Library System Programming Project

CS-4347 Database Systems (Milestone 2)

Description

This SQL programming project involves the creation of a database host application that interfaces with a backend SQL database implementing a Library Management System. Users of the system are understood to be librarians (not book borrowers).

This is a team project.

Basic Functionality

This **Milestone 2** focuses on basic functionality and primarily involves implementing the programming logic. The final milestone, Milestone 3, will involve interacting with the your Milestone 2 features via a GUI interface. Milestone 3 will be evaluated with a schedule live demonstration via Teams screen share.

Programming Language(s) and Frameworks

Your host application should interact with a SQL database. Milestone 2 requires implementing only the logic that manipulates your SQLj database.

Your application GUI may be programmed with Java, Python, C#, C++, Ruby, Python, Javascript, or PHP.

Approved SQL databases are SQLite, MySQL, PostgreSQL, and Microsoft SQL Server.

If you would like to use any other language or SQL database not specifically listed above, you ***must*** obtain prior approval from the TA that they are able to effectively evaluate the language of your submission.

Submission

Your submission should be a single archive file (zip, gzip, or tar) that contains all source code and a `readme.pdf` file that provides instruction on how to build and run (language and version, compiler version, third party module includes/imports.

Functional Requirements

1) GUI (20 points)

All GUI requirements are Milestone 3, not required for Milestone 2.

2) Book Search and Availability

Be able to search for a book, given any combination of ISBN, title, and/or Author(s). Your search interface should provide a single text search field (like Google) and be case insensitive. Your query should support substring matching (e.g. search for “william” should return author “William Jones”, author “Sam Williamson”, and book title “Houses of Williamsburg, Virginia”). For each postitive hit, you should then display the following four columns in your search results:

- ISBN
- Book title
- Book author(s) (displayed as a comma separated list)
- Book availability (is the book currently checked out?)

For example, your program may implement a function or method named `search()` that takes a single string as an argument. The string should be interpreted as case-insensitive.

```
search("will");

# May find positive 'hits' such as...
# Red color for explanitory purposes only
NO ISBN        TITLE                              AUTHORS                      STATUS
01 0923398364  Houses of Williamsburg, Virginia   John Smith                   IN
02 0891787631  College Algebra                    Will McDaniels, Mary Jones   IN
03 T619875682  Wills and Trusts for Retirement    Barbara Brown, ESQ           OUT
04 X2723786q6  The History of the Han Dynasty     Dawn Williamson              IN
```

Note that a search for a full ISBN should return no more than one result.

3) Book Loans [20 points]

Checking Out Books

- Be able to select a book for checkout either by:
  - Providing both an ISBN number and Borrower ID directly to a `checkout()` function

- Each BORROWER is permitted a maximum of 3 active BOOK_LOANS. If a BORROWER already has 3 BOOK_LOANS that are still out, then the checkout (i.e. create new BOOK_LOANS tuple) should fail and return a useful error message. Due date should automatically populate as exactly 14 calendar days from day of checkout.

- If a book is already checked out by any BORROWER (including the BORROWER currently attempting to checkout), then the checkout should fail and return a useful error message that the book is not available.

- If a borrower has an unpaid fine, do not permit them to checkout a book.

Checking In Books

- Be able to locate BOOK_LOANS tuples by searching on any of BOOKS.isbn, BORROWER.card_no, and/or any substring of BORROWER name. Once checked out books are located, provide a way of selecting potentially 1-3 potentially checkouts to check in.

4) Borrower Management [20 points]

- Be able to create new BORROWERs in the system.
- All name, SSN, 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 SSN, then your system should reject and return a useful error message.

5) Fines [20 points]

- fine_amt attribute is a dollar amount that should have two decimal places. The data type should be fixed-decimal, not `float` or `real`.
- `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 mechanism (i.e. method/function) that updates/refreshes entries in the FINES table. In reality, this would occur as a cron/batch script that automatically executed daily during library closed hours.
- 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].
- When updating fines, 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.
  - Do not allow paying partial FINES. Successful payment should clear all fines.
  - Display of Fines should provide a mechanism to filter out previously paid fines (either by default or choice).

Schema

The schema for the library database is derived from (but NOT the same as) the library schema in the textbook (Figure 6.6). The *actual* schema used for this project is provided below. You are permitted to modify or augment this schema provided that your system (a) is backwards compatible with the given schema, and (b) adheres to the written requirements. As long as your system supports the documented functionality, you may add any features you deem useful.