SQL Project Report: Pharmaceutical Management System Using SQL

---

1. Project Title:

Pharmaceutical Management System Using SQL

---

2. Objective / Problem Statement:

This project aims to build and analyze a comprehensive pharmaceutical management system using SQL. It focuses on handling key healthcare operations such as medicine inventory, prescriptions, patient and doctor data, suppliers, restocking logic, and sales transactions. The project also includes automation through triggers and stored procedures.

---

3. Tools & Technologies Used:

- **Database:** MySQL
- **Tools:** MySQL Workbench
- **Languages:** SQL

---

4. Dataset Description:

- **Tables Created:** Medicines, Patients, Doctors, Prescriptions, Sales, Suppliers, Prescription_Details, Medicine_Locations, Medicine_Suppliers, Restock_Alerts, Deleted_Prescriptions, Expired_Medicine_Log, Expired_Stock
- **Key Data Points:**
  - Medicines and their prices, stock, expiry
  - Patients' demographic data
  - Prescription details including dosage and duration
  - Sales transactions and billing data
  - Supplier connections to medicines
  - Automated restock and expiry tracking

---

5. SQL Tasks and Analysis:

✓Schema Design and Data Insertion

All relevant tables were created with constraints and relationships, and sample data was inserted for meaningful analysis.

✅Queries for Business Insights

1. Medicines below stock threshold

```
SELECT * FROM Medicines WHERE Stock_Quantity < 10;
```

2. Expired medicines

```
SELECT * FROM Medicines WHERE Expiry_Date < CURDATE();
```

3. Top 3 most sold medicines

```
SELECT Medicine_ID, SUM(Quantity) AS Total_Sold
FROM Sales
GROUP BY Medicine_ID
ORDER BY Total_Sold DESC
LIMIT 3;
```

4. Revenue by medicine

```
SELECT s.Medicine_ID, m.Medicine_Name, SUM(s.Quantity * m.Price_Per_Unit) AS Total_Revenue
FROM Sales s
JOIN Medicines m ON s.Medicine_ID = m.Medicine_ID
GROUP BY s.Medicine_ID, m.Medicine_Name;
```

5. Patients per doctor

```
SELECT Doctor_ID, COUNT(DISTINCT Patient_ID) AS Total_Patients
FROM Prescriptions
GROUP BY Doctor_ID;
```

6. Sales in last 30 days

```
SELECT Sale_Date, SUM(Quantity) AS Total_Quantity
FROM Sales
WHERE Sale_Date >= CURDATE() - INTERVAL 30 DAY
GROUP BY Sale_Date;
```

7. Prescribed but not sold

```
SELECT DISTINCT pd.Medicine_ID
FROM Prescription_Details pd
LEFT JOIN Sales s ON pd.Medicine_ID = s.Medicine_ID
WHERE s.Medicine_ID IS NULL;
```

8. Prescriptions by doctor in last 6 months

```
SELECT Doctor_ID, COUNT(*) AS Total_Prescriptions
FROM Prescriptions
WHERE Prescription_Date >= CURDATE() - INTERVAL 6 MONTH
GROUP BY Doctor_ID;
```

9. Sold but not prescribed

```
SELECT DISTINCT s.Medicine_ID
FROM Sales s
LEFT JOIN Prescription_Details pd ON s.Medicine_ID = pd.Medicine_ID
WHERE pd.Medicine_ID IS NULL;
```

10. Prescriptions with > 3 medicines

```
SELECT Prescription_ID
FROM Prescription_Details
GROUP BY Prescription_ID
HAVING COUNT(DISTINCT Medicine_ID) > 3;
```

11. Patients purchasing from multiple cities

```
SELECT s.Patient_ID
FROM Sales s
JOIN Medicine_Locations ml ON s.Medicine_ID = ml.Medicine_ID
GROUP BY s.Patient_ID
HAVING COUNT(DISTINCT ml.Location) > 1;
```

12. Suppliers of out-of-stock medicines

```
SELECT DISTINCT s.Supplier_ID, s.Supplier_Name
FROM Suppliers s
JOIN Medicine_Suppliers ms ON s.Supplier_ID = ms.Supplier_ID
JOIN Medicines m ON ms.Medicine_ID = m.Medicine_ID
WHERE m.Stock_Quantity = 0;
```

13. Most prescribed medicine per diagnosis

```
SELECT Diagnosis, Medicine_ID, COUNT(*) AS Count
FROM Prescriptions p
JOIN Prescription_Details pd ON p.Prescription_ID = pd.Prescription_ID
GROUP BY Diagnosis, Medicine_ID
ORDER BY Diagnosis, Count DESC;
```

14. City-wise sales quantity

```
SELECT m.Medicine_ID, p.City, SUM(s.Quantity) AS Total_Sold
FROM Sales s
JOIN Patients p ON s.Patient_ID = p.Patient_ID
JOIN Medicines m ON s.Medicine_ID = m.Medicine_ID
GROUP BY m.Medicine_ID, p.City;
```

15. Doctors with no prescriptions

```
SELECT d.Doctor_ID, d.Name
FROM Doctors d
LEFT JOIN Prescriptions p ON d.Doctor_ID = p.Doctor_ID
WHERE p.Prescription_ID IS NULL;
```

16. **Same-day prescription and sale**

```
SELECT DISTINCT s.Medicine_ID, s.Patient_ID, s.Sale_Date
FROM Sales s
JOIN Prescriptions p ON s.Patient_ID = p.Patient_ID AND s.Sale_Date = p.Prescription_Date
JOIN Prescription_Details pd ON p.Prescription_ID = pd.Prescription_ID
WHERE s.Medicine_ID = pd.Medicine_ID;
```

17. **Late purchases after prescription**

```
SELECT s.Patient_ID, s.Medicine_ID, DATEDIFF(s.Sale_Date, p.Prescription_Date) AS Days_After
FROM Sales s
JOIN Prescriptions p ON s.Patient_ID = p.Patient_ID
JOIN Prescription_Details pd ON p.Prescription_ID = pd.Prescription_ID AND s.Medicine_ID = pd.Medicine_ID
WHERE DATEDIFF(s.Sale_Date, p.Prescription_Date) > 15;
```

18. **Doctor prescribing most medicines**

```
SELECT p.Doctor_ID, COUNT(pd.Medicine_ID) AS Total_Medicines
FROM Prescriptions p
JOIN Prescription_Details pd ON p.Prescription_ID = pd.Prescription_ID
GROUP BY p.Doctor_ID
ORDER BY Total_Medicines DESC
LIMIT 1;
```

19. **Unauthorized patient purchases**

```
SELECT DISTINCT s.Patient_ID, s.Medicine_ID
FROM Sales s
LEFT JOIN (
    SELECT p.Patient_ID, pd.Medicine_ID
    FROM Prescriptions p
    JOIN Prescription_Details pd ON p.Prescription_ID = pd.Prescription_ID
) AS prescribed_meds ON s.Patient_ID = prescribed_meds.Patient_ID AND s.Medicine_ID =
prescribed_meds.Medicine_ID
WHERE prescribed_meds.Medicine_ID IS NULL;
```

---

6. Triggers and Procedures:

Trigger: Reduce stock after sale

```
CREATE TRIGGER reduce_stock_after_sale
AFTER INSERT ON Sales
FOR EACH ROW
BEGIN
  UPDATE Medicines
  SET Stock_Quantity = Stock_Quantity - NEW.Quantity
  WHERE Medicine_ID = NEW.Medicine_ID;
END;
```

□ Trigger: Update last_prescribed_date

```
CREATE TRIGGER update_last_prescribed
AFTER INSERT ON Prescription_Details
FOR EACH ROW
BEGIN
  DECLARE presc_date DATE;
  SELECT Prescription_Date INTO presc_date
  FROM Prescriptions
  WHERE Prescription_ID = NEW.Prescription_ID;

  UPDATE Medicines
  SET Last_Prescribed_Date = presc_date
  WHERE Medicine_ID = NEW.Medicine_ID;
END;
```

□ Trigger: Restock alert

```
CREATE TRIGGER trigger_restock_alert
AFTER UPDATE ON Medicines
FOR EACH ROW
BEGIN
  IF NEW.Stock_Quantity < 10 AND OLD.Stock_Quantity >= 10 THEN
    INSERT INTO Restock_Alerts (Alert_ID, Medicine_ID, Alert_Date, Note)
    VALUES (
      CONCAT('A', LPAD(FLOOR(RAND() * 10000), 4, '0')),
      NEW.Medicine_ID,
      CURDATE(),
      'Stock below threshold'
    );
  END IF;
END;
```

□ Procedure: Add prescription with multiple medicines

```
CREATE PROCEDURE AddPrescription (...)
-- Full logic with JSON iteration
```

□ Procedure: Generate bill

```
CREATE PROCEDURE GenerateBill (...)
-- Returns line item and total billing
```

□□ Procedure: Patients by doctor

```
CREATE PROCEDURE PatientsByDoctor (...)
-- Lists patients treated by a given doctor
```

□ Procedure: Move expired medicines

```
CREATE PROCEDURE ExpireMedicines()
-- Moves expired medicines to Expired_Stock table
```

□ Procedure: Sales summary by category

```
CREATE PROCEDURE SalesSummaryByCategory (...)
-- Returns total quantity and revenue for a category
```

---

7. Insights / Outcomes:

- Automatic restocking alerts reduce manual effort.
- Sales trends and prescription data help track demand.
- Revenue and city-wise analysis help in business expansion planning.
- Triggers ensure data consistency and real-time monitoring.
- Stored procedures streamline complex operations.

---

8. Visualizations (Optional Description):

- Bar chart: Top-selling medicines
- Line graph: Daily sales trend
- Pie chart: Category-wise revenue
- Heatmap: Sales by city and medicine

---

9. GitHub / Portfolio Link (If Any):

Add your GitHub repository link or portfolio page here.