CSU 44D01 Project

Online Food Ordering System for Restaurant

Submitted By – Mihir Shekhar
Student ID – 21355037

Contents of Project Report

➢ INTRODUCTION :

In this project, the idea is to develop an ER Model for information to be represented for an application of our OWN choice and implement it as a MySQL database using the concepts we studied in the CSU 44D01 module.

This is the database required for Online Food ordering System without having to go to the restaurant. It gives the platform to global food ordering system search marketplace. Designing a database for an Online Food Ordering System for a restaurant involves structuring and organizing data to efficiently store and retrieve information related to the restaurant's operations.

When it comes to online food ordering system, the data that we can model into our database is very large, so I decided to put the most important data only. Here's a brief introduction to the key considerations in database design for such a system as the following:

➢ Entities:

1. **Menu_Item**:
   This entity will used to store each dish or item details on the menu.
   Attributes: with attributes like ItemID, Name, Description, Price, and Category.

2. **Customer:**
   This entity used to store customer details such as CustomerID, Name, contact_information, and address.

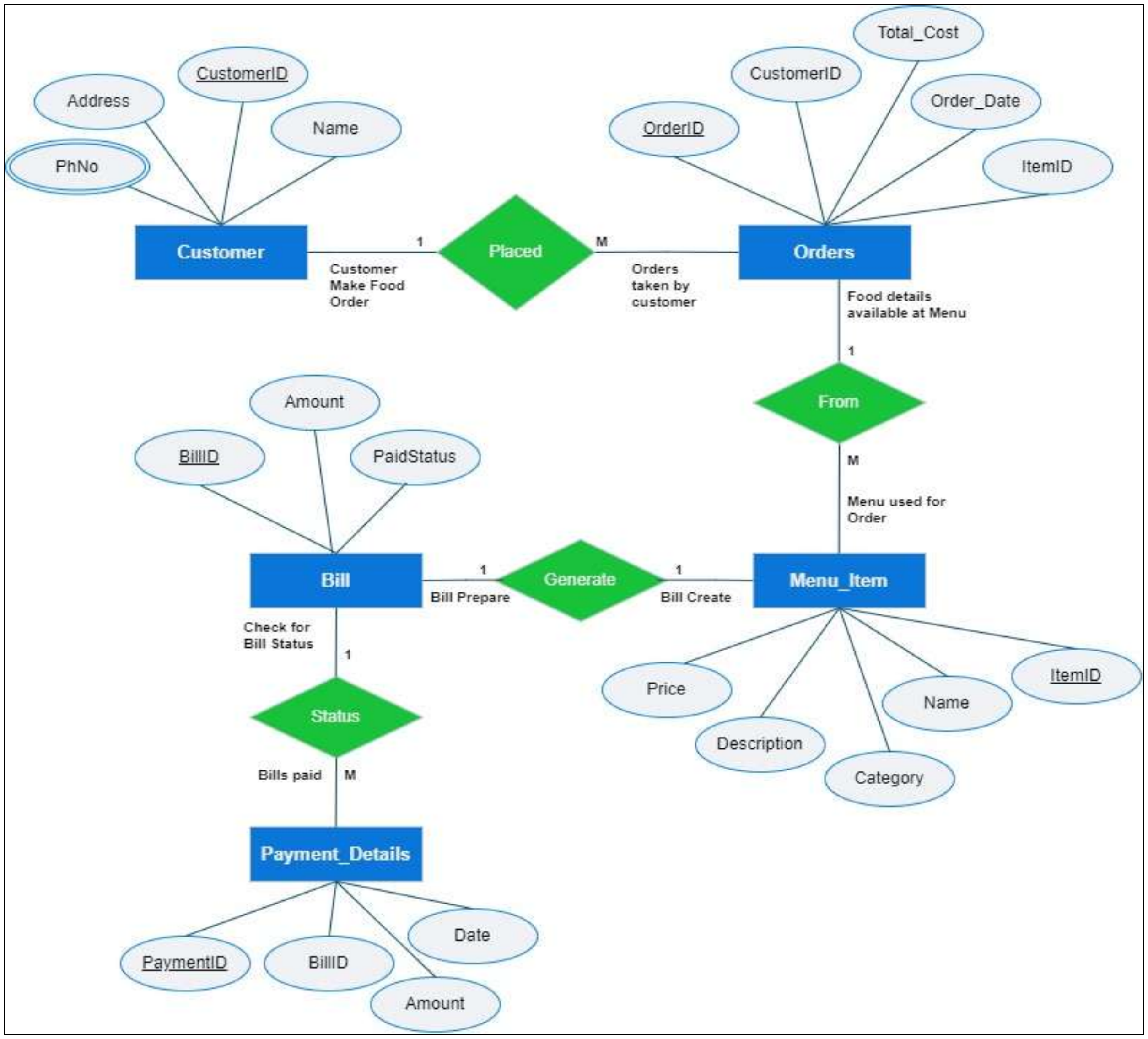
3. **Orders:**
   This entity used to track information about each order, including the customer placing the order, items ordered, total cost etc.

4. **Bill:**
   This entity will have the information regarding billing details like Bill amount, Bill paid status of customer for order etc.

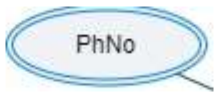
5. **Payment_detail:**
   Record details of payment transactions associated with orders as per bill.

➢ Entity Relationship Diagram:

Following is the ER diagram representing the relationships of entities I have used in my model.



➢ Symbol used in E-R Diagram are as following:

	Primary Key, i.e. oval with attribute underline		Relationship with its cardinality
	Attribute of entity		Participation
	Entity		Multi-Valued Attribute

➢ Mapping to Relational Schema:

In this section of relational schema mapping, I have translated the conceptual database planned within the past segment (ERM) to a logical database.
Following are the assumptions I've made for this project:

1. Customer, Orders, Menu_Item, Bill and Payment_Details all have unique IDs.
2. A customer can place order.
3. Orders can be placed with the help of items listed in Menu. So, the orders are associated with Menu_Item.
4. As the order is placed, base on price given in menu, the Bill will be generated, we can check the bill payment status.
5. The amount of bill will be paid by the customer as per his/her convenient method for payment.
6. It is assumed that the bill amount is paid by the customer at the time of order placed.

- **Mapping to Relational Table:**

Customer (CustomerID (PK), Name, PhNo, Address)

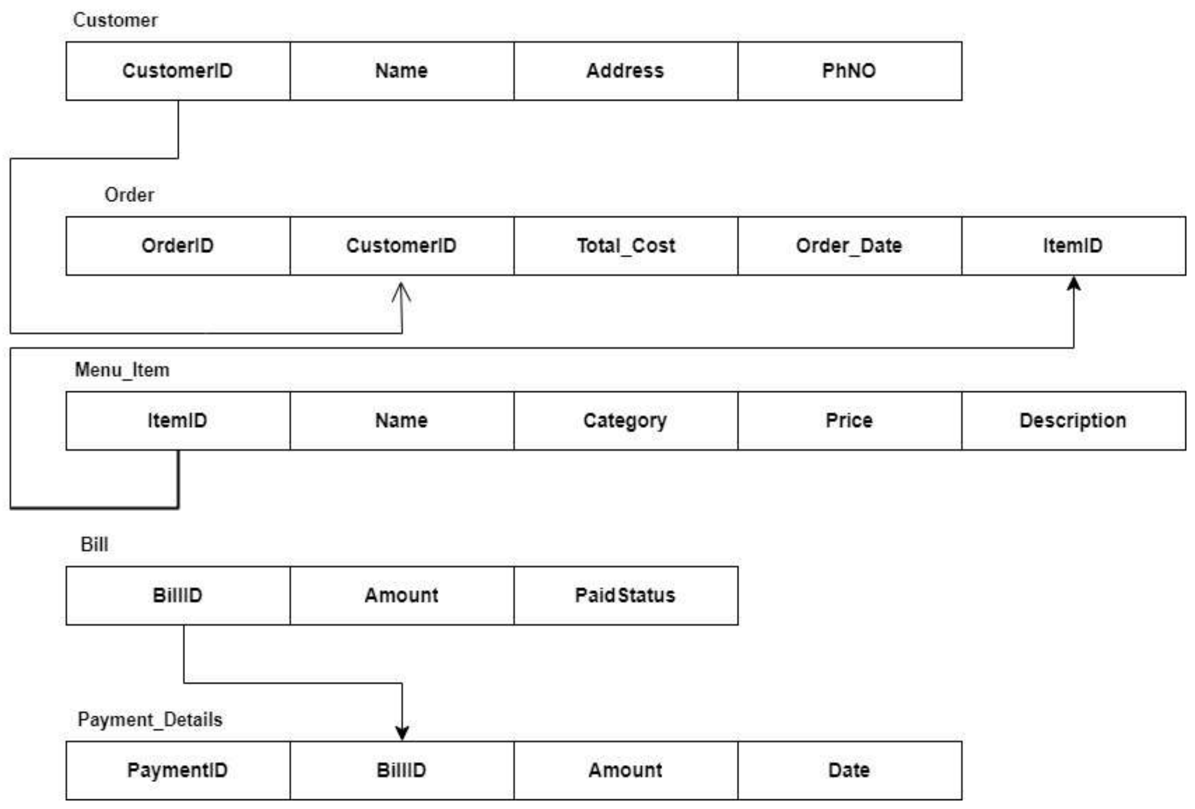
Orders (OrderID (PK), CustomerID (FK), ItemID (FK), Order_Date,Total_Cost)

Menu_Item (ItemID (PK), Name, Description, Price, Category)

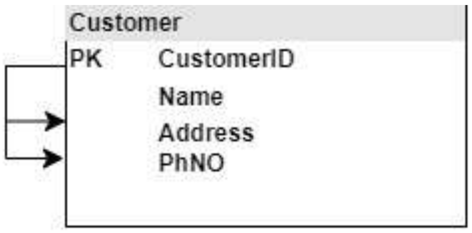
Bill(BillID (PK), Amount, PaidStatus)

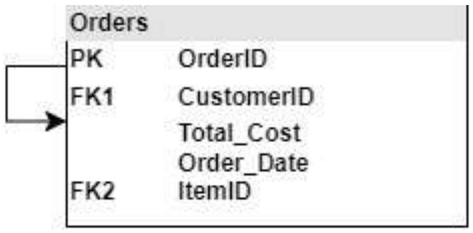
Payment_Detail(PaymentID(PK), BillID (FK), Amount, Date)

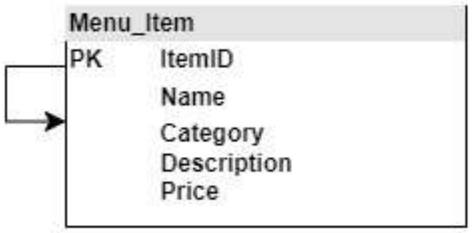
➢ Relational Schema Diagram:

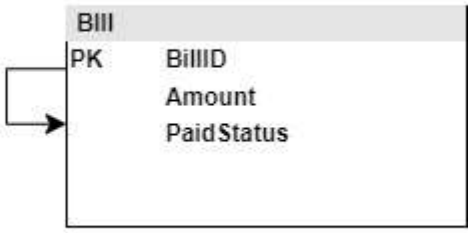


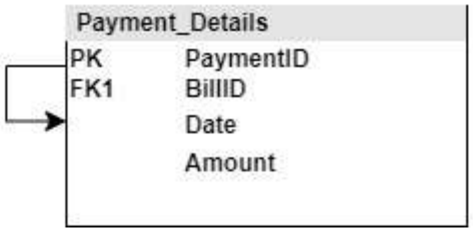
➢ Functional Dependency and Normalization:











Normalization:

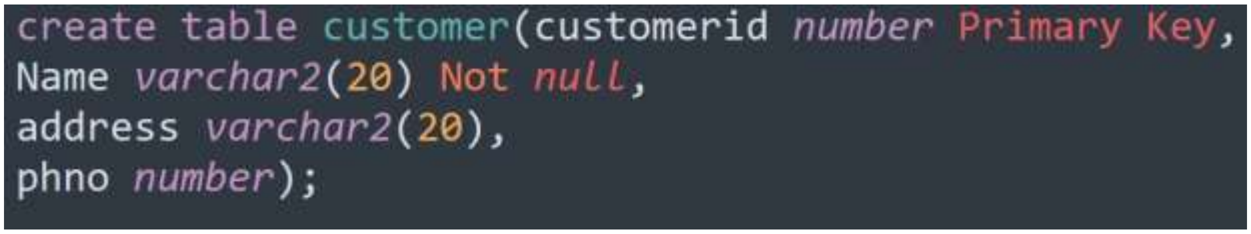
There is a multivalued attribute in the Customer table which is the PhNo attribute. As a result 1NF was applied, all the columns will have atomic values. Moving forward applying 2NF there was a separate table order which then contained the order details with having foreign key customerID in new table order. All resulting tables were in BCNF form and hence all normal forms were satisfory for every table in the database.

➢ Creating Database:

We will do explanation and SQL Code for creating the database Tables (including any constraints).

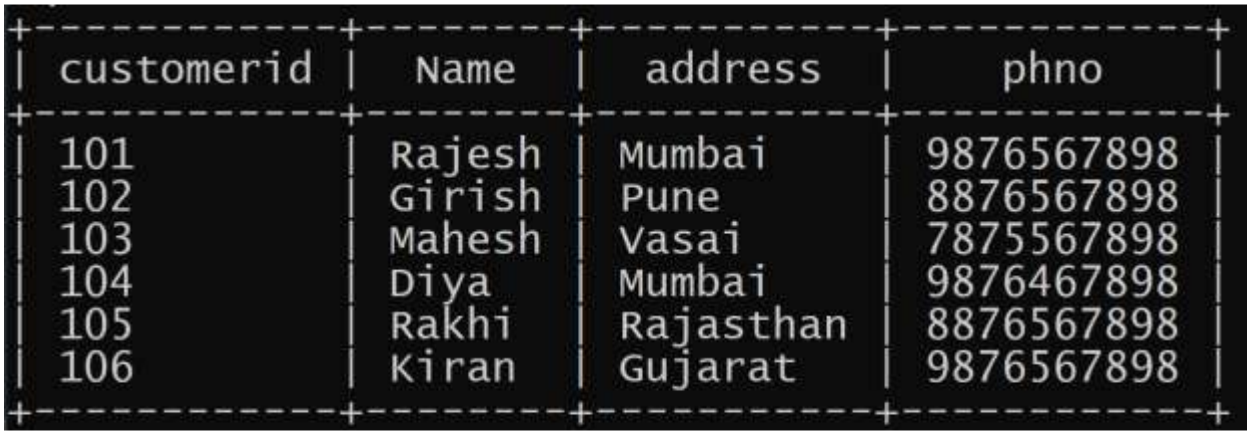
Creating Tables:

The table customer had to be created since it will become master tables for other tables.



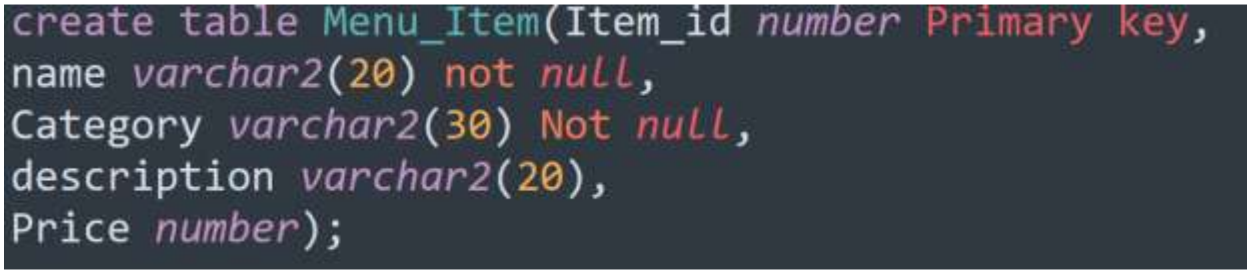
```
create table customer(customerid number Primary Key,
Name varchar2(20) Not null,
address varchar2(20),
phno number);
```

We can store the values here, and got the data. Here, the CustomerID is primary key, so it will only allowed unique numbers.



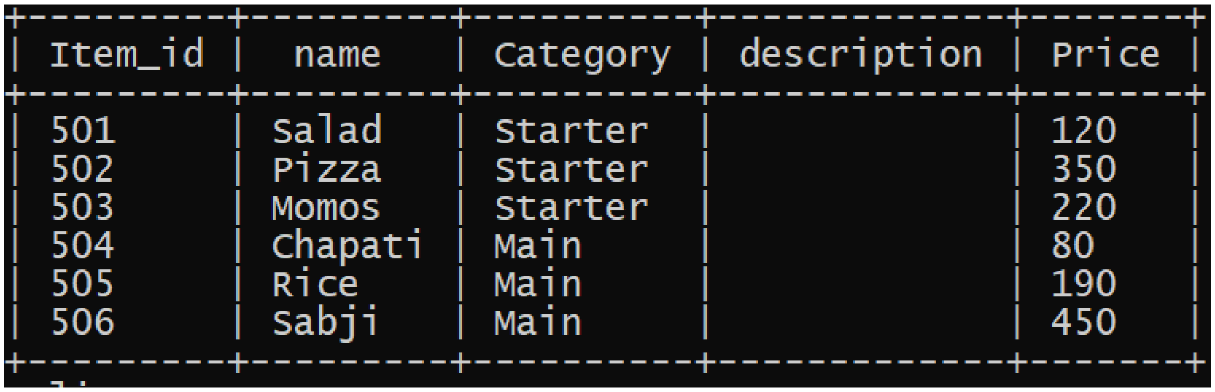
```
+------------+--------+-----------+------------+
| customerid |  Name  |  address  |    phno    |
+------------+--------+-----------+------------+
| 101        | Rajesh | Mumbai    | 9876567898 |
| 102        | Girish | Pune      | 8876567898 |
| 103        | Mahesh | Vasai     | 7875567898 |
| 104        | Diya   | Mumbai    | 9876467898 |
| 105        | Rakhi  | Rajasthan | 8876567898 |
| 106        | Kiran  | Gujarat   | 9876567898 |
+------------+--------+-----------+------------+
```

The table Menu_Item need to be created since it will become master tables for other tables.



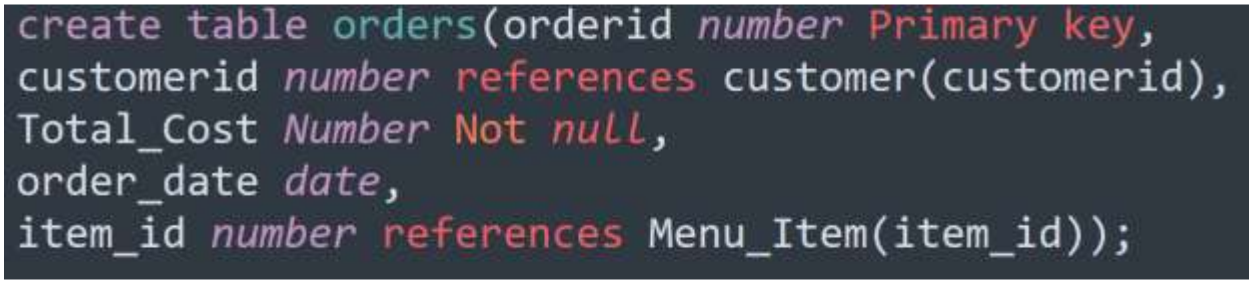
```
create table Menu_Item(Item_id number Primary key,
name varchar2(20) not null,
Category varchar2(30) Not null,
description varchar2(20),
Price number);
```

The data store at Menu_item will be:



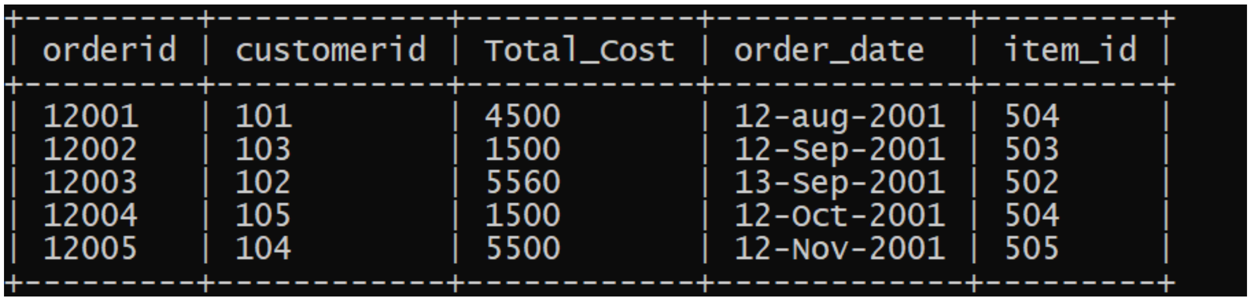
```
+---------+---------+----------+------------+-------+
| Item_id |  name   | Category | description | Price |
+---------+---------+----------+------------+-------+
| 501     | Salad   | Starter  |            | 120   |
| 502     | Pizza   | Starter  |            | 350   |
| 503     | Momos   | Starter  |            | 220   |
| 504     | Chapati | Main     |            | 80    |
| 505     | Rice    | Main     |            | 190   |
| 506     | Sabji   | Main     |            | 450   |
+---------+---------+----------+------------+-------+
```

Now we will create table orders as below with proper constrains with primary key and foreign key:



```
create table orders(orderid number Primary key,
customerid number references customer(customerid),
Total_Cost Number Not null,
order_date date,
item_id number references Menu_Item(item_id));
```

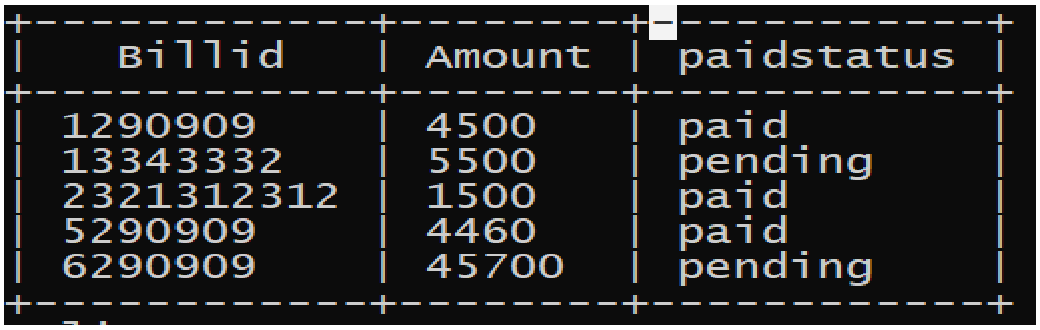
After saving the data will be:



```
+---------+------------+------------+-------------+---------+
| orderid | customerid | Total_Cost | order_date  | item_id |
+---------+------------+------------+-------------+---------+
| 12001   | 101        | 4500       | 12-aug-2001 | 504     |
| 12002   | 103        | 1500       | 12-Sep-2001 | 503     |
| 12003   | 102        | 5560       | 13-Sep-2001 | 502     |
| 12004   | 105        | 1500       | 12-Oct-2001 | 504     |
| 12005   | 104        | 5500       | 12-Nov-2001 | 505     |
+---------+------------+------------+-------------+---------+
```

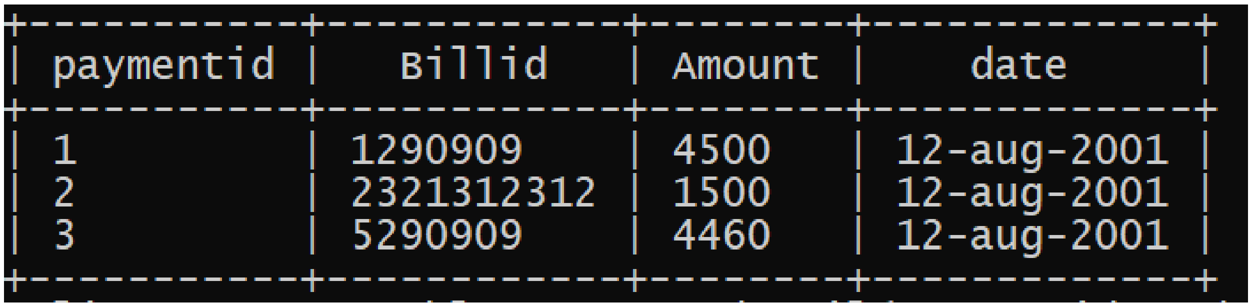
Next all tables will be created.

The table bill:



```
+------------+--------+------------+
|   Billid   | Amount | paidstatus |
+------------+--------+------------+
| 1290909    | 4500   | paid       |
| 13343332   | 5500   | pending    |
| 2321312312 | 1500   | paid       |
| 5290909    | 4460   | paid       |
| 6290909    | 45700  | pending    |
+------------+--------+------------+
```

The table payment_details:

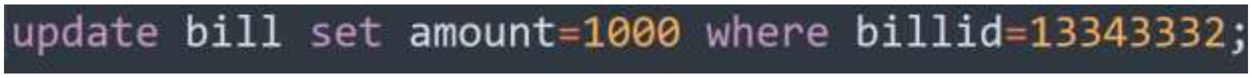


```
+-----------+------------+--------+-------------+
| paymentid |   Billid   | Amount |    date     |
+-----------+------------+--------+-------------+
| 1         | 1290909    | 4500   | 12-aug-2001 |
| 2         | 2321312312 | 1500   | 12-aug-2001 |
| 3         | 5290909    | 4460   | 12-aug-2001 |
+-----------+------------+--------+-------------+
```

➢ **Update Operations:**

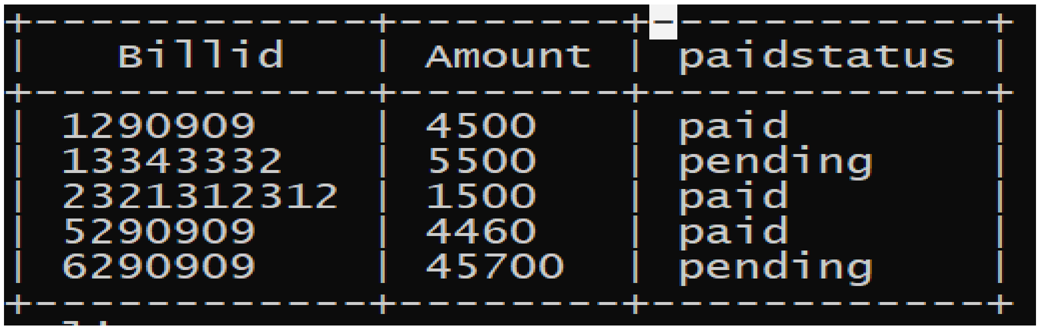
Update operations can be used for a range of actions. In this example, I will show how a bill amount can be change if the person will get some amount as discount in bill.

Creating update command:



```
update bill set amount=1000 where billid=13343332;
```

A view of original Table:

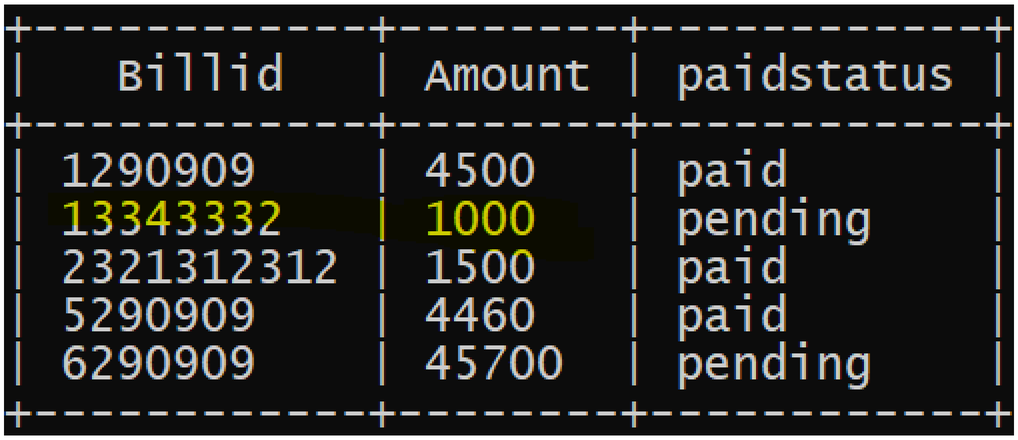


```
+-------------+---------+--------------+
|   Billid    | Amount  | paidstatus   |
+-------------+---------+--------------+
| 1290909     | 4500    | paid         |
| 13343332    | 5500    | pending      |
| 2321312312  | 1500    | paid         |
| 5290909     | 4460    | paid         |
| 6290909     | 45700   | pending      |
+-------------+---------+--------------+
```

Now we can display the new values:

select * from bill;

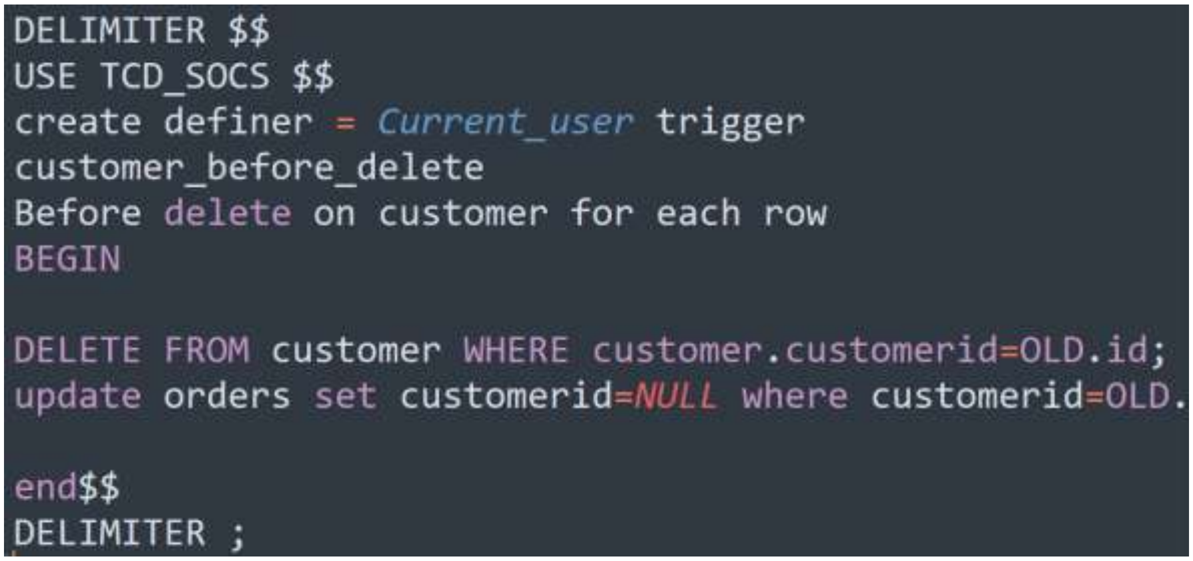
The amount of Billid 13343332 is change to new amount Rs. 1000.



```
+-------------+---------+--------------+
|   Billid    | Amount  | paidstatus   |
+-------------+---------+--------------+
| 1290909     | 4500    | paid         |
| 13343332    | 1000    | pending      |
| 2321312312  | 1500    | paid         |
| 5290909     | 4460    | paid         |
| 6290909     | 45700   | pending      |
+-------------+---------+--------------+
```

➢ **Trigger Operations:**

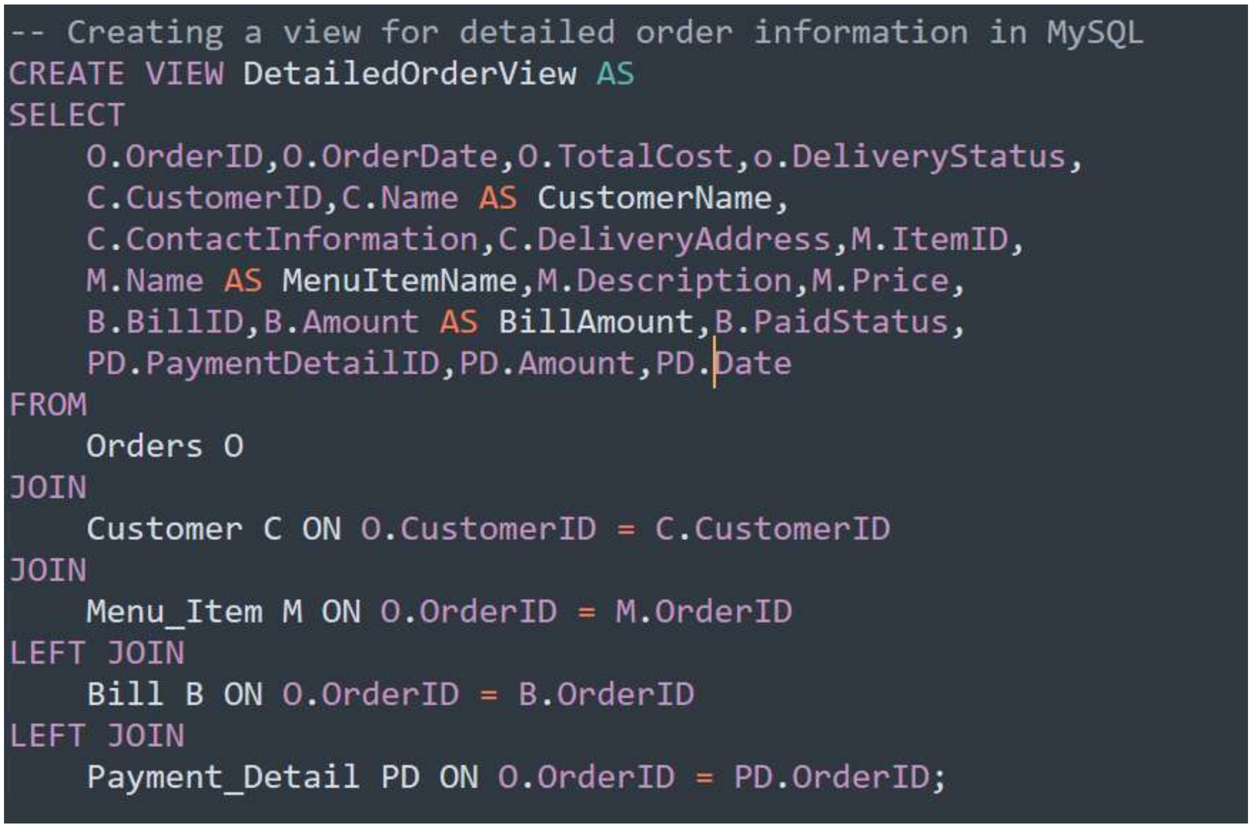
Trigger command can be used for the wide purpose in the database. I used it in case of make sure for integrity management of the database on deletion of tupple. For example before delete operations on customer table we take backup of customer id in variable. We set the all customer id of order table to null of deleted customer for each row.



```
DELIMITER $$
USE TCD_SOCS $$
create definer = Current_user trigger
customer_before_delete
Before delete on customer for each row
BEGIN

DELETE FROM customer WHERE customer.customerid=OLD.id;
update orders set customerid=NULL where customerid=OLD.

end$$
DELIMITER ;
```

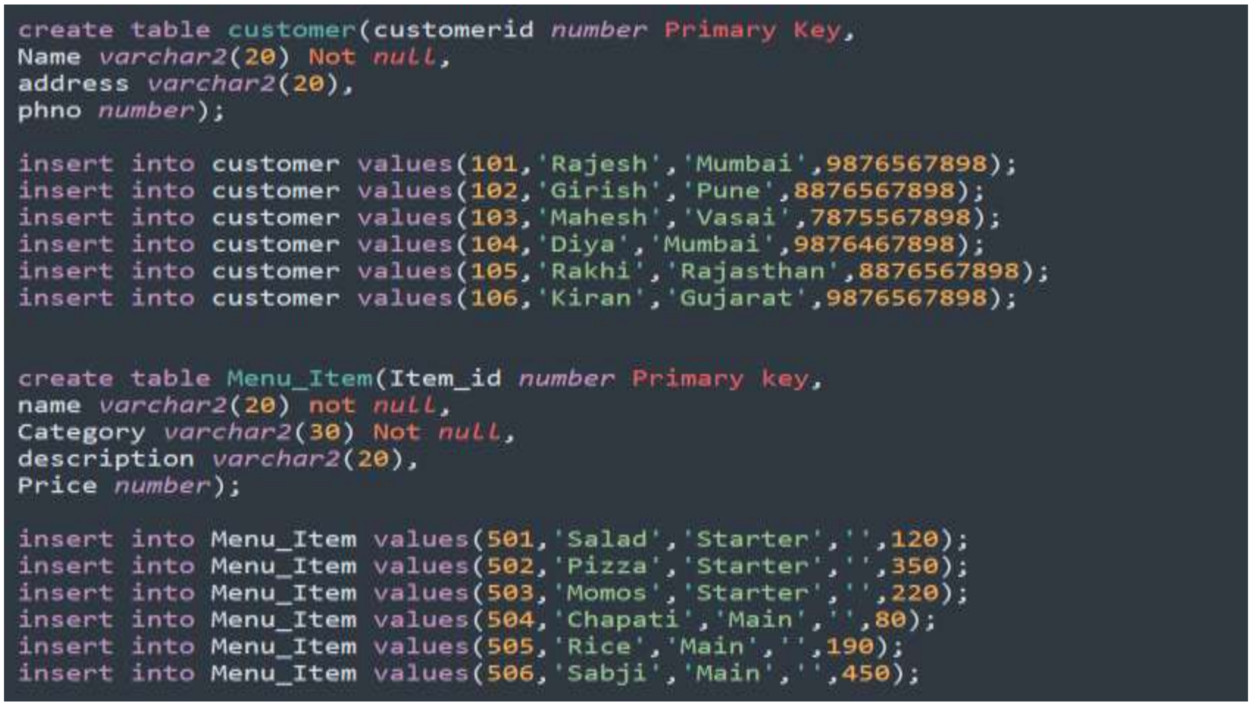
➢ **View Creation:**

Views are often used for one or more purpose. It is used to create table information that is frequently used to see data or to make security on data from specific people access. After creating view you can view data from it as normal table.

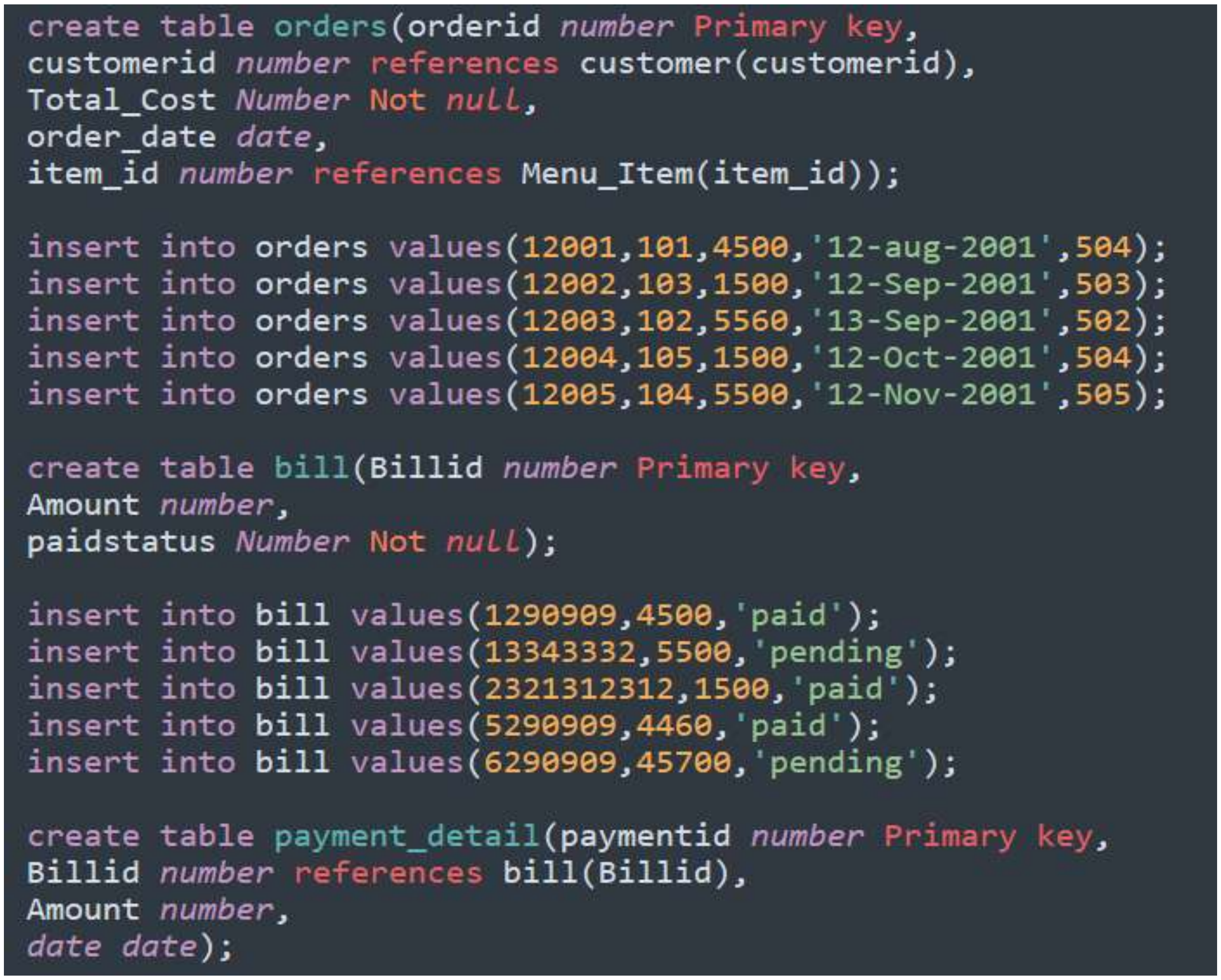


```sql
-- Creating a view for detailed order information in MySQL
CREATE VIEW DetailedOrderView AS
SELECT
    O.OrderID,O.OrderDate,O.TotalCost,o.DeliveryStatus,
    C.CustomerID,C.Name AS CustomerName,
    C.ContactInformation,C.DeliveryAddress,M.ItemID,
    M.Name AS MenuItemName,M.Description,M.Price,
    B.BillID,B.Amount AS BillAmount,B.PaidStatus,
    PD.PaymentDetailID,PD.Amount,PD.Date
FROM
    Orders O
JOIN
    Customer C ON O.CustomerID = C.CustomerID
JOIN
    Menu_Item M ON O.OrderID = M.OrderID
LEFT JOIN
    Bill B ON O.OrderID = B.OrderID
LEFT JOIN
    Payment_Detail PD ON O.OrderID = PD.OrderID;
```

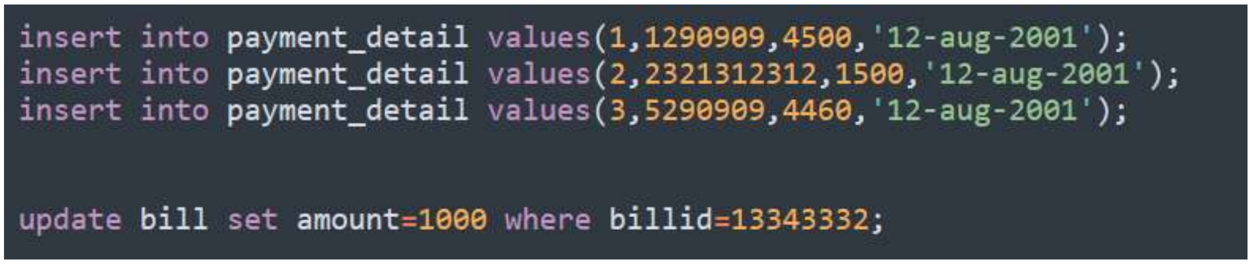
➢ **My all Code:**



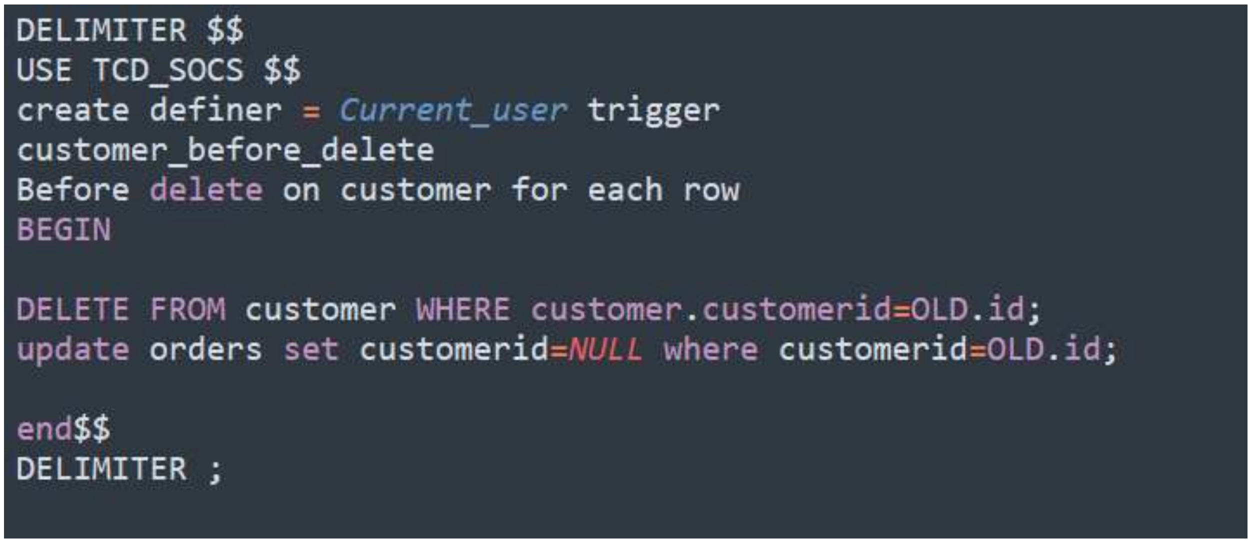
```
create table customer(customerid number Primary Key,
Name varchar2(20) Not null,
address varchar2(20),
phno number);

insert into customer values(101,'Rajesh','Mumbai',9876567898);
insert into customer values(102,'Girish','Pune',8876567898);
insert into customer values(103,'Mahesh','Vasai',7875567898);
insert into customer values(104,'Diya','Mumbai',9876467898);
insert into customer values(105,'Rakhi','Rajasthan',8876567898);
insert into customer values(106,'Kiran','Gujarat',9876567898);


create table Menu_Item(Item_id number Primary key,
name varchar2(20) not null,
Category varchar2(30) Not null,
description varchar2(20),
Price number);

insert into Menu_Item values(501,'Salad','Starter','',120);
insert into Menu_Item values(502,'Pizza','Starter','',350);
insert into Menu_Item values(503,'Momos','Starter','',220);
insert into Menu_Item values(504,'Chapati','Main','',80);
insert into Menu_Item values(505,'Rice','Main','',190);
insert into Menu_Item values(506,'Sabji','Main','',450);
```



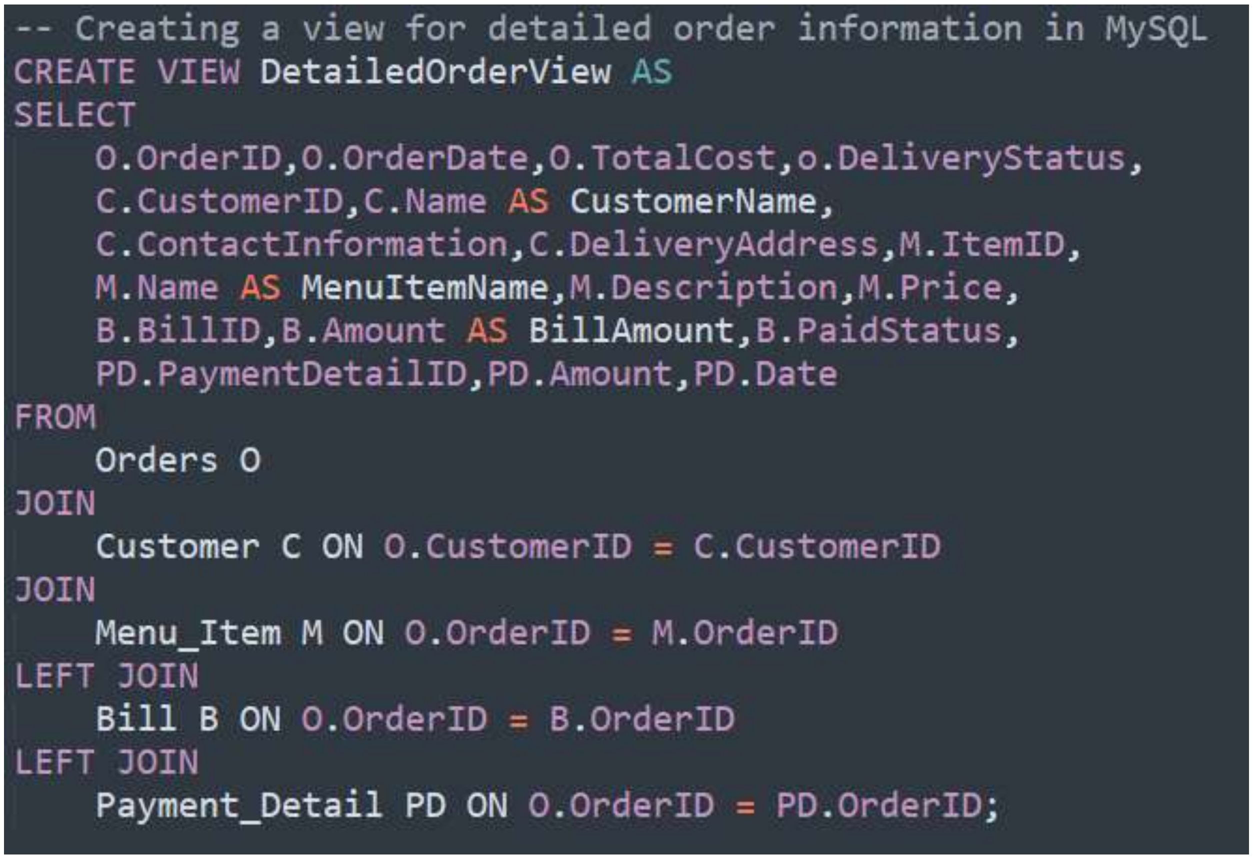
```
create table orders(orderid number Primary key,
customerid number references customer(customerid),
Total_Cost Number Not null,
order_date date,
item_id number references Menu_Item(item_id));

insert into orders values(12001,101,4500,'12-aug-2001',504);
insert into orders values(12002,103,1500,'12-Sep-2001',503);
insert into orders values(12003,102,5560,'13-Sep-2001',502);
insert into orders values(12004,105,1500,'12-Oct-2001',504);
insert into orders values(12005,104,5500,'12-Nov-2001',505);

create table bill(Billid number Primary key,
Amount number,
paidstatus Number Not null);

insert into bill values(1290909,4500,'paid');
insert into bill values(13343332,5500,'pending');
insert into bill values(2321312312,1500,'paid');
insert into bill values(5290909,4460,'paid');
insert into bill values(6290909,45700,'pending');

create table payment_detail(paymentid number Primary key,
Billid number references bill(Billid),
Amount number,
date date);
```



```sql
insert into payment_detail values(1,1290909,4500,'12-aug-2001');
insert into payment_detail values(2,2321312312,1500,'12-aug-2001');
insert into payment_detail values(3,5290909,4460,'12-aug-2001');


update bill set amount=1000 where billid=13343332;
```



```sql
DELIMITER $$
USE TCD_SOCS $$
create definer = Current_user trigger
customer_before_delete
Before delete on customer for each row
BEGIN

DELETE FROM customer WHERE customer.customerid=OLD.id;
update orders set customerid=NULL where customerid=OLD.id;

end$$
DELIMITER ;
```



```sql
-- Creating a view for detailed order information in MySQL
CREATE VIEW DetailedOrderView AS
SELECT
    O.OrderID,O.OrderDate,O.TotalCost,o.DeliveryStatus,
    C.CustomerID,C.Name AS CustomerName,
    C.ContactInformation,C.DeliveryAddress,M.ItemID,
    M.Name AS MenuItemName,M.Description,M.Price,
    B.BillID,B.Amount AS BillAmount,B.PaidStatus,
    PD.PaymentDetailID,PD.Amount,PD.Date
FROM
    Orders O
JOIN
    Customer C ON O.CustomerID = C.CustomerID
JOIN
    Menu_Item M ON O.OrderID = M.OrderID
LEFT JOIN
    Bill B ON O.OrderID = B.OrderID
LEFT JOIN
    Payment_Detail PD ON O.OrderID = PD.OrderID;
```

Different roles and permissions have been assigned which gives different privileges to everyone.

```
CREATE USER 'DBA_Admin'@'localhost' IDENTIFIED BY 'rootpassword"';

CREATE USER 'SC’@'localhost' IDENTIFIED BY 'user1@';

Create USER 'manager'@'localhost' IDENTIFIED by 'food123';

grant ALL PRIVILEGES on orderonline to 'DBA_Admin'@'localhost';
grant ALL PRIVILEGES on SC to 'DBA_Admin'@'localhost';
grant ALL PRIVILEGES on manger to 'DBA_Admin'@'localhost';
```

Explanation:
DBA_Admin : Is the database admin who will have the access to every table of the database and will have all user privileges as well.