NAAN MUDHALVAN HACKATHON SET 1& 2 ANSWERS

HACKATHON SET - 1

Use Case Title: Library Management System (LMS)

Student Name: Sandhiya sri S

Register Number: C2S22885

Institution: NPR Arts & Science College,Natham

Department: BSc Computer Science

Date of Submission: 18.03.2025

1. Problem Statement

Libraries face challenges in managing their operations, including tracking book loans, monitoring overdue items, and maintaining an organized database of available books. The Library Management System (LMS) aims to create a robust database system that simplifies these processes, ensuring that library staff can efficiently manage book lending and provide a better experience for users.

2. Proposed Solution

The proposed solution is a SQL-based Library Management System that includes the following features:

- Book Management: Staff can add new books with details such as title, author, genre, and availability status.
- Loan Tracking: The system tracks book loans, including issue and return dates, allowing for efficient monitoring of borrowed books.
- Overdue Monitoring: Automated notifications for overdue books and the capability to generate reports on late returns.
- User Management: Users (students) can easily borrow and return books, with their borrowing history accessible for review.

The system will utilize a well-defined database schema to ensure data integrity and optimize query performance.

3. Technologies & Tools Considered

- Database Management System: MySQL or Oracle Database
- Programming Language: SQL for database operations

- **Development Tools: MySQL Workbench or Oracle SQL Developer for database design and management**
- **Version Control: GitHub for repository management**

4. Database Schema & Data Flow

Database Schema

The database schema includes the following key tables:

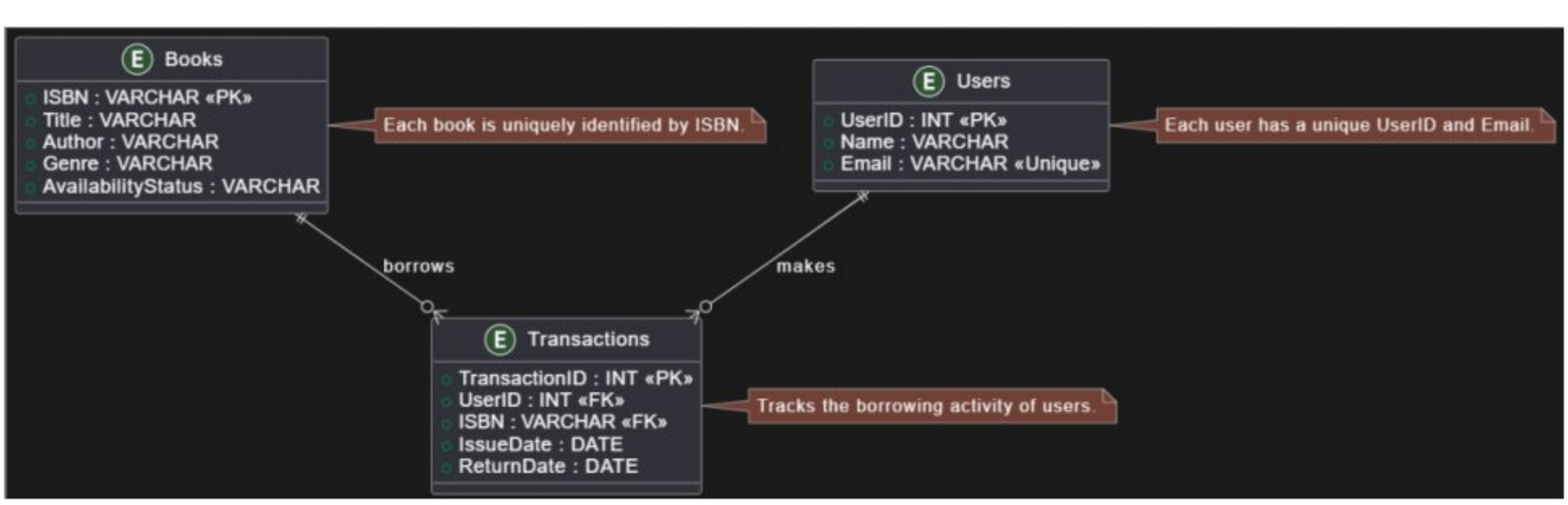
- **Books Table:**
  - **ISBN (VARCHAR, Primary Key)**
  - **Title (VARCHAR)**
  - **Author (VARCHAR)**
  - **Genre (VARCHAR)**
  - **AvailabilityStatus (VARCHAR)**
- **Users Table:**
  - **UserID (INT, Primary Key)**
  - **Name (VARCHAR)**
  - **Email (VARCHAR, Unique)**
- **Transactions Table:**
  - **TransactionID (INT, Primary Key)**
  - **UserID (INT, Foreign Key)**
  - **ISBN (VARCHAR, Foreign Key)**
  - **IssueDate (DATE)**
  - **ReturnDate (DATE)**

Data Flow

1. **When a user borrows a book, a new transaction is created in the Transactions table.**
2. **The availability status of the book is updated in the Books table.**

3. **Queries can be used to fetch overdue books by comparing the current date with the return date.**
4. **Reports can be generated to show user borrowing history and overdue items.**

Entity-Relationship Diagram (ERD)



5. Feasibility & Challenges

- **Feasibility: The proposed LMS is practical due to its straightforward design and the availability of SQL databases. Implementation is feasible within a short timeframe, given the well-defined scope and existing technologies.**
- **Challenges: Potential challenges include ensuring data integrity during concurrent transactions and managing user access levels. These can be addressed by implementing proper locking mechanisms and user authentication protocols.**

6. Expected Outcome & Impact

The expected outcome is a fully functional Library Management System that streamlines library operations. Benefits include:

- **Improved efficiency in managing book lending and returns.**
- **Reduced instances of overdue books through automated tracking.**
- **Enhanced user experience for students borrowing and returning books.**

7. Future Enhancements

Future enhancements could include:

- **A web interface for easier access to the LMS.**

- Integration with mobile applications for notifications and book searches.
- Advanced reporting features for library staff to analyze borrowing trends.

8. SQL Scripts

Table Creation Scripts

sql

```
CREATE TABLE Books (
    ISBN VARCHAR(13) PRIMARY KEY,
    Title VARCHAR(255),
    Author VARCHAR(255),
    Genre VARCHAR(100),
    AvailabilityStatus VARCHAR(20)
);

CREATE TABLE Users (
    UserID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(255),
    Email VARCHAR(255) UNIQUE
);

CREATE TABLE Transactions (
    TransactionID INT PRIMARY KEY AUTO_INCREMENT,
    UserID INT,
    ISBN VARCHAR(13),
    IssueDate DATE,
    ReturnDate DATE,
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
```

FOREIGN KEY (ISBN) REFERENCES Books(ISBN)

);

Sample Data Insertion

sql

INSERT INTO Books (ISBN, Title, Author, Genre, AvailabilityStatus) VALUES

('978-3-16-148410-0', 'The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction', 'Available'),

('978-1-56619-909-4', '1984', 'George Orwell', 'Dystopian', 'Available');

INSERT INTO Users (Name, Email) VALUES

(Pranitha R', 'pranithacs90393@gmail.com'),

('Jane Smith', thakshithapandian3@gmail.com');

Example Queries and Outputs

- Fetch all available books:

sql

SELECT * FROM Books WHERE AvailabilityStatus = 'Available';

Example Output:

ISB	Title	Author	Genre	Availability Status
978-3-16-148410-0	The Great Gatsby	F. Scott Fitzgerald	Fiction	Available
978-1-56619-909-4	198	George Orwell	Dystopian	Available

- Track overdue books:

sql

SELECT * FROM Transactions WHERE ReturnDate < CURDATE();

Example Output:

TransactionID	UserID	ISBN	IssueDate	ReturnDate
1	1	978-3-16-148410-0	2025-03-01	2025-03-10
2	2	978-1-56619-909-4	2025-03-05	2025-03-15

9. GitHub Repository Structure

- /sql_scripts: Contains all SQL scripts for table creation and data insertion.
- /documentation: Contains this document and any additional documentation.
- README.md: Overview of the project, setup instructions, and usage examples.
- GitHub link:
- https://github.com/Sandhiyasri2004/Sandhiyasri

HACKATHON SET - 2

Use Case Title: Online Food Ordering System Project Document

Student Name: Sandhiya sri S

Register Number: C2S22885

Institution: NPR Arts & Science College,Natham

Department: BSc Computer Science

Date of Submission: 18.03.2025

1. Problem Statement

The food industry is experiencing a surge in demand for online food ordering and delivery services. Customers seek a convenient and efficient way to order meals from their favorite restaurants, while restaurants aim to expand their reach and streamline their order management processes. The Online Food Ordering System

(OFOS) aims to create a comprehensive platform that bridges the gap between customers and restaurants, providing a seamless ordering experience.

2. Proposed Solution

The proposed solution is a SQL-based Online Food Ordering System that includes the following features:

- Restaurant Management: Restaurants can register and manage their menu items, including details such as dish name, description, price, and availability.
- Customer Management: Customers can create accounts, browse menus, place orders, and track their order status.
- Order Tracking: The system tracks orders, including order details, delivery status, and customer information.
- Payment Integration: Secure payment processing for customers to complete their orders.
- Reporting and Analytics: Comprehensive reporting and analytics for restaurants to monitor sales, popular dishes, and customer trends.

The system will utilize a well-defined database schema to ensure data integrity and optimize query performance.

3. Technologies & Tools Considered

- Database Management System: MySQL or PostgreSQL
- Programming Language: SQL for database operations, with potential integration of a backend language (e.g., Python, Java, or Node.js) for application logic
- Development Tools: MySQL Workbench, PostgreSQL, or integrated development environments (IDEs) for database design and management
- Web Framework: React, Angular, or Vue.js for building the user interface
- Payment Gateway: Stripe, PayPal, or other secure payment processing services
- Version Control: GitHub for repository management

4. Database Schema & Data Flow

Database Schema

The database schema includes the following key tables:

Restaurants Table:

- **RestaurantID (INT, Primary Key)**
- **Name (VARCHAR)**
- **Description (TEXT)**
- **Address (VARCHAR)**
- **ContactInfo (VARCHAR)**

Menu Items Table:

- **ItemID (INT, Primary Key)**
- **RestaurantID (INT, Foreign Key)**
- **Name (VARCHAR)**
- **Description (TEXT)**
- **Price (DECIMAL)**
- **Availability (BOOLEAN)**

Orders Table:

- **OrderID (INT, Primary Key)**
- **CustomerID (INT, Foreign Key)**
- **RestaurantID (INT, Foreign Key)**
- **ItemID (INT, Foreign Key)**
- **OrderDate (DATETIME)**
- **DeliveryStatus (VARCHAR)**
- **TotalAmount (DECIMAL)**

Customers Table:

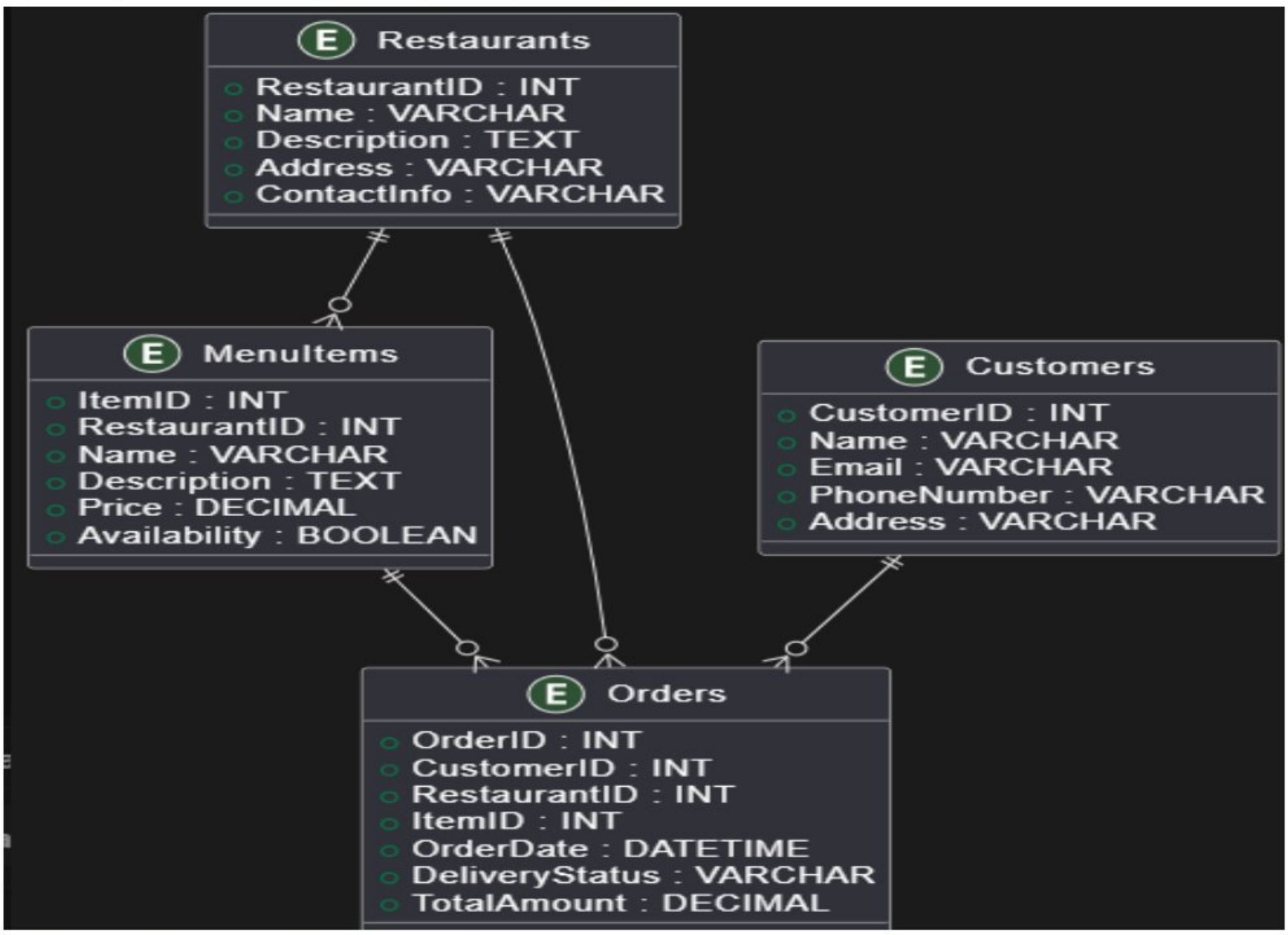
- **CustomerID (INT, Primary Key)**
- **Name (VARCHAR)**

- **Email (VARCHAR, Unique)**
- **PhoneNumber (VARCHAR)**
- **Address (VARCHAR)**

Data Flow

1. **Customers browse the available restaurants and their menus.**
2. **Customers select items and place an order.**
3. **The order is recorded in the Orders table, and the restaurant's inventory is updated accordingly.**
4. **Customers can track the status of their order.**
5. **Restaurants can generate reports and analytics based on order data.**

Entity-Relationship Diagram (ERD)



5. Feasibility & Challenges

Feasibility

The proposed Online Food Ordering System is feasible due to the availability of SQL databases and web development technologies. The well-defined scope and the integration of common features in the food delivery industry make the implementation achievable within a reasonable timeframe.

Challenges

Potential challenges include ensuring secure payment processing, handling high-volume transactions, and providing a user-friendly interface for both customers and restaurants. These can be addressed by leveraging established payment gateways, implementing scalable database and application architectures, and focusing on user experience design.

6. Expected Outcome & Impact

The expected outcome is a fully functional Online Food Ordering System that streamlines the food ordering and delivery process. Benefits include:

- **Improved customer experience with a convenient and user-friendly ordering platform.**
- **Increased sales and reach for participating restaurants.**
- **Efficient order management and tracking for restaurants.**
- **Comprehensive reporting and analytics to support business decision-making.**

7. Future Enhancements

Future enhancements could include:

- **Integration with restaurant management systems for seamless inventory and order synchronization.**
- **Mobile application development for a more accessible and responsive user experience.**
- **Personalized recommendations and loyalty programs to enhance customer engagement.**
- **Integration with delivery services for automated order fulfillment.**
- **Advanced analytics and forecasting capabilities to support strategic planning.**

8. SQL Scripts

The following SQL script creates the database schema:

sql

```
-- Create Customers Table
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(255),
    Email VARCHAR(255) UNIQUE,
    Phone VARCHAR(20),
    Address VARCHAR(255)
);

-- Create Menu Table
CREATE TABLE Menu (
    MenuID INT PRIMARY KEY AUTO_INCREMENT,
    ItemName VARCHAR(255),
    Category VARCHAR(100),
    Price DECIMAL(10, 2),
    Description TEXT,
    Available BOOLEAN DEFAULT TRUE
);

-- Create Orders Table
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY AUTO_INCREMENT,
    CustomerID INT,
```

```
    OrderDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    Status VARCHAR(20),
    TotalAmount DECIMAL(10, 2),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

-- Create OrderDetails Table
CREATE TABLE OrderDetails (
    OrderDetailID INT PRIMARY KEY AUTO_INCREMENT,
    OrderID INT,
    MenuID INT,
    Quantity INT,
    Subtotal DECIMAL(10, 2),
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
    FOREIGN KEY (MenuID) REFERENCES Menu(MenuID)
);

-- Create Payments Table
CREATE TABLE Payments (
    PaymentID INT PRIMARY KEY AUTO_INCREMENT,
    OrderID INT,
    PaymentDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    PaymentMethod VARCHAR(50),
    Amount DECIMAL(10, 2),
    Status VARCHAR(20),
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
```

```
);

-- Create Inventory Table
CREATE TABLE Inventory (
    IngredientID INT PRIMARY KEY AUTO_INCREMENT,
    IngredientName VARCHAR(255),
    QuantityAvailable INT,
    UnitPrice DECIMAL(10, 2)
);

-- Create InventoryTransactions Table
CREATE TABLE InventoryTransactions (
    TransactionID INT PRIMARY KEY AUTO_INCREMENT,
    IngredientID INT,
    OrderID INT,
    QuantityUsed INT,
    TransactionDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (IngredientID) REFERENCES Inventory(IngredientID),
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
);
```

5. Example SQL Queries

Here are some example SQL queries for managing the food ordering system:

5.1 Fetch All Orders for a Customer

sql

```
SELECT o.OrderID, o.OrderDate, o.Status, o.TotalAmount
FROM Orders o
```

JOIN Customers c ON o.CustomerID = c.CustomerID

WHERE c.Email = 'customer@example.com';

Console Output:

```
OrderID | OrderDate           | Status    | TotalAmount
-------------------------------------------------------
1       | 2025-03-17 12:00:00 | Completed| 25.50
2       | 2025-03-18 13:00:00 | Pending   | 15.75
```

5.2 Track Pending Deliveries

sql

SELECT * FROM Orders WHERE Status = 'Pending';

Console Output:

```
OrderID | CustomerID | OrderDate           | Status    | TotalAmount
-----------------------------------------------------------------
2       | 3          | 2025-03-18 13:00:00 | Pending   | 15.75
```

5.3 Get Sales Trends

sql

SELECT OrderDate, SUM(TotalAmount) AS TotalSales

FROM Orders

GROUP BY OrderDate

ORDER BY OrderDate;

Console Output:

```
OrderDate           | TotalSales
-------------------------------
2025-03-17 00:00:00 | 25.50
2025-03-18 00:00:00 | 15.75
```

5.4 Calculate Total Order Amount with Tax and Discounts

sql

```
DELIMITER //

CREATE PROCEDURE CalculateTotalOrderAmount(IN order_id INT, OUT
total_amount DECIMAL(10, 2))

BEGIN

    DECLARE subtotal DECIMAL(10, 2);

    DECLARE tax_rate DECIMAL(5, 2) DEFAULT 0.08; -- 8% tax

    DECLARE discount DECIMAL(10, 2) DEFAULT 0.00; -- No discount


    SELECT SUM(Subtotal) INTO subtotal FROM OrderDetails WHERE OrderID =
order_id;

    SET total_amount = subtotal + (subtotal * tax_rate) - discount;

END //

DELIMITER ;
```

Console Usage:

sql

Copy

```
CALL CalculateTotalOrderAmount(1, @total);

SELECT @total;
```

Console Output:

```
@total
-----------
27.54
```

Expected Outputs

1. Well-Structured Database Schema

- **The schema includes tables for Customers, Orders, Menu, Payments, and Inventory, with appropriate relationships and constraints.**

2. Efficient SQL Queries

- **Queries provided allow for effective retrieval and management of data related to orders, customers, and inventory.**

3. Comprehensive Documentation

- **The document includes a detailed explanation of the database structure, SQL scripts, and usage instructions.**

4. GitHub Repository

- **While the document does not create a GitHub repository, it provides instructions for setting one up with the SQL scripts included.**

GitHub link:

https://github.com/Sandhiyasri2004/Sandhiyasri