Wegmans 2 Database Report

Ryan Cervantes, Jorge Flores, Jarod Godlewski, Spencer Lowitt

Progress

- E-R/ Relational Design
    - Our overarching design was kept mostly the same. During phase 2 we found that calculated values did not need to stored in the database, as they could just be created with SQL queries as needed. As such, we did not need to overcomplicate the design. However, there were a few tweaks done here and there in order to add more information and functionality to the program, which will be detailed later in the report. The only significant change was the addition of the “Admin” table, which we made in order to verify the credentials of users attempting to make changes to the database.
- Postgres Databases
    - By phase 3 our SQL database was completed, and most of the typical, smaller updates we needed to make could be completed through the application. However, when bigger bulk changes needed to be made, such as adding dummy data for testing, we were able to implement scripts to create several thousand entries without needing to enter them all by hand. This was made possible through our understanding of how JBDC interacted with postrgres in the backend.
- Data used in database
    - In order to accomodate our new Administrators table, we created three administrators with usernames and passwords so that they can be authenticated when logging in. There were also a few errors in the way the data was written out in order for it to be copied into our database, so this was also fixed. We also randomly generated data for testing which we were able to insert and remove from the database at will.
- Application concept
    - The end of phase 2 marked the solidification of our application’s concept and design. In this phase, we did not change any of the details of the application from phase 2, instead we added more functionality in order to make the application more robust in terms of what each user of the application can do.
        - Administrators
            - Query the best 3 items across stores or in a particular store
            - Query the worst 3 items across stores or in a particular store
            - Get the customer who has spent the most money
            - Query the worst store in sales
            - Query the best store in sales
            - Registering customers to the database
- Application Implementation
    - Main implementation of the application to place in this phase. We hooked up the database to the application, implemented the command line parser and created

    functions to house the queries we wrote in previous phases. We also added authentication for both customers and administrators.

  - In order to more effectively implement the command-line we used the 3rd party libraries:
    - Picocli - use to implement the parsing of commands.
    - Apache-ant - used for parsing strings as if they were inputs into the command line
    - Postgresql - used to drive the JBDC functions we used

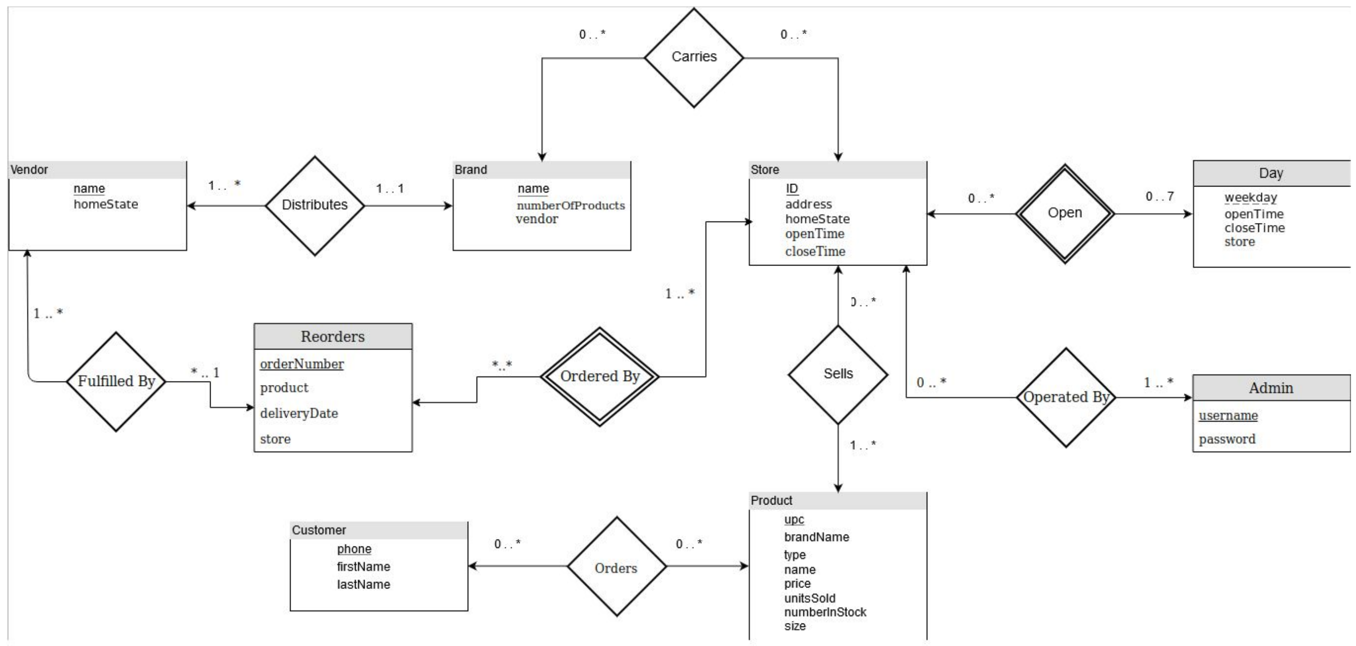
Description

This report will outline our project for the prestigious Wegmans2 company. Wegmans2 is a growing company and needs to upgrade their business to a more modern system. As such, they have hired us to create a database that will track multiple aspects of their day-to-day administrative and retail factors. Additionally, they have asked us to create a programmer-friend command-line interface for this backend.

As specified by Wegmans2, the database must keep track of retail and administrative factors of their business. For the retail aspects, we were tasked to keep track of customers and any items they have purchased through our application. For the administrative aspects, we were tasked to keep track of information about physical stores (i.e. address, open time, close time), the items they sell (name, price, brand, vendors, number in stock), and restocking (items requested, when they were delivered, what store requested them).

The application itself must have a dual purpose. First, it must serve as a customer portal through which customers are able to purchase items. This includes all the intermediary steps of a larger company's online presence such as logging in, choosing a store, browsing inventory, adding items to a cart, and checking out. Secondly, it must serve as an administrative hub where by administrators may modify the operations of wegmans. This includes things such as updating item prices, adding/removing items to a store's inventory, requesting reorders, and approving reorders. Finally, Wegmans2 has also asked us to make the application have the ability to gather business statistics about their business. This will include things such as finding the MVP customer, finding the best/worst stores, and finding the worst/best items.

Database Design

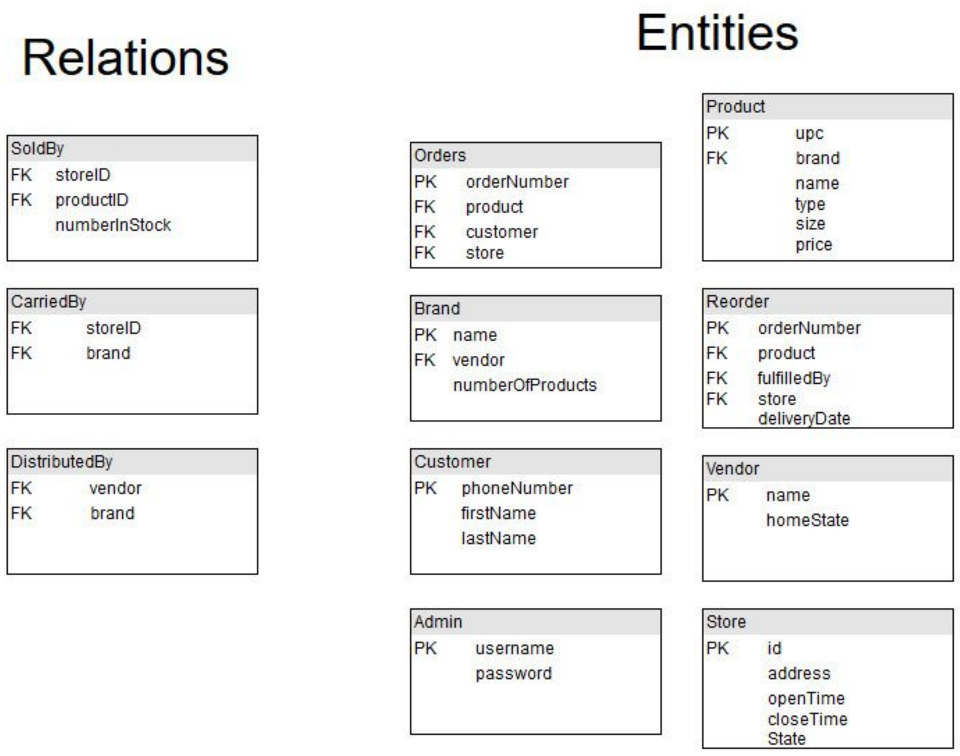


The above E-R diagram models the different entities within the domain of the application. We designed this application with 6 entities that would interact with each other to yield the desired functionality: Store, Product, Brand, Vendor, and Customer. From there, we could use different relationships to describe how the entities interacted with one another. The general idea of the domain is as follows:

"A **Store** <u>sells</u> multiple **Products**, which can be bought by a **Customer.** Each product is carried by a **Brand** and brands are <u>distributed</u> by a **Vendor**. When a store runs out of a specific product, an administrator may have a store <u>order</u> a **Reorder** which will then be fulfilled.

<u>Underlined</u> - weak entity set
Bold - strong entity set

Reduction of ER Diagram to Relational Database Tables



Relationships:

- *SoldBy:*
  - **storeID** - an 8 character value to define which store has the product in stock [References Store(id)]
    productID - an identifier for the product, using the upc [References Product(upc)]
    numberInStock - an integer keeping track of how many of each product is available at the given store
  - This table was created from the “Sells” relationship between Store and Product. It is the table that essentially keeps track of each store’s inventory. As the numberInStock attribute from the ER diagram in Product would require each instance of a product to also have a connection to each store, it was moved to this relation and subsequently this table. This way, we can keep track of not just which stores sell which product, but also how many of each are at each store.
- *CarriedBy*

- **storeID** - an 8 character value to define which store carries said brand [References Store(id)]
  brand - a 50 character character field that represents the name of the brand [References Brand(name)]
- This table was created from the “Carries” relationship between Store and Brand. It is a way to keep track of which brands can be found at which stores, and so it is a simple adaptation using two foreign keys to connect to the two tables.

- *Orders*
  - **orderNumber** - an internally generated 10 character string that represents the order number of a particular order each time a customer purchases something [Primary Key]
    customer - the phone number of the customer who purchased this order [References Customer(phoneNumber)]
    product - the upc of one of the products purchased in this order [References Product(upc)]
    store - the ID of the store that these products were ordered from [References Store(id)]
  - The Orders table was created from the “Orders” relationship between customer and product in the ER diagram. As we need to keep track of past orders anywhere in the system, this table is able to hold not only when and where an item was purchased, but also who purchased it. We can also use this table to calculate things such as popular products, total units sold, which stores sell the most product, etc.

- *DistrbutedBy*
  - **vendor** - the name of the vendor that distributes this brand, as a max 50 character string [References Vendor(name)]
    brand - the brand that is being distributed by said vendor, as a max 50 character string [References Brand(name)]
  -

Entities:

- *Store:*
  - **id** - a set of 8 numbers help by 8 characters to uniquely identify a store [Primary Key]
    address - a physical location for the store, represented by a maximum 50 characters
    openTime - a 4 character integer, that represents the opening time in 24 hour format
    closeTime - a 4 character integer, that represents the closing time in 24 hour format
    state - the state that the store is in, represented by 2 characters (e.g. CA, NY)
  - This table was created directly from the Store entity in the ER diagram. The open and close time columns are integers in order to easily calculate the amount of hours a store is open.
- *Brand*
  - **name** - the name of the brand, maxed out at 50 characters [Primary Key]
    numberOfProducts - an integer representing the amount of products under this brand
    vendor - the name of the vendor that distributes this brand [References vendor(name)]

    - The brand table was created directly from the Brand entity, in a trivial one-to-one conversion. It references the Vendor table in order to ensure that it can be distributed properly.
- *Customer*
    - **phoneNumber** - a 10 character string representing the customer's unique phone number [Primary Key]
      firstName - the customer's first name, maxing out at 50 characters.
      lastName - the customer's last name, maxing out at 50 characters.
    - The customer table was created directly from the Customer entity in the ER diagram, in a trivial one-to-one conversion.
- *Product*
    - **upc** - a 12 character string of numbers that uniquely identifies each product [Product Key]
      brand - name of the brand that creates the product [References Brand(name)]
      name - the standard name of the product for usual information, maxes out at 50 characters
      type - the type of product (e.g., meat, beverage, etc.), 50 character limit
      size - the physical size of the product's packaging: small, medium, or large
      price - a double precision that represents the cost of the item, standard across all locations and rounded to two decimal places.
    - The product table was created mostly from the product entity in the ER diagram. The numberInStock attribute was moved to the soldBy relationship.
- *Reorder*
    - **orderNumber** - a generated 8 character identifier for the reorder [Primary Key]
      product - upc of the product that is being requested [References Product(upc)]
      store - the ID of the store making the order [References Store(id)]
      deliveryDate - the date that the reorder was fulfilled by the vendor; if null, it is an unfulfilled order, uses SQL Date datatype.
      fulfilledBy
    - The Reorder table was created from the Reorder entity in the ER Diagram. Since we need to keep track of both the current orders that stores are making, as well as keep a history of past reorders, we added a delivery date attribute. If it is null, then we'll know that it is an unfilled order. Otherwise, we can sort by delivery date as well as other constraints.
- *Vendor*
    - **name** - the name of the vendor, a 50 character max string [Primary Key]
      homeState - a 2 character string that represents the state the vendor is based in.
    - The Vendor table was created directly from the Vendor entity in the ER diagram, in a trivial one-to-one conversion.
- *Admin*
    - **username** - the username of the administrator [Primary Key]
      password - the password of the administrator

  - The admin table was added to the diagram in phase 3, in order to authenticate users trying to edit the database.

The main entity sets are all contained within their own tables, as well as all the relations; the relation "Ordered by" relation is captured by reorders table. Furthermore, this table interacts with the Vendors table, which will query the reorders table for reorders to fulfill, and then stamp it with a delivery date and then update the data in the database, in order to show that the order has been completed without losing the history of the order. Additionally, we have created indexes on the following:

- Order by Customer

  -
    ```
    CREATE INDEX cust_index
      ON orders(customer);
    ```
  - Admin queries often want to get information on past reorders by customer, so this index is useful.
- Product by Brand

  -
    ```
    CREATE INDEX brand_ind
      ON product(brand);
    ```
  - Customers wanting to get information about a certain store's products by a brand that they like will be able to take advantage of this index. It also helps when admins are requesting reorders for certain products.
- Product by Type

  -
    ```
    CREATE INDEX type_index
      ON product(type);
    ```
  - Similarly, customers will be able to search for a certain product they're looking for, such as snacks or cleaning supplies
- Reorder by DeliveryDate

  -
    ```
    CREATE INDEX reorder_by_date
      ON reorder(deliverydate);
    ```
  - This index is useful for the fulfillment of reorders. As the delivery date not existing is the way that the unfulfilled orders is determined, being able to quickly get the ones that do exist allows for faster execution of the fulfillment methods.
- SoldBy by Product ID and Store ID

  -
    ```
    CREATE INDEX soldby_index
      ON soldby(productid, storeid);
    ```
  - This table has the most queries associated with it, as it handles things such as finding stock number, which stores sell what products, etc. As there are no queries that use either the store ID or the product ID, instead of using both at the same time, having the index on both covers almost all soldBy queries we use. This is helpful as they are also the most common queries. They are also useful for finding calculated values.

Description of Application

For our application are planning on creating a supermarket shopping application. Our application will mimic two sides of operating a retail business: administration and customer interaction. A user will be able to access certain features based on the account with which he/she will use to sign in. Customer's will have a limited access to the information presented in the database compared to that of an administrator because they will only need to see information relevant to buying items. On the other hand, an administrator will have a wide range of actions and information relating to the management of stores, products, vendors, and brands.

Use Case #1: The Customer

For our application we chose to recreate an online shopping experience via the command line. When shopping online, any given user can do a multitude of actions in order to help them purchase items online. As an example, we have the following scenario.

"Bob lives in Massachusetts and does not have time to drive to Wegmans to get groceries. He chooses instead to find a store online and purchase his items online. He wants to find milk, eggs, and bread."

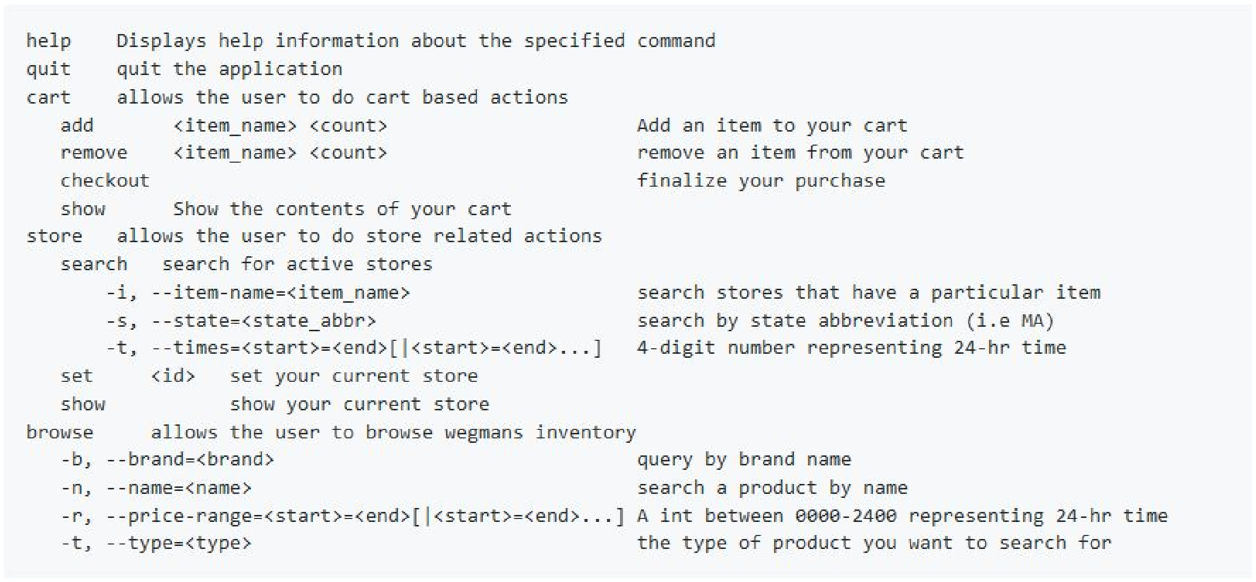
In this specific use case, there is only a limited amount of information a user is able to see. When logged on the user will only be able to take actions that are relevant to his/her shopping experience.

In order to accomplish his goal of buying his items, he will need to do the following:
- Log in
- Find a store in his area
- Browse the store's inventory
- Add items to his shopping cart
- Purchase the items

So, Bob will only be able to run commands relevant to each of the steps in his process of buying items and, therefore, will be relegated to certain data in the database. More generally, users that have an account will be able to see the following information:
- Each store's name, state, street address', hours of operation and inventory
- Each Products upc, brand name, price, type, and if the item is in stock
- Their cart's inventory, current total

Notice that information is intentionally left out of the customers view such as products' the vendor, the number of items a store has in stock. Also, customers will not be able to see certain relations such as reorders, or vendor sales because it is not information a customer would need to know when shopping.

Detailed Command Outline (Customer)



```
help    Displays help information about the specified command
quit    quit the application
cart    allows the user to do cart based actions
   add       <item_name> <count>                     Add an item to your cart
   remove    <item_name> <count>                     remove an item from your cart
   checkout                                          finalize your purchase
   show      Show the contents of your cart
store   allows the user to do store related actions
   search   search for active stores
       -i, --item-name=<item_name>                   search stores that have a particular item
       -s, --state=<state_abbr>                      search by state abbreviation (i.e MA)
       -t, --times=<start>=<end>[|<start>=<end>...]   4-digit number representing 24-hr time
   set     <id>   set your current store
   show           show your current store
browse     allows the user to browse wegmans inventory
   -b, --brand=<brand>                               query by brand name
   -n, --name=<name>                                 search a product by name
   -r, --price-range=<start>=<end>[|<start>=<end>...] A int between 0000-2400 representing 24-hr time
   -t, --type=<type>                                 the type of product you want to search for
```

Use Case #2: The Administrator

Our application is designed to deal with two types of users. First is the typical user or customer that will use the application for everyday shopping needs. However, the second is the administrator that will deal with the inner workings of the stores. These users will be able to manipulate data need to run a store as well as retrieve data as if they were going to report numbers in a sales report. They will manually be able to restock inventory, remove or add items to the inventory of the stores, change the prices of items in the store, and retrieve metrics about how stores are doing.

“*Bob has been hired as a store admin for Wegmans2 and needs complete tasks for his boss. He needs to update prices of “healthcare” items to reflect a 50% off sale across all stores. Also, his boss has noticed that they have run out of Wegmans brand soap in many stores and would like him to order more from the vendors. Also, the boss’s cousin needs to be registered into the online shopping program”*

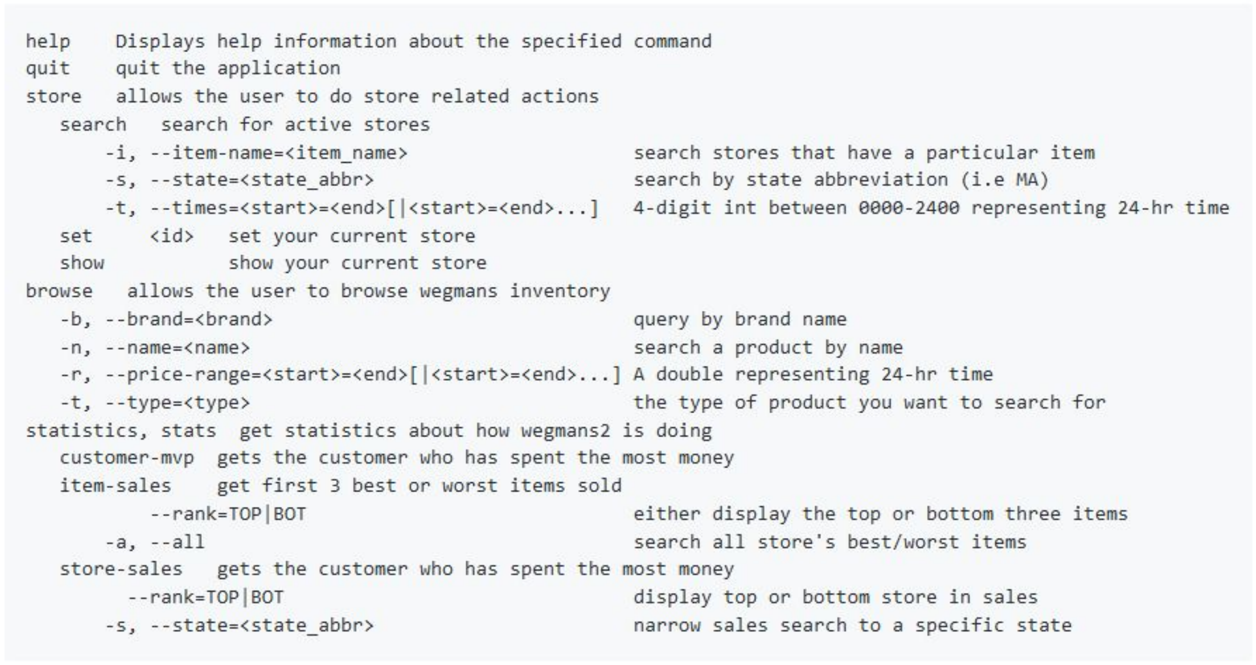
An example list of actions that an admin will be able to perform is the following.

- Log in under an admin prompt
- Select a store
- Remove an item from a store’s inventory
- Add an item to a store’s inventory

- Update a price of an item across all stores
- Request a reorder
- Fulfill a reorder
- Register a customer
- Find the MVP Customer
- Find 3 best/worst selling stores
- Find 3 best/worst selling items across stores
- Find 3 best/worst selling items in a single store

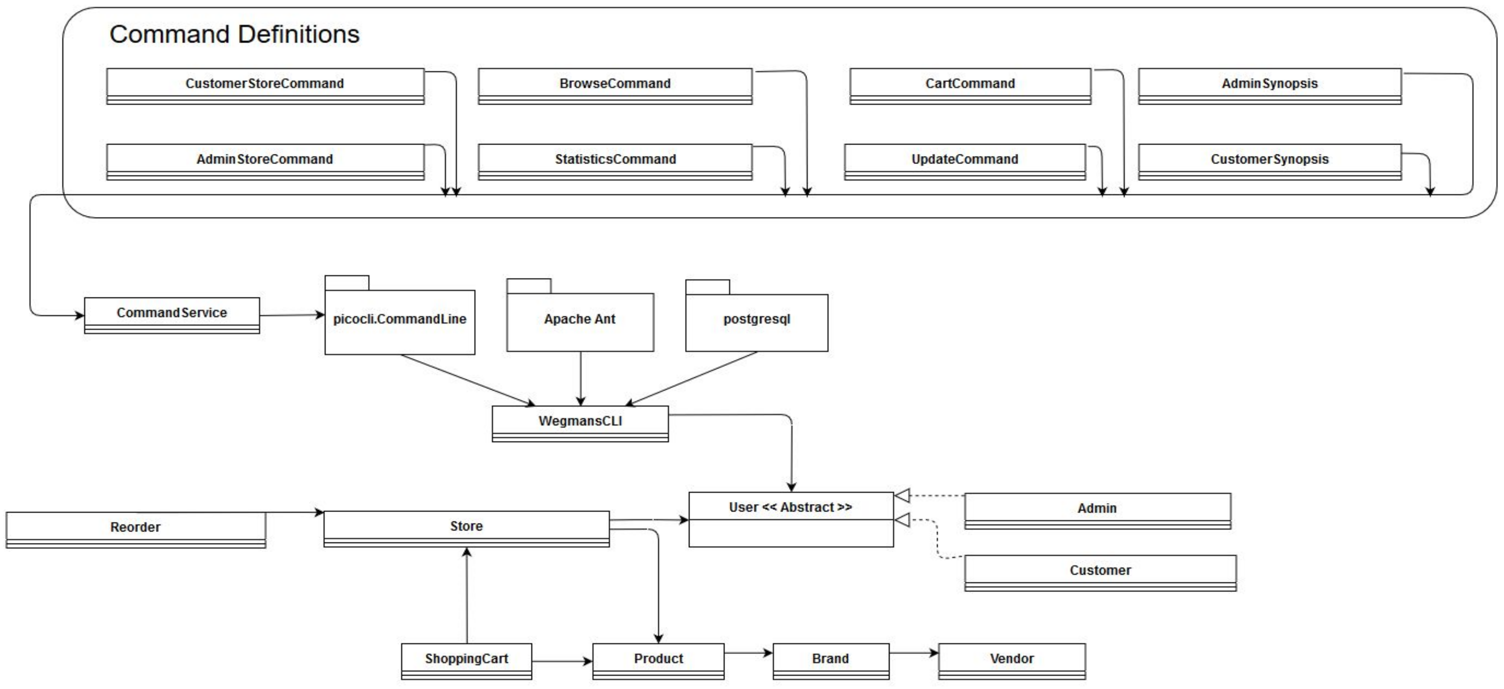
In the case of the administrator, there will be a lot more data that they will be able manipulate and see. However, he most poignant difference between the two, however is noted by the list of actions that they can take. Since they are tasked with updating values within stores' inventory, prices, and initiating reorders, they will be able to see all the backend information that will be hidden from normal customers such as the vendors, reorder history, registered customer list, etc.

Detailed Command Outline (Admin)



```
help    Displays help information about the specified command
quit    quit the application
store   allows the user to do store related actions
   search   search for active stores
       -i, --item-name=<item_name>                  search stores that have a particular item
       -s, --state=<state_abbr>                     search by state abbreviation (i.e MA)
       -t, --times=<start>=<end>[|<start>=<end>...]   4-digit int between 0000-2400 representing 24-hr time
   set     <id>   set your current store
   show           show your current store
browse   allows the user to browse wegmans inventory
   -b, --brand=<brand>                              query by brand name
   -n, --name=<name>                                search a product by name
   -r, --price-range=<start>=<end>[|<start>=<end>...] A double representing 24-hr time
   -t, --type=<type>                                the type of product you want to search for
statistics, stats  get statistics about how wegmans2 is doing
   customer-mvp  gets the customer who has spent the most money
   item-sales    get first 3 best or worst items sold
           --rank=TOP|BOT                           either display the top or bottom three items
       -a, --all                                    search all store's best/worst items
   store-sales   gets the customer who has spent the most money
         --rank=TOP|BOT                             display top or bottom store in sales
       -s, --state=<state_abbr>                     narrow sales search to a specific state
```

UML Diagram for the Program (Collapsed)



The program's architecture has been redesigned from Phase 2, making it more streamlined. Seen above is a condensed version of the UML for out application that doesn't show methods and attributes. These can be seen with the attached PDF version of the UML.

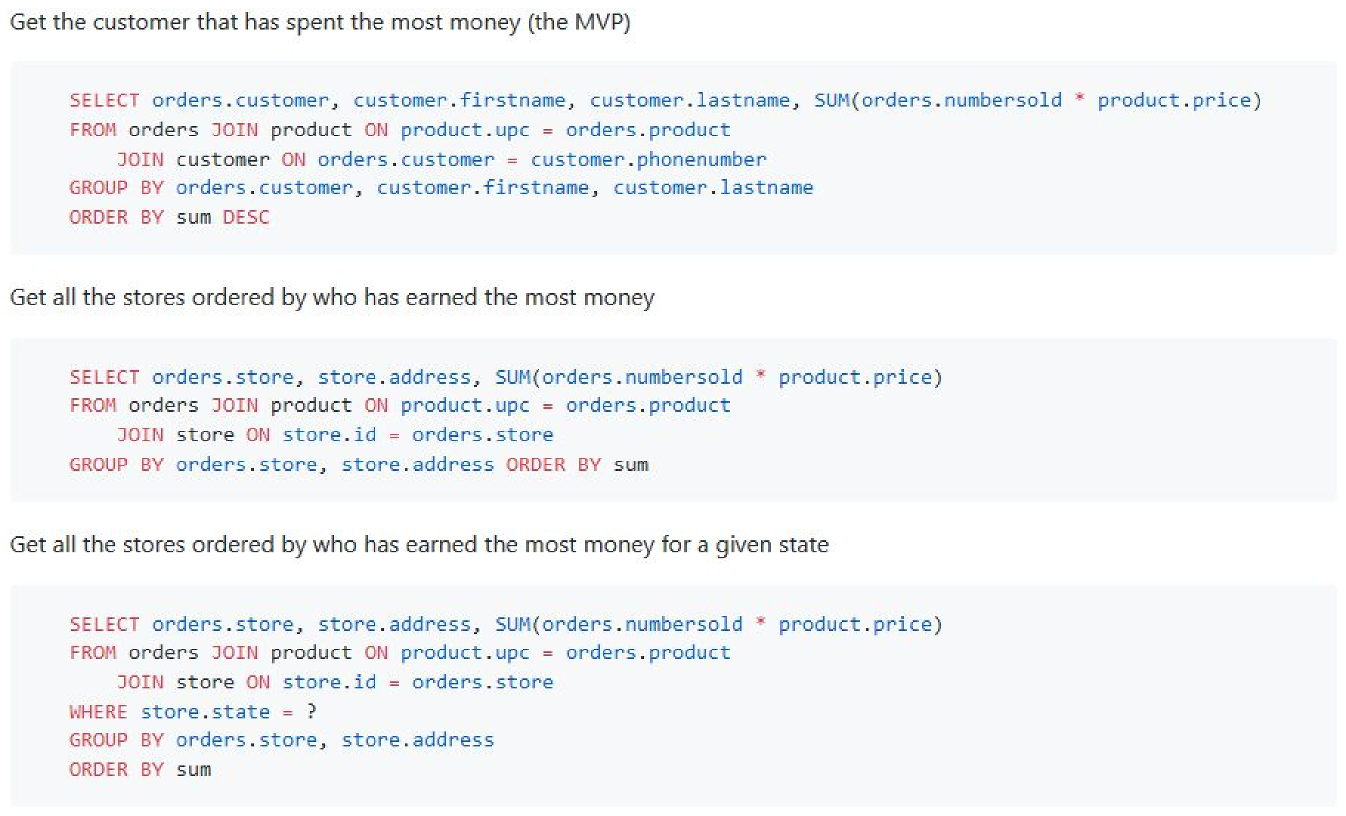
The biggest difference here is the use of the new "Command" classes that are utilized by the "picocli" package. Picocli is a command line interface creation tool that allows users to declare parsing rules and execution rules for a program. Usually this program is used for regular command line tools where only one command is executed, our program uses this continually after a user inputs their command, thus creating a custom, continuous application for the user. Each of the commands, subcommands, and options map to specific queries within the User, Customer, and Admin classes. When any of the commands shown above is run, the proper parameters are passed so that the query can be run.

Similar to Phase 2, the segmentation of functionality lies with the User, Customer, and Admin classes. Each subclass (Customer and Admin) inherit most functionality from the User's shared set of methods (i.e browsing a store's, inventory, searching for stores). Customers will be able to add things to a shopping cart, while Admins may not. On the other hands, Admins may perform a host of different actions such as running statistical queries, adding/removing customers, updating prices etc.

Finally, we have our "value objects" which are the reorder, store, product, brand, and vendor classes. These classes allow us to programmatically use the results from the postgres database. Instead of manipulating large batches of strings, we convert them into their respective Java Objects to more easily pass information throughout our program.

SQL Queries

The following queries are ones are a subset of queries that can be used by an administrator. Specifically, these are queries that allow administrators to gain certain statistical information about how Wegmans2 is doing. A list of full SQL queries can be found in the Git Repository's README.md file.



Get the customer that has spent the most money (the MVP)

```sql
SELECT orders.customer, customer.firstname, customer.lastname, SUM(orders.numbersold * product.price)
FROM orders JOIN product ON product.upc = orders.product
    JOIN customer ON orders.customer = customer.phonenumber
GROUP BY orders.customer, customer.firstname, customer.lastname
ORDER BY sum DESC
```

Get all the stores ordered by who has earned the most money

```sql
SELECT orders.store, store.address, SUM(orders.numbersold * product.price)
FROM orders JOIN product ON product.upc = orders.product
    JOIN store ON store.id = orders.store
GROUP BY orders.store, store.address ORDER BY sum
```

Get all the stores ordered by who has earned the most money for a given state

```sql
SELECT orders.store, store.address, SUM(orders.numbersold * product.price)
FROM orders JOIN product ON product.upc = orders.product
    JOIN store ON store.id = orders.store
WHERE store.state = ?
GROUP BY orders.store, store.address
ORDER BY sum
```

In the first query, the orders, product, and customer tables are joined together in order to get the most money. The condition **product.upc = orders.product** allows the query to get the **price** for that item from the product table. The condition **orders.customer = customer.phonenumber** then allows to get the name of the person who has made that order. After using the SUM aggregation on multiplication of **numberSold** from orders and the **price** from the product table (in order to get the money spent on that product) and then groups that by the customer who spent that money. Ordering it by the sum descending allows the first thing in the result set to be the customer who has spent the most money.

The second query works on the same principle, with the **store.id = orders.store** condition filtering through the order being made at that specific store. It finds the total money spent in the same way, using the sum of the price and number purchased. It is ordered by sum ascending by default, in order to find the least earned money. Appending a " DESC" string in the Java program then allows it to display the stores with most earned money.

The third query is almost the same, but in order to find that same information for a specific state instead of sitewide, an extra **WHERE** store.state = ? clause is added, with the ? being replaced with the appropriate state in the application using prepared statements, in order to avoid SQL injection.

Table Creation:

```
CREATE TABLE SoldBy
  (
     storeid       CHARACTER(8),
     productid     CHARACTER(12),
     numberinstock INTEGER,
     FOREIGN KEY (storeid) REFERENCES store(id)
     FOREIGN KEY (productid) REFERENCES product(upc)
  );

CREATE TABLE reorder
  (
     ordernumber    CHAR(8) NOT NULL,
     product        CHAR(12) NOT NULL,
     store          CHAR(8) NOT NULL,
     deliverydate   DATE,
     stockrequested INTEGER NOT NULL,
     fulfilledby    VARCHAR(50),
     PRIMARY KEY (ordernumber),
     FOREIGN KEY (product) REFERENCES product(upc),
     FOREIGN KEY (store) REFERENCES store(id),
     FOREIGN KEY (fulfilledby) REFERENCES vendor(NAME),
  );
```

Shown here is a sample of some of the table creation queries we used when making the database. As you can see, almost every attribute references another attribute in the database. Our entire database is based around these foreign key connections, in order to be absolutely sure that no data is referenced or created that doesn't exist elsewhere in the database. If new information needs to be added, the admin users can create it from the ground up to avoid errors down the line. In the actual entity tables attributes that aren't primary keys are usually also foreign keys, to ensure that they cannot be null and cause errors.

Team Report

Overall, the project was a success and we were able to implement a fairly robust command line application that modelled an online shopping application. However, at the start, we were very confused about our program at first. We didn't know what our application should model, we didn't know how in depth it should go, etc. This lead to a poor overall effort when it came for Phase 1 submission. We solved this issue in Phase 2 by meeting more to discuss the project as a whole, and brainstorm better ideas. After coming up with the idea of Wegmans2 and mapping out the application, we were more confident with creating a list of features our application would implement. This was a big improvement since we were able to map out features that we wanted our project to implement, which meant for a clear roadmap for development. From that point onwards, development was very straightforward and enjoyable. Those who implemented the application were able to easily create the features outlined in each of the specific use cases.

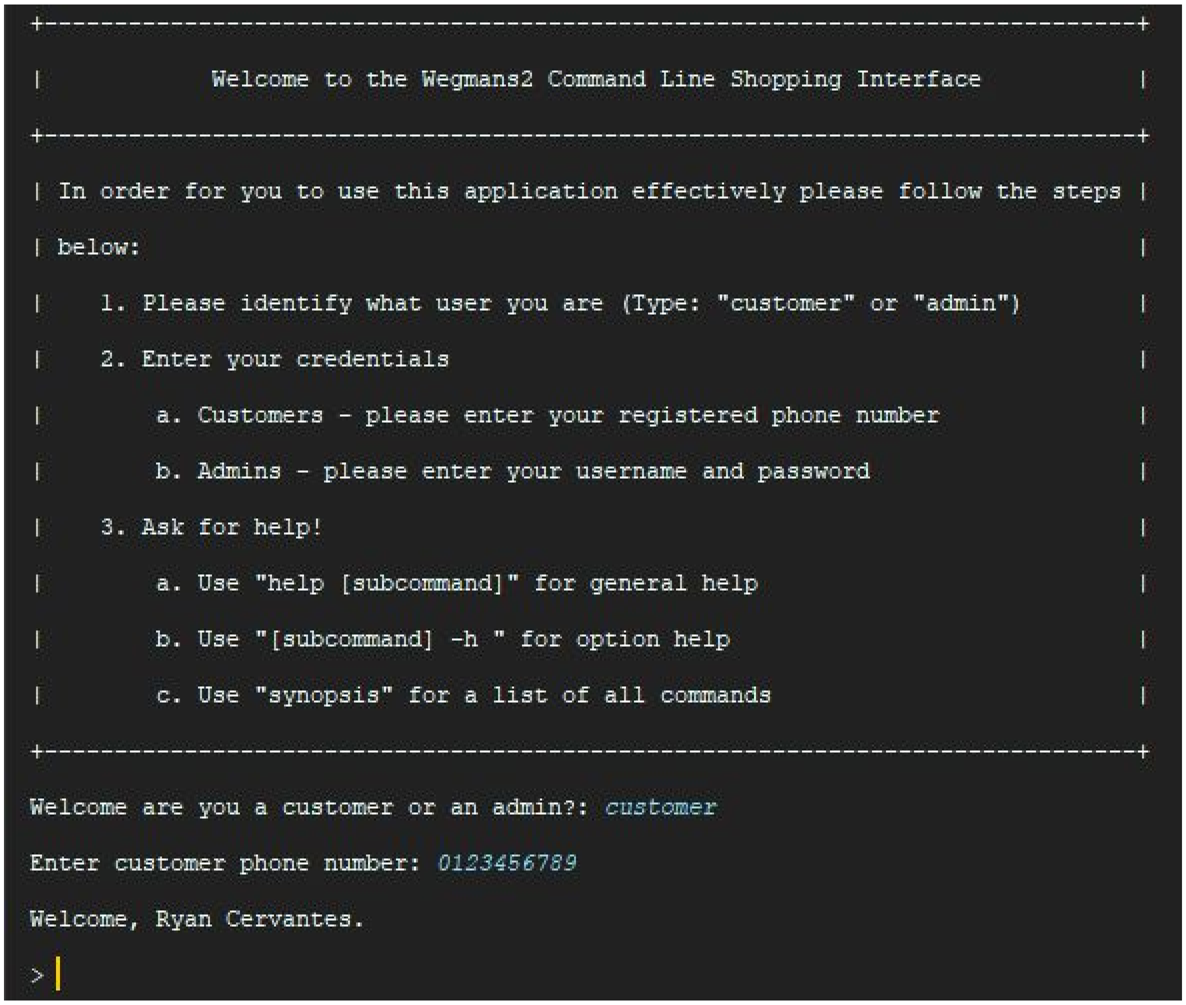
Responsibilities:

- Jorge Flores
    - Design of E-R Diagram with all other members
    - Design of database schema with all other members
    - Handled all technical database issues
        - Created database on postgres
        - Updated tables when necessary
        - Answered and consulted on all things table related
    - Fixing generated data to adhere with database
    - Populating Database with generated data
    - Writing **ALL** queries implemented in the application
    - Writing the Phase 2 report with Ryan Cervantes
    - Implementation of application with Ryan Cervantes
        - Researching application integration with PostgreSQL
        - Writing backend to interact with PostgreSQL
        - Working with Ryan to integrate backend with front-end logic
        - Debugging application
    - Editing and writing the final presentation
        - Present E-R Diagram/ Tables
        - Helping Ryan Create supplemental tables

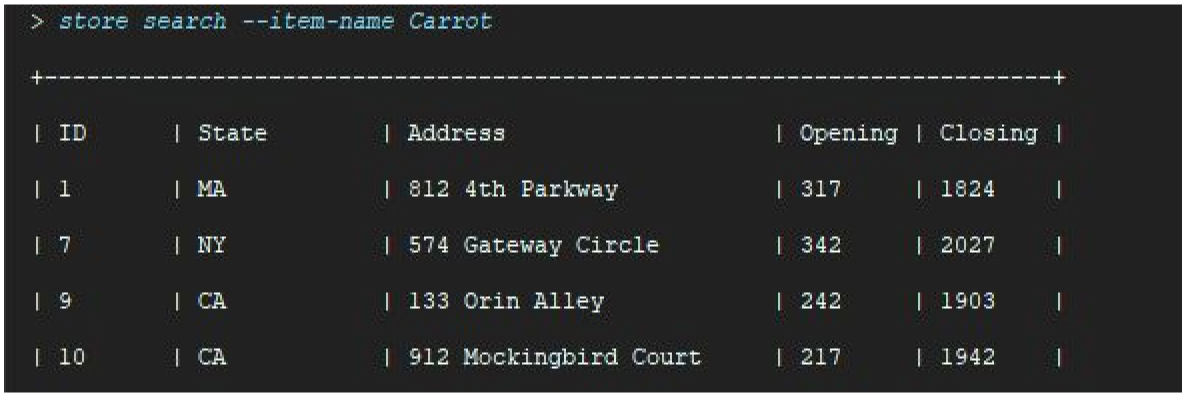
- Ryan Cervantes
    - Team leader to keep tasks organized deadlines, kept priorities straight, check in on people to do tasks they volunteered for
    - Design of E-R Diagram with all other members
    - Design of database schema with all other members
    - Assisting design/writing of queries with Jorge Flores
    - Writing most of Phase 3 Report
    - Editing and writing the final presentation
        - Creation and presentation of demo
        - Creating supplemental tables
    - Design of entire application
        - Creation of concept, explaining to others
        - Design and creation of UML Diagrams
    - Implementation of entire application with Jorge Flores
        - Researching libraries to use
        - Learning and implementing libraries (picocli, ant, etc)
        - Implementing CLI logic and parsing
        - Integrating parsing and logic with Jorge's Backend
        - Debugging application
    - Creation of README.md file
    - Building and shipping the application
- Jarod Godlewski
    - Design of E-R Diagram with all other members
    - Design of Database Schema with all other members
    - Sample data generation for Phase 2
    - Helped write Phase 2 report
    - Presenting data generation in Final Presentation

- Spencer Lowitt
    - Design of E-R Diagram with all other members
    - Design of Database Schema with all other members
    - Sample Queries for Phase 2 report
    - Create final presentation slides

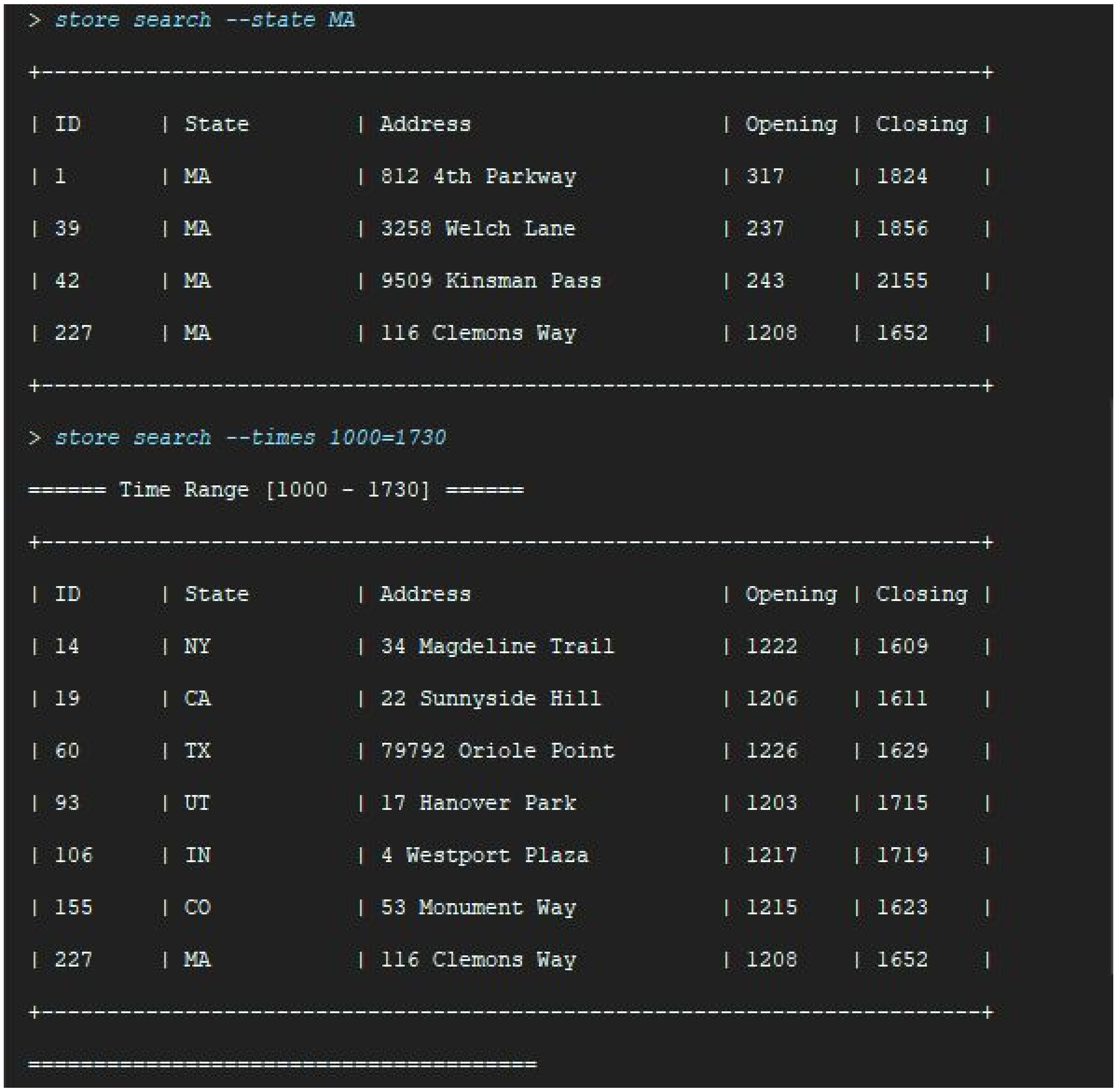
Customer Demo

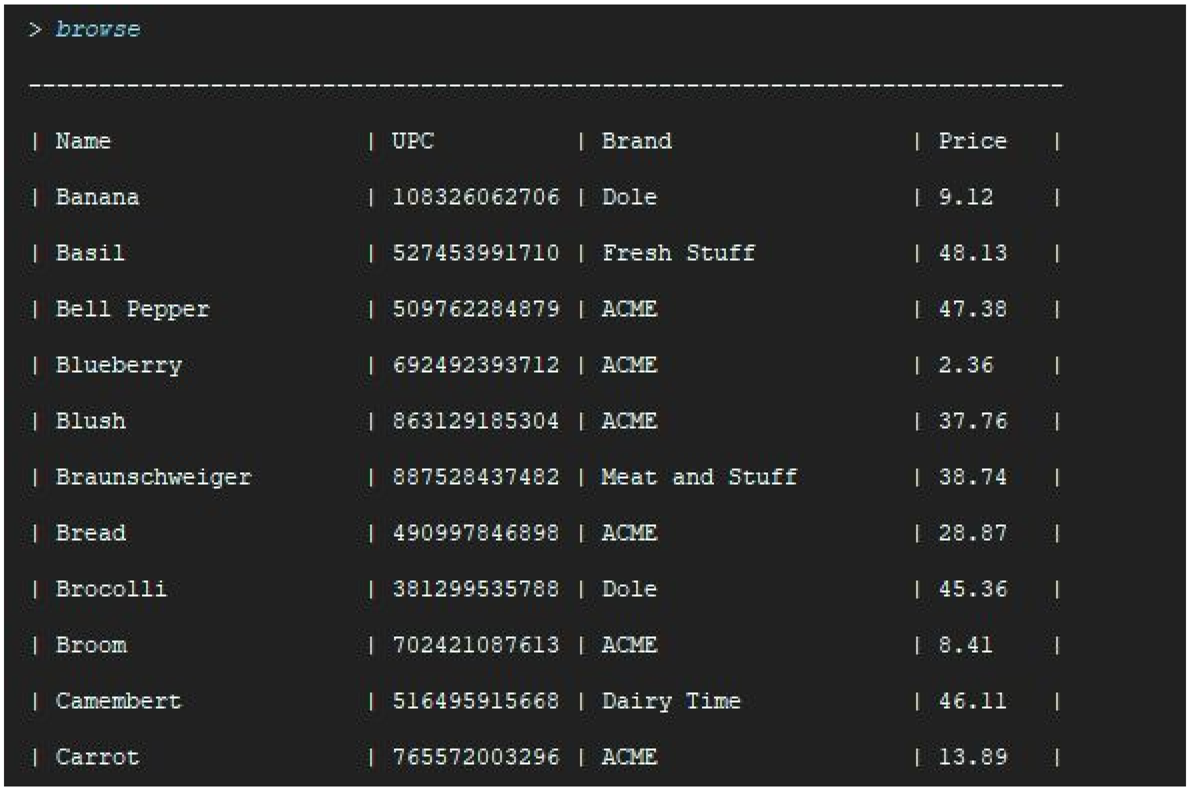


```
+------------------------------------------------------------------------------+
|            Welcome to the Wegmans2 Command Line Shopping Interface          |
+------------------------------------------------------------------------------+
| In order for you to use this application effectively please follow the steps |
| below:                                                                       |
|    1. Please identify what user you are (Type: "customer" or "admin")        |
|    2. Enter your credentials                                                 |
|        a. Customers - please enter your registered phone number              |
|        b. Admins - please enter your username and password                   |
|    3. Ask for help!                                                          |
|        a. Use "help [subcommand]" for general help                           |
|        b. Use "[subcommand] -h " for option help                             |
|        c. Use "synopsis" for a list of all commands                          |
+------------------------------------------------------------------------------+
Welcome are you a customer or an admin?: customer
Enter customer phone number: 0123456789
Welcome, Ryan Cervantes.
>
```

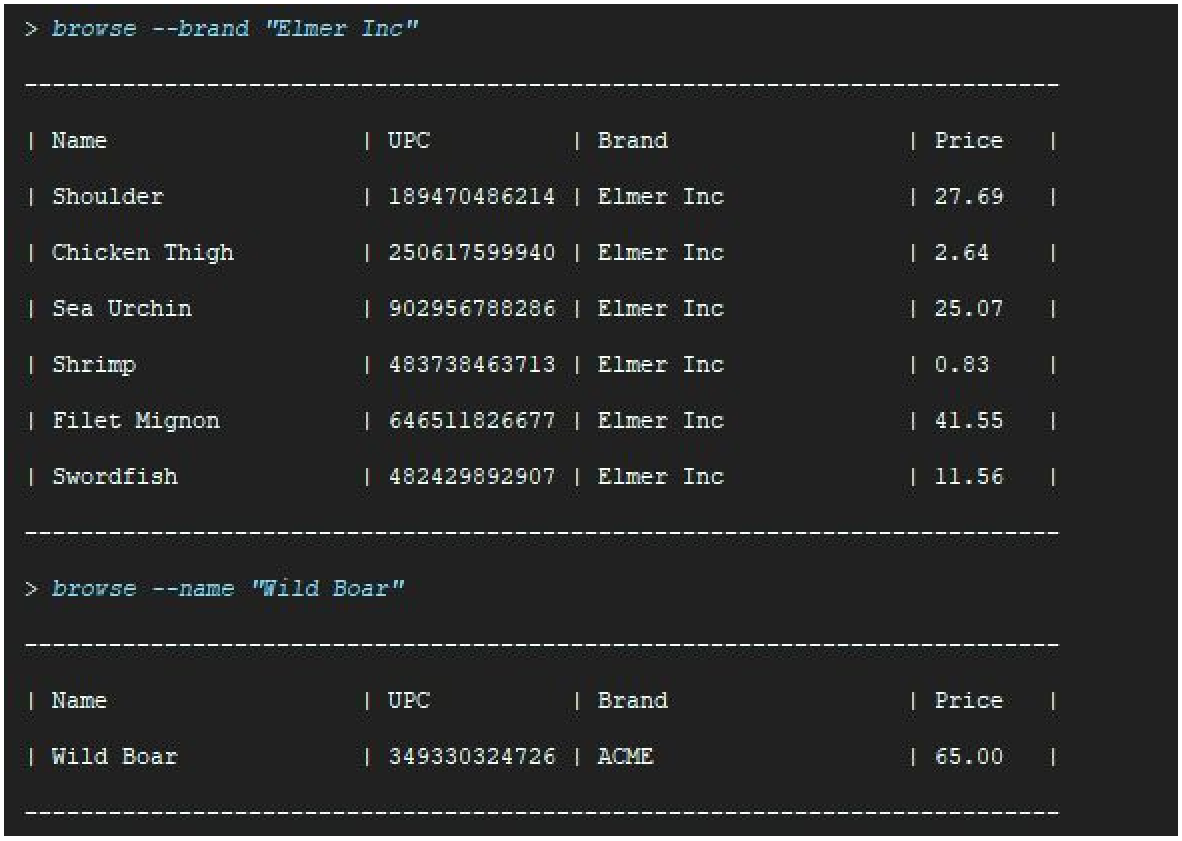


```
> store search --item-name Carrot
+-----------------------------------------------------------------------+
| ID      | State       | Address                     | Opening | Closing |
| 1       | MA          | 812 4th Parkway             | 317     | 1824    |
| 7       | NY          | 574 Gateway Circle          | 342     | 2027    |
| 9       | CA          | 133 Orin Alley              | 242     | 1903    |
| 10      | CA          | 912 Mockingbird Court       | 217     | 1942    |
```

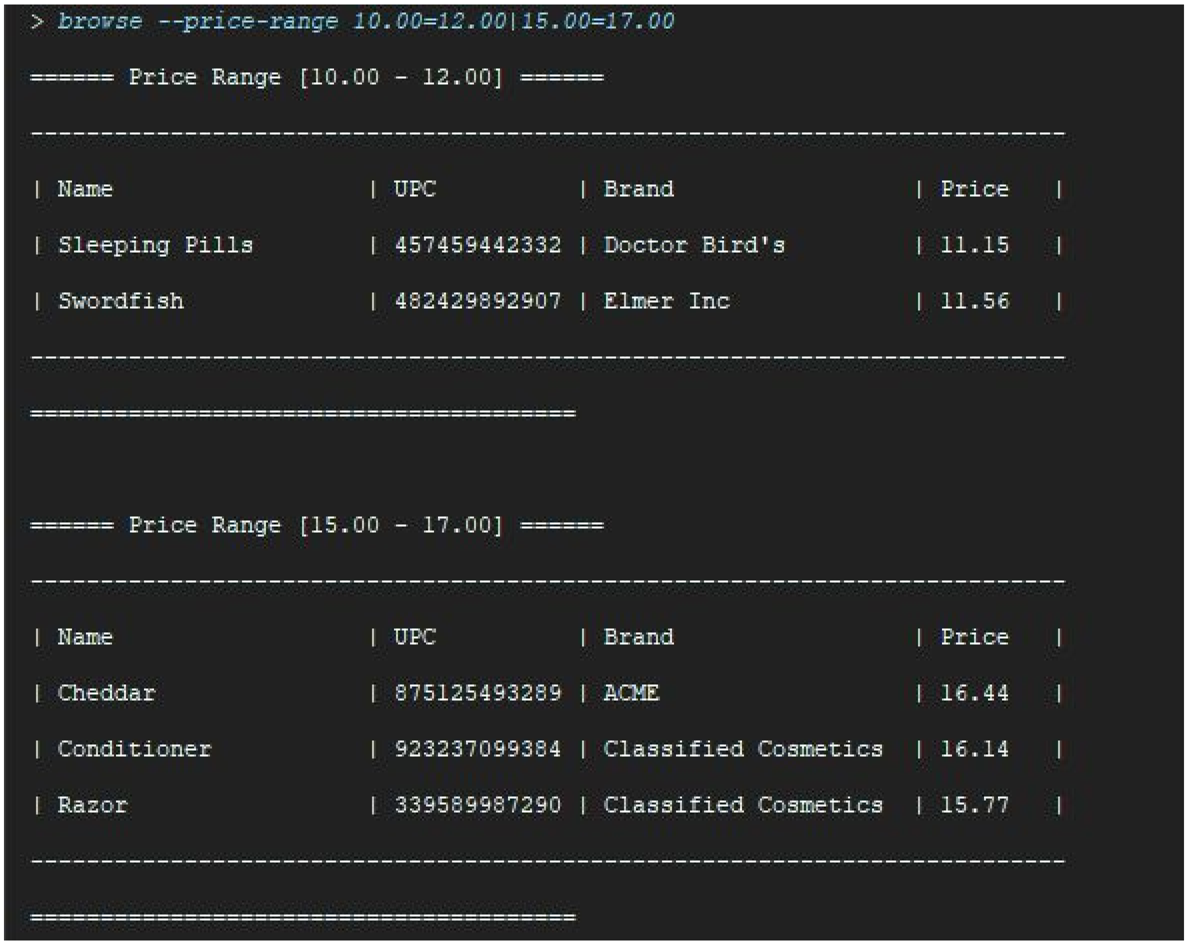
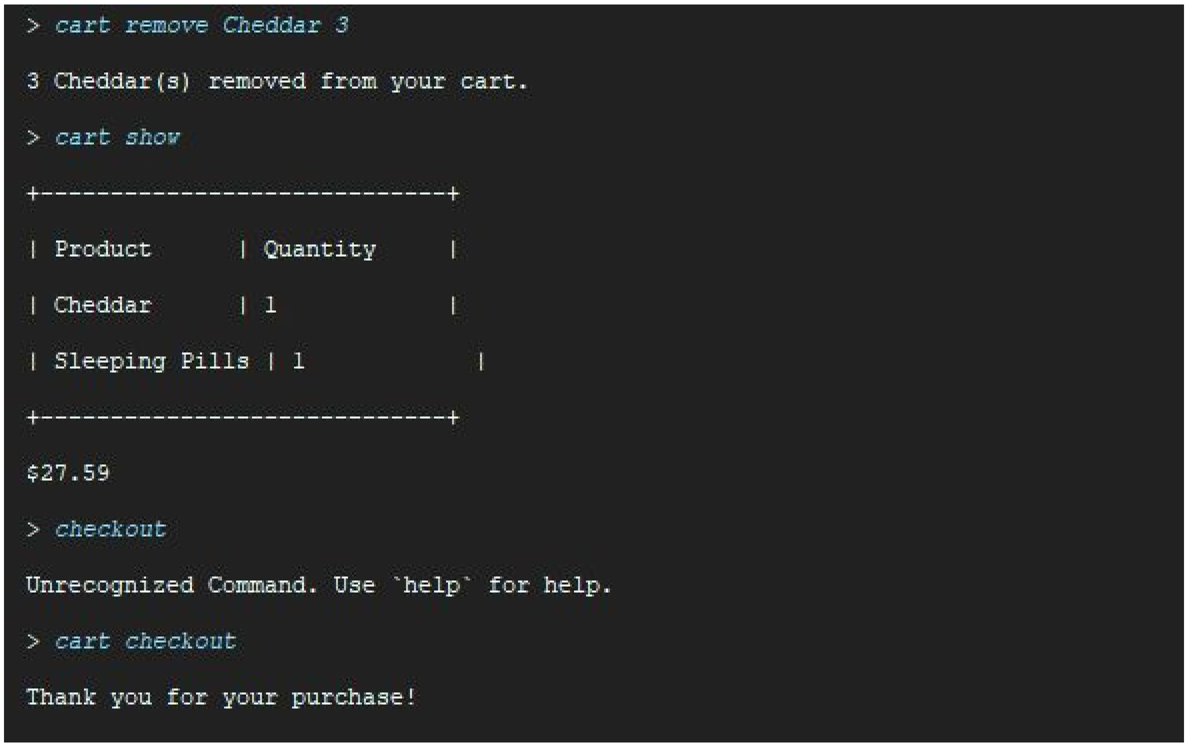

```
> store search --state MA
+-----------------------------------------------------------------------------+
| ID       | State         | Address                      | Opening | Closing |
| 1        | MA            | 812 4th Parkway              | 317     | 1824    |
| 39       | MA            | 3258 Welch Lane              | 237     | 1856    |
| 42       | MA            | 9509 Kinsman Pass            | 243     | 2155    |
| 227      | MA            | 116 Clemons Way              | 1208    | 1652    |
+-----------------------------------------------------------------------------+
> store search --times 1000=1730
====== Time Range [1000 - 1730] ======
+-----------------------------------------------------------------------------+
| ID       | State         | Address                      | Opening | Closing |
| 14       | NY            | 34 Magdeline Trail           | 1222    | 1609    |
| 19       | CA            | 22 Sunnyside Hill            | 1206    | 1611    |
| 60       | TX            | 79792 Oriole Point           | 1226    | 1629    |
| 93       | UT            | 17 Hanover Park              | 1203    | 1715    |
| 106      | IN            | 4 Westport Plaza             | 1217    | 1719    |
| 155      | CO            | 53 Monument Way              | 1215    | 1623    |
| 227      | MA            | 116 Clemons Way              | 1208    | 1652    |
+-----------------------------------------------------------------------------+
========================================
```



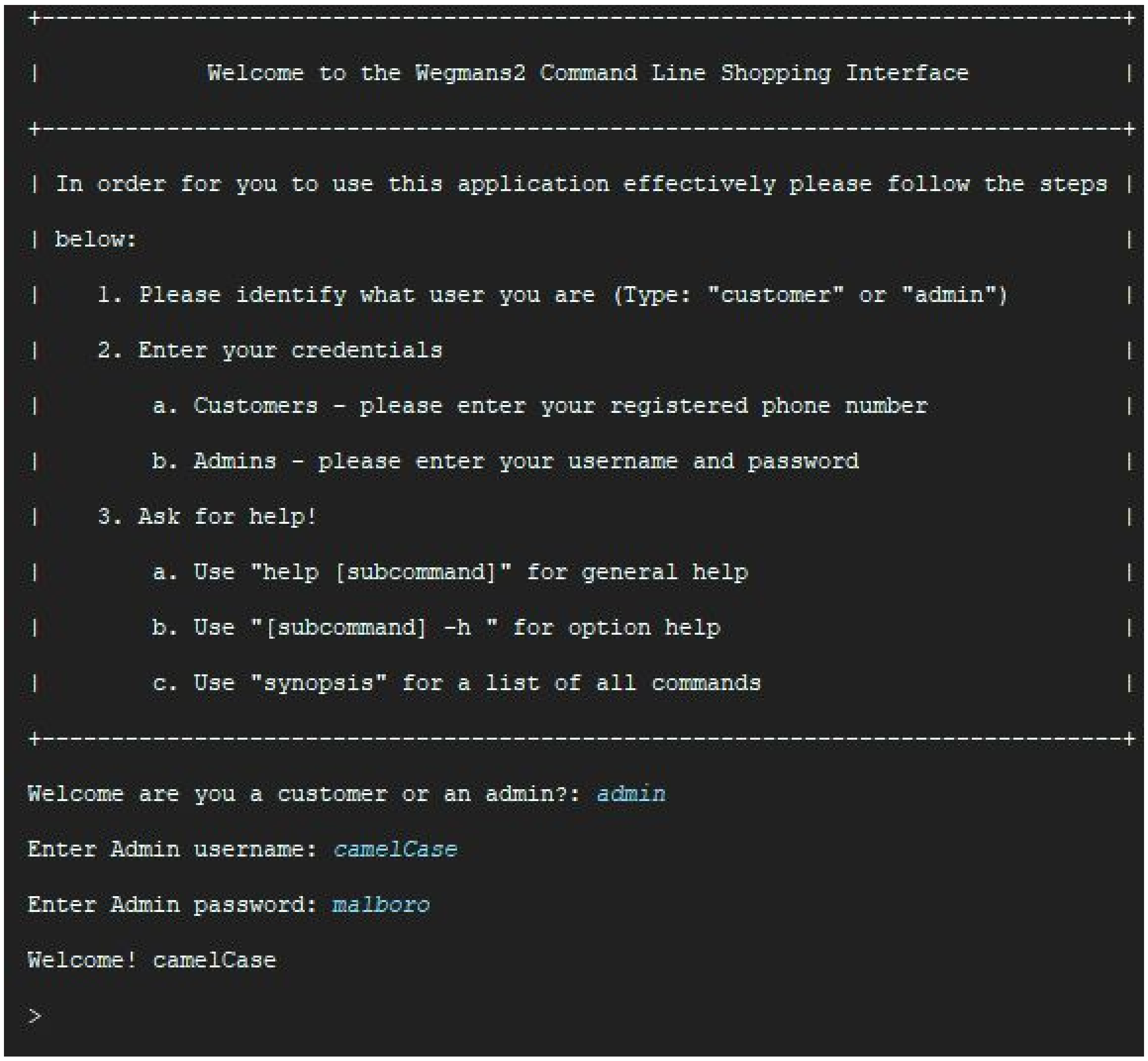
```
> browse
--------------------------------------------------------------------------
| Name                  | UPC          | Brand                 | Price    |
| Banana                | 108326062706 | Dole                  | 9.12     |
| Basil                 | 527453991710 | Fresh Stuff           | 48.13    |
| Bell Pepper           | 509762284879 | ACME                  | 47.38    |
| Blueberry             | 692492393712 | ACME                  | 2.36     |
| Blush                 | 863129185304 | ACME                  | 37.76    |
| Braunschweiger        | 887528437482 | Meat and Stuff        | 38.74    |
| Bread                 | 490997846898 | ACME                  | 28.87    |
| Brocolli              | 381299535788 | Dole                  | 45.36    |
| Broom                 | 702421087613 | ACME                  | 8.41     |
| Camembert             | 516495915668 | Dairy Time            | 46.11    |
| Carrot                | 765572003296 | ACME                  | 13.89    |
```



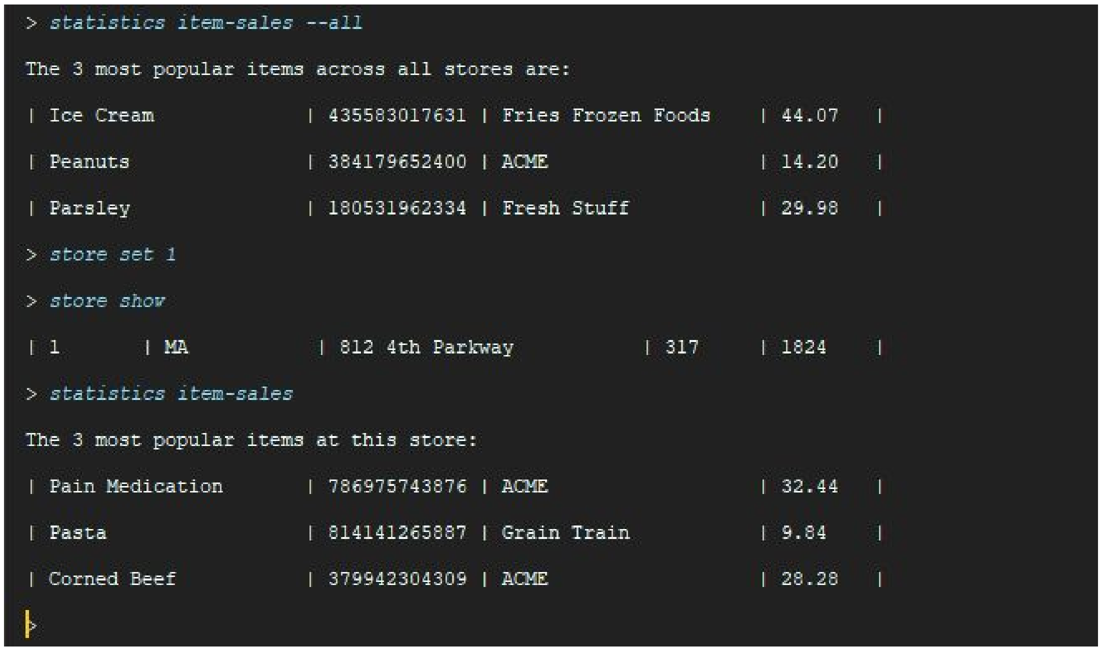
```
> browse --brand "Elmer Inc"
--------------------------------------------------------------------------
| Name                  | UPC          | Brand                 | Price    |
| Shoulder              | 189470486214 | Elmer Inc             | 27.69    |
| Chicken Thigh         | 250617599940 | Elmer Inc             | 2.64     |
| Sea Urchin            | 902956788286 | Elmer Inc             | 25.07    |
| Shrimp                | 483738463713 | Elmer Inc             | 0.83     |
| Filet Mignon          | 646511826677 | Elmer Inc             | 41.55    |
| Swordfish             | 482429892907 | Elmer Inc             | 11.56    |
--------------------------------------------------------------------------
> browse --name "Wild Boar"
--------------------------------------------------------------------------
| Name                  | UPC          | Brand                 | Price    |
| Wild Boar             | 349330324726 | ACME                  | 65.00    |
--------------------------------------------------------------------------
```


```
> browse --price-range 10.00=12.00|15.00=17.00

====== Price Range [10.00 - 12.00] ======

-------------------------------------------------------------------------------

| Name                   | UPC          | Brand                   | Price    |

| Sleeping Pills         | 457459442332 | Doctor Bird's           | 11.15    |

| Swordfish              | 482429892907 | Elmer Inc               | 11.56    |

-------------------------------------------------------------------------------

=========================================


====== Price Range [15.00 - 17.00] ======

-------------------------------------------------------------------------------

| Name                   | UPC          | Brand                   | Price    |

| Cheddar                | 875125493289 | ACME                    | 16.44    |

| Conditioner            | 923237099384 | Classified Cosmetics    | 16.14    |

| Razor                  | 339589987290 | Classified Cosmetics    | 15.77    |

-------------------------------------------------------------------------------

=========================================
```

```
> cart remove Cheddar 3

3 Cheddar(s) removed from your cart.

> cart show

+-----------------------------+

| Product       | Quantity     |

| Cheddar       | 1            |

| Sleeping Pills | 1            |

+-----------------------------+

$27.59

> checkout

Unrecognized Command. Use `help` for help.

> cart checkout

Thank you for your purchase!
```

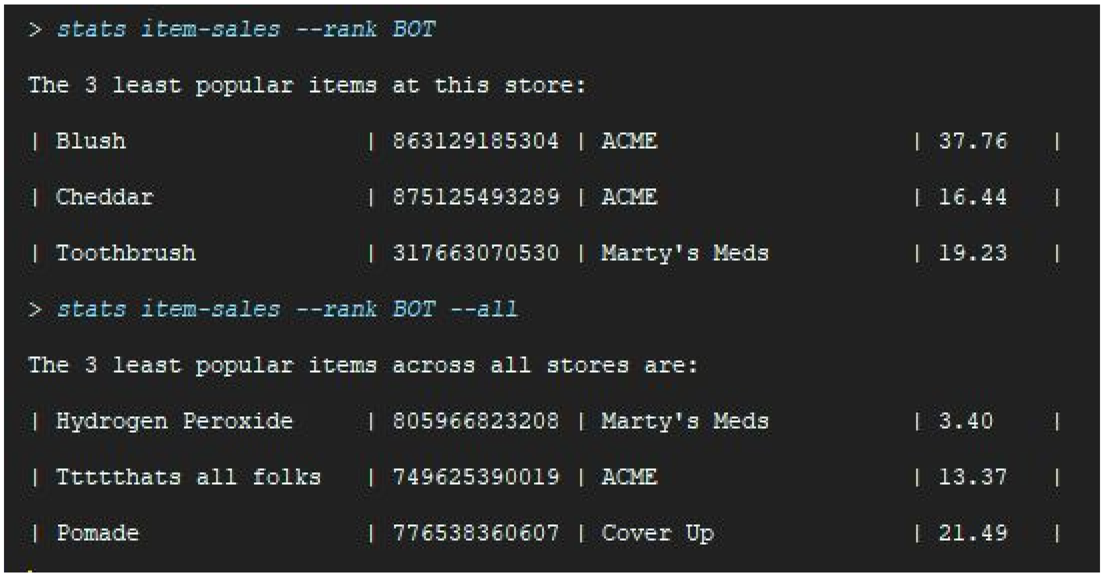
Admin Demo



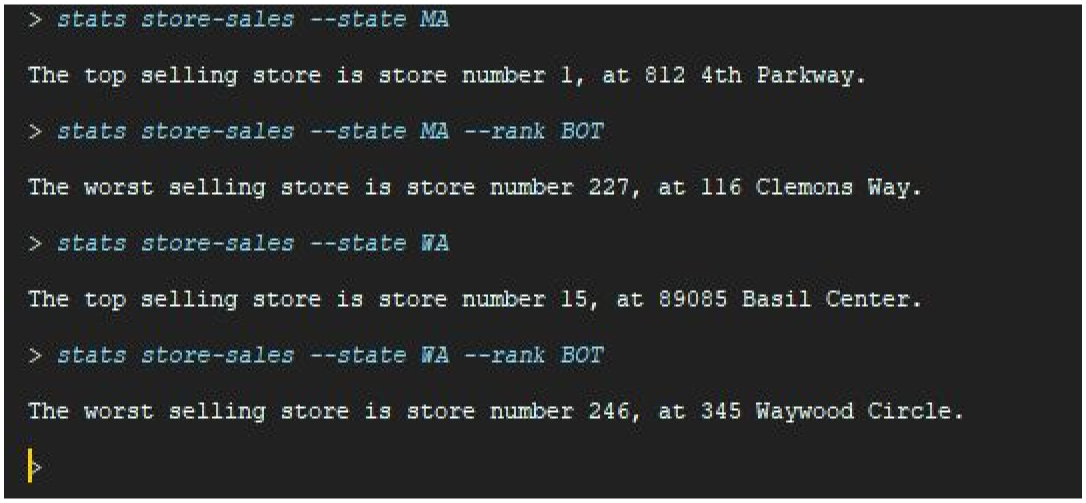
```
+------------------------------------------------------------------------------+
|            Welcome to the Wegmans2 Command Line Shopping Interface           |
+------------------------------------------------------------------------------+
| In order for you to use this application effectively please follow the steps |
| below:                                                                       |
|    1. Please identify what user you are (Type: "customer" or "admin")        |
|    2. Enter your credentials                                                 |
|        a. Customers - please enter your registered phone number              |
|        b. Admins - please enter your username and password                   |
|    3. Ask for help!                                                          |
|        a. Use "help [subcommand]" for general help                           |
|        b. Use "[subcommand] -h " for option help                             |
|        c. Use "synopsis" for a list of all commands                          |
+------------------------------------------------------------------------------+
Welcome are you a customer or an admin?: admin
Enter Admin username: camelCase
Enter Admin password: malboro
Welcome! camelCase
>
```



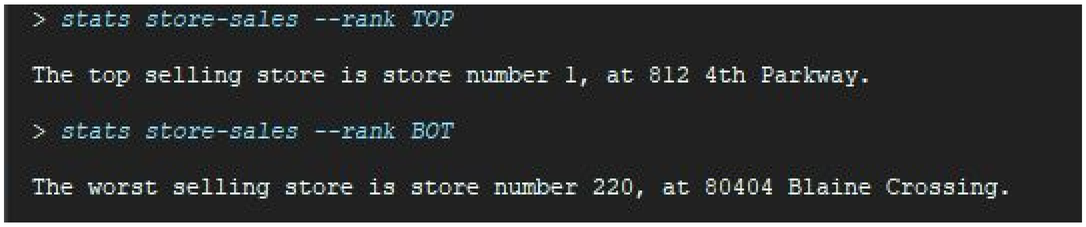
```
> statistics item-sales --all

The 3 most popular items across all stores are:

| Ice Cream            | 435583017631 | Fries Frozen Foods    | 44.07    |

| Peanuts              | 384179652400 | ACME                  | 14.20    |

| Parsley              | 180531962334 | Fresh Stuff           | 29.98    |

> store set 1

> store show

| 1       | MA          | 812 4th Parkway          | 317     | 1824     |

> statistics item-sales

The 3 most popular items at this store:

| Pain Medication      | 786975743876 | ACME                  | 32.44    |

| Pasta                | 814141265887 | Grain Train           | 9.84     |

| Corned Beef          | 379942304309 | ACME                  | 28.28    |

>
```



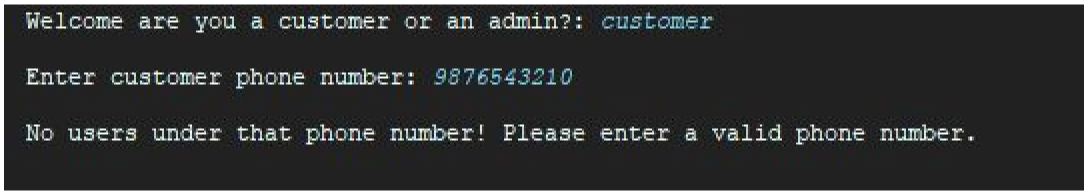
```
> stats item-sales --rank BOT

The 3 least popular items at this store:

| Blush                | 863129185304 | ACME                  | 37.76    |

| Cheddar              | 875125493289 | ACME                  | 16.44    |

| Toothbrush           | 317663070530 | Marty's Meds          | 19.23    |

> stats item-sales --rank BOT --all

The 3 least popular items across all stores are:

| Hydrogen Peroxide    | 805966823208 | Marty's Meds          | 3.40     |

| Tttthats all folks   | 749625390019 | ACME                  | 13.37    |

| Pomade               | 776538360607 | Cover Up              | 21.49    |
```



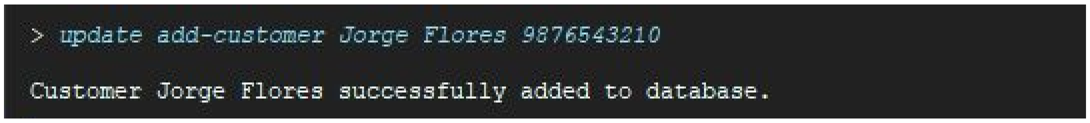
```
> stats store-sales --state MA

The top selling store is store number 1, at 812 4th Parkway.

> stats store-sales --state MA --rank BOT

The worst selling store is store number 227, at 116 Clemons Way.

> stats store-sales --state WA

The top selling store is store number 15, at 89085 Basil Center.

> stats store-sales --state WA --rank BOT

The worst selling store is store number 246, at 345 Waywood Circle.

>
```



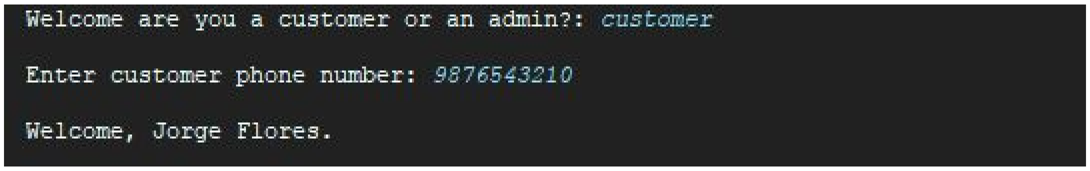
```
> stats store-sales --rank TOP

The top selling store is store number 1, at 812 4th Parkway.

> stats store-sales --rank BOT

The worst selling store is store number 220, at 80404 Blaine Crossing.
```



```
Welcome are you a customer or an admin?: customer

Enter customer phone number: 9876543210

No users under that phone number! Please enter a valid phone number.
```



```
> update add-customer Jorge Flores 9876543210

Customer Jorge Flores successfully added to database.
```



```
Welcome are you a customer or an admin?: customer

Enter customer phone number: 9876543210

Welcome, Jorge Flores.
```