Grocery Store Database

Jason Fagerberg, Ling Li, Matthew Mykietyn

COMP 2670 Database Management Systems

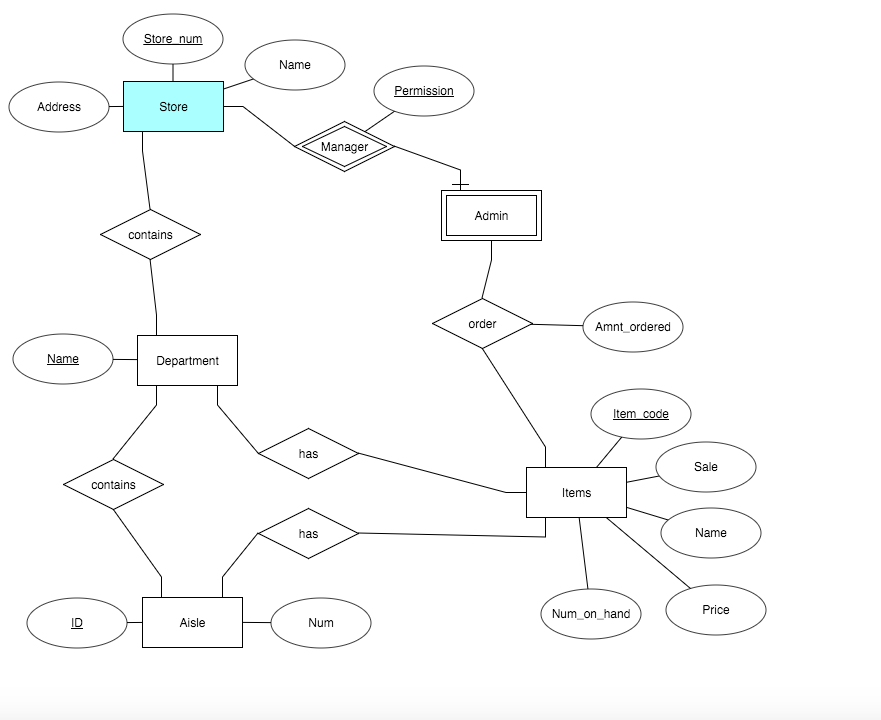
Spring 2016

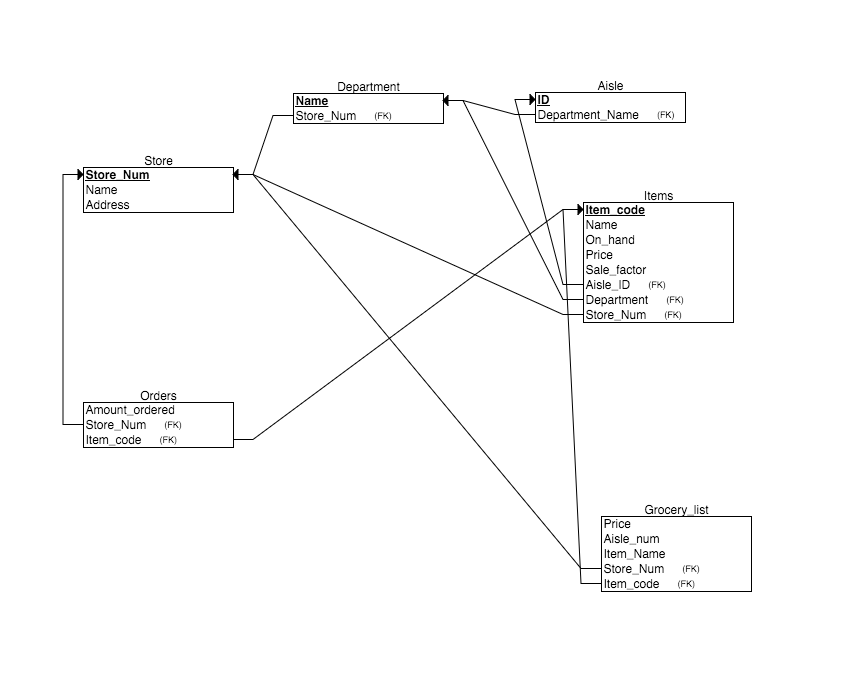
13 April 2016

**Abstract:**

Our goal for this project was to create a grocery store database with duel-functionality. We wanted this database to serve both the employees and the customers of any grocery store. On the company side, this database was meant to track inventory and handle future orders with ease. This database’s company inventory and ordering side should only be accessible by an admin who has the proper permissions. This grocery store database was meant to be able to handle the creation of a grocery list that a user could use to streamline their overall shopping experience by giving them the location of each item. We also wanted this grocery list to show a final price so the customer could budget more effectively.

What we provided is the underline database where an application could be built on top of as well as the SQL commands that would be needed. The base of our store is contained in the store, department, and aisle tables. The information in these tables is generally not directly searched for, rather they are meant to build relations with the other, more queried tables. The store contains the private information of store\_num where all of the queried tables are related to. This is built this way so this database could, in theory, handle multiple stores. The more query-able tables on the company side are the items table and the item\_orders table. Currently there is no check for permission to edit these tables. However, if there was an application to manage this database, then we would have the admin table, that is created by a relation to the store table, play a larger role in determining if the changes were allowed to be made. The items table simply contains all the items in a store as well as their location, amount left in store, and their price. The items\_orders tables contain the id of items that are currently on order and how much was ordered. The more searched user side table is the grocery\_list table. This table contains the ids of items in the store that a user may want to buy. I believe we have created a great substructure to solve the problem that we had laid out.

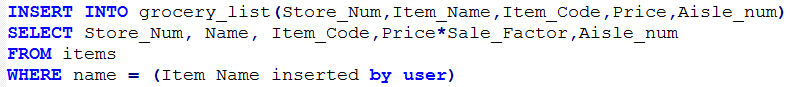
**ER Diagram:**

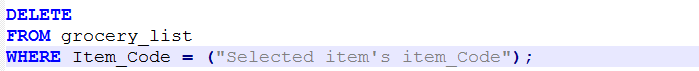
**Relations:**

**Samples/Examples:**

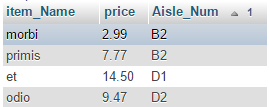
Since we did not have time to create a UI using PHP or any other application we will quickly go over a variety of SQL statements and how we would have implemented them. We will also provide screenshots of the returned tables in order to show that they do in fact work as we described.

**USER SIDE:** Our idea for the user side of this database was something like a web applet where you could actively update, remove and add items to your grocery list.

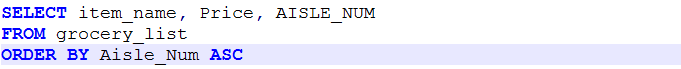
1. This of course would have buttons that were coded with simple commands such as



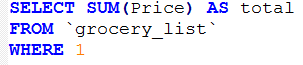
1. The list itself would hide nonessential information like the item\_code and the Store\_num just so the UI looks a little cleaner and streamlined.



(Example of info provided)



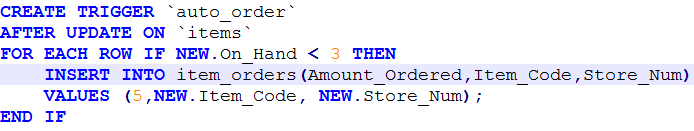
(Code to produce this)

1. On the bottom of the list we would have liked to have a live updating total price counter (Code to do this and test result)

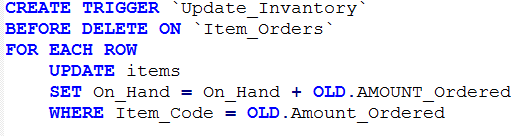
Overall the grocery\_list UI’s statements would be simple to code. This would have made the applet very simplistic and fast, however we did not have the appropriate time to design and code the UI.

**COMPANY SIDE:**

The idea for the company side involved some more complex but automatic transactions that would occur any time a barcode (which contains the item\_code) is scanned.

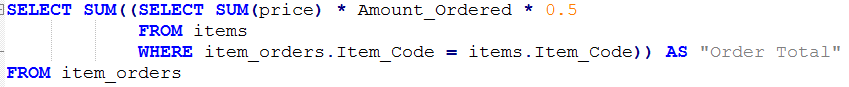
1. Any time an item is scanned on a register the system should check to see of that item is close to being out of stock. If it is the item should automatically be added to the list of items that are going to be ordered. We accomplished this by the use of the trigger shown before.

This code automatically adds any item that had an On\_Hand count of less than 3 after that item was updated by the register

1. Another automatic function we added was one that would take the Amount\_Ordered from the Item\_Orders list and update the items On\_Hand count with that Amount Ordered once the order was fulfilled or DELETED in an SQL sense.

When an order is received the admin accepting the shipment will scan the barcode on the pallet containing the items. This barcode would tell the database to delete all the items in that order list and on each of those deletes, this trigger will automatically add the amount ordered to the new On\_Hand count.

1. On an admin computer we would also like to make basic order information available. This information would include things like how much the order will cost and how many units there are incoming.



* 1. This query sums the price of all the ordered items multiplied by the amount ordered and then that sum is divided in half because usually stores operate with at least 50% profit margin.



* 1. This query returns the amount of items in an order and returns that value so he admin would know how many people to put on in order to successfully stock that order to the shelf.

**User Manual:**

Unfortunately, we have UI that we need to describe but we do have functionality built into our database. In theory there would be two user interfaces that we would need. Those would be a company side application and a user side application. Due to time constraints we were unable to get functioning examples of these two programs. So both interfaces would be connected to the same database, however the company side would have a specific login credentials based on a store number. This extra permission would allow the company to do things such as edit what the actual store contains and orders for new items.

The company side application would start with a login screen where an Admin would need to input the store number and the permission number. Our database has a sample set of these credentials which are a store number of 9818 and a permission number of 88958. After a successful login there would be an option menu where the admin would be able to select, Add Item, change item attribute, add to order, check in order, and register mode. Once the Add item option was selected a series of option menus would pop up asking to fill in the information of the item’s name, base price, department, and aisle number. An item code will automatically be identified to keep all the keys unique. The store number will be added automatically based on the login credentials. The edit item attribute button would have you scan an items barcode (get its item code) and then it would ask you what you want to change out of the options of, change inventory count, change location, and change item name. Each option would bring up a dialog box where the admin could change that items attributes in the database based on what the admin chose to change. Add to existing order would have the admin scan an item, then input the amount they want to order. This option would add that item’s information into the item Order table with the inputted amount they want to order. Check in order would have the admin scan in a barcode that would contain the item codes of all the incoming items in an order, and remove them from the order. There is a built in trigger that will update the inventory automatically. Register mode would open the register interface that would take each item code from its scanned barcode and then put it in its own grocer list table where the customer would be able to see their order and purchase it. Every time an item is scanned in this mode a trigger will automatically decrement it’s On hand count.

The user side would include a simple app where a grocery list can be built and seen before a person even walks into the store. There would be a list of buttons labeled, search of item, and remove item. By searching for an item name, a user can add a specific item to their list and see the total for their order automatically update. The remove button will remove an item from the grocery list that user selects. This will also automatically update the total order cost on the bottom of the screen.

**Discussion:**

Liked: Playing with the SQL, seeing what can be done with such simplistic functions

Disliked: Trying to like the database, UI development

Easiest: Generating random data

Hardest: Coordinating a team for one goal

What We Learned: It is very hard to get three people to work on one thing to reach one goal. It is also very difficult to make a usable UI and also link the database to it. Relations are extremely important in reality and one messed up relation can break a database. Building a comprehensive database is much harder in practice than it is in reality.

**Conclusion:**

There is still a lot to do with this project. Almost everything that is left to be done is in connecting and creating the UI to interact with the database. Right now everything this database can do is limited to only phpmyadmin and it’s built in SQL compiler. The SQL commands that we need to make the buttons on the UI work would actually be very simple Insert, delete, or Update buttons. So that aspect would not be that difficult to do. However there still much to do with learning php to link the database and creating a usable UI.

What we did produce is a dynamic database that updates based on actions taken upon another part of the database. There is functionality for a person to build a grocery list and see where all the items are and their price. We can also see how much a customer would pay for their order. On the other hand, we have the functionality to have an admin see the details of an order including its price, how much is coming in, and what is coming in. This database automatically adds to the item orders when the count on any one item falls to 2 or less. It also automatically updates when the order is deleted signaling that it has been delivered.