TheMallCatalog Database

CPSC 471 L01, Winter 2017. Group 8

Mohammed Hameed <10149565>, Ryan Anderson <10156070>,
Arjun Cosare <30021317>, Matthew Powaschuk <30022573>

Abstract

This paper serves to explain the implementation of the database of our web-based mall stock organization system *TheMallCatalog*, more specifically, using a version of MySQL. This database services a system designed for users of a mall, increasing the convenience of their experience at the mall, should they choose to use our system beforehand. Our main users of concern being the mall store employees, customers, and a system administrator. To design this system, both an ER diagram and relational schema diagram were used and will be shown and analyzed in this paper. As we used multiple diagrams in the design of this system, any differences between them will be explored. All assumptions made in relation to our design will be clearly explained. Related to the database management system that we used, how we implemented it, with my MySQL 5.6.35 will be described along with all SQL queries used. Our user interface design was based on HTML, with the help of the bootstrap framework, and its design choices will furthermore be explored.

Introduction

"Sold out" are words we believe a customer should never have to hear unexpectedly, this feeling of having wasted a trip and many other inconveniences still trouble malls despite recent technological advances in mall systems. We would offer as a solution our *TheMallCatalog*, an online tool used by mall patrons and workers alike that serves as a way to communicate and treat mall inventory, otherwise scattered between stores, as an unified online database, with customers coming in to pick up items later, still getting that classic mall experience. Advantages of taking this approach is that malls can now offer new perks to customers previously unavailable to them, for example, stores can now offer online-only sales of items for special prices called store events. This system also conveniently functions with minimal work done by the workers as well, for the average mall worker, they can be designated to upload the store's stock to the database, which they would already be keeping track of, making this a trivial task for them. Additionally, whenever a new item is added, they upload that item to our system, and all corresponding information with it, name, cost, etc, as well as creating and maintaining store events. The Mall administrator is the overseer who manages the creation and deletion of stores, account management, as well as offering help to those stores in implementing this system.

Project Design

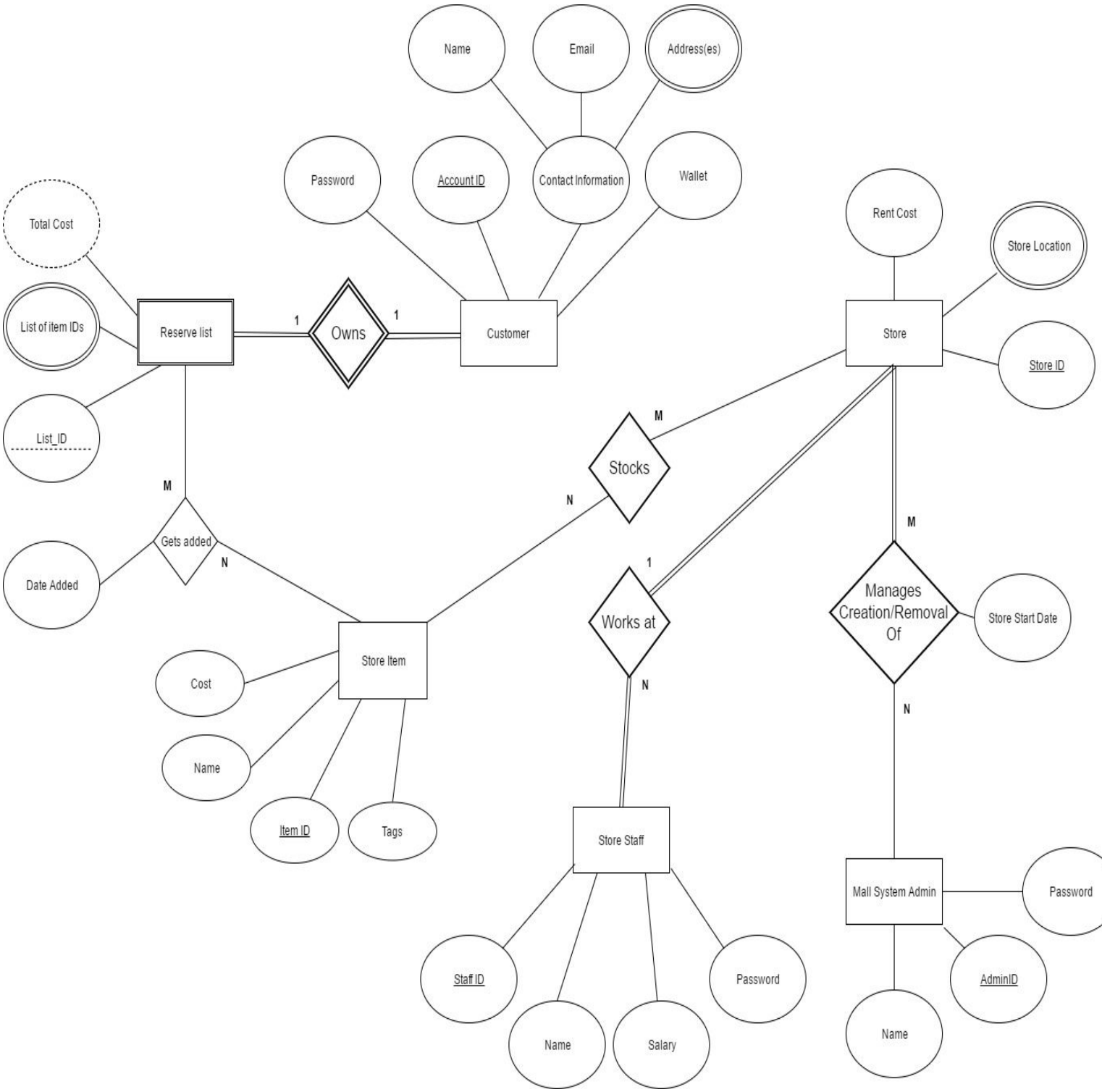
In this section, we will describe our users and the details of their functionality. Our first user is the customer, the ones who use our system for the convenience it offers to their shopping experience. Before diving into that however, it is important to know that all users are “accounts” in our system, where their username, password, and contact information are kept, this is the case to give us the ability to implement a login system. Onto the customers, each customer has a reserve list, which allows them to gather items they are interested in while they browse the mall catalog. In this catalog, they can search for items, and add them to their list from the same page. Within the catalog, the customer can search by store, item, or store and item simultaneously. They can view the name of the item, its cost, the quantity in stock, and the store in which the item is located. Occasionally, a store item’s cost will temporarily change in the customer catalog, this is accomplished through stores hosting store events, which are akin to sales. Once the customer is satisfied with the contents of their reserve lists, they head into the store to pay for their guaranteed items either through the funds in their wallet associated with their account, or another, more traditional, method. Additionally, a customer can pre-load funds into another user’s wallet, allowing for monetary gifts to be given through our system. The customer can delete their account at any time.

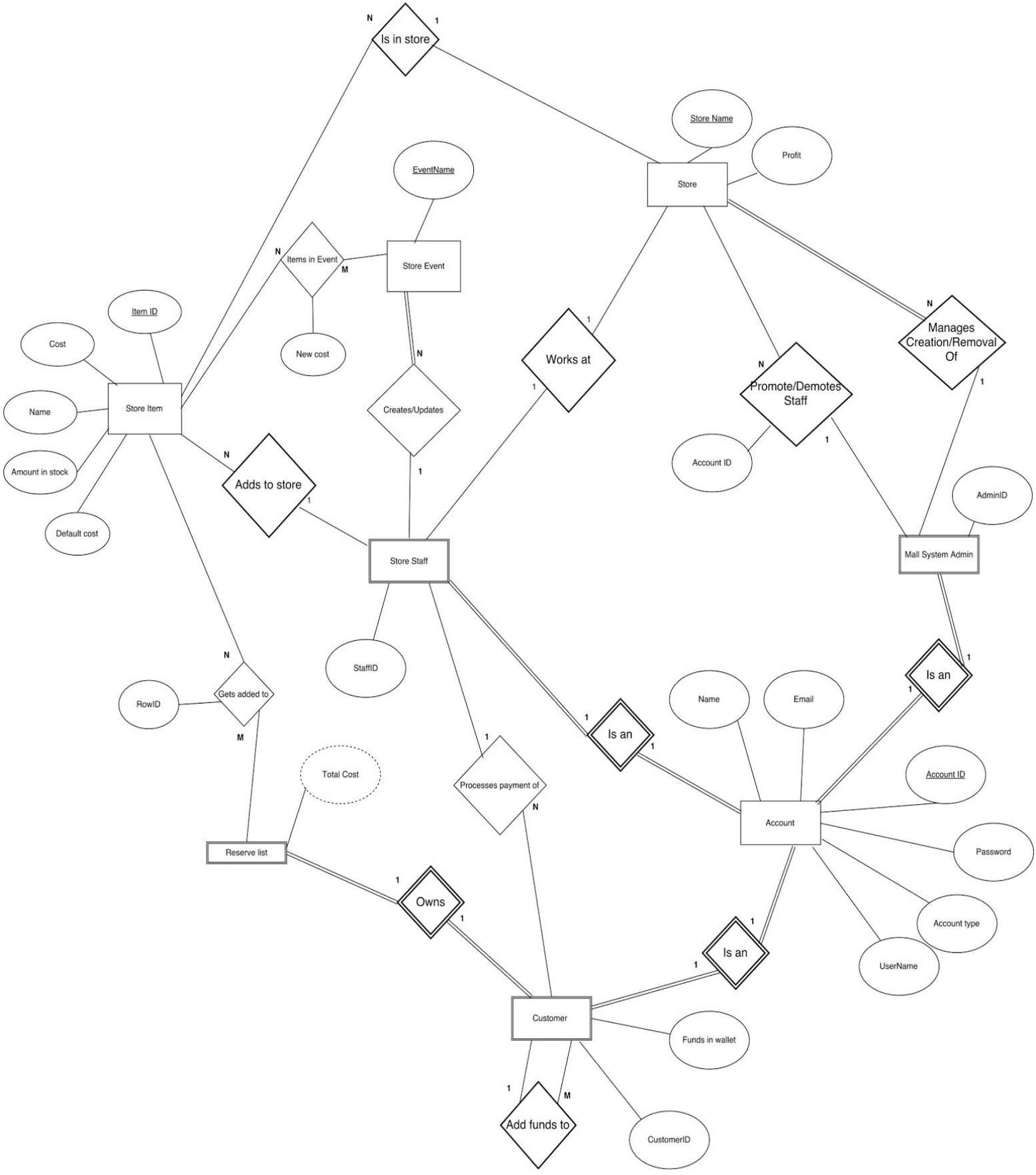
On to store workers, in this scenario, only one store worker account need access our system per store. Preferably the manager or supervisor would set it up, and all workers would access the same account. This account is in charge of updating or adding items to their store in our catalog, this would occur either when a new item is brought into the store, or the price or amount of an item changes, The worker logged into the staff account will also verify a customer’s order once they arrive in the store, and if the order is correct, they will process payment either through the customer’s wallet on our system, or another payment method. Finally, the staff account is used to set up and end store events, which is a function that allows them to dictate temporary new prices for items. These events end when the staff account deletes it, reverting the item’s cost back to it’s default.

Finally we have our system administrator, a mall employee whose job is to oversee the addition of stores to our system, as well as the addition or removal of the staff account from that store. More specifically, all accounts will exist as customer accounts when created, and it is the admin’s job to turn customer accounts into staff accounts. It is the responsibility of an admin to be up to date on what needs to be updated and when.

We first designed our system using two ER diagrams, one that we presented in class. As well as a final and more refined diagram. Find them both, in that order, on the next two pages.

Note: if any of the diagrams appear low-resolution, you may need to zoom in on the pdf document and it’ll clear up.


Name
Email
Address(es)
Password
Account ID
Contact Information
Wallet
Total Cost
List of item IDs
Reserve list
1
Owns
1
Customer
List_ID
Rent Cost
Store Location
Store
Store ID
M
Stocks
N
M
Gets added
N
Date Added
1
Works at
Manages Creation/Removal Of
Store Start Date
N
N
Store Item
Cost
Name
Item ID
Tags
Store Staff
Staff ID
Name
Salary
Password
Mall System Admin
Password
AdminID
Name


Is in store
Store Name
Profit
Store
EventName
Items in Event
Store Event
Item ID
Cost
New cost
Name
Store Item
Amount in stock
Default cost
Creates/Updates
Works at
Promote/Demotes Staff
Manages Creation/Removal Of
Account ID
Adds to store
AdminID
Mall System Admin
Store Staff
StaffID
Gets added to
RowID
Is an
Name
Email
Is an
Account ID
Total Cost
Processes payment of
Account
Password
Reserve list
Owns
Account type
UserName
Is an
Customer
Funds in wallet
Add funds to
CustomerID

ER comparisons

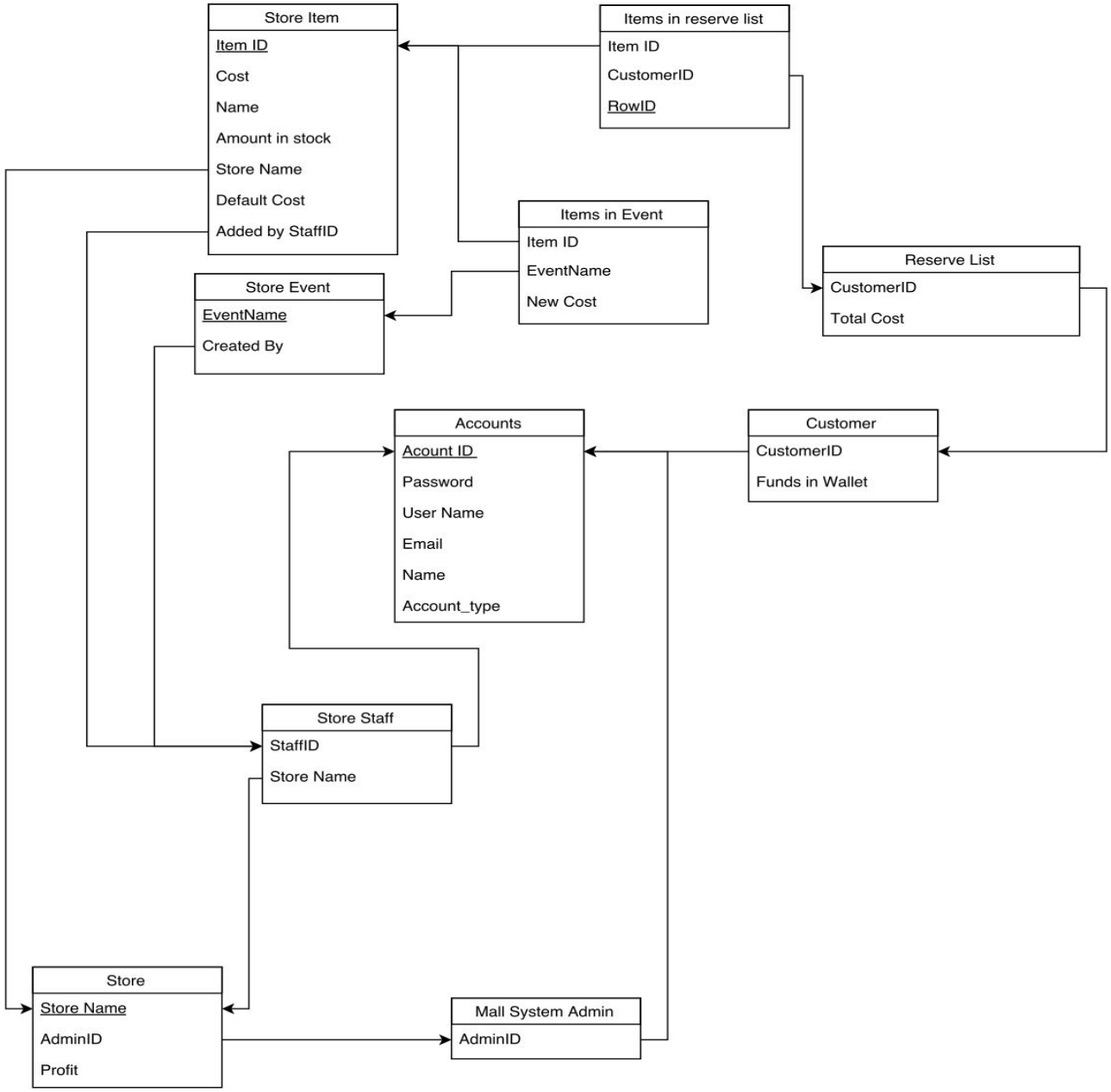
In this section we will describe the changes made from our initial, to our final ER diagram, going by entity

- Store
    - Primary Key StoreID changed to Store Name because we make the assumption that there is only 1 of each store in a mall.
    - Profit attribute added to represent payments made to the store through a customer's funds in wallet, the reason for this is explained under the "Other assumptions" section.
    - Rent cost, and store location removed as it serves no purpose in our final system.
    - Relationships between store and administrator are now N:1 respectively, as we assume that at most one admin is assigned to a store.
    - Store start date removed from it's creation/removal relationship with mall system admin, as our system has no need of knowing it.
- Store Item
    - Tags removed from store item, they are now searched by name.
    - Store items now have attribute "amount in stock"
    - A store item is in a store now, changed from stocking a store, as the verb "stocks", doesn't fit for an inanimate object.
    - Due to a new relationship with Store event, Store item now keeps track of its default cost, this is so that when an event ends, an item's cost can be reset to what it was previously.
- Store Event
    - Store Event is a new entity type that represents a group of items from the same store that are created by the store staff, through the store entity, and offers a special discount.
    - It has single attribute and primary key, EventID.
    - A single event may affect many store items and a store item can be in many events at a time, making this relationship M:N, this relationship will also store the item's 'new cost' during that event.
    - Each event is managed by a single store staff account, which it why "Creates/Updates" is N:1.
- Reserve List
    - List ID is removed, and lists will be searched by the owning customer, as each customer can only have one.
    - List of items IDs removed as this attribute is entirely captured by the M:N relationship Gets added to.
    - Reserve list has its identifying relationship with another weak entity, this is the case as although reserve list could be identified directly with an account, we do not want staff or admins to have access to a reserve list.
- Customer
    - AccountID changed to CustomerID, which is a foreign key pointing to AccountID in Account, customer is now a weak entity identified by Account.
    - Contact information moved to account, and is no longer composite within account.

    - Password moved to account.
    - Customers now have a relation with reserve list which allows them to pay for the items in it.
    - Customers now have a recursive relationship with itself where they can pre-load funds into their “wallet”, money that they have stored in their account, as well as the wallets of others, as an online gift.
    - `Wallet` changed to `funds in wallet` for clarity.
- Store Staff
    - StaffID is a foreign key pointing to AccountID in Account, Store Staff is now a weak entity identified by Account.
    - The Works at relationship is now 1:1, as there is only 1 staff account per store.
    - Store Staff now has relationship with Store Item, where the staff adds the item to the store that they work for.
    - Added relationship “Creates/updates” with Store event, meaning that the store staff is responsible for making their store’s events and changing them when they desire.
    - Name and salary removed as they are not necessary.
    - All store staff are accounts, Password now kept in account.
- Mall Administrator
    - AdminID is a foreign key pointing to AccountID in Account, Mall Administrator is now a weak entity identified by Account.
    - Mall administrator is now an account, with name and password moved to said entity type.
    - Admins now add and remove staff from stores, as well as still creating and managing them.
- Account
    - Account is an entity type to aid our login system. It holds core account information.
    - New attribute Account Type, represents whether the account is a customer, staff, or admin.
    - All “is an” relationships are now identifying relationships, for customers, staff and admins.

Implementation

In order to aid our implementation of this system, we turned our ER diagram into a relational schema diagram, find it below.



Design notes

In the process of transferring our ER diagram into a relational schema diagram, a few assumptions were made which are not entirely obvious at first, in this small section we will describe those assumptions here

- The “Add funds to” relationship does not add an attribute to customer, as it only changes the value of “funds in wallet”.
- RowID exists to uniquely identify a relationship, in its occurrence in items_in_reserve list, this is so that we can extract a single row.
- Within reserve list, total cost is not present in this diagram as it is derived and unnecessary to represent directly.
- CustomerID, StaffID, and AdminID, are all foreign keys to AccountID.
- Within reserve list, CustomerID is a foreign key to the CustomerID in customer, which in turn is a foreign key to AccountID.
- For the “Processes payments of” relationship, no attributes or relations are added, as a customer will tell the worker their username in person, and the worker will go into their account and subtract the money from their funds in wallet, or accept other payment.

Other assumptions

Here we will go over any other assumptions made in our implementation, that does not directly relate to either diagram.

- It is assumed that all staff are removed before deleting a store.
- It is assumed that an admin will promote an existing account to staff, not create a new one, meaning that a store worker will have to register for a customer account themselves before being added to a store. After promotion, they can only login and view the staff panel, no longer having access to customer.
- The “Profit” attribute in the store entity exists to store the funds from a customer’s wallet that was used to pay for items on their reserve list, since when adding funds to their wallet, their credit card payments are made to us, at the end of every month, we would pay the stores the equivalent to their profit attribute and reset it to 0.
- When an item is added to a reserve list, that item’s stock is also decremented within the store.
- Only one item of each name can be added to a reserve list.
- A staff account cannot see events, it is assumed they will keep track of it themselves, as they will likely be hosting this event offline in-store simultaneously.

DBMS

We rented our domain from NixiHost.com, which provided us with a centralized cPanel interface for the design of our website. Along with this, MySQL version 5.6.35 is also hosted on this web server, and is the database management system that we chose. We chose it because of its easy implementation into php. In addition, a program offered by cPanel ‘phpMyAdmin’ could be used to automatically translate SQL queries into php code. Allowing quick addition of queries into our code. In this next section we will list every single SQL statement in our system by file, as well as briefly explaining its purpose in the transaction it is used within.

STAFF.php

$sql = "SELECT Store_Name FROM Store_Staff WHERE '$StaffID' = Staff_ID;";

- Gets Store of Staff that is using the page.

$sql = "SELECT AccountID FROM Accounts WHERE '$customerUserName' = Username;";

- Gets Account ID of given customer, used to modify customers account.

$sql = "SELECT Funds_In_Wallet FROM Customers WHERE '$customerID' = Customer_ID;";

- Get’s the amount of funds that is associated with the given customer.

$sql = "SELECT Item_ID FROM Items_In_Reserve_List WHERE Item_ID IN (SELECT Item_ID FROM Store_Item WHERE Store_Name = '$StaffStore');";

- Get’s a list of items that are in the given customers reserveList that exist within the Staffs store.

$sql = "SELECT Cost FROM Store_Item WHERE '$ItemInList' = Item_ID;";

- Get’s the cost of a individuals item in the previous sql statement table. Used to total the cost of all the items a customer is checking out at the Staffs store.

$sql = "UPDATE Store SET Profit = (Profit + '$StoreItemCost') WHERE '$StaffStore' = Store_Name;";

- Updates the profit, or earnings, of the store which sold the item to the customer.

$sql = "UPDATE Customers SET Funds_In_Wallet = (Funds_In_Wallet - '$StoreItemCost') WHERE '$customerID' = Customer_ID;";

- Updates Customer's wallet to be reduced by the cost of all the items in the reserve list

$sql = "UPDATE Reserve_List SET Total_Cost = (Total_Cost - '$StoreItemCost') WHERE '$customerID' = Customer_ID;";

- Updates the reverse list of the given customer to be reduced by the cost of all the items that were removed from the reserve list.

$sql = "DELETE FROM Items_In_Reserve_List WHERE Item_ID IN (SELECT Item_ID FROM Store_Item WHERE Store_Name = '$StaffStore');";

- Removes the items that have been ‘Checked Out’ by the customer from the customers Reserve list.

$sql = "INSERT INTO `Store_Item` VALUES ('NULL', '$ItemName', '$Cost', '$Num', '$StaffID','$StaffStore',NULL,'$Cost')"

- When a staff wants to add an item to a store, adds the given variables to the Store_Items Realtion.

$sql = "SELECT Item_ID from Store_Item WHERE '$ItemName'= Name;";

- Gets the item ID assosiated with the ItemName. *Note ItemName is unique

$sql = "SELECT Item_ID FROM Items_In_Reserve_List WHERE '$ItemID' = Item_ID";

- Gets the Item_ID from the Items in the reserveList to check if a staff can remove an item, as only items that are not reserved by anyone can be removed.

$sql = "DELETE FROM Store_Item WHERE Name = '$ItemName' AND Store_Name = '$StaffStore'";

- Removes a Store_Item from the Store_Item relation with the given variables from The staff.

$sql = "UPDATE Store_Item SET Amount_In_Stock = '$Num' WHERE Item_ID = '$ItemID'";

- Updates the amount in stock variable of a store item to the value entered by the user.

$sql = "UPDATE Store_Item SET Default_Cost = '$Cost' WHERE Item_ID = '$ItemID'";

- Updates the amount of the default cost of a store item to the value entered by the user.

$sql = "SELECT Event_Name FROM Store_Item WHERE Item_ID = '$ItemID'";

- Gets the Event_Name of the Item being updated, used to check to see if the Default_Cost and Actual Cost of the Store_Item should be updated or not.
  Items on event will have the event price, even if they are updated later by a user.

$sql = "UPDATE Store_Item SET Cost = '$Cost' WHERE Item_ID = '$ItemID'";

- Updates the actual cost of a store item to the value entered in by the user.

ADMIN.php

$sql = "SELECT Store_Name FROM Store";

- Gets the name of all stores in our database, used when admin wants to view all current stores, or when an admin wants the retrieve the tuple of a store they wish to delete,

$sql = "INSERT INTO Store (Store_Name, AdminID, Profit) VALUES ('$store_name', '$id', '0');";

-Creates a store with the specified values in the attributes, used when the administrator creates a new store.

$sql = "DELETE FROM Store WHERE Store_Name = '$store_name';";

-Used to delete a store with name store_name from the database.

$sql = "SELECT Store_Name, username FROM Accounts, Store_Staff WHERE Staff_ID = AccountID";

-Retrieves all unique store-staff combinations, used by admin when they want to view all staff and the stores at which they work.

$sql = "SELECT username FROM Accounts, Store_Staff WHERE Store_Name = '$store_name' AND Staff_ID = AccountID";

-Similar query to the one above, but returns all staff for a single store, rather than all of them.

$sql = "SELECT AccountID FROM Accounts WHERE Username = '$staff_username';";

-Retrieves the generic “Account” that equals the username provided, used before either promoting that account to staff, or removing that account from a store.

$sql = "INSERT INTO Store_Staff (Staff_ID, Store_Name) VALUES ('$staff_id','$store_name');";

-Creates a store staff tuple with the ID of the generic account of the previous query at the store where the staff is to be added,

$sql = "UPDATE Accounts SET Account_Type = 'staff' WHERE AccountID = '$staff_id';";

- Changes the Account_Type attribute from customer to staff for the account being promoted.

$sql = "SELECT * FROM Store_Staff WHERE Staff_ID = '$staff_id' AND Store_Name = '$store_name';"

-Selects the Store_Staff tuple to be deleted from the store that has Store_Name = store_name

$sql = "DELETE FROM Store_Staff WHERE Staff_ID = '$staff_id' AND Store_Name = '$store_name';";

-Simply removes the specified staff member's tuple from the Store_Staff relation.

$sql = "UPDATE Accounts SET Account_Type = 'customer' WHERE AccountID = '$staff_id';";

-Demotes the Account_Type back to customer, if they are transferring to another store, they will be promoted again when added back.

USER_CATALOG.php/CATALOG.php

In terms, of SQL queries, we will treat these files as one because catalog.php is simply user_catalog.php but with less functionality, due to not having a reserve list available.

$sql = "SELECT * FROM Store_Item WHERE Name = '$item_name';";

-This is the query used if the user checked the radio box to search by item name only, retrieves all items with the name item_name.

$sql = "SELECT * FROM Store_Item WHERE Store_Name = '$store_name';";

-This is the query used if the user checked the radio box to search by store name only, retrieves all items from store with the name store_name.

$sql = "SELECT * FROM Store_Item WHERE Name = '$item_name' AND Store_Name = '$store_name';";

-This is the query used if the user checked the radio box to search by store name and item name, retrieves all items with name item_name from store with name store_name.

$sql = "SELECT * FROM Store_Item;";

-This is the query used if the fields are blank, it will display every item in our database.

The following statements are only present in user_catalog.php and relate to reserving an item.

$sql = "SELECT * FROM Store_Item WHERE Name = '$item_name' AND Store_Name = '$store_name';";

-This will retrieve the tuple of the item that the user requests to reserve, from the store that the user wishes to reserve from.

$sql = "SELECT * FROM Items_In_Reserve_List WHERE Item_ID = '$item_id' AND Customer_ID = '$cid';";

-This will retrieve any occurrences of the store item that already occur within the reserve list, as we make the assumption that a customer can only reserve one of each item, this will be used to check that.

$sql = "INSERT INTO Items_In_Reserve_List (Item_ID, Customer_ID, RowID) VALUES ('$item_id', '$cid', 'NULL');";

-This is the query that inserts the item requested into the customer's reserve list, by inserting into the relation representing the M:N relation.

$sql = "UPDATE Reserve_List SET `Total_Cost`=(`Total_Cost` + $item_cost) WHERE `Customer_ID` = $cid";

-This query will update the Total_Cost of the reserve list by adding the cost of the item added to it.

$sql = "UPDATE `Store_Item` SET `Amount_In_Stock`=(`Amount_In_Stock`- 1) WHERE `Item_ID` = $item_id";

-This query will decrement the Amount_In_Stock of the item reserved so that no other user can reserve this item while it is in this customer's reserve list.

LOGIN.php

$sql = "SELECT Username, Password, Account_Type, AccountID FROM Accounts WHERE Username = '$username' AND Password = '$password'";

- Gets the info for the account whom's password and username are the same as the one Entered in by the user.

EVENTS.php

$sql = "INSERT INTO Events (Event_Name, Created_By) VALUES ('$event_name', '$id');";

-This query is executed when a store staff attemps the create an event with event_name and id.

$sql = "DELETE FROM Events WHERE Event_Name = '$event_name'";

-This query is executed when a store staff wants to delete an event by name event_name.

$sql = "SELECT * FROM Items_In_Events WHERE Event_Name = '$event_name';";

-Retrieves all tuples of Items_In_Events from a specific event specified by event_name.

$sql2 = "UPDATE Store_Item SET Cost = '$cost' WHERE Item_ID = '$item_id';";

-This changes the active cost of item with item_id to the new cost specified by the event it is contained in.

$sql2 = "SELECT Default_Cost FROM Store_Item WHERE Item_ID = '$item_id';";

-This retrieves the item's default cost with Item_ID = $item_id, used when ending an event so that we know what the original cost of the item was.

$sql3 = "UPDATE Store_Item SET Cost = '$default_cost' WHERE Item_ID = '$item_id';";

-This sets the Store_item's cost back to it's default cost where Item_Id = $item_id.

The following queries are used when adding an item to a store event.

$sql = "SELECT Item_ID FROM Store_Item WHERE Name = '$item_name';";

- Retrieves the Item_ID associated with $item_name.

$sql = "INSERT INTO Items_In_Events (Event_Name, New_Cost, Item_ID) VALUES ('$event_name', '$new_item_cost', '$item_id');";

-Inserts the item into the relation that represents the M:N relationship between Items and events, along with the Event_Name and Item_ID, the new cost associated with this event is also added within the tuple.

$sql = "DELETE FROM Items_In_Events WHERE Event_Name = '$event_name' AND Item_ID = '$item_id'";

-This is the query used to delete a single item from an event, matching it by event_name and item_id.

REGISTER.php

$sql = "INSERT INTO Accounts (Password, Email, Username, name, Account_Type) VALUES ('$password', '$email', '$username', '$name', 'customer');";

- When an account is created insert an account into the database, tries to insert an account Values is later checked to see if this was successful.

$sql2 = "SELECT AccountID FROM Accounts WHERE Username = '$username';";

- Gets the AccountID for the given user name. *note userName is unique.

$sql = "INSERT INTO Customers (Customer_ID) VALUES ('$customer_id');";

- If it is new account the account is inserted into the customer's relation, all new accounts Are customers.

$sql2 = "INSERT INTO Reserve_List (Customer_ID, Total_Cost) VALUES ('$customer_id', '0');";

- Creates (adds) a reserve list for the new account.

REMOVE_ACCOUNT.php

$sql = "DELETE FROM Accounts WHERE AccountID = '$id';";

- Removes an account with the given AccountID from the Accounts relation
  *note this will cascade throughout the database when removed.

RESERVE_LIST.php

$sql = "SELECT * FROM Store_Item WHERE Name = '$delete_item'";

- Gets all store items where the name is the same as the one that is planned to be Deleted.

$sql = "DELETE FROM Items_In_Reserve_List WHERE Item_ID = '$item_id' AND Customer_ID = '$id';";

- Attempts to deleted items in the reserve list of the user which is the item_id.

$sql = "UPDATE Reserve_List SET Total_Cost=(Total_Cost - $item_cost) WHERE Customer_ID = '$id';";

- Adds the cost of the given item to the total cost of the given customers Reserve list.

$sql = "INSERT INTO Items_In_Reserve_List (Item_ID, Customer_ID, RowID) VALUES ('$item_id', '$id', 'NULL');";

- Inserts the given item into the Items in Reserve List relation to the given customers reserve list.

CUSTOMER.php

$sql = "SELECT Funds_In_Wallet FROM Customers WHERE Customer_ID = '$cid';";

- Gets the Funds in the wallet of the given customer.

$sql = "SELECT Total_Cost FROM Reserve_List WHERE Customer_ID = '$cid';";

- Gets the total cost of the reserve list of the given customer.

WALLET.php

$sql = "UPDATE `Customers` SET `Funds_In_Wallet`=(`Funds_In_Wallet`+ $amount_add) WHERE `Customer_ID` = $id";

-This query will add the amount specified to the customer's wallet, by incrementing it by amount_add.

The following queries are used when sending funds to another user

$sql = "SELECT AccountID FROM Accounts WHERE Username = '$send_username'";

-Retrieves the AccountID associated with the username that the calling customer intends to send funds to

$sql = "SELECT Funds_In_Wallet FROM Customers WHERE Customer_ID = '$idfrom'";

-Retrieves the current amount of money that the sending customer has in their wallet, this will be used to check if they have enough to send the amount requested.

$sql = "UPDATE `Customers` SET `Funds_In_Wallet`=(`Funds_In_Wallet`- $amount_send) WHERE `Customer_ID` = $idfrom";

-Removes the funds from the sending customer's account before sending it off.

$sql = "UPDATE `Customers` SET `Funds_In_Wallet`=(`Funds_In_Wallet`+ $amount_send) WHERE `Customer_ID` = $idto";

-Adds the funds sent to the account of the receiving customer.

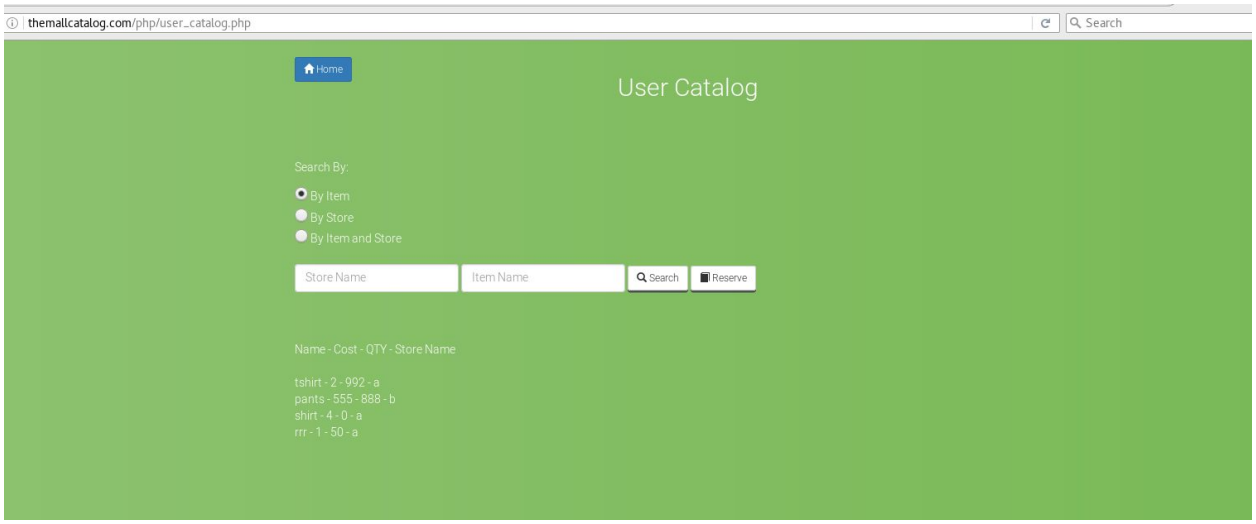
$sql = "UPDATE `Customers` SET `Funds_In_Wallet`=(`Funds_In_Wallet`+ $amount_send) WHERE `Customer_ID` = $idfrom";

-This query only executes if the database has failed to send the funds after already removing them from the sender, this simply puts the same amount of funds back into the sender's wallet.

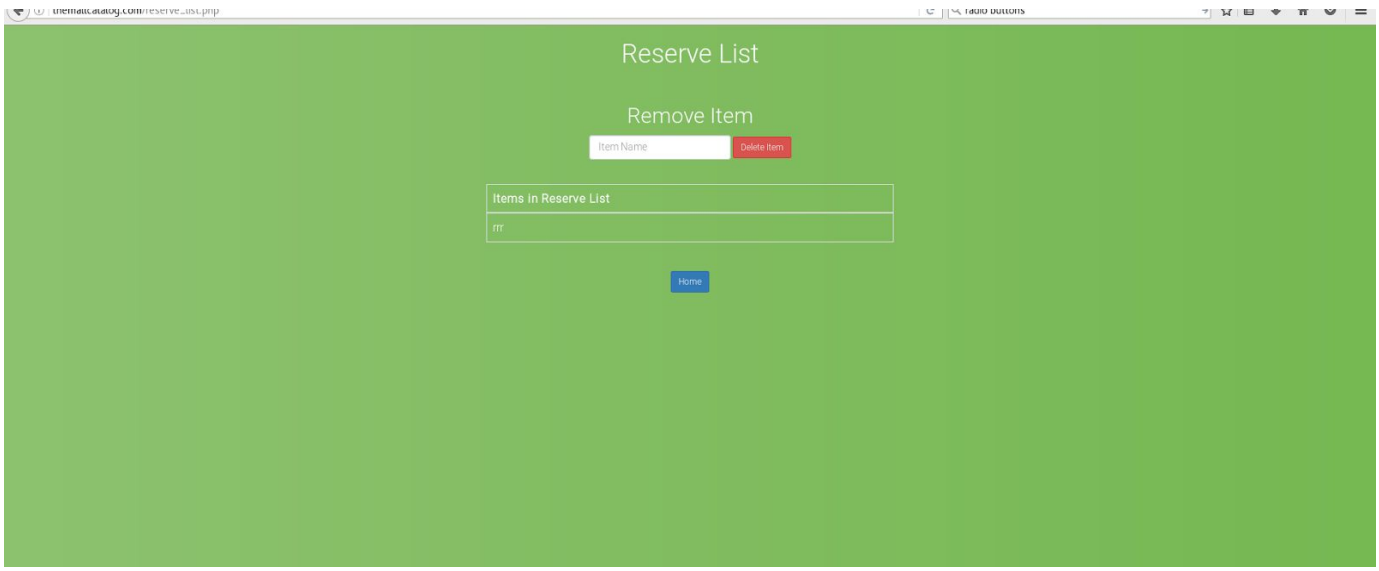
User Interface

With the majority of our backgrounds being composed of color #76b852, we intended our interface to have a unified, interconnected feel. With page transitioning that is easy on the eyes. To aid with our implementation, we made use of bootstrap, a simple open source HTML and CSS framework with which we set up our background and text font.

Our forms and buttons are simple HTML assets to which we link some php functionality, each button will post something different to a php file. Aside from basic forms and buttons, we also use radio buttons on the admin, staff, and catalog pages. These are used to specify certain options within the page, for example in catalog, the user can search by item, store, or item and store. These buttons will affect which of the forms are required to be filled out.



Additionally, some of our buttons are styled with colors or images, to help differentiate which buttons are associated with what form, and what the purpose of the button is, for example all buttons leading back to the home page are blue and all buttons that either log out or perform an action that the user should be wary of are red, such as deleting account or removing an item from the reserve list.



References:

Login System is based off the code found at:

Flat HTML5/CSS3 Login Form. (n.d.). Retrieved April 6, 2017, from http://codepen.io/colorlib/pen/rxddKy

Taken many code examples from:

HTML. (n.d.). Retrieved April 10, 2017, from https://www.w3schools.com/