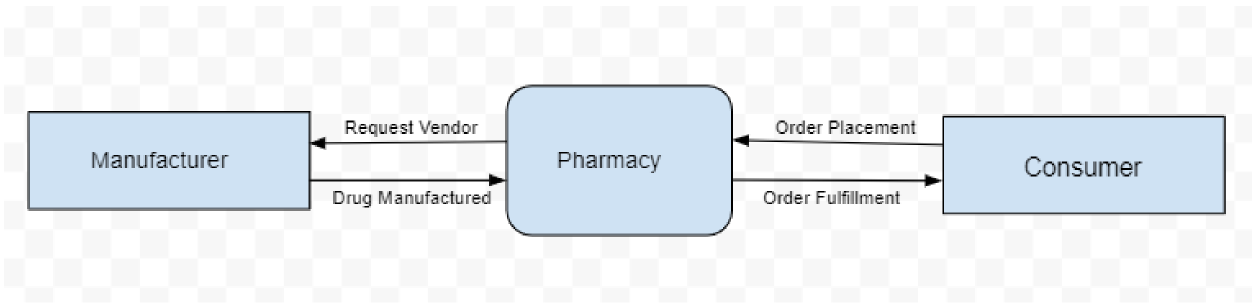
Title: Pharmacy Management

Introduction

Our project is based on the database of a Pharmacy with multiple locations. Think of Walgreens or CVS Pharmacy with multiple locations all over the country.

Our pharmacy business operates on a business to consumer model. The business will request prescription medication as inventory from various drug manufacturers and vendors. In turn, this inventory is sold to consumers possessing the appropriate prescriptions made out by their physicians.

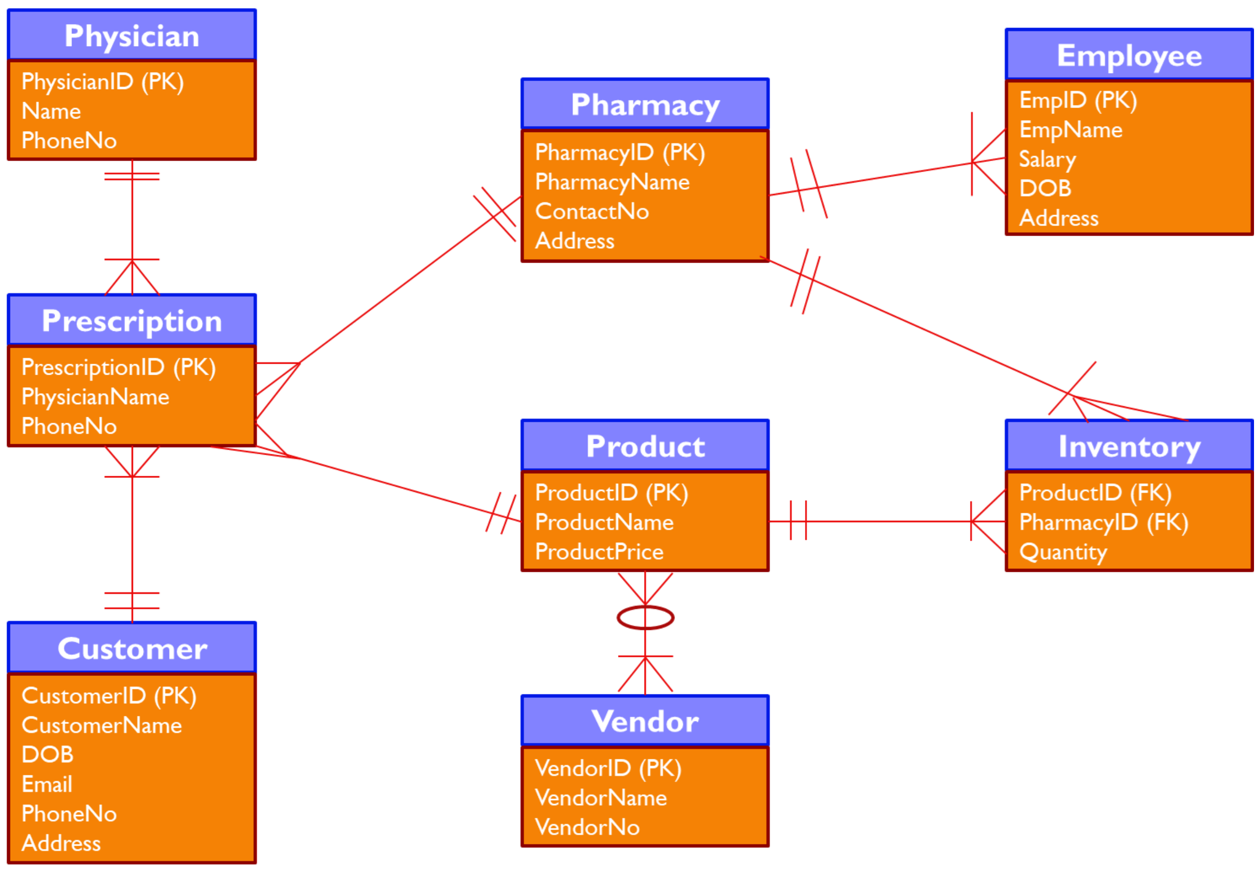


As our business deals with medications that can harm patients if not consumed properly, our pharmacy must keep records of all prescription orders, to identify all key stakeholders within the transaction. These records will be kept in our main pharmacy database. The main purpose of this database is to keep accurate records of inventory and how it relates to their customers. Specifically, the database manages inventory records, prescription records, tracks the sales and provision of medications, and keeps a database of customers, employees, and associated physicians.

The specific entities that we will track in our database are as follows:

- Physician
- Pharmacy
- Patient
- Vendor
- Prescription
- Product
- Employee
- Inventory

ER Diagram



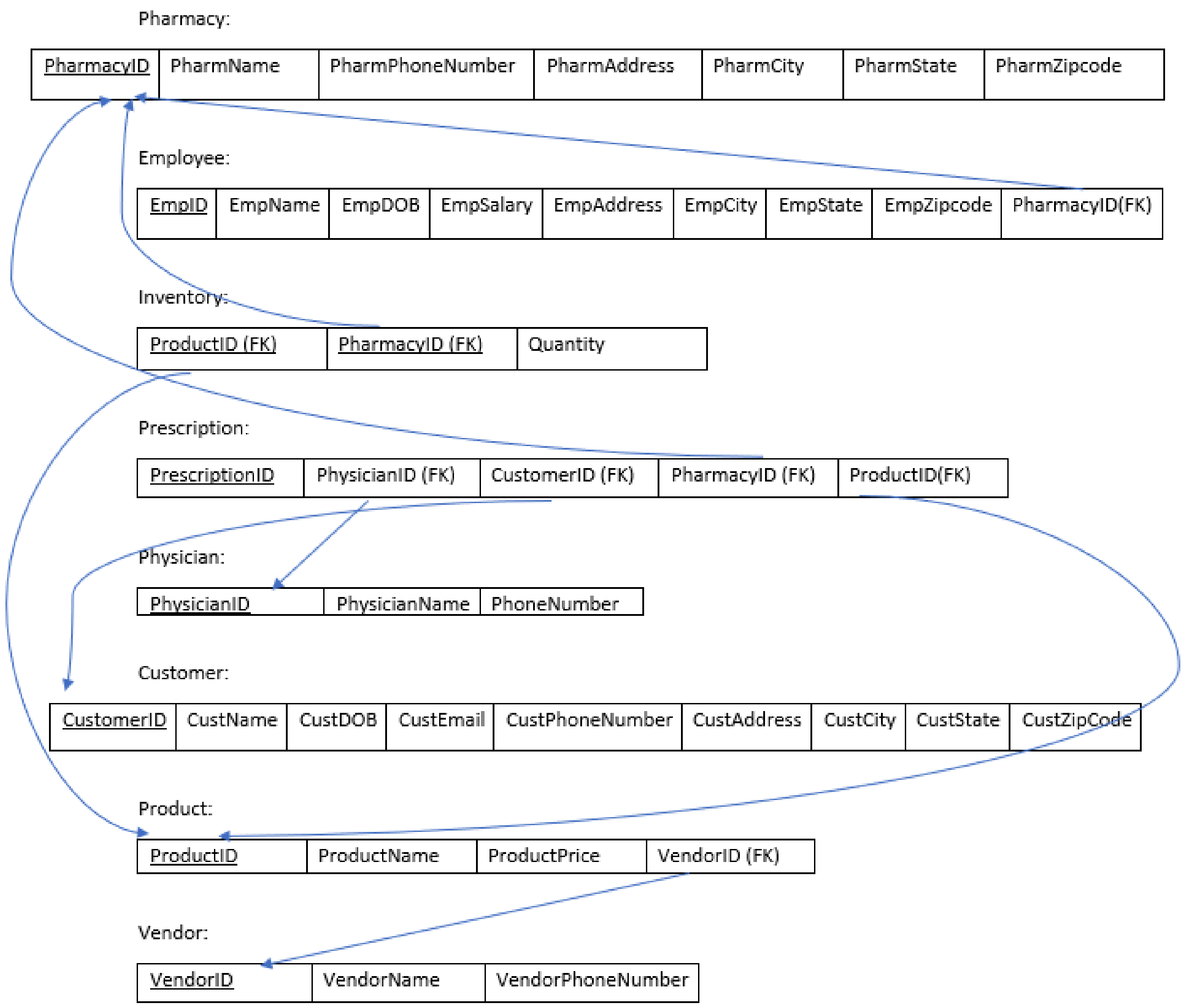
A customer must have at least one prescription, and can have many prescriptions. A prescription must have one and only one customer. A physician can prescribe at least one prescription and can prescribe many and A prescription must be prescribed by one and only physician. A pharmacy can have at least one prescription and can have many and A prescription must have one and only pharmacy. A product can have at least one prescription and can have many and a prescription must have one and only product.

A pharmacy can have at least one employee and can have many and An employee can work at one and only pharmacy.A pharmacy can have at least one inventory and can have many and an inventory can have one and only pharmacy.

A product can have at least one inventory and can have many and an inventory must have one and only product. A vendor can have at least one product and can have many and a product must have one and only vendor.
There is a relationship similar to ternary which we learned in class between physician, customer, product, and pharmacy. This relationship is pivotal in the ER diagram due to the idea that the prescription order ties all of these entities together. The physician can send in one to many

orders, and the pharmacy also has to have one to many orders. There can be up to many products that go into a prescription order, and customers can pick up a minimum of one, up to an infinite number of prescriptions. The reason why they all have a one to many relationship is due to the idea that the central entity relies on at least one of each entity to create a prescription order.

Logical Schema



The logical schema has arrows to show referential integrity constraints. The arrows start at the dependent table and point to the parent table. For example, The pharmacy table has three referential arrows from the employee, inventory, and prescription table all pointing to the pharmacy table’s primary key, PharmacyID.

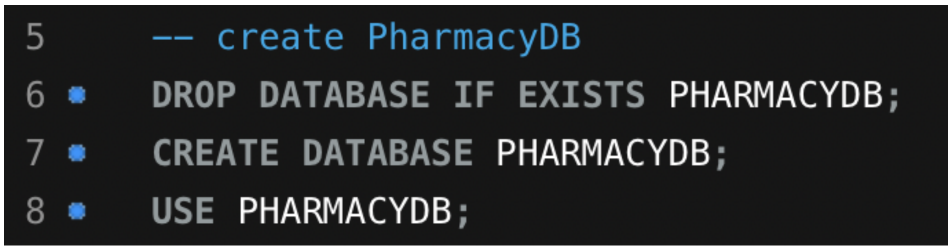
Each table has at least one primary key. Within the Inventory table, the ProductID and the PharmacyID both serve as primary and foreign keys. Essentially, the transformed logical schema meets the 1^{st} normal form because it does not have multi-varied attributes and all the tables have at least one primary key. It also meets the 2^{nd} normal form because all non-key attributes are fully function dependent on the entire primary key. The 3rd normal form is satisfied because there aren’t any transitive dependencies.

The transition from first normal form to second normal form would allow for there to be functional dependencies where non-key attributes are dependent on the full primary key. The 3rd normal form ensures that there are no transitive dependencies within the table. For Instance, the first normal form would have both pharmacyID and EmployeeID with other attributes that relied on each. This would not satisfy the 3rd normal form because there are transitive dependencies within each primary key. Therefore, we used an interactive process to break down partial and transitive dependencies to 2nd and 3rd normal form, respectively.

Database Creation:

The pharmacy's database was created in MySQL, titled PHARMACYDB. The following sets of code were used to construct the database, and subsequent related entity tables.

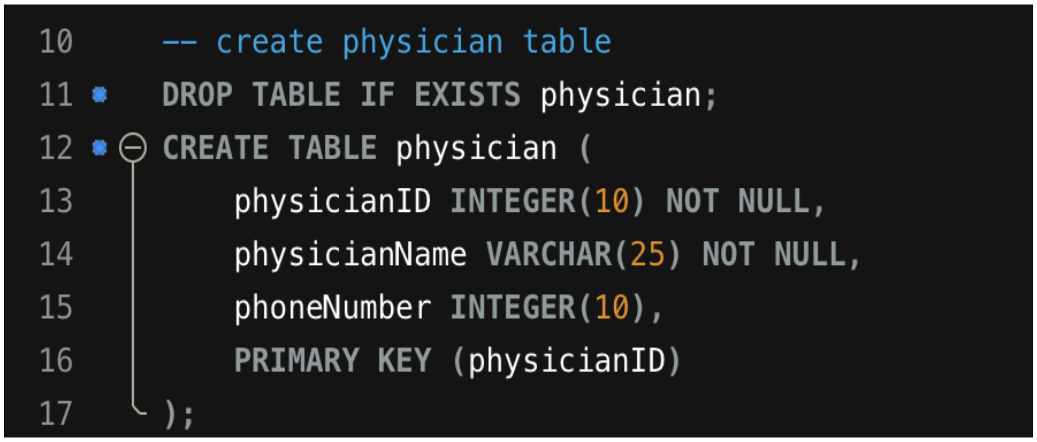
Create Pharmacy Database:



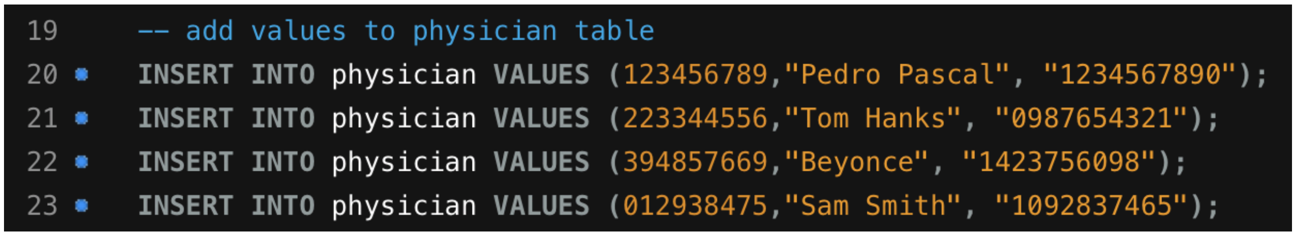
```sql
-- create PharmacyDB
DROP DATABASE IF EXISTS PHARMACYDB;
CREATE DATABASE PHARMACYDB;
USE PHARMACYDB;
```

Create Physician entity & add dummy values:

Along with the SQL code which creates the entities inside the Pharmacy Database, dummy values were entered into the entities in order to depict a more realistic database. These dummy values are used later when performing queries on the database.

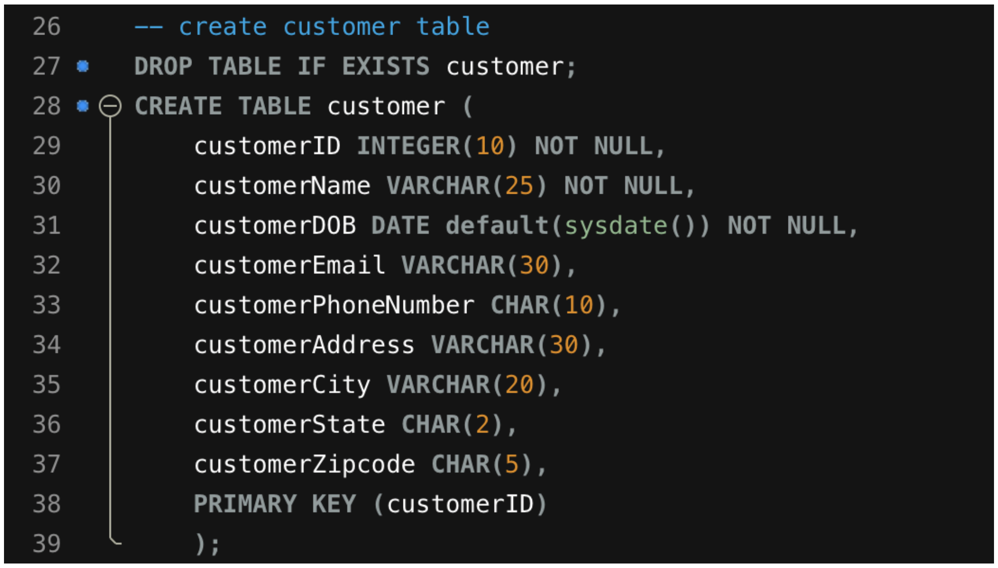


```sql
-- create physician table
DROP TABLE IF EXISTS physician;
CREATE TABLE physician (
    physicianID INTEGER(10) NOT NULL,
    physicianName VARCHAR(25) NOT NULL,
    phoneNumber INTEGER(10),
    PRIMARY KEY (physicianID)
);
```

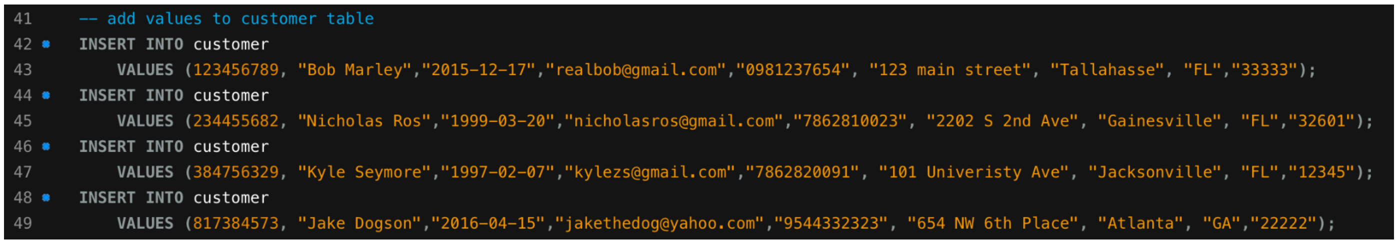


```sql
-- add values to physician table
INSERT INTO physician VALUES (123456789,"Pedro Pascal", "1234567890");
INSERT INTO physician VALUES (223344556,"Tom Hanks", "0987654321");
INSERT INTO physician VALUES (394857669,"Beyonce", "1423756098");
INSERT INTO physician VALUES (012938475,"Sam Smith", "1092837465");
```

Create Customer entity & add dummy values:

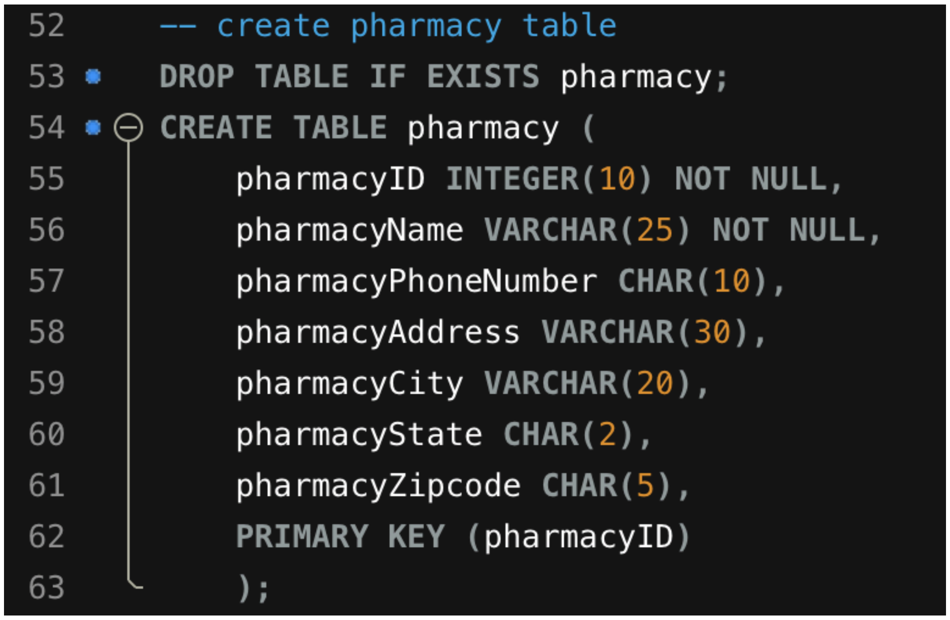


```
-- create customer table
DROP TABLE IF EXISTS customer;
CREATE TABLE customer (
        customerID INTEGER(10) NOT NULL,
        customerName VARCHAR(25) NOT NULL,
        customerDOB DATE default(sysdate()) NOT NULL,
        customerEmail VARCHAR(30),
        customerPhoneNumber CHAR(10),
        customerAddress VARCHAR(30),
        customerCity VARCHAR(20),
        customerState CHAR(2),
        customerZipcode CHAR(5),
        PRIMARY KEY (customerID)
        );
```

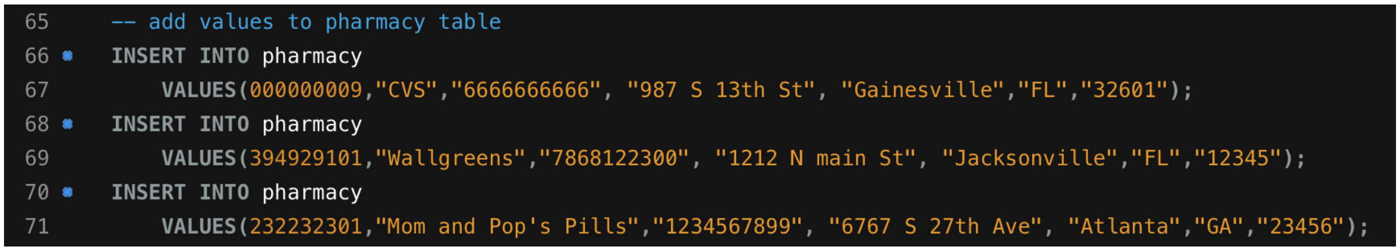


```
-- add values to customer table
INSERT INTO customer
    VALUES (123456789, "Bob Marley","2015-12-17","realbob@gmail.com","0981237654", "123 main street", "Tallahasse", "FL","33333");
INSERT INTO customer
    VALUES (234455682, "Nicholas Ros","1999-03-20","nicholasros@gmail.com","7862810023", "2202 S 2nd Ave", "Gainesville", "FL","32601");
INSERT INTO customer
    VALUES (384756329, "Kyle Seymore","1997-02-07","kylezs@gmail.com","7862820091", "101 Univeristy Ave", "Jacksonville", "FL","12345");
INSERT INTO customer
    VALUES (817384573, "Jake Dogson","2016-04-15","jakethedog@yahoo.com","9544332323", "654 NW 6th Place", "Atlanta", "GA","22222");
```

Create Pharmacy entity & add dummy values:

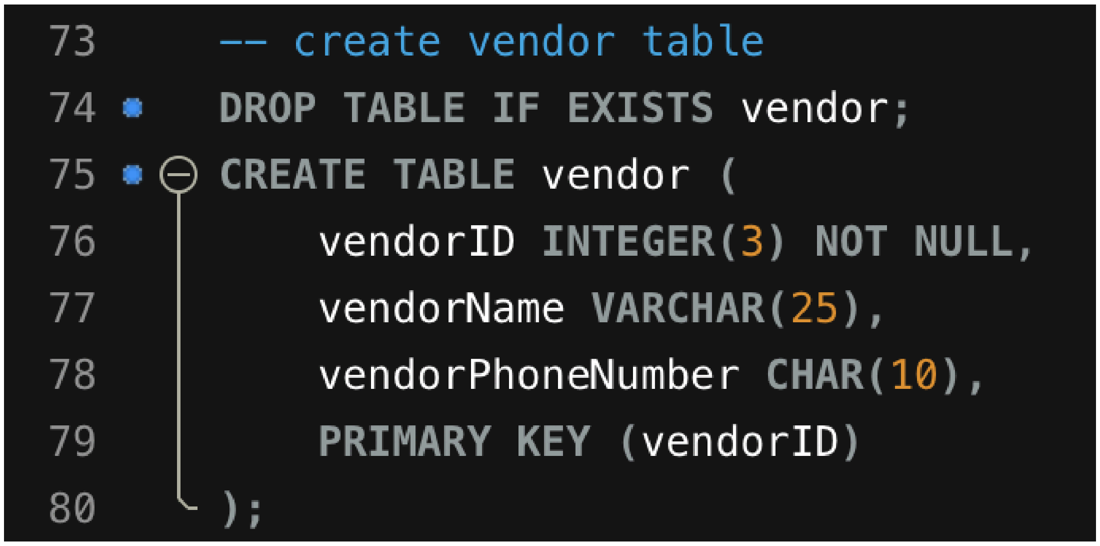


```
-- create pharmacy table
DROP TABLE IF EXISTS pharmacy;
CREATE TABLE pharmacy (
        pharmacyID INTEGER(10) NOT NULL,
        pharmacyName VARCHAR(25) NOT NULL,
        pharmacyPhoneNumber CHAR(10),
        pharmacyAddress VARCHAR(30),
        pharmacyCity VARCHAR(20),
        pharmacyState CHAR(2),
        pharmacyZipcode CHAR(5),
        PRIMARY KEY (pharmacyID)
        );
```

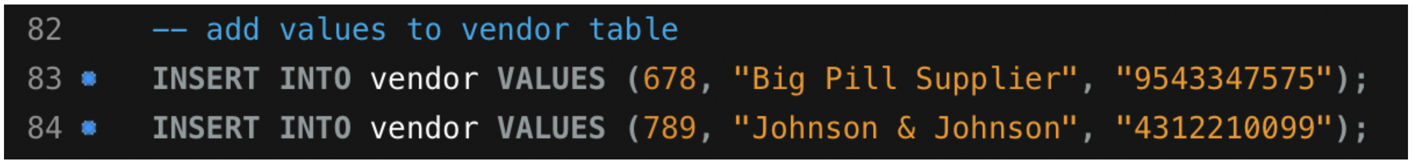


```sql
-- add values to pharmacy table
INSERT INTO pharmacy
    VALUES(000000009,"CVS","6666666666", "987 S 13th St", "Gainesville","FL","32601");
INSERT INTO pharmacy
    VALUES(394929101,"Wallgreens","7868122300", "1212 N main St", "Jacksonville","FL","12345");
INSERT INTO pharmacy
    VALUES(232232301,"Mom and Pop's Pills","1234567899", "6767 S 27th Ave", "Atlanta","GA","23456");
```

Create Vendor entity & add dummy values:

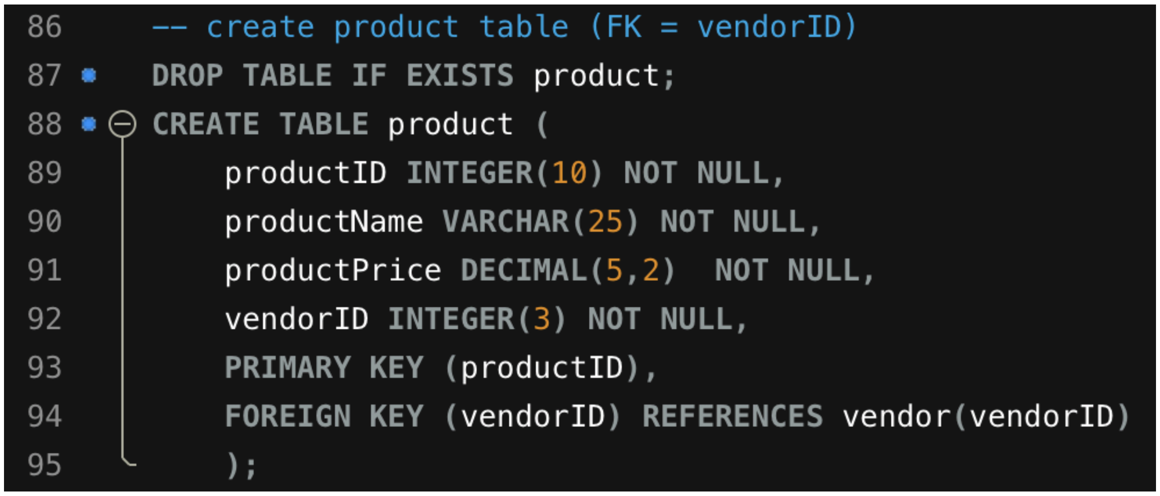


```sql
-- create vendor table
DROP TABLE IF EXISTS vendor;
CREATE TABLE vendor (
        vendorID INTEGER(3) NOT NULL,
        vendorName VARCHAR(25),
        vendorPhoneNumber CHAR(10),
        PRIMARY KEY (vendorID)
);
```

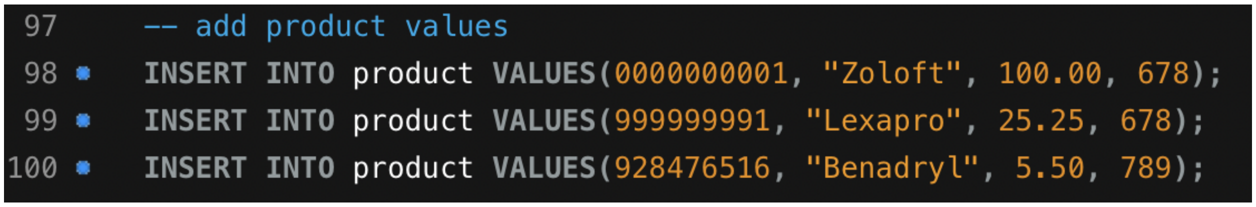


```sql
-- add values to vendor table
INSERT INTO vendor VALUES (678, "Big Pill Supplier", "9543347575");
INSERT INTO vendor VALUES (789, "Johnson & Johnson", "4312210099");
```

Create Product entity & add dummy values:

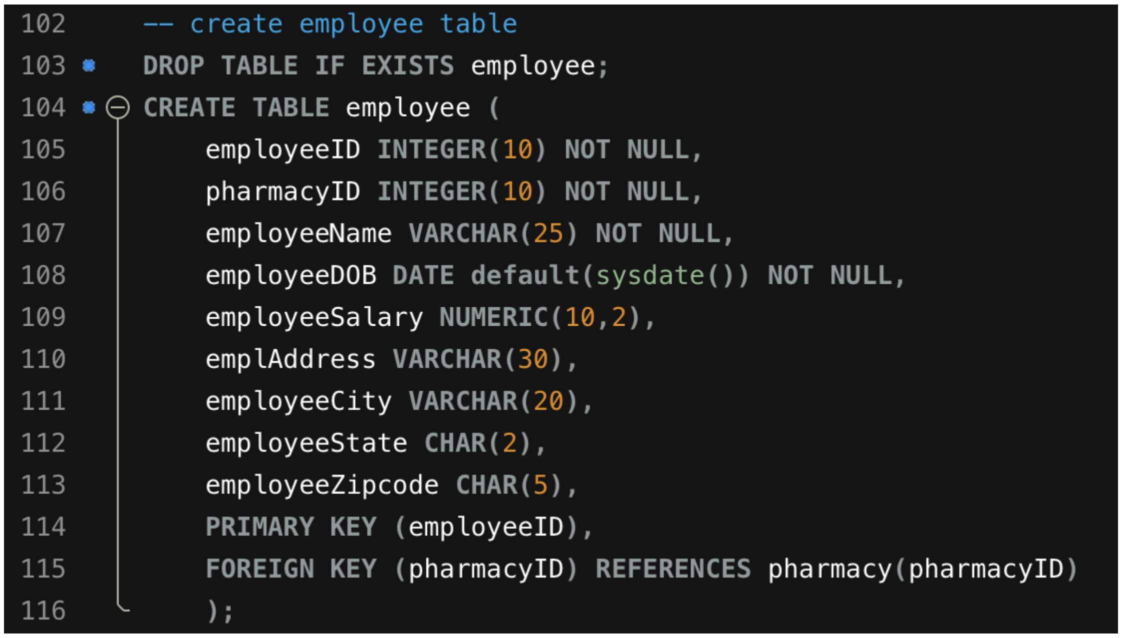


```sql
-- create product table (FK = vendorID)
DROP TABLE IF EXISTS product;
CREATE TABLE product (
    productID INTEGER(10) NOT NULL,
    productName VARCHAR(25) NOT NULL,
    productPrice DECIMAL(5,2)  NOT NULL,
    vendorID INTEGER(3) NOT NULL,
    PRIMARY KEY (productID),
    FOREIGN KEY (vendorID) REFERENCES vendor(vendorID)
    );
```

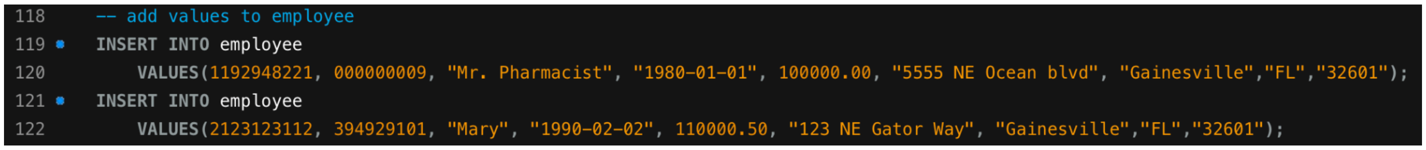


```sql
-- add product values
INSERT INTO product VALUES(0000000001, "Zoloft", 100.00, 678);
INSERT INTO product VALUES(999999991, "Lexapro", 25.25, 678);
INSERT INTO product VALUES(928476516, "Benadryl", 5.50, 789);
```

Create Employee entity & add dummy values:



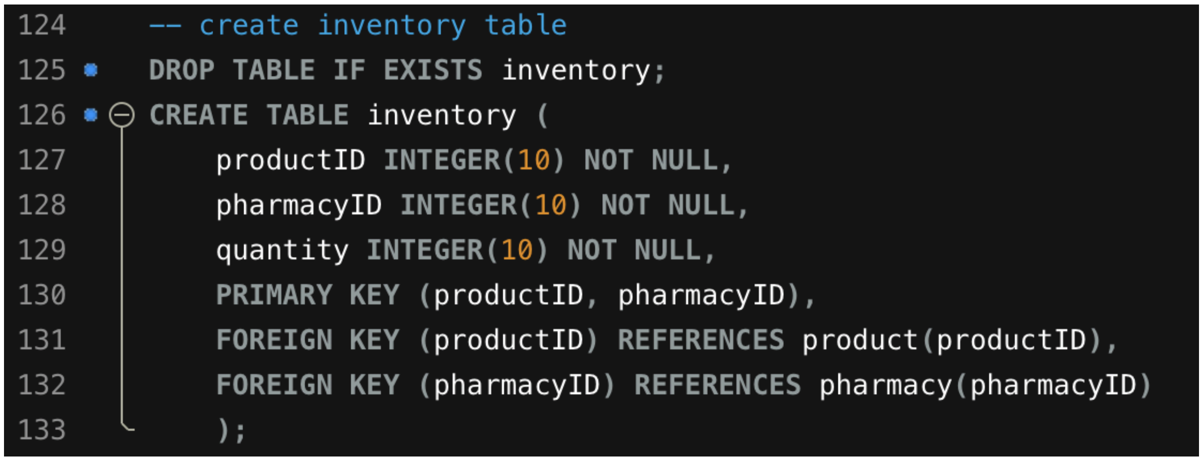
```sql
-- create employee table
DROP TABLE IF EXISTS employee;
CREATE TABLE employee (
    employeeID INTEGER(10) NOT NULL,
    pharmacyID INTEGER(10) NOT NULL,
    employeeName VARCHAR(25) NOT NULL,
    employeeDOB DATE default(sysdate()) NOT NULL,
    employeeSalary NUMERIC(10,2),
    emplAddress VARCHAR(30),
    employeeCity VARCHAR(20),
    employeeState CHAR(2),
    employeeZipcode CHAR(5),
    PRIMARY KEY (employeeID),
    FOREIGN KEY (pharmacyID) REFERENCES pharmacy(pharmacyID)
    );
```



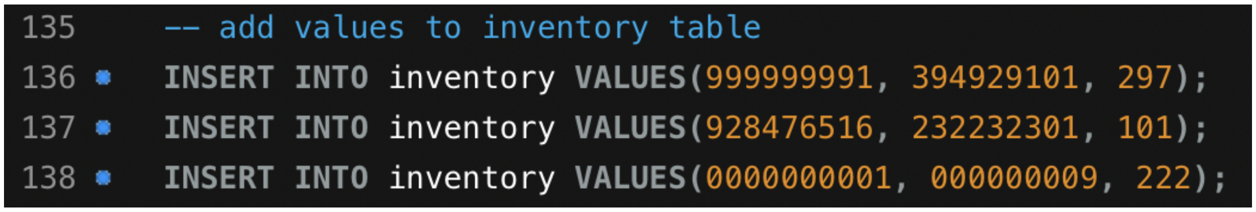
```sql
-- add values to employee
INSERT INTO employee
    VALUES(1192948221, 000000009, "Mr. Pharmacist", "1980-01-01", 100000.00, "5555 NE Ocean blvd", "Gainesville","FL","32601");
INSERT INTO employee
    VALUES(2123123112, 394929101, "Mary", "1990-02-02", 110000.50, "123 NE Gator Way", "Gainesville","FL","32601");
```

Create Inventory entity & add dummy values:

The Inventory table is an associative entity which records all products held in inventory per pharmacy location, and records the on-hand inventory. The primary key for this entity is a composite key made up of foreign keys **productID** and **pharmacyID**.



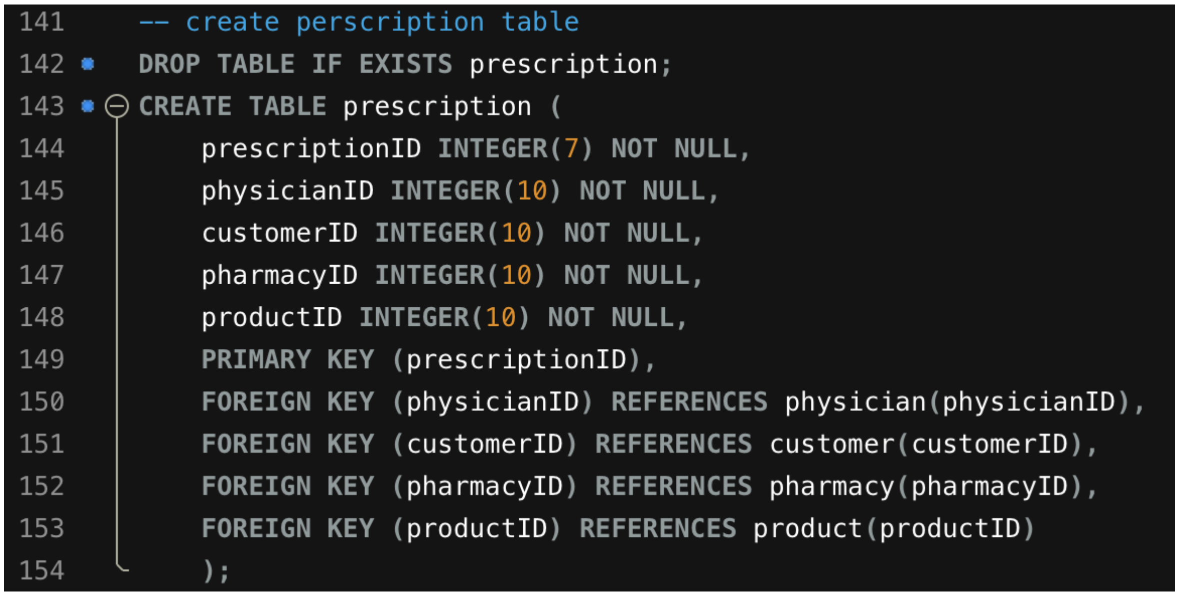
```sql
-- create inventory table
DROP TABLE IF EXISTS inventory;
CREATE TABLE inventory (
    productID INTEGER(10) NOT NULL,
    pharmacyID INTEGER(10) NOT NULL,
    quantity INTEGER(10) NOT NULL,
    PRIMARY KEY (productID, pharmacyID),
    FOREIGN KEY (productID) REFERENCES product(productID),
    FOREIGN KEY (pharmacyID) REFERENCES pharmacy(pharmacyID)
    );
```



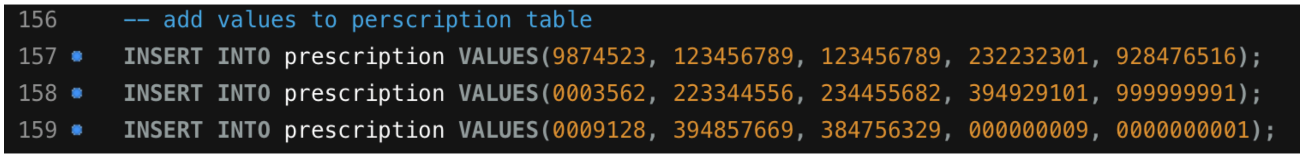
```sql
-- add values to inventory table
INSERT INTO inventory VALUES(999999991, 394929101, 297);
INSERT INTO inventory VALUES(928476516, 232232301, 101);
INSERT INTO inventory VALUES(0000000001, 000000009, 222);
```

Create Prescription entity & add dummy values:

The Prescription table is an associative entity which records all prescription orders completed in our pharmacies. A new primary key was created for this entity, **prescriptionID**, rather than using all 4 foreign keys as a composite primary key.



```sql
-- create perscription table
DROP TABLE IF EXISTS prescription;
CREATE TABLE prescription (
    prescriptionID INTEGER(7) NOT NULL,
    physicianID INTEGER(10) NOT NULL,
    customerID INTEGER(10) NOT NULL,
    pharmacyID INTEGER(10) NOT NULL,
    productID INTEGER(10) NOT NULL,
    PRIMARY KEY (prescriptionID),
    FOREIGN KEY (physicianID) REFERENCES physician(physicianID),
    FOREIGN KEY (customerID) REFERENCES customer(customerID),
    FOREIGN KEY (pharmacyID) REFERENCES pharmacy(pharmacyID),
    FOREIGN KEY (productID) REFERENCES product(productID)
    );
```



```sql
-- add values to perscription table
INSERT INTO prescription VALUES(9874523, 123456789, 123456789, 232232301, 928476516);
INSERT INTO prescription VALUES(0003562, 223344556, 234455682, 394929101, 999999991);
INSERT INTO prescription VALUES(0009128, 394857669, 384756329, 000000009, 0000000001);
```

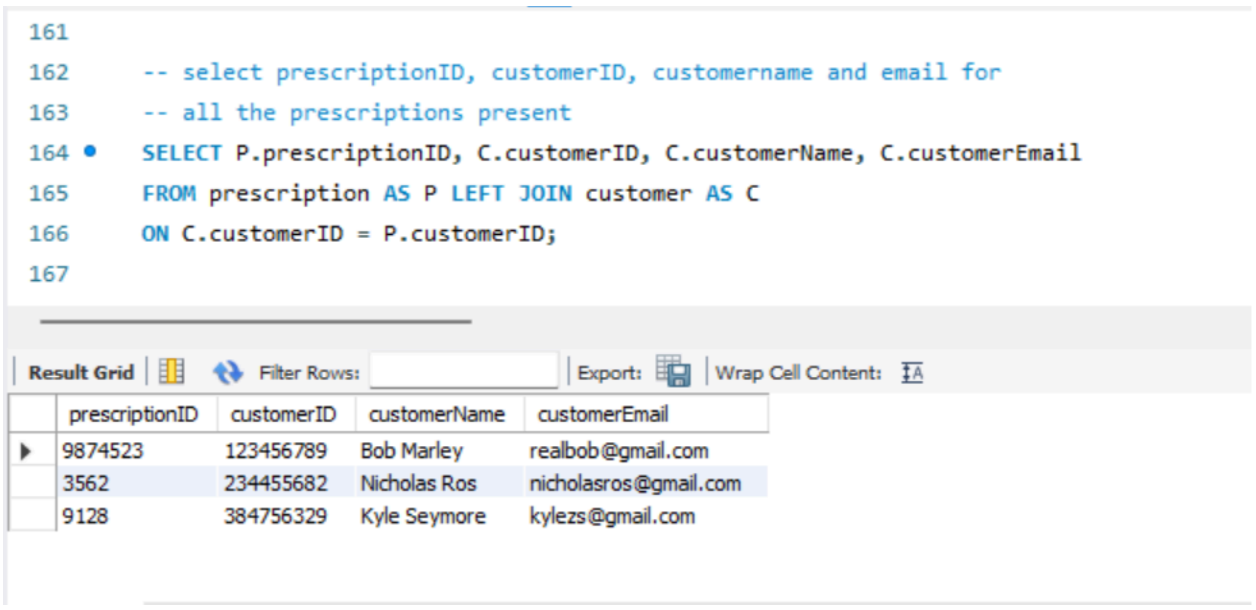
Database Management:

Query the database

Real-world scenario and corresponding SQL code:

1. The database administrator may need to view customer information and their associated prescriptions. Then we should select prescription ID, customer ID, customer name, and email for all the prescriptions present.

SELECT P.prescriptionID, C.customerID, C.customerName, C.customerEmail
FROM prescription **AS** P **LEFT JOIN** customer AS C
ON C.customerID = P.customerID;

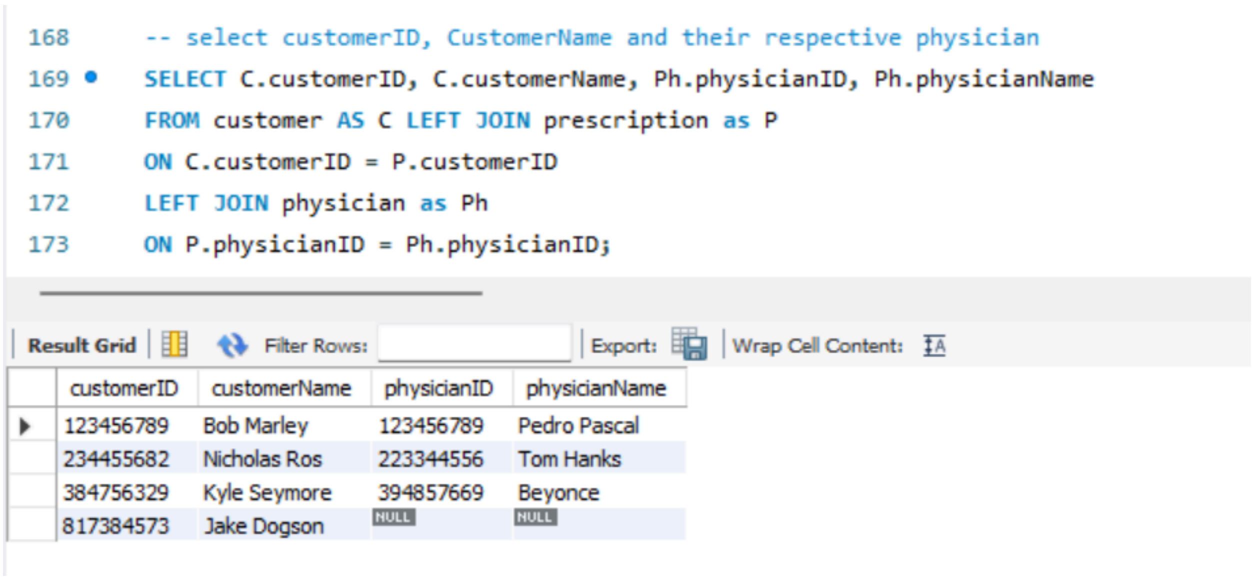


```

-- select prescriptionID, customerID, customername and email for
-- all the prescriptions present
SELECT P.prescriptionID, C.customerID, C.customerName, C.customerEmail
FROM prescription AS P LEFT JOIN customer AS C
ON C.customerID = P.customerID;

```

prescriptionID	customerID	customerName	customerEmail
9874523	123456789	Bob Marley	realbob@gmail.com
3562	234455682	Nicholas Ros	nicholasros@gmail.com
9128	384756329	Kyle Seymore	kylezs@gmail.com

2. When there is a need to see customers and the physicians who made a prescription for the customer. Then we should select the customer ID, customer name, and their respective physician details like physician ID and physician name.

SELECT C.customerID, C.customerName, Ph.physicianID, Ph.physicianName
FROM customer **AS** C **LEFT JOIN** prescription as P
ON C.customerID = P.customerID
LEFT JOIN physician as Ph
ON P.physicianID = Ph.physicianID;

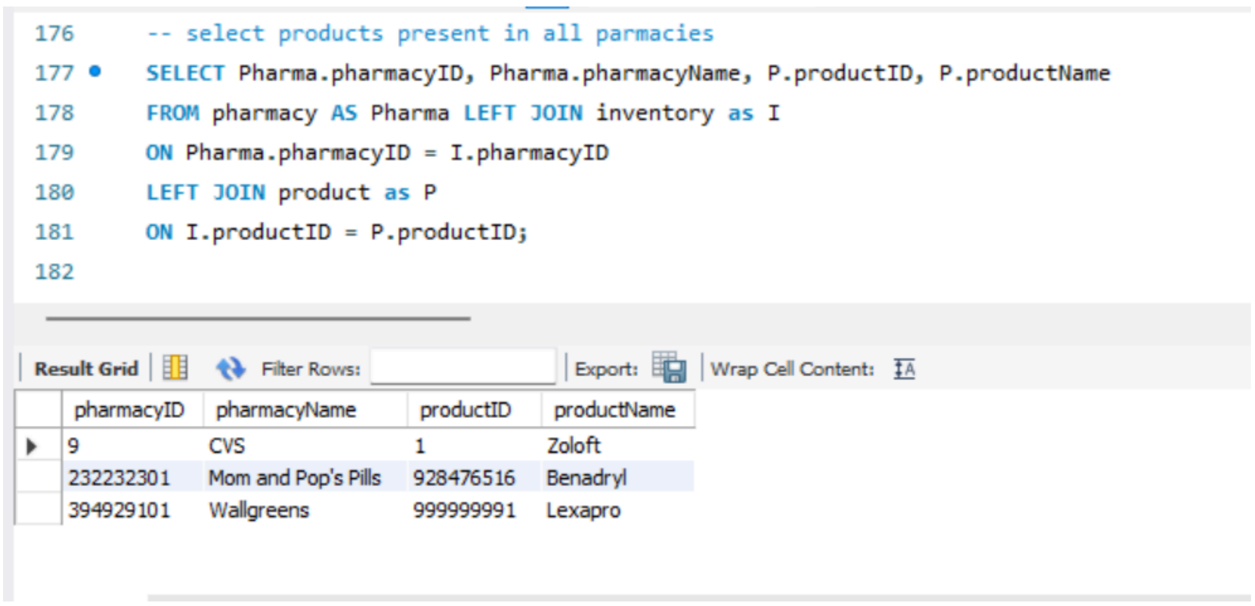


```
-- select customerID, CustomerName and their respective physician
SELECT C.customerID, C.customerName, Ph.physicianID, Ph.physicianName
FROM customer AS C LEFT JOIN prescription as P
ON C.customerID = P.customerID
LEFT JOIN physician as Ph
ON P.physicianID = Ph.physicianID;
```

customerID	customerName	physicianID	physicianName
123456789	Bob Marley	123456789	Pedro Pascal
234455682	Nicholas Ros	223344556	Tom Hanks
384756329	Kyle Seymore	394857669	Beyonce
817384573	Jake Dogson	NULL	NULL

3. A database administrator would like to see which products are being sold in each different store. We can select Pharmacy ID, Pharmacy Name, ProductID and the Product Name using the query below.

SELECT Pharma.pharmacyID, Pharma.pharmacyName, P.productID, P.productName
FROM pharmacy AS Pharma **LEFT JOIN** inventory as I
ON Pharma.pharmacyID = I.pharmacyID
LEFT JOIN product as P
ON I.productID = P.productID;

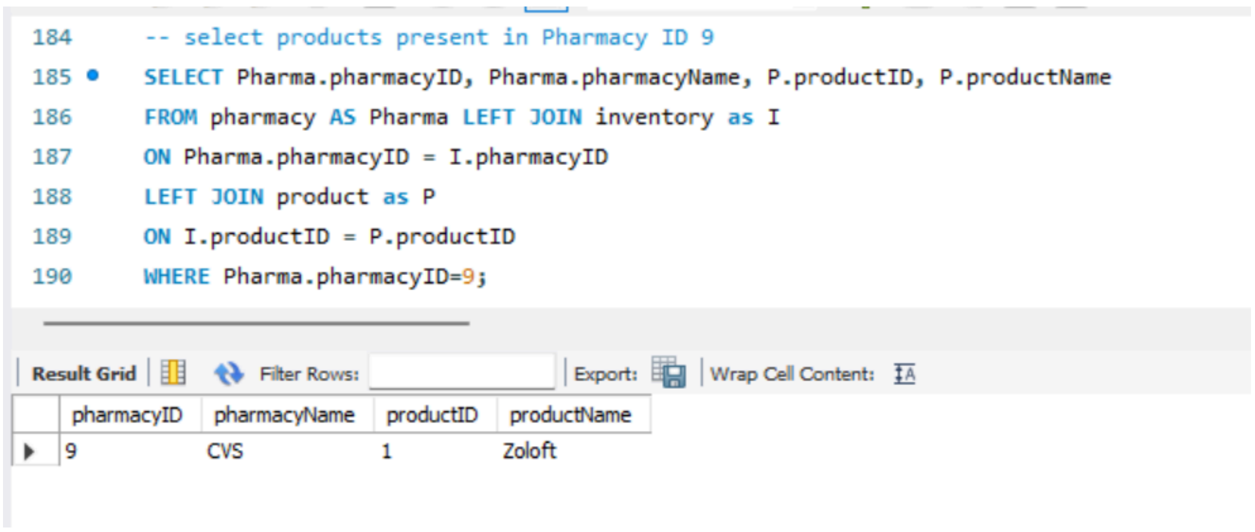


```
-- select products present in all parmacies
SELECT Pharma.pharmacyID, Pharma.pharmacyName, P.productID, P.productName
FROM pharmacy AS Pharma LEFT JOIN inventory as I
ON Pharma.pharmacyID = I.pharmacyID
LEFT JOIN product as P
ON I.productID = P.productID;
```

pharmacyID	pharmacyName	productID	productName
9	CVS	1	Zoloft
232232301	Mom and Pop's Pills	928476516	Benadryl
394929101	Wallgreens	999999991	Lexapro

4. A database administrator may need to see what products are being used in any particular pharmacy. In this example we want to look at then we can select Pharmacy ID, Pharmacy Name, Product ID, and Product Name, then use a selection criteria of Pharmacy ID=9 in the WHERE clause.

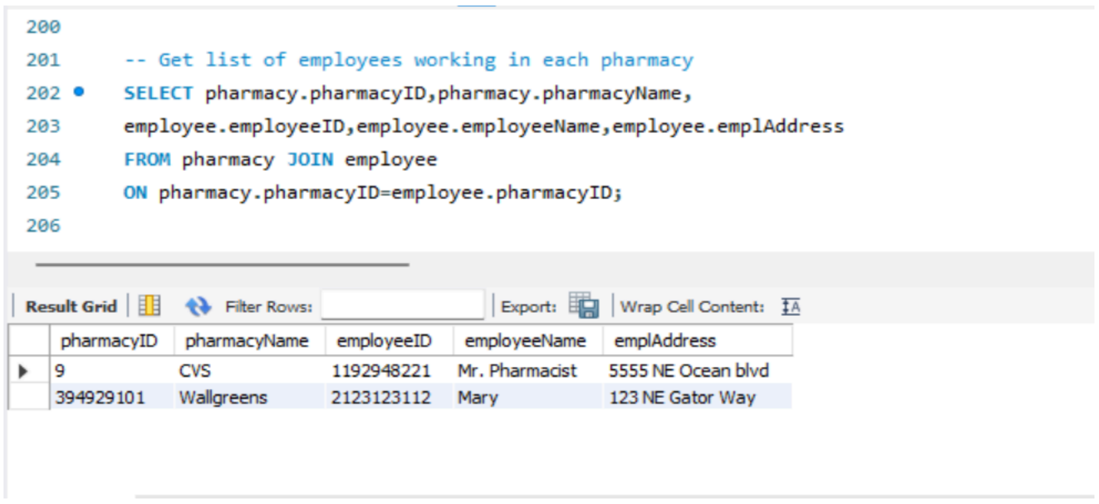
SELECT Pharma.pharmacyID, Pharma.pharmacyName, P.productID, P.productName
FROM pharmacy **AS** Pharma **LEFT JOIN** inventory **AS** I
ON Pharma.pharmacyID = I.pharmacyID
LEFT JOIN product as P
ON I.productID = P.productID
WHERE Pharma.pharmacyID = 9;


```
-- select products present in Pharmacy ID 9
SELECT Pharma.pharmacyID, Pharma.pharmacyName, P.productID, P.productName
FROM pharmacy AS Pharma LEFT JOIN inventory as I
ON Pharma.pharmacyID = I.pharmacyID
LEFT JOIN product as P
ON I.productID = P.productID
WHERE Pharma.pharmacyID=9;
```

pharmacyID	pharmacyName	productID	productName
9	CVS	1	Zoloft

5. If the pharmacy DB administrator needs to see all employees staffed in each of the pharmacy locations then we can join the pharmacy and employee table.

SELECT pharmacy.pharmacyID, pharmacy.pharmacyName, employee.employeeID, employee.employeeName, employee.emplAddress
FROM pharmacy **JOIN** employee
ON pharmacy.pharmacyID = employee.pharmacyID;


```
-- Get list of employees working in each pharmacy
SELECT pharmacy.pharmacyID,pharmacy.pharmacyName,
employee.employeeID,employee.employeeName,employee.emplAddress
FROM pharmacy JOIN employee
ON pharmacy.pharmacyID=employee.pharmacyID;
```

pharmacyID	pharmacyName	employeeID	employeeName	emplAddress
9	CVS	1192948221	Mr. Pharmacist	5555 NE Ocean blvd
394929101	Wallgreens	2123123112	Mary	123 NE Gator Way

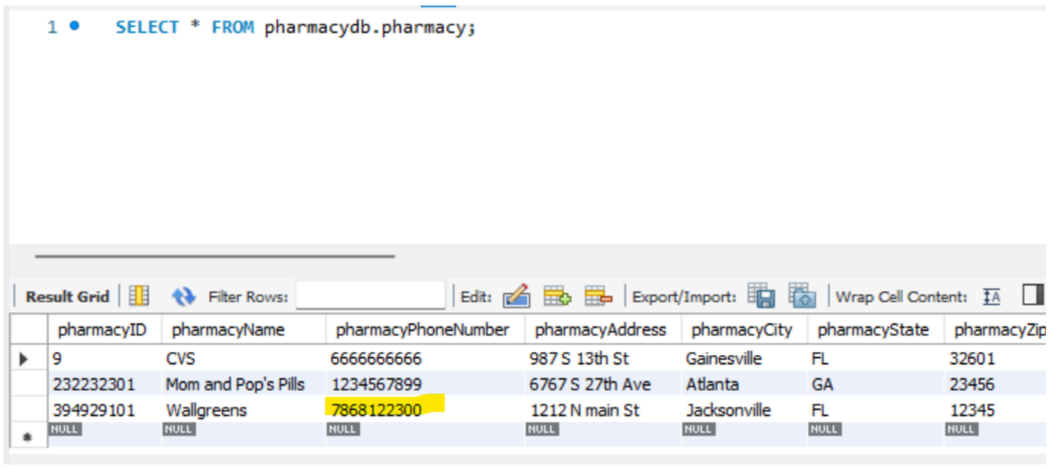
Update the Database

Just like how important it is to retrieve the data, sometimes it is also important to update and delete the data whenever you want to have the latest and up-to-date information.

6. In a situation in which one of the pharmacy locations changed their phone number, we can use an update query to update a particular pharmacy field, specifically the Pharmacy Phone Number.

UPDATE pharmacy
SET pharmacyPhoneNumber = '4567896659'
WHERE pharmacyID = '394929101';

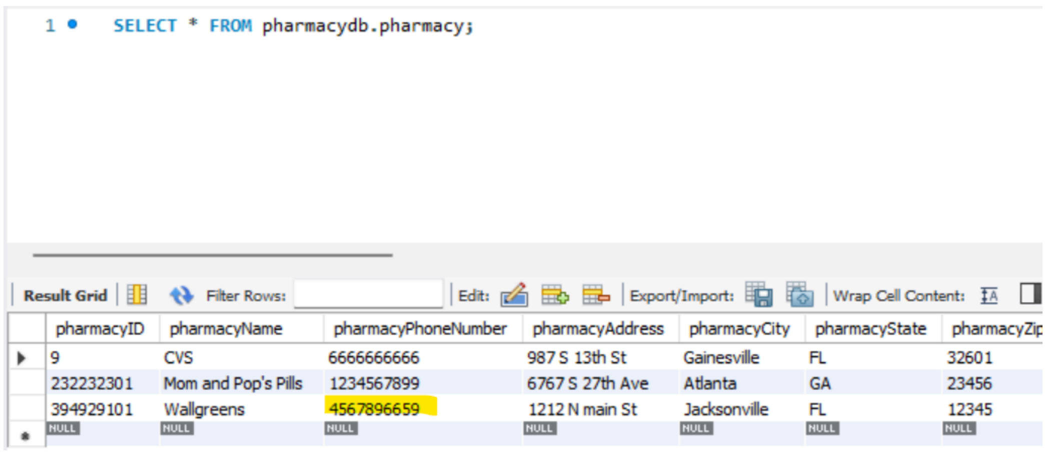
BEFORE:



```
SELECT * FROM pharmacydb.pharmacy;
```

pharmacyID	pharmacyName	pharmacyPhoneNumber	pharmacyAddress	pharmacyCity	pharmacyState	pharmacyZip
9	CVS	6666666666	987 S 13th St	Gainesville	FL	32601
232232301	Mom and Pop's Pills	1234567899	6767 S 27th Ave	Atlanta	GA	23456
394929101	Wallgreens	7868122300	1212 N main St	Jacksonville	FL	12345
NULL	NULL	NULL	NULL	NULL	NULL	NULL

AFTER:



```
SELECT * FROM pharmacydb.pharmacy;
```

pharmacyID	pharmacyName	pharmacyPhoneNumber	pharmacyAddress	pharmacyCity	pharmacyState	pharmacyZip
9	CVS	6666666666	987 S 13th St	Gainesville	FL	32601
232232301	Mom and Pop's Pills	1234567899	6767 S 27th Ave	Atlanta	GA	23456
394929101	Wallgreens	4567896659	1212 N main St	Jacksonville	FL	12345
NULL	NULL	NULL	NULL	NULL	NULL	NULL