# DBMS ASSIGNMENT

Nikki Gautam

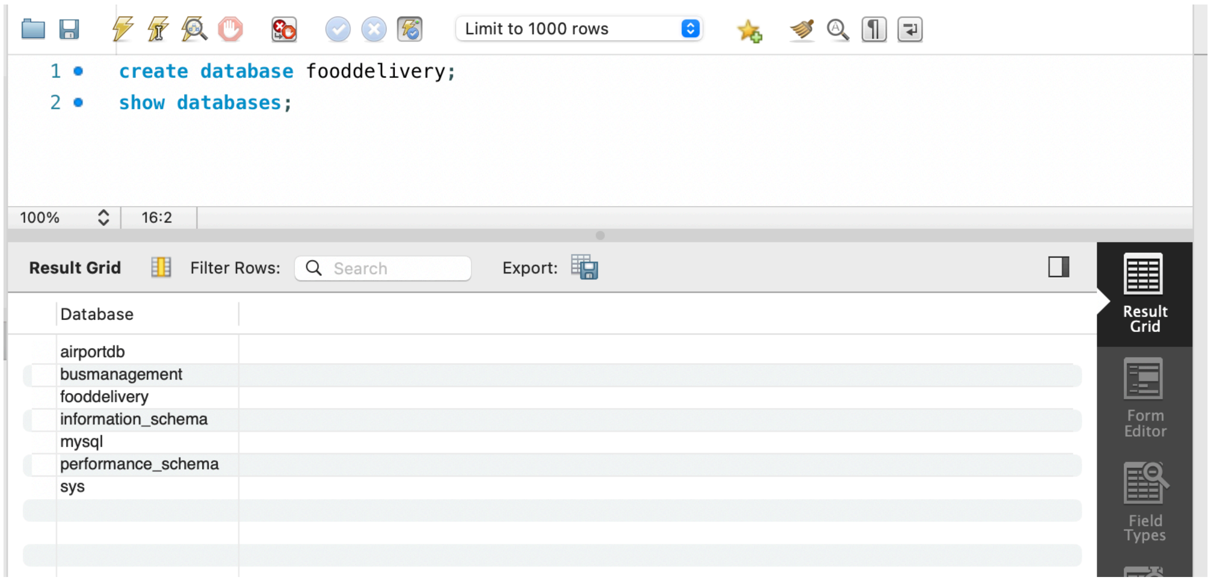
## Question 2:

Language Used (for querying): **SQL**

## Part A:

For the ER Model created in the earlier question, create the database tables, and normalize them. You are free to modify the original ER diagram with new findings / changes. You are also free to leave some tables in a denormalized state if you can justify it in the final report (part C).

## Database Created: fooddelivery



```
create database fooddelivery;
show databases;
```

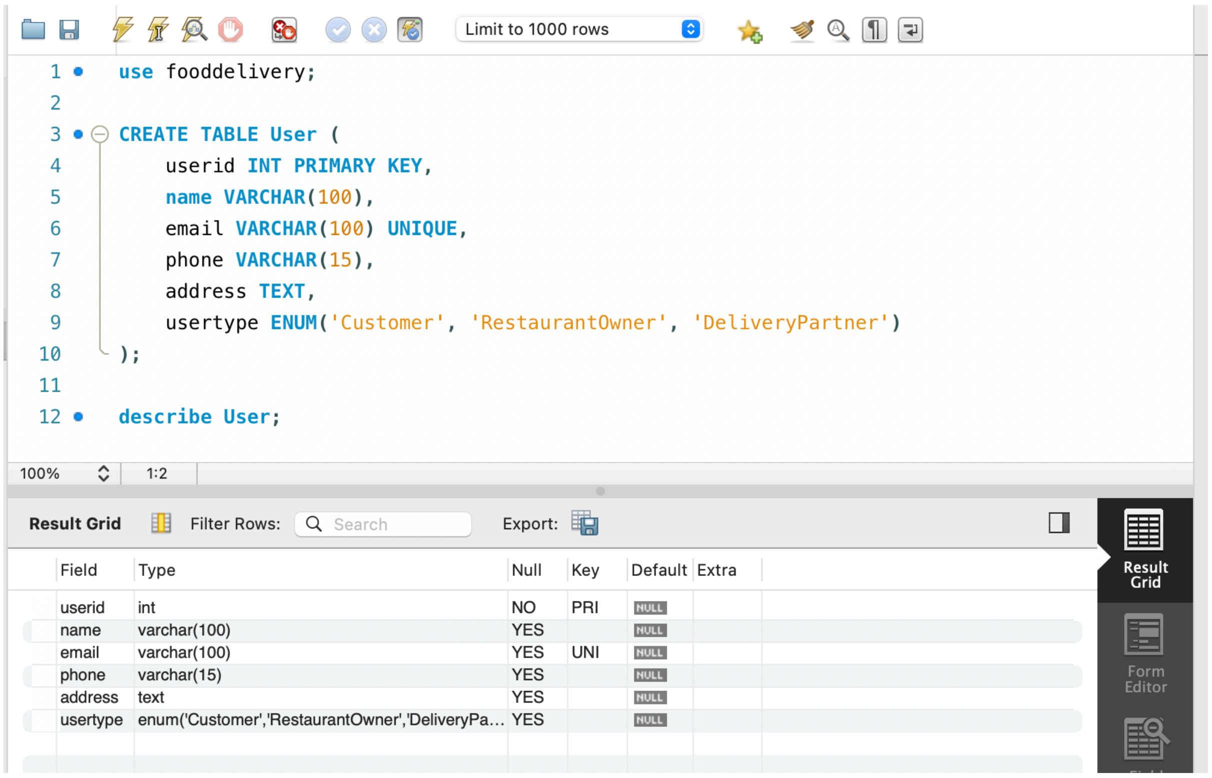
```
Database
airportdb
busmanagement
fooddelivery
information_schema
mysql
performance_schema
sys
```

## Database Tables:

### 1. User table

Used to store information of the users who interact with the system

```
CREATE TABLE User (
    userid INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(15),
    address TEXT,
    usertype ENUM('Customer', 'RestaurantOwner',
'DeliveryPartner')
);
```



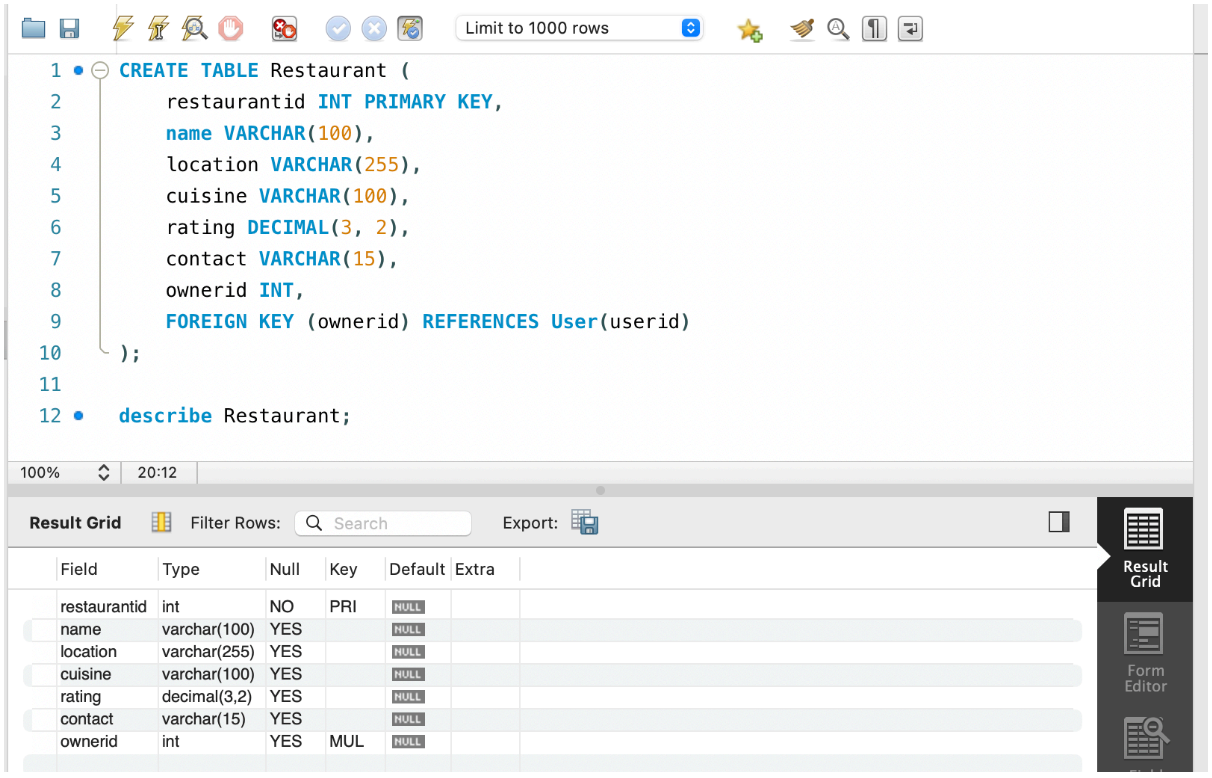
```
use fooddelivery;

CREATE TABLE User (
    userid INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(15),
    address TEXT,
    usertype ENUM('Customer', 'RestaurantOwner', 'DeliveryPartner')
);

describe User;
```

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| userid | int | NO | PRI | NULL | |
| name | varchar(100) | YES | | NULL | |
| email | varchar(100) | YES | UNI | NULL | |
| phone | varchar(15) | YES | | NULL | |
| address | text | YES | | NULL | |
| usertype | enum('Customer','RestaurantOwner','DeliveryPa... | YES | | NULL | |

## 2. Restaurant table

Used to store information of the Restaurant in consideration

```
CREATE TABLE Restaurant (
    restaurantid INT PRIMARY KEY,
    name VARCHAR(100),
    location VARCHAR(255),
    cuisine VARCHAR(100),
    rating DECIMAL(3, 2),
    contact VARCHAR(15),
    ownerid INT,
    FOREIGN KEY (ownerid) REFERENCES User(userid)
);
```



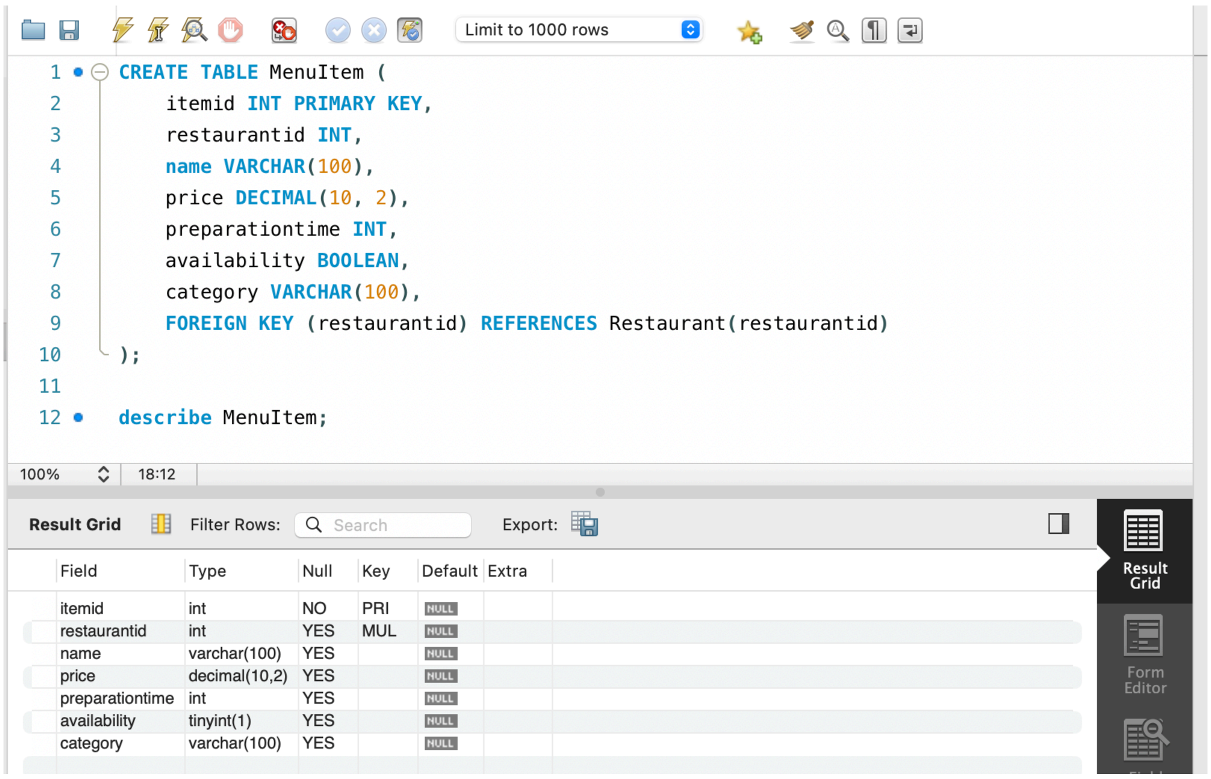
```
CREATE TABLE Restaurant (
    restaurantid INT PRIMARY KEY,
    name VARCHAR(100),
    location VARCHAR(255),
    cuisine VARCHAR(100),
    rating DECIMAL(3, 2),
    contact VARCHAR(15),
    ownerid INT,
    FOREIGN KEY (ownerid) REFERENCES User(userid)
);

describe Restaurant;
```

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| restaurantid | int | NO | PRI | NULL | |
| name | varchar(100) | YES | | NULL | |
| location | varchar(255) | YES | | NULL | |
| cuisine | varchar(100) | YES | | NULL | |
| rating | decimal(3,2) | YES | | NULL | |
| contact | varchar(15) | YES | | NULL | |
| ownerid | int | YES | MUL | NULL | |

## 3. MenuItem table

Used to store details of dishes available to the customers

```
CREATE TABLE MenuItem (
    itemid INT PRIMARY KEY,
    restaurantid INT,
    name VARCHAR(100),
    price DECIMAL(10, 2),
    preparationtime INT,
    availability BOOLEAN,
    category VARCHAR(100),
    FOREIGN KEY (restaurantid) REFERENCES
Restaurant(restaurantid)
);
```



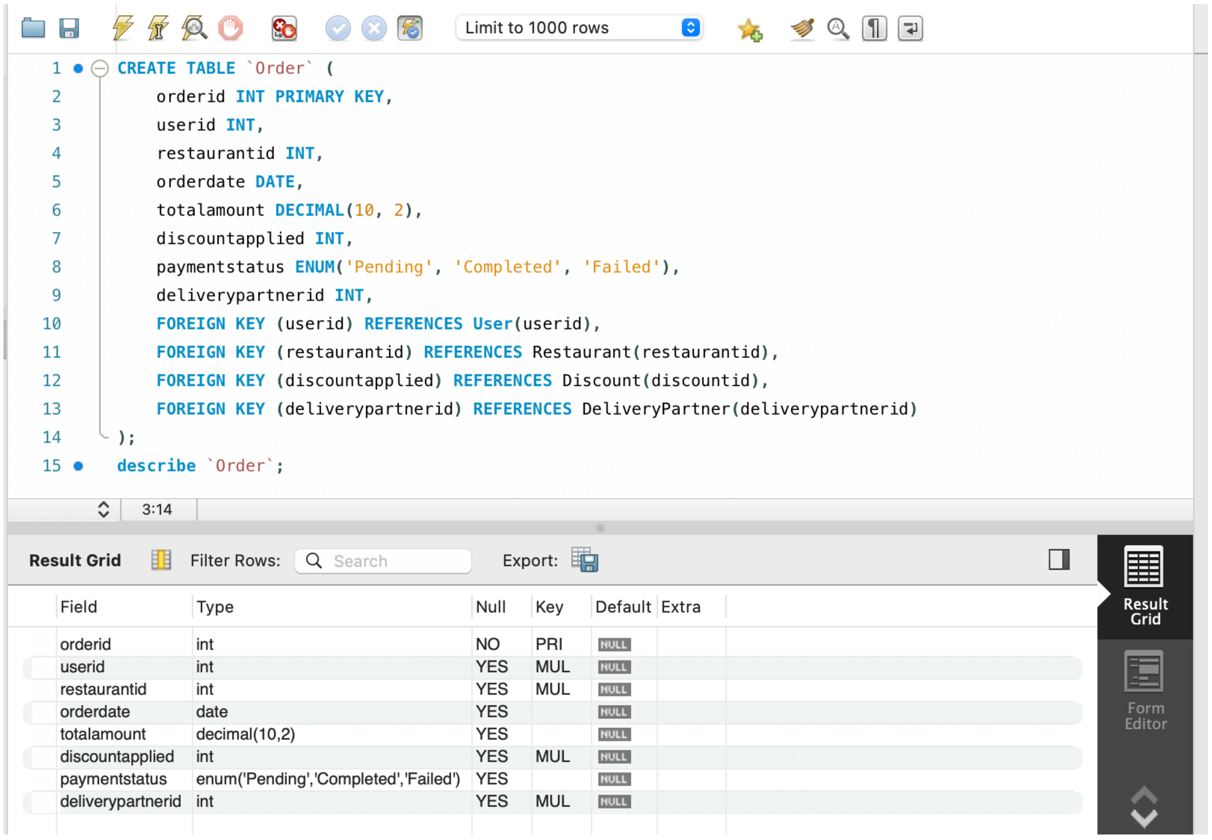
```
CREATE TABLE MenuItem (
    itemid INT PRIMARY KEY,
    restaurantid INT,
    name VARCHAR(100),
    price DECIMAL(10, 2),
    preparationtime INT,
    availability BOOLEAN,
    category VARCHAR(100),
    FOREIGN KEY (restaurantid) REFERENCES Restaurant(restaurantid)
);

describe MenuItem;
```

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| itemid | int | NO | PRI | NULL | |
| restaurantid | int | YES | MUL | NULL | |
| name | varchar(100) | YES | | NULL | |
| price | decimal(10,2) | YES | | NULL | |
| preparationtime | int | YES | | NULL | |
| availability | tinyint(1) | YES | | NULL | |
| category | varchar(100) | YES | | NULL | |

## 4. Order table

Used to store information corresponding to orders registered

```
CREATE TABLE `Order` (
    orderid INT PRIMARY KEY,
    userid INT,
    restaurantid INT,
    orderdate DATE,
    totalamount DECIMAL(10, 2),
    discountapplied INT,
    paymentstatus ENUM('Pending', 'Completed',
'Failed'),
    deliverypartnerid INT,
    FOREIGN KEY (userid) REFERENCES User(userid),
    FOREIGN KEY (restaurantid) REFERENCES
Restaurant(restaurantid),
    FOREIGN KEY (discountapplied) REFERENCES
Discount(discountid),
    FOREIGN KEY (deliverypartnerid) REFERENCES
DeliveryPartner(deliverypartnerid)
);
```



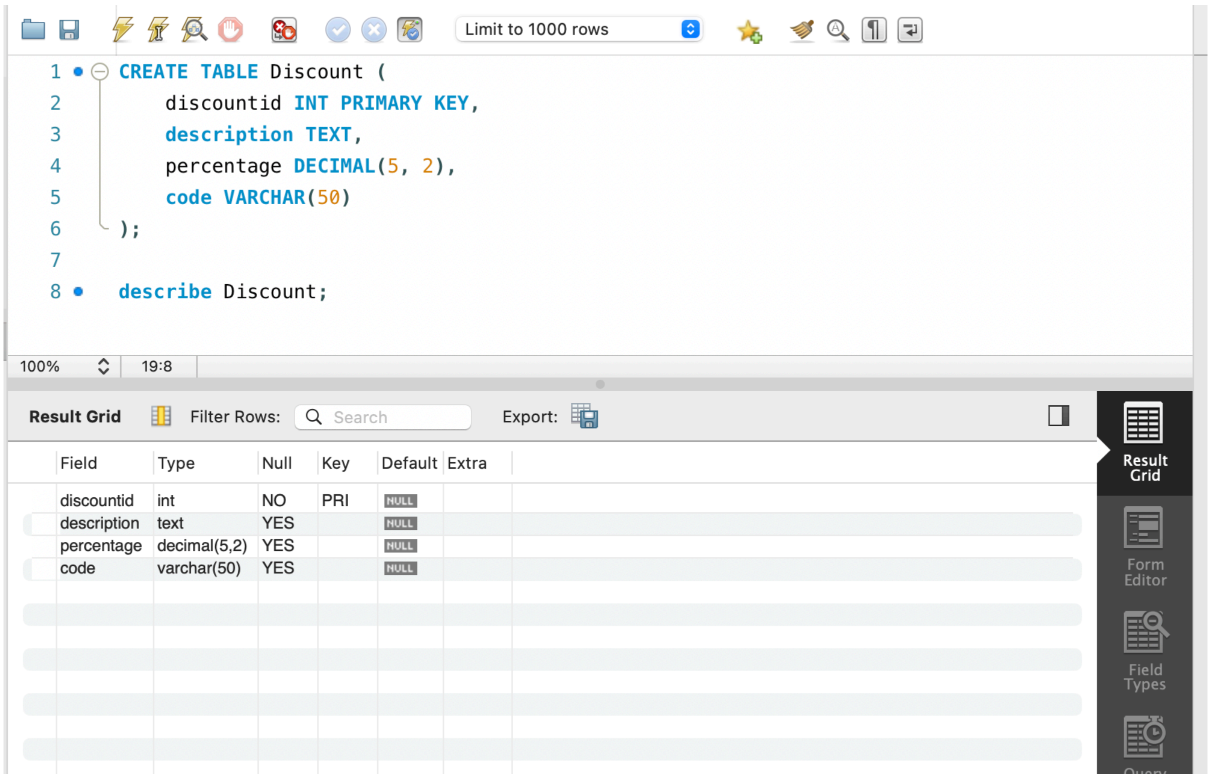
```
CREATE TABLE `Order` (
    orderid INT PRIMARY KEY,
    userid INT,
    restaurantid INT,
    orderdate DATE,
    totalamount DECIMAL(10, 2),
    discountapplied INT,
    paymentstatus ENUM('Pending', 'Completed', 'Failed'),
    deliverypartnerid INT,
    FOREIGN KEY (userid) REFERENCES User(userid),
    FOREIGN KEY (restaurantid) REFERENCES Restaurant(restaurantid),
    FOREIGN KEY (discountapplied) REFERENCES Discount(discountid),
    FOREIGN KEY (deliverypartnerid) REFERENCES DeliveryPartner(deliverypartnerid)
);
describe `Order`;
```

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| orderid | int | NO | PRI | NULL | |
| userid | int | YES | MUL | NULL | |
| restaurantid | int | YES | MUL | NULL | |
| orderdate | date | YES | | NULL | |
| totalamount | decimal(10,2) | YES | | NULL | |
| discountapplied | int | YES | MUL | NULL | |
| paymentstatus | enum('Pending','Completed','Failed') | YES | | NULL | |
| deliverypartnerid | int | YES | MUL | NULL | |

## 5. Discount table

Used to store information regarding discounts available to the customers

```
CREATE TABLE Discount (
    discountid INT PRIMARY KEY,
    description TEXT,
    percentage DECIMAL(5, 2),
    code VARCHAR(50)
);
```



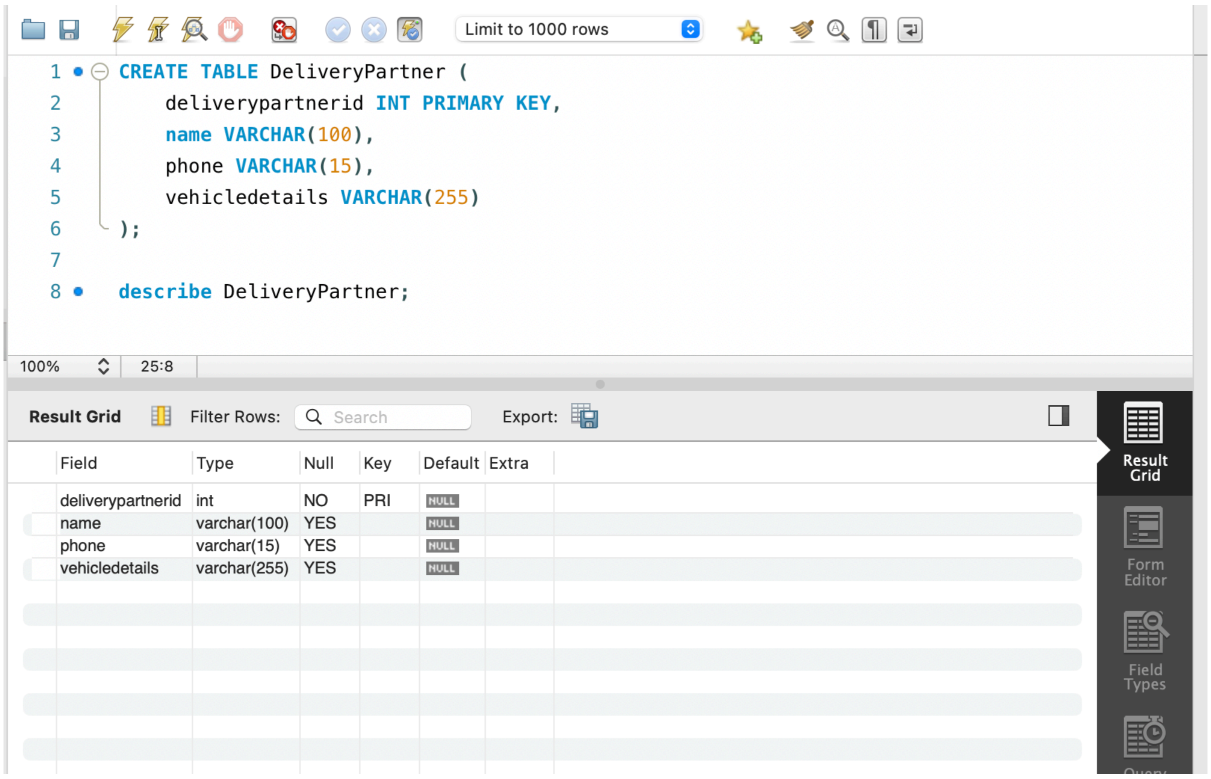
```
CREATE TABLE Discount (
    discountid INT PRIMARY KEY,
    description TEXT,
    percentage DECIMAL(5, 2),
    code VARCHAR(50)
);

describe Discount;
```

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| discountid | int | NO | PRI | NULL | |
| description | text | YES | | NULL | |
| percentage | decimal(5,2) | YES | | NULL | |
| code | varchar(50) | YES | | NULL | |

## 6. DeliveryPartner table

Used to represent delivery partner information in association with the restaurant

```
CREATE TABLE DeliveryPartner (
    deliverypartnerid INT PRIMARY KEY,
    name VARCHAR(100),
    phone VARCHAR(15),
    vehicledetails VARCHAR(255)
);
```



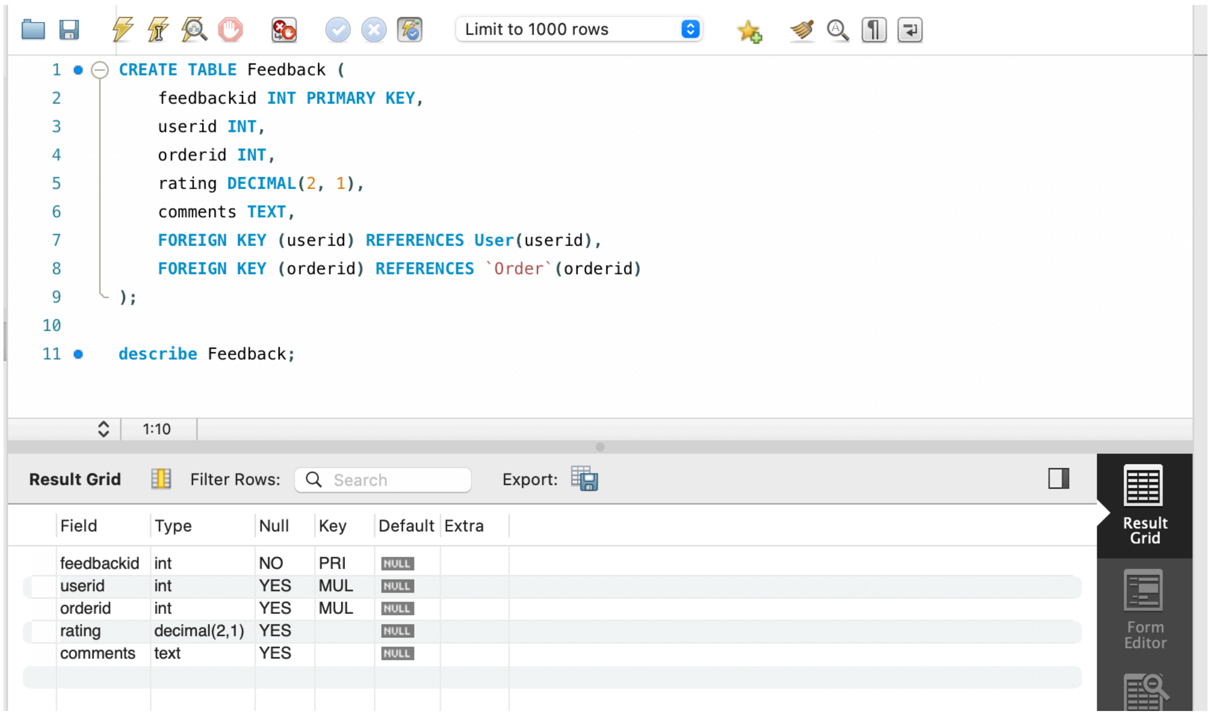
```
CREATE TABLE DeliveryPartner (
    deliverypartnerid INT PRIMARY KEY,
    name VARCHAR(100),
    phone VARCHAR(15),
    vehicledetails VARCHAR(255)
);

describe DeliveryPartner;
```

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| deliverypartnerid | int | NO | PRI | NULL | |
| name | varchar(100) | YES | | NULL | |
| phone | varchar(15) | YES | | NULL | |
| vehicledetails | varchar(255) | YES | | NULL | |

## 7. Feedback table

Used to keep a record of customer feedback corresponding to the orders

```
CREATE TABLE Feedback (
    feedbackid INT PRIMARY KEY,
    userid INT,
    orderid INT,
    rating DECIMAL(2, 1),
    comments TEXT,
    FOREIGN KEY (userid) REFERENCES User(userid),
    FOREIGN KEY (orderid) REFERENCES `Order`(orderid)
);
```



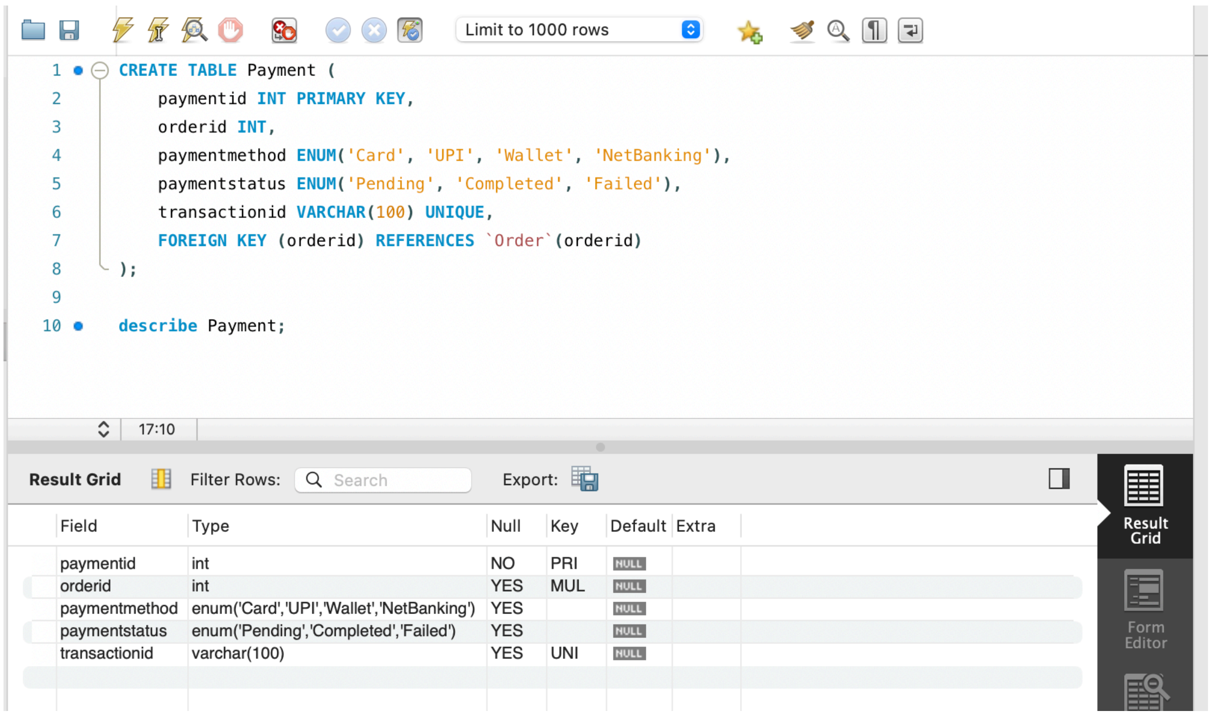
```
CREATE TABLE Feedback (
    feedbackid INT PRIMARY KEY,
    userid INT,
    orderid INT,
    rating DECIMAL(2, 1),
    comments TEXT,
    FOREIGN KEY (userid) REFERENCES User(userid),
    FOREIGN KEY (orderid) REFERENCES `Order`(orderid)
);

describe Feedback;
```

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| feedbackid | int | NO | PRI | NULL | |
| userid | int | YES | MUL | NULL | |
| orderid | int | YES | MUL | NULL | |
| rating | decimal(2,1) | YES | | NULL | |
| comments | text | YES | | NULL | |

## 8. Payment table

Used to store information of payments made

```
CREATE TABLE Payment (
    paymentid INT PRIMARY KEY,
    orderid INT,
    paymentmethod ENUM('Card', 'UPI', 'Wallet',
'NetBanking'),
    paymentstatus ENUM('Pending', 'Completed',
'Failed'),
    transactionid VARCHAR(100) UNIQUE,
    FOREIGN KEY (orderid) REFERENCES `Order`(orderid)
);
```



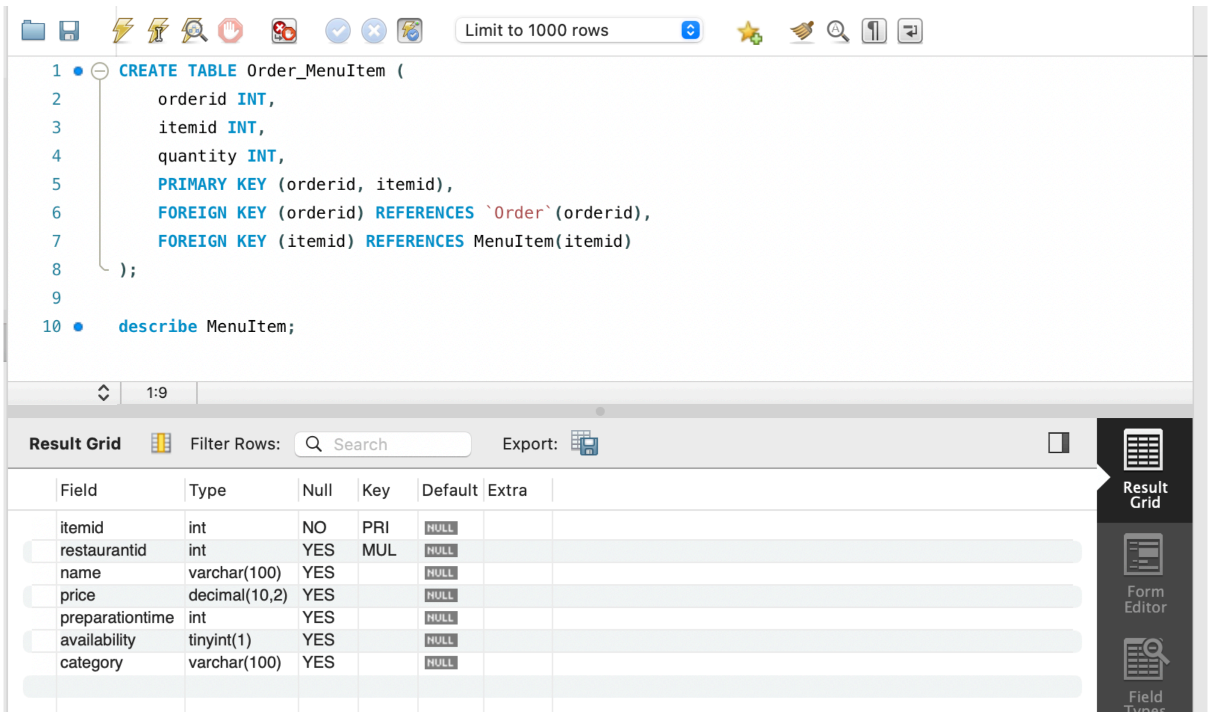
```
CREATE TABLE Payment (
    paymentid INT PRIMARY KEY,
    orderid INT,
    paymentmethod ENUM('Card', 'UPI', 'Wallet', 'NetBanking'),
    paymentstatus ENUM('Pending', 'Completed', 'Failed'),
    transactionid VARCHAR(100) UNIQUE,
    FOREIGN KEY (orderid) REFERENCES `Order`(orderid)
);

describe Payment;
```

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| paymentid | int | NO | PRI | NULL | |
| orderid | int | YES | MUL | NULL | |
| paymentmethod | enum('Card','UPI','Wallet','NetBanking') | YES | | NULL | |
| paymentstatus | enum('Pending','Completed','Failed') | YES | | NULL | |
| transactionid | varchar(100) | YES | UNI | NULL | |

## 9. Order_MenuItem table

Used to store information of orders places by customers

```
CREATE TABLE Order_MenuItem (
    orderid INT,
    itemid INT,
    quantity INT,
    PRIMARY KEY (orderid, itemid),
    FOREIGN KEY (orderid) REFERENCES
`Order`(orderid),
    FOREIGN KEY (itemid) REFERENCES MenuItem(itemid)
);
```



```
CREATE TABLE Order_MenuItem (
    orderid INT,
    itemid INT,
    quantity INT,
    PRIMARY KEY (orderid, itemid),
    FOREIGN KEY (orderid) REFERENCES `Order`(orderid),
    FOREIGN KEY (itemid) REFERENCES MenuItem(itemid)
);

describe MenuItem;
```

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| itemid | int | NO | PRI | NULL | |
| restaurantid | int | YES | MUL | NULL | |
| name | varchar(100) | YES | | NULL | |
| price | decimal(10,2) | YES | | NULL | |
| preparationtime | int | YES | | NULL | |
| availability | tinyint(1) | YES | | NULL | |
| category | varchar(100) | YES | | NULL | |

## **Normalization (**Applied in the solution proposed**):**

### 1. User (Already in 3NF)

- Each column is functionally dependent on the primary key (userid)
- There are no transitive dependencies

### 2. Restaurant (Already in 3NF)

- Each column is functionally dependent on the primary key (restaurantid)
- The foreign key "**ownerid**" references "**userid**" in the "**User**" table

### 3. MenuItem

- **1NF**: All attributes contain atomic values
- **2NF**: All non-key attributes are fully dependent on the primary key (itemid)
- **3NF**: No transitive dependencies

### 4. Order (LEFT IN DENORMALIZED STATE)

- Including "**discountapplied**" and "**deliverypartnerid**" in this table avoids the need for extra joins while fetching data related to an order's discount and delivery

### 5. Discount (Already in 3NF)

- Each column is functionally dependent on the primary key (discountid)
- No transitive dependencies

### 6. DeliveryPartner (Already in 3NF)

- Each column is functionally dependent on the primary key (deliverypartnerid)

### 7. Feedback (Already in 3NF)

- Each column is functionally dependent on the primary key (feedbackid)
- Foreign keys "**userid**" and "**ordered**" ensure referential integrity

**8. Payment (Already in 3NF)**

- Each column is functionally dependent on the primary key (paymentid)
- Foreign key "**ordered**" ensures referential integrity

**9. Order_MenuItem (Bridge Table) (Already in 3NF)**

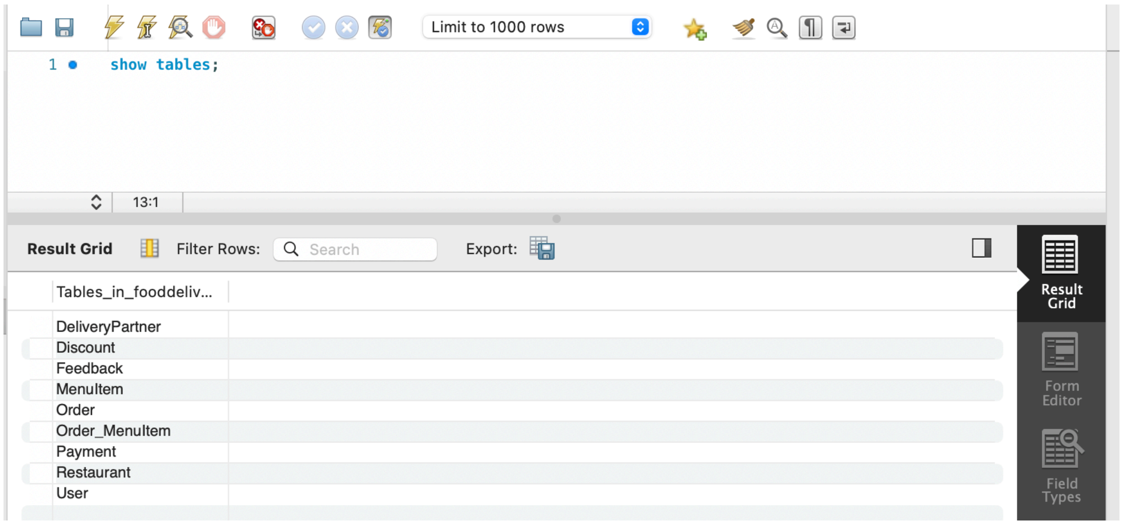
- **Purpose**: Represents the M:N relationship between "**Order**" and "**MenuItem**"
- Each column is fully dependent on the composite primary key (orderid, itemid)

## Denormalized Tables (Justification for keeping them as they are):

1. **Order**:
   - Attributes like discountapplied and deliverypartnerid are included to simplify queries related to discounts and delivery partners
   - Reducing joins improves performance for frequently accessed data

2. **Order_MenuItem**:
   - Maintains normalization while efficiently handling the M:N relationship

Final glimpse of "**fooddelivery**":



```
show tables;
```

| Tables_in_fooddeliv... |
|---|
| DeliveryPartner |
| Discount |
| Feedback |
| MenuItem |
| Order |
| Order_MenuItem |
| Payment |
| Restaurant |
| User |