SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR
Constituent of Symbiosis International (Deemed University), Pune
(Established under Section 3 of the UGC Act of 1956 wide notification number F-9-12/2001-U-3 of Government of India)

DBMS MINI PROJECT REPORT

Pharmacy Management System

Submitted By:

Renuka Wadetwar	22070521159
Poornima Mendhekar	22070521162
Mansee Dakhole	22070521164

V SEM B

Submitted To:

Dr. Giridhar Urkude
Assistant Professor

INDEX

1. INTRODUCTION

Pharmacies manage a wide range of data every day, including customer details, prescriptions, insurance records, drug inventory, and employee information. Handling this data manually or through inefficient systems often leads to errors, such as stock mismanagement, billing inaccuracies, and difficulties in tracking prescriptions. A well-structured database system can make these tasks easier and faster to manage. By moving away from manual methods and embracing automation, pharmacies can ensure better data accuracy, smoother operations, and improved customer service. Using a strong database management system (DBMS) improves how well a pharmacy operates. It helps organize and manage data more effectively. Additionally, it ensures that data is easy to access, retrieve, and secure.

2. PROBLEM STATEMENT

The goal of this project is to develop a Pharmacy Management System that automates important daily tasks like managing customer information, processing prescriptions, and keeping track of drug inventory. The current manual methods often lead to mistakes, such as incorrect billing and stock issues. To solve these problems, the system will use various database features, including functions, procedures, views, joins, and exceptions to make operations smoother. This will help with tasks like checking stock levels, applying discounts, and notifying staff of necessary actions regarding low inventory or expired drugs, among other functions. Overall, this system aims to improve the pharmacy's efficiency, reduce errors, and ensure that important information is easy to access and secure.

3. DESCRIPTION

This project develops a Pharmacy Management System that organizes and automates daily pharmacy tasks using a database. The system includes several tables, each with its details and purpose:

1. Insurance:

Attributes: InsuranceID, CompName, StartDate, EndDate, CoInsurance

This table keeps track of insurance plans for customers, showing which customers have insurance and what their coverage includes.

2. Customer:

Attributes: AadharNo, Fname, Lname, Gender, DoB, Address, Phone, InsuranceID

This table keeps track of customer information, including their personal details and insurance, making it easy to manage their records.

3. Prescription:

Attributes: PrespID, AadharNo, DocID, PrespDate

This table helps track prescriptions for customers, recording important details like the doctor and date, so their history is easy to access.

4. PrescribedDrugs:

Attributes: PrespID, DrugName, PrespQty, RefillLimit

This table tracks the specific drugs prescribed to customers, detailing the quantity and refill limits for each medication.

5. Medicine:

Attributes: DrugName, BatchNo, Price, StockQty, Manufacturer, MedType, ExpiryDate

This table contains details about the medicines available in the pharmacy, helping to manage stock and expiration dates.

6. Employee:

Attributes: EmpID, Fname, Lname, AadharNo, Phone, License, Role, StartDate, EndDate, Salary, DoB

This table holds information about pharmacy employees, including their roles and contact details.

7. Orders:

Attributes: OrderID, PrespID, EmpID, OrderDate

This table records orders made for prescriptions, ensuring proper order management.

8. OrderedDrugs:

Attributes: OrderID, DrugName, BatchNo, OrderedQty, Price

This table tracks the drugs included in each order, ensuring accurate dispensing and billing.

9. Bill:

Attributes: OrderID, AadharNo, TotalAmt, InsrPay, CustPay

This table keeps track of billing information for each order, including total amounts and payment details.

10. Employee_Notification:

Attributes: EmpID, NotificationID

This table links employees to important notifications, keeping them informed about updates.

11.Notification:

Attributes: NotificationID, Type, Message

This table stores alerts and reminders for staff, such as low stock or important updates.

12. Employee_Disposed_Drugs:

Attributes: EmpID, DrugName, BatchNo, DisposalDate

This table records drugs that employees have disposed of, helping manage safety.

13. Disposed:

Attributes: DrugName, BatchNo, DisposalDate, DispQty, Company

This table tracks disposed drugs, ensuring accurate record-keeping for safety regulations.

4. ER DIAGRAM

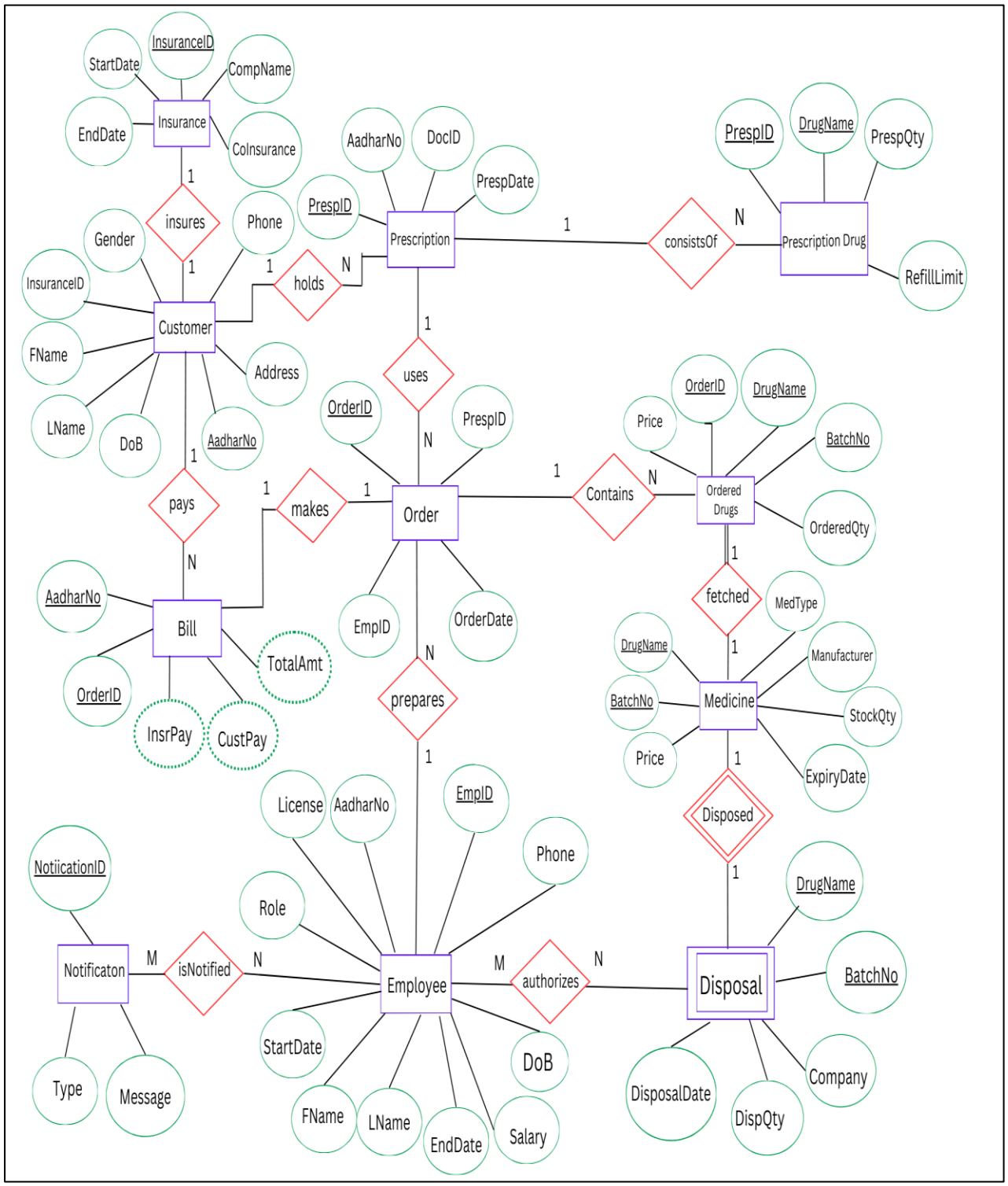

Figure 1: ER Diagram for Pharmacy Management System

5. SCHEMA DIAGRAM

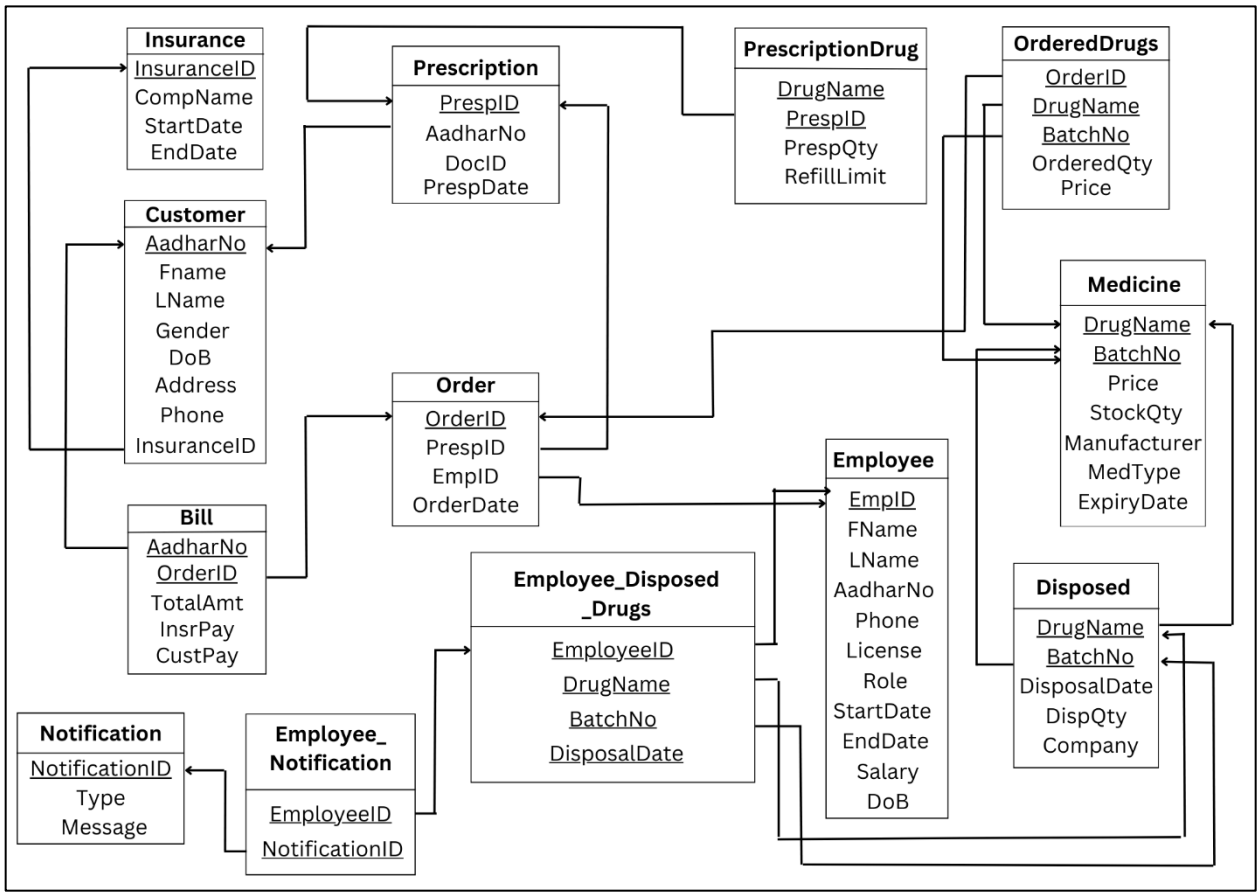

Figure 2: Schema Diagram for Pharmacy Management System

Key Relationships in the above diagram:

1. A customer can have one insurance policy, therefore the relationship is One-to-One.

2. A customer can have multiple prescriptions, therefore the relationship is One-to-Many.

3. A prescription can include multiple drugs, therefore the relationship is One-to-Many.

4. A prescription can generate multiple orders, therefore the relationship is One-to-Many.

5. An order can include multiple ordered drugs, therefore the relationship is One-to-Many.

6. Ordered drugs are linked to medicines, therefore the relationship is Many-to-One.

7. Each order is linked to a bill, therefore the relationship is One-to-One.

8. An employee prepares multiple orders, therefore the relationship is One-to-Many.

9. An employee can receive multiple notifications, therefore the relationship is One-to-Many.

10. A notification can be sent to multiple employees, therefore the relationship is One-to-Many.

11. A medicine can be disposed of through multiple disposal records, therefore the relationship is One-to-Many.

12. An employee disposes of multiple drugs, therefore the relationship is One-to-Many.

13. A drug disposal record is linked to a specific employee disposal entry, therefore the relationship is One-to-One.

6. NORMALIZATION

1. Insurance (InsuranceID, CompName, StartDate, EndDate)
Functional Dependency: InsuranceID → CompName, StartDate, EndDate

2. Customer (AadharNo, Fname, Lname, Gender, DoB, Address, Phone, InsuranceID)
Functional Dependency: AadharNo → Fname, Lname, Gender, DoB, Address, Phone, InsuranceID

3. Prescription (PrespID, AadharNo, DocID, PrespDate)
Functional Dependency: PrespID → AadharNo, DocID, PrespDate

4. PrescriptionDrug (PrespID, DrugName, PrespQty, RefillLimit)
Functional Dependency: (PrespID, DrugName) → PrespQty, RefillLimit

5. Order (OrderID, PrespID, EmpID, OrderDate)
Functional Dependency: OrderID → PrespID, EmpID, OrderDate

6. OrderedDrugs (OrderID, DrugName, BatchNo, OrderedQty, Price)
Functional Dependency: (OrderID, DrugName, BatchNo) → OrderedQty, Price

7. Bill (OrderID, AadharNo, TotalAmt, InsrPay, CustPay)
Functional Dependency: (OrderID, AadharNo) → TotalAmt, InsrPay, CustPay

8. Employee (EmpID, Fname, Lname, AadharNo, Phone, License, Role, StartDate, EndDate, Salary, DoB)
Functional Dependency: EmpID → Fname, Lname, AadharNo, Phone, License, Role, StartDate, EndDate, Salary, DoB

9. Employee_Notification (EmpID, NotificationID)
Functional Dependency: (EmpID, NotificationID)

10. Notification (NotificationID, Type, Message)
Functional Dependency: NotificationID → Type, Message

11. Employee_Disposed_Drugs(EmpID, DrugName, BatchNo, DisposalDate)
Functional Dependency: (EmpID, DrugName, BatchNo) → DisposalDate

12. Disposed (DrugName, BatchNo, DisposalDate, DispQty, Company)
Functional Dependency: (DrugName, BatchNo) → DisposalDate, DispQty, Company

13. Medicine (DrugName, BatchNo, Price, StockQty, Manufacturer, MedType, ExpiryDate)
Functional Dependency: (DrugName, BatchNo) → Price, StockQty, Manufacturer, MedType,ExpiryDate

7. CONSTRUCTION OF DATABASE

```
SQL> CREATE TABLE Insurance (
  2      InsuranceID VARCHAR(10) PRIMARY KEY,
  3      CompName VARCHAR(50),
  4      StartDate DATE,
  5      EndDate DATE,
  6      CoInsurance VARCHAR(50)
  7  );

Table created.

SQL> CREATE TABLE Customer (
  2      AadharNo VARCHAR(12) PRIMARY KEY,
  3      Fname VARCHAR(50),
  4      Lname VARCHAR(50),
  5      Gender CHAR(1),
  6      DoB DATE,
  7      Address VARCHAR(100),
  8      Phone VARCHAR(15),
  9      InsuranceID VARCHAR(10),
 10      FOREIGN KEY (InsuranceID) REFERENCES Insurance(InsuranceID)
 11  );

Table created.

SQL> CREATE TABLE Prescription (
  2      PrespID VARCHAR(10) PRIMARY KEY,
  3      AadharNo VARCHAR(12),
  4      DocID VARCHAR(10),
  5      PrespDate DATE,
  6      FOREIGN KEY (AadharNo) REFERENCES Customer(AadharNo)
  7  );

Table created.

SQL> CREATE TABLE PrescriptionDrug (
  2      PrespID VARCHAR(10),
  3      DrugName VARCHAR(50),
  4      PrespQty INT,
  5      RefillLimit INT,
  6      PRIMARY KEY (PrespID, DrugName),
  7      FOREIGN KEY (PrespID) REFERENCES Prescription(PrespID)
  8  );

Table created.
```

Figure 3: Creating Insurance, Customer, Prescription and PrescriptionDrug tables

```
SQL> CREATE TABLE Employee (
  2      EmpID VARCHAR(10) PRIMARY KEY,
  3      Fname VARCHAR(50),
  4      Lname VARCHAR(50),
  5      AadharNo VARCHAR(12),
  6      Phone VARCHAR(15),
  7      License VARCHAR(20),
  8      Role VARCHAR(50),
  9      StartDate DATE,
 10      EndDate DATE,
 11      Salary DECIMAL(10, 2),
 12      DoB DATE
 13  );

Table created.

SQL> CREATE TABLE Orders (
  2      OrderID VARCHAR(10) PRIMARY KEY,
  3      PrespID VARCHAR(10),
  4      EmpID VARCHAR(10),
  5      OrderDate DATE,
  6      FOREIGN KEY (PrespID) REFERENCES Prescription(PrespID),
  7      FOREIGN KEY (EmpID) REFERENCES Employee(EmpID)
  8  );

Table created.

SQL> CREATE TABLE OrderedDrugs (
  2      OrderID VARCHAR(10),
  3      DrugName VARCHAR(50),
  4      BatchNo VARCHAR(10),
  5      OrderedQty INT,
  6      Price DECIMAL(10, 2),
  7      PRIMARY KEY (OrderID, DrugName, BatchNo),
  8      FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
  9  );

Table created.

SQL> CREATE TABLE Bill (
  2      OrderID VARCHAR(10),
  3      AadharNo VARCHAR(12),
  4      TotalAmt DECIMAL(10, 2),
  5      InsrPay DECIMAL(10, 2),
  6      CustPay DECIMAL(10, 2),
  7      PRIMARY KEY (OrderID, AadharNo),
  8      FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
  9      FOREIGN KEY (AadharNo) REFERENCES Customer(AadharNo)
 10  );

Table created.
```

```
SQL> CREATE TABLE Employee_Notification (
  2      EmpID VARCHAR(10),
  3      NotificationID VARCHAR(10),
  4      PRIMARY KEY (EmpID, NotificationID),
  5      FOREIGN KEY (EmpID) REFERENCES Employee(EmpID)
  6  );

Table created.

SQL> CREATE TABLE Notification (
  2      NotificationID VARCHAR(10) PRIMARY KEY,
  3      Type VARCHAR(50),
  4      Message VARCHAR(70)
  5  );

Table created.

SQL> CREATE TABLE Employee_Disposed_Drugs (
  2      EmpID VARCHAR(10),
  3      DrugName VARCHAR(50),
  4      BatchNumber VARCHAR(10),
  5      DisposalDate DATE,
  6      PRIMARY KEY (EmpID, DrugName, BatchNumber),
  7      FOREIGN KEY (EmpID) REFERENCES Employee(EmpID)
  8  );

Table created.

SQL> CREATE TABLE Disposed (
  2      DrugName VARCHAR(50),
  3      BatchNo VARCHAR(10),
  4      DisposalDate DATE,
  5      DispQty INT,
  6      Company VARCHAR(50),
  7      PRIMARY KEY (DrugName, BatchNo)
  8  );

Table created.

SQL> CREATE TABLE Medicine (
  2      DrugName VARCHAR(50),
  3      BatchNo VARCHAR(10),
  4      Price DECIMAL(10, 2),
  5      StockQty INT,
  6      Manufacturer VARCHAR(50),
  7      MedType VARCHAR(50),
  8      ExpiryDate DATE,
  9      PRIMARY KEY (DrugName, BatchNo)
 10  );

Table created.
```

Figure 4: Creating Employee, Orders, OrderedDrugs, Bill, Employee_Notification, Notification, Employee_Disposed_Drugs, Disposed and Medicine tables

8. IMPLEMENTATION

1. In-Built Functions

The in-built functions in the Pharmacy Management System streamline key tasks. The SUM function calculates total sales, the COUNT function tracks the number of prescriptions per customer, and the MAX function finds the highest price of medicines. These functions improve data accuracy and efficiency.

```
SQL> SELECT SUM(TotalAmt) AS TotalSales FROM Bill;

TOTALSALES
----------
     55000
```

Figure 5: The SUM() function is used to calculate the total sales amount from the Bill table

```
SQL> SELECT COUNT(PrespID) AS PrescriptionCount FROM Prescription WHERE AadharNo = '123456789012';

PRESCRIPTIONCOUNT
-----------------
                1
```

Figure 6: The COUNT() function is used to count the number of prescriptions associated with a specific Aadhar number in the Prescription table

```
SQL> SELECT MAX(Price) AS MaxPrice FROM Medicine;

  MAXPRICE
----------
       300
```

Figure 7: MAX() is used to retrieve the highest price from the Medicine table

2. Procedure

- The procedure CheckDrugStock checks the stock quantity of a specified drug and prints its availability. If the drug is not found, it informs that the drug does not exist in the inventory.

```
SQL> CREATE OR REPLACE PROCEDURE CheckDrugStock (p_DrugName IN VARCHAR2) IS
  2      StockCount NUMBER;
  3  BEGIN
  4      SELECT StockQty INTO StockCount
  5      FROM Medicine
  6      WHERE DrugName = p_DrugName;
  7
  8      IF StockCount > 0 THEN
  9          DBMS_OUTPUT.PUT_LINE('The drug ' || p_DrugName || ' is in stock with quantity: ' || Stock
Count);
 10      ELSE
 11          DBMS_OUTPUT.PUT_LINE('The drug ' || p_DrugName || ' is out of stock.');
 12      END IF;
 13  EXCEPTION
 14      WHEN NO_DATA_FOUND THEN
 15          DBMS_OUTPUT.PUT_LINE('The drug ' || p_DrugName || ' does not exist in the inventory.');
 16  END CheckDrugStock;
 17  /

Procedure created.

SQL> BEGIN
  2      CheckDrugStock('DrugA');
  3  END;
  4  /
The drug DrugA is in stock with quantity: 50

PL/SQL procedure successfully completed.
```

Figure 8: Procedure to check and print the stock quantity of a specific drug

- The GenerateOrderReport procedure fetches and prints the Order ID, Prescription ID, and Employee ID for each order in the Orders table.

```
SQL> CREATE OR REPLACE PROCEDURE GenerateOrderReport
  2  IS
  3  BEGIN
  4      FOR rec IN (SELECT OrderID, PrespID, EmpID FROM Orders) LOOP
  5          DBMS_OUTPUT.PUT_LINE('Order ID: ' || rec.OrderID || ', Prescription ID: ' || rec.PrespID || ', Employee ID: ' || rec.EmpID);
  6      END LOOP;
  7  END GenerateOrderReport;
  8  /

Procedure created.

SQL> SET SERVEROUTPUT ON;
SQL>
SQL> BEGIN
  2      GenerateOrderReport;
  3  END;
  4  /
Order ID: ORD001, Prescription ID: PRES001, Employee ID: EMP001
Order ID: ORD002, Prescription ID: PRES002, Employee ID: EMP002
Order ID: ORD003, Prescription ID: PRES003, Employee ID: EMP003
Order ID: ORD004, Prescription ID: PRES004, Employee ID: EMP004
Order ID: ORD005, Prescription ID: PRES005, Employee ID: EMP005
Order ID: ORD006, Prescription ID: PRES006, Employee ID: EMP006
Order ID: ORD007, Prescription ID: PRES007, Employee ID: EMP007
Order ID: ORD008, Prescription ID: PRES008, Employee ID: EMP008
Order ID: ORD009, Prescription ID: PRES009, Employee ID: EMP009
Order ID: ORD010, Prescription ID: PRES010, Employee ID: EMP010

PL/SQL procedure successfully completed.
```

Figure 9: Procedure to generate order report

3. User-defined Functions

- The user-defined function CalculateDiscount figures out how much discount to give based on the total amount spent, using different rates for different spending levels. It then returns the amount of discount.

```
SQL> CREATE OR REPLACE FUNCTION CalculateDiscount(TotalAmount IN NUMBER)
  2  RETURN NUMBER
  3  IS
  4      DiscountAmount NUMBER;
  5  BEGIN
  6      IF TotalAmount < 1000 THEN
  7          DiscountAmount := 0;
  8      ELSIF TotalAmount >= 1000 AND TotalAmount < 5000 THEN
  9          DiscountAmount := TotalAmount * 0.05;
 10      ELSIF TotalAmount >= 5000 AND TotalAmount < 10000 THEN
 11          DiscountAmount := TotalAmount * 0.10;
 12      ELSE
 13          DiscountAmount := TotalAmount * 0.15;
 14      END IF;
 15
 16      RETURN DiscountAmount;
 17  END CalculateDiscount;
 18  /

Function created.

SQL> SET SERVEROUTPUT ON;
SQL>
SQL> DECLARE
  2      DiscountAmount NUMBER;
  3  BEGIN
  4      DiscountAmount := CalculateDiscount(950);
  5      DBMS_OUTPUT.PUT_LINE('The discount for amount 950 is: ' || DiscountAmount);
  6
  7      DiscountAmount := CalculateDiscount(2000);
  8      DBMS_OUTPUT.PUT_LINE('The discount for amount 2000 is: ' || DiscountAmount);
  9
 10      DiscountAmount := CalculateDiscount(7000);
 11      DBMS_OUTPUT.PUT_LINE('The discount for amount 7000 is: ' || DiscountAmount);
 12
 13      DiscountAmount := CalculateDiscount(12000);
 14      DBMS_OUTPUT.PUT_LINE('The discount for amount 12000 is: ' || DiscountAmount);
 15  END;
 16  /
The discount for amount 950 is: 0
The discount for amount 2000 is: 100
The discount for amount 7000 is: 700
The discount for amount 12000 is: 1800

PL/SQL procedure successfully completed.
```

Figure 10: User-defined function to calculate the discount

- This function counts how many prescriptions a customer has based on their Aadhar number.

```
SQL> CREATE OR REPLACE FUNCTION GetPrescriptionsByCustomer(p_AadharNo IN VARCHAR2)
  2  RETURN NUMBER
  3  IS
  4      PrescriptionCount NUMBER;
  5  BEGIN
  6      SELECT COUNT(*) INTO PrescriptionCount
  7      FROM Prescription
  8      WHERE AadharNo = p_AadharNo;
  9
 10      RETURN PrescriptionCount;
 11  END GetPrescriptionsByCustomer;
 12  /

Function created.

SQL> SET SERVEROUTPUT ON;
SQL>
SQL> DECLARE
  2      Result NUMBER;
  3  BEGIN
  4      Result := GetPrescriptionsByCustomer('123456789012'); -- Replace with the desired Aadhar number
  5      DBMS_OUTPUT.PUT_LINE('Prescription Count: ' || Result);
  6  END;
  7  /
Prescription Count: 1

PL/SQL procedure successfully completed.
```

Figure 11: User-defined function to calculate prescription count by Aadhar number

4. Triggers

- The trigger trg_Uppercase_DrugName automatically converts any drug name to uppercase before it is inserted or updated in the Medicine table.
- An example INSERT statement adds an analgesic drug called "aspirin" to the Medicine table, and an UPDATE statement changes the name of a drug associated with a specific batch number. Finally, the SELECT statement retrieves all records from the Medicine table.

```
SQL> CREATE OR REPLACE TRIGGER trg_Uppercase_DrugName
  2  BEFORE INSERT OR UPDATE ON Medicine
  3  FOR EACH ROW
  4  BEGIN
  5      :NEW.DrugName := UPPER(:NEW.DrugName);
  6  END;
  7  /

Trigger created.

SQL> INSERT INTO Medicine (DrugName, BatchNo, Price, StockQty, Manufacturer, MedType, ExpiryDate)
  2  VALUES ('aspirin', 'BATCH001', 100.00, 50, 'PharmaCorp', 'Analgesic', TO_DATE('2025-01-01', 'YYYY
-MM-DD'));

1 row created.

SQL> UPDATE Medicine SET DrugName = 'ibuprofen' WHERE BatchNo = 'BATCH002';

1 row updated.
```

Figure 12: Trigger to convert drug names to uppercase on insert/update

5. Views

- The view CustomerPrescriptionView combines customer first and last names with their prescription IDs and dates by joining the Customer and Prescription tables on AadharNo. This simplifies data access by allowing users to retrieve relevant information without writing complex queries.

```
SQL> CREATE OR REPLACE VIEW CustomerPrescriptionView AS
  2  SELECT
  3      c.Fname AS CustomerFirstName,
  4      c.Lname AS CustomerLastName,
  5      p.PrespID,
  6      p.PrespDate
  7  FROM
  8      Customer c
  9  JOIN
 10      Prescription p ON c.AadharNo = p.AadharNo;

View created.
```

Figure 13: View to retrieve specific customer and prescription details

- The TotalBillView provides a summarized view of the total amounts billed to each customer in the pharmacy.

```
SQL> CREATE VIEW TotalBillView AS
  2  SELECT
  3      B.AadharNo,
  4      SUM(B.TotalAmt) AS TotalAmount
  5  FROM
  6      Bill B
  7  GROUP BY
  8      B.AadharNo;

View created.

SQL> SELECT * FROM TotalBillView;

AADHARNO     TOTALAMOUNT
------------ -----------
323456789012        3000
723456789012        7000
923456789012        9000
023456789012       10000
223456789012        2000
623456789012        6000
423456789012        4000
123456789012        1000
523456789012        5000
823456789012        8000

10 rows selected.
```

Figure 14: View to summarize totals for each customer

6. Joins

- List all customers along with their prescriptions.

```
SQL> SELECT
  2      c.Fname AS CustomerFirstName,
  3      c.Lname AS CustomerLastName,
  4      p.PrespID AS PrescriptionID
  5  FROM
  6      Customer c
  7  INNER JOIN
  8      Prescription p ON c.AadharNo = p.AadharNo;
```

Figure 15: Inner join

- List all customers and their prescriptions, even if some customers don't have any prescriptions.

```
SQL> SELECT
  2      c.Fname AS CustomerFirstName,
  3      c.Lname AS CustomerLastName,
  4      p.PrespID AS PrescriptionID
  5  FROM
  6      Customer c
  7  LEFT OUTER JOIN
  8      Prescription p ON c.AadharNo = p.AadharNo;
```

Figure 16: Left Outer join

- List all prescriptions and their associated customers, even if some prescriptions are not linked to any customer.

```
SQL> SELECT
  2      c.Fname AS CustomerFirstName,
  3      c.Lname AS CustomerLastName,
  4      p.PrespID AS PrescriptionID
  5  FROM
  6      Customer c
  7  RIGHT OUTER JOIN
  8      Prescription p ON c.AadharNo = p.AadharNo;
```

Figure 17: Right Outer join

- A Cross join generates a Cartesian product, returning all possible combinations of rows from two tables. For example, if one table has 10 rows and another has 5, the result will have 50 rows (10 x 5).

```
SQL> SELECT
  2      Presp.PrespID,
  3      Emp.EmpID,
  4      Emp.Fname,
  5      Emp.LNAME
  6  FROM
  7      Prescription Presp
  8  CROSS JOIN
  9      employee9 Emp;
```

Figure 18: Cross join

7. Exception

- This procedure checks if the employee exists and prints their salary. It handles exceptions by notifying the user if no employee is found or if any other error occurs during execution.

```
SQL> CREATE OR REPLACE PROCEDURE CheckEmployeeSalary(p_EmpID IN VARCHAR2) IS
  2      v_Salary NUMBER;
  3  BEGIN
  4      SELECT Salary INTO v_Salary FROM Employee WHERE EmpID = p_EmpID;
  5
  6      DBMS_OUTPUT.PUT_LINE('Salary for Employee ID ' || p_EmpID || ' is: ' || v_Salary);
  7  EXCEPTION
  8      WHEN NO_DATA_FOUND THEN
  9          DBMS_OUTPUT.PUT_LINE('Error: No employee found with ID ' || p_EmpID);
 10      WHEN OTHERS THEN
 11          DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
 12  END CheckEmployeeSalary;
 13  /

Procedure created.

SQL> BEGIN
  2      CheckEmployeeSalary('EMP001');
  3  END;
  4  /
Salary for Employee ID EMP001 is: 50000

PL/SQL procedure successfully completed.
```

Figure 19: An exception to check whether the employee exists or not

9. CONCLUSION

In conclusion, this project successfully implements a comprehensive database management system for a pharmacy, addressing key areas such as customer details, prescription tracking, drug inventory, employee management, and billing processes. By automating various operations through the use of SQL-based tables, functions, procedures, and triggers, the system ensures data accuracy and efficiency. The use of views and joins simplifies data retrieval, making it easier for users to access and manage important information. Overall, the Pharmacy Management System enhances operational efficiency, minimizes errors, and streamlines day-to-day tasks, ensuring a smooth workflow in the pharmacy environment.