Robert Burrus

Alex Henry

CS405G Spring 2014 Project – Toys and Games

**Part I – Finalized Database Design**

**ER Diagram**

customerID

password

CustomerHasOrder

itemQuantity

orderID

quantity

CustomerOrder

orderStatus

timestamp

isManager

password

staffID

Staff

name

price

numberInStock

itemID

Customer

CartHasItem

Item

OrderHasItem

**Database Schema Design**

|  |  |
| --- | --- |
| OrderHasItem | |
| orderID | INT (PK) (FK references CustomerOrder.orderID) not null |
| itemID | INT (PK) (FK references Item.itemID) not null |
| quantity | INT not null |

|  |  |
| --- | --- |
| CartHasItem | |
| itemID | INT (PK) (FK references Item.itemID) not null |
| cartID | INT (PK) (FK references Customer.customerID) not null |
| itemQuantity | INT not null |

|  |  |
| --- | --- |
| Item | |
| itemID | INT (PK) not null |
| numberInStock | INT not null |
| name | Char(30) not null |
| price | INT not null |

|  |  |
| --- | --- |
| CustomerOrder | |
| orderID | INT (PK) auto-increment, not null |
| orderStatus | Char(30) not null |
| TimeStamp | Datetime |

|  |  |
| --- | --- |
| Staff | |
| staffID | INT (PK) not null |
| Password | Char(20) not null |
| isManager | Tinyint(1) not null |

|  |  |
| --- | --- |
| Customer | |
| customerID | INT (PK) not null |
| password | CHAR (20) not null |

|  |  |
| --- | --- |
| CustomerHasOrder | |
| customerID | INT (FK references Customer.customerID) not null |
| orderID | INT (PK) (FK references CustomerOrder.orderID) not null |

**Functional Dependencies / Form**

**OrderHasItem**

FD: Where A, B, and C are orderID, itemID, and quantity respectively, we have AB->C because the quantity of item in the order depends on not only the order, but also the item in the order. AB is our key.

Normalization: This relation is in BCNF because it is in 1NF and every determinant is a candidate key. We know it to be in 1NF because the domain of each attribute contains only atomic values and there are no multivalued attributes.

**CartHasItem**

FD: Where A, B, and C are itemID, cartID, and itemQuantity respectively, we have AB->C because the quantity of item in the cart depends on not only the order, but also the item in the order. AB is our key.

Normalization: This relation is in BCNF because it is in 1NF and every determinant is a candidate key. We know it to be in 1NF because the domain of each attribute contains only atomic values and there are no multivalued attributes.

**CustomerOrder**

FD: Where A, B, and C are orderID, orderStatus, and TimeStamp respectively, we have A->B and A->C because the order status and time stamp depend on the particular order we are looking at. A is our key.

Normalization: This relation is in BCNF because it is in 1NF and every determinant is a candidate key. We know it to be in 1NF because the domain of each attribute contains only atomic values and there are no multivalued attributes.

**Item**

FD: Where A, B, C, and D are itemID, numberInStock, name, and price respectively, we have A->B, A->C, and A->D because the number in stock, and name, and the price all depend on the particular item we are examining. A is our key.

Normalization: This relation is in BCNF because it is in 1NF and every determinant is a candidate key. We know it to be in 1NF because the domain of each attribute contains only atomic values and there are no multivalued attributes.

**Staff**

FD: Where A, B, and C are staffID, password, and isManager respectively, we have A->B and A->C because the password of the staff member and whether he/she is a manager depends on the particular staff member. A is our key.

Normalization: This relation is in BCNF because it is in 1NF and every determinant is a candidate key. We know it to be in 1NF because the domain of each attribute contains only atomic values and there are no multivalued attributes.

**Customer**

FD: Where A and B are customerID and password respectively, we have A->B because the password of the customer is dependent on the particular customer. A is our key.

Normalization: This relation is in BCNF because it is in 1NF and every determinant is a candidate key. We know it to be in 1NF because the domain of each attribute contains only atomic values and there are no multivalued attributes.

**CustomerHasOrder**

FD: Where A and B are customerID and orderID respectively, we have B->A because the order holds the information as to what customer placed it. If we have the orderID we can deduce the customerID, because a customer can have multiple orders the reverse of this statement does not hold. B is our key.

Normalization: This relation is in BCNF because it is in 1NF and every determinant is a candidate key. We know it to be in 1NF because the domain of each attribute contains only atomic values and there are no multivalued attributes.

**Part II – Description of Programs**

**Program Flow**

A visitor to our site (both customers and staff) starts at the “Project Page”. From there, they can either click on a link to shop unregistered, sign in or register as a customer, or sign in as a staff member. If they shop unregistered, they are redirected to a page listing all inventory but are unable to take any action. If they clicked the “Customer Page” link, they are redirected to sign in / register for shopping. If the customerID and password are valid they will continue on to the Registered Shop page where they can add items to their cart, view their cart, purchase the items in their cart, and view past orders. If the user was a staff member, they could click on the Staff Login link on the original Project Page and they would be prompted to enter their staffID and password. From the staff page, all staff would be able to view inventory, update inventory, and view / ship pending customer orders. If the staff member is a manager they will have to additional pages to look at: Sales Stats and Item Promotions. Sales stats gives the manager a look at the number of each item sold in the specified time frame. Item Promotion allows the manager to set new item prices.

**Data Structures**

We did not use any data structures in this project, only the regular php variables.

**Algorithms**

The only “algorithms” we used were nested for loops to query our databases for rows of information and then iterate over those rows to get specific columns. Outside of that, there were not any algorithms used that were more complicated than simple loop cosntructs.

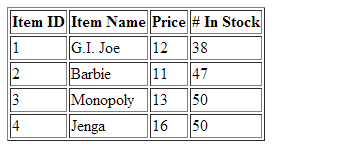
**Part III – Program Functions**

**Sample Input / Output**

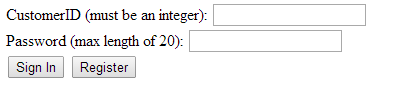
Home Page that both customers and staff see once visiting the site:



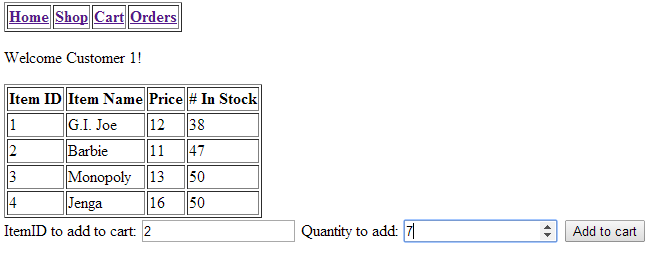
unregisteredShop.php:



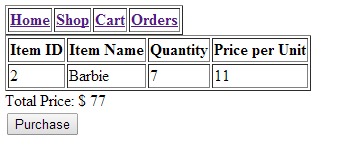
customerAccess.php - Login / Register:



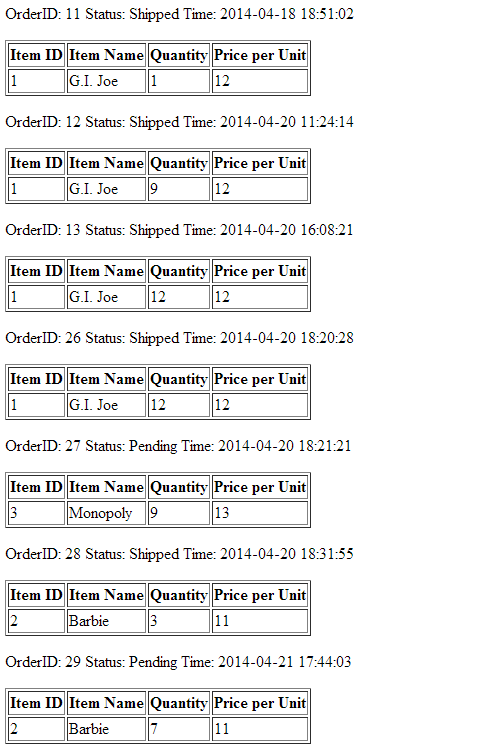
registeredShop.php – This page appears after a user signs in or registers. Notice the menu options at the top. Adding 7 Barbie’s to my Cart:



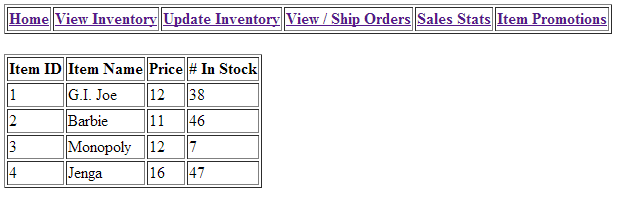
customerCart.php -- My Cart after adding the Barbie’s:



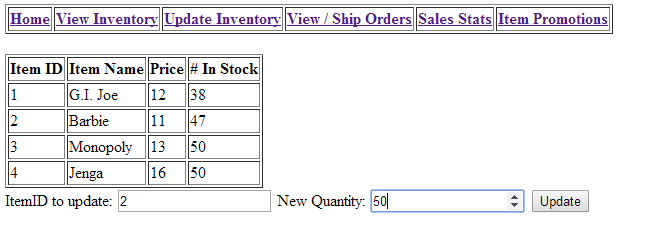
customerOrders.php -- Viewing Orders After Barbie Purchase (Order 29):

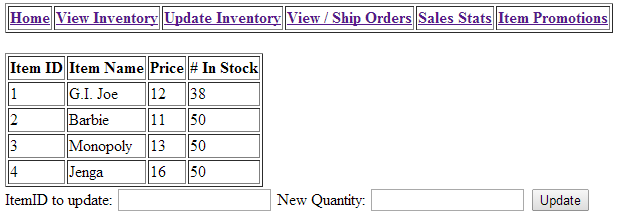


viewInventory.php – Manager viewing the inventory:

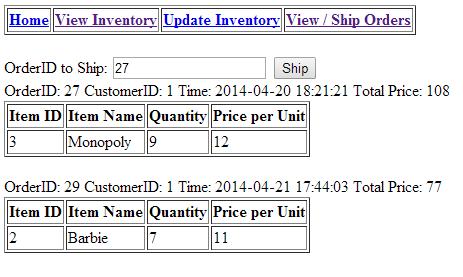


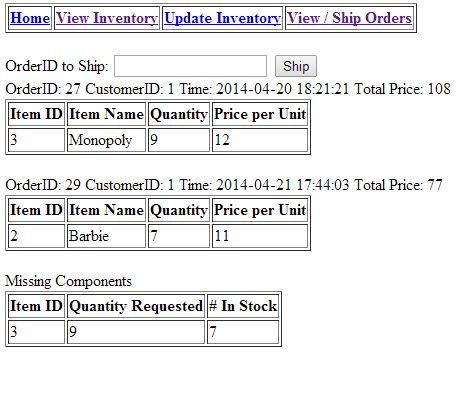
updateInventory.php -- Manager about to update Inventory (Before and After):



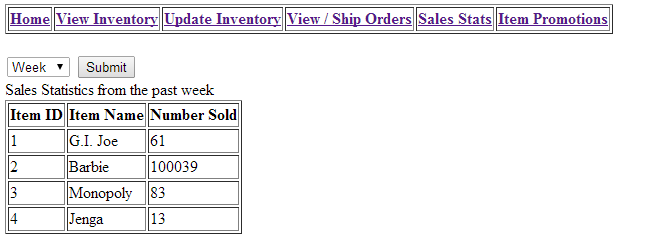


viewAndShipOrders.php -- Staff Viewing / Shipping Order 27 (all Items not in stock so we see the output of missing components:

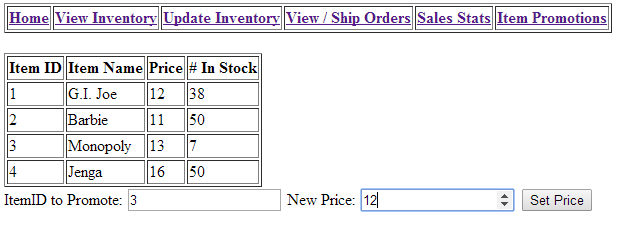


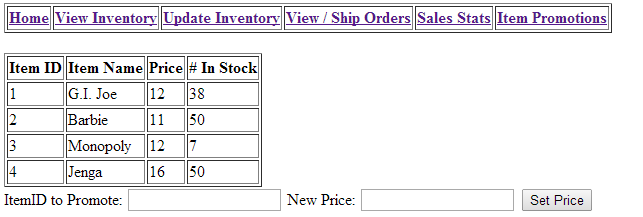


salesStatistics.php -- Manager looking at Sales Stats from past week:



itemPromotion.php -- Manager setting New Price for Monopoly:





Part IV – Testing

We manually tested each of our pages with input that was expected to work and input that would cause an error. Because of the way we implemented our PHP and HTML, entering input of the wrong type (i.e. entering a string where a number is required) would not be allowed. The button to “Submit” or “Update” would not execute until the correct type was entered for all input boxes.

All of our error handling is done within the PHP / HTML code. For example, in viewAndShipOrders.php entering an orderID that has either already shipped or does not exist brings up a message that tells the user “That is not an existing order number or that order has already shipped!”

Most of our testing was unit testing, checking to make sure each page worked properly as we created it. Our functions are largely split off into different pages and so testing each function individually was easy. At varying intervals throughout the project cycle we would go through a regression test on each function to ensure that new implementation of functions did not break previously working code.

**Project Experience**

Robert Burrus

Overall my project experience was good. I had never dealt with PHP or server-side databases. I gained a good understanding of how to implement a simple front-end application using a back-end database. PHP was easy to work with, as was writing scripts in MySQL. Good assignment to help tie down the concepts learned in class.

Alex Henry

As a whole, the project experience was rewarding and beneficial to my ability to work with databases. I can say with certainty that I have a more thorough and comprehensive understanding of database systems and what it takes to interact with them in a web environment. Prior to this project I had only used basic HTML & CSS, so the opportunity to improve my skills in the desirable field of web programming was an added bonus. I was happy to complete the project and very pleased with the results that Robert and I were able to achieve.