Michael Landis

Dr. Weitian Tong

CSCI 3432

April 29, 2024

Final Report

Introduction

My application for the final project is a retail management system. This encompasses all retail information from transactions, inventory, employees, products, and sales. I chose this retail management system because of my involvement with the Salty Dog T-shirt shop last summer and I could see some of the issues present with the current system for restocking and keeping locations with a high enough inventory. This project could also be used for other retail stores, but it tackled some of the issues that I saw working in retail. Since the main issues were with maintaining inventory counts, I put my main focus on making restocking easier and creating automated restock sheets based on sales. I also wanted this system to be able to act as the system to execute sales.

Main Components:

1. Employees can be created and deleted. To create an employee, the user must input the name, primary work location, and position title – which will change their privileges in the future.
2. Products can be added to the database, but not deleted because old products will still exist in transactions and people can return old products. When creating a new product, inventory details must be set for each location so the restocking system can properly function. Users can view products and the recent transactions that the products were involved in along with the inventory quantities at each store location.
3. Transactions can be executed and the inventory of the store location is updated accordingly. Each transaction happens at a store by an employee and each product on the transaction is kept in the sale table, where the quantity of each product on a transaction is tracked. Users can view recent transactions and inspect them for what was sold, when, where, and any notes that were left on the transactions.
4. Inventory for the warehouse can be updated manually, as in my application the products come to the warehouse from the printshop where each product must be counted to ensure there are no errors in the newly printed shirts or products. Inventory for each store location comes from the warehouse and can be restocked based on the location restock quantity or from the product sales from the previous day, especially helpful for daily restocks at the beginning of the day, which happened every day at my job.
5. Users can look at the most sold products from the past day, week, and month along with the total sales from each day based on a store location.

Database Details

My database uses 6 tables – Employee, Location, Transaction, ProductsSold (AKA Sales), Inventory, and Product.

The relationships between the entities are the following:

1. An Employee can work at many locations along with executing many transactions. There should be no employee who exists without working at a location, but an employee can exist without completing any transactions. There is a foreign key in the transaction table that corresponds to an employee. The employee and location are connected by a relation table so that there can be many employees at each location and one employee can work at many locations.
2. The products sold table has 2 foreign keys, one for the transaction that it is a part of and one for the product that it is tracking, each product in a transaction has a products sold entry that tracks the quantity sold and the price of those products. This table creates a many-to-many relationship between the transaction and product tables.
3. The transaction table also has 2 foreign keys, relating an employee and location to each transaction. A transaction cannot exist without both of these so if an employee is deleted, the manager of the location is replaced as the employee who completed the transaction.
4. The inventory table relates the product and location table, keeping track of the inventory for each product at the store along with the reorder level to keep track of the minimum stock quantity for each product.

Relation Model with constraints:

- Employee
    - EmployeeID – auto-incrementing integer, primary key
    - First Name – string – not null
    - Last Name – string – not null
    - Position – string – not null
- Location
    - LocationID – auto-incrementing integer, primary key
    - Location Name – string – not null, unique
    - Address – string – not null
    - City – string – not null
    - State – string – not null

- EmployeeLocationAssignment (For many-to-many between Employee and Location)
    - EmployeeID – integer, foreign key – not null
    - LocationID – integer, foreign key - not null
- Transaction
    - TransactionID – auto-incrementing integer, pimary key
    - Date – date – not null
    - Total – numeric(10, 2) – not null
    - Payment Method – string – not null
    - EmployeeID – integer, foreign key – not null

    - LocationID – integer, foreign key – not null
    - Notes – string
- Product
    - ProductID – auto-incrementing integer, primary key
    - Product Name – string – not null
    - Description – string
    - Price – numeric(10, 2) – not null
    - Size – string – not null
- ProductsSold
    - SaleID – auto-incrementing integer, primary key
    - TransactionID – integer, foreign key – not null
    - ProductID – integer, foreign key – not null
    - Quantity – integer – not null
    - Total Price – numeric(10, 2) – not null
- Inventory
    - LocationID – integer, foreign key – not null
    - ProductID – integer, foreign key – not null
    - Storck Quantity – integer – not null
    - Reorder Level – integer – not null

Boyce-Codd Normal Form or Third Normal Form:

My database does not comply with either of these. This is because in my Product table, I have the product name, which includes the shirt type and the shirt color. If I had separated these two from the product name, then my database would be of BCNF as I have no dependencies that occur with a non-superkey.

Functionality

Basic Functions:

1. Create and delete employees
2. Create product entities
3. Update warehouse inventory
4. Search products
5. Search transactions by location
6. Search inventory by location
7. Analyze transactions and sales by location, using aggregate queries

Advanced Functions:

1. Restocking a location with sufficient inventory based on either the set reorder level or on the previous days' sales. I did this by looking for the products sold from the previous day using an aggregate query, and then creating a transfer form that the warehouse would use to make sure all the correct products were sent to the store. Once sent, these products would be moved from the warehouse to the location in the inventory table.

2. Once a transaction begins, the quantity of products sold must be taken from the inventory to keep the inventory consistent. Once the products were sold they would be taken from inventory and this transaction would be saved for future analysis. The information from each transaction is used alongside the restocking program to make sure every product is restocked correctly. A transaction can also allow a store to go below their inventory as it is being tracked because there is a chance that discrepancies happen, but if there is a negative inventory an alert would be shown on the inventory table for that location showing that the inventory of the specific product needs to be checked.

Implementation

The backend for my project was implemented using Java with a Spring framework as a Maven project for creating requests to the database and transferring information back to the web interface. I used a PostgreSQL database as it is something I have worked with before and its implementation with Java is simple, using the Java SQL library. I used Java servlets to transfer this information to and from the backend. The frontend web interface was made using HTML and CSS inside JavaServer Pages. I used JSPs because they made it easy to redirect the user from one page to another and to allow each page to present the needed information for the task at hand. JSPs also made calling servlets and getting information from these servlets very easy through requests, responses, and session details. The only confirmation I did of users was checking that users sign in with an ID that is in the employee database. I did that because, in real practice, there would be a login page before this for each location to log in to at the beginning of the day, and then each employee would input their ID once the store was logged in.

How to Use

Begin by importing the database that has preset values for all tables. Once the database is imported, change the DatabaseConnection.java file so that the username, password, and URL are correct for the environment you are working in. Be sure the username and password are to a user that has full access to the database, if not, then some of the functions in the web app will not work.

Navigate to the root directory in the command prompt (rms2) and run the following command:

```
mvn clean install
```

This will install all the dependencies for the project. After this, run the following command:

```
mvn tomcat7:run
```

The program will not be accessible through localhost:8080/rms2/ in your browser. Use the ID 1 to enter the program and you will be logged in as Jessica Sampson and will have access to all functions of the web app. As of submission, the database has employees up to an ID of 21 so anything between 1 and 21 will work, but will be using different store locations for the transactions that are created.

Experiences

Throughout the creation of this project, I learned a lot, from creating a database management system to creating a full-stack project. I was familiar with creating a database system before this so this was a true presentation of what I have learned through this class. On the other hand, I have never created a web interface or full-stack project. I had to watch many YouTube videos and look at many examples of

how to use Spring and Maven, the ones posted to the course website were helpful when looking at design and the project from my Software Engineering class helped create a project that is built correctly, sending and receiving information from the right places. I also learned how to use HTML, CSS, and JSPs through the use of servlets. This was not only challenging, but it was rewarding. I enjoyed seeing the progress that I made in creating this project. Going back and redoing this project, I would have made sure that I was consistent with Boyce-Codd Normal From as it would have been pretty easy to obtain from where I stand now. I also would have created a location login page before the employee logs into their account.

References

Throughout the creation of my database, I used the class website to determine my foreign key constraints and the basic design of my database. While making my web interface I used my project from Software Engineering as a baseline as I have never made a web interface before. Along with this, I had to use Google to learn how to use servlets and create my Maven Spring project. There was no single source I used, but I needed help on how to connect each part of my project since this was all new to me. When I ran into issues, Stack Overflow was my go-to in my process to fix them after I tried debugging on my own. I also used YouTube to learn how to create and build a Spring project with Maven.