**Case Study**

**SQL Database Management System for Restaurant Management System**

**Overview**

A restaurant management system (RMS) is a software solution that simplifies and automates various restaurant operations such as order management, inventory control, staff management, customer relationship management, and financial transactions. Implementing an SQL database management system (DBMS) for an RMS enhances data organization, accessibility, and overall efficiency.

**Objectives**

1. **Efficient Order Management:** Streamline the process of taking and processing orders.
2. **Inventory Control:** Track stock levels and automate restocking processes.
3. **Staff Management:** Maintain staff records, schedules, and payroll information.
4. **Customer Relationship Management:** Manage customer data and loyalty programs.
5. **Financial Management:** Record and analyze financial transactions, including sales and expenses.

**Database Design**

The SQL database for the RMS will consist of multiple interrelated tables. Here are the primary tables and their relationships:

1. **Customers Table:**
   * + customer\_id (Primary Key)
     + name
     + contact\_number
     + email
     + address
2. **Staff Table:**
   * + staff\_id (Primary Key)
     + name
     + role
     + contact\_number
     + email
     + salary
     + schedule
3. **Menu Table:**
   * + menu\_id (Primary Key)
     + item\_name
     + category
     + price
     + description
4. **Orders Table:**
   * + order\_id (Primary Key)
     + customer\_id (Foreign Key)
     + staff\_id (Foreign Key)
     + order\_date
     + total\_amount
5. **Order\_Details Table:**
   * + order\_detail\_id (Primary Key)
     + order\_id (Foreign Key)
     + menu\_id (Foreign Key)
     + quantity
     + price
6. **Inventory Table:**
   * + inventory\_id (Primary Key)
     + item\_name
     + category
     + quantity
     + supplier
     + last\_restocked
7. **Transactions Table:**
   * + transaction\_id (Primary Key)
     + order\_id (Foreign Key)
     + transaction\_date
     + amount
     + payment\_method

**Database Schema**

CREATE TABLE Customers (

customer\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR (100),

contact\_number VARCHAR (15),

email VARCHAR (100),

address TEXT

);

CREATE TABLE Staff (

staff\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR (100),

role VARCHAR (50),

contact\_number VARCHAR (15),

email VARCHAR (100),

salary DECIMAL (10, 2)

);

CREATE TABLE Menu (

menu\_id INT PRIMARY KEY AUTO\_INCREMENT,

item\_name VARCHAR (100),

category VARCHAR (50),

price DECIMAL (5, 2));

CREATE TABLE Orders (

order\_id INT PRIMARY KEY AUTO\_INCREMENT,

customer\_id INT,

staff\_id INT,

order\_date DATETIME,

total\_amount DECIMAL (10, 2),

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id),

FOREIGN KEY (staff\_id) REFERENCES Staff(staff\_id)

);

CREATE TABLE Order\_Details (

order\_detail\_id INT PRIMARY KEY AUTO\_INCREMENT,

order\_id INT,

menu\_id INT,

quantity INT,

price DECIMAL (5, 2),

FOREIGN KEY (order\_id) REFERENCES Orders(order\_id),

FOREIGN KEY (menu\_id) REFERENCES Menu(menu\_id)

);

CREATE TABLE Inventory (

inventory\_id INT PRIMARY KEY AUTO\_INCREMENT,

item name VARCHAR (100),

category VARCHAR (50),

quantity INT,

supplier VARCHAR (100),

last\_restocked DATE

);

CREATE TABLE Transactions (

transaction\_id INT PRIMARY KEY AUTO\_INCREMENT,

order\_id INT,

transaction\_date DATETIME,

amount DECIMAL (10, 2),

payment\_method VARCHAR (50),

FOREIGN KEY (order\_id) REFERENCES Orders(order\_id)

);

**Use Cases**

1. **Order Placement:**
   * When a customer places an order, entries are made in the Orders and Order\_Details tables.
   * The total\_amount is calculated and updated in the Orders table.
   * Relevant inventory\_id entries are updated to reflect the reduced stock.
2. **Inventory Management:**
   * Track inventory levels with the Inventory table.
   * Automate notifications for low stock items and schedule restocking.
   * Maintain records of suppliers and last restocking dates.
3. **Staff Management:**
   * Maintain staff schedules, roles, and payroll information in the Staff table.
   * Track performance and assign tasks based on role and availability.
4. **Customer Management:**
   * Store customer information and loyalty points in the Customers table.
   * Implement loyalty programs and personalized marketing based on customer data.
5. **Financial Management:**
   * Record financial transactions in the Transactions table.
   * Generate reports for sales, expenses, and profitability analysis.

**Conclusion**

Implementing an SQL database management system for a restaurant management system significantly enhances the efficiency and accuracy of restaurant operations. By leveraging the power of SQL for data management, restaurants can streamline their processes, provide better customer service, and make data-driven decisions to boost profitability and growth.

**ER Diagram Details**

**Entities and Attributes:**

* + **Customers**
    - customer\_id (PK)
    - name
    - contact\_number
    - email
    - address
  + **Staff**
    - staff\_id (PK)
    - name
    - role
    - contact\_number
    - email
    - salary
    - schedule
  + **Menu**
    - menu\_id (PK)
    - item\_name
    - category
    - price
    - description
  + **Orders**
    - order\_id (PK)
    - customer\_id (FK)
    - staff\_id (FK)
    - order\_date
    - total\_amount
  + **Order\_Details**
    - order\_detail\_id (PK)
    - order\_id (FK)
    - menu\_id (FK)
    - quantity
    - price
  + **Inventory**
    - inventory\_id (PK)
    - item\_name
    - category
    - quantity
    - supplier
    - last\_restocked
  + **Transactions**
    - transaction\_id (PK)
    - order\_id (FK)
    - transaction\_date
    - amount
    - payment\_method

1. **Relationships:**
   * **Customers** to **Orders**: One-to-Many (One customer can place multiple orders)
   * **Staff** to **Orders**: One-to-Many (One staff member can handle multiple orders)
   * **Orders** to **Order\_Details**: One-to-Many (One order can have multiple order details)
   * **Menu** to **Order\_Details**: One-to-Many (One menu item can be in multiple order details)
   * **Orders** to **Transactions**: One-to-One (One order is associated with one transaction)