**CSCE 4523 Database Management Systems**

**Homework 4**

**By: Shane Norden and Max Olvey**

**Objectives**

The objectives of this homework are to get further practice with high level languages and learn how to use the high level languages in oder to interact with a database via ODBC or JBDC. In our case, it was JDBC. The goal was to implement a database similar to homework one and write a program that could interact with sql from the command line. It should take inputs from the user and directly interact with the sql, then produce changes in a manner that is easy to read for the user.

**Approach**

We decided to use the example code as a base of sorts, however all it really ended up being a skeleton due to many of the already given functions being altered to fit our use cases. Many new imports were brought in so that new information could be used. In the main file, the user interacts with a switch case showing a table full of options. The user’s input is tracked with a scanner function. Depending on the users option, different functions would be called that would handle that specific case. In some cases, there were 2 functions.

For option 1 (find all available copies at a given bookstore), it would prompt the user for a bookstore and a city. From there, a function would be called that natural joined bookstore, book, and copy, and produce all books and prices at the bookstore location. The query itself would also return a Boolean function. It would return false if the query returned no information (ie the query or information asked for doesn’t meet the constraints given or asked for) and true if the resulting query does produce information. If the query does exist, the query would print and send a message confirming the query. If the query doesn’t exist, then it would print a message saying that the query does not exist.

Option 2 was significantly more complicated. Option 2 (Purchase an available copy from a particular bookstore) required a lot of information based on how the purchase table was created. Upon choosing option 2, it would ask the user for a book title they wish to purchase. From there it would call a Boolean function that would select the bookName, copyID, bookstoreName, and city from a natural joined book tale Boookstore, Book, and copy table. From there it would be similar to option one where if the query returned empty, it would state that the book isn’t available and end the function. If it was true, it would ask the user for the copyID they’d like to purchase. Once that occurred, it would call a function that would find the current date, the current time, and the number of rows in the purchase table then insert that values into the purchase table.

Option 3 went about as smoothly as it could, I just had to build a simple query to join purchase copy and bookstore together so I could list everything out.

Option 4 was less straightforward, because I had an issue with the query function. I had to instead use executeupdate so I could directly delete the requested purchase.

Number 5 was the most challenging for me because there were like three things happening at once. First I had to grab the data from the user (which was pretty simple), but then I had to calculate the new bookID for it. I settled on using the MAX function to return the biggest bookID and then just add one. I then call it again to get the bookID for the insert into the copy table, and use a similar trick for the copyID. I originally tried to make it auto increment inside of the sql file but it was giving me all kinds of issue so I pivoted to the described method.The other big issue I had was with the date formatting, but ended up using a similar approach to how Shane formatted his date for the purchase method.

Option 7 is simply an extension of functionality to show everything in all of the tables of the dataset to help with debugging, but I left it in so if you want to make edits then see it you still can.

**Results**

The scanner worked well as it took in the user inputs. There were some errors with determining whether the query returned was empty or not. Reading the ResultSet documentation further proved to be unhelpful as when the query returned nothing, it would return true. If it returned a query, it would be false. Further investigation proved that the truth logic was flawed however it was already implemented. The query function worked well except for when we needed to insert and delete data, so I used execute update instead. I had to build a function for getting the maximum id numbers for copy and book so I could correctly add a new id in without overlapping with previous entries.

Shane worked on options 1, 2, and 6 while Max did 3,4, 7, and 5.

**Typescript**

Include the typescript here (or upload it separately).