Practical SQL

Designing and Querying an E-Library Database

Aisah Taufik Hidayat Abdullah

# Project’s Objectives

## Introduction

This report outlines the design and implementation of a database system for an e-library application. The system is designed to manage multiple libraries, handle diverse book collections, and support user interactions such as borrowing and holding books. We will cover the database design, implementation in PostgreSQL, and querying for analysis.

## Objective

To create a robust database schema for an e-library system that efficiently manages multiple libraries, tracks book inventories, and supports user interactions with books, including borrowing and holding. The database design aims to provide a comprehensive solution for managing books and users while ensuring data integrity and ease of access.

# Designing The Database

## Mission Statement

Our goal is to develop a database schema that supports:

* Multiple libraries each with unique collections of books.
* Efficient tracking of book availability and user interactions.
* Support for user borrowing and placing holds on books.
* Accurate record-keeping of loans and holds with defined constraints.

## Table Structures

1. Library Table

Table Name: Library

Description: Stores information about libraries.

Fields:

* + library\_id (INT, Primary Key, AUTO\_INCREMENT)
  + name (VARCHAR, NOT NULL)
  + location (VARCHAR, NOT NULL)

1. Book Table

Table Name: Book

Description: Contains details about books available in the libraries.

Fields:

* book\_id (INT, Primary Key, AUTO\_INCREMENT)
* title (VARCHAR, NOT NULL)
* author (VARCHAR, NOT NULL)
* category (VARCHAR, NOT NULL)
* quantity (INT, NOT NULL, CHECK(quantity >= 0))
* library\_id (INT, Foreign Key, references Library(library\_id))

1. Users Table

Table Name: User

Description: Holds user information for interacting with the library system.

Fields:

* user\_id (INT, Primary Key, AUTO\_INCREMENT)
* username (VARCHAR, UNIQUE, NOT NULL)
* email (VARCHAR, UNIQUE, NOT NULL)
* password (VARCHAR, NOT NULL)

1. Loan Table

Table Name: Loan

Description: Manages loan transactions of books by users.

Fields:

* loan\_id (INT, Primary Key, AUTO\_INCREMENT)
* user\_id (INT, Foreign Key, references Users(user\_id))
* book\_id (INT, Foreign Key, references Book(book\_id))
* loan\_date (DATE, NOT NULL)
* due\_date (DATE, NOT NULL)
* return\_date (DATE)

1. Hold Table

Table Name: Hold

Description: Tracks holds placed by users on unavailable books.

Fields:

* hold\_id (INT, Primary Key, AUTO\_INCREMENT)
* user\_id (INT, Foreign Key, references Users(user\_id))
* book\_id (INT, Foreign Key, references Book(book\_id))
* hold\_date (DATE, NOT NULL)
* status (ENUM('active', 'expired', 'fulfilled'), NOT NULL)

## Table Relationships

Library and Book: One-to-Many (One library has many books)

User and Loan: One-to-Many (One user can have many loans)

Book and Loan: One-to-Many (One book can be loaned multiple times)

User and Hold: One-to-Many (One user can place many holds)

Book and Hold: One-to-Many (One book can have many holds)

## Business Rules

Books:

* Title, author, and quantity are required fields.
* Quantity must be non-negative.

Loans:

* Must include loan date and due date.
* Return date can be NULL if the book hasn’t been returned.

Holds:

* Status must be one of 'active', 'expired', or 'fulfilled'.

# Implementing the Design

## Creating Tables in PostgreSQL

To implement the database schema, the following SQL statements were used:

|  |
| --- |
| CREATE TABLE Library (  library\_id SERIAL PRIMARY KEY,  name VARCHAR(255) NOT NULL,  location VARCHAR(255) NOT NULL  );  CREATE TABLE Book (  book\_id SERIAL PRIMARY KEY,  title VARCHAR(255) NOT NULL,  author VARCHAR(255) NOT NULL,  category VARCHAR(255) NOT NULL,  quantity INT NOT NULL CHECK(quantity >= 0),  library\_id INT REFERENCES Library(library\_id)  );  CREATE TABLE Users (  user\_id SERIAL PRIMARY KEY,  username VARCHAR(255) UNIQUE NOT NULL,  email VARCHAR(255) UNIQUE NOT NULL,  password VARCHAR(255) NOT NULL  );  CREATE TABLE Loan (  loan\_id SERIAL PRIMARY KEY,  user\_id INT REFERENCES Users(user\_id),  book\_id INT REFERENCES Book(book\_id),  loan\_date DATE NOT NULL,  due\_date DATE NOT NULL,  return\_date DATE  );  CREATE TABLE Hold (  hold\_id SERIAL PRIMARY KEY,  user\_id INT REFERENCES Users(user\_id),  book\_id INT REFERENCES Book(book\_id),  hold\_date DATE NOT NULL,  status ENUM('active', 'expired', 'fulfilled') NOT NULL  ); |

## Populating the Database

The following Python script generates and saves dummy data into CSV files, which are then imported into the PostgreSQL database:

|  |
| --- |
| import pandas as pd  from faker import Faker  import random  from datetime import datetime, timedelta  # Initialize Faker and define constants  fake = Faker()  NUM\_LIBRARIES = 10  NUM\_BOOKS = 100  NUM\_USERS = 40  NUM\_LOANS = 80  NUM\_HOLDS = 50  # Generate data for each table  # Library Data  libraries = [{'library\_id': i, 'name': fake.company(), 'location': fake.address()} for i in range(1, NUM\_LIBRARIES + 1)]  # Book Data  books = [{'book\_id': i, 'title': fake.catch\_phrase(), 'author': fake.name(), 'category': random.choice(['Self-Improvement', 'Biography', 'Fantasy', 'Romance', 'Science Fiction']), 'quantity': random.randint(1, 10), 'library\_id': random.randint(1, NUM\_LIBRARIES)} for i in range(1, NUM\_BOOKS + 1)]  # User Data  users = [{'user\_id': i, 'username': fake.user\_name(), 'email': fake.email(), 'password': fake.password()} for i in range(1, NUM\_USERS + 1)]  # Loan Data  loans = [{'loan\_id': i, 'user\_id': random.randint(1, NUM\_USERS), 'book\_id': random.randint(1, NUM\_BOOKS), 'loan\_date': fake.date\_between(start\_date='-30d', end\_date='today'), 'due\_date': fake.date\_between(start\_date='today', end\_date='+14d'), 'return\_date': random.choice([None, fake.date\_between(start\_date='-30d', end\_date='today')])} for i in range(1, NUM\_LOANS + 1)]  # Hold Data  holds = [{'hold\_id': i, 'user\_id': random.randint(1, NUM\_USERS), 'book\_id': random.randint(1, NUM\_BOOKS), 'hold\_date': fake.date\_between(start\_date='-30d', end\_date='today'), 'status': random.choice(['active', 'expired', 'fulfilled'])} for i in range(1, NUM\_HOLDS + 1)]  # Convert to DataFrames and save as CSV  pd.DataFrame(libraries).to\_csv('Library.csv', index=False)  pd.DataFrame(books).to\_csv('Book.csv', index=False)  pd.DataFrame(users).to\_csv('Users.csv', index=False)  pd.DataFrame(loans).to\_csv('Loan.csv', index=False)  pd.DataFrame(holds).to\_csv('Hold.csv', index=False) |

The data generated by the Python script, including information about libraries, books, users, loans, and holds, was successfully created and then imported into a PostgreSQL database. This process involved generating realistic, yet fictional, data using the Faker library, saving the data as CSV files, and subsequently importing these CSV files into PostgreSQL to populate the relevant database tables. This allowed for a comprehensive dataset to be available within PostgreSQL for further analysis.

# Retrieve Data

## Querying and Analysis

These queries provide essential insights into the operation of the e-library system. They help the library staff make informed decisions about collection management, user engagement, and resource allocation. By regularly running these queries, the library can ensure it meets the needs of its users while maintaining an efficient and well-balanced collection.

### Most Popular Book Categories

Importance:

* Collection Development: Understanding which book categories are most prevalent in the library system helps librarians and administrators make informed decisions about expanding or reducing specific genres. If a particular category, such as "Fantasy" or "Science Fiction," has a high count, it may indicate a strong demand, suggesting that more books in these categories should be acquired.
* User Satisfaction: By ensuring that the library's collection reflects user interests, the library can enhance user satisfaction and engagement. Popular categories might indicate the genres users are most interested in, guiding future acquisitions.
* Budget Allocation: Libraries often operate with limited budgets, so knowing which categories are most popular helps in prioritizing funds. Investing in more books from popular categories could lead to better utilization of resources.

|  |
| --- |
| SELECT category, COUNT(\*) AS book\_count  FROM Book  GROUP BY category  ORDER BY book\_count DESC; |

### Users with Overdue Books

Importance:

* Overdue Management: Identifying users who have overdue books is crucial for managing the library's inventory. It allows the library to take action, such as sending reminders or applying fines, to encourage timely returns.
* Inventory Control: Overdue books reduce the number of available copies for other users. By managing overdue items effectively, the library can ensure that books are returned on time and are available for other patrons to borrow.
* User Accountability: Keeping track of overdue books helps in maintaining a record of user reliability. This data can be used to enforce rules or policies regarding borrowing limits or access privileges for users with frequent overdue returns.

|  |
| --- |
| SELECT U.username, B.title, L.due\_date  FROM Loan L  JOIN Users U ON L.user\_id = U.user\_id  JOIN Book B ON L.book\_id = B.book\_id  WHERE L.return\_date IS NULL AND L.due\_date < CURRENT\_DATE; |

### Current Holds on Books

Importance:

* Hold Queue Management: Understanding which books are currently on hold allows the library to manage the hold queue effectively. This is particularly important for popular books with limited copies, ensuring that they are distributed fairly among users.
* User Satisfaction: By monitoring active holds, the library can ensure that users are informed when a book becomes available. This enhances user satisfaction by minimizing wait times and improving communication.
* Resource Optimization: Active holds indicate which books are in high demand but not immediately available. The library can use this information to consider purchasing additional copies or managing loan periods more strictly to reduce wait times.

|  |
| --- |
| SELECT U.username, B.title, H.hold\_date  FROM Hold H  JOIN Users U ON H.user\_id = U.user\_id  JOIN Book B ON H.book\_id = B.book\_id  WHERE H.status = 'active'; |

### Average Number of Books Borrowed per User

Importance:

* User Engagement: This query helps the library understand overall user engagement by measuring how many books, on average, a user borrows. A higher average might indicate strong user engagement and a well-utilized library, while a lower average could suggest areas for improvement in user outreach or collection appeal.
* Service Evaluation: Tracking borrowing patterns can help the library evaluate the effectiveness of its services. If the average is low, the library might investigate why users are not borrowing more books and address potential barriers, such as limited collection diversity or inconvenient loan periods.
* Policy Development: Insights from this query can inform policy decisions, such as adjusting the maximum number of books users can borrow or the duration of loan periods, to better match user needs and maximize resource utilization.

|  |
| --- |
| SELECT AVG(book\_count) AS avg\_books\_borrowed  FROM (  SELECT user\_id, COUNT(\*) AS book\_count  FROM Loan  GROUP BY user\_id  ) AS user\_loans; |

### Library with the Most Books

Importance:

* Resource Allocation: Knowing which library has the most books can help in making decisions about resource allocation. For instance, if one library has a significantly larger collection, it may indicate an imbalance that could be addressed by redistributing resources or expanding facilities at other locations.
* Strategic Planning: This information is valuable for long-term strategic planning. Libraries with large collections may serve as central hubs, so understanding their capacity can guide decisions about where to expand or upgrade services.
* User Access: The library with the most books may also attract more users, so understanding this dynamic helps in planning user services, events, and staffing to match the demand.

|  |
| --- |
| SELECT L.name, COUNT(B.book\_id) AS total\_books  FROM Library L  JOIN Book B ON L.library\_id = B.library\_id  GROUP BY L.name  ORDER BY total\_books DESC  LIMIT 1; |

# References

* Towards Data Science. (2020). Practical SQL: Create and Query a Relational Database. Retrieved from <https://towardsdatascience.com/practical-sql-create-and-query-a-relational-database-8bac84d78703>
* Towards Data Science. (2020). Practical SQL: Designing and Creating a Relational Database. Retrieved from <https://towardsdatascience.com/practical-sql-designing-and-creating-a-relational-database-de31c40b853f>
* GitHub. (2019). Hospital Management System by Anwesh90. Retrieved from https://github.com/anwesh90/Hospital-Management-System
* Medium. (2020). Creating a Modern Library Database. Retrieved from <https://medium.com/@danishman/creating-a-modern-library-database-b7dff4313f28>
* Vertabelo. (2021). Database Design for a Library System. Retrieved from <https://vertabelo.com/blog/database-for-library-system/>