**RESTAURANT MANAGEMENT SYSTEM**

**Database Design Document**

**V 3.0**

**By**

|  |  |  |
| --- | --- | --- |
| 1 | Abu Bakar | NUM-BSCS-2022-41 |
| 2 | Danish Abdullah Khan | NUM-BSCS-2022-05 |
| 3 | Zunaira Akbar | NUM-BSCS-2022-34 |



**Department of Computer Sciences**

**Namal University**

**Mianwali, Pakistan**

**Submission Date: 30th June 2024**

**REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Approved by** |
| *30/6/24* | *V.3.0* | *Change the Database Structure a little bit just Physical Structure*  *Change the Project Descriptions we can’t put that context we give in Project proposal.*  *Change the name of Project from “Food Management System” to “Rstaurant management System”* | **Mam Asiya** |
| *10/6/24* | *V.2.0* | *Change the ERD*  *Give the attributes datatypes according to their nature.*  *Define primary keys and foreign keys.*  *Define the Relations between the tables* | **Mam Asiya** |
| *22/04/24* | *V 1.0* | *Specify the changes implemented after the submission of the previous document. These changes should be based on the suggestions given by the person who approved the document.* |  |

*Instructions:*

* *Place the latest revisions at the top of the table.*
* *The Revision History pertains only to changes in the document's content or any updates made after a suggestion from the approving authority. It does not apply to the template's formatting.*

**TABLE OF CONTENTS**

[CHAPTER 1: PROJECT OVERVIEW 4](#_Toc170839581)

[1.1. INTRODUCTION: 4](#_Toc170839582)

[1.2. PROBLEM STATEMENT: 4](#_Toc170839583)

[1.3. PROJECT OBJECTIVES: 4](#_Toc170839584)

[1.4. DOCUMENT OBJECTIVES: 5](#_Toc170839585)

[CHAPTER 2: DETAILED DATABASE DESIGN 6](#_Toc170839586)

[2.1. ENTITY: 6](#_Toc170839588)

[2.2. DATA DICTIONARY: 6](#_Toc170839589)

[2.3. RELATIONSHIPS: 8](#_Toc170839590)

[2.4. ENTITY RELATIONSHIP DIAGRAM: 8](#_Toc170839591)

[Chapter 3 : Logical DATABASE DESIGN 9](#_Toc170839592)

[3.1. RELATIONAL SCHEMA: 9](#_Toc170839593)

[3.2. FUNCTIONAL DEPENDENCIES: 9](#_Toc170839594)

[3.3. NORMALIZATION: 10](#_Toc170839595)

[Chapter 4 : Physical DATABASE DESIGN 11](#_Toc170839596)

[4.1. STRUCTURE OF THE TABLES: 11](#_Toc170839597)

[4.2. DATA SAMPLES INSIDE TABLES: 12](#_Toc170839598)

[4.3. QUERIES RESULTS: 13](#_Toc170839599)

[Chapter 5 Interface Design 17](#_Toc170839600)

[5.1. LANGUAGE/FRAMEWORK: 17](#_Toc170839601)

[5.2. DATABASE CONNECTIVITY: 17](#_Toc170839602)

[5.3. STORED PROCEDURES AND FUNCTIONS: 17](#_Toc170839603)

[*5.4.* INTERFACES: 20](#_Toc170839604)

[Chapter 6 : CONCLUSION 23](#_Toc170839605)

[6.1. Lessons Learned 23](#_Toc170839606)

[6.2. Challenges and Solutions 23](#_Toc170839607)

[6.3. Future Work and Improvements 23](#_Toc170839608)

[6.4. Final Thoughts 24](#_Toc170839609)

[REFERENCES 24](#_Toc170839610)

# PROJECT OVERVIEW

## INTRODUCTION:

Welcome to the Restaurant Management System project! Our goal is to create a system that helps restaurants keep track of their food inventory, monitor expiration dates, and manage usage efficiently. This system will replace the old, time-consuming, and error-prone methods, reducing food waste and improving resource management for everyone from kitchen staff to managers and suppliers.

## PROBLEM STATEMENT:

Managing food items in a restaurant can be quite challenging. Currently, there's no single place to keep track of everything, making it difficult to know when food is about to spoil. Most tracking is done manually with paper files or separate spreadsheets, which can lead to mistakes and inefficiency. Our Restaurant Management System aims to solve this by offering a central database where all food-related information can be stored. This will help track available stock and automatically alert staff when items are about to expire. Additionally, it will generate detailed reports to keep everyone informed about the food inventory.

## PROJECT OBJECTIVES:

**Centralized Database Development:**

* **Objective:** Establish a centralized database to manage food inventory data, encompassing item specifics, quantities, and expiration dates.
* **Measurable:** Finalize the database setup by the end of the semester.
* **Achievable:** Within reach using the current resources available.
* **Relevance:** Directly fulfills the requirement for structured data storage.

**Reporting Capabilities:**

* **Objective:** Produce reports detailing food consumption, waste, and stock levels to support data-centric decision-making.
* **Measurable:** Create and validate reporting features by the semester's conclusion.
* **Achievable:** Aligned with the project's resource capability.
* **Relevance:** Enables informed decision-making processes.

**Enhancing Inventory Management Efficiency:**

* **Objective:** Improve overall inventory management workflows by introducing tools for monitoring, ordering, and restocking.
* **Measurable:** Demonstrate enhanced efficiency within the semester.
* **Achievable:** Realistic considering the project's circumstances.
* **Relevance:** Addresses identified issues concerning food management.

## DOCUMENT OBJECTIVES:

**Introduction:**

* **Purpose:** Provide an overview of the project and its importance.
* **Content:** Briefly describe the necessity for effective food inventory management and outline the system's objectives.

**Problem Statement:**

* **Purpose:** Clearly outline the challenges or issues targeted by the system.
* **Content:** Address the current issues concerning food inventory tracking, wastage, and data management.

**Project Objectives:**

* **Purpose:** Specify the system's objectives.
* **Content:**
  + **Centralized Database Creation:** Detail the objective of establishing a centralized database for food inventory data.
  + **Automated Expiration Date Tracking:** Specify the aim of efficient expiration date monitoring.
  + **Reporting Functionality:** Emphasize the importance of generating pertinent reports.
  + **Inventory Management Efficiency Improvement:** Discuss the goal of optimizing overall inventory management processes.

**Database Schema:**

* **Purpose:** Explain the database structure.
* **Content:** Provide information on the tables, fields, and relationships pertaining to food inventory data.

# DETAILED DATABASE DESIGN



## ENTITY:

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Entity Name** | **Description** |
| 01 | Customer | This entity represents the customer who places the order. |
| 02 | Order | An order represents a transaction made by a customer. |
| 03 | Payment | A payment records the transaction details of an order. |
| 04 | Menu | A menu item represents the food or drink options available. |
| 05 | MenuType | A menu type categorizes the menu items. |
| 06 | OrderDetail | Order detail captures specific items and quantities in an order. |
| 07 | Rating | A rating provides customer feedback on menu items. |

## DATA DICTIONARY:

**Customer:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | CustomerID | INT | PK | Unique identifier for each customer. |
| 02 | CustomerType | VARCHAR(20) |  | Type of customer (e.g., regular, VIP). |
| 03 | Email | VARCHAR(50) | UNIQUE | Email address of the customer. |
| 04 | Phone | VARCHAR(20) |  | Phone number of the customer. |
| 05 | Address | VARCHAR(100) |  | Address of the customer. |

**Orders:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | OrderID | INT | PK | Unique identifier for each order. |
| 02 | CustomerID | INT | FK | Reference to the customer who placed the order. |
| 03 | OrderDate | DATE |  | Date when the order was placed. |

**Payment:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | PaymentID | INT | PK | Unique identifier for each payment. |
| 02 | OrderID | INT | FK | Reference to the order for the payment. |
| 03 | PaymentAmount | DECIMAL(10,2) |  | Amount paid. |
| 04 | PaymentDate | DATE |  | Date of the payment. |
| 05 | PaymentMethod | VARCHAR(50) |  | Method used for the payment. |

**Menu:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | MenuItemID | INT | PK | Unique identifier for each menu item. |
| 02 | MenuName | VARCHAR(50) |  | Name of the menu item. |
| 03 | Price | DECIMAL(10,2) |  | Price of the menu item. |
| 04 | Description | VARCHAR(255) |  | Description of the menu item. |
| 05 | MenuTypeID | INT | FK | Reference to the type of menu item. |

**MenuType:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | MenuTypeID | INT | PK | Unique identifier for each menu type. |
| 02 | TypeName | VARCHAR(50) |  | Name of the menu type. |

**OrderDetail:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | OrderDetailID | INT | PK | Unique identifier for each order detail. |
| 02 | OrderID | INT | FK | Reference to the order. |
| 03 | MenuItemID | INT | FK | Reference to the menu item. |
| 04 | Quantity | INT |  | Quantity of the menu item ordered. |
| 05 | Price | DECIMAL(10,2) |  | Price of the ordered item. |
| 06 | SpecialInstructions | VARCHAR(255) |  | Any special instructions for the order. |

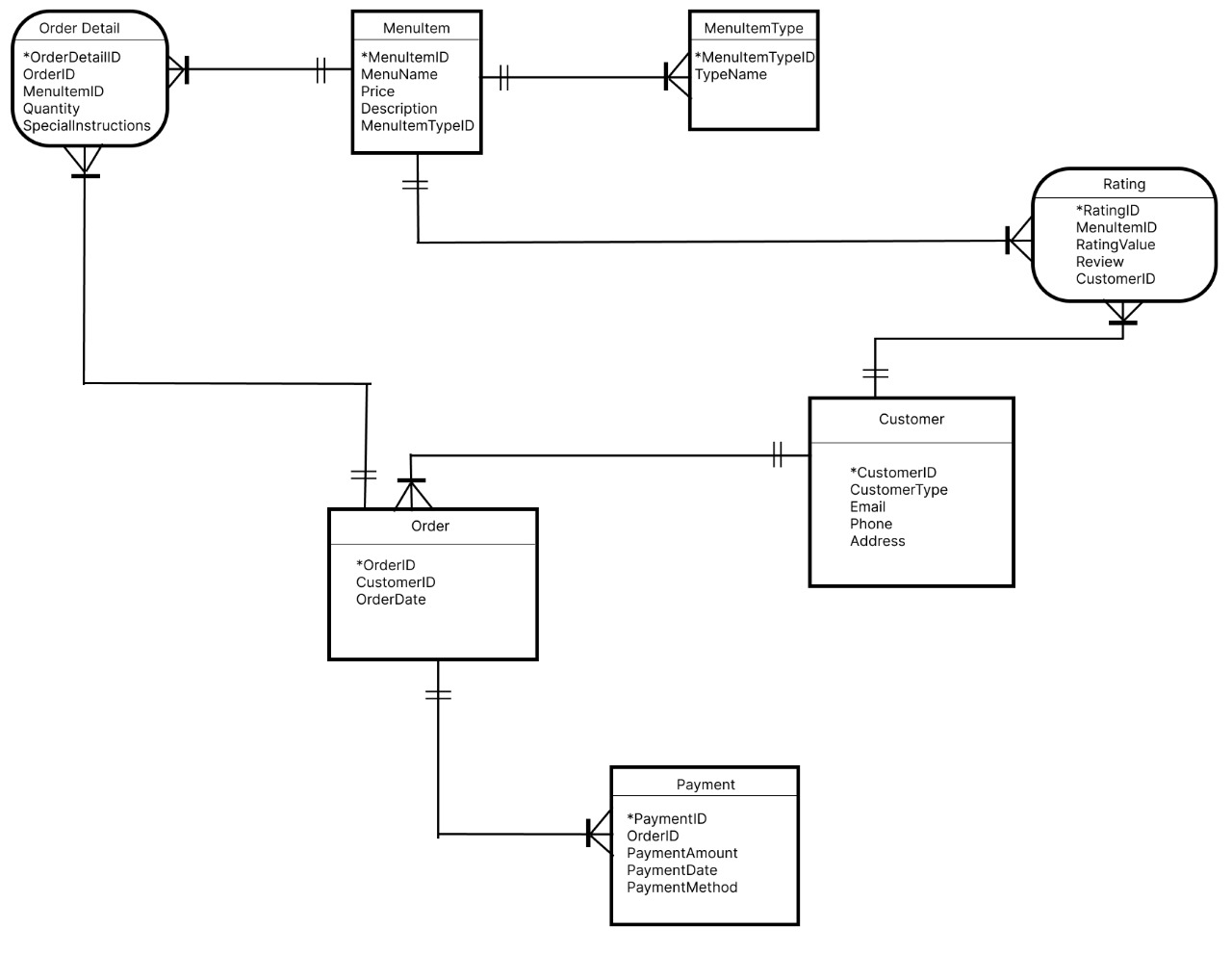
**Rating:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Name** | **Data Type** | **Constraint** | **Description** |
| 01 | RatingID | INT | PK | Unique identifier for each rating. |
| 02 | MenuItemID | INT | FK | Reference to the rated menu item. |
| 03 | RatingValue | INT |  | Rating value given by the customer. |
| 04 | Review | VARCHAR(255) |  | Customer review for the menu item. |
| 05 | CustomerID | INT | FK | Reference to the customer who gave the rating. |

## RELATIONSHIPS:

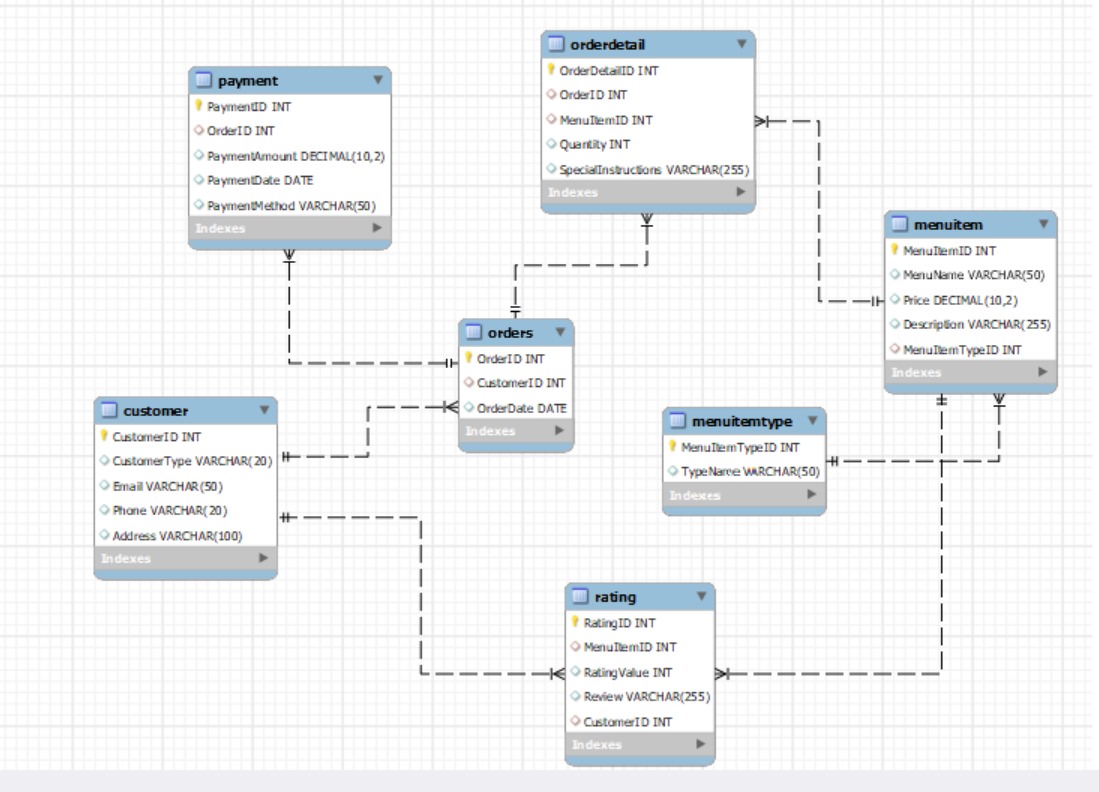
|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No** | **Participating Entities** | **Relation** | **Business Rule** |
| 01 | User, Order | User places Order | A user may place multiple orders. An order is placed by exactly one user. |
| 02 | Order, Payment | Order has Payment | An order may have multiple payments. A payment is for exactly one order. |
| 03 | Order, OrderDetail | Order has OrderDetail | An order has multiple order details. An order detail belongs to exactly one order. |
| 04 | Menu, OrderDetail | Menu is in OrderDetail | A menu item may appear in multiple order details. An order detail references exactly one menu item. |
| 05 | Menu, Rating | Menu receives Rating | A menu item may receive multiple ratings. A rating is for exactly one menu item. |
| 06 | User, Rating | User gives Rating | A user may give multiple ratings. A rating is given by exactly one user. |
| 07 | MenuType, Menu | MenuType categorizes Menu | A menu type may categorize multiple menu items. A menu item belongs to exactly one menu type. |

## ENTITY RELATIONSHIP DIAGRAM:



# : Logical DATABASE DESIGN

## RELATIONAL SCHEMA:



## FUNCTIONAL DEPENDENCIES:

**1. Customer table:**

- CustomerID → CustomerType, Email, Phone, Address

Example: If CustomerID is 1, it determines the CustomerType as 'Registered', Email as 'john@example.com', Phone as '123-456-7890', and Address as '123 Main St, Anytown'.

**2. MenuType table:**

- MenuTypeID → TypeName

Example: If MenuTypeID is 1, it determines the TypeName as 'Pizza'.

**3. Menu table:**

- MenuItemID → MenuName, Price, Description, MenuTypeID

Example: If MenuItemID is 301, it determines the MenuName as 'Margherita Pizza', Price as 12.99, Description as 'Classic pizza with tomatoes', and MenuTypeID as 1 (belonging to the 'Pizza' menu type).

**4. Orders table:**

- OrderID → CustomerID, OrderDate

Example: If OrderID is 101, it determines the CustomerID as 1 (the customer who placed the order) and the OrderDate as '2024-06-01'.

**5. OrderDetail table:**

- OrderDetailID → OrderID, MenuItemID, Quantity, Price, SpecialInstructions

Example: If OrderDetailID is 401, it determines the OrderID as 101 (the order it belongs to), MenuItemID as 301 (the specific menu item ordered), Quantity as 2, Price as 25.98 (the price of the menu item), and SpecialInstructions as 'No onions'.

**6. Payment table:**

- PaymentID → OrderID, PaymentAmount, PaymentDate, PaymentMethod

Example: If PaymentID is 201, it determines the OrderID as 101 (the order for which the payment was made), PaymentAmount as 50.00, PaymentDate as '2024-06-01', and PaymentMethod as 'Credit Card'.

**7. Rating table:**

- RatingID → MenuItemID, RatingValue, Review, CustomerID

Example: If RatingID is 501, it determines the MenuItemID as 301 (the menu item being rated), RatingValue as 5 (out of 5), Review as 'Excellent taste!', and CustomerID as 1 (the customer who provided the rating).

## NORMALIZATION:

Our ERD doesn’t contain any anomaly, so we only draw the 3NF Normalization.

A screenshot of a computer screen

Description automatically generated

# : Physical DATABASE DESIGN

## STRUCTURE OF THE TABLES:

DESCRIBE CUSTOMER;

DESCRIBE ORDERS;

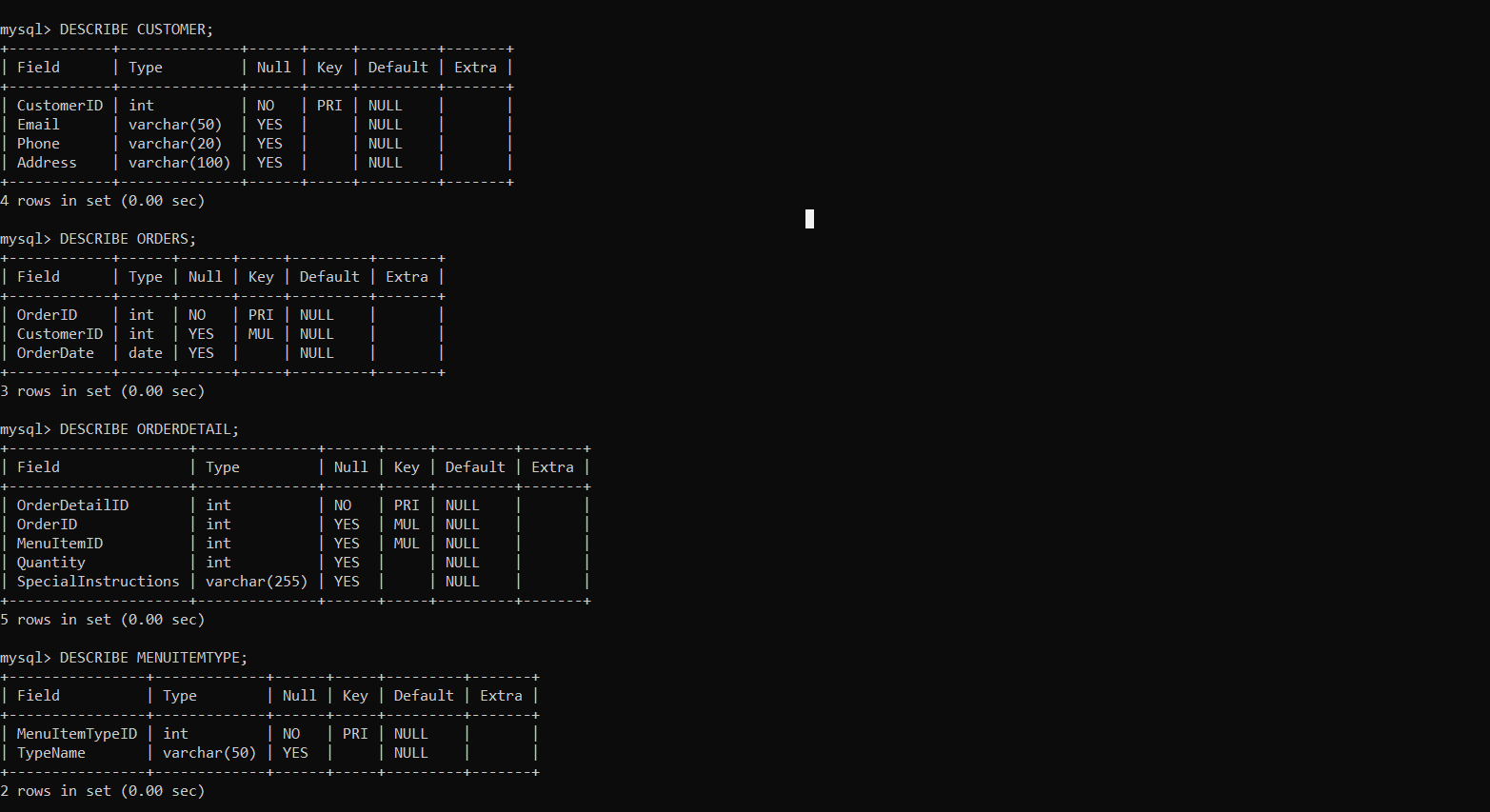
DESCRIBE ORDERDETAIL;

DESCRIBE MENUITEMTYPE;

DESCRIBE MENUITEM;

DESCRIBE PAYMENT;

DESCRIBE RATING;



A screenshot of a computer

Description automatically generated

## DATA SAMPLES INSIDE TABLES:

SELECT \* FROM Customer;

SELECT \* FROM Orders;

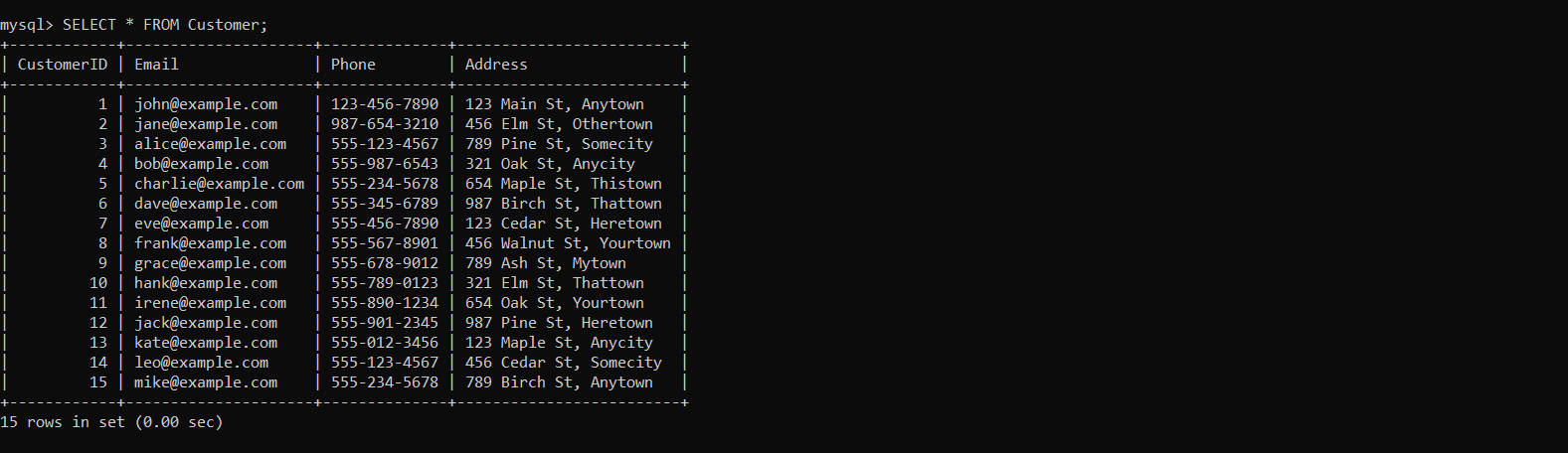
SELECT \* FROM OrderDetail;

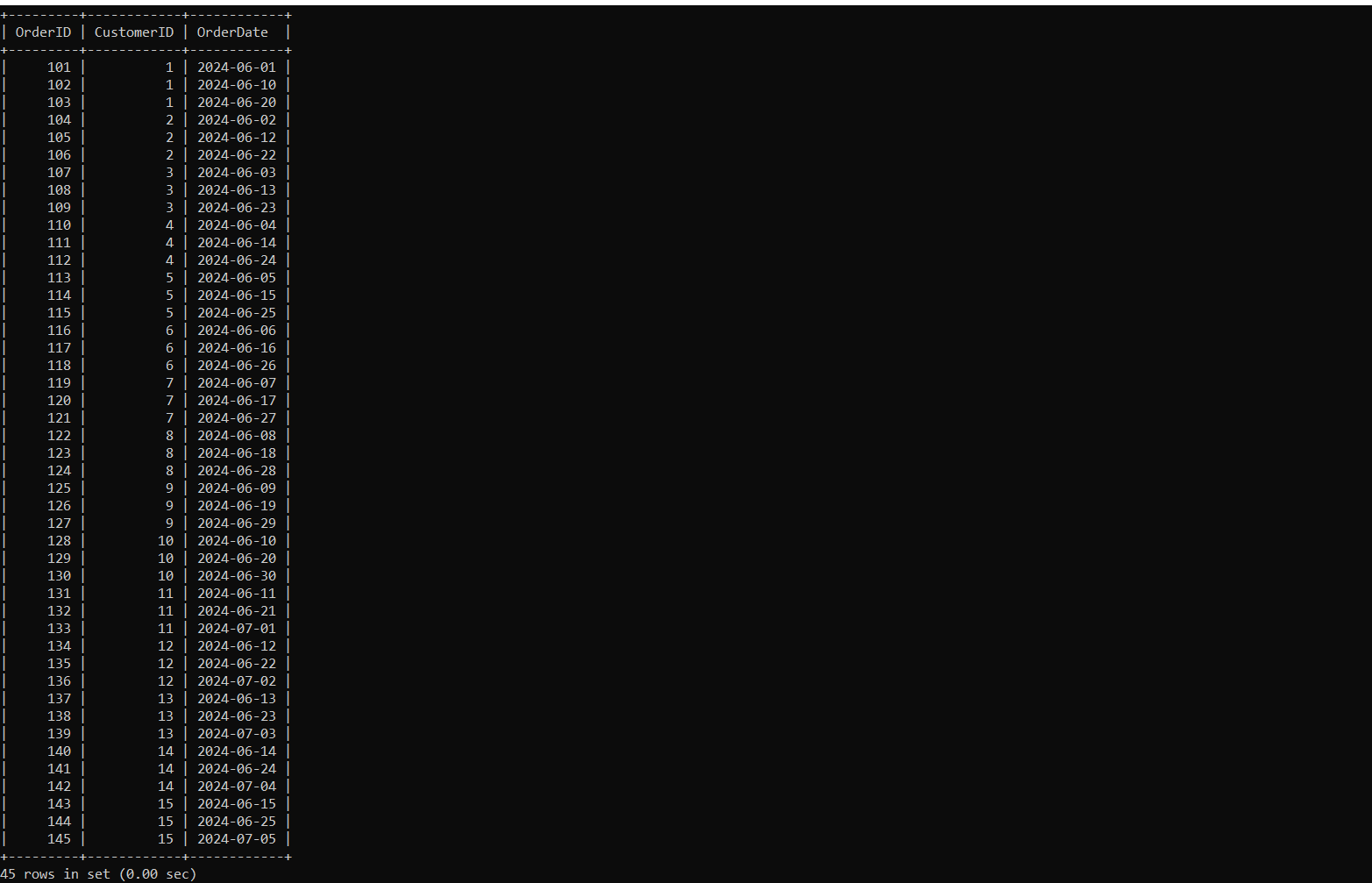
SELECT \* FROM MenuItemType;

SELECT \* FROM MenuItem;

SELECT \* FROM Payment;

SELECT \* FROM Rating;





A black background with a white light

Description automatically generated with medium confidenceA black screen with blue lines

Description automatically generated

A screen shot of a computer

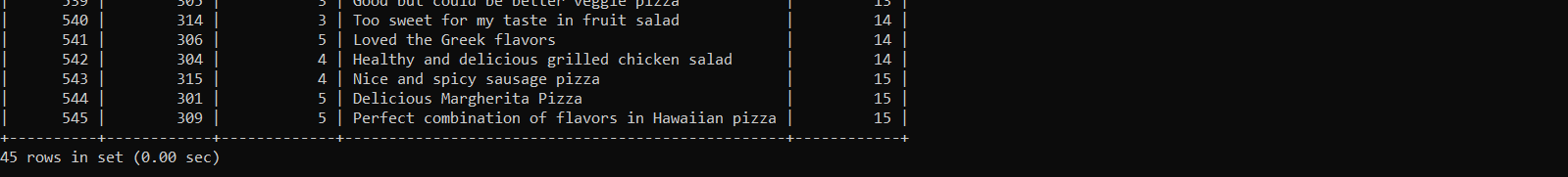
Description automatically generated

A black screen with a black background

Description automatically generatedA black screen with a black background

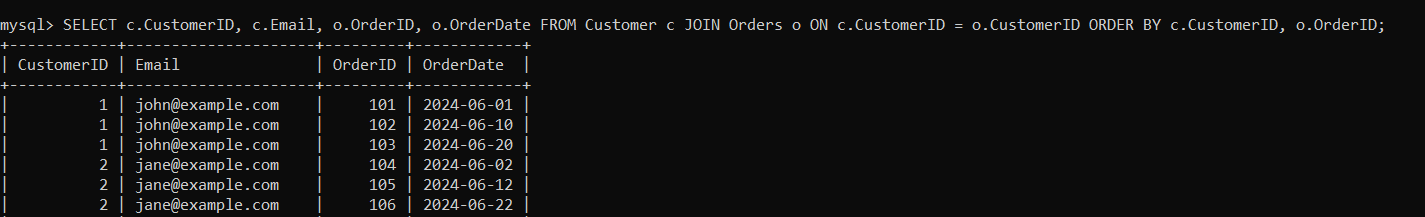
Description automatically generated

A screen shot of a computer

Description automatically generated

## QUERIES RESULTS:

* SELECT c.CustomerID, c.Email, o.OrderID, o.OrderDate FROM Customer c JOIN Orders o ON c.CustomerID = o.CustomerID ORDER BY c.CustomerID, o.OrderID;



A black screen with a black background

Description automatically generated

* SELECT c.CustomerID, c.Email, SUM(p.PaymentAmount) AS TotalAmountPaid FROM Customer c JOIN Orders o ON c.CustomerID = o.CustomerID JOIN Payment p ON o.OrderID = p.OrderID GROUP BY c.CustomerID;

A black screen with a black background

Description automatically generated

* SELECT PaymentMethod, SUM(PaymentAmount) AS TotalAmount FROM Payment GROUP BY PaymentMethod ORDER BY TotalAmount DESC;

A black screen with white text

Description automatically generated

* SELECT m.MenuItemID,m.MenuName,AVG(r.RatingValue) AS AverageRating FROM Menu m JOIN Rating r ON m.MenuItemID = r.MenuItemID GROUP BY m.MenuItemID, m.MenuName ORDER BY AverageRating DESC;

A computer screen shot of a black screen

Description automatically generated

* SELECT c.CustomerID, c.Email, r.MenuItemID, r.RatingValue, r.Review FROM Customer c JOIN Rating r ON c.CustomerID = r.CustomerID;

A screen shot of a computer program

Description automatically generated

A black screen with white text

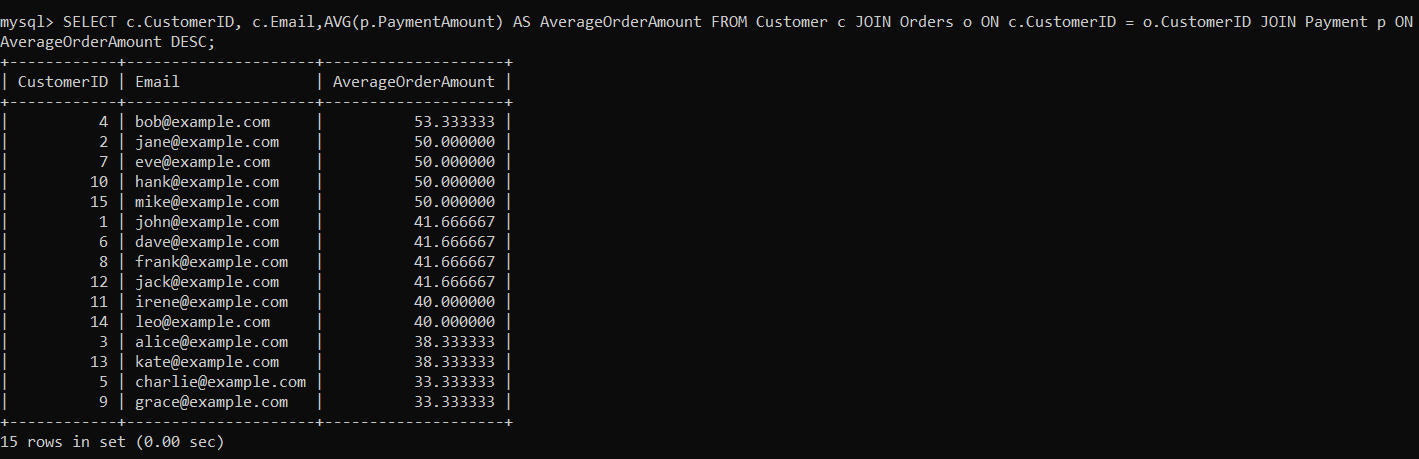
Description automatically generated

* SELECT m.MenuName, r.RatingValue, r.Review FROM Menu m JOIN Rating r ON m.MenuItemID = r.MenuItemID WHERE r.RatingValue >= 3;

A screen shot of a computer

Description automatically generated

* SELECT c.CustomerID,c.CustomerType,c.Email,AVG(p.PaymentAmount) AS AverageOrderAmount FROM Customer c JOIN Orders o ON c.CustomerID = o.CustomerID JOIN Payment p ON o.OrderID = p.OrderID GROUP BY c.CustomerID ORDER BY AverageOrderAmount DESC;



* SELECT MenuName, (SELECT AVG(RatingValue) FROM Rating WHERE MenuItemID = Menu.MenuItemID) AS AvgRating FROM Menu;

A computer screen with text on it

Description automatically generated

* SELECT o.OrderID, c.Email, od.MenuItemID, od.Quantity, od.Price FROM (SELECT OrderID, CustomerID FROM Orders) AS o JOIN Customer c ON o.CustomerID = c.CustomerID JOIN OrderDetail od ON o.OrderID = od.OrderID;

A black screen with a black background

Description automatically generated

A black screen with a black background

Description automatically generated

* SELECT p.PaymentID, p.OrderID, p.PaymentAmount, p.PaymentDate FROM (SELECT \* FROM Payment WHERE PaymentMethod = 'Credit Card') AS p;

A black screen with white lines

Description automatically generated

* SELECT c.CustomerID,c.Email,COUNT(o.OrderID) AS TotalOrders,SUM(od.Quantity) AS TotalItemsOrdered FROM Customer c LEFT JOIN Orders o ON c.CustomerID = o.CustomerID LEFT JOIN OrderDetail od ON o.OrderID = od.OrderID GROUP BY c.CustomerID, c.Email HAVING COUNT(o.OrderID) >= 1 AND SUM(od.Quantity) >= 1 ORDER BY TotalOrders DESC;

A computer screen with text on it

Description automatically generated

# Interface Design

## LANGUAGE/FRAMEWORK:

We chose Python for our GUI client because it's easy to understand and write, allowing for quick development. We used Tkinter, a simple yet powerful library for building GUIs in Python. By combining Python and Tkinter with Visual Studio, we created a robust environment that makes coding and debugging efficient. Tkinter supports various widgets and features like event handling and customization, making our GUI client versatile and powerful for different use cases.

## DATABASE CONNECTIVITY:

To connect our GUI client with the MySQL database, we used the MySQL-connector-python library. This library allows Python applications to interact easily with MySQL databases. After installing the library with “***pip install mysql-connector-python,”*** we used credentials like host, username, password, and database name to set up the connection. We wrapped our connection and database calls in a try-except-finally block to manage errors effectively and ensure that connections are properly closed to avoid resource leaks.

## STORED PROCEDURES AND FUNCTIONS:

Here's a list of stored procedures and their objectives for our Restaurant Management System:

**Customer Management System**

* **InsertCustomer**: Add new customer records to the database.
* **UpdateCustomer**: Modify existing customer records.
* **DeleteCustomer**: Remove customer records from the database.
* **SearchCustomersByEmail**: Search for customers based on their email.

**Order Management System**

* **InsertOrder**: Add new orders.
* **UpdateOrder**: Update existing orders.
* **DeleteOrder**: Delete orders.
* **SearchOrdersByCustomerID**: Search orders by CustomerID.

**Menu Item Management System**

* **InsertMenuItem**: Insert a new menu item into the MenuItem table.
* **UpdateMenuItem**: Update an existing menu item in the MenuItem table.
* **DeleteMenuItem**: Delete an existing menu item from the MenuItem table.
* **SearchMenuItemsByMenuItemTypeID**: Retrieve menu items based on a specific type ID.

**Menu Item Type Management System**

* **InsertMenuItemType**: Insert a new menu item type into the MenuItemType table.
* **UpdateMenuItemType**: Update an existing menu item type in the MenuItemType table.
* **DeleteMenuItemType**: Delete an existing menu item type from the MenuItemType table.
* **SearchMenuItemTypes**: Retrieve menu item types based on the type name.

**Order Detail Management System**

* **InsertOrderDetail**: Insert a new order detail into the OrderDetail table.
* **UpdateOrderDetail**: Update an existing order detail in the OrderDetail table.
* **DeleteOrderDetail**: Delete an existing order detail from the OrderDetail table.
* **SearchOrderDetailsByOrderID**: Retrieve order details based on a specific order ID.

**Payment Management System**

* **InsertPayment**: Insert a new payment into the Payment table.
* **UpdatePayment**: Update an existing payment in the Payment table.
* **DeletePayment**: Delete an existing payment from the Payment table.
* **SearchPaymentsByOrderID**: Retrieve payments based on a specific order ID.

**Rating Management System**

* **InsertRating**: Insert a new rating into the Rating table.
* **UpdateRating**: Update an existing rating in the Rating table.
* **DeleteRating**: Delete an existing rating from the Rating table.
* **SearchRatingsByMenuItemID**: Retrieve ratings based on a specific menu item ID.
* **SearchRatingsByCustomerID**: Retrieve ratings based on a specific customer ID.

**STORED PROCEDURES CODE**

DELIMITER //

CREATE PROCEDURE InsertCustomer(IN p\_email VARCHAR(50), IN p\_phone VARCHAR(20), IN p\_address VARCHAR(100)) BEGIN INSERT INTO Customer (Email, Phone, Address) VALUES (p\_email, p\_phone, p\_address); END //

CREATE PROCEDURE UpdateCustomer(IN p\_customer\_id INT, IN p\_email VARCHAR(50), IN p\_phone VARCHAR(20), IN p\_address VARCHAR(100)) BEGIN UPDATE Customer SET Email = p\_email, Phone = p\_phone, Address = p\_address WHERE CustomerID = p\_customer\_id; END //

CREATE PROCEDURE DeleteCustomer(IN p\_customer\_id INT) BEGIN DELETE FROM Customer WHERE CustomerID = p\_customer\_id; END //

CREATE PROCEDURE SearchCustomersByEmail(IN p\_email VARCHAR(50)) BEGIN SELECT \* FROM Customer WHERE Email LIKE CONCAT('%', p\_email, '%'); END //

CREATE PROCEDURE InsertOrder(IN p\_CustomerID INT, IN p\_OrderDate DATE) BEGIN INSERT INTO Orders (CustomerID, OrderDate) VALUES (p\_CustomerID, p\_OrderDate); END //

CREATE PROCEDURE UpdateOrder(IN p\_OrderID INT, IN p\_CustomerID INT, IN p\_OrderDate DATE) BEGIN UPDATE Orders SET CustomerID = p\_CustomerID, OrderDate = p\_OrderDate WHERE OrderID = p\_OrderID; END //

CREATE PROCEDURE DeleteOrder(IN p\_OrderID INT) BEGIN DELETE FROM Orders WHERE OrderID = p\_OrderID; END //

CREATE PROCEDURE SearchOrdersByCustomerID(IN p\_CustomerID INT) BEGIN SELECT \* FROM Orders WHERE CustomerID = p\_CustomerID; END //

CREATE PROCEDURE InsertMenuItemType(IN type\_name VARCHAR(50)) BEGIN INSERT INTO MenuItemType (TypeName) VALUES (type\_name); END //

CREATE PROCEDURE UpdateMenuItemType(IN type\_id INT, IN type\_name VARCHAR(50)) BEGIN UPDATE MenuItemType SET TypeName = type\_name WHERE MenuItemTypeID = type\_id; END //

CREATE PROCEDURE DeleteMenuItemType(IN type\_id INT) BEGIN DELETE FROM MenuItemType WHERE MenuItemTypeID = type\_id; END //

CREATE PROCEDURE SearchMenuItemTypes() BEGIN SELECT \* FROM MenuItemType; END //

CREATE PROCEDURE InsertMenuItem(IN menu\_name VARCHAR(50), IN price DECIMAL(10, 2), IN description VARCHAR(255), IN type\_id INT) BEGIN INSERT INTO MenuItem (MenuName, Price, Description, MenuItemTypeID) VALUES (menu\_name, price, description, type\_id); END //

CREATE PROCEDURE UpdateMenuItem(IN item\_id INT, IN menu\_name VARCHAR(50), IN price DECIMAL(10, 2), IN description VARCHAR(255), IN type\_id INT) BEGIN UPDATE MenuItem SET MenuName = menu\_name, Price = price, Description = description, MenuItemTypeID = type\_id WHERE MenuItemID = item\_id; END //

CREATE PROCEDURE DeleteMenuItem(IN item\_id INT) BEGIN DELETE FROM MenuItem WHERE MenuItemID = item\_id; END //

CREATE PROCEDURE SearchMenuItemsByMenuItemTypeID(IN type\_id INT) BEGIN SELECT \* FROM MenuItem WHERE MenuItemTypeID = type\_id; END //

CREATE PROCEDURE InsertOrderDetail(IN order\_id INT, IN item\_id INT, IN quantity INT, IN instructions VARCHAR(255)) BEGIN INSERT INTO OrderDetail (OrderID, MenuItemID, Quantity, SpecialInstructions) VALUES (order\_id, item\_id, quantity, instructions); END //

CREATE PROCEDURE UpdateOrderDetail(IN detail\_id INT, IN order\_id INT, IN item\_id INT, IN quantity INT, IN instructions VARCHAR(255)) BEGIN UPDATE OrderDetail SET OrderID = order\_id, MenuItemID = item\_id, Quantity = quantity, SpecialInstructions = instructions WHERE OrderDetailID = detail\_id; END //

CREATE PROCEDURE DeleteOrderDetail(IN detail\_id INT) BEGIN DELETE FROM OrderDetail WHERE OrderDetailID = detail\_id; END //

CREATE PROCEDURE SearchOrderDetailsByOrderID(IN order\_id INT) BEGIN SELECT \* FROM OrderDetail WHERE OrderID = order\_id; END //

CREATE PROCEDURE InsertPayment(IN order\_id INT, IN amount DECIMAL(10, 2), IN pay\_date DATE, IN method VARCHAR(50)) BEGIN INSERT INTO Payment (OrderID, PaymentAmount, PaymentDate, PaymentMethod) VALUES (order\_id, amount, pay\_date, method); END //

CREATE PROCEDURE UpdatePayment(IN payment\_id INT, IN order\_id INT, IN amount DECIMAL(10, 2), IN pay\_date DATE, IN method VARCHAR(50)) BEGIN UPDATE Payment SET PaymentAmount = amount, PaymentDate = pay\_date, PaymentMethod = method WHERE PaymentID = payment\_id AND orderID = order\_id; END //

CREATE PROCEDURE DeletePayment(IN payment\_id INT) BEGIN DELETE FROM Payment WHERE PaymentID = payment\_id; END //

CREATE PROCEDURE SearchPaymentsByOrderID(IN order\_id INT) BEGIN SELECT \* FROM Payment WHERE OrderID = order\_id; END //

CREATE PROCEDURE InsertRating(IN item\_id INT, IN rating\_value INT, IN review VARCHAR(255), IN cust\_id INT) BEGIN INSERT INTO Rating (MenuItemID, RatingValue, Review, CustomerID) VALUES (item\_id, rating\_value, review, cust\_id); END //

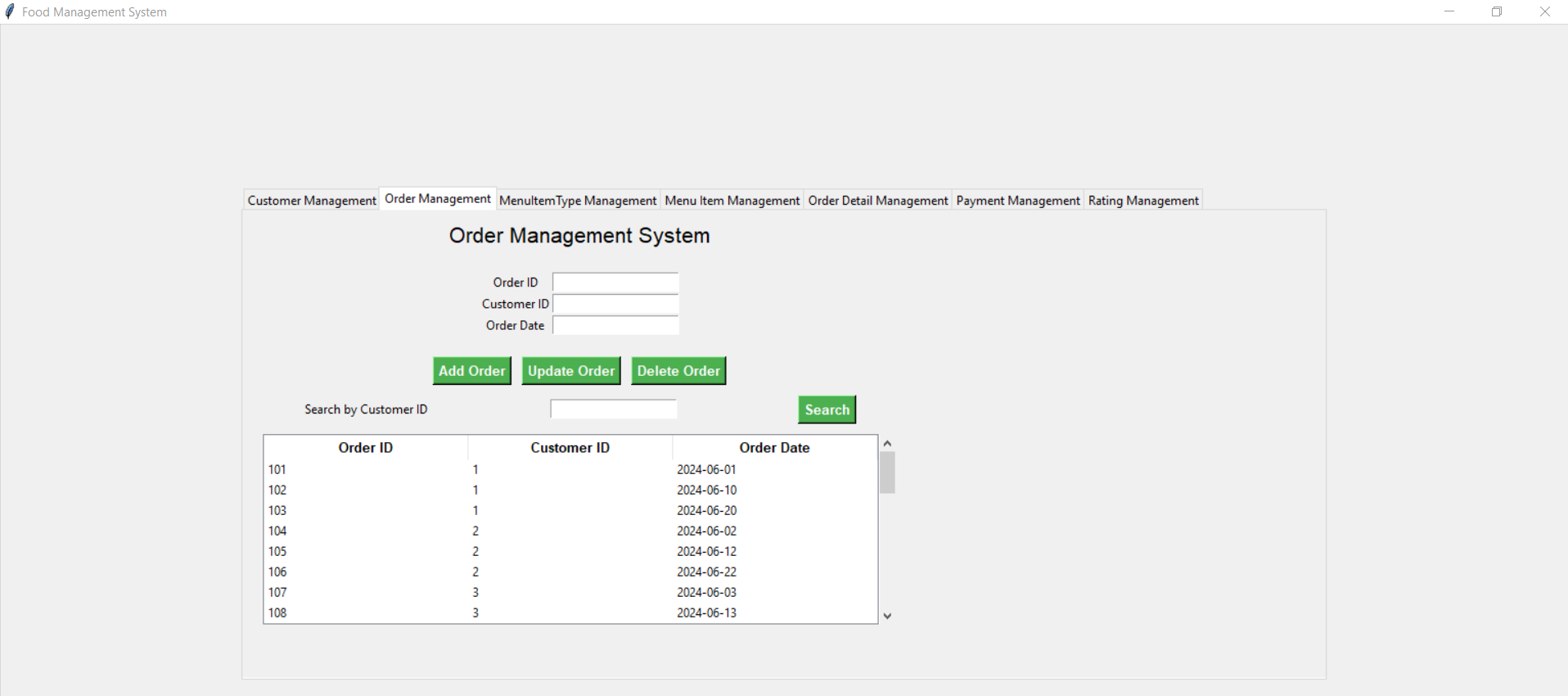
CREATE PROCEDURE UpdateRating(IN rating\_id INT, IN menu\_item\_id INT, IN rating\_value INT, IN review VARCHAR(255), IN customer\_id INT) BEGIN UPDATE Rating SET MenuItemID = menu\_item\_id, RatingValue = rating\_value, Review = review, CustomerID = customer\_id WHERE RatingID = rating\_id; END //

CREATE PROCEDURE DeleteRating(IN rating\_id INT) BEGIN DELETE FROM Rating WHERE RatingID = rating\_id; END //

CREATE PROCEDURE SearchRatingsByMenuItemID(IN item\_id INT) BEGIN SELECT \* FROM Rating WHERE MenuItemID = item\_id; END //

DELIMITER;

## INTERFACES:



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# : CONCLUSION

## 6.1. Lessons Learned

**Technical Skills:**

**Database Management:** Gained a solid understanding of MySQL, including stored procedures, database creation, data insertion, MySQL queries.

**UI Development:** Improved skills with Tkinter to build user-friendly Python interfaces.

**Error Handling:** Enhanced ability to manage exceptions in Python and MySQL, making the application more dependable.

**Project Management:**

**Time Management:** Learned to effectively divide time between development, testing, and documentation.

**Documentation:** Recognized the importance of keeping detailed and up-to-date documentation for smooth project flow and handover.

**Collaboration:**

**Teamwork:** Benefited from effective communication and teamwork, leading to better problem-solving and creativity.

**Feedback:** Understood the importance of incorporating feedback from team members and users to improve the project.

## 6.2. Challenges and Solutions

**Design Challenges:**

**Database Schema:** Ensuring a complete and normalized database schema was challenging, but iterative design and team reviews helped.

**Implementation Challenges:**

**Stored Procedures:** Overcoming the complexity of stored procedures was achieved through extensive study and testing.

**Testing Challenges:**

**Integration Testing:** Ensuring all components worked together smoothly required both automated scripts and thorough manual testing.

## 6.3. Future Work and Improvements

**Additional Features:**

User **Roles and Permissions:** Implement a user roles and permissions system to manage various levels of access (e.g., admin, manager, customer).

Order **History:** Maintain a detailed order history for customers, allowing them to view past orders and reorder easily.

**Optimizations:**

**Performance Tuning:** Improve database query and stored procedure performance for large datasets.

**UI Enhancements:** Make the GUI more responsive and user-friendly, using modern frameworks like VS Code with Tkinter.

**Broader Applications:**

**Mobile App:** Create a mobile version for managing ratings on the go.

**Data Analytics:** Integrate analytics to provide insights and trends, aiding food management decisions.

## 6.4. Final Thoughts

**Personal Insights:**

**Growth:** This project was a major learning experience in both technical skills and project management.

**Real-World Impact:** Building an application that addresses a real need was incredibly rewarding.

**Overall Impact:**

**User Impact:** The system will make managing ratings and reviews more efficient, boosting customer satisfaction.

**Professional Development:** This project has significantly contributed to our professional growth, preparing us for future challenges.

**Acknowledgments:**

* **Team Members:** Abubakar, Appi Zunaira Akbar, Danish Abdullah Khan
* **Instructor:** Mam Asiya Batool
* **Special Thanks:** ChatGPT and Google for their help in creating Python GUIs.

# REFERENCES

**MySQL Documentation:**[*https://dev.mysql.com/doc/*](https://dev.mysql.com/doc/)

**Tkinter Documentation:** [*https://docs.python.org/3/library/tkinter.html*](https://docs.python.org/3/library/tkinter.html)