOOKING_ID from bookingride b, distance
d,promo_code p where b.CUSTOMER_ID=p.CUSTOMER_ID and
to_char(b.START_TIME,'HH')>to_char(p.START_DATE,'HH') and
to_char(b.START_TIME,'HH')<to_char(p.END_DATE,'HH') and
d.START_LOCATION=b.START_LOCATION and d.END_LOCATION=b.END_LOCATION;

BEGIN
   OPEN getPrice;
   LOOP
      FETCH getPrice INTO Rate,DISCOUNT_PRICE,bookingid;
      EXIT WHEN (getPrice%NOTFOUND);
         IF (DISCOUNT_PRICE!= 0) THEN
          DBMS_OUTPUT.PUT_LINE('Discount applied  : ' || DISCOUNT_PRICE);
            IF((Rate-DISCOUNT_PRICE)<0) THEN
               DBMS_OUTPUT.PUT_LINE('Final Fare : 0');
            ELSE
               DBMS_OUTPUT.PUT_LINE('Final Fare : ' || (Rate-DISCOUNT_PRICE));
            END IF;

         END IF;
   END LOOP;
   CLOSE getPrice;
END;
```

Triggers:

1) Trigger to update the values of booking if the customer cancels the ride.

```
create or replace TRIGGER resetBooking
  BEFORE
    UPDATE
  ON bookingride
  FOR EACH ROW
BEGIN
        IF(:NEW.is_cancelled=1) THEN
        :NEW.START_TIME:=null;
        :NEW.End_TIME:=null;
        :NEW.LIVE_LOCATION:=null;
        :NEW.END_LOCATION:=null;
        :NEW.START_LOCATION:=null;
        DBMS_OUTPUT.PUT_LINE('Ride cancelled');
        END IF;
END;
```

2) Trigger to update the availability of car by checking whether the driver has reached the end location.

```
create or replace TRIGGER driverAvailable
 BEFORE

  UPDATE
 ON bookingride

 FOR EACH ROW

DECLARE
 driverloc  varchar(40);
BEGIN

        IF(:NEW.LIVE_LOCATION=:OLD.END_LOCATION) THEN
        update  driver set isavailable=1 where driver_id=:OLD.driver_id;

        DBMS_OUTPUT.PUT_LINE('Driver is available');
        END IF;
END;
```