**Phase 2: Happy Drugstore DBMS Design**

Team members: Yoonjoung Kim, Rimon Chemali, Linda Neeley

**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 regulatory compliance 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 Regulatory Compliance data

**4. Major User Views**

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

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

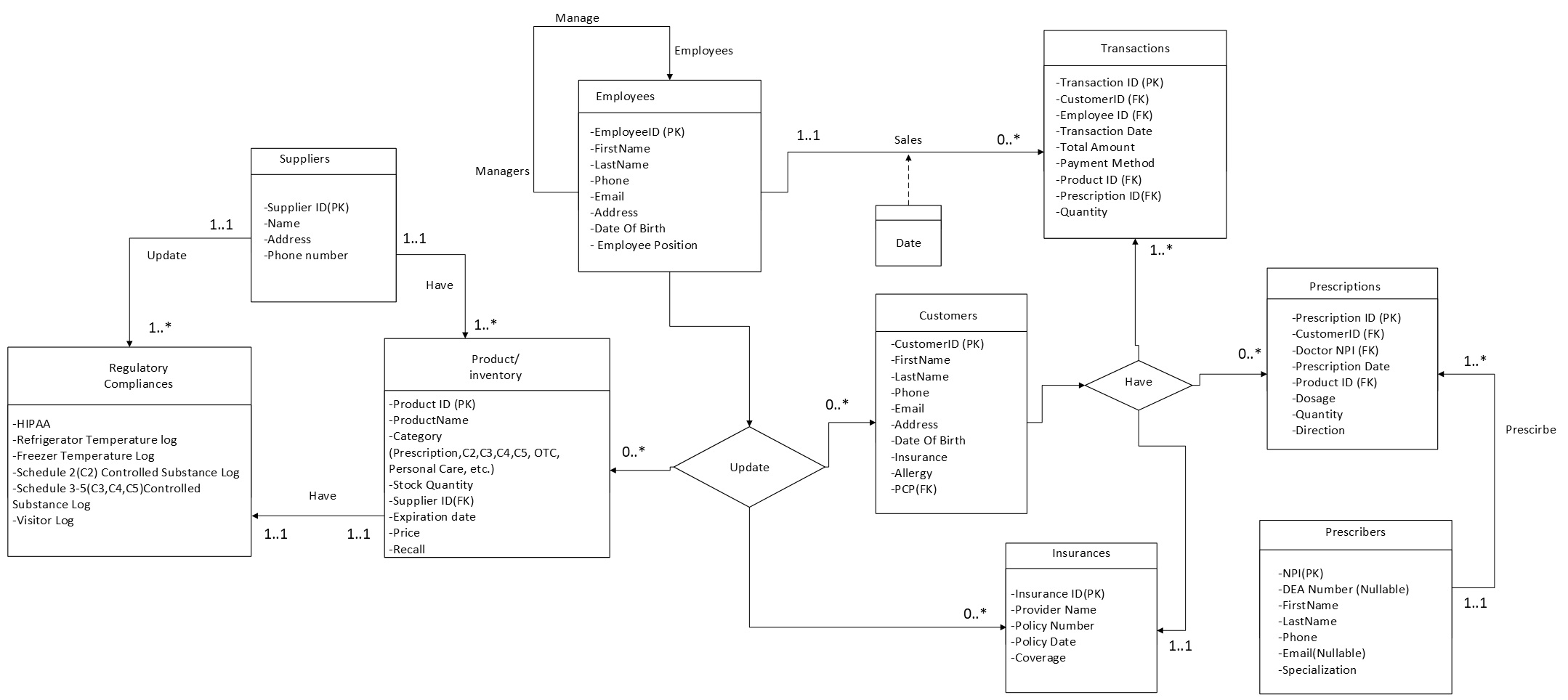
* Maintain (Add, Update, Delete)
* Query (Read-Only Access – Retrieve Data)
* Report (Generate Summaries)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 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 (Inventory)** | Maintain | x | x |  | x |  |  |
|  | Query | x | x | x | x | x |  |
|  | Report | x |  |  | x |  |  |
| **Regulatory** **Compliance** | Maintain | x |  |  |  |  |  |
|  | Query | 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.**

**✓ Specify attribute domains, primary keys, foreign keys and the parent tables referenced by each foreign key.**

**SQLite does not support the CREATE DOMAIN command, so constraints must be defined directly within the table structure.**

**1) Regulatory\_Compliances**

CREATE TABLE Regulatory\_Compliances (

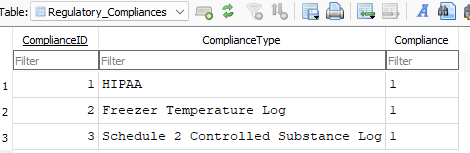
ComplianceID INT NOT NULL,

ComplianceType VARCHAR(100) NOT NULL,

Compliance BOOLEAN DEFAULT NULL

PRIMARY KEY (ComplianceID)

);



* **Primary Key:** ComplianceID
* **Functional Dependency:** ComplianceID → ComplianceType, Compliance

Compliance Type -> Compliance

* **Example Dependency:** ComplianceID 1 is associated with 'HIPAA', thus ComplianceType does not depend on any other attributes.
* **1NF:** Each cell contains atomic values, ensuring no multi-valued attributes.
* **2NF:** ComplianceType is fully dependent on ComplianceID, the primary key.
* **3NF:** There is no transitive dependency.
* **BCNF:** The primary key (ComplianceID) uniquely determines ComplianceType, ensuring no functional dependency violations.

**2) Suppliers**

CREATE TABLE Suppliers (

SupplierID INT NOT NULL,

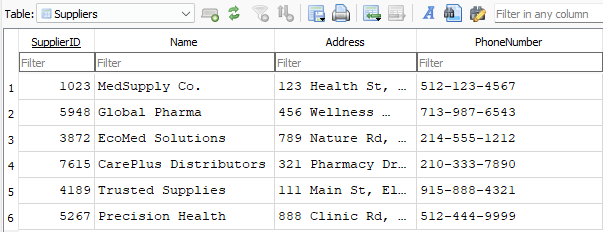
Name VARCHAR(100),

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.

**3) Product\_Inventory**

CREATE TABLE Product\_Inventory (

ProductID INT NOT NULL,

ProductName VARCHAR(100),

Category VARCHAR(50),

StockQuantity INT,

SupplierID INT,

ExpirationDate DATE,

Price DECIMAL(10,2),

Recall BOOLEAN DEFAULT FALSE,

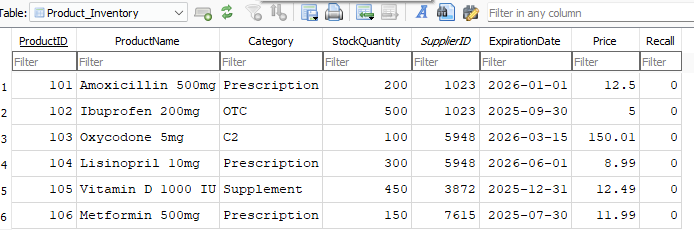
PRIMARY KEY (ProductID),

FOREIGN KEY (SupplierID)

REFERENCES Suppliers(SupplierID)

ON UPDATE CASCADE

ON DELETE SET NULL

);

* **Primary Key:** ProductID
* **Functional Dependency:** ProductID → ProductName, Category, StockQuantity, SupplierID, ExpirationDate, Price, Recall
* **Example Dependency:** ProductID 101 is associated with 'Amoxicillin 500mg', proving ProductName does not depend on Category.
* **1NF:** No multi-valued attributes exist; each cell holds atomic data.
* **2NF:** All attributes are fully dependent on ProductID, ensuring compliance.
* **3NF:** No attribute depends on another non-primary attribute, eliminating transitive dependencies.
* **BCNF:** Every determinant uniquely determines its attributes, fulfilling BCNF requirements.

**4) Employees**

CREATE TABLE Employees (

EmployeeID INT NOT NULL,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Phone VARCHAR(15),

Email VARCHAR(100),

Address VARCHAR(255),

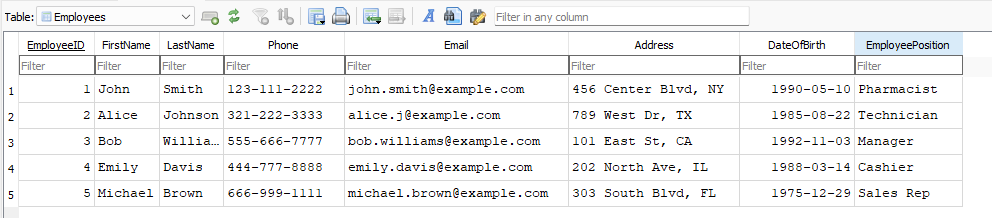
DateOfBirth DATE,

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.

**5) Insurances**

CREATE TABLE Insurances (

InsuranceID INT NOT NULL,

ProviderName VARCHAR(100),

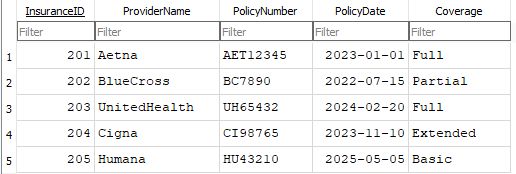
PolicyNumber VARCHAR(50),

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) 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),

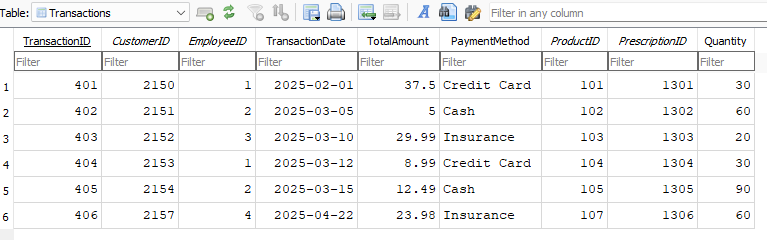
FOREIGN KEY (ProductID)

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

**7) 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)

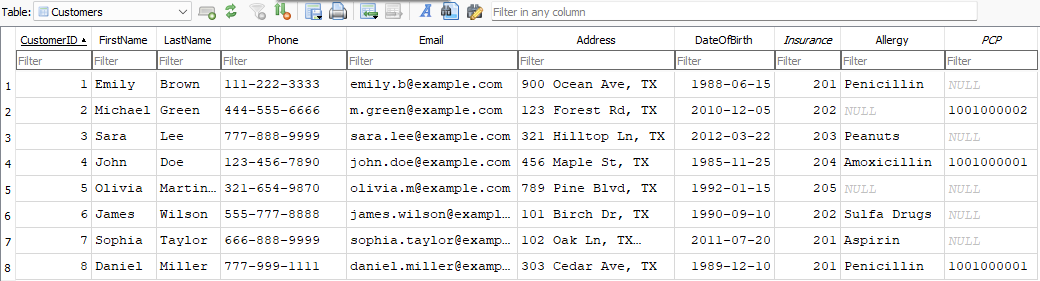
REFERENCES Insurances(InsuranceID)

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.

**8) Prescribers**

CREATE TABLE Prescribers (

NPI VARCHAR(10) NOT NULL,

DEANumber VARCHAR(9),

FirstName VARCHAR(50),

LastName VARCHAR(50),

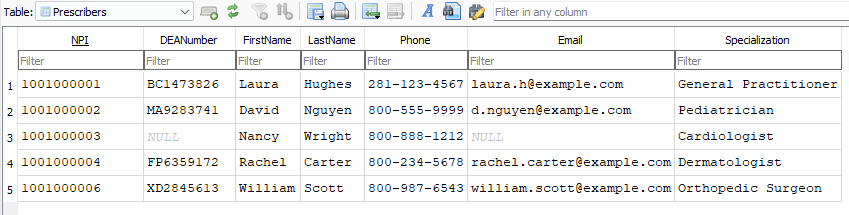
Phone VARCHAR(15),

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.

**9) 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),

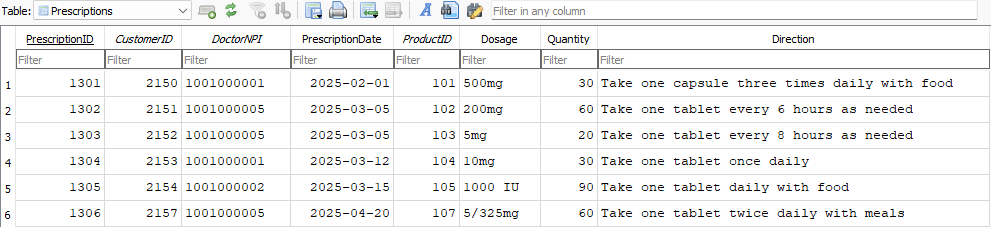
FOREIGN KEY (DoctorNPI)

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.

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

**7. Complete List of Use Cases**

✓ actor and step-by-step descriptions for every use case. Number all use cases. Minimally, you must have

➢ for each entity, at least one insert, one delete, one update, and one aggregate query ➢ and for each set of entities that have a direct relationship, one joint query

✓ put the needed SQL statements after each use case.

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' and products' information.
* Customer: Engages with the system and makes drug purchases.

**7.1) Employee information**

1. Total Transactions by Employees

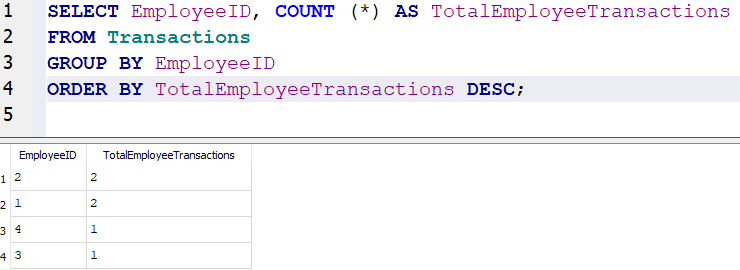
Actor: Manager, Sales Rep

Steps:

1. Manger/Sales Rep logs into system.

2. Navigate to the employee records.

3. View total transactions by each employee.



User: Manager

Steps:

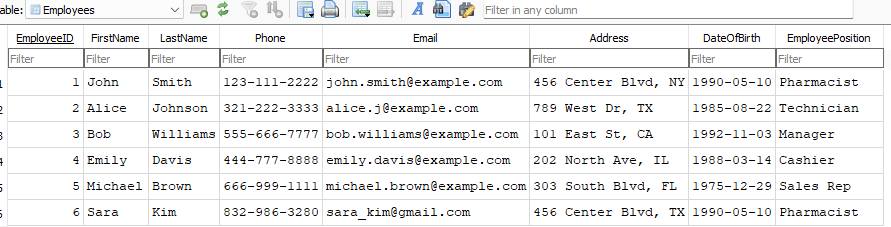
1. Manager logs into the system.

2. Navigate to the employee records.

3. View the total transactions made by each employee.

INSERT INTO Employees

VALUES (6, 'Sara', 'Kim', '832-986-3280’, 'sara\_kim@gmail.com', '456 Center Blvd, TX', '1990-05-10', 'Pharmacist');



3. Remove Employee

Actor: Manager

Steps:

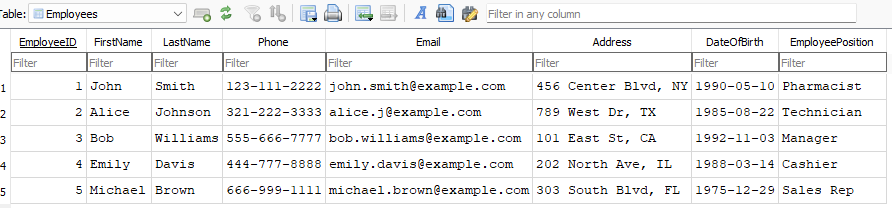
1. Manager enters the system.

2. Search for an employee using employee ID.

3. Delete the employee record.

DELETE FROM Employees

WHERE EmployeeID = 4;



4. Update Employee Phone number

Actor: Manager

Steps:

1. Manager enters the system.

2. Search for employee using employee ID.

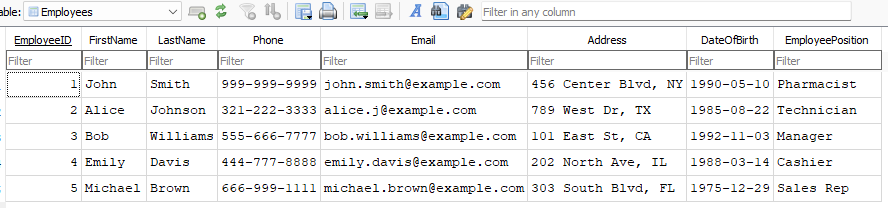
3. Update employee phone number.

4. Save changes made.

UPDATE Employees

SET Phone = ‘999-999-9999’

WHERE EmployeeID = 1;



**7.2) Suppliers**

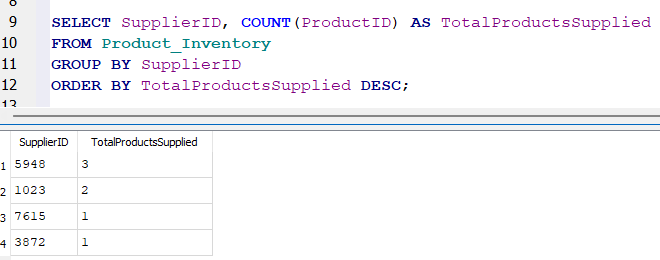
5. Total Products From Suppliers in descending order

Actor: Pharmacist

Steps:

1. Pharmacists opens the system.

2. Review product numbers from each supplier.



6. Add New Supplier

Actor: Manager

Steps:

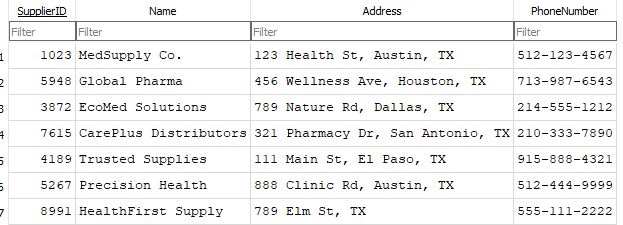
1. Manager enters the system.

2. Insert new supplier information.

3. Save new supplier information.

INSERT INTO Suppliers VALUES

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



7. Remove Supplier

Actor: Manager

Steps:

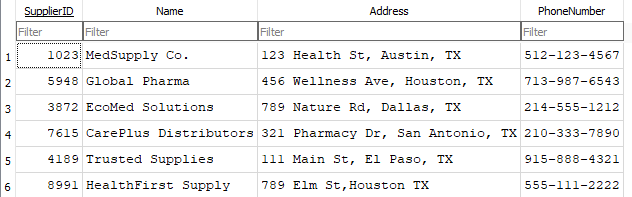
1. Manager searches for suppliers in the system.

2. Delete supplier using supplier ID.

3. Confirm all changes.

DELETE FROM Suppliers

WHERE SupplierID = 5267;



8. Update Supplier Address

Actor: Manager

Steps:

1. Manager searches for suppliers on the system using supplier ID.

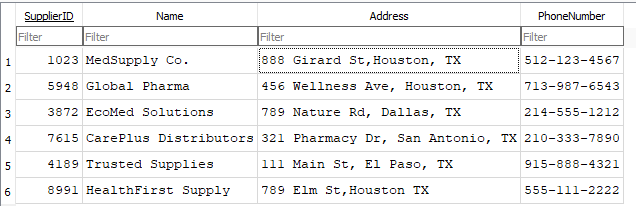
2. Update the address.

3. Save all changes.

UPDATE Suppliers

SET Address = '888 Girard St, Houston, TX'

WHERE SupplierID = 1023;



**7.3) Transactions**

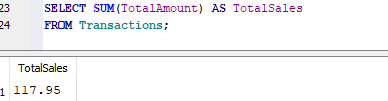
9. Find Total Sales

Actor: Manager

Steps:

1. Manager accesses transactions.

2. View total sales from all transactions.



10. Find Total Cash Sales

Actor: Manager

Steps:

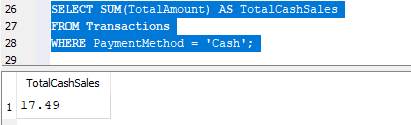
1. Manager searches transactions by payment method.

2. View total from total cash sales.

SELECT SUM(TotalAmount) AS TotalCashSales

FROM Transactions

WHERE PaymentMethod = 'Cash';



11. Add New Transaction

Actor: Cashier

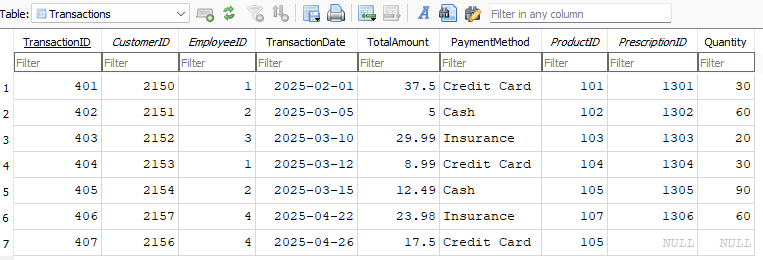
Steps:

1. Cashier enters transactions details.

2. Save new transaction information.

INSERT INTO Transactions VALUES

(407, 2156, 4, '2025-04-26', 17.50, 'Credit Card', 105, NULL, NULL);



12. Void Transaction

Actor: Manager

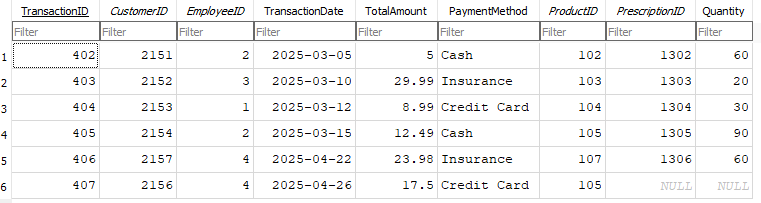
Steps:

1. Manager searches for transactions using transaction ID.

2. Delete selected transaction and save changes.

DELETE FROM Transactions

WHERE TransactionID = 401;



13. Update Transaction Total

Actor: Manager

Steps:

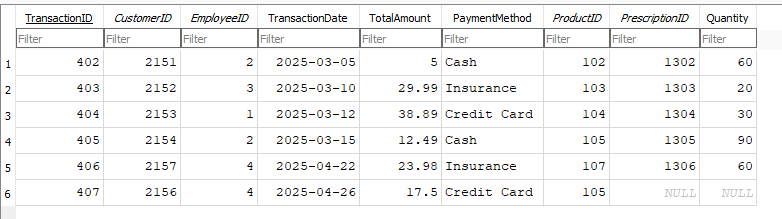
1. Manager searches up transaction using transaction ID.

2. Update total amount to correct value and save changes.

UPDATE Transactions

SET TotalAmount = 39.89

WHERE TransactionID = 404;



**7.4) Prescriptions**

14. Tracking Prescriptions by Customer:

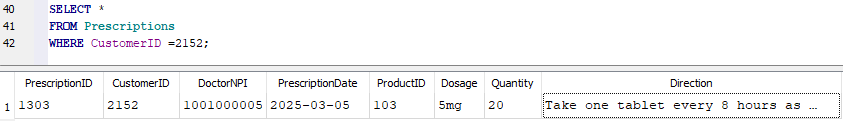
Actor: Pharmacists

Steps:

1. Pharmacist enters the system.

2. Search customer using customer’s ID.

3. View prescriptions from customer.



15. Add Prescriptions

Actor: Pharmacist

Steps:

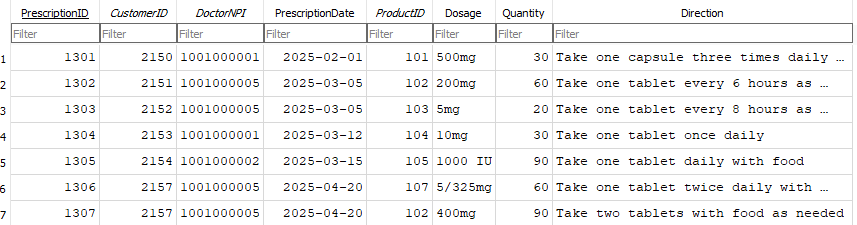
1. Pharmacist enters the system.

2. Insert prescriptions values.

3. Confirm and save information.

INSERT INTO Prescriptions VALUES

(1307, 2157, 1001000005, '2025-04-20', 102, ‘400mg’, 90, 'Take two tablets with food as needed');



16. Delete Prescription

Actor: Pharmacist

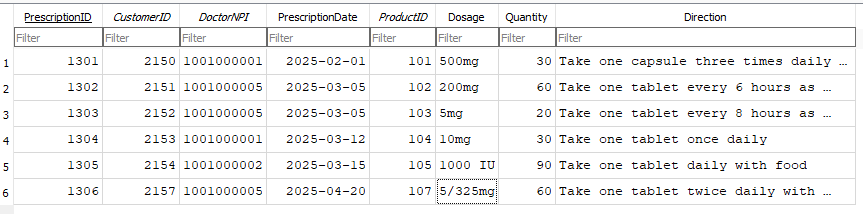
Steps:

1. Pharmacist searches for prescription using prescription ID.

2. Delete prescription and confirm changes.

DELETE FROM Prescriptions

WHERE PrescriptionID = 1307;



17. Providing Dosage Directions for Patients

Actor: Pharmacist

Steps:

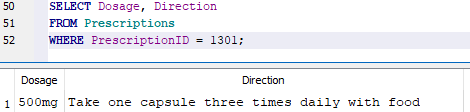
1. Pharmacists searches prescription using prescription ID.

2. View prescription dosages and directions.

SELECT Dosage, Direction

FROM Prescriptions

WHERE PrescriptionID = 1301;



**7.5) Prescribers**

18. Find a Prescriber

Actor: Technician

Steps:

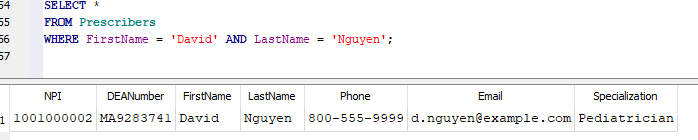
1. Technician searches for prescriber using first and last name.

2. View prescriber information.

SELECT \*

FROM Prescribers

WHERE FirstName = 'David' AND LastName = 'Nguyen';



19. Add New Doctor

Actor: Manager

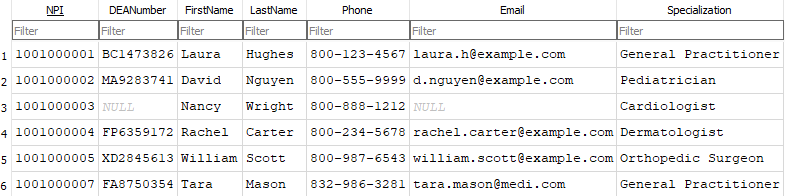
Steps:

1. Manager inserts new values into prescribers' data.

2. Confirm and save all new changes.

INSERT INTO Prescribers VALUES

(1001000007, 'FA8750354', 'Tara', 'Mason', '832-986-3281, 'tara.mason@medi.com', 'General Practitioner');



20. Delete Doctor

Actor: Manager

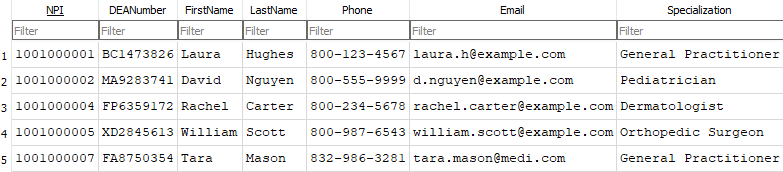
Steps:

1. Manager searches prescribers using NPI.

2. Delete selected prescriber and save changes.

DELETE FROM Prescribers

WHERE NPI = 1001000003;



21. Update Doctor Phone

Actor: Technician

Steps:

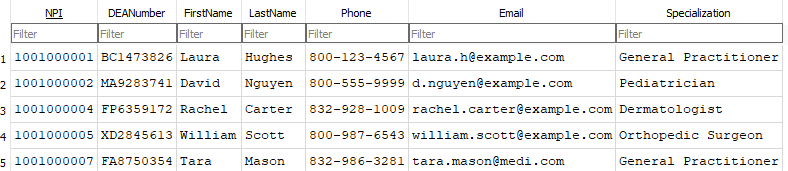
1. Technician searches for prescribers using NPI.

2. Update prescriber number and save changes.

UPDATE Prescribers

SET Phone = ‘832-928-1009'

WHERE NPI= 1001000004;



**7.6) Insurances**

22. Find Total Policies Per Insurance Provider

Actor: Manager

Steps:

1. Manager searches insurances using provider name.

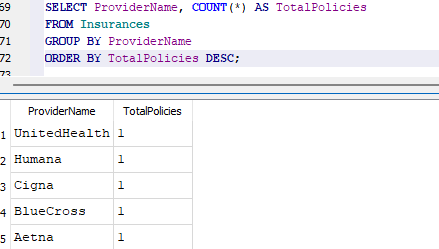
2. Total number of policies per provider are displayed.

SELECT ProviderName, COUNT(\*) AS TotalPolicies

FROM Insurances

GROUP BY ProviderName

ORDER BY TotalPolicies DESC;



23. Add New Insurance Provider

Actor: Manager

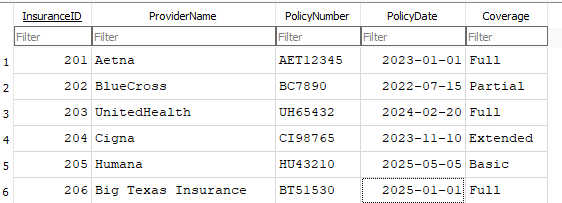
Steps:

1. Manager inserts values into the new insurance profile.

2. Confirm and submit information.

INSERT INTO Insurances

VALUES (206, 'Big Texas Insurance', 'BT51530', ‘2025-01-01’, 'Full');



24. Delete Insurance Provider

Actor: Manager

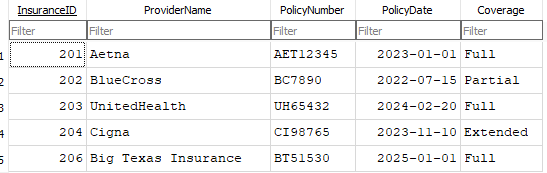
Steps:

1. Manager searches insurance provider using insurance ID.

2. Delete selected insurance provider and save data.

DELETE FROM Insurances

WHERE InsuraceID = 205;



25. Update Insurance

Actor: Manager

Steps:

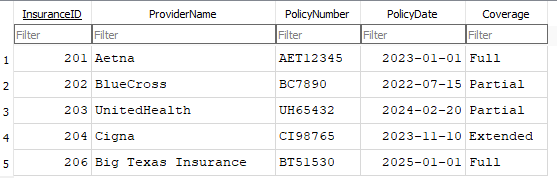
1. Manager searches for insurance by using the insurance ID.

2. Update coverage to partial and save data.

UPDATE Insurances

SET Coverage = ‘Partial’

WHERE InsuranceID = 203;



**7.7) Customers**

26. Find Total Number of Customers (Aggregate)

Actor: Sales Representative

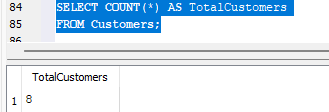
Steps:

1. Sales Rep opens the customer information portal.

2. View total number of customers.

SELECT COUNT(\*) AS TotalCustomers

FROM Customers;



27. Add New Customer

Actor: Cashier

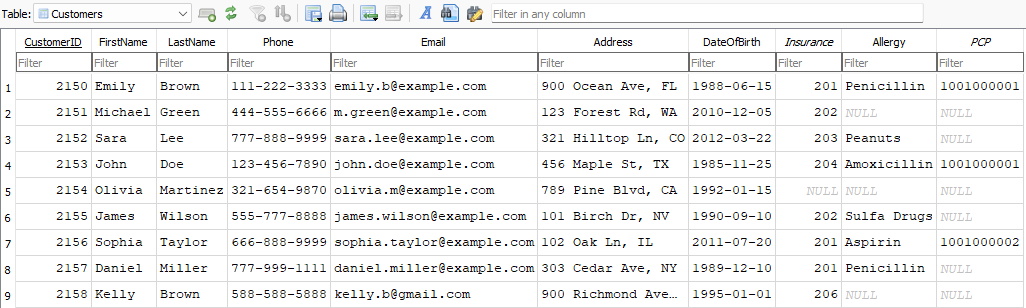
Steps:

1. Cashier enters new customer information.

2. Save and submit information.

INSERT INTO Customers VALUES

(2158, 'Kelly', 'Brown', '588-588-5888', 'kelly.b@gmail.com', '900 Richmond Ave, TX', ‘1995-01-01’, 206, NULL, NULL);



28. Delete Customer

Actor: Manager

Steps:

1. Manager searches for customers using customer ID.

2. Confirm and delete customer information from the database.

DELETE FROM Customers

WHERE CustomerID = 2158;



29. Update Customer Information

Actor: Cashier

Steps:

1. Cashier searches for customer using customer ID.

2. Update new phone number and save changes.

UPDATE Customers

SET Phone = ‘832-940-2940'

WHERE CustomerID = 2155;



**7.8) Product/Inventory**

30. Find Total Product Stock (Aggregate)

Actor: Pharmacists

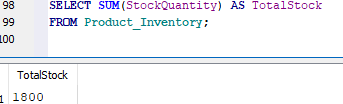
Steps:

1. Pharmacist opens the product inventory.

2. View the total stock available.

SELECT SUM(StockQuantity) AS TotalStock

FROM Product\_Inventory;



31. Add New Product

Actor: Pharmacist

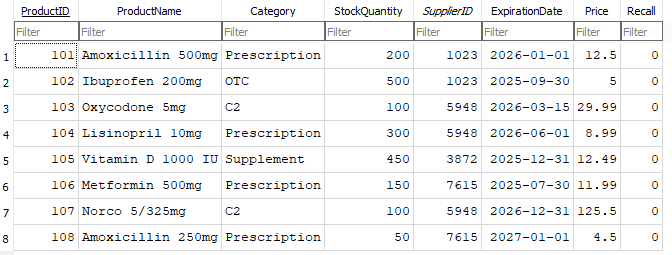
Steps:

1. Pharmacist enters product inventory values.

2. Confirm and save the data.

INSERT INTO Product\_Inventory VALUES

(108, 'Amoxicillin 250mg', 'Prescription', 50, 7615, '2027-01-01', 4.50, FALSE);



32. Delete Product

Actor: Manager

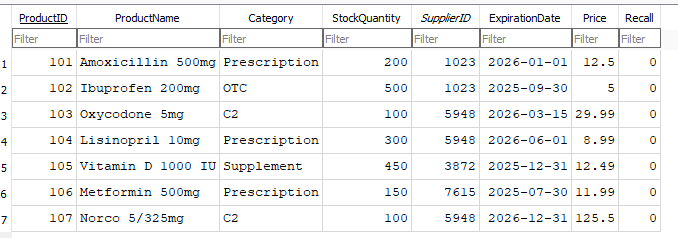
Steps:

1. Manager searches for product using product ID.

2. Delete selected product and save changes.

DELETE FROM Product\_Inventory

WHERE ProductID = 101;



33. Update Product Price

Actor: Manager

Steps:

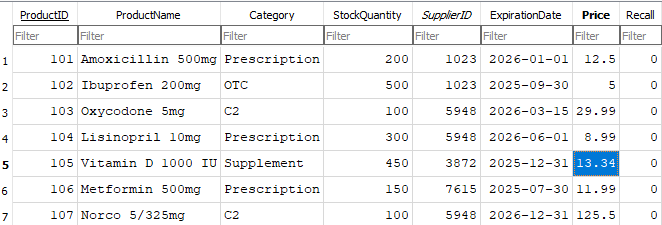
1. Manager searches for product using product ID.

2. Update new price and save changes.

UPDATE Product\_Inventory

SET Price = 13.34

WHERE ProductID = 105;



**7.9) Regulatory Compliances**

34. Find the Total Number of Regulatory Compliances

Actor: Manager

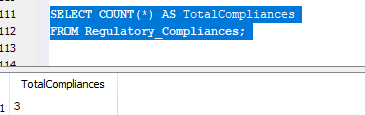
Steps:

1. Manager opens the compliances portal.

2. View total compliances.

SELECT COUNT(\*) AS TotalCompliances

FROM Regulatory\_Compliances;



35. Add New Regulatory Compliance

Actor: Manager

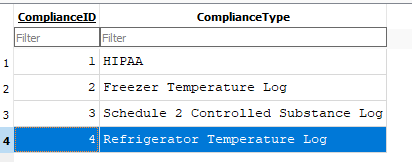
Steps:

1. Manager enters the compliance portal.

2. Enters new compliance values and saves changes.

INSERT INTO Regulatory\_Compliances VALUES

(4, 'Refrigerator Temperature Log');



36. Delete Compliance

Actor: Manager

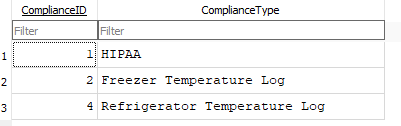
Steps:

1. Manager searches for compliance using compliance ID.

2. Delete selected compliance and save changes.

DELETE FROM Regulatory\_Compliances

WHERE ComplianceID = 3;



37. Update Compliance Status

Actor: Manager

Steps:

1. Search for compliance using compliance ID.

2. Update status to non-compliant.

3. Confirm and save changes.

UPDATE Regulatory\_Compliances

SET Status = 'Non-Compliant'

WHERE ComplianceID = 4728592;

**7.10) Joint Queries**

38. Customers and Prescriptions – Retrieve customer details and their prescriptions

Actor: Technician

Steps:

1. Technician combines customer and prescription information.

2. View combined data results.

SELECT

c.CustomerID,

c.FirstName,

c.LastName,

pr.PrescriptionID,

pr.PrescriptionDate,

pr.ProductID

FROM Customers AS c

JOIN Prescriptions AS pr ON c.CustomerID=pr.CustomerID;

39. Prescriptions and Products – Get details of products prescribed in prescriptions:

Actor: Pharmacist

Steps:

1. Pharmacist runs join query on prescription and prodcut data.

2. View results.

SELECT

pr.PrescriptionID,

pd.ProductName,

pd.Category,

pd.Price

FROM

Prescriptions AS pr

JOIN

Products AS pd ON pr.ProductID = pd.ProductID;

40. Products and Suppliers – Retrieve details of products and their suppliers.

Actor: Pharmacist

Steps:

1. Pharmacist queries joined supplier and product data.

2. View which supplier provides each product.

SELECT

pd.ProductID,

pd.ProductName,

s.Name AS SupplierName,

s.PhoneNumber

FROM

Products AS pd

JOIN

Suppliers AS s ON pd.SupplierID = s.SupplierID;

41. Transactions and Customers – Fetch transaction details along with customer information

Actor: Cashier

Steps:

1. Cashier joins transaction and customer data.

2. View transaction data with customer names.

SELECT

t.TransactionID,

t.TransactionDate,

t.TotalAmount,

c.FirstName,

c.LastName

FROM

Transactions AS t

JOIN

Customers AS c ON t.CustomerID = c.CustomerID;

42. Transactions and Employees – Link transactions with the employees handling them

Actor: Manager

Steps:

1. Manager review transaction handling details.

2. Use join to view employee per transaction.

SELECT

t.TransactionID,

t.TransactionDate,

t.TotalAmount,

e.FirstName AS EmployeeFirstName,

e.LastName AS EmployeeLastName,

e.Position

FROM

Transactions AS t

JOIN

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

43. Prescriptions and prescribers – Get prescription details and the prescriber who issued them

Actor: Pharmacist

Steps:

1. Pharmacist queries join on prescriptions and prescribers.

2. View issuing doctor for each prescription.

SELECT

pr.PrescriptionID,

pr.PrescriptionDate,

p.FirstName AS PrescriberFirstName,

p.LastName AS PrescriberLastName,

p.Specialization

FROM Prescriptions AS pr

JOIN Prescribers AS p ON pr.DoctorNPI = p.NPI;

44. Customers and Insurances – Retrieve customer details along with their insurance provider

Actor: Sales Representative

Steps:

1. Sales Rep joins customer and insurance data.

2. View results in a report.

SELECT

c.CustomerID,

c.FirstName,

c.LastName,

i.ProviderName,

i.PolicyNumber,

i.PolicyDate

FROM

Customers AS c

JOIN

Insurances AS i ON c.Insurance = i.InsuranceID;

45. Retrieve Schedule2(C2) Log – Keeping track of C2 transactions with Date, customer information, and quantity

Actor: Pharmacist

Steps:

1. Pharmacist queries all transactions involving C2 category drugs.

2. View customer, product, and quantity details.

SELECT

t.TransactionID,

t.TransactionDate,

t.Quantity,

c.FirstName AS PatientFirstName,

c.LastName AS PatientLastName,

p.ProductName,

p.Category

FROM Transactions t

JOIN Prescriptions pr ON t.PrescriptionID = pr.PrescriptionID

JOIN Customers c ON pr.CustomerID = c.CustomerID

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

WHERE p.Category = 'C2';

**8. Test Plan and Records**

Test plan and records. Verify every use case works as specified. Fix any bugs. Include the screen captures of the actual output of the following queries: (15 pts)

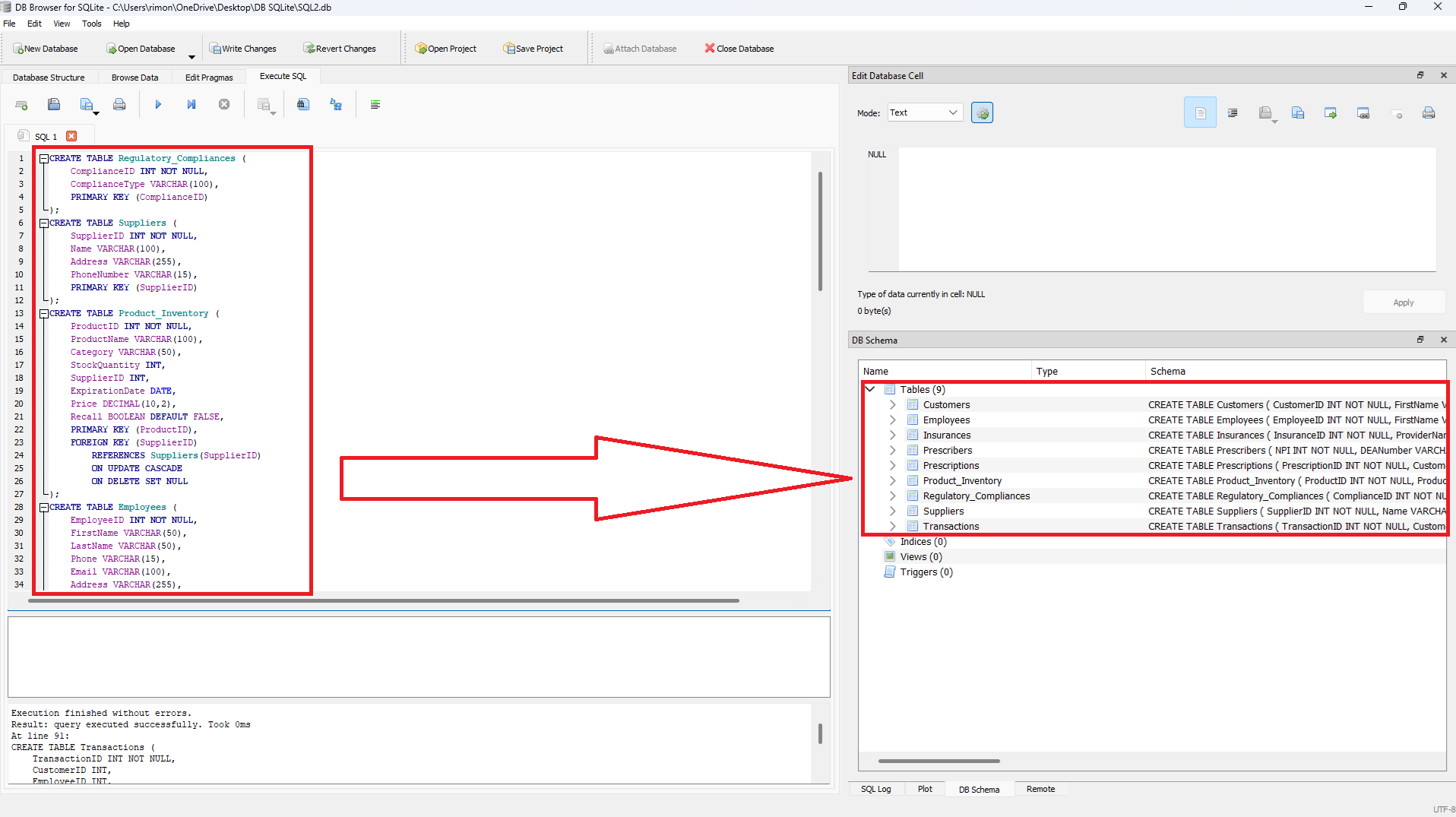
✓ show all data of each table

✓ one aggregate query for each table

✓ one joint query for each set of entities that have a direct relationship

This demonstrates the creation of a multi-user relational database system using SQL. It includes the creation of tables, insertion of sample data, and examples of joint queries. As mentioned earlier, the choice of DBMS used for this project is: **DB Browser for SQLite (DB4S)**

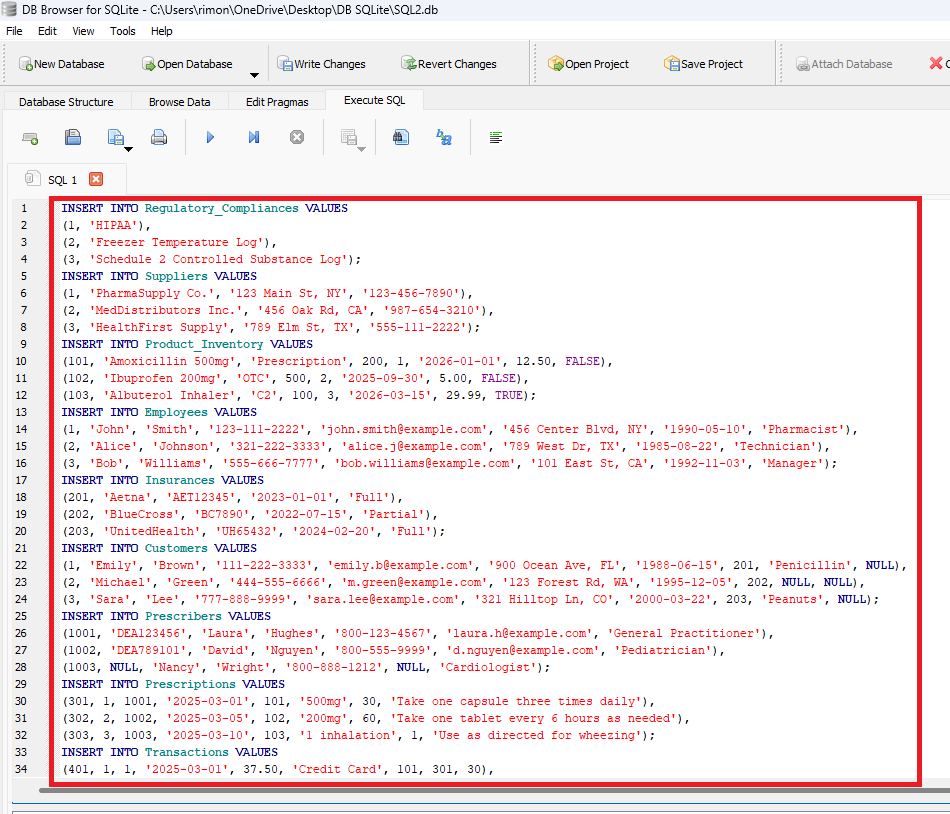
**8.1) Creation of Tables**



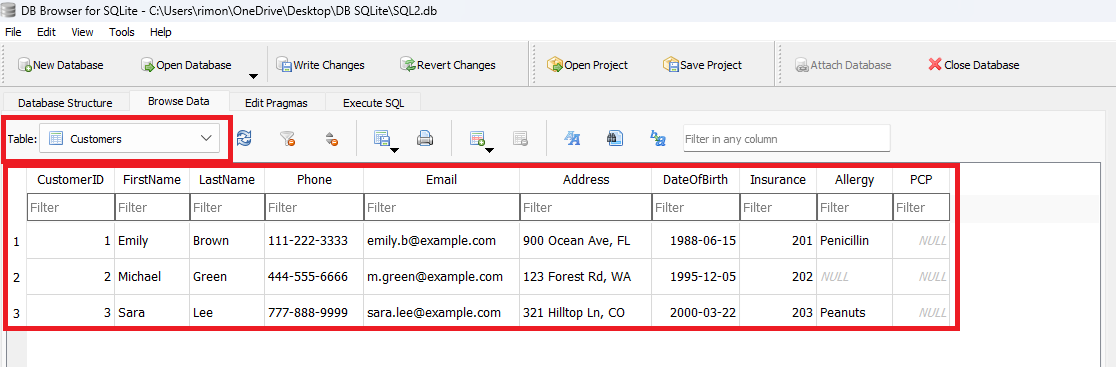
**8.2) Insertion of Sample Data into tables:**

For this DBMS project, the group added 3 rows of data to each and every relation table. All tables have a Cardinality of 3.

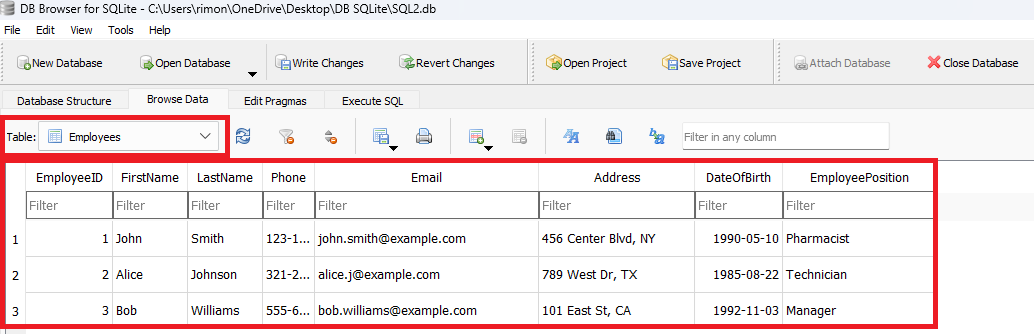
See the commands below:



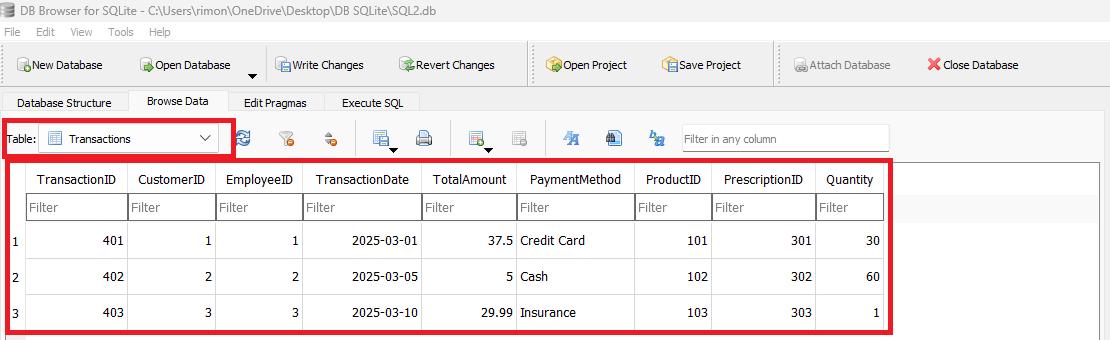
Customers table after inserting three rows of data (Cardinality of 3):



Employees table (Cardinality of 3):



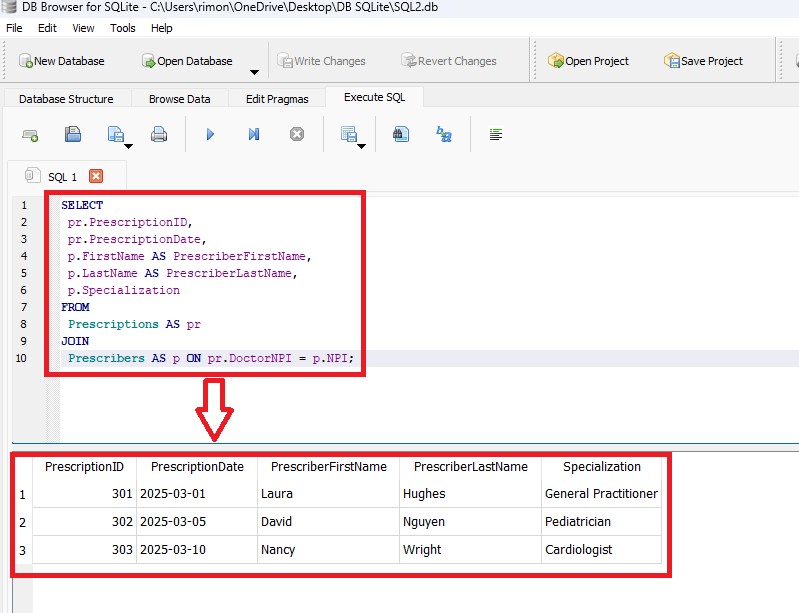
Transactions table (Cardinality of 3):



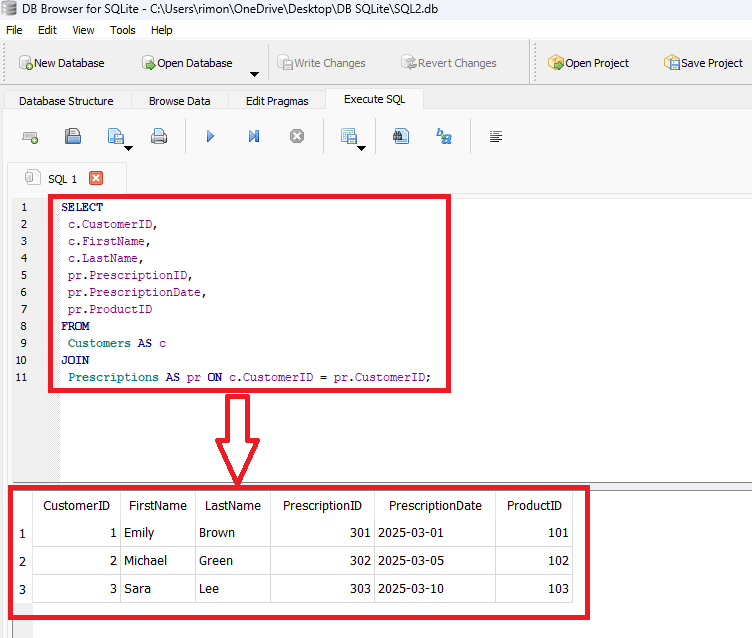
**8.3) Joint Queries**

Below are three JOIN queries listed from (i) to (iii). For each JOIN execution, the SQL code and results are placed below:

1. Prescriptions and prescribers – Get prescription details and the prescriber who issued them



ii. Customers and Prescriptions – Retrieve customer details and their prescriptions



iii. Retrieve Schedule2(C2) Log – Keeping track of C2 transactions with Date, customer information, and quantity

