Pharmaceutical Inventory and Sales Management System

Team Members:

1. S M THANUSHA
SRN: PES1UG23CS493

2. RISHIKA
SRN: PES1UG23CS479

Problem Statement :

To design and implement a database-driven Pharmacy Management System that efficiently manages medicine inventory, customer records, employee details, purchases, and sales billing operations in a secure and organized manner.

Description of the Statement:

This project presents the design and development of a Pharmacy Management System aimed at automating and streamlining daily pharmacy operations. The system manages key functions such as medicine inventory tracking, supplier and purchase records, customer details, employee management, and sales billing. Using a structured relational database, the application ensures accurate stock updates, reliable transaction handling, and efficient data retrieval. The solution enhances operational efficiency, reduces manual errors, and provides a centralized platform for managing pharmacy workflows.

User Requirement Specification :

The system must allow users to manage pharmacy operations including medicine inventory, supplier records, customer details, employee accounts, purchases, and sales billing. It should automatically update stock levels, calculate totals using triggers, ensure data accuracy through constraints, and provide a simple, secure, and reliable interface for daily pharmacy management tasks.

List of Softwares/Tools/Programming Languages Used

The project uses XAMPP (Apache & MySQL), phpMyAdmin, VS Code, GitHub, PHP, HTML, CSS, JavaScript, Bootstrap, and SQL queries including triggers, functions, joins, and aggregates for implementing the Pharmacy Management System.

System Functionalities :

● System Functionality 1: Medicine Inventory Management

Allows the admin to add, update, search, and delete medicines while maintaining accurate real-time stock levels.

● System Functionality 2: Sales & Billing Management

Enables creating new sales, adding items to a bill, calculating totals automatically using triggers, and generating the final invoice.

● System Functionality 3: Purchase & Supplier Management

Records supplier details and purchase entries, updates stock based on purchases, and maintains purchase history for auditing.

● System Functionality 4: Customer Management

Stores customer information and links customers to their sales for record-keeping and billing history.

● System Functionality 5: Employee Login & Role Management

Provides secure login for employees and administrators, ensuring only authorized users can access restricted modules.

● System Functionality 6: Automated Stock & Total Calculation (Triggers)

Uses SQL triggers to compute item total price, update inventory, and recalculate sale totals whenever sales items are inserted or deleted.

● **System Functionality 7: Reporting & Query Support**
Supports aggregate queries, joins, and nested queries for generating insights such as daily sales, top-selling medicines, and low-stock alerts.

[SCREENSHOTS ARE ON GIT]

DDL Commands

```
CREATE TABLE suppliers (
  Sup_ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  Sup_Name VARCHAR(100) NOT NULL,
  Sup_Add VARCHAR(255),
  Sup_Phone VARCHAR(20),
  Sup_Mail VARCHAR(100)
) ENGINE=InnoDB;

CREATE TABLE meds (
  Med_ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  Med_Name VARCHAR(150) NOT NULL,
  Med_Qty INT NOT NULL DEFAULT 0,
  Category VARCHAR(50),
  Med_Price DECIMAL(10,2) NOT NULL DEFAULT 0.00,
  Location_Rack VARCHAR(50)
) ENGINE=InnoDB;

CREATE TABLE customer (
  C_ID INT NOT NULL PRIMARY KEY,
```

```
  C_Fname VARCHAR(100),
  C_Lname VARCHAR(100),
  C_Age INT,
  C_Sex CHAR(1),
  C_Phno VARCHAR(20),
  C_Mail VARCHAR(100)
) ENGINE=InnoDB;

CREATE TABLE employee (
  E_ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  E_Fname VARCHAR(100),
  E_Lname VARCHAR(100),
  E_Bdate DATE,
  E_Age INT,
  E_Sex CHAR(1),
  E_Type VARCHAR(50),
  E_Date DATE,
  E_Add VARCHAR(255),
  E_Mail VARCHAR(100),
  E_Phone VARCHAR(20),
  E_Sal DECIMAL(10,2)
) ENGINE=InnoDB;

CREATE TABLE emplogin (
  E_Username VARCHAR(80) PRIMARY KEY,
  E_Password VARCHAR(255) NOT NULL,
```

```
  E_ID INT,
  FOREIGN KEY (E_ID) REFERENCES employee(E_ID)
     ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE=InnoDB;

CREATE TABLE admin (
  A_Username VARCHAR(80) PRIMARY KEY,
  A_Password VARCHAR(255) NOT NULL
) ENGINE=InnoDB;

CREATE TABLE purchase (
  P_ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  Med_ID INT NOT NULL,
  Sup_ID INT,
  P_Qty INT NOT NULL,
  P_Cost DECIMAL(10,2),
  P_Date DATE,
  Mfg_Date DATE,
  Exp_Date DATE,
  FOREIGN KEY (Med_ID) REFERENCES meds(Med_ID)
     ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (Sup_ID) REFERENCES suppliers(Sup_ID)
     ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE=InnoDB;

CREATE TABLE sales (
```

```
  SALE_ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  S_Date DATE DEFAULT (CURRENT_DATE),
  S_Time TIME DEFAULT (CURRENT_TIME),
  TOTAL_AMT DECIMAL(12,2) NOT NULL DEFAULT 0.00,
  C_ID INT,
  E_ID INT,
  FOREIGN KEY (C_ID) REFERENCES customer(C_ID)
      ON DELETE SET NULL ON UPDATE CASCADE,
  FOREIGN KEY (E_ID) REFERENCES employee(E_ID)
      ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE=InnoDB;

CREATE TABLE sales_items (
  SALE_ITEM_ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  SALE_ID INT NOT NULL,
  MED_ID INT NOT NULL,
  SALE_QTY INT NOT NULL DEFAULT 1,
  TOT_PRICE DECIMAL(12,2) NOT NULL DEFAULT 0.00,
  FOREIGN KEY (SALE_ID) REFERENCES sales(SALE_ID)
      ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (MED_ID) REFERENCES meds(Med_ID)
      ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB;
```

CRUD operation Screenshots:

[I've just given an instance rest of it is in the .sql i uploaded]

```
MariaDB [pharmacy]> INSERT INTO meds (Med_Name, Med_Qty, Category, Med_Price, Location_Rack) VALUES
    -> ('Dolo 650 MG', 120, 'Tablet', 1.00, 'A1'),
    -> ('Paracetamol 500mg', 300, 'Tablet', 0.75, 'A1'),
    -> ('Amoxicillin 500mg', 150, 'Capsule', 4.50, 'B1'),
    -> ('Gelusil Antacid', 200, 'Tablet', 1.25, 'A3'),
    -> ('Cetirizine 10mg', 500, 'Tablet', 0.90, 'A2');
Query OK, 5 rows affected (0.006 sec)
Records: 5  Duplicates: 0  Warnings: 0

MariaDB [pharmacy]> SELECT * FROM meds;
+--------+------------------+---------+----------+-----------+---------------+
| Med_ID | Med_Name         | Med_Qty | Category | Med_Price | Location_Rack |
+--------+------------------+---------+----------+-----------+---------------+
|      1 | Dolo 650 MG      |     120 | Tablet   |      1.00 | A1            |
|      2 | Paracetamol 500mg |     300 | Tablet   |      0.75 | A1            |
|      3 | Amoxicillin 500mg |     150 | Capsule  |      4.50 | B1            |
|      4 | Gelusil Antacid  |     200 | Tablet   |      1.25 | A3            |
|      5 | Cetirizine 10mg  |     500 | Tablet   |      0.90 | A2            |
+--------+------------------+---------+----------+-----------+---------------+
5 rows in set (0.001 sec)

MariaDB [pharmacy]> SELECT * FROM meds WHERE Med_ID = 2;
+--------+------------------+---------+----------+-----------+---------------+
| Med_ID | Med_Name         | Med_Qty | Category | Med_Price | Location_Rack |
+--------+------------------+---------+----------+-----------+---------------+
|      2 | Paracetamol 500mg |     300 | Tablet   |      0.75 | A1            |
+--------+------------------+---------+----------+-----------+---------------+
1 row in set (0.007 sec)

MariaDB [pharmacy]> UPDATE meds
    -> SET Med_Qty = 350,
    ->     Med_Price = 0.85
    -> WHERE Med_ID = 2;
Query OK, 1 row affected (0.007 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [pharmacy]> DELETE FROM meds WHERE Med_ID = 2;
Query OK, 1 row affected (0.003 sec)

MariaDB [pharmacy]> SELECT * FROM meds WHERE Med_ID = 2;
Empty set (0.002 sec)

MariaDB [pharmacy]> SELECT * FROM meds;
+--------+------------------+---------+----------+-----------+---------------+
| Med_ID | Med_Name         | Med_Qty | Category | Med_Price | Location_Rack |
+--------+------------------+---------+----------+-----------+---------------+
|      1 | Dolo 650 MG      |     120 | Tablet   |      1.00 | A1            |
|      3 | Amoxicillin 500mg |     150 | Capsule  |      4.50 | B1            |
|      4 | Gelusil Antacid  |     200 | Tablet   |      1.25 | A3            |
|      5 | Cetirizine 10mg  |     500 | Tablet   |      0.90 | A2            |
+--------+------------------+---------+----------+-----------+---------------+
4 rows in set (0.001 sec)

MariaDB [pharmacy]>
```

Note: Commands were executed in MariaDB because XAMPP on macOS uses MariaDB as its default SQL server, which is fully compatible with standard MySQL syntax.

10. Triggers, Procedures/Functions, Nested query, Join, Aggregate queries

A. Function Used in the System

Function: fn_sale_total()

```
DELIMITER $$

CREATE FUNCTION fn_sale_total(p_sale_id INT)

RETURNS DECIMAL(12,2)

DETERMINISTIC

BEGIN

   DECLARE v_total DECIMAL(12,2) DEFAULT 0.00;

   SELECT COALESCE(SUM(TOT_PRICE), 0.00)

   INTO v_total

   FROM sales_items

   WHERE SALE_ID = p_sale_id;

   RETURN v_total;

END $$

DELIMITER ;
```

Description: Calculates the total amount of a sale by adding all item totals.

B. Triggers Used in the System

1. BEFORE INSERT Trigger – Automatically sets item total price

```
DELIMITER $$
CREATE TRIGGER trg_si_bi_set_tot_price
BEFORE INSERT ON sales_items
FOR EACH ROW
BEGIN
  DECLARE v_price DECIMAL(10,2);
  SELECT Med_Price INTO v_price
  FROM meds
  WHERE Med_ID = NEW.MED_ID;

  IF v_price IS NULL THEN
    SET NEW.TOT_PRICE = 0.00;
  ELSE
    SET NEW.TOT_PRICE = v_price * NEW.SALE_QTY;
  END IF;
END $$
DELIMITER ;
```

2. AFTER INSERT Trigger – Updates stock & sale total

```
DELIMITER $$
CREATE TRIGGER trg_si_ai_update_stock_total
AFTER INSERT ON sales_items
FOR EACH ROW
BEGIN
  UPDATE meds
  SET Med_Qty = GREATEST(Med_Qty - NEW.SALE_QTY, 0)
```

```
  WHERE Med_ID = NEW.MED_ID;


  UPDATE sales
  SET TOTAL_AMT = fn_sale_total(NEW.SALE_ID)
  WHERE SALE_ID = NEW.SALE_ID;
END $$
DELIMITER ;
```

3. AFTER DELETE Trigger – Restores stock & recalculates total

```
DELIMITER $$
CREATE TRIGGER trg_si_ad_restore_stock_total
AFTER DELETE ON sales_items
FOR EACH ROW
BEGIN
  UPDATE meds
  SET Med_Qty = Med_Qty + OLD.SALE_QTY
  WHERE Med_ID = OLD.MED_ID;


  UPDATE sales
  SET TOTAL_AMT = fn_sale_total(OLD.SALE_ID)
  WHERE SALE_ID = OLD.SALE_ID;
END $$
DELIMITER ;
```

C. Nested Query Example

```
SELECT Med_Name, Med_Price
FROM meds
WHERE Med_Price > (
  SELECT AVG(Med_Price)
  FROM meds
);
```

Description: Displays medicines priced above the average price.

D. Join Query Example

```
SELECT s.SALE_ID, c.C_Fname, m.Med_Name, si.SALE_QTY
FROM sales s
JOIN customer c ON s.C_ID = c.C_ID
JOIN sales_items si ON s.SALE_ID = si.SALE_ID
JOIN meds m ON si.MED_ID = m.Med_ID;
```

Description: Shows sale details with customer and medicine information.

E. Aggregate Query Example

```
SELECT Category, COUNT(*) AS Total_Medicines, SUM(Med_Qty) AS Total_Stock
FROM meds
GROUP BY Category;
```

Description: Counts medicines per category and total available stock.

Code Snippets for Invoking the Procedures / Functions / Triggers

A. Invoking the Function (`fn_sale_total`)

```
SELECT fn_sale_total(1) AS TotalAmount;
```

Description:
This calls the function to compute the total amount for **Sale ID = 1**.

B. Trigger Invocation – BEFORE & AFTER INSERT

```
INSERT INTO sales_items (SALE_ID, MED_ID, SALE_QTY)

VALUES (1, 3, 2);
```

Description:
When this statement runs:

- The **BEFORE INSERT** trigger automatically sets `TOT_PRICE`.
- The **AFTER INSERT** trigger updates stock and recalculates the sale total.

C. Trigger Invocation – AFTER DELETE

```
DELETE FROM sales_items

WHERE SALE_ITEM_ID = 10;
```

Description:
Deleting a sales item activates the **AFTER DELETE** trigger, which restores medicine stock and recalculates the sale total.

D. Viewing Function & Trigger Definitions

```
SHOW CREATE FUNCTION fn_sale_total;

SHOW CREATE TRIGGER trg_si_bi_set_tot_price;

SHOW CREATE TRIGGER trg_si_ai_update_stock_total;

SHOW CREATE TRIGGER trg_si_ad_restore_stock_total;
```

Description:
These commands display the full SQL definitions of the function and all triggers used in the project.

E. Complete Invocation Example – Sale Creation + Trigger Execution + Function Call

```
INSERT INTO sales (C_ID, E_ID) VALUES (200001, 1);

SET @sid = LAST_INSERT_ID();


INSERT INTO sales_items (SALE_ID, MED_ID, SALE_QTY)

VALUES (@sid, 1, 2);


SELECT fn_sale_total(@sid) AS ComputedTotal;
```

Description:

Step-by-step demonstration:

1. A sale is created.
2. A sales item is added → triggers execute automatically.
3. The function is called to fetch the updated total.