Name: Marcus Andrew Acord-Serventi

**Table of Contents**

[Project Direction Overview 2](#_Toc101790907)

[Use Cases and Fields 2](#_Toc101790908)

[Structural Database Rules 6](#_Toc101790909)

[Conceptual Entity-Relationship Diagram 7](#_Toc101790910)

[Full DBMS Physical ERD 8](#_Toc101790911)

[MS Server SQL Reset Script 9](#_Toc101790912)

[MS Server SQL Initialization Script 9](#_Toc101790913)

[MS Server SQL Procedures Script 12](#_Toc101790914)

[MS Server SQL Data Fill Script 22](#_Toc101790915)

[MS Server SQL Queries 33](#_Toc101790916)

[MS Server SQL Indexes 37](#_Toc101790917)

[History Table Demonstration 39](#_Toc101790918)

[Data Visualizations 41](#_Toc101790919)

[Summary and Reflection 41](#_Toc101790920)

# Project Direction Overview

For my project I want to revisit a web app that I created in CS 602, server-side programming. We were tasked to develop a simple shopping cart application for a website. I developed Phil’s Pills, a simple online store for pharmacy ordering. To do this, we had to create a front-end design, coupled with back-end access (I used PHP), that ultimately hooked into a database using MyPHP and MySQL. The course taught us a lot of the fundamentals of server-side programming, but almost no database information. I basically had to rely on the template we used in the assignments in order to create an exceedingly simple SQL database. I remember when I got stuck at one point because I needed to JOIN two tables to display the data correctly, I didn’t know how, and had to ask my facilitator (I’m assuming this is probably a very basic function of SQL operations). I want to revisit this project because I’m wondering how I could have done things better and the base project gives me some reference as to how a database should work. After taking the course, I felt like databases were never really explained, but we were expected to know and use them; I felt like it would have been wise to take CS669 BEFORE CS602, so I want to reflect and see how better I would have done in 602 by completing this project.

Some examples of the databases we need to use are for products, for user information, and orders. Obviously we need to display all of our products, their prices, UPC codes, quantity in stock, etc. This is the core of the application. We need a way to store our user information, so user name, password, name, e-mail, shipping address, and a check for admin users. Lastly, once a user places an order, we need to create their order and store it in the database; this could be an order number that increments, what they ordered, the price, the quantity, the data ordered, and the order status. This could get a bit complicated as we need a way to track and update the order as needed.

The programming work, as far as UI and interface is already done (albeit, not the best). However, the database portion is so scant and basic, it really begs to be improved and updated and I’m eager to see what I learn in this course to revisit how I could have done things better on my previous project. At the end of the project, I’d probably want to take what I know, and redevelop my app to meet an improved database schema.

# Use Cases and Fields

Case Number:

1. A user should be able to create a new account from the website landing page.
2. Once an account is created, a user should be able to login to the website.
3. Users should be able to view and update their profile.

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it’s Needed |
| PK: UserName | The user’s login username | So user’s can login to the system using their user name |
| Password | The user’s login Password | So user’s can login to the system using their password |
| E-Mail | The user’s email address | So we can contact the user |
| Phone | The user’s phone number | So we can contact the user |
| FirstName | The user’s first name | Mainly for shipping and billing purposes |
| LastName | The user’s last name | Mainly for shipping and billing purposes |
| Address | The user’s address | Address needed to ship products to the user |

1. A user should be able to view a list of products.
2. A user should be able select n products from the list and buy a n numbers of product.
3. Admins should be able to update the product’s data (quantity, price, etc.).

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it’s Needed |
| PK: UPC | The UPC code of the product (like a barcode) | Unique identifier of the product; may be the NDC code or the barcode number of the product |
| ProductName | Name of the product | To easily identify the product to users |
| ProductDescription | A description of the product | Describes the product; important to extended duration or extended release, etc. |
| Size | The size (think mg for pills) of the product | So we can find pills of certain sizes when needed |
| Price | Price per pill of the product | Needed to charge customers and create order logs |
| StockQuantity | Number of product in stock | Needed to maintain how much stock of the product there is; customers should not be able to order more than is in stock and we need to be able to check stock levels of products |

1. Once a user has bought a product, an order should be created and stored.
2. Admins should be able to update the order information or cancel an order.

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it’s Needed |
| PK: OrderNumber | The order number | Unique identifier of the product; may be the NDC code or the barcode number of the product |
| DateOrdered | When the order was placed | We need this to provide a date in time for the customer and for records; we can also use this to ensure an order is completed timely |
| FK: UserName | The user who ordered the product | Identifies which order belongs to which user. Links orders and users. |

\*Note this is a one-to-many relationship. I failed to implement this in server-side programming because I did not know SQL. An order can contain several different product UPCs and quantities. This data should also, somehow, be joined from the previous table.

1. Additionally, for medications requiring prescriptions, we need 2x more tables: A Prescription table and a prescriber table.
2. Prescribers write prescriptions, so we need a way to store our prescription information.

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it’s Needed |
| PK: PrescriptionNumber | The number of the prescription | Need to keep unique track of each prescription |
| FK: PrescriberNumber | The unique identifier of the prescriber | Needed to link the prescription to its prescriber |
| FK: UPC | The UPC of the requested product | Needed to link the prescription to its products. Note: there can be multiple products to one prescription. |
| FK: UserName | The username of the person the prescription belongs to | Prescriptions belong to certain users so we need to identify who they belong to |
| Dosage | Dosing size of the medication | Each prescription should have a dosage size since drugs come in multiple dosage sizes we need to know how much medication to dispense |
| Frequency | Prescription frequency to take medication | Each prescription has instructions for how often to take medication; this needs to be identified and stated on medication dispenses |
| Quantity | The number of medication (pills, normally) to dispense | Prescribers specify in prescriptions how much of each medication to dispense (eg, 90 pills, etc.); need to know how much to dispense |

1. We need to maintain a table for all our prescriber’s information. This links prescribers and prescriptions.

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it’s Needed |
| PK: PrescriberNumber | The unique identifier of the prescriber | Needed to link the prescription to its prescriber |
| Name | Name of the prescriber | Easy lookup for prescribers |
| Phone | Phone number of the prescriber | In case contact is needed |
| Address\_id | Address of the prescriber from the Address table | Regulatory – must maintain as part of prescriptions |
| LicenseNumber | License number of the prescriber | Regulatory – must maintain as part of prescriptions |

1. Lastly, since orders to products is a many-to-many relationship, we need a bridging table to link the two. We create line items, that have the product’s UPC, along with the order number it links to and the quantity.

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it’s Needed |
| PK: LineItemCode | The unique identifier of the line item | Unique synthetic key for the table |
| FK: UPC | The UPC of the item in the line item | Need to link the line item back to the product |
| FK: OrderNumber | The OrderNumber that contains the line item | Need to link the line item to a corresponding order |
| Quantity | The quantity of the product the line item contains | While a line item may have the same products, the number of products will vary across orders |

1. Normalizing the database meant splitting address into a separate table.

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it’s Needed |
| PK: Address\_id | Synthetic key for the address | Synthetic key that we can uniquel increment and relate to address |
| StreetAddress | Street address (eg, 123 Fake Street) | Need to know to deliver items |
| State | The state (eg, MA); longer varchar is longer state code is needed (such as APO/FPO) | Need to know the state to deliver items |
| City | City | Need to know the City to deliver items |
| Zipcode | Zipcode; VARCHAR10 for extended zipcodes (such as 55555-5555) | Need to know the Zipcode to deliver items |

1. Normalizing the database means we need to split dosage and frequency to a different table from prescriptions.

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it’s Needed |
| PK: Regimen\_id | Synthetic key for the regimen | Synthetic key that we can uniquely increment and relate to regimen |
| Dosage | The dosage to be given (eg, 30mg, 40 milliliters, etc.) | Need to know how much to give the patient |
| Frequency | How often the dosage should be given (eg, once per 12 hours, etc.) | Need to know how often dosage to be given to patient |

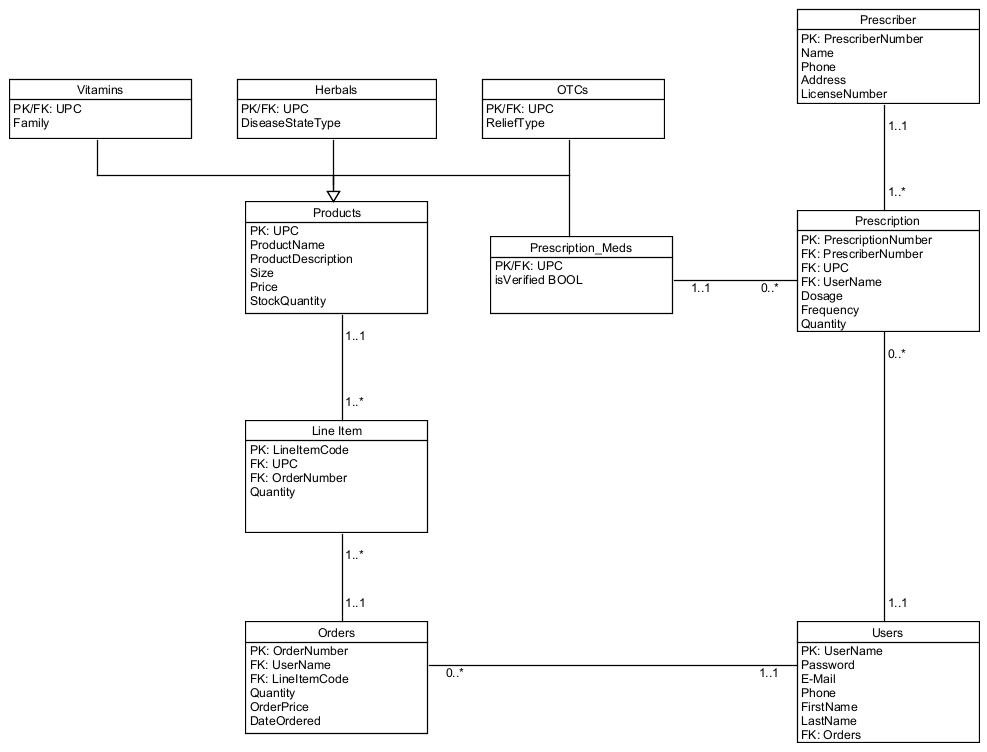
1. We need a table to track price changes.

|  |  |  |
| --- | --- | --- |
| Field | What it stores | Why it’s Needed |
| PK: PriceChangeID | Synthetic key sequence of price changes | Need a synthetic key to keep track of all price changes |
| OldPrice | The previous price | Need to track what the price used to be |
| NewPrice | The new or updated price | Need to be able to tell what the price was changed to |
| UPC | The UPC of the product | Need to know which product the price change applies to |
| ChangeDate | The date the change was made | Need to know the date the price change was made |

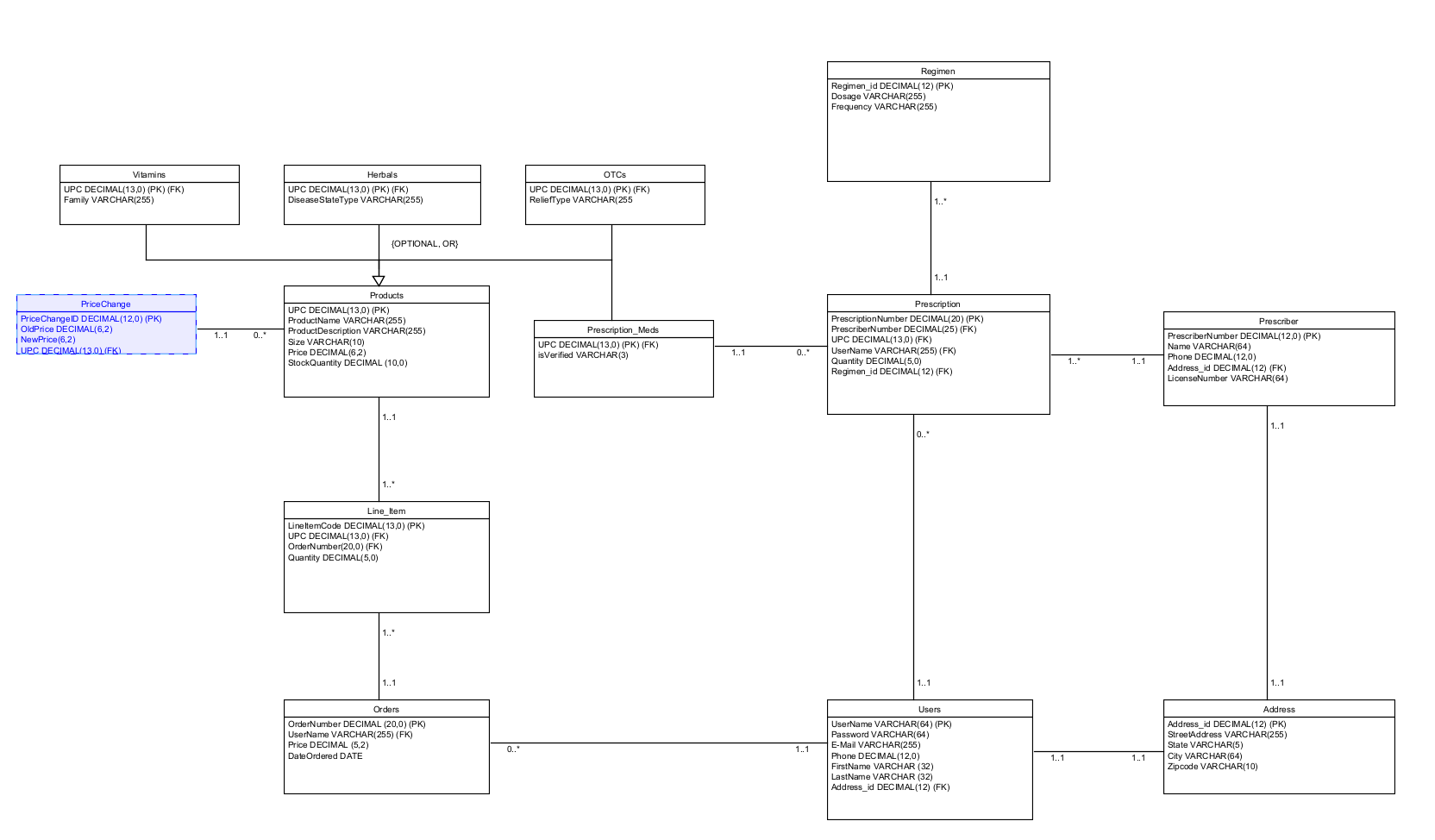
# Structural Database Rules

1. A user may place many orders; each order must be placed by 1 user.
2. An order contains 1 or more line items; a line item belongs to 1 order.
3. A line item contains 1 product; a product may be found in many line items.
4. A product (eg, prescription medication) may require a prescription if it’s a prescription medication; a prescription is written for 1 product.
5. A prescriber can write many prescriptions; a prescription can be written by 1 and only 1 prescriber.
6. A user may have many prescriptions; a prescription belongs to 1 user.
7. A product may be a vitamin, herbal, OTC medication, OR prescription medication, or none of these.
8. A prescription may have multiple regimens; a regimen may be used by a prescription.
9. A prescriber has 1 address; an address belongs to 1 prescriber.
10. A user has 1 address; an address belongs to 1 user.
11. A product can have 0 to many price changes; price changes belong to 1 product.

# Conceptual Entity-Relationship Diagram



# Full DBMS Physical ERD



We added all data types and all of our PKs, FKs, and specialization-generalizations to our concept diagram to create our physical ERD. We stuck to the rule of creating a synthetic key for each table, except under product, where we must use the product’s UPC (since it is tied to barcode). Most of the types are VARCHAR, but a few outliers:

1. PrescriberNumber should be a number and it should auto-increment as more prescribers are added to the database.
2. LicenseNumber should be a number with no decimal spots.
3. PrescriptionNumber should be a number and it should auto-increment as more prescriptions are added to the database.
4. isVerified should be either true or false, depending on when the prescription verification is complete.
5. UPC is a 13 digit number tied to the product’s barcode for scanning and inventory management purposes.
6. StockQuantity should be a number and should never have a decimal (how can you have half a pill?).
7. OrderNumber should be a number and it should auto-increment as more orders are added to the database.
8. Quantity should be a number with no decimals.
9. OrderPrice should be a decimal with 2 decimal spots to reflect the order’s price.
10. DateOrdered should be automatically updated based on the date the order was placed. Should be a DATE type.

For specialization-generalization, products can be either vitamins, OTC, herbals, or prescription medications. They cannot be a combination of any of the other sub-types. In addition, a product could be none of these things (maybe Phil’s Pills sells pill holders or glucometers, for instance). Since it can be any of those we assign {OR} or none of those {OPTIONAL} giving us a specialization-generalization of {OPTIONAL, OR}.

To normalize the database, I made 2 key changes: I split address into a separate table since it needs to be done for 1NF normalization and I split prescription dosage/frequency into a separate table to give them 3NF normalization.

# MS Server SQL Reset Script

Reset Script

-- removing foreign key constraints so tables drop without error

ALTER TABLE Users DROP CONSTRAINT fk\_Users\_Address\_id;

ALTER TABLE Prescriber DROP CONSTRAINT fk\_Prescriber\_Address\_id;

ALTER TABLE Prescription DROP CONSTRAINT fk\_Prescription\_PrescriberNumber;

ALTER TABLE Prescription DROP CONSTRAINT fk\_Prescription\_UPC;

ALTER TABLE Prescription DROP CONSTRAINT fk\_Prescription\_UserName;

ALTER TABLE Prescription DROP CONSTRAINT fk\_Prescription\_Regimen\_id;

ALTER TABLE Vitamins DROP CONSTRAINT fk\_Vitamins\_UPC;

ALTER TABLE Herbals DROP CONSTRAINT fk\_Herbals\_UPC;

ALTER TABLE OTCs DROP CONSTRAINT fk\_OTCs\_UPC;

ALTER TABLE Prescription\_Meds DROP CONSTRAINT fk\_Prescription\_Meds\_UPC;

ALTER TABLE Line\_Item DROP CONSTRAINT fk\_Line\_Item\_UPC;

ALTER TABLE Line\_Item DROP CONSTRAINT fk\_Line\_Item\_OrderNumber;

ALTER TABLE Orders DROP CONSTRAINT fk\_Orders\_UserName;

-- initial drop table commands

DROP TABLE Users;

DROP TABLE Prescriber;

DROP TABLE Address;

DROP TABLE Prescription;

DROP TABLE Regimen;

DROP TABLE Products;

DROP TABLE Vitamins;

DROP TABLE Herbals;

DROP TABLE OTCs;

DROP TABLE Prescription\_Meds;

DROP TABLE Line\_Item;

DROP TABLE Orders;

-- initial drop sequence commands

DROP SEQUENCE PrescriberNumber\_seq;

DROP SEQUENCE Address\_id\_seq;

DROP SEQUENCE PrescriptionNumber\_seq;

DROP SEQUENCE Regimen\_id\_seq;

DROP SEQUENCE LineItemCode\_seq;

DROP SEQUENCE OrderNumber\_seq;

# MS Server SQL Initialization Script

Initialize Tables Script

-- creating starter tables

CREATE TABLE Users(

Username VARCHAR(64) PRIMARY KEY,

Password VARCHAR(64) NOT NULL,

E\_Mail VARCHAR(255) NOT NULL,

Phone DECIMAL(12,0),

FirstName VARCHAR(32) NOT NULL,

LastName VARCHAR(32) NOT NULL,

Address\_id DECIMAL(12) -- Needs foreign key

);

CREATE TABLE Prescriber(

PrescriberNumber DECIMAL(12) PRIMARY KEY, -- needs sequence

Name VARCHAR(64) NOT NULL,

Phone DECIMAL(12),

Address\_id DECIMAL(12), -- Needs foreign key

LicenseNumber VARCHAR(64) NOT NULL

);

CREATE TABLE Address(

Address\_id DECIMAL(12) PRIMARY KEY NOT NULL, -- needs sequence

StreetAddress VARCHAR(255) NOT NULL,

City VARCHAR(64) NOT NULL,

State VARCHAR(5) NOT NULL,

Zipcode VARCHAR(10) NOT NULL

);

CREATE TABLE Prescription(

PrescriptionNumber DECIMAL(20) PRIMARY KEY NOT NULL, -- needs sequence

PrescriberNumber DECIMAL(12) NOT NULL, -- Needs foreign Key

UPC DECIMAL(13) NOT NULL, -- Needs Foreign Key

UserName VARCHAR(64) NOT NULL, -- Needs Foreign Key

Quantity DECIMAL(5,0) NOT NULL,

Regimen\_id DECIMAL(12) NOT NULL -- Needs foreign key

);

CREATE TABLE Regimen(

Regimen\_id DECIMAL(12) PRIMARY KEY NOT NULL, -- needs sequence

Dosage VARCHAR(255) NOT NULL,

Frequency VARCHAR(255) NOT NULL

);

CREATE TABLE Products(

UPC DECIMAL(13) PRIMARY KEY NOT NULL,

ProductName VARCHAR(255) NOT NULL,

ProductDescription VARCHAR(255),

Size VARCHAR(64),

Price DECIMAL(6,2) NOT NULL,

StockQuantity DECIMAL(10) NOT NULL

);

CREATE TABLE Vitamins(

UPC DECIMAL(13) PRIMARY KEY, -- needs foreign key

Family VARCHAR(255)

);

CREATE TABLE Herbals(

UPC DECIMAL(13) PRIMARY KEY, -- needs foreign key

DiseaseStateType VARCHAR(255)

);

CREATE TABLE OTCs(

UPC DECIMAL(13) PRIMARY KEY, -- needs foreign key

ReliefType VARCHAR(255)

);

CREATE TABLE Prescription\_Meds(

UPC DECIMAL(13) PRIMARY KEY, -- needs foreign key

DrugClass DECIMAL(1)

);

CREATE TABLE Line\_Item(

LineItemCode DECIMAL(13) PRIMARY KEY NOT NULL, -- needs sequence

UPC DECIMAL(13) NOT NULL, -- needs foreign key

OrderNumber DECIMAL(20), -- needs foreign key (also, is this needed? not sure, think its needed to create the 'bridge' between orders)

Quantity DECIMAL(5) NOT NULL

);

CREATE TABLE Orders(

OrderNumber DECIMAL(20) PRIMARY KEY NOT NULL, -- needs sequence

UserName VARCHAR(64) NOT NULL, -- needs foreign key

Price DECIMAL(5,2),

DateOrdered DATE NOT NULL

);

-- adding foreign key constraints

ALTER TABLE Users ADD CONSTRAINT fk\_Users\_Address\_id FOREIGN KEY (Address\_id) REFERENCES Address(Address\_id);

ALTER TABLE Prescriber ADD CONSTRAINT fk\_Prescriber\_Address\_id FOREIGN KEY (Address\_id) REFERENCES Address(Address\_id);

ALTER TABLE Prescription ADD CONSTRAINT fk\_Prescription\_PrescriberNumber FOREIGN KEY (PrescriberNumber) REFERENCES Prescriber(PrescriberNumber);

ALTER TABLE Prescription ADD CONSTRAINT fk\_Prescription\_UPC FOREIGN KEY (UPC) REFERENCES Prescription\_Meds(UPC);

ALTER TABLE Prescription ADD CONSTRAINT fk\_Prescription\_UserName FOREIGN KEY (UserName) REFERENCES Users(UserName);

ALTER TABLE Prescription ADD CONSTRAINT fk\_Prescription\_Regimen\_id FOREIGN KEY (Regimen\_id) REFERENCES Regimen(Regimen\_id);

ALTER TABLE Vitamins ADD CONSTRAINT fk\_Vitamins\_UPC FOREIGN KEY (UPC) REFERENCES Products(UPC);

ALTER TABLE Herbals ADD CONSTRAINT fk\_Herbals\_UPC FOREIGN KEY (UPC) REFERENCES Products(UPC);

ALTER TABLE OTCs ADD CONSTRAINT fk\_OTCs\_UPC FOREIGN KEY (UPC) REFERENCES Products(UPC);

ALTER TABLE Prescription\_Meds ADD CONSTRAINT fk\_Prescription\_Meds\_UPC FOREIGN KEY (UPC) REFERENCES Products(UPC);

ALTER TABLE Line\_Item ADD CONSTRAINT fk\_Line\_Item\_UPC FOREIGN KEY (UPC) REFERENCES Products(UPC);

ALTER TABLE Line\_Item ADD CONSTRAINT fk\_Line\_Item\_OrderNumber FOREIGN KEY (OrderNumber) REFERENCES Orders(OrderNumber);

ALTER TABLE Orders ADD CONSTRAINT fk\_Orders\_UserName FOREIGN KEY (UserName) REFERENCES Users(UserName);

-- creating sequences for synthetics

CREATE SEQUENCE PrescriberNumber\_seq START WITH 1;

CREATE SEQUENCE Address\_id\_seq START WITH 1;

CREATE SEQUENCE PrescriptionNumber\_seq START WITH 1;

CREATE SEQUENCE Regimen\_id\_seq START WITH 1;

CREATE SEQUENCE LineItemCode\_seq START WITH 1;

CREATE SEQUENCE OrderNumber\_seq START WITH 1;

# MS Server SQL Procedures Script

Create Procedures Script

-- creating procedure to add address

CREATE OR ALTER PROCEDURE add\_address

@StreetAddress VARCHAR(255),

@City VARCHAR(64),

@State VARCHAR(5),

@Zipcode VARCHAR(10)

AS

BEGIN

DECLARE @Address\_id\_seq INT = NEXT VALUE FOR Address\_id\_seq;

INSERT INTO Address(Address\_id, StreetAddress, City, State, Zipcode)

VALUES(@Address\_id\_seq,

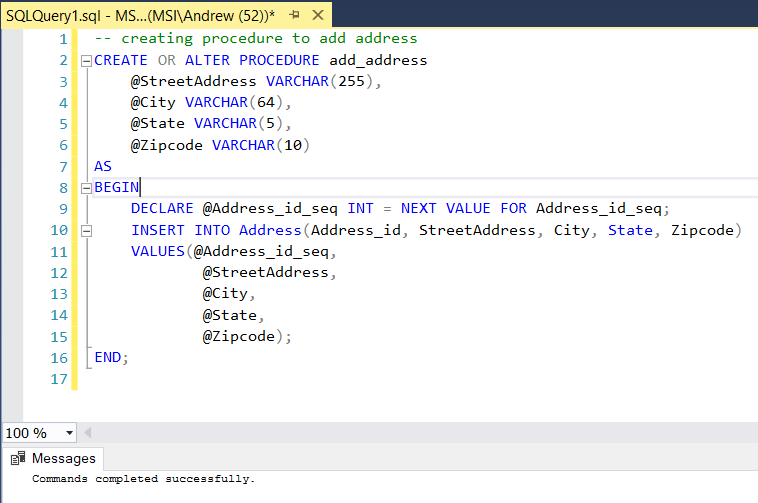
@StreetAddress,

@City,

@State,

@Zipcode);

END;



-- creating procedure to add user

CREATE OR ALTER PROCEDURE add\_user

@UserName VARCHAR(64),

@Password VARCHAR(64),

@E\_Mail VARCHAR(255),

@Phone DECIMAL(12,0),

@FirstName VARCHAR(32),

@LastName VARCHAR(32),

@Address\_id DECIMAL(12)

AS

BEGIN

INSERT INTO Users(Username, Password, E\_Mail, Phone, FirstName, LastName, Address\_id)

VALUES(@UserName,

@Password,

@E\_Mail,

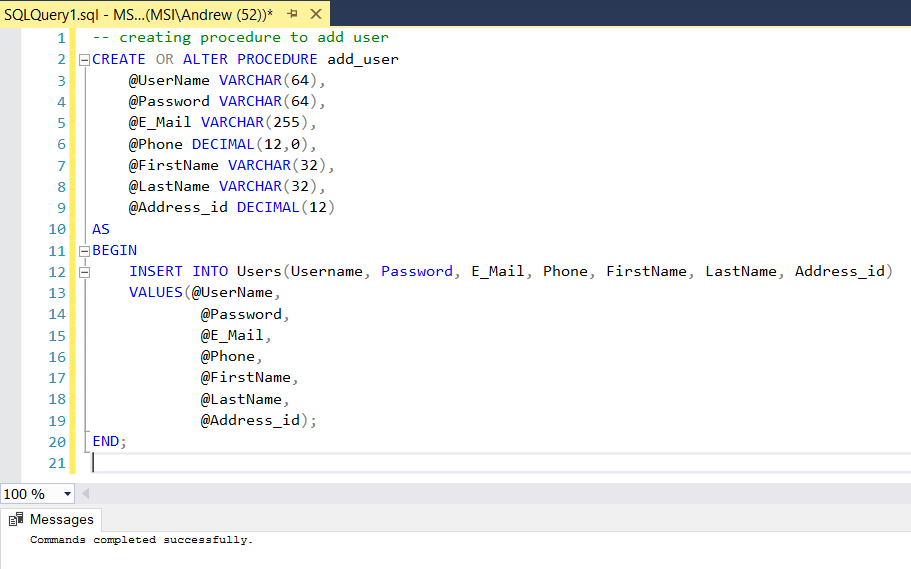
@Phone,

@FirstName,

@LastName,

@Address\_id);

END;



-- creating procedure to add prescriber

CREATE OR ALTER PROCEDURE add\_prescriber

@Name VARCHAR(64),

@Phone DECIMAL(12),

@Address\_id DECIMAL(12),

@LicenseNumber VARCHAR(64)

AS

BEGIN

DECLARE @PrescriberNumber\_seq INT = NEXT VALUE FOR PrescriberNumber\_seq;

INSERT INTO Prescriber(PrescriberNumber, Name, Phone, Address\_id, LicenseNumber)

VALUES(@PrescriberNumber\_seq,

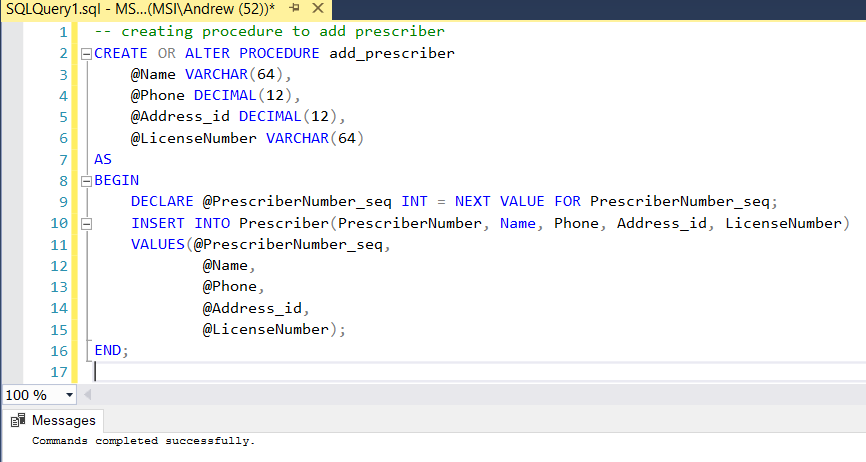
@Name,

@Phone,

@Address\_id,

@LicenseNumber);

END;



-- creating procedure to add regimen

CREATE OR ALTER PROCEDURE add\_regimen

@dosage VARCHAR(255),

@frequency VARCHAR(255)

AS

BEGIN

DECLARE @Regimen\_id\_seq INT = NEXT VALUE FOR Regimen\_id\_seq;

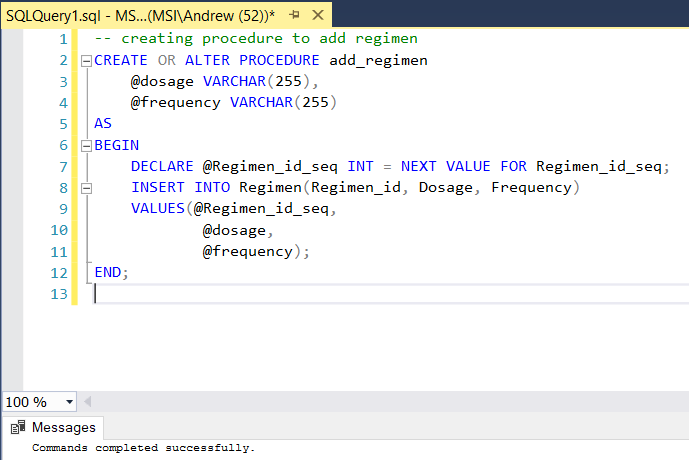
INSERT INTO Regimen(Regimen\_id, Dosage, Frequency)

VALUES(@Regimen\_id\_seq,

@dosage,

@frequency);

END;



-- creating procedure to add prescription

CREATE OR ALTER PROCEDURE add\_prescription

@PrescriberNumber DECIMAL(12),

@UPC DECIMAL(13),

@UserName VARCHAR(64),

@Quantity DECIMAL(5,0),

@Regimen\_id DECIMAL(12)

AS

BEGIN

DECLARE @PrescriptionNumber\_seq INT = NEXT VALUE FOR PrescriptionNumber\_seq;

INSERT INTO Prescription(PrescriptionNumber,

PrescriberNumber,

UPC,

UserName,

Quantity,

Regimen\_id)

VALUES(@PrescriptionNumber\_seq,

@PrescriberNumber,

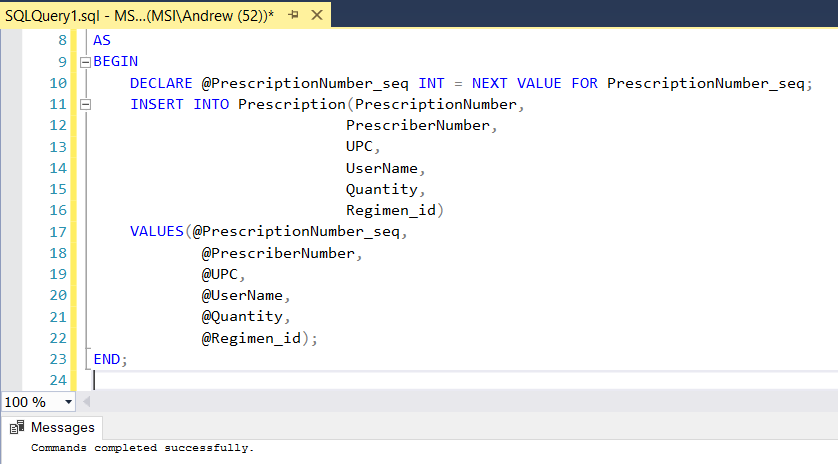
@UPC,

@UserName,

@Quantity,

@Regimen\_id);

END;



-- creating procedure to add product

CREATE OR ALTER PROCEDURE add\_product

@UPC DECIMAL(13),

@ProductName VARCHAR(255),

@ProductDescription VARCHAR(255),

@Size VARCHAR(64),

@Price DECIMAL(6,2),

@StockQuantity DECIMAL(10)

AS

BEGIN

INSERT INTO Products(UPC,

ProductName,

ProductDescription,

Size,

Price,

StockQuantity)

VALUES(@UPC,

@ProductName,

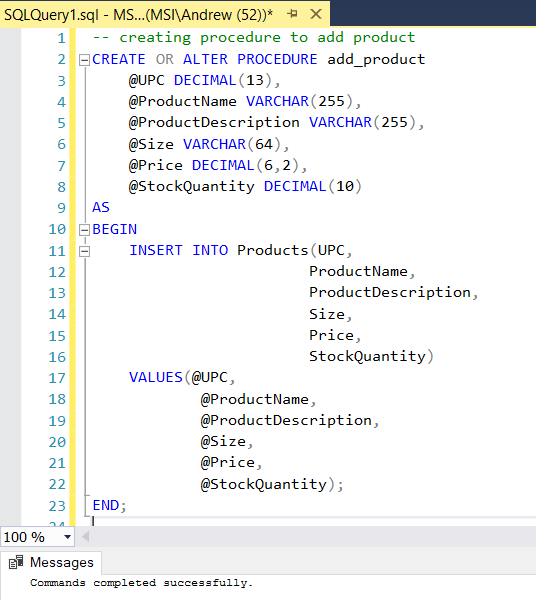
@ProductDescription,

@Size,

@Price,

@StockQuantity);

END;



-- creating procedure to add vitamin

CREATE OR ALTER PROCEDURE add\_vitamin

@Family VARCHAR(255),

@UPC DECIMAL(13),

@ProductName VARCHAR(255),

@ProductDescription VARCHAR(255),

@Size VARCHAR(64),

@Price DECIMAL(6,2),

@StockQuantity DECIMAL(10)

AS

BEGIN

INSERT INTO Products(UPC,

ProductName,

ProductDescription,

Size,

Price,

StockQuantity)

VALUES(@UPC,

@ProductName,

@ProductDescription,

@Size,

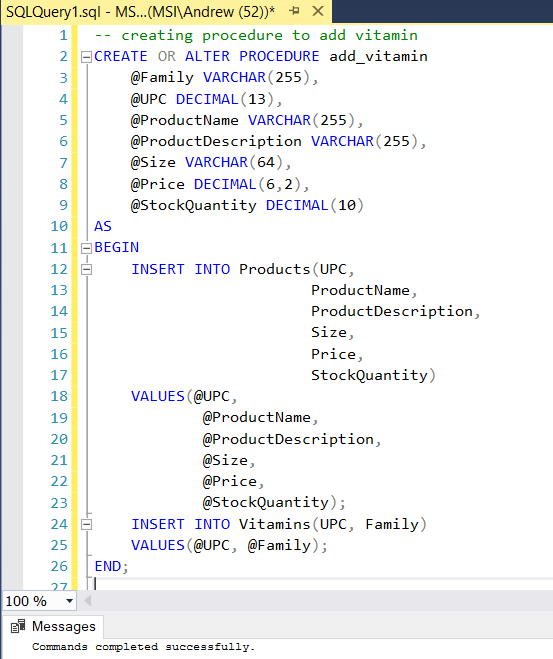
@Price,

@StockQuantity);

INSERT INTO Vitamins(UPC, Family)

VALUES(@UPC, @Family);

END;



-- creating procedure to add herbal

CREATE OR ALTER PROCEDURE add\_herbal

@DiseaseStateType VARCHAR(255),

@UPC DECIMAL(13),

@ProductName VARCHAR(255),

@ProductDescription VARCHAR(255),

@Size VARCHAR(64),

@Price DECIMAL(6,2),

@StockQuantity DECIMAL(10)

AS

BEGIN

INSERT INTO Products(UPC,

ProductName,

ProductDescription,

Size,

Price,

StockQuantity)

VALUES(@UPC,

@ProductName,

@ProductDescription,

@Size,

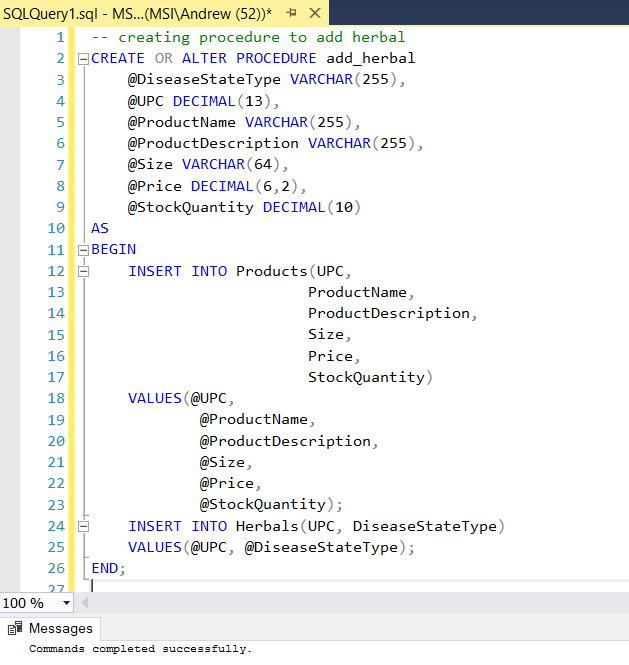
@Price,

@StockQuantity);

INSERT INTO Herbals(UPC, DiseaseStateType)

VALUES(@UPC, @DiseaseStateType);

END;



-- creating procedure to add OTC

CREATE OR ALTER PROCEDURE add\_OTC

@ReliefType VARCHAR(255),

@UPC DECIMAL(13),

@ProductName VARCHAR(255),

@ProductDescription VARCHAR(255),

@Size VARCHAR(64),

@Price DECIMAL(6,2),

@StockQuantity DECIMAL(10)

AS

BEGIN

INSERT INTO Products(UPC,

ProductName,

ProductDescription,

Size,

Price,

StockQuantity)

VALUES(@UPC,

@ProductName,

@ProductDescription,

@Size,

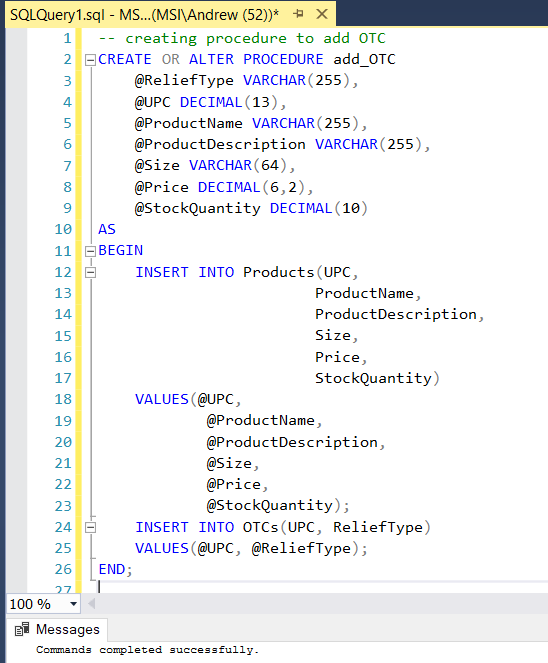
@Price,

@StockQuantity);

INSERT INTO OTCs(UPC, ReliefType)

VALUES(@UPC, @ReliefType);

END;



-- creating procedure to add prescription medication

CREATE OR ALTER PROCEDURE add\_prescription\_med

@DrugClass DECIMAL(1),

@UPC DECIMAL(13),

@ProductName VARCHAR(255),

@ProductDescription VARCHAR(255),

@Size VARCHAR(64),

@Price DECIMAL(6,2),

@StockQuantity DECIMAL(10)

AS

BEGIN

INSERT INTO Products(UPC,

ProductName,

ProductDescription,

Size,

Price,

StockQuantity)

VALUES(@UPC,

@ProductName,

@ProductDescription,

@Size,

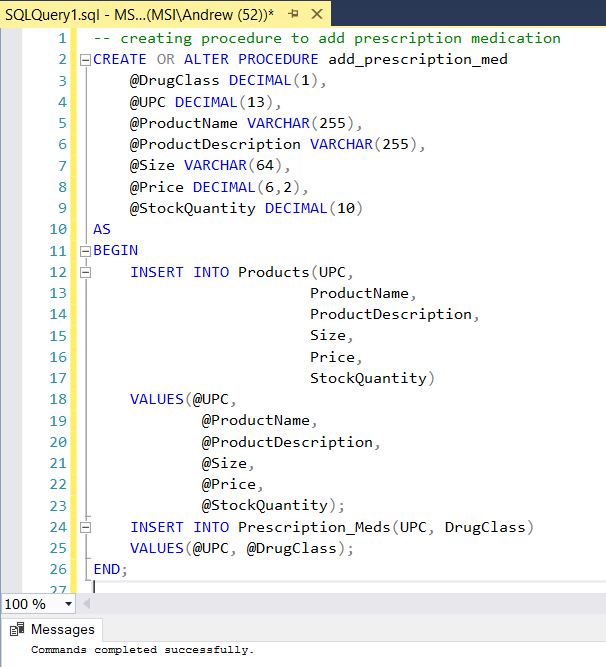
@Price,

@StockQuantity);

INSERT INTO Prescription\_Meds(UPC, DrugClass)

VALUES(@UPC, @DrugClass);

END;



-- creating procedure to add line items

CREATE OR ALTER PROCEDURE add\_line\_item

@UPC DECIMAL(13),

@OrderNumber DECIMAL(20),

@Quantity DECIMAL(5)

AS

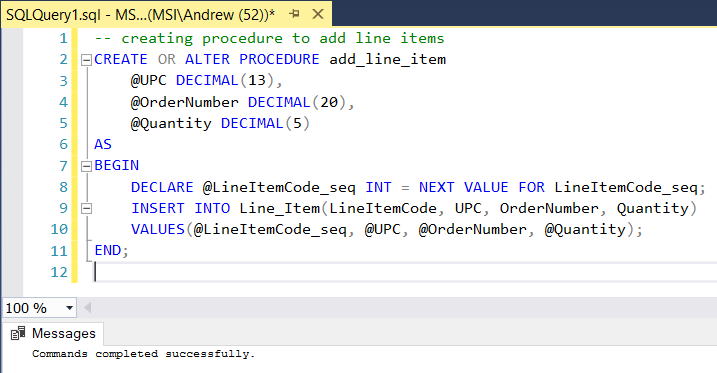
BEGIN

DECLARE @LineItemCode\_seq INT = NEXT VALUE FOR LineItemCode\_seq;

INSERT INTO Line\_Item(LineItemCode, UPC, OrderNumber, Quantity)

VALUES(@LineItemCode\_seq, @UPC, @OrderNumber, @Quantity);

END;



-- creating procedure to add order

CREATE OR ALTER PROCEDURE add\_order

@UserName VARCHAR(64),

@DateOrdered DATE

AS

BEGIN

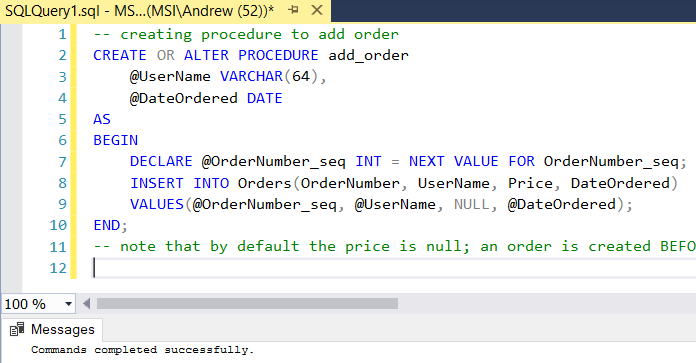
DECLARE @OrderNumber\_seq INT = NEXT VALUE FOR OrderNumber\_seq;

INSERT INTO Orders(OrderNumber, UserName, Price, DateOrdered)

VALUES(@OrderNumber\_seq, @UserName, NULL, @DateOrdered);

END;

-- note that by default the price is null; an order is created BEFORE the individual line items are created so the price is not known upon creation (line items require an order number). The price needs to be filled in later (probably by the application) once all the line items have been completed and the order has been finished. Because of this, we have a column for the order’s price, but it is set to null by default.



# MS Server SQL Data Fill Script

-- adding addresses

EXECUTE add\_address '123 Fake Street', 'Boston', 'MA', '12345';

EXECUTE add\_address '701 Winnsoro Cv', 'Round Rock', 'TX', '78669';

EXECUTE add\_address '5925 Almeda Ave', 'Houston', 'TX', '77469';

EXECUTE add\_address '3320 Harmon Ave', 'Austin', 'TX', '78705';

EXECUTE add\_address '5932 Walsh Point', 'Colorado Springs', 'CO', '80919';

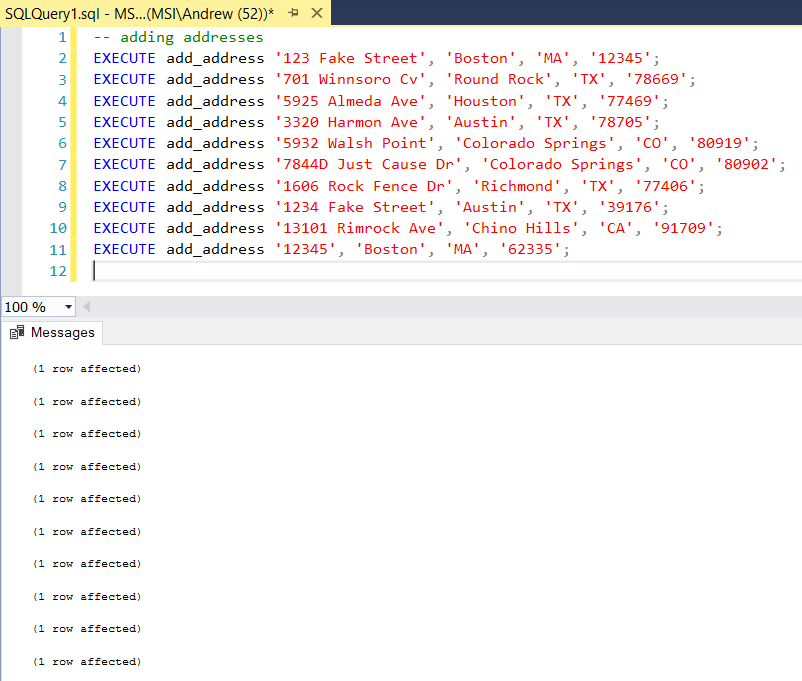
EXECUTE add\_address '7844D Just Cause Dr', 'Colorado Springs', 'CO', '80902';

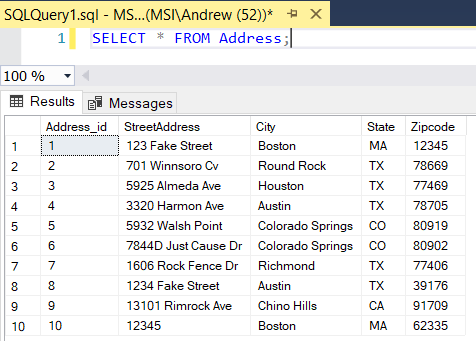
EXECUTE add\_address '1606 Rock Fence Dr', 'Richmond', 'TX', '77406';

EXECUTE add\_address '1234 Fake Street', 'Austin', 'TX', '39176';

EXECUTE add\_address '13101 Rimrock Ave', 'Chino Hills', 'CA', '91709';

EXECUTE add\_address '12345', 'Boston', 'MA', '62335';





-- adding users

EXECUTE add\_user 'nodcomandr', 'password123', 'andrew.a@gmail.com', 8326381234, 'Andrew', 'A', 1;

EXECUTE add\_user 'JohnSmith', 'JohnSmith123', 'john.smith@gmail.com', 5551234567, 'John', 'Smith', 2;

EXECUTE add\_user 'JaneSmith', 'JaneSmith123', 'jane.smith@gmail.com', 1234567555, 'Jane', 'Smith', 3;

EXECUTE add\_user 'GeorgeWashington', 'george123', 'BigGW@gmail.com', 5555551234, 'George', 'Washington', 4;

EXECUTE add\_user 'AbrahamLincoln', 'emancipation123', 'civilwar.pres@gmail.com', 8325551234, 'Abraham', 'Lincoln', 5;

EXECUTE add\_user 'TeddyRoosevelt', 'roosevelt123', 'theodore.r@gmail.com', 4132563346, 'Theodore', 'Roosevelt', 6;

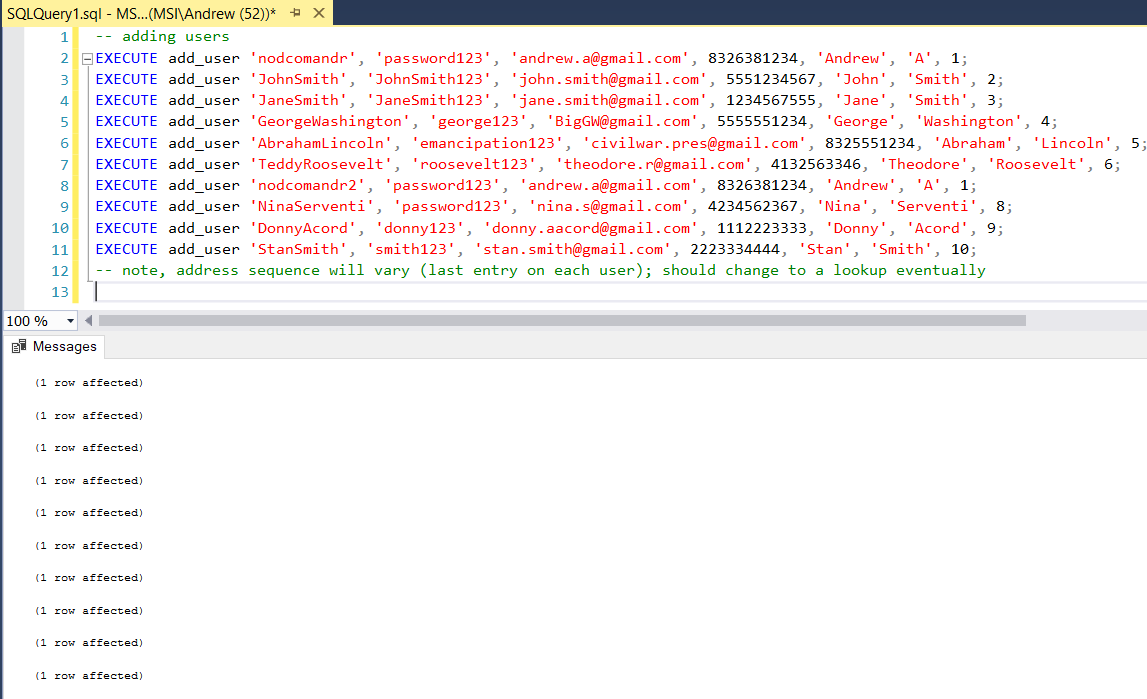
EXECUTE add\_user 'nodcomandr2', 'password123', 'andrew.a@gmail.com', 8326381234, 'Andrew', 'A', 1;

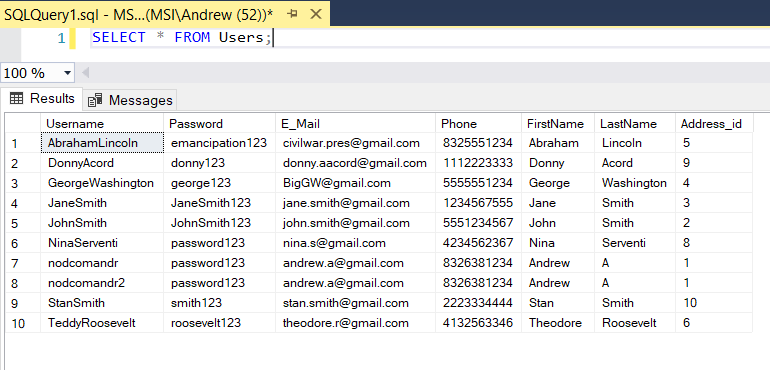
EXECUTE add\_user 'NinaServenti', 'password123', 'nina.s@gmail.com', 4234562367, 'Nina', 'Serventi', 8;

EXECUTE add\_user 'DonnyAcord', 'donny123', 'donny.aacord@gmail.com', 1112223333, 'Donny', 'Acord', 9;

EXECUTE add\_user 'StanSmith', 'smith123', 'stan.smith@gmail.com', 2223334444, 'Stan', 'Smith', 10;

-- note, address sequence will vary (last entry on each user); should change to a lookup eventually





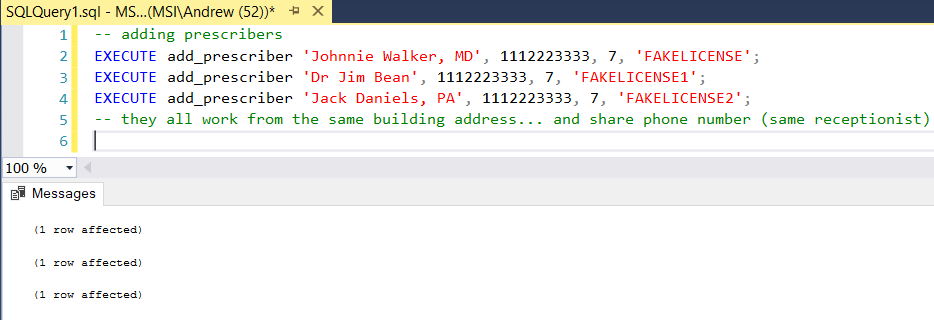
-- adding prescribers

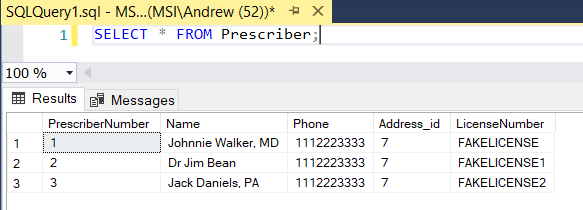
EXECUTE add\_prescriber 'Johnnie Walker, MD', 1112223333, 7, 'FAKELICENSE';

EXECUTE add\_prescriber 'Dr Jim Bean', 1112223333, 7, 'FAKELICENSE1';

EXECUTE add\_prescriber 'Jack Daniels, PA', 1112223333, 7, 'FAKELICENSE2';

-- they all work from the same building address... and share phone number (same receptionist)





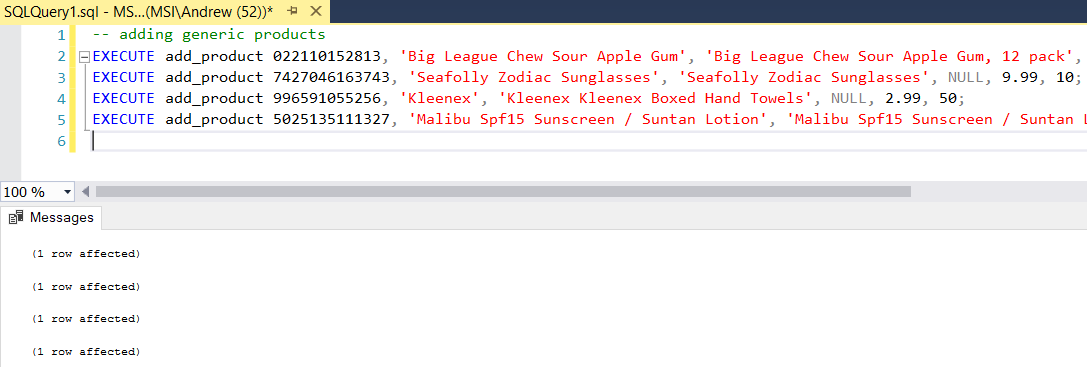
-- adding generic products

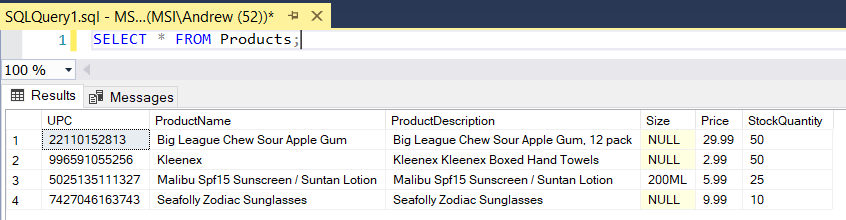
EXECUTE add\_product 022110152813, 'Big League Chew Sour Apple Gum', 'Big League Chew Sour Apple Gum, 12 pack', NULL, 29.99, 50;

EXECUTE add\_product 7427046163743, 'Seafolly Zodiac Sunglasses', 'Seafolly Zodiac Sunglasses', NULL, 9.99, 10;

EXECUTE add\_product 996591055256, 'Kleenex', 'Kleenex Kleenex Boxed Hand Towels', NULL, 2.99, 50;

EXECUTE add\_product 5025135111327, 'Malibu Spf15 Sunscreen / Suntan Lotion', 'Malibu Spf15 Sunscreen / Suntan Lotion', '200ML', 5.99, 25;

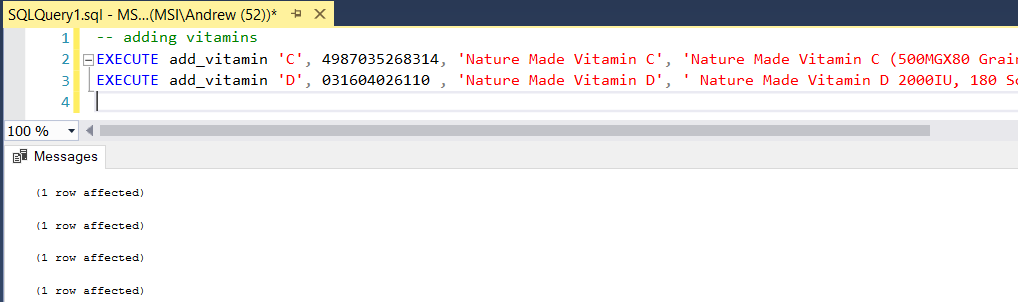


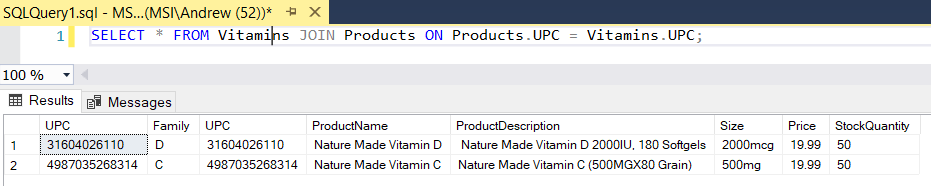


-- adding vitamins

EXECUTE add\_vitamin 'C', 4987035268314, 'Nature Made Vitamin C', 'Nature Made Vitamin C (500MGX80 Grain)', '500mg', 19.99, 50;

EXECUTE add\_vitamin 'D', 031604026110 , 'Nature Made Vitamin D', ' Nature Made Vitamin D 2000IU, 180 Softgels ', '2000mcg', 19.99, 50;

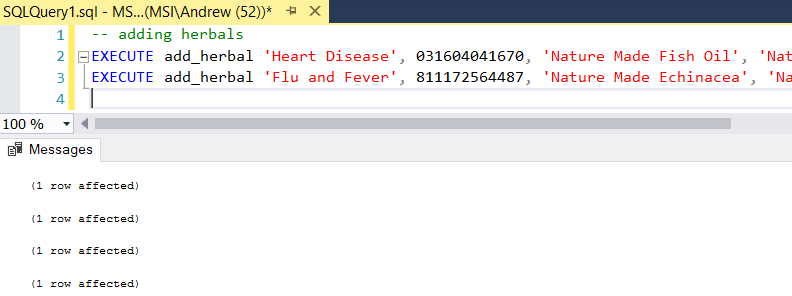


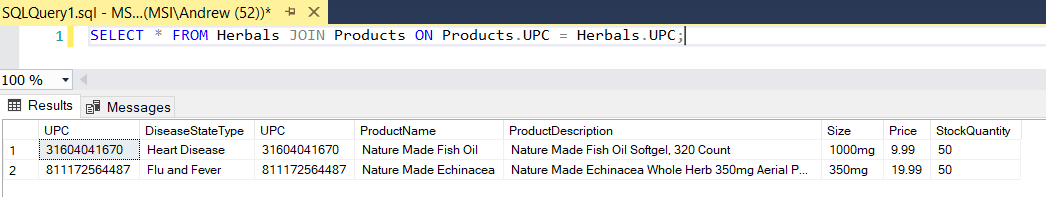


-- adding herbals

EXECUTE add\_herbal 'Heart Disease', 031604041670, 'Nature Made Fish Oil', 'Nature Made Fish Oil Softgel, 320 Count', '1000mg', 9.99, 50;

EXECUTE add\_herbal 'Flu and Fever', 811172564487, 'Nature Made Echinacea', 'Nature Made Echinacea Whole Herb 350mg Aerial Part Herbal Supplement Capsules - 100 CT', '350mg', 19.99, 50;

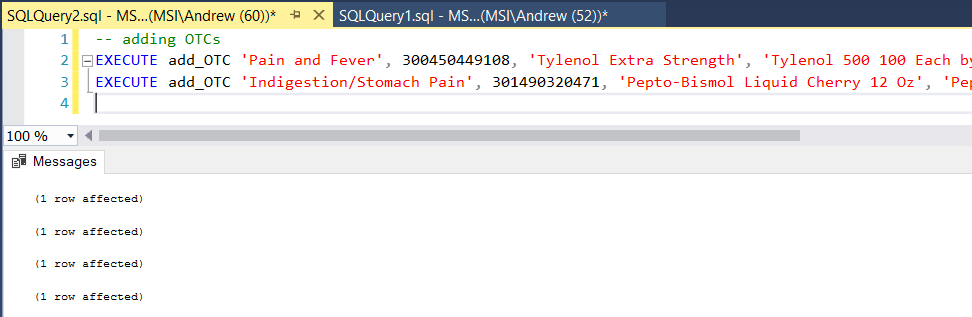


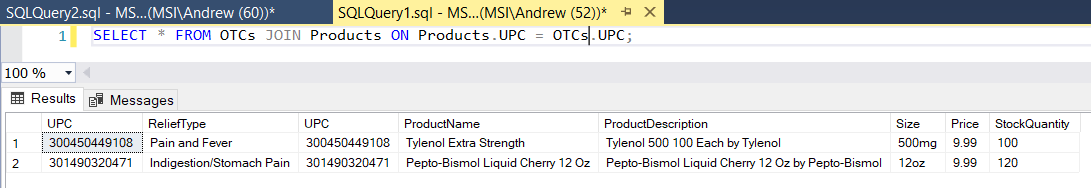


-- adding OTCs

EXECUTE add\_OTC 'Pain and Fever', 300450449108, 'Tylenol Extra Strength', 'Tylenol 500 100 Each by Tylenol', '500mg', 9.99, 100;

EXECUTE add\_OTC 'Indigestion/Stomach Pain', 301490320471, 'Pepto-Bismol Liquid Cherry 12 Oz', 'Pepto-Bismol Liquid Cherry 12 Oz by Pepto-Bismol', '12oz', 9.99, 120;





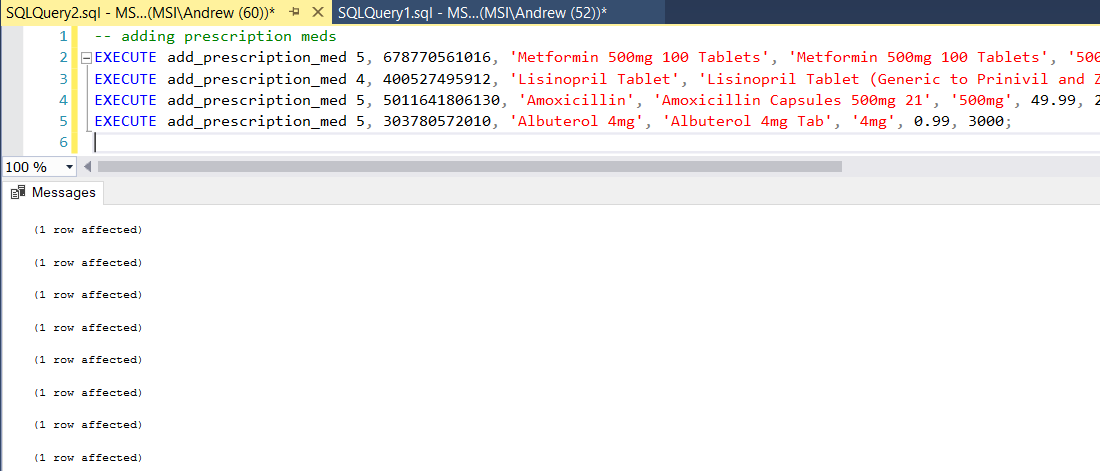
-- adding prescription meds

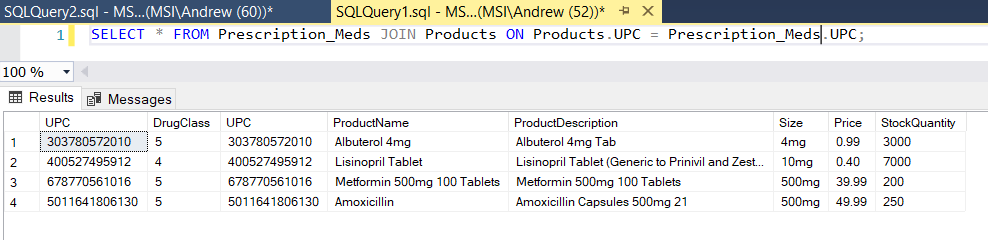
EXECUTE add\_prescription\_med 5, 678770561016, 'Metformin 500mg 100 Tablets', 'Metformin 500mg 100 Tablets', '500mg', 39.99, 200;

EXECUTE add\_prescription\_med 4, 400527495912, 'Lisinopril Tablet', 'Lisinopril Tablet (Generic to Prinivil and Zestril) Size: 10 Mg', '10mg', 0.40, 7000;

EXECUTE add\_prescription\_med 5, 5011641806130, 'Amoxicillin', 'Amoxicillin Capsules 500mg 21', '500mg', 49.99, 250;

EXECUTE add\_prescription\_med 5, 303780572010, 'Albuterol 4mg', 'Albuterol 4mg Tab', '4mg', 0.99, 3000;





-- adding dosing regimen

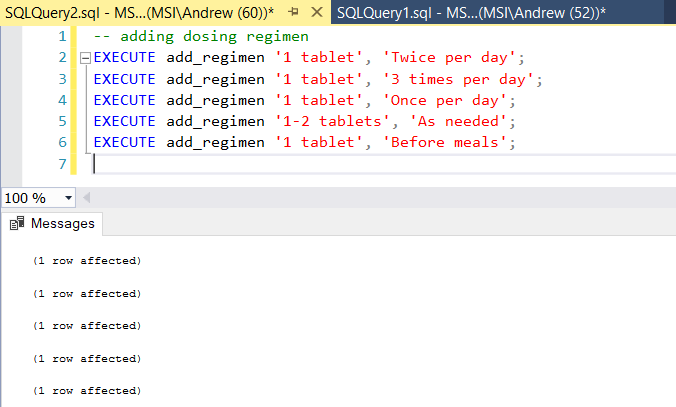
EXECUTE add\_regimen '1 tablet', 'Twice per day';

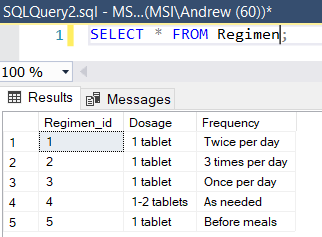
EXECUTE add\_regimen '1 tablet', '3 times per day';

EXECUTE add\_regimen '1 tablet', 'Once per day';

EXECUTE add\_regimen '1-2 tablets', 'As needed';

EXECUTE add\_regimen '1 tablet', 'Before meals';





-- adding prescriptions

-- need to make these more dynamic and less hard coded - initial implementation for testing only (technical debt) (hard coding prescriber numbers and regimen IDs is not good!)

EXECUTE add\_prescription 1, 5011641806130, 'nodcomandr', 1, 3;

EXECUTE add\_prescription 2, 303780572010, 'nodcomandr', 60, 4;

EXECUTE add\_prescription 3, 400527495912, 'nodcomandr', 90, 3;

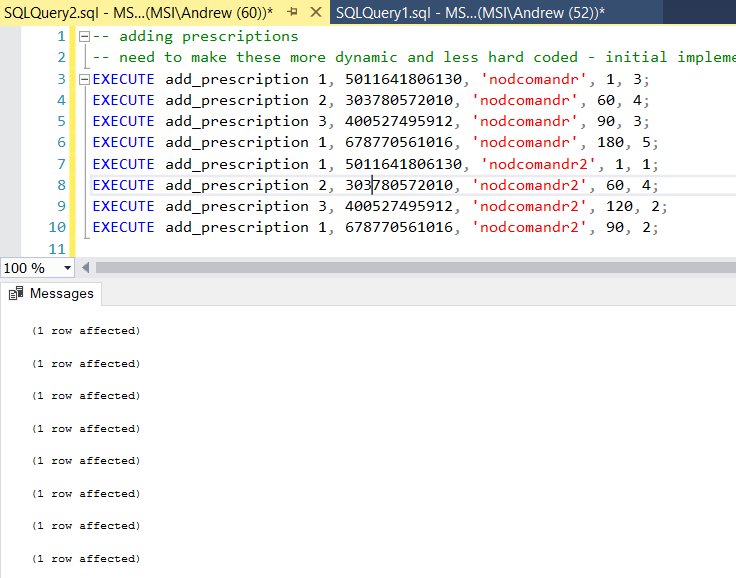
EXECUTE add\_prescription 1, 678770561016, 'nodcomandr', 180, 5;

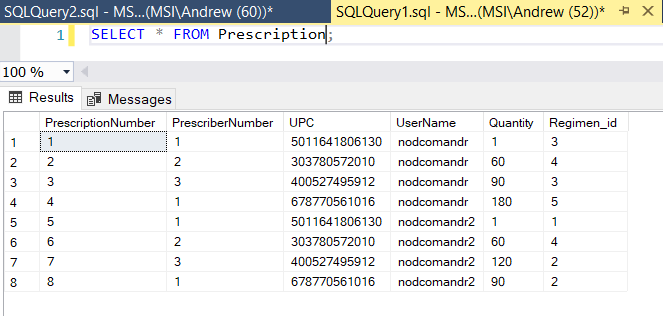
EXECUTE add\_prescription 1, 5011641806130, 'nodcomandr2', 1, 1;

EXECUTE add\_prescription 2, 303780572010, 'nodcomandr2', 60, 4;

EXECUTE add\_prescription 3, 400527495912, 'nodcomandr2', 120, 2;

EXECUTE add\_prescription 1, 678770561016, 'nodcomandr2', 90, 2;





-- adding orders

EXECUTE add\_order 'nodcomandr', '01-JAN-2022';

EXECUTE add\_order 'nodcomandr', '05-JAN-2022';

EXECUTE add\_order 'nodcomandr', '10-JAN-2022';

EXECUTE add\_order 'nodcomandr', '15-JAN-2022';

EXECUTE add\_order 'nodcomandr2', '02-JAN-2022';

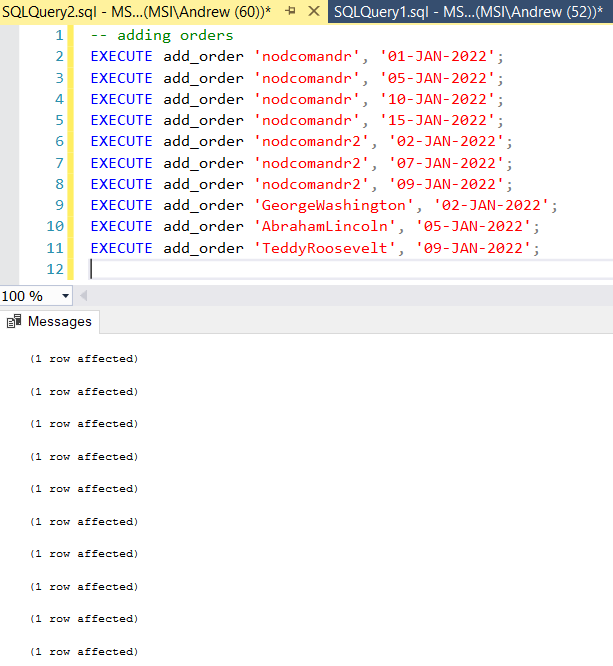
EXECUTE add\_order 'nodcomandr2', '07-JAN-2022';

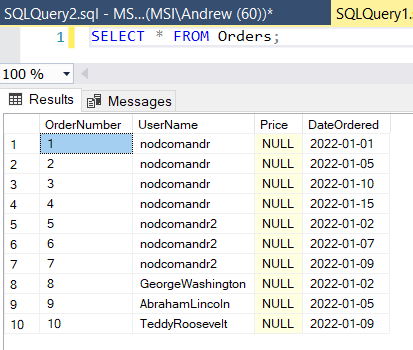
EXECUTE add\_order 'nodcomandr2', '09-JAN-2022';

EXECUTE add\_order 'GeorgeWashington', '02-JAN-2022';

EXECUTE add\_order 'AbrahamLincoln', '05-JAN-2022';

EXECUTE add\_order 'TeddyRoosevelt', '09-JAN-2022';





-- adding line items

EXECUTE add\_line\_item 5011641806130, 1, 1;

EXECUTE add\_line\_item 303780572010, 2, 60;

EXECUTE add\_line\_item 400527495912, 2, 90;

EXECUTE add\_line\_item 678770561016, 3, 180;

EXECUTE add\_line\_item 301490320471, 4, 2;

EXECUTE add\_line\_item 303780572010, 5, 60;

EXECUTE add\_line\_item 300450449108, 6, 1;

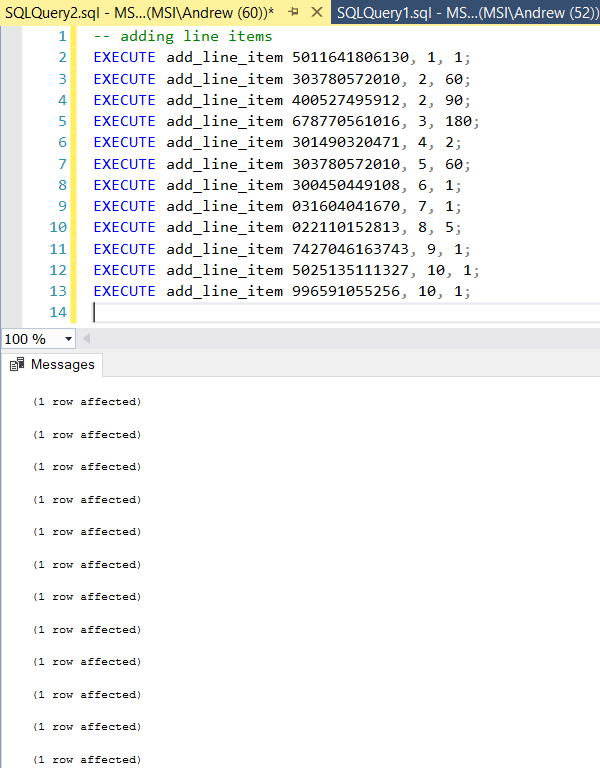
EXECUTE add\_line\_item 031604041670, 7, 1;

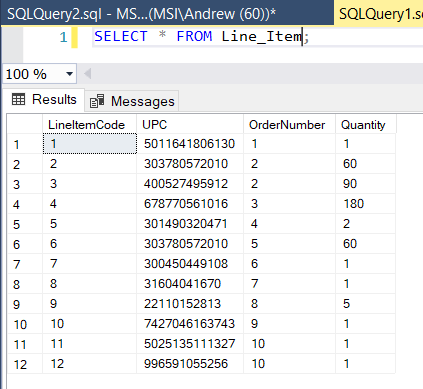
EXECUTE add\_line\_item 022110152813, 8, 5;

EXECUTE add\_line\_item 7427046163743, 9, 1;

EXECUTE add\_line\_item 5025135111327, 10, 1;

EXECUTE add\_line\_item 996591055256, 10, 1;





# MS Server SQL Queries

1. So the first question to answer by the first query: Can I pull a list of all users, their addresses, any prescriptions they might have, and any orders they have placed? Think of this as a master list of users and their info. This can give us an overall snapshot of our users, their addresses, their orders, and any prescriptions they have.

SELECT \*

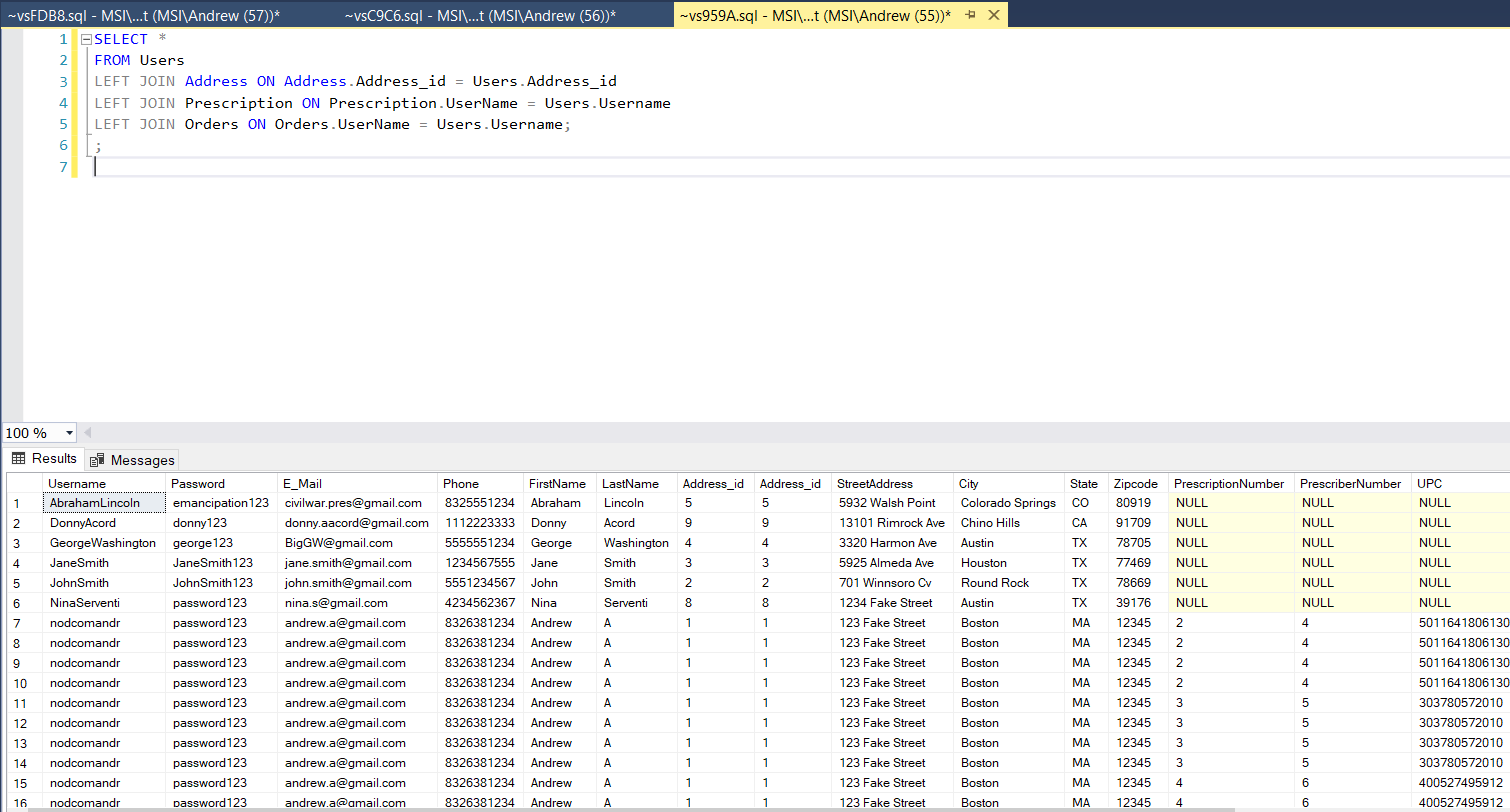
FROM Users

LEFT JOIN Address ON Address.Address\_id = Users.Address\_id

LEFT JOIN Prescription ON Prescription.UserName = Users.Username

LEFT JOIN Orders ON Orders.UserName = Users.Username;

;



1. Can we get a full list of all vitamins we have in stock? Herbals? OTC meds? Prescription meds? All products, regardless of subtype?

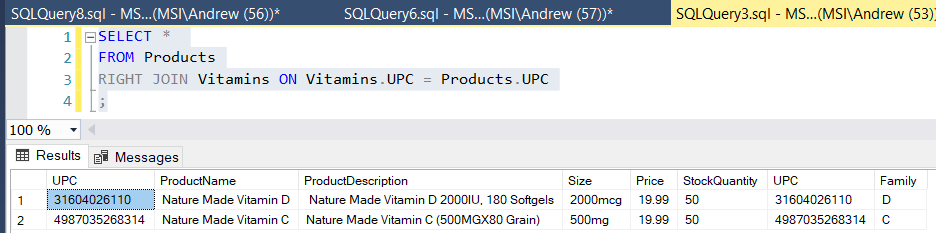
Vitamins

SELECT \*

FROM Products

RIGHT JOIN Vitamins ON Vitamins.UPC = Products.UPC

;



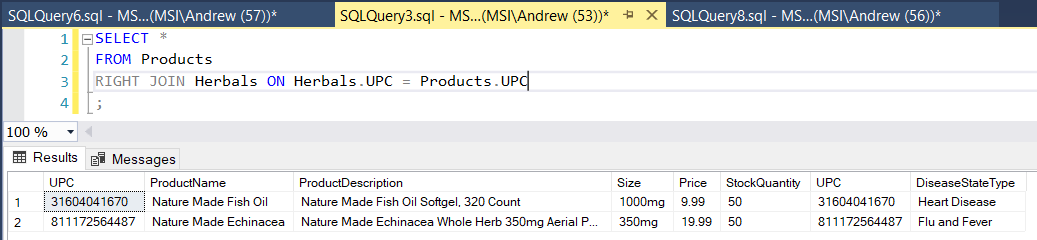
Herbals

SELECT \*

FROM Products

RIGHT JOIN Herbals ON Herbals.UPC = Products.UPC

;



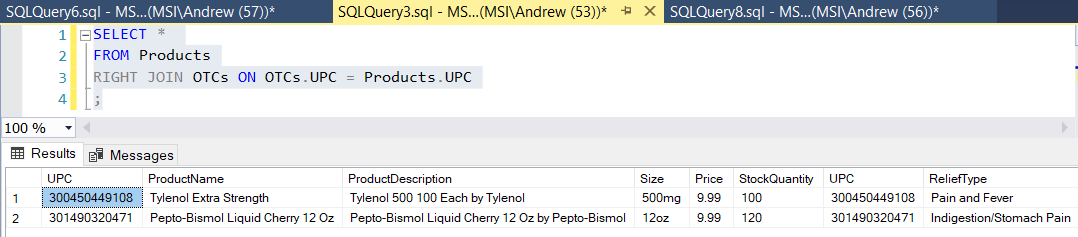
OTC

SELECT \*

FROM Products

RIGHT JOIN OTCs ON OTCs.UPC = Products.UPC

;



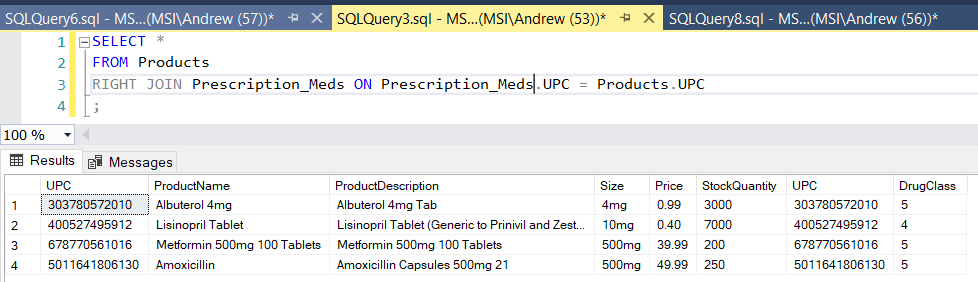
Prescription Meds

SELECT \*

FROM Products

RIGHT JOIN Prescription\_Meds ON Prescription\_Meds.UPC = Products.UPC

;

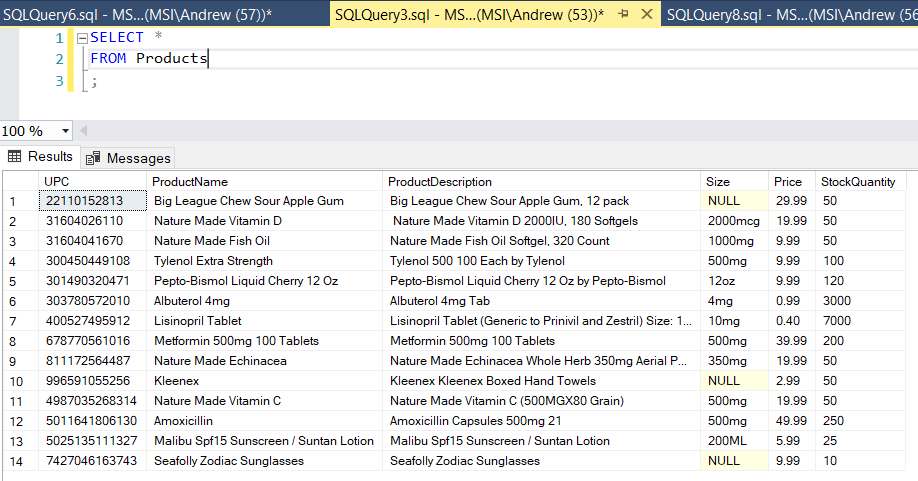


All products

SELECT \*

FROM Products

;



All products with subtypes

SELECT \*

FROM Products

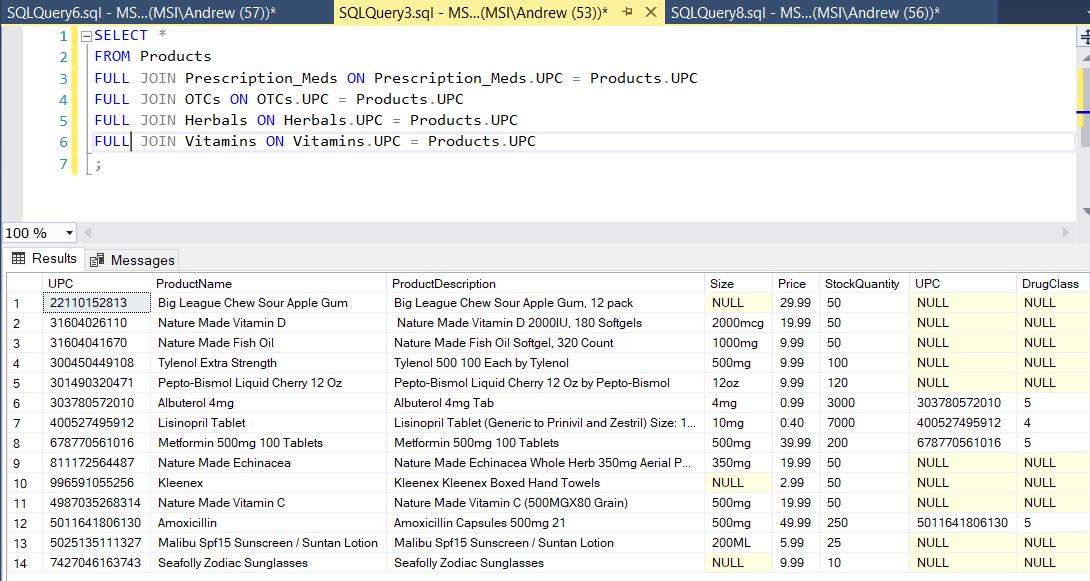
FULL JOIN Prescription\_Meds ON Prescription\_Meds.UPC = Products.UPC

FULL JOIN OTCs ON OTCs.UPC = Products.UPC

FULL JOIN Herbals ON Herbals.UPC = Products.UPC

FULL JOIN Vitamins ON Vitamins.UPC = Products.UPC

;



1. Let’s say that we want to see all orders placed by a user. We can do that by joining our users table and our orders table, then adding a WHERE clause to show only a particular username. This is handy for the app development so that it will show a user’s orders. (Note this only shows orders – if we want to see the actual items, we would also need to JOIN line items and Product UPCs).

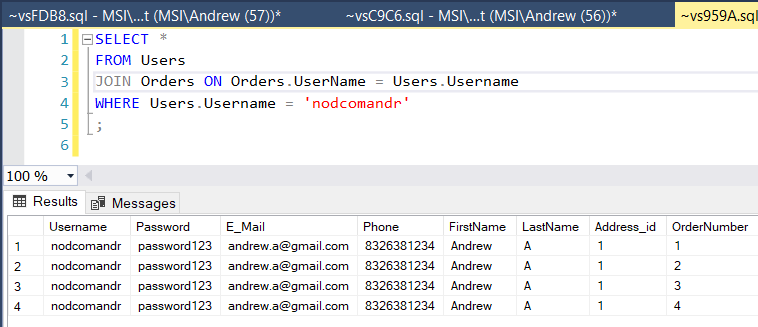
SELECT \*

FROM Users

LEFT JOIN Orders ON Orders.UserName = Users.Username

WHERE Users.Username = 'nodcomandr'

;



To take it a step further, we can do some joins and see what the user ordered and when:

SELECT Users.UserName, Products.ProductName, Products.UPC, Orders.DateOrdered, Line\_Item.Quantity

FROM Products

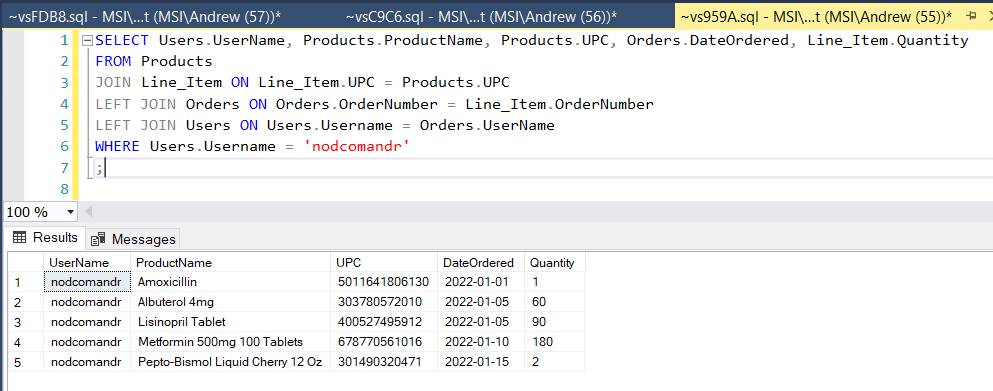
JOIN Line\_Item ON Line\_Item.UPC = Products.UPC

LEFT JOIN Orders ON Orders.OrderNumber = Line\_Item.OrderNumber

LEFT JOIN Users ON Users.Username = Orders.UserName

WHERE Users.Username = 'nodcomandr'

;



Using this query we can see that user Nodcomandr has placed 4 orders (with the one on 1-5 containing two products).

# MS Server SQL Indexes

So we don’t need to place indexes on our primary keys (good!). But we do need indexes on all of our foreign keys. Additionally, we identify 3 commonly used items to create indexes on (product stock, product name, and product’s price – all useful for displaying on a website, which means they will be frequently accessed).

-- indexing foreign keys

CREATE INDEX Users\_Address\_id\_index

ON Users(Address\_id);

CREATE INDEX Prescriber\_Address\_id\_index

ON Prescriber(Address\_id);

CREATE INDEX PrescriberNumber\_index

ON Prescription(PrescriberNumber);

CREATE INDEX Prescription\_UPC\_index

ON Prescription(UPC);

CREATE INDEX Prescription\_UserName\_index

ON Prescription(UserName);

CREATE INDEX Prescription\_Regimen\_id\_index

ON Prescription(Regimen\_id);

CREATE INDEX Line\_Item\_UPC\_index

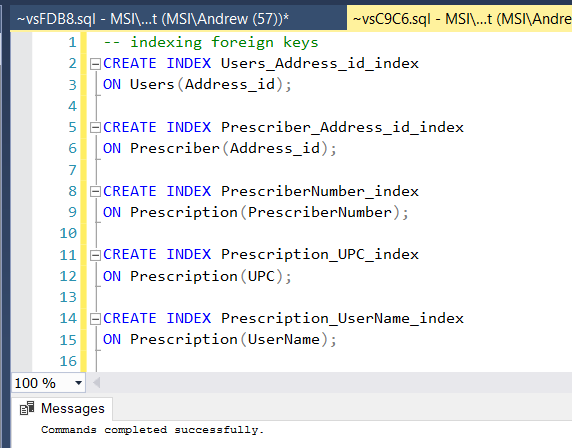
ON Line\_Item(UPC);

CREATE INDEX Line\_Item\_OrderNumber\_index

ON Line\_Item(OrderNumber);

CREATE INDEX Orders\_UserName\_index

ON Orders(UserName);



-- we would want an index on product's stock; this way we can quickly check if a product is in stock or out of stock

CREATE INDEX Products\_StockQuantity\_index

ON Products(StockQuantity);

-- we would want an index on an item's name; while the system uses UPC for products, we refer to products by their name, so placing an index here only makes sense

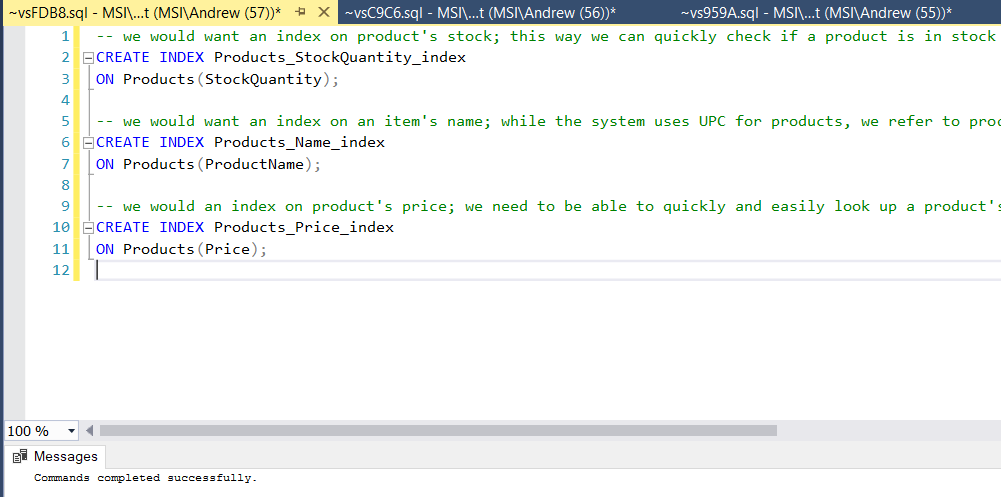
CREATE INDEX Products\_Name\_index

ON Products(ProductName);

-- we would an index on product's price; we need to be able to quickly and easily look up a product's price, especially useful for displaying products on a website quickly

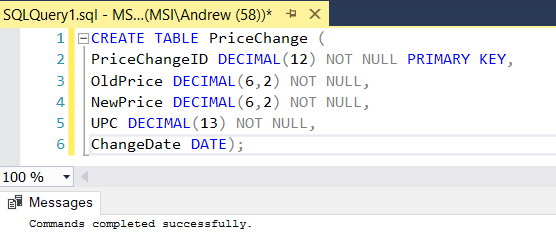
CREATE INDEX Products\_Price\_index

ON Products(Price);



# History Table Demonstration

We first need to create a table to store our price changes. We link this table in our ERD to products. We also need a new synthetic key for our price changes. Once we do that, we create a trigger on that table. Whenever an update is executed on the table, if the price is different, we insert a record of the price change into our new table that has the synthetic key, the old price, the new price, the UPC, and the date of the change.



-- creating table to store changes in price

CREATE TABLE PriceChange (

PriceChangeID DECIMAL(12) NOT NULL PRIMARY KEY,

OldPrice DECIMAL(6,2) NOT NULL,

NewPrice DECIMAL(6,2) NOT NULL,

UPC DECIMAL(13) NOT NULL,

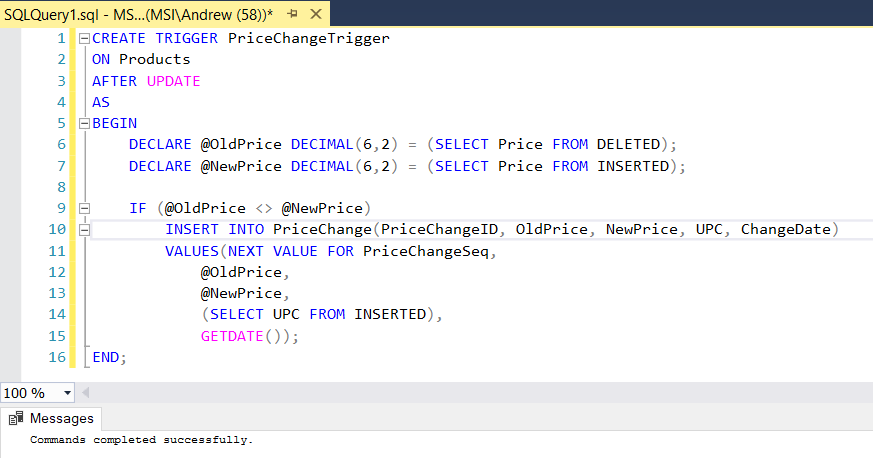
ChangeDate DATE);

ALTER TABLE PriceChange

ADD CONSTRAINT FK\_UPC

FOREIGN KEY (UPC) REFERENCES Products(UPC);

CREATE SEQUENCE PriceChangeSeq START WITH 1;



-- Trigger to update price change tables

CREATE TRIGGER PriceChangeTrigger

ON Products

AFTER UPDATE

AS

BEGIN

DECLARE @OldPrice DECIMAL(6,2) = (SELECT Price FROM DELETED);

DECLARE @NewPrice DECIMAL(6,2) = (SELECT Price FROM INSERTED);

IF (@OldPrice <> @NewPrice)

INSERT INTO PriceChange(PriceChangeID, OldPrice, NewPrice, UPC, ChangeDate)

VALUES(NEXT VALUE FOR PriceChangeSeq,

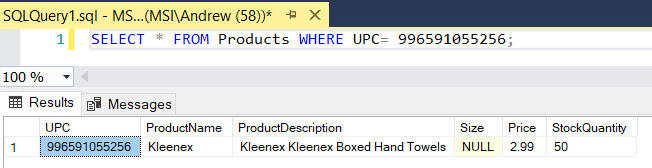
@OldPrice,

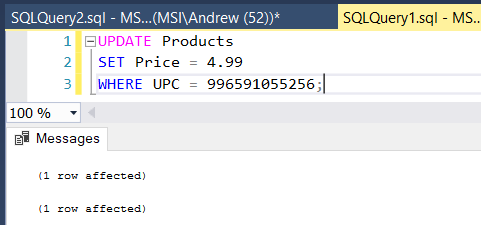
@NewPrice,

(SELECT UPC FROM INSERTED),

GETDATE());

END;

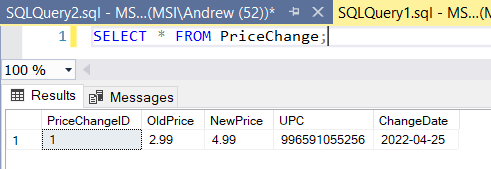


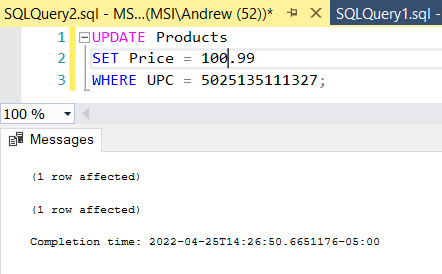


UPDATE Products

SET Price = 4.99

WHERE UPC = 996591055256;

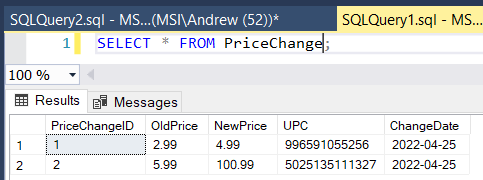


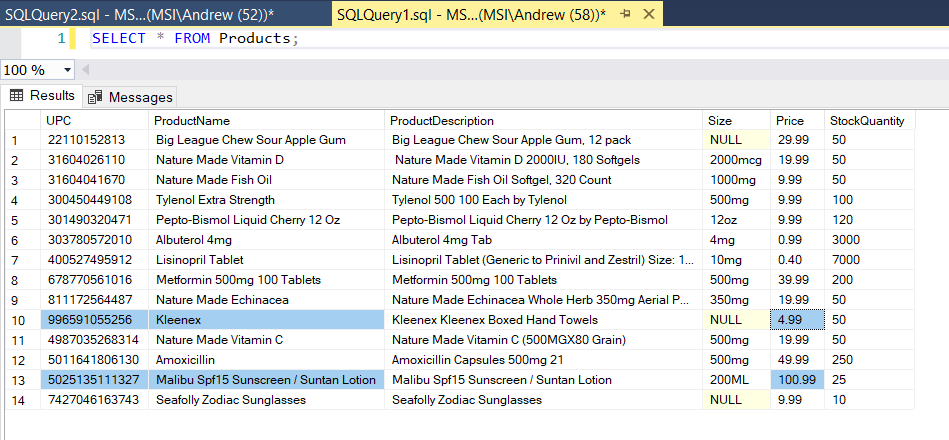


UPDATE Products

SET Price = 100.99

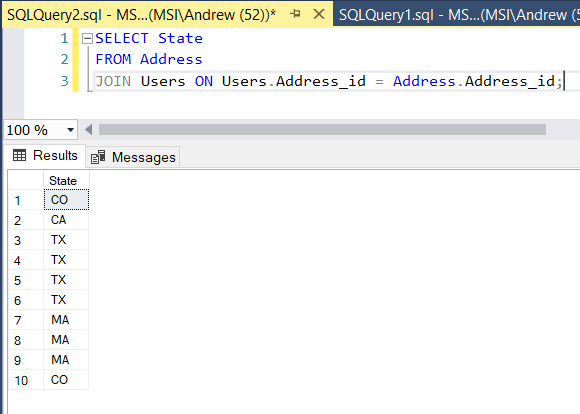
WHERE UPC = 5025135111327;





# Data Visualizations

For my first story, I wanted to see where the majority of userbase was located. Different states have different laws and requirements for pharmacies, so to ensure we are in compliance, we need a list of all the states we sell products to and we need to know where the majority of our users are located. This could also potentially help in picking a location to place a physical location for a pharmacy; being located near the majority of your user base helps cut down costs and shipping times. We can do this through a simple SELECT statement and joining the addresses (note that we don’t want ALL addresses – our address table also includes addresses of prescribers, so we need to make sure to filter the results, which in this case is easily accomplished by joining the Address\_id on the users table):

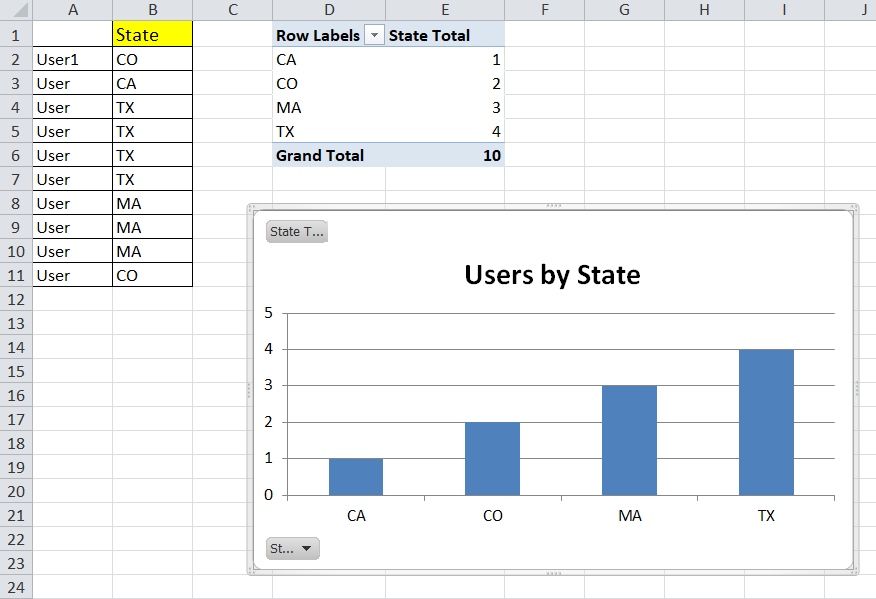


SELECT State

FROM Address

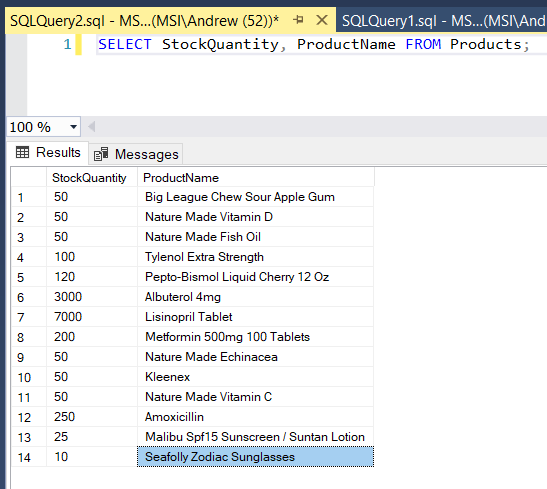
JOIN Users ON Users.Address\_id = Address.Address\_id;

Next we want to populate these results into Excel, create a simple pivot table to summarize results, and create our chart.



By looking at the graph we can see that the majority of the users of our database fall within either Texas or Massachusetts. If we were considering where to place a physical location, we would probably first consider these two locations (although obviously there are many other factors to consider as well). We also are more likely to get audited from our highest user locations so we want to ensure we have the highest levels of compliance within these two states.

The next query I came up with is to take a look at our stock levels of product. This was tricky because some products (like individual pills) we have thousands of. Others, we have only a few stock. The solution was simply: once the data was exported, take any pills and divide them by 30 (for a 30 day supply). The result was this:



We can see that the majority of our products we keep around 50 in stock. We have a few products where we keep more and a few products where we keep less. We may need to adjust stock levels, based on how many we sell, so I felt like checking the stock of products only makes sense. By looking at the visual representation of the stocks of our product, we can fine tune and adjust our stock levels based on what we think our demand is going to be.

# Summary and Reflection

Finally done! I felt like the project this week was much easier than previous weeks. Updated the ERD to reflect the new PriceChange table, create a new trigger to capture price changes, do some queries and then create some visualizations for them. This week was much easier than prior weeks (and probably by design since we have a final coming up!).

One note from last week: I purposefully did not do any checking on the state section of the address. Why? State addresses usually are only 2 characters, yet I purposefully allowed for up to 5 characters. In some rare cases the state section can have up to 5 characters, some of which can be numbers. How? Through APOs! When I was in Afghanistan, we had a temporary APO state that we used for mail and the state location was APOA5. By checking for, and removing numbers from our state inputs, we disallow this type of input.

Glad to be done, however, as a new programmer, I definitely learned a LOT by completing the project! I feel like I can safely use SQL in the workplace and I know how to create, design, and retrieve data with SQL now.