THE CATHOLIC UNIVERSITY OF EASERN AFRICA (CUEA)

A.M.E.C.E.A

FACULTY OF SCIENCE.

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE.

CMT 302: ADVANCED DATABASE SYSTEMS

GROUP PROJECT: PHARMACY INVENTORY MANAGEMENT

MEMBERS:

1049424 - KABEYI EUGENE MATHEWS

1050902 - DENG PAUL GARANG

1062635 - MWAURA NJERI

1049400 - SILAH RYAN THINDI

1048775 - DERRICK MURITHI

Description

The proposed system is an integrated Medical Management System that would be able to integrate the processes of customer management, drug manufacturing, prescription handling, sales, purchase, and supply chain management within any medical environment. It integrates main stakeholders, such as customers, doctors, distributors, employees, and manufacturers of drugs for an integrated approach to the effective management of medical resources.

Overview

The various modules of the system are interconnected as follows:

Customer Management: This module stores and manages patient details.

Tracking of Prescriptions: The prescription made and the patient requirement accordingly.

Medi-Drug Inventory: Drug stock, cost, expiry, and discount that are continuously updated.

Sale/Purchase: Sales and purchase transactions are input for proper and transparent transaction processing.

Supply Chain: Product tracking between distributor-manufacturer.

Employee Management: Consolidated employee information that is involved in various transactions.

The system uses a relational database to ensure data integrity and efficient data cross-referencing between modules.

Goals

Centralized Data Management: Ensure that the customers, doctors, and distributors operate on the same network.

Real-Time Inventory Tracking: Maintain drug stock, expiry date, and discounts to minimize wastage of drugs and ensure timely stock replacements.

Improved Customer Care: Accurately record patient and prescription information for better and more personalized care.

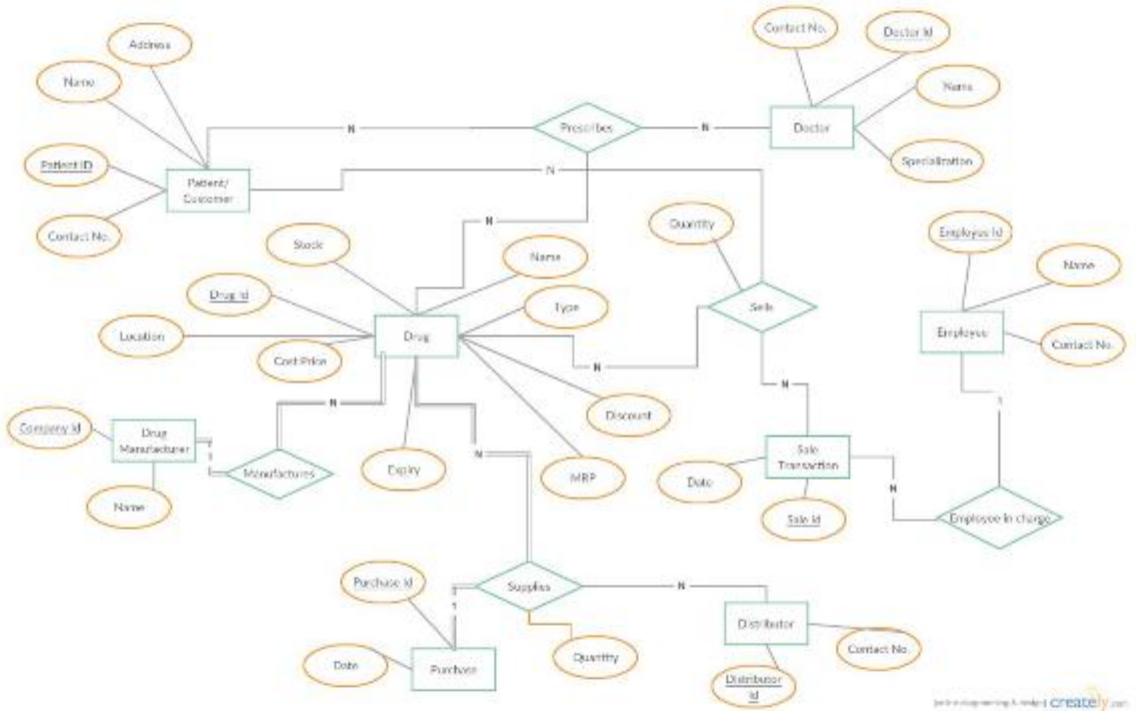
Smooth Transactions: Automate and simplify the buying-selling process to reduce manual effort and minimize errors.

Regulatory Compliance: Ensure that all prescriptions, drugs, and transactions meet industrial regulations and standards.

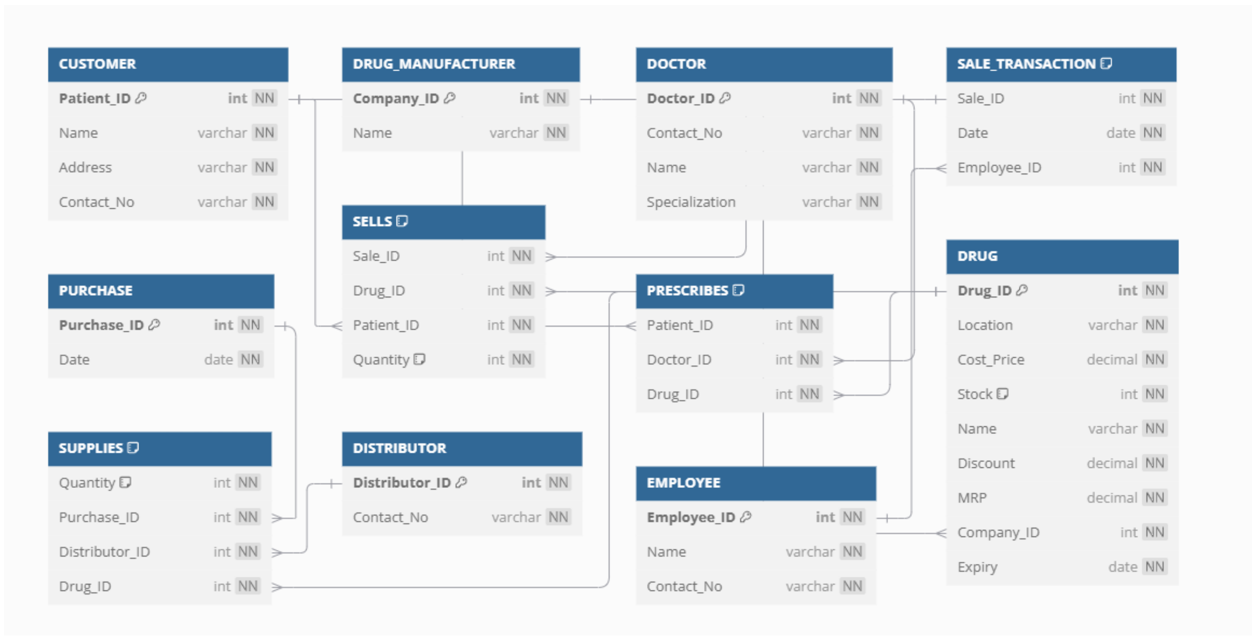
Seamless Integration: Allow for smooth coordination between manufacturers, distributors, and medical institutions.

Scalability and Adaptability: Design the system to adapt to future needs and expand to accommodate additional features or higher user loads.

ER DIAGRAM



SQL SCHEMA



CUSTOMER

Patient_ID	int	NN
Name	varchar	NN
Address	varchar	NN
Contact_No	varchar	NN

DRUG_MANUFACTURER

Company_ID	int	NN
Name	varchar	NN

DOCTOR

Doctor_ID	int	NN
Contact_No	varchar	NN
Name	varchar	NN
Specialization	varchar	NN

SALE_TRANSACTION

Sale_ID	int	NN
Date	date	NN
Employee_ID	int	NN

SELLS

Sale_ID	int	NN
Drug_ID	int	NN
Patient_ID	int	NN
Quantity	int	NN

PURCHASE

Purchase_ID	int	NN
Date	date	NN

PRESCRIBES

Patient_ID	int	NN
Doctor_ID	int	NN
Drug_ID	int	NN

DRUG

Drug_ID	int	NN
Location	varchar	NN
Cost_Price	decimal	NN
Stock	int	NN
Name	varchar	NN
Discount	decimal	NN
MRP	decimal	NN
Company_ID	int	NN
Expiry	date	NN

SUPPLIES

Quantity	int	NN
Purchase_ID	int	NN
Distributor_ID	int	NN
Drug_ID	int	NN

DISTRIBUTOR

Distributor_ID	int	NN
Contact_No	varchar	NN

EMPLOYEE

Employee_ID	int	NN
Name	varchar	NN
Contact_No	varchar	NN

CREATING DATABASE

```
CREATE DATABASE MedicalManagement;

USE MedicalManagement;
```

CREATING TABLES

1. CUSTOMER Table

```
CREATE TABLE CUSTOMER (
    Patient_ID INT PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Address VARCHAR(255) NOT NULL,
    Contact_No VARCHAR(15) NOT NULL
);
```

2. DRUG_MANUFACTURER Table

```
CREATE TABLE DRUG_MANUFACTURER (
    Company_ID INT PRIMARY KEY,
    Name VARCHAR(255) NOT NULL
);
```

3. DOCTOR Table

```
CREATE TABLE DOCTOR (
Doctor_ID INT PRIMARY KEY,
    Contact_No VARCHAR(15) NOT NULL,
    Name VARCHAR(255) NOT NULL,
    Specialization VARCHAR(255) NOT NULL
);
```

4. SALE_TRANSACTION Table

```
CREATE TABLE SALE_TRANSACTION (
    Sale_ID INT NOT NULL,
    Date DATE NOT NULL,
    Employee_ID INT NOT NULL,
    PRIMARY KEY (Sale_ID, Employee_ID),
    FOREIGN KEY (Employee_ID) REFERENCES EMPLOYEE(Employee_ID)
);
```

5. PURCHASE Table

```
CREATE TABLE PURCHASE (
    Purchase_ID INT PRIMARY KEY,
    Date DATE NOT NULL
);
```

6. SELLS Table

```
CREATE TABLE SELLS (
    Sale_ID INT NOT NULL,
    Drug_ID INT NOT NULL,
    Patient_ID INT NOT NULL,
    Quantity INT NOT NULL CHECK (Quantity > 0),
    PRIMARY KEY (Sale_ID, Drug_ID, Patient_ID),
    FOREIGN KEY (Sale_ID) REFERENCES SALE_TRANSACTION(Sale_ID),
    FOREIGN KEY (Drug_ID) REFERENCES DRUG(Drug_ID),
    FOREIGN KEY (Patient_ID) REFERENCES CUSTOMER(Patient_ID)
);
```

7. PRESCRIBES Table

```
CREATE TABLE PRESCRIBES (
    Patient_ID INT NOT NULL,
    Doctor_ID INT NOT NULL,
    Drug_ID INT NOT NULL,
    PRIMARY KEY (Patient_ID, Doctor_ID, Drug_ID),
    FOREIGN KEY (Patient_ID) REFERENCES CUSTOMER(Patient_ID),
    FOREIGN KEY (Doctor_ID) REFERENCES DOCTOR(Doctor_ID),
    FOREIGN KEY (Drug_ID) REFERENCES DRUG(Drug_ID)
);
```

8. DRUG Table

```
CREATE TABLE DRUG (
    Drug_ID INT PRIMARY KEY,
    Location VARCHAR(255) NOT NULL,
    Cost_Price DECIMAL(10, 2) NOT NULL,
    Stock INT NOT NULL CHECK (Stock >= 0),
    Name VARCHAR(255) NOT NULL,
    Discount DECIMAL(5, 2) NOT NULL,
    MRP DECIMAL(10, 2) NOT NULL,
    Company_ID INT NOT NULL,
    Expiry DATE NOT NULL,
    FOREIGN KEY (Company_ID) REFERENCES DRUG_MANUFACTURER(Company_ID)
);
```

9. SUPPLIES Table

```
CREATE TABLE SUPPLIES (
    Quantity INT NOT NULL CHECK (Quantity > 0),
    Purchase_ID INT NOT NULL,
    Distributor_ID INT NOT NULL,
    Drug_ID INT NOT NULL,
    PRIMARY KEY (Distributor_ID, Drug_ID, Purchase_ID),
    FOREIGN KEY (Purchase_ID) REFERENCES PURCHASE(Purchase_ID),
    FOREIGN KEY (Distributor_ID) REFERENCES DISTRIBUTOR(Distributor_ID),
    FOREIGN KEY (Drug_ID) REFERENCES DRUG(Drug_ID)
);
```

10. DISTRIBUTOR Table

```
CREATE TABLE DISTRIBUTOR (
    Distributor_ID INT PRIMARY KEY,
    Contact_No VARCHAR(15) NOT NULL
);
```

11. EMPLOYEE Table

```
CREATE TABLE EMPLOYEE (
    Employee_ID INT PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Contact_No VARCHAR(15) NOT NULL
);
```

<u>CRUD OPERATIONS</u>

1. **CUSTOMER**

```
-- Create (Insert)
INSERT INTO CUSTOMER (Patient_ID, Name, Address, Contact_No) VALUES
(1, 'John Doe', '123 Elm Street', '1234567890'),
(2, 'Jane Smith', '456 Oak Avenue', '0987654321'),
(3, 'Alice Johnson', '789 Pine Road', '1122334455'),
(4, 'Bob Brown', '321 Maple Lane', '2233445566'),
(5, 'Emily Davis', '654 Cedar Drive', '3344556677');

-- Read (Select)
SELECT * FROM CUSTOMER;

-- Update
UPDATE CUSTOMER
SET Address = '555 Birch Blvd'
WHERE Patient_ID = 3;

-- Delete
DELETE FROM CUSTOMER WHERE Patient_ID = 5;
```

2. **DRUG_MANUFACTURER**

```
-- Create (Insert)
INSERT INTO DRUG_MANUFACTURER (Company_ID, Name) VALUES
(1, 'PharmaCorp'),
(2, 'MediLife'),
(3, 'Wellness Inc'),
(4, 'BioHealth'),
(5, 'CareMeds');

-- Read (Select)
SELECT * FROM DRUG_MANUFACTURER;

-- Update
UPDATE DRUG_MANUFACTURER
SET Name = 'Global Pharma'
WHERE Company_ID = 2;

-- Delete
DELETE FROM DRUG_MANUFACTURER WHERE Company_ID = 5;
```

3. DOCTOR

```
-- Create (Insert)
INSERT INTO DOCTOR (Doctor_ID, Contact_No, Name, Specialization) VALUES
(1, '9876543210', 'Dr. Sarah Lee', 'Cardiology'),
(2, '8765432109', 'Dr. Michael Chen', 'Neurology'),
(3, '7654321098', 'Dr. Laura King', 'Pediatrics'),
(4, '6543210987', 'Dr. Kevin White', 'Orthopedics'),
(5, '5432109876', 'Dr. Emily Adams', 'Dermatology');

-- Read (Select)
SELECT * FROM DOCTOR;

-- Update
UPDATE DOCTOR
SET Specialization = 'General Medicine'
WHERE Doctor_ID = 4;

-- Delete
DELETE FROM DOCTOR WHERE Doctor_ID = 5;
```

4. SALE_TRANSACTION

```
-- Create (Insert)
INSERT INTO SALE_TRANSACTION (Sale_ID, Date, Employee_ID) VALUES
(1, '2024-11-01', 101),
(2, '2024-11-02', 102),
(3, '2024-11-03', 103),
(4, '2024-11-04', 104),
(5, '2024-11-05', 105);
```

-- Read (Select)
*SELECT * FROM SALE_TRANSACTION;*

-- Update
UPDATE SALE_TRANSACTION
SET Date = '2024-11-10'
WHERE Sale_ID = 3;

-- Delete
DELETE FROM SALE_TRANSACTION WHERE Sale_ID = 5;

5. PURCHASE

-- Create (Insert)
INSERT INTO PURCHASE (Purchase_ID, Date) VALUES
(1, '2024-11-01'),
(2, '2024-11-02'),
(3, '2024-11-03'),
(4, '2024-11-04'),
(5, '2024-11-05');

-- Read (Select)
*SELECT * FROM PURCHASE;*

-- Update
UPDATE PURCHASE
SET Date = '2024-12-01'
WHERE Purchase_ID = 2;

-- Delete
DELETE FROM PURCHASE WHERE Purchase_ID = 5;

6. DRUG

-- Create (Insert)
INSERT INTO DRUG (Drug_ID, Location, Cost_Price, Stock, Name, Discount, MRP, Company_ID, Expiry) VALUES
(1, 'Aisle 1', 10.50, 100, 'Paracetamol', 5, 12.00, 1, '2025-01-01'),
(2, 'Aisle 2', 15.00, 200, 'Ibuprofen', 10, 18.00, 2, '2025-06-01'),
(3, 'Aisle 3', 8.00, 150, 'Cough Syrup', 7, 10.00, 3, '2025-03-01'),
(4, 'Aisle 4', 20.00, 80, 'Antibiotics', 5, 25.00, 4, '2025-08-01'),
(5, 'Aisle 5', 30.00, 50, 'Vitamin C', 0, 35.00, 5, '2025-10-01');

-- Read (Select)
*SELECT * FROM DRUG;*

```
-- Update
UPDATE DRUG
SET Stock = 120
WHERE Drug_ID = 3;

-- Delete
DELETE FROM DRUG WHERE Drug_ID = 5;
```

7. DISTRIBUTOR

```
-- Create (Insert)
INSERT INTO DISTRIBUTOR (Distributor_ID, Contact_No) VALUES
(1, '1234567890'),
(2, '0987654321'),
(3, '1122334455'),
(4, '2233445566'),
(5, '3344556677');

-- Read (Select)
SELECT * FROM DISTRIBUTOR;

-- Update
UPDATE DISTRIBUTOR
SET Contact_No = '9998887777'
WHERE Distributor_ID = 3;

-- Delete
DELETE FROM DISTRIBUTOR WHERE Distributor_ID = 5;
```

8. EMPLOYEE

```
-- Create (Insert)
INSERT INTO EMPLOYEE (Employee_ID, Name, Contact_No) VALUES
(101, 'Alice Walker', '9876543210'),
(102, 'Bob Martin', '8765432109'),
(103, 'Charlie Young', '7654321098'),
(104, 'Diana Prince', '6543210987'),
(105, 'Evan Stone', '5432109876');

-- Read (Select)
SELECT * FROM EMPLOYEE;

-- Update
UPDATE EMPLOYEE
```

```
SET Name = 'Ethan Stone'
WHERE Employee_ID = 105;

-- Delete
DELETE FROM EMPLOYEE WHERE Employee_ID = 105;
```

9. SELLS

```
S -- Create (Insert)
INSERT INTO SELLS (Sale_ID, Drug_ID, Patient_ID, Quantity) VALUES
(1, 1, 1, 2),
(2, 2, 2, 5),
(3, 3, 3, 3),
(4, 4, 4, 1),
(5, 5, 1, 4);

-- Read (Select)
SELECT * FROM SELLS;

-- Update
UPDATE SELLS
SET Quantity = 6
WHERE Sale_ID = 2 AND Drug_ID = 2 AND Patient_ID = 2;

-- Delete
DELETE FROM SELLS
WHERE Sale_ID = 5 AND Drug_ID = 5 AND Patient_ID = 1;
```

10. PRESCRIBES

```
-- Create (Insert)
INSERT INTO PRESCRIBES (Patient_ID, Doctor_ID, Drug_ID) VALUES
(1, 1, 1),
(2, 2, 2),
(3, 3, 3),
(4, 4, 4),
(5, 5, 5);

-- Read (Select)
SELECT * FROM PRESCRIBES;

-- Update
UPDATE PRESCRIBES
SET Drug_ID = 4
WHERE Patient_ID = 3 AND Doctor_ID = 3 AND Drug_ID = 3;
```

```
-- Delete
DELETE FROM PRESCRIBES
WHERE Patient_ID = 5 AND Doctor_ID = 5 AND Drug_ID = 5;
```

11. SUPPLIES

```
-- Create (Insert)
INSERT INTO SUPPLIES (Quantity, Purchase_ID, Distributor_ID, Drug_ID) VALUES
(100, 1, 1, 1),
(200, 2, 2, 2),
(150, 3, 3, 3),
(80, 4, 4, 4),
(50, 5, 5, 5);

-- Read (Select)
SELECT * FROM SUPPLIES;

-- Update
UPDATE SUPPLIES
SET Quantity = 250
WHERE Purchase_ID = 2 AND Distributor_ID = 2 AND Drug_ID = 2;

-- Delete
DELETE FROM SUPPLIES
WHERE Purchase_ID = 5 AND Distributor_ID = 5 AND Drug_ID = 5;
```

STORED PROCEDURES

1. **CUSTOMER**

```
-- Create Procedure for INSERT
DELIMITER //
CREATE PROCEDURE AddCustomer(IN p_Name VARCHAR(255), IN
p_Address VARCHAR(255), IN p_Contact_No VARCHAR(20))
BEGIN
    INSERT INTO CUSTOMER (Name, Address, Contact_No) VALUES
(p_Name, p_Address, p_Contact_No);
END //
DELIMITER ;

-- Read Procedure
DELIMITER //
CREATE PROCEDURE GetCustomers()
BEGIN
```

```
    SELECT * FROM CUSTOMER;
END //
DELIMITER ;

-- Update Procedure
DELIMITER //
CREATE PROCEDURE UpdateCustomer(IN p_Patient_ID INT, IN
p_Address VARCHAR(255))
BEGIN
    UPDATE CUSTOMER SET Address = p_Address WHERE Patient_ID =
p_Patient_ID;
END //
DELIMITER ;

-- Delete Procedure
DELIMITER //
CREATE PROCEDURE DeleteCustomer(IN p_Patient_ID INT)
BEGIN
    DELETE FROM CUSTOMER WHERE Patient_ID = p_Patient_ID;
END //
DELIMITER ;
```

2. DRUG_MANUFACTURER

```
-- Create Procedure for INSERT

DELIMITER //

CREATE PROCEDURE AddDrugManufacturer(IN p_Name VARCHAR(255))

BEGIN

    INSERT INTO DRUG_MANUFACTURER (Name) VALUES (p_Name);

END //

DELIMITER ;


-- Read Procedure

DELIMITER //

CREATE PROCEDURE GetDrugManufacturers()

BEGIN

    SELECT * FROM DRUG_MANUFACTURER;
```

```
END //
DELIMITER ;

-- Update Procedure
DELIMITER //
CREATE PROCEDURE UpdateDrugManufacturer(IN p_Company_ID INT, IN p_Name VARCHAR(255))
BEGIN
    UPDATE DRUG_MANUFACTURER SET Name = p_Name WHERE Company_ID = p_Company_ID;
END //
DELIMITER ;

-- Delete Procedure
DELIMITER //
CREATE PROCEDURE DeleteDrugManufacturer(IN p_Company_ID INT)
BEGIN
    DELETE FROM DRUG_MANUFACTURER WHERE Company_ID = p_Company_ID;
END //
DELIMITER ;
```

3. DOCTOR

```
-- Create Procedure for INSERT
DELIMITER //
CREATE PROCEDURE AddDoctor(IN p_Name VARCHAR(255), IN p_Contact_No VARCHAR(20), IN p_Specialization VARCHAR(255))
BEGIN
    INSERT INTO DOCTOR (Name, Contact_No, Specialization) VALUES (p_Name, p_Contact_No, p_Specialization);
END //
```

```
DELIMITER ;

-- Read Procedure
DELIMITER //
CREATE PROCEDURE GetDoctors()
BEGIN
    SELECT * FROM DOCTOR;
END //
DELIMITER ;

-- Update Procedure
DELIMITER //
CREATE PROCEDURE UpdateDoctor(IN p_Doctor_ID INT, IN p_Specialization
VARCHAR(255))
BEGIN
    UPDATE DOCTOR SET Specialization = p_Specialization WHERE Doctor_ID =
p_Doctor_ID;
END //
DELIMITER ;

-- Delete Procedure
DELIMITER //
CREATE PROCEDURE DeleteDoctor(IN p_Doctor_ID INT)
BEGIN
    DELETE FROM DOCTOR WHERE Doctor_ID = p_Doctor_ID;
END //
DELIMITER ;
```

4. SALE_TRANSACTION

```
-- Create Procedure for INSERT
```

```
DELIMITER //
CREATE PROCEDURE AddSaleTransaction(IN p_Date DATE, IN p_Employee_ID INT)
BEGIN
    INSERT INTO SALE_TRANSACTION (Date, Employee_ID) VALUES (p_Date, p_Employee_ID);
END //
DELIMITER ;

-- Read Procedure
DELIMITER //
CREATE PROCEDURE GetSaleTransactions()
BEGIN
    SELECT * FROM SALE_TRANSACTION;
END //
DELIMITER ;

-- Update Procedure
DELIMITER //
CREATE PROCEDURE UpdateSaleTransaction(IN p_Sale_ID INT, IN p_Date DATE)
BEGIN
    UPDATE SALE_TRANSACTION SET Date = p_Date WHERE Sale_ID = p_Sale_ID;
END //
DELIMITER ;

-- Delete Procedure
DELIMITER //
CREATE PROCEDURE DeleteSaleTransaction(IN p_Sale_ID INT)
BEGIN
    DELETE FROM SALE_TRANSACTION WHERE Sale_ID = p_Sale_ID;
```

```
END //
DELIMITER ;
```

5. PURCHASE

```
-- Create Procedure for INSERT
DELIMITER //
CREATE PROCEDURE AddPurchase(IN p_Date DATE)
BEGIN
    INSERT INTO PURCHASE (Date) VALUES (p_Date);
END //
DELIMITER ;

-- Read Procedure
DELIMITER //
CREATE PROCEDURE GetPurchases()
BEGIN
    SELECT * FROM PURCHASE;
END //
DELIMITER ;

-- Update Procedure
DELIMITER //
CREATE PROCEDURE UpdatePurchase(IN p_Purchase_ID INT, IN p_Date DATE)
BEGIN
    UPDATE PURCHASE SET Date = p_Date WHERE Purchase_ID = p_Purchase_ID;
END //
DELIMITER ;

-- Delete Procedure
```

```
DELIMITER //

CREATE PROCEDURE DeletePurchase(IN p_Purchase_ID INT)

BEGIN

    DELETE FROM PURCHASE WHERE Purchase_ID = p_Purchase_ID;

END //

DELIMITER ;
```

6. DRUG

```
-- Create Procedure for INSERT

DELIMITER //

CREATE PROCEDURE AddDrug(IN p_Location VARCHAR(255), IN p_Cost_Price
DECIMAL(10,2), IN p_Stock INT, IN p_Name VARCHAR(255), IN p_Discount DECIMAL(5,2),
IN p_MRP DECIMAL(10,2), IN p_Company_ID INT, IN p_Expiry DATE)

BEGIN

    INSERT INTO DRUG (Location, Cost_Price, Stock, Name, Discount, MRP, Company_ID,
Expiry)

    VALUES (p_Location, p_Cost_Price, p_Stock, p_Name, p_Discount, p_MRP,
p_Company_ID, p_Expiry);

END //

DELIMITER ;


-- Read Procedure

DELIMITER //

CREATE PROCEDURE GetDrugs()

BEGIN

    SELECT * FROM DRUG;

END //

DELIMITER ;


-- Update Procedure

DELIMITER //
```

```
CREATE PROCEDURE UpdateDrug(IN p_Drug_ID INT, IN p_Stock INT)
BEGIN
    UPDATE DRUG SET Stock = p_Stock WHERE Drug_ID = p_Drug_ID;
END //
DELIMITER ;

-- Delete Procedure
DELIMITER //
CREATE PROCEDURE DeleteDrug(IN p_Drug_ID INT)
BEGIN
    DELETE FROM DRUG WHERE Drug_ID = p_Drug_ID;
END //
DELIMITER ;
```

7. SELLS

```
-- Create Procedure for INSERT
DELIMITER //
CREATE PROCEDURE AddSell(IN p_Sale_ID INT, IN p_Drug_ID INT, IN p_Patient_ID INT,
IN p_Quantity INT)
BEGIN
    INSERT INTO SELLS (Sale_ID, Drug_ID, Patient_ID, Quantity)
    VALUES (p_Sale_ID, p_Drug_ID, p_Patient_ID, p_Quantity);
END //
DELIMITER ;

-- Read Procedure
DELIMITER //
CREATE PROCEDURE GetSells()
BEGIN
    SELECT * FROM SELLS;
```

END //

DELIMITER ;

-- Update Procedure

DELIMITER //

CREATE PROCEDURE UpdateSell(IN p_Sale_ID INT, IN p_Drug_ID INT, IN p_Patient_ID INT, IN p_Quantity INT)

BEGIN

UPDATE SELLS SET Quantity = p_Quantity WHERE Sale_ID = p_Sale_ID AND Drug_ID = p_Drug_ID AND Patient_ID = p_Patient_ID;

END //

DELIMITER ;

-- Delete Procedure

DELIMITER //

CREATE PROCEDURE DeleteSell(IN p_Sale_ID INT, IN p_Drug_ID INT, IN p_Patient_ID INT)

BEGIN

DELETE FROM SELLS WHERE Sale_ID = p_Sale_ID AND Drug_ID = p_Drug_ID AND Patient_ID = p_Patient_ID;

END //

DELIMITER ;

8. PRESCRIBES

-- Create Procedure for INSERT

DELIMITER //

CREATE PROCEDURE AddPrescription(IN p_Patient_ID INT, IN p_Doctor_ID INT, IN p_Drug_ID INT)

BEGIN

INSERT INTO PRESCRIBES (Patient_ID, Doctor_ID, Drug_ID)

VALUES (p_Patient_ID, p_Doctor_ID, p_Drug_ID);

```
END //
DELIMITER ;

-- Read Procedure
DELIMITER //
CREATE PROCEDURE GetPrescriptions()
BEGIN
    SELECT * FROM PRESCRIBES;
END //
DELIMITER ;

-- Update Procedure
DELIMITER //
CREATE PROCEDURE UpdatePrescription(IN p_Patient_ID INT, IN p_Doctor_ID INT, IN
p_Drug_ID INT)
BEGIN
    UPDATE PRESCRIBES SET Drug_ID = p_Drug_ID WHERE Patient_ID = p_Patient_ID
AND Doctor_ID = p_Doctor_ID;
END //
DELIMITER ;

-- Delete Procedure
DELIMITER //
CREATE PROCEDURE DeletePrescription(IN p_Patient_ID INT, IN p_Doctor_ID INT, IN
p_Drug_ID INT)
BEGIN
    DELETE FROM PRESCRIBES WHERE Patient_ID = p_Patient_ID AND Doctor_ID =
p_Doctor_ID AND Drug_ID = p_Drug_ID;
END //
DELIMITER ;
```

9. SUPPLIES

```
-- Create Procedure for INSERT

DELIMITER //

CREATE PROCEDURE AddSupply(IN p_Quantity INT, IN p_Purchase_ID INT, IN
p_Distributor_ID INT, IN p_Drug_ID INT)

BEGIN

    INSERT INTO SUPPLIES (Quantity, Purchase_ID, Distributor_ID, Drug_ID)

    VALUES (p_Quantity, p_Purchase_ID, p_Distributor_ID, p_Drug_ID);

END //

DELIMITER ;


-- Read Procedure

DELIMITER //

CREATE PROCEDURE GetSupplies()

BEGIN

    SELECT * FROM SUPPLIES;

END //

DELIMITER ;


-- Update Procedure

DELIMITER //

CREATE PROCEDURE UpdateSupply(IN p_Purchase_ID INT, IN p_Distributor_ID INT, IN
p_Drug_ID INT, IN p_Quantity INT)

BEGIN

    UPDATE SUPPLIES SET Quantity = p_Quantity WHERE Purchase_ID = p_Purchase_ID
AND Distributor_ID = p_Distributor_ID AND Drug_ID = p_Drug_ID;

END //

DELIMITER ;
```

```
-- Delete Procedure
DELIMITER //
CREATE PROCEDURE DeleteSupply(IN p_Purchase_ID INT, IN p_Distributor_ID INT, IN
p_Drug_ID INT)
BEGIN
    DELETE FROM SUPPLIES WHERE Purchase_ID = p_Purchase_ID AND Distributor_ID =
p_Distributor_ID AND Drug_ID = p_Drug_ID;
END //
DELIMITER ;
```

10. DISTRIBUTOR

```
-- Create Procedure for INSERT
DELIMITER //
CREATE PROCEDURE AddDistributor(IN p_Contact_No VARCHAR(20))
BEGIN
    INSERT INTO DISTRIBUTOR (Contact_No) VALUES (p_Contact_No);
END //
DELIMITER ;


-- Read Procedure
DELIMITER //
CREATE PROCEDURE GetDistributors()
BEGIN
    SELECT * FROM DISTRIBUTOR;
END //
DELIMITER ;


-- Update Procedure
DELIMITER //
```

CREATE PROCEDURE UpdateDistributor(IN p_Distributor_ID INT, IN p_Contact_No VARCHAR(20))

BEGIN

UPDATE DISTRIBUTOR SET Contact_No = p_Contact_No WHERE Distributor_ID = p_Distributor_ID;

END //

DELIMITER ;

-- Delete Procedure

DELIMITER //

CREATE PROCEDURE DeleteDistributor(IN p_Distributor_ID INT)

BEGIN

DELETE FROM DISTRIBUTOR WHERE Distributor_ID = p_Distributor_ID;

END //

DELIMITER ;

11. EMPLOYEE

-- Create Procedure for INSERT

DELIMITER //

CREATE PROCEDURE AddEmployee(IN p_Name VARCHAR(255), IN p_Contact_No VARCHAR(20))

BEGIN

INSERT INTO EMPLOYEE (Name, Contact_No) VALUES (p_Name, p_Contact_No);

END //

DELIMITER ;

-- Read Procedure

DELIMITER //

CREATE PROCEDURE GetEmployees()

BEGIN

```
    SELECT * FROM EMPLOYEE;
END //
DELIMITER ;

-- Update Procedure
DELIMITER //
CREATE PROCEDURE UpdateEmployee(IN p_Employee_ID INT, IN p_Name
VARCHAR(255), IN p_Contact_No VARCHAR(20))
BEGIN
    UPDATE EMPLOYEE SET Name = p_Name, Contact_No = p_Contact_No WHERE
Employee_ID = p_Employee_ID;
END //
DELIMITER ;

-- Delete Procedure
DELIMITER //
CREATE PROCEDURE DeleteEmployee(IN p_Employee_ID INT)
BEGIN
    DELETE FROM EMPLOYEE WHERE Employee_ID = p_Employee_ID;
END //
DELIMITER ;
```

Calling Procedures:

```
CALL AddCustomer('Jane Doe', '123 Pine St', '9876543210');
CALL GetCustomers();
CALL UpdateCustomer(1, '456 Oak Ave');
CALL DeleteCustomer(1);
```

TRIGGERS

1. CUSTOMER

```
-- Trigger on INSERT: Ensures the customer’s contact number is not empty
DELIMITER //
CREATE TRIGGER BeforeInsertCustomer
BEFORE INSERT ON CUSTOMER
FOR EACH ROW
BEGIN
   IF NEW.Contact_No IS NULL THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Contact No cannot be NULL';
   END IF;
END //
DELIMITER ;

-- Trigger on UPDATE: Ensures the address is not empty when updating
DELIMITER //
CREATE TRIGGER BeforeUpdateCustomer
BEFORE UPDATE ON CUSTOMER
FOR EACH ROW
BEGIN
   IF NEW.Address IS NULL THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Address cannot be NULL';
   END IF;
END //
DELIMITER ;

-- Trigger on DELETE: Ensures no customer can be deleted if they have related sales
DELIMITER //
```

```
CREATE TRIGGER BeforeDeleteCustomer
BEFORE DELETE ON CUSTOMER
FOR EACH ROW
BEGIN
    DECLARE customer_exists INT;
    SELECT COUNT(*) INTO customer_exists
    FROM SELLS
    WHERE Patient_ID = OLD.Patient_ID;
    IF customer_exists > 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot delete customer with
existing sales records';
    END IF;
END //
DELIMITER ;
```

2. DRUG_MANUFACTURER

```
-- Trigger on INSERT: Ensures the manufacturer name is not empty
DELIMITER //
CREATE TRIGGER BeforeInsertDrugManufacturer
BEFORE INSERT ON DRUG_MANUFACTURER
FOR EACH ROW
BEGIN
    IF NEW.Name IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Manufacturer name cannot be
NULL';
    END IF;
END //
DELIMITER ;


-- Trigger on UPDATE: Ensures the name is not updated to NULL
```

```
DELIMITER //
CREATE TRIGGER BeforeUpdateDrugManufacturer
BEFORE UPDATE ON DRUG_MANUFACTURER
FOR EACH ROW
BEGIN
   IF NEW.Name IS NULL THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Manufacturer name cannot be
NULL';
   END IF;
END //
DELIMITER ;

-- Trigger on DELETE: Prevents deleting a manufacturer if drugs are still associated
DELIMITER //
CREATE TRIGGER BeforeDeleteDrugManufacturer
BEFORE DELETE ON DRUG_MANUFACTURER
FOR EACH ROW
BEGIN
   DECLARE manufacturer_exists INT;
   SELECT COUNT(*) INTO manufacturer_exists
   FROM DRUG
   WHERE Company_ID = OLD.Company_ID;
   IF manufacturer_exists > 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot delete manufacturer with
existing drugs';
   END IF;
END //
DELIMITER ;
```

3. DOCTOR

```
-- Trigger on INSERT: Ensures doctor's contact number is not empty
DELIMITER //
CREATE TRIGGER BeforeInsertDoctor
BEFORE INSERT ON DOCTOR
FOR EACH ROW
BEGIN
    IF NEW.Contact_No IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Doctor contact number cannot be
NULL';
    END IF;
END //
DELIMITER ;

-- Trigger on UPDATE: Ensures specialization is not NULL
DELIMITER //
CREATE TRIGGER BeforeUpdateDoctor
BEFORE UPDATE ON DOCTOR
FOR EACH ROW
BEGIN
    IF NEW.Specialization IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Doctor specialization cannot be
NULL';
    END IF;
END //
DELIMITER ;

-- Trigger on DELETE: Prevents deletion of doctor if they have related prescriptions
```

```
DELIMITER //
CREATE TRIGGER BeforeDeleteDoctor
BEFORE DELETE ON DOCTOR
FOR EACH ROW
BEGIN
    DECLARE doctor_exists INT;
    SELECT COUNT(*) INTO doctor_exists
    FROM PRESCRIBES
    WHERE Doctor_ID = OLD.Doctor_ID;
    IF doctor_exists > 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot delete doctor with existing
prescriptions';
    END IF;
END //
DELIMITER ;
```

4. SALE_TRANSACTION

```
-- Trigger on INSERT: Ensures employee ID is valid
DELIMITER //
CREATE TRIGGER BeforeInsertSaleTransaction
BEFORE INSERT ON SALE_TRANSACTION
FOR EACH ROW
BEGIN
    DECLARE employee_exists INT;
    SELECT COUNT(*) INTO employee_exists
    FROM EMPLOYEE
    WHERE Employee_ID = NEW.Employee_ID;
    IF employee_exists = 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid Employee ID';
    END IF;
END //
DELIMITER ;

-- Trigger on DELETE: Prevents deletion of sale transactions if they are linked to sales
DELIMITER //
CREATE TRIGGER BeforeDeleteSaleTransaction
BEFORE DELETE ON SALE_TRANSACTION
FOR EACH ROW
BEGIN
    DECLARE sale_exists INT;
    SELECT COUNT(*) INTO sale_exists
    FROM SELLS
    WHERE Sale_ID = OLD.Sale_ID;
    IF sale_exists > 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot delete sale transaction with existing sales';
    END IF;
END //
DELIMITER ;
```

5. PURCHASE

```
-- Trigger on INSERT: Ensures the purchase date is not in the future
DELIMITER //
CREATE TRIGGER BeforeInsertPurchase
BEFORE INSERT ON PURCHASE
FOR EACH ROW
BEGIN
    IF NEW.Date > CURDATE() THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Purchase date cannot be in the future';
    END IF;
END //
DELIMITER ;

-- Trigger on DELETE: Ensures a purchase cannot be deleted if related to supplies
DELIMITER //
CREATE TRIGGER BeforeDeletePurchase
BEFORE DELETE ON PURCHASE
FOR EACH ROW
BEGIN
    DECLARE purchase_exists INT;
    SELECT COUNT(*) INTO purchase_exists
    FROM SUPPLIES
    WHERE Purchase_ID = OLD.Purchase_ID;
    IF purchase_exists > 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot delete purchase with
existing supplies';
    END IF;
END //
DELIMITER ;
```

6. DRUG

```
-- Trigger on INSERT: Ensures drug stock is not negative
DELIMITER //
CREATE TRIGGER BeforeInsertDrug
BEFORE INSERT ON DRUG
FOR EACH ROW
BEGIN
    IF NEW.Stock < 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Drug stock cannot be negative';
    END IF;
END //
DELIMITER ;

-- Trigger on UPDATE: Ensures stock is not negative
DELIMITER //
CREATE TRIGGER BeforeUpdateDrug
BEFORE UPDATE ON DRUG
FOR EACH ROW
BEGIN
    IF NEW.Stock < 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Drug stock cannot be negative';
    END IF;
END //
DELIMITER ;

-- Trigger on DELETE: Prevents deletion if drug is associated with supplies
DELIMITER //
CREATE TRIGGER BeforeDeleteDrug
BEFORE DELETE ON DRUG
FOR EACH ROW
BEGIN
    DECLARE drug_exists INT;
    SELECT COUNT(*) INTO drug_exists
    FROM SUPPLIES
    WHERE Drug_ID = OLD.Drug_ID;
    IF drug_exists > 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot delete drug with existing
supplies';
```

```
    END IF;
END //
DELIMITER ;
```

7. SELLS

```
-- Trigger on INSERT: Ensures quantity is positive
DELIMITER //
CREATE TRIGGER BeforeInsertSell
BEFORE INSERT ON SELLS
FOR EACH ROW
BEGIN
    IF NEW.Quantity <= 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Quantity cannot be zero or negative';
    END IF;
END //
DELIMITER ;

-- Trigger on UPDATE: Ensures quantity is positive
DELIMITER //
CREATE TRIGGER BeforeUpdateSell
BEFORE UPDATE ON SELLS
FOR EACH ROW
BEGIN
    IF NEW.Quantity <= 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Quantity cannot be zero or negative';
    END IF;
END //
DELIMITER ;
```

8. PRESCRIBES

```
-- Trigger on INSERT: Ensures that the drug prescribed is in stock
DELIMITER //
CREATE TRIGGER BeforeInsertPrescription
BEFORE INSERT ON PRESCRIBES
FOR EACH ROW
BEGIN
    DECLARE drug_in_stock INT;
    SELECT Stock INTO drug_in_stock
    FROM DRUG
    WHERE Drug_ID = NEW.Drug_ID;
    IF drug_in_stock <= 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Drug is out of stock';
    END IF;
END //
DELIMITER ;
```

9. SUPPLIES

```
-- Trigger on INSERT: Ensures supply quantity is not negative
DELIMITER //
CREATE TRIGGER BeforeInsertSupply
BEFORE INSERT ON SUPPLIES
FOR EACH ROW
BEGIN
    IF NEW.Quantity <= 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Supply quantity cannot be zero or
negative';
    END IF;
END //
```

```
DELIMITER ;


-- Trigger on UPDATE: Ensures supply quantity is not negative
DELIMITER //
CREATE TRIGGER BeforeUpdateSupply
BEFORE UPDATE ON SUPPLIES
FOR EACH ROW
BEGIN
   IF NEW.Quantity <= 0 THEN
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Supply quantity cannot be zero or negative';
   END IF;
END //
DELIMITER ;
```

10. DISTRIBUTOR

```
-- Trigger on INSERT: Ensures the contact number is valid
DELIMITER //
CREATE TRIGGER BeforeInsertDistributor
BEFORE INSERT ON DISTRIBUTOR
FOR EACH ROW
BEGIN
   IF NEW.Contact_No IS NULL THEN
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Distributor contact number cannot be NULL';
   END IF;
END //
DELIMITER ;


-- Trigger on DELETE: Prevents deletion of distributor if related to supplies
```

```
DELIMITER //
CREATE TRIGGER BeforeDeleteDistributor
BEFORE DELETE ON DISTRIBUTOR
FOR EACH ROW
BEGIN
    DECLARE distributor_exists INT;
    SELECT COUNT(*) INTO distributor_exists
    FROM SUPPLIES
    WHERE Distributor_ID = OLD.Distributor_ID;
    IF distributor_exists > 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot delete distributor with
existing supplies';
    END IF;
END //
DELIMITER ;
```

11. EMPLOYEE

```
-- Trigger on INSERT: Ensures employee name is not empty
DELIMITER //
CREATE TRIGGER BeforeInsertEmployee
BEFORE INSERT ON EMPLOYEE
FOR EACH ROW
BEGIN
    IF NEW.Name IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Employee name cannot be NULL';
    END IF;
END //
DELIMITER ;


-- Trigger on DELETE: Prevents deletion of employee if related to sale transactions
```

```
DELIMITER //

CREATE TRIGGER BeforeDeleteEmployee

BEFORE DELETE ON EMPLOYEE

FOR EACH ROW

BEGIN

    DECLARE employee_exists INT;

    SELECT COUNT(*) INTO employee_exists

    FROM SALE_TRANSACTION

    WHERE Employee_ID = OLD.Employee_ID;

    IF employee_exists > 0 THEN

        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot delete employee with
existing sale transactions';

    END IF;

END //

DELIMITER ;
```

ADVANCED QUERIES

1. **Get the Total Sales by Each Customer**

```
SELECT C.Patient_ID, C.Name, SUM(S.Quantity * D.MRP) AS Total_Sales

FROM CUSTOMER C

JOIN SELLS S ON C.Patient_ID = S.Patient_ID

JOIN DRUG D ON S.Drug_ID = D.Drug_ID

GROUP BY C.Patient_ID, C.Name

ORDER BY Total_Sales DESC;
```

2. Find All Doctors Who Have Prescribed Drugs to a Specific Customer

```
SELECT DISTINCT D.Name AS Doctor_Name, D.Specialization

FROM DOCTOR D
```

```
JOIN PRESCRIBES P ON D.Doctor_ID = P.Doctor_ID
JOIN CUSTOMER C ON P.Patient_ID = C.Patient_ID
WHERE C.Name = 'John Doe';
```

3. List All Drugs That Are Out of Stock

```
SELECT D.Name AS Drug_Name, D.Stock, D.Expiry
FROM DRUG D
WHERE D.Stock <= 0;
```

4. Get the Total Number of Sales for Each Drug

```
SELECT D.Name AS Drug_Name, COUNT(S.Sale_ID) AS Total_Sales
FROM DRUG D
JOIN SELLS S ON D.Drug_ID = S.Drug_ID
GROUP BY D.Drug_ID, D.Name
HAVING COUNT(S.Sale_ID) > 5
ORDER BY Total_Sales DESC;
```

5. Get the Most Expensive Drug Sold to Each Customer

```
SELECT C.Name AS Customer_Name,
    (SELECT D.Name
     FROM DRUG D
     JOIN SELLS S ON D.Drug_ID = S.Drug_ID
     WHERE S.Patient_ID = C.Patient_ID
     ORDER BY D.MRP DESC LIMIT 1) AS Most_Expensive_Drug
FROM CUSTOMER C;
```

6. List All Employees Who Sold Drugs to a Specific Customer

```
SELECT DISTINCT E.Name AS Employee_Name
FROM EMPLOYEE E
JOIN SALE_TRANSACTION ST ON E.Employee_ID = ST.Employee_ID
JOIN SELLS S ON ST.Sale_ID = S.Sale_ID
JOIN CUSTOMER C ON S.Patient_ID = C.Patient_ID
WHERE C.Name = 'John Doe';
```

7. Find the Total Value of Drugs Supplied by Each Distributor

```
SELECT D.Name AS Distributor_Name, SUM(S.Quantity * DR.MRP) AS Total_Value
FROM DISTRIBUTOR D
JOIN SUPPLIES S ON D.Distributor_ID = S.Distributor_ID
JOIN DRUG DR ON S.Drug_ID = DR.Drug_ID
GROUP BY D.Distributor_ID, D.Name
ORDER BY Total_Value DESC;
```

8. Get the Most Prescribed Drug by Each Doctor

```
SELECT D.Name AS Doctor_Name,
       (SELECT DR.Name
        FROM DRUG DR
        JOIN PRESCRIBES P ON DR.Drug_ID = P.Drug_ID
        WHERE P.Doctor_ID = D.Doctor_ID
        GROUP BY DR.Drug_ID
        ORDER BY COUNT(P.Drug_ID) DESC LIMIT 1) AS Most_Prescribed_Drug
FROM DOCTOR D;
```

9. Get the Total Purchase Value of Drugs Supplied in Each Purchase

```
SELECT P.Purchase_ID, SUM(S.Quantity * DR.MRP) AS Total_Purchase_Value
FROM PURCHASE P
JOIN SUPPLIES S ON P.Purchase_ID = S.Purchase_ID
JOIN DRUG DR ON S.Drug_ID = DR.Drug_ID
GROUP BY P.Purchase_ID;
```

VIEWS

1. View for Total Sales by Each Customer

This view shows the total sales amount for each customer by multiplying the quantity sold by the MRP of each drug.

*CREATE VIEW Total_Sales_By_Customer AS SELECT C.Patient_ID, C.Name AS Customer_Name, SUM(S.Quantity * D.MRP) AS Total_Sales FROM CUSTOMER C JOIN SELLS S ON C.Patient_ID = S.Patient_ID JOIN DRUG D ON S.Drug_ID = D.Drug_ID GROUP BY C.Patient_ID, C.Name ORDER BY Total_Sales DESC;*

2. View for Doctors and Their Most Prescribed Drugs

This view shows each doctor along with the most prescribed drug. This is useful to analyze which drugs are frequently prescribed by each doctor.

CREATE VIEW Doctors_Most_Prescribed_Drugs AS SELECT D.Doctor_ID, D.Name AS Doctor_Name, (SELECT DR.Name FROM DRUG DR JOIN PRESCRIBES P ON DR.Drug_ID = P.Drug_ID WHERE P.Doctor_ID = D.Doctor_ID GROUP BY DR.Drug_ID ORDER BY COUNT(P.Drug_ID) DESC LIMIT 1) AS Most_Prescribed_Drug FROM DOCTOR D;

3. View for Out of Stock Drugs

This view shows all drugs that are currently out of stock (Stock <= 0).

CREATE VIEW Out_Of_Stock_Drugs AS SELECT D.Drug_ID, D.Name AS Drug_Name, D.Stock, D.Expiry FROM DRUG D WHERE D.Stock <= 0;

4. View for Employee Sales

This view shows the total sales value handled by each employee. It aggregates the quantity of drugs sold by each employee.

*CREATE VIEW Employee_Sales AS SELECT E.Employee_ID, E.Name AS Employee_Name, SUM(S.Quantity * D.MRP) AS Total_Sales_Value FROM EMPLOYEE E JOIN SALE_TRANSACTION ST ON E.Employee_ID = ST.Employee_ID JOIN SELLS S ON*

ST.Sale_ID = S.Sale_ID JOIN DRUG D ON S.Drug_ID = D.Drug_ID GROUP BY E.Employee_ID, E.Name ORDER BY Total_Sales_Value DESC;

5. View for Supplier Information and Drug Stocks

This view shows the total stock of each drug supplied by each distributor.

CREATE VIEW Supplier_Drug_Stock AS SELECT D.Name AS Distributor_Name, DR.Name AS Drug_Name, SUM(S.Quantity) AS Total_Stock FROM DISTRIBUTOR D JOIN SUPPLIES S ON D.Distributor_ID = S.Distributor_ID JOIN DRUG DR ON S.Drug_ID = DR.Drug_ID GROUP BY D.Distributor_ID, DR.Drug_ID ORDER BY Distributor_Name, Drug_Name;

6. View for Customer Purchases and Drugs

This view shows all customers, the drugs they purchased, and the total quantity purchased.

CREATE VIEW Customer_Drug_Purchases AS SELECT C.Patient_ID, C.Name AS Customer_Name, D.Name AS Drug_Name, SUM(S.Quantity) AS Total_Quantity FROM CUSTOMER C JOIN SELLS S ON C.Patient_ID = S.Patient_ID JOIN DRUG D ON S.Drug_ID = D.Drug_ID GROUP BY C.Patient_ID, D.Drug_ID ORDER BY Customer_Name, Drug_Name;

7. View for Purchase History with Drug and Supplier Details

This view shows the purchase history along with drug names, quantities, and the distributor supplying the drugs.

CREATE VIEW Purchase_History AS SELECT P.Purchase_ID, P.Date AS Purchase_Date, DR.Name AS Drug_Name, S.Quantity AS Quantity_Supplied, D.Name AS Distributor_Name FROM PURCHASE P JOIN SUPPLIES S ON P.Purchase_ID = S.Purchase_ID JOIN DRUG

DR ON S.Drug_ID = DR.Drug_ID JOIN DISTRIBUTOR D ON S.Distributor_ID =

D.Distributor_ID ORDER BY P.Date DESC;

8. View for Sale Transactions with Employee and Drug Details

This view shows the details of sale transactions, including the employee handling the sale, drugs sold, and their quantities.

CREATE VIEW Sale_Transactions_Details AS SELECT ST.Sale_ID, ST.Date AS Sale_Date,

E.Name AS Employee_Name, D.Name AS Drug_Name, S.Quantity AS Quantity_Sold FROM

SALE_TRANSACTION ST JOIN EMPLOYEE E ON ST.Employee_ID = E.Employee_ID JOIN

SELLS S ON ST.Sale_ID = S.Sale_ID JOIN DRUG D ON S.Drug_ID = D.Drug_ID ORDER BY

Sale_Date DESC;

9. View for Prescription History

This view shows the prescription history for each patient, including the doctor who prescribed the drug and the drug name.

CREATE VIEW Prescription_History AS SELECT C.Name AS Patient_Name, D.Name AS

Doctor_Name, DR.Name AS Drug_Name FROM PRESCRIBES P JOIN CUSTOMER C ON

P.Patient_ID = C.Patient_ID JOIN DOCTOR D ON P.Doctor_ID = D.Doctor_ID JOIN DRUG

DR ON P.Drug_ID = DR.Drug_ID ORDER BY Patient_Name, Doctor_Name;

10. View for Drug Availability by Manufacturer

This view shows the availability (stock) of each drug grouped by the manufacturer.

CREATE VIEW Drug_Availability_By_Manufacturer AS SELECT M.Name AS

Manufacturer_Name, DR.Name AS Drug_Name, D.Stock AS Stock_Available FROM DRUG D

JOIN DRUG_MANUFACTURER M ON D.Company_ID = M.Company_ID ORDER BY Manufacturer_Name, Drug_Name;

REPORTS

1. Total Sales Report by Customer

This report shows the total sales for each customer, helping the business understand which customers are driving the most revenue.

*SELECT C.Name AS Customer_Name, SUM(S.Quantity * D.MRP) AS Total_Sales_Value FROM CUSTOMER C JOIN SELLS S ON C.Patient_ID = S.Patient_ID JOIN DRUG D ON S.Drug_ID = D.Drug_ID GROUP BY C.Patient_ID ORDER BY Total_Sales_Value DESC;*

2. Doctor Prescription Report

This report shows which drugs each doctor is prescribing the most to their patients. It helps identify the most popular drugs among different doctors.

SELECT D.Name AS Doctor_Name, DR.Name AS Drug_Name, COUNT(P.Drug_ID) AS Prescription_Count FROM DOCTOR D JOIN PRESCRIBES P ON D.Doctor_ID = P.Doctor_ID JOIN DRUG DR ON P.Drug_ID = DR.Drug_ID GROUP BY D.Doctor_ID, DR.Drug_ID ORDER BY Prescription_Count DESC;

3. Stock Availability Report

This report shows the current stock levels of each drug, which helps in monitoring inventory levels.

```
SELECT D.Name AS Drug_Name, D.Stock AS Available_Stock, D.Expiry AS Expiry_Date
FROM DRUG D WHERE D.Stock > 0 ORDER BY D.Stock DESC;
```

4. Total Purchase Report

This report shows the total purchase value for each purchase, helping track purchase expenses and drug stock replenishments.

```
SELECT P.Purchase_ID, P.Date AS Purchase_Date, SUM(S.Quantity * DR.MRP) AS
Total_Purchase_Value FROM PURCHASE P JOIN SUPPLIES S ON P.Purchase_ID =
S.Purchase_ID JOIN DRUG DR ON S.Drug_ID = DR.Drug_ID GROUP BY P.Purchase_ID
ORDER BY Total_Purchase_Value DESC;
```

5. Drug Sales Report

This report displays the most popular drugs in terms of sales, helping identify the top-selling drugs.

```
SELECT DR.Name AS Drug_Name, SUM(S.Quantity) AS Total_Quantity_Sold, SUM(S.Quantity
* DR.MRP) AS Total_Sales_Value FROM DRUG DR JOIN SELLS S ON DR.Drug_ID =
S.Drug_ID GROUP BY DR.Drug_ID ORDER BY Total_Sales_Value DESC;
```

6. Employee Sales Performance Report

This report evaluates the sales performance of each employee, helping track which employees are driving the most sales.

```
SELECT E.Name AS Employee_Name, SUM(S.Quantity * DR.MRP) AS Total_Sales_Value
FROM EMPLOYEE E JOIN SALE_TRANSACTION ST ON E.Employee_ID = ST.Employee_ID
JOIN SELLS S ON ST.Sale_ID = S.Sale_ID JOIN DRUG DR ON S.Drug_ID = DR.Drug_ID
GROUP BY E.Employee_ID ORDER BY Total_Sales_Value DESC;
```

7. Customer Prescription Report

This report displays the drugs prescribed to each customer, helping to analyze the treatment trends of each patient.

```
SELECT C.Name AS Customer_Name, DR.Name AS Drug_Name, COUNT(P.Drug_ID) AS
Prescription_Count FROM CUSTOMER C JOIN PRESCRIBES P ON C.Patient_ID =
P.Patient_ID JOIN DRUG DR ON P.Drug_ID = DR.Drug_ID GROUP BY C.Patient_ID,
DR.Drug_ID ORDER BY Prescription_Count DESC;
```

8. Drug Manufacturer Report

This report lists each drug manufacturer and the total value of drugs supplied by them, helping track which manufacturers contribute the most to stock.

```
SELECT M.Name AS Manufacturer_Name, SUM(S.Quantity * D.MRP) AS Total_Supply_Value
FROM DRUG_MANUFACTURER M JOIN DRUG D ON M.Company_ID = D.Company_ID
JOIN SUPPLIES S ON D.Drug_ID = S.Drug_ID GROUP BY M.Company_ID ORDER BY
Total_Supply_Value DESC;
```

9. Sales Transactions Report

This report lists the sales transactions, including the employee who made the sale and the drugs involved, helping analyze each sale.

```
SELECT ST.Sale_ID, ST.Date AS Sale_Date, E.Name AS Employee_Name, D.Name AS
Drug_Name, S.Quantity AS Quantity_Sold FROM SALE_TRANSACTION ST JOIN
EMPLOYEE E ON ST.Employee_ID = E.Employee_ID JOIN SELLS S ON ST.Sale_ID =
S.Sale_ID JOIN DRUG D ON S.Drug_ID = D.Drug_ID ORDER BY Sale_Date DESC;
```

10. Distributor Drug Supply Report

This report lists each distributor and the drugs they supply, including the quantity supplied for each drug.

```
SELECT D.Name AS Distributor_Name, DR.Name AS Drug_Name, SUM(S.Quantity) AS
Quantity_Supplied FROM DISTRIBUTOR D JOIN SUPPLIES S ON D.Distributor_ID =
S.Distributor_ID JOIN DRUG DR ON S.Drug_ID = DR.Drug_ID GROUP BY
D.Distributor_ID, DR.Drug_ID ORDER BY Distributor_Name, Drug_Name;
```

11. Drug Expiry Report

This report shows all drugs that are close to expiry, helping ensure that products are rotated and replaced before they expire.

```
SELECT D.Name AS Drug_Name, D.Expiry AS Expiry_Date FROM DRUG D WHERE
D.Expiry <= CURDATE() + INTERVAL 1 MONTH ORDER BY D.Expiry;
```