SQL Programming Project

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 group project.

Programming Language(s) and Frameworks

Your host application should have a GUI interface. You may implement either a native GUI application or a web GUI interface.

Your application GUI may be programmed with Java, Ruby, Python, Javascript, or PHP. You *may* use a web programming framework like Django (Python) or Node.js (Javascript). You *may* use an ORM (object-relational mapping) framework, like Hibernate, provided that your SQL schema is compatible with the requirements. Approved SQL databases are MySQL, PostgreSQL, MS SQL Server, and SQLite.

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

Functional Requirements

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

All interface with the Library database (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 SQL connector. Initial database creation and population may be done from command line or other admin tool.

Overall design will be judged on intuitive usability, ***not*** aesthetic design.

2) Book Search and Availability [20 points]

Using your GUI, 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 title "Houses of Williamsburg, Virginia"). You should then display the following in your search results:

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

3) Book Loans [20 points]

<u>Checking Out Books</u>

- Once found in a GUI search, be able to check out a book after being prompted for a 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 default to 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 a book is already checked out, then the checkout should fail and return a useful error message.

So any time we search any one of the variables given, bring all the books and click on them to check in out and out by who ?

Checking In Books

- Using your GUI, be able to check in a book. For examle, be able to locate BOOK_LOANS tuples by searching on any of BOOKS.book_id, BORROWER.card_no, and/or any substring 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. today as the date_in in corresponding BOOK_LOANS tuple).

4) Borrower Management [20 points]

- Using your GUI, 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.

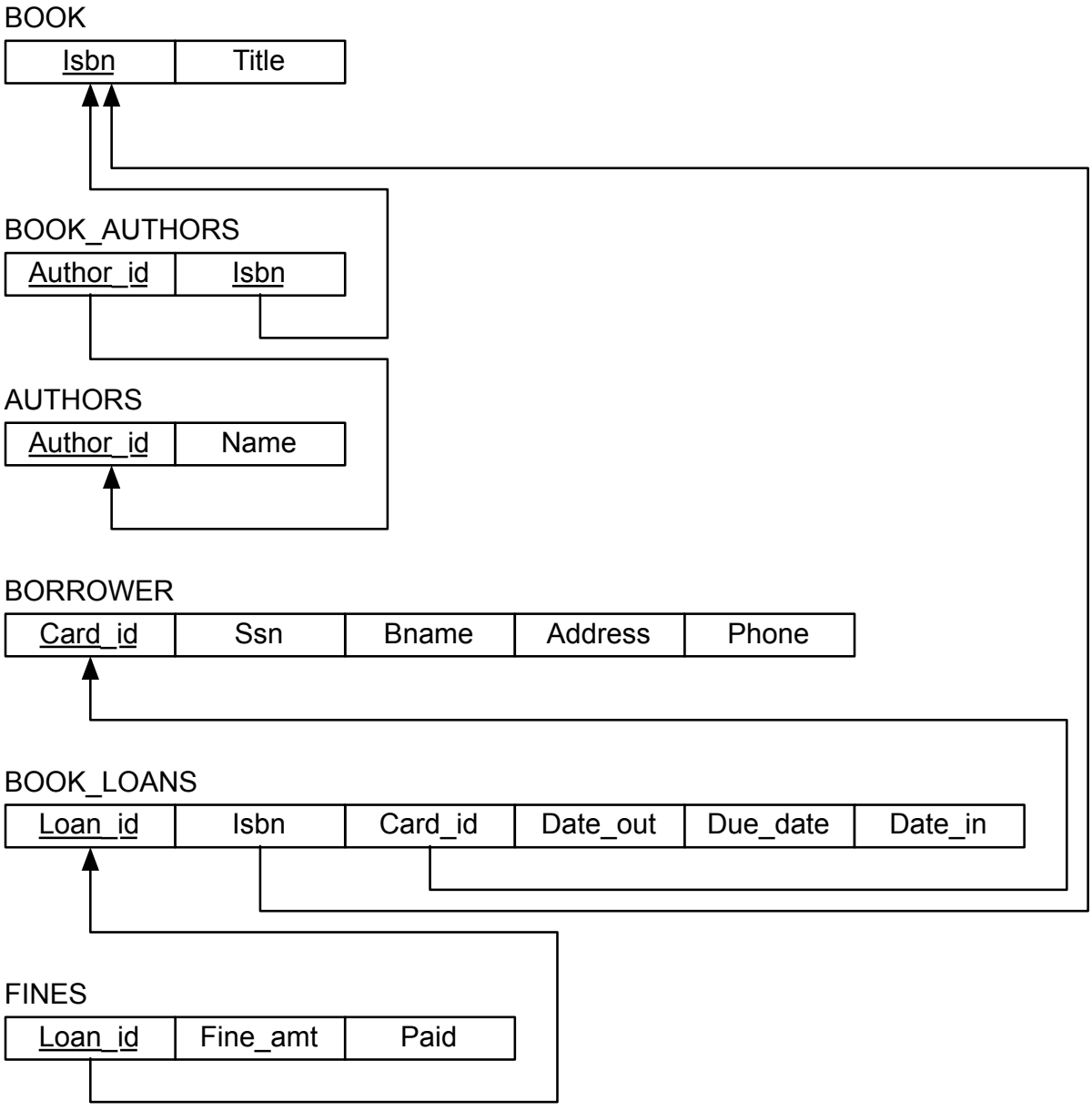
Multiple primary keys essenitally

5) Fines [20 points]

- 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).

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.



Data

- Baseline data to initialize your database is provided in the eLearning programming assignment folder.
- All data is provided in plain text CSV files. Note that there is *not* a one-to-one, file-to-table correspondence. Part of your system design task to map (i.e. normalize) these data onto your schema and tables.
- All book id's are 10-character ISBN numbers (i.e. some contain alpha characters). Note that some may contain leading zeroes. These are part of ISBN and should *not* be truncated!

Submission

You will be required to submit the following files:

- A design document that describes your system architecture including design decisions and assumptions. (1.5-3 pages 12 point font, *not* including any schema diagrams or system architecture figures). Format should be PDF.
- A Quick Start user guide for librarian system users (1-2 pages).
- A `readme.txt` file that describes how to compile, build, and install your application. It should include any technical dependencies (language, frameworks, platform, OS, software libraries, software versions, etc.).
- All application source code, including any build files (e.g. make, ant, maven, etc.).
- All files must be zipped together into a single file. This file should be named `<netid>_cs6360.zip`, where `<net-id>` is your Net ID.
  Example: `cid021000_cs6360.zip`

Grading

Milestone 1 will be graded by the TA offline.

Milestone 2 will be graded by the TA in person with a live demonstration. If your team is unable to bring a laptop computer to demonstrate your application, please let me know as soon as possible so I can make alternate arrangements to demo your project in the computer lab.

An online sign-up mechanism will be made available to reserve a specific time with the TA for evaluation and grading. 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 team will have 20 minutes to demonstrate their system and execute the test cases provided at grading time. Test cases will be designed to validate use cases from the published requirements.