Corey Clow, Tejaswini Jagtap, Zach Morgan, Himani Munshi
Professor Passino
Principles of Data Management, CSCI-320-01
14 April 2019

Group 3 Phase 3 Report: Retail Domain

1. Description

- Considering the power our customers have today - they can decide our store's fate by deciding to either buy our products or not - it's crucial that we remain a step ahead of them. That is why retail data is so important.
- This database is primarily designed for vendors, administrators, online, and offline customers. These different users access the database through different applications made specifically for them.
- Our database allows customers to either shop online or offline. Online customers directly become frequent shoppers (members) of all the stores in the database.
- Our database also allows anonymous customers who to make a purchase without any registration provided they shop from a physical store.
- The online customer application enables customers to view their cart or checkout, view product categories, search for product, or view their personal information.
- The checkout register application for customers who shop from physical stores enables them to view their cart or checkout, scan their product to cart, view their information, or cancel checkout.
- The vendor application provides vendors access to view unshipped order requests, orders in transit, completed orders, or all orders.
- The administrator application allows administrators to get information of the data in the database by writing SQL queries.

2. Database design (high-level)

As a high-level description of our design, we focused on four main concepts: product, store, customer, and vendor. With these established as the pillars of our retail domain, lower level questions can be answered such as:

- What defines a given pillar?
- How do they interact with each other?
- How do they map to the real world?

2.1. Product

In the retail domain, a product is simply something sold, such as a loaf of bread, a chair, or a pair of jeans. Analyzing this pillar, we subcategorized it into:

- Cleaning products
- Drug products
- Food products
- Furniture products
- Clothing products

Not only does this design make it easy for someone to see the general purpose of a product by its category, but it also allows certain attributes only relevant to that category to be stored separately. This means if the retailer wants to start selling a new type of product in the future, it is possible to plug and play a new category without restructuring the higher-level product concept.

In terms of interaction, it also allows general top-level statements about all products, such as “A product has a brand” and “A product is sold in a certain package”, or specific statements about certain categories, such as “A food product is made of ingredients”. Using this hierarchical design means other pillars can interact with whichever layer of abstraction they see fit.

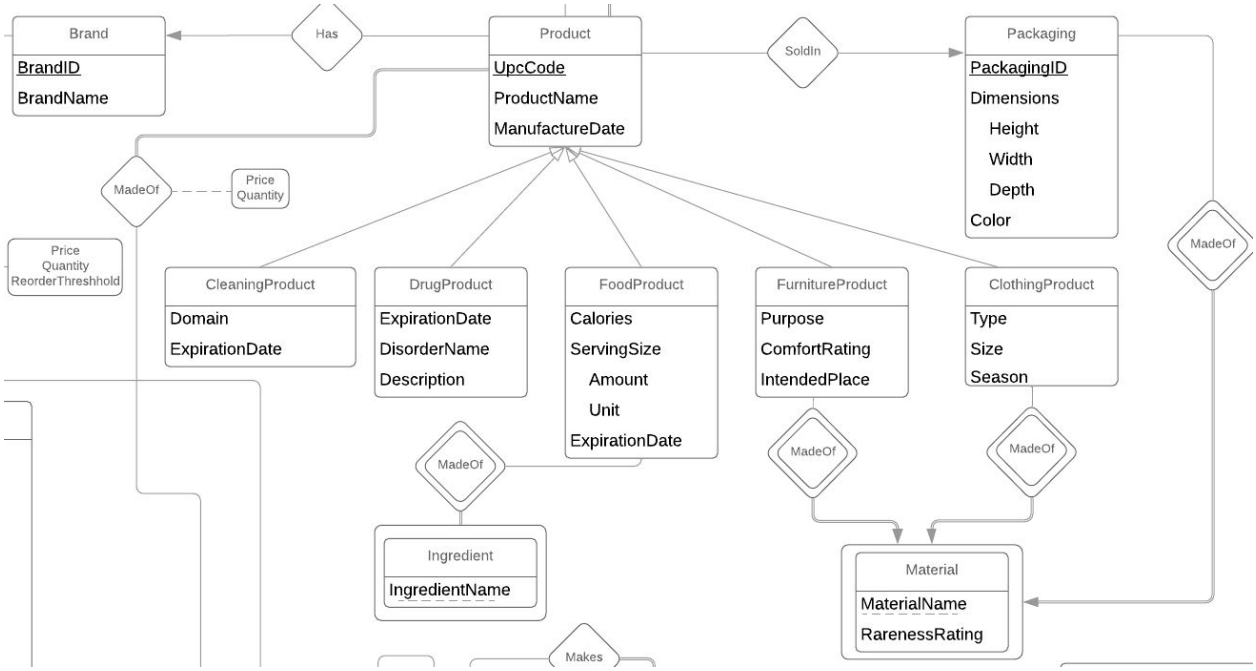


Figure 1. Product pillar

2.2. Store

A store is someone who sells products to customers. To distinguish between a physical store and an online store, we went with another hierarchical design where a physical store is a subcategory of a store. Because online stores share many attributes and processes with physical stores except physical location, an online store is denoted by being an instance of the upper-level store concept with no corresponding subcategory instance. If there were more distinguishing properties of an online store it would require another subcategory, but because the two are so similar both in the real world and in our interpretation, this is not necessary.

The hierarchical design for stores offers the same benefit as our design of products, i.e., we can make general statements about all stores such as “Stores make sales of products with

customers”, and “Stores purchase products from vendors”, or specific statements like “If a store has no corresponding physical location, its sales must be shipped”.

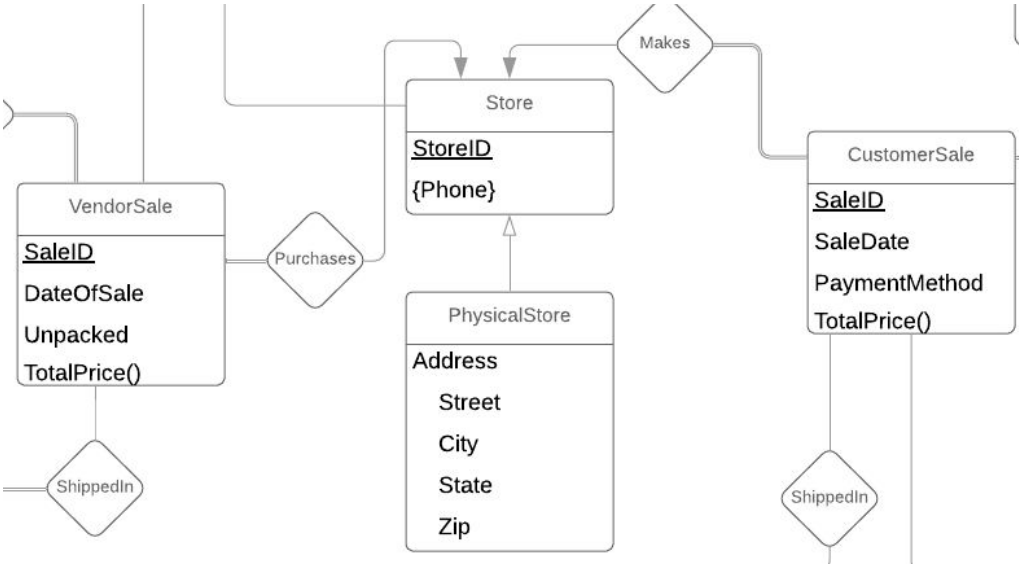


Figure 2. Store pillar

2.3. Customer

A customer is the purchasing recipient of a sale made with a store. In the real world, customers can be anonymous, e.g., someone who pays with cash and walks out the door, or have some sort of membership where they reveal details about themselves. To distinguish between the two in our design, we only have a frequent shopper concept, with no “anonymous customer” concept. The reasoning behind this is because if we did, it would be possible to group all the sales made by an anonymous customer, which raises the question, would they still be anonymous? Additionally, in keeping with the real world, it is not possible to track all of the sales made by an anonymous customer. If someone wants to receive the benefits of becoming a frequent shopper, they can sign up through our online application.

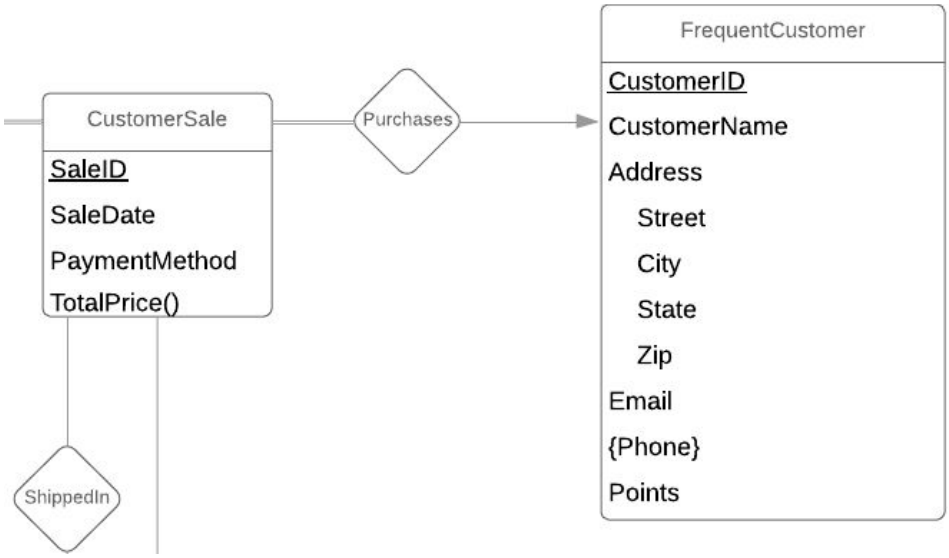


Figure 3. Customer pillar

2.4. Vendor

A vendor is someone who sells products of certain brands to stores, analogous to the store in the store-customer relationship. In this case, the vendor makes the sale with a store, and the store is the purchasing recipient of the sale. The main difference is sales made by vendors must have a corresponding shipment because, in the real world, retail stores order from vendors all across the country.

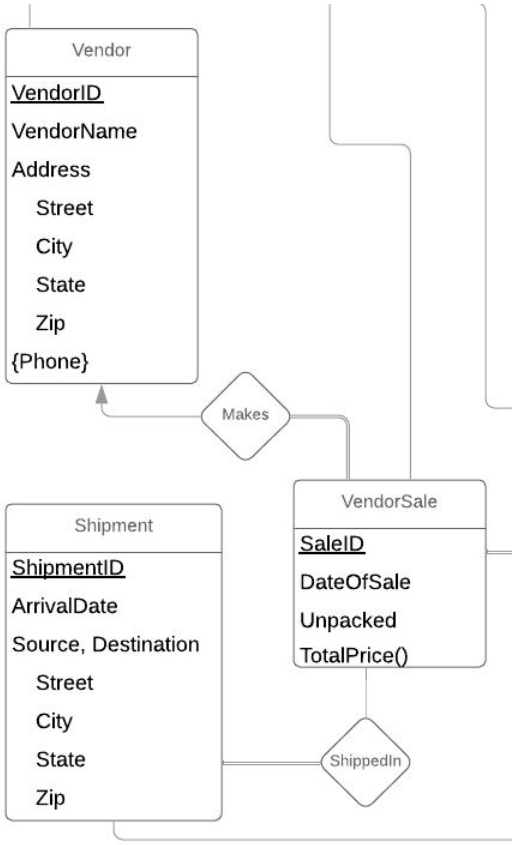
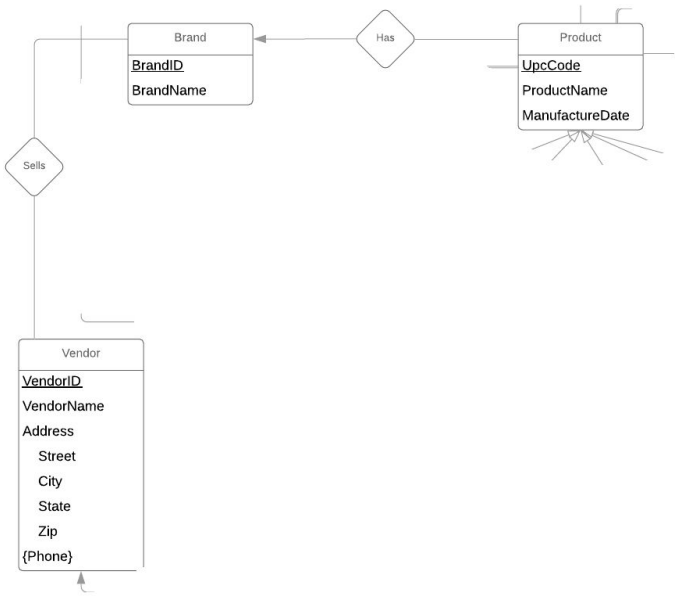


Figure 4. Vendor pillar

3. Database design (tables)

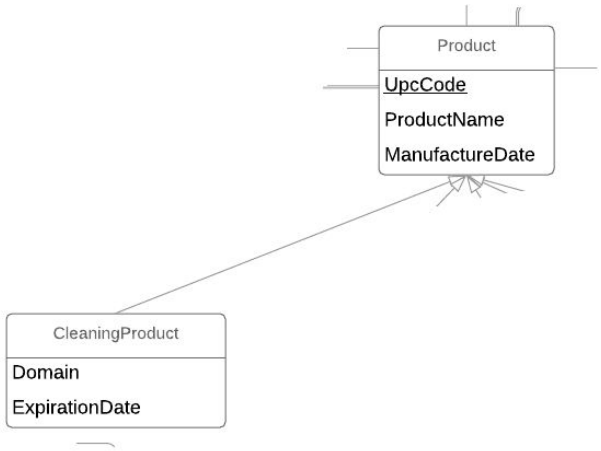
The following tables outline each relation that resulted from reducing our ER diagram, as well as the domain, or allowed values of each attribute. For the actual sample data, see the corresponding CSV files.

- Brand (Primary Key: BrandID)



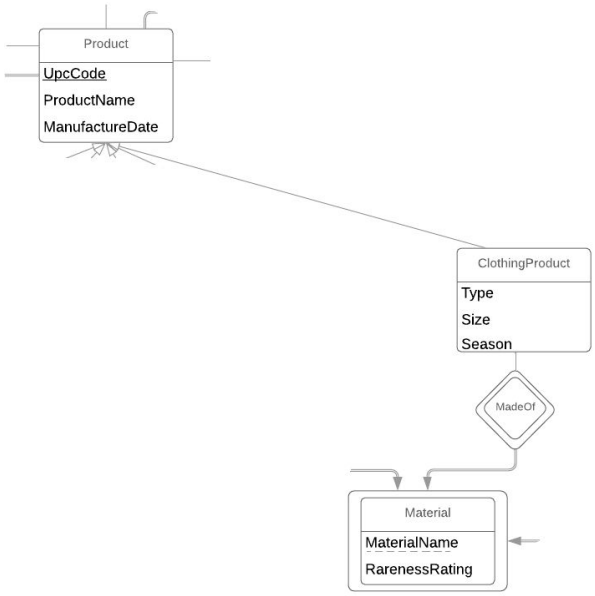
Attribute	BrandID	BrandName
Domain	Primary key, integer, >= 1	varchar(20)

- CleaningProduct (Primary Key: ProductUpcCode)



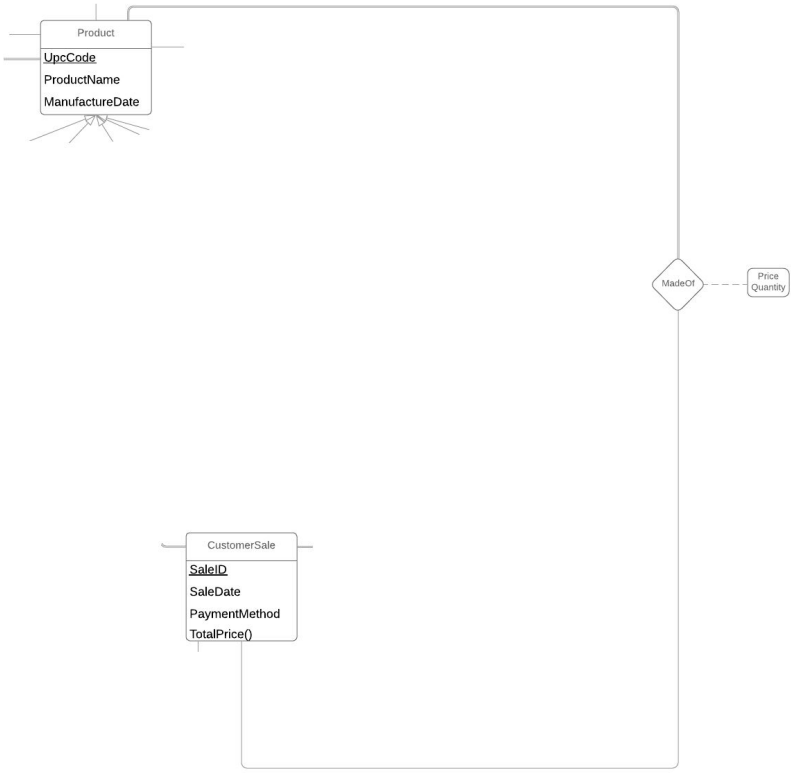
Attribute	ProductUpcCode	Domain	ExpirationDate
Domain	Primary key, integer >= 1	varchar(20)	Epoch timestamp (integer)

- ClothingProduct (Primary Key: ProductUpcCode)



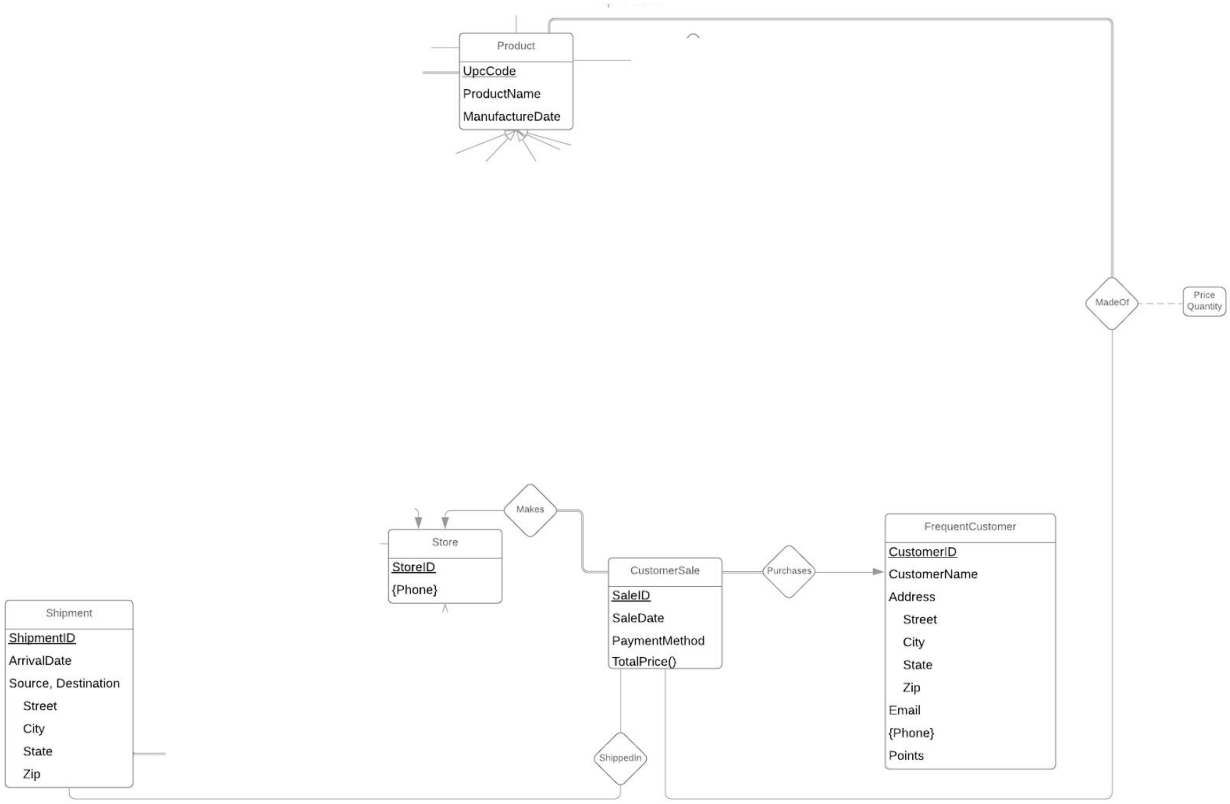
Attribute	ProductUpcCode	Size	Type	Season	MaterialName
Domain	Primary key, integer >= 1	varchar(5)	varchar(15)	varchar(15) Constraint: not null	Foreign key, varchar(20) Constraint: not null

- CustomerProductSale (Primary Key: {SaleID, ProductID})



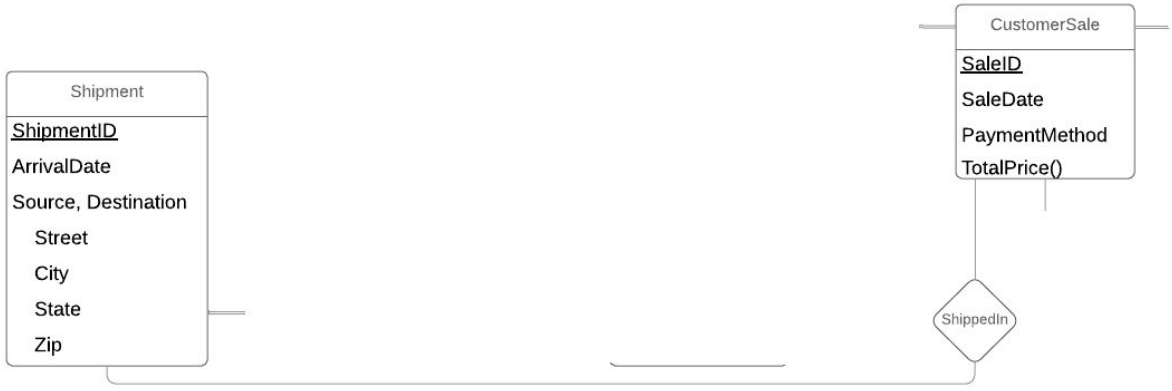
Attribute	SaleID	ProductID	Price	Quantity
Domain	Foreign key, integer >= 1	Foreign key, integer >= 1	Integer > 0 Constraint: not null	Integer > 0 Constraint: not null

- CustomerSale (Primary Key: SaleID)



Attribute	SaleID	SaleDate	PaymentMethod	StoreID	CustomerID
Domain	Primary key, integer >= 1	Epoch timestamp (integer) Constriant: not null	varchar(15) Constraint: not null	Foreign key, integer >= 1 Constraint: not null	Foreign key, integer >= 1 Constraint: not null

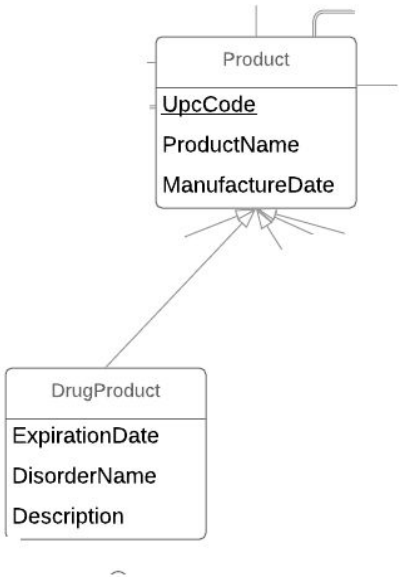
- CustomerSaleShippedIn (Primary Key: {SaleID, ShipmentID})



Attribute	SaleID	ShipmentID

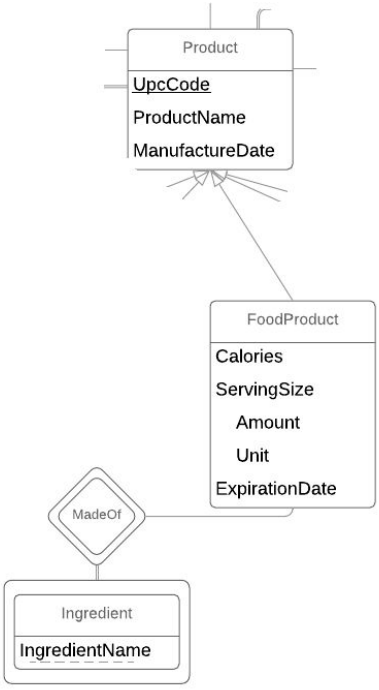
Domain	Foreign key, integer >= 1	Foreign key, integer >= 1

- DrugProduct (Primary Key: ProductUpcCode)



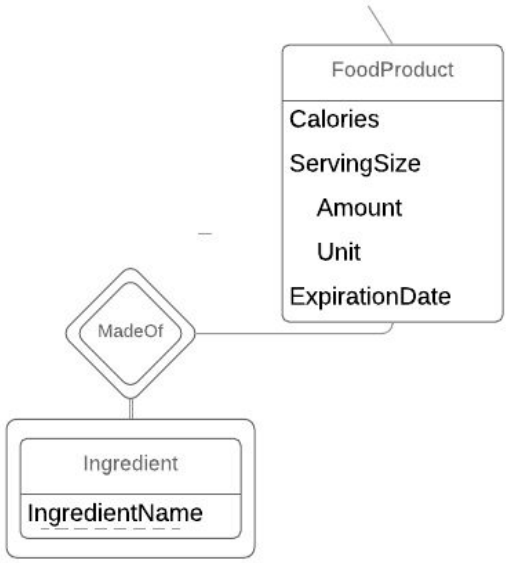
Attribute	ProductUpcCode	ExpirationDate	DisorderName	Description
Domain	Primary key, integer >= 1	Epoch timestamp (integer) Constraint: not null	varchar(30)	varchar(150)

- FoodProduct (Primary Key: ProductUpcCode)



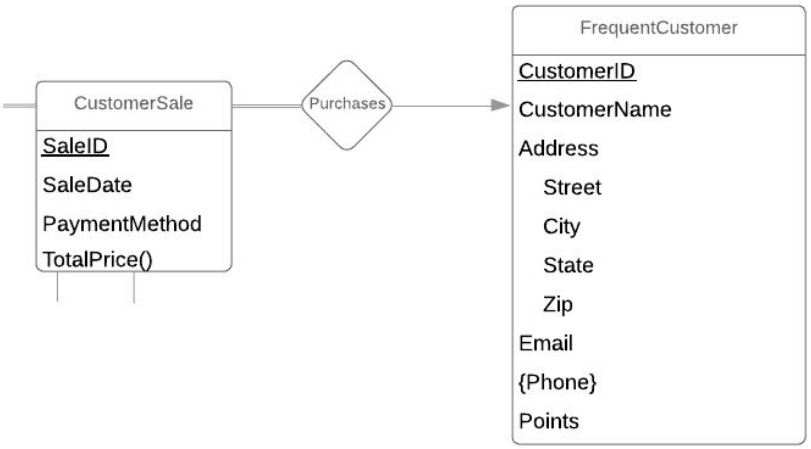
Attribute	ProductUpcCode	Calories	Amount	Unit	ExpirationDate
Domain	Primary key, integer >= 1	Integer	float(6) Constraint: not null	varchar(2) Constraint: not null	Epoch timestamp (integer)

- FoodProductIngredient (Primary Key: {IngredientName, ProductUpcCode})



Attribute	IngredientName	ProductUpcCode
Domain	Foreign key, varchar(35)	Foreign key, integer

- FrequentCustomer (Primary Key: CustomerID)



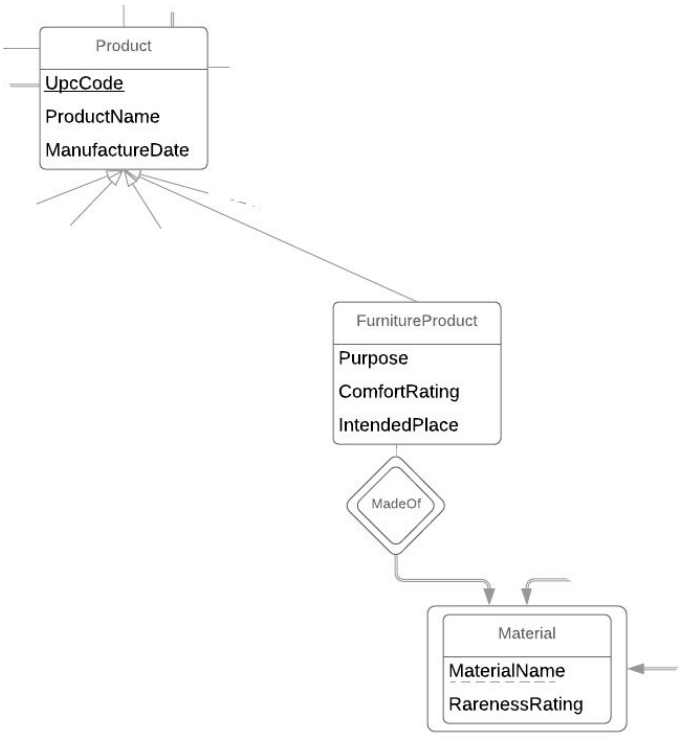
Attribute	CustomerID	Customer Name	Street	City	State	Zip	Points	Email

Domain	Primary key, integer >= 1	varchar(20) Constraint: not null	varchar (30)	varchar(20)	varchar(15)	varchar(15)	Integer > -1	Varchar (255)

- FrequentCustomerPhone (Primary Key: {CustomerID, AreaCode, Prefix, LineNumber})

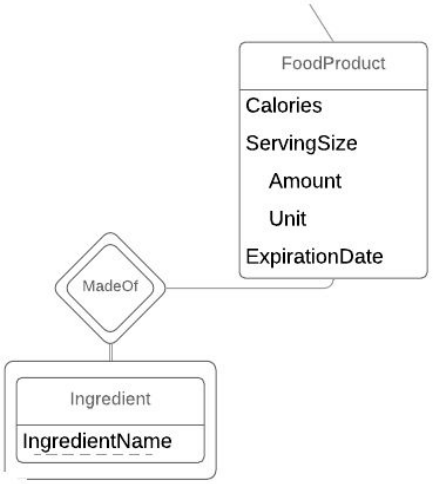
Attribute	CustomerID	AreaCode	Prefix	LineNumber
Domain	Foreign key, integer >= 1	Integer	Integer	Integer

- FurnitureProduct (Primary Key: ProductUpcCode)



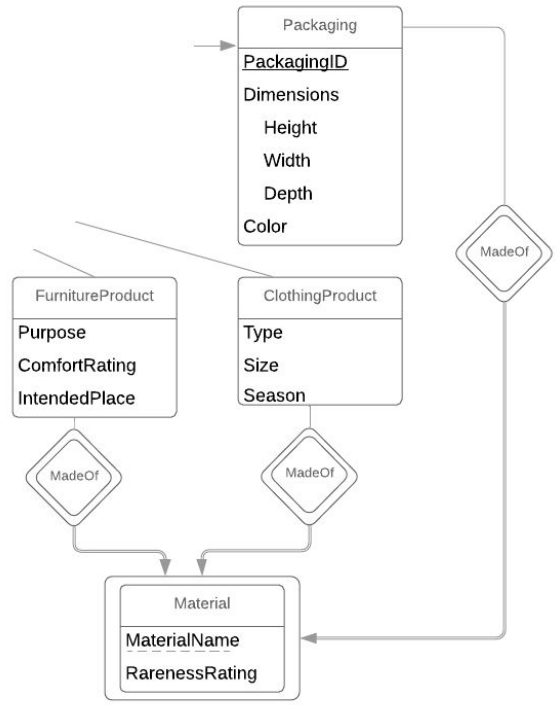
Attribute	ProductUpcCode	Purpose	ComfortRating	IntendedPlace	MaterialName
Domain	Primary key, integer >= 1	varchar(20)	-1 < Integer < 11	varchar(30)	Foreign key, varchar(20) Constraint: not null

- Ingredient (Primary Key: IngredientName)



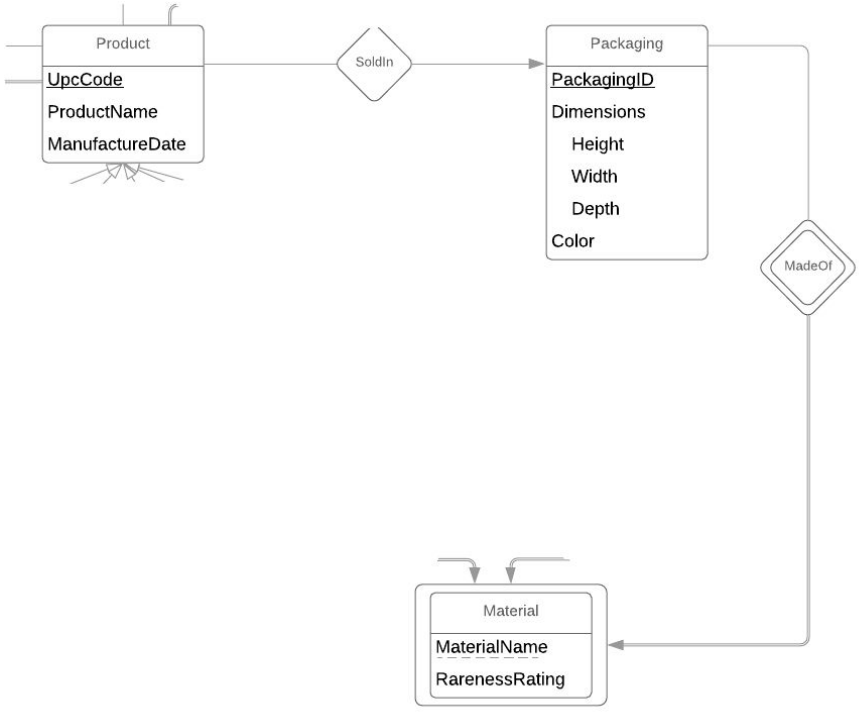
Attribute	IngredientName
Domain	Primary key, varchar(35)

- Material (Primary Key: MaterialName)



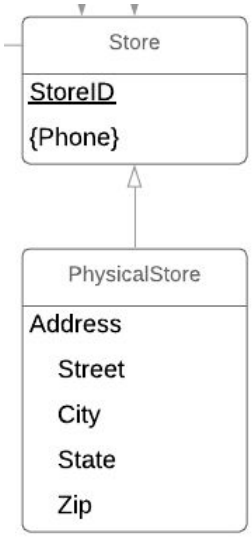
Attribute	MaterialName	RarenessRating
Domain	Primary key, varchar(20)	-1<Integer<101

- Packaging (Primary Key: PackagingID)



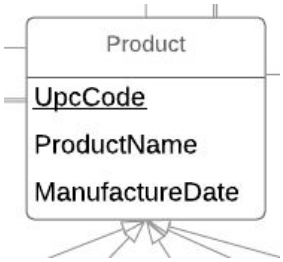
Attribute	PackagingID	Height	Width	Depth	Color	MaterialName
Domain	Primary key, integer >= 1	float(5) > 0.0	float(5) > 0.0	float(5) > 0.0	varchar(10)	Foreign key, varchar(20) Constraint: not null

- PhysicalStore (Primary Key: StoreID)



Attribute	StoreID	Street	City	State	Zip
Domain	Primary key, integer >= 1	Varchar(30) Constraint: not null	Varchar(30) Constraint: not null	Varchar(5) Constraint: not null	Integer Constraint: not null

- Product (Primary Key: ProductUpcCode)

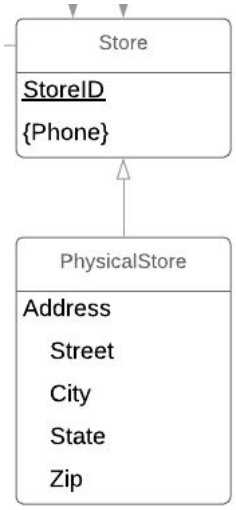


Attribute	ProductUpcCode	ProductName	ManufactureDate	PackagingID	BrandID
Domain	Primary key, integer >= 1	Varchar(50) Constraint: not null	Epoch timestamp (integer)	Foreign key, integer Constraint: not null	Foreign key, integer Constraint: not null

- Shipment (Primary Key: ShipmentID)

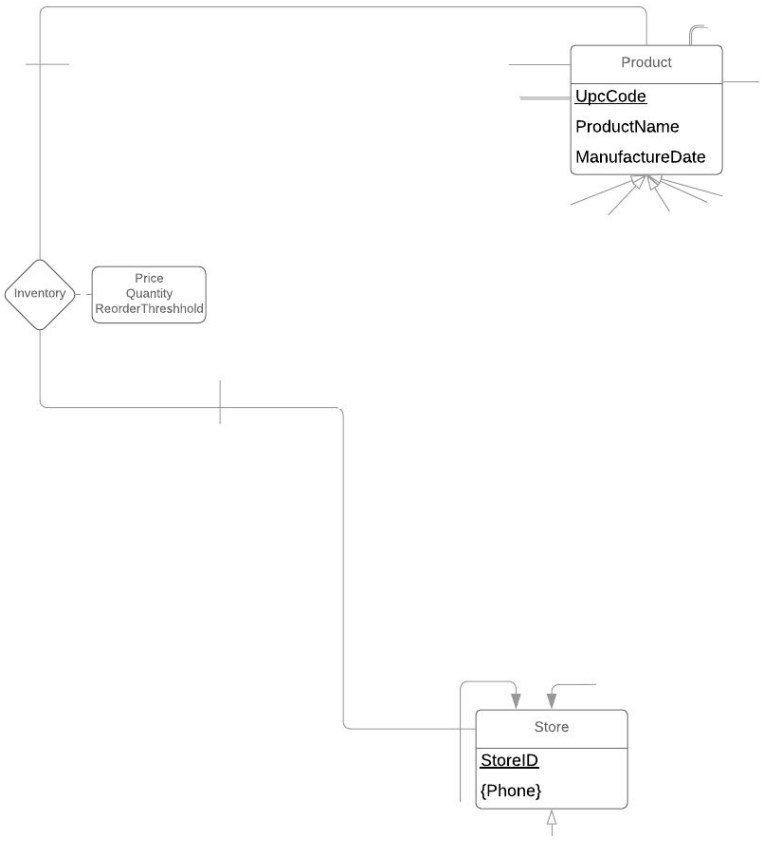
Attribute	Shipment ID	ArrivalDate	SrcStreet	SrcState	SrcZip	DstStreet	DstState	DstZip
Domain	Primary key, integer >= 1	Epoch timestamp (integer)	Varchar(30) Constraint: not null	Varchar(5) Constraint: not null	Integer, not null Constraint: not null	Varchar(30), not null Constraint: not null	Varchar(5), not nul Constraint: not nulll	Integer, not nul Constraint: not nulll

- Store (Primary Key: StoreID)



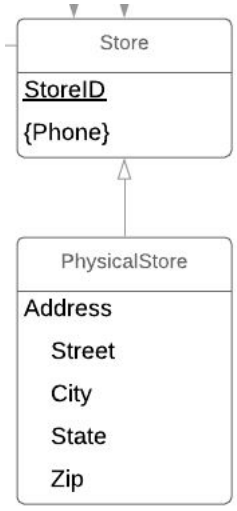
Attribute	StoreID
Domain	Primary key, integer >= 1

- StoreInventory (Primary Key: {StoreID, ProductID})



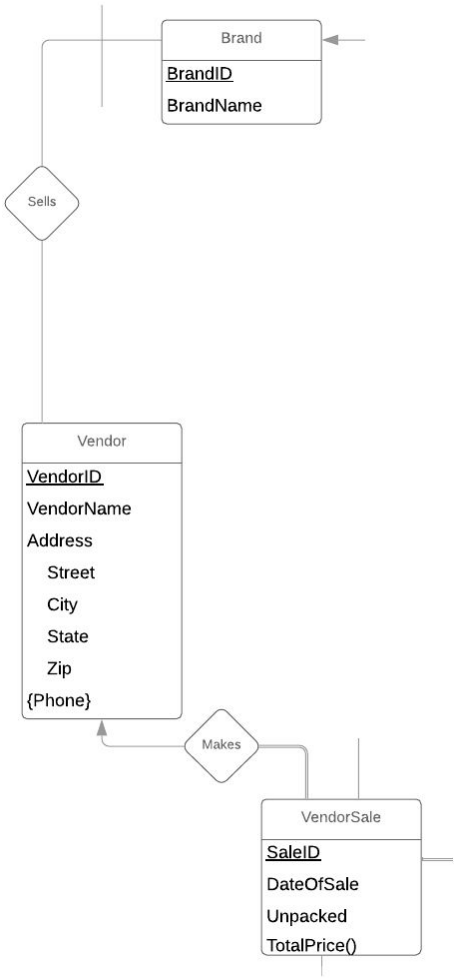
Attribute	StoreID	ProductID	Quantity	Price	ReorderThreshold
Domain	Foreign key, integer >= 1	Foreign key, integer >= 1	Integer > -1 Constraint: not null	Integer > 0 Constraint: not null	Integer >= 0

- StorePhone (Primary Key: {StoreID, AreaCode, Prefix, LineNumber})



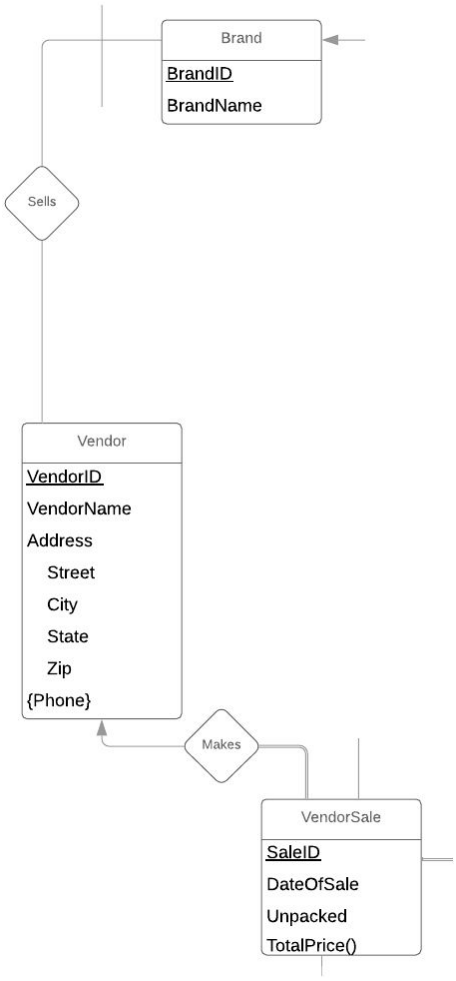
Attribute	StoreID	AreaCode	Prefix	LineNumber
Domain	Foreign key, integer >= 1	Integer	Integer	Integer

- Vendor (Primary Key: VendorID)



Attribute	VendorID	VendorName	Street	City	State	Zip
Domain	Primary key, integer >= 1	Varchar(30) Constraint: not null	Varchar(30) Constraint: not null	Varchar(30) Constraint: not null	Varchar(5) Constraint: not null	Integer Constraint: not null

- VendorBrand (Primary Key: {VendorID, BrandID})



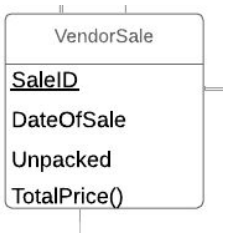
Attribute	VendorID	BrandID
Domain	Foreign key, integer >= 1	Foreign key, integer >= 1

- VendorPhone (Primary Key: {VendorID, AreaCode, Prefix, LineNumber})

Attribute	VendorID	AreaCode	Prefix	LineNumber
Domain	Foreign key, integer >= 1	Integer	Integer	Integer

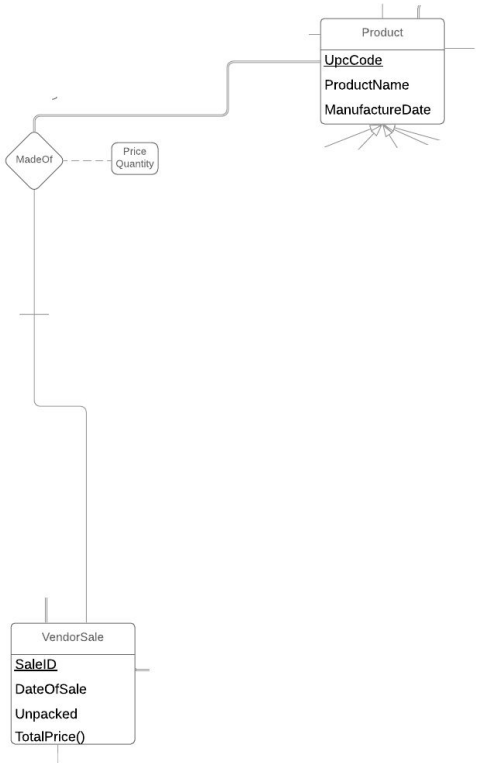
		Constraint: not null	Constraint: not null	Constraint: not null

- VendorSale (Primary Key: SaleID)



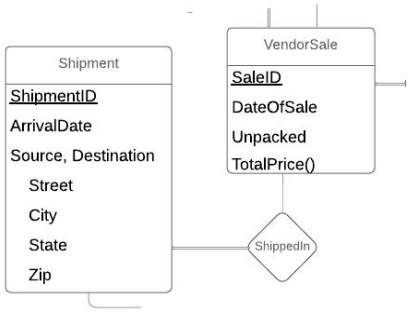
Attribute	SaleID	VendorID	StoreID	DateOfSale	Unpacked
Domain	Primary key, integer >= 1	Foreign key, integer >= 1	Foreign key, integer >= 1	Epoch timestamp (integer) Constraint: not null	True/False (boolean)

- VendorSaleProduct (Primary Key: {VendorSaleID, ProductUpcCode})



Attribute	VendorSaleID	ProductUpcCode	Price	Quantity
Domain	Foreign key, integer >= 1	Foreign key, integer >= 1	Integer > 0	Integer > 0#

- VendorSaleShippedIn (Primary Key: {SaleID, ShipmentID})



Attribute	SaleID	ShipmentID
Domain	Foreign key, integer >=1	Foreign key, integer >= 1

4. Sample of SQL queries from application

4.1. Getting the online store id

getOnlineStoreId method in Data.java.

```
SELECT StoreID FROM Store
WHERE StoreID NOT IN (SELECT StoreID from PhysicalStore)
```

Our store tables are arranged in a hierarchical structure with a Store supertable and PhysicalStore subtable. This means to differentiate between an online store and physical store, an online store simply does not have a corresponding row in the physical store table.

Thus by using a subquery to fetch all the store IDs in the PhysicalStore table, we can find the ID in the Store table that does not have a physical location.

4.2. Searching for products using case insensitive phrases

getBrowsingProducts method in Data.java.

```
WITH OnlineStoreInventory AS
  (SELECT * FROM StoreInventory WHERE StoreID = {Online Store ID})
SELECT * FROM OnlineStoreInventory
INNER JOIN Product
ON OnlineStoreInventory.ProductID = Product.ProductUPCCode
WHERE LOWER(ProductName) LIKE LOWER('%{Search Phrase}%');
```

Using the method to get the online store ID as described in the previous query, we can get every item stocked in the online store by simply doing a SELECT * where the store ID is equal

to the ID found previously. To clean up the overall query, a WITH statement is used to rename the table results.

It is important to note that the online store inventory simply references a product ID as a foreign key. Thus to get other information about products in the inventory, we must do an inner join. Since the two fields are named differently in each table, we must use ON where we specify each column.

Finally, to get only items resulting from a user entered search phrase, we do string comparison in the WHERE block, using % to mean the product name must only contain the phrase anywhere in it.

4.3. Categorizing products based on type of product

getBrowsingProducts method in Data.java.

```
WITH OnlineStoreInventory AS
  (SELECT * FROM StoreInventory WHERE StoreID = {Online Store ID}),
SpecificProduct AS
  (SELECT * FROM Product NATURAL JOIN {Product Table Name})
SELECT * FROM OnlineStoreInventory
INNER JOIN SpecificProduct
ON OnlineStoreInventory.ProductID = SpecificProduct.ProductUPCCode;
```

Getting the online store inventory is the same process as described in the previous query. Since our products are arranged into a hierarchical structure, with a Product super table and a specific subtable, e.g. FurnitureProduct, to get all the information about a product, we must NATURAL JOIN the upper and lower tables.

Now that we have the inventory (which only points to IDs and doesn't have the information about a product) and all product information based on the specified category, we can join the two tables, so based on a given inventory row, you have everything you need.

Since the fields we want to join on are different in both tables, we must specify each in the ON block at the end.

4.4. Updating frequent customer address

setOnlineCustomerAddress method in Data.java.

```
UPDATE FrequentCustomer
SET
Street = '{street}', City = '{city}',
```

```
State = '{state}',
Zip = {zip number}
WHERE CustomerID = {Frequent Customer ID};
```

This query is mainly used when a customer registers a new account online. It is also used on the customer profile page where the user is given the option to change any criteria listed on their profile, such as name, address, or add phone numbers.

Once the user enters all information they wish to update on their address, we can string format each field into the query, so the fields are set appropriately. To prevent updating every customer with this information, we must specify the customer ID in the WHERE block.

4.5. Adding a frequent customer phone number

addOnlineCustomerPhone method in Data.java.

```
INSERT INTO FrequentCustomerPhone
(CustomerId, AreaCode, Prefix, LineNumber)
VALUES
({Customer ID},
 {Customer Phone Area Code},
 {Customer Phone Prefix},
 {Customer Phone Line Number});
```

During the creation of a new online account, and during editing of an existing accounts profile, the user can add any number of phone numbers.

After they enter one long phone number, it is parsed using a regex and string formatted into the above query. In order to follow database best practices and make all of our fields atomic as possible, we store the phone number in three separate fields, namely area code, prefix, and line number.

4.6. Getting which vendor sells a specific item

Fire method in ReorderTrigger.java

```
SELECT * FROM Product
NATURAL JOIN Brand
NATURAL JOIN VendorBrand
NATURAL JOIN Vendor
WHERE ProductUPCCode = {Product UPC Code};
```

When a customer purchases items from a store, there is a chance that their purchase will lower the stores inventory quantity below their reorder threshold. When this happens, a trigger is fired. To create the sale, we must find out which vendor sells the item that dipped below its reorder threshold.

Natural joining each of these tables allows us to finally get which vendor sells it, so we can create a sale with their ID.

4.7. Viewing packed shipments that have arrived

getArrivedPackedSales method in Data.java.

```
SELECT *
FROM VendorSale
NATURAL JOIN
VendorSaleShippedIn
NATURAL JOIN
Shipment WHERE
ArrivalDate <= {Current timestamp} AND
    Unpacked = FALSE AND
    StoreID = {Current store id};
```

After a store inventory product dips below its reorder threshold and the update trigger inserts a new vendor sale, the vendor must go into the application to confirm the sale and generate a shipment.

Once a shipment arrival time is before the current time, a store has the ability to open, or ‘unpack’ the sale to increase their inventory.

By natural joining VendorSale with VendorSaleShippedIn and then Shipment, we can access fields that are spread out across each table. For example, Unpacked is stored in VendorSale, ArrivalDate is stored in Shipment, and VendorSaleShippedIn is the linking table that resulted from reducing out ER diagram.

4.8. Creating Packaging table

Used during creation/population of database through H2 browser.

```
CREATE TABLE PACKAGING(
    PACKAGINGID INT NOT NULL,
    HEIGHT FLOAT(5) NOT NULL,
    WIDTH FLOAT(5) NOT NULL,
```

```
    DEPTH FLOAT(5) NOT NULL,
    COLOR VARCHAR(10),
    MATERIALNAME VARCHAR(20) NOT NULL,
    PRIMARY KEY(PACKAGINGID),
    FOREIGN KEY(MATERIALNAME) REFERENCES MATERIAL(MATERIALNAME),
    CHECK(HEIGHT > 0.0),
    CHECK(WIDTH > 0.0),
    CHECK(DEPTH > 0.0)
);
```

The packaging table has five main attributes, namely height, width, depth, color, and material name. By specifying NOT NULL on each field, we are saying that these fields must have a value when inserting or updating a row. This makes sense because packaging in real life **must** have dimensions, however, if packaging is transparent, such as plastic, then it can be argued that it does not have any color and is thus null, which is why null is allowed for color.

MaterialName is a foreign key because there is a separate Material table. This table contains other information specific to material that would not make sense in the packaging table. For example RarenessRating is a measure of how rare a material is, if for example something was made out of gold, it has a high rareness rating.

The constraints at the bottom make sense in real life as well because any packaging has non-zero dimensions, even if it is a piece of paper, it is just small.

4.9. Creating FurnitureProduct table

Used during creation/population of database through H2 browser.

```
CREATE TABLE FURNITUREPRODUCT(
    PRODUCTUPCCODE INT NOT NULL,
    PURPOSE VARCHAR(20),
    COMFORTRATING INT,
    INTENDEDPLACE VARCHAR(30),
    MATERIALNAME VARCHAR(20) NOT NULL,
    PRIMARY KEY(PRODUCTUPCCODE),
    FOREIGN KEY(MATERIALNAME) REFERENCES MATERIAL(MATERIALNAME),
    CHECK(COMFORTRATING > -1),
    CHECK(COMFORTRATING < 11)
);
```

The NOT NULL tags in the creation of this table are similar reasoning as the SQL statement in the section above, as well as the MaterialName field being a foreign key into the Material table.

The constraints at the bottom specify the ComfortRating field must be between 0 and 10. This makes sense in the real world because many things are rated on a finite scale, e.g., Amazon reviews (except these are 1-5).

5. User interface

5.1. Description (with screenshot)

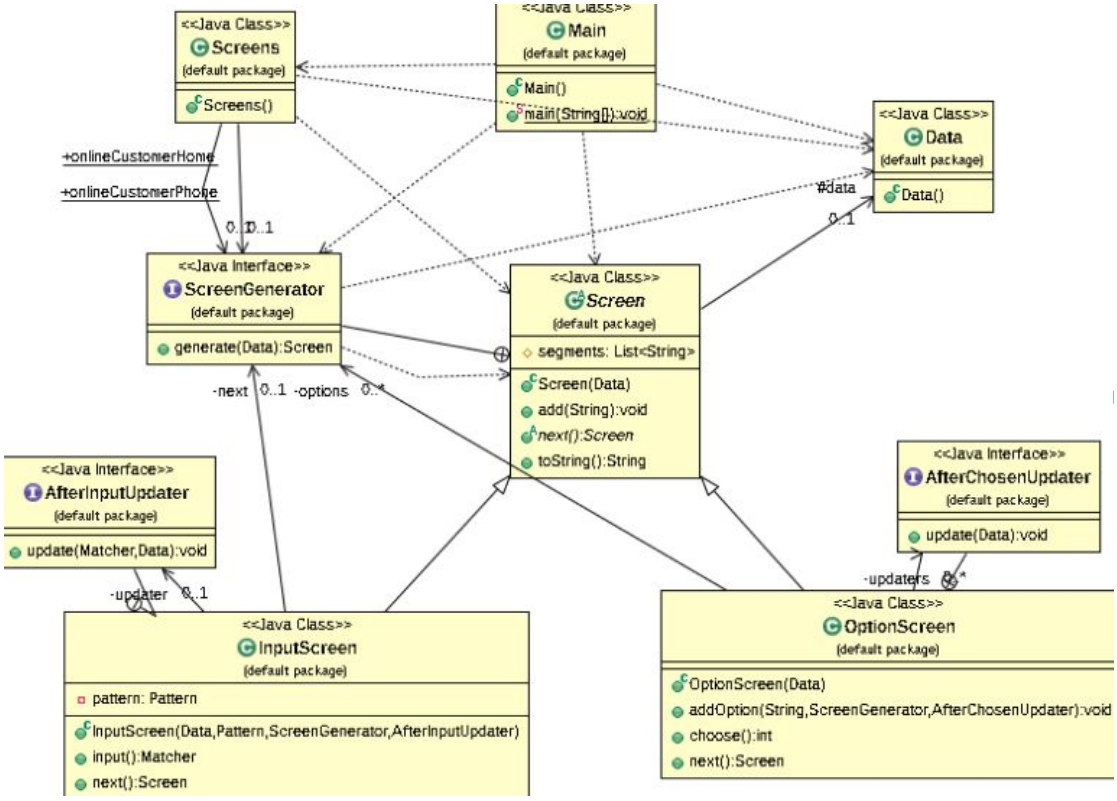
The user interface of the application is command line based. There are two possible types of screens the user can interact with. An “Option Screen”, where the user is presented with 2 or more options. A selection is made by typing in the corresponding number for the desired option, then enter. The second type is an “Input Screen”, where the user is presented with a prompt to provide text input. The input provided is checked against a regular expression pattern for validity, and then if valid, is used for the appropriate database interaction. Refer to the following diagrams for examples of the user interface.

```
Retail Center

1) Administrator
2) Online Customer
3) Vendor
4) Unpack Arrived Shipments
5) Checkout Register

Choose an application: 6
Option must be between 1 and 5. Please try again.
2
Welcome,

1) New customer sign-up
2) Existing customer login

Please choose an option: 1
Enter first and last name (e.g. Bob Evans): Corey Clow
Enter address in format <Street, City, State, ZIP> (e.g. 907 William Street, Lake Villa, IL, 60046): NOT AN ADDRESS FORMAT
Input does not match specified format. Please try again.
```

5.2. Underlying structure and DB interaction (with UML diagram)

The driver class of the application “Main” holds the state of the application by having the current displayed “Screen” object, and continually forwards the state by repeatedly calling the “next” method on Screen. Within the Screen being displayed (either an InputScreen or OptionScreen) the next Screen to be transitioned to after it finishes is defined.

Shared between all Screen is the “Data” object. This object contains a direct connection to the H2 Database for the application. The object exposes certain abstracted functionality on the Database and internally performs the necessary SQL operations. This contains all actual database interactions within the Data class and provides an easy to understand API for the Screens.



5.3. Users and use cases

The following are the four possible users of the application.

5.3.1. Administrator

This user has the ability to run any SQL query through the command line. Because of this, the application is simply a pass-through where whatever they enter is executed. This means they have the most power of all users and can insert rows, select columns, delete rows, drop tables, etc.

```
Retail Center

1) Administrator
2) Online Customer
3) Vendor
4) Unpack Arrived Shipments
5) Checkout Register

Choose an application: 1
Enter SQL statement: SELECT * FROM Product WHERE ProductName LIKE '%tes%'

PRODUCTUPCCODE, PRODUCTNAME, MANUFACTUREDATE, PACKAGINGID, BRANDID
9, Wine - Peller Estates Late, 1527290271, 28, 36
60, Neomycin Polymyxin B Sulfates and Dexamethasone, 1544563411, 43, 44

Enter SQL statement: NOT A VALID SQL STATEMENT
Syntax error in SQL statement "NOT[*] A VALID SQL STATEMENT"; SQL statement:
NOT A VALID SQL STATEMENT [42000-199] Please try again.
```

5.3.2. Online Customer

This user has the ability to:

- Create an account
- Manage their account when they have one
- View available products
- Manage their cart
- Make purchases

The data associated with each of these actions:

5.3.2.1. Account Creation

Provides: first and last name, address (Street, city, state, zip), email address, phone number (optional)

```
Retail Center

1) Administrator
2) Online Customer
3) Vendor
4) Unpack Arrived Shipments
5) Checkout Register

Choose an application: 2
Welcome,

1) New customer sign-up
2) Existing customer login

Please choose an option: 1
Enter first and last name (e.g. Bob Evans): Corey Clow
Enter address in format <Street, City, State, ZIP> (e.g. 907 William Street, Lake Villa, IL, 60046): 907 William Street, Lake Villa, IL, 60046
Enter email address: cmc4533@rit.edu
You may choose to add any number of phone numbers.

1) Add phone number
2) Finish

Please choose an option: 1
Enter phone number (e.g. 123-456-7890): 123-123-1234
```

5.3.2.2. Existing Account Management

Provides: Same as account creation, however, the user picks which one they want to provide
Gets: All of their current information

```
Welcome,

1) New customer sign-up
2) Existing customer login

Please choose an option: 2
Enter your customer ID: 12
Hello, Morissa Farlambe, ID: 12. You have 78 frequent shopper points.

1) View cart / checkout
2) View product categories
3) Search for product
4) View customer information
5) Log out

Please choose an option:
```

5.3.2.3. View Available Products

Provides: Either -> Search phrase or Product category (one of: Cleaning supplies, Medicine, Food, Furniture, Clothes)
Gets: Requested product(s) and all of the data associated with it (price, manufacturer date, etc.. differs for product types)

```
Hello, Pearle Kettlestring, ID: 23. You have 0 frequent shopper points.

1) View cart / checkout
2) View product categories
3) Search for product
4) View customer information
5) Log out

Please choose an option: 3
Enter product search phrase: tes
Available products:

1) Wine - Peller Estates Late                          $48.00          x221
2) Neomycin Polymyxin B Sulfates and Dexamethasone     $44.00          x23

Select a product to add to your cart, or alternatively,

3) Search for product
4) View product categories
5) Return home

Choose an option: 4
The following product categories are available:

1) Cleaning supplies
2) Medicine
3) Food
4) Furniture
5) Clothing

Choose the category you wish to view:
```

5.3.2.4. Manage Cart

Provides: Product UPC for adding
Gets: Details of product(s) currently in the cart, and total price

```
You have 2 distinct products currently in your cart:

Product Name                                    Price                                   Quantity
1) Chocolate Eclairs                            $54.00                                   x13
2) Colour Catcher                               $10.00                                   x15

Total price: $852.00

Select a product from the cart, or alternatively,

3) Checkout
4) Return home

Please choose an option: 3
Use points towards purchase?

1) Yes
2) No

Please choose an option:
```

5.3.2.5. Make Purchase

Provides: Decision to use points or not

Gets: Order summary, similar to cart view

```
Use points towards purchase?

1) Yes
2) No

Please choose an option: 1
Enter card number (16-digits): 1231231231231231
Thank you for shopping with us!
Pearle Kettlestring, your order has been placed and will ship to your address on file.

Order Summary:
Product Name                                    Price                                   Quantity
Chocolate Eclairs                               $54.000000                               x13
Colour Catcher                                  $10.000000                               x15

Total price: $852.00
Press enter to return home:
```

5.3.3. Checkout Register

This user has the ability to

- Make sales for both frequent shoppers and anonymous customers
- Get a customer's information

The data associated with each of these actions:

```
Retail Center

1) Administrator
2) Online Customer
3) Vendor
4) Unpack Arrived Shipments
5) Checkout Register

Choose an application: 5
Enter your store ID: 20
Welcome,

1) Create frequent shopper sale
2) Create anonymous sale

Please choose an option: 1
Enter Frequent shopper ID: 20
Welcome Stafford Hasluck!
Customer ID: 20
Points: 102

1) View cart / checkout
2) Scan product to cart
3) View Customer Information
4) Cancel checkout

Please choose an option: 2
Enter UPC code: 191
```

5.3.3.1. Make a Sale

Identical to Online Customer making a purchase, except the register, just provides a UPC code to add to a sale, and after the sale is confirmed the registered user has to indicate the payment type (one of: credit, cash, check) and enter the corresponding details. Also, the registered user only has to provide a customer ID if it is a frequent shopper sale.

```
You have 1 distinct products currently in your cart:

Product Name                                    Price                                   Quantity
1) Cufflinks                                    $7349.00                                 x1

Total price: $7349.00

Select a product from the cart, or alternatively,

2) Checkout
3) Return home

Please choose an option:
```

5.3.3.2. Get Frequent Shopper/Registered Customer Information

Provides: Desired customer ID
Gets: All information associated with this customer (identical to the response from online customer application)

5.3.4. Vendor

This user is able to:

- View unshipped order requests, orders in transit, completed orders, or just all orders.

Each processed and delivered order has the following data associated with it: SaleID, StoreID, Full Address (street, city, state, zip), Date of Sale, Date of Delivery, a total price, and an array (1 or greater length) of products making up this sale, which each have a product name, quantity, and price.

Order Requests have all of the above data except for a date of sale, and date of delivery.

In Transit, orders have all of the above data except for a date of delivery.

All of this data can only be interacted with in a read-only way, except for Order Requests, where the vendor user has the ability to either confirm or reject the request.

The following screenshot is an image of a vendor logging in, and then viewing an overview of their orders that need to be processed. All 3 screens that have order overviews of different

subsets of orders have the same visual structure, just with different text.

```
1) Administrator
2) Online Customer
3) Vendor
4) Unpack Arrived Shipments
5) Checkout Register

Choose an application: 3
Welcome, enter vendor ID: 5
Welcome Andonis Traice, ID: 5.
1) View unshipped order requests
2) View orders in transit
3) View All Orders
4) Log out1
Orders Waitng to Be Processed:

1) Sale ID: 192
For Store: 16 Located at 298 Novick Drive Lawrenceville, GA 86478

Summary Of Products Purchased:
Product: Crackers - Graham
Purchased 93 at $14 each

Total Price: 1302

2) Sale ID: 198
For Store: 42 Located at 10 Merchant Way Fort Wayne, IN 3844

Summary Of Products Purchased:
Product: Figs
Purchased 62 at $90 each

Total Price: 5580

3) Sale ID: 210
For Store: 16 Located at 298 Novick Drive Lawrenceville, GA 86478

Summary Of Products Purchased:
Product: Calypso - Pineapple Passion
Purchased 13 at $55 each

Total Price: 715

4) Sale ID: 249
For Store: 31 Located at 685 Luster Road Providence, RI 40470

Summary Of Products Purchased:
Product: Calypso - Pineapple Passion
Purchased 97 at $117 each

Total Price: 11349

5) Return Home
Select an order to interact with.
```

5.3.5. Others

Other users/features include the following:

- Someone who periodically runs the application that records the increase in inventory resulting from unpacking an arrived shipment. In the real world, this maps to someone who is responsible for grabbing e.g. a Fedex delivery and opening it.
- Triggers that generate VendorSale entries when a store inventory item drops below its reorder threshold.

6. Overall process reflection

6.1. What went well

Our group usually met on Wednesdays and Fridays at 4:00PM. On weeks where there wasn't anything due, we either decided in class nothing was urgent enough to meet, or we met anyways and started planning out how we would complete the assignment on time. This included items such as:

- Design
    - Database
    - Application
- Reviewing what needs to be completed for the next assignment
    - Making sure everyone understood each item
    - Distributing the work between individuals
    - Ensuring everything will be finished on time
- Helping each other if anyone was stuck

Everyone completed their work on time and we communicated well through Slack.

6.2. What we worked through

We did not face any interpersonal issues that needed to be worked through. We all worked well together and understood each others workload.

The one problem we faced technology-wise was first getting our populated database to work on different computers via the H2 browser and our Java project. After populating the database using the H2 browser and CSVREAD statements, we found that copying the "retail.mv.db" file to another computer resulted in a blank database with no tables after opening it in the H2 browser.

After much debugging, we discovered this was because we copied the database file while it was still opened by H2. This meant that the file was in an inconsistent state and had some sort of lock on it that did not transfer well to other computers.

After closing all H2 processes and copying the file from the original computer again, this time we received a "database corrupt" error message. This was finally solved by using H2's Recovery and RunScript tools that converted the binary file into a human-readable sequence of SQL statements that could be used to build the database from scratch on another computer.

6.3. What we learned

Over the process of finishing this project, we learned the following things:

- How to construct SQL queries
- How to use H2 with Java
- How to make a robust command line application that interacts with a database
- Improved technical writing from the reports (concise, informative, to-the-point)
- Some information about the retail domain, e.g. UPC codes, etc.

7. Contributions

- Database design: All
- Reports: All
- Application design: Corey Clow, Zach Morgan
- UI Storyboards: Corey Clow
- Implementation of command line engine: Corey Clow
- Phase 1 CSV file: Zach Morgan
- Phase 2 CSV files: Himani Munshi, Tejaswini Jagtap
- Populating database: Himani Munshi, Tejaswini Jagtap
- Online customer sign-up, home, information: Corey Clow
- Online customer cart: Himani Munshi, Tejaswini Jagtap
- Checkout Register Application: Himani Munshi, Tejaswini Jagtap
- Vendor application: Zach Morgan