COS 103 – Database Systems and Applications

Name	:	Chaw Thiri Win

Table of Contents

System Overview 3
Task – 1 (Entity-Relationship Diagram) 3
Task-2-DDL 4
  Create and Use Database 4
  Table User 4
  Table Food 5
  Table Comboset 5
  Table Comboset_Item 6
  Table Transactions 7
  Table Food_Order and Combo_Order 7
  Inserting 8
    Insert into User 9
    Insert into Food 9
    Insert into Comboset 10
    Insert into Comboset_Item 11
    Insert into Transactions 11
    Insert into Food_Order 12
    Insert into Combo_Order 13
Task-3-DML 14

System Overview

This System represents a Fast-food Restaurant which is own by Ronald. The system capture the information of each food items which include item name, item type, and price. This restaurant also sells combo items at a promotional price.

Task – 1 (Entity-Relationship Diagram)

This diagram shown below is an Entity-Relationship Diagram of a Fast-Food Restaurant. The diagram contains 7 tables with their multiplicity, cardinality, and participation. It involves several key entities. In User: contains personal information such as username, address, phone number, password, role, and nationality. In food: details food items, including name, type, and price with tax. Comboset: Defines combination meal sets with a name, set number, and price. Transaction: Represents customer transactions, linking to users, and includes details like total amount, quantity, tax, and date-time. Food_Order: Represents the food items in a transaction, connecting food items with their quantities. Combo_Order: Records ordered combo sets in a transaction, along with quantities. Comboset_Item: Maps food items to combo sets, indicating which items belong to each combo set. The relationships between entities are shown through foreign keys, ensuring that users, food, combosets, and transactions are properly connected for tracking and management purposes.

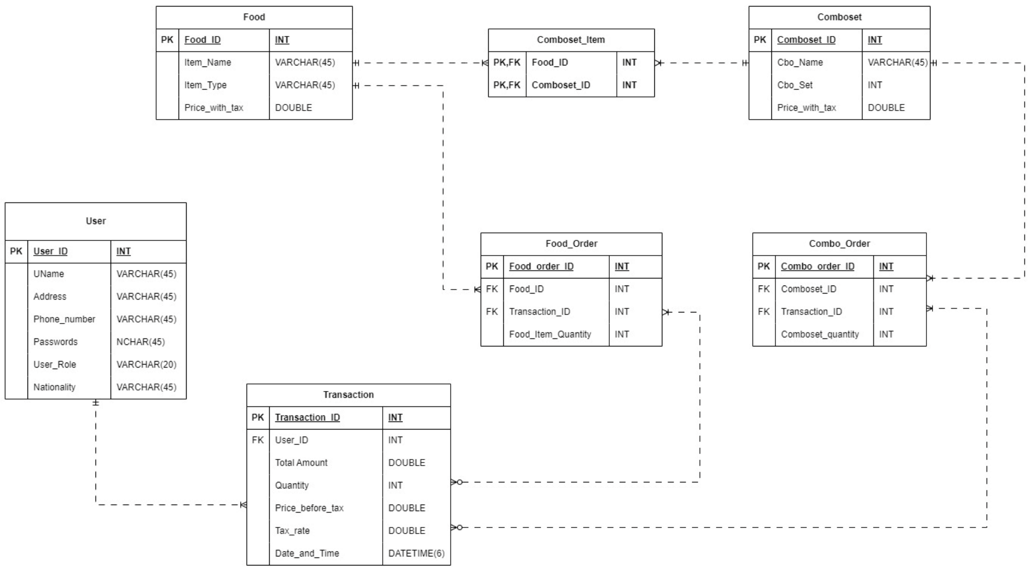
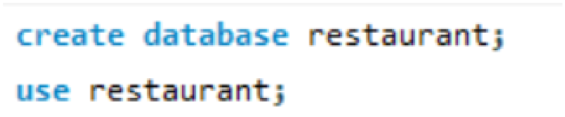


Figure - Entity-Relationship Diagram (Task-1)

Task-2-DDL

DDL (Data Definition Language): DDL commands are used to define and modify the structure of the database and its objects, like tables, schemas, indexes, etc. It includes operations like creating, altering, and deleting tables.

Create and Use Database

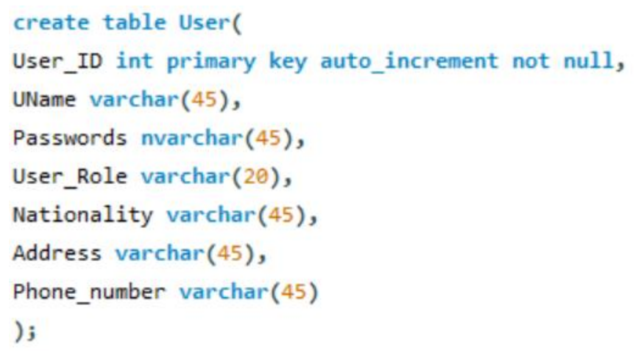


```
create database restaurant;
use restaurant;
```

Figure: create and use database

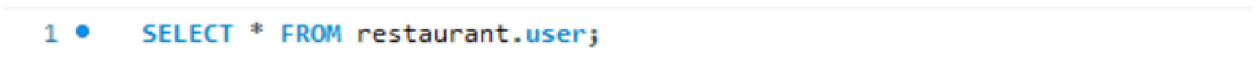
First, we create a database which name is restaurant. And then we use the database to creating tables, inserting values, altering, and deleting. The given restaurant database begins by defining the User table, which has data about workers (such as managers and cashiers) such as their role, country, address, and phone number.

Table User

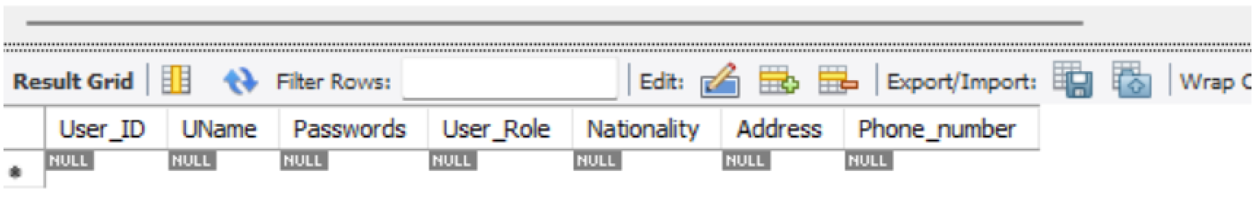


```
create table User(
User_ID int primary key auto_increment not null,
UName varchar(45),
Passwords nvarchar(45),
User_Role varchar(20),
Nationality varchar(45),
Address varchar(45),
Phone_number varchar(45)
);
```

After creating a table, the table will be result like this as we haven't added its values. Before adding the values, it'll be set to null by default.



```
SELECT * FROM restaurant.user;
```



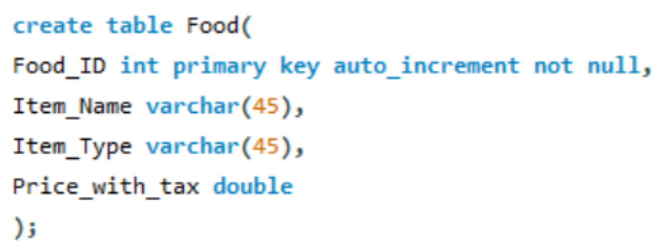
User_ID	UName	Passwords	User_Role	Nationality	Address	Phone_number
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure: Result of Create Table User

This is the SQL query for creating a table named User. These are known as table definition.

Table Food

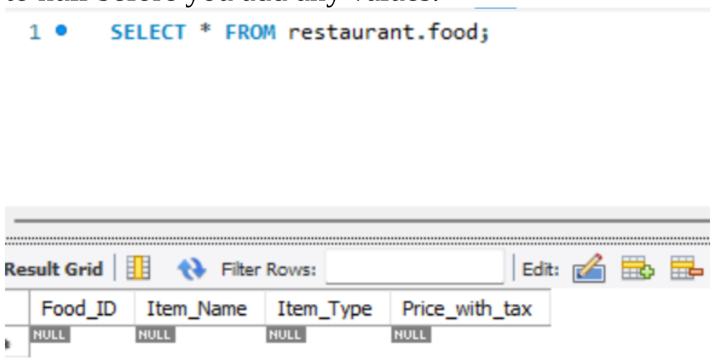
Subsequently, a Food table is made to include details on specific menu items, including drinks and food, along with fields for the item's name, kind, and price (tax included).



```sql
create table Food(
Food_ID int primary key auto_increment not null,
Item_Name varchar(45),
Item_Type varchar(45),
Price_with_tax double
);
```

Figure: Create Table Food

Since we haven't put the values to the table, it will look like this once it is created. By default, it will be set to null before you add any values.



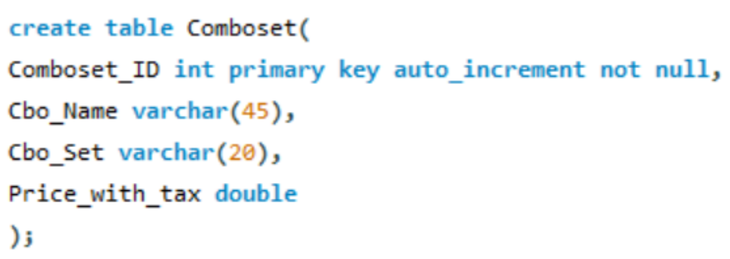
```sql
SELECT * FROM restaurant.food;
```

Food_ID	Item_Name	Item_Type	Price_with_tax
NULL	NULL	NULL	NULL

Figure: Result of Create Table Food

The comboname, set identification, and price with tax are all stored in fields of the Comboset table, which contains information about the combination meals that the restaurant offers.

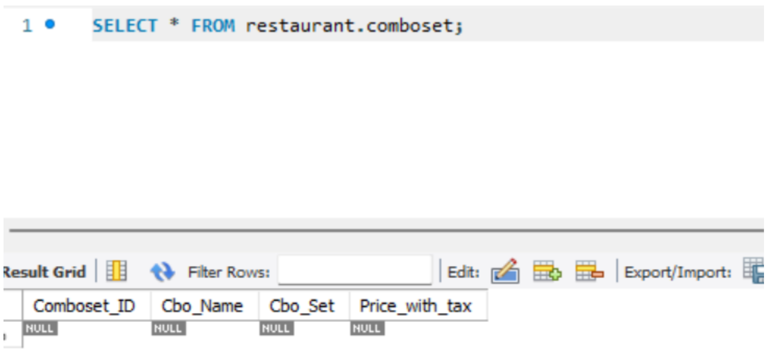
Table Comboset



```sql
create table Comboset(
Comboset_ID int primary key auto_increment not null,
Cbo_Name varchar(45),
Cbo_Set varchar(20),
Price_with_tax double
);
```

Figure: Create Table Comboset

As mentioned above, the tables will initially contain null values in every field upon creation unless entries are added to them.



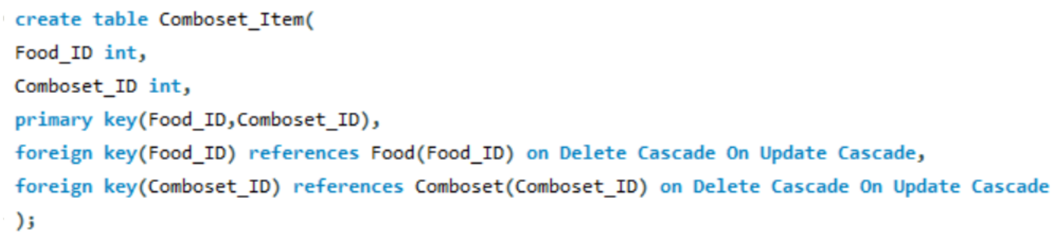
```
SELECT * FROM restaurant.comboset;
```

Comboset_ID	Cbo_Name	Cbo_Set	Price_with_tax
NULL	NULL	NULL	NULL

Figure: Result of Create Table Comboset

Table Comboset_Item

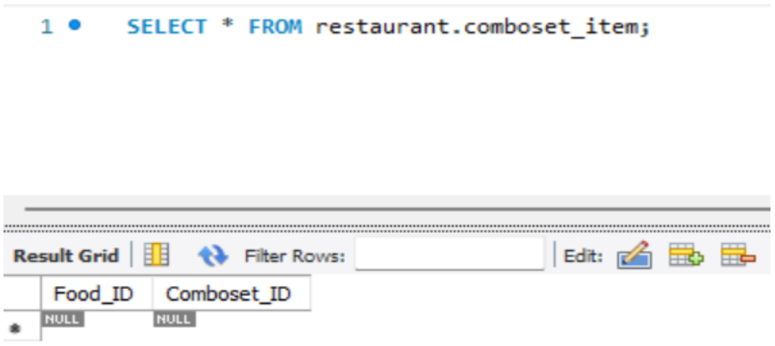
A Comboset_Item database is constructed to build associations between food items and combination sets. By using foreign keys to connect food items to their respective combo sets, this functions as a junction table that permits several food items to be a part of a single combo set.



```
create table Comboset_Item(
Food_ID int,
Comboset_ID int,
primary key(Food_ID,Comboset_ID),
foreign key(Food_ID) references Food(Food_ID) on Delete Cascade On Update Cascade,
foreign key(Comboset_ID) references Comboset(Comboset_ID) on Delete Cascade On Update Cascade
);
```

Figure: Create Table Comboset_Item

All of the fields of the tables will initially be set to null before data is added later.



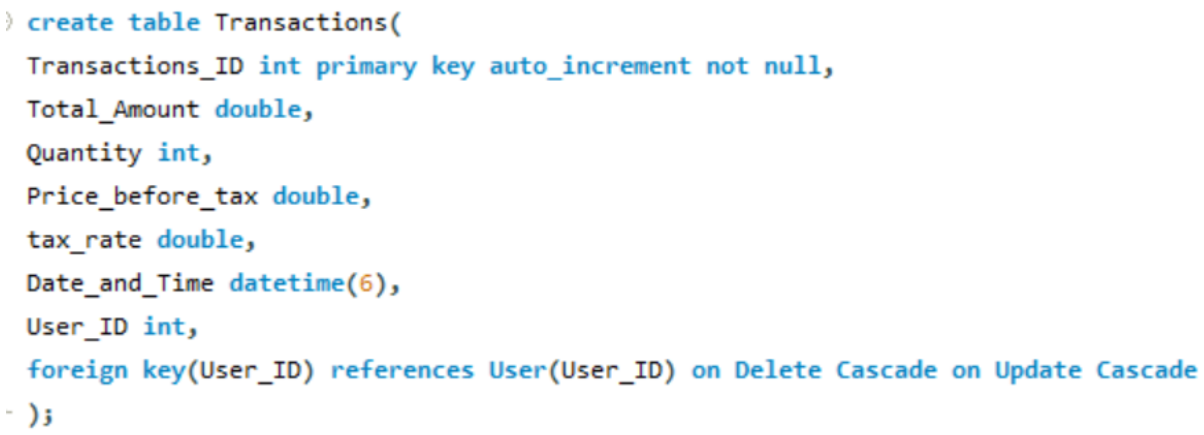
```
SELECT * FROM restaurant.comboset_item;
```

Food_ID	Comboset_ID
NULL	NULL

Figure: Result of Create Table Comboset_Item

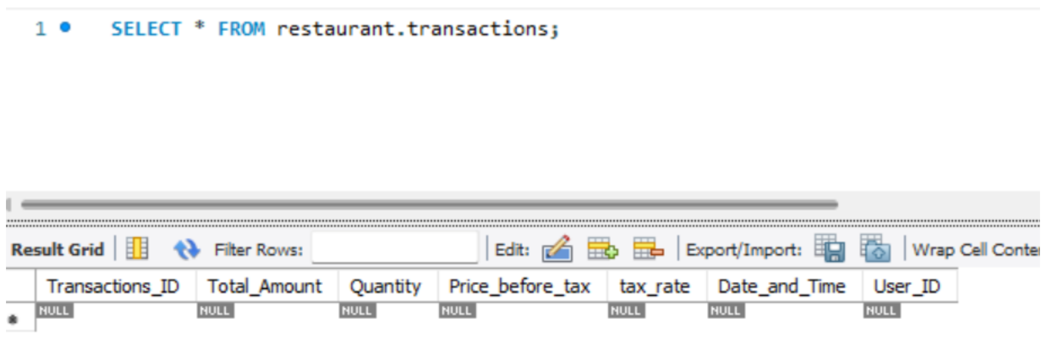
Table Transactions

The amount, quantity, price before tax, tax rate, date, time, and user (cashier) who handled the transaction are all recorded in the Transactions database, which keeps track of sales.



```
create table Transactions(
Transactions_ID int primary key auto_increment not null,
Total_Amount double,
Quantity int,
Price_before_tax double,
tax_rate double,
Date_and_Time datetime(6),
User_ID int,
foreign key(User_ID) references User(User_ID) on Delete Cascade on Update Cascade
);
```

Figure: Create Table Transactions



```
SELECT * FROM restaurant.transactions;
```

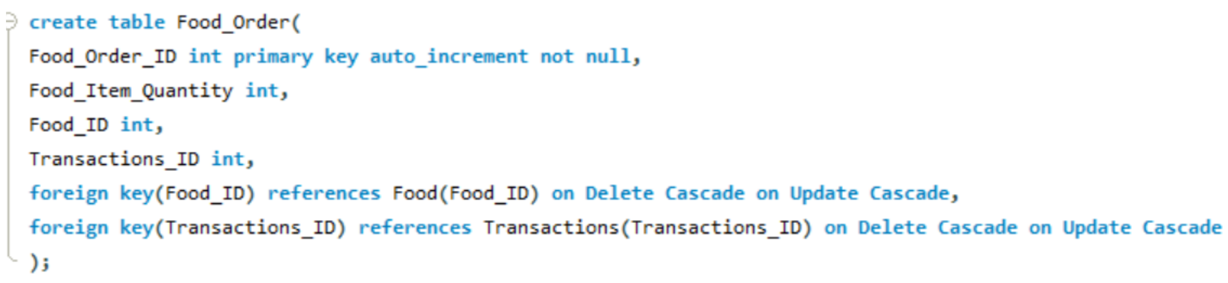
Transactions_ID	Total_Amount	Quantity	Price_before_tax	tax_rate	Date_and_Time	User_ID
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure: Result of Create Table Transactions

Table Food_Order and Combo_Order

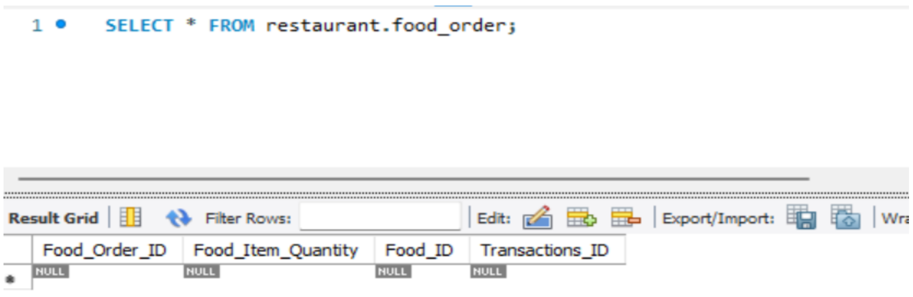
The Food_Order and Combo_Order tables use foreign keys to refer to the specific food items or combination sets in each transaction.

The initial data, which includes user information, food and beverage items, combo set details, sample transactions, and matching meal and combination orders, are then added to these tables by the script. This structure allows a thorough record of restaurant operations, tying menu items and combinations to transactions handled by different users.



```
create table Food_Order(
Food_Order_ID int primary key auto_increment not null,
Food_Item_Quantity int,
Food_ID int,
Transactions_ID int,
foreign key(Food_ID) references Food(Food_ID) on Delete Cascade on Update Cascade,
foreign key(Transactions_ID) references Transactions(Transactions_ID) on Delete Cascade on Update Cascade
);
```

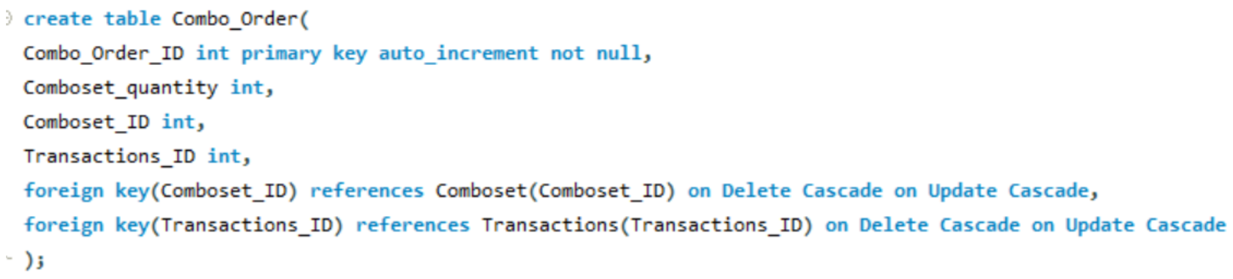
Figure: Create Table Food_order



```
SELECT * FROM restaurant.food_order;
```

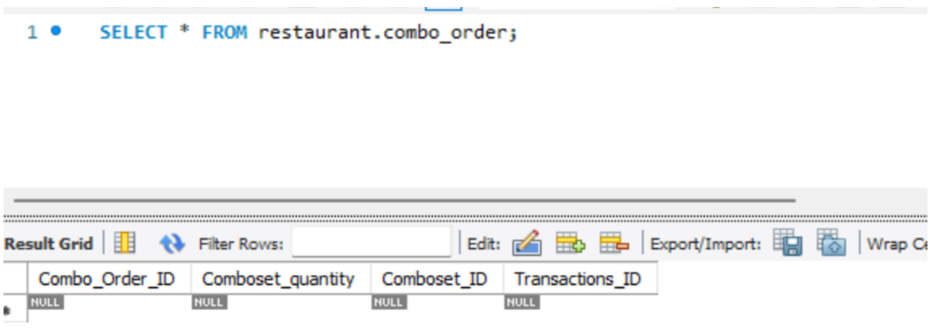
Food_Order_ID	Food_Item_Quantity	Food_ID	Transactions_ID
NULL	NULL	NULL	NULL

Figure: Result of Create Table Food_order



```
create table Combo_Order(
Combo_Order_ID int primary key auto_increment not null,
Comboset_quantity int,
Comboset_ID int,
Transactions_ID int,
foreign key(Comboset_ID) references Comboset(Comboset_ID) on Delete Cascade on Update Cascade,
foreign key(Transactions_ID) references Transactions(Transactions_ID) on Delete Cascade on Update Cascade
);
```

Figure: Create Table Combo_Order



```
SELECT * FROM restaurant.combo_order;
```

Combo_Order_ID	Comboset_quantity	Comboset_ID	Transactions_ID
NULL	NULL	NULL	NULL

Figure: Result of Create Table Combo_Order

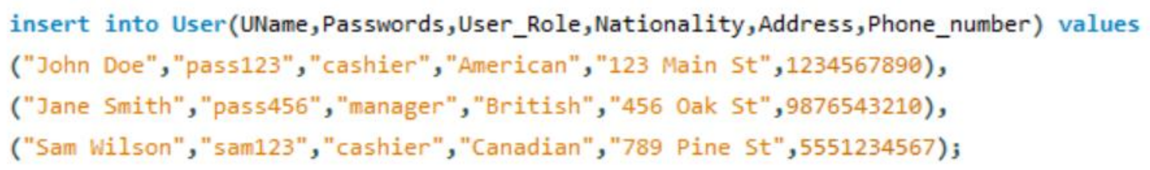
Inserting

Inserting values into SQL tables means adding new data into a specific table in a database. We use the INSERT INTO command for this. We specify the table name, the columns we want to add data to, and the actual values to insert. The structure looks like this:

- **INSERT INTO**: Tells the database you're adding new data.
- **Table name**: The name of the table where you're adding data.
- **Columns**: The specific parts of the table (columns) that will receive the new data.
- **Values**: The actual data you're adding for each column.

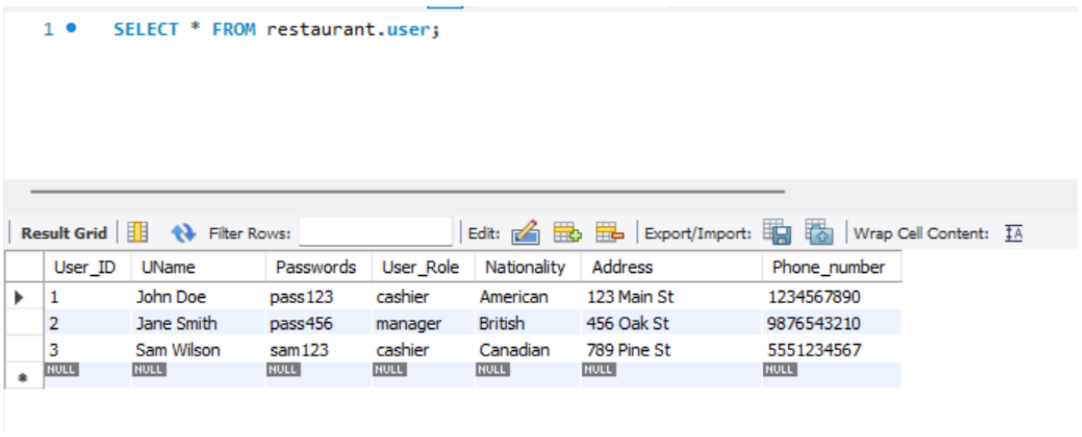
Insert into User

User Table: The INSERT statements include two cashiers and one manager as new users in the User table. A variety of employee records are created as each user has fields for their name, password, role, country, address, and phone number.



```
insert into User(UName,Passwords,User_Role,Nationality,Address,Phone_number) values
("John Doe","pass123","cashier","American","123 Main St",1234567890),
("Jane Smith","pass456","manager","British","456 Oak St",9876543210),
("Sam Wilson","sam123","cashier","Canadian","789 Pine St",5551234567);
```

Figure: Insert Into User



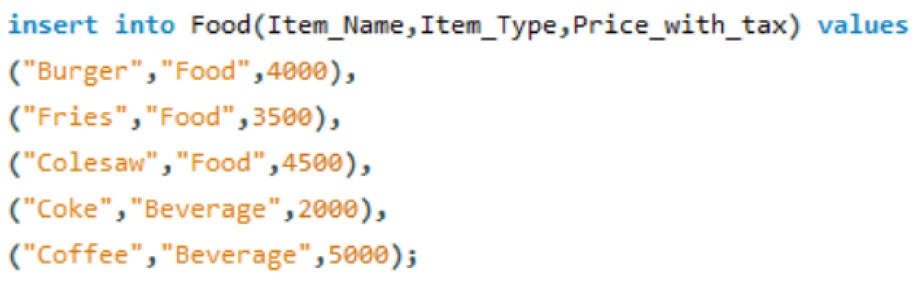
```
SELECT * FROM restaurant.user;
```

User_ID	UName	Passwords	User_Role	Nationality	Address	Phone_number
1	John Doe	pass123	cashier	American	123 Main St	1234567890
2	Jane Smith	pass456	manager	British	456 Oak St	9876543210
3	Sam Wilson	sam123	cashier	Canadian	789 Pine St	5551234567
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure: Result of Insert Into User

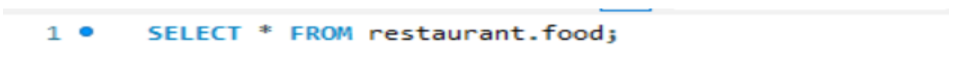
Insert into Food

Food Table: Five menu items that fall into the food or beverage categories are displayed on the food table. The price, item name, and type are all included in each entry, and these details will be used as transaction references in other tables.

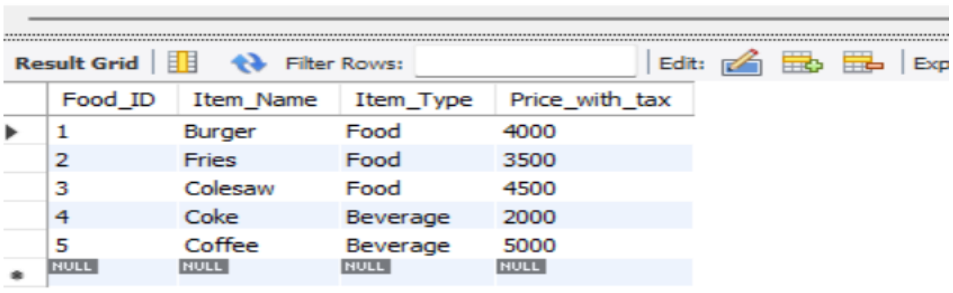


```
insert into Food(Item_Name,Item_Type,Price_with_tax) values
("Burger","Food",4000),
("Fries","Food",3500),
("Colesaw","Food",4500),
("Coke","Beverage",2000),
("Coffee","Beverage",5000);
```

Figure: Insert Into Food



```sql
SELECT * FROM restaurant.food;
```

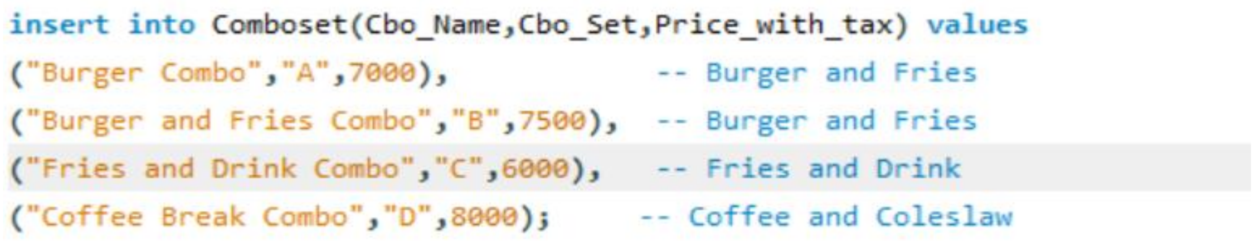


Food_ID	Item_Name	Item_Type	Price_with_tax
1	Burger	Food	4000
2	Fries	Food	3500
3	Colesaw	Food	4500
4	Coke	Beverage	2000
5	Coffee	Beverage	5000
NULL	NULL	NULL	NULL

Figure: Result of Insert Into Food

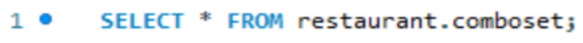
Insert into Comboset

Comboset Table: Four combo meal options, each with a distinct name, set identifier, and price, are added to the Comboset table. Customers may choose from a variety of meal and drink combinations using these combos.

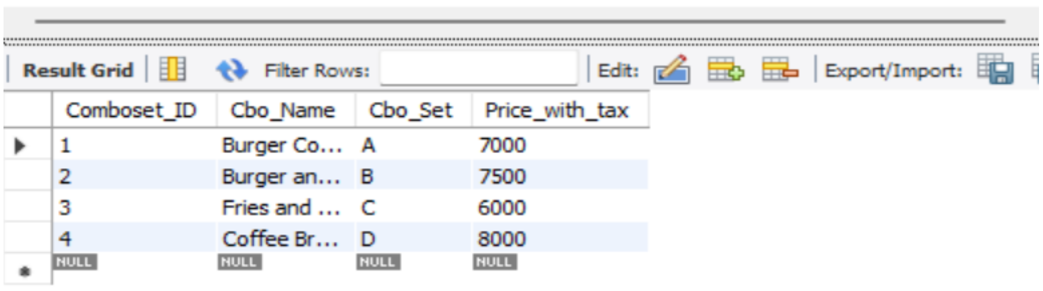


```sql
insert into Comboset(Cbo_Name,Cbo_Set,Price_with_tax) values
("Burger Combo","A",7000),              -- Burger and Fries
("Burger and Fries Combo","B",7500),   -- Burger and Fries
("Fries and Drink Combo","C",6000),    -- Fries and Drink
("Coffee Break Combo","D",8000);      -- Coffee and Coleslaw
```

Figure: Insert Into Comboset



```sql
SELECT * FROM restaurant.comboset;
```

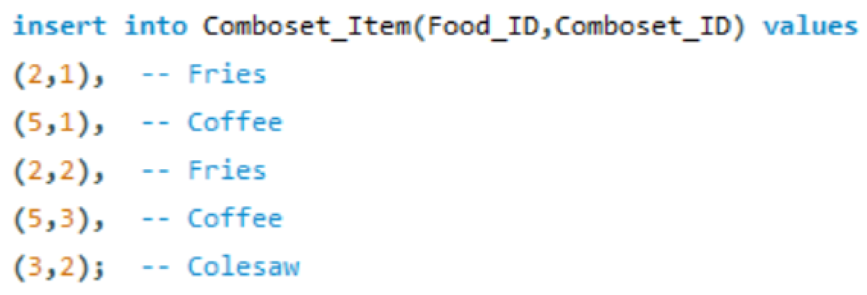


Comboset_ID	Cbo_Name	Cbo_Set	Price_with_tax
1	Burger Co...	A	7000
2	Burger an...	B	7500
3	Fries and ...	C	6000
4	Coffee Br...	D	8000
NULL	NULL	NULL	NULL

Figure: Result of Insert Into Comboset

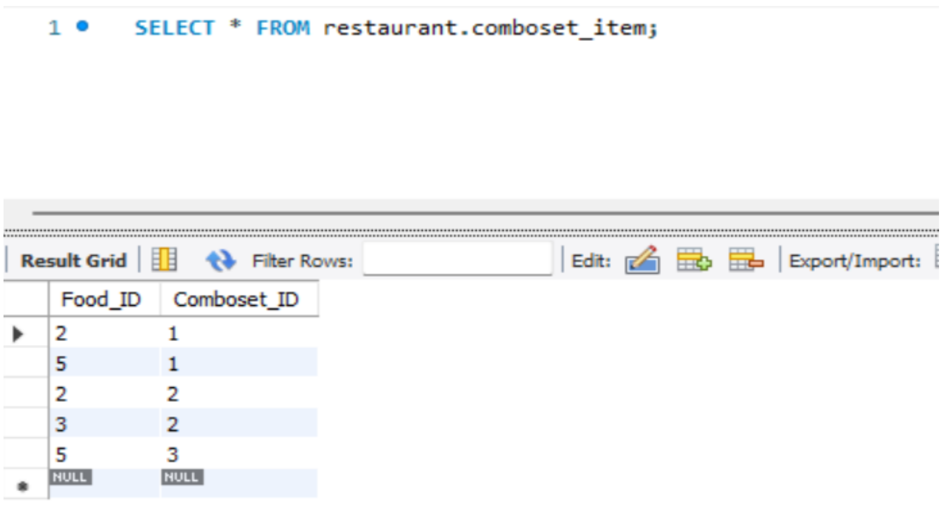
Insert into Comboset_Item

Comboset_Item Table: This join table connects specific foods to the combination sets with which they go together. By matching their IDs, each combination is linked to certain food products, defining the constituent parts of each combo.



```
insert into Comboset_Item(Food_ID,Comboset_ID) values
(2,1),  -- Fries
(5,1),  -- Coffee
(2,2),  -- Fries
(5,3),  -- Coffee
(3,2);  -- Colesaw
```

Figure: Insert Into Comboset_Item



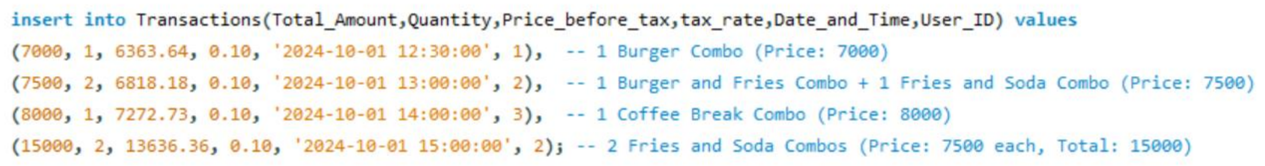
```
SELECT * FROM restaurant.comboset_item;
```

Food_ID	Comboset_ID
2	1
5	1
2	2
3	2
5	3
NULL	NULL

Figure: Result of Insert Into Comboset_Item

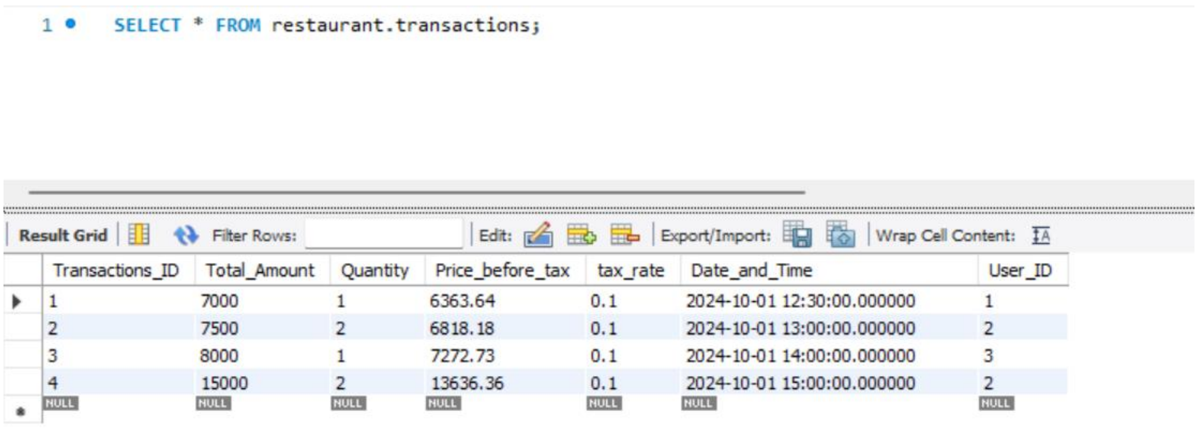
Insert into Transactions

Transactions Table: This table records sales information such as the amount sold, the number of items, the price before taxes, and the tax rate. Timestamped and associated with a particular user (cashier) who handled the sale is another feature of each transaction.



```
insert into Transactions(Total_Amount,Quantity,Price_before_tax,tax_rate,Date_and_Time,User_ID) values
(7000, 1, 6363.64, 0.10, '2024-10-01 12:30:00', 1),  -- 1 Burger Combo (Price: 7000)
(7500, 2, 6818.18, 0.10, '2024-10-01 13:00:00', 2),  -- 1 Burger and Fries Combo + 1 Fries and Soda Combo (Price: 7500)
(8000, 1, 7272.73, 0.10, '2024-10-01 14:00:00', 3),  -- 1 Coffee Break Combo (Price: 8000)
(15000, 2, 13636.36, 0.10, '2024-10-01 15:00:00', 2); -- 2 Fries and Soda Combos (Price: 7500 each, Total: 15000)
```

Figure: Insert Into Transactions



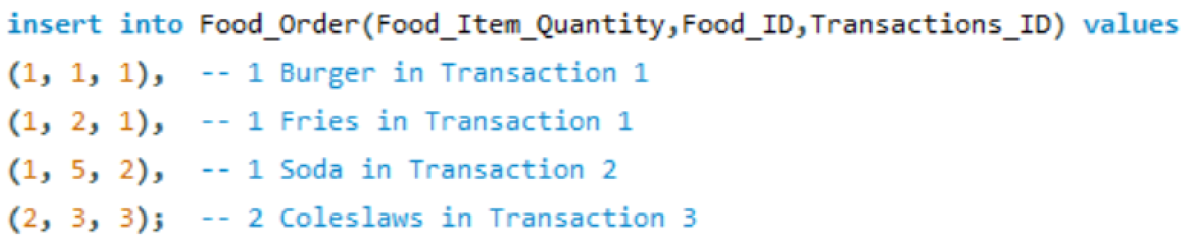
```
SELECT * FROM restaurant.transactions;
```

	Transactions_ID	Total_Amount	Quantity	Price_before_tax	tax_rate	Date_and_Time	User_ID
▶	1	7000	1	6363.64	0.1	2024-10-01 12:30:00.000000	1
	2	7500	2	6818.18	0.1	2024-10-01 13:00:00.000000	2
	3	8000	1	7272.73	0.1	2024-10-01 14:00:00.000000	3
	4	15000	2	13636.36	0.1	2024-10-01 15:00:00.000000	2
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure: Result of Insert Into Transactions

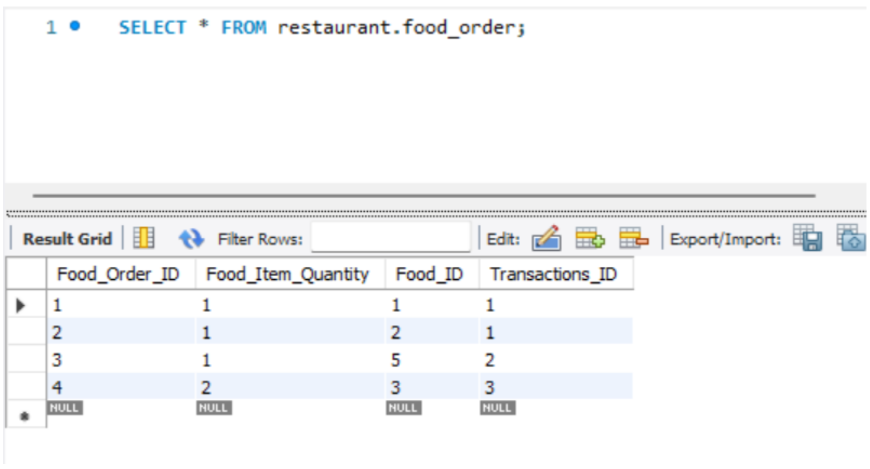
Insert into Food_Order

The Food_Order Table is a table that keeps track of the specific food items ordered in each transaction, along with the amount of each item. The Food and Transactions tables are connected to the Food_Order entries, guaranteeing that every order is connected to the appropriate transaction and goods.



```
insert into Food_Order(Food_Item_Quantity,Food_ID,Transactions_ID) values
(1, 1, 1),  -- 1 Burger in Transaction 1
(1, 2, 1),  -- 1 Fries in Transaction 1
(1, 5, 2),  -- 1 Soda in Transaction 2
(2, 3, 3);  -- 2 Coleslaws in Transaction 3
```

Figure: Insert Into Food_Order



```
SELECT * FROM restaurant.food_order;
```

	Food_Order_ID	Food_Item_Quantity	Food_ID	Transactions_ID
▶	1	1	1	1
	2	1	2	1
	3	1	5	2
	4	2	3	3
*	NULL	NULL	NULL	NULL

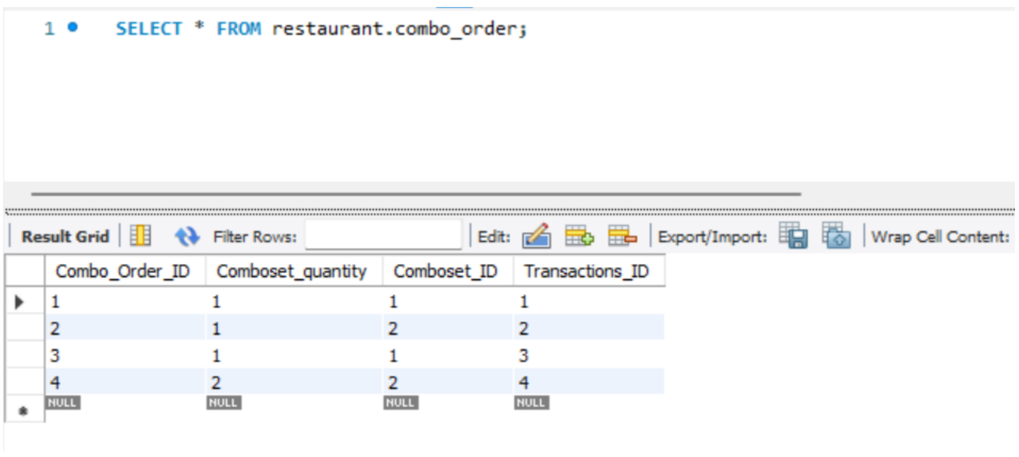
Figure: Result of Insert Into Food_Order

Insert into Combo_Order

Combo_Order Table: During transactions, combination sets are tracked in the Combo Order table. It allows for comprehensive tracking of combination sales by keeping track of the quantity of each combo requested and connecting these entries to the particular transaction as well as the combo set in question.

```
insert into Combo_Order(Comboset_quantity,Comboset_ID,Transactions_ID) values
(1, 1, 1),  -- 1 Burger Combo in Transaction 1 (Transaction 1 at 12:30 PM)
(1, 2, 2),  -- 1 Fries and Soda Combo in Transaction 2 (Transaction 2 at 01:00 PM)
(1, 1, 3),  -- 1 Burger Combo in Transaction 3 (Transaction 3 at 02:00 PM)
(2, 2, 4);  -- 2 Fries and Soda Combos in Transaction 4 (Transaction 4 at 03:00 PM)
```

Figure: Insert Into Combo_Order



```
SELECT * FROM restaurant.combo_order;
```

Combo_Order_ID	Comboset_quantity	Comboset_ID	Transactions_ID
1	1	1	1
2	1	2	2
3	1	1	3
4	2	2	4
NULL	NULL	NULL	NULL

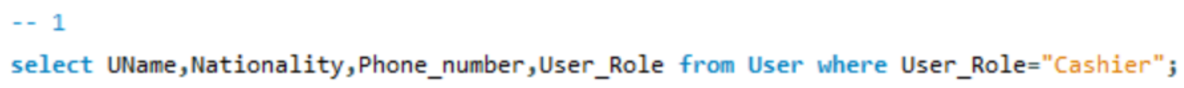
Figure: Result of Insert Into Combo_Order

Task-3-DML

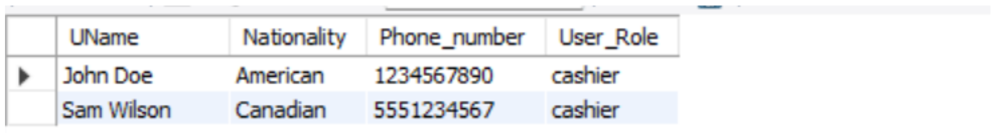
A collection of SQL commands known as DML (Data Manipulation Language) are used to work with the data kept in databases. You may use it to get, add, edit, and remove data. The data included in the database tables is directly altered by these procedures. There are four primary DML commands:

- **SELECT**: Retrieves data from the database. It allows you to query one or more tables to fetch specific data.
- **INSERT**: Adds new records (rows) into a table.
- **UPDATE**: Modifies existing data in a table.
- **DELETE**: Removes data from a table.

Q1.



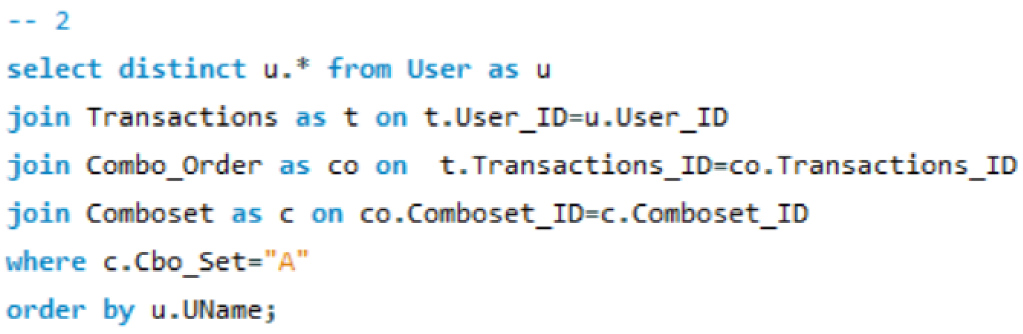
```sql
-- 1
select UName,Nationality,Phone_number,User_Role from User where User_Role="Cashier";
```



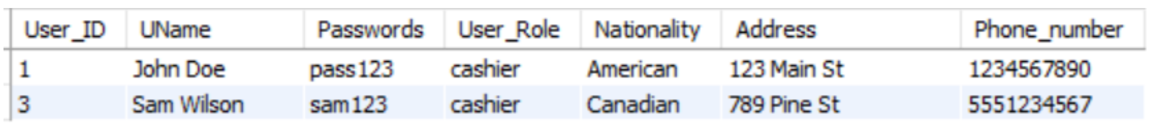
	UName	Nationality	Phone_number	User_Role
▶	John Doe	American	1234567890	cashier
	Sam Wilson	Canadian	5551234567	cashier

Figure: Query result showing the name, nationality, and phone number of all cashiers.

Q2.



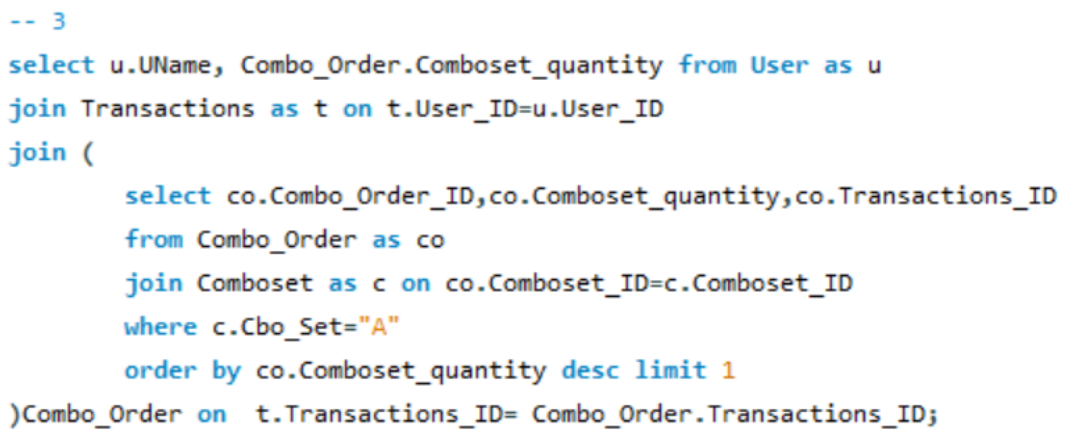
```sql
-- 2
select distinct u.* from User as u
join Transactions as t on t.User_ID=u.User_ID
join Combo_Order as co on  t.Transactions_ID=co.Transactions_ID
join Comboset as c on co.Comboset_ID=c.Comboset_ID
where c.Cbo_Set="A"
order by u.UName;
```



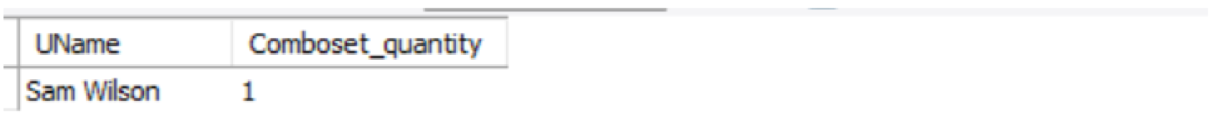
User_ID	UName	Passwords	User_Role	Nationality	Address	Phone_number
1	John Doe	pass123	cashier	American	123 Main St	1234567890
3	Sam Wilson	sam123	cashier	Canadian	789 Pine St	5551234567

Figure: Query result showing the details of cashier that sold the combo named “Combo A” in alphabetical order of cashier name.

Q3.



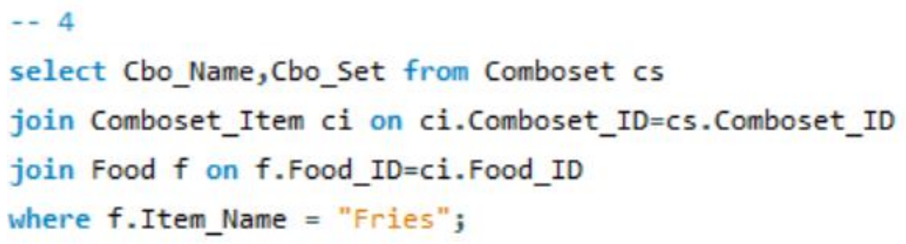
```
-- 3
select u.UName, Combo_Order.Comboset_quantity from User as u
join Transactions as t on t.User_ID=u.User_ID
join (
        select co.Combo_Order_ID,co.Comboset_quantity,co.Transactions_ID
        from Combo_Order as co
        join Comboset as c on co.Comboset_ID=c.Comboset_ID
        where c.Cbo_Set="A"
        order by co.Comboset_quantity desc limit 1
)Combo_Order on  t.Transactions_ID= Combo_Order.Transactions_ID;
```



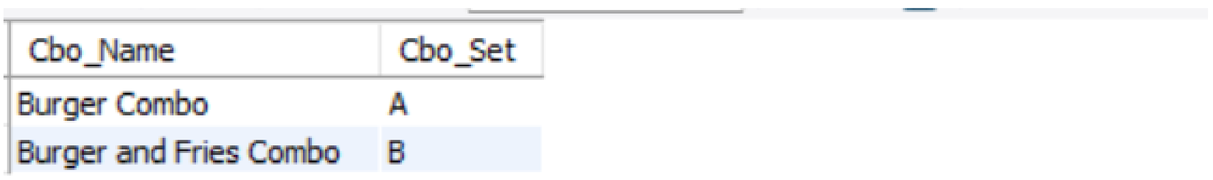
UName	Comboset_quantity
Sam Wilson	1

Figure: Query result showing “Which cashier sold the most ‘Combo A’ ”.

Q4.



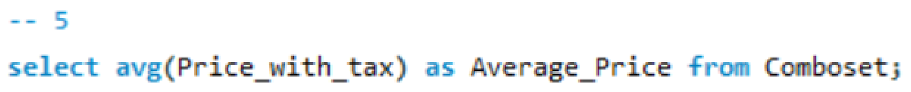
```
-- 4
select Cbo_Name,Cbo_Set from Comboset cs
join Comboset_Item ci on ci.Comboset_ID=cs.Comboset_ID
join Food f on f.Food_ID=ci.Food_ID
where f.Item_Name = "Fries";
```



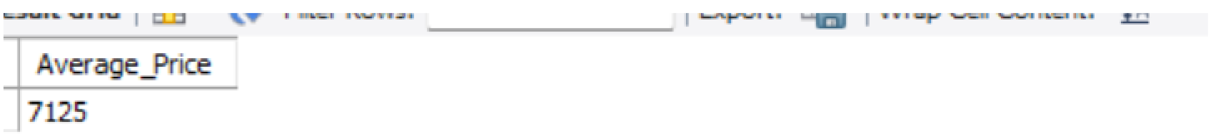
Cbo_Name	Cbo_Set
Burger Combo	A
Burger and Fries Combo	B

Figure: Query result showing all the combos that has “Fries” as one of its combo items.

Q5.



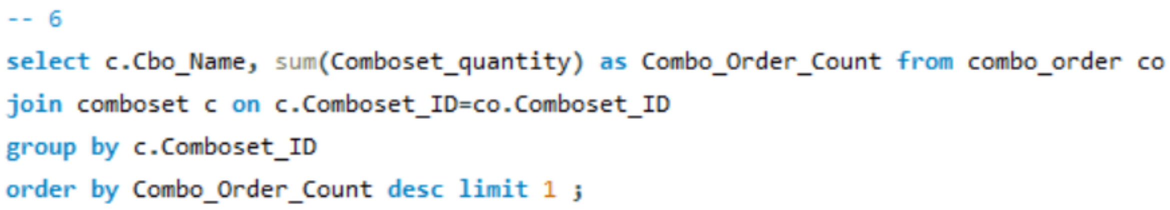
```
-- 5
select avg(Price_with_tax) as Average_Price from Comboset;
```



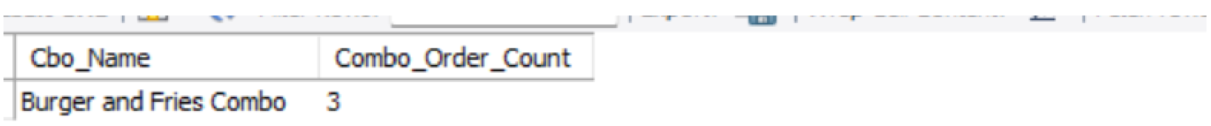
Average_Price
7125

Figure: Query result showing “What is the average price of each combo?”

Q6.



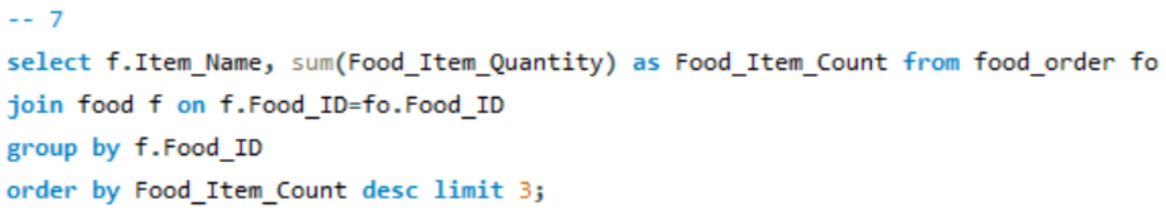
```
-- 6
select c.Cbo_Name, sum(Comboset_quantity) as Combo_Order_Count from combo_order co
join comboset c on c.Comboset_ID=co.Comboset_ID
group by c.Comboset_ID
order by Combo_Order_Count desc limit 1 ;
```



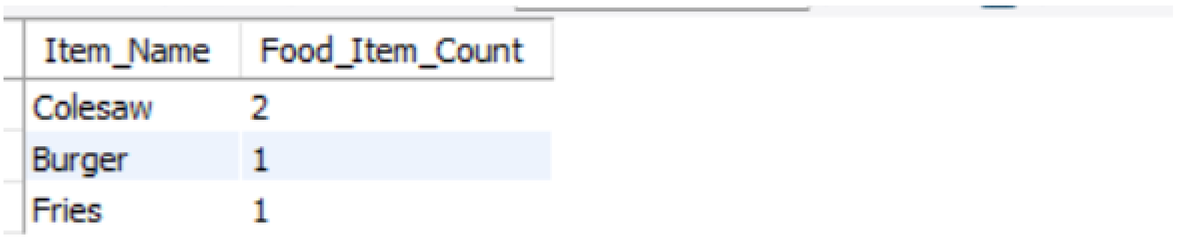
Cbo_Name	Combo_Order_Count
Burger and Fries Combo	3

Figure: Query result showing “Display the name of the most popular combo?”

Q7.



```
-- 7
select f.Item_Name, sum(Food_Item_Quantity) as Food_Item_Count from food_order fo
join food f on f.Food_ID=fo.Food_ID
group by f.Food_ID
order by Food_Item_Count desc limit 3;
```



Item_Name	Food_Item_Count
Colesaw	2
Burger	1
Fries	1

Figure: Query result showing “the first THREE most popular Ala-carte item and sort the result based on the most popular item on top.”