**DATABASE DESIGN PROJECT 1**

By: Dhananjay Singh (dxs145530)

UTD Id: 2021250625

1. ***Requirement Specifications*:**

**1) GUI [~~25 points~~][10 points]**(15 points were re-assigned for the extra FINES tracking requirement)

All interface with the Library (queries, updates, deletes, etc.) must be done from a graphical user interface of your original design. Your GUI application will interface with the Library database via an appropriate MySQL connector. Initial database creation and population may be done from command line or other admin tool.

**2) Book Search and Availability [25 points]**

Using your GUI, be able to search for a book, given any combination of **book\_id**, **title**, and/or **Author(s)**, which may be either **author\_name** **OR** (radio\_button?) any combination of parts of an author's name. Your query should support substring matching. You should then display:

* book\_id
* title
* author(s)
* branch\_id
* How many copies are owned by a specified branch
* Book availability at each branch (i.e. How many copies not already checked out?).

The multiple copies at different branch locations should display on separate lines, to facilitate location-specific checkout. However, all authors of a book should be displayed on the same line.

**3) Book Loans [25 points]**

**Checking Out Books**

* Using your GUI, be able to check out a book, given the combination of BOOK\_COPIES(book\_id, branch\_id) and BORROWER(Card\_no), i.e. create a new tuple in BOOK\_LOANS.  Generate a new unique primary key for loan\_id. The date\_out should be today’s date. The due\_date should be 14 days after the date\_out.
* Each BORROWER is permitted a maximum of 3 BOOK\_LOANS. If a BORROWER already has 3 BOOK\_LOANS, then the checkout (i.e. create new BOOK\_LOANS tuple) should *fail* and return a useful error message.
* If the number of BOOK\_LOANS for a given book at a branch already equals the No\_of\_copies (i.e. There are no more book copies available at your library\_branch), then the checkout should *fail* and return a useful error message.

**Checking In Books**

* Using your GUI, be able to check in a book. Be able to locate BOOK\_LOANS tuples by searching on any of book\_id, Card\_no, and/or any part of BORROWER name. Once located, provide a way of selecting one of potentially multiple results and a button (or menu item) to check in (i.e. enter a value for **date\_in** in corresponding BOOK\_LOANS tuple).

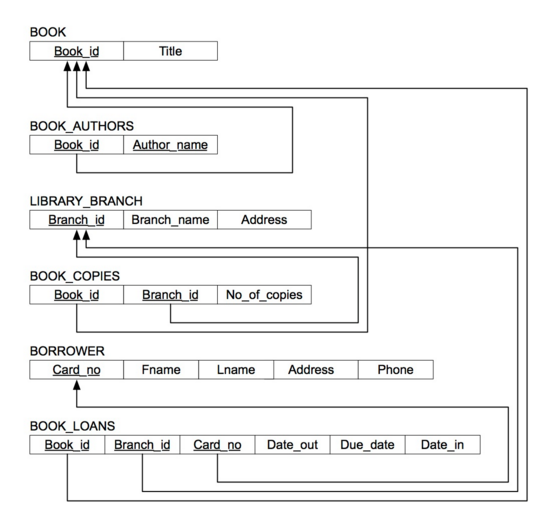
**4) Borrower Management [25 points]**

* Using your GUI, be able to create new borrowers in the system.
* All name and address attributes are required to create a new account (i.e. value must be not null).
* You must devise a way to automatically generate new **card\_no** primary keys for each new tuple that uses a compatible format with the existing borrower IDs.
* Borrowers are allowed to possess exactly one library card. If a new borrower is attempted withe same fname, lname, and address, then your system should *reject* and return a useful error message.

**5. FINES**

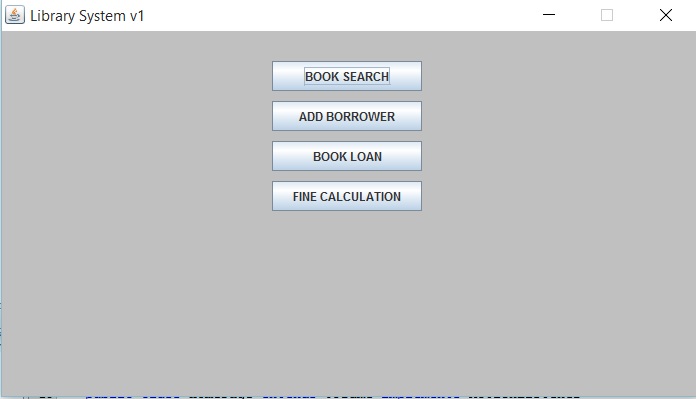
* Create a new table FINES(loan\_id, fine\_amt, paid)
  + The primary key loan\_id is also a **foreign key** that references BOOK\_LOANS(loan\_id)
  + fine\_amt attribute is a dollar amount that should have **two decimal places**.
  + paid attribute is a **boolean** value (or integer 0/1) that idicates whether a fine has been paid.
  + Fines are assessed at a rate of $0.25/day (twenty-five cents per day).
* You should provide a button, menu item, etc. that updates/refreshes entries in the FINES table. In reality, this would occur as a cron/batch script that executed daily.
* There are two scenarios for late books
  + (1) Late books that have been returned — the fine will be [(the difference in days between the due\_date and date\_in) \* $0.25].
  + (2) Late book that are still out — the *estimated* fine will be [(the difference between the due\_date and TODAY) \* $0.25].
* If a row already exists in FINES for a particular late BOOK\_LOANS record, then
  + If paid == FALSE, do not create a new row, only update the fine\_amt if different than current value.
  + If paid == TRUE, do nothing.
* Provide a mechanism for librarians to enter payment of fines (i.e. to update a FINES record where paid == TRUE)
  + Do not allow payment of a fine for books that are not yet returned.
  + Display of Fines should be grouped by card\_no. i.e. SUM the fine\_amt for each Borrower.
  + Display of Fines should provide a mechanism to filter ￼out previously paid fines (either by default or choice).

**The schema for this application is as follows :**

****

* 1. **The Home Screen:**

The requirement seems best suited for a stand-alone operation where-in the functions /features to be provided can be navigated through to from a single screen as follows:

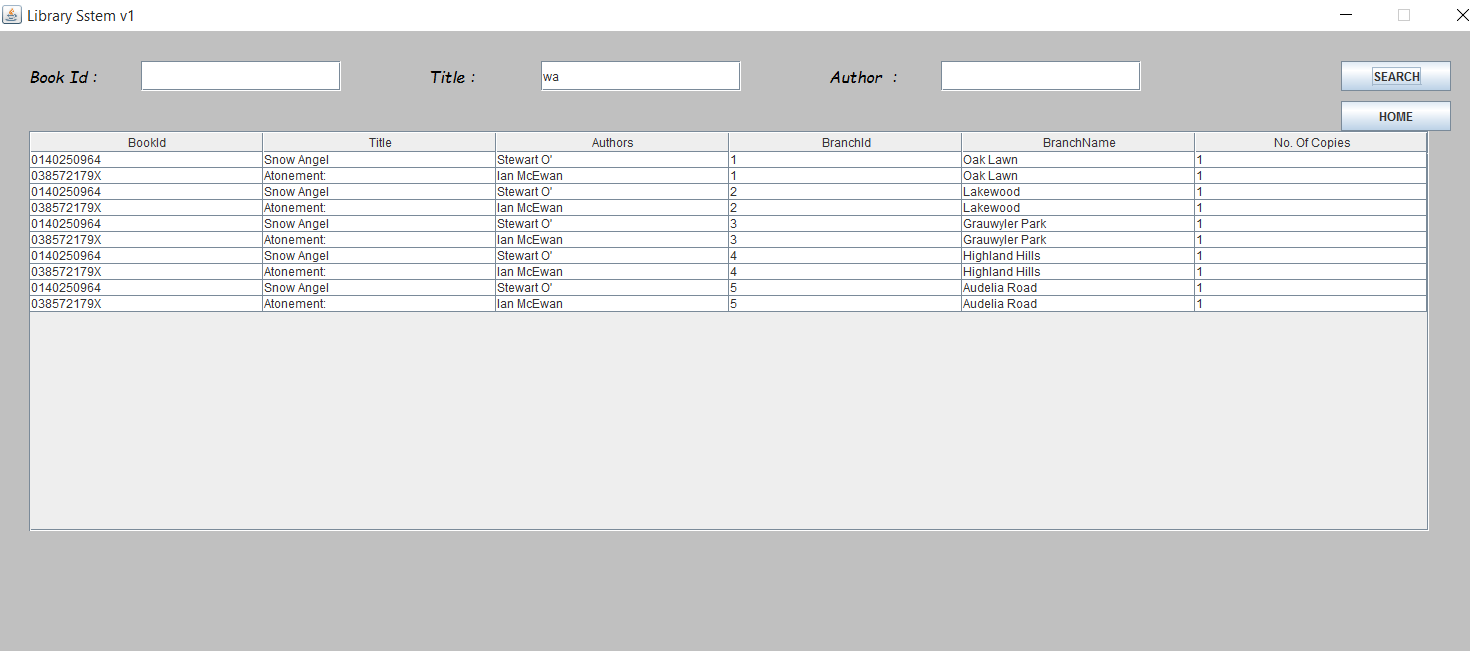


This makes it easier for the user to navigate through all the features easily.

* 1. **Book search :**

On clicking on *Book Search* button the user is redirected to a page where there is a table with fields like book\_id, the title of the book, the authors, the branch-by-branch availability of those books.

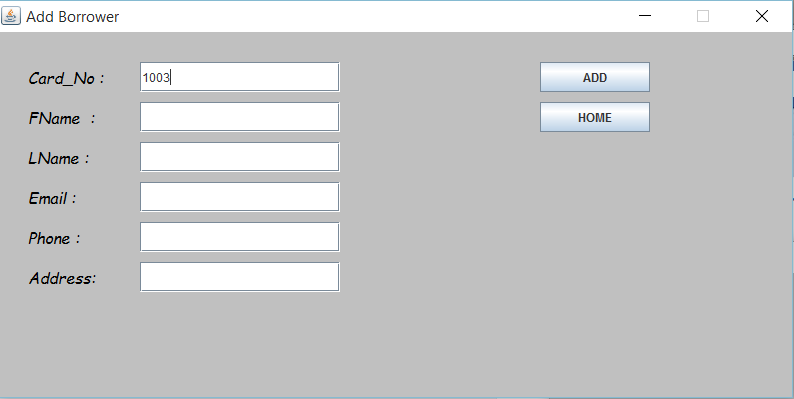
The user can now specify his/her search filters in the text fields above and get the information in a clear, tabular manner.



Note: The home button is present on every page for providing ease in navigation.

* 1. **Add Borrower :**

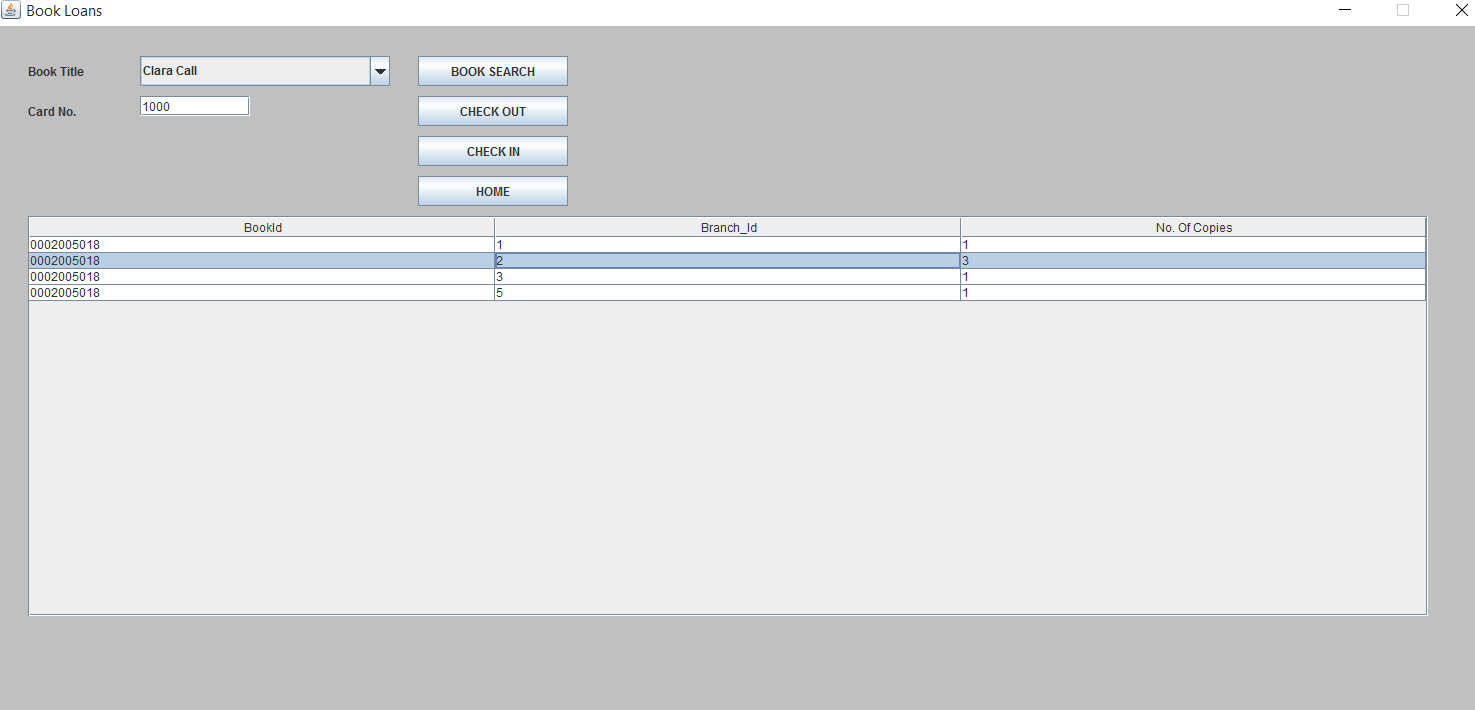
This page is responsible for addition of a borrower into the system. The card\_no is automatically generated as well as the application logic checks for already existing card holder. This guarantees that no person holds more than one card. The screen looks like the following :



* 1. **Book Loans Subsystem:**

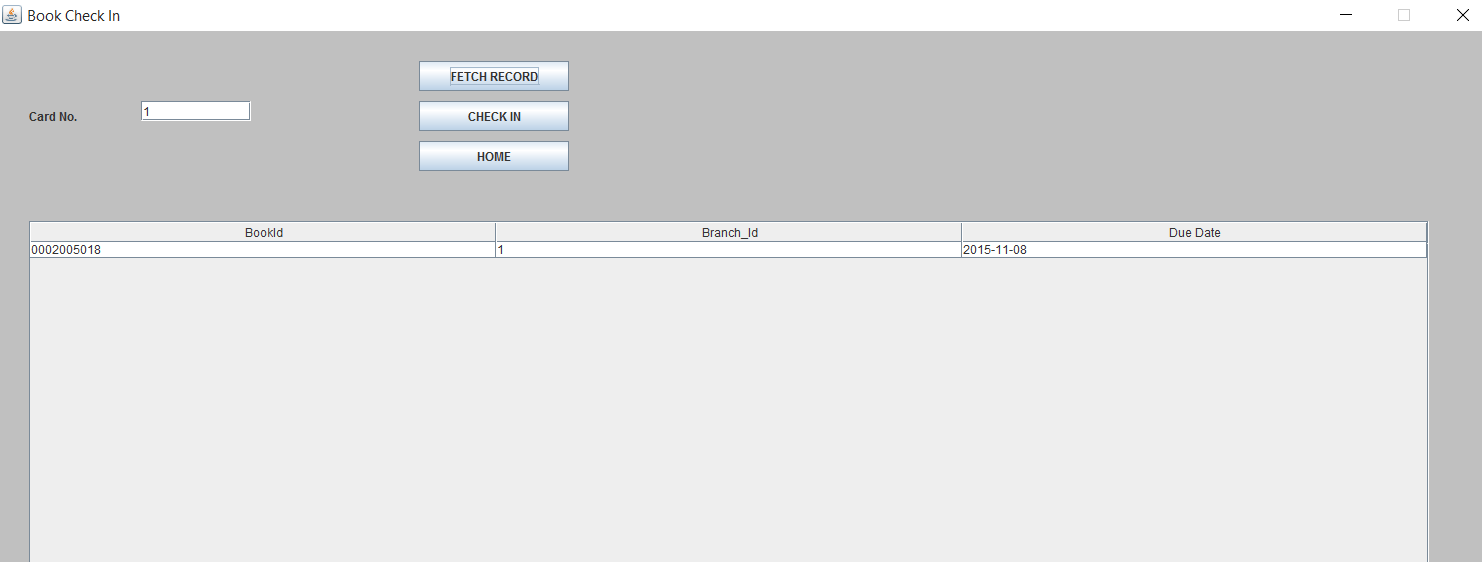
This screen is responsible for managing book loans ie. Chek in and check out of books. The Check out subsystem as shown below is a pretty straight forward feature. We just have to check if the loan limit for the member has exceeded its maximum allowance. The screen has a tabular format. The title of the desired book can be selected from a drop down menu and accordingly its availability in terms of branch\_id, branch\_name and book\_id is listed in the table.

The table is selectable and a book can be selected for check out.



The check\_in system is implemented as a part of the book\_loans system because independent of the type of operations, same tables in the back ground data base have to be updated.

The check in though, works in a similar fashion as the checkout. The user enters the card number and all loan records of the holder are presented in a tabular fashion. The user can now check in a book by selecting a table entry, hence, providing for a single screen check in function.



* 1. **Fine Payment :**

Now, this is a scheduled operation and is also dependent on whether a user checks in or checks out a book. By clicking on the fine payment button on the home screen, the application calculates the fines for all users.

Plus, the user with a particular card number can fetch his/her due amount which again is shown in a tabular fashion.

Now, the user can either pay the amount or can go back to the home screen.

