Food Ordering System

TEAM MEMBERS

Prabhjyot Kaur

Anushka Sawant

Ishika Kedia

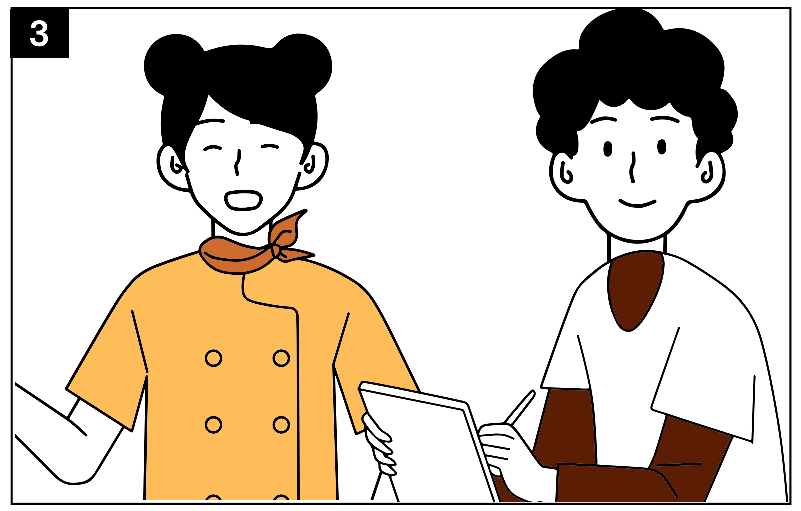
Simran Patel

Dabawalas

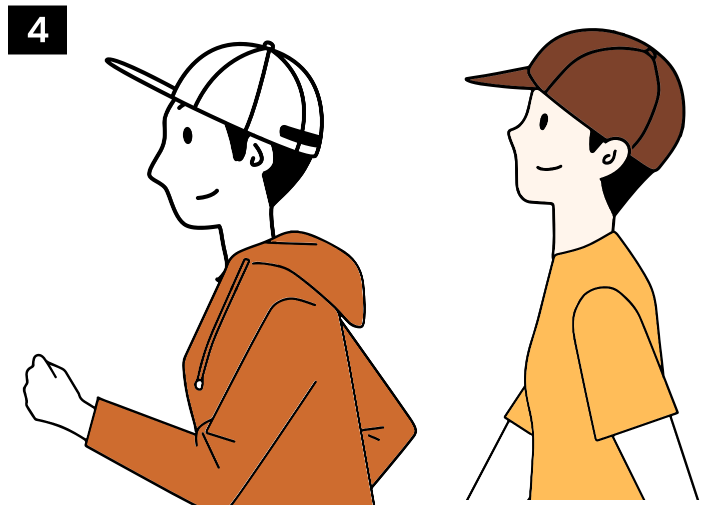
India developed the dabbawala meal delivery system in busy metropolitan areas, such as Mumbai, in response to the increased number of workers in cities, this meal delivery system relied on delivery men called dabbawalas.
The dabbawalas constitute a lunchbox delivery and return system that delivers hot lunches from homes and restaurants to people at work

People calling their nearest restaurant to order food from their selective menu.

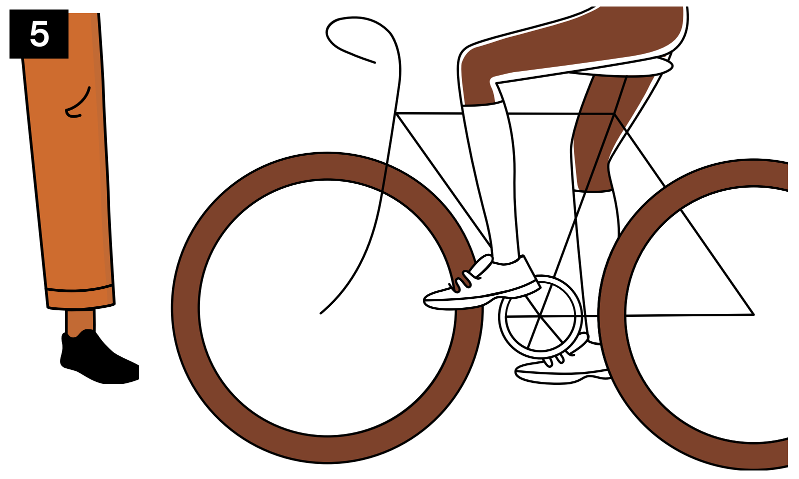
Restaurants accepting some orders or declining a few due to shortage of delivery boys.



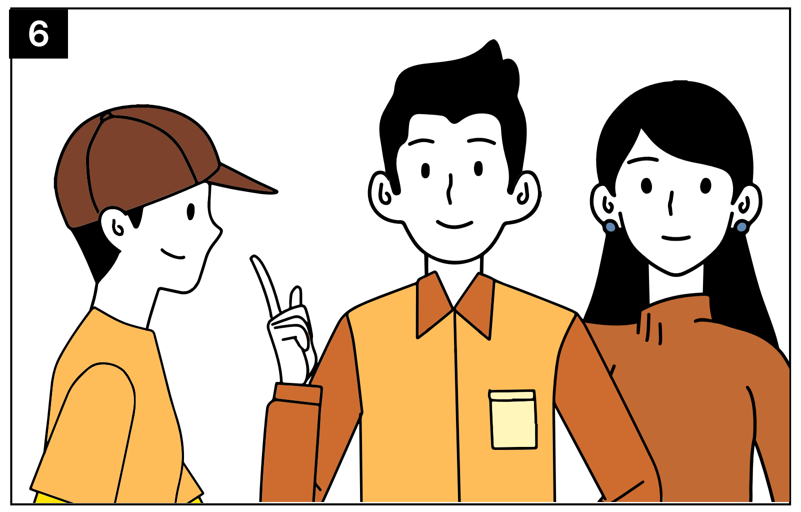
Chef preparing orders.



Some people rushing to the restaurant in order to pick



Getting late deliveries.



Families receiving their deliveries.

Changes after Online Food Ordering System was introduced

Online food ordering allows customers to place an order at virtually any time, from anywhere, saving them time and resources typically spent on travelling to pick up a meal.

During COVID-19, customers may continue to enjoy the minimal contact that online ordering can offer

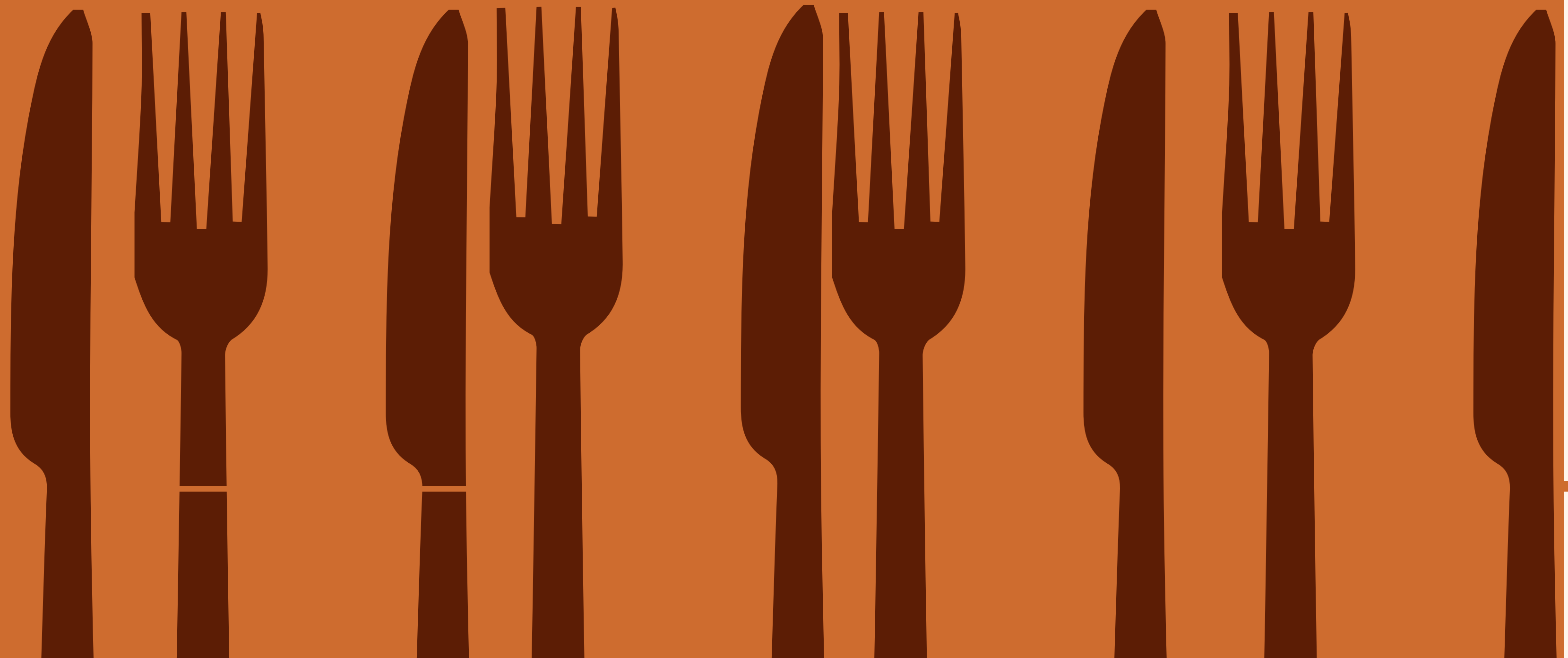
For customers, online ordering also opens their doors to nearly limitless dining options and allows them to browse restaurants and cuisines they may not have been familiar with previously.

Zomato

- The aim of developing the Online Food Ordering System project is to replace the traditional way of taking orders with a computerized system.
- Another important reason for developing this project is to prepare order summary reports quickly and in the correct format at any point of timewhen required.

Swiggy

COMPONENTS OF FOOD ORDERING SYSTEM



Website/App

First is a website or mobile app for customers to view the restaurant's dishes and place an online order.

Admin Management Interface

The second is an admin management interface for the restaurants to receive and manage the customer's orders.

Software

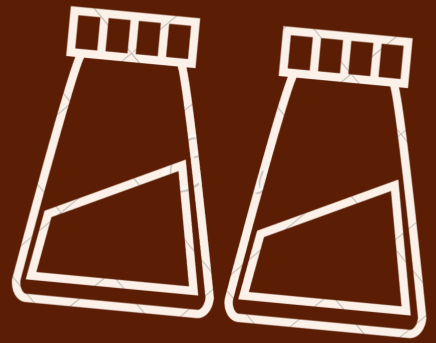
The third is software that manages the orders efficiently, meaning it has the capability to manage different orders at once.

Database

Fourth is a database for storing, accessing, updating and managing the data effectively.

DATABASE

12 Tables



5 Views

3 Triggers

35 Queries

4 Functions

6 Stored Procedures

2 Indexes

TABLES

User	Table with all user information.
Staff	Table with all staff information.
Restaurant	Table with all restaurant and cuisine information.
Food menu	Table with all food menu options.
Drinks menu	Table with all drinks menu options.
Offers	Table with all offer related information.

TABLES

Cart	Table with all user and their cart information.
Orders	Table with all the order information to date.
Payment	Table with all payment information.
Payment Details	Table with all user and their sensitive payment detail information.
Bill	Table with all information about bill with respect to order.
Feedback	Table with all information of feedback with respect to orders they recieved.

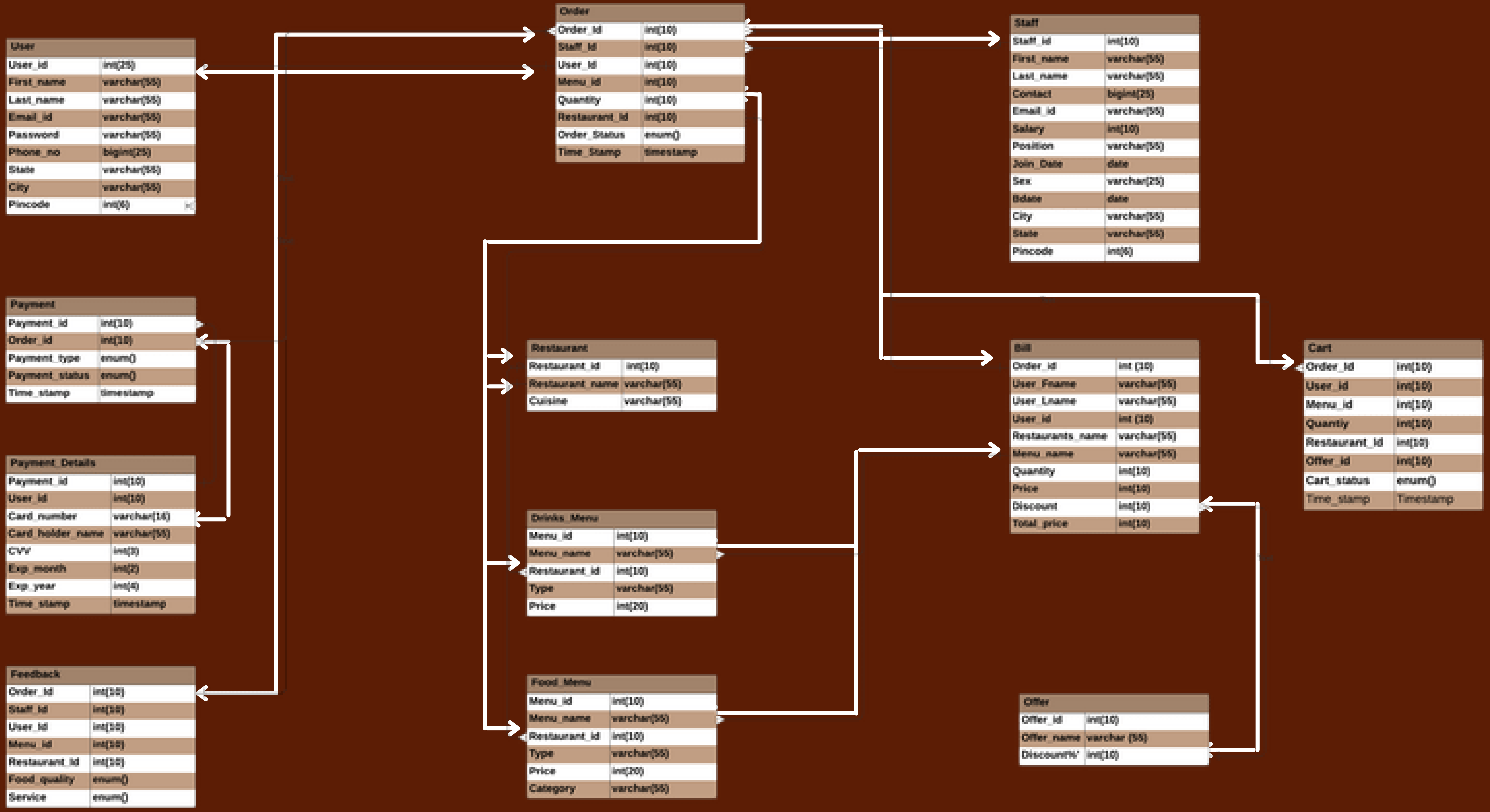
ER DIAGRAM

The purpose of ER Diagram is to represent the entity framework infrastructure.

ER Diagram stands for Entity Relationship Diagram

ER diagrams are created based on three basic concepts: entities, attributes and relationships.

ERD is a diagram that displays the relationship of entity sets stored in a database.



User	
User_id	int(25)
First_name	varchar(55)
Last_name	varchar(55)
Email_id	varchar(55)
Password	varchar(55)
Phone_no	bigint(25)
State	varchar(55)
City	varchar(55)
Pincode	int(6)

Order	
Order_id	int(10)
Staff_id	int(10)
User_id	int(10)
Menu_id	int(10)
Quantity	int(10)
Restaurant_id	int(10)
Order_Status	enum()
Time_Stamp	timestamp

Staff	
Staff_id	int(10)
First_name	varchar(55)
Last_name	varchar(55)
Contact	bigint(25)
Email_id	varchar(55)
Salary	int(10)
Position	varchar(55)
Join_Date	date
Sex	varchar(25)
Bdate	date
City	varchar(55)
State	varchar(55)
Pincode	int(6)

Payment	
Payment_id	int(10)
Order_id	int(10)
Payment_type	enum()
Payment_status	enum()
Time_stamp	timestamp

Restaurant	
Restaurant_id	int(10)
Restaurant_name	varchar(55)
Cuisine	varchar(55)

Bill	
Order_id	int (10)
User_Fname	varchar(55)
User_Lname	varchar(55)
User_id	int (10)
Restaurants_name	varchar(55)
Menu_name	varchar(55)
Quantity	int(10)
Price	int(10)
Discount	int(10)
Total_price	int(10)

Cart	
Order_id	int(10)
User_id	int(10)
Menu_id	int(10)
Quantiy	int(10)
Restaurant_id	int(10)
Offer_id	int(10)
Cart_status	enum()
Time_stamp	Timestamp

Payment_Details	
Payment_id	int(10)
User_id	int(10)
Card_number	varchar(16)
Card_holder_name	varchar(55)
CVV	int(3)
Exp_month	int(2)
Exp_year	int(4)
Time_stamp	timestamp

Drinks_Menu	
Menu_id	int(10)
Menu_name	varchar(55)
Restaurant_id	int(10)
Type	varchar(55)
Price	int(20)

Feedback	
Order_id	int(10)
Staff_id	int(10)
User_id	int(10)
Menu_id	int(10)
Restaurant_id	int(10)
Food_quality	enum()
Service	enum()

Food_Menu	
Menu_id	int(10)
Menu_name	varchar(55)
Restaurant_id	int(10)
Type	varchar(55)
Price	int(20)
Category	varchar(55)

Offer	
Offer_id	int(10)
Offer_name	varchar (55)
Discount%'	int(10)

KEYS

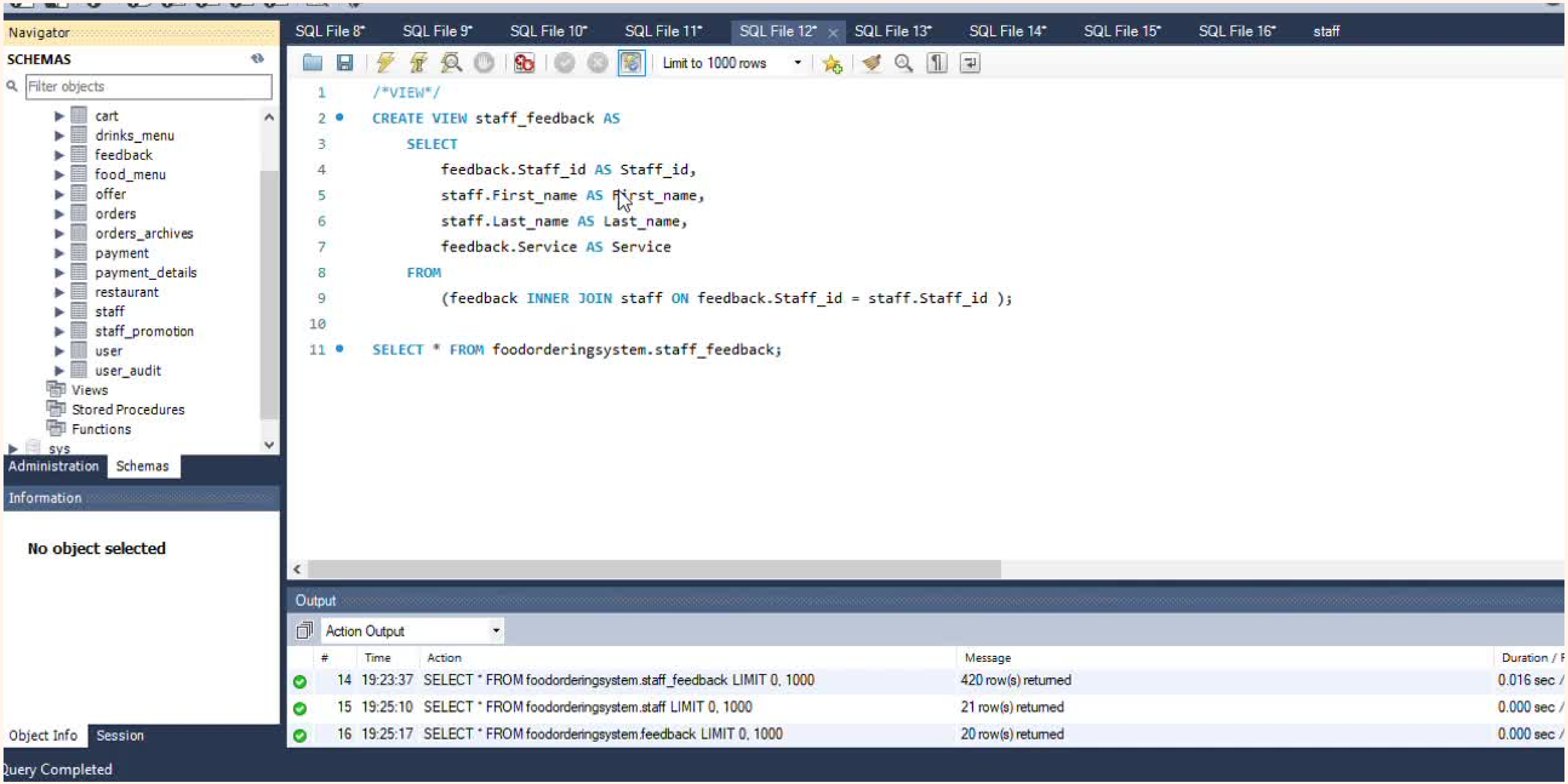
Primary

1. User_id in the User table
2. Staff_id in Staff table
3. Payment_id in the Payment table
4. Menu_id in Food_menu and Drinks_menu table
5. Restaurant_id in Restaurant table
6. Order_id in the Orders table
7. Offer_id in Offers table

Foreign

1. Restaurant_id in Food_menu and Drinks_menu table
2. Staff_id, User_id, Menu_id and Restaurant_id in the Orders table
3. Payment_id and User_id in Payment_details table
4. Order_id and User_id in Bill table
5. Order_id, User_id, Menu_id and Restaurant_id in Cart table
6. Order_id, Staff_id, Menu_id, Restaurant_id in Feedback table

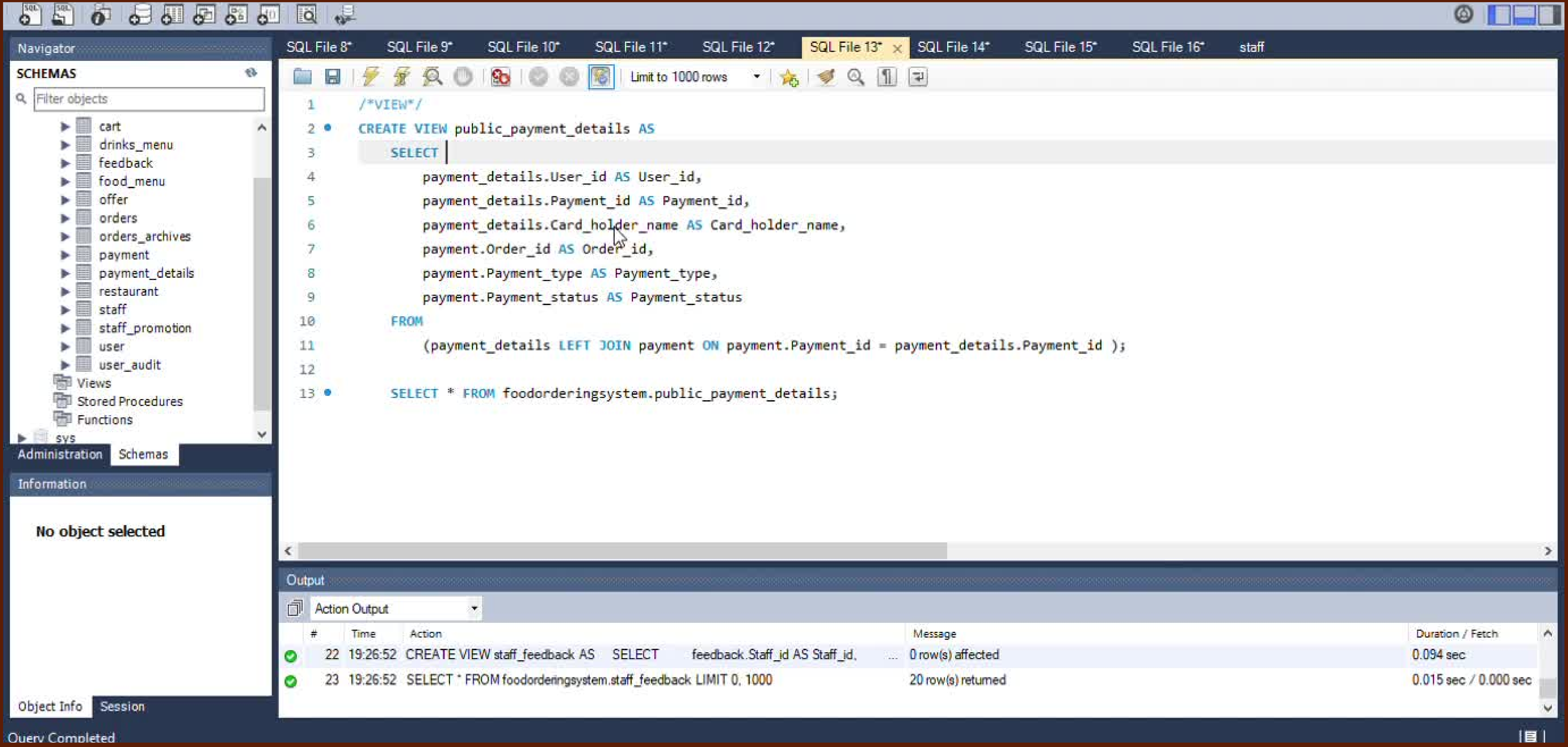
VIEWS



```sql
/*VIEW*/
CREATE VIEW staff_feedback AS
    SELECT
        feedback.Staff_id AS Staff_id,
        staff.First_name AS First_name,
        staff.Last_name AS Last_name,
        feedback.Service AS Service
    FROM
        (feedback INNER JOIN staff ON feedback.Staff_id = staff.Staff_id );

SELECT * FROM foodorderingsystem.staff_feedback;
```

```
#   Time      Action                                                        Message               Duration / F
14  19:23:37  SELECT * FROM foodorderingsystem.staff_feedback LIMIT 0, 1000  420 row(s) returned   0.016 sec /
15  19:25:10  SELECT * FROM foodorderingsystem.staff LIMIT 0, 1000           21 row(s) returned    0.000 sec /
16  19:25:17  SELECT * FROM foodorderingsystem.feedback LIMIT 0, 1000        20 row(s) returned    0.000 sec /
```

Staff_feedback view shows the service feedbacks received from the users.



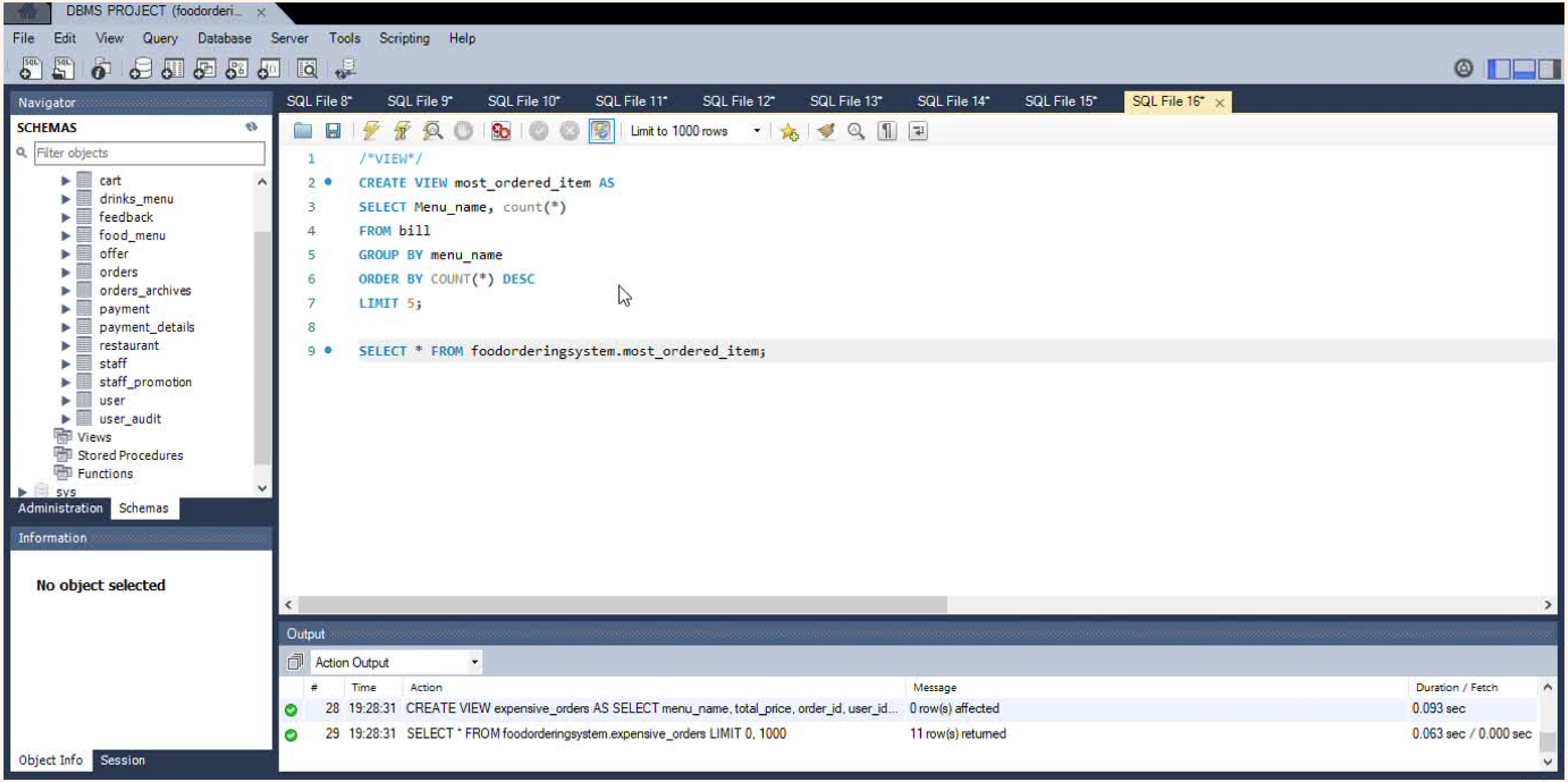
```
/*VIEW*/
CREATE VIEW public_payment_details AS
    SELECT
        payment_details.User_id AS User_id,
        payment_details.Payment_id AS Payment_id,
        payment_details.Card_holder_name AS Card_holder_name,
        payment.Order_id AS Order_id,
        payment.Payment_type AS Payment_type,
        payment.Payment_status AS Payment_status
    FROM
        (payment_details LEFT JOIN payment ON payment.Payment_id = payment_details.Payment_id );

    SELECT * FROM foodorderingsystem.public_payment_details;
```

No object selected

```
#   Time      Action                                                                   Message             Duration / Fetch
22  19:26:52  CREATE VIEW staff_feedback AS   SELECT      feedback.Staff_id AS Staff_id,   ...  0 row(s) affected   0.094 sec
23  19:26:52  SELECT * FROM foodorderingsystem.staff_feedback LIMIT 0, 1000           20 row(s) returned  0.015 sec / 0.000 sec
```

Query Completed

Public_payment_details view doesn't show sensitive information like Cardholder name, CVV, etc.

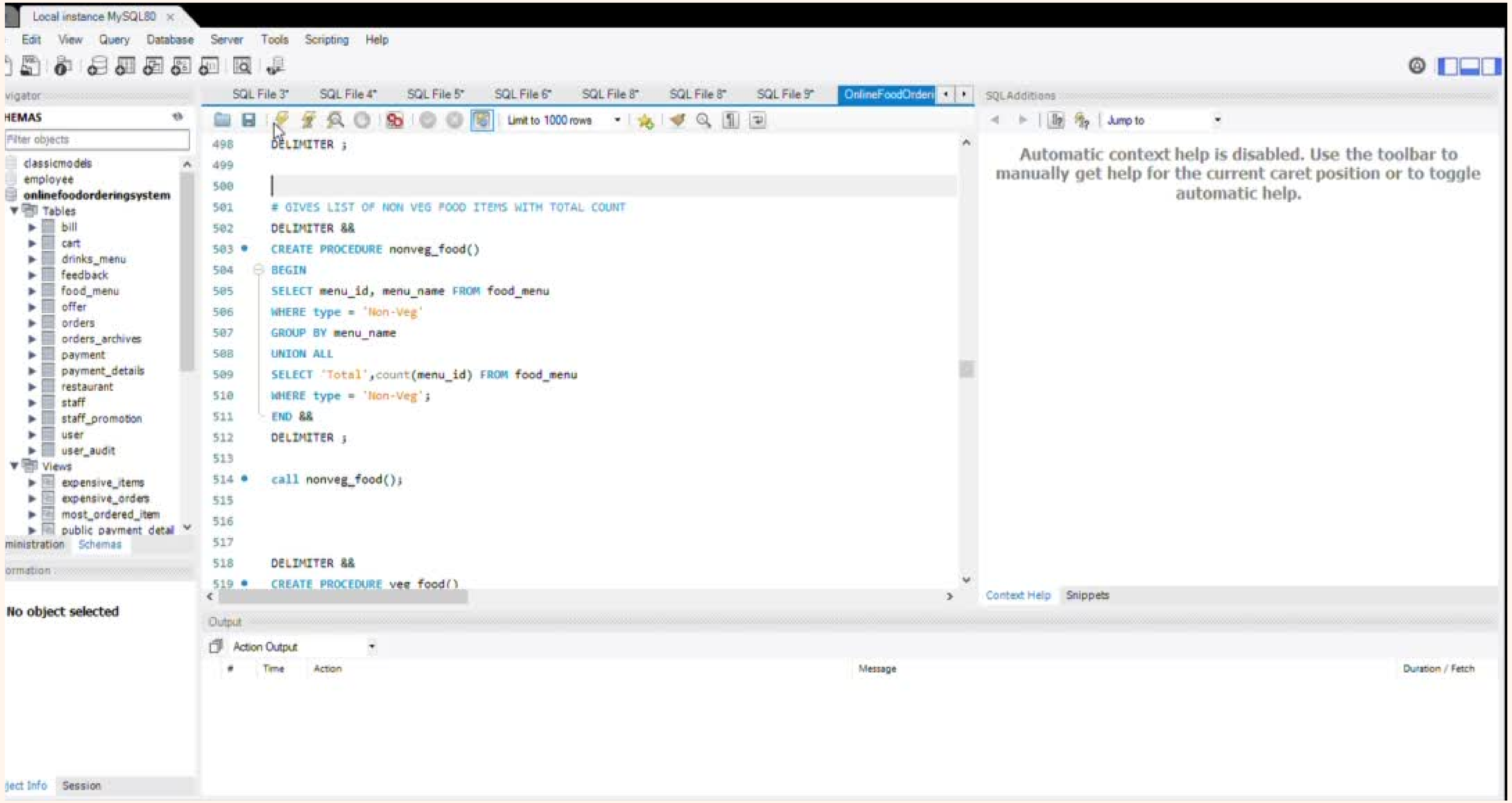


```sql
/*VIEW*/
CREATE VIEW most_ordered_item AS
SELECT Menu_name, count(*)
FROM bill
GROUP BY menu_name
ORDER BY COUNT(*) DESC
LIMIT 5;

SELECT * FROM foodorderingsystem.most_ordered_item;
```

```
#   Time      Action                                                                        Message               Duration / Fetch
28  19:28:31  CREATE VIEW expensive_orders AS SELECT menu_name, total_price, order_id, user_id...  0 row(s) affected     0.093 sec
29  19:28:31  SELECT * FROM foodorderingsystem.expensive_orders LIMIT 0, 1000                   11 row(s) returned    0.063 sec / 0.000 sec
```

Most_ordered_item view shows the food items which are mostly ordered .

STORED PROCEDURES

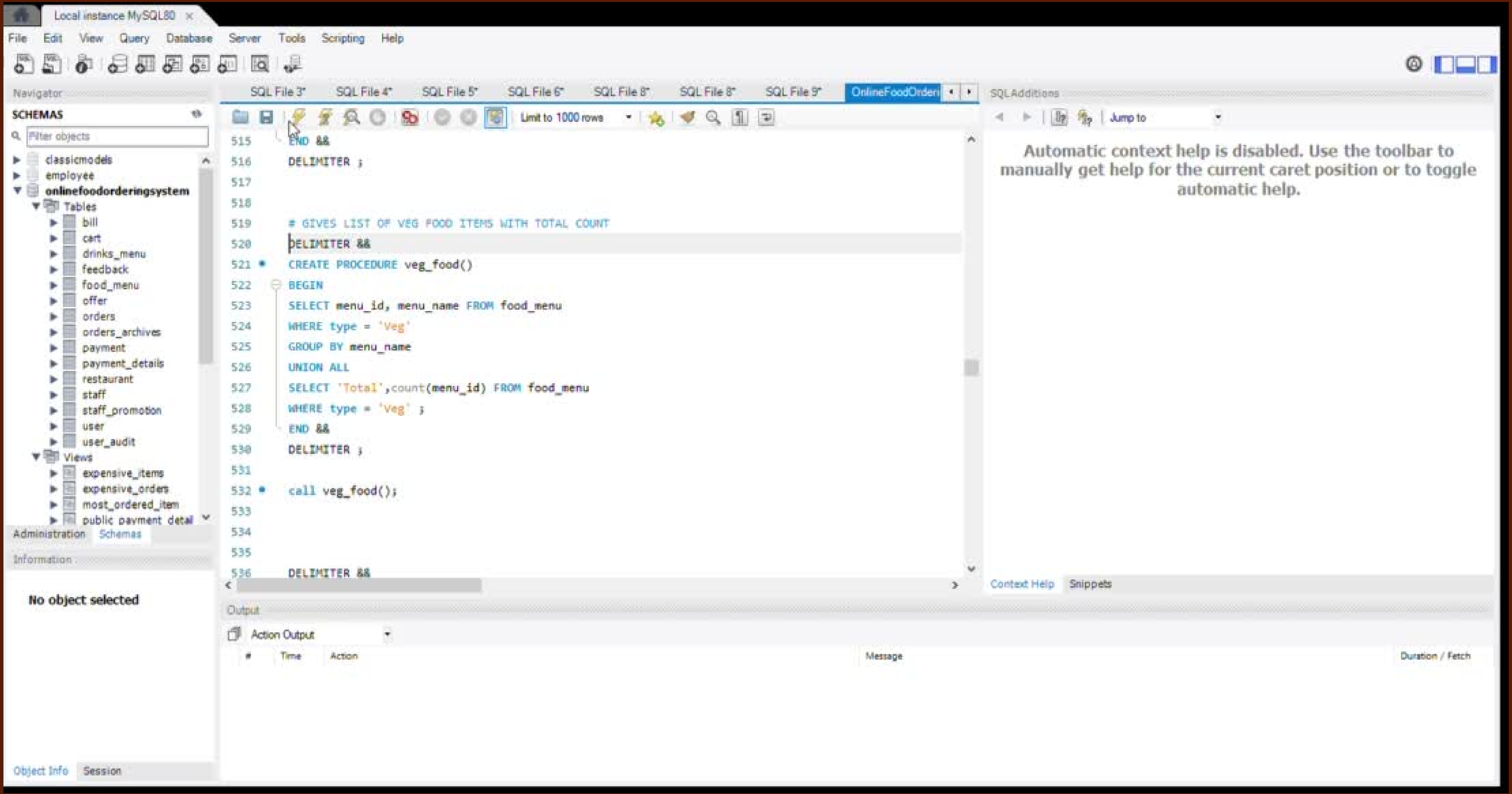


```
DELIMITER ;


# GIVES LIST OF NON VEG FOOD ITEMS WITH TOTAL COUNT
DELIMITER &&
CREATE PROCEDURE nonveg_food()
BEGIN
SELECT menu_id, menu_name FROM food_menu
WHERE type = 'Non-Veg'
GROUP BY menu_name
UNION ALL
SELECT 'Total',count(menu_id) FROM food_menu
WHERE type = 'Non-Veg';
END &&
DELIMITER ;

call nonveg_food();



DELIMITER &&
CREATE PROCEDURE veg_food()
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

No object selected

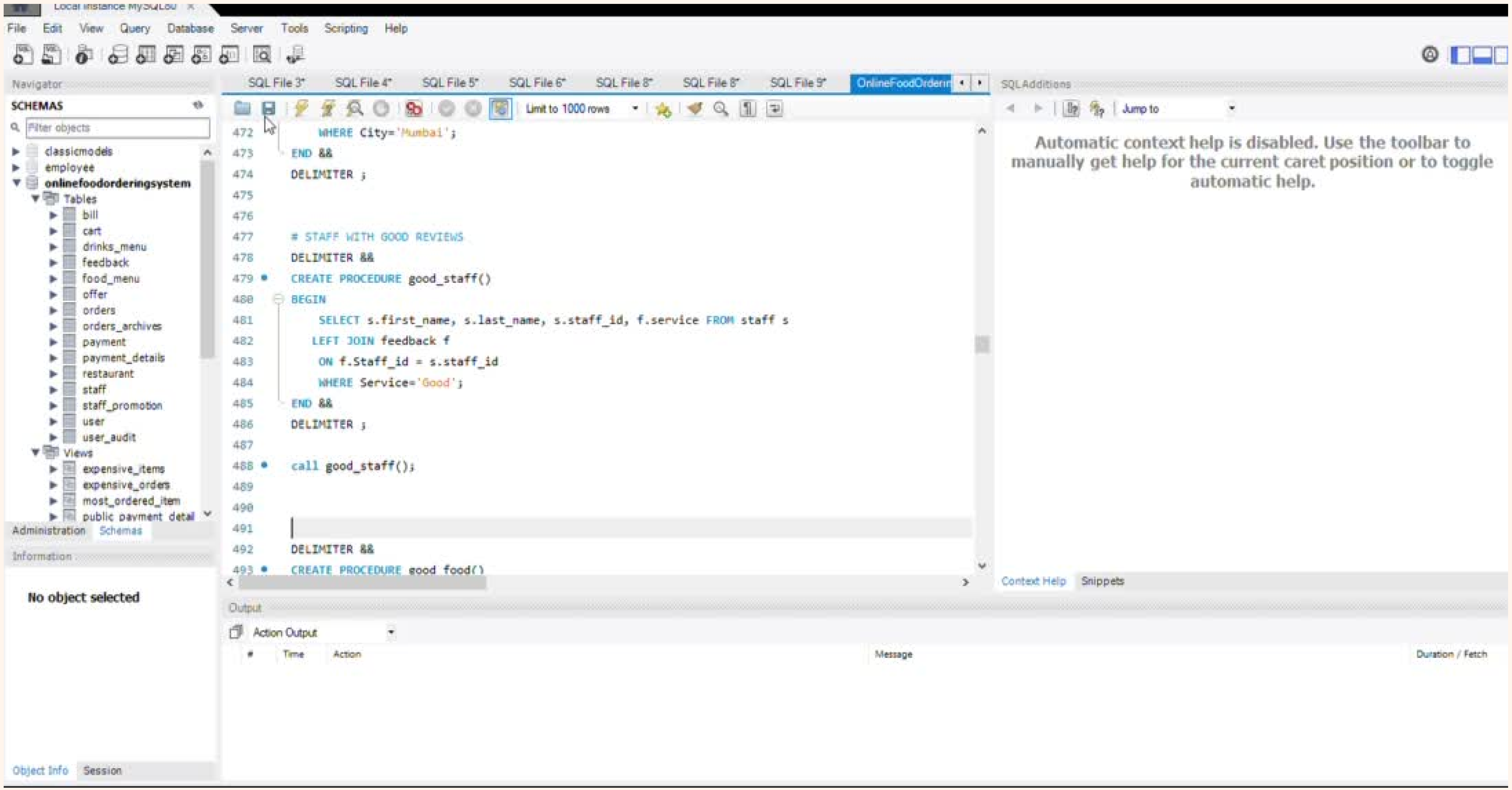
Nonveg_food(): This stored procedure gives us all the non-veg food items from the menu and their count.



```
END &&
DELIMITER ;


# GIVES LIST OF VEG FOOD ITEMS WITH TOTAL COUNT
DELIMITER &&
CREATE PROCEDURE veg_food()
BEGIN
SELECT menu_id, menu_name FROM food_menu
WHERE type = 'Veg'
GROUP BY menu_name
UNION ALL
SELECT 'Total',count(menu_id) FROM food_menu
WHERE type = 'Veg' ;
END &&
DELIMITER ;

call veg_food();



DELIMITER &&
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Veg_food(): This stored procedure gives us all the veg food items from the menu and their count.



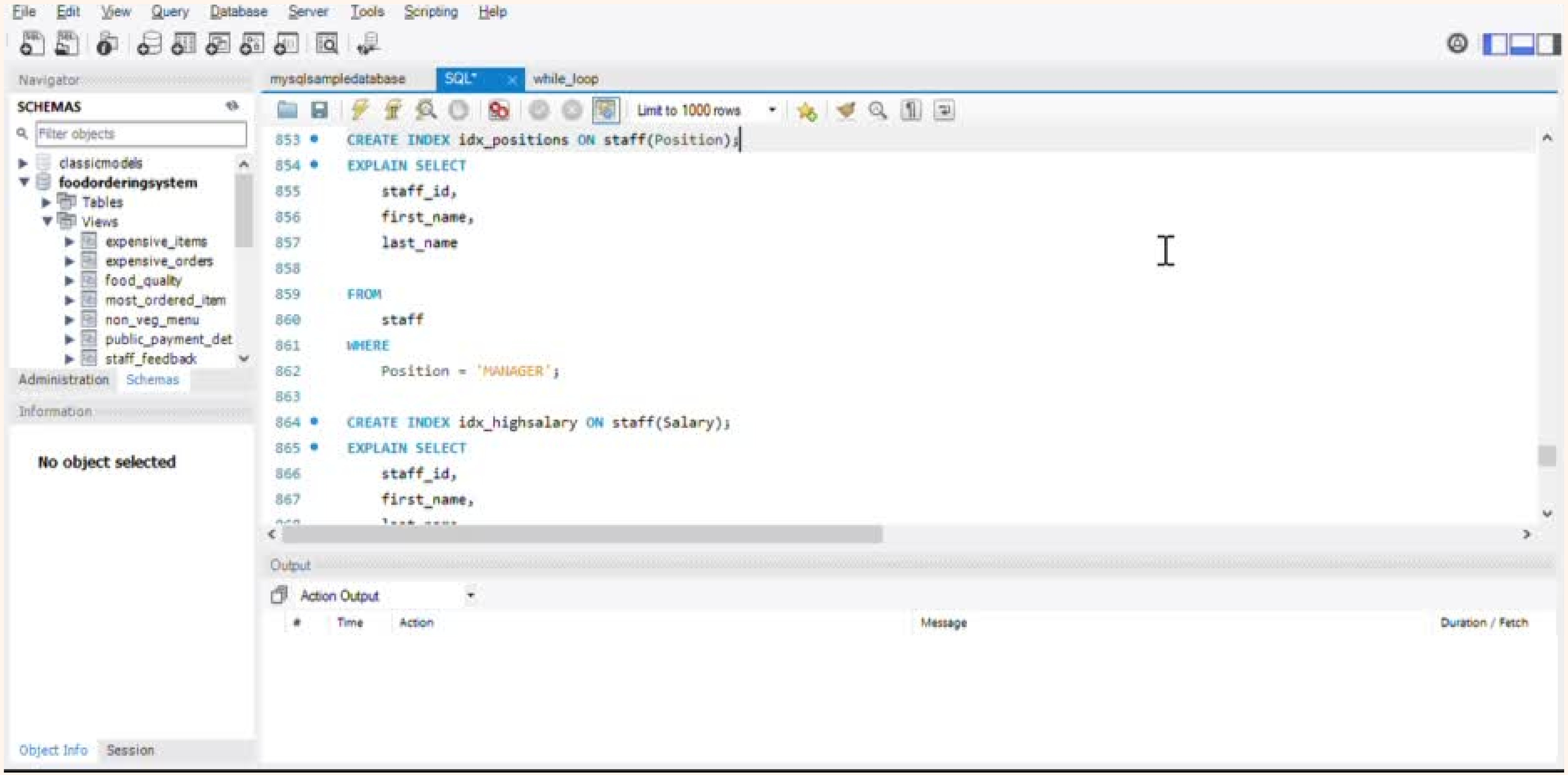
No object selected

```sql
    WHERE City='Mumbai';
END &&
DELIMITER ;


# STAFF WITH GOOD REVIEWS
DELIMITER &&
CREATE PROCEDURE good_staff()
BEGIN
    SELECT s.first_name, s.last_name, s.staff_id, f.service FROM staff s
   LEFT JOIN feedback f
    ON f.Staff_id = s.staff_id
    WHERE Service='Good';
END &&
DELIMITER ;

call good_staff();



DELIMITER &&
CREATE PROCEDURE good_food()
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Good_staff(): This stored procedure gives us all the full names of people from staff who got good service feedbacks.

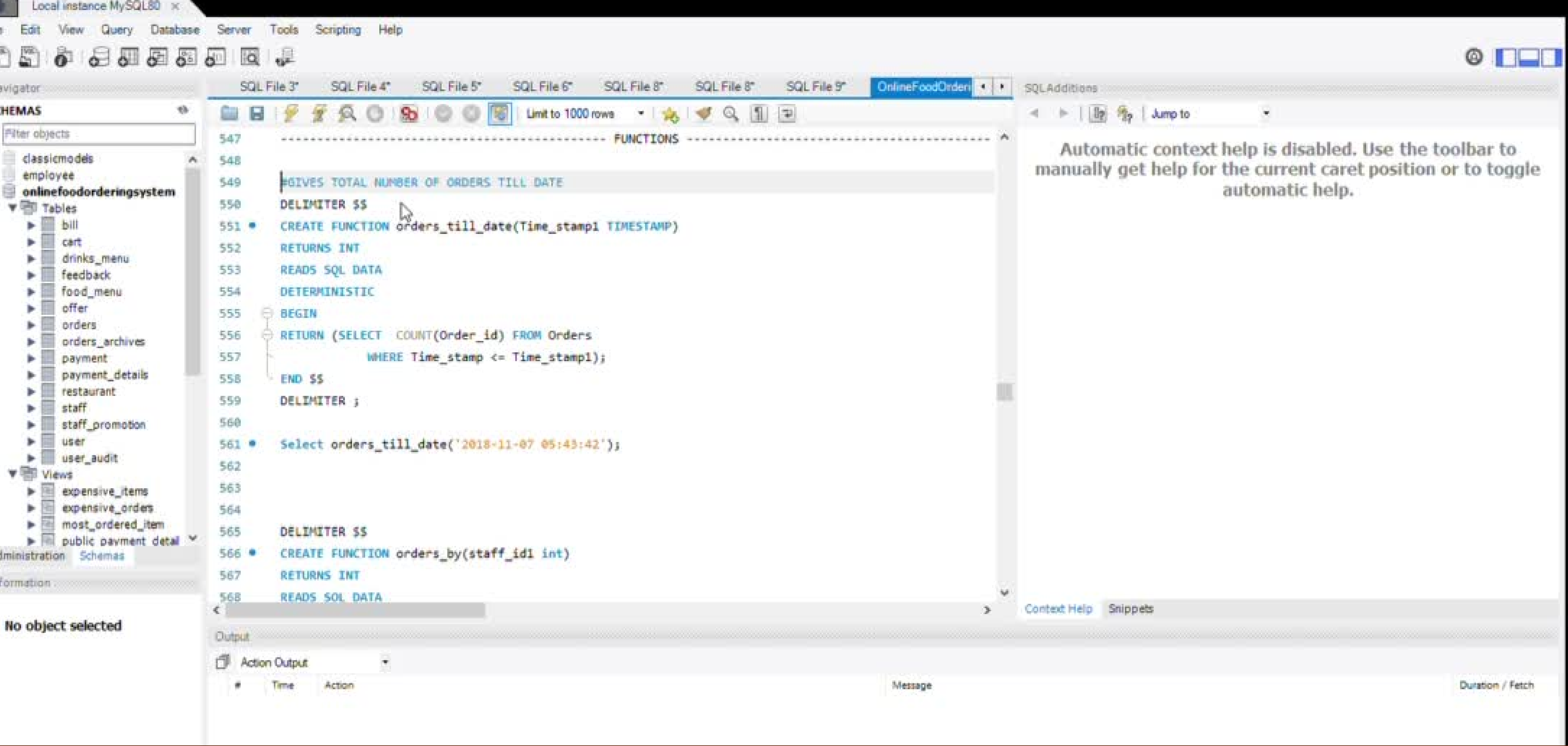
INDEXES



```
CREATE INDEX idx_positions ON staff(Position);
EXPLAIN SELECT
    staff_id,
    first_name,
    last_name

FROM
    staff
WHERE
    Position = 'MANAGER';

CREATE INDEX idx_highsalary ON staff(Salary);
EXPLAIN SELECT
    staff_id,
    first_name,
```

The idx_positions gives us the position of the staff. Specially the staff at the managers position . The idx_highsalary gives us the salary of the staff.

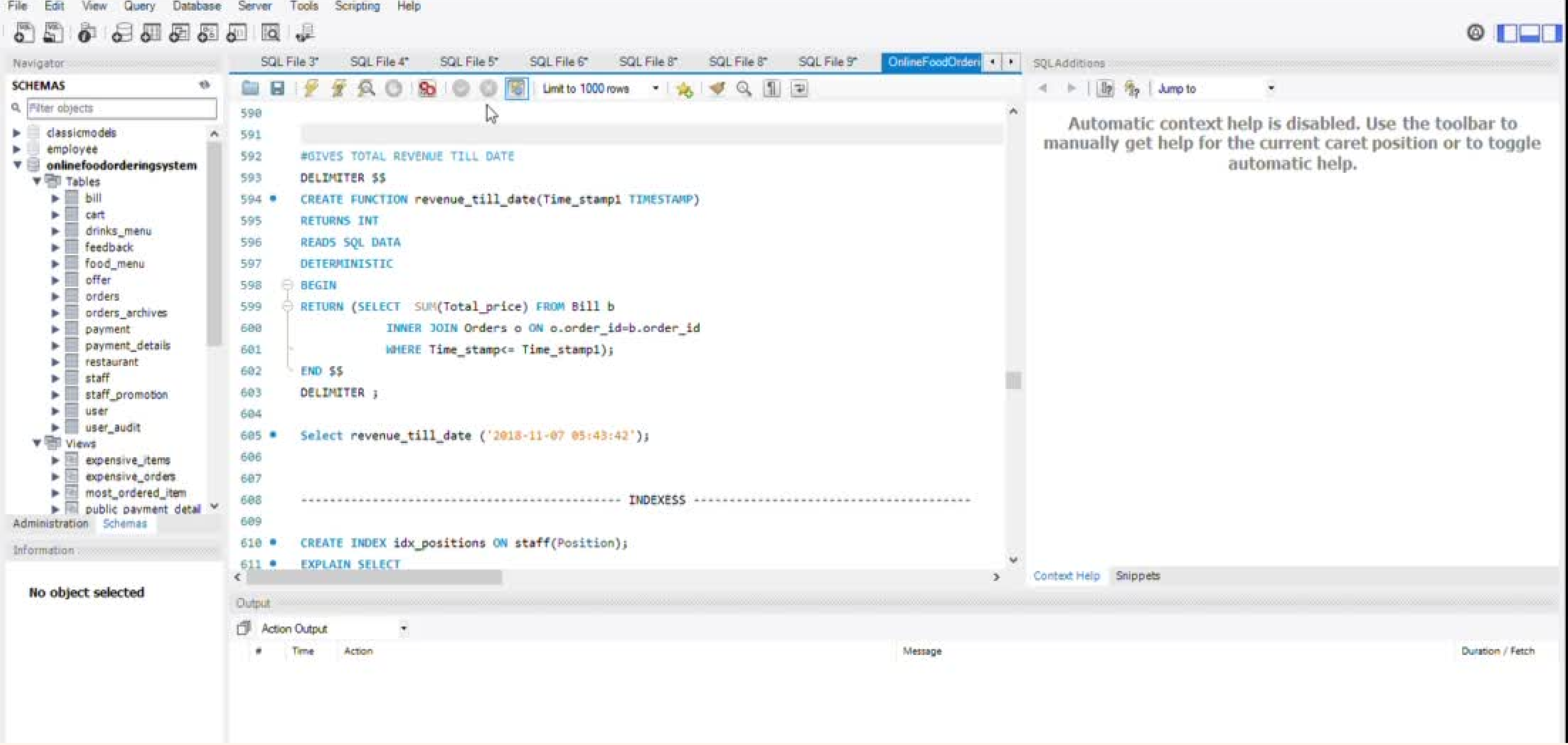
FUNCTIONS



```
------------------------------------------------- FUNCTIONS -------------------------------------------------

#GIVES TOTAL NUMBER OF ORDERS TILL DATE
DELIMITER $$
CREATE FUNCTION orders_till_date(Time_stamp1 TIMESTAMP)
RETURNS INT
READS SQL DATA
DETERMINISTIC
BEGIN
RETURN (SELECT  COUNT(Order_id) FROM Orders
            WHERE Time_stamp <= Time_stamp1);
END $$
DELIMITER ;

Select orders_till_date('2018-11-07 05:43:42');



DELIMITER $$
CREATE FUNCTION orders_by(staff_id1 int)
RETURNS INT
READS SQL DATA
```

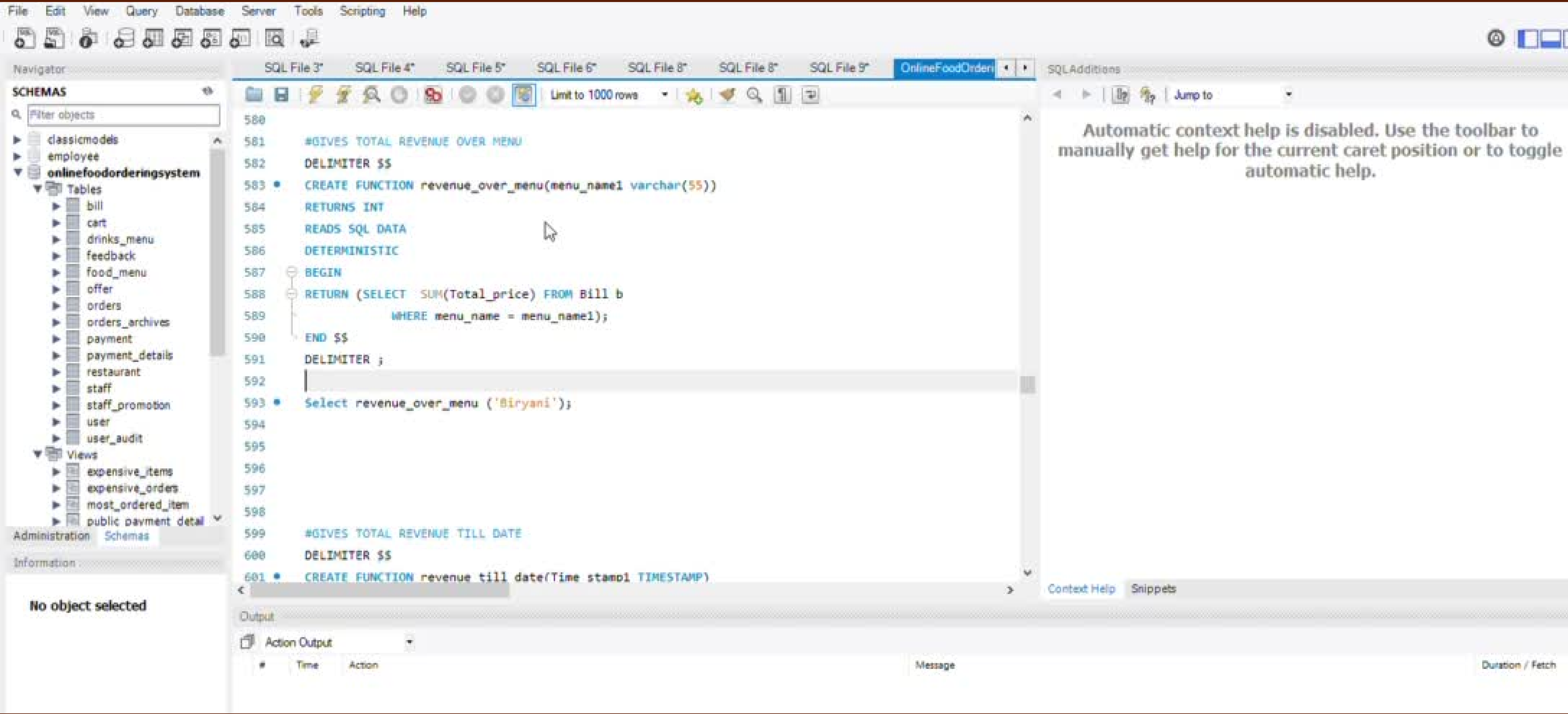
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Orders_till_date(): This function gives us a count of a total number of orders uptill a date, with timestamp parameter.



```sql

#GIVES TOTAL REVENUE TILL DATE
DELIMITER $$
CREATE FUNCTION revenue_till_date(Time_stamp1 TIMESTAMP)
RETURNS INT
READS SQL DATA
DETERMINISTIC
BEGIN
RETURN (SELECT  SUM(Total_price) FROM Bill b
          INNER JOIN Orders o ON o.order_id=b.order_id
          WHERE Time_stamp<= Time_stamp1);
END $$
DELIMITER ;

Select revenue_till_date ('2018-11-07 05:43:42');



------------------------------------------- INDEXESS ------------------------------------------

CREATE INDEX idx_positions ON staff(Position);
EXPLAIN SELECT
```

Revenue_till_date(): This function gives us the sum of the total revenue of orders till given date, with the timestamp parameter.

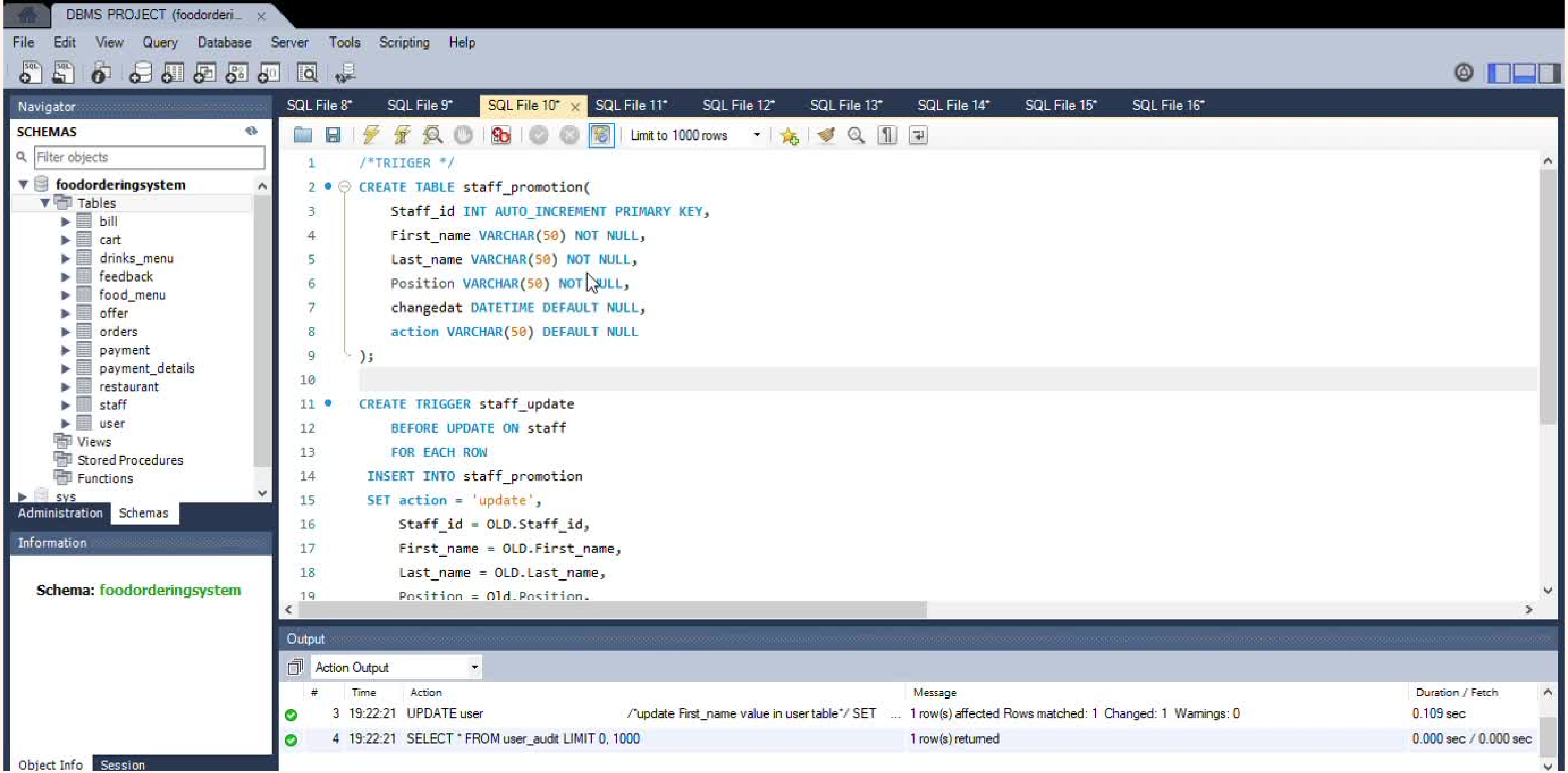


```
#GIVES TOTAL REVENUE OVER MENU
DELIMITER $$
CREATE FUNCTION revenue_over_menu(menu_name1 varchar(55))
RETURNS INT
READS SQL DATA
DETERMINISTIC
BEGIN
RETURN (SELECT  SUM(Total_price) FROM Bill b
            WHERE menu_name = menu_name1);
END $$
DELIMITER ;

Select revenue_over_menu ('Biryani');





#GIVES TOTAL REVENUE TILL DATE
DELIMITER $$
CREATE FUNCTION revenue_till_date(Time_stamp1 TIMESTAMP)
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Revenue_over_menu(): This function gives us the sum of total revenue over one food item from menu.

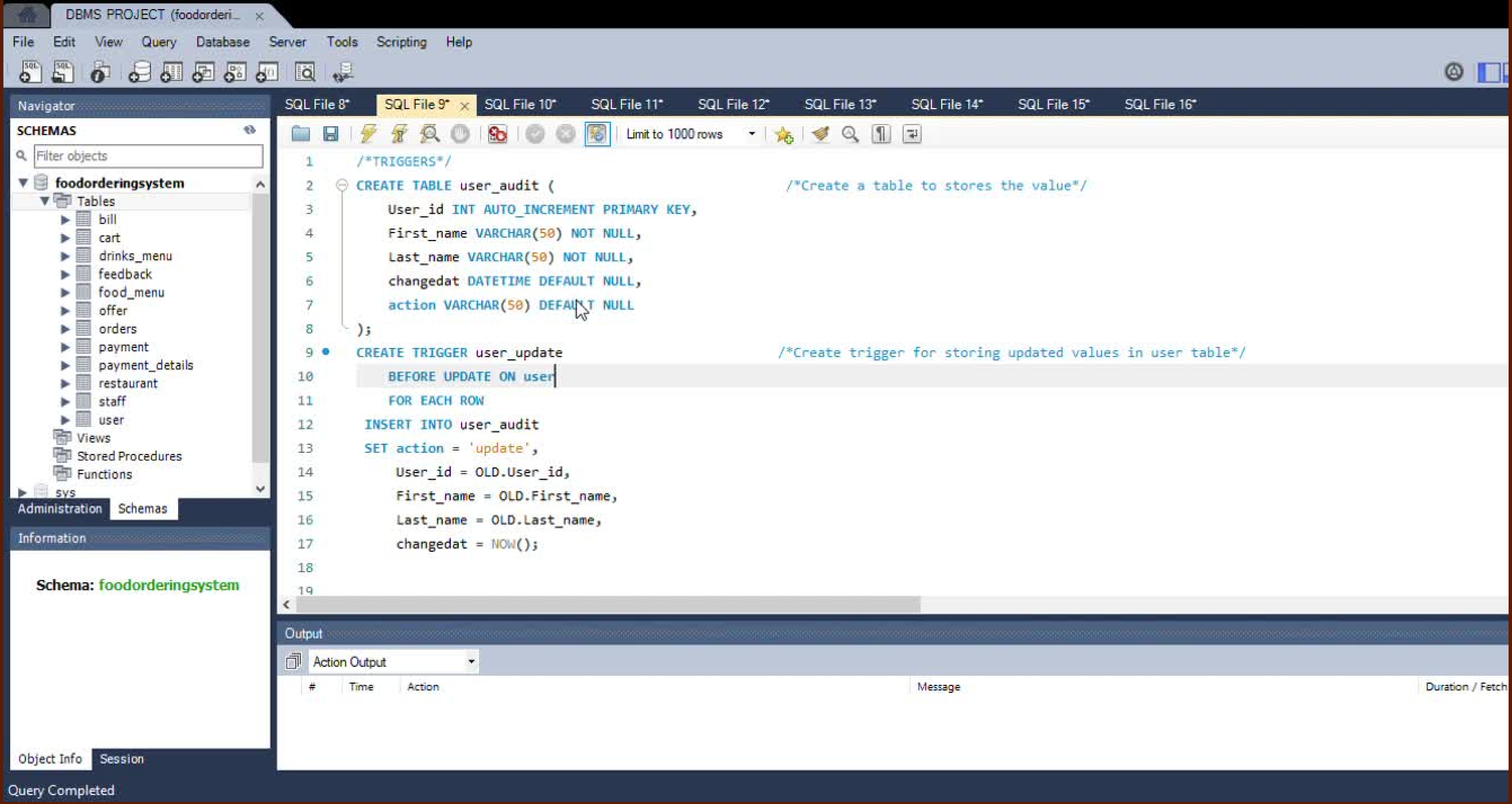
TRIGGERS



```sql
/*TRIIGER */
CREATE TABLE staff_promotion(
    Staff_id INT AUTO_INCREMENT PRIMARY KEY,
    First_name VARCHAR(50) NOT NULL,
    Last_name VARCHAR(50) NOT NULL,
    Position VARCHAR(50) NOT NULL,
    changedat DATETIME DEFAULT NULL,
    action VARCHAR(50) DEFAULT NULL
);

CREATE TRIGGER staff_update
    BEFORE UPDATE ON staff
    FOR EACH ROW
 INSERT INTO staff_promotion
 SET action = 'update',
     Staff_id = OLD.Staff_id,
     First_name = OLD.First_name,
     Last_name = OLD.Last_name,
     Position = Old.Position,
```

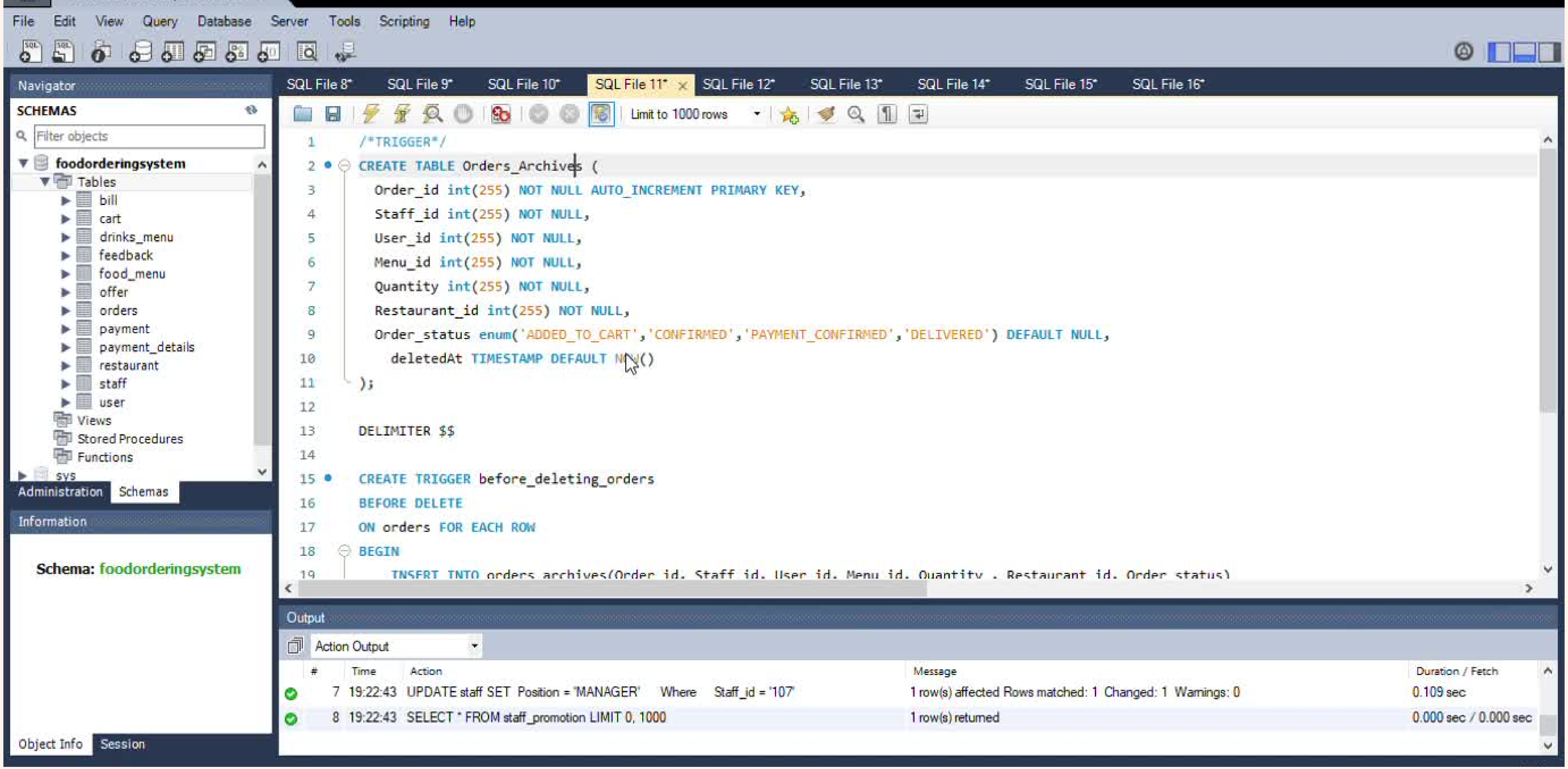
```
#  Time      Action                                                          Message                                                      Duration / Fetch
3  19:22:21  UPDATE user    /*update First_name value in user table*/ SET ...  1 row(s) affected Rows matched: 1  Changed: 1  Warnings: 0    0.109 sec
4  19:22:21  SELECT * FROM user_audit LIMIT 0, 1000                            1 row(s) returned                                            0.000 sec / 0.000 sec
```

Staff_update(): This trigger is invoked when data in the staff table is updated. The updated value is stored in Staff_promotion table.



```sql
/*TRIGGERS*/
CREATE TABLE user_audit (                                   /*Create a table to stores the value*/
    User_id INT AUTO_INCREMENT PRIMARY KEY,
    First_name VARCHAR(50) NOT NULL,
    Last_name VARCHAR(50) NOT NULL,
    changedat DATETIME DEFAULT NULL,
    action VARCHAR(50) DEFAULT NULL
);
CREATE TRIGGER user_update                                  /*Create trigger for storing updated values in user table*/
    BEFORE UPDATE ON user
    FOR EACH ROW
 INSERT INTO user_audit
 SET action = 'update',
     User_id = OLD.User_id,
     First_name = OLD.First_name,
     Last_name = OLD.Last_name,
     changedat = NOW();
```

User_update(): This trigger is invoked when data in the user table is updated, the updated value is stored in user_audit table.



Schema: foodorderingsystem

```
/*TRIGGER*/
CREATE TABLE Orders_Archives (
  Order_id int(255) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  Staff_id int(255) NOT NULL,
  User_id int(255) NOT NULL,
  Menu_id int(255) NOT NULL,
  Quantity int(255) NOT NULL,
  Restaurant_id int(255) NOT NULL,
  Order_status enum('ADDED_TO_CART','CONFIRMED','PAYMENT_CONFIRMED','DELIVERED') DEFAULT NULL,
    deletedAt TIMESTAMP DEFAULT NOW()
);

DELIMITER $$

CREATE TRIGGER before_deleting_orders
BEFORE DELETE
ON orders FOR EACH ROW
BEGIN
    INSERT INTO orders_archives(Order_id, Staff_id, User_id, Menu_id, Quantity , Restaurant_id, Order_status)
```

#	Time	Action	Message	Duration / Fetch
7	19:22:43	UPDATE staff SET Position = 'MANAGER' Where Staff_id = '107'	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.109 sec
8	19:22:43	SELECT * FROM staff_promotion LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

Before_deleting_orders(): This trigger is invoked when a value in orders table is deleted, the deleted value is stored in orders_archives table.

CONCLUSION

- The online food delivery system is the need of the hour because of the recent changes in the industry and the increasing use of the internet and so is a database for effective management of online food delivery system.
- Orders are recorded easily by this system; Information needed in making an order to the customer is provided by the system. Receiving orders and modifying its data is possible and it also helps the admin in controlling all the Food system.
- Furthermore, budgeting can be optimized.
- Mistakes of duplicate entries, calculation problem, miscommunication can be eliminated.
- The functionality of updated details is available to the manager at all times.
- To sum up, database management is an integral part of the food ordering system.

THANK YOU!