Database Design for the Doordash Application

Instructor:

Nurcan Yuruk, Ph.D.

Team Members:

Bharath Challa -BXC200008

Giftson Vasanth A – GXA210001

INDEX

Data Collection

Doordash is an all-in-one application that provides on-demand delivery or Pickup from restaurants, nearby grocery, and convenience stores, and more. In our project we have collected all data specific to pick up and delivery of the orders placed by the customer.

- Each user must create an account to enjoy services of the application. Each customer is identified by the Customer Id, and all the below details are also collected from the user.
    - First Name, Last Name, Email, Country code, Mobile Number, Password, Dashpass Address
- Customer can store multiple address for the food to be delivered. We assume that each address is stored with unique address id and contains street no., street name, city, state, and pin code which are stored in database to ensure the hassle-free delivery.
    - street no., street name, city, state, and pin code
- Once the user logs in, they can start ordering the food they need by exploring the restaurants. We give unique key to each restaurant – restaurant id. Each restaurant has
    - Name, Cuisine category (Mexican, Chinese, Indian, Italian, Mediterranean), Email Id, Number to contact and the ratings which are given by the users.
- Users can provide review about the restaurant in words and can also provide ratings out of 5 stars and the date on which the review is provided is also stored.
    - i.e. we store Review Description, Rating, Date
- After choosing the restaurant, user can choose the food they need. Each food is identified by the Food Id and contains,
    - Food Name, Description, Category (Salad, Veg, Non-Veg), Options (Regular, Medium, Large) Calories, Price, Food Images, and the location in which the image is stored in the server.
- Once the user chooses the food, the order is generated. Each order has,
    - Order id, contact number for anu questions regarding the order, promo code if applied for the order, Total order price along with tax information.

- Order placed by the users can either be picked up or delivery. The date and time for the pickup or the delivery of the order is given by the user.

- For the delivery, the user should provide the delivery address. The delivery fee is then calculated based on the distance. User can add the delivery tip to the door dasher. For each delivery of the order the order status is also maintained. Hence for each order scheduled for delivery we store,

    - Delivery Address, Delivery Fee, Delivery Status, Delivery Tip.

- For the orders some time the offer code is applied based on the trustworthiness of the customer to honor their faithfulness to their restaurant if they have ordered multiple times.
  Note: This is an assumption.

    - Each offer is identified by Offer Id and contains offer id, discount percentage or the discount amount

- Once the user has ordered, the application takes the user to the payment page. The user can add multiple payments methods like Credit Card, Venmo, PayPal and also through Apple account if they use Apple machines.

    - Each payment method is identified by the payment id and the venmo id, paypal id, apple id, card information like card no., cvc, expiry month and the expiry year is stored.

- Each payment for the order is tracked by separate transaction id.

    - For each transaction, transaction date, time, and the transaction status of is recorded.

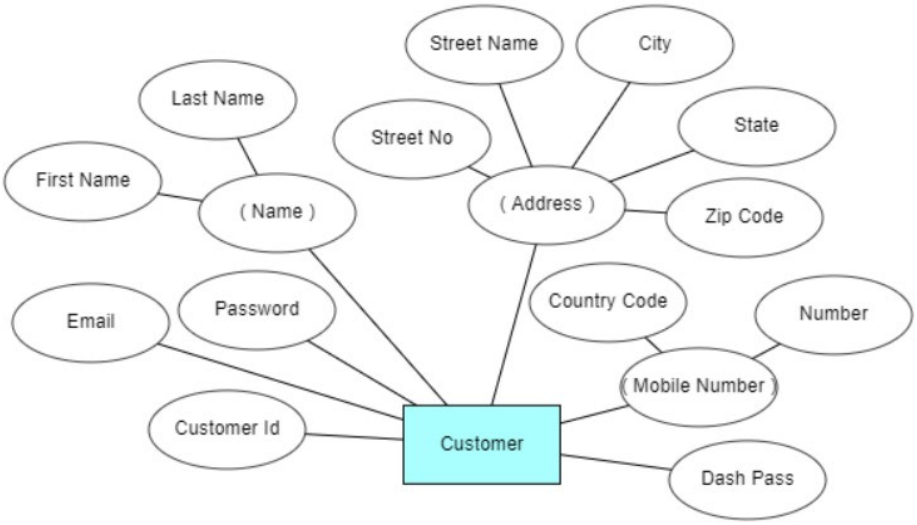
- Once the ordering and payment is done, customer can come and collect the order from the restaurant. In case of the delivery order door dasher is assigned to each order to deliver food to the user.

- Each door dasher is identified by the SSN. Other information about the door dasher like name, driving license id, Email, contact no, bank account number to credit the salary, ratings, number of orders fulfilled by the door dasher is also recorded.

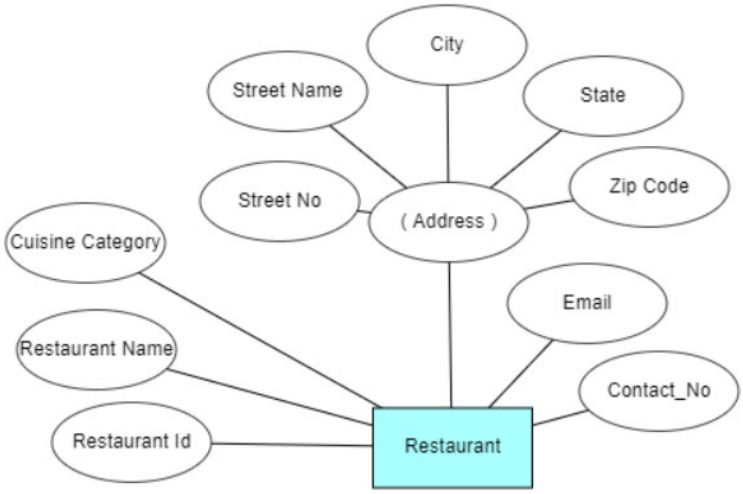
    - We also register the vehicle information which the door dasher uses. Vehicle information includes Vehicle Plate No , State in which the vehicle is registered and the vehicle type.

Identifying Entities and the attributes:

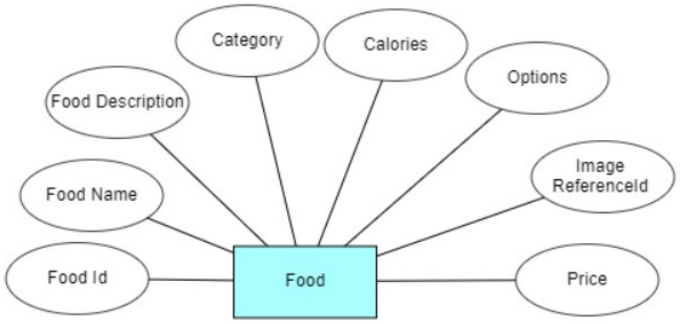
Customer/User:



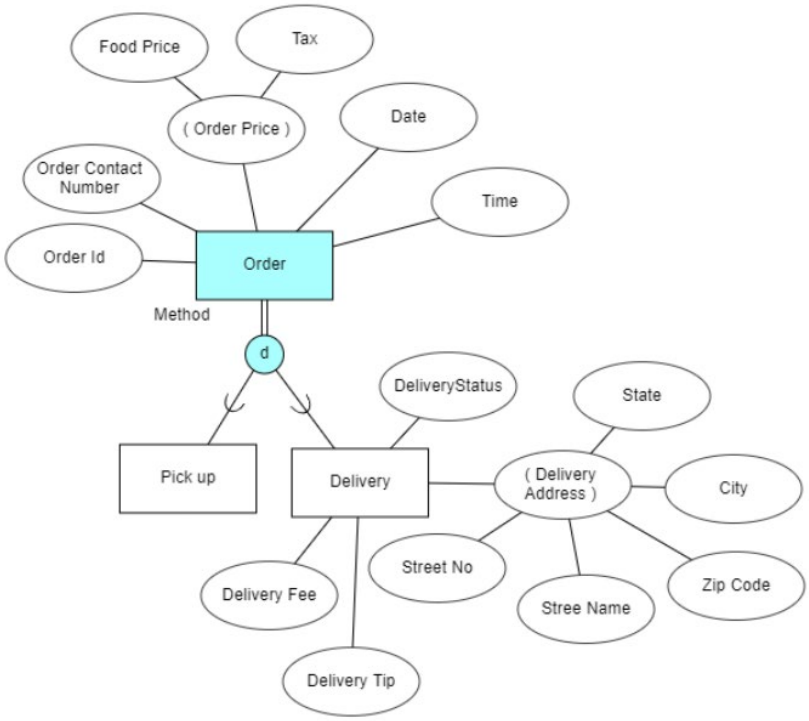
Restaurant:



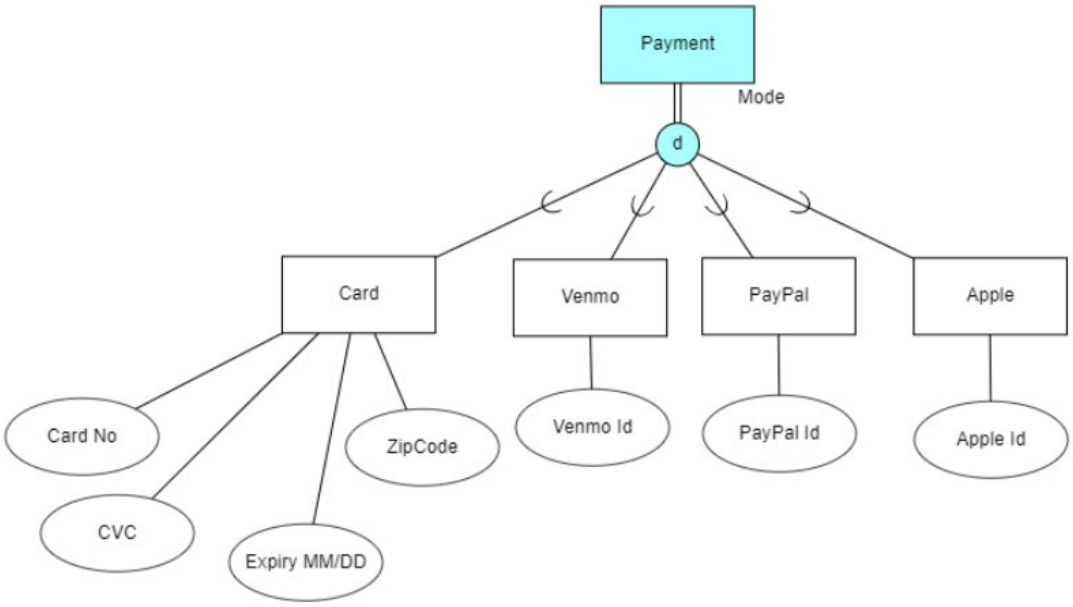
Food:


Category
Calories
Options
Food Description
Image ReferenceId
Food Name
Food Id
Food
Price

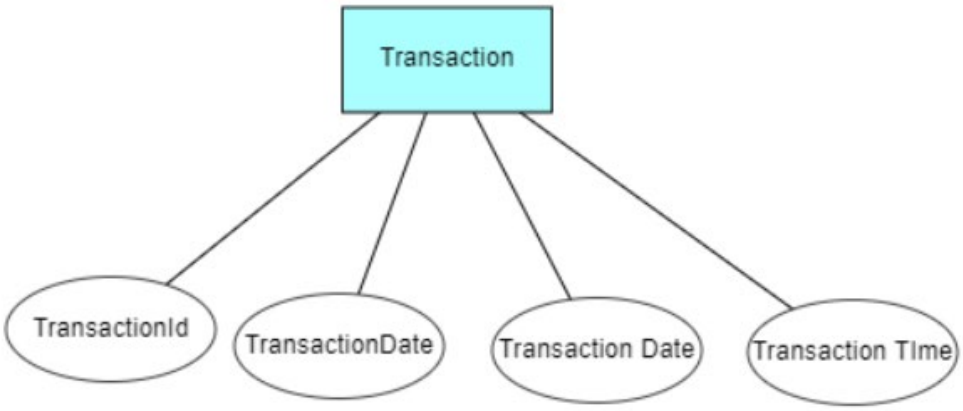
Order:


Food Price
Tax
(Order Price)
Date
Order Contact Number
Time
Order Id
Order
Method
d
DeliveryStatus
State
Pick up
Delivery
(Delivery Address)
City
Street No
Zip Code
Delivery Fee
Stree Name
Delivery Tip

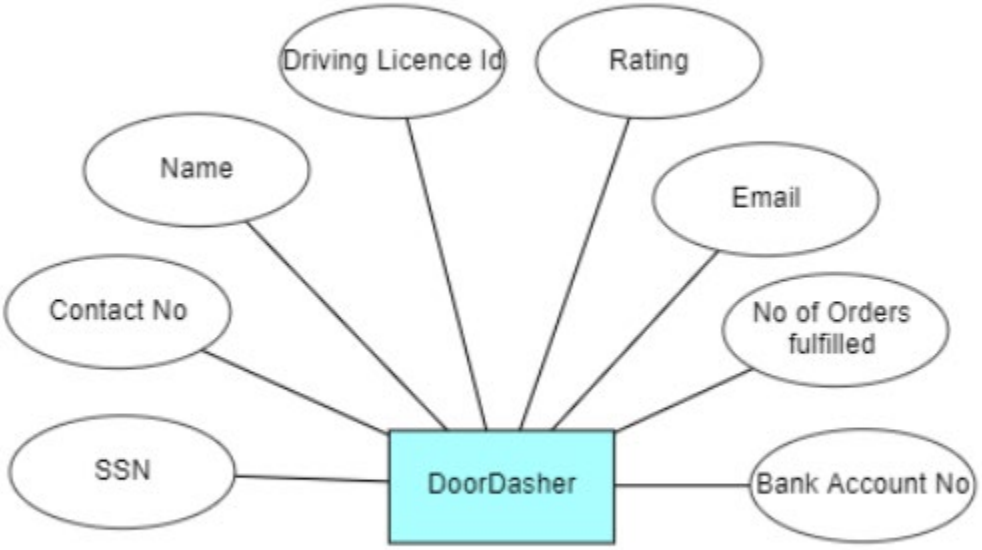
Payment



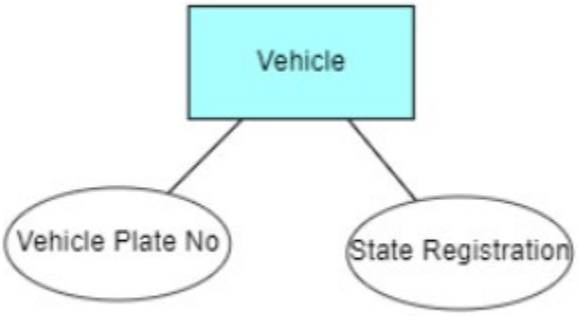
Transaction:



Doordasher:



Vehicle



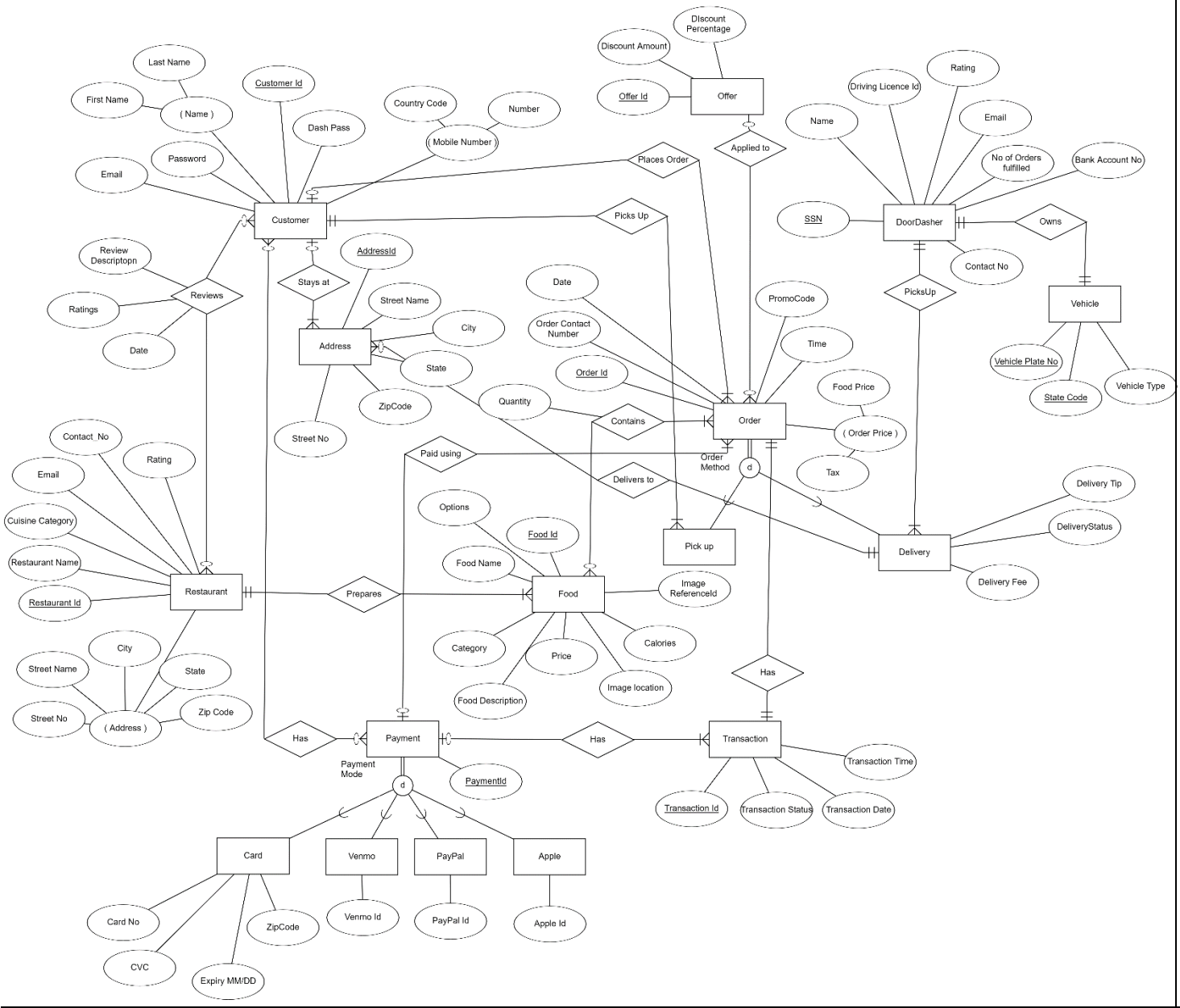
Identifying Relationships, Cardinality and Participation

Relationships	Entities	Cardinality	Participation
Reviews	Customer Restaurant	Many to Many One customer can provide review to multiple restaurants. One Restaurant can be commented by many customers	Both customer and restaurants are not mandatory to participate in this relationship Hence Partial particiation
Stays at	Customer Address	One to Many One Customer can have multiple addresses for their orders to be delivered One address is unique is unique to one customer	Since Address is weak entity, it involves in total participation. Customer may not require to provided address unless they order. So partial participation
Prepares	Restaurant Food	One to Many One restaurant can prepare many food items. But food item with one food id cannot be in multiple restaurants	Both food and restaurant should participate in this relationship. Hence total participation
Contains	Food Order	Many to Many One order contains many food items. Also one food item can be present in multiple orders	Order should participate in relation. So total One food item chosen by customer participates in the relationship. Hence partial
Places order	Customer Order	One to Many One Customer can place multiple orders. One order is associated with only one customer.	All customers using the application may not participate in this relation. Hence partial. Order must participate in this relation. Hence total

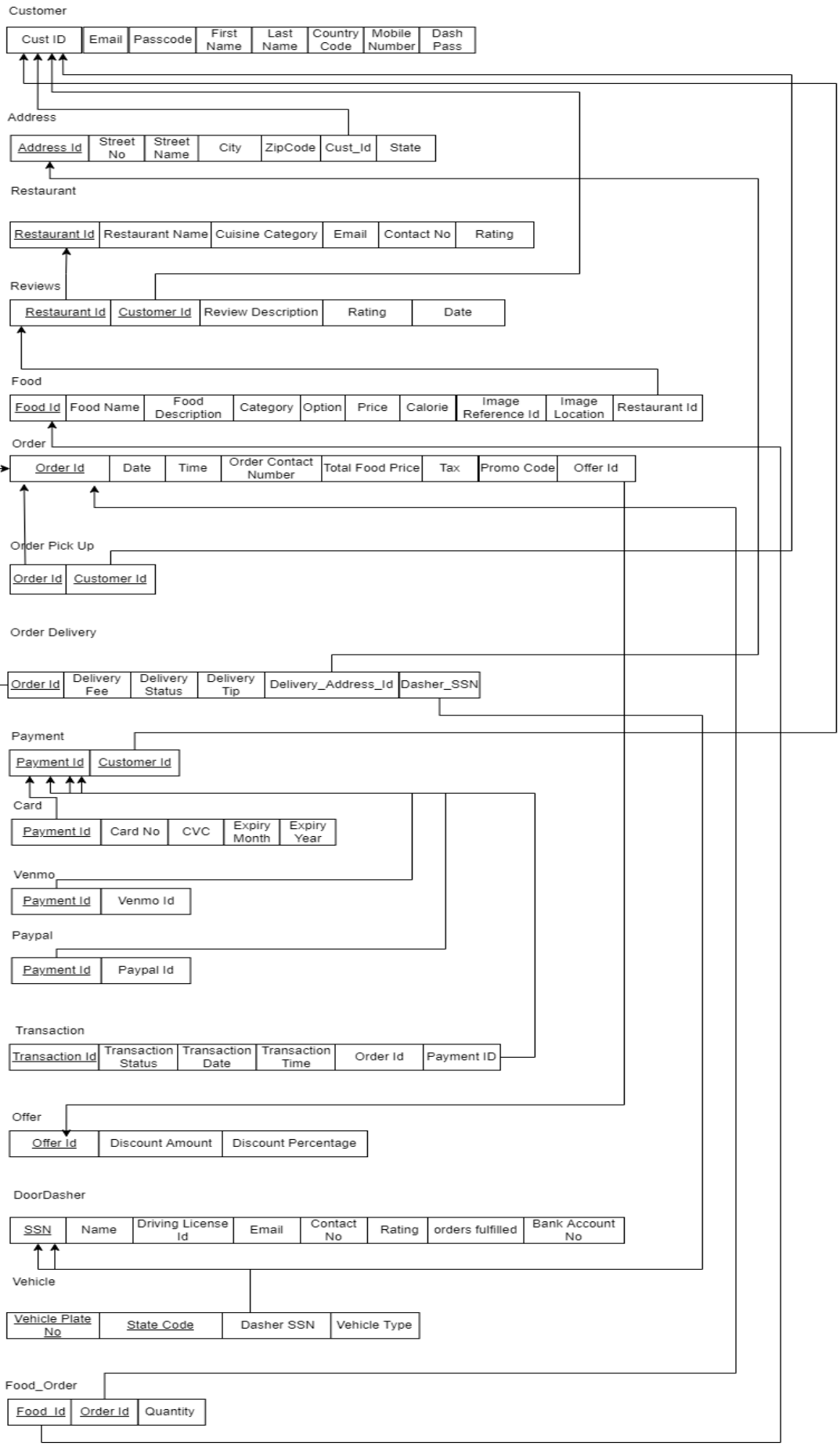
Applied to	Offer Order	One to Many One offer id can be applied to multiple orders. For one order, at maximum only one offer can be applied	Both participation of entities are partial as entities offer, order may or may not participate in relationship
Has	Customer Payment	One to Many One customer can have multiple payment methods, hence can have different payment ids. But one payment id cannot be shared by multiple customers	Participation of both entities in this relationship is partial as customer can either add or not add payment methods
Paid	Order Payment	Many to One Many orders can be paid using one payment method chosen by the customers. But one of the payment ids is chosen for one order	Order must participate. Hence the participation is total. Not all payment method participates. Hence it is partial for Payment
Has	Payment Transaction	One to Many We can pay using one payment method for multiple orders. And for each we maintain unique transaction. Hence relationship is one to Many	One of the payment ids participate. Hence partial. Transaction should participate. Hence it is total
Has	Order Transaction	One to one For each order, we have one transaction and vice versa	All orders and transaction participate. Hence both entities are in total participation

Picks Up	Customer Pick UP	One to Many One customer can pick up multiple orders. One order can be associated with only one customer id.	Customer may or may not participate in the relationship. Hence the relationship is partial. All orders must participate in the relationship. Hence total.
Picks Up	Doordasher Delivery	One to Many One doordasher can pick up multiple orders. But one order can be picked up by only one doordasher	All doordashers may not participate. Hence partial. But orders must participate. Hence total
Owns	Doordasher Vehicle	One to One Each door dasher can have only one vehicle. And one vehicle can be registered against only one doordasher	Both entities are involved in total participation.

ER Diagram


Last Name
First Name
(Name)
Customer Id
Dash Pass
Country Code
Number
(Mobile Number)
Password
Email
Customer
Discount Percentage
Discount Amount
Offer Id
Offer
Applied to
Places Order
Picks Up
Rating
Driving Licence Id
Name
Email
No of Orders fulfilled
Bank Account No
SSN
DoorDasher
Owns
Contact No
PicksUp
Vehicle
Vehicle Plate No
State Code
Vehicle Type
Review Descriptopn
Ratings
Date
Reviews
Stays at
AddressId
Street Name
City
State
ZipCode
Street No
Address
Date
Order Contact Number
Order Id
Quantity
Contains
Order
PromoCode
Time
Food Price
(Order Price)
Tax
Order Method
d
Paid using
Delivers to
Pick up
Delivery
Delivery Tip
DeliveryStatus
Delivery Fee
Contact_No
Rating
Email
Cuisine Category
Restaurant Name
Restaurant Id
Restaurant
Prepares
Options
Food Id
Food Name
Food
Image ReferenceId
Category
Price
Calories
Image location
Food Description
City
Street Name
State
Street No
(Address)
Zip Code
Has
Payment
Payment Mode
d
PaymentId
Has
Transaction
Has
Transaction Time
Transaction Id
Transaction Status
Transaction Date
Card
Venmo
PayPal
Apple
Card No
CVC
Expiry MM/DD
ZipCode
Venmo Id
PayPal Id
Apple Id

Mapping ER to Relational Schema



Identification of Functional Dependencies:

1.Customer

CustId - > Email, Password, First Name, Last Name, Country Code, Mobile Number, DashPass

2.Address

AddressId -> Street No, Street Name, City, ZipCode, CustId

ZipCode -> City,State

3.Restaurant

RestaurantId -> Restaurant Name, Cuisine Category, Email, Contact no.

4.Reviews

RestId,CustId -> Review Description, Rating, Date

5.Food

Food Id - > Food Name, Food Description, Category, Option, Calorie, Image Reference Id, Image Location, Restaurant Id

Image Reference Id - > Image Location

6.Order

OrderId -> Date, Time, Order Contact Number, Total Food Price, Tax, Promo Code

7.Order Pickup

OrderId, CustomerId (composite primary key and also both are foreign keys)

8.Order Delivery

Order Id -> Delivery Fee, Status, Delivery Tip, Address Id, DoorDasher SSN

9.Payment

PaymentId, CustomerId (composite primary key and both are foreign keys)

10.Card

PaymentId - > Card No, CVC, Expiry Date, Expiry Month, ZipCode

Card No -> CVC, Expiry Date, Expiry Month

11.Venmo

PaymentId -> VenmoId

12. Paypal

PaymentId -> Paypal Id

13.Transaction

Transaction Id -> Transaction Status, TransactionDate, Transaction Time, Order Id, Payment Id

14. Offer

OfferId -> Discount amount, Discount Percentage, Order Id

15. Doordasher

SSN - > Name, Email, Driving License Id, Contact No, Rating, Orders Fulfilled, BankAccountNo

16. Vehicle Id,

Vehicle Plate No , State Code – Dasher SSN, Vehicle Type

Dasher SSN – Vehicle Plate No, State Code Vehicle Type

17. Food Order

Food Id, Order Id – Quantity

Normalization:

- All relations are in 1 NF and in 2 NF.

1.Address

AddressId -> Street No, Street Name, City, ZipCode, CustId

ZipCode -> City,State

The relation is not in 3 NF, because of transitive dependency.

We split the relation into two – Address and ZipCode

Address

Address Id	Street No	Street Name	ZipCode	Cust_Id

ZipCode

ZipCode	City	StateName

2.Food

Food Id - > Food Name, Food Description,Category, Option, Calorie, Image Reference Id, Image Location

Image Reference Id - > Image Location

The relation is not in 3 NF, because of transitive dependency

We split the relations.

Food

Food Id	Food Name	Food Description	Category	Option	Price	Calorie	Image Reference Id	Restaurant Id

Image

Image Reference Id	Image Location

3. Card

PaymentId - > Card No, CVC, Expiry Date, Expiry Month

Card No - > CVC, Expiry Date, Expiry Month

The relation is not in 3 NF, because of transitive dependency. So we split the relations.

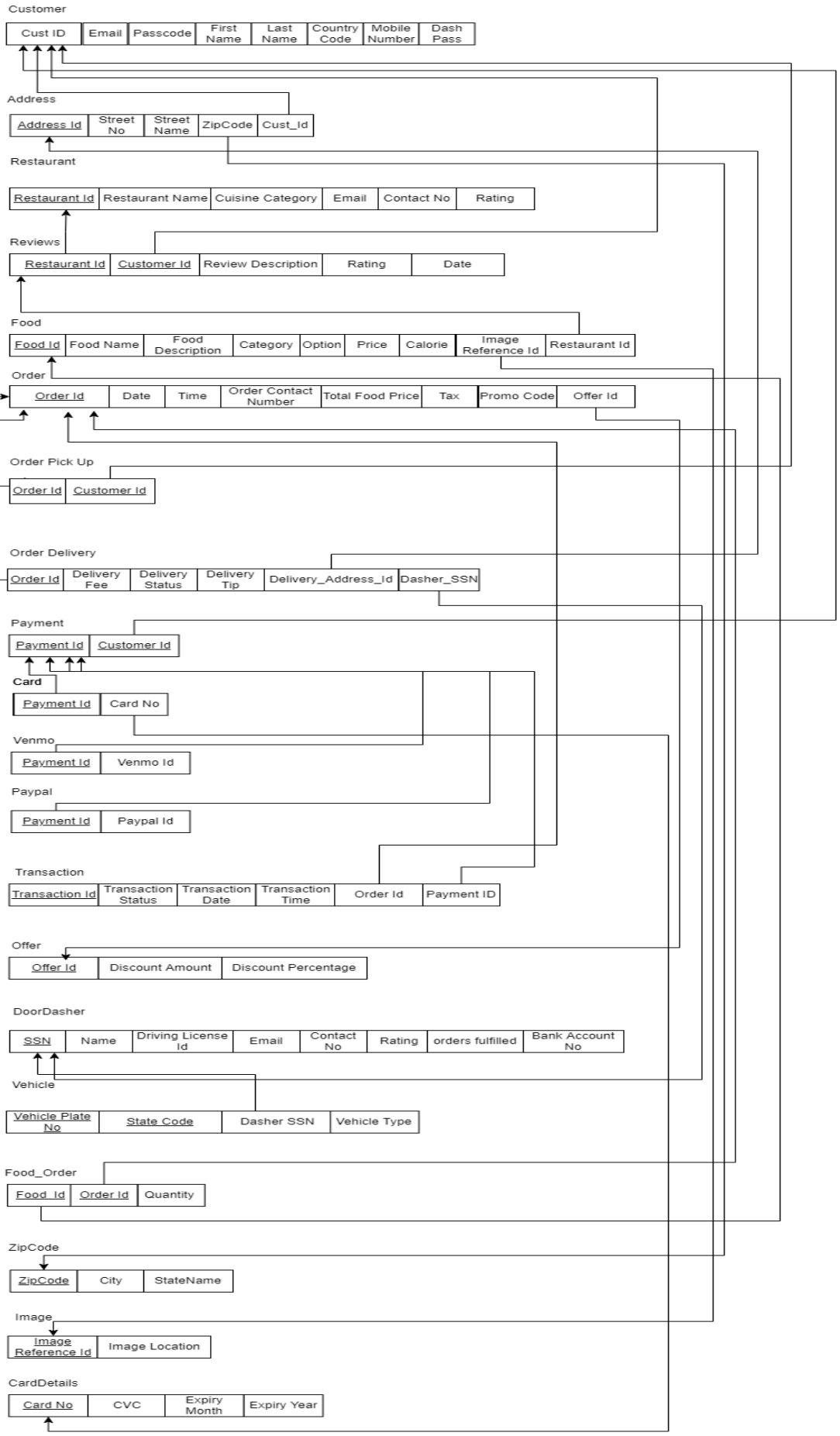
Card

Payment Id	Card No

CardDetails

Card No	CVC	Expiry Month	Expiry Year

Final Relational Schema after Normalization



SQL Queries in creation of tables

```
create table Customer
(
    CustomerId   integer not null,
    Email        varchar(50),
    Passcode     varchar(20) not null,
    FirstName    varchar(30) not null,
    LastName     varchar(30),
    CountryCode  number(2) DEFAULT 0,
    MobileNumber number(10) not null,
    DashPass     varchar(10) DEFAULT 'InActive',
    PRIMARY KEY (CustomerId)
);


create table ZipCode
(
    Zipcode   number(6),
    City      varchar(20) NOT NULL,
    StateName varchar(20),
    PRIMARY KEY (Zipcode)
);


create table Address
(
    AddressId  integer,
    StreetNo   integer,
    StreetName  varchar(50) NOT NULL,
    Zipcode    number(6) NOT NULL,
    CustomerId integer,
    PRIMARY KEY (AddressId),
    FOREIGN KEY (Zipcode) REFERENCES ZipCode (Zipcode) ON DELETE SET NULL,
    FOREIGN KEY (CustomerId) REFERENCES Customer (CustomerId) ON DELETE CASCADE
);

create table Restaurant
(
    RestaurantId    integer,
    RestaurantName  varchar(30) NOT NULL,
    CuisineCategory varchar(20),
    Email           varchar(50),
    ContactNo       number(10) NOT NULL,
    Rating          number (3,2),
    PRIMARY KEY (RestaurantId)
);
```

```
create table Reviews
(
    RestaurantId      integer,
    CustomerId        integer,
    ReviewDescription varchar(500),
    Rating            number(3,2) NOT NULL,
    ReviewDate        date,
    PRIMARY KEY (RestaurantId, CustomerId),
    FOREIGN KEY (RestaurantId) REFERENCES Restaurant (RestaurantId) ON DELETE CASCADE,
    FOREIGN KEY (CustomerId) REFERENCES Customer (CustomerId) ON DELETE CASCADE
);

create table Image
(
    ImageReferenceId integer,
    ImageLocation    varchar(100),
    PRIMARY KEY (ImageReferenceId)
);

create table Food
(
    FoodId           integer,
    FoodName         varchar(20) NOT NULL,
    FoodDescription  varchar(100),
    Category         varchar(20),
    Options          varchar(10),
    Price            number(8,2) NOT NULL,
    Calorie          integer,
    ImageReferenceId integer,
    RestaurantId     integer,
    PRIMARY KEY (FoodId),
    FOREIGN KEY (RestaurantId) REFERENCES Restaurant (RestaurantId) ON DELETE CASCADE,
    FOREIGN KEY (ImageReferenceId) REFERENCES Image (ImageReferenceId) ON DELETE SET NULL
);

create table OrderDetails
(
    OrderId            integer,
    OrderDate          date NOT NULL,
    OrderTime          timestamp NOT NULL,
    OrderContactNumber number(10) NOT NULL,
    Price              number(10,2) NOT NULL,
    Tax                number(10,3) DEFAULT 8.025,
    PromoCode          varchar(30) DEFAULT NULL,
    OfferId            integer DEFAULT NULL,
    PRIMARY KEY (OrderId),
    FOREIGN KEY (OfferId) REFERENCES Offer(OfferId) ON DELETE SET NULL
);
```

```
Create Table OrderPickUp
(
    OrderId    integer,
    CustomerId integer,
    PRIMARY KEY (OrderId, CustomerId),
    FOREIGN KEY (CustomerId) REFERENCES Customer (CustomerId) ON DELETE CASCADE,
    FOREIGN KEY (OrderId) REFERENCES OrderDetails (OrderId) ON DELETE CASCADE
);

create table DoorDasher
(
    SSN               char(9),
    DasherName        varchar(40) NOT NULL,
    DrivingLicenseId  integer NOT NULL,
    Email             varchar(50),
    ContactNumber     number(10) NOT NULL,
    Rating            number(3,2),
    OrdersFulfilled   integer DEFAULT 0,
    BankAccountNumber number(20) NOT NULL,
    PRIMARY KEY (SSN)
);

Create Table OrderDelivery
(
    OrderId           integer,
    DeliveryFee       number(6,2),
    DeliveryStatus    number(1) DEFAULT 0,
    DeliveryTip       number(6,2),
    DeliveryAddressId integer NOT NULL,
    DoorDasherSSN     char(9),
    PRIMARY KEY (OrderId),
    FOREIGN KEY (DeliveryAddressId) REFERENCES Address (AddressId),
    FOREIGN KEY (OrderId) REFERENCES OrderDetails (OrderId) ON DELETE CASCADE,
    FOREIGN KEY (DoorDasherSSN) REFERENCES DoorDasher (SSN) ON DELETE SET NULL
);

create table Payment
(
    PaymentId  integer,
    CustomerId integer,
    PRIMARY KEY (PaymentId),
    FOREIGN KEY (CustomerId) REFERENCES Customer (CustomerId) ON DELETE CASCADE
);

create table CardDetails
(
    CardNo      number(16),
    CVC         number(3) NOT NULL,
    ExpiryMonth varchar(3) NOT NULL,
    ExpiryYear  number(4) NOT NULL,
    PRIMARY KEY (CardNo)
);
```

```
create table Card
(
    PaymentId integer,
    CardNo    number(16) NOT NULL,
    PRIMARY KEY (PaymentId),
    FOREIGN KEY (PaymentId) REFERENCES Payment (PaymentId) ON DELETE CASCADE,
    FOREIGN KEY (CardNo) REFERENCES CardDetails (CardNo) ON DELETE CASCADE
);

create table Venmo
(
    PaymentId integer,
    VenmoId   varchar(30) NOT NULL,
    PRIMARY KEY (PaymentId),
    FOREIGN KEY (PaymentId) REFERENCES Payment (PaymentId) ON DELETE CASCADE
);

create table Paypal
(
    PaymentId integer,
    PayPalId  varchar(30) NOT NULL,
    PRIMARY KEY (PaymentId),
    FOREIGN KEY (PaymentId) REFERENCES Payment (PaymentId) ON DELETE CASCADE
);


create table Transaction
(
    TId       integer,
    TStatus   number(1) NOT NULL,
    TDate     date,
    Ttime     timestamp,
    OrderId   integer,
    PaymentId integer,
    PRIMARY KEY (TId),
    FOREIGN KEY (PaymentId) REFERENCES Payment (PaymentId) ON DELETE CASCADE,
    FOREIGN KEY (OrderId) REFERENCES OrderDetails (OrderId) ON DELETE CASCADE
);


create table Offer
(
    OfferId            integer,
    DiscountAmount     number(4),
    DiscountPercentage number(3) DEFAULT 0.000,
    PRIMARY KEY (OfferId),
);
```

```
create table Vehicle
(
    VehiclePlateNo number(4),
    StateCode      varchar(5) NOT NULL,
    DasherSSN      char(9),
    VehicleType    varchar(10) DEFAULT 'Car',
    PRIMARY KEY (VehiclePlateNo),
    FOREIGN KEY (DasherSSN) references DoorDasher (SSN) ON DELETE CASCADE
);

create table FoodOrder
(
    FoodId  integer,
    OrderId integer,
    PRIMARY KEY (FoodId, OrderId),
    FOREIGN KEY (FoodId) references Food (FoodId) ON DELETE CASCADE,
    FOREIGN KEY (OrderId) REFERENCES OrderDetails (OrderId) ON DELETE CASCADE
);
```

PL SQL – Triggers and Stored Procedures

Stored Procedure 1

To increase the food price of all food items in the restaurant by some percentage during inflation.

```
-- Increasing the food price by percentage of amount
CREATE OR REPLACE PROCEDURE increase_food_price_by_percent(
    rest_id IN RESTAURANT.RESTAURANTID%TYPE,
    percentage IN number
) AS
    thisFood FOOD%ROWTYPE;
    CURSOR FoodCur IS
        SELECT F.*
        FROM RESTAURANT R,
             FOOD F
        WHERE R.RESTAURANTID = rest_id
          AND R.RESTAURANTID = F.RESTAURANTID
            FOR UPDATE;

BEGIN
    OPEN FoodCur;
    LOOP
        FETCH FoodCur INTO thisFood;
        EXIT WHEN (FoodCur%NOTFOUND);
        UPDATE FOOD
        SET PRICE = PRICE * (1 + percentage / 100)
        WHERE FOODID = thisFood.FOODID;
        dbms_output.put_line(thisFood.FOODNAME || ' current price is '
            || thisFood.PRICE);
    END LOOP;
    CLOSE FoodCur;
END;
```

Stored Procedure 2

Finding the faithful customers who have placed more than N orders

---

```
-- Finding the Loyal Customers who placed at least N orders
CREATE TABLE LoyalCustomers
(
    CUSTOMERID   NUMBER,
    ORDERS_COUNT NUMBER
);


CREATE OR REPLACE PROCEDURE find_customers_placed_at_least_n_orders(no_of_orders IN number)
AS
    CURSOR CustomerCur IS
        SELECT O.CUSTOMERID AS CID, COUNT(*) AS ORDERS_COUNT
        FROM CUSTOMER C, ORDERPICKUP O
        WHERE C.CUSTOMERID = O.CUSTOMERID
        GROUP BY O.CUSTOMERID;
    thisCustomer CustomerCur%ROWTYPE;
    orders_count number;
BEGIN
    DELETE FROM LoyalCustomers;
    OPEN CustomerCur;
    LOOP
        FETCH CustomerCur INTO thisCustomer;
        EXIT WHEN (CustomerCur%NOTFOUND);
        orders_count := thisCustomer.ORDERS_COUNT;
        IF orders_count >= no_of_orders THEN
            INSERT INTO LoyalCustomers
            VALUES (thisCustomer.CID, thisCustomer.ORDERS_COUNT);
            dbms_output.put_line(thisCustomer.CID || ' has placed '
                || thisCustomer.ORDERS_COUNT || ' orders');
        END IF;
    END LOOP;
    CLOSE CustomerCur;
END;
```

Trigger 1

Trigger that updates the overall rating of the restaurant when the additional review is provided by averaging all the ratings for the restaurant

---

```
-- Trigger to update the restaurant overall rating when review gets modified or added
CREATE TRIGGER update_restaurant_rating
    AFTER DELETE OR INSERT OR UPDATE OF RATING
    ON REVIEWS
    FOR EACH ROW
DECLARE
    no_of_ratings number;
    total_rating  number;
    new_rating    number;
BEGIN
    /* assume that RATING is non-null field */
    SELECT COUNT(*) INTO no_of_ratings FROM REVIEWS RW WHERE RW.RESTAURANTID = :OLD.RESTAURANTID
;
    SELECT SUM(RATING) INTO total_rating FROM REVIEWS RW WHERE RW.RESTAURANTID = :OLD.RESTAURANT
ID;
    new_rating := (total_rating / no_of_ratings);
    UPDATE RESTAURANT
    SET rating = new_rating
    WHERE RESTAURANTID = :OLD.RESTAURANTID;
END;
```

Another way of declaring triggers-

```
- Trigger to update the restaurant overall rating when review gets modified or added using all c
onditions
CREATE OR REPLACE TRIGGER update_restaurant_rating_v2
    AFTER DELETE OR INSERT OR UPDATE OF RATING
    ON REVIEWS
    FOR EACH ROW
DECLARE
    no_of_ratings  number;
    rating_diff    number;
    curr_rating    number;
    updated_rating number;
BEGIN
    /* assume that RATING is non-null field */
    SELECT COUNT(*) INTO no_of_ratings FROM REVIEWS RW WHERE RW.RESTAURANTID = :OLD.RESTAURANTID
;
    SELECT RATING INTO curr_rating FROM RESTAURANT WHERE RESTAURANTID = :OLD.RESTAURANTID;

    IF DELETING THEN
        updated_rating := (curr_rating * (no_of_ratings + 1) - :OLD.rating) / no_of_ratings;
    END IF;
```

```
    IF INSERTING THEN
         updated_rating := (curr_rating * (no_of_ratings - 1) + :NEW.rating) / no_of_ratings;
     END IF;

    IF UPDATING THEN
         rating_diff := :NEW.RATING - :OLD.RATING;
         updated_rating := (curr_rating * (no_of_ratings) + rating_diff) / no_of_ratings;
     END IF;

     UPDATE RESTAURANT
     SET rating = updated_rating
     WHERE RESTAURANTID = :OLD.RESTAURANTID;
END;
```

Trigger 2

Update the number of orders fulfilled by the door dasher when the delivery is completed

---

```
CREATE OR REPLACE TRIGGER update_fulfilled_orders
    AFTER
        UPDATE OF DeliveryStatus
    ON ORDERDELIVERY
    FOR EACH ROW
DECLARE
    FULFILLED number;
BEGIN
    FULFILLED := 1;
    IF UPDATING AND :new.DeliveryStatus = FULFILLED THEN
        UPDATE DoorDasher DD
        SET OrdersFulfilled = OrdersFulfilled + 1
        WHERE DD.SSN = :new.DoorDasherSSN;
    END IF;
END;
```