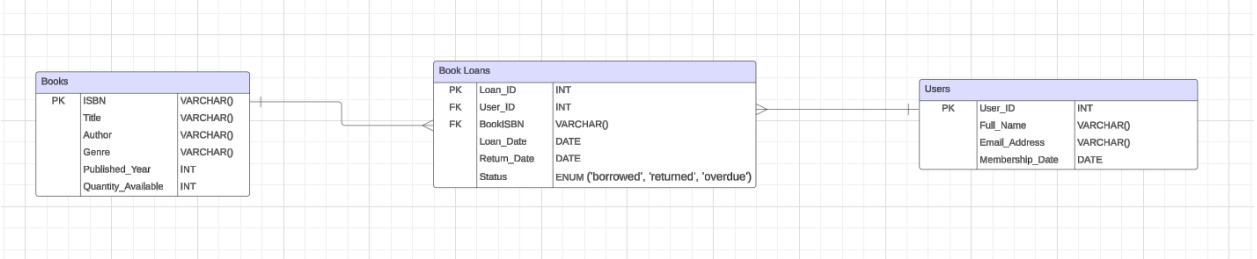
Database

Lab - 4

SE 2141

Alair, Shawn Khennee S.

Part 1 : Conceptual Design



https://lucid.app/lucidchart/6444ee76-445e-4805-a374-5e5f420202a9/edit?viewport_loc=-1658%2C388%2C3677%2C1783%2C0_0&invitationId=inv_0be1aaad-cc0d-4864-a3d7-22cb538d5bbe

Entity : - Book Attributes : - ISBN (PK) - Title - Author - Genre - Published_Year - Quantity_Available	Entity : - Book Loan Attributes : - Loan_Id (PK) - User_Id (FK) - BookISBN (FK) - Loan_Date - Return_Date - Status	Entity : - Users Attributes : - User_Id - Full_Name - Email_Address - Membership_Date

Part 2 : Logical Design

Book Table :

```
CREATE TABLE Books (
    ISBN VARCHAR(20) PRIMARY KEY,
    Title VARCHAR(255) NOT NULL,
    Author VARCHAR(255) NOT NULL,
    Genre VARCHAR(100),
    Published_Year INT,
    Quantity_Available INT
    CHECK (Quantity_Available >= 0)
);
```

Book Loan Table :

```
CREATE TABLE Book_Loans (
    Loan_ID SERIAL PRIMARY KEY,
    User_ID INT NOT NULL,
    Book_ISBN VARCHAR(20) NOT NULL,
    Loan_Date DATE NOT NULL,
    Return_Date DATE,
    Status VARCHAR(50)
    CHECK (Status IN ('borrowed', 'returned', 'overdue')),
    FOREIGN KEY (User_ID)
    REFERENCES Users(User_ID)
    ON DELETE CASCADE,
    FOREIGN KEY (Book_ISBN)
    REFERENCES Books(ISBN)
    ON DELETE CASCADE
);
```

Users Table :

```
CREATE TABLE Users (
    User_ID INT PRIMARY KEY,
    Full_Name VARCHAR(255) NOT NULL,
    Email VARCHAR(255) NOT NULL UNIQUE,
    Membership_Date DATE NOT NULL
);
```

Part 3 : SQL Queries

a. Insert a new book in the library with a quantity of 5.

```
INSERT INTO Books (Title, Author, ISBN, Genre, Published_Year,
Quantity_Available)
VALUES
('The Great Pinoy', 'Jose Suoberon', '9780743273565', 'Fiction', 1925, 5),
('One vs 100', 'Krystal Siaotong', '9780451524935', 'Dystopian', 1949, 3),
('Karaoke Crypto', 'Francis Amada', '9780061120084', 'Fiction', 1960, 4),
('Kingkong Chips', 'Joshua Samenian', '9781503280786', 'Adventure', 1851, 2);
```

b. Add a new user to the system.

```
INSERT INTO Users (User_ID, Full_Name, Email, Membership_Date)
VALUES
(1, 'Jane Doe', 'janedoe@example.com', '2024-12-10'),
(2, 'John Doe', 'johndoe@example.com', '2024-11-15'),
(3, 'Jin Doe', 'jindoe@example.com', '2024-08-20');
```

c. Record a book loan for a user.

```
INSERT INTO Book_Loans (User_ID, Book_ISBN, Loan_Date, Return_Date, Status)
VALUES
(1, '9780743273565', '2024-12-10', NULL, 'borrowed'),
(2, '9780451524935', '2024-11-20', NULL, 'borrowed'),
(3, '9780061120084', '2024-08-15', '2024-09-01', 'returned'),
(2, '9780061120084', '2024-11-25', NULL, 'borrowed');
```

d. Find all books borrowed by a specific user.

```
SELECT Books.Title,
  Books.Author,
  Books.Genre,
  Books.Published_Year,
  Book_Loans.Loan_Date,
  Book_Loans.Status
FROM Book_Loans
JOIN Books
ON Book_Loans.Book_ISBN = Books.ISBN
WHERE Book_Loans.User_ID = 2;
```

e. List all overdue loans.

```
SELECT Users.Full_Name,
```

```
  Books.Title,
  Books.Author,
  Book_Loans.Loan_Date,
  Book_Loans.Return_Date,
  Book_Loans.Status
FROM Book_Loans
JOIN Books
ON Book_Loans.Book_ISBN = Books.ISBN
JOIN Users
ON Book_Loans.User_ID = Users.User_ID
WHERE Book_Loans.Status = 'overdue';
```

Part 4 :

To anticipate borrowing books when no duplicates are accessible, a database trigger is executed on the Book_Loans table. The trigger checks the Quantity_Available within the Books table some time recently, embedding an advance. In the event that no duplicates are accessible, it raises an exemption and squares the operation. For substantial credits, the trigger decreases the Quantity_Available by one. To optimize execution, a file on the ISBN column guarantees proficient lookups. Furthermore, application-level checks can inquire accessibility and inform clients some time recently submitting an advance request. This combined approach keeps up information judgment, anticipates over-borrowing, and upgrades the client involvement.

Fast Retrieval :

```sql
CREATE INDEX idx_status ON Book_Loans(Status);

CREATE INDEX idx_user_book_status ON Book_Loans(User_ID, Book_ISBN, Status);

SELECT Users.Full_Name,
  Books.Title,
  Books.Author,
  Book_Loans.Loan_Date,
  Book_Loans.Return_Date,
  Book_Loans.Status
FROM Book_Loans
JOIN Books
ON Book_Loans.Book_ISBN = Books.ISBN
JOIN Users
ON Book_Loans.User_ID = Users.User_ID
WHERE Book_Loans.Status = 'overdue';
```

```
QUERY PLAN
"Nested Loop  (cost=0.28..5.90 rows=1 width=1674) (actual time=0.045..0.046 rows=1 loops=1)"
"  ->  Nested Loop  (cost=0.14..3.48 rows=1 width=1162) (actual time=0.038..0.039 rows=1 loops=1)"
"        ->  Seq Scan on book_loans  (cost=0.00..1.05 rows=1 width=188) (actual time=0.016..0.017 rows=1 loops=1)"
"              Filter: ((status)::text = 'overdue'::text)"
"              Rows Removed by Filter: 3"
"        ->  Index Scan using books_pkey on books  (cost=0.14..2.36 rows=1 width=1090) (actual time=0.019..0.020 rows=1 loops=1)"
"              Index Cond: ((isbn)::text = (book_loans.book_isbn)::text)"
"  ->  Index Scan using users_pkey on users  (cost=0.14..2.36 rows=1 width=520) (actual time=0.006..0.006 rows=1 loops=1)"
"        Index Cond: (user_id = book_loans.user_id)"
"Planning Time: 1.145 ms"
"Execution Time: 0.163 ms"
```

Part 5 : Reflection

Since the database grows every second, retrieving data takes longer, this means we can have ***slow queries***.To help us analyze and optimize our queries using the EXPLAIN tool.

Having thousands to millions of data can lead to ***storage problems*** and it can cost much. We can optimize it by dividing the database into partitions similar to disk partitions when using multiple drives to save space.

We can get delays if there are ***too many users*** accessing the database at the same time. We can implement a read-replica setup, where copies of the database handle read-requests which can reduce the load on the main database.

If one has a large database it can be a hassle to have it backed-up, which can lead to ***backup delays*** and it will take more time for the database to be available. By using incremental backups we can have the database backup faster and efficiently.

When having multiple databases, ***consistency*** is important so that data will always be accurate across the different databases/servers. Databases like PostgreSQL have a logical replication that ensures consistency.