My Library Query GUI is programmed with “Swing” widget toolkit for Java.

The accomplishments for all the functional requirements will be discussed as followed,

1) Graphical User Interface and Overall Design

Download and import mysql-connector-java-8.0.15.jar to connect Java and MySQL.

The overall design is mainly based on following requirements.

First, I decide to put all the text search fields and functional buttons at the top of Graphical user interface. For the many buttons and text field, I create two classes (Item and Item2) implemented by JPanel to manage them

Then as the requirement 2, 3 and 5 ask for returning search results or updating results. we need one text field to three different tables to display the error information and results.

However, for the normal BorderLayOut design in JFrame, the “Center” region will extend as large as possible while other regions will be reduced too small to display. To solve this problem, I decide to encapsulate all the text search fields and functional buttons in a JPanel as a controlPanel, which is put in the north region. Thus, the north region can be controlled by the function of JPanel “setPreferredSize” to fix the size.

The other text result field and tables are encapsulated in another JPanel as a tablePanel, which is in the center region. Each Table is also wrapped by JPanel with vertical design by using BoxLayOut.Y_axis. All of them are divided by empty box within adding box. createRigidArea.

For the three tables, which are implemented by JTable, I use vectors of string to update the columns titles and vectors of vectors (Vector<Vector<String>>) to update the data

2) Book Search and Availability

As the requirement, my search interface should provide a single text search field to return the given search results.

To search for a book given any combination of ISBN, title or Authors. we must combine all the necessary table by using inner join:

“book inner join book_authors on book.Isbn = book_authors.Isbn inner join authors on book_authors.Author_id = authors.Author_id inner join book_loans on book_loans.Isbn = book.Isbn "

To support substring matching, we apply the syntax as below:

lower(book.title) like '%search %' , lower(author.name) like '%search %', lower(book.Isbn) like '%search %'

For the Book availability (is the book currently checked out?), we can use “if (a.date_in is null, 'No', 'Yes')”

Once we meet the problem that one book(book.Isbn) were borrowed twice or more times, we can try to filter out the old book_loans by using "select a.loan_id from book_loans a inner join book_loans where book_loans.isbn = a.isbn order by a.loan_id desc limit 1"

3) Book Loans

Checking out books:

After showing the result books in the book table, we can select one of the book to check out with typing in the card_id. I decide to share the text field of card id with Checking process because we may both need the card id keyword.

To generate a new unique primary key for loan_id, we can set loan_id as auto increment while creating the table book_loans.

I decide to use text area to return an error message. For example, when a borrower has reached the 3 books limit, it will return "ERROR: A Maximum of Three Borrowed Books". If a book has been borrowed, it will return "ERROR: A Maximum of Three Borrowed Books".

Checking in books:

To search a book that you want to check in, I supply three options, ISBN, Borrower Card_ID and Borrower name. We can get all potential results based on anyone of these three keywords. This step is similar with the book search.

Then we can select one of the potential results and finish checking in, the selected result will also be updated the "Date In" as today's date automatically.

Meanwhile, after pressing the "Check In" button, I hope we can update the fine amount in the fines table if the book is overdue. **Logically, we have to assume all the returned book are definitely in the fines table when initialize the testing cases.**

To avoid the repeating refresh of the fine_amount and fine_paid in step_5, I want to control the fine whose book was returned in this step, and update/refresh the fines later in step 5.

4) Borrower Management

In this step, I still supply three keywords to create a new account with the simple insert syntax: "insert into borrower (ssn, bname, address) VALUES('','','')"

To automatically generate new card_no primary keys, I still use the auto increment in the card_no of borrower table.

The text area will also return the error message "ERROR: The ssn has been used" when a new borrower is attempted withe same SSN

5) Fines

I separate this part as three sub steps: Fine Update, Pay Fine and Fine Filter.

For sub step 1, as mentioned in step 3, I decide to update or refresh the fines of loan_id which is not returned in this step. Because after returning the book, the fine amount will be fixed. It is meaningless to refresh and update them daily.

I select and insert all the overdue loan_id which is not returned :

“select loan_id from book_loans where loan_id not in (select loan_id from fines) and date_in is null and current_date() > due_date” and "insert into fines(loan_id, paid) values (" + loan_id + ", '0')"

For sub step 2, I design a button “Pay Fine” to update the unpaid loan_id to paid by selecting a row you want to pay in the check in table, because all the book loan information will be showed in that table.

“update fines a inner join book_loans b on a.loan_id = b.loan_id set paid = 1 where a.loan_id = '" + loan_id + ’”;

At the same time, the fine table will return the borrower card ID and fine amount history.

“select card_id, sum(fine_amt) from book_loans a inner join fines b on a.loan_id = b.loan_id group by card_id”

For sub step 3, I design a button “Fine filter” to filter out the previously paid fines, the refreshed fine amount will be returned in the fine table.

"select card_id, sum(fine_amt) from book_loans a inner join fines b on a.loan_id = b.loan_id " + "where paid = 0 group by card_id;"

After all, to verify my designation of the GUI, I also write a database file (library.sql) based on supplied book.csv and borrowers.csv to simulate all the possible condition we can meet