**Final Phase: Happy Drugstore DBMS Design**

Team members: Yoonjoung Kim, Rimon Chemali, Linda Neeley

CS5318

**1. Abstract**

This document will cover the purpose, design, potential outcomes, and other non-limiting factors for the implementation of a database management application. On a high level, this DBMS will be used for a new drug store and will help the store manage customer information, prescriptions and sales data with various data groups for various end users. The different data groups (with their attributes, domains, keys, relations…) and different user groups (access and views) will be thoroughly explained in this document.

Throughout the entire design of the DBMS, from mission statement to DBMS push, each step will be documented and consulted with the client.

To account for any risks associated when implementing the DBMS, the following may be addressed and possibly avoided, but not limited to:

* Data Security and Patient Privacy: Ensuring sensitive information, such as patient information or drug inventory data, are protected from unauthorized users.
* Data Integrity: Maintaining tracing methods to sustain accurate and consistent data.
* Compliance of products: Following state-level and federal standards for drug management.
* Cost Overruns: Managing budgets, marketing tactics and services to avoid cost run-aways.
* User Training and Adoption: Ensuring staff adopt the system and train how to use the new system.
* Scalable system: Designing the system to handle future growth (whether data volume or user volume) without performance drop.

**2. Misson Statement**

The proposed drugstore DBMS will implement a secure, efficient, and scalable system that integrates internal drugstore users with drug/patient data, in order to accomplish safe, non-risk, information access and management.

**3. Mission Objective**

1. To maintain (enter, update, and delete) product(drug) information
2. To maintain (enter, update, and delete) customer information
3. To maintain (enter, update, and delete) employee information
4. To maintain (enter, update, and delete) doctor information
5. To maintain (enter, update, and delete) suppliers information
6. To maintain (enter, update, and delete) prescription data
7. To maintain (enter, update, and delete) sales data
8. To maintain (enter, update, and delete) insurance information
9. To maintain (enter, update, and delete) regulatory compliance information
10. To perform searches on drug and product information
11. To perform searches on customer information
12. To perform searches on employee information
13. To perform searches on doctor information
14. To perform searches on suppliers information
15. To perform searches on prescription data
16. To perform searches on sales data
17. To perform searches on insurance information
18. To perform searches on orders information
19. To track status of inventory levels
20. To track status of drug reorders from suppliers
21. To track status of prescriptions
22. To track status of insurance information
23. To track status of drug recalls
24. To report on drug and product data
25. To report on customer data
26. To report on employee data
27. To report on doctor data
28. To report on suppliers data
29. To report on prescription data
30. To report on sales data
31. To report on insurance data
32. To report on orders data

**4. Major User Views**

**Users:** Manager (Pharmacist in Charge), Pharmacist, Pharmacy technician, Customer, Cashier, Sales Representative

**Access Type**: Maintain, Query, Report

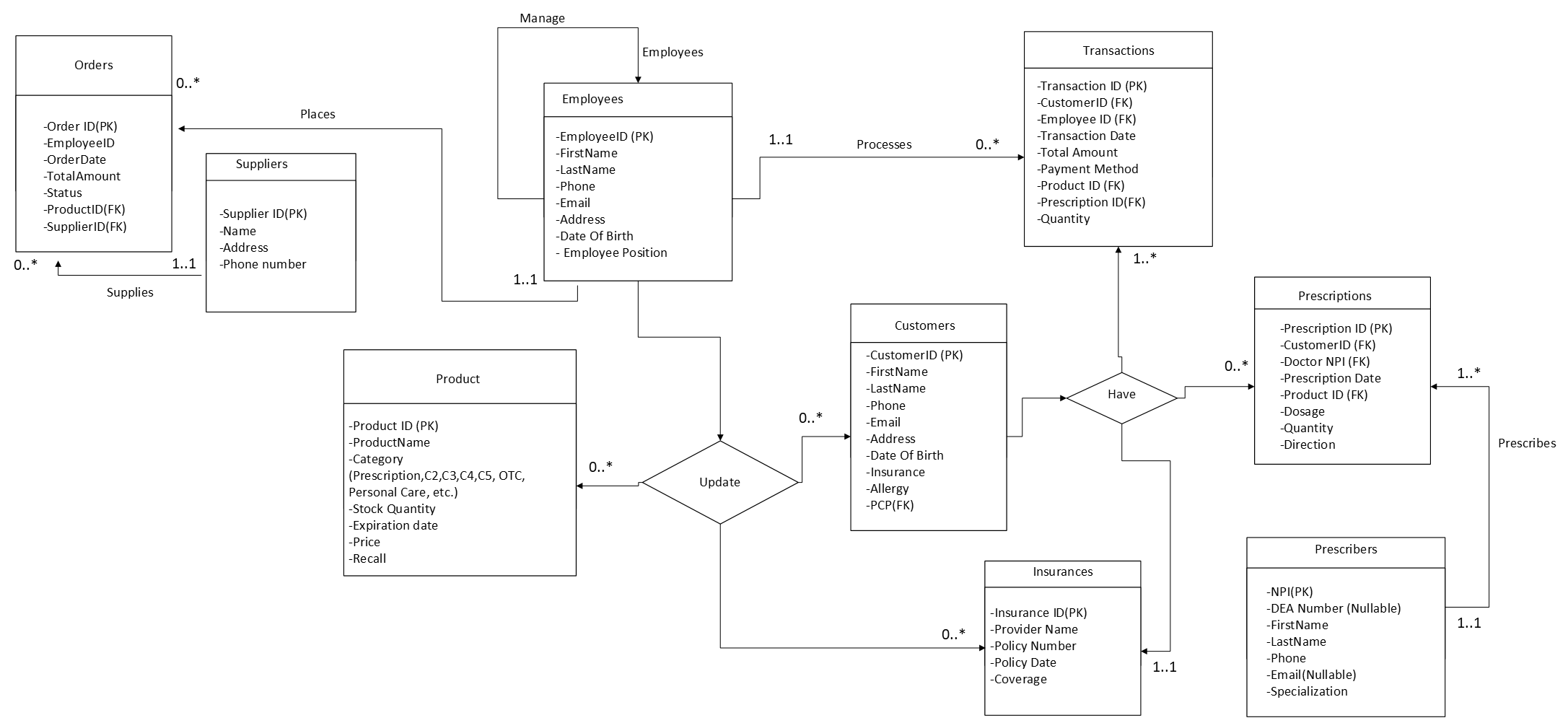
* Maintain (Add, Update, Delete)
* Query (Read-Only Access – Retrieve Data)
* Report (Generate Summaries)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Data | Access Type | Manager (PIC) | Pharmacist | Pharmacy Technician | Sales Rep | Cashier | Customer |
| **Customers** | Maintain | x | x | x | x |  |  |
|  | Query | x | x | x | x | x | x |
|  | Report | x |  |  |  |  |  |
| **Employees** | Maintain | x |  |  |  |  |  |
|  | Query | x |  |  | x |  |  |
|  | Report | x |  |  |  |  |  |
| **Prescriptions** | Maintain | x | x |  |  |  |  |
|  | Query | x | x | x |  |  | x |
|  | Report | x | x |  |  |  |  |
| **Suppliers** | Maintain | x | x | x | x |  |  |
|  | Query | x | x | x | x |  |  |
|  | Report | x |  |  |  |  |  |
| **Prescribers** | Maintain | x | x | x |  |  |  |
|  | Query | x | x | x |  |  |  |
|  | Report | x |  |  |  |  |  |
| **Insurances & Billing** | Maintain | x |  |  |  |  |  |
|  | Query | x | x | x |  |  |  |
|  | Report | x |  |  |  |  |  |
| **Transactions** | Maintain | x | x | x | x | x |  |
|  | Query | x | x | x | x | x | x |
|  | Report | x |  |  |  |  |  |
| **Products** | Maintain | x | x |  | x |  |  |
|  | Query | x | x | x | x | x |  |
|  | Report | x |  |  | x |  |  |
| **Orders** | Maintain | x | x | x | x |  |  |
|  | Query | x | x | x | x |  |  |
|  | Report | x |  |  |  |  |  |

**5. E/R Diagram**

This is the Entity-Relationship (E/R) diagram for the Happy Drugstore Database Management System (DBMS). The design ensures that all entities are directly or indirectly related, with relationships per entity. This DBMS is tailored to support the Happy Drugstore's operational needs, enabling efficient data storage and access.

<The remainder of this page is intentionally blank. the ER model is on next page.>



**6. Relational Model** **and BCNF verification.**

The following tables represent the Happy Drugstore DBMS structure, derived from the E/R diagram. Each SQL command shows the integrity of each attribute, domain, schema and relational keys with references. The choice of DBMS used for this project is: DB Browser for SQLite (DB4S)

**6.1) Suppliers**

CREATE TABLE Suppliers (

SupplierID INT NOT NULL,

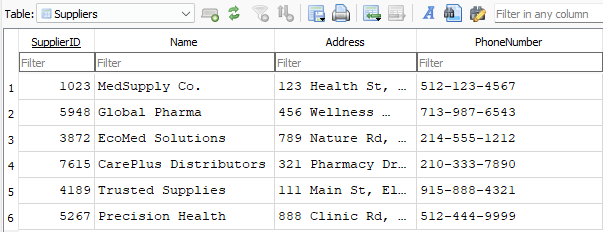
Name VARCHAR(100),

Address VARCHAR(255),

PhoneNumber VARCHAR(15),

PRIMARY KEY (SupplierID)

);



* **Primary Key:** SupplierID
* **Functional Dependency:** SupplierID → Name, Address, PhoneNumber
* **Example Dependency:** SupplierID 1023 is associated with 'MedSupply Co.', proving Name does not depend on any external attributes.
* **1NF:** Each attribute contains a single, atomic value.
* **2NF:** Name, Address, and PhoneNumber fully depend on SupplierID, the primary key.
* **3NF:** Since SupplierID is the sole determinant, there are no transitive dependencies.
* **BCNF:** The table satisfies BCNF as SupplierID uniquely determines all other attributes.

**6.2) Product\_Inventory**

CREATE TABLE Product\_Inventory (

ProductID INT NOT NULL,

ProductName VARCHAR(100),

Category VARCHAR(50),

StockQuantity INT,

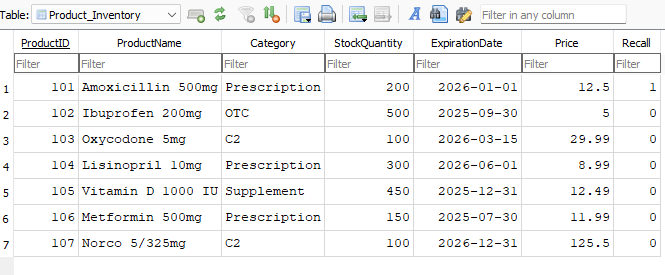
ExpirationDate DATE,

Price DECIMAL(10,2),

Recall BOOLEAN DEFAULT FALSE,

PRIMARY KEY (ProductID)

);



* Primary Key: ProductID
* Functional Dependency: ProductID → ProductName, Category, StockQuantity, ExpirationDate, Price, Recall
* Example Dependency: ProductID 101 is associated with 'Amoxicillin 500mg', proving StockQuantity depends only on ProductID.
* 1NF: Each attribute contains atomic values.
* 2NF: All attributes are fully dependent on ProductID.
* 3NF: No transitive dependency between non-prime attributes.
* BCNF: Every determinant is a candidate key.

**6.3) Employees**

CREATE TABLE Employees (

EmployeeID INT NOT NULL,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Phone VARCHAR(15),

Email VARCHAR(100),

Address VARCHAR(255),

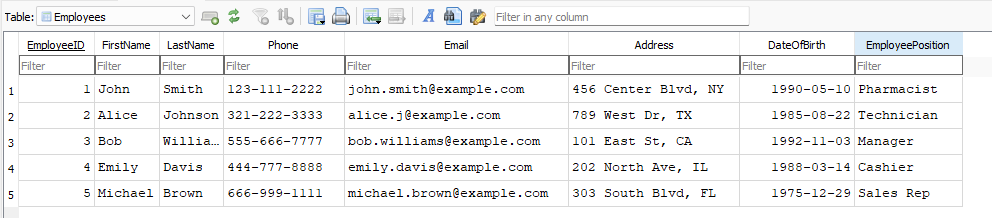
DateOfBirth DATE,

EmployeePosition VARCHAR(50),

PRIMARY KEY (EmployeeID),

CHECK (EmployeePosition IN ('Pharmacist', 'Technician', 'Cashier', 'Manager','Sales Rep'))

);



* **Primary Key:** EmployeeID
* **Functional Dependency:** EmployeeID → FirstName, LastName, Phone, Email, Address, DateOfBirth, EmployeePosition
* **Example Dependency:** EmployeeID 1 is associated with 'John Smith', proving that FirstName does not depend on EmployeePosition.
* **1NF:** Each field contains atomic data.
* **2NF:** FirstName, LastName, Phone, and EmployeePosition all rely on the primary key EmployeeID.
* **3NF:** No transitive dependencies, since each attribute directly depends on EmployeeID.
* **BCNF:** Each subset of the primary key does not determine non-primary attributes.

**6.4) Insurances**

CREATE TABLE Insurances (

InsuranceID INT NOT NULL,

ProviderName VARCHAR(100),

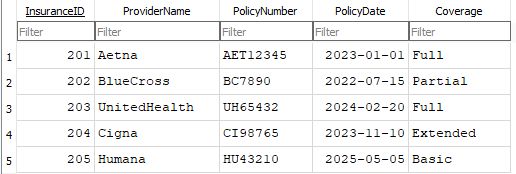
PolicyNumber VARCHAR(50),

PolicyDate DATE,

Coverage VARCHAR(255),

PRIMARY KEY (InsuranceID)

);



* **Primary Key:** InsuranceID
* **Functional Dependency:** InsuranceID → ProviderName, PolicyNumber, PolicyDate, Coverage
* **Example Dependency:** InsuranceID 201 is associated with 'Aetna', proving ProviderName does not depend on PolicyDate.
* **1NF:** Each field contains atomic values
* **2NF:** All attributes fully depend on InsuranceID, the primary key.
* **3NF:** No transitive dependencies exist since each attribute is determined solely by InsuranceID.
* **BCNF:** Every determinant uniquely determines its attributes.

**6.5) Transactions**

CREATE TABLE Transactions (

TransactionID INT NOT NULL,

CustomerID INT,

EmployeeID INT,

TransactionDate DATE,

TotalAmount DECIMAL(10,2),

PaymentMethod VARCHAR(50),

ProductID INT,

PrescriptionID INT,

Quantity INT,

PRIMARY KEY (TransactionID),

FOREIGN KEY (CustomerID)

REFERENCES Customers(CustomerID),

FOREIGN KEY (EmployeeID)

REFERENCES Employees(EmployeeID),

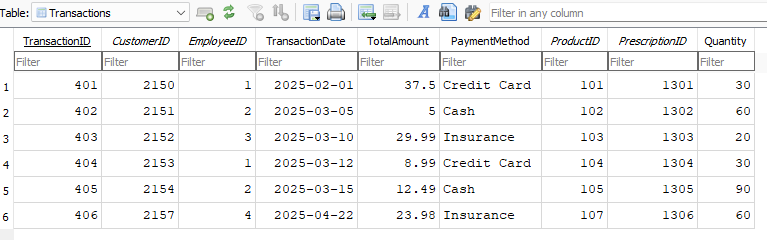
FOREIGN KEY (ProductID)

REFERENCES Product\_Inventory(ProductID),

FOREIGN KEY (PrescriptionID)

REFERENCES Prescriptions(PrescriptionID)

);



* **Primary Key:** TransactionID
* **Functional Dependency:** TransactionID → CustomerID, EmployeeID, TransactionDate, TotalAmount, PaymentMethod, ProductID, PrescriptionID, Quantity
* **Example Dependency:** TransactionID 401 is associated with CustomerID 2150 and ProductID 101, proving these do not depend on PaymentMethod.
* **1NF:** Every field contains atomic values.
* **2NF:** All attributes fully depend on TransactionID, the primary key.
* **3NF:** No transitive dependency exists; attributes depend directly on TransactionID.
* **BCNF:** Each subset of the primary key does not functionally depend on

**6.6) Customers**

CREATE TABLE Customers (

CustomerID INT NOT NULL,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Phone VARCHAR(15),

Email VARCHAR(100),

Address VARCHAR(255),

DateOfBirth DATE,

Insurance INT,

Allergy VARCHAR(100),

PCP VARCHAR(10),

PRIMARY KEY (CustomerID),

FOREIGN KEY (Insurance)

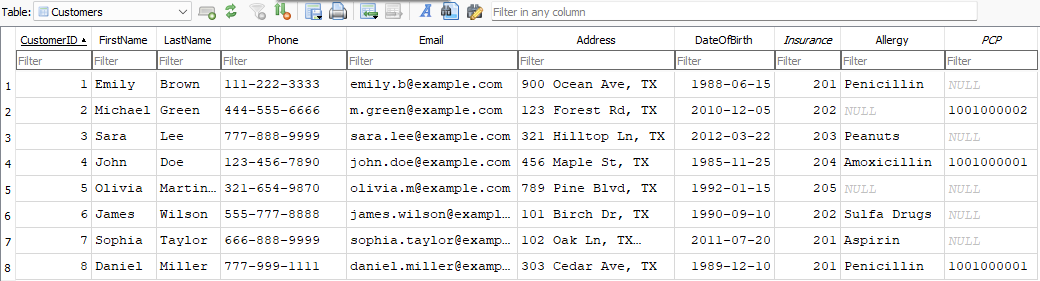
REFERENCES Insurances(InsuranceID)

ON DELETE SET NULL

FOREIGN KEY (PCP)

REFERENCES Prescribers(NPI)

);



* **Primary Key:** CustomerID
* **Functional Dependency:** CustomerID → FirstName, LastName, Phone, Email, Address, DateOfBirth, Insurance, Allergy, PCP
* **Example Dependency:** CustomerID 2150 is associated with 'Emily Brown', proving that FirstName does not depend on Insurance.
* **1NF:** No repeating groups; each field contains atomic data.
* **2NF:** All attributes fully depend on CustomerID, ensuring compliance.
* **3NF:** No transitive dependencies exist; Insurance is properly referenced via a foreign key.
* **BCNF:** Every determinant uniquely determines its attributes.

**6.7) Prescribers**

CREATE TABLE Prescribers (

NPI VARCHAR(10) NOT NULL,

DEANumber VARCHAR(9),

FirstName VARCHAR(50),

LastName VARCHAR(50),

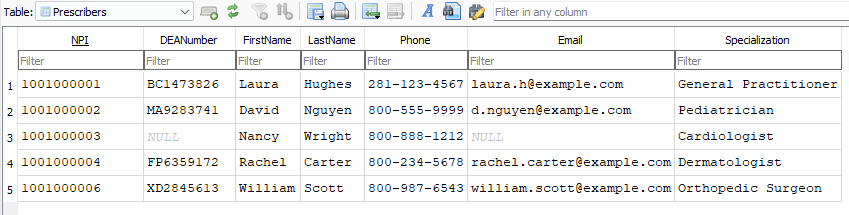
Phone VARCHAR(15),

Email VARCHAR(100),

Specialization VARCHAR(100),

PRIMARY KEY (NPI)

);



* **Primary Key:** NPI
* **Functional Dependency:** NPI → DEANumber, FirstName, LastName, Phone, Email, Specialization
* **Example Dependency:** NPI 1001000001 is associated with 'Laura Hughes', proving Specialization does not depend on DEANumber.
* **1NF:** Each field contains atomic values.
* **2NF:** All attributes fully depend on NPI, the primary key.
* **3NF:** No transitive dependencies exist; DEANumber does not determine other attributes.
* **BCNF:** Every determinant uniquely determines its attributes.

**6.8) Prescriptions**

CREATE TABLE Prescriptions (

PrescriptionID INT NOT NULL,

CustomerID INT,

DoctorNPI VARCHAR(10),

PrescriptionDate DATE,

ProductID INT,

Dosage VARCHAR(50),

Quantity INT,

Direction TEXT,

PRIMARY KEY (PrescriptionID),

FOREIGN KEY (CustomerID)

REFERENCES Customers(CustomerID),

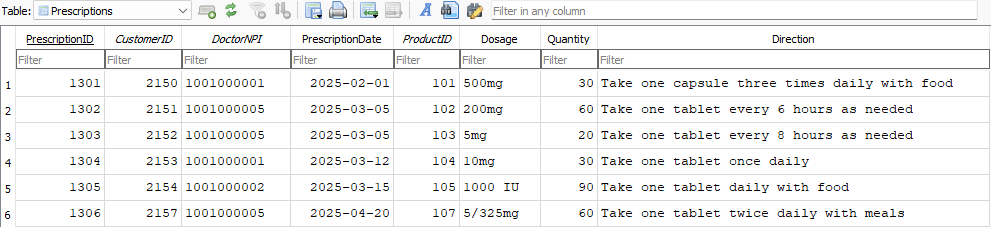
FOREIGN KEY (DoctorNPI)

REFERENCES Prescribers(NPI),

FOREIGN KEY (ProductID)

REFERENCES Product\_Inventory(ProductID)

);



* **Primary Key:** PrescriptionID
* **Functional Dependency:** PrescriptionID → CustomerID, DoctorNPI, PrescriptionDate, ProductID, Dosage, Quantity, Direction
* **Example Dependency:** PrescriptionID 1301 is associated with CustomerID 2150 and ProductID 101, proving Quantity does not depend on DoctorNPI.
* **1NF:** Each field contains atomic values.
* **2NF:** All attributes fully depend on PrescriptionID, the primary key.
* **3NF:** No transitive dependencies exist.
* **BCNF:** Each subset of the primary key does not functionally depend on non-primary attributes.

**6.9) Orders**

CREATE TABLE Orders (

OrderID INT NOT NULL,

EmployeeID INT,

OrderDate DATE NOT NULL,

TotalAmount DECIMAL(10,2),

Status VARCHAR(20) CHECK (Status IN ('Pending', 'Shipped', 'Delivered', 'Canceled')),

ProductID INT,

SupplierID INT,

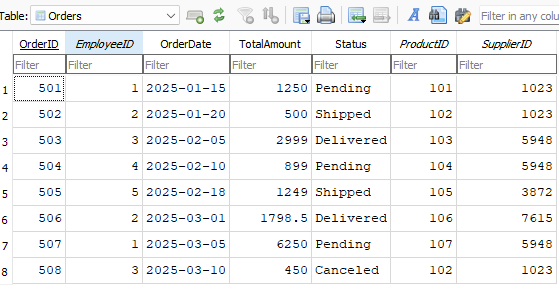
PRIMARY KEY (OrderID),

FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID),

FOREIGN KEY (ProductID) REFERENCES Product\_Inventory(ProductID),

FOREIGN KEY (SupplierID) REFERENCES Suppliers(SupplierID)

);



* Primary Key: OrderID
* Functional Dependency: OrderID → EmployeeID, OrderDate, TotalAmount, Status, ProductID, SupplierID
* Example Dependency: OrderID 501 maps to ProductID 101 and SupplierID 1023, showing all attributes depend on OrderID.
* 1NF: No repeating groups, only atomic values.
* 2NF: Attributes fully functionally dependent on OrderID.
* 3NF: No transitive dependencies among non-prime attributes.
* BCNF: Every determinant is a candidate key.

**7. Complete List of Use Cases**

All actors (users):

* **Manager**: Keeps track of documents and supervises operations.
* **Pharmacist**: Manages inventory and prescriptions.
* **Pharmacy Technician**: Helps process prescriptions.
* **Cashier**: Processes transactions.
* **Sales Representative**: Keeps track of customers', ’orders’, and products' information.
* **Customer**: Engages with the system and makes drug purchases.

The below use cases from sections **7.1** to **7.8** will also include at least one aggregate query for each table. The **JOIN** queries are found in section **7.9**.

**7.1) Suppliers Use Cases**

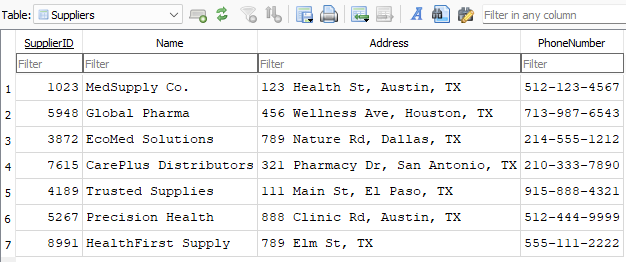
1) Add New Supplier

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the supplier management section.
3. Enters new supplier information.
4. Reviews the information.
5. Clicks 'Save' to store the supplier data.

INSERT INTO Suppliers VALUES

(8991, 'HealthFirst Supply', '789 Elm St, TX', '555-111-2222');



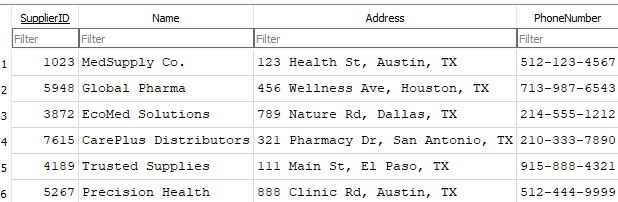
2) Delete Supplier

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the supplier list.
3. Selects the supplier to delete.
4. Reviews the supplier information.
5. Clicks 'Delete' and confirms.

DELETE FROM Suppliers

WHERE SupplierID = 8991;



3) Update Supplier Address

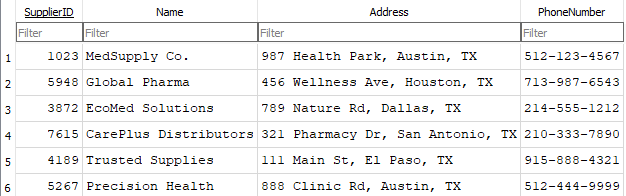
Actor: Manager  
Steps:

1. Manager logs into the system.
2. Searches for the supplier by ID.
3. Updates the supplier's address.
4. Reviews the updated details.
5. Saves the changes.

UPDATE Suppliers

SET Address = '987 Health Park, Austin, TX'

WHERE SupplierID = 1023;



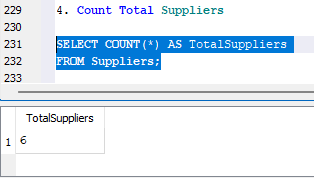
4) Count Total Suppliers (AGGREGATE)

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the reports section.
3. Selects supplier data.
4. Runs the report to count suppliers.

SELECT COUNT(\*) AS TotalSuppliers

FROM Suppliers;



**7.2) Product\_Inventory Use Cases**

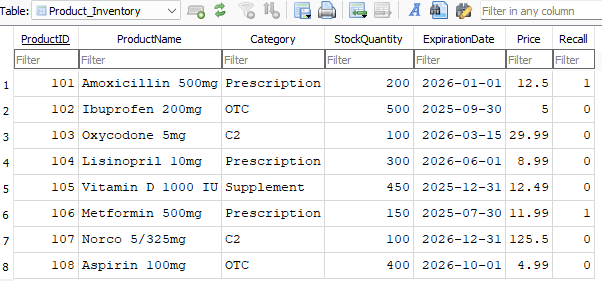
5) Add New Product

Actor: Technician  
Steps:

1. Technician logs into the system.
2. Navigates to the product inventory section.
3. Enters new product details including name, category, stock quantity, expiration date, price, and recall status.
4. Reviews the details.
5. Clicks 'Save' to store the new product.

INSERT INTO Product\_Inventory VALUES

(108, 'Aspirin 100mg', 'OTC', 400, '2026-10-01', 4.99, FALSE);



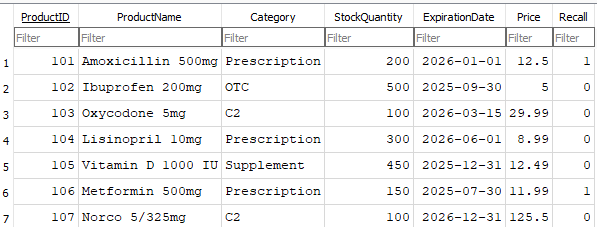
6) Delete Product

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the product inventory.
3. Selects the product to delete.
4. Reviews product details.
5. Clicks 'Delete' and confirms deletion.

DELETE FROM Product\_Inventory

WHERE ProductID = 108;



7) Update Product Price

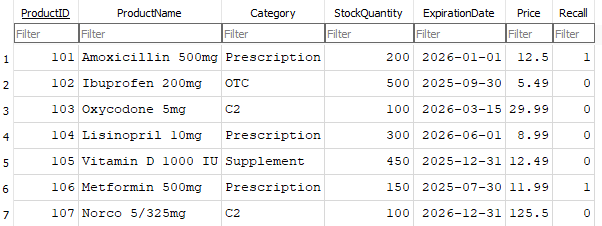
Actor: Technician  
Steps:

1. Technician logs into the system.
2. Searches for the product by Product ID.
3. Updates the product price.
4. Reviews the changes.
5. Saves the updated information.

UPDATE Product\_Inventory

SET Price = 5.49

WHERE ProductID = 102;



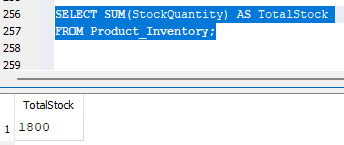
8) Find Total Stock Quantity (AGGREGATE)

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the reports section.
3. Selects product inventory data.
4. Runs the report to calculate the total stock quantity.
5. Reviews the report output.

SELECT SUM(StockQuantity) AS TotalStock

FROM Product\_Inventory;



**7.3) Employees Use Cases**

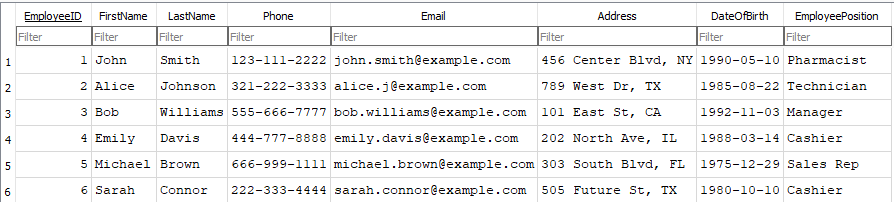
9. Add New Employee

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the employee management section.
3. Inputs new employee details including name, contact information, date of birth, and position.
4. Reviews the details.
5. Clicks 'Save' to store the new employee.

INSERT INTO Employees VALUES

(6, 'Sarah', 'Connor', '222-333-4444', 'sarah.connor@example.com', '505 Future St, TX', '1980-10-10', 'Cashier');



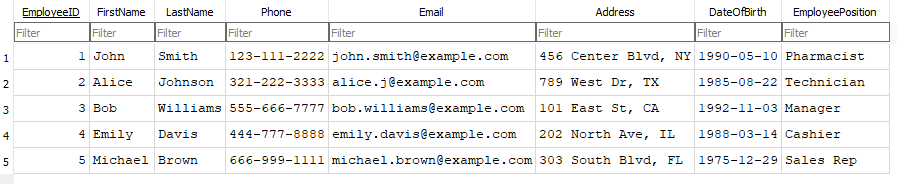
10) Delete Employee

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Searches for the employee.
3. Reviews the employee record.
4. Clicks 'Delete' and confirms deletion.

DELETE FROM Employees

WHERE EmployeeID = 6;



11) Update Employee Email

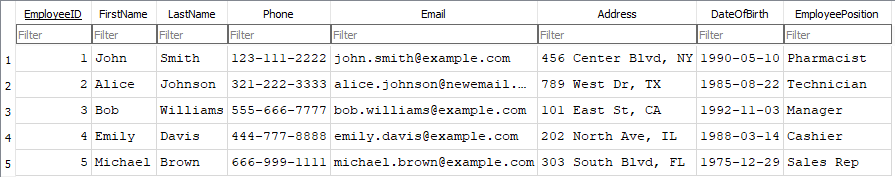
Actor: Manager  
Steps:

1. Manager logs into the system.
2. Searches for the employee by Employee ID.
3. Updates the employee's email address.
4. Reviews the updated details.
5. Saves the changes.

UPDATE Employees

SET Email = 'alice.johnson@newemail.com'

WHERE EmployeeID = 2;



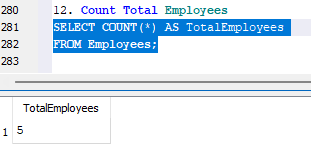
12) Count Total Employees (AGGREGATE)

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the reports section.
3. Runs the report to count all employees.
4. Reviews the report output.

SELECT COUNT(\*) AS TotalEmployees

FROM Employees;



**7.4) Insurances Use Cases**

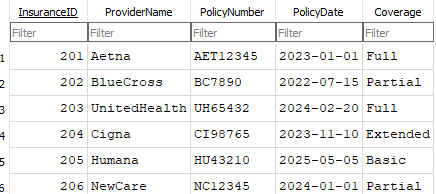
13) Add New Insurance Provider

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the insurance management section.
3. Enters new insurance provider information.
4. Reviews the details.
5. Clicks 'Save' to add the provider.

INSERT INTO Insurances VALUES

(206, 'NewCare', 'NC12345', '2024-01-01', 'Partial');



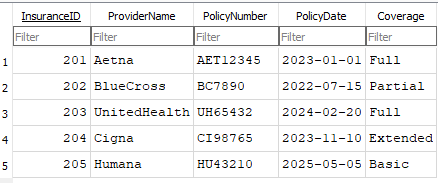
14) Delete Insurance Provider

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Searches for the insurance provider.
3. Reviews the provider's information.
4. Clicks 'Delete' and confirms.

DELETE FROM Insurances

WHERE InsuranceID = 206;



15) Update Insurance Coverage

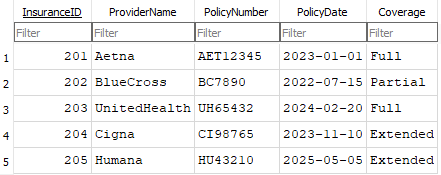
Actor: Manager  
Steps:

1. Manager logs into the system.
2. Searches for the insurance provider.
3. Updates the coverage details.
4. Reviews changes.
5. Saves the updated information.

UPDATE Insurances

SET Coverage = 'Extended'

WHERE InsuranceID = 205;



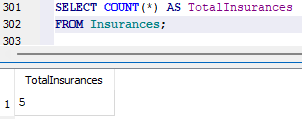
16) Count Insurance Providers (AGGREGATE)

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the reports section.
3. Selects insurance data.
4. Runs the report to count providers.

SELECT COUNT(\*) AS TotalInsurances

FROM Insurances;



**7.5) Customers Use Cases**

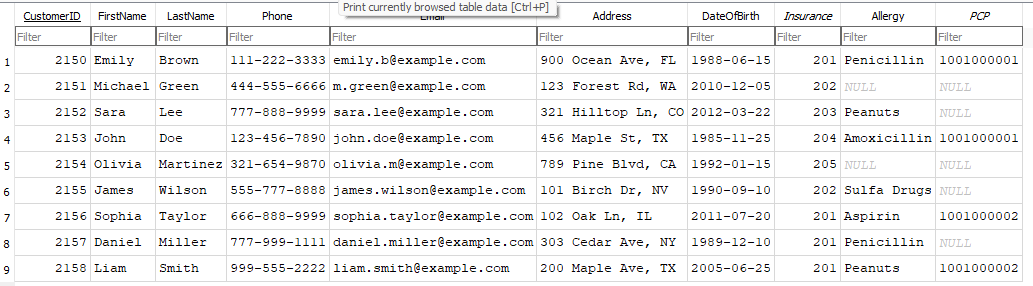
17) Add New Customer

Actor: Technician  
Steps:

1. Technician logs into the system.
2. Navigates to the customer management section.
3. Inputs new customer information.
4. Reviews the information.
5. Clicks 'Save' to add the customer.

INSERT INTO Customers VALUES

(2158, 'Liam', 'Smith', '999-555-2222', 'liam.smith@example.com', '200 Maple Ave, TX', '2005-06-25', 201, 'Peanuts', '1001000002');



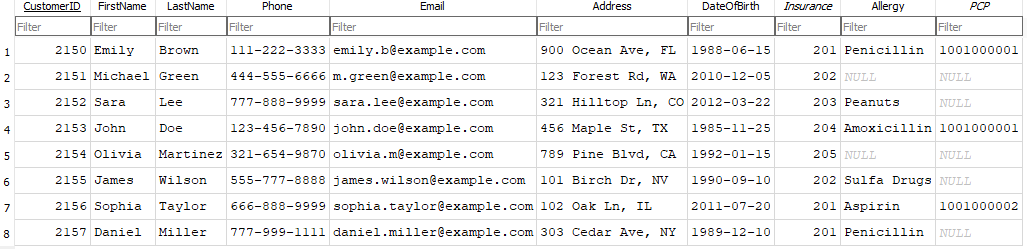
18) Delete Customer

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Searches for the customer.
3. Reviews the customer information.
4. Clicks 'Delete' and confirms.

DELETE FROM Customers

WHERE CustomerID = 2158;



19. Update Customer Phone Number

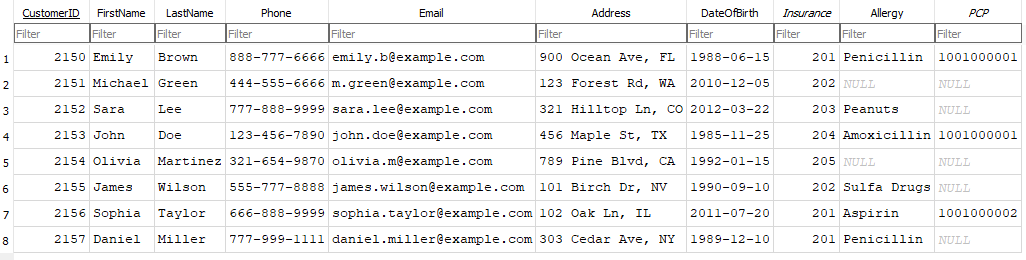
Actor: Technician  
Steps:

1. Technician logs into the system.
2. Searches for the customer by ID.
3. Updates the customer's phone number.
4. Reviews the updated information.
5. Saves the changes.

UPDATE Customers

SET Phone = '888-777-6666'

WHERE CustomerID = 2150;



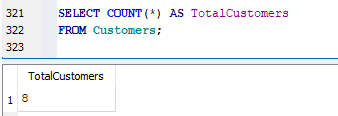
20) Count Total Customers (AGGREGATE)

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the reports section.
3. Runs the report to count all customers.

SELECT COUNT(\*) AS TotalCustomers

FROM Customers;



**7.6) Prescribers Use Cases**

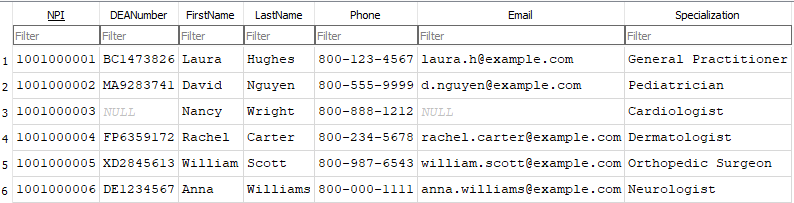
21) Add New Prescriber

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the prescriber management section.
3. Inputs new doctor information.
4. Reviews the information.
5. Clicks 'Save' to add the prescriber.

INSERT INTO Prescribers VALUES

('1001000006', 'DE1234567', 'Anna', 'Williams', '800-000-1111', 'anna.williams@example.com', 'Neurologist');



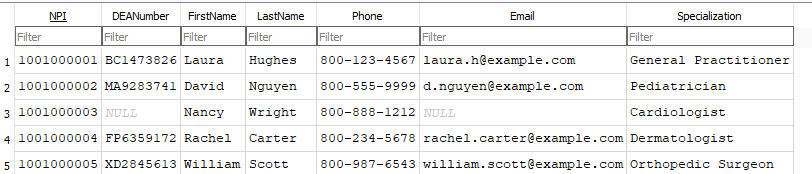
22) Delete Prescriber

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Searches for the prescriber.
3. Reviews the prescriber information.
4. Clicks 'Delete' and confirms.

DELETE FROM Prescribers

WHERE NPI = '1001000006';



23) Update Prescriber Email

Actor: Manager  
Steps:

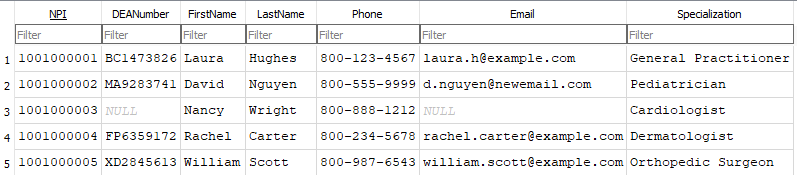
1. Manager logs into the system.
2. Searches for the prescriber by NPI.
3. Updates the prescriber's email.
4. Reviews the changes.

Saves the updated information.

UPDATE Prescribers

SET Email = 'd.nguyen@newemail.com'

WHERE NPI = '1001000002';



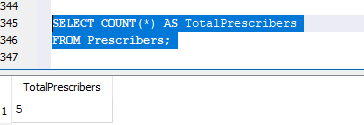
24) Count Total Prescribers (AGGREGATE)

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the reports section.
3. Runs the report to count prescribers.

SELECT COUNT(\*) AS TotalPrescribers

FROM Prescribers;



**7.7) Transactions Use Cases**

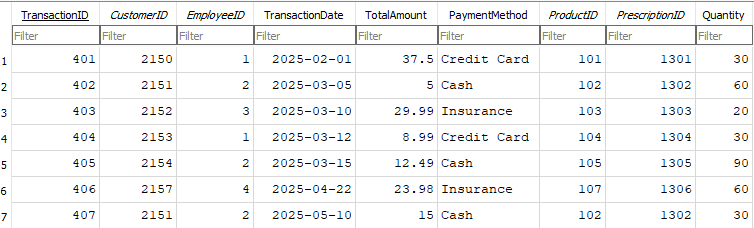
25) Add New Transaction

Actor: Cashier  
Steps:

1. Cashier logs into the system.
2. Navigates to the transaction processing section.
3. Inputs the customer, employee, product, prescription, and payment details.
4. Reviews the information.
5. Clicks 'Save' to record the transaction.

INSERT INTO Transactions VALUES

(407, 2151, 2, '2025-05-10', 15.00, 'Cash', 102, 1302, 30);



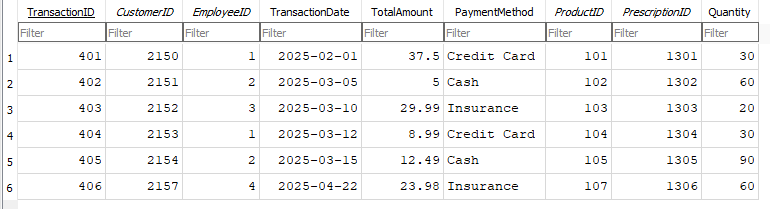
26) Delete Transaction

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Searches for the transaction.
3. Reviews the transaction details.
4. Clicks 'Delete' and confirms.

DELETE FROM Transactions

WHERE TransactionID = 407;



27) Update Transaction Payment Method

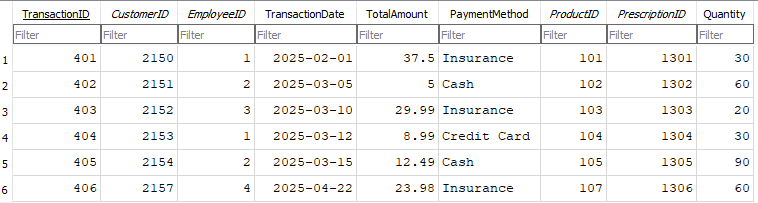
Actor: Cashier  
Steps:

1. Cashier logs into the system.
2. Searches for the transaction.
3. Updates the payment method.
4. Reviews the changes.
5. Saves the updated transaction.

UPDATE Transactions

SET PaymentMethod = 'Insurance'

WHERE TransactionID = 401;



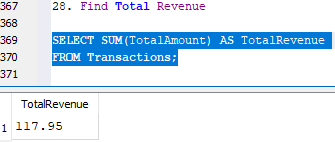
28) Find Total Revenue (AGGREGATE)

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the reports section.
3. Selects transaction data.
4. Runs the report to calculate total revenue.
5. Reviews the output.

SELECT SUM(TotalAmount) AS TotalRevenue

FROM Transactions;



**7.8) Prescription Use Cases**

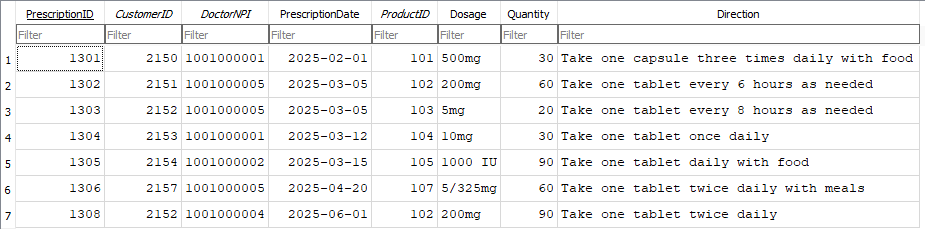
29) Add New Prescription

Actor: Pharmacist  
Steps:

1. Pharmacist logs into the system.
2. Navigates to the prescription entry section.
3. Inputs new prescription details including Customer ID, Doctor NPI, Prescription Date, Product ID, Dosage, Quantity, and Directions.
4. Reviews the entered information.
5. Clicks the 'Save' button to store the prescription.

INSERT INTO Prescriptions VALUES

(1308, 2152, '1001000004', '2025-06-01', 102, '200mg', 90, 'Take one tablet twice daily');



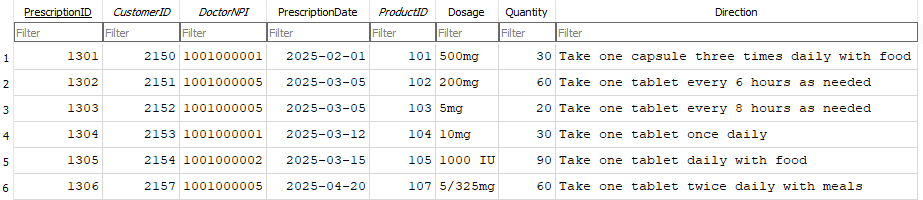
30) Delete Prescription

Actor: Pharmacist  
Steps:

1. Pharmacist logs into the system.
2. Navigates to the prescription management section.
3. Searches for the prescription using Prescription ID.
4. Reviews the prescription details.
5. Selects the delete option.
6. Confirms the deletion.

DELETE FROM Prescriptions

WHERE PrescriptionID = 1308;



31) Update Prescription Dosage

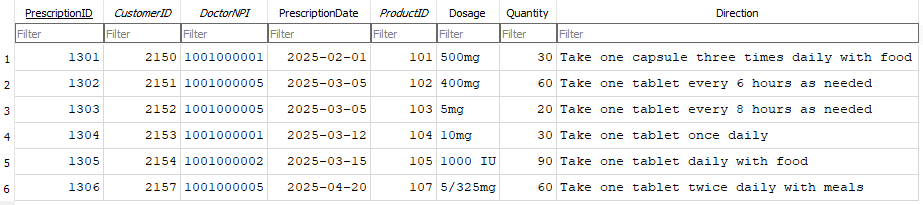
Actor: Pharmacist  
Steps:

1. Pharmacist logs into the system.
2. Navigates to the prescription management section.
3. Searches for the prescription using Prescription ID.
4. Updates the dosage details.
5. Reviews changes.
6. Saves the updated prescription.

UPDATE Prescriptions

SET Dosage = '400mg'

WHERE PrescriptionID = 1302;



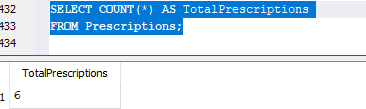
32) Count Total Prescriptions (AGGREGATE)

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the reports section.
3. Selects the prescription data report.
4. Runs the report to count all prescriptions.
5. Reviews the report output.

SELECT COUNT(\*) AS TotalPrescriptions

FROM Prescriptions;



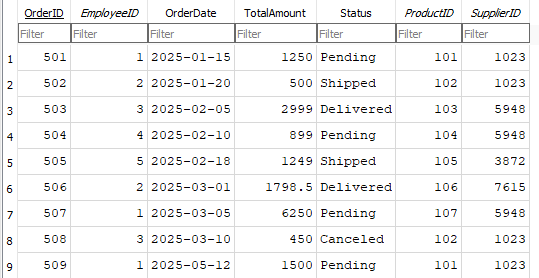
33) Add New Order

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the order management section.
3. Inputs new order details including Employee ID, Order Date, Total Amount, Status, Product ID, and Supplier ID.
4. Reviews the entered order information.
5. Clicks 'Save' to store the new order in the system.

INSERT INTO Orders (OrderID, EmployeeID, OrderDate, TotalAmount, Status, ProductID, SupplierID)

VALUES (509, 1, '2025-05-12', 1500.00, 'Pending', 101, 1023);



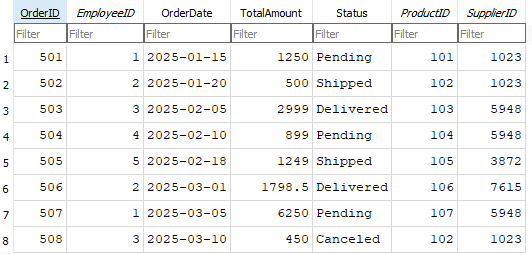
34) Delete Order

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the order management section.
3. Searches for the order using the Order ID.
4. Reviews the order details.
5. Clicks 'Delete' and confirms the deletion.

DELETE FROM Orders

WHERE OrderID = 509;



35) Update Order Status

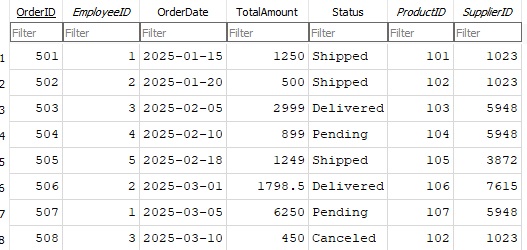
Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the order management section.
3. Searches for the order by Order ID.
4. Updates the order status to 'Shipped'.
5. Reviews the changes.
6. Saves the updated order information.

UPDATE Orders

SET Status = 'Shipped'

WHERE OrderID = 501;



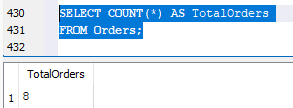
36) Count Total Orders (AGGREGATE)

Actor: Manager  
Steps:

1. Manager logs into the system.
2. Navigates to the reports or analytics section.
3. Selects the order data report.
4. Runs the report to count all existing orders.
5. Reviews the total count.

SELECT COUNT(\*) AS TotalOrders

FROM Orders;



**7.9) Join Queries**

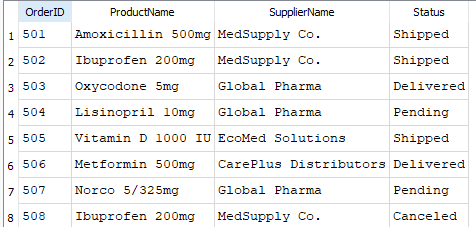
37) List Orders with Product, Supplier Info, and Status

SELECT o.OrderID, p.ProductName, s.Name AS SupplierName, o.OrderDate, o.Status

FROM Orders o

JOIN Product\_Inventory p ON o.ProductID = p.ProductID

JOIN Suppliers s ON o.SupplierID = s.SupplierID;



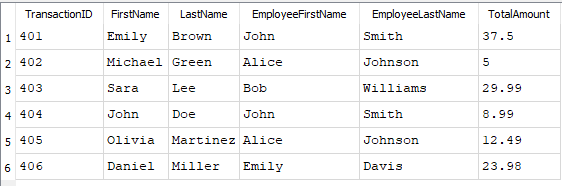
38) List Transactions with Customer and Employee Info

SELECT t.TransactionID, c.FirstName, c.LastName, e.FirstName AS EmployeeFirstName, e.LastName AS EmployeeLastName, t.TotalAmount

FROM Transactions t

JOIN Customers c ON t.CustomerID = c.CustomerID

JOIN Employees e ON t.EmployeeID = e.EmployeeID;



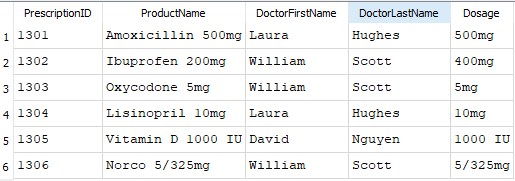
39) Prescriptions with Product and Doctor Info

SELECT pr.PrescriptionID, p.ProductName, d.FirstName AS DoctorFirstName, d.LastName AS DoctorLastName, pr.Dosage

FROM Prescriptions pr

JOIN Product\_Inventory p ON pr.ProductID = p.ProductID

JOIN Prescribers d ON pr.DoctorNPI = d.NPI;

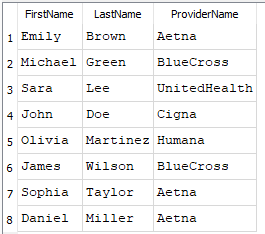


40) Customers with Insurance Providers

SELECT c.FirstName, c.LastName, i.ProviderName

FROM Customers c

LEFT JOIN Insurances i ON c.Insurance = i.InsuranceID;



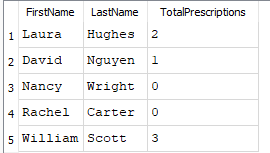
41) Doctors and Prescription Counts

SELECT d.FirstName, d.LastName, COUNT(pr.PrescriptionID) AS TotalPrescriptions

FROM Prescribers d

LEFT JOIN Prescriptions pr ON d.NPI = pr.DoctorNPI

GROUP BY d.NPI;



42) C2 Controlled Substance Transactions

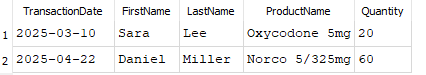
SELECT t.TransactionDate, c.FirstName, c.LastName, p.ProductName, t.Quantity

FROM Transactions t

JOIN Customers c ON t.CustomerID = c.CustomerID

JOIN Product\_Inventory p ON t.ProductID = p.ProductID

WHERE p.Category = 'C2';



43) Recalled Products with Supplier Contact Info

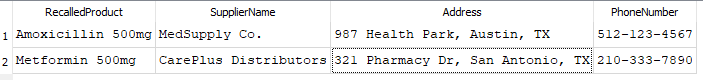
SELECT p.ProductName AS RecalledProduct, s.Name AS SupplierName, s.Address, s.PhoneNumber

FROM Product\_Inventory p

JOIN Orders o ON p.ProductID = o.ProductID

JOIN Suppliers s ON o.SupplierID = s.SupplierID

WHERE p.Recall = TRUE;



44) List all transactions that were paid using Insurance

SELECT t.TransactionID, c.FirstName, c.LastName, t.TotalAmount

FROM Transactions t

JOIN Customers c ON t.CustomerID = c.CustomerID

WHERE t.PaymentMethod = 'Insurance';



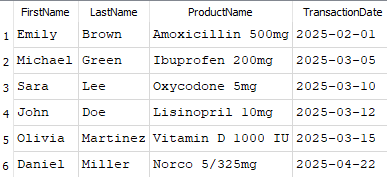
45) Find Customers who made Transactions and what Products they bought

SELECT c.FirstName, c.LastName, p.ProductName, t.TransactionDate

FROM Transactions t

JOIN Customers c ON t.CustomerID = c.CustomerID

JOIN Product\_Inventory p ON t.ProductID = p.ProductID;



**8. Conclusion**

The successful design and implementation of the Happy Drugstore DBMS represent a significant achievement toward enhancing operational efficiency, ensuring data integrity, and improving customer service within the pharmacy environment. This database management system was carefully constructed to align with the purpose, mission, and objectives established from the beginning of the project.

The DBMS was designed to manage critical data groups, including customer information, prescriptions, product inventory, employee records, prescriber details, insurance policies, sales transactions, supplier relationships, and compliance tracking. It provides secure, scalable, and efficient access to information across various internal users such as managers, pharmacists, technicians, and cashiers.

Key considerations such as data security, patient privacy, compliance with regulatory standards, cost control, user training, and future scalability were systematically addressed to ensure the DBMS not only meets immediate operational needs but also remains sustainable and adaptable to future growth.

The mission objectives were fulfilled by enabling comprehensive data management capabilities:

* Full CRUD (Create, Read, Update, Delete) operations across all entities.
* Search and reporting functionalities to support decision-making.
* Inventory tracking and recall management for safe drug dispensation.
* Insurance and prescription management to streamline billing and compliance.
* Order and supplier tracking for effective procurement.

Additionally, careful normalization (up to BCNF) ensured data integrity and eliminated redundancy. Join queries were crafted to support complex reporting needs, such as tracking C2 controlled substances and managing drug recalls.

This comprehensive effort culminates in a robust DBMS ready for deployment, ensuring Happy Drugstore can operate more efficiently, deliver better customer experiences, and maintain high standards of legal compliance and data protection.

The Happy Drugstore DBMS project stands as a strong foundation for future enhancements, such as advanced analytics, mobile integration, and broader regulatory reporting modules.

**Demo:** [Recording-20250503\_162155.webm](https://uhdowntown-my.sharepoint.com/:v:/g/personal/chemalir1_gator_uhd_edu/ET5ySXTn5PVIpsBO-qCiY3kBHieUDiDvM_4f6w2MbHbGEw?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJTdHJlYW1XZWJBcHAiLCJyZWZlcnJhbFZpZXciOiJTaGFyZURpYWxvZy1MaW5rIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXcifX0%3D&e=hXmCV0)

**9. References**

1. SQLite Documentation - https://www.sqlite.org/docs.html