SQL Programming Project

CS-6360 Database Design

Due: October 11, 2016 11:59pm

Description

This SQL programming project involves the creation of a database host application that interfaces with a backend SQL database to implement a DBMS for a library circulation desk. Users of the system are understood to be librarians (not book borrowers).

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

Programming Language(s) and Frameworks

Your application must utilize a SQL database as its back end data store. Approved databases for use are MySQL, PostgreSQL, Oracle Express, MS SQL Server, and SQLite.

Your host application must use a GUI interface. You may choose to implement either a native GUI application or a web application interface. Approved application languages are Java, C#, PHP, Python, Ruby, and Javascript. You are free to use any third party libraries, frameworks, or templates to implement your GUI (e.g. Swing, JSP, ASP, Rails, Node.js, Django, etc.).

Grading

You will be required to demonstrate your application for the TA for grading. 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.

An online sign-up mechanism will be made available to reserve a specific time with the TA for your demo. 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.

Each student will have approximately 10 minutes to demonstrate their system and execute the test cases provided at the demo. Test cases will be designed to validate use cases created from the stated requirements.

Functional Requirements

1) Graphical User Interface (GUI), Overall Design, and Architecture [20 points]

All librarian user interaction with the Library Circulation Desk DBMS (queries, updates, deletes, etc.) must be done from a graphical user interface of your original design. The initial import of the dataset is considered to be an administor task and my be performed via any method (Workbench, import script, command line, etc.).

During your presentation you will be expected to execute test cases through your GUI as well as perform raw SQL queries to validate your schema. GUI design will be judged primarily on function and usability, not look-and-feel.

2) Book Search and Availability [25 points]

Using your GUI, provide a single search field to locate a book given any combination of ISBN, title, and/or Author(s). This is similar to the single search field on the UTD library website: http://www.utdallas.edu/library/.

Your query should support substring matching. For example, search for "will" should return results whose title include "Will", "Willing", or "Swill", or whose author name contains "Will", "Willy", "William", etc.

Search output should display the following information for each book in the result set:

- ISBN
- Book title
- All book author(s)
- Availability status

3) Book Loans [25 points]

Checking Out Books

- Using your GUI, be able to check out a book from the library, which will create a new record in the BOOK_LOANS table. The **Date_out** should be today's date. The **Due_date** should be 14 days after the **Date_out**. **Date_in** should be NULL at checkout time.

- A checkout requires selecting a combination of BOOK and a BORROWER. The BOOK to be checked out should be selected by direct input of the book ISBN. The borrower's library card number should be selected by direct input of the borrower's Card_id.

- Attempts to checkout a book copy that is not available should be rejected and display an appropriate error message.

- Each BORROWER is permitted a maximum of 3 BOOK_LOANS. If a BORROWER already has 3 active BOOK_LOANS, then the checkout (i.e. attempt to create a new BOOK_LOANS tuple) should fail and display an appropriate error message.

- BORROWERS are not permitted to check out a book if they have either an unpaid fine or a currently overdue book. The system should reject such attempts and display an appropriate 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 book information or BORROWER information. Once an active BOOK_LOANS record is located, provide a way of checking in the book, i.e. updating the **Date_in** value from NULL to the current date.
- If an overdue book is checked in, a FINES record should be created (see Requirement #5)

4) Borrower Management [15 points]

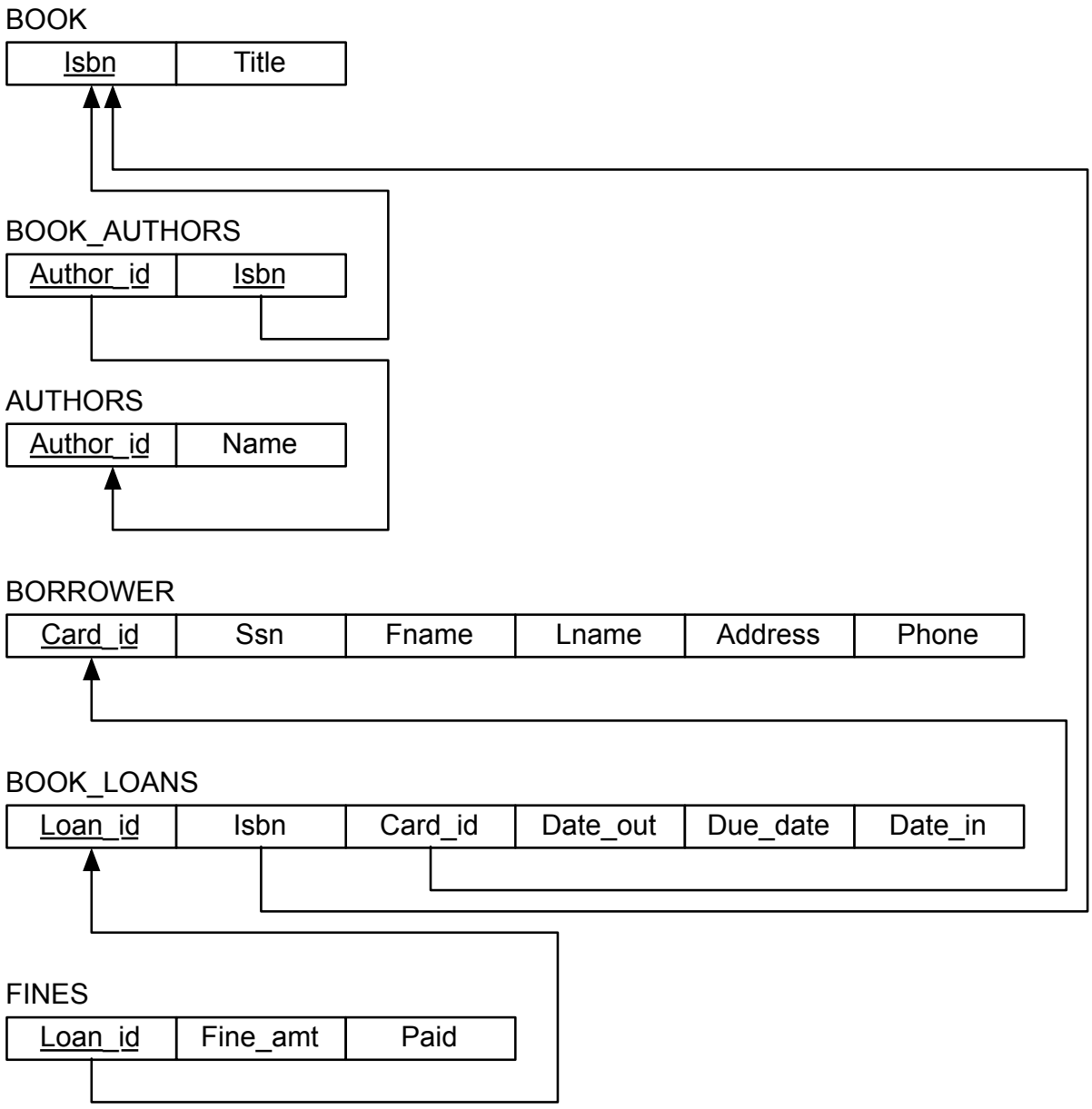
- Using your GUI, be able to create new a BORROWER record in the system.
- BORROWER **Name**, **Ssn**, and **Address** attribute values are required to create a new account (i.e. values must be NOT NULL).
- Borrowers are allowed to possess exactly one library card. If a new borrower is attempted with a duplicate **Ssn**, then your system should reject the new BORROWER and return an appropriate error message.

5) Fines [15 points]

- Using your GUI, be able display FINES information for a specified BORROWER. You should provide a mechanism to display paid, unpaid, or both.
- Using your GUI, be able display all *overdue* books for a specified BORROWER, i.e. BOOK_LOANS whose **Due_date** has passed, but whose **Date_in** is NULL
- Fines may not be assessed (i.e. create a record in the FINES table) until a book is returned.
- Fines shall be assessed at a rate of $0.25/day (twenty-five cents per day).
- Using your GUI, provide a mechanism for librarians to enter payment of fines (i.e. update a FINES record to set **Paid** attribute to be true.

Schema

The schema for the library database is derived from (but NOT the same as) the library schema in the textbook (Figure 6.6, page 204). 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.



Data

- Baseline data to initialize your database is provided in the eLearning programming assignment folder. All data is provided in plain text CSV files.
- Not all files may use the same CSV format. For example, some files may use comma delineation while others use tab delimiters. Some files may use quotes around strings, some may not.
- Note that there is *not* a one-to-one, file-to-table correspondence. For example, data to popultate the BOOK, AUTHORS, and BOOK_AUTHORS tables is found in the single file `books.csv`.
- Some of the data contains "noise". Part of your system design task to clean and normalize these data, i.e. map it onto your schema and tables.
- You may use either ISBN-10 or ISBN-13 as the BOOK.Isbn primary key. Note that some ISBN-10 values may contain leading zeroes. These are part of the attribute value and should *not* be truncated!

Submission

You will be required to submit the following files for grading:

- A **Design Document** that contains a high-level description of your system architecture including design decisions and assumptions. It should be 1.5-3 pages 12 point font text, *not* including any schema diagrams or system architecture figures. Clearly indicate any and all third party libraries and/or software used. Document format should be PDF.

- A **Quick Start Guide** (a brief User Manual) for librarian system users (1-2 pages). Document format should be PDF.

- A **`readme.txt`** file that describes how to compile, build, and install your application. It should include any technical dependencies (language and version, frameworks, platform, OS, software libraries, software versions, etc.). Build scripts (Ant, Maven, Makefile, etc.) are encouraged, but not required. Document format should be plain ASCII text.

- All **application source code**, including any build files.

- Any **scripts** created to clean or normalize raw csv data.

- You do not need to submit data files.

- All submission files must be zipped together into a single file. This file should be named `<netid>_library.zip`, where `<netid>` is your Net ID.
  Example: `cid021000_library.zip`