



DOORDASH

DoorDash Food Delivery Service

CS-6360:001 - Database Design

Team Number: **31**

Team Members:

1. Manan Dalal (**MUD200000**)
2. Vishesh Mehta (**VJM190001**)
3. Fenil Godhani (**FKG210000**)

Overview:

About DoorDash:

- ⇒ DoorDash is a prepared food delivery service founded in 2013 by Stanford students Tony Xu, Stanley Tang, Andy Fang, and Evan Moore.
- ⇒ A Y combinator-backed company, DoorDash is one of several technology companies that use logistics services to offer food delivery from restaurants on-demand.
- ⇒ DoorDash launched in Palo Alto and, as of May 2019, had expanded to more than 4,000 cities and offers a selection of 340,000 stores across the U.S. and Canada.
- ⇒ The company is currently worth more than \$13 billion and is the largest third-party delivery service in the USA, surpassing Grubhub in 2019.

Why we chose DoorDash?

- ⇒ As the company explains, DoorDash creates opportunities by empowering local businesses and in turn, generating new ways for people to earn, work and live.
- ⇒ The purpose of choosing DoorDash database is to dig deeper into learning how a food delivery service as DoorDash has revolutionized the people's eating habits and standards and has helped generate jobs.
- ⇒ The business model used by DoorDash is intriguing and we wanted to learn about the data requirements of such a database to help us find some meaningful real life database storage problem solving experience.

How does DoorDash Work?

- ⇒ DoorDash has several users that are either customers, dashers or business partners.
- ⇒ Business partners are the restaurant/store owners who partner with DoorDash.
- ⇒ Dashers are the people who work for DoorDash and deliver food door-to-door to the customers.
- ⇒ Once the customers provide their location details, DoorDash displays all the partnered restaurants/stores in a 'x' mile radius from the provided location.
- ⇒ A customer can then search for or filter out restaurants based on various parameters like cuisine, ratings, etc..., add items to his/her cart and place an order.
- ⇒ Once the order is prepared, a dasher picks up the contents and delivers them to the customer's address.
- ⇒ The customer must prepay and can give a review for that order, restaurant or/and the dasher.
- ⇒ The customer can also apply for a refund if he/she finds the order to be unsatisfactory.
- ⇒ The customer and the restaurant can also cancel an order if something goes wrong.

Assumptions

- ⇒ Food is always available between the time a restaurant opens until it closes. That means that we would not check the quantity of the available food.
- ⇒ Customers will be shown restaurants within a particular radius only.
- ⇒ Customers are only allowed to order food from one restaurant at a time. That means that a single order cannot have items from multiple restaurants.

Data/Functional Requirements

Customers:

- ⇒ Create/Update Login and Contact Information
- ⇒ They can search for restaurants based on name, cuisine, menu items etc...
- ⇒ They can add items to their cart and place orders.
- ⇒ Can add instructions and specific requests within the order.
- ⇒ Receive updates and track the status of their order.
- ⇒ Pay, and ask for refund for their order.
- ⇒ Can tip the dasher.
- ⇒ View their order histories.
- ⇒ Can provide feedback/reviews for the order, restaurant or/and the dasher.
- ⇒ Can become a member and enjoy other benefits by buying DashPass.

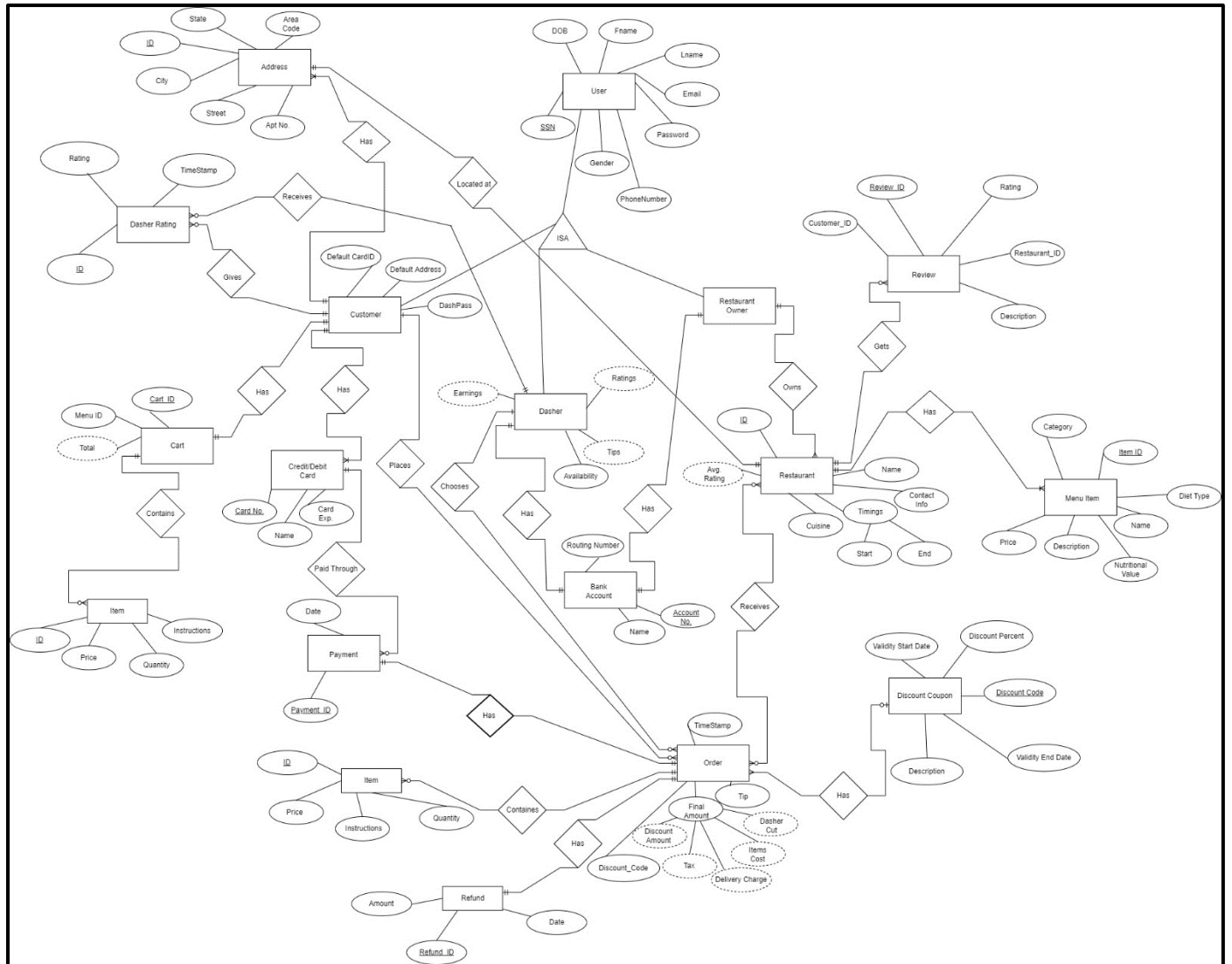
Restaurants:

- ⇒ Create a profile and can add new items/edit existing items to the menu.
- ⇒ They can receive orders, update status of the order and get information of the dasher.
- ⇒ They can discontinue with DoorDash with mutual consensus.
- ⇒ Receive payments from the customer via DoorDash.
- ⇒ Can view insightful data such as most/least items ordered, best/least rated items, user feedbacks etc....
- ⇒ Can add bank information to receive payments.

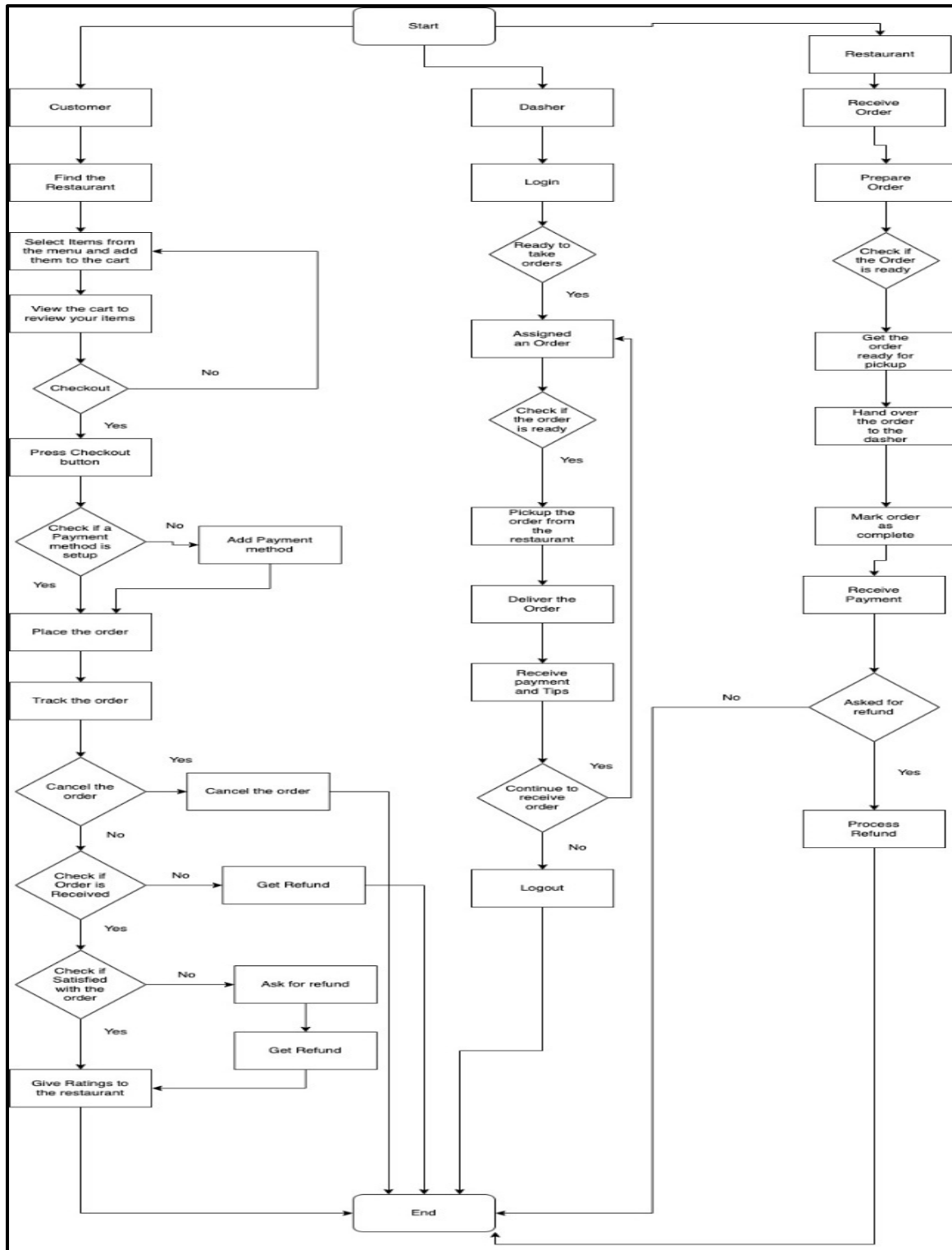
Dashers:

- ⇒ Choose from available orders to pick up around their area/current location.
- ⇒ Know when the order is available for pickup.
- ⇒ Can contact customer/restaurant in case there is discrepancy.
- ⇒ De-Register incase they want to discontinue.
- ⇒ Can receive payments/fees from DoorDash and tips from customers.
- ⇒ Can add bank/wallet information to receive payments.

ER/EER Diagram



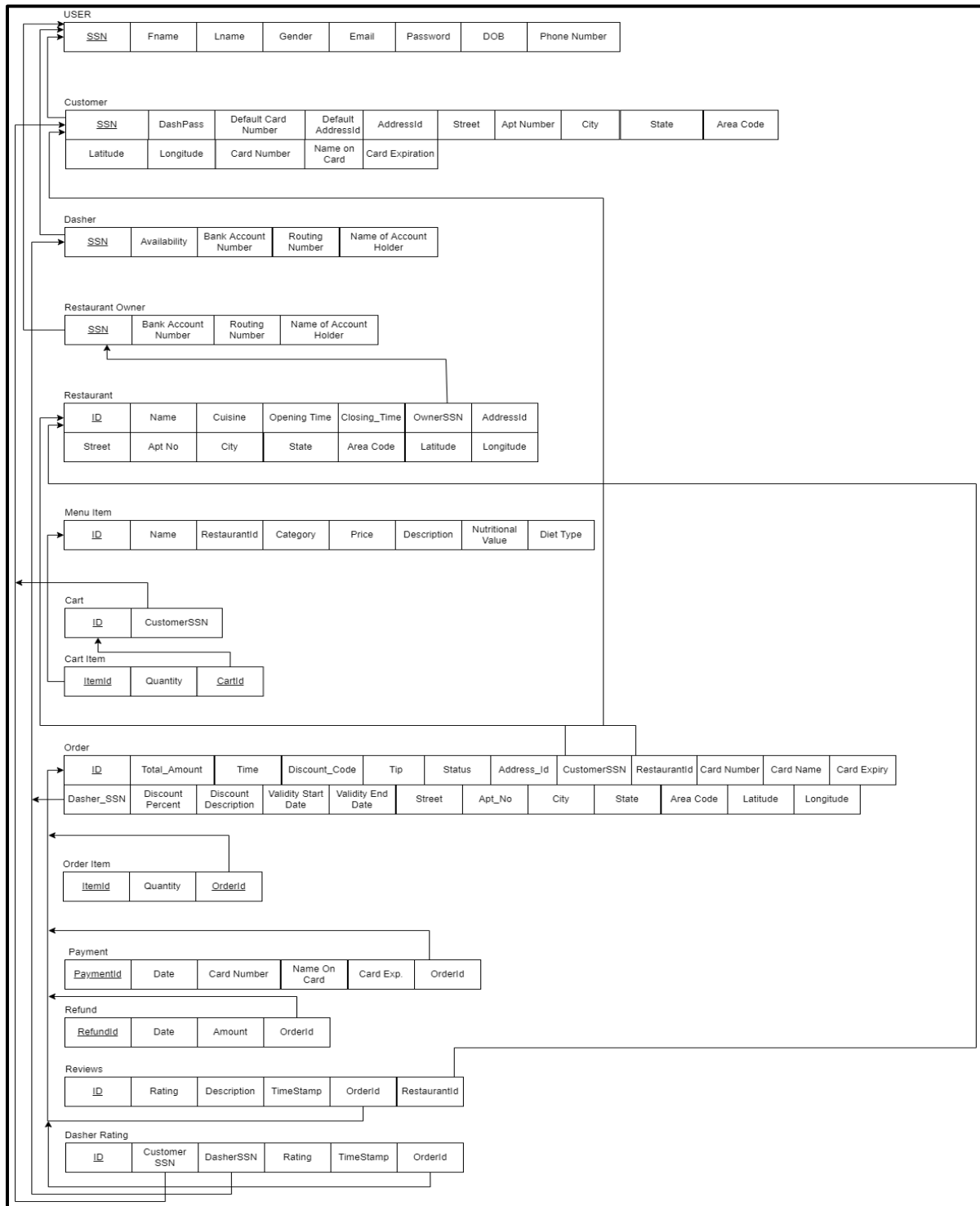
Work-Flow Diagram



Mapping of ER/EER Diagram to Relational Schema

Initial Relational Model

Below is the initial relational model derived from the ER/EER using the proper conversion rules.



Normalizing Initial Schema Design

1-NF: Removing multiple values from the attribute and create another record(entry) to achieve 1-NF.

Example: In the Relating USER, there can be multiple values for phone numbers and emails.

NOT 1-NF

<u>SSN</u>	FName	LName	Gender	Email	Password	DOB	PhoneNumber
123456789	Albert	Einstein	Male	albert@gmail.com einstein@gmail.com	iamalbert	1875-10-20	1234568991
234567890	Isaac	Newton	Male	isaac@gmail.com	lwatchapplesfall	1878-12-10	3048702822 4901890480

1-NF

<u>SSN</u>	FName	LName	Gender	Email	Password	DOB	PhoneNumber
123456789	Albert	Einstein	Male	albert@gmail.com	iamalbert	1875-10-20	1234568991
123456789	Albert	Einstein	Male	einstein@gmail.com	iamalbert	1875-10-20	1234568991
234567890	Isaac	Newton	Male	isaac@gmail.com	lwatchapplesfall	1878-12-10	3048702822
234567890	Isaac	Newton	Male	isaac@gmail.com	lwatchapplesfall	1878-12-10	4901890480

Like the above example, we do this operation for all such values in our schema. Thus, we can say that our database schema relation is now in 2-NF

2-NF: The 2nd Normal Form states that there must be no partial dependencies in any relation. This means that all attributes must solely depend on the entire key and not just on a subset of the key.

- ➔ On examining all our relations, we can observe that the only the relations CART-ITEM and ORDER-ITEM have composite keys.
- ➔ But the non-prime attribute 'quantity' is functionally dependent on the entire key. Hence, we can say that it does not have any partial dependency.
- ➔ All other relations have only one attribute as key and hence for those relations having a partial dependency is not a possibility.

Thus, we can say that our database schema relation is now in 2-NF.

3-NF: The 3rd Normal Form states that there must be no transitive dependencies in any relation. This means that all attributes must solely depend on the key attribute and not on some other non-key attribute.

➔ We can see in the relation CUSTOMER that the attributes such as are functionally dependent 'Street', 'Apt Number', 'City', 'AreaCode', 'State', 'Longitude' and Latitude are dependent on the attribute 'AddressId' and attributes like 'CardExp' and 'NameOnCard' are dependent on 'CardNumber'. This is the very definition of a transitive dependency.

➔ Thus, we can now separate the relation CUSTOMER into three different relations such as:

R1: Customer (SSN, DashPass, Default CardNo, Default AddressId, AddressId, Street, Apt, City, State, Longitude, Latitude, AreaCode, CardNo, NameOnCard, CardExp)

Turns into the following:

R1: Customer (SSN, DashPass, Default CardNo, DefaultAddressId)

R2: Customer_address (AddressId, Street, Apt, City, State, Longitude, Latitude, AreaCode)

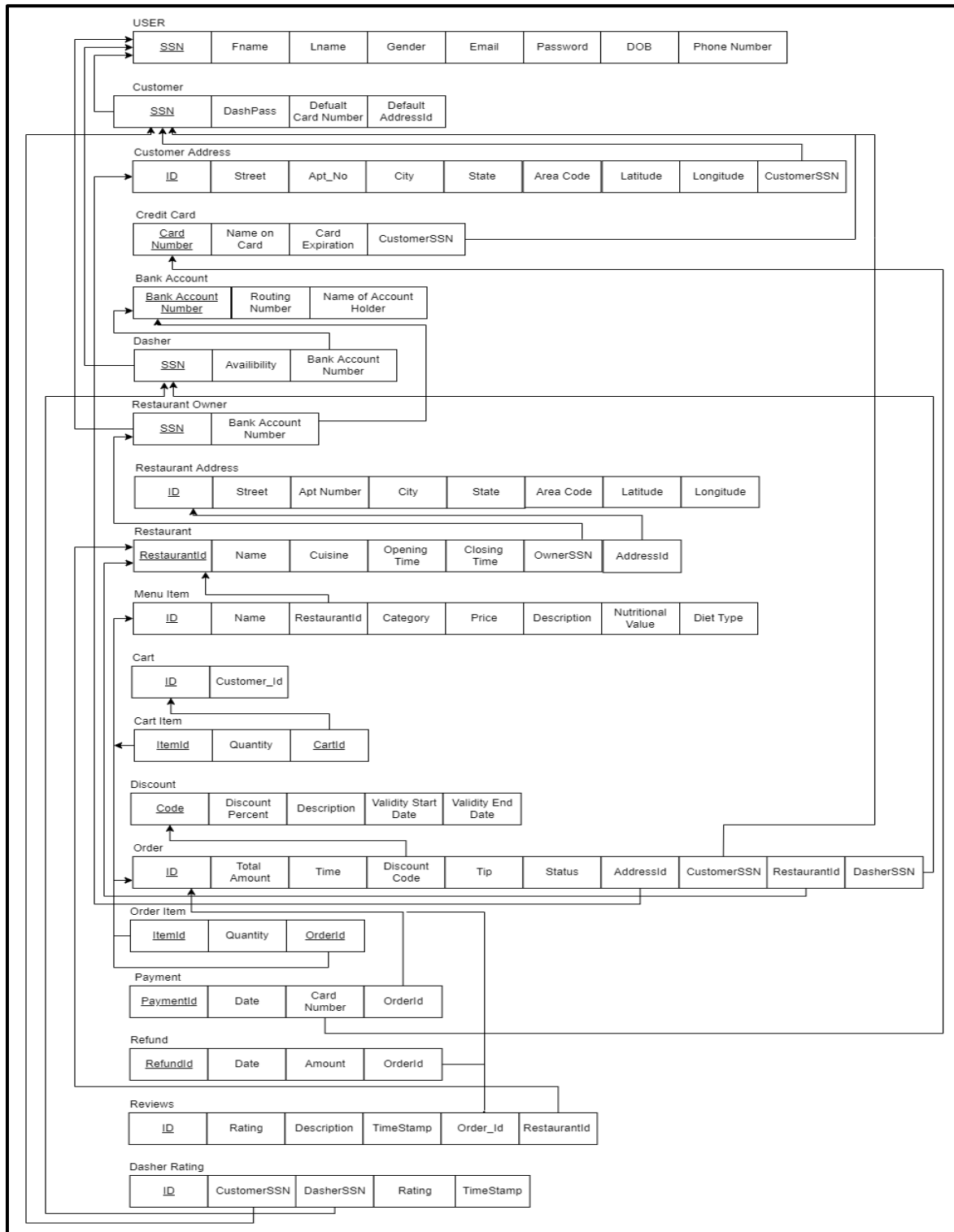
R3: Credit_Card (CardNo, NameOnCard, CardExp)

➔ Similarly, we apply same logic to the other relations having transitive dependencies.

Once all the transitive dependencies are resolved, we can say that our database schema relation is now in 3-NF.

Final Relational Model

After applying normalization, the below diagram is the normalized relational schema of our system.



SQL Statements

Creating Tables

After performing normalization on the original set of tables, the following are all the tables that our system would require to function properly along with their MySQL definitions:

1. User

```
DROP TABLE IF EXISTS `user`;
CREATE TABLE `user` (
  SSN INT NOT NULL,
  FName VARCHAR(15) NOT NULL,
  LName VARCHAR(15) NOT NULL,
  Gender ENUM('Male', 'Female') DEFAULT NULL,
  Email VARCHAR(50) NOT NULL,
  DOB DATE,
  PhoneNumber BIGINT NOT NULL,
  Pwd VARCHAR(20) NOT NULL,
  PRIMARY KEY(SSN),
  CHECK (length(SSN) = 9),
  CHECK (length(PhoneNumber) = 10)
);
```

2. Customer

```
DROP TABLE IF EXISTS customer;
CREATE TABLE customer (
  SSN INT NOT NULL,
  DashPass BOOLEAN NOT NULL DEFAULT 0,
  DefaultCardId INT,
  DefaultAddressId INT,
  PRIMARY KEY(SSN),
  FOREIGN KEY (SSN) REFERENCES `user` (SSN) ON DELETE CASCADE,
  CHECK (length(SSN) = 9)
);
```

3. Customer Address

```
DROP TABLE IF EXISTS customer_address;
CREATE TABLE customer_address (
  AddressId INT AUTO_INCREMENT NOT NULL,
  StreetName VARCHAR(50) NOT NULL,
  AptNumber INT,
  City VARCHAR(20) NOT NULL,
  State VARCHAR(20) NOT NULL,
  AreaCode INT NOT NULL,
  Latitude INT,
  Longitude INT,
  CustomerSSN INT NOT NULL,
  PRIMARY KEY(AddressId),
  FOREIGN KEY (CustomerSSN) REFERENCES Customer(SSN),
  CHECK (length(AreaCode) = 5),
  CHECK (length(CustomerSSN) = 9)
);
```

4. Credit Card

```
DROP TABLE IF EXISTS credit_card;
CREATE TABLE credit_card (
    CardNumber INT AUTO_INCREMENT,
    NameOnCard VARCHAR(40) NOT NULL,
    CardExp DATE,
    CustomerSSN INT,
    PRIMARY KEY(CardNumber),
    FOREIGN KEY (CustomerSSN) REFERENCES customer(SSN),
    CHECK (length(CustomerSSN) = 9)
);
```

5. Bank Account

```
DROP TABLE IF EXISTS bank_account;
CREATE TABLE bank_account (
    BankAccountNumber INT NOT NULL,
    RoutingNumber INT NOT NULL,
    AccountHolderName VARCHAR(50) NOT NULL,
    PRIMARY KEY(BankAccountNumber)
);
```

6. Dasher

```
DROP TABLE IF EXISTS dasher;
CREATE TABLE dasher (
    SSN INT NOT NULL,
    Availability BOOLEAN NOT NULL DEFAULT 0,
    BankAccountNumber INT NOT NULL,
    PRIMARY KEY(SSN),
    FOREIGN KEY (SSN) REFERENCES `user` (SSN) ON DELETE CASCADE,
    FOREIGN KEY (BankAccountNumber) REFERENCES bank_account(BankAccountNumber) ON DELETE CASCADE,
    CHECK (length(SSN) = 9)
);
```

7. Restaurant Owner

```
DROP TABLE IF EXISTS restaurant_owner;
CREATE TABLE restaurant_owner (
    SSN INT NOT NULL,
    BankAccountNumber INT NOT NULL,
    PRIMARY KEY(SSN),
    FOREIGN KEY (SSN) REFERENCES `user` (SSN) ON DELETE CASCADE,
    FOREIGN KEY (BankAccountNumber) REFERENCES bank_account(BankAccountNumber) ON DELETE CASCADE,
    CHECK (length(SSN) = 9)
);
```

8. Restaurant Address

```
DROP TABLE IF EXISTS restaurant_address;
CREATE TABLE restaurant_address (
    AddressId INT AUTO_INCREMENT NOT NULL,
    StreetName VARCHAR(50) NOT NULL,
    AptNumber INT,
    City VARCHAR(20) NOT NULL,
    State VARCHAR(20) NOT NULL,
    AreaCode INT NOT NULL,
    Latitude INT,
    Longitude INT,
    PRIMARY KEY(AddressId),
    CHECK (length(AreaCode) = 5)
);
```

9. Restaurant

```
DROP TABLE IF EXISTS restaurant;
CREATE TABLE restaurant (
    RestaurantId INT AUTO_INCREMENT NOT NULL,
    RestaurantName VARCHAR(20) NOT NULL,
    Cuisine VARCHAR(20),
    OpeningTime TIME,
    ClosingTime TIME,
    OwnerSSN INT NOT NULL,
    AddressId INT NOT NULL,
    PRIMARY KEY(RestaurantId),
    FOREIGN KEY (OwnerSSN) REFERENCES restaurant_owner(SSN),
    FOREIGN KEY (AddressId) REFERENCES restaurant_address(AddressId) ON DELETE CASCADE,
    CHECK (length(OwnerSSN) = 9)
);
```

10. Menu Item

```
DROP TABLE IF EXISTS menu_item;
CREATE TABLE menu_item (
    ItemId INT AUTO_INCREMENT,
    RestaurantId INT NOT NULL,
    `Name` VARCHAR(20) NOT NULL,
    Category VARCHAR(25),
    Price DOUBLE NOT NULL,
    `Description` VARCHAR(100),
    NutritionalValue INT,
    DietType VARCHAR(20),
    PRIMARY KEY(ItemId),
    FOREIGN KEY (RestaurantId) REFERENCES Restaurant(RestaurantId)
);
```

11. Cart

```
DROP TABLE IF EXISTS cart;
CREATE TABLE cart (
    CartId INT,
    CustomerSSN INT,
    PRIMARY KEY(CartId),
    FOREIGN KEY (CustomerSSN) REFERENCES Customer(SSN),
    CHECK (length(CustomerSSN) = 9)
);
```

12. Cart Item

```
DROP TABLE IF EXISTS cart_item;
CREATE TABLE cart_item (
    ItemId INT NOT NULL,
    CartId INT NOT NULL,
    Quantity INT NOT NULL,
    PRIMARY KEY(ItemId, CartId),
    FOREIGN KEY (ItemId) REFERENCES menu_item(ItemId),
    FOREIGN KEY (CartId) REFERENCES cart(CartId)
);
```

13. Discount Coupon

```
DROP TABLE IF EXISTS discount_coupon;
CREATE TABLE discount_coupon (
    CouponCode VARCHAR(10) NOT NULL,
    DiscountPercentage INT NOT NULL,
    `Description` VARCHAR(100),
    ValidityStart DATE NOT NULL,
    ValidityEnd DATE NOT NULL,
    PRIMARY KEY(CouponCode),
    CHECK (DiscountPercentage < 100)
);
```

14. Order

```
DROP TABLE IF EXISTS `order`;
CREATE TABLE `order` (
    OrderId INT AUTO_INCREMENT NOT NULL,
    TotalAmount DOUBLE NOT NULL,
    `TimeStamp` TIMESTAMP NOT NULL,
    DicountCode VARCHAR(10),
    TipAmount DOUBLE NOT NULL DEFAULT 0,
    OrderStatus enum("placed", "accepted", "prepared", "picked up", "delivered", "refunded") NOT NULL,
    AddressId INT NOT NULL,
    CardNumber INT NOT NULL,
    CustomerSSN INT NOT NULL,
    RestaurantId INT NOT NULL,
    DasherSSN INT,
```

```
    PRIMARY KEY(OrderId),
    FOREIGN KEY (DicountCode) REFERENCES discount_coupon(CouponCode),
    FOREIGN KEY (AddressId) REFERENCES customer_address(AddressId),
    FOREIGN KEY (CardNumber) REFERENCES credit_card(CardNumber),
    FOREIGN KEY (CustomerSSN) REFERENCES customer(SSN),
    FOREIGN KEY (DasherSSN) REFERENCES dasher(SSN),
    FOREIGN KEY (RestaurantId) REFERENCES restaurant(RestaurantId),
    CHECK (length(DasherSSN) = 9),
    CHECK (length(CustomerSSN) = 9)
);
```

15. Order Item

```
DROP TABLE IF EXISTS order_item;
CREATE TABLE order_item (
    ItemId INT NOT NULL,
    OrderId INT NOT NULL,
    Quantity INT NOT NULL,
    PRIMARY KEY(ItemId, OrderId),
    FOREIGN KEY (ItemId) REFERENCES menu_item(ItemId),
    FOREIGN KEY (OrderId) REFERENCES `order`(OrderId)
);
```

16. Payment

```
DROP TABLE IF EXISTS payment;
CREATE TABLE payment (
    PaymentId INT AUTO_INCREMENT NOT NULL,
    `Date` TIMESTAMP,
    CardNumber INT,
    OrderId INT,
    PRIMARY KEY(PaymentId),
    FOREIGN KEY (CardNumber) REFERENCES credit_card(CardNumber),
    FOREIGN KEY (OrderId) REFERENCES `order`(OrderId)
);
```

17. Refund

```
DROP TABLE IF EXISTS refund;
CREATE TABLE refund (
    RefundId INT AUTO_INCREMENT NOT NULL,
    `Date` TIMESTAMP NOT NULL,
    Amount DOUBLE NOT NULL,
    OrderId INT NOT NULL,
    PRIMARY KEY(RefundId),
    FOREIGN KEY (OrderId) REFERENCES `order`(OrderId)
);
```

18. Review

```
DROP TABLE IF EXISTS review;
CREATE TABLE review (
    ReviewId INT AUTO_INCREMENT NOT NULL,
    Rating DOUBLE,
    `Description` VARCHAR(100),
    `TimeStamp` TIMESTAMP NOT NULL,
    OrderId INT NOT NULL,
    RestaurantId INT,
    PRIMARY KEY(ReviewId),
    FOREIGN KEY (OrderId) REFERENCES `order`(OrderId),
    FOREIGN KEY (RestaurantId) REFERENCES restaurant(RestaurantId)
);
```

19. Dasher Rating

```
DROP TABLE IF EXISTS dasher_rating;
CREATE TABLE dasher_rating (
    Id INT AUTO_INCREMENT,
    CustomerSSN INT,
    DasherSSN INT,
    Rating DOUBLE,
    `TimeStamp` TIMESTAMP,
    PRIMARY KEY(Id),
    FOREIGN KEY (CustomerSSN) REFERENCES customer(SSN),
    FOREIGN KEY (DasherSSN) REFERENCES dasher(SSN),
    CHECK (length(DasherSSN) = 9),
    CHECK (length(CustomerSSN) = 9)
);
```


Inserting Data

As shown below, here are some sample insertion operations on all the tables of the database system.

```
INSERT into `user` values ('123456789','Vishesh','Mehta','Male','mvishesh42@gmail.com','1997-11-16','4699274591',MD5('vishesh'));
INSERT into `user` values ('234567891','Manan','Dalal','Male','md10@gmail.com','1998-12-10','7489271894',MD5('manan'));
INSERT into `user` values ('456789123','Rohan','Prasad','Male','rp007@gmail.com','1999-05-03','5683271894',MD5('rohan'));

INSERT into customer (SSN, DashPass) values ('123456789','1');

INSERT into customer_address values (0,'955 W President George Bush Hwy','5303','Richardson','Texas','75080','101','211','123456789');

INSERT into credit_card values ('11001','Vishesh Mehta','2025-08-26', 123456789);

INSERT into bank_account values ('110000001','1111000','Manan Dalal');
INSERT into bank_account values ('110000003','1111002','Rohan Prasad');

INSERT into dasher values ('456789123','1','110000003');

INSERT into restaurant_owner values ('234567891','110000001');

INSERT into restaurant_address values (0,'1012 Lincoln Street',null,'Richardson','Texas','75080','101','112');
```

```
INSERT into restaurant values (0,'Crunch','Indian','08:00:00','22:00:00','234567891',1);

INSERT into menu_item values (0, 1, 'Paneer Burger','Burgers','9.99','Spicy Indian Burger','180','Vegeterian');
INSERT into menu_item values (0, 1, 'Paneer Makhni Pizza','Pizza','12.99','Spicy Indian Pizza','240','Vegeterian');
INSERT into menu_item values (0, 1, 'Chicken Tikka Pizza','Pizza','15.99','Spicy Indian Pizza','290','Non-Vegeterian');

INSERT into cart values ('1','123456789');

INSERT into cart_item values (1,1,2);

INSERT into discount_coupon values ('BC12','10','This is a coupon code for 10 percent discount','2022-10-02','2022-10-10');
```

```
INSERT into `order` values (0, 30.52, TIMESTAMP("2022-03-23", "13:10:11"),'BC12','5','placed',1,11001,123456789,1,345678912);
INSERT into `order` values (0, 84.50, TIMESTAMP("2022-03-28", "12:15:11"),null,'5','delivered',2,11001,123456789,8,456789123);

INSERT into order_item values (1,1,1);
INSERT into order_item values (2,1,1);

INSERT into payment values (0,TIMESTAMP("2022-03-23", "13:09:11"),11001,1);
INSERT into payment values (0,TIMESTAMP("2022-03-23", "13:09:11"),11001,2);

INSERT into refund values (0,TIMESTAMP("2022-03-23", "14:09:11"),15,2);

INSERT into review values (0,4.5,'Great Food',TIMESTAMP("2022-03-23", "14:09:11"),1, 1);
INSERT into review values (0,4,'Decent',TIMESTAMP("2022-03-23", "15:09:11"),2, 1);

INSERT into dasher_rating values (1,123456789,345678912,4.0,TIMESTAMP("2022-03-23", "15:09:11"));
INSERT into dasher_rating values (2,123456789,345678912,3.0,TIMESTAMP("2022-03-23", "15:09:11"));
```

Insights into Data

1. To find average rating of all restaurants.

```
SELECT r.RestaurantName as "Restaurant Name", AVG(re.Rating) AS "Average Rating"
FROM review re
INNER JOIN restaurant r ON r.RestaurantId = re.RestaurantId
GROUP BY re.RestaurantId;
```

Restaurant Name	Average Rating
Crunch	4.375
Taco Bell	3.5
Olive Garden	4.333333333333333
Bay leaf	3.8
Subway	3.1666666666666665

2. To check how many restaurant owners own more than 1 restaurant.

```
SELECT ro.SSN as "Owner SSN", concat(u.FName, ' ', u.LName) as "Owner Name"
FROM restaurant_owner ro
INNER JOIN `user` u ON u.SSN = ro.SSN
WHERE ro.SSN NOT IN (
    SELECT OwnerSSN
    FROM restaurant
    GROUP BY OwnerSSN
    HAVING count(*) = 1
);
```

Owner SSN	Owner Name
234567891	Manan Dalal
507891234	Darshil Dhanani

3. To check how many users have a DashPass and how many don't.

```
SELECT COUNT(*) as "No of Customers", DashPass
FROM customer
GROUP BY DashPass
ORDER BY DashPass;
```

Result Grid			Filter Rows:
	No of Customers	DashPass	
▶	1	0	
	3	1	

4. To find the restaurants who get the maximum orders.

```
SELECT o.RestaurantId, r.RestaurantName, COUNT(*) as "No of Orders"
FROM `order` o
INNER JOIN restaurant r
ON o.RestaurantId = r.RestaurantId
GROUP BY o.RestaurantId
ORDER BY count(o.RestaurantId) DESC;
```

Result Grid				Filter Rows:	Ex
	RestaurantId	RestaurantName	No of Orders		
▶	2	Bay leaf	5		
	1	Crunch	3		
	3	Subway	3		
	8	Olive Garden	3		
	7	Taco Bell	2		

5. To find out how many times customers are asking for refund.

```
SELECT count(o.OrderId) AS "Refund Requests", o.CustomerSSN, concat(u.Fname, ' ', u.Lname) AS "Customer Name"
FROM customer c
INNER JOIN `user` u
ON u.SSN = c.SSN
LEFT OUTER JOIN `order` o
ON c.SSN = o.CustomerSSN
WHERE OrderId IN (
    SELECT OrderId
    FROM refund
)
GROUP BY o.CustomerSSN
ORDER BY count(o.OrderId) DESC
```

Result Grid			
		Filter Rows:	Export:
	Refund Requests	CustomerSSN	Customer Name
▶	3	567891234	Lipi Patel
	2	767891234	Harin Desai
	1	123456789	Vishesh Mehta

Views

1. Creating a view that shows the total number of restaurants in different cities.

```
CREATE VIEW restuarants_per_city AS
SELECT count(*) AS "Number of Restaurants", City
FROM restaurant_address
GROUP BY City;




SELECT * FROM restuarants_per_city;
```

Result Grid	Filter Rows:	Export
Number of Restaurants	City	
3	Richardson	
2	Brooklyn	

2. To visualize all important information of a customer.

```
CREATE VIEW customer_info AS
SELECT u.SSN, concat(u.Fname, ' ', u.Lname) AS "Name", u.Email, u.PhoneNumber, c.DashPass,
       cc.CardNumber, cc.CardExp, ca.StreetName, ca.AptNumber, ca.City, ca.AreaCode
FROM `user` u
INNER JOIN customer c ON u.SSN = c.SSN
INNER JOIN credit_card cc ON cc.CustomerSSN = c.SSN
INNER JOIN customer_address ca ON ca.CustomerSSN = c.SSN;

SELECT * FROM customer_info;
```

Result Grid	 Filter Rows:	Export: 	Wrap Cell Content: 							
SSN	Name	Email	PhoneNumber	DashPass	CardNumber	CardExp	StreetName	AptNumber	City	AreaCode
123456789	Vishesh Mehta	mvishesh42@gmail.com	4699274591	1	11001	2025-08-26	955 W President George Bush Hwy	5303	Richardson	75080
123456789	Vishesh Mehta	mvishesh42@gmail.com	4699274591	1	11001	2025-08-26	955 W President George Bush Hwy	4301	Richardson	75080
123456789	Vishesh Mehta	mvishesh42@gmail.com	4699274591	1	11002	2023-05-21	955 W President George Bush Hwy	5303	Richardson	75080
123456789	Vishesh Mehta	mvishesh42@gmail.com	4699274591	1	11002	2023-05-21	955 W President George Bush Hwy	4301	Richardson	75080
567891234	Lipi Patel	lpi05@gmail.com	1289321894	0	11003	2024-02-01	955 W President George Bush Hwy	6231	Austin	75100
567891234	Lipi Patel	lpi05@gmail.com	1289321894	0	11003	2024-02-01	918 E Larry Road	1110	Brooklyn	81001
567891235	Divy Patel	divy@gmail.com	1289321855	1	11006	2024-03-22	121 Mdaren Street	1102	Brooklyn	81001
567891235	Divy Patel	divy@gmail.com	1289321855	1	11006	2024-03-22	881 Richard Ave	1104	Richardson	75080
767891234	Harin Desai	harin@gmail.com	1289551894	1	11004	2025-08-10	121 Mdaren Street	1101	Brooklyn	81001
767891234	Harin Desai	harin@gmail.com	1289551894	1	11005	2025-04-28	121 Mdaren Street	1101	Brooklyn	81001

3. To get all important details of an order.

```
CREATE VIEW order_info AS
SELECT o.OrderId, o.TotalAmount, o.OrderStatus, o.RestaurantId, r.RestaurantName, o.CardNumber, cc.CardExp,
       o.DasherSSN, concat(u.Fname, ' ', u.LName) AS "Dasher Name", o.CustomerSSN, concat(uc.Fname, ' ', uc.LName) AS "Customer Name",
       ca.StreetName, ca.AptNumber, ca.City, ca.State, ca.AreaCode, re.RefundId, re.Amount
FROM `order` o
INNER JOIN restaurant r ON o.RestaurantId = r.RestaurantId
INNER JOIN credit_card cc ON o.CardNumber = cc.CardNumber
INNER JOIN `user` u ON o.DasherSSN = u.SSN
INNER JOIN `user` uc ON o.CustomerSSN = uc.SSN
INNER JOIN customer_address ca ON o.AddressId = ca.AddressId
LEFT OUTER JOIN refund re ON o.OrderId = re.OrderId
ORDER BY o.OrderId;

SELECT * FROM order_info;
```

OrderId	TotalAmount	OrderStatus	RestaurantId	RestaurantName	CardNumber	CardExp	DasherSSN	Dasher Name	CustomerSSN	Customer Name	StreetName	AptNumber	City	State	AreaCode	RefundId	Amount
1	30.52	refunded	1	Crunch	10001	2023-08-26	345678912	Pingu Patel	123456789	Vishesh Mehta	955 W President George Bush Hwy		Richardson	Texas	75080	1	15
2	21.52	delivered	1	Crunch	10001	2023-08-26	345678912	Pingu Patel	123456789	Vishesh Mehta	955 W President George Bush Hwy		Richardson	Texas	75080		
3	52.1	delivered	7	Taco Bell	10002	2023-05-21	345678912	Pingu Patel	123456789	Vishesh Mehta	955 W President George Bush Hwy		Richardson	Texas	75080		
4	84.5	delivered	8	Olive Garden	10001	2023-08-26	456789123	Rohan Prasad	123456789	Vishesh Mehta	955 W President George Bush Hwy		Richardson	Texas	75080		
5	99.12	delivered	2	Bay leaf	10003	2024-02-01	569891234	Nahit Nakrani	567891234	Lip Patel	955 W President George Bush Hwy		Austin	Texas	75100		
6	90.52	delivered	3	Subway	10003	2024-02-01	569891234	Nahit Nakrani	567891234	Lip Patel	955 W President George Bush Hwy		Austin	Texas	75100	3	10.11
7	78.52	delivered	2	Bay leaf	10003	2024-02-01	569891234	Nahit Nakrani	567891234	Lip Patel	955 W President George Bush Hwy		Austin	Texas	75100	4	22.26
8	23.52	delivered	2	Bay leaf	10003	2024-02-01	569891234	Nahit Nakrani	567891234	Lip Patel	955 W President George Bush Hwy		Austin	Texas	75100	5	3.5
9	51.52	delivered	2	Bay leaf	10004	2023-08-10	569891234	Nahit Nakrani	767891234	Harin Desai	121 Midaren Street		Brooklyn	New York	81001		
10	41.31	delivered	3	Subway	10005	2023-04-28	569891234	Nahit Nakrani	767891234	Harin Desai	121 Midaren Street		Brooklyn	New York	81001		
11	48.93	delivered	2	Bay leaf	10005	2023-04-28	569891234	Nahit Nakrani	767891234	Harin Desai	121 Midaren Street		Brooklyn	New York	81001	6	28.5
12	78.77	delivered	3	Subway	10004	2023-08-10	569891234	Nahit Nakrani	767891234	Harin Desai	121 Midaren Street		Brooklyn	New York	81001	7	10.2
13	43.2	delivered	1	Crunch	10006	2024-03-22	456789123	Rohan Prasad	567891235	Divy Patel	881 Richard Ave		Richardson	Texas	75080		
14	44.51	delivered	7	Taco Bell	10006	2024-03-22	345678912	Pingu Patel	567891235	Divy Patel	881 Richard Ave		Richardson	Texas	75080		

4. To find out how well each dasher is performing based on their average rating.

```
CREATE VIEW average_dasher_rating AS
SELECT dr.DasherSSN, concat(u.Fname, ' ', u.LName) as "Name", avg(Rating) AS "Average Rating"
FROM dasher_rating dr
INNER JOIN `user` u ON u.SSN = dr.DasherSSN
GROUP BY dr.DasherSSN
ORDER BY avg(Rating) DESC;

SELECT * FROM average_dasher_rating;
```

DasherSSN	Name	Average Rating
456789123	Rohan Prasad	4.166666666666667
345678912	Pingu Patel	3.8333333333333335
569891234	Nishit Nakrani	3.375

PL/SQL Statements

Triggers

1. Creating a trigger that can satisfy the condition that “An order cannot be deleted if it has not been delivered or refunded”

```
DELIMITER //
CREATE TRIGGER delete_pending
BEFORE DELETE
on `order`
FOR EACH ROW
BEGIN
    IF OLD.OrderStatus != 'delivered' OR OLD.OrderStatus != "refunded"
    THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot delete an incomplete order.', MYSQL_ERRNO = 1001;
    END IF;
END //
DELIMITER ;
```

Result Grid		Filter Rows:	Edit
	OrderId	OrderStatus	
	20	prepared	
▶*	NULL	NULL	

35	13:42:41	DELETE FROM `order` WHERE OrderId = 20	Error Code: 1001. Cannot delete an incomplete order.	0.000 sec
----	----------	--	--	-----------

2. Creating a trigger that can satisfy the condition that “A customer can only have items from a single restaurant in his cart at a single time. If he adds an item from another restaurant, then the cart should be emptied first.”

```
DELIMITER //
CREATE TRIGGER adding_cart_item
BEFORE INSERT
ON cart_item
FOR EACH ROW
BEGIN
    DECLARE new_restaurant_id INTEGER;
    DECLARE new_cart_id INTEGER;
    DECLARE old_restaurant_id INTEGER;
    DECLARE old_item_id INTEGER;
    SET @new_restaurant_id := (SELECT RestaurantId FROM menu_item WHERE ItemId = NEW.ItemId);
    SET @old_item_id := (SELECT ItemId FROM cart_item WHERE CartId = NEW.CartId LIMIT 1);
    SET @old_restaurant_id := (SELECT RestaurantId FROM menu_item WHERE ItemId = @old_item_id);

    IF @old_restaurant_id != @new_restaurant_id THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot add item from another restaurant', MYSQL_ERRNO = 1001;
    END IF;
END //
DELIMITER ;
```

36 13:43:41 insert into cart_item values (6, 2, 2)

Error Code: 1001. Cannot add item from another restaurant

0.016 sec

Stored Procedures

1. A Customer can search for restaurants using filters like cuisine, city etc..

```
DELIMITER //
CREATE PROCEDURE Restaurant_Search(IN cuisine1 VARCHAR(30), IN city1 varchar(20))
BEGIN
    SELECT *
    FROM restaurant r
    WHERE Cuisine = cuisine1 AND (
        SELECT City
        FROM restaurant_address
        WHERE AddressId = r.AddressId) = city1;
END //
DELIMITER ;

CALL Restaurant_Search('Indian', 'Brooklyn');
```

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	RestaurantId	RestaurantName	Cuisine	OpeningTime	ClosingTime	OwnerSSN	AddressId
	2	Bay leaf	Indian	08:00:00	22:00:00	907891234	2

2. DoorDash can find out the earnings of each restaurant between a particular period.

```
DELIMITER //
CREATE PROCEDURE Total_Earnings(IN startDate TimeStamp, IN endDate TimeStamp)
BEGIN
    SELECT RestaurantId, SUM(TotalAmount) AS "Total Earnings", startDate, endDate
    FROM `order`
    WHERE `TimeStamp` BETWEEN startDate AND endDate
    GROUP BY RestaurantId
    ORDER BY SUM(TotalAmount) DESC;
END //
DELIMITER ;

CALL Total_Earnings(TIMESTAMP("2022-02-23", "15:09:11"), TIMESTAMP("2022-04-24", "15:09:11"));
```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
RestaurantId	Total Earnings	startDate	endDate
2	261.61	2022-02-23 15:09:11	2022-04-24 15:09:11
8	232.97	2022-02-23 15:09:11	2022-04-24 15:09:11
3	170.60000000000002	2022-02-23 15:09:11	2022-04-24 15:09:11
7	96.61	2022-02-23 15:09:11	2022-04-24 15:09:11
1	95.24000000000001	2022-02-23 15:09:11	2022-04-24 15:09:11

3. A Restaurant can change the status of an order as it gets processed and completed.

```
DELIMITER //
```

```
CREATE PROCEDURE Change_Order_Status (IN OrderId1 INT, IN newStatus VARCHAR(20))
```

```
BEGIN
```

```
    UPDATE `order`
```

```
    SET OrderStatus = newStatus
```

```
    WHERE OrderId = OrderId1;
```

```
END //
```

```
DELIMITER ;
```



```
select OrderId, OrderStatus from `order` where OrderId = 1;
```

```
CALL Change_Order_Status(1, 'refunded');
```

```
select OrderId, OrderStatus from `order` where OrderId = 1;
```

Result Grid			Filter Rows:	
	OrderId	OrderStatus		
▶	1	delivered		
*	NULL	NULL		

Before

Result Grid			Filter Rows:	
	OrderId	OrderStatus		
▶	1	refunded		
*	NULL	NULL		

After