DBMS MINIPROJECT

RESTAURANT INVENTORY MANAGEMENT SYSTEM

GitHub link:

UE21CS351A:DATABASE MANAGEMENT SYSTEM

MINI PROJECT-USER REQUIREMENT SPECIFICATION

MEMBERS: PES1UG21CS166-DARSH AGGARWAL

PES1UG21CS183-DISHA JAIN

Table of Contents

INTRODUCTION

Purpose of the project:

A restaurant inventory management system efficiently tracks and controls food and ingredient levels, minimizes waste, ensures menu consistency, optimizes ordering, and enhances profitability by providing real-time data and analytics to streamline operations and improve customer service.

Scope of the Project

The scope of the project encompasses designing and implementing a comprehensive software solution. It will include features for real-time tracking of food and beverage inventory, automated reordering based on predefined thresholds, supplier management, menu synchronization, cost analysis, and reporting.

The system will be user-friendly, accessible from multiple devices, and capable of generating insightful analytics. Additionally, it can be integrate with point-of-sale (POS) systems and barcode scanners for seamless data entry.

The project aims to enhance operational efficiency, reduce costs, minimize wastage, and improve overall inventory .

PROJECT DESCRIPTION

PROJECT OVERVIEW

The project involves developing a restaurant inventory management system, a software solution designed to streamline food and beverage inventory processes.

It will enable real-time tracking, automated reordering, supplier management, cost analysis, and reporting. The system will integrate with existing POS systems and barcode scanners, enhancing operational efficiency, reducing costs, and minimizing wastage.

The primary goal is to optimize restaurant profitability and customer satisfaction by providing comprehensive inventory control and data analytics.

MAJOR PROJECT FUNCTIONALITIES

The project will include the following major functionalities:

Inventory Tracking:

Real-time monitoring of food and beverage stock levels to ensure accurate inventory counts.

Automated Reordering:

System-triggered reorder requests based on predefined inventory thresholds to prevent stockouts.

Menu Synchronization:

Ensuring that the menu reflects current inventory availability to prevent customer disappointment.

Cost Analysis:

Analyzing costs associated with inventory management and identifying cost-saving opportunities.

Reporting:

Generating detailed reports on inventory levels, usage patterns, and financial insights.

Integration:

Seamless integration with point-of-sale (POS) systems and barcode scanners for data synchronization.

User-Friendly Interface:

An intuitive and user-friendly interface for ease of use by restaurant staff.

Analytics:

Providing data analytics to make informed decisions regarding purchasing and menu planning.

Accessibility:

SYSTEM FEATURES AND FUNCTIONAL REQUIREMENTS

System Feature 1: Inventory Tracking

Functional Requirements:

Entities:

- **Input:** Users should be able to input new inventory items, edit existing items, and delete items when necessary.
- **Database Entities:** The system should maintain a database of inventory items, including fields for item name, category, quantity on hand, unit price, reorder point, supplier information, and storage location.

System Feature 2: Automated Reordering

Functional Requirements:

Entities:

- **Input:** Users should be able to set reorder points for each inventory item.
- **Database Entities:** The system should track the reorder points for each item and generate purchase orders automatically when inventory falls below these points. Purchase orders should include item details, quantities, and supplier information.

System Feature 3: Menu Synchronization

Functional Requirements:

Entities:

- **Input:** Users should have the ability to manually adjust the menu if needed.
- **Database Entities:** The system should automatically update the menu based on the current inventory availability. Menu items should be linked to inventory items to prevent offering out-of-stock items.

System Feature 4: Integration

Functional Requirements:

Entities:

- **Input:** Users should be able to configure integration settings with POS systems, barcode scanners, and accounting software.
- **Database Entities:** The system should establish seamless integration with these external systems through APIs or data import/export functionality, ensuring accurate data synchronization.

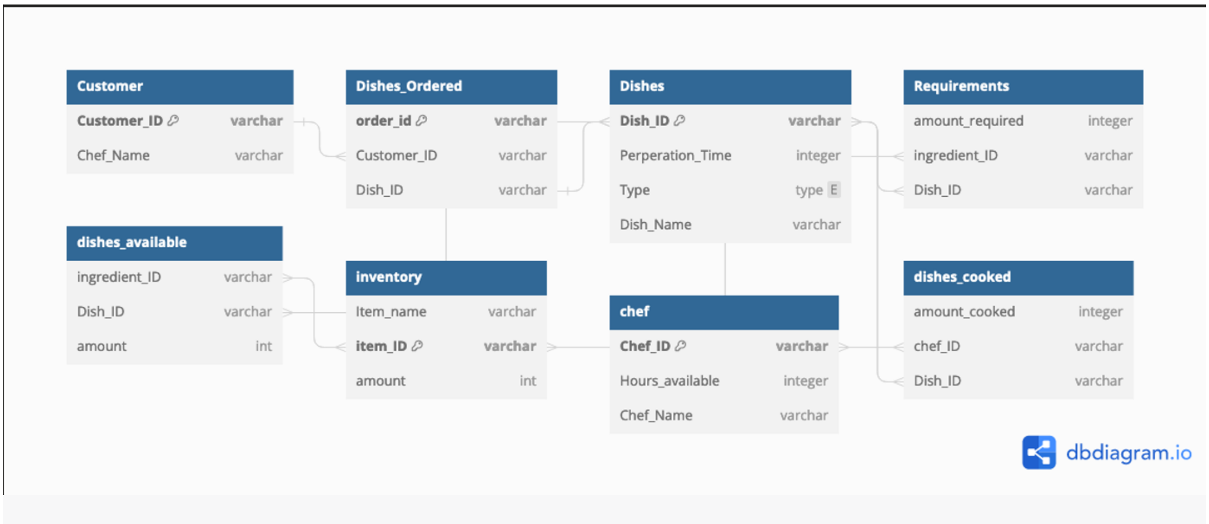
System Feature 5: Accessibility

Functional Requirements:

Entities:

- **Input:** Users should be able to access the system from various devices, including desktop computers, mobile devices, and tablets.
- **Database Entities:** The system should have a responsive user interface that provides accessibility and usability across different devices and screen sizes.

RELATIONAL SCHEMA



Customer	
Customer_ID	**varchar**
Chef_Name	varchar

Dishes_Ordered	
order_id	**varchar**
Customer_ID	varchar
Dish_ID	varchar

Dishes	
Dish_ID	**varchar**
Perperation_Time	integer
Type	type E
Dish_Name	varchar

Requirements	
amount_required	integer
ingredient_ID	varchar
Dish_ID	varchar

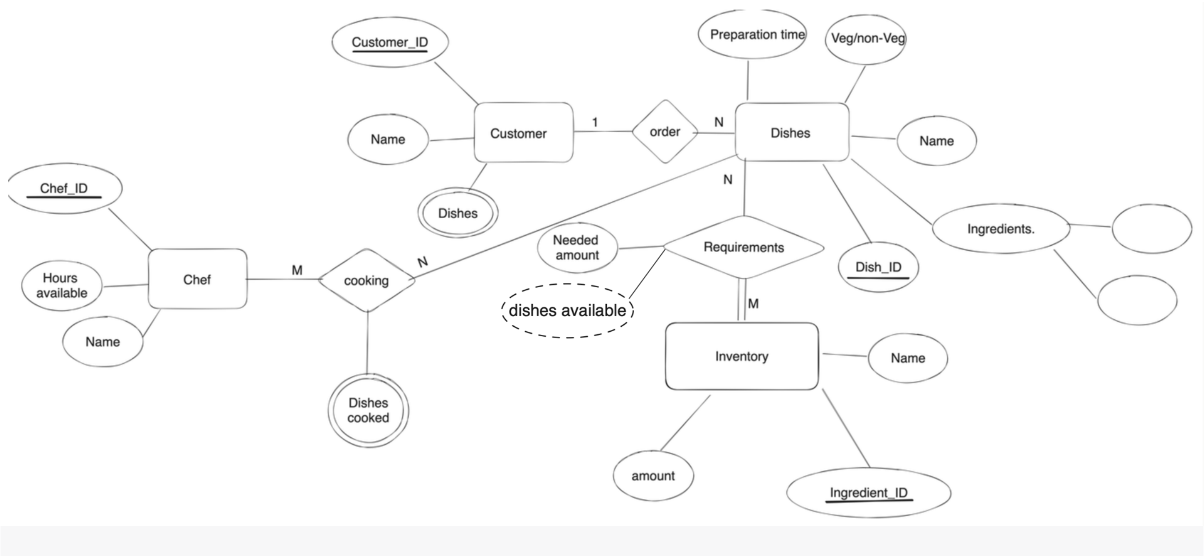
dishes_available	
ingredient_ID	varchar
Dish_ID	varchar
amount	int

inventory	
Item_name	varchar
item_ID	**varchar**
amount	int

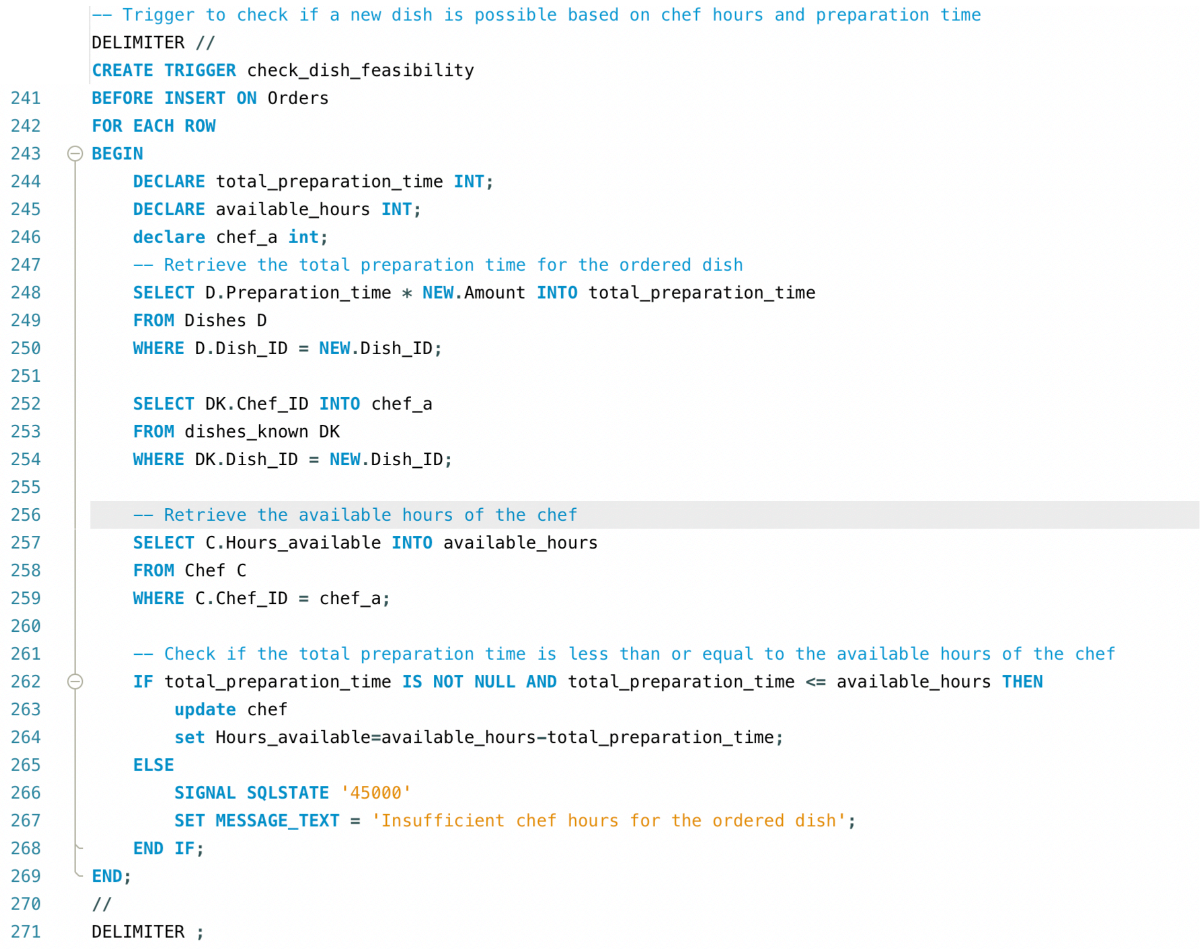
chef	
Chef_ID	**varchar**
Hours_available	integer
Chef_Name	varchar

dishes_cooked	
amount_cooked	integer
chef_ID	varchar
Dish_ID	varchar

ER DIAGRAM

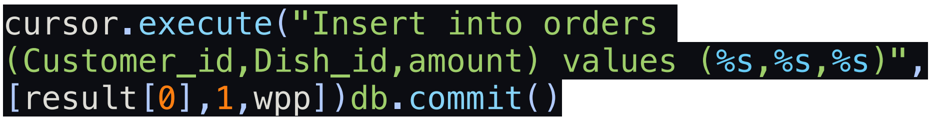


Trigger definition (any one)



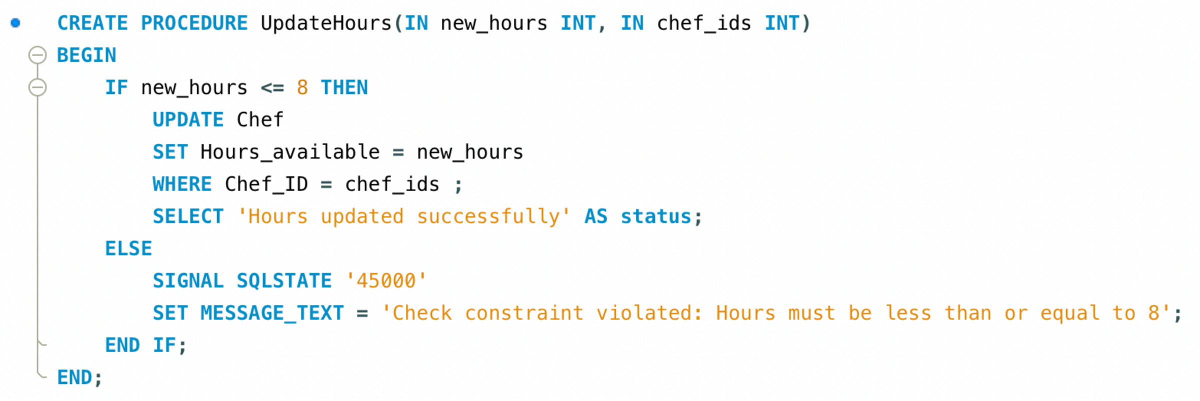
```
-- Trigger to check if a new dish is possible based on chef hours and preparation time
DELIMITER //
CREATE TRIGGER check_dish_feasibility
BEFORE INSERT ON Orders
FOR EACH ROW
BEGIN
    DECLARE total_preparation_time INT;
    DECLARE available_hours INT;
    declare chef_a int;
    -- Retrieve the total preparation time for the ordered dish
    SELECT D.Preparation_time * NEW.Amount INTO total_preparation_time
    FROM Dishes D
    WHERE D.Dish_ID = NEW.Dish_ID;

    SELECT DK.Chef_ID INTO chef_a
    FROM dishes_known DK
    WHERE DK.Dish_ID = NEW.Dish_ID;

    -- Retrieve the available hours of the chef
    SELECT C.Hours_available INTO available_hours
    FROM Chef C
    WHERE C.Chef_ID = chef_a;

    -- Check if the total preparation time is less than or equal to the available hours of the chef
    IF total_preparation_time IS NOT NULL AND total_preparation_time <= available_hours THEN
        update chef
        set Hours_available=available_hours-total_preparation_time;
    ELSE
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Insufficient chef hours for the ordered dish';
    END IF;
END;
//
DELIMITER ;
```

Statements that execute the trigger:



```
cursor.execute("Insert into orders
(Customer_id,Dish_id,amount) values (%s,%s,%s)",
[result[0],1,wpp])db.commit()
```

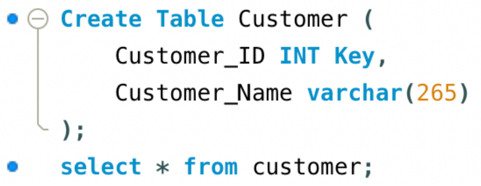
Procedure definition (any one)



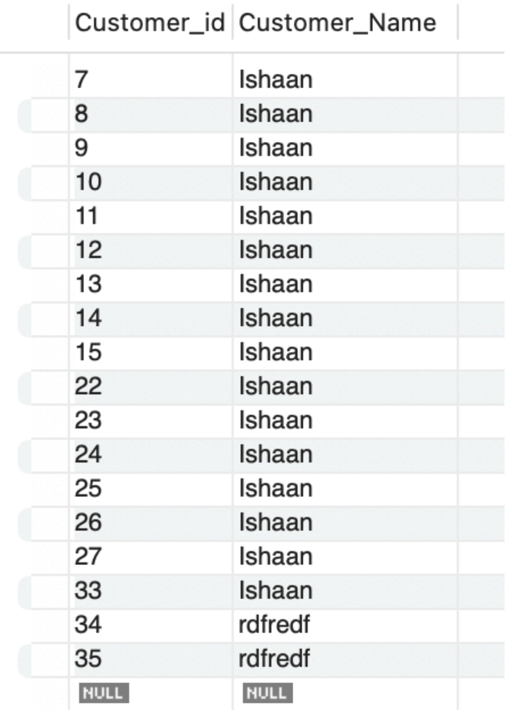
```
CREATE PROCEDURE UpdateHours(IN new_hours INT, IN chef_ids INT)
BEGIN
    IF new_hours <= 8 THEN
        UPDATE Chef
        SET Hours_available = new_hours
        WHERE Chef_ID = chef_ids ;
        SELECT 'Hours updated successfully' AS status;
    ELSE
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Check constraint violated: Hours must be less than or equal to 8';
    END IF;
END;
```

Queries – any 4 sample queries

1.Creating a table and displaying it.

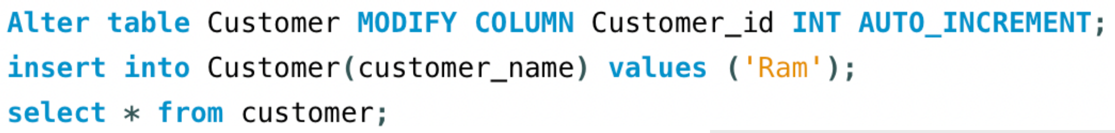


```
Create Table Customer (
    Customer_ID INT Key,
    Customer_Name varchar(265)
);
select * from customer;
```

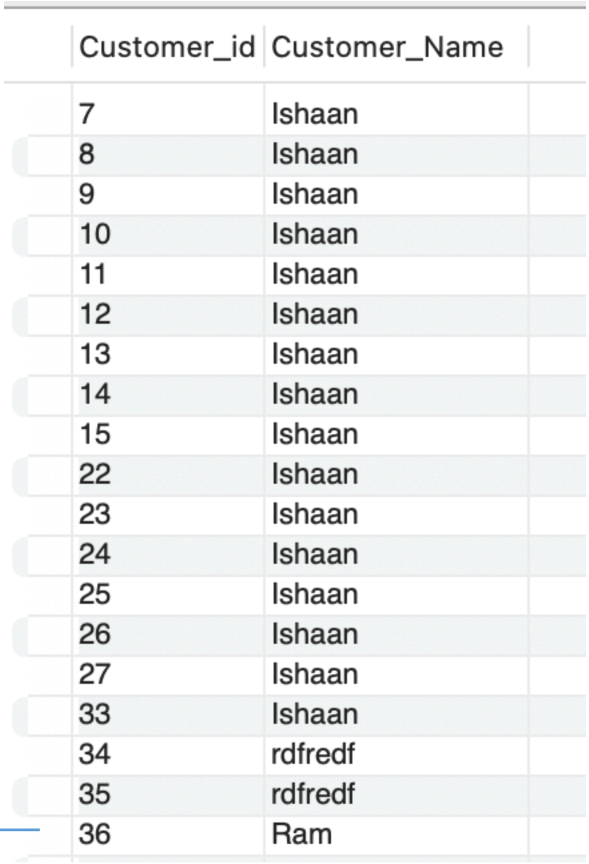


Customer_id	Customer_Name
7	Ishaan
8	Ishaan
9	Ishaan
10	Ishaan
11	Ishaan
12	Ishaan
13	Ishaan
14	Ishaan
15	Ishaan
22	Ishaan
23	Ishaan
24	Ishaan
25	Ishaan
26	Ishaan
27	Ishaan
33	Ishaan
34	rdfredf
35	rdfredf
NULL	NULL

2. Adding auto-increment constraint to the column of customer id.



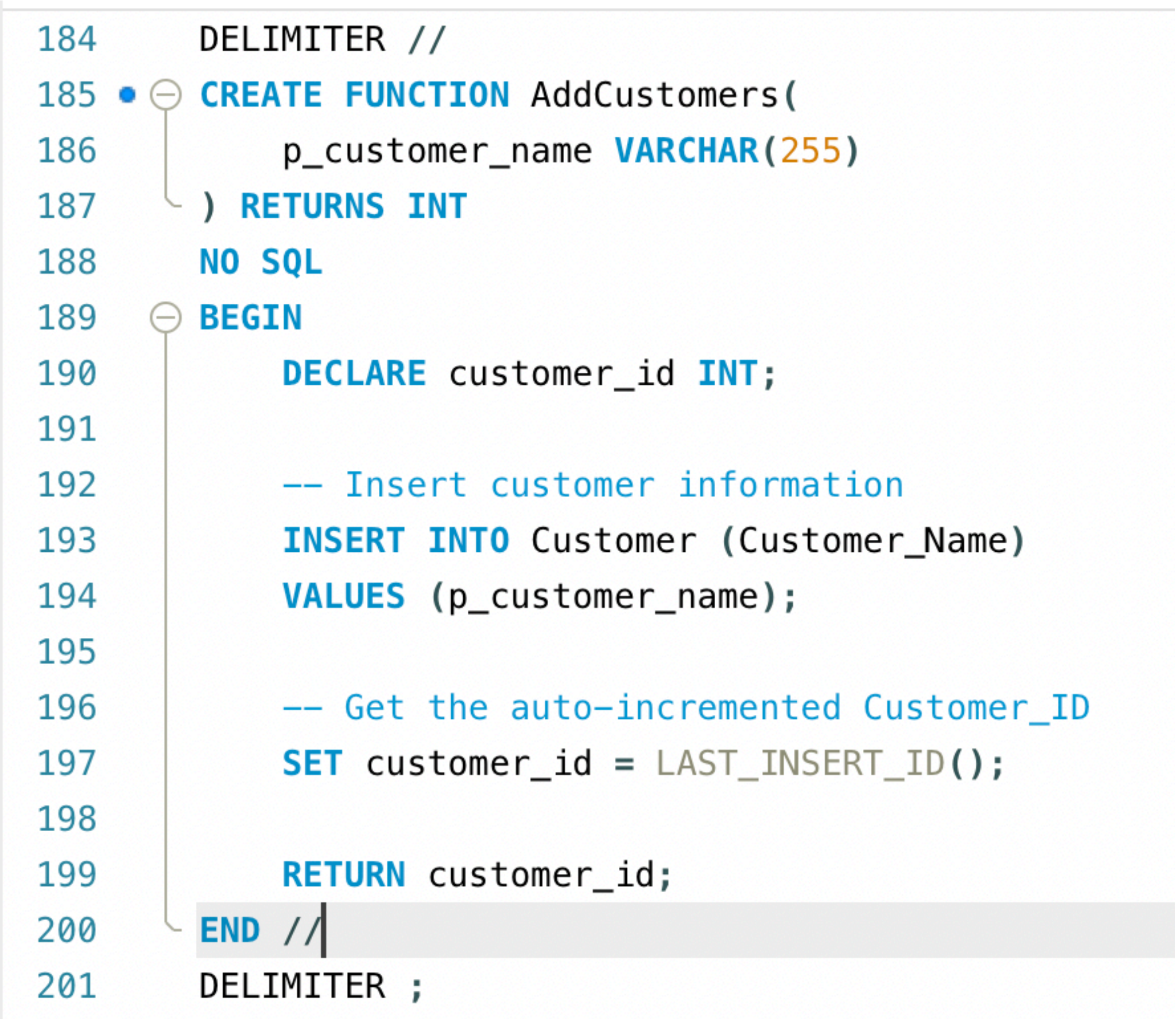
```
Alter table Customer MODIFY COLUMN Customer_id INT AUTO_INCREMENT;
insert into Customer(customer_name) values ('Ram');
select * from customer;
```



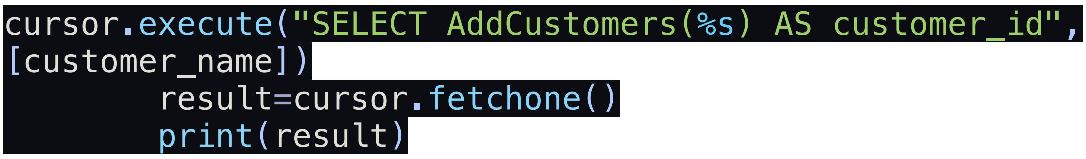
Customer_id	Customer_Name
7	Ishaan
8	Ishaan
9	Ishaan
10	Ishaan
11	Ishaan
12	Ishaan
13	Ishaan
14	Ishaan
15	Ishaan
22	Ishaan
23	Ishaan
24	Ishaan
25	Ishaan
26	Ishaan
27	Ishaan
33	Ishaan
34	rdfredf
35	rdfredf
36	Ram

Customer_id auto incremented and given

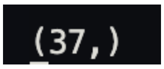
3. Creating a function that returns the customer_id on basis of customer_name.



```
DELIMITER //
CREATE FUNCTION AddCustomers(
    p_customer_name VARCHAR(255)
) RETURNS INT
NO SQL
BEGIN
    DECLARE customer_id INT;

    -- Insert customer information
    INSERT INTO Customer (Customer_Name)
    VALUES (p_customer_name);

    -- Get the auto-incremented Customer_ID
    SET customer_id = LAST_INSERT_ID();

    RETURN customer_id;
END //
DELIMITER ;
```

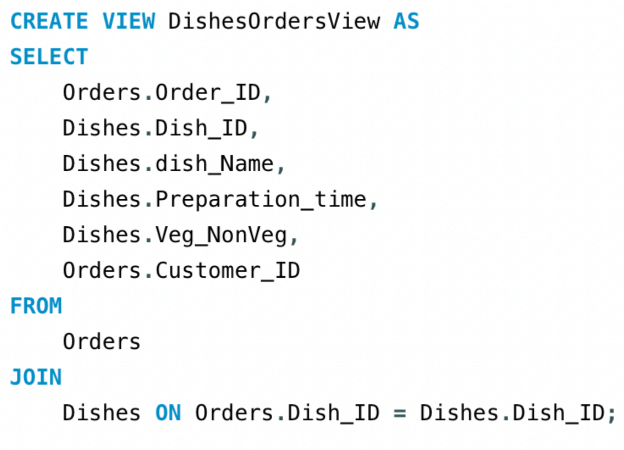


```
cursor.execute("SELECT AddCustomers(%s) AS customer_id",
[customer_name])
        result=cursor.fetchone()
        print(result)
```



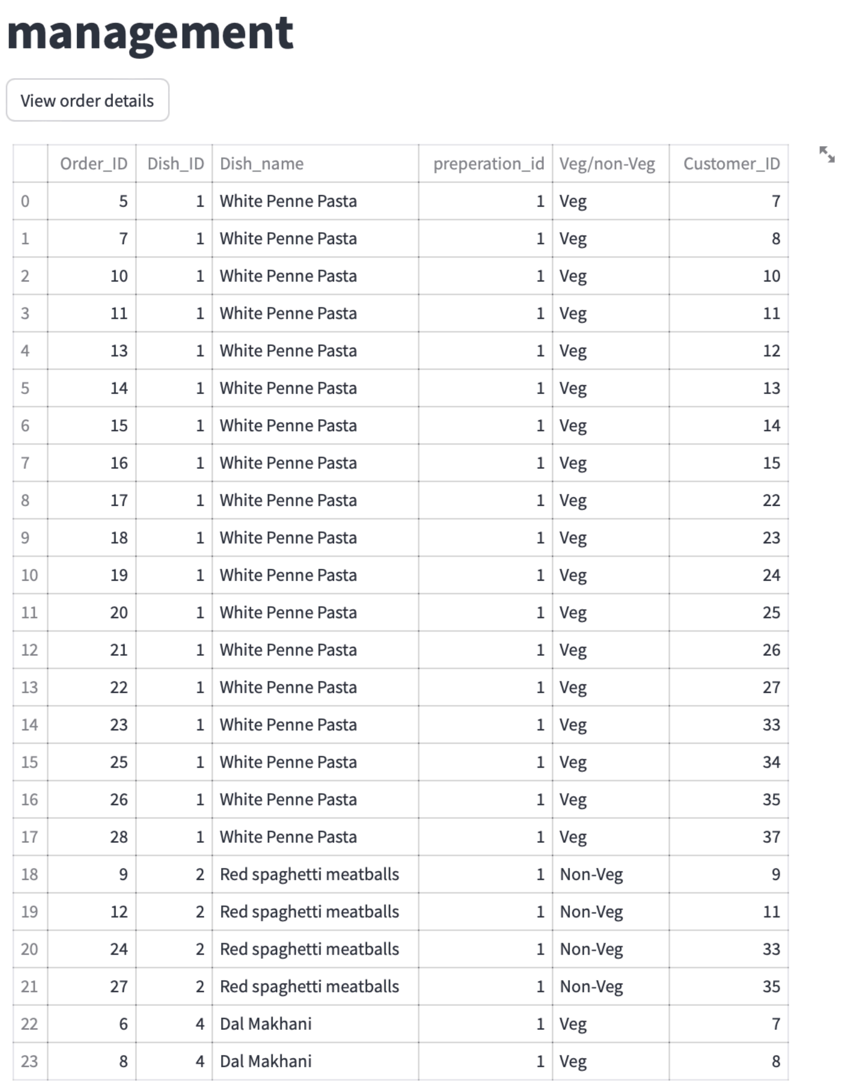
```
(37,)
```

4. Creating a function that returns the customer_id on basis of customer_name.



```
CREATE VIEW DishesOrdersView AS
SELECT
    Orders.Order_ID,
    Dishes.Dish_ID,
    Dishes.dish_Name,
    Dishes.Preparation_time,
    Dishes.Veg_NonVeg,
    Orders.Customer_ID
FROM
    Orders
JOIN
    Dishes ON Orders.Dish_ID = Dishes.Dish_ID;
```

```
if st.button ('view ingredient requirements'):
    cursor.execute("SELECT * FROM DishIngredientsView")
    data_t=cursor.fetchall()
    data_t=pd.DataFrame(data_t,columns=['Dish_name', 'Dish_ID',
'ingridient_ID', 'Ingredient_name', 'amount_Required'])
    st.table(data_t)
```



management

	Order_ID	Dish_ID	Dish_name	preperation_id	Veg/non-Veg	Customer_ID
0	5	1	White Penne Pasta	1	Veg	7
1	7	1	White Penne Pasta	1	Veg	8
2	10	1	White Penne Pasta	1	Veg	10
3	11	1	White Penne Pasta	1	Veg	11
4	13	1	White Penne Pasta	1	Veg	12
5	14	1	White Penne Pasta	1	Veg	13
6	15	1	White Penne Pasta	1	Veg	14
7	16	1	White Penne Pasta	1	Veg	15
8	17	1	White Penne Pasta	1	Veg	22
9	18	1	White Penne Pasta	1	Veg	23
10	19	1	White Penne Pasta	1	Veg	24
11	20	1	White Penne Pasta	1	Veg	25
12	21	1	White Penne Pasta	1	Veg	26
13	22	1	White Penne Pasta	1	Veg	27
14	23	1	White Penne Pasta	1	Veg	33
15	25	1	White Penne Pasta	1	Veg	34
16	26	1	White Penne Pasta	1	Veg	35
17	28	1	White Penne Pasta	1	Veg	37
18	9	2	Red spaghetti meatballs	1	Non-Veg	9
19	12	2	Red spaghetti meatballs	1	Non-Veg	11
20	24	2	Red spaghetti meatballs	1	Non-Veg	33
21	27	2	Red spaghetti meatballs	1	Non-Veg	35
22	6	4	Dal Makhani	1	Veg	7
23	8	4	Dal Makhani	1	Veg	8