|  |
| --- |
| University OF Texas at dallas |
| Library Management System |
|  |
|  |
| **Preeti Duraipandian(pxd171230)** |
| **10/23/2017** |

|  |
| --- |
|  |

Contents

[Objective 2](#_Toc496547325)

[SCHEMA DIAGRAM 3](#_Toc496547326)

[Population Of Data 4](#_Toc496547327)

[Java Code and JavaFX 4](#_Toc496547328)

[UI Design 5](#_Toc496547329)

[Overview of queries/Updates for each functionalities 5](#_Toc496547330)

[Book Search 5](#_Toc496547331)

[Checkout 5](#_Toc496547332)

[Return a book 6](#_Toc496547333)

[Pay fine for one book at a time 6](#_Toc496547334)

[Pay Total Fine for a borrower 6](#_Toc496547335)

[Refresh 6](#_Toc496547336)

[Add a new borrower 7](#_Toc496547337)

## Objective

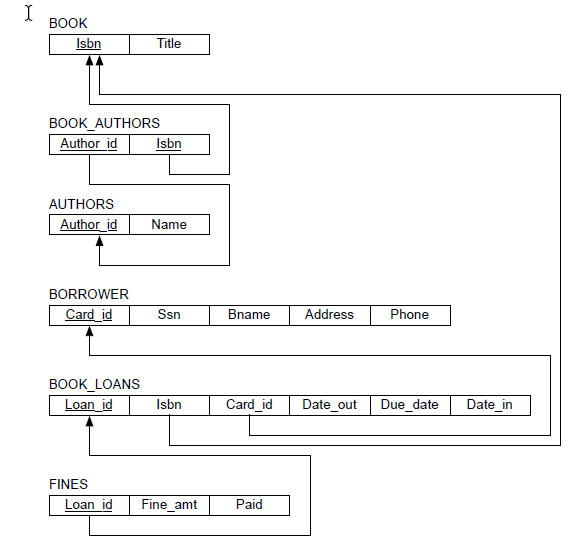
To design a Library Management System that can perform the following functions

* Search for a book
* Checkout a selected book
* Return the book
* Add new borrowers
* Calculate and update fines for overdue books
* Collect fines for overdue books from borrowers

Ensure smooth functioning of all these functions by updating the database correctly and retrieving t he correct information from the database for each of the above actions.

## SCHEMA DIAGRAM

The following schema was used for the design of the Library Management System. As is obvious the underlined attributes in each table are the primary keys and the attributes that are pointing to attributes in other tables are foreign keys. By those definitions following are the primary and foreign keys of each of the tables.



|  |  |  |
| --- | --- | --- |
| **Table Name** | **Primary Key** | **Foreign Keys** |
| Book | ISBN | None |
| Book\_Author | (Author\_ID,ISBN) | ISBN to Book.ISBN  Author\_Id to Author.Author\_Id |
| Author | Author\_id(Auto Increment) | None |
| Borrower | Card\_ID(Auto Increment) |  |
| Book\_loans | Loan ID(Auto\_Increment) | ISBN to book.ISBN  Card\_id to borrower.card\_id |
| Fines | Loan\_Id | Loan\_Id to book\_loans.loan\_id |

Schema is created using the following command,

CREATE SCHEMA LIBRARY;

All above tables were created using CREATE TABLE commands. For eg. Books table was created by running the following command.

CREATE TABLE book(ISBN CHAR(10) NOT NULL ,Title VARCHAR(170),PRIMARY KEY(ISBN));

## Population Of Data

The initial data was available as .csv files for borrower details and book details. These files were imported into the database using the workbench GUI. However alternately this could have been done by running the following command.

LOAD DATA LOCAL INFILE ‘books.csv’ INTO TABLE master\_table LINES TERMINATED BY ‘\n’;

Here, the master\_table was created in the library database specifically for the purpose of importing the data from .csv file into database and then later using that table to import data into other tables described in the schema above. Master table had the following schema.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ISBN10 | ISBN13 | Title | Author | Cover | Publisher | Pages |

No constraints were defined for this table as this was only a temporary table for importing data into other tables.

Similar to the master\_table for books, a new master table for borrower was also created and it was called master\_borrower. Data from borrower.csv was imported into this table. Following is the schema for master\_borrower table.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ID | First\_name | Last\_Name | EMAIL | Address | City | State | Phone |

The appropriate tables were populated by running the appropriate Insert into commands. For example the book table was populated by running the following command.

INSERT INTO BOOK(ISBN,TITLE) SELECT ISBN10,Title FROM MASTER\_TABLE;

Similarly the author table was populated by running the following command

INSERT INTO AUTHOR(Author\_Id,Name) SELECT UNIQUE 0,Author FROM master\_table;

There is a 0 in the second query because Author\_Id is an auto-incremented field. Hence it really doesn’t matter what I am inserting, MySQL will correct it.

There is a tricky part in the second table that is borrower table where I concatenated the fields into a single field. Hence that insert command was something like this.

INSERT INTO borrower(Card\_Id,BName,Address,Phone) SELECT 0,CONCAT(First\_Name,’ ‘,Last\_Name),CONCAT(Address,’ ’,City,’ ‘,State),Phone FROM Master\_Borrower;

Once all tables are populated, I proceeded to the development of the UI and Java-MySQL Connectivity.

## Java Code and JavaFX

I chose JavaFX for the design of UI and used **scenebuilder** to design the layouts for each page.

The Java Project is divided into following sections.

1. Java Code(ch.makery.library.code)- This contains the main Class of the JAVAFX project. The application gets launched from here and the different fxml files that are required are loaded from here.
2. Java Model(ch.makery.library.model)- This contains all the class structures for all the tables that have been used in the UI.
3. Java Util(ch.mkaery.library.util) – This contains all the DB utility functions that almost all interactions with mySQL would require.
4. Java View(ch.makery.library.View)- This contains the FXML elements on which the actual UI components lie and the Java Controller classes that initialise these components, handle these components(buttons) and get or set values to these components(TextFields,TableViews). Almost all of the queries that need to be run for each action have been specified and run from the controller class functions.

### UI Design

To sum up the UI design very briefly- there is a single Tab Pane consisting of 5 tabs. One each for Book Search and Checkout, Add new Borrower, Return Book, Pay fine for one book at a time, Pay total fine per borrower. Identical refresh functionalities have been provided on both Pay Fine pages. Each tab has it’s own separate fxml file and hence it’s own controller. These fxml files have been directly included in the main Tab Pane fxml file to keep the Tab Pane Controller file less cluttered and to make the design more modular.

## Overview of queries/Updates for each functionalities

### Book Search

For this purpose, every time the Search Button is clicked, I create a new View called **Matched\_Books,** that would contain the following schema.

|  |  |  |
| --- | --- | --- |
| Book.ISBN | Book.Title | Author.Name |

This view gets populated with the UNION of the following queries.

1. SELECT B.isbn,B.Title,A.Name from book as B,book\_author as BA,author AS A where B.ISBN=BA.ISBN AND BA.author\_id=A.author\_id AND B.Title LIKE ‘SearchString’
2. SELECT B.isbn,B.Title,A.Name from book as B,book\_author as BA,author AS A where B.ISBN=BA.ISBN AND BA.author\_id=A.author\_id AND A.Name LIKE ‘SearchString’
3. SELECT B.isbn,B.Title,A.Name from book as B,book\_author as BA,author AS A where B.ISBN=BA.ISBN AND BA.author\_id=A.author\_id AND B.ISBN LIKE ‘SearchString’

Once this view is created and a result is obtained after running a query to display all attributes of the view, I check if each of this entry is present in the book\_loans table. If it is then check if it has it’s DATE\_Out value as NOT NULL as well as DATE\_IN value also as NOT NULL, implying the book has been checkout and has not been returned yet.

Based on that it is decided if the book is currently available or not.

### Checkout

Before checking out a selected ISBN (a book), following checks are made.

1. Selected ISBN is not allowed to be checked out if it is present in the book\_loans table and has it’s DATE\_OUT and DATE\_IN value as NOT NULL.
2. It is checked if the card\_id against which the checkout is being made doesn’t have 3 or more books in his name that have not been returned yet.

SELECT card\_id,COUNT(\*) FROM book\_Loans where Date\_Out IS NOT NULL AND DATE\_IN IS NULL AND Card\_ID=cardId+ GROUP BY card\_Id HAVING COUNT(\*)>2

Once these checks are satisfied, it is allowed to checkout a book. To mark that a book is checked out, that book makes an entry into the book\_loans table, the date out is set to the current date and due\_date is set to a date 14 days later,date\_in is set to NULL.

### Return a book

For a person to return the book, it is necessary to find out what books that person has borrowed. For this, a substring matching search box is provided, where one can type the borrower Name or the ISBN or the Card Id of the borrower and anything matching with that substring will be returned. Again this search is similar to what was done for book search, only the attributes on which it is applied differs.

When the person clicks the return button, The date\_in value for that specific ISBN in the book\_loans table is set to Current Date. Following is the update that happens.

Update book\_loans set Date\_in=curr\_date where ISBN=selISBN;

In the above query, cur\_date is obtained from Java functions and selISBN is received from the UI.

### Pay fine for one book at a time

Once a person returns a book, the librarian can come to this tab to check what fine amount he owes for the book he just returned and fine amounts for the books that he has returned in the past but has not paid the amount till this date. This needs to be done only after clicking the refresh button whose functionality will be explained in some time.

There is a filter to filter out paid or unpaid fine amounts for that borrower.

For the sake of example, if we are filtering out unpaid entries, we run the following

SELECT BL.Card\_Id,BO.BName,B.ISBN,B.Title,BL.Date\_Out,BL.Due\_Date,F.Fine\_Amt FROM book as B,borrower as BO,fines as F,book\_loans as BL where Bo.card\_Id=BL.Card\_Id AND DATE\_IN IS NOT NULL AND F.Loan\_Id=BL.LOan\_Id AND BL.ISBN=B.ISBN AND BL.Card\_Id=searchText AND Paid=false

This is just one of the select statements that the actual query executes. Other two would match based on the Borrower Name and ISBN.

In case someone wants to pay fine for already paid book, an error message is thrown.

Once the fine is paid, the fines table for that loan\_id is updated to Paid.

### Pay Total Fine for a borrower

When the librarian enters the borrower name or his ID, his total fine amount (sum of the fine amounts for all the books that were overdue in his name) would appear. This way he can pay the total fine that he owes.

As with the Pay one Book at a time, for correct fine amount that he needs to pay, the librarian must press the Refresh Button before searching the borrower’s fine amount.

When Pay Fine button is pressed, if the correct amount is entered as Paid Amount then all the unpaid entries in the fine table are updated as Paid.

### Refresh

The following three functions are done in the refresh button handler.

1. Update fines for books that have not been returned yet. Fine is updated to difference of current date and due date multiplied by $0.25.

update fines,book\_loans set fine\_amt=0.25\*DateDiff(CURDATE(),Due\_Date) where book\_loans.loan\_id=fines.loan\_id and Date\_In IS NULL

1. Update fines for books that have been returned but since then the fine amount is not paid. Fine amount will be difference between date IN and due date multiplied by $0.25.

update fines,book\_loans set fine\_amt=0.25\*DateDiff(Date\_IN,Due\_Date) where book\_loans.loan\_id=fines.loan\_id and Date\_In IS NOT NULL AND Paid=false

1. If some book became overdue since the last time the fines table was refreshed, it will not be present in the fines table. These books need to be inserted into the fines table with correct fine amount.

insert into fines SELECT book\_loans.loan\_id,0.25\*DATEDIFF(CURDATE(),Due\_Date),0 from book\_Loans where CURDATE()>Due\_Date AND Date\_IN IS NULL AND Loan\_id NOT IN(Select Loan\_Id from fines)

### Add a new borrower

Not much here. When all details for a borrower is entered, with those details a new entry is created in the borrower table.