Tini Grosso

cgrosso@bu.edu

Abstract

Reusable transactions and procedures, a history table, and questions and queries for SoapSafari hand-made soap business

Soapsafari inventory control project

Database Design Document—MET CS669 Iteration 5

Contents

[Project Direction Overview 1](#_Toc53651683)

[Use Cases and Corresponding Structural Database Rules 3](#_Toc53651684)

[Use Case 1: Adding a newly-purchased ingredient to the inventory 3](#_Toc53651685)

[Structural Database Rules for Use Case 1: 4](#_Toc53651686)

[Use Case 2: Making a recipe 5](#_Toc53651687)

[Structural Database Rules for Use Case 2: 6](#_Toc53651688)

[Use Case 3: Logging a trip to a vendor 8](#_Toc53651689)

[Use Case 4: Soap is purchased by unidentified people 9](#_Toc53651690)

[Structural Database Rules for Use Case 4: 9](#_Toc53651691)

[Use Case 5: Soap is ordered by a new customer 10](#_Toc53651692)

[Structural Database Rules for Use Case 5: 11](#_Toc53651693)

[Use Case 6: Inventory is Purchased from a New Vendor 12](#_Toc53651694)

[Structural Database Rule for Use Case 6: 12](#_Toc53651695)

[Use Case 7: User/Business Owner Withdraws Items for Personal Use 13](#_Toc53651696)

[Structural Database Rule for Use Case 7: 13](#_Toc53651697)

[Adding Specialization/Generalization to the SoapSafari DB 14](#_Toc53651698)

[Breaking Apart the Ingredient Entity Specialties: 14](#_Toc53651699)

[Breaking Apart the Vendor Entity Specialties: 15](#_Toc53651700)

[Structural Database Rules for Specialization/Generalization 16](#_Toc53651701)

[SoapSafari Bridging Entities (defining more Structural DB Rules) 17](#_Toc53651702)

[RECIPE\_COMPONENT: 17](#_Toc53651703)

[LINE\_ITEM: 18](#_Toc53651704)

[Adding a History Table 19](#_Toc53651705)

[Structural Database Rules, Summarized: 20](#_Toc53651706)

[Normalization 22](#_Toc53651707)

[SoapSafari Attributes, Summarized, Ordered by Table 23](#_Toc53651708)

[SoapSafari Relationship Classification and Associative Mapping 28](#_Toc53651709)

[Conceptual Entity Relationship Diagram (ERD) 31](#_Toc53651710)

[DBMS Physical EERD 32](#_Toc53651711)

[SoapSafari Indexing 33](#_Toc53651712)

[Primary Keys 33](#_Toc53651713)

[Foreign Keys 33](#_Toc53651714)

[Query-Driven Indexes 35](#_Toc53651715)

[“Order of Operations” for Running Procedures and Scripts 37](#_Toc53651716)

[SoapSafari SQL Transactional Stored Procedures: 38](#_Toc53651717)

[SS2AddIngredient (Addresses Use Case 1): 38](#_Toc53651718)

[SS2MakeRecipe (Addresses Use Case 2) (Farina, 2019) (Microsoft, 2020): 40](#_Toc53651719)

[SS2AddPersonalUSE (Addresses Use Case 7) 44](#_Toc53651720)

[Triggers 47](#_Toc53651721)

[Negative quantities in Ingredients: 47](#_Toc53651722)

[Price History Trigger: 49](#_Toc53651723)

[Questions and Queries: 52](#_Toc53651724)

[Works Cited 56](#_Toc53651725)

# Project Direction Overview

For this database project, I developed an inventory control system for my handmade-soap business, SoapSafari.

The database stores an inventory of all ingredients and supplies necessary to make soap. The ingredients that are used in the actual products include fixed, fragrance and essential oils; botanicals; colorants; distilled water and sodium hydroxide. The supplies include such items as plastic wrap, paper towels, labels, shrink wrap and disposable gloves. Other items (“durable goods”) necessary for soapmaking include soap molds, stick blenders, safety goggles, plastic pails, a digital kitchen scale and glass thermometers, which are purchased infrequently but do eventually wear out or break over time.

Please allow me, for two paragraphs, to digress into some history, to help you understand why I am chose this for a project: I started this business unwittingly in March of 2007, when I was homeschooling one of my kids for middle school, teaching a chemistry class at a homeschool coop, and in need of an exothermic chemical reaction as a lab demonstration, for which I chose to make some soap with crude directions. After it had cured, I put an ugly bar of it in our shower, and even my husband raved at how nice it made his skin feel! We both wanted to know WHY, and upon research, I learned that “saponification” (the chemical reaction between oils and sodium hydroxide that produces soap) ALSO produces a high percentage of the amazing humectant *glycerin*, which draws moisture to your skin, and which commercial soap-makers extract from their products and sell to the cosmetics industry (Athena, 2012), thus causing “regular” soap to make your skin feel dry and tight. (Commercial “glycerin soap” is a misnomer, by the way—some glycerin is added back into the soap so manufacturers can call it that, but it is only translucent because there is a high percentage of alcohol in it, which also dries out your skin. (Ida, 2014))

I was hooked. I went crazy with ingredients, new recipes, experiments, and literally hundreds of bars of soap curing all over the house, scenting the air with intoxicating fragrances. I also sold it at craft fairs for a few years. (Since taking an outside job in 2015 and going back to school in 2018, though, I have let my soap business take an extreme backseat to my current busy life.)

I would love to resurrect this business, if for no other reason than that handmade soap is wonderful, it feels and smells so good, and I still want to share it with others! The reason this database will be useful and important is that my previous record-keeping had become painful enough that fighting with broken spread sheets and scribbled notes on lined paper was a serious impediment to continuing with my business at all.

This database will eventually be able to provide all tax information for the current tax year; for now, we will gather some inventory data in a query as a demo. The numbers that will be trivial to produce will save many days of tedious work. The database may also someday be connected to a Java application program. The database will do such things as:

1. Keep separate inventories of entities of different ingredients, supplies, and durable goods; these will be simple to modify
2. Keep a history of price changes of the products I buy.
3. Keep a list of vendors, also simple to modify
4. Keep an inventory of ready-to-sell products
5. For each of the entities in 1), keep name, purchase date, expiration date (if applicable), price, units purchased, units available at start of year, current units available, vendor and low inventory threshold, for example.
6. Keep sales receipts.
7. Keep track of finished soap batches and their quantities (the number of bars of soap from a particular batch) extracted for “personal use.”
8. Have a procedure to be able to read the most common recipes that I make and decrement inventory for those ingredients.
9. Notify me that inventory is low on an ingredient, so I can order more.
10. Provide me with Cost of Goods Sold for schedule C (or at least the values for the line item numbers 9 and 35-42 for TurboTax, or whatever line numbers they are this year) at the end of the year.

# Use Cases and Corresponding Structural Database Rules

Here are some use cases for the SoapSafari database:

## Use Case 1: Adding a newly-purchased ingredient to the inventory

1. The user logs into the application.
2. The user has purchased an ingredient from a vendor and directs the application to add this purchase to inventory.
3. Inventory is updated with the specific ingredient and its relevant information.
4. User logs out of the application.

Necessary fields in a table for the above Use Case are listed below (See also Breaking Apart the Inventory Entity Specialties, under Adding Specialization/Generalization to the SoapSafari DB):

|  |  |  |
| --- | --- | --- |
| **Inventory Field** | **What it Stores** | **Why it’s Needed** |
| Inv\_ID | Sythetic key for this table | The same item may be newly purchased while there is still older inventory in stock. These items, although the same thing, will have different purchase dates, expiration dates, and possibly different unit prices. |
| Inv\_Type | Specialization discriminator:  I = Ingredient  S = Supply  D = Durable\_Good | To identify the correct specialization for the Inventory supertype |
| Item\_Name | The name of the inventory instance | To identify the inventory instance with a human-readable name |
| Description | Description of ingredient | Some ingredients, particularly fragrance oils, may require further elaboration about their qualities |
| PurchaseDate | The date the purchase of this oil was made | To timestamp the purchase for tax reasons |
| Exp\_Date | The date after which the product must be discarded | To ensure inventory complies to safety rules |
| Cost | The cost of the product | To know what the purchase price of the instance was |
| Units\_Purchased | The number of units purchased | To be able to calculate the cost per unit of the product |
| Unit\_Name | The kind of unit (ounces, sps.., etc) | To be able to accurately keep track of different kinds of inventory, depending on how its measured |
| Yr\_Start\_Units | How much product was in inventory to begin with on Jan 1 of this year or thereafter | The first number necessary to determine how much was used by the end of the year |
| Current\_Units | The current number of units of this oil in inventory | The second number necessary to determine how much was used by the end of the year; also, to be able to maintain a current inventory |
| Vendor\_ID | PK of vendor from whom this product was purchased | To relate this table to the Vendor table |
| Low\_Inv\_Thresh | An amount below which a notification is sent | To alert user to buy more inventory of this item |

## Structural Database Rules for Use Case 1:

For this use case, the user is interacting with the database via an app, through which the Inventory table will be updated with the relevant ingredient purchase information. Inventory consists of mandatory Ingredients (such as fixed oils and essential oils), Supplies (consumables, such as paper towels and plastic wrap), and Durable Goods (such as stick blenders and molds), but in this use case, only Ingredient needs updating. Inventory will also relate to a Vendor table (see Use Case 3) because every item in Inventory is purchased from a vendor.

I see two entities in this use case: Inventory and Vendor. Inventory, as just mentioned, is a generalization for 3 specializations: Ingredient (I), Supply (S) and Durable\_Good (D). (Ingredient contains an attribute that specifies the Ingredient Type, which is Fixed\_Oil (X), Fragrance\_Oil (F), Essential\_Oil (E), Botanical (B), Colorant (C), Liquid (L), or Hydroxide (H).)

Here is the Structural Database Rule I see:

1. An inventory item may be purchased from one vendor (optional, singular);

A vendor may sell many inventory items (optional, plural).

## Use Case 2: Making a recipe

1. The user logs into the application.
2. The user chooses to make a particular soap recipe, which would specify certain ingredients (fixed oils, liquid and hydroxide) and allow the user to choose other optional ingredients (essential oils, fragrance oils, botanicals and colorants).
3. The user chooses a name for this batch (usually involving the kinds of scents chosen, since each recipe can and probably will be customized).
4. The app first checks each ingredient in the recipe to determine if there is enough on-hand to make the recipe; if an ingredient is insufficient, an error message is displayed.
5. If all ingredients are on-hand, the app queries the database to subtract those same items from inventory.
6. If an ingredient in inventory dips below a certain level, a reminder-to-reorder message is displayed.
7. A batch id, recipe id, batch name, production date, number of bars of soap made, number of bars remaining (same as starting, initially), and cost of the batch are added to a batch table.
8. The user closes the application.
9. The user will then go make the recipe of soap.

Necessary fields in a table for the above Use Case that are not already listed in Use Case 1 are listed below:

|  |  |  |
| --- | --- | --- |
| **Recipe Field** | **What it Stores** | **Why it’s Needed** |
| Recipe\_ID | Synthetic Key for this table | Recipe is uniquely identified for inventory |
| Recipe\_Name | Name of the soap recipe | For a human to easily remember which batch of soap was made |

|  |  |  |
| --- | --- | --- |
| **Batch Field** | **What it Stores** | **Why it’s Needed** |
| Batch\_ID | Synthetic Key for this table | Needs a unique, nonnull identifier |
| Recipe\_ID | PK of the recipe from which this batch was made | To relate the Batch to the individual ingredients in it, for consumer safety and to calculate the cost |
| Prod\_Date | The date the soap was made | After a specified period of time, it should be used or discounted |
| Batch\_Name\_ID | PK of the batch name table as FK here | To avoid redundancy in the batch table, since different batches of soap could feasibly have the same name |
| Cost\_Of\_Batch | The total cost of the ingredients in the whole batch of soap | To determine the cost of an individual bar of soap |
| Num\_Bars\_Start | The number of bars of soap produced in this batch | To determine how many bars this batch made |
| Num\_Bars\_Left | The number of bars left to use or sell of this batch (initially, the same as Num\_Bars\_Start | For inventory and tax purposes |

|  |  |  |
| --- | --- | --- |
| **Batch\_Name Field** | **What it Stores** | **Why it’s Needed** |
| Batch\_Name\_ID | Synthetic Key for this table | Since different batches could have the same name, the name is uniquely identified. |
| Soap\_Name | Name of the batch of soap that was made | For a human to easily see the name of the batch of soap |

## Structural Database Rules for Use Case 2:

For this use case, as in the first, the user is interacting with the database via an app, which will interact with a recipe. The recipe relates to the ingredients (different ones in the Ingredient table), as well as with the batch table. Because Recipe and Ingredient have a M:N relationship, they will interrelate via a bridge entity, Recipe Component (see SoapSafari Bridging Entities, later in this document).

Triggers will execute if certain ingredients drop below a defined low-level.

The cost of the batch will be determined with the assistance of the app to solicit user input for customization, and procedural code that will calculate the cost of the ingredients in the recipe plus the cost of the add-ons.

A recipe can be made many times, and therefore can produce many batches of soap, but in order for it to be soap it must be the result of at least one fixed oil, a liquid and a hydroxide compound. The finished product will be a batch of soap that is logged by the fields shown above. It is possible that something goes terribly wrong with the soapmaking and the whole product is thrown away, but if a batch of soap is made, it is the result of one recipe.

I see three new entities in this use case: Recipe, Batch and Batch\_Name; Recipe and Batch share a 1:M relationship, as do Batch and Batch\_Name.

Here are the Structural Database Rules I see:

1. A recipe may produce many batches of soap (optional, plural);

A batch of soap is the result of one recipe (mandatory, singular).

1. A batch of soap must have one name (mandatory, singular);

The name of a batch of soap may apply to more than one batch (optional, plural).

## Use Case 3: Logging a trip to a vendor

1. The user logs into the application.
2. The user chooses to log that a round-trip was made to a local vendor.
3. User closes application.

Necessary fields in a table for the above Use Case are listed below. (See also Breaking Apart the Vendor Entity Specialties, under Adding Specialization/Generalization to the SoapSafari DB):

|  |  |  |
| --- | --- | --- |
| **Vendor Field** | **What it Stores** | **Why it’s Needed** |
| Vendor\_ID | Unique nonnull value for this row | To relate inventory items to their vendor |
| Addr\_ID | PK of Address table | To be able to know the location of the vendor for driving or shipping |
| Is\_Local | A “boolean” discriminator | Indicates whether the vendor instance has an address to which the user can drive |
| Is\_Remote | A “boolean” discriminator | Indicates whether this vender instance is one that can be used for online ordering and shipping |
| Vendor\_Name | Name of Vendor | Identifies the Vendor so a human can recognize the name. |
| Mileage | Mileage from my home to the Vendor’s address | To know how many miles it is from the user’s home to the Vendor’s address. |
| Trip\_Tally | The number of round-trips made to the Vendor’s location this year | To be able to tally up the correct number of miles driven for the year to add to Schedule C at the end of the year. |
| Phone\_Num | The phone number of the vendor | To be able to call the vendor, if necessary |
| URL | The web address of the vendor | To be able to look up the URL of the vendor when wanting to place a new order |
| Email | The email address of the vendor | To be able to contact someone at the vendor’s business |

This use case requires no additional entities nor Structural Business Rules than the ones enumerated in Use Case 1.

## Use Case 4: Soap is purchased by unidentified people

1. The user logs into the application.
2. The application is used to record the purchase of several bars of soap to Anonymous Customer.
3. The user closes the application.

|  |  |  |
| --- | --- | --- |
| **Sales\_Receipt Field** | **What it Stores** | **Why it’s needed** |
| Receipt\_ID | Unique, nonnull value for this receipt | To be able to associate a customer with a purchase |
| Cust\_ID | The ID of the Batch being sold | To subtract the correct product from the Batch table |
| Sales\_Date | The number of bars sold | To subtract the correct amount of product from the Batch table, and to calculate the total price |

## Structural Database Rules for Use Case 4:

For this use case, the user is interacting with the database via an app.

A sales receipt should, in theory, keep track of how many bars of which batch of soap has been sold, and for how much. However, Sales\_Receipt and Batch have a M:N relationship (soap from a particular batch can go onto many sales receipts, and many sales receipts can contain soap from many batches). Therefore, a bridge entity, Line\_Item, will be addressed under SoapSafari Bridging Enties, later in this document.

Because a bridging entity is involved, this use case just pertains to the relationship between Sales\_Receipt and Customer, which is fairly simple. The Sales\_Receipt will contain fields for a receipt ID, a customer ID, and the sales date.

Many bars of soap (if not most) are sold to anonymous people (or are consumed for personal use--see Use Case 7). Therefore, we will have a “dummy entity” for our anonymous soap consumer, and require that each sales receipt be associated with a customer.

There are two new entities in this use case: Sales\_Receipt and Customer (see Use Case 5 for more detail about Customer).

Here is the Structural Database Rule I see:

4) A customer may be associated with many sales receipts (optional, plural);

A sales receipt must be associated with one customer (mandatory, singular).

## Use Case 5: Soap is ordered by a new customer

1. The user logs into the app.
2. The user enters new customer info into the customer table.
3. The user enters sales information into a sales receipt.
4. The user records in the sales receipt table the Cust\_ID of the customer who made the purchase.
5. User closes the app.

For this use case, a customer’s address will be logged into the Address table, and the customer’s email and phone info will be logged into the Customer table, if the Customer wants to offer these pieces of information. A customer will have, optionally, one address, phone number and email address. An address, however, can be associated with multiple customers.

An address may associate with one zip-code, if the customer gives it to us; the zip-code will define only one city and one state.

Here are additional fields needed in the additional tables for this use case:

|  |  |  |
| --- | --- | --- |
| **Customer Field** | **What it Stores** | **Why it’s needed** |
| Cust\_ID | Synthetic key for customer table | To uniquely identify a customer instance |
| Addr\_ID | PK to Address table, as FK in this table | To be able to unite a customer with the customer’s address |
| Cust\_FName | Customer’s First Name | For a human to be able to identify a customer by first name |
| Cust\_LName | Customer’s Last Name | For a human to be able to identify a customer by last name |
| Phone | Customer’s Phone Number | To be able to contact a customer by phone or text |
| Email | Customer’s Email Address | To be able to contact the customer via email, if necessary |

|  |  |  |
| --- | --- | --- |
| **Address Field** | **What it Stores** | **Why it’s needed** |
| Addr\_ID | Synthetic Key for Address table | Uniquely identify an Address instance |
| Addr\_Line1 | First line of an address | A necessary piece of an address, if we have one |
| Addr\_Line2 | Second line of an address | Some customers and businesses have a second address line |
| Zip\_ID | PK of Zip table as FK in this table | To relate Addresses to ZipCodes |

|  |  |  |
| --- | --- | --- |
| **Zip Field** | **What it Stores** | **Why it’s needed** |
| Zip\_ID | Synthetic Key to Zip table | Uniquely identify a ZipCode instance |
| Zip\_Code | A Zip Code | Uniquely determines a specific city and state |
| City\_ID | PK of City table as FK in this table | To relate a city name to this zip code |
| State\_ID | PK of State table as FK in this table | To relate a state to a zip code |

|  |  |  |
| --- | --- | --- |
| **City Field** | **What it Stores** | **Why it’s needed** |
| City\_ID | Synthetic Key to City table | Uniquely identifies the name of a city |
| City\_Name | The name of the city | So a human can read the name of the city |

|  |  |  |
| --- | --- | --- |
| **State Field** | **What it Stores** | **Why it’s needed** |
| State\_ID | Synthetic Key to State table | Uniquely identifies the state code of a U.S. state |
| State\_Code | The 2-letter code of the U.S. state | A U.S. citizen (and possibly others) will be able to see and recognize a state with this 2-letter code |

Here are the entities that are added to the ERD as a result of this use case: Customer, Address, Zip and City. Sales\_Receipt is already in our ERD, but now relates to Customer.

Here are the Structural Database Rules that go with this use case:

## Structural Database Rules for Use Case 5:

5) A customer may have one address (optional, singular);

An address may be associated with many customers (optional, plural).

1. An address may have only one zip code associated with it (optional, singular);

A zip code may have many addresses associated with it (optional, plural).

1. A zip code must have one city associated with it (mandatory, singular);

A city must be associated with at least one zip code (mandatory, plural).

1. A zip code must have one state associated with it (mandatory, singular);

A state must be associated with at least one zip code (mandatory, plural).

## Use Case 6: Inventory is Purchased from a New Vendor

1. The user logs into the app.
2. The user enters new vendor info into the vendor table.
3. The user enters the new inventory item purchased in the inventory table.
4. The user records in the inventory table the Vendor\_ID associated with this item.
5. User closes the app.

(See use case 3 for Vendor table fields.)

For this use case, a vendor’s id, address id, name, phone number, specialization discriminator (local or remote), URL, and email address will be logged into the Vendor table. Vendor and Customer both relate to the Address table.

A vendor has one address, but an address might be associated with more than one vendor.

There are no additional entities needed for this use case, but here is a Structural Database Rule that goes with this use case:

## Structural Database Rule for Use Case 6:

1. A vendor must have one address (mandatory, singular);

An address may have more than one vendor associated with it (optional, plural).

## Use Case 7: User/Business Owner Withdraws Items for Personal Use

1. The user logs into the app.
2. The user elects to designate some product for personal use, and logs these items into the Personal\_Use table and withdraws them from the Batch.
3. The user closes the app.

For this use case, the important thing is to be able to withdraw the quantity used and keep track of the actual cost. No receipt is necessary. Thus, the Personal\_Use table will relate to the Batch table directly. The business owner may withdraw soap from a particular batch many times (i.e., more than one Pers\_Use\_ID can relate to a batch of soap), but each withdrawal associates with only one soap batch. Here is the table we will need and the attributes:

|  |  |  |
| --- | --- | --- |
| **Personal\_Use Field** | **What it Stores** | **Why it’s Needed** |
| Pers\_Use\_ID | Synthetic Key for Personal\_Use table | Unique, nonnull value for this row |
| Batch\_ID | PK of Batch table as FK here | To relate the withdrawal of a personal use item to the batch from which it was withdrawn |
| Quantity | The number of bars of this product | To be able to calculate the cost |

The new entity from this use case is Personal\_Use. Here is the new Structural Business Rule associated with this use case:

## Structural Database Rule for Use Case 7:

1. A personal use withdrawal indicates one batch of soap (mandatory, singular);

The same batch of soap may be withdrawn from several times for personal use (optional, plural).

# Adding Specialization/Generalization to the SoapSafari DB

There are two places in my database in which I envision specialization/generalization to be relevant: Inventory and Vendor.

Inventory is comprised of three entities: Ingredient, Supply and Durable\_Good. These three entities share some common attributes but are different and are used differently. The specialized Inventory entity exhibits a total completeness and disjointness (mandatory or). The Inventory attributes listed in Use Case 1 called “Units\_Purchased” and “Unit\_Name” will be removed from the supertype, combined, and placed into each of the 3 subtypes (Ingredient, Supply and Durable\_Good) as “Oz\_Purchased,” “Nbr\_Purchased” and “Nbr\_Purchased,” respectively.

Ingredient is what actually comprises soap. There are seven different types of ingredients (fixed oils, fragrance oils, essential oils, botanicals, colorants, liquids and hydroxides), but they share the same attributes, so they do not need specialization—just an Ingredient\_Type attribute.

Supply consists of other consumables but they are not ingredients in soap and do not expire. Specifically, Supply contains plastic wrap, paper towels, labels, packaging and disposable gloves.

Durable\_Good consists of items that may wear out or break over time but normally last several years and are used over and over. These items are soap molds, stick blenders, safety goggles, plastic pails and glass thermometers.

Vendor will maintain a list of (big surprise here) vendors from whom Inventory is purchased. These vendors can be online, local, or both online and local; but there are no additional vendors for whom those choices do not apply, so Vendor exhibits total completeness (mandatory and). For example, I have placed online orders with Jedwards International in Braintree, MA, for shipment to my home in NH; but once, when I needed a large amount of cocoa butter, I decided to drive there from my home to save the shipping charges.

Let’s look back at our Field lists for both Inventory and for Vendor and decide how to distribute them into their specializations:

## Breaking Apart the Ingredient Entity Specialties:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Inventory Field** | **Stays in Supertype** | **Moves or Added to Ingredient Subtype** | **Moves or Added to Supply Subtype** | **Moves or Added to Durable**  **\_Good**  **Subtype** | **Comments** |
| Inv\_ID | Checkbox Checked | Checkbox Checked | Checkbox Checked | Checkbox Checked |  |
| Vendor\_ID | Checkbox Checked |  |  |  |  |
| Inv\_Type | Checkbox Checked |  |  |  |  |
| Item\_Name |  | Checkbox Checked | Checkbox Checked | Checkbox Checked | Will become Ingred\_Name, Supply\_Name, or  Dur\_Good  \_Name |
| Description |  | Checkbox Checked |  |  |  |
| Ingred\_Type |  | Checkbox Checked |  |  |  |
| Purchase\_Date | Checkbox Checked |  |  |  |  |
| Exp\_Date |  | Checkbox Checked |  |  |  |
| Cost | Checkbox Checked |  |  |  |  |
| Units\_  Purchased |  | Checkbox Checkedrenamed  Oz\_Purchased | Checkbox Checkedrenamed Nbr\_Purchased | Checkbox Checked  renamed Nbr\_Purchased |  |
| Unit\_Name | removed |  |  |  |  |
| Yr\_Start\_Units |  | Checkbox Checked | Checkbox Checked |  |  |
| Current\_Units |  | Checkbox Checked | Checkbox Checked |  |  |
| Low\_Inv\_  Thresh |  | Checkbox Checked | Checkbox Checked |  |  |

## Breaking Apart the Vendor Entity Specialties:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Vendor Field** | **Stays in SuperType** | **Moves or Added to Local\_Vendor Subtype** | **Moves or Added to Remote\_Vendor Subtype** | **Comments** |
| Vendor\_ID | Checkbox Checked | Checkbox Checked | Checkbox Checked |  |
| Addr\_ID | Checkbox Checked |  |  |  |
| Is\_Local | Checkbox Checked |  |  |  |
| Is\_Remote | Checkbox Checked |  |  |  |
| Vendor\_Name | Checkbox Checked |  |  |  |
| Mileage |  | Checkbox Checked |  |  |
| Trip\_Tally |  | Checkbox Checked |  |  |
| Phone\_Num | Checkbox Checked |  |  |  |
| URL |  |  | Checkbox Checked |  |
| Email | Checkbox Checked |  |  |  |

The above new specialization/generalization relationships will not impact the use cases, but will add new entities and therefore new Structural Business Rules:

## Structural Database Rules for Specialization/Generalization

1. An inventory item is an ingredient, a supply or a durable good.
2. A vendor is local, remote or both.

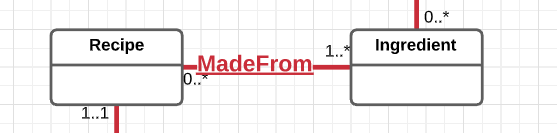
# SoapSafari Bridging Entities (defining more Structural DB Rules)

Two of our Structure Database Rules define M:N relationships, which cannot be modeled as such in our database. They need to be broken apart, each with a bridging entity. Here are the rules, and here are their replacements with bridging entities:

## RECIPE\_COMPONENT:

PREVIOUS: A recipe must be made from several different ingredients (mandatory, plural);

An ingredient may be in many different recipes (optional, plural).



The new entity here will be called Recipe\_Component, and will contain Foreign Keys to both Recipe and Ingredient.

Here are the attributes in the Recipe\_Component table:

|  |  |  |
| --- | --- | --- |
| **Recipe\_Component Field** | **What it Stores** | **Why it’s Needed** |
| Recipe\_Comp\_ID | Synthetic Key for the Recipe\_Component table | Unique nonnull PK |
| Recipe\_ID | PK of Recipe table used here as FK | To bridge the Recipe table to the Ingredient table |
| Inventory\_ID | PK of Inventory/Ingredient table used here as FK | To bridge the Ingredient table to the Recipe table |
| Quantity | The number of units of this ingredient in the recipe | To know how to make the recipe and how much to withdraw from inventory |

The NEW business rules generated from this reification are:

1. A recipe must have at least 1 (3, actually) recipe component (mandatory, plural);

A recipe component must have a recipe (mandatory, singular).

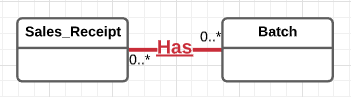
1. An ingredient may be in many recipe components (optional, plural);

A recipe component must have an ingredient (mandatory, singular).

## LINE\_ITEM:

PREVIOUS: A bar of soap from a particular batch can be in many sales receipts (optional, plural);

A sales receipt can have soaps from many different batches on it (optional, plural).



The new entity here will be called Line\_Item, and will contain Foreign Keys to both Sales\_Receipt and Batch.

Here are the attributes in the Line\_Item table:

|  |  |  |
| --- | --- | --- |
| **Line\_Item Field** | **What it Stores** | **Why it’s Needed** |
| Line\_Item\_ID | Synthetic Key for the Line\_Item table | Unique nonnull PK |
| Batch\_ID | PK of Batch table used here as FK | To bridge the Batch table to the Receipt table |
| Receipt\_ID | PK of Receipt table used here as FK | To bridge the Receipt table to the Batch table |
| Quantity | The number of units (bars of soap) of this batch on the receipt | To know how much to charge for the soap, and to be able to subtract this many bars from this batch’s quantity |
| Price\_per\_bar | The price per bar of the soap on this line item | To know how much to charge for the soap and for tax reasons |

NEW business rules generated from this reification are:

1. A batch may be on at least 1 line item (optional, plural);

A line item must have a batch that it references (mandatory, singular).

1. A sales receipt must have at least 1 line item (mandatory, plural);

A line item must be in a sales receipt (mandatory, singular).

# Adding a History Table

It was determined that a history of price changes would be beneficial for SoapSafari. Markets fluctuate, and prices, especially on 50-lb. pails of fixed oil, fluctuate constantly, and it would be nice to have a record of prices so that I can see if a product I am about to purchase is a good value or not.

Here are the new attributes that will be in the Price\_History table:

|  |  |  |
| --- | --- | --- |
| **Price\_History Field** | **What it Stores** | **Why it’s Needed** |
| Price\_Hist\_ID | Synthetic Key for the History table | Unique nonnull PK |
| Inv\_ID | PK of Inventory table used here as FK | To relate the Inventory table to the Price\_History table |
| Purchase\_Date | The date the purchase was made | To make the change of the price relevant |
| Old\_Cost | The previous cost of this product, in the same size from the same vendor | To track the change |
| New\_Cost | The new cost of this product, in the same size from the same vendor as before | To track the change |

Here are the new Structural Database Rules that will go with the new association between Inventory and Price\_History:

1. Items in Inventory may have many price changes (optional, plural);

Each price change is associated with one purchase of inventory (singular, mandatory).

(SEE ALSO SECTION ON TRIGGERS)

# Structural Database Rules, Summarized:

1. An inventory item may be purchased from one vendor (optional, singular);

A vendor may sell many inventory items (optional, plural).

1. A recipe may produce many batches of soap (optional, plural);

A batch of soap is the result of one recipe (mandatory, singular).

1. A batch of soap must have one name (mandatory, singular);

The name of a batch of soap may apply to more than one batch (optional, plural).

1. A customer may be associated with many sales receipts (optional, plural);

A sales receipt must be associated with one customer (mandatory, singular).

1. A customer may have one address (optional, singular);

An address may be associated with many customers (optional, plural).

1. An address may have only one zip code associated with it (optional, singular);

A zip code may have many addresses associated with it (optional, plural).

1. A zip code must have one city associated with it (mandatory, singular);

A city must be associated with at least one zip code (mandatory, plural).

1. A zip code must have one state associated with it (mandatory, singular);

A state must be associated with at least one zip code (mandatory, plural).

1. A vendor must have one address (mandatory, singular);

An address may have more than one vendor associated with it (optional, plural).

1. A personal use withdrawal indicates one batch of soap (mandatory, singular);

The same batch of soap may be withdrawn from several times for personal use (optional, plural).

1. An inventory item is an ingredient, a supply or a durable good.
2. A vendor is local, remote or both.
3. A recipe must have at least 1 (3, actually) recipe component (mandatory, plural);

A recipe component must have a recipe (mandatory, singular).

1. An ingredient may be in many recipe components (optional, plural);

A recipe component must have an ingredient (mandatory, singular).

1. A batch may be on at least 1 line item (optional, plural);

A line item must have a batch that it references (mandatory, singular).

1. A sales receipt must have at least 1 line item (mandatory, plural);

A line item must be in a sales receipt (mandatory, singular).

1. Items in Inventory may have many price changes (optional, plural);

Each price change is associated with one purchase of inventory (singular, mandatory).

# Normalization

Since previous iterations of this project, three additional tables have been added for the sake of normalization. The Zip table now references a City and a State table, and Batch references Batch\_Name (for the reasons cited in the Structural Database Rules). The “Units\_Purchased” attribute was also pulled out of the Inventory table and added to each respective subtype (as Oz\_Purchased and Nbr\_Purchased), because ingredients are in ounces, and supplies and durable goods are in random numbers of units.

The street addresses were not pulled apart, although they could have been. It seemed best to me to keep the addresses together to save on complexity.

# SoapSafari Attributes, Summarized, Ordered by Table

All synthetic IDs will be of type Decimal(12)\*, which will allow more than enough room for a growing inventory list with unique IDs for a long, long time.

All phone numbers and addresses will be domestic (USA) only.

Phone numbers allow for VARCHAR(12)\*\*, which would include 2 hyphens.

Zip code allows for “zip plus 4,” plus a hyphen.

|  |  |  |  |
| --- | --- | --- | --- |
| **Table** | **Attribute** | **Datatype** | **Reasoning** |
| Inventory | Inv\_ID | DECIMAL(12) | \*Unique, nonnull PK |
| Inventory | Vendor\_ID | DECIMAL(12) | \*PK of Vendor table  as FK |
| Inventory | Inv\_Type | CHAR(1) | Specialization discriminator only needs a single CHAR |
| Inventory | Purchase\_Date | DATE | It’s a date |
| Inventory | Cost | DECIMAL(7,2) | This would allow an inventory purchase of up to $99,999.99. Possibly if I were purchasing drums of costly essential oils I could spend in the 10’s of thousands, but then I wouldn’t be in grad school studying CS! (7,2) is more than enough here. |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| Ingredient | Inv\_ID | DECIMAL(12) | \*PK, FK of supertype |
| Ingredient | Ingred\_Name | VARCHAR(255) | Human-recognizable name of Ingredient will certainly be less than 255 characters. |
| Ingredient | Ingred\_Type | CHAR(1) | Only needs a single CHAR |
| Ingredient | Description | VARCHAR(1024) | Some ingredients should have further elaboration, but 1024 is sufficient. |
| Ingredient | Exp\_Date | DATE | It’s a date. |
| Ingredient | Oz\_Purchased | DECIMAL(7,2) | As with Cost, above, I will not be purchasing more than 10,000 ounces of anything. One 50-lb. pail of a fixed oil is 800 oz, so 5 places is sufficient. I added decimal places for consistency with Yr\_Start\_Units and Current\_Units. |
| Ingredient | Yr\_Start\_Units | DECIMAL(7,2) | See Ingredient.Oz\_Purchased |
| Ingredient | Current\_Units | DECIMAL(7,2) | See Inventory.Units\_Purchased |
| Ingredient | Low\_Inv\_Thresh | DECIMAL(6,2) | Similar to reasoning in Inventory.Units Purchased, except that less space will be needed. |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| Supply | Inv\_ID | DECIMAL(12) | \*PK, FK of supertype |
| Supply | Supply\_Name | VARCHAR(255) | Human-recognizable name of Supply will certainly be less than 255 characters. |
| Supply | Nbr\_Purchased | DECIMAL(7,2) | See Ingredient.Oz\_Purchased |
| Supply | Yr\_Start\_Units | DECIMAL(7,2) | See Ingredient.Yr\_Start\_Units |
| Supply | Current\_Units | DECIMAL(7,2) | See Ingredient.Current\_Units |
| Supply | Low\_Inv\_Thresh | DECIMAL(6,2) | See Ingredient.Low\_Inv\_Thresh |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| Durable\_Good | Inv\_ID | DECIMAL(12) | \*PK, FK of supertype |
| Durable\_Good | Nbr\_Purchased | DECIMAL(5) | See Ingredient.Oz\_Purchased, except these will not be fractional |
| Durable\_Good | Dur\_Good\_Name | VARCHAR(255) | Human-recognizable name of Durable\_Good will certainly be less than 255 characters |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| Price\_History | Price\_Hist\_ID | VARCHAR(12) | \*Unique, nonnull PK |
| Price\_History | Inv\_ID | VARCHAR(12) | PK of Inventory table as FK |
| Price\_History | Purchase\_Date | DATE | It’s a date |
| Price\_History | Old\_Cost | DECIMAL(7,2) | Compatible with Cost from Inv table |
| Price\_History | New\_Cost | DECIMAL(7,2) | Compatible with Cost from Inventory table |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| Recipe | Recipe\_ID | DECIMAL(12) | \*Unique, nonnull PK |
| Recipe | Recipe\_Name | VARCHAR(255) | Sufficient for the name of a soap recipe, which would include some of the oils (e.g., ‘Avocado & Cocoa Butter Soap’) |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| Batch | Batch\_ID | DECIMAL(12) | \*Unique, nonnull PK |
| Batch | Batch\_Name\_ID | DECIMAL(12) | \*PK of Batch\_Name table as FK |
| Batch | Prod\_Date | DATE | It’s a date |
| Batch | Recipe\_ID | DECIMAL(12) | \*PK of Recipe table as FK |
| Batch | Num\_Bars\_Start | DECIMAL(5) | Normally 20-30 bars is max for SoapSafari. 5 digits sufficient. |
| Batch | Num\_Bars\_Left | DECIMAL(5) | See Num\_Bars\_Start, above. Num\_Bars\_Left will initially be Num\_Bars\_Start. |
| Batch | Cost\_Of\_Batch | DECIMAL(6,2) | Even if a LOT of really expensive essential oil added, 4 places left of decimal are sufficient for cost, which normally only needs 2. |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| Batch\_Name | Batch\_Name\_ID | DECIMAL(12) | \*Unique, nonnull PK |
| Batch\_Name | Soap\_Name | VARCHAR(255) | Sufficient for a descriptive name for this batch of soap, which would likely include some scents and oils (e.g., ‘Lavender-Spearmint Avocado Soap’) |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| Vendor | Vendor\_ID | DECIMAL(12) | \*Unique, nonnull PK |
| Vendor | Vendor\_Name | VARCHAR(255) | Will be sufficient for even a very long vendor name |
| Vendor | Address\_ID | DECIMAL(12) | \*PK of Address table as FK |
| Vendor | Is\_Local | CHAR(1) | Boolean needs only 1 CHAR |
| Vendor | Is\_Remote | CHAR(1) | Boolean needs only 1 CHAR |
| Vendor | Phone\_Num | VARCHAR(12) | \*\*domestic number |
| Vendor | Email | VARCHAR(255) | Will be sufficient for even a very long, creative email address |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| Local\_Vendor | Vendor\_ID | DECIMAL(12) | \*PK, FK of supertype |
| Local\_Vendor | Vendor\_Mileage | DECIMAL((6,2) | Allows for 9999 miles, which is more than enough to drive from NH to San Diego 3 times. |
| Local\_Vendor | Trip\_Tally | DECIMAL(4) | Allows for driving to the same Vendor 3 times every day for nearly the whole year, which will never happen. |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| Remote\_Vendor | Vendor\_ID | DECIMAL(12) | \*PK, FK of supertype |
| Remote\_Vendor | URL | VARCHAR(1024) | Sufficient for a pasted URL that consumes more than 10 lines |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| Sales\_Receipt | Sales\_Receipt\_ID | DECIMAL(12) | \*Unique, nonnull PK |
| Sales\_Receipt | Cust\_ID | DECIMAL(12) | \*PK of Customer table as FK |
| Sales\_Receipt | Sales\_Date | DATE | It’s a date |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| Customer | Cust\_ID | DECIMAL(12) | \*Unique, nonnull PK |
| Customer | Cust\_FName | VARCHAR(255) | Sufficient for a very long first name |
| Customer | Cust\_LName | VARCHAR(255) | Sufficient for a very long, hyphenated last name |
| Customer | Addr\_ID | DECIMAL(12) | \*PK of Address table as FK |
| Customer | Phone\_Num | VARCHAR(12) | \*\*domestic number |
| Customer | Email | VARCHAR(255) | See Vendor.Email |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| Address | Addr\_ID | DECIMAL(12) | \*Unique, nonnull PK |
| Address | Addr\_Line1 | VARCHAR(255) | Sufficient for a very long street address |
| Address | Addr\_Line2 | VARCHAR(255) | Sufficient for a very long 2nd line of a street address |
| Address | Zip\_ID | DECIMAL(12) | \*PK of Zip code table as FK |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| Zip | Zip\_ID | DECIMAL(12) | \*Unique, nonnull PK |
| Zip | Zip\_Code | VARCHAR(10) | “Zip plus 4” with hyphen allowed |
| Zip | City\_ID | DECIMAL(12) | \*PK of City table as FK |
| Zip | State\_ID | DECIMAL(12) | \*PK of State table as FK |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| City | City\_ID | DECIMAL(12) | \*Unique, nonnull PK |
| City | City\_Name | VARCHAR(255) | More than accommodates the longest domestic place name (Wikipedia, 2020) |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| State | State\_ID | DECIMAL(12) | \*Unique, nonnull PK |
| State | State\_Code | CHAR(2) | U.S. States only need 2 characters |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| Line\_Item | Line\_Item\_ID | DECIMAL(12) | \*Unique, nonnull PK |
| Line\_Item | Sales\_Receipt\_ID | DECIMAL(12) | \*PK of Sales\_Receipt table as FK |
| Line\_Item | Batch\_ID | DECIMAL(12) | \*PK of Batch table as FK |
| Line\_Item | Quantity | DECIMAL(5) | Allowing a purchase of up to 10,000-99,000 items is sufficient |
| Line\_Item | Price\_per\_bar | DECIMAL(5,2) | Allowing a bar of soap to cost 999.99 is more than sufficient. |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| Recipe\_Component | Recipe\_Comp\_ID | DECIMAL(12) | \*Unique, nonnull PK |
| Recipe\_Component | Recipe\_ID | DECIMAL(12) | \*PK of Recipe table as FK |
| Recipe\_Component | Inventory\_ID | DECIMAL(12) | \*PK of Ingredient table as FK |
| Recipe\_Component | Quantity | DECIMAL(7,2) | Ingredient components are in ounces and fractions of ounces; this is more than enough |
| ++++++++++++ | ++++++++++++ | ++++++++++++ | ++++++++++++ |
| Personal\_Use | Pers\_Use\_ID | DECIMAL(12) | \*Unique, nonnull PK |
| Personal\_Use | Batch\_ID | DECIMAL(12) | \*PK of Batch table as FK |
| Personal\_Use | Quantity | DECIMAL(5) | See Line\_Item.Quantity |

# SoapSafari Relationship Classification and Associative Mapping

The Structural Database Rules described above actually define the participation and plurality constraints on both sides of the entity relationships, but we will spell them out and reiterate so that they are easier to envision:

1. Inventory/Vendor have a 1:M association. An inventory item may be purchased from one vendor, and a vendor may sell many of the items in inventory.

Inventory and Vendor will each have a synthetic PK as a unique, non-null identifier.

Inventory (on the many side) will contain an FK referencing Vendor’s PK.

1. Recipe/Recipe\_Component have a 1:M association. A recipe must have a least 1 recipe component (actually there MUST be at least 3: a fixed oil, a liquid and a hydroxide), and a recipe component must have an associated recipe or it would not be the component of a recipe.

Recipe\_Component and Recipe will each have a synthetic PK as a unique, non-null identifier.

Recipe\_Component (on the many side) will contain an FK referencing Recipe’s PK.

1. Ingredient/Recipe\_Component have a 1:M association. An ingredient may be in many recipe components, and a recipe component must have a corresponding ingredient.

Recipe\_Component and Ingredient will each have a synthetic PK as a unique, non-null identifier.

Recipe\_Component (on the many side) will contain an FK referencing Ingredient’s PK.

1. Recipe/Batch have a 1:M association. A recipe may produce many finished products, and a finished product is the result of one recipe.

Recipe and Batch will each have a synthetic PK as a unique, non-null identifier.

Batch (on the many side) will contain an FK referencing Recipe’s PK.

1. Batch/Batch\_Name have a 1:M association. A batch of soap has only one name associated with it, and a name for a batch of soap may apply to more than one batch.

Batch and Batch\_Name will each have a synthetic PK as a unique, non-null identifier.

Batch (on the many side) will contain an FK referencing Batch\_Name’s PK.

1. Batch/Line\_Item have a 1:M association. A batch may be referenced on at least one line item, and a line item must have batch on it, to which it refers.

Batch and Line\_Item will each have a synthetic PK as a unique, non-null identifier.

Line\_Item (on the many side) will contain an FK referencing Batch’s PK.

1. Sales\_Receipt/Line\_Item have a 1:M association. A sales receipt must have at least one line item, and a line item must be in one sales receipt.

Sales\_Receipt and Line\_Item will each have a synthetic PK as a unique, non-null identifier.

Line\_Item (on the many side) will contain an FK referencing Sales\_Receipt’s PK.

1. Inventory is in a “mandatory-or” inheritance relationship with Ingredient, Supply or Durable\_Good.

Inventory will have a synthetic PK as a unique, non-null identifier; its 3 specialization entities will have the same PK as their supertype, via an FK constraint.

1. Vendor is in a “mandatory-and” inheritance relationship with Local\_Vendor and Remote\_Vendor.

Vendor will have a synthetic PK as a unique, non-null identifier; its 2 specialization entities will have the same PK as their supertype, via an FK constraint.

1. Customer/Sales\_Receipt have a 1:M association. A customer may have many sales receipts, and a sales receipt may be associated with one customer.

Customer and Sales\_Receipt will each have a synthetic PK as a unique, non-null identifier.

Sales\_Receipt (on the many side) will contain an FK referencing Customer’s PK.

1. Customer/Address have a 1:M association. A customer may have one address, and an address may be associated with many customers.

Customer and Address will each have a synthetic PK as a unique, non-null identifier.

Customer (on the many side) will contain an FK referencing Address’s PK.

1. Address/Zip have a 1:M association. A zip code may have many addresses, and an address may be associated with one zip code.

Zip and Address will each have a synthetic PK as a unique, non-null identifier.

Address (on the many side) will contain an FK referencing Zip’s PK.

1. Zip/City have a 1:M association. A zip code, if it exists in the DB, has one city that it corresponds to. A city must have a zip code, but because city names are not unique, a city can have many different zip codes associated with it.

Zip and City will each have a synthetic PK as a unique, non-null identifier.

Zip (on the many side) will contain an FK referencing City’s PK.

1. Zip/State have a 1:M association. A zip code, if it exists in the DB, has one state that it corresponds to. A state must have at least one zip code associated with it.

Zip and State will each have a synthetic PK as a unique, non-null identifier.

Zip (on the many side) will contain an FK referencing State’s PK.

1. Vendor/Address have a 1:M association. A vendor may have one address, and an address may be associated with many vendors.

Vendor and Address will each have a synthetic PK as a unique, non-null identifier.

Vendor (on the many side) will contain an FK referencing Address’s PK.

1. Batch/Personal\_Use have a 1:M association. A personal use item must be withdrawn from one batch, and a batch of soap may be withdrawn from for many personal use events.

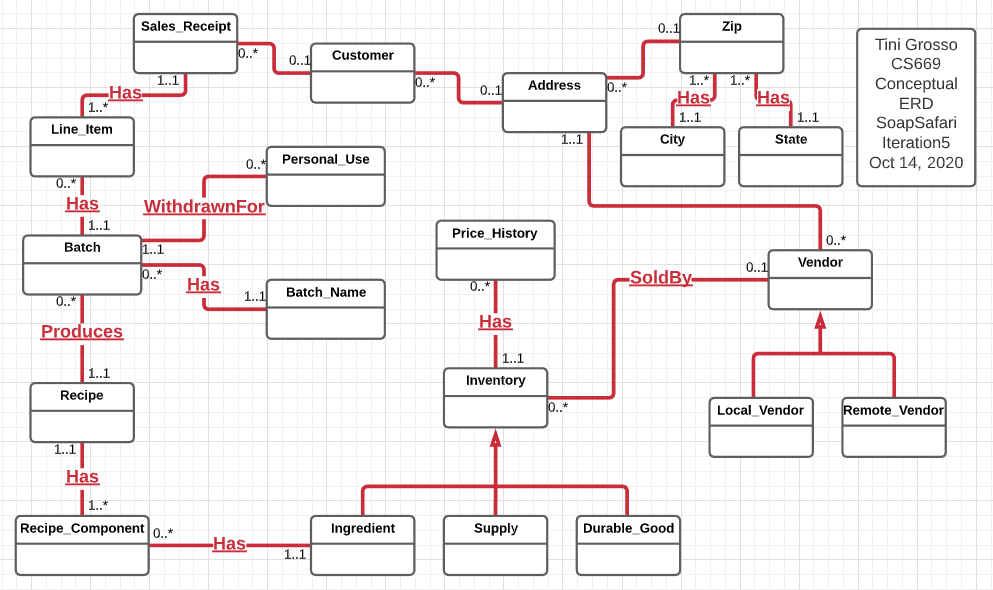
Batch and Personal\_Use will each have a synthetic PK as a unique, non-null identifier.

Personal\_Use (on the many side) will contain an FK referencing Batch’s PK.

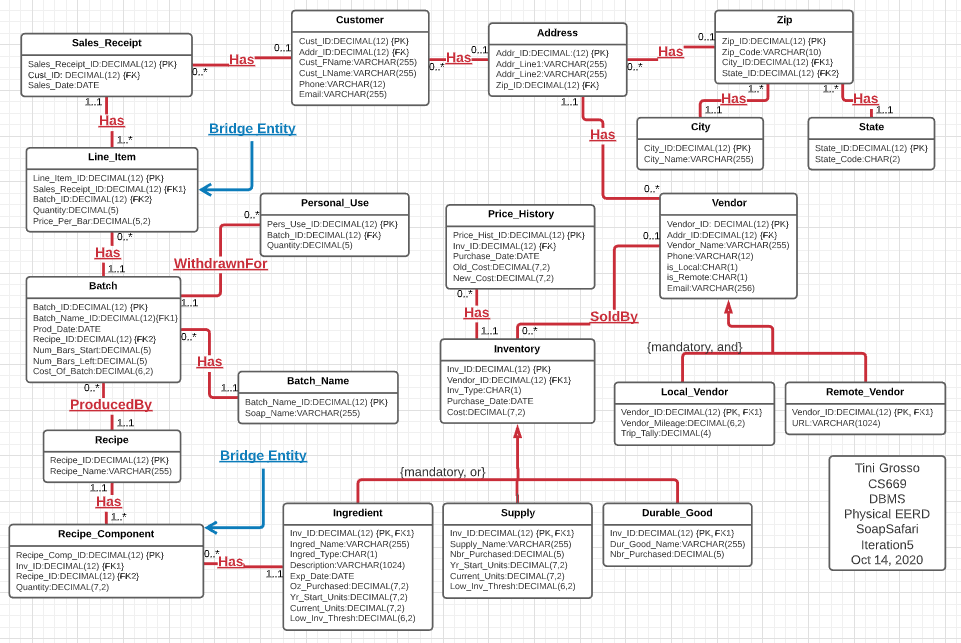
1. Price\_History/Inventory have a 1:M association. An item in inventory may have many price changes, but each price change is associated with one purchase of inventory.

Price\_History and Inventory each have a synthetic PK as a unique, non-null identifier. Price\_History (on the many side) will contain an FK referencing Inventory’s PK.

# Conceptual Entity Relationship Diagram (ERD)



# DBMS Physical EERD



# SoapSafari Indexing

## Primary Keys

Twenty entities exist in the SoapSafari EERD; here is the list of them, with their corresponding primary keys (which have implicitly-created indexes already), so they will NOT need explicit indexes:

Inventory.Inv\_ID

Ingredient.Inv\_ID

Supply.Inv\_ID

Durable\_Good.Inv\_ID

Price\_Hist\_ID

Vendor.Vendor\_ID

Local\_Vendor.Vendor\_ID

Remote\_Vendor.Vendor\_ID

Zip.Zip\_ID

City.City\_ID

State.State\_ID

Address.Addr\_ID

Customer.Cust\_ID

Sales\_Receipt.Sales\_Receipt\_ID

Line\_Item.Line\_Item\_ID

Batch.Batch\_ID

Batch\_Name.Batch\_ID

Personal\_Use.Pers\_Use\_ID

Recipe.Recipe\_ID

Recipe\_Component.Recipe\_Comp\_ID

## Foreign Keys

There are seventeen foreign key columns in the EERD. (Four of the above tables have no FK, and four have 2 FKs each.) However, five of them are also PKs that reference their supertype as an FK, so we will leave those off this list. Here is a table of the remaining fifteen:

|  |  |  |
| --- | --- | --- |
| **Column** | **Unique?** | **Description** |
| Address.Zip\_ID | Not unique | The FK in Address references a zip code. Many addresses can have the same zip code. |
| Zip.City\_ID | Not unique | The FK in Zip references a city. Many zip codes can be associated with one city name. |
| Zip.State\_ID | Not unique | The FK in Zip references a state code. Many zip codes are associated with one state code. |
| Customer.Addr\_\_ID | Not unique | The FK in Customer references an address. Many customers can have the same address. |
| Vendor.Addr\_ID | Not unique | The FK in Vendor references an address. Many vendors can have the same address. |
| Inventory.Vendor\_ID | Not unique | The FK in Inventory references a particular vendor. Many inventory items may come from the same vendor, |
| Sales\_Receipt.Cust\_ID | Not unique | The FK in Sales\_Receipt references a customer. Many sales receipts can be associated with one customer. |
| Batch.Recipe\_ID | Not unique | The FK in Batch references a recipe. A batch can be made many times from the same recipe. |
| Batch.Batch\_Name\_ID | Not unique | The FK in Batch references a batch name. A batch name can refer to many different batches. |
| Line\_Item.Sales\_Receipt\_ID | Not unique | The FK in Line\_Item references a receipt. Many line items may be on one receipt. |
| Line\_Item.Batch\_ID | Not unique | The FK in Line\_Item references a batch of soap. Many line items may refer to the same batch of soap. |
| Recipe\_Component.Inv\_ID | Not unique | The FK in Recipe\_Component references an inventory item that is an ingredient. Many recipe components may refer to the same ingredient. |
| Recipe\_Component.Recipe\_ID | Not unique | The FK in Recipe\_Component references a recipe. Many recipe components may produce the same recipe. |
| Personal\_Use.Batch\_ID | Not unique | The FK in Personal\_Use references a batch of soap. Many of the same product many be withdrawn for personal use. |
| Price\_History.Inv\_ID | Not unique | The FK in Price\_History references a change in price in Inventory. Many price changes may refer to the same inventory item. |

## Query-Driven Indexes

There are several places where I can see a need for an index:

1. I have frequently been asked by customers such things as: do I have a recipe that uses Patchouli oil, or do I have a soap mold with a Lion motif? Therefore, I believe a frequently-used WHERE query will involve Recipe\_Component.Inv\_ID (which is already indexed as an FK in our SoapSafari database), WHERE Ingredient.Ingred\_Type

is, for example, ‘E’ (essential oil). Also, to find a lion-themed soap mold, I would want to search the Durable\_Good.Dur\_Good\_Name for a substring WHERE the substring ‘Lion’ is in the name. I have hundreds of molds; therefore, I want to index the column Durable\_Good.Dur\_Good\_Name.

1. For tax reasons, querying the database where:
2. purchase date is between Jan 1 and Dec 31 of the current tax year (last year), and
3. the starting number of units at the beginning of the year (from a history table?) > 0

will be important. Therefore, Inventory.Purchase\_Date, and Ingredient.Start\_Units will be indexed. (Supply.Start\_Units will also be used in the same type of WHERE clause; however, I only have a small handful of supplies. I have many hundreds of ingredients.

Here are the query-driven indexes I created (see also SQL in the section above):

|  |  |  |
| --- | --- | --- |
| **Column** | **Unique?** | **Description** |
| Ingredient.Ingred\_Type | Not unique | Ingredient refers to 7 different ingredient types. There are many ingredients in each type. |
| Durable\_Good.Dur\_Good\_Name | Not unique | It is *probable* that Durable\_Good will contain a list of unique names; however, it is also *possible* that this is not the case. |
| Inventory.Purchase\_Date | Not unique | Inventory will have many items with the same purchase date. |
| Ingredient.Start\_Units |  | Ingredient will likely have many items with the same number of start units |

# “Order of Operations” for Running Procedures and Scripts

Please execute SQL files in the following order:

SS2Create (must be first)

SS2FillData (must be second)

SS2NegTrigger

SS2TestNegTrigger (sometime after SS2NegTrigger)

SS2PriceHistTrig

SS2TestPriceHistTrig (sometime after SS2PriceHistTrig)

SS2AddIngredient

SS2ExecuteAddIngredient (sometime after SS2AddIngredient)

SS2AddPersonalUse

SS2ExecuteAddPersUse (sometime after SS2AddPersonalUse)

SS2ExecuteNotEnuffBars (sometime after SS2AddPersonalUse)

SS2MakeRecipe

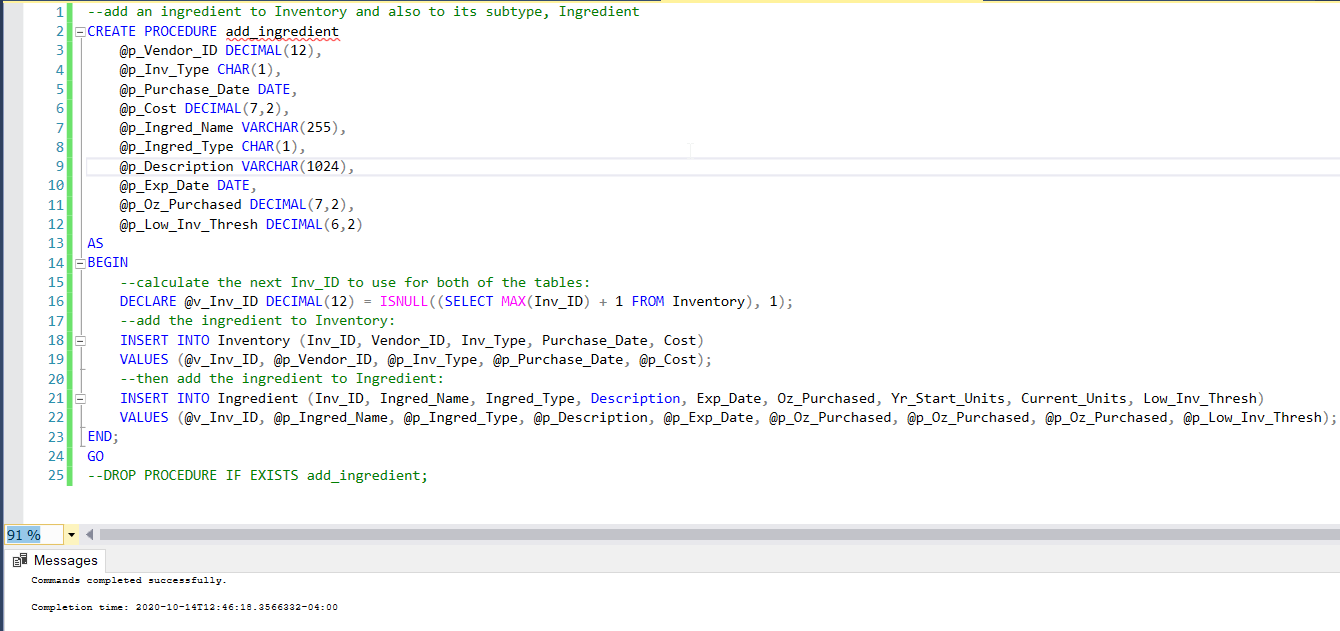
SS2ExecuteMakeRecipe (sometime after SS2MakeRecipe)

SS2QueriesQuestions (sometime after TestPriceHistTrig)

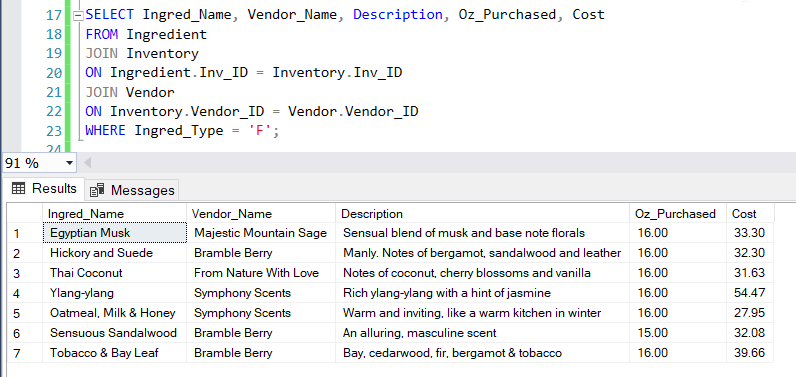
# SoapSafari SQL Transactional Stored Procedures:

## SS2AddIngredient (Addresses Use Case 1):

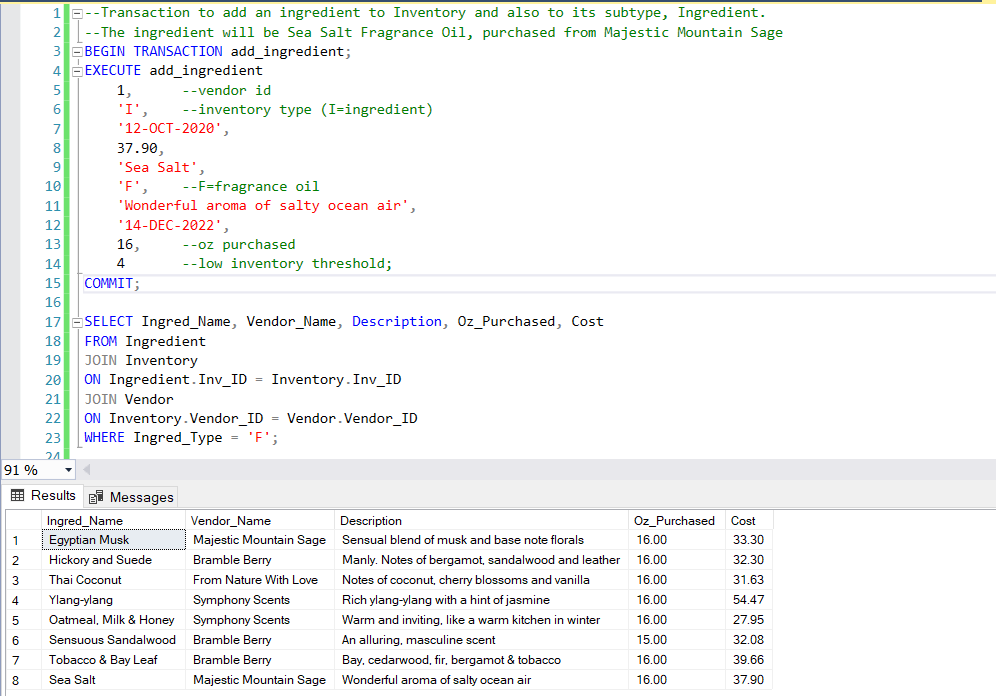
This procedure takes all necessary parameters and creates a new addition to the Inventory and Ingredient tables. This procedure (or variations of it) will be very useful to SoapSafari:



The SELECT clause shows the list of all fragrance oils in the database, along with some useful attributes, before execution of add\_ingredient:

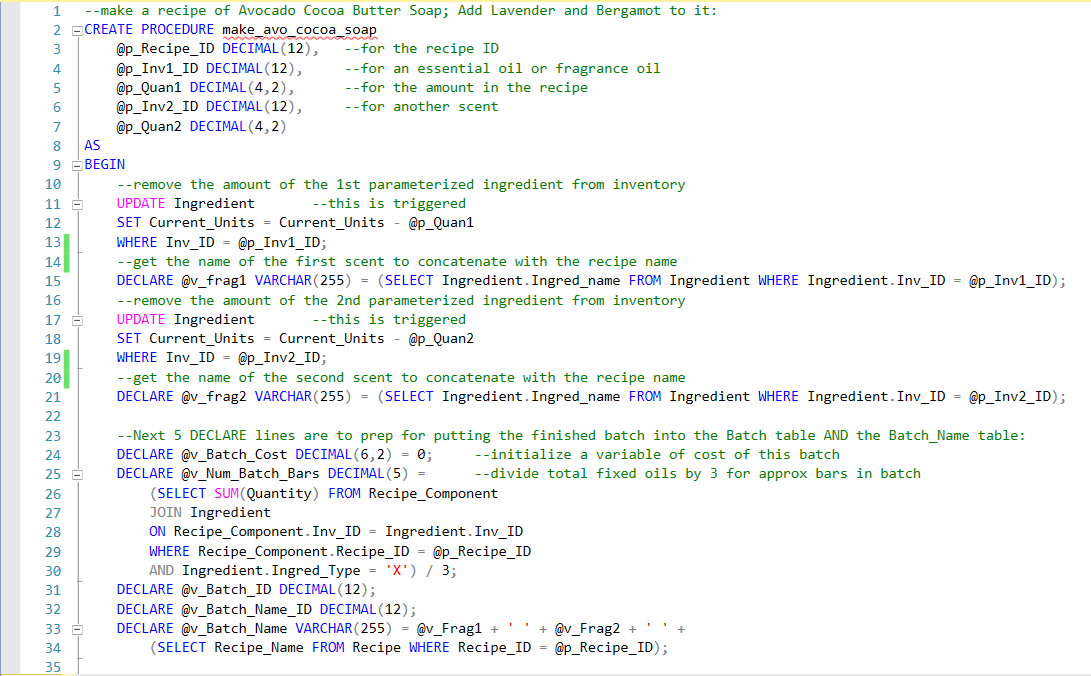


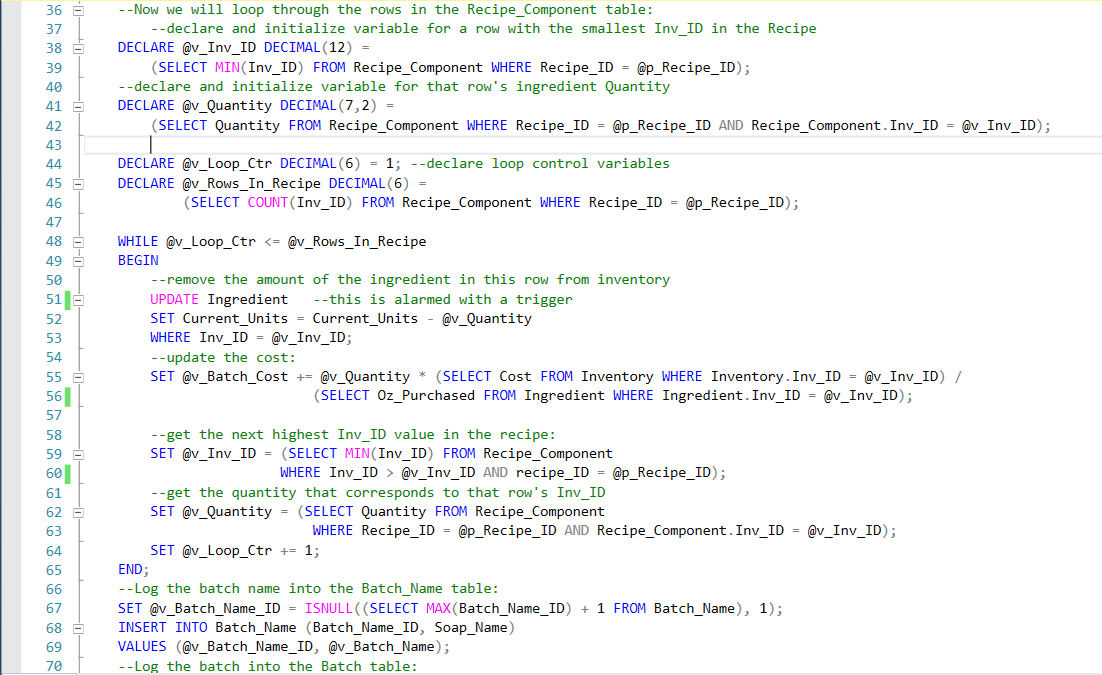
And here is the list of fragrance oils AFTER execution of add\_ingredient (Sea Salt fragrance oil):

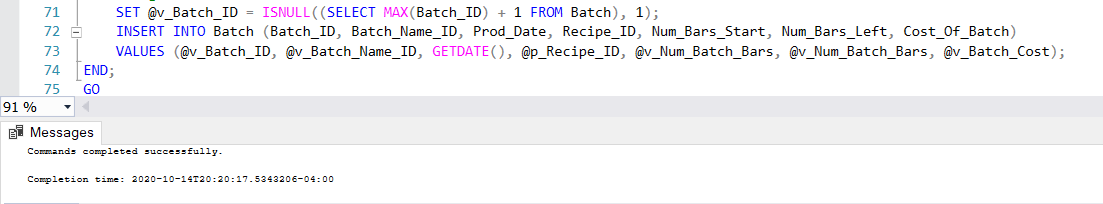


## SS2MakeRecipe (Addresses Use Case 2) (Farina, 2019) (Microsoft, 2020):

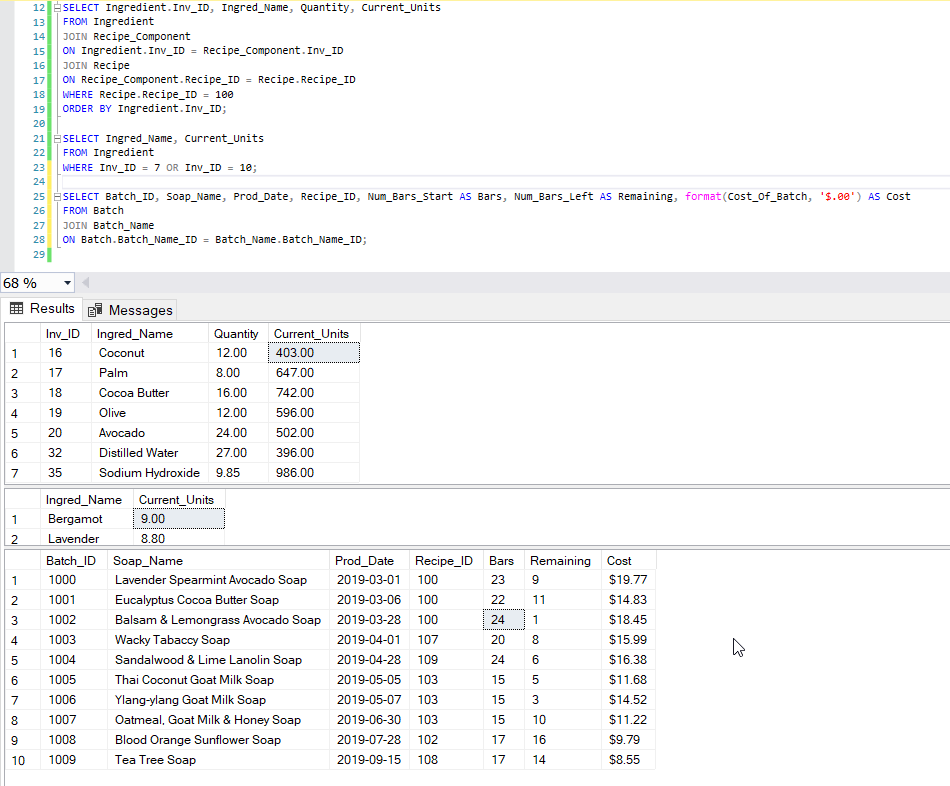
This procedure will be extremely useful. It allows running a recipe that is stored in the database, adding parameterized ingredients, such as essential oils or botanicals, and, upon execution, will subtract the recipe quantities from inventory and add the finished batch of soap to the Batch table, along with the calculated cost of the batch.



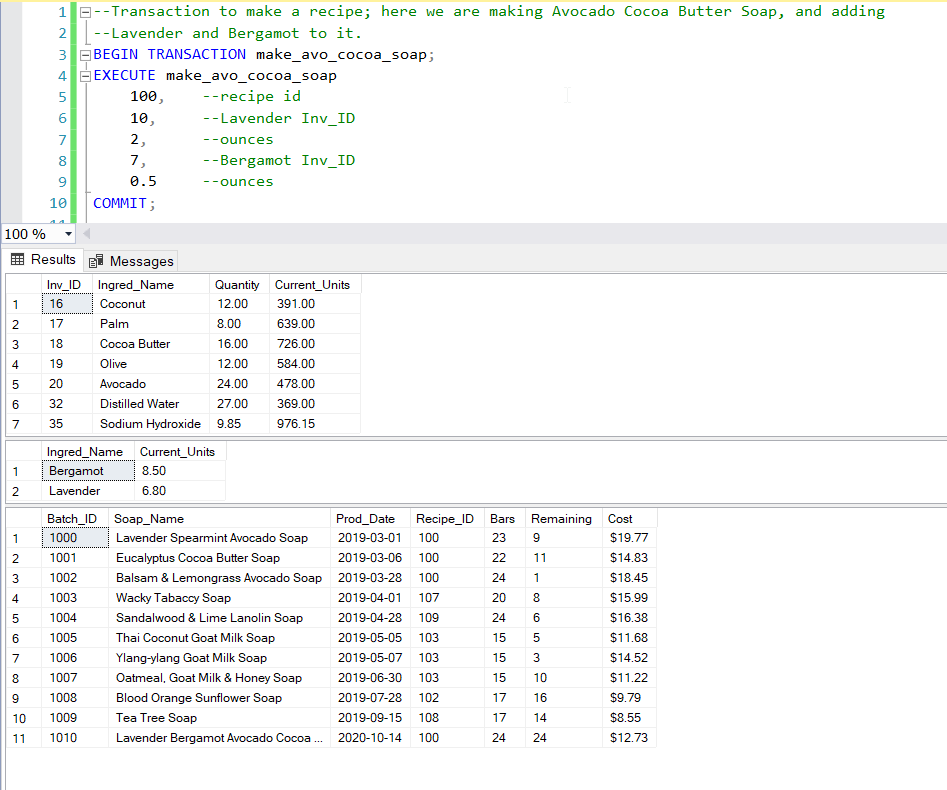




Prior to the executing of the above procedure, we can see the ingredients and quantities of fixed oils in the recipe, the total number of ounces of each ingredient in inventory, the quantities of the parameterized essential oils we will add to the recipe, and the total Batches of finished soap:

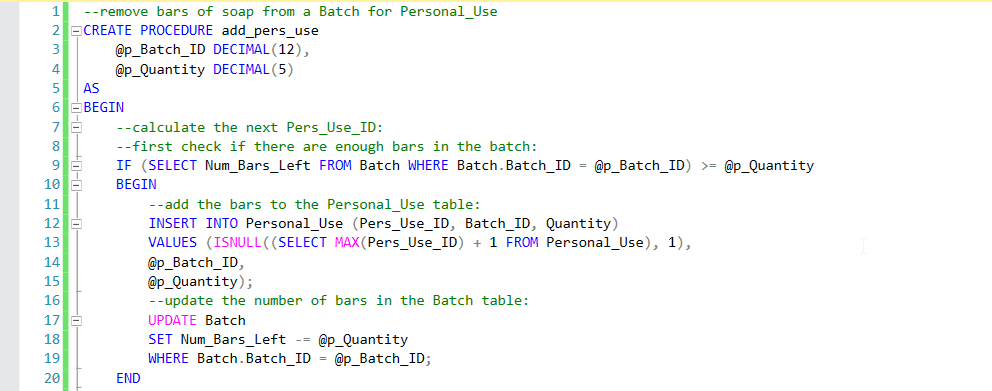


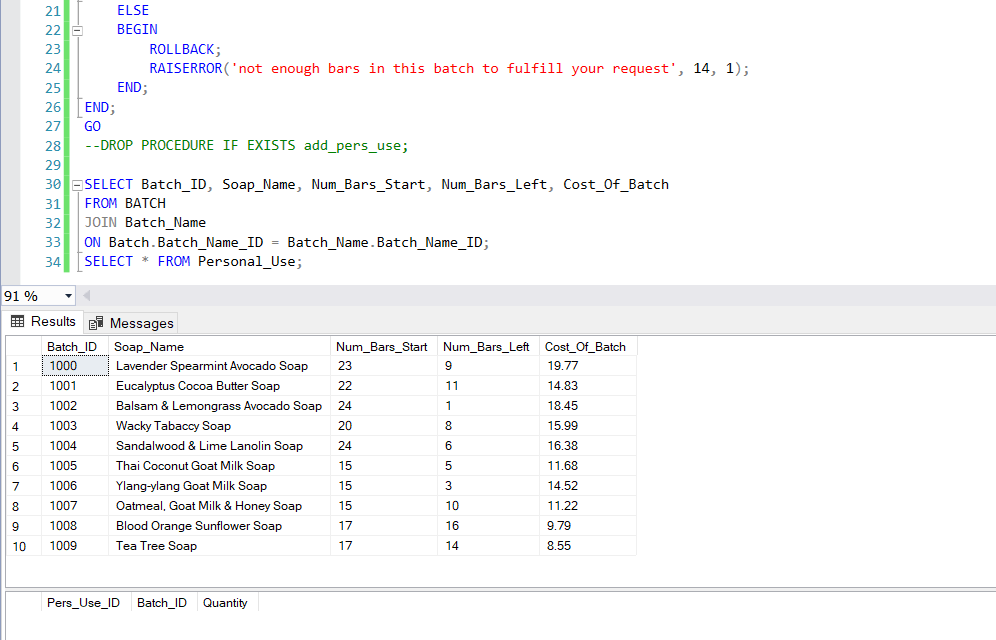
After execution, we can see that the ingredients in inventory are reduced by the recipe quantities, and that the finished batch of soap is added to the Batch table:



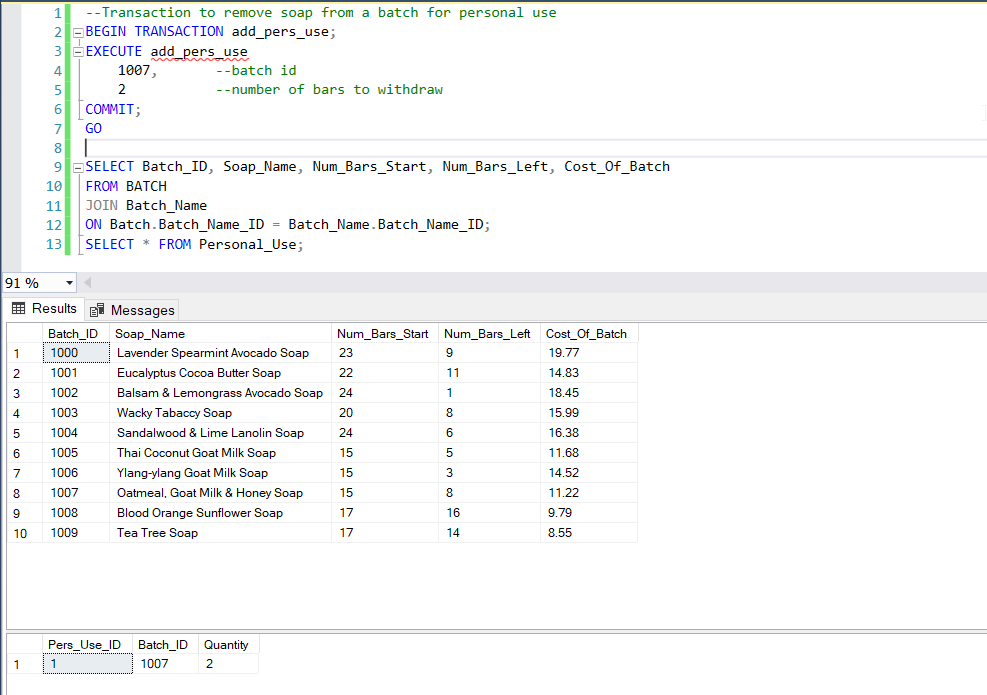
## SS2AddPersonalUSE (Addresses Use Case 7)

A LOT of my soap is withdrawn for personal use. This procedure allows me to withdraw bars and actually know exactly what they cost to produce, so I do not chalk them up as a business expense. Below is a screenshot of the add\_pers\_use Procedure, with Batch and Personal\_Use tables shown before the withdrawal of two bars of Oatmeal, Goat Milk & Honey Soap:

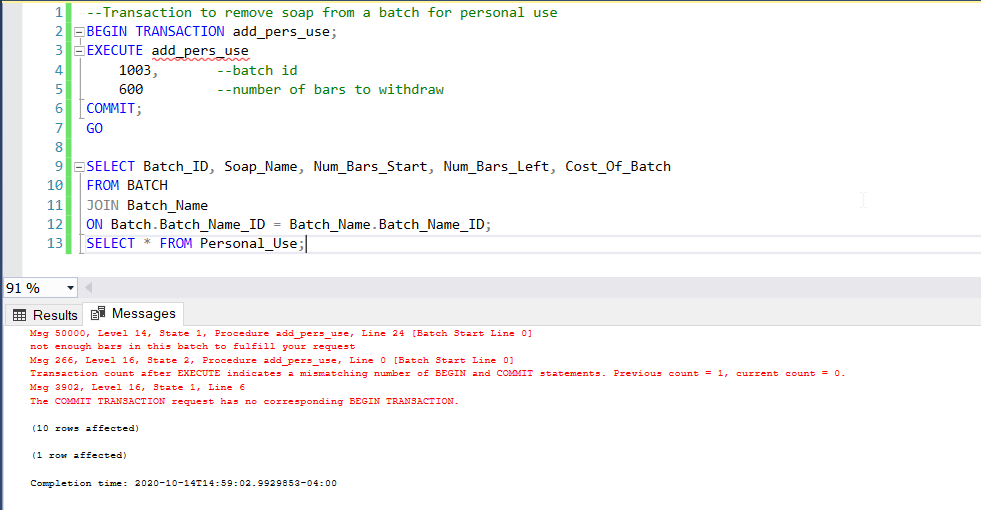




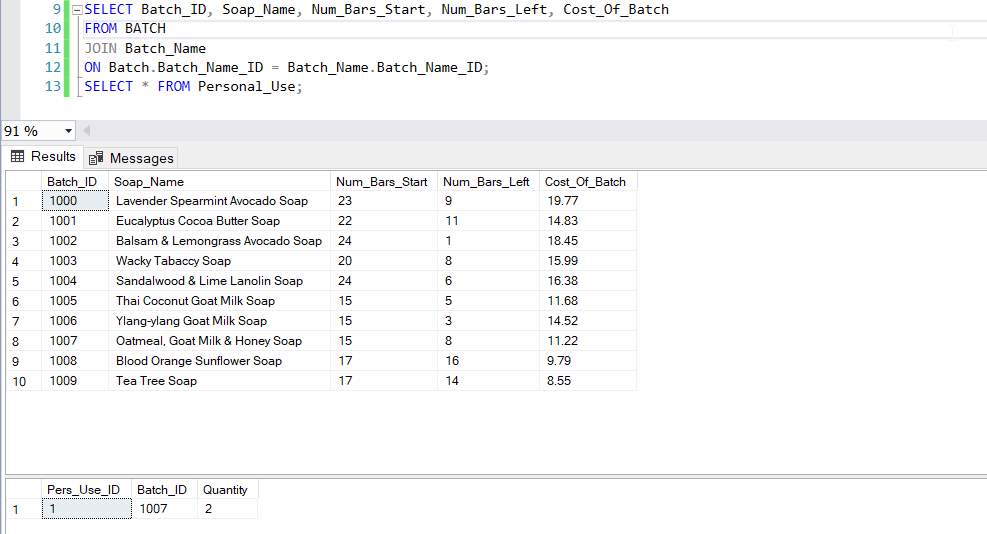
Two bars of Batch number 1007 withdrawn for personal use:



Negative test: withdraw 600 bars from batch number 1003, the Wacky Tabaccy Soap:



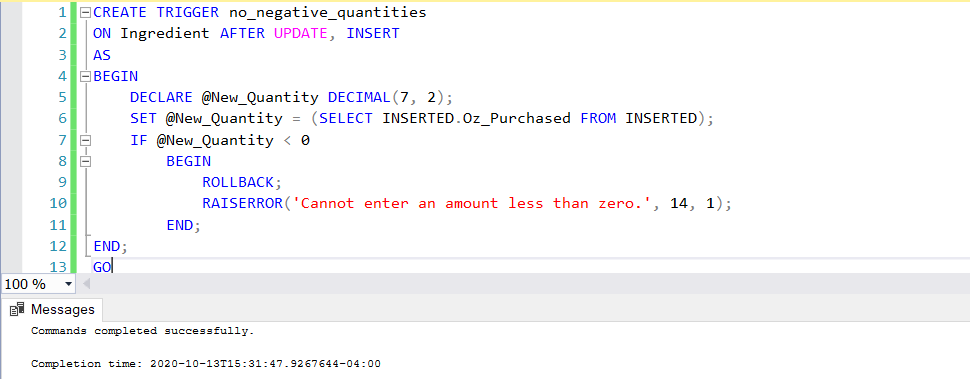
Results are that no soap was withdrawn:



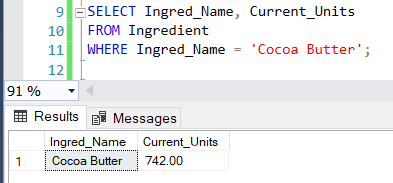
# Triggers

## Negative quantities in Ingredients:

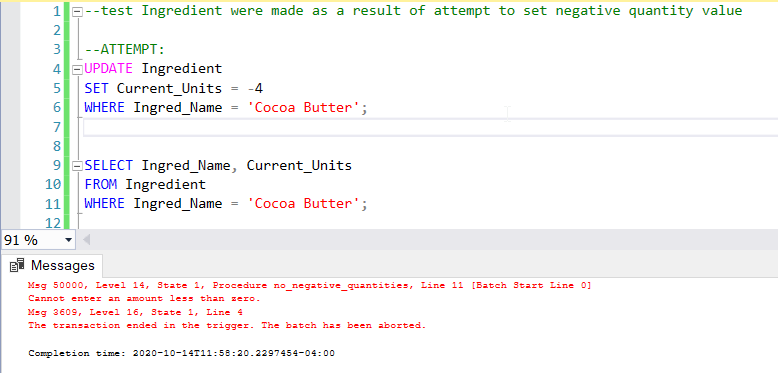
Anyone can “fat-finger” something; error-checking is good to make sure a negative quantity is not added to the ingredient table:



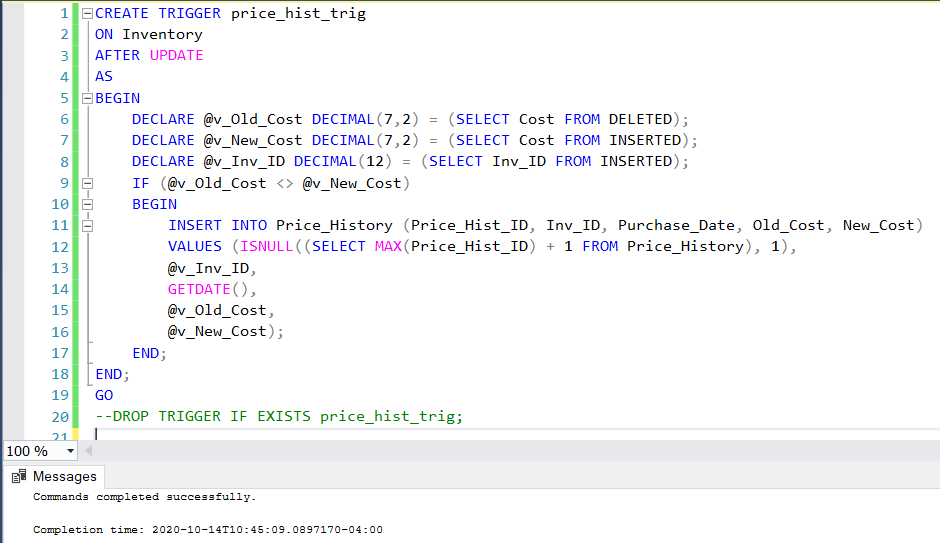
Current Cocoa Butter Units:



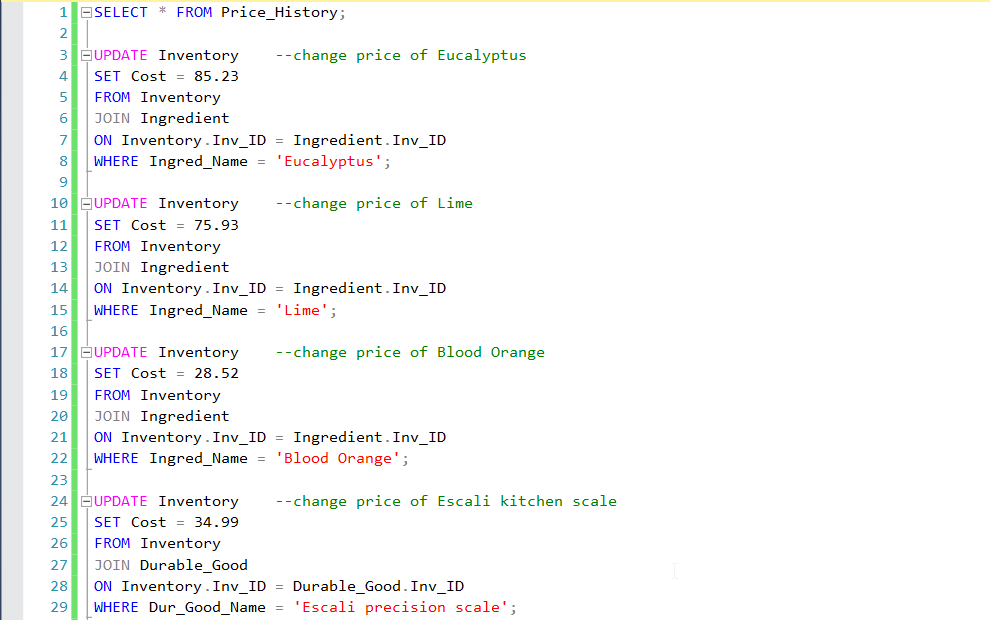
Attempt to change Current\_Units to a negative number:

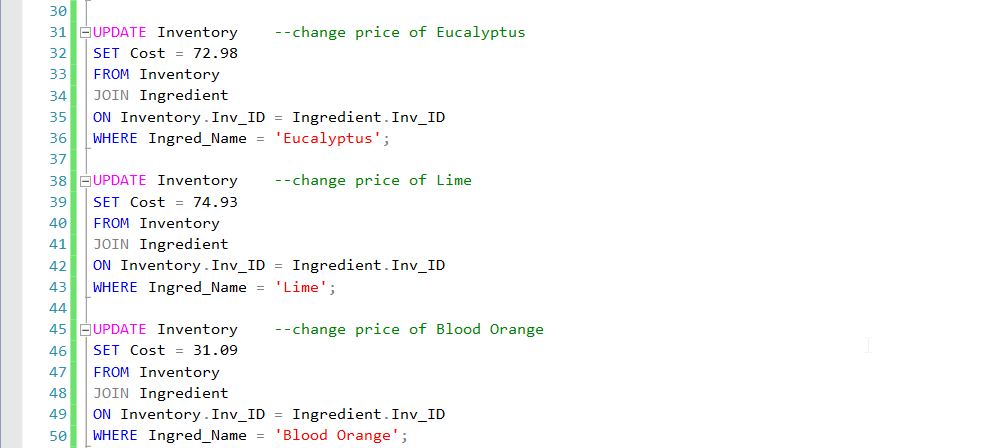


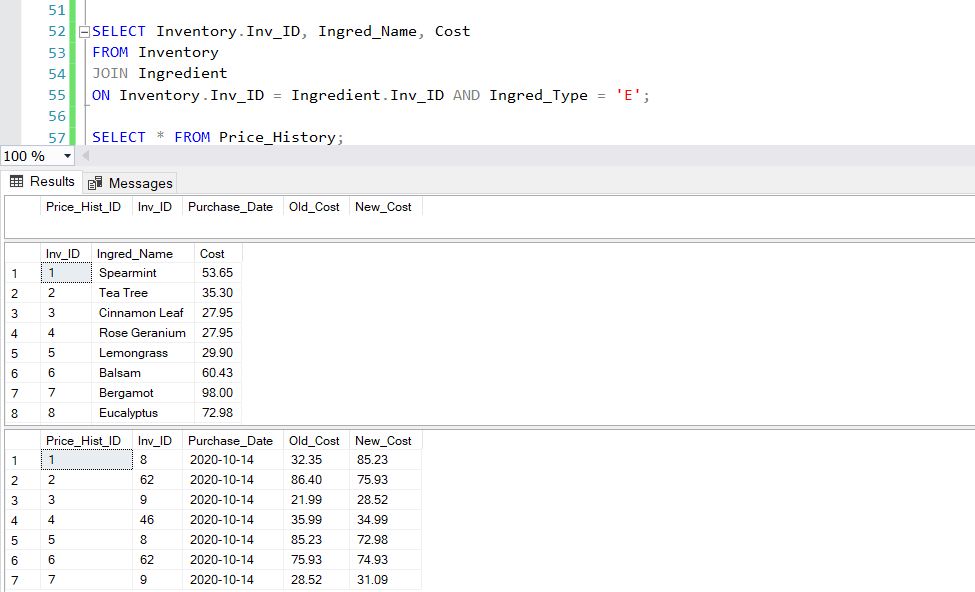
## Price History Trigger:



Before changes are made to the cost of some essential oils, the Price\_History table is empty; after changes, the table is automatically updated:







(SEE ALSO HISTORY TABLE, ABOVE)

# Questions and Queries:

Here are some questions and corresponding queries that my database will be useful for:

My coconut oil was just recalled--are there known customers who purchased soap with it?

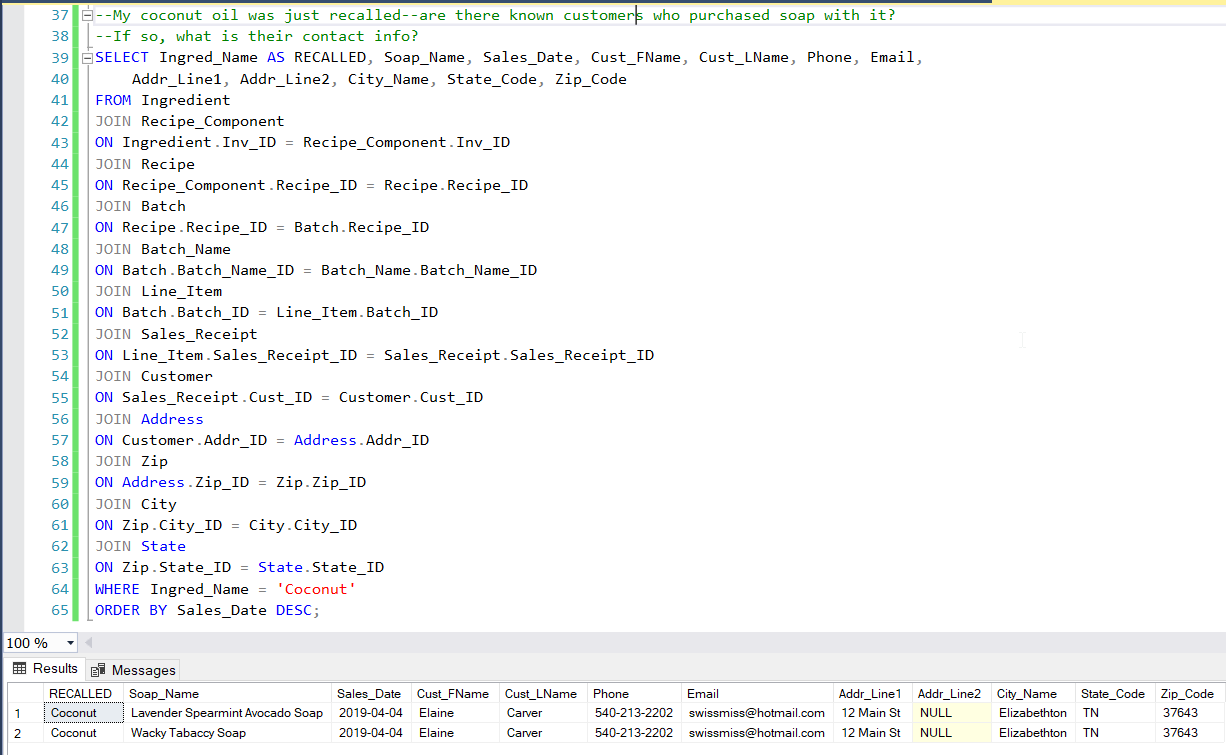
If so, what is their contact info?

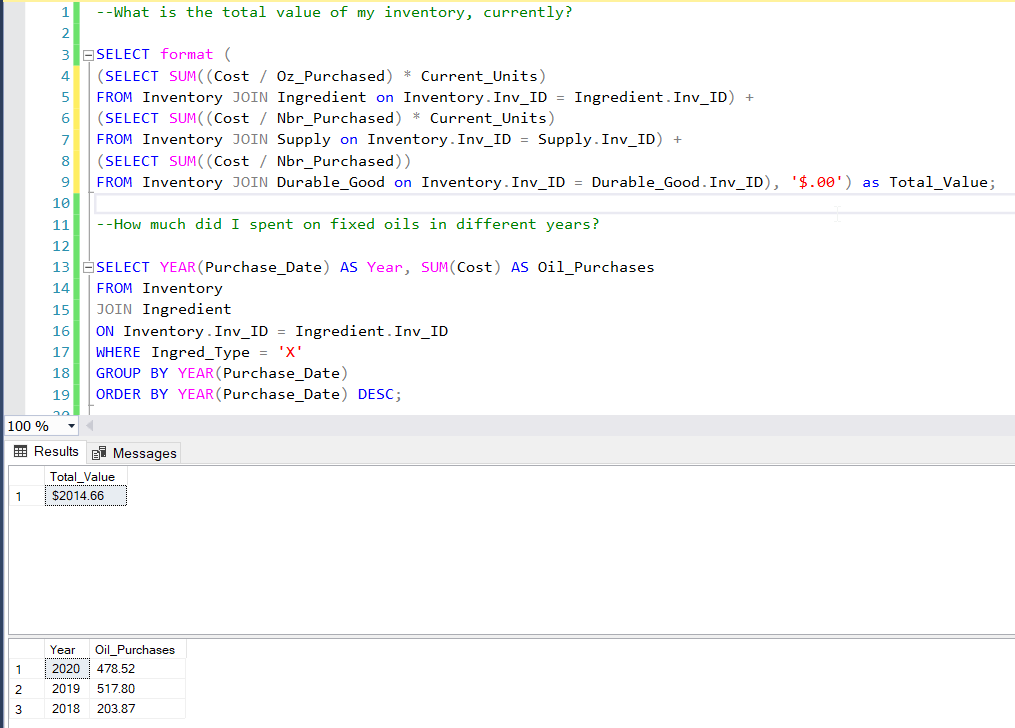
What is the total value of my inventory, currently?

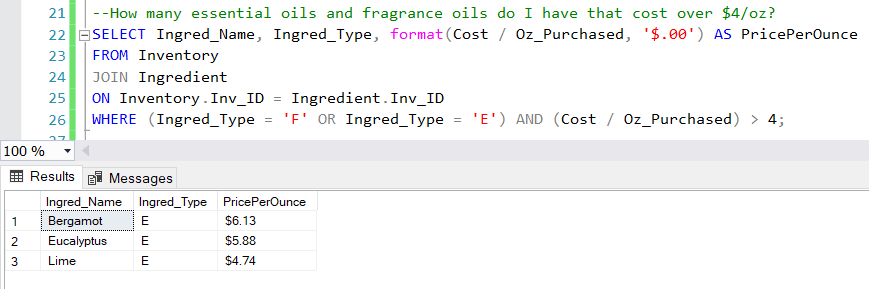
How much did I spend on fixed oils in different years?

How many essential oils and fragrance oils do I have that cost over $4/oz?

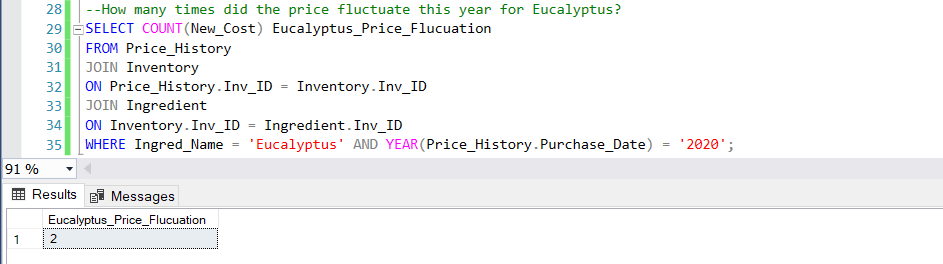
How many times did the price fluctuate this year for Eucalyptus?







How many times did the price fluctuate this year for Eucalyptus?



My database for SoapSafari will allow me to keep track of inventory so low items can be reordered, inventory will be continually updated each time a recipe is made and/or an item is added or deleted, numbers can be produced for schedule C at the end of the year in a trivial manner, and withdrawing items for personal use is simple now.

There are many more procedures to be added to make this database fully operational, including those for processing values for schedule C, and exception-handling. This project has compelled me to learn more SQL than I thought I ever could or would, including extremely valuable skills in database design; and it has been a great journey. Many thanks to Warren Mansur and Jim Gibson of Boston University’s MET CS Department, for the impartment of knowledge and help.

Sincerely,

Tini Grosso

# Works Cited

Athena. (2012, March 30). *Glycerin and a 100% glycerin Soap?* Retrieved from simplyeden: https://www.simply-eden.com/blogs/interesting-facts/5902187-glycerin-and-a-100-glycerin-soap#:~:text=A%20100%25%20glycerin%20soap%20is,of%20the%20soap%2Dmaking%20process.&text=Glycerin%20alone%20cannot%20make%20a%20bar%20of%20soap.

Farina, D. (2019, Sept 12). *SQL Server Loop through Table Rows without Cursor*. Retrieved from MSSQLTips: https://www.mssqltips.com/sqlservertip/6148/sql-server-loop-through-table-rows-without-cursor/

Ida. (2014, July 29). *Glycerin Soap & Glycerin*. Retrieved from Chagrin Valley Soap & Salve Company: https://www.chagrinvalleysoapandsalve.com/blog/posts/glycerin-soap-glycerin/

Microsoft. (2020, August 19). *How to iterate through a result set by using Transact-SQL in SQL Server*. Retrieved from Microsoft Support: https://support.microsoft.com/en-us/help/111401/how-to-iterate-through-a-result-set-by-using-transact-sql-in-sql-serve

US Government. (2019). *Form 1040 Schedule C*. Retrieved from Internal Revenue Service: https://www.irs.gov/pub/irs-pdf/f1040sc.pdf

Wikipedia. (2020, Sept 29). *List of long place names*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/List\_of\_long\_place\_names

