Programming Project #1 (SQL Application)

Programming Project #1 (SQL Application)

This is the link to submit your SQL Library Project.

This SQL programming project involves the creation of a database host application that interfaces with a SQL database that implements a Library Management System. Assume that the users of the system are librarians (not book borrowers).

This is an individual (not group) project. All work, design, and coding must be done individually by you.

~~**Due Date: Monday, October 19 11:59pm**~~

Due Date: Sunday, October 25 11:59pm

Programming Language(s) and Frameworks

To build your host application you may implement either a native GUI application or a web interface. Your applications may be programmed with Java, Ruby, Python, or PHP. You may use a web programming framework like Django (Python) or Node.js (Javascript). You may use either MySQL, PostgreSQL, or MS SQL Server as your database. You may NOT use a platform-specific language, e.g. Objective-C, Swift (Apple) or C#, F# (Microsoft). If you would like to use any other language not specifically listed above, you must obtain approval from the TA that she is able to evaluate the language of your submission.

Schema

The schema for the library database is derived from (but NOT the same as) the library schema in the textbook (Chapter 6 Exercises). The actual schema used for this project is provided in this folder. You are permitted to modify or augment this schema provided that your system adheres to the written requirements. As long as it supports the documented functionality, you may add any features you deem useful.

Data

- Baseline data to initialize your database is provided in this folder.
  - There is no initial data in the BOOK_LOANS table.
- The use cases executed for grading will be based upon the provided data.
- Data is provided in plain text CSV files. It is your task to map these onto the schema and tables, i.e. there is not a one-to-one, file-to-table correspondence.
- All book id's are 10-character ISBN numbers (i.e. some contain alpha characters).
- Replace primary key of BOOK_LOANS with a new synthetic key that is INTEGER type named "loan_id". This will allow a BORROWER to check out a book more than once from the same branch.

Submission

You will be required to submit the following files:

- All application source code, including any build files (e.g. make, ant, maven, etc.)
- An SQL schema creation file that includes appropriate all appropriate CREATE TABLE commands and (potentially) ALTER TABLE commands and/or CREATE TRIGGER commands to implement any integrity constraints.

- Table data import file(s) that include includes either (a) a sequence of INSERT INTO commands, or (b) A file will load directives and the .dat on which it operates
- A 2+ page written description of your system that includes (a) Quick Start user guide for librarian system users, (b) high-level architecture, design decisions, and justifications, (c) technical dependencies (software libraries, software versions, etc.) and build instructions.
- All files must be zipped together into a single file. This file should be named ***<net-id>*_cs6360.zip**, where ***<net-id>*** is your Net ID. Example: **cid021000_cs6360.zip**

Grading

- You will be required to demonstrate your application for the TA. If you are unable to bring a laptop computer to demonstrate your application, please let me know as soon as possible so I can make alternate arrangements.
- Each student will have 10-15 minutes to demonstrate their system and execute the test cases provided at grading time.
- A sign-up mechanism will be made available to reserve a specific time for evaluation. As a courtesy to the TAs and those waiting behind you, **please be on time for your scheduled slot** and have your application already launched and ready to go.

Notice: Requirements *will* change before the due date. One objective of this project is the ability to accomodate requirement changes after beginning a database project. There will be at least one schema change, one functional requirement change, and one data change.

The follow are a *preliminary* summary weight of your functional requirements:

1) GUI [25 points]

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.

SQL Library Project New Requirements

The following new requirements are now added to the SQL Library Project.

1. enable the tracking and payment of **library fines**.

This requirement will be worth 15 points, which will be taken from the 25 points GUI requirements.

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).

Java SQL stub code

Attached Files: DBTest.java (2.495 KB)

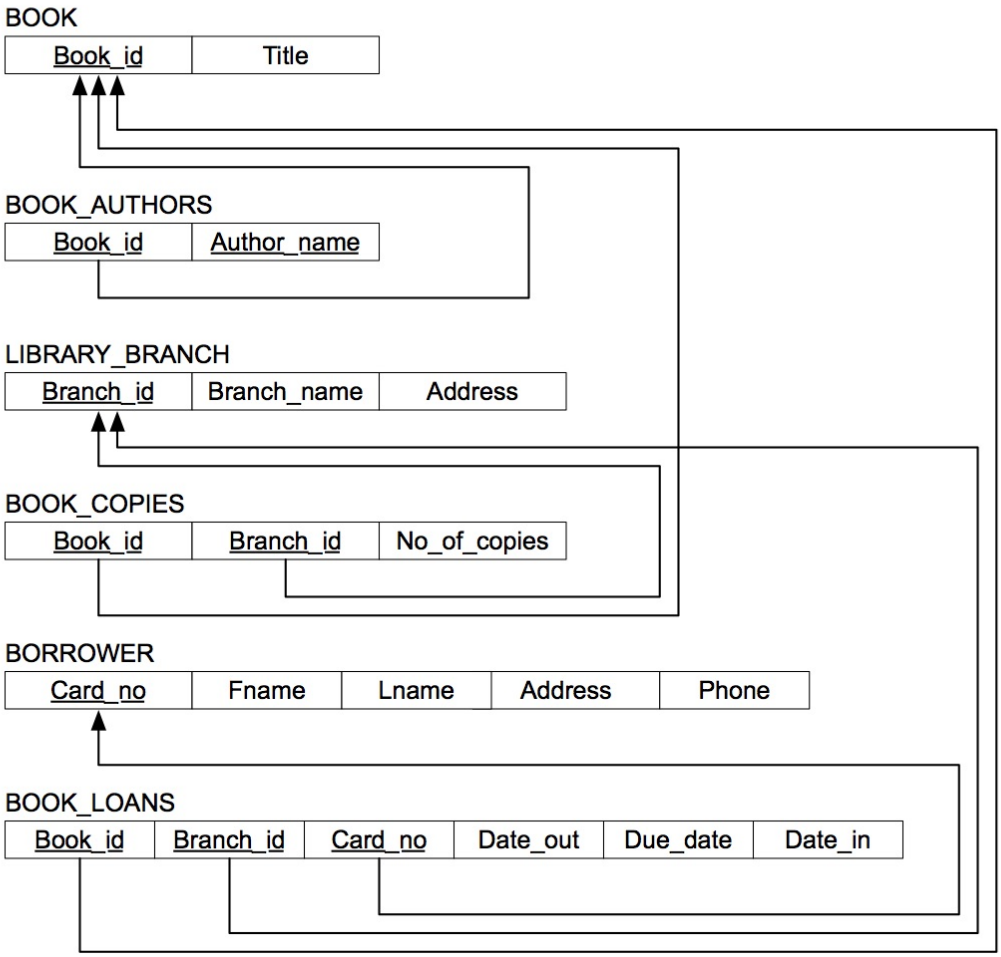
Attached is an example Java code which connects to a MySQL database. This examples assumes that the database you connect to has the Company schema created from the textbook. You must also have the Java MySQL Connector installed to be able to

```
import java.sql.*
```

http://dev.mysql.com/downloads/connector/

This requires that the connector `.jar` file be placed in your Java ClassPath (or alternatively use `-cp` option when you launch Java).

Library Project Schema v1.0



Library Data Set

Attached Files: book_copies.csv (20.705 MB)
borrowers.csv (82.576 KB)
library_branch.csv (241 B)

books.csv (37.917 MB)

Here are the raw CSV files which can be normalized and/or reformatted for import into SQL.