University Database Technical Report

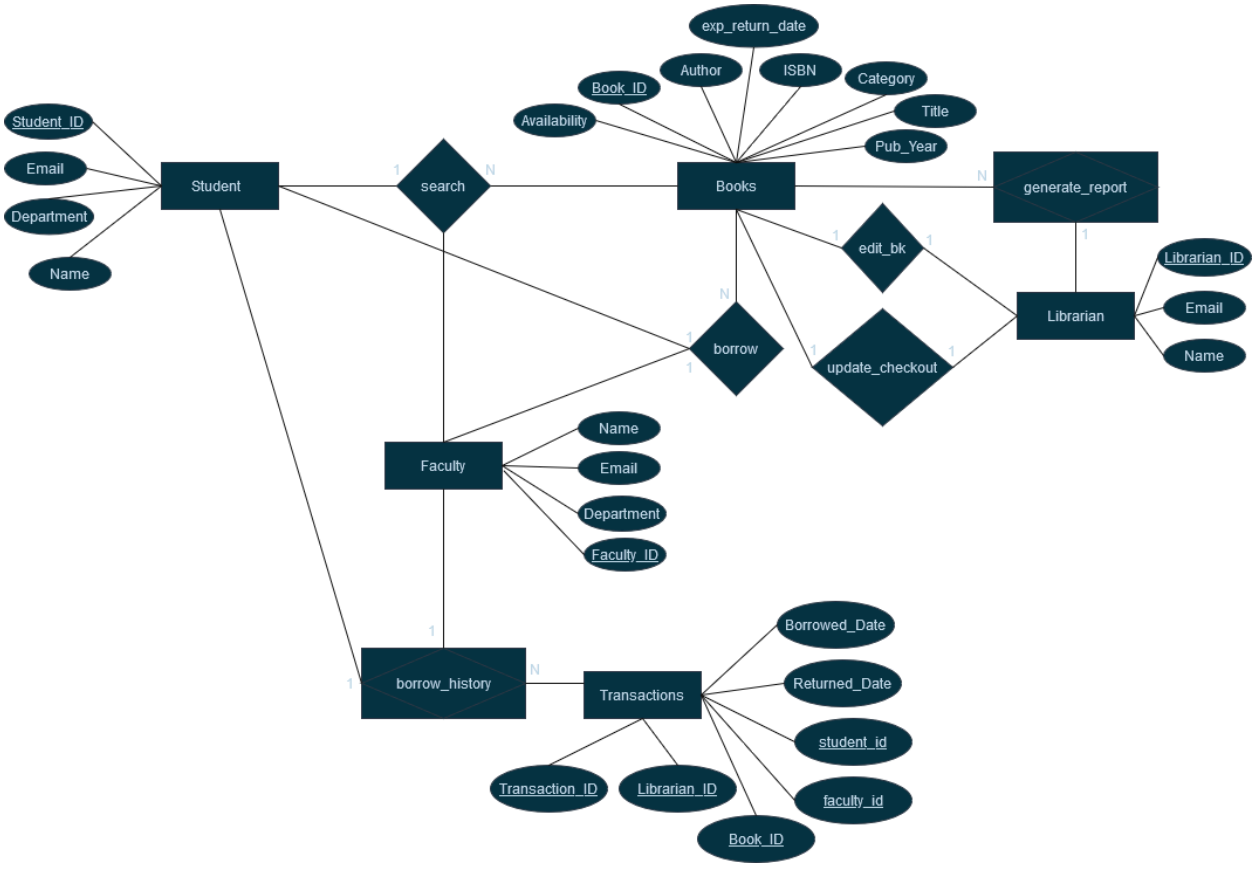
Introduction

Imagine a bustling university library filled with wonderful and interesting books and staffed with amazing librarians ready to help students and faculty with borrowing from the library's vast collection. However, the university struggles with inefficiencies in overseeing loans, tracking book information, and validating users.

Throughout the semester, we have learned about the various critical roles that database management systems (DBMS) have in the modern tech landscape. They provide a framework for data organization and storage, ensure data integrity and security, include data accessibility and retrieval, implement concurrency control and transaction management, and offer enhanced scalability and performance.

The university library database project was an opportunity for Boosung and myself to apply the various aforementioned concepts into our project. We can help rid these inefficiencies by creating an efficient database schema that interacts with user-friendly website for students, faculty, and librarians to perform their respective tasks.

Entity-Relationship Modeling (ERM)



Database Schema

Database Entities

1. Books: A table representing a book entity.
    i. Book_ID (primary key): A unique identification number that is char(9) that cannot be null.

   ii. Title: The book's title which is of type char(255).
   iii. Author(s): The book's author(s) which is of type char(255).
   iv. ISBN: The book's International Standard Book Number (ISBN) which is of type VARCHAR(13) to account for the fact that leading zeros may be truncated and hyphens can sometimes be included. It is also UNIQUE because no two books can have the same ISBN.
   v. Pub_Year: The book's year of publication which is of type INTEGER. We place no constraints on what year the book can be published from.
   vi. Category: The book's literary category (e.g. fantasy, fiction, horror, etc.) which is of type char(255).
   vii. Availability: A book's availability indicates whether or not a student or faculty member could check it out. This attribute is represented by a BOOLEAN.
   viii. Exp_Return_Date: The book's expected return date based on who checked out the book. The expected return date has a DATE data type and it has a constraint where the expected return date, when assigned, must be greater than the current date.
2. Students: A table representing a student entity.
   i. Student_ID (primary key): A unique identification number for a student that is of type char(9).
   ii. Email: The email address of the student that can be used to reach said student. The email address is varchar(255) and atop of that it must be unique. No two students or faculty members can have the same email address.
   iii. Department: A student's academic department (e.g. Computer Science, Mathematics, Physics, etc.). This is of type varchar(100).
   iv. Name: A student's full name (first name, middle name, and last name). This is of type varchar(100).
3. Faculty: A table representing faculty entity.
   i. Faculty_ID (primary key): A unique identification number for a faculty member that is of type char(9).
   ii. Email: The email address of the faculty member that can be used to reach said student. The email address is varchar(255) and atop of that it must be unique. No two students or faculty members can have the same email address.
   iii. Department:A faculty member's academic department (e.g. Computer Science, Mathematics, Physics, etc.). This is of type varchar(100).
   iv. Name: A faculty member's full name (first name, middle name, and last name). This is of type varchar(100).
4. Librarian: A table representing a librarian entity.
   i. Librarian_ID (primary key): A unique identification number for a librarian that is of type char(9).
   ii. Email: The email address of a librarian that can be used to reach said student. The email address is varchar(255) and atop of that it must be unique. No two librarians, students, or faculty members can have the same email address.

iii. Name:A librarian's full name (first name, middle name, and last name). This is of type varchar(100).

5. Transactions: A table representing a transaction entity.
   i. Transaction_ID (primary key): A unique identification number for a transaction that is of type char(9).
   ii. Librarian_ID (foreign key): Look at the librarian table for more details.
   iii. Student_ID (foreign key): Look at the student table for more details.
   iv. Faculty_ID (foreign key): Look at the faculty table for more details.
   v. Book_ID (foreign key): Look at the book table for more details.
   vi. Returned_Date: The return date of a loaned book. It has the year data type and it is constrained to have a date greater than the borrowed date.
   vii. Borrowed_Date: The borrow date of a loaned book. It has the year data type and it is constrained to have a date less than the returned date.

Relations

1. Faculty and students can search the book table by author(s), ISBN, category, title, publication year, and availability.
2. Faculty and students can borrow books from the book table. Students have a loan time of one week while faculty members have two weeks. This is reflected in the Exp_Return_Date attribute of the book which can be null if the book is available.
3. Students and faculty can generate a borrow history report from the transaction table which shows all of their active and past loans.
4. Librarians can update the entries in the book table and manually mark books as available or not.
5. Librarians can generate reports based on book availability and overdue books.

Notable SQL Queries

There are basic SQL queries for adding a book (librarian) and borrowing a book (student or faculty):

```
INSERT INTO books (title, author, isbn,
pub_year, category, availability,
exp_return_date)
VALUES ('title_value', 'author_value',
'isbn_value', 'pub_year_value',
'category_value', 'availability_value',
NULL);
```

```
await db.run('INSERT INTO books
(book_id, title, author, isbn,
pub_year, category, availability,
exp_return_date) VALUES (?, ?, ?,
?, ?, ?, ?, ?)', [isbn, title,
author, isbn, pub_year, category,
availability, null]);
```

On the left hand side, you can see the standard SQL query we learned earlier in class. Using the JavaScript sqlite3 library, we can easily translate that query into JavaScript code (right). This can be done for getting all the books a student borrowed as well:

```
db = await openDb();
const query = `
SELECT books.title, books.author, transactions.book_id, transactions.checkout_date
FROM transactions
INNER JOIN books ON transactions.book_id = books.isbn
WHERE transactions.student_id = ? AND transactions.checkin_date IS NULL`;
const borrowedBooks = await db.all(query, [studentId]);
```

Here, you can see that we can store the general query as a string, while we input the studentId parameter as a variable using the sqlite3 library. Other than the variable addition, the query is identical to the one we used in class.

```
db = await openDb();
const query = `
SELECT books.category, COUNT(*) as count
FROM transactions
INNER JOIN books ON transactions.book_id = books.isbn
GROUP BY books.category`;
const borrowedBooksPerGenre = await
      db.all(query);
```

This is the query to get all the books per genre.

```
const conditions = [];
      const params = [];
      if (author) {
          conditions.push("author LIKE ?");
          params.push('%' + author + '%');
      }
      if (isbn) {
          conditions.push("isbn = ?");
          params.push(isbn);
      }
      if (category) {
          conditions.push("category LIKE ?");
          params.push('%' + category + '%');
      }
      if (title) {
          conditions.push("title LIKE ?");
          params.push('%' + title + '%');
      }
      const queryString = `SELECT * FROM books WHERE ${conditions.join(' AND ')}`;
      console.log("Executing query:", queryString, params);  // Log the final query
```

Lastly, sqlite3 also allows the use of conditionals to decide what query filtering conditions to implement. If we only pass in the author and category variables, the query will look like this:

```
SELECT * FROM books WHERE author LIKE ? AND category LIKE ?
```

Host Language Code

The host language utilized in our project is JavaScript in tandem with Node.js to help build a backend to interact with SQLite using the node module sqlite3.