# TCSS 450 Final Project - SockBank

Yang(Vivian) Tang and Kevin Nguyen

July 13, 2020

## Contents

# 1 Final Report: A Sockfull Analysis

**Abstract**

This report discusses the design and implementation of our TCSS 445 Database System and Design cumulative project. The task at hand was to provide a solution to a real-world problem that could be solved using database systems. We chose to model an online store to address the issue of people being left with only one half of a pair of socks after losing one during the laundry. Specifically mentioned in this paper is the motivation behind the topic chosen, the objective in mind (both personally, and for the project), and the design decisions made for our application.

## 1.1 Introduction

Thanks to the advancements of technology, buying (and even selling) clothes have become more accessible than ever. Naturally, when we were thinking of systems that could be modeled after a database, we thought of an online retailer. It is something we are accustomed to interacting with regularly. But we are always on the consumer side, adding content to the databases, and never on the administrative side, managing the systems. And so the interest had led us here; to modeling our project after an online retailer.

## 1.2 Objective and Scope

The objective of our project, initially, was to create a front end interface online store that exclusively sold socks. It was hard to imagine an online store being able to solve a problem that many people have. We realized a phenomenon that many people, if not everyone, has had once in their life: people tend to lose one sock when doing laundry! Therefore we decided to be an online retailer that sold socks to people, and that could sell just one sock to people that lost one half of a pair.

The scope of our project changed midway. At first, we wanted to create an online store where customers could shop for socks. But that is boring. Going back to our initial goal of being on the administrative side, we decided our database application will deal with the administrative team of owning an online retailer. That is, checking stocks, managing orders, managing customers, extrapolating which items are good sellers, and vice versa by tracking returns. In doing so, we can generate conclusions from the data, making it a more exciting project.

## 1.3 Other Existing Work

Since there are countless online retailers, there are numerous other works that is similar to our model. Although, it is our idea that is unique. To be able to purchase a single sock is unusual.

## 1.4 Main Body

### 1.4.1 Design

The database used to design the stores is similar to that of other online stores (we speculate). This section will discuss in detail for each relation what attributes we chose to include in our design, and what attributes we considered but decided to leave out.

The *CUSTOMER* relation includes important things about a customer, but nothing unneeded about the customer for our purposes. What we mean is, we only store information about a customer that is truly necessary for our applications. For example, we do not store birthday or credit card information.Perhaps in the future, if we implemented a front end shopping interface, we would have to store this information, but for the purposes of this project, this is enough. Especially it is composed of an *ID, Fname, Lname, Phone, email address, city, state,* and *postal*.

The *ORDERS* relation consists of *cust_id, order_id, order_date, card_number*, and *total*. Here we use the *order_id* to uniquely identify an order and the *cust _id* to determine whose order this is. We store the card number in the event that we need to issue a refund to a user.

The *ORDER _ITEM* relation is composed of *item_id, order_id*, and total. Here we track the quantity of a specific item is purchased, and the associated order it came from. By design, an order consists of multiple *order_item* tuples.

The *ITEM* relation represents an item to be sold in our database. An item consists of *item_title, item_id, item_title, unit_price, stock, manu_id, and item_discount*. *Manu_id* is included to identify which manufacturer produced this item. The other attributes are typical ones that is expected to represent an item.

The RETURNS relation was added to include another dimension of complexity to the application, as well as to more properly model the real world. Tuples from this relation are of the form *customer_id item_id, order_id,* qty, and total. This relation is a child of the *ORDER_ITEM* relation. This is to model a return in the real world, i.e. customers can return part of a purchase, and are not required to return all items from a purchase. We keep track of the *customer_id* so that we may keep track of which customers return what. From this attribute and aggregate functions we can determine for a given customer how many returns are made.

The *BLACKLISTED(card_number)*relation represents the credit card numbers our store will not accept. Card numbers are added to this relation because the customer using the added number has made too many returns to the store. This is especially important because our store sells socks. It is both unhygienic and unethical to resell used socks. For the store, having a customer make too many returns is unprofitable, so we choose to blacklist their credit cards.

The *MANUFACTURER* relation was added to complete the model. It consists of an *manu_id, manu_name, manu_phone* and *manu_email.* Manufacturers create the products that we sell; for our model it would be useful to see which products are more profitable, and consequently which manufacturers are more profitable.

The *SHIPMENT* and *WAREHOUSE* relations were copied from lecture. Nothing special was added to them, but were included to provide a more complete model of the problem addressed.

### 1.4.2 Implementation

After we designed the database, we ran our SQL script on Oracle Live. Then, we created the relationship schema diagram, the ER Diagram and used normalization of BCNF to make sure that we have the appropriate design. After we satisfied our database design, we started the coding work of the implantation of our web. We used the HTML, CSS, SQL and PHP languages to code.

Firstly, we created my Homepage using HTML and CSS. On that page, we created the AdminSockBank title and added the logo image. After that, we created four statistical sums. The administration team of SockBank is then able to keep track of total sales, total customers, total orders and total warehouses. We want to encourage the administration team of our SockBank company by those numbers. Being able to see every single increase on those numbers might motivating them and fostering working enthusiasm.

Secondly, we created the Search Customer Page, the same title, logo and navigation bar after that we created one heading, input box and submit button. After entering the right Customer ID, we can get customer details from the customer table. If we entered the id that is not assigned to any customer , the web will show the Error Message that this id is not assigned to any customer. If we enter a valid id, then we get Customer Details, and from there we also get that Customer Order details. After getting the required data, we can go back to the homepage by clicking on the link which we created for that purpose. Similarly, we created the Search Order page. For all the processing work for Search Customer and Search Order, we created a different file Fuction.php which handles all the backhand working. search_customer.php is the Search Customer work file, and searh_order.php is the Search Order work file.

Thirdly, we got data from the Customer table and created the Pie Chart Customer Distribution Chart to make our website look nice. We created the Pie Chart using the Google Chart libraries. Fourthly, we created five buttons on our homepage, including Next Item On Sale, Revenue By Item, Find Target Customer, Find Target Customer and Blacklist Cards. By clicking any of the buttons, we will go to the specified page displaying the appropriate information. We created a PHP file for each of the buttons. The queries we used for each of the buttons is in Appendix D.

Connection.php in that connection process is defined. index.php is the main homepage file. That is the main hierarchy of our site and all the working and functionality of our site.

## 1.5 An overview of the architecture

The architecture of our application involves the following entities: the database, the server, the client, and us (the business). The database type used is a MySQL database. This is because Codeanywhere supports MySQL.

The server used is Codeanywhere Inc.s cloud IDE. We chose this rather than Microsoft Visual Studio because we are both Mac users, with no access to Windows machines at home. But also, for our purposes we thought Codeanywhere would be more convenient, by the name, we could code and collaborate anywhere.

The client of our application is ourselves. In designing the project we are the store, as well as the developers for the system. Thinking back this is a bit strange for a store of our scale, i.e. to develop its systems in house is a trend for larger companies and corporations.

The Internet is the connection between information from our database on Codeanywheres cloud and its front end interface. Trivially, it is the way we would sell our products, and thus it is the way our customers would interact with the database, which would in turn affect our current front end interface.

In sum, it is the customers who populate the database, then we as administrators will monitor trends in the database to make informed decisions that would benefit the store. These decisions include advertising plans, restocking of items, decreasing purchases made to a unprofitable manufacturer, etc.

## 1.6 Conclusions and Contributions

In conclusion, through this course we have gained hands-on experience in developing a database system and thus a deeper understanding of how a retailers database system functions. This project allowed us to think critically about what elements are involved in a retail management system where we applied the ER model to each entity. We believe we completed the project to the best of our abilities as a team of 2, but most definitely if we had a team of 3 we could have a less ugly interface.

In the end we both think this project was completed fairly by both of us. Not all the contributions were equal all the time though. Realistically, when one of us was too busy the other stepped up and did more. This switch off of workload responsibility happened many times, leading to us having a very fair work contribution in all aspects.

## 1.7 Future Work

This project can be continued in many ways. Most obvious is to create an online front end store interface for customers to buy products. This way we can add additional content to the database without manually typing in every entry into relations. This would make it easier to generate more interesting data, where we could see obvious trends happening with the store. But also it would make the concept of an online retailer more complete.

Yet another thing we can work on is finalizing what happens during a credit card transaction. The first step would be to determine how transactions are carried out, which APIs are used, and how to validate with the provided credit card credentials with the credit card issuer. Adding onto this, we would like to learn to what extent do companies keep credit card data, and for what reason! Or is it instead that companies connect to VISA, AMEX, DISCOVER, etc., APIs to do this.

Another possible future work would be to reinvent the front-end user interface for the project. We would like to make it more dynamic, and flexible for real use. Specifically adding forms to the interface so an employee of the store could manipulate the data. We both think it would be a great (and extremely difficult) learning experience to create an interface which is up to par with industry standards systems.

## 2 Appendix A: ER Diagram

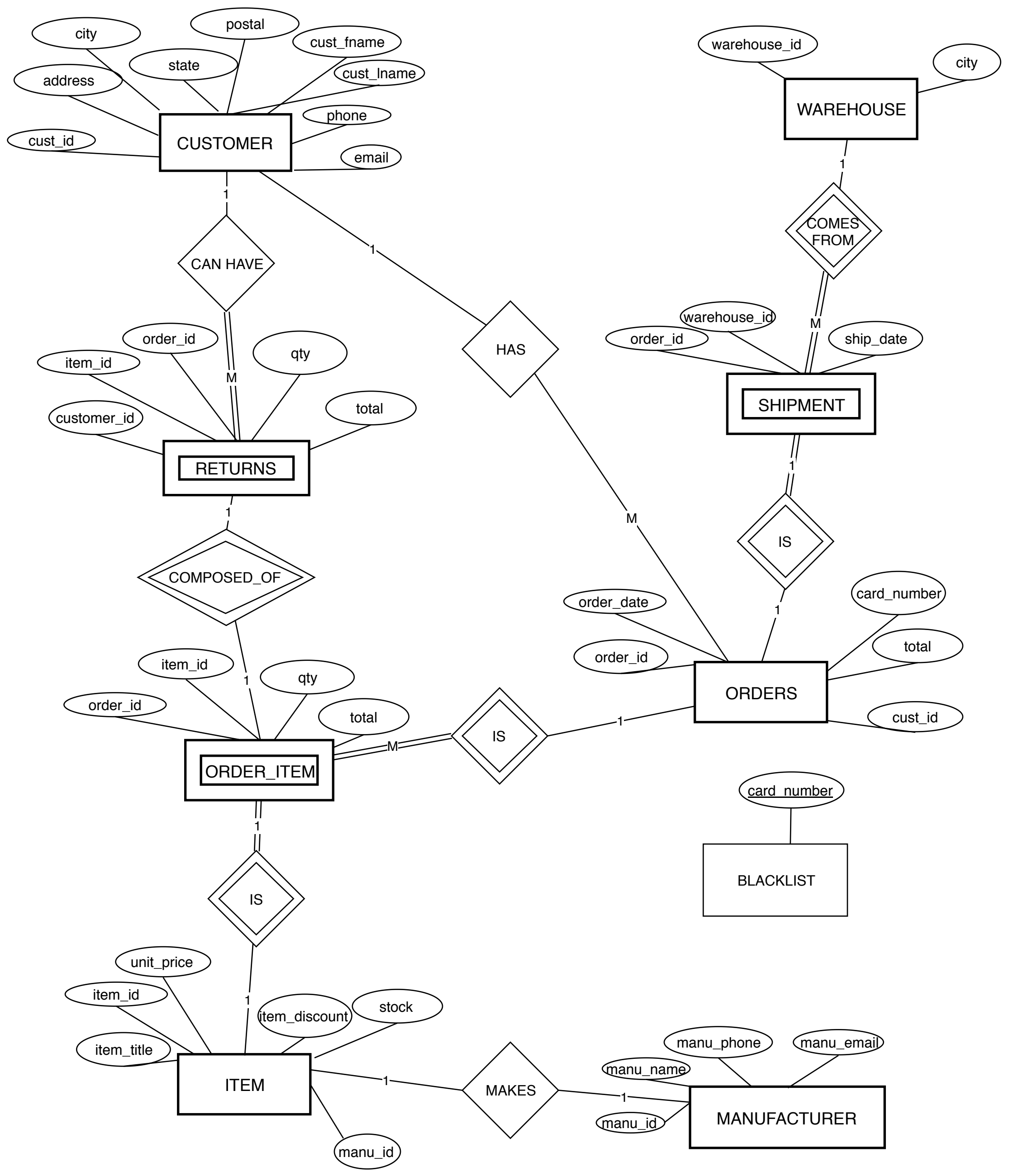
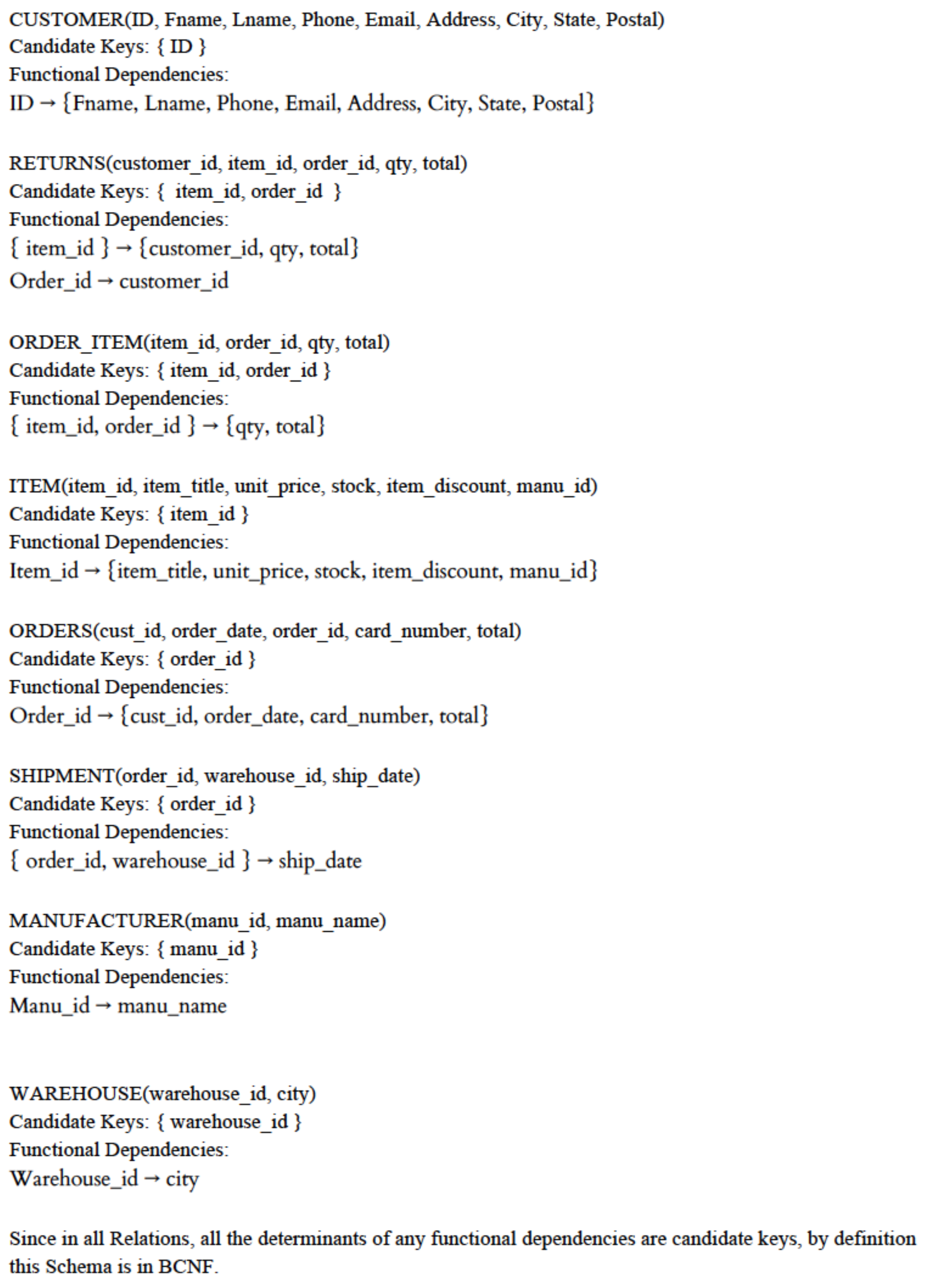


Figure 1: ER Diagram

# 3 Appendix B: Normalization Proof to BCNF



CUSTOMER(ID, Fname, Lname, Phone, Email, Address, City, State, Postal)
Candidate Keys: { ID }
Functional Dependencies:
ID → {Fname, Lname, Phone, Email, Address, City, State, Postal}

RETURNS(customer_id, item_id, order_id, qty, total)
Candidate Keys: { item_id, order_id }
Functional Dependencies:
{ item_id } → {customer_id, qty, total}
Order_id → customer_id

ORDER_ITEM(item_id, order_id, qty, total)
Candidate Keys: { item_id, order_id }
Functional Dependencies:
{ item_id, order_id } → {qty, total}

ITEM(item_id, item_title, unit_price, stock, item_discount, manu_id)
Candidate Keys: { item_id }
Functional Dependencies:
Item_id → {item_title, unit_price, stock, item_discount, manu_id}

ORDERS(cust_id, order_date, order_id, card_number, total)
Candidate Keys: { order_id }
Functional Dependencies:
Order_id → {cust_id, order_date, card_number, total}

SHIPMENT(order_id, warehouse_id, ship_date)
Candidate Keys: { order_id }
Functional Dependencies:
{ order_id, warehouse_id } → ship_date

MANUFACTURER(manu_id, manu_name)
Candidate Keys: { manu_id }
Functional Dependencies:
Manu_id → manu_name

WAREHOUSE(warehouse_id, city)
Candidate Keys: { warehouse_id }
Functional Dependencies:
Warehouse_id → city

Since in all Relations, all the determinants of any functional dependencies are candidate keys, by definition this Schema is in BCNF.

Figure 2: Demonstration of Boyce-Codd Normal Form

## 4 Appendix C: Relational Schema Diagram

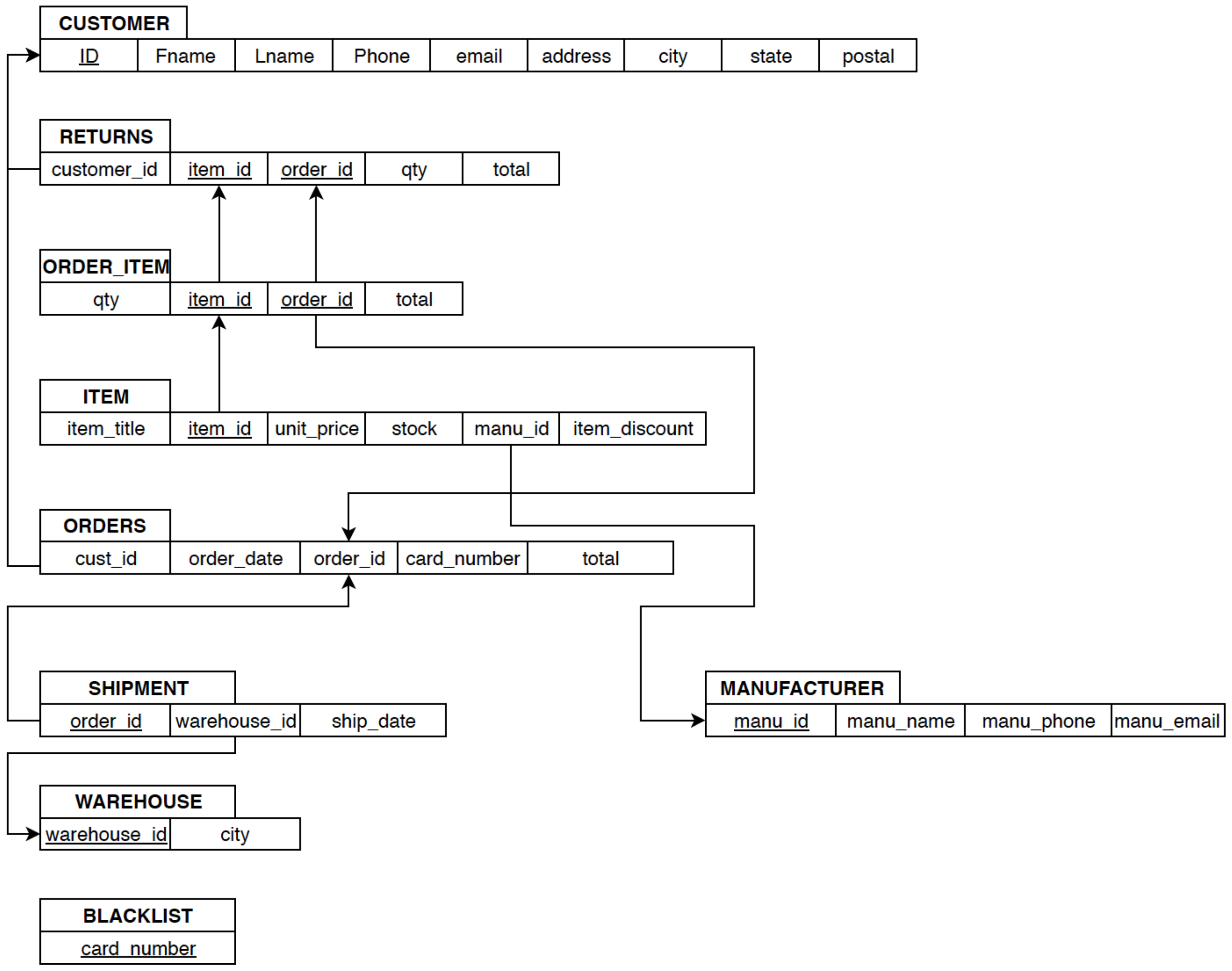
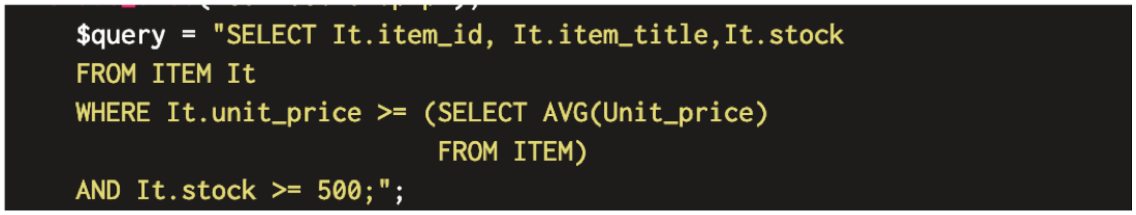


Figure 3: Relational Schema Diagram

## 5 Appendix D: Screenshots of functional SQL Queries

### 5.1 Query 1 : Next Item On Sales

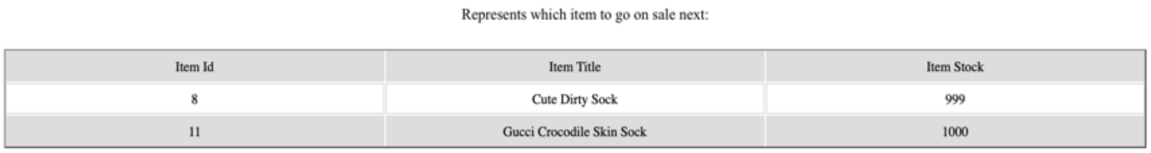
This query is used to choose which items should go on sale next. This is determined by a items unit price being greater than or equal to the average price of items, and whose quantity is greater than 500. It represents items that are overly expensive and are plentiful in stock.



```
$query = "SELECT It.item_id, It.item_title,It.stock
FROM ITEM It
WHERE It.unit_price >= (SELECT AVG(Unit_price)
                        FROM ITEM)
AND It.stock >= 500;";
```

Figure 4: Query 1

Table 1: Result of Query 1

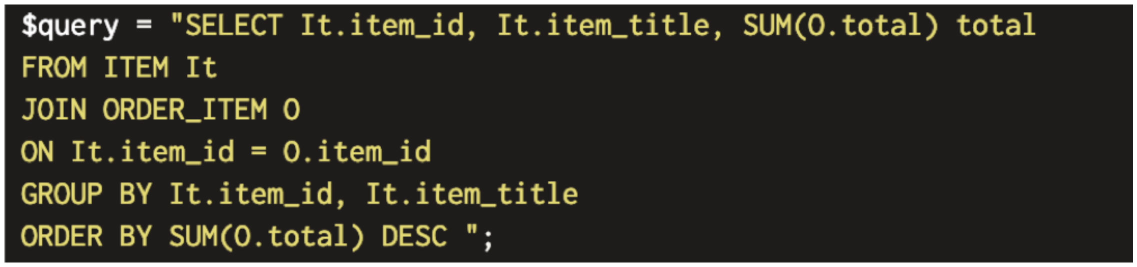


Represents which item to go on sale next:

| Item Id | Item Title | Item Stock |
|---|---|---|
| 8 | Cute Dirty Sock | 999 |
| 11 | Gucci Crocodile Skin Sock | 1000 |

### 5.2 Query 2 : Revenue Per Product

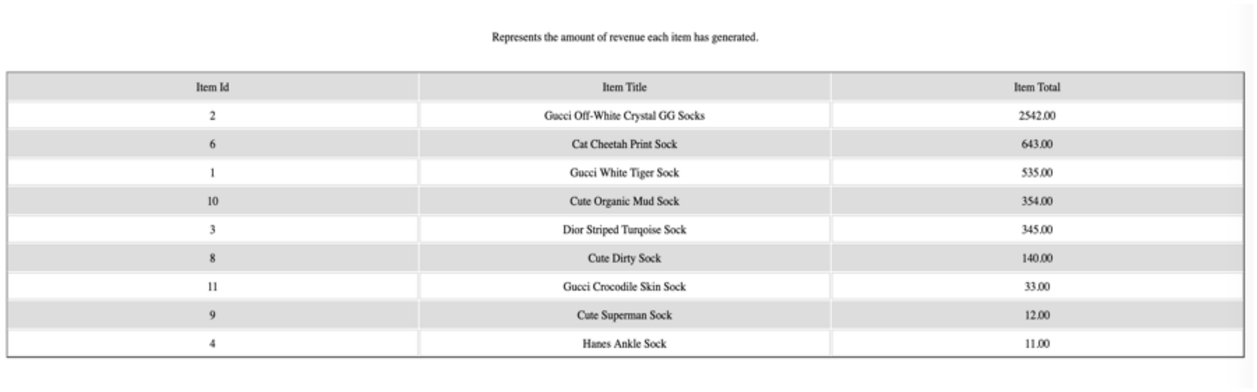
This query is used to choose which items the store should order more of. This shows which items produce the most income for the store, and so we should order more of that item.



```
$query = "SELECT It.item_id, It.item_title, SUM(O.total) total
FROM ITEM It
JOIN ORDER_ITEM O
ON It.item_id = O.item_id
GROUP BY It.item_id, It.item_title
ORDER BY SUM(O.total) DESC ";
```

Figure 5: Query 2

Table 2: Result of Query 2

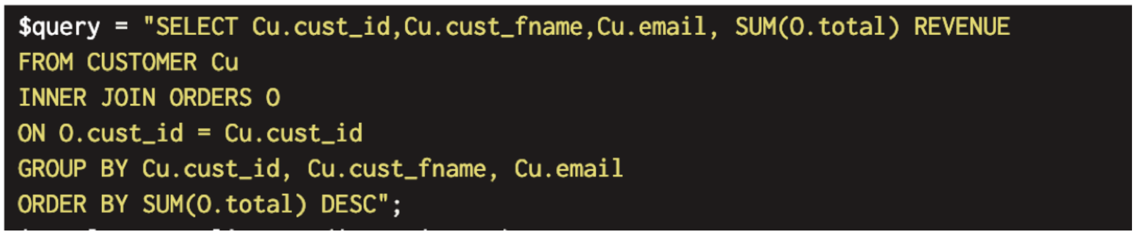


Represents the amount of revenue each item has generated.

| Item Id | Item Title | Item Total |
|---|---|---|
| 2 | Gucci Off-White Crystal GG Socks | 2542.00 |
| 6 | Cat Cheetah Print Sock | 643.00 |
| 1 | Gucci White Tiger Sock | 535.00 |
| 10 | Cute Organic Mud Sock | 354.00 |
| 3 | Dior Striped Turqoise Sock | 345.00 |
| 8 | Cute Dirty Sock | 140.00 |
| 11 | Gucci Crocodile Skin Sock | 33.00 |
| 9 | Cute Superman Sock | 12.00 |
| 4 | Hanes Ankle Sock | 11.00 |

## 5.3 Query 3 : Target Customers

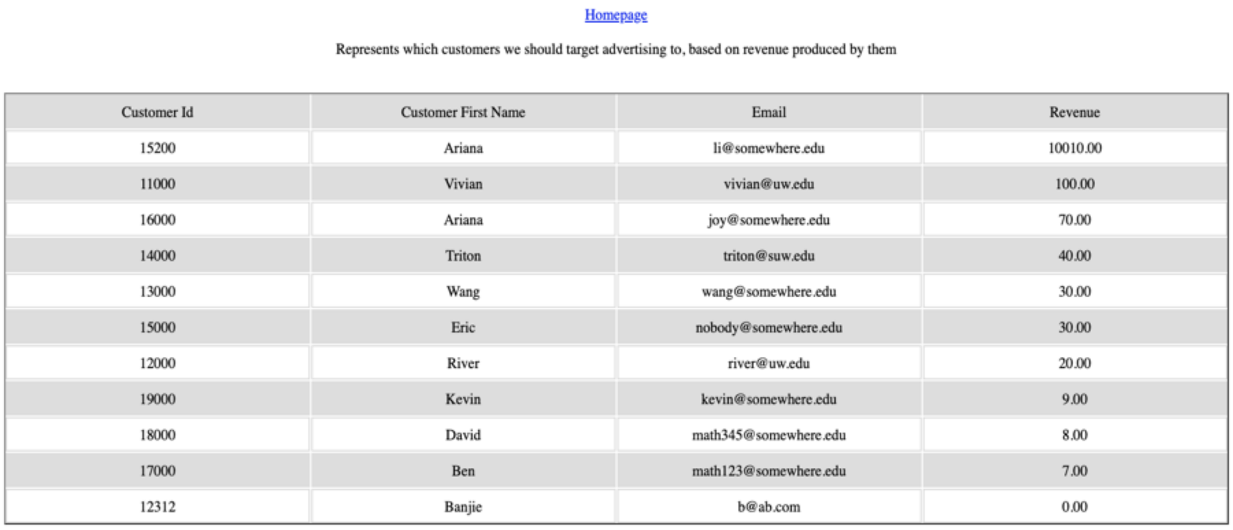
This query will be used to determine which customers should be targeted for advertising. It represents which customers buy from the store most. In real life, this information is useful in that we can run different ads to both loyal and lesser customers.



```
$query = "SELECT Cu.cust_id,Cu.cust_fname,Cu.email, SUM(O.total) REVENUE
FROM CUSTOMER Cu
INNER JOIN ORDERS O
ON O.cust_id = Cu.cust_id
GROUP BY Cu.cust_id, Cu.cust_fname, Cu.email
ORDER BY SUM(O.total) DESC";
```

Figure 6: Query 3

Table 3: Result of Query 3



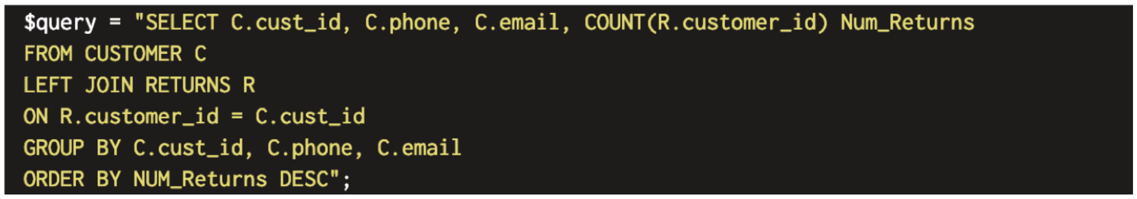
Homepage

Represents which customers we should target advertising to, based on revenue produced by them

| Customer Id | Customer First Name | Email | Revenue |
|---|---|---|---|
| 15200 | Ariana | li@somewhere.edu | 10010.00 |
| 11000 | Vivian | vivian@uw.edu | 100.00 |
| 16000 | Ariana | joy@somewhere.edu | 70.00 |
| 14000 | Triton | triton@suw.edu | 40.00 |
| 13000 | Wang | wang@somewhere.edu | 30.00 |
| 15000 | Eric | nobody@somewhere.edu | 30.00 |
| 12000 | River | river@uw.edu | 20.00 |
| 19000 | Kevin | kevin@somewhere.edu | 9.00 |
| 18000 | David | math345@somewhere.edu | 8.00 |
| 17000 | Ben | math123@somewhere.edu | 7.00 |
| 12312 | Banjie | b@ab.com | 0.00 |

## 5.4 Query 4 : Monitor Returns

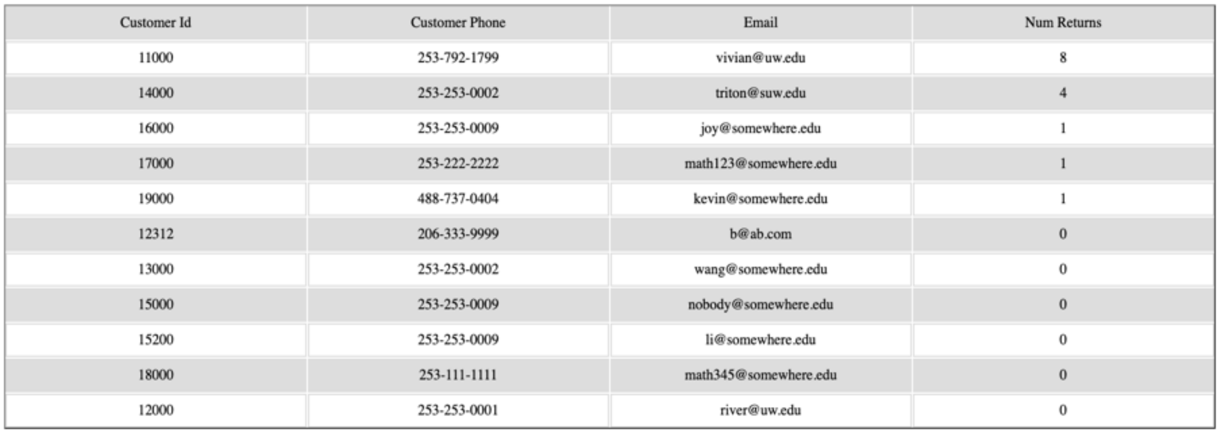
This query will be used to determine which customers are bad. Customers that are bad frequently make returns. The user can use *customerID* to identify which cards have been used by this customer, and then blacklist this customers cards.



```
$query = "SELECT C.cust_id, C.phone, C.email, COUNT(R.customer_id) Num_Returns
FROM CUSTOMER C
LEFT JOIN RETURNS R
ON R.customer_id = C.cust_id
GROUP BY C.cust_id, C.phone, C.email
ORDER BY NUM_Returns DESC";
```

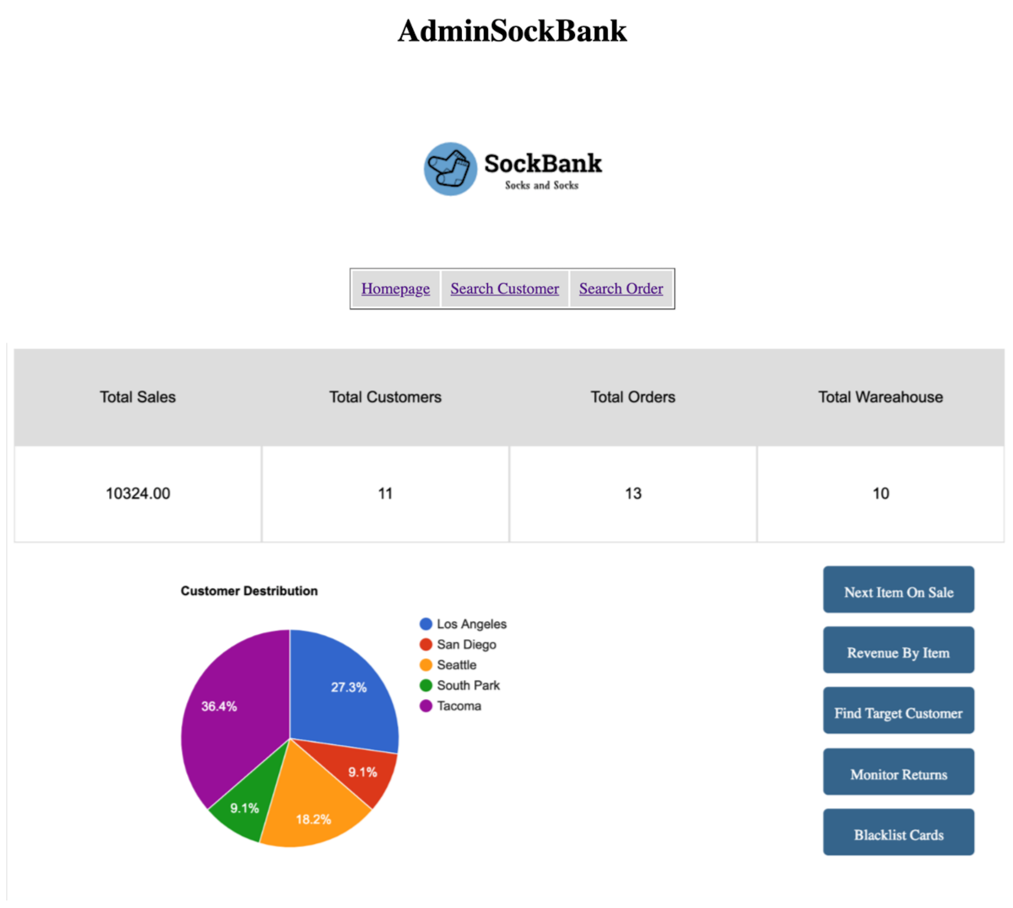
Figure 7: Query 1

Table 4: Result of Query 4



| Customer Id | Customer Phone | Email | Num Returns |
|---|---|---|---|
| 11000 | 253-792-1799 | vivian@uw.edu | 8 |
| 14000 | 253-253-0002 | triton@suw.edu | 4 |
| 16000 | 253-253-0009 | joy@somewhere.edu | 1 |
| 17000 | 253-222-2222 | math123@somewhere.edu | 1 |
| 19000 | 488-737-0404 | kevin@somewhere.edu | 1 |
| 12312 | 206-333-9999 | b@ab.com | 0 |
| 13000 | 253-253-0002 | wang@somewhere.edu | 0 |
| 15000 | 253-253-0009 | nobody@somewhere.edu | 0 |
| 15200 | 253-253-0009 | li@somewhere.edu | 0 |
| 18000 | 253-111-1111 | math345@somewhere.edu | 0 |
| 12000 | 253-253-0001 | river@uw.edu | 0 |

## 6 Appendix E: Screen-shots of functional Web Interface



| Total Sales | Total Customers | Total Orders | Total Wareahouse |
|---|---|---|---|
| 10324.00 | 11 | 13 | 10 |

Figure 8: Homepage of SockBank

Figure 8 shows the home page of our SockBank. The home page displays general information about SockBank and helps the administration term to make decisions. The user can click the *Search Customer* to go the Search Customer page. The user is also able to Search Order page by clicking the *Search Order*. The homepage displays the amount of total sales in dollar, the number of total customers, the number of total orders and the number of total warehouses. The pie chart describes our customer distribution. Finally, the user can click the five buttons on the bottom-right to see more information. The results of the first 4 button are in Appendix D.

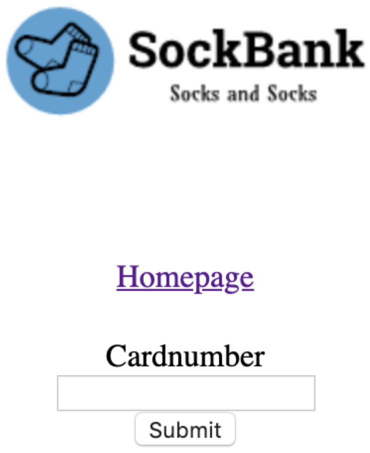


Figure 9: BlackList

Figure 9 shows the page when clicking ***Blacklist Card***. The user can put a customer's card into blacklist by entering the customer's card number. The user is able to go back to homepage by clicking homepage.

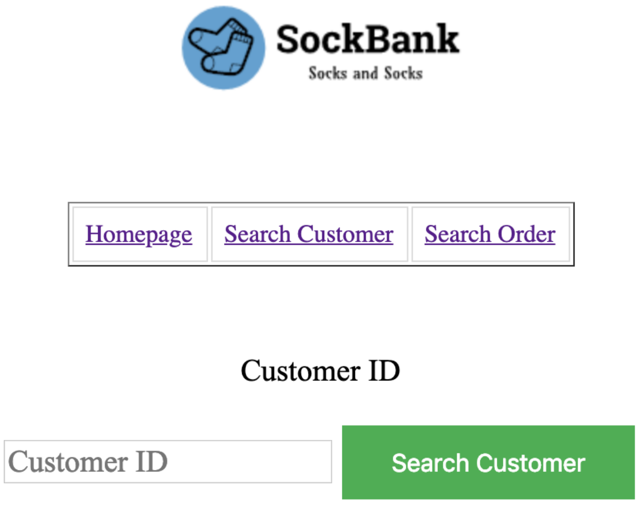
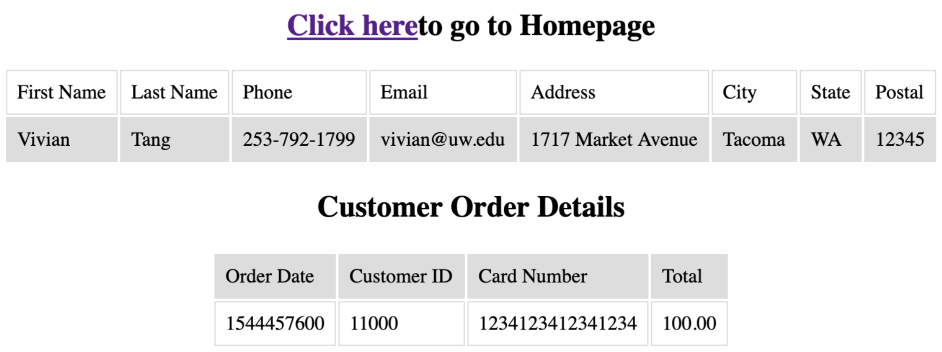


Figure 10: Search Customer of SockBank

Figure 10 shows the page when clicking ***Search Customer***. The user is able to search the customer 's information and orders by entering the customer's ID and clicking ***Search Customer***. The user is able to go back to homepage by clicking homepage.


**Click here**to go to Homepage

| First Name | Last Name | Phone | Email | Address | City | State | Postal |
|---|---|---|---|---|---|---|---|
| Vivian | Tang | 253-792-1799 | vivian@uw.edu | 1717 Market Avenue | Tacoma | WA | 12345 |

**Customer Order Details**

| Order Date | Customer ID | Card Number | Total |
|---|---|---|---|
| 1544457600 | 11000 | 1234123412341234 | 100.00 |

Figure 11: The Result Page of Search Customer When Enter 11000

Figure 11 shows an example output of Search Customer. In particular, Figure 11 shows the result of search Customer When entered 11000, which is the CustomerID of Vivian.

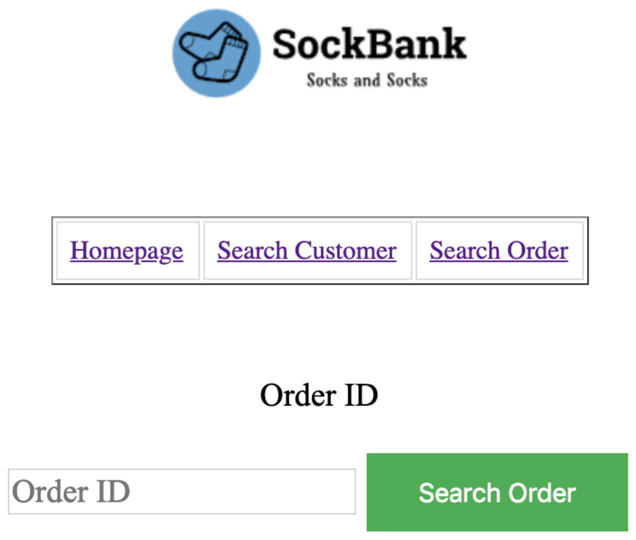


Figure 12: Search Order of SockBank

Figure 12 shows the page when clicking ***Search Order***. The user is able to search the Order 's details by entering the Order's ID and clicking ***Search Order***. The user is able to go back to homepage by clicking homepage.