

# Database Design

## Final Project Report



**Course Number - CS6360.001**

**Team Number - 13**

**Project Title - Uber-3**

### **Team Members**

Soumya Mukhija (sxm200134)

Praticha Suhas Damale (psd190006)

Adithya Sundararajan Iyer (asi200000)

# Table of Contents

<b>Table of Contents</b>	<b>2</b>
<b>Introduction</b>	<b>3</b>
How Uber Works	3
Data Requirements and Interaction of Entities	4
<b>(Enhanced) Entity-Relationship Diagram</b>	<b>7</b>
<b>Relational Schema and Normalization</b>	<b>8</b>
Mapping EER Diagram into Relational Schema	8
Informal Design Guidelines for Relational Databases	9
Functional Dependencies and Successive Normalization	9
3NF Normalized Relational Schema	13
<b>Table Creation Queries (With Screenshots)</b>	<b>15</b>
<b>PL/SQL Procedures &amp; Triggers</b>	<b>29</b>
Stored Procedures	29
Triggers	30

# Introduction

## **How Uber Works**

Uber is a mobility-as-a-service provider, which means that they deliver the service of mobility to their customer base. Uber is responsible for helping people travel from a source to their choice of destination, providing a multitude of options for their comfort and convenience.

The system consists of a car, a driver, and a customer. The driver is assumed to always own the car that they drive. A customer makes a trip request, choosing their pickup location, drop location, and the type of car that they wish to travel in (*Premium*, which includes Black and Black SUV, or *Economy*, which includes UberX, UberXL, Comfort, and Uber Pet). A randomly selected driver near the pickup location can choose to either accept or decline that request. If the request is declined, it is forwarded to another randomly chosen driver in that area, and the process repeats itself.

Once the request is accepted by a driver, they can view the route to the pickup location of the customer. Once they reach the pickup point, the customer has 5 minutes to arrive, after which the ride is auto-canceled. After picking up the customer, the driver can view the route to the drop location on their app interface.

When the customer is dropped off, they see an option to rate the driver on their application and add a tip. Similarly, the driver gets an option to rate the customer as well. Both these ratings are used to calculate the average total rating of the respective customer and the driver.

Once the trip is complete, the customer can choose to pay with their card, cash, Paypal, or Uber money (which is virtual cash that is pre-added by the customer using their card). One can choose to save multiple cards in their account for easy payment. All completed trips must be paid for.

Uber also utilizes the concept of Uber Points. Each time a customer makes a trip, they earn some points based on the car type chosen and the fare paid (The number of points is twice the fare for *Economy* cars, and thrice the fare for *Premiums*). The aggregate of these points is used to determine the type of a customer, which can be among Blue, Gold, Diamond, and Platinum. The better the type of a customer, the more perks they receive while traveling with Uber.

## Data Requirements and Interaction of Entities

Using the above data, we have extracted the main components of the system and used that information to build an initial Entity-Relationship (ER) diagram.

1. **Person:** A Person can be a Driver, Customer, or both. All Persons on Uber provide their email IDs, passwords, full names, and uniquely identifying contact numbers. They can choose to provide an online display picture link for easy recognition by others, their date of birth, and sex.

\*Note that even though overlaps between a Driver and a Customer are allowed, they both view different interfaces on the application.

- a. **Customer:** This is a subset of the Person entity. Each Person on the Customer interface has a few attributes that distinguish them from the Driver interface, including a customer rating, a unique referral link, the customer type, wallet balance (the amount of Uber money added and not yet used), and their Uber points.

Customers are allowed to make and cancel trip requests. They can also have a list of saved cards or saved addresses for faster access to the Uber services. Every time a customer makes a trip, the database system automatically adds the trip and payment information, all of which can be viewed by the customer.

- b. **Driver:** Similar to above, each Driver has a few attributes distinguishing them from the Customer subset. These attributes include their SSN, address, information about their driver's license, and Driver's Rating.

Each driver is assumed to own at least 1 car. Drivers are employees in Uber, so they have also linked salary accounts in the application where their pay is dispensed.

2. **Cars:** A car has several properties including a uniquely identifying Vehicle Identification Number (VIN), model, brand, color, a unique license plate number, insurance information, and capacity. It would also have a type, which a user has the flexibility to choose while requesting a ride. Premium cars are more expensive and worth more Uber points than Comfort.
3. **Trip Requests:** A customer makes a trip request, choosing from options such as car type, pickup location, and drop-off location. Based on these choices, they are shown an estimated rate for the ride. After booking, the customer can cancel the ride up to 5 minutes before the driver arrives. Upon cancelation, customers have to provide a reason from a list of options.

On the other hand, a driver can either accept or decline a trip request made by the customer. If they decline it, the request is forwarded to another randomly chosen driver in that area meeting the customer's preferences of car type. A trip can be either complete or canceled, but not both.

**a. Canceled Trips:**

- i. This is a disjoint subset of Trip Requests.
- ii. Trips canceled by the customers only consist of a cancellation reason, a penalty, and a timestamp in the database.

**b. Completed Trips:**

- i. A trip that is completed implies that a customer was provided service by the driver.
- ii. They were picked up and dropped off at their choice of location in their choice of car type.
- iii. Because of this, the Completed Trips entity contains a larger amount of information, including tips provided, timestamps of different milestones during the trip, ratings provided by both parties, the ride charge, and a total fare (including taxes) and discount applied.
- iv. Completed trips also have payment details and car details added to them.

4. **Payment History:** Payment is added every time a trip is completed, and then each payment entry is stored in the database. The attributes of this entity include an identifying transaction ID, the total amount paid including taxes and tips, payment mode (Cash/Card/Paypal/Uber Money) a Paypal ID (in the case of Paypal payment). The user also has an option to make payment from a saved card.

5. **Saved Cards:** These are cards that the user has saved in the database for easy access while making payments. These consist of a card nickname given by the user, type of card (Debit/Credit), card network (Visa/Mastercard etc.), full name on the card, card number, and expiration date. Cards are uniquely identified by the card numbers.

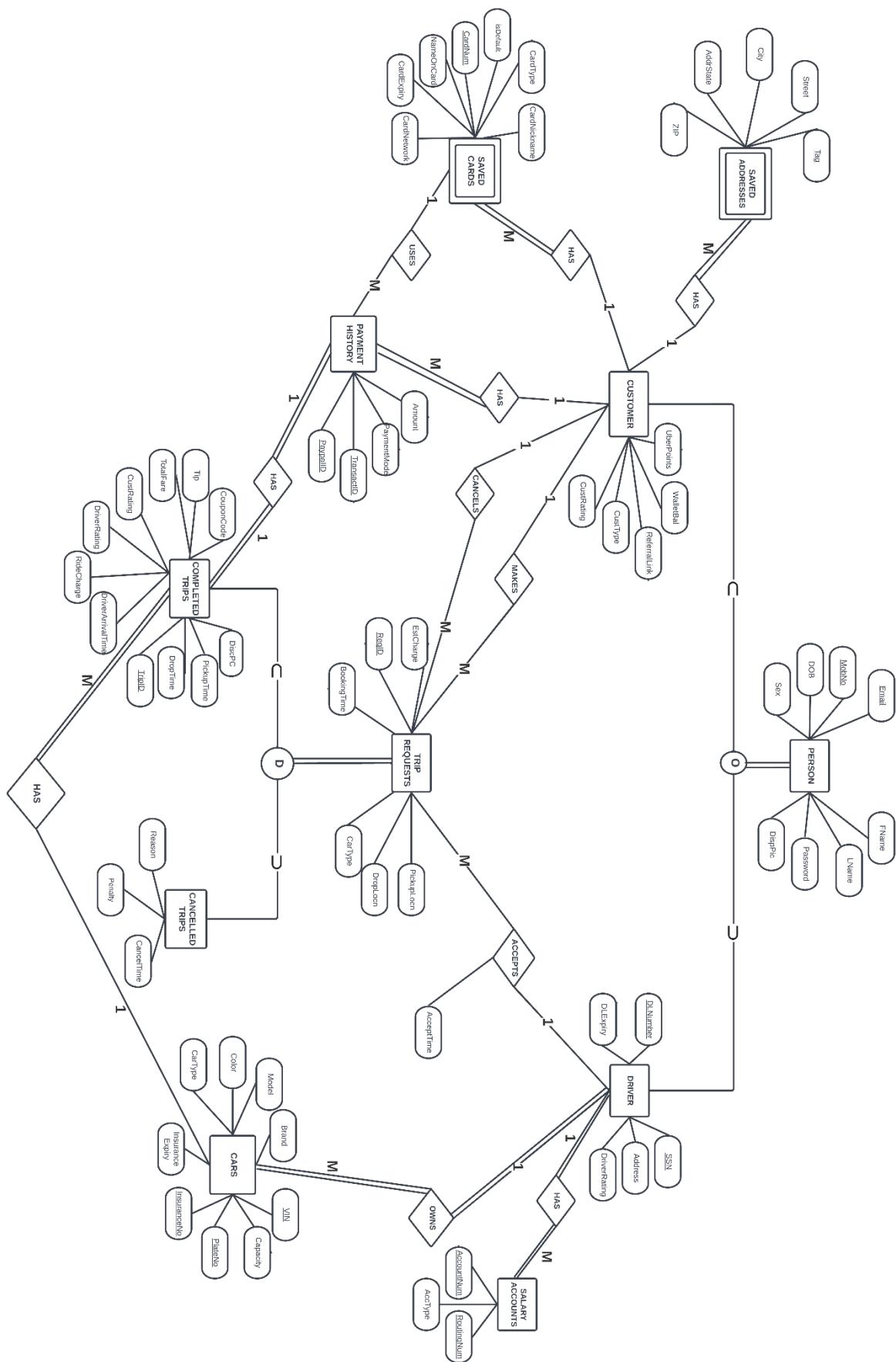
Saved cards are used by the customer while making a payment, so they are linked with both the Customer and the Payment History entities. A saved card would not exist without a customer, so it is a weak entity of Customer in the system.

6. **Saved Addresses:** These are a list of addresses that a customer might go frequently to. They appear on the top of the application as soon as a person enters the Customer interface. Each of these addresses is assigned a unique tag by the user (such as '*Home*'), and they have a street, city, state, and zip code.

A saved address would not exist without a customer, so it is a weak entity of Customer in the system.

7. **Salary Accounts:** Drivers are employees of Uber, so they each have a salary account linked with their profile that they receive their payments in. This entity includes a unique account ID, an account number, a routing number and the account type (savings/checking). A driver can have multiple salary accounts.

# (Enhanced) Entity-Relationship Diagram



# Relational Schema and Normalization

This section consists of 4 parts:

1. Mapping EER Diagram into Relational Schema
2. Informal Design Guidelines for Relational Databases
3. Functional Dependencies and Successive Normalization
4. 3NF Normalized Relational Schema

## **Mapping EER Diagram into Relational Schema**

\*Note - The primary keys have been underlined, and the foreign keys have been *italicized*.

1. **PERSON** (MobNo, FName, LName, Email, Password, DOB, Sex, DispPic)
2. **DRIVER** (SSN, Address, DLNumber, DLExpiry, DriverRating, *MobNo*)  
Foreign-Key(*MobNo*) REFERENCES PERSON(*MobNo*)
3. **CARS** (VIN, Brand, Model, Color, PlateNo, InsuranceNo, InsuranceExpiry, Capacity, CarType, SSN)  
Foreign-Key(*SSN*) REFERENCES DRIVER(*SSN*)
4. **SALARY\_ACCOUNTS** (AccountNum, RoutingNum, AccType, SSN)  
Foreign-Key(*SSN*) REFERENCES DRIVER(*SSN*)
5. **CUSTOMER** (MobNo, CustType, UberPoints, WalletBal, ReferralLink, CustRating)  
Foreign-Key(*MobNo*) REFERENCES PERSON(*MobNo*)
6. **SAVED\_ADDRESSES** (MobNo, Tag, Street, City, AddrState, ZIP)  
Foreign-Key(*MobNo*) REFERENCES CUSTOMER(*MobNo*)
7. **SAVED\_CARDS** (CardNum, NameOnCard, CardExpiry, CardType, CardNetwork, CardNickname, isDefault, *MobNo*)  
Foreign-Key(*MobNo*) REFERENCES CUSTOMER(*MobNo*)
8. **TRIP\_REQUESTS** (ReqID, BookingTime, CarType, PickupLocn, DropLocn, EstCharge, *MobNo*)  
Foreign-Key(*MobNo*) REFERENCES CUSTOMER(*MobNo*)
9. **ACCEPTS** (RequestID, SSN, AcceptTime)  
Foreign-Key(*RequestID*) REFERENCES TRIP\_REQUESTS(*ReqID*)  
Foreign-Key(*SSN*) REFERENCES DRIVER(*SSN*)

10. **CANCELLED\_TRIPS** (RequestID, CancelTime, Reason, Penalty)  
 Foreign-Key(RequestID) REFERENCES TRIP\_REQUESTS(ReqID)
11. **COMPLETE\_TRIPS** (TripID, DriverArrivalTime, PickupTime, DropTime, CouponCode, DiscPC, RideCharge, TotalFare, Tip, CustRating, DriverRating, SSN, RequestID, VIN)  
 Foreign-Key(SSN) REFERENCES DRIVER(SSN)  
 Foreign-Key(RequestID) REFERENCES TRIP\_REQUESTS(ReqID)  
 Foreign-Key(VIN) REFERENCES CARS(VIN)
12. **PAYMENT\_HISTORY** (TransactID, Amount, PaymentMode, PaypalID, CardNum, TripID, MobNo)  
 Foreign-Key(CardNum) REFERENCES SAVED\_CARDS(CardNum)  
 Foreign-Key(TripID) REFERENCES COMPLETE\_TRIPS(TripID)  
 Foreign-Key(MobNo) REFERENCES CUSTOMER(MobNo)

## Informal Design Guidelines for Relational Databases

- Semantics of the attributes are easily explainable, and each tuple in a relationship would represent one entity/relationship instance.
- There is no wasteful or redundant storage of information.
- The insertion, updation, and deletion anomalies will be handled by normalizing the table to 3NF form.
- Relations have been designed to reduce the occurrence of NULL values.

## Functional Dependencies and Successive Normalization

All attributes would contain atomic values. None of the entities or relationships have multivalued, composite or nested attributes. Hence the designed schema is in 1NF.

### PERSON

PERSON {MobNo, FName, LName, Email, Password, DOB, Sex, DispPic}

**FD1:**  $\text{MobNo} \rightarrow \{\text{FName}, \text{LName}, \text{Email}, \text{Password}, \text{DOB}, \text{Sex}, \text{DispPic}\}$

There is no partial or transitive dependency. So we can say it is already in 3NF.

## DRIVER

DRIVER {SSN, Address, DLNumber, DLExpiry, DriverRating, MobNo}

**FD2:** SSN → Address, DLNumber, DriverRating, MobNo

**FD3:** DLNum → DLExpiry

There is no partial dependency. So we can say it is in 2NF.

Thus we see that transitive dependency exists. This violates 3NF. To eliminate this transitive dependency and normalize to 3NF, we create the following new tables:

DL\_INFO {DLNum, DLExpiry}

DRIVER {SSN, Address, DLNumber, DriverRating, MobNo}

## CARS

CARS {VIN, Brand, Model, Color, PlateNo, InsuranceNo, InsuranceExpiry, Capacity, CarType, SSN}

**FD4:** VIN → Brand, Model, Color, PlateNo, InsuranceNo, Capacity, CarType, SSN

**FD5:** InsuranceNo → InsuranceExpiry

There is no partial dependency. So we can say it is in 2NF.

Thus we see that transitive dependency exists. This violates 3NF. To eliminate this transitive dependency and normalize to 3NF, we create the following new tables:

INSURANCE {InsuranceNo, InsuranceExpiry}

CARS {VIN, Brand, Model, Color, PlateNo, InsuranceNo, Capacity, CarType, SSN}

## SALARY\_ACCOUNTS

SALARY\_ACCOUNTS {AccountNum, RoutingNum, AccType, SSN}

**FD6:** AccountNum, RoutingNum → AccType, SSN

There is no partial or transitive dependency. So we can say it is already in 3NF.

## CUSTOMER

CUSTOMER {MobNo, CustType, UberPoints, WalletBal, ReferralLink, CustRating}

**FD7:** MobNo → CustType, UberPoints, WalletBal, ReferralLink, CustRating

There is no partial or transitive dependency. So we can say it is already in 3NF.

## SAVED\_ADDRESSES

SAVED\_ADDRESSES {MobNo, Tag, Street, City, AddrState, ZIP}

**FD8:** MobNo, Tag → Street, City, AddrState, ZIP

There is no partial or transitive dependency. So we can say it is already in 3NF.

## SAVED\_CARDS

SAVED\_CARDS {CardNum, NameOnCard, CardExpiry, CardType, CardNetwork, CardNickname, isDefault, MobNo}

**FD9:** CardNum → NameOnCard, CardExpiry, CardType, CardNetwork, CardNickname, isDefault, MobNo

There is no partial or transitive dependency. So we can say it is already in 3NF.

## TRIP\_REQUESTS

SAVED\_CARDS {ReqID, BookingTime, CarType, PickupLocn, DropLocn, EstCharge, MobNo}

**FD10:** ReqID → BookingTime, CarType, PickupLocn, DropLocn, EstCharge, MobNo

There is no partial or transitive dependency. So we can say it is already in 3NF.

## ACCEPTS

ACCEPTS {RequestID, SSN, AcceptTime}

**FD11:** RequestID, SSN → AcceptTime

There is no partial or transitive dependency. So we can say it is already in 3NF.

## CANCELLED\_TRIPS

CANCELLED\_TRIPS {RequestID, CancelTime, Reason, Penalty}

**FD12:** RequestID → CancelTime, Reason, Penalty

There is no partial or transitive dependency. So we can say it is already in 3NF.

## COMPLETE\_TRIPS

COMPLETE\_TRIPS {TripID, DriverArrivalTime, PickupTime, DropTime, CouponCode, DiscPC, RideCharge, TotalFare, Tip, CustRating, DriverRating, SSN, RequestID, VIN}

**FD13:** TripID → DriverArrivalTime, PickupTime, DropTime, CouponCode, RideCharge, TotalFare, Tip, CustRating, DriverRating, SSN, RequestID, VIN

**FD14:** CouponCode → DiscPC

There is no partial dependency. So we can say it is in 2NF.

Thus we see that transitive dependency exists. This violates 3NF. To eliminate this transitive dependency and normalize to 3NF, we create the following new tables:

COUPONS {CouponCode, DiscPC}

COMPLETE\_TRIPS {TripID, DriverArrivalTime, PickupTime, DropTime, CouponCode, RideCharge, TotalFare, Tip, CustRating, DriverRating, SSN, RequestID, VIN}

## PAYMENT\_HISTORY

PAYMENT\_HISTORY {TransactID, Amount, PaymentMode, PaypalID, CardNum, TripID, MobNo}

**FD15:** TransactID → Amount, PaymentMode, PaypalID, CardNum, TripID, MobNo

There is no partial or transitive dependency. So we can say it is already in 3NF.

## 3NF Normalized Relational Schema

\*Note - The primary keys have been underlined, and the foreign keys have been *italicized*.

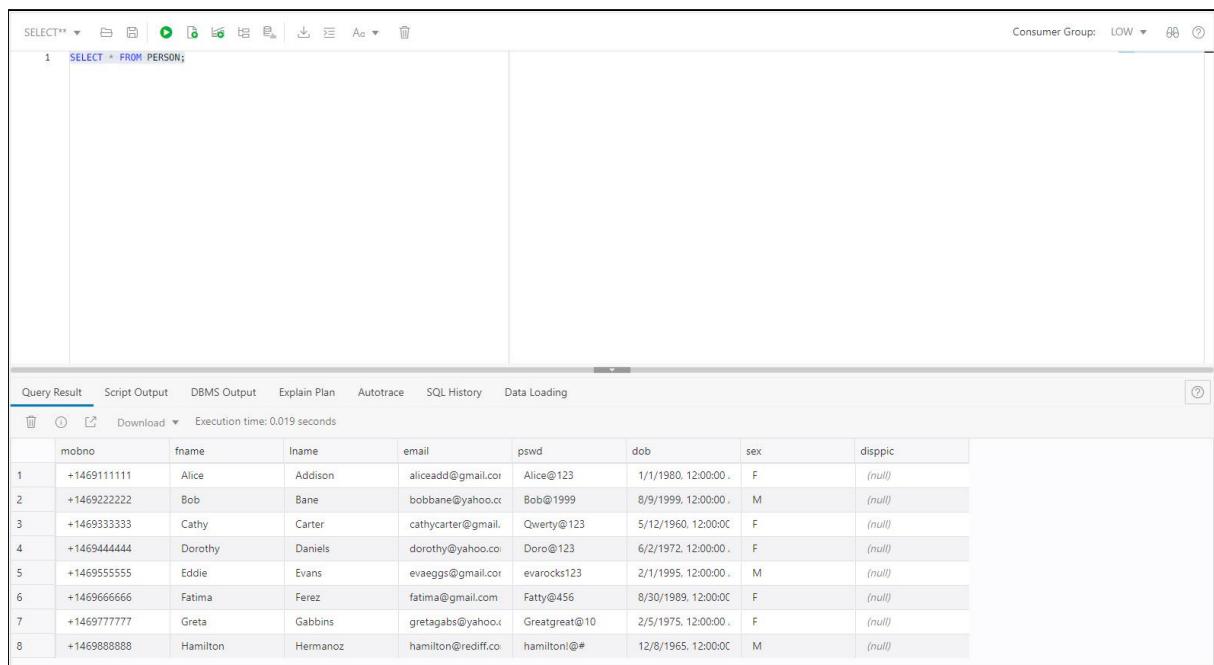
1. **PERSON** (MobNo, FName, LName, Email, Password, DOB, Sex, DispPic)
2. **DL\_INFO** {DLNum, DLExpiry}
3. **DRIVER** (SSN, Address, DriverRating, *DLNumber, MobNo*)  
Foreign-Key(MobNo) REFERENCES PERSON(MobNo)  
Foreign-Key(*DLNumber*) REFERENCES DL\_INFO(DLNum)
4. **INSURANCE** {InsuranceNo, InsuranceExpiry}
5. **CARS** (VIN, Brand, Model, Color, PlateNo, Capacity, CarType, *InsuranceNo, SSN*)  
Foreign-Key(SSN) REFERENCES DRIVER(SSN)  
Foreign-Key(InsuranceNo) REFERENCES INSURANCE(InsuranceNo)
6. **SALARY\_ACCOUNTS** (AccountNum, RoutingNum, AccType, SSN)  
Foreign-Key(SSN) REFERENCES DRIVER(SSN)
7. **CUSTOMER** (MobNo, CustType, UberPoints, WalletBal, ReferralLink, CustRating)  
Foreign-Key(MobNo) REFERENCES PERSON(MobNo)
8. **SAVED\_ADDRESSES** (MobNo, Tag, Street, City, AddrState, ZIP)  
Foreign-Key(MobNo) REFERENCES CUSTOMER(MobNo)
9. **SAVED\_CARDS** (CardNum, NameOnCard, CardExpiry, CardType, CardNetwork, CardNickname, isDefault, MobNo)  
Foreign-Key(MobNo) REFERENCES CUSTOMER(MobNo)
10. **TRIP\_REQUESTS** (ReqID, BookingTime, CarType, PickupLocn, DropLocn, EstCharge, MobNo)  
Foreign-Key(MobNo) REFERENCES CUSTOMER(MobNo)
11. **ACCEPTS** (RequestID, SSN, AcceptTime)  
Foreign-Key(RequestID) REFERENCES TRIP\_REQUESTS(ReqID)  
Foreign-Key(SSN) REFERENCES DRIVER(SSN)

12. **CANCELLED\_TRIPS** (RequestID, CancelTime, Reason, Penalty)  
Foreign-Key(RequestID) REFERENCES TRIP\_REQUESTS(ReqID)
13. **COUPONS** {CouponCode, DiscPC}
14. **COMPLETE\_TRIPS** (TripID, DriverArrivalTime, PickupTime, DropTime, RideCharge, TotalFare, Tip, CustRating, DriverRating, SSN, RequestID, VIN, CouponCode)  
Foreign-Key(SSN) REFERENCES DRIVER(SSN)  
Foreign-Key(RequestID) REFERENCES TRIP\_REQUESTS(ReqID)  
Foreign-Key(VIN) REFERENCES CARS(VIN)  
Foreign-Key(CouponCode) REFERENCES COUPONS(CouponCode)
15. **PAYMENT\_HISTORY** (TransactID, Amount, PaymentMode, PaypalID, CardNum, TripID, MobNo)  
Foreign-Key(CardNum) REFERENCES SAVED\_CARDS(CardNum)  
Foreign-Key(TripID) REFERENCES COMPLETE\_TRIPS(TripID)  
Foreign-Key(MobNo) REFERENCES CUSTOMER(MobNo)

# Table Creation Queries (With Screenshots)

## 1. PERSON

```
CREATE TABLE Person
(
    MobNo      varchar(13)          ,
    FName      varchar(50)         NOT NULL,
    LName      varchar(50)         NOT NULL,
    Email      varchar(50)         NOT NULL UNIQUE,
    Pswd       char(30)           NOT NULL,
    DOB        DATE               ,
    Sex        char(1)            ,
    DispPic    varchar(200)        ,
    PRIMARY KEY (MobNo)
);
```

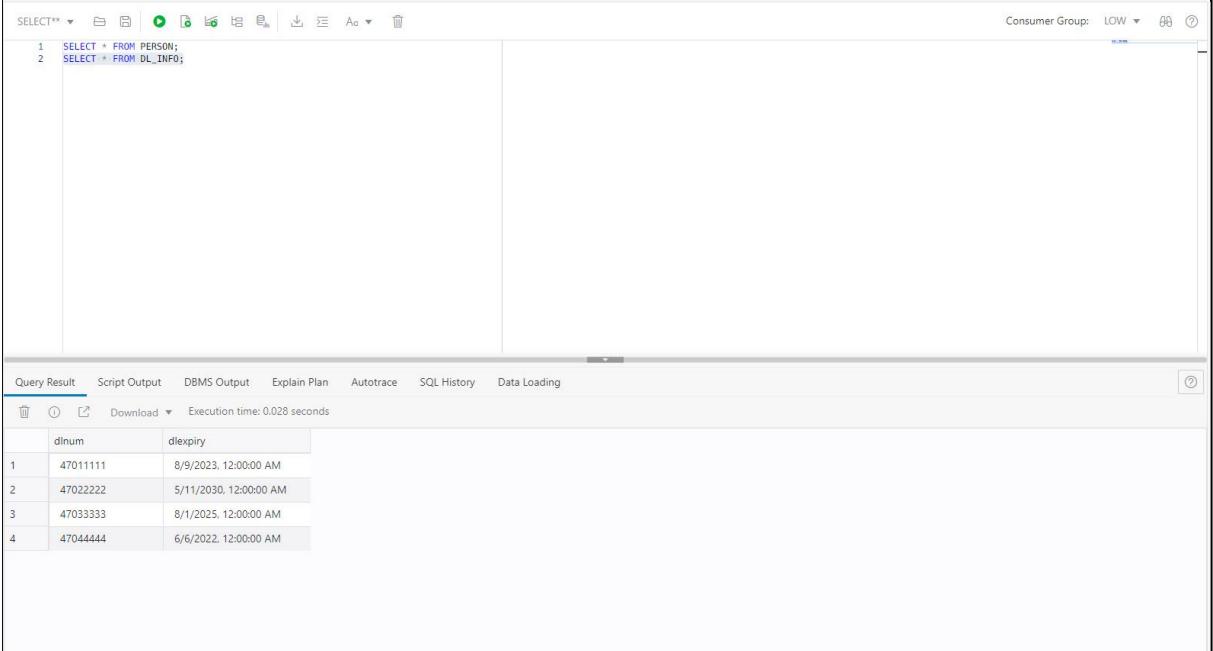


The screenshot shows the Oracle SQL Developer interface. The top panel displays the SQL command: `SELECT * FROM PERSON;`. The bottom panel shows the results of the query:

	mobno	fname	lname	email	pswd	dob	sex	disppic
1	+1469111111	Alice	Addison	aliceadd@gmail.com	Alice@123	1/1/1980, 12:00:00 .	F	(null)
2	+1469222222	Bob	Bane	bobbane@yahoo.cc	Bob@1999	8/9/1999, 12:00:00 .	M	(null)
3	+1469333333	Cathy	Carter	cathycarter@gmail.	Qwerty@123	5/12/1960, 12:00:0C	F	(null)
4	+1469444444	Dorothy	Daniels	dorothy@yahoo.co	Doro@123	6/2/1972, 12:00:00 .	F	(null)
5	+1469555555	Eddie	Evans	evaeggs@gmail.com	evarocks123	2/1/1995, 12:00:00 .	M	(null)
6	+1469666666	Fatima	Ferez	fatima@gmail.com	Fatty@456	8/30/1989, 12:00:0C	F	(null)
7	+1469777777	Greta	Gabbins	gretagabs@yahoo.c	Greatgreat@10	2/5/1975, 12:00:00 .	F	(null)
8	+1469888888	Hamilton	Hermanoz	hamilton@redif.co	hamilton@#	12/8/1965, 12:00:0C	M	(null)

## 2. DL\_INFO

```
CREATE TABLE DL_Info
(
    DLNum      char(8) ,
    DLExpiry    DATE        NOT NULL,
    PRIMARY KEY (DLNum)
);
```



The screenshot shows the Oracle SQL Developer interface. At the top, there are two tabs: 'Query Result' (which is selected) and 'Script Output'. Below the tabs, there is a toolbar with various icons. The main area contains two SQL statements:

```
1  SELECT * FROM PERSON;
2  SELECT * FROM DL_INFO;
```

The 'Query Result' tab is active, and the results for the second query are displayed in a table:

	dinum	dlexpiry
1	47011111	8/9/2023, 12:00:00 AM
2	47022222	5/11/2030, 12:00:00 AM
3	47033333	8/1/2025, 12:00:00 AM
4	47044444	6/6/2022, 12:00:00 AM

The execution time is listed as 0.028 seconds.

## 3. DRIVER

```
CREATE TABLE Driver
(
    SSN           char(11)          ,
    Addr         varchar(100)       NOT NULL,
    DriverRating   decimal(2,1)      ,
    DLNumber      char(8)        NOT NULL UNIQUE,
    MobNo        varchar(13)       NOT NULL,
    PRIMARY KEY (SSN),
    FOREIGN KEY (MobNo) REFERENCES Person(MobNo) ON DELETE CASCADE,
    FOREIGN KEY (DLNumber) REFERENCES DL_INFO(DLNum) ON DELETE CASCADE
);
```

```

SELECT * FROM PERSON;
SELECT * FROM DL_INFO;
SELECT * FROM DRIVER;

```

	ssn	addr	driverrating	dlnumber	mobno
1	111-11-1111	7488, Preston Rd	5	47011111	+1469222222
2	111-22-2222	1211, Campbell Dr	4.7	47022222	+1469444444
3	123-33-0091	56, Frankford Rd	5	47033333	+1469555555
4	233-90-1122	4599, Crest Blvd	4.9	47044444	+1469888888

## 4. INSURANCE

```

CREATE TABLE INSURANCE
(
    InsuranceNo      varchar(15)      ,
    InsuranceExpiry DATE            NOT NULL,
    PRIMARY KEY(InsuranceNo)
);

```

```

SELECT * FROM PERSON;
SELECT * FROM DL_INFO;
SELECT * FROM DRIVER;
SELECT * FROM INSURANCE;

```

	insuranceno	insuranceexpiry
1	12311111	6/9/2023, 12:00:00 AM
2	12311112	4/5/2023, 12:00:00 AM
3	123222222	4/12/2024, 12:00:00 AM
4	4561231234	2/3/2025, 12:00:00 AM
5	98711231	6/1/2024, 12:00:00 AM
6	99123123	3/9/2026, 12:00:00 AM

## 5. CARS

```
CREATE TABLE Cars
(
    VIN           char(17)          ,
    Brand         varchar(50)        NOT NULL,
    Model         varchar(50)        NOT NULL,
    Color         varchar(20)        NOT NULL,
    PlateNo       varchar(20)        NOT NULL UNIQUE,
    Capacity       int              DEFAULT 3   NOT NULL ,
    Cartype       varchar(20)        NOT NULL,
    InsuranceNo  varchar(15)        NOT NULL UNIQUE,
    SSN           char(11)          NOT NULL ,
    PRIMARY KEY(VIN),
    FOREIGN KEY (SSN) REFERENCES Driver(SSN) ON DELETE CASCADE,
    FOREIGN KEY (InsuranceNo) REFERENCES INSURANCE(InsuranceNo) ON DELETE CASCADE
);
```

The screenshot shows the Oracle SQL Developer interface. In the top-left pane, there is a script editor with the following code:

```
1 SELECT * FROM PERSON;
2 SELECT * FROM DL_INFO;
3 SELECT * FROM DRIVER;
4 SELECT * FROM INSURANCE;
5 SELECT * FROM CARS;
```

In the bottom-right pane, the 'Query Result' tab is selected, displaying the following table of data:

	vin	brand	model	color	plateno	capacity	cartype	insuranceno	ssn	
1	2Q8D365848Z41111	Dodge	Charger	Green	AZ008		3	Economy	98711231	233-90-1122
2	4Y1SL65848Z41111	Audi	Q3	Red	FL0099		5	SUV	123222222	111-22-2222
3	4Y1SL65848Z41143	Toyota	Camry	White	TX456		3	Economy	123111111	111-11-1111
4	4Y1SL65848Z42256	BMW	S6	Orange	TX342		3	SUV	123111112	111-11-1111
5	8CD365848Z4111	Toyota	Corolla	Brown	TX124		3	Economy	4561231234	123-33-0091
6	8HB0D365848Z4111	Mercedes	GLB-22	Black	WS887		5	Black SUV	99123123	233-90-1122

## 6. SALARY\_ACCOUNTS

```
CREATE TABLE Salary_Accounts
(
    AccountNum      char(10)        NOT NULL,
    RoutingNum      char(9)         NOT NULL,
    AccType         varchar(10)      NOT NULL,
    SSN             char(11)         NOT NULL,
    PRIMARY KEY(AccountNum,RoutingNum),
    FOREIGN KEY (SSN) REFERENCES Driver(SSN) ON DELETE CASCADE
);
```

```

SELECT* FROM PERSON;
SELECT* FROM INFO;
SELECT* FROM ORDER;
SELECT* FROM INSURANCE;
SELECT* FROM CARS;
SELECT* FROM SALARY_ACCOUNTS;

```

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

	accountnum	routingnum	acctype	ssn
1	1011329666	111000222	Saving	111-22-2222
2	1011329778	111009288	Checking	123-33-0091
3	6066329666	994144632	Checking	111-11-1111
4	7889911921	111000923	Saving	233-90-1122

## 7. CUSTOMER

```

CREATE TABLE Customer
(
    MobNo      varchar(13)          ,
    CustType    varchar(15)        DEFAULT 'Blue' NOT NULL ,
    UberPoints  int              DEFAULT 0 NOT NULL,
    WalletBal   decimal(10,2)     DEFAULT 0 NOT NULL,
    ReferLink   varchar(70)       NOT NULL,
    CustRating  decimal(2,1)      ,

    PRIMARY KEY (MobNo),
    FOREIGN KEY (MobNo) REFERENCES Person(MobNo) ON DELETE CASCADE
);

```

The screenshot shows the Oracle SQL Developer interface. The top navigation bar includes tabs for 'Consumer Group: LOW' and 'Consumer Group: HIGH'. The main area displays a query editor with the following SQL code:

```
SELECT** FROM PERSON;
SELECT * FROM CUSTINFO;
SELECT * FROM DRIVER;
SELECT * FROM INSURANCE;
SELECT * FROM CARS;
SELECT * FROM SALARY_ACCOUNTS;
SELECT * FROM CUSTOMER;
```

Below the query editor is a toolbar with icons for file operations like New, Open, Save, Print, and Delete. The bottom navigation bar includes tabs for 'Query Result', 'Script Output', 'DBMS Output', 'Explain Plan', 'Autotrace', 'SQL History', and 'Data Loading'. The status bar at the bottom indicates an execution time of 0.023 seconds.

	mobno	custtype	uberpoints	walletbal	referlink	custrating
1	+1469111111	Blue	123	500.23	wiufbqvwhbfkaduwqhf	5
2	+1469333333	Blue	450	0	ehivbwvriebvbu	4.8
3	+1469444444	Platinum	1000	877.9	bieuwugfyewfviewwf	5
4	+1469666666	Gold	566	23.65	hcbuvveygeuw	5
5	+1469777777	Blue	200	45.66	beauwyfbuuvew	4.9

## 8. SAVED ADDRESSES

```
CREATE TABLE Saved_Addresses
(
    MobNo      varchar(13)      NOT NULL,
    Tag        int              NOT NULL,
    Street     varchar(30)       NOT NULL,
    City       varchar(30)       NOT NULL,
    AddrState  varchar(30)       NOT NULL,
    ZIP        int              NOT NULL,
    PRIMARY KEY (MobNo,Tag),
    FOREIGN KEY (MobNo) REFERENCES Customer(MobNo) ON DELETE CASCADE
);
```

```

SELECT** FROM PERSON;
SELECT * FROM DL_INFO;
SELECT * FROM DRIVER;
SELECT * FROM INSURANCE;
SELECT * FROM CARS;
SELECT * FROM SALARY_ACCOUNTS;
SELECT * FROM CUSTOMER;
SELECT * FROM SAVED_ADDRESSES;

```

Consumer Group: LOW    00    ⓘ

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

Download    Execution time: 0.018 seconds

	mobno	tag	street	city	addrstate	zip
1	+1469111111	1	12, Preston Rd	Dallas	TX	75252
2	+1469111111	2	34, Campbell Rd	Dallas	TX	75050
3	+1469444444	1	45, Colorado Dr	Tampa	FL	67505
4	+1469444444	2	65, Johnson Dr	Fairfax	VA	34533
5	+1469444444	3	3455, James Dr	Dallas	TX	75090
6	+1469666666	1	7433, Ash Dr	Dallas	TX	75455
7	+1469777777	1	7677, Frankford Rd	Dallas	TX	75252

## 9. SAVED\_CARDS

```

CREATE TABLE Saved_Cards
(
    CardNum          varchar(16)      ,
    NameOnCard       varchar(50)      NOT NULL,
    CardExpiry       DATE           NOT NULL,
    CardType         varchar(50)      NOT NULL,
    CardNetwork      varchar(30)      NOT NULL,
    CardNickname     varchar(30)      ,
    isDefault        CHAR(1)        ,
    MobNo            varchar(13)      NOT NULL,
    PRIMARY KEY (CardNum),
    FOREIGN KEY (MobNo) REFERENCES Customer(MobNo) ON DELETE CASCADE
);

```

The screenshot shows the Oracle SQL Developer interface. The top menu bar includes 'File', 'Edit', 'Tools', 'Database', 'Help', and 'About'. Below the menu is a toolbar with icons for SELECT, INSERT, UPDATE, DELETE, and other database operations. The main area displays a query window with the following code:

```
1 SELECT * FROM PERSON;
2 SELECT * FROM DL_INFO;
3 SELECT * FROM DRIVER;
4 SELECT * FROM INSURANCE;
5 SELECT * FROM CARS;
6 SELECT * FROM SALARY_ACCOUNTS;
7 SELECT * FROM CUSTOMER;
8 SELECT * FROM SAVED_ADDRESSES;
9 SELECT * FROM SAVED_CARDS;
```

To the right of the code, there's a 'Consumer Group' dropdown set to 'LOW'. Below the code, the results pane shows the following table:

	cardnum	nameoncard	cardexpiry	cardtype	cardnetwork	cardnickname	isdefault	mobno
1	1234123412341234	Alice Addison	9/7/2024, 12:00:00 AM	Credit	VISA	(null)	1	+1469111111
2	3455612234118977	Fatima Ferez	7/1/2025, 12:00:00 AM	Debit	BBT	(null)	1	+1469666666
3	4588989976776566	Dorothy Daniels	9/7/2026, 12:00:00 AM	Credit	MasterCard	Dos Card	1	+1469444444
4	767754556532122	Greta Gabbins	8/1/2022, 12:00:00 AM	Credit	Amex	(null)	(null)	+1469777777
5	989987883435122	Simon Daniels	8/4/2023, 12:00:00 AM	Debit	Chase	Sams Card	0	+1469444444

Below the table, the message 'Execution time: 0.006 seconds' is displayed.

## 10. TRIP REQUESTS

```
CREATE TABLE Trip_Requests
(
    ReqID           int GENERATED BY DEFAULT ON NULL AS IDENTITY,
    BookingTime     TIMESTAMP        NOT NULL,
    CarType         varchar(50)      NOT NULL,
    PickupLocn     varchar(50)      NOT NULL,
    DropLocn       varchar(50)      NOT NULL,
    EstCharge      decimal(5,2)      NOT NULL,
    MobNo          varchar(13)      NOT NULL,
    PRIMARY KEY(ReqID),
    FOREIGN KEY(MobNo) REFERENCES Customer(MobNo) ON DELETE CASCADE
);
```

```

SELECT** *
 1  SELECT * FROM PERSON;
 2  SELECT * FROM DL_INFO;
 3  SELECT * FROM DRIVER;
 4  SELECT * FROM INSURANCE;
 5  SELECT * FROM CAR;
 6  SELECT * FROM CARRIER_ACCOUNTS;
 7  SELECT * FROM CUSTOMER;
 8  SELECT * FROM SAVED_ADDRESSES;
 9  SELECT * FROM SAVED_CARDS;
10  SELECT * FROM TRIP_REQUESTS;

```

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

	reqid	bookingtime	cartye	pickuplocn	droplocn	estcharge	mobno
1	1	2022-04-29T13:59:59.990Z	Economy	7421 Frankford Rd	UT Dallas	8.6	+1469111111
2	2	2022-04-27T13:24:59.990Z	SUV	UT Dallas	Downtown Dallas	43.98	+1469333333
3	3	2022-03-13T10:24:59.990Z	Premium	3455, James Dr	Airport Dallas	56.87	+1469444444
4	4	2022-03-13T06:30:59.600Z	Economy	7433, Ash Dr	UT Dallas	23.45	+1469666666
5	5	2022-03-13T06:30:59.600Z	SUV	7488 Wrick Blvd	23, Preston Rd	45.55	+1469777777
6	9	2022-04-25T07:30:59.600Z	Economy	3455 West Dr	67 Bose Rd	34.65	+1469444444

## 11. ACCEPTS

```

CREATE TABLE Accepts
(
    RequestID      int      ,
    SSN            char(11)  ,
    AcceptTime     TIMESTAMP NOT NULL,
    PRIMARY KEY(RequestID,SSN),
    FOREIGN KEY (RequestID) REFERENCES Trip_Requests(ReqID) ON DELETE CASCADE,
    FOREIGN KEY (SSN) REFERENCES Driver(SSN)
);

```

```

SELECT * FROM PERSON;
SELECT * FROM DL_INFO;
SELECT * FROM DRIVER;
SELECT * FROM INSURANCE;
SELECT * FROM CAR;
SELECT * FROM CARRIER_ACCOUNTS;
SELECT * FROM CUSTOMER;
SELECT * FROM SAVED_ADDRESSES;
SELECT * FROM SAVED_CARDS;
SELECT * FROM TRIP_REQUESTS;
SELECT * FROM ACCEPTS;

```

Query Result

	requestid	ssn	accepttime
1	1	111-11-1111	2022-04-29T14:01:59.990Z
2	2	111-22-2222	2022-04-27T13:26:59.990Z
3	3	123-33-0091	2022-03-13T10:29:59.990Z
4	4	233-90-1122	2022-03-13T06:35:59.600Z
5	5	111-11-1111	2022-03-13T06:33:59.600Z

## 12. CANCELLED\_TRIPS

```

CREATE TABLE Cancelled_Trips
(
    RequestID      int          NOT NULL,
    CancelTime     TIMESTAMP     NOT NULL,
    Reason         varchar(30)   NOT NULL,
    Penalty        decimal(5,2)  NOT NULL,
    PRIMARY KEY(RequestID),
    FOREIGN KEY (RequestID) REFERENCES Trip_Requests(ReqID) ON DELETE CASCADE
);

```

```

1 SELECT * FROM PERSON;
2 SELECT * FROM DL_INFO;
3 SELECT * FROM DRIVER;
4 SELECT * FROM INSURANCE;
5 SELECT * FROM CARS;
6 SELECT * FROM SALARY_ACCOUNTS;
7 SELECT * FROM CUSTOMER;
8 SELECT * FROM SAVED_ADDRESSES;
9 SELECT * FROM SAVED_CARDS;
10 SELECT * FROM TRIP_REQUESTS;
11 SELECT * FROM ACCEPTS;
12 SELECT * FROM CANCELLED_TRIPS;

```

Query Result

requestid	canceltime	reason	penalty
1	9 2022-04-25T07:45:59.600Z	Too late	5.5

## 13.COUPONS

```

CREATE TABLE Coupons
(
    CouponCode  varchar(15),
    DiscPC  int,
    PRIMARY KEY (CouponCode)
);

```

```

1 SELECT * FROM PERSON;
2 SELECT * FROM DL_INFO;
3 SELECT * FROM DRIVER;
4 SELECT * FROM INSURANCE;
5 SELECT * FROM CARS;
6 SELECT * FROM SALARY_ACCOUNTS;
7 SELECT * FROM CUSTOMER;
8 SELECT * FROM SAVED_ADDRESSES;
9 SELECT * FROM SAVED_CARDS;
10 SELECT * FROM TRIP_REQUESTS;
11 SELECT * FROM ACCEPTS;
12 SELECT * FROM CANCELLED_TRIPS;
13 SELECT * FROM COUPONS;

```

Query Result

couponcode	discpc
1 NOPE00	0
2 MEMO30	15
3 UBER50	20

## 14. COMPLETE\_TRIPS

```

CREATE TABLE Complete_Trips
(
    TripID           int GENERATED BY DEFAULT ON NULL AS IDENTITY ,
    DriverArrivalTime   TIMESTAMP      NOT NULL,
    PickupTime        TIMESTAMP      NOT NULL,
    DropTime          TIMESTAMP      NOT NULL,
    RideCharge        decimal(5,2),
    TotalFare         decimal(5,2)     NOT NULL,
    Tip               decimal(5,2)     NOT NULL,
    CustRating        decimal(2,1)     NOT NULL,
    DriverRating       decimal(2,1)     NOT NULL,
    SSN               char(11)        NOT NULL,
    RequestID         int            NOT NULL,
    VIN                char(17)        NOT NULL,
    CouponCode        varchar(15),
    PRIMARY KEY(TripID),
    FOREIGN KEY (SSN) REFERENCES Driver(SSN),
    FOREIGN KEY (RequestID) REFERENCES Trip_Requests(ReqID) ,
    FOREIGN KEY (VIN) REFERENCES Cars(VIN) ,
    FOREIGN KEY (CouponCode) REFERENCES COUPONS(CouponCode)
);

```

•

The screenshot shows the Oracle SQL Developer interface. At the top, there's a toolbar with various icons. Below it is a consumer group dropdown set to 'LOW'. The main area has a code editor on the left containing the SQL for creating the 'Complete\_Trips' table. On the right is a large 'Query Result' window. The results show five rows of trip data:

	tripid	driverarrivaltime	pickuptime	droptime	ridecharge	totalfare	tip	custrating	driverrating	ssn	requestid	VIN	couponcode
1	1	2022-04-29T14:16:59.990Z	2022-04-29T14:18:59.990Z	2022-04-29T14:48:59.990Z	8.6	8.6	5	5	5	111-11-1111	1	4Y1SL65848Z4143	(null)
2	2	2022-04-27T13:35:59.990Z	2022-04-27T13:40:59.990Z	2022-04-27T14:00:59.990Z	43.98	37.38	10	5	5	111-22-2222	2	4Y1SL65848Z4111	UBER50
3	3	2022-03-13T10:45:59.990Z	2022-03-13T10:54:59.990Z	2022-03-13T11:24:59.990Z	56.87	56.87	0	4.8	4.5	123-33-0091	3	8C8D365848Z4111	(null)
4	5	2022-03-13T06:40:59.600Z	2022-03-13T06:59:59.600Z	2022-03-13T07:30:59.600Z	23.45	23.45	2	5	5	233-90-1122	4	2QBD365848Z4111	(null)
5	7	2022-03-13T06:49:59.600Z	2022-03-13T06:52:59.600Z	2022-03-13T07:30:59.600Z	45.55	45.55	1	5	5	111-11-1111	5	4Y1SL65848Z42256	MEMO30

## 15. PAYMENT\_HISTORY

```
CREATE TABLE Payment_History
(
    TransactID      int GENERATED BY DEFAULT ON NULL AS IDENTITY,
    Amount          decimal(6,2)        NOT NULL,
    PaymentMode     varchar(20)       NOT NULL,
    PaypalID        varchar(20),
    CardNum         varchar(16),
    TripID          int              NOT NULL,
    MobNo           varchar(13)       NOT NULL,

    PRIMARY KEY(TransactID),
    FOREIGN KEY (CardNum) REFERENCES Saved_Cards(CardNum) ON DELETE SET NULL,
    FOREIGN KEY (TripID) REFERENCES Complete_Trips(TripID) ON DELETE SET NULL,
    FOREIGN KEY (MobNo) REFERENCES Customer(MobNo) ON DELETE SET NULL
);
```

The screenshot shows the Oracle SQL Developer interface. At the top, there is a toolbar with various icons. Below the toolbar, the consumer group is set to 'LOW'. The main area contains two panes. The left pane displays the SQL code for creating the Payment\_History table. The right pane shows the results of a query that lists five rows of payment history data.

transactid	amount	paymentmode	paypalid	cardnum	tripid	mobno
1	1	13.6	Card	(null)	1234123412341234	1 +1469111111
2	2	47.38	Cash	(null)	(null)	2 +1469333333
3	3	56.87	Paypal	4CFD6677901254	(null)	3 +1469444444
4	4	25.45	Card	(null)	3455612234118977	5 +1469666666
5	5	46.55	Card	(null)	7677545565332122	7 +1469777777

# PL/SQL Procedures & Triggers

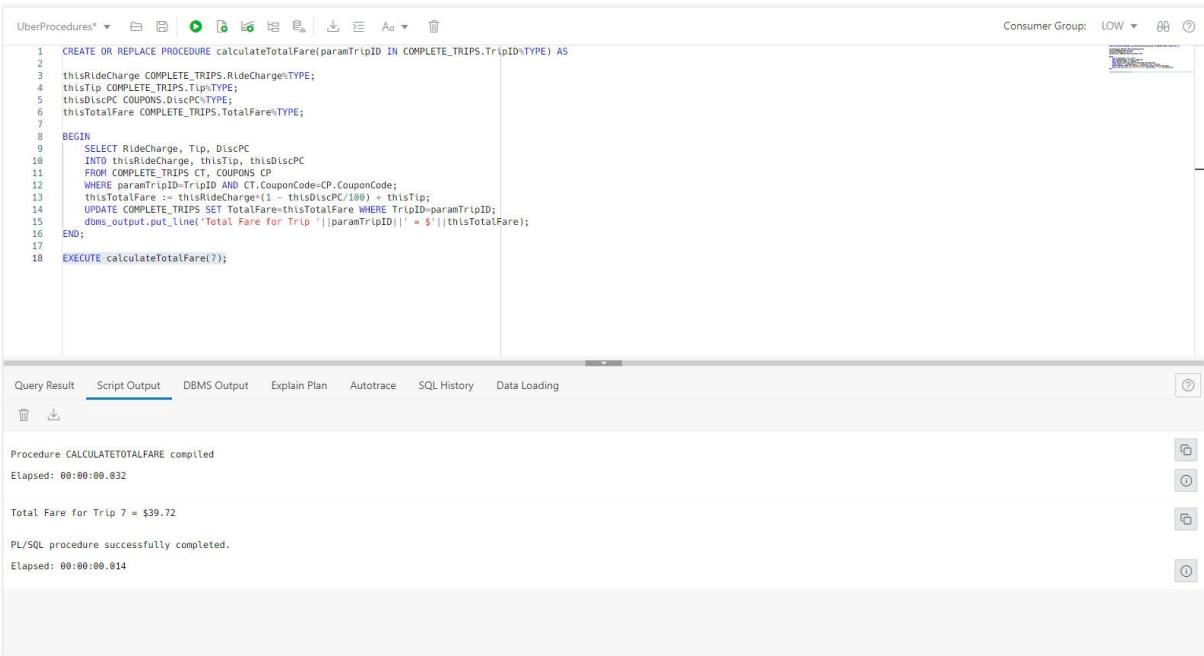
## Stored Procedures

### 1. Procedure to compute the total fare of a completed trip.

```
CREATE OR REPLACE PROCEDURE calculateTotalFare(paramTripID IN
COMPLETE_TRIPS.TripID%TYPE) AS

thisRideCharge COMPLETE_TRIPS.RideCharge%TYPE;
thisTip COMPLETE_TRIPS.Tip%TYPE;
thisDiscPC COUPONS.DiscPC%TYPE;
thisTotalFare COMPLETE_TRIPS.TotalFare%TYPE;

BEGIN
    SELECT RideCharge, Tip, DiscPC
    INTO thisRideCharge, thisTip, thisDiscPC
    FROM COMPLETE_TRIPS CT, COUPONS CP
    WHERE paramTripID=TripID AND CT.CouponCode=CP.CouponCode;
    thisTotalFare := thisRideCharge*(1 - thisDiscPC/100) + thisTip;
    UPDATE COMPLETE_TRIPS SET TotalFare=thisTotalFare WHERE
    TripID=paramTripID;
    dbms_output.put_line('Total Fare for Trip '||paramTripID||' =
$'||thisTotalFare);
END;
```



The screenshot shows the Oracle SQL Developer interface with the code for the `calculateTotalFare` procedure. The code is identical to the one provided above. Below the code, the results of the compilation and execution are displayed:

```
Procedure CALCULATETOTALFARE compiled
Elapsed: 00:00:00.032
Total Fare for Trip 7 = $39.72
PL/SQL procedure successfully completed.
Elapsed: 00:00:00.014
```

Compilation and Execution of procedure

```

1 CREATE OR REPLACE PROCEDURE calculateTotalFare(paramTripID IN COMPLETE_TRIPS.TripID%TYPE) AS
2   thisRideCharge COMPLETE_TRIPS.RideCharge%TYPE;
3   thisTip COMPLETE_TRIPS.Tip%TYPE;
4   thisDiscPC COUPONS.DiscPC%TYPE;
5   thisTotalFare COMPLETE_TRIPS.TotalFare%TYPE;
6
7   BEGIN
8     SELECT RideCharge, Tip, DiscPC
9       INTO thisRideCharge, thisTip, thisDiscPC
10      FROM COMPLETE_TRIPS CT, COUPONS CP
11     WHERE paramTripID=TripID AND CT.CouponCode=CP.CouponCode;
12
13     thisTotalFare := thisRideCharge*(1 - thisDiscPC/100) + thisTip;
14
15     UPDATE COMPLETE_TRIPS SET TotalFare=thisTotalFare WHERE TripID=paramTripID;
16
17     dbms_output.put_line('Total Fare for Trip '||paramTripID||' = '$||thisTotalFare);
18
19   END;
20
21   EXECUTE calculateTotalFare(7);
22
23   SELECT * FROM COMPLETE_TRIPS WHERE COMPLETE_TRIPS.TRIPID=7;

```

Query Result

	tripid	driverarrivaltime	pickuptime	droptime	ridecharge	totalfare	tip	custrating	driverrating	ssn	requestid	vin	couponcode
1	7	2022-03-13T06:49:5	2022-03-13T06:52:5	2022-03-13T07:30:5	45.55	39.72	1	5	5	111-11-1111	5	4Y1SL65848Z42256	MEMO30

Updated value in table after procedure execution

## 2. Procedure to compute the average rating of a driver.

```

CREATE OR REPLACE PROCEDURE avgDriverRating(paramSSN IN Driver.SSN%TYPE) AS

thisAvg Driver.DriverRating%TYPE;

BEGIN
  SELECT AVG(DriverRating)
  INTO thisAvg
  FROM Complete_Trips
  WHERE SSN=paramSSN;
  UPDATE Driver SET DriverRating=thisAvg WHERE SSN=paramSSN;
  dbms_output.put_line('Updated rating for this driver is ' || thisAvg);
END;

```

The screenshot shows the Oracle SQL Developer interface. In the top-left pane, there is a code editor with the following PL/SQL procedure:

```

23  -- Procedure to compute the average rating of a driver
24  CREATE OR REPLACE PROCEDURE avgDriverRating(paramSSN IN Driver.SSN%TYPE) AS
25
26    thisAvg Driver.DriverRating%TYPE;
27
28    BEGIN
29      SELECT AVG(DriverRating)
30        INTO thisAvg
31        FROM Complete_Trips
32        WHERE SSN=paramSSN;
33        UPDATE Driver SET DriverRating=thisAvg WHERE SSN=paramSSN;
34        dbms_output.put_line('Updated rating for this driver is ' || thisAvg);
35
36    END;
37
38  EXECUTE avgDriverRating('123-33-0091');

```

In the top-right pane, there is a graphical representation of the database schema. Below the code editor, the tabs "Query Result" and "Script Output" are selected. The "Script Output" tab shows the following output:

```

Procedure AVGDIVERATING compiled
Elapsed: 00:00:00.012

Updated rating for this driver is 4.5

PL/SQL procedure successfully completed.
Elapsed: 00:00:00.012

```

## Compilation and Execution of procedure

The screenshot shows the Oracle SQL Developer interface. In the top-left pane, there is a code editor with the same PL/SQL procedure as before, plus an additional SELECT statement at the end:

```

23  -- Procedure to compute the average rating of a driver
24  CREATE OR REPLACE PROCEDURE avgDriverRating(paramSSN IN Driver.SSN%TYPE) AS
25
26    thisAvg Driver.DriverRating%TYPE;
27
28    BEGIN
29      SELECT AVG(DriverRating)
30        INTO thisAvg
31        FROM Complete_Trips
32        WHERE SSN=paramSSN;
33        UPDATE Driver SET DriverRating=thisAvg WHERE SSN=paramSSN;
34        dbms_output.put_line('Updated rating for this driver is ' || thisAvg);
35
36    END;
37
38  EXECUTE avgDriverRating('123-33-0091');
39
40  SELECT * FROM Driver WHERE Driver.SSN='123-33-0091';

```

In the bottom-right pane, there is a table viewer showing the results of the SELECT statement. The table has columns: ssn, addr, driverrating, dlnumber, and mobno. There is one row with the following values:

	ssn	addr	driverrating	dlnumber	mobno
1	123-33-0091	56, Frankford Rd	4.5	47033333	+1469555555

## Updated value in table after procedure execution

# Triggers

## 1. Trigger to ensure Car Insurance has not already expired

```
CREATE OR REPLACE TRIGGER insuranceExpiry
BEFORE INSERT OR UPDATE
ON INSURANCE FOR EACH ROW
BEGIN
    IF (:new.InsuranceExpiry < sysdate)
        THEN raise_application_error(-20101, 'This insurance has expired. Please
renew or enter another.');
    END IF;
END;
```

The screenshot shows the Oracle SQL Developer interface with the 'UberProcedures' tab selected. The code area contains the trigger definition. The status bar at the bottom indicates the trigger was compiled successfully in 0.00046 seconds. A red error message box displays three errors: ORA-20101, ORA-06512, and ORA-04088.

```
41
42
43
44 -- Trigger to ensure Car Insurance has not already expired
45 CREATE OR REPLACE TRIGGER insuranceExpiry
46 BEFORE INSERT OR UPDATE
47 ON INSURANCE FOR EACH ROW
48 BEGIN
49     IF (:new.InsuranceExpiry < sysdate)
50         THEN raise_application_error(-20101, 'This insurance has expired. Please
renew or enter another.');
51     END IF;
52 END;
53
54 INSERT INTO INSURANCE VALUES ('12375148', TO_DATE('2021-06-09', 'YYYY-MM-DD'));
```

Trigger INSURANCEEXPIRY compiled  
Elapsed: 00:00:00.046

ORA-20101: This insurance has expired. Please renew or enter another.  
ORA-06512: at "ADMIN.INSURANCEEXPIRY", line 3  
ORA-04088: error during execution of trigger 'ADMIN.INSURANCEEXPIRY'

## 2. Trigger to check if expiry of Driver's License is in the past

```
CREATE OR REPLACE TRIGGER driversLicenseExpiry
BEFORE INSERT OR UPDATE
ON DL_INFO FOR EACH ROW
BEGIN
    IF (:new.DLExpiry < sysdate)
        THEN raise_application_error(-20102, 'This license has expired. Please
renew before entering details in the system.');
    END IF;
END;
```

The screenshot shows the Oracle SQL Developer interface with the code editor open. The code is as follows:

```

55
56
57  -- Trigger to check if expiry of Driver's License is in the past
58  CREATE OR REPLACE TRIGGER driversLicenseExpiry
59  BEFORE INSERT OR UPDATE
60  ON DL_INFO FOR EACH ROW
61  BEGIN
62      IF (:new.DLExpiry < sysdate)
63          THEN raise_application_error(-20102, 'This license has expired. Please renew before entering details in the system.');
64      END IF;
65  END;
66
67  INSERT INTO DL_INFO VALUES ('47083459', TO_DATE('2020-06-06', 'YYYY-MM-DD'));
68
69

```

The 'Script Output' tab is selected, showing the results of the compilation:

Trigger DRIVERSLICENSEEXPIRY compiled  
Elapsed: 00:00:00.079

ORA-20102: This license has expired. Please renew before entering details in the system.  
ORA-06512: at "ADMIN.DRIVERSLICENSEEXPIRY", line 3  
ORA-04088: error during execution of trigger 'ADMIN.DRIVERSLICENSEEXPIRY'

### 3. Trigger to check if a customer's saved card is expired or valid.

```

CREATE OR REPLACE TRIGGER cardExpiry
BEFORE INSERT OR UPDATE
ON Saved_Cards FOR EACH ROW
BEGIN
    IF (:new.CardExpiry < sysdate)
        THEN raise_application_error(-20103, 'This card is expired.');
    END IF;
END;

```

The screenshot shows the Oracle SQL Developer interface with the code editor open. The code is as follows:

```

69
70  -- Trigger to check if a customer's saved card is expired or valid.
71  CREATE OR REPLACE TRIGGER cardExpiry
72  BEFORE INSERT OR UPDATE
73  ON Saved_Cards FOR EACH ROW
74  BEGIN
75      IF (:new.CardExpiry < sysdate)
76          THEN raise_application_error(-20103, 'This card is expired.');
77      END IF;
78  END;
79
80
81  INSERT INTO SAVED_CARDS VALUES ('3455669234118977', 'Lorem Ipsum', TO_DATE('2021-07-01', 'YYYY-MM-DD'), 'Credit', 'Forex', NULL, 1, '+1469666666');
82
83

```

The 'Script Output' tab is selected, showing the results of the compilation:

Trigger CARDEXPIRY compiled  
Elapsed: 00:00:00.087

ORA-20103: This card is expired.  
ORA-06512: at "ADMIN.CARDEXPIRY", line 3  
ORA-04088: error during execution of trigger 'ADMIN.CARDEXPIRY'

# THANK YOU!