Project Title: Restaurant Data Analysis Management for Informed Decision-Making

Submission Date: 12/15/2023

Submitted By: Group 11

Group Members:

Soumith Roy Akula (02084467)

Uday Reddy Polepally (02115349)

Yashwanth Reddy Kistipati (02086419)

Advisor: Yukui Luo, PhD

Abstract

The Restaurant Data Analysis Management System is a comprehensive software solution designed to empower restaurant owners and managers with data-driven insights for informed decision-making. Recognizing the significance of operational efficiency, enhanced customer experiences, and data-driven decision-making in the competitive restaurant industry, this project focuses on creating a robust database and implementing data analysis techniques.

1. Introduction

1.1 Problem Statement

The Restaurant Data Analysis Management System is a comprehensive solution designed to address the challenges faced by restaurants in today's dynamic and competitive industry. The project aims to leverage the power of data analysis to enhance decision-making processes, streamline operations, and ultimately improve the overall performance and customer satisfaction of restaurants.

1.2 Project Objectives

Operational Efficiency Enhancement: Simplify daily restaurant operations, order handling, and menu management through effective data management and analysis.

Customer Experience Improvement: Understand customer preferences, order history, and ratings to make informed decisions for enhancing the overall customer experience.

Data-Driven Decision-Making: Facilitate data-driven decision-making by acquiring and analysing data related to menus, orders, and customer feedback.

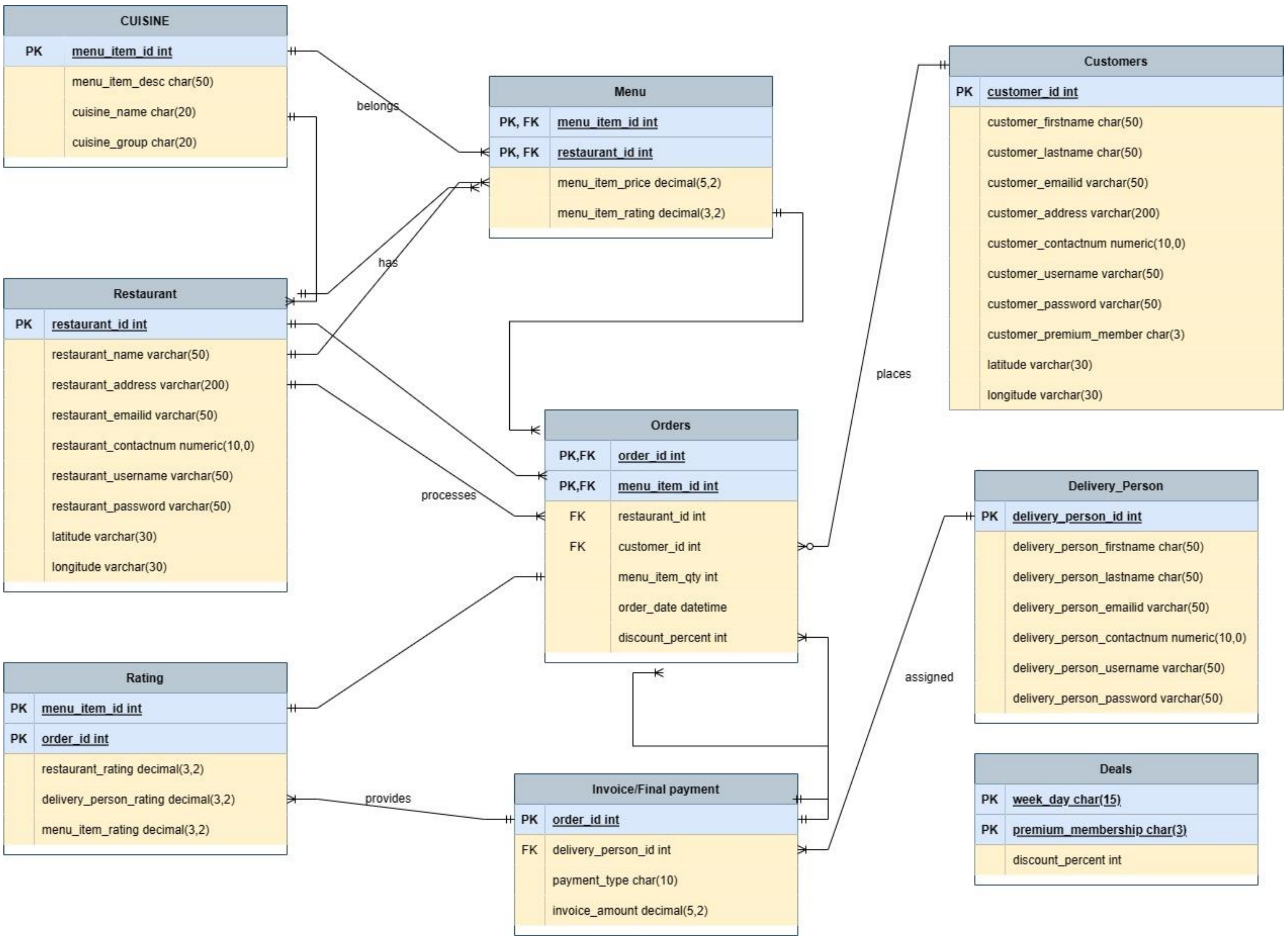
Secure User Access Control: Establish a secure user management system to control access to the database, ensuring data integrity and confidentiality.

Improved Customer Service: Utilize data analysis to identify the most frequently ordered menu items and customer preferences, leading to tailored offerings and improved customer service.

Efficient Order Management: Enable real-time tracking and management of incoming orders for restaurant owners, ensuring efficient order processing.

2. Database Design

2.1 Entity-Relationship Diagram


CUISINE
PK menu_item_id int
menu_item_desc char(50)
cuisine_name char(20)
cuisine_group char(20)
belongs
Menu
PK, FK menu_item_id int
PK, FK restaurant_id int
menu_item_price decimal(5,2)
menu_item_rating decimal(3,2)
Customers
PK customer_id int
customer_firstname char(50)
customer_lastname char(50)
customer_emailid varchar(50)
customer_address varchar(200)
customer_contactnum numeric(10,0)
customer_username varchar(50)
customer_password varchar(50)
customer_premium_member char(3)
latitude varchar(30)
longitude varchar(30)
has
Restaurant
PK restaurant_id int
restaurant_name varchar(50)
restaurant_address varchar(200)
restaurant_emailid varchar(50)
restaurant_contactnum numeric(10,0)
restaurant_username varchar(50)
restaurant_password varchar(50)
latitude varchar(30)
longitude varchar(30)
places
processes
Orders
PK,FK order_id int
PK,FK menu_item_id int
FK restaurant_id int
FK customer_id int
menu_item_qty int
order_date datetime
discount_percent int
Delivery_Person
PK delivery_person_id int
delivery_person_firstname char(50)
delivery_person_lastname char(50)
delivery_person_emailid varchar(50)
delivery_person_contactnum numeric(10,0)
delivery_person_username varchar(50)
delivery_person_password varchar(50)
assigned
Rating
PK menu_item_id int
PK order_id int
restaurant_rating decimal(3,2)
delivery_person_rating decimal(3,2)
menu_item_rating decimal(3,2)
provides
Invoice/Final payment
PK order_id int
FK delivery_person_id int
payment_type char(10)
invoice_amount decimal(5,2)
Deals
PK week_day char(15)
PK premium_membership char(3)
discount_percent int

Relationships:

a. Restaurant-Cuisine (1:M): One restaurant can offer multiple cuisines, represented by the Menu table's foreign key (menu_item_id) referencing the Cuisine table.

b. Restaurant-Menu (1:M): One restaurant can have multiple items in its menu, represented by the Menu table's foreign key (res_id) referencing the Restaurant table. Foreign Key(s): res_id in the "Menu" table references res_id in the "Restaurant" table. Description: This relationship establishes a connection between restaurants and their menus. It allows each restaurant to have a list of menu items associated with it.

c. Customer-Orders (1:M): One customer can place multiple orders, represented by the Orders table's foreign key (cus_id) referencing the Customer table. Foreign Key(s): cus_id in the "Orders" table references cus_id in the "Customer" table. Description: This relationship links each order to a specific customer, enabling the system to associate orders with individual customers.

d. Restaurant-Orders (1:M): One restaurant can receive multiple orders, represented by the Orders table's foreign key (res_id) referencing the Restaurant table. Foreign Key(s): res_id in the "Orders" table references res_id in the "Restaurant" table. Description: This relationship associates each order with a specific restaurant. It allows tracking orders back to the restaurant from which they were placed.

e. Delivery_Person-Invoice (1:M): One delivery person can be associated with multiple invoices, represented by the Invoice table's foreign key (dp_id) referencing the Delivery_Person table.

f. Orders-Invoice (1:1): Each order has a corresponding invoice, represented by the Invoice table's foreign key (order_id) referencing the Orders table.

g. Customer-Rating (1:M): One customer can provide multiple ratings, represented by the Rating table's foreign key (cus_id) referencing the Customer table.

h. Orders-Rating (1:1): Each order can have a corresponding rating, represented by the Rating table's foreign key (order_id) referencing the Orders table. Foreign Key(s): order_id, menu_item_id in the "Rating" table reference order_id, menu_item_id in the "Orders" table. Description: This relationship associates ratings with specific orders and menu items, enabling the system to store and retrieve feedback for both.

i. Menu-Rating (1:1): Each menu item can have a corresponding rating, represented by the Rating table's foreign key (menu_item_id) referencing the Menu table.

2.2 Table Descriptions

RESTAURANT Table: Stores information about different restaurants.
Attributes: res_id: Unique identifier for each restaurant.
res_name: Name of the restaurant.
res_location: Location or address of the restaurant.
res_contact: Contact information for the restaurant.
res_manager: Name of the restaurant manager.

CUSTOMER Table: Contains details about the customers who place orders.
Attributes: cus_id: Unique identifier for each customer.
cus_firstname: First name of the customer.
cus_lastname: Last name of the customer.
cus_contact: Contact information for the customer.

CUISINE Table: Holds information about different cuisines associated with menu items.
Attributes: cuisine_id: Unique identifier for each cuisine.
cuisine_name: Name of the cuisine.

MENU Table: Represents the menu items offered by each restaurant.
Attributes: menu_item_id: Unique identifier for each menu item.
menu_item_desc: Description of the menu item.
res_id: Foreign key referencing the restaurant to which the menu item belongs.
cuisine_id: Foreign key referencing the cuisine associated with the menu item.

ORDERS Table: Contains details about the orders placed by customers.
Attributes: order_id: Unique identifier for each order.
res_id: Foreign key referencing the restaurant receiving the order.
cus_id: Foreign key referencing the customer placing the order.
menu_item_id: Foreign key referencing the menu item in the order.

order_date: Date and time when the order was placed.

DELIVERY_PERSON Table: Stores information about the delivery personnel.
Attributes: dp_id: Unique identifier for each delivery person.
dp_firstname: First name of the delivery person.
dp_lastname: Last name of the delivery person.

RATING Table: Contains ratings provided by customers for various aspects.
Attributes: rating_id: Unique identifier for each rating.
order_id: Foreign key referencing the order for which the rating is given.
res_id: Foreign key referencing the restaurant being rated.
menu_item_id: Foreign key referencing the menu item being rated.
dp_id: Foreign key referencing the delivery person being rated.
restaurant_rating: Rating for the restaurant.
menu_item_rating: Rating for the menu item.
delivery_person_rating: Rating for the delivery person.

INVOICE Table: Contains details about invoices generated for orders.
Attributes: invoice_id: Unique identifier for each invoice.
order_id: Foreign key referencing the order for which the invoice is generated.
dp_id: Foreign key referencing the delivery person associated with the order.
Invoice_amount: Total amount in the invoice.
payment_type: Type of payment used.

DEALS Table: Table stores information about special promotions and discounts offered by restaurants.
Attributes: week_day: Represents the day of the week for which the deal is applicable.
premium_membership: Indicates whether the deal is exclusive to premium members.
discount_percent: Specifies the percentage of discount offered in the deal.

3. Data Collection

3.1 Sample Data Generation:

For the purpose of meaningful analysis and system testing, realistic sample data has been generated to simulate various scenarios within the Restaurant Data Analysis Management system. This sample data covers diverse aspects of the system, including information about restaurants, customers, menu items, orders, delivery personnel, deals, invoices, and ratings. This ensures that the database is representative of real-world scenarios, allowing for comprehensive testing and analysis.

3.2 Data Initialization Process:

The data initialization process involves populating the tables with the generated sample data. This step is crucial for creating a functional and representative database that mirrors the actual usage of the system. The sample data is inserted into corresponding tables using SQL INSERT statements, establishing relationships between entities through foreign keys. This process ensures that the database is well-structured and contains meaningful information for subsequent analysis and testing.

By executing the provided SQL commands, the system is initialized with sample data, enabling users to interact with the application in a manner that closely resembles real-world usage. This initialization process is fundamental for evaluating the system's performance, functionality, and effectiveness in handling various scenarios.

i. **CREATE TABLE:** This command is used to create tables and define columns, primary keys, foreign keys, and constraints.

ii. **ALTER TABLE:** It allows modification of an existing table, including adding or dropping columns, defining primary and foreign keys, etc.

iii. **JOIN:** SQL uses JOIN operations (e.g., INNER JOIN, LEFT JOIN, RIGHT JOIN) to combine rows from multiple tables based on related columns between them.

iv. **FOREIGN KEY Constraint:** This constraint ensures the referential integrity of the data ina table by specifying that the values in a column must match values in a related table's primary key.

There are mainly three types of relationships used:

● one-to-one relationships

● one-to-many relationships

● many-to-many relationships

By establishing and maintaining these relationships using keys and SQL commands, databases organize data efficiently and ensure data consistency across multiple tables.

3.3 Data Integrity:

Data integrity in the database is maintained using:

● **Referential Integrity:** This is maintained through foreign keys. It ensures the relationships between tables are preserved. A foreign key in one table points to the primary key in another table.

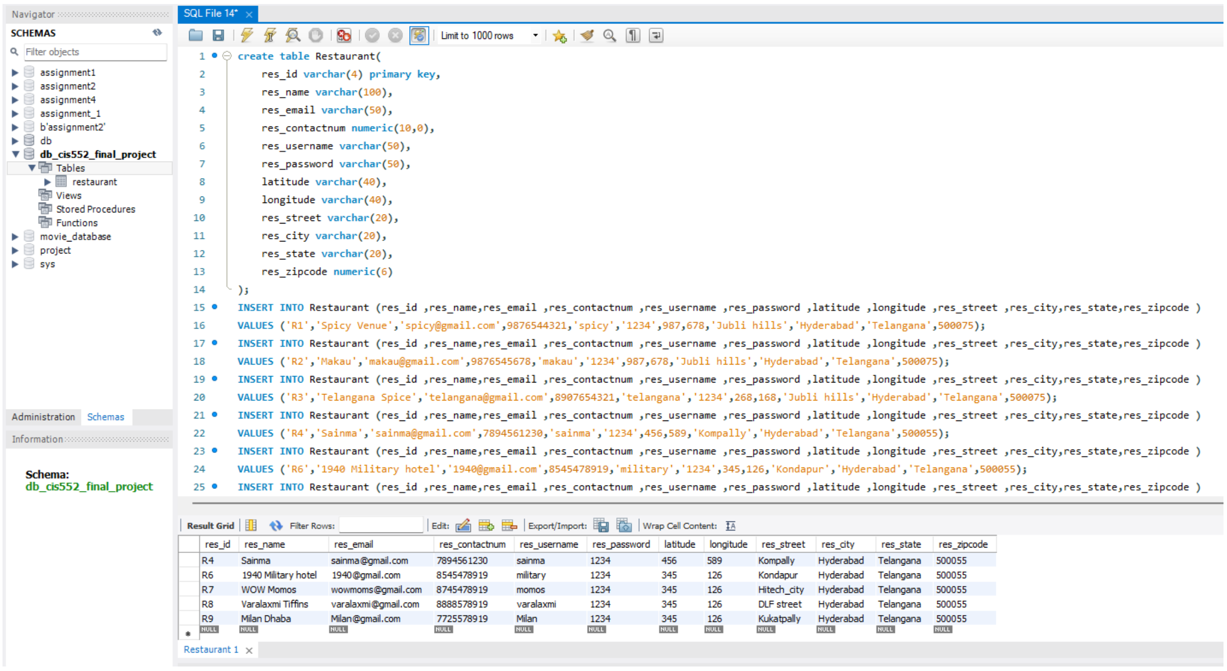
● **Constraints:** SQL provides various constraints like NOT NULL, UNIQUE, CHECK, and DEFAULT constraints. These enforce rules at the column level.

● **Indexes:** While not directly related to integrity, indexes improve data retrieval performance. They ensure that data is stored and retrieved efficiently while maintaining the integrity of relationships.

● **Triggers:** Triggers are special stored procedures that are automatically executed or fired when certain events occur (like an INSERT, UPDATE, or DELETE operation). They help maintain data integrity by allowing custom checks, modifications, or actions to be performed when specific conditions are met.

Using above rules for Data collection, Data initialization and Data integrity methods we have created the tables for our database. The following are the screenshots of database tables that we crated and inserted values.

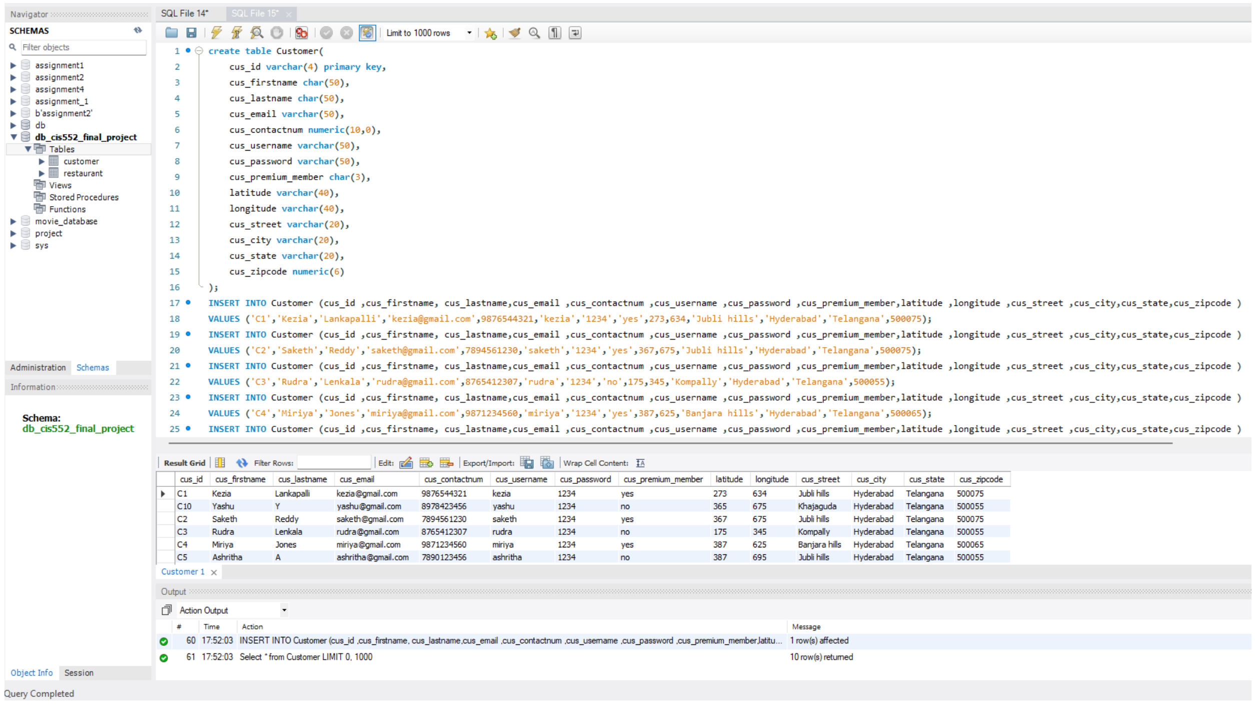
1. Restaurant table



```sql
create table Restaurant(
    res_id varchar(4) primary key,
    res_name varchar(100),
    res_email varchar(50),
    res_contactnum numeric(10,0),
    res_username varchar(50),
    res_password varchar(50),
    latitude varchar(40),
    longitude varchar(40),
    res_street varchar(20),
    res_city varchar(20),
    res_state varchar(20),
    res_zipcode numeric(6)
);
INSERT INTO Restaurant (res_id ,res_name,res_email ,res_contactnum ,res_username ,res_password ,latitude ,longitude ,res_street ,res_city,res_state,res_zipcode )
VALUES ('R1','Spicy Venue','spicy@gmail.com',9876544321,'spicy','1234',987,678,'Jubli hills','Hyderabad','Telangana',500075);
INSERT INTO Restaurant (res_id ,res_name,res_email ,res_contactnum ,res_username ,res_password ,latitude ,longitude ,res_street ,res_city,res_state,res_zipcode )
VALUES ('R2','Makau','makau@gmail.com',9876545678,'makau','1234',987,678,'Jubli hills','Hyderabad','Telangana',500075);
INSERT INTO Restaurant (res_id ,res_name,res_email ,res_contactnum ,res_username ,res_password ,latitude ,longitude ,res_street ,res_city,res_state,res_zipcode )
VALUES ('R3','Telangana Spice','telangana@gmail.com',8907654321,'telangana','1234',268,168,'Jubli hills','Hyderabad','Telangana',500075);
INSERT INTO Restaurant (res_id ,res_name,res_email ,res_contactnum ,res_username ,res_password ,latitude ,longitude ,res_street ,res_city,res_state,res_zipcode )
VALUES ('R4','Sainma','sainma@gmail.com',7894561230,'sainma','1234',456,589,'Kompally','Hyderabad','Telangana',500055);
INSERT INTO Restaurant (res_id ,res_name,res_email ,res_contactnum ,res_username ,res_password ,latitude ,longitude ,res_street ,res_city,res_state,res_zipcode )
VALUES ('R6','1940 Military hotel','1940@gmail.com',8545478919,'military','1234',345,126,'Kondapur','Hyderabad','Telangana',500055);
INSERT INTO Restaurant (res_id ,res_name,res_email ,res_contactnum ,res_username ,res_password ,latitude ,longitude ,res_street ,res_city,res_state,res_zipcode )
```

res_id	res_name	res_email	res_contactnum	res_username	res_password	latitude	longitude	res_street	res_city	res_state	res_zipcode
R4	Sainma	sainma@gmail.com	7894561230	sainma	1234	456	589	Kompally	Hyderabad	Telangana	500055
R6	1940 Military hotel	1940@gmail.com	8545478919	military	1234	345	126	Kondapur	Hyderabad	Telangana	500055
R7	WOW Momos	wowmoms@gmail.com	8745478919	momos	1234	345	126	Hitech_city	Hyderabad	Telangana	500055
R8	Varalaxmi Tiffins	varalaxmi@gmail.com	8888578919	varalaxmi	1234	345	126	DLF street	Hyderabad	Telangana	500055
R9	Milan Dhaba	Milan@gmail.com	7725578919	Milan	1234	345	126	Kukatpally	Hyderabad	Telangana	500055
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

2. Customer Table



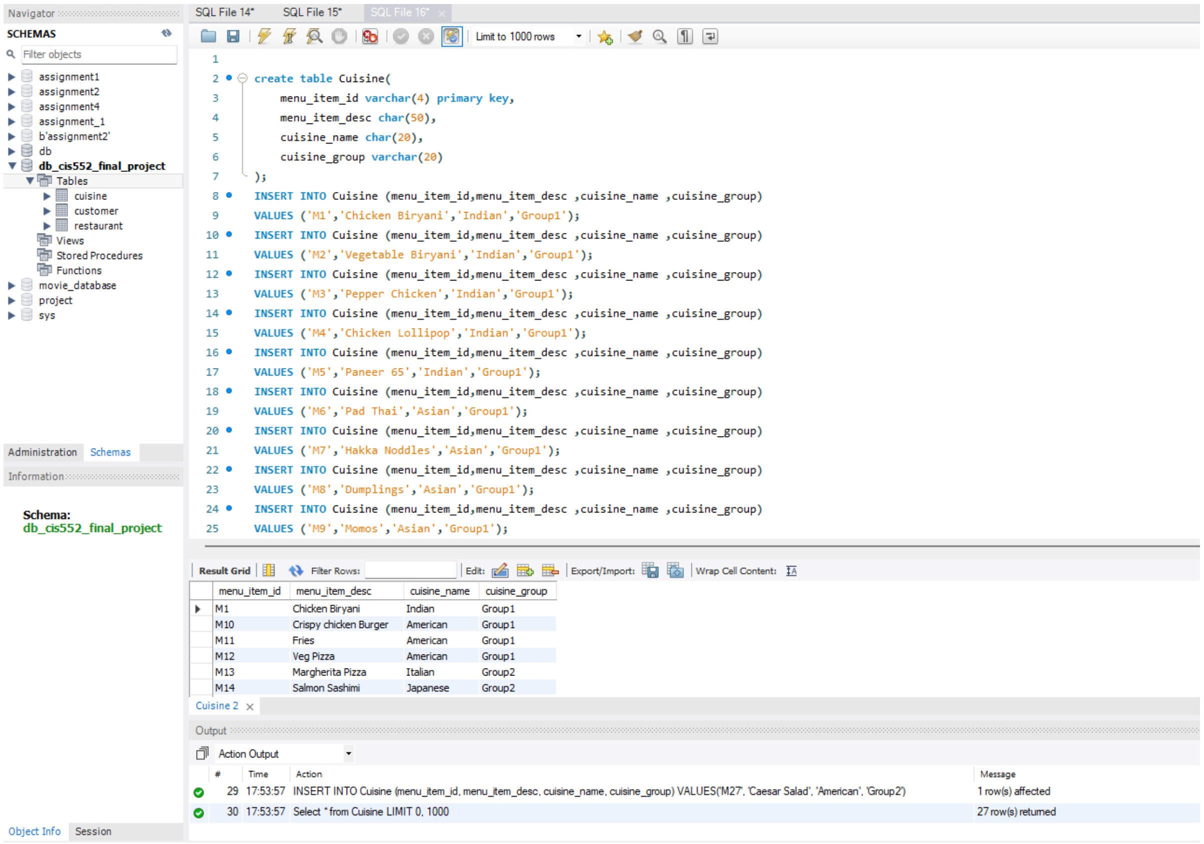
```sql
create table Customer(
    cus_id varchar(4) primary key,
    cus_firstname char(50),
    cus_lastname char(50),
    cus_email varchar(50),
    cus_contactnum numeric(10,0),
    cus_username varchar(50),
    cus_password varchar(50),
    cus_premium_member char(3),
    latitude varchar(40),
    longitude varchar(40),
    cus_street varchar(20),
    cus_city varchar(20),
    cus_state varchar(20),
    cus_zipcode numeric(6)
);
INSERT INTO Customer (cus_id ,cus_firstname, cus_lastname,cus_email ,cus_contactnum ,cus_username ,cus_password ,cus_premium_member,latitude ,longitude ,cus_street ,cus_city,cus_state,cus_zipcode )
VALUES ('C1','Kezia','Lankapalli','kezia@gmail.com',9876544321,'kezia','1234','yes',273,634,'Jubli hills','Hyderabad','Telangana',500075);
INSERT INTO Customer (cus_id ,cus_firstname, cus_lastname,cus_email ,cus_contactnum ,cus_username ,cus_password ,cus_premium_member,latitude ,longitude ,cus_street ,cus_city,cus_state,cus_zipcode )
VALUES ('C2','Saketh','Reddy','saketh@gmail.com',7894561230,'saketh','1234','yes',367,675,'Jubli hills','Hyderabad','Telangana',500075);
INSERT INTO Customer (cus_id ,cus_firstname, cus_lastname,cus_email ,cus_contactnum ,cus_username ,cus_password ,cus_premium_member,latitude ,longitude ,cus_street ,cus_city,cus_state,cus_zipcode )
VALUES ('C3','Rudra','Lenkala','rudra@gmail.com',8765412307,'rudra','1234','no',175,345,'Kompally','Hyderabad','Telangana',500055);
INSERT INTO Customer (cus_id ,cus_firstname, cus_lastname,cus_email ,cus_contactnum ,cus_username ,cus_password ,cus_premium_member,latitude ,longitude ,cus_street ,cus_city,cus_state,cus_zipcode )
VALUES ('C4','Miriya','Jones','miriya@gmail.com',9871234560,'miriya','1234','yes',387,625,'Banjara hills','Hyderabad','Telangana',500065);
INSERT INTO Customer (cus_id ,cus_firstname, cus_lastname,cus_email ,cus_contactnum ,cus_username ,cus_password ,cus_premium_member,latitude ,longitude ,cus_street ,cus_city,cus_state,cus_zipcode )
```

cus_id	cus_firstname	cus_lastname	cus_email	cus_contactnum	cus_username	cus_password	cus_premium_member	latitude	longitude	cus_street	cus_city	cus_state	cus_zipcode
C1	Kezia	Lankapalli	kezia@gmail.com	9876544321	kezia	1234	yes	273	634	Jubli hills	Hyderabad	Telangana	500075
C10	Yashu	Y	yashu@gmail.com	8978423456	yashu	1234	no	365	675	Khajaguda	Hyderabad	Telangana	500055
C2	Saketh	Reddy	saketh@gmail.com	7894561230	saketh	1234	yes	367	675	Jubli hills	Hyderabad	Telangana	500075
C3	Rudra	Lenkala	rudra@gmail.com	8765412307	rudra	1234	no	175	345	Kompally	Hyderabad	Telangana	500055
C4	Miriya	Jones	miriya@gmail.com	9871234560	miriya	1234	yes	387	625	Banjara hills	Hyderabad	Telangana	500065
C5	Ashritha	A	ashritha@gmail.com	7890123456	ashritha	1234	no	387	695	Jubli hills	Hyderabad	Telangana	500055

```
#   Time      Action                                                                                                              Message
60  17:52:03  INSERT INTO Customer (cus_id ,cus_firstname, cus_lastname,cus_email ,cus_contactnum ,cus_username ,cus_password ,cus_premium_member,latitu...  1 row(s) affected
61  17:52:03  Select * from Customer LIMIT 0, 1000                                                                                 10 row(s) returned
```

Query Completed

3. Cuisine Table

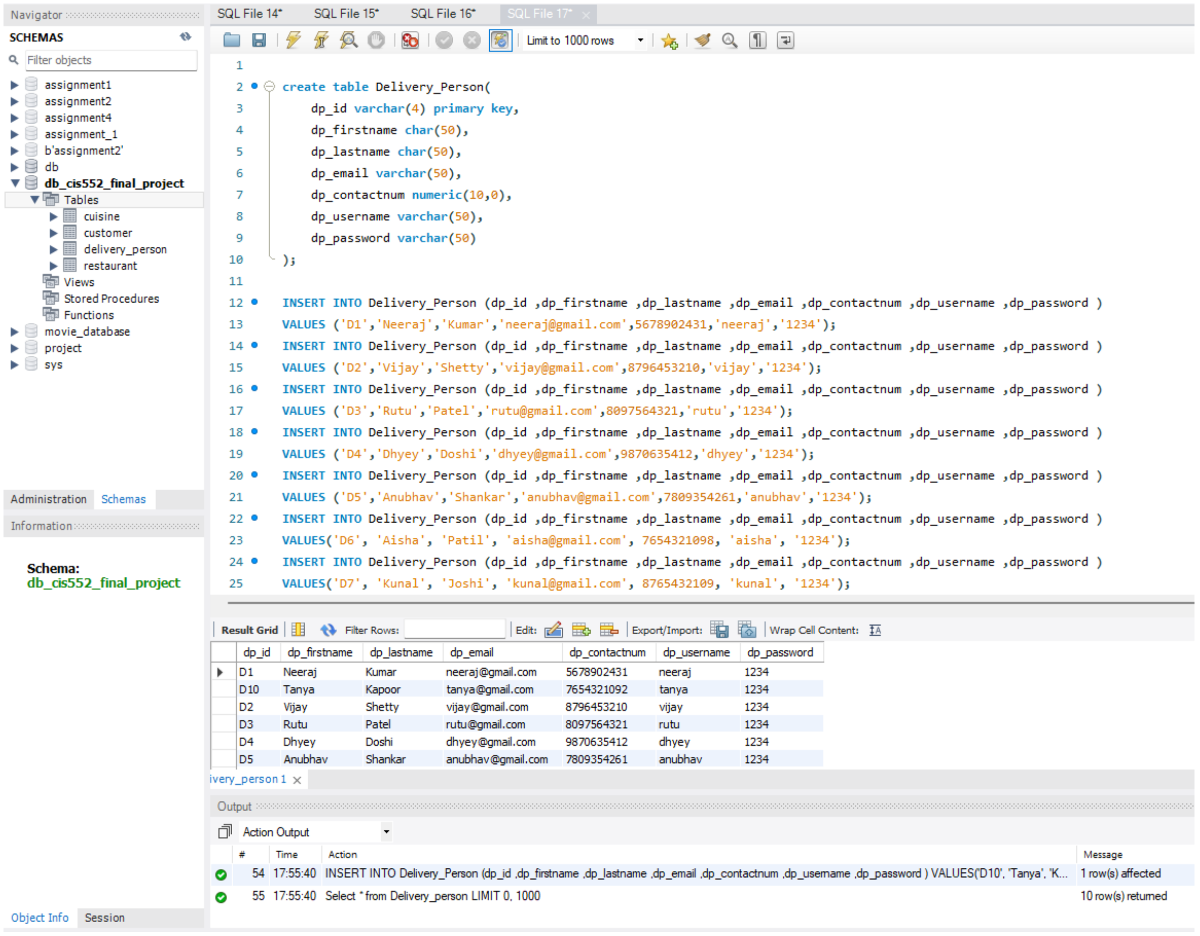


```sql
create table Cuisine(
    menu_item_id varchar(4) primary key,
    menu_item_desc char(50),
    cuisine_name char(20),
    cuisine_group varchar(20)
);
INSERT INTO Cuisine (menu_item_id,menu_item_desc ,cuisine_name ,cuisine_group)
VALUES ('M1','Chicken Biryani','Indian','Group1');
INSERT INTO Cuisine (menu_item_id,menu_item_desc ,cuisine_name ,cuisine_group)
VALUES ('M2','Vegetable Biryani','Indian','Group1');
INSERT INTO Cuisine (menu_item_id,menu_item_desc ,cuisine_name ,cuisine_group)
VALUES ('M3','Pepper Chicken','Indian','Group1');
INSERT INTO Cuisine (menu_item_id,menu_item_desc ,cuisine_name ,cuisine_group)
VALUES ('M4','Chicken Lollipop','Indian','Group1');
INSERT INTO Cuisine (menu_item_id,menu_item_desc ,cuisine_name ,cuisine_group)
VALUES ('M5','Paneer 65','Indian','Group1');
INSERT INTO Cuisine (menu_item_id,menu_item_desc ,cuisine_name ,cuisine_group)
VALUES ('M6','Pad Thai','Asian','Group1');
INSERT INTO Cuisine (menu_item_id,menu_item_desc ,cuisine_name ,cuisine_group)
VALUES ('M7','Hakka Noddles','Asian','Group1');
INSERT INTO Cuisine (menu_item_id,menu_item_desc ,cuisine_name ,cuisine_group)
VALUES ('M8','Dumplings','Asian','Group1');
INSERT INTO Cuisine (menu_item_id,menu_item_desc ,cuisine_name ,cuisine_group)
VALUES ('M9','Momos','Asian','Group1');
```

menu_item_id	menu_item_desc	cuisine_name	cuisine_group
M1	Chicken Biryani	Indian	Group1
M10	Crispy chicken Burger	American	Group1
M11	Fries	American	Group1
M12	Veg Pizza	American	Group1
M13	Margherita Pizza	Italian	Group2
M14	Salmon Sashimi	Japanese	Group2

```
#   Time      Action                                                                                                              Message
29  17:53:57  INSERT INTO Cuisine (menu_item_id, menu_item_desc, cuisine_name, cuisine_group) VALUES('M27', 'Caesar Salad', 'American', 'Group2')  1 row(s) affected
30  17:53:57  Select * from Cuisine LIMIT 0, 1000                                                                                 27 row(s) returned
```

4. Delivery_Person Table

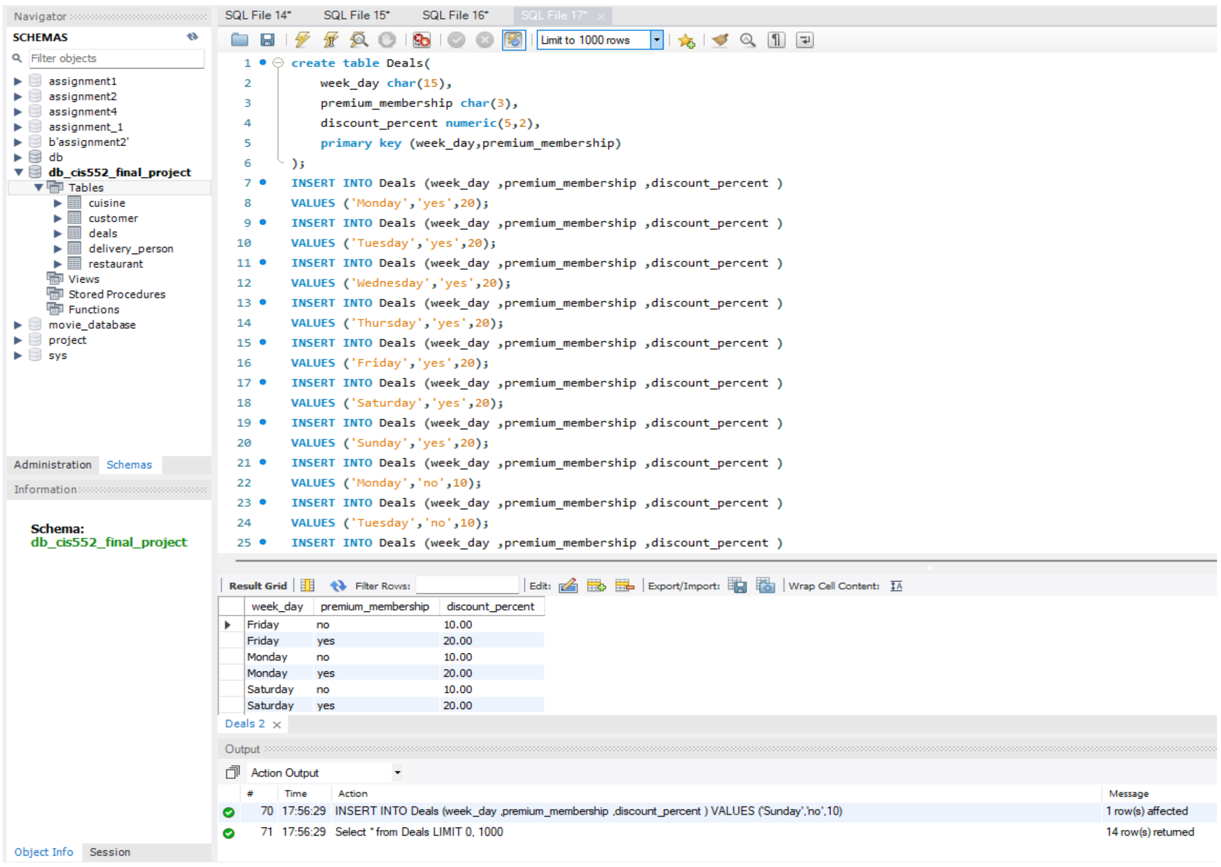


```sql
create table Delivery_Person(
    dp_id varchar(4) primary key,
    dp_firstname char(50),
    dp_lastname char(50),
    dp_email varchar(50),
    dp_contactnum numeric(10,0),
    dp_username varchar(50),
    dp_password varchar(50)
);

INSERT INTO Delivery_Person (dp_id ,dp_firstname ,dp_lastname ,dp_email ,dp_contactnum ,dp_username ,dp_password )
VALUES ('D1','Neeraj','Kumar','neeraj@gmail.com',5678902431,'neeraj','1234');
INSERT INTO Delivery_Person (dp_id ,dp_firstname ,dp_lastname ,dp_email ,dp_contactnum ,dp_username ,dp_password )
VALUES ('D2','Vijay','Shetty','vijay@gmail.com',8796453210,'vijay','1234');
INSERT INTO Delivery_Person (dp_id ,dp_firstname ,dp_lastname ,dp_email ,dp_contactnum ,dp_username ,dp_password )
VALUES ('D3','Rutu','Patel','rutu@gmail.com',8097564321,'rutu','1234');
INSERT INTO Delivery_Person (dp_id ,dp_firstname ,dp_lastname ,dp_email ,dp_contactnum ,dp_username ,dp_password )
VALUES ('D4','Dhyey','Doshi','dhyey@gmail.com',9870635412,'dhyey','1234');
INSERT INTO Delivery_Person (dp_id ,dp_firstname ,dp_lastname ,dp_email ,dp_contactnum ,dp_username ,dp_password )
VALUES ('D5','Anubhav','Shankar','anubhav@gmail.com',7809354261,'anubhav','1234');
INSERT INTO Delivery_Person (dp_id ,dp_firstname ,dp_lastname ,dp_email ,dp_contactnum ,dp_username ,dp_password )
VALUES('D6', 'Aisha', 'Patil', 'aisha@gmail.com', 7654321098, 'aisha', '1234');
INSERT INTO Delivery_Person (dp_id ,dp_firstname ,dp_lastname ,dp_email ,dp_contactnum ,dp_username ,dp_password )
VALUES('D7', 'Kunal', 'Joshi', 'kunal@gmail.com', 8765432109, 'kunal', '1234');
```

dp_id	dp_firstname	dp_lastname	dp_email	dp_contactnum	dp_username	dp_password
D1	Neeraj	Kumar	neeraj@gmail.com	5678902431	neeraj	1234
D10	Tanya	Kapoor	tanya@gmail.com	7654321092	tanya	1234
D2	Vijay	Shetty	vijay@gmail.com	8796453210	vijay	1234
D3	Rutu	Patel	rutu@gmail.com	8097564321	rutu	1234
D4	Dhyey	Doshi	dhyey@gmail.com	9870635412	dhyey	1234
D5	Anubhav	Shankar	anubhav@gmail.com	7809354261	anubhav	1234

```
#   Time      Action                                                                                                                          Message
54  17:55:40  INSERT INTO Delivery_Person (dp_id ,dp_firstname ,dp_lastname ,dp_email ,dp_contactnum ,dp_username ,dp_password ) VALUES('D10', 'Tanya', 'K...  1 row(s) affected
55  17:55:40  Select * from Delivery_person LIMIT 0, 1000                                                                                     10 row(s) returned
```

5. Deals Table



```sql
create table Deals(
    week_day char(15),
    premium_membership char(3),
    discount_percent numeric(5,2),
    primary key (week_day,premium_membership)
  );
INSERT INTO Deals (week_day ,premium_membership ,discount_percent )
VALUES ('Monday','yes',20);
INSERT INTO Deals (week_day ,premium_membership ,discount_percent )
VALUES ('Tuesday','yes',20);
INSERT INTO Deals (week_day ,premium_membership ,discount_percent )
VALUES ('Wednesday','yes',20);
INSERT INTO Deals (week_day ,premium_membership ,discount_percent )
VALUES ('Thursday','yes',20);
INSERT INTO Deals (week_day ,premium_membership ,discount_percent )
VALUES ('Friday','yes',20);
INSERT INTO Deals (week_day ,premium_membership ,discount_percent )
VALUES ('Saturday','yes',20);
INSERT INTO Deals (week_day ,premium_membership ,discount_percent )
VALUES ('Sunday','yes',20);
INSERT INTO Deals (week_day ,premium_membership ,discount_percent )
VALUES ('Monday','no',10);
INSERT INTO Deals (week_day ,premium_membership ,discount_percent )
VALUES ('Tuesday','no',10);
INSERT INTO Deals (week_day ,premium_membership ,discount_percent )
```

Result Grid

week_day	premium_membership	discount_percent
Friday	no	10.00
Friday	yes	20.00
Monday	no	10.00
Monday	yes	20.00
Saturday	no	10.00
Saturday	yes	20.00

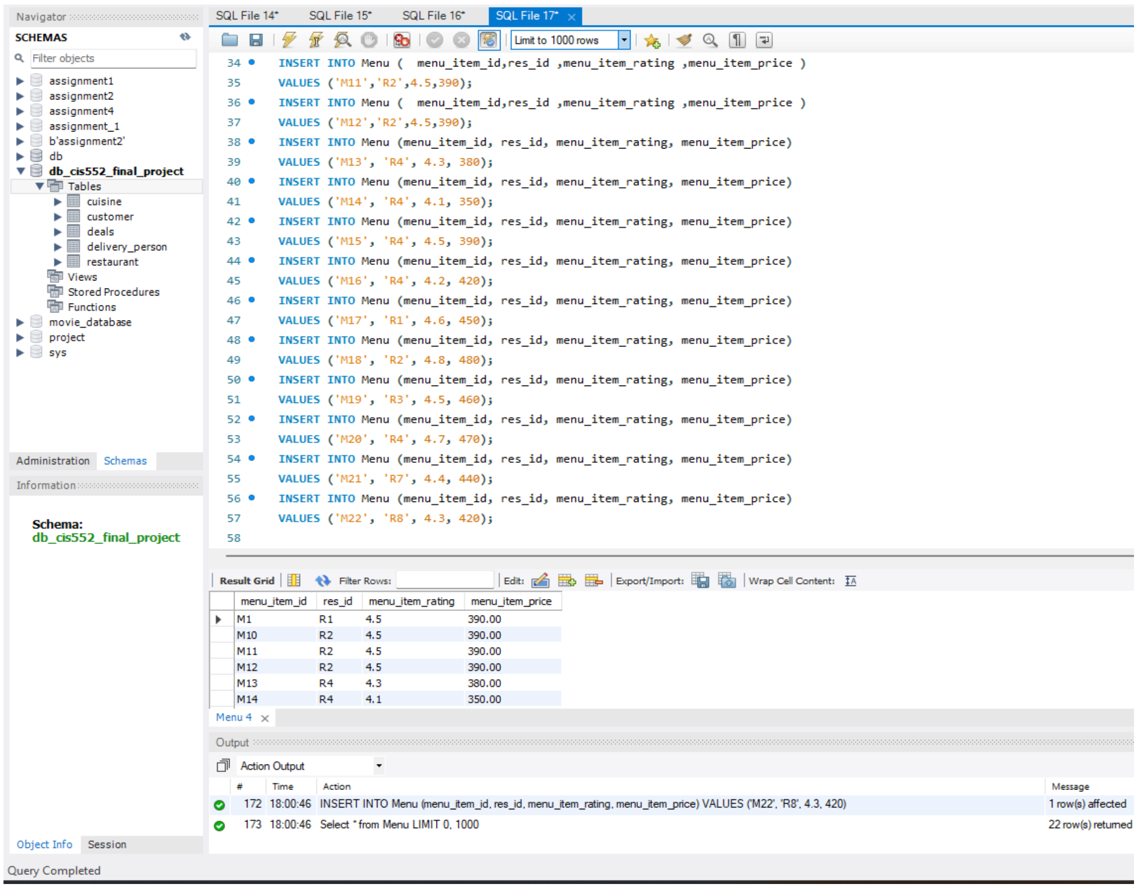
Deals 2

Output

Action Output

```
#   Time      Action                                                                                        Message
70  17:56:29  INSERT INTO Deals (week_day ,premium_membership ,discount_percent ) VALUES ('Sunday','no',10)  1 row(s) affected
71  17:56:29  Select * from Deals LIMIT 0, 1000                                                             14 row(s) returned
```

6. Menu Table



```sql
INSERT INTO Menu (  menu_item_id,res_id ,menu_item_rating ,menu_item_price )
VALUES ('M11','R2',4.5,390);
INSERT INTO Menu (  menu_item_id,res_id ,menu_item_rating ,menu_item_price )
VALUES ('M12','R2',4.5,390);
INSERT INTO Menu (menu_item_id, res_id, menu_item_rating, menu_item_price)
VALUES ('M13', 'R4', 4.3, 380);
INSERT INTO Menu (menu_item_id, res_id, menu_item_rating, menu_item_price)
VALUES ('M14', 'R4', 4.1, 350);
INSERT INTO Menu (menu_item_id, res_id, menu_item_rating, menu_item_price)
VALUES ('M15', 'R4', 4.5, 390);
INSERT INTO Menu (menu_item_id, res_id, menu_item_rating, menu_item_price)
VALUES ('M16', 'R4', 4.2, 420);
INSERT INTO Menu (menu_item_id, res_id, menu_item_rating, menu_item_price)
VALUES ('M17', 'R1', 4.6, 450);
INSERT INTO Menu (menu_item_id, res_id, menu_item_rating, menu_item_price)
VALUES ('M18', 'R2', 4.8, 480);
INSERT INTO Menu (menu_item_id, res_id, menu_item_rating, menu_item_price)
VALUES ('M19', 'R3', 4.5, 460);
INSERT INTO Menu (menu_item_id, res_id, menu_item_rating, menu_item_price)
VALUES ('M20', 'R4', 4.7, 470);
INSERT INTO Menu (menu_item_id, res_id, menu_item_rating, menu_item_price)
VALUES ('M21', 'R7', 4.4, 440);
INSERT INTO Menu (menu_item_id, res_id, menu_item_rating, menu_item_price)
VALUES ('M22', 'R8', 4.3, 420);
```

Result Grid

menu_item_id	res_id	menu_item_rating	menu_item_price
M1	R1	4.5	390.00
M10	R2	4.5	390.00
M11	R2	4.5	390.00
M12	R2	4.5	390.00
M13	R4	4.3	380.00
M14	R4	4.1	350.00

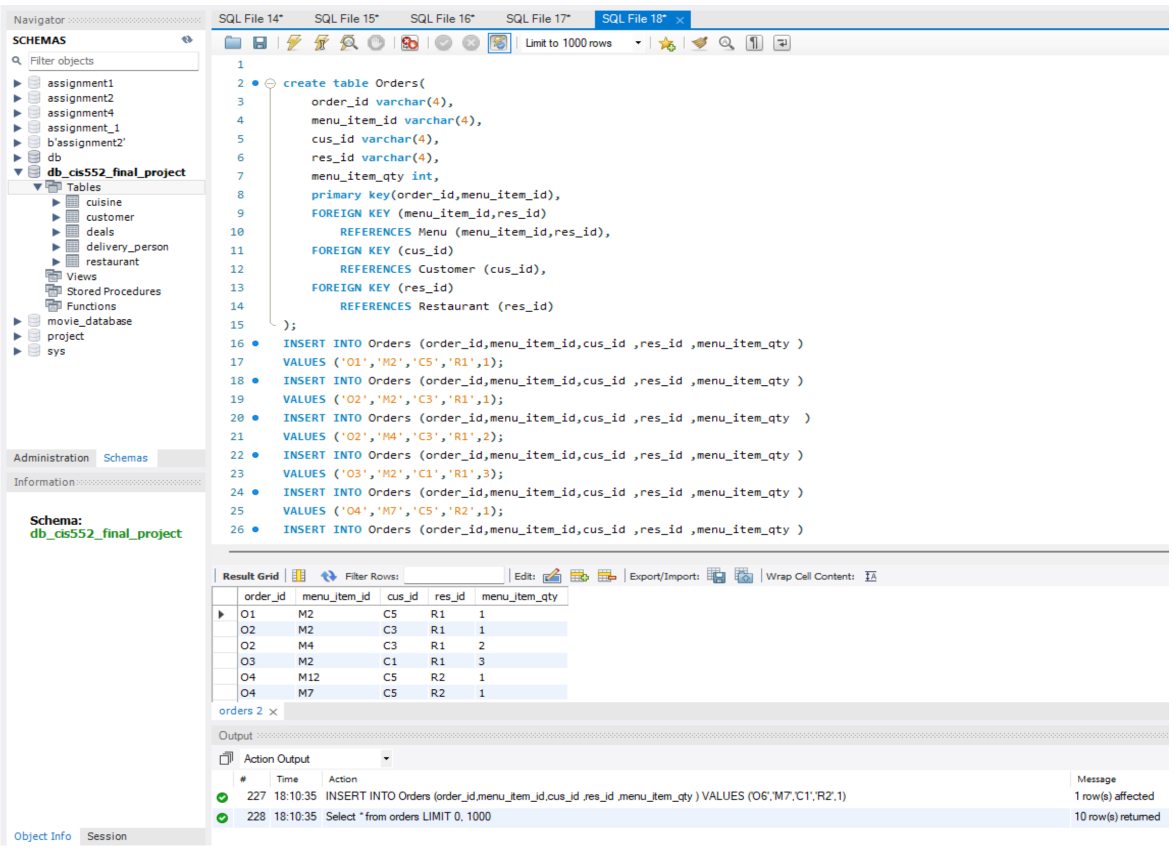
Menu 4

Output

Action Output

```
#    Time      Action                                                                                               Message
172  18:00:46  INSERT INTO Menu (menu_item_id, res_id, menu_item_rating, menu_item_price) VALUES ('M22', 'R8', 4.3, 420)  1 row(s) affected
173  18:00:46  Select * from Menu LIMIT 0, 1000                                                                     22 row(s) returned
```

7. Orders Table

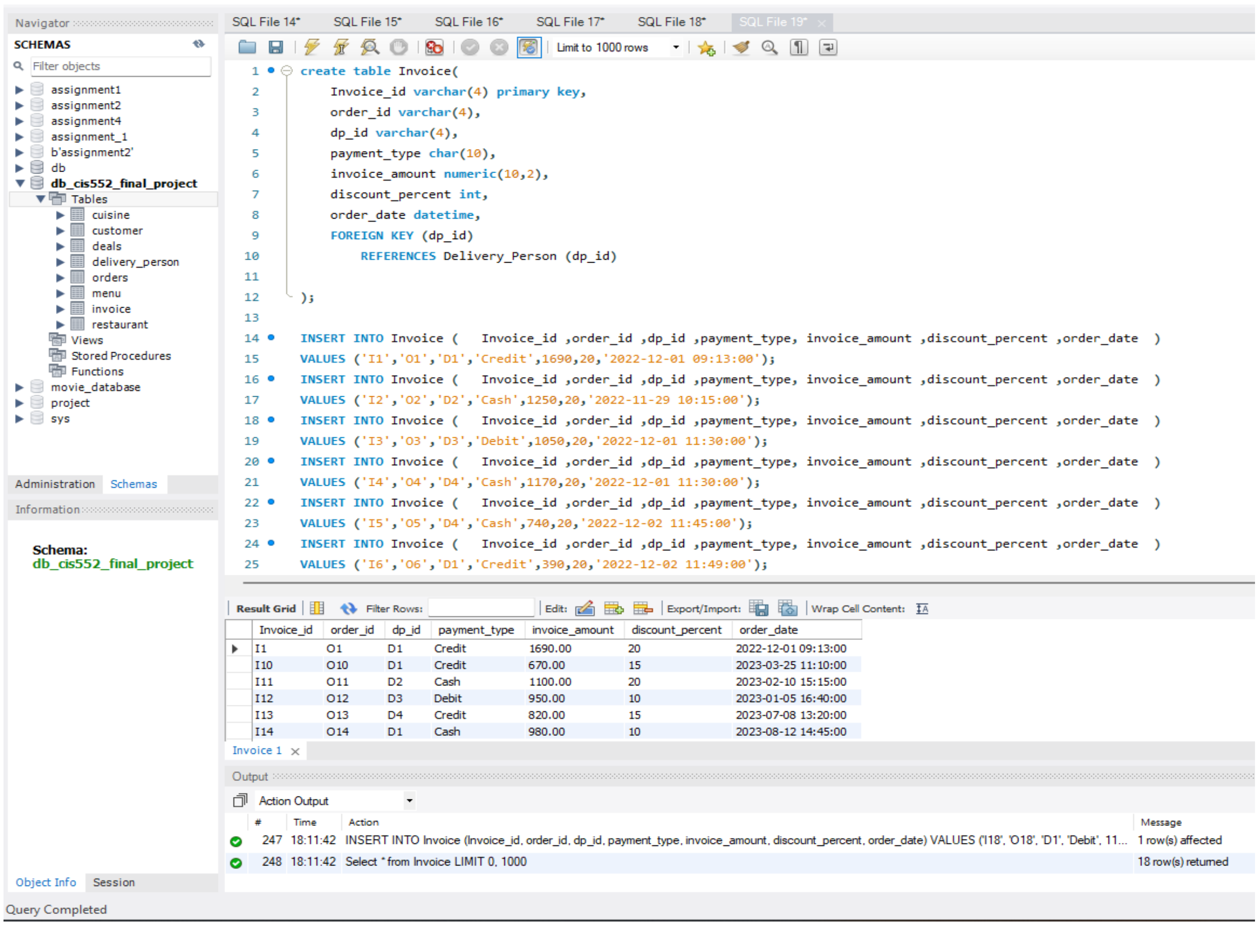


```sql

create table Orders(
    order_id varchar(4),
    menu_item_id varchar(4),
    cus_id varchar(4),
    res_id varchar(4),
    menu_item_qty int,
    primary key(order_id,menu_item_id),
    FOREIGN KEY (menu_item_id,res_id)
        REFERENCES Menu (menu_item_id,res_id),
    FOREIGN KEY (cus_id)
        REFERENCES Customer (cus_id),
    FOREIGN KEY (res_id)
        REFERENCES Restaurant (res_id)
);
INSERT INTO Orders (order_id,menu_item_id,cus_id ,res_id ,menu_item_qty )
VALUES ('O1','M2','C5','R1',1);
INSERT INTO Orders (order_id,menu_item_id,cus_id ,res_id ,menu_item_qty )
VALUES ('O2','M2','C3','R1',1);
INSERT INTO Orders (order_id,menu_item_id,cus_id ,res_id ,menu_item_qty  )
VALUES ('O2','M4','C3','R1',2);
INSERT INTO Orders (order_id,menu_item_id,cus_id ,res_id ,menu_item_qty )
VALUES ('O3','M2','C1','R1',3);
INSERT INTO Orders (order_id,menu_item_id,cus_id ,res_id ,menu_item_qty )
VALUES ('O4','M7','C5','R2',1);
INSERT INTO Orders (order_id,menu_item_id,cus_id ,res_id ,menu_item_qty )
```

order_id	menu_item_id	cus_id	res_id	menu_item_qty
O1	M2	C5	R1	1
O2	M2	C3	R1	1
O2	M4	C3	R1	2
O3	M2	C1	R1	3
O4	M12	C5	R2	1
O4	M7	C5	R2	1

```
#    Time      Action                                                                                                   Message
227  18:10:35  INSERT INTO Orders (order_id,menu_item_id,cus_id ,res_id ,menu_item_qty ) VALUES ('O6','M7','C1','R2',1)  1 row(s) affected
228  18:10:35  Select * from orders LIMIT 0, 1000                                                                       10 row(s) returned
```

8. Invoice Table



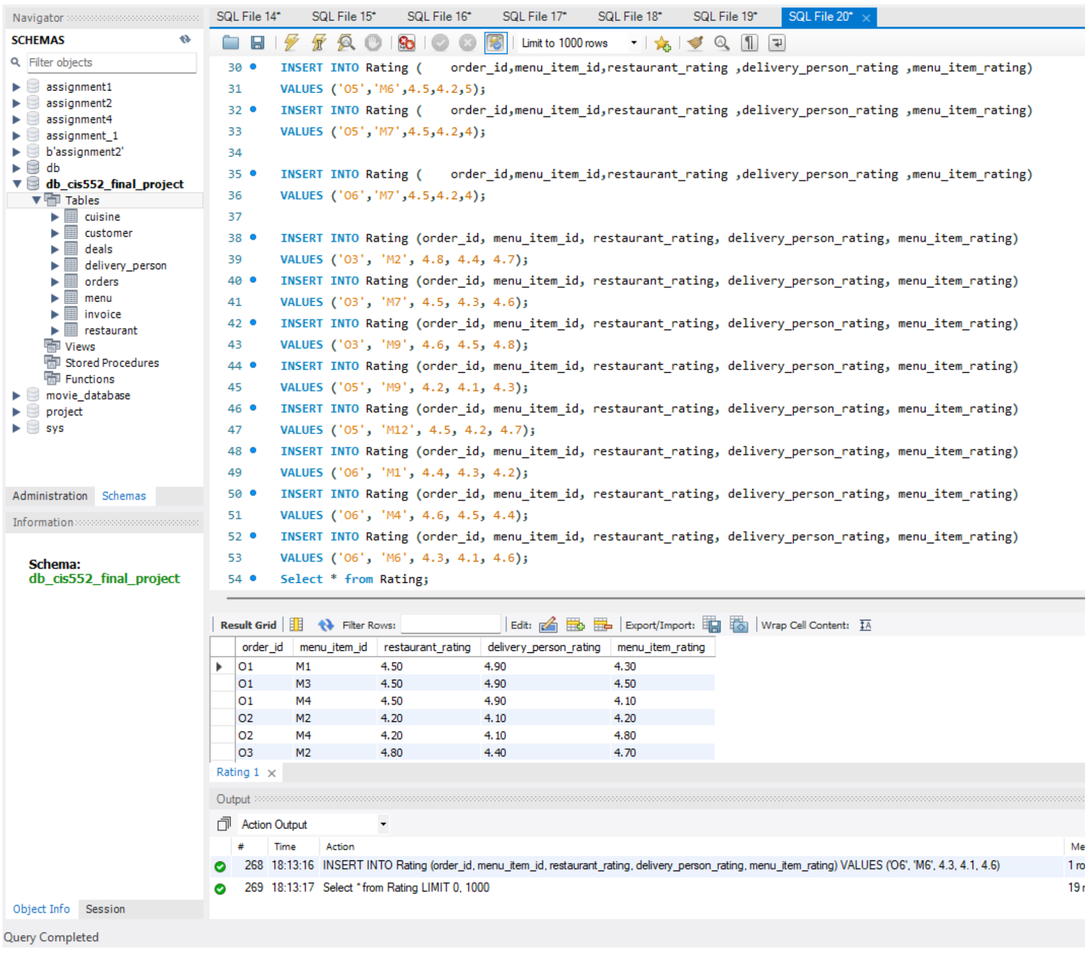
```sql
create table Invoice(
    Invoice_id varchar(4) primary key,
    order_id varchar(4),
    dp_id varchar(4),
    payment_type char(10),
    invoice_amount numeric(10,2),
    discount_percent int,
    order_date datetime,
    FOREIGN KEY (dp_id)
        REFERENCES Delivery_Person (dp_id)

);

INSERT INTO Invoice (   Invoice_id ,order_id ,dp_id ,payment_type, invoice_amount ,discount_percent ,order_date  )
VALUES ('I1','O1','D1','Credit',1690,20,'2022-12-01 09:13:00');
INSERT INTO Invoice (   Invoice_id ,order_id ,dp_id ,payment_type, invoice_amount ,discount_percent ,order_date  )
VALUES ('I2','O2','D2','Cash',1250,20,'2022-11-29 10:15:00');
INSERT INTO Invoice (   Invoice_id ,order_id ,dp_id ,payment_type, invoice_amount ,discount_percent ,order_date  )
VALUES ('I3','O3','D3','Debit',1050,20,'2022-12-01 11:30:00');
INSERT INTO Invoice (   Invoice_id ,order_id ,dp_id ,payment_type, invoice_amount ,discount_percent ,order_date  )
VALUES ('I4','O4','D4','Cash',1170,20,'2022-12-01 11:30:00');
INSERT INTO Invoice (   Invoice_id ,order_id ,dp_id ,payment_type, invoice_amount ,discount_percent ,order_date  )
VALUES ('I5','O5','D4','Cash',740,20,'2022-12-02 11:45:00');
INSERT INTO Invoice (   Invoice_id ,order_id ,dp_id ,payment_type, invoice_amount ,discount_percent ,order_date  )
VALUES ('I6','O6','D1','Credit',390,20,'2022-12-02 11:49:00');
```

Invoice_id	order_id	dp_id	payment_type	invoice_amount	discount_percent	order_date
I1	O1	D1	Credit	1690.00	20	2022-12-01 09:13:00
I10	O10	D1	Credit	670.00	15	2023-03-25 11:10:00
I11	O11	D2	Cash	1100.00	20	2023-02-10 15:15:00
I12	O12	D3	Debit	950.00	10	2023-01-05 16:40:00
I13	O13	D4	Credit	820.00	15	2023-07-08 13:20:00
I14	O14	D1	Cash	980.00	10	2023-08-12 14:45:00

```
#    Time      Action                                                                                                                                         Message
247  18:11:42  INSERT INTO Invoice (Invoice_id, order_id, dp_id, payment_type, invoice_amount, discount_percent, order_date) VALUES ('I18', 'O18', 'D1', 'Debit', 11...  1 row(s) affected
248  18:11:42  Select * from Invoice LIMIT 0, 1000                                                                                                            18 row(s) returned
```

Query Completed

9. Rating Table



```
INSERT INTO Rating (    order_id,menu_item_id,restaurant_rating ,delivery_person_rating ,menu_item_rating)
VALUES ('O5','M6',4.5,4.2,5);
INSERT INTO Rating (    order_id,menu_item_id,restaurant_rating ,delivery_person_rating ,menu_item_rating)
VALUES ('O5','M7',4.5,4.2,4);

INSERT INTO Rating (    order_id,menu_item_id,restaurant_rating ,delivery_person_rating ,menu_item_rating)
VALUES ('O6','M7',4.5,4.2,4);

INSERT INTO Rating (order_id, menu_item_id, restaurant_rating, delivery_person_rating, menu_item_rating)
VALUES ('O3', 'M2', 4.8, 4.4, 4.7);
INSERT INTO Rating (order_id, menu_item_id, restaurant_rating, delivery_person_rating, menu_item_rating)
VALUES ('O3', 'M7', 4.5, 4.3, 4.6);
INSERT INTO Rating (order_id, menu_item_id, restaurant_rating, delivery_person_rating, menu_item_rating)
VALUES ('O3', 'M9', 4.6, 4.5, 4.8);
INSERT INTO Rating (order_id, menu_item_id, restaurant_rating, delivery_person_rating, menu_item_rating)
VALUES ('O5', 'M9', 4.2, 4.1, 4.3);
INSERT INTO Rating (order_id, menu_item_id, restaurant_rating, delivery_person_rating, menu_item_rating)
VALUES ('O5', 'M12', 4.5, 4.2, 4.7);
INSERT INTO Rating (order_id, menu_item_id, restaurant_rating, delivery_person_rating, menu_item_rating)
VALUES ('O6', 'M1', 4.4, 4.3, 4.2);
INSERT INTO Rating (order_id, menu_item_id, restaurant_rating, delivery_person_rating, menu_item_rating)
VALUES ('O6', 'M4', 4.6, 4.5, 4.4);
INSERT INTO Rating (order_id, menu_item_id, restaurant_rating, delivery_person_rating, menu_item_rating)
VALUES ('O6', 'M6', 4.3, 4.1, 4.6);
Select * from Rating;
```

order_id	menu_item_id	restaurant_rating	delivery_person_rating	menu_item_rating
O1	M1	4.50	4.90	4.30
O1	M3	4.50	4.90	4.50
O1	M4	4.50	4.90	4.10
O2	M2	4.20	4.10	4.20
O2	M4	4.20	4.10	4.80
O3	M2	4.80	4.40	4.70

```
#    Time      Action                                                                                                                              Me
268  18:13:16  INSERT INTO Rating (order_id, menu_item_id, restaurant_rating, delivery_person_rating, menu_item_rating) VALUES ('O6', 'M6', 4.3, 4.1, 4.6)  1 ro
269  18:13:17  Select * from Rating LIMIT 0, 1000                                                                                                  19 r
```

Schema: db_cis552_final_project

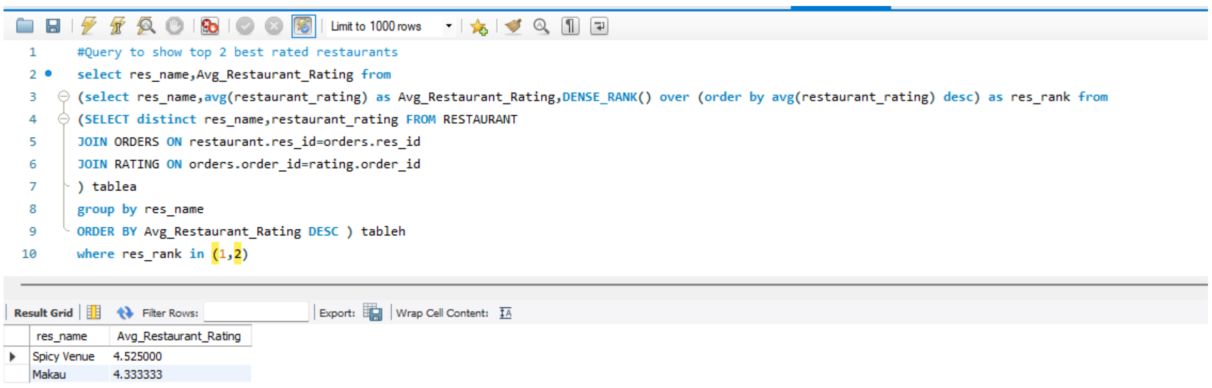
Query Completed

4. SQL Queries

4.1 Query Examples

i. Query to show top 2 best rated restaurants:

The query identifies the top 2 best-rated restaurants. It uses a subquery to calculate the average restaurant rating for each restaurant, assigns a dense rank based on the descending order of average ratings, and then filters the results to include only those with a rank of 1 or 2. Finally, it selects the restaurant name and average rating for display.

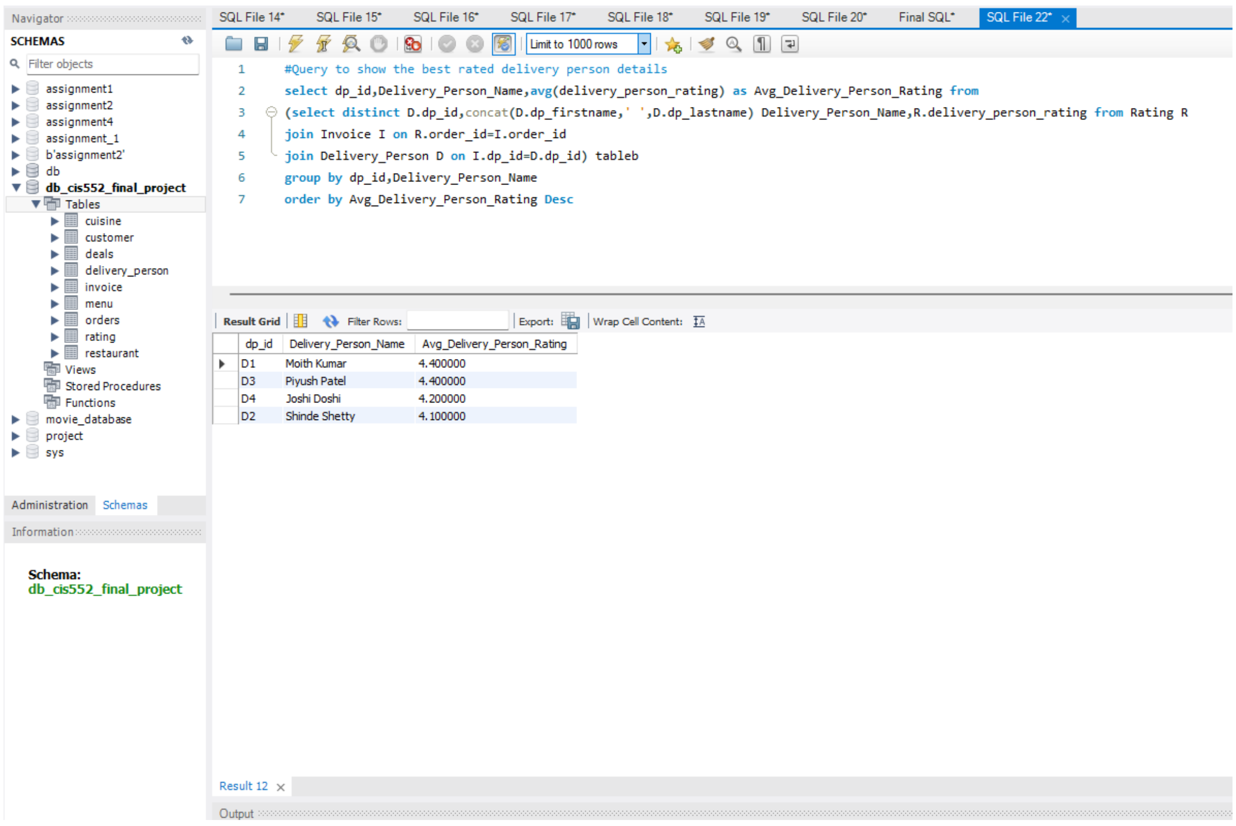


```sql
#Query to show top 2 best rated restaurants
select res_name,Avg_Restaurant_Rating from
(select res_name,avg(restaurant_rating) as Avg_Restaurant_Rating,DENSE_RANK() over (order by avg(restaurant_rating) desc) as res_rank from
(SELECT distinct res_name,restaurant_rating FROM RESTAURANT
JOIN ORDERS ON restaurant.res_id=orders.res_id
JOIN RATING ON orders.order_id=rating.order_id
) tablea
group by res_name
ORDER BY Avg_Restaurant_Rating DESC ) tableh
where res_rank in (1,2)
```

res_name	Avg_Restaurant_Rating
Spicy Venue	4.525000
Makau	4.333333

ii. Query to show the best rated delivery person details:

This SQL query retrieves the delivery person ID (dp_id), concatenated full name (Delivery_Person_Name), and the average rating (Avg_Delivery_Person_Rating) for each delivery person. The data is obtained by joining the Rating, Invoice, and Delivery_Person tables. The inner subquery (tableb) ensures distinct combinations of delivery person details. The outer query then calculates the average rating for each delivery person, grouping the results by dp_id and Delivery_Person_Name, and finally, it orders the output by the average rating in descending order, showing the best-rated delivery persons first.

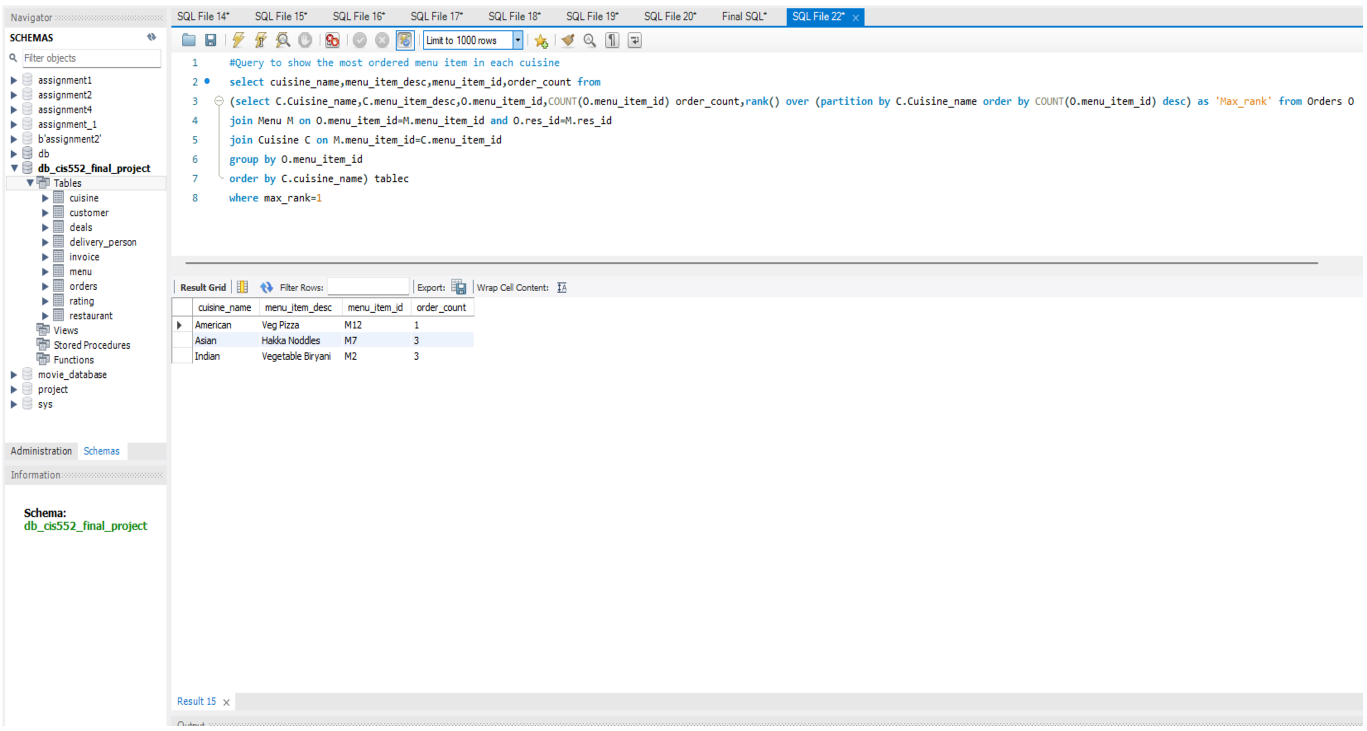


```sql
#Query to show the best rated delivery person details
select dp_id,Delivery_Person_Name,avg(delivery_person_rating) as Avg_Delivery_Person_Rating from
(select distinct D.dp_id,concat(D.dp_firstname,' ',D.dp_lastname) Delivery_Person_Name,R.delivery_person_rating from Rating R
join Invoice I on R.order_id=I.order_id
join Delivery_Person D on I.dp_id=D.dp_id) tableb
group by dp_id,Delivery_Person_Name
order by Avg_Delivery_Person_Rating Desc
```

dp_id	Delivery_Person_Name	Avg_Delivery_Person_Rating
D1	Moith Kumar	4.400000
D3	Piyush Patel	4.400000
D4	Joshi Doshi	4.200000
D2	Shinde Shetty	4.100000

iii. Query to show the most ordered menu item in each cuisine:

This query retrieves the most ordered menu item in each cuisine by joining the Orders, Menu, and Cuisine tables. It utilizes the RANK() window function to rank menu items based on order count within their respective cuisines. The results are filtered to include only the top-ranked items in each cuisine, providing information such as cuisine name, menu item description, ID, and the corresponding order count for the most popular menu item in each cuisine.

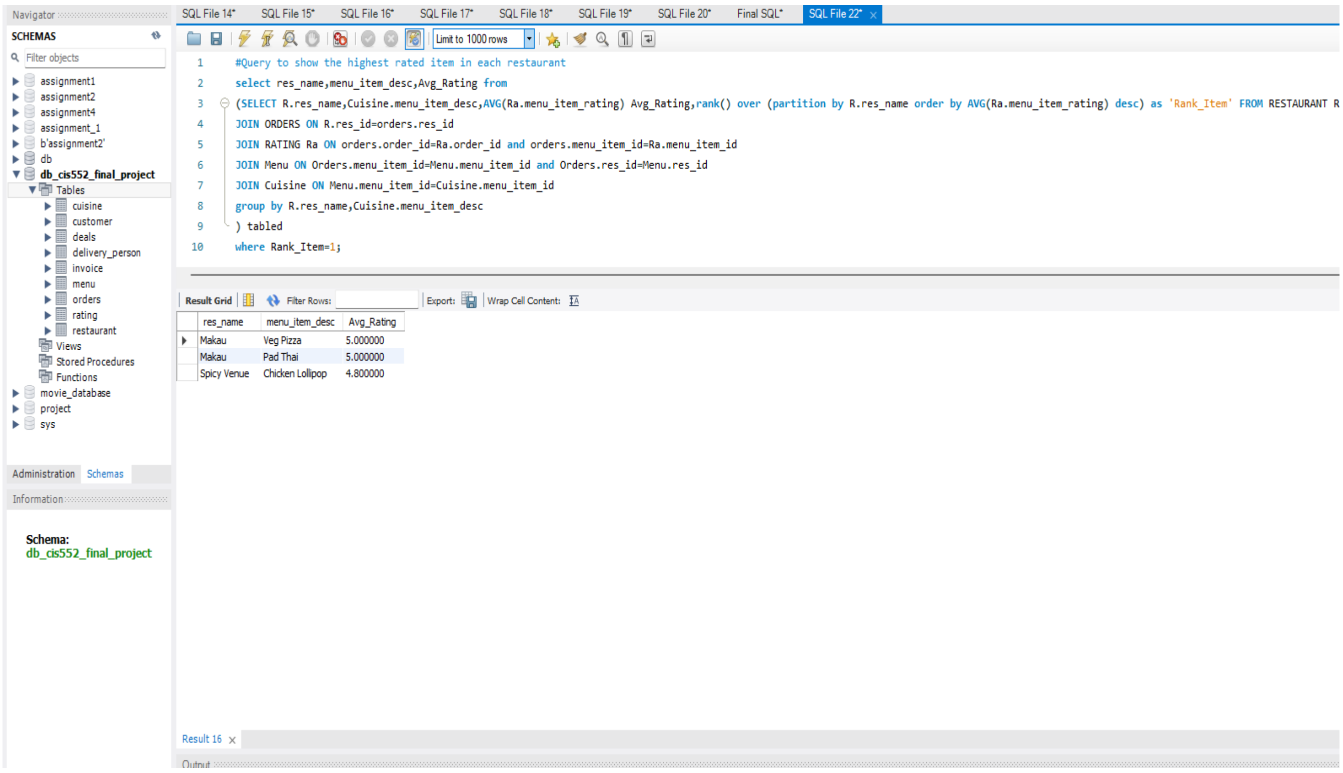


```
#Query to show the most ordered menu item in each cuisine
select cuisine_name,menu_item_desc,menu_item_id,order_count from
(select C.Cuisine_name,C.menu_item_desc,O.menu_item_id,COUNT(O.menu_item_id) order_count,rank() over (partition by C.Cuisine_name order by COUNT(O.menu_item_id) desc) as 'Max_rank' from Orders O
join Menu M on O.menu_item_id=M.menu_item_id and O.res_id=M.res_id
join Cuisine C on M.menu_item_id=C.menu_item_id
group by O.menu_item_id
order by C.cuisine_name) tablec
where max_rank=1
```

cuisine_name	menu_item_desc	menu_item_id	order_count
American	Veg Pizza	M12	1
Asian	Hakka Noddles	M7	3
Indian	Vegetable Biryani	M2	3

iv. Query to show the highest rated item in each restaurant:

This query retrieves the highest-rated menu item in each restaurant by joining the RESTAURANT, ORDERS, RATING, MENU, and CUISINE tables. It uses the AVG() function to calculate the average rating for each menu item within each restaurant. The RANK() window function is then applied to rank the items based on their average ratings within each restaurant. The final results are filtered to include only the top-ranked items in each restaurant, providing information such as restaurant name, menu item description, and the corresponding average rating for the highest-rated item in each restaurant.

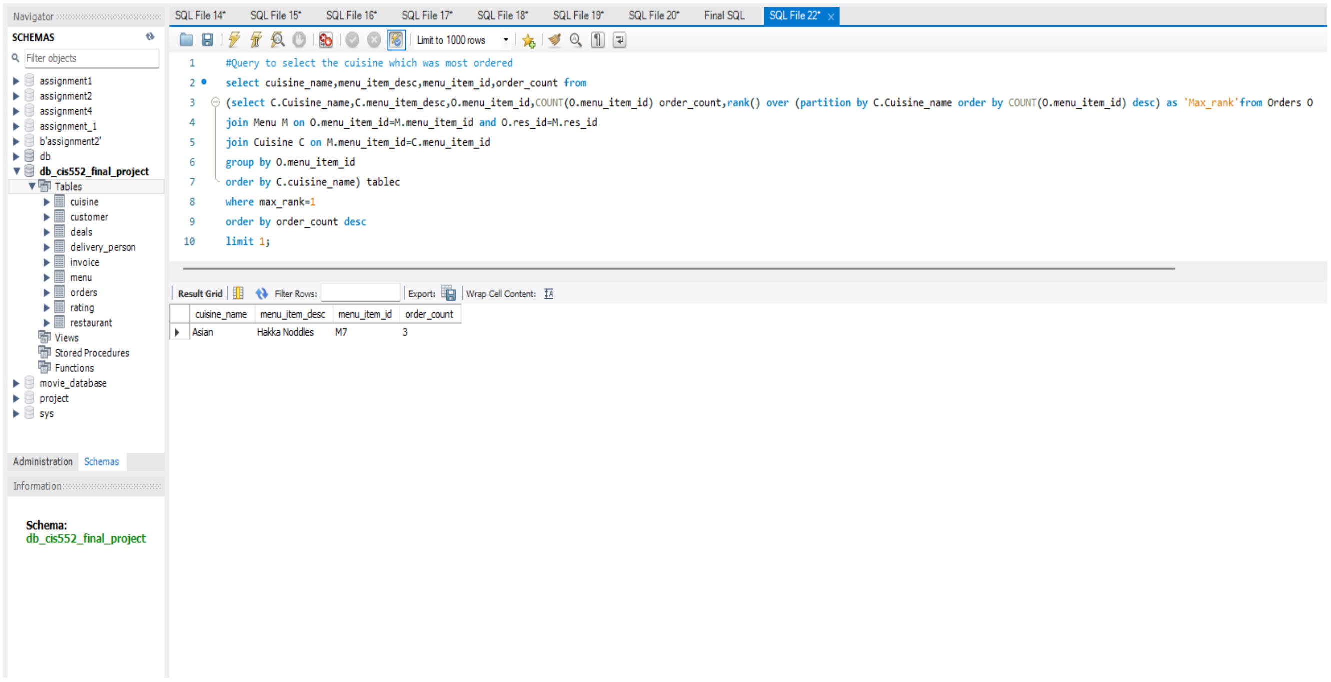


```sql
#Query to show the highest rated item in each restaurant
select res_name,menu_item_desc,Avg_Rating from
(SELECT R.res_name,Cuisine.menu_item_desc,AVG(Ra.menu_item_rating) Avg_Rating,rank() over (partition by R.res_name order by AVG(Ra.menu_item_rating) desc) as 'Rank_Item' FROM RESTAURANT R
JOIN ORDERS ON R.res_id=orders.res_id
JOIN RATING Ra ON orders.order_id=Ra.order_id and orders.menu_item_id=Ra.menu_item_id
JOIN Menu ON Orders.menu_item_id=Menu.menu_item_id and Orders.res_id=Menu.res_id
JOIN Cuisine ON Menu.menu_item_id=Cuisine.menu_item_id
group by R.res_name,Cuisine.menu_item_desc
) tabled
where Rank_Item=1;
```

res_name	menu_item_desc	Avg_Rating
Makau	Veg Pizza	5.000000
Makau	Pad Thai	5.000000
Spicy Venue	Chicken Lollipop	4.800000

v. Query to select the cuisine which was most ordered:

This query retrieves the highest-rated menu item in each restaurant by joining the RESTAURANT, ORDERS, RATING, MENU, and CUISINE tables. It uses the AVG() function to calculate the average rating for each menu item within each restaurant. The RANK() window function is then applied to rank the items based on their average ratings within each restaurant. The final results are filtered to include only the top-ranked items in each restaurant, providing information such as restaurant name, menu item description, and the corresponding average rating for the highest-rated item in each restaurant.



```sql
#Query to select the cuisine which was most ordered
select cuisine_name,menu_item_desc,menu_item_id,order_count from
(select C.Cuisine_name,C.menu_item_desc,O.menu_item_id,COUNT(O.menu_item_id) order_count,rank() over (partition by C.Cuisine_name order by COUNT(O.menu_item_id) desc) as 'Max_rank'from Orders O
join Menu M on O.menu_item_id=M.menu_item_id and O.res_id=M.res_id
join Cuisine C on M.menu_item_id=C.menu_item_id
group by O.menu_item_id
order by C.cuisine_name) tablec
where max_rank=1
order by order_count desc
limit 1;
```

cuisine_name	menu_item_desc	menu_item_id	order_count
Asian	Hakka Noddles	M7	3

4.2 Query Optimization Techniques

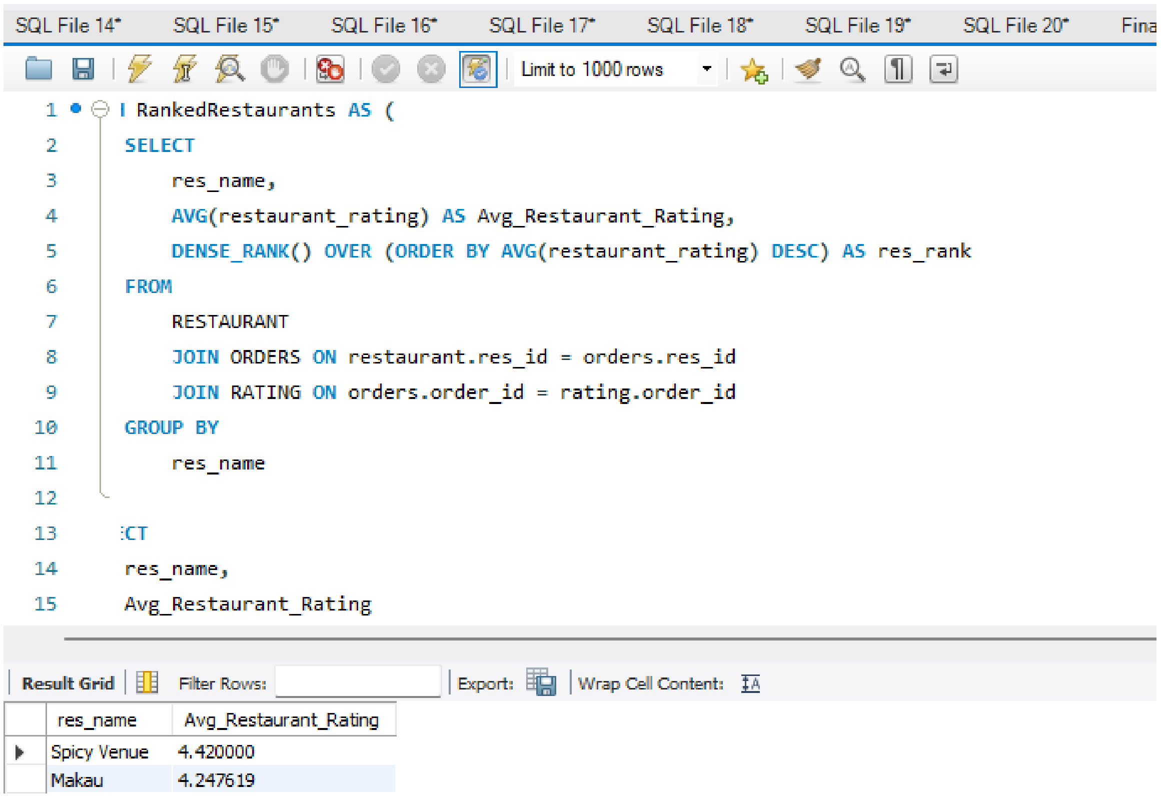
Query optimization is a crucial aspect of database management that aims to enhance the performance of database queries by minimizing execution time and resource usage. Here are some general query optimization techniques:

i. Use Indexes: Indexes help speed up data retrieval by providing a quick lookup mechanism. Ensure that columns involved in WHERE clauses, JOIN conditions, and ORDER BY clauses are properly indexed.
ii. *Avoid SELECT: Retrieve only the columns needed for the query rather than using SELECT * to fetch all columns. This reduces the amount of data transferred and processed.
iii. Optimize JOIN Operations: Use INNER JOINs when possible, as they are generally more efficient than OUTER JOINs. Consider using appropriate JOIN types based on the relationship between tables.
iv. Use WHERE Clause Effectively: Apply filters in the WHERE clause to limit the number of rows processed. This reduces the amount of data that needs to be read and improves query performance.
v. Aggregate Functions and GROUP BY: Be mindful of using aggregate functions and GROUP BY clauses. Ensure that they are applied only when necessary to avoid unnecessary calculations.
vi. Avoid Subqueries if Possible: Subqueries can be resource intensive. Consider using JOINs or other techniques to achieve the same result without the need for subqueries.
vii. Use EXISTS and IN Sparingly: EXISTS and IN can be inefficient for large datasets. Consider alternative approaches, such as JOINs or EXISTS with correlated subqueries, for better performance.
viii. Optimize ORDER BY and LIMIT: If possible, avoid sorting large result sets. Use ORDER BY and LIMIT judiciously and ensure that the columns used for sorting are indexed.
ix. Partitioning: Partition large tables to improve query performance by restricting the amount of data that needs to be scanned.
x. Use Proper Data Types: Choose appropriate data types for columns to minimize storage space and optimize query performance.
xi. Database Design: A well-designed database schema with normalized tables and appropriate relationships can contribute to efficient query execution.

Here We tried to Optimize the couple of queries from above Queries using Optimization techniques:

1. Query to show top 2 best rated restaurants:

- Table Aliases: Use meaningful table aliases to improve code readability.
- Avoid Redundant Subquery: You can eliminate the redundant subquery by using the window function directly in the outer query.

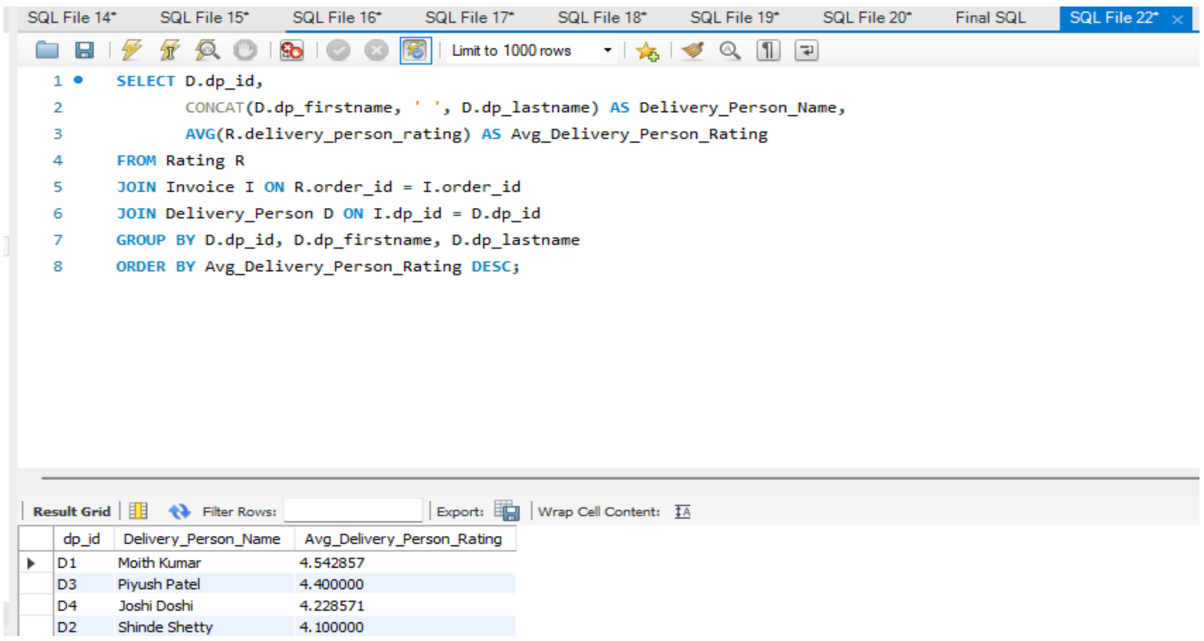


```sql
I RankedRestaurants AS (
SELECT
    res_name,
    AVG(restaurant_rating) AS Avg_Restaurant_Rating,
    DENSE_RANK() OVER (ORDER BY AVG(restaurant_rating) DESC) AS res_rank
FROM
    RESTAURANT
    JOIN ORDERS ON restaurant.res_id = orders.res_id
    JOIN RATING ON orders.order_id = rating.order_id
GROUP BY
    res_name

:CT
res_name,
Avg_Restaurant_Rating
```

res_name	Avg_Restaurant_Rating
Spicy Venue	4.420000
Makau	4.247619

2. Query to show the best rated delivery person details:

- Removed Subquery: The subquery was removed, and the join conditions were integrated directly into the main query for better clarity.
- Avoided DISTINCT: The DISTINCT keyword was eliminated, assuming that the join conditions are appropriately set to avoid duplicate records.
- Simplified Concatenation: The CONCAT function was used directly in the SELECT clause for simplicity.



```sql
SELECT D.dp_id,
       CONCAT(D.dp_firstname, ' ', D.dp_lastname) AS Delivery_Person_Name,
       AVG(R.delivery_person_rating) AS Avg_Delivery_Person_Rating
FROM Rating R
JOIN Invoice I ON R.order_id = I.order_id
JOIN Delivery_Person D ON I.dp_id = D.dp_id
GROUP BY D.dp_id, D.dp_firstname, D.dp_lastname
ORDER BY Avg_Delivery_Person_Rating DESC;
```

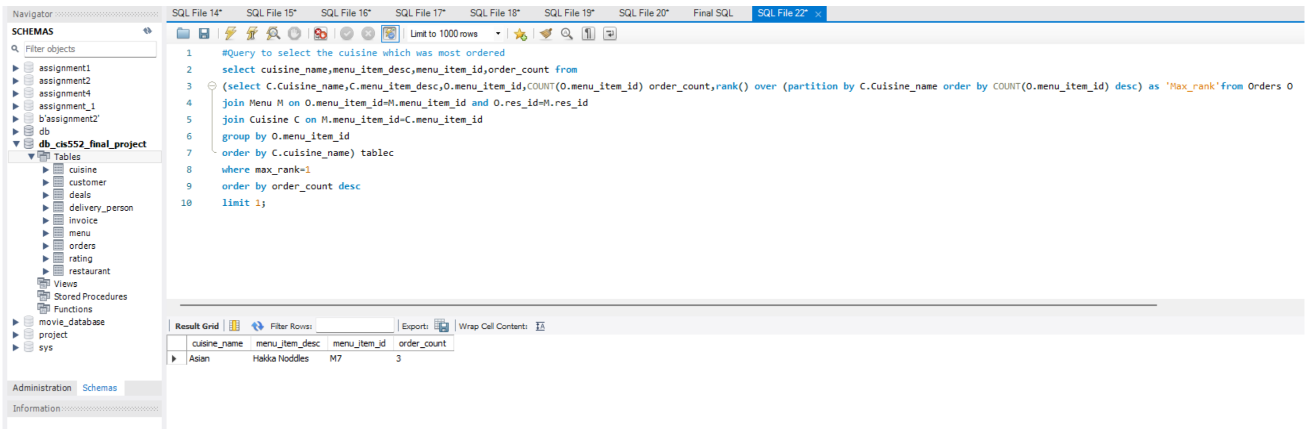
dp_id	Delivery_Person_Name	Avg_Delivery_Person_Rating
D1	Moith Kumar	4.542857
D3	Piyush Patel	4.400000
D4	Joshi Doshi	4.228571
D2	Shinde Shetty	4.100000

4.3 TRC queries

Tuple Relational Calculus (TRC) is a formal language used for relational databases. It is a non-procedural query language, meaning that it specifies what data to retrieve rather than how to retrieve it. TRC operates on sets of tuples, which represent rows in a relational database table.

i. TRC to select the cuisine which was most ordered

{(cuisine_name,menu_item_desc,menu_item_id,order_count)| ∃O,M,C(Orders(O)∧Menu(M)∧Cuisine(C)∧O.menu_item_id=M.menu_item_id∧O.res_id=M.res_id∧ M.menu_item_id=C.menu_item_id∧C.cuisine_name=cuisine_name∧M.menu_item_desc=menu_item_desc∧O.menu_item_id=menu_item_id ∧COUNT(∧COUNT(O.menu_item_id)=order_count RANK(COUNT(O.menu_item_id))=1} ORDER BY order_count DESC LIMIT 1

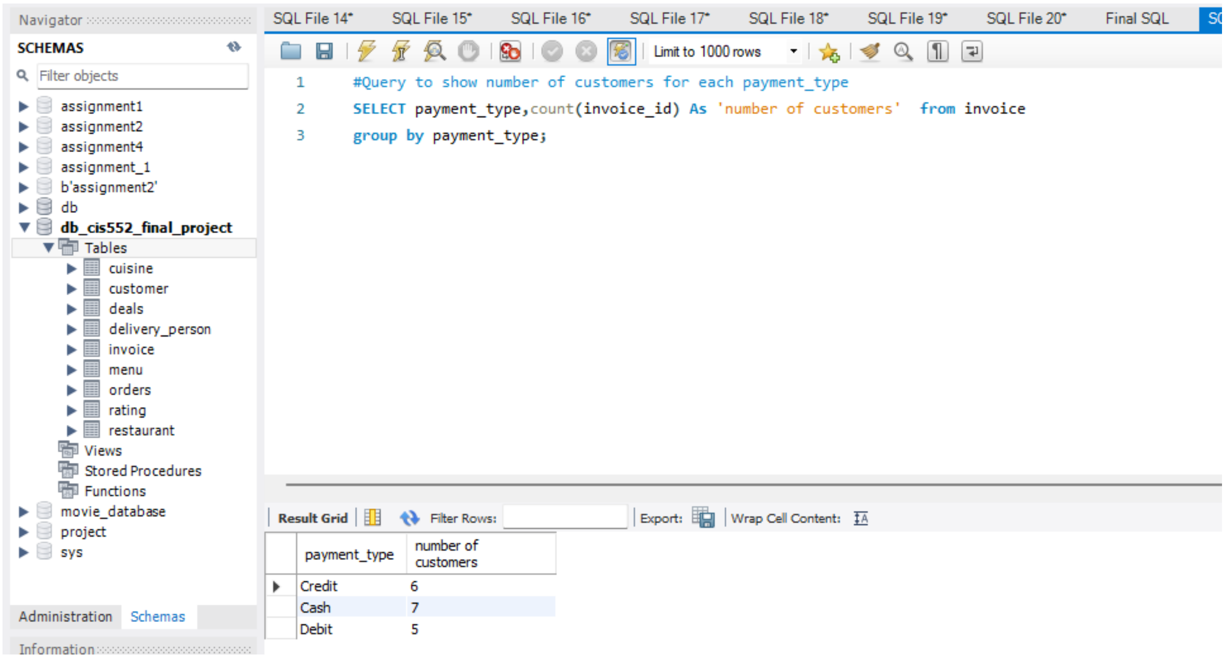


```
#Query to select the cuisine which was most ordered
select cuisine_name,menu_item_desc,menu_item_id,order_count from
(select C.Cuisine_name,C.menu_item_desc,O.menu_item_id,COUNT(O.menu_item_id) order_count,rank() over (partition by C.Cuisine_name order by COUNT(O.menu_item_id) desc) as 'Max_rank'from Orders O
join Menu M on O.menu_item_id=M.menu_item_id and O.res_id=M.res_id
join Cuisine C on M.menu_item_id=C.menu_item_id
group by O.menu_item_id
order by C.cuisine_name) tablec
where max_rank=1
order by order_count desc
limit 1;
```

cuisine_name	menu_item_desc	menu_item_id	order_count
Asian	Hakka Noddles	M7	3

ii. TRC to show number of customers for each payment type

{(payment_type, COUNT(invoice_id))|∃invoice(invoice_id,payment_type)}



```
#Query to show number of customers for each payment_type
SELECT payment_type,count(invoice_id) As 'number of customers'  from invoice
group by payment_type;
```

payment_type	number of customers
Credit	6
Cash	7
Debit	5

5. Conclusion

In conclusion, the restaurant management system project has successfully used delivered a robust and well-structured relational database to address the complex operational requirements of a modern restaurant. By designing tables for entities such as restaurants, cuisines, menus, orders, invoices, and ratings, we've created a comprehensive system that ensures data integrity through foreign key relationships. Realistic sample data was generated to facilitate meaningful analysis and testing of the system's functionality. The implementation supports key restaurant management operations, including menu management, order placement, and invoice generation. Furthermore, the system's SQL queries enable valuable insights into restaurant performance, customer preferences, and efficient delivery operations. While the project has achieved its primary objectives, future enhancements could focus on scalability, user interface integration, and heightened security measures to meet the evolving needs of the restaurant industry. Overall, this project provides a strong foundation for a versatile and efficient restaurant management system.

6. Future Recommendations

i. Scalability: While the system is designed to handle typical restaurant operations, scalability may be a consideration for large-scale enterprises. Future enhancements could involve optimizations for performance in such scenarios.

ii. User Interface: The current implementation focuses on the database and backend functionality. Integration with a user interface for easy interaction with the system would be a logical next step.

iii. Security Measures: Implementing robust security measures, such as encryption and access controls, would be essential for protecting sensitive customer and business data.

iv. User Interface (UI) Enhancement: Develop an intuitive and user-friendly interface for both restaurant staff and customers. This can include features such as an interactive menu, easy order placement, and a visually appealing dashboard for restaurant administrators.

v. Mobile Application Development: Consider creating a mobile application to extend the reach and convenience of the restaurant management system. Mobile apps can offer on-the-go access for customers, delivery personnel, and restaurant staff.

vi. Integration with Online Platforms: Integrate the system with popular online food delivery platforms to broaden the restaurant's customer base. Seamless integration with platforms like UberEats, DoorDash, or Grubhub can streamline order processing and increase visibility.

vii. Advanced Analytics and Reporting: Enhance the analytical capabilities of the system by incorporating advanced reporting tools and data visualization. This can help restaurant owners and managers make data-driven decisions based on sales trends, customer feedback, and operational performance.

Github:

The complete project details can be found in my github account by following this link,

Restaurant Data Analysis Management for Informed Decision-Making