Contents

Introduction

Library management is a crucial aspect in any institution that requires organization, management, and efficient use of available resources to enhance the user experience. It is established that the Library Management System developed for this purpose, uses a structured SQL database to store huge amount of data regarding books, authors, genres, members and borrowing records. The database is constructed on SQLite and allows it to execute elaborate query assistance that helps to solve commonplace procedures of library work and develop strategic long-term plans. The schema has been implemented with considerable effort in order to meet the objectives of data consistency and the elimination of excessive duplication of information recorded within the tables of the system, thereby making the system functional and expandable. The purpose of this report is to describe the series of SQL queries that represent the database's capacity for effective library data administration and analysis.

Project Topic: Library Management System

Database Schema

Tables

- Books: Contains details about the books in the library.
- Members: Stores information about library members.
- Borrow: Records details of book borrowings.
- Authors: Information about book authors.
- Genres: Different genres available in the library.

Columns and Data Types

- Books: BookID (nominal), Title (nominal), AuthorID (nominal), GenreID (nominal), PublishYear (interval), CopiesAvailable (ratio).
- Members: MemberID (nominal), Name (nominal), MembershipType (ordinal: Regular, Premium, VIP), JoinDate (interval).
- Borrow: BorrowID (nominal), MemberID (nominal), BookID (nominal), BorrowDate (interval), ReturnDate (interval).
- Authors: AuthorID (nominal), Name (nominal), BirthYear (interval).
- Genres: GenreID (nominal), GenreName (nominal).

Relationships

Foreign Keys

- Books.AuthorID references Authors.AuthorID.
- Books.GenreID references Genres.GenreID.
- Borrow.BookID references Books.BookID.
- Borrow.MemberID references Members.MemberID.

Table Creating Queries

```
CREATE TABLE Authors (
  AuthorID INTEGER PRIMARY KEY,
  Name TEXT,
  BirthYear INTEGER
);

CREATE TABLE Genres (
  GenreID INTEGER PRIMARY KEY,
  GenreName TEXT
);

CREATE TABLE Books (
  BookID INTEGER PRIMARY KEY,
  Title TEXT,
  AuthorID INTEGER,
  GenreID INTEGER,
  PublishYear INTEGER,
  CopiesAvailable INTEGER,
      FOREIGN KEY (AuthorID) REFERENCES Authors
(AuthorID),
  FOREIGN KEY (GenreID) REFERENCES Genres (GenreID)
);

CREATE TABLE Members (
  MemberID INTEGER PRIMARY KEY,
  Name TEXT,
  MembershipType TEXT,
  JoinDate DATE
);

CREATE TABLE Borrow (
  BorrowID INTEGER PRIMARY KEY,
  MemberID INTEGER,
  BookID INTEGER,
  BorrowDate DATE,
  ReturnDate DATE,
    FOREIGN KEY (MemberID) REFERENCES Members
(MemberID),
  FOREIGN KEY (BookID) REFERENCES Books (BookID)
);
```

Data Generation: Explanation of Methods Used

In the Library Management System database project, realistic data was created with a lot of precision in order to mimic true to life library environment. Regarding the Authors table, names were selected from the list of famous authors, while birth years were generated randomly with the help of Python. randint function that creates years within the range of 1900 to 2000. This approach effectively creates a balance between the latest authors and the classic ones so as to imitate a library.

For the Books table, since it contained numerous books, titles themselves were created in the form of dynamic string, with an added integer number after the string “Book Title”. The fields AuthorID and GenreID were filled in a random manner resembling the manner of a library that contains rather vast and diverse assortment of books. The structure of the inputs of each book was given a publication year ranging from 1900 to 2023 to maintain realistic publication of books. The number of copy that is available was set randomly between 0 and 20 to mimic the variation within the number of books in circulation attributed to popularity.

For the tables Members, some random data for the fields ‘membership date’ have been created using a function random_date() which generates a date which is within a believable range. Members’ JoinDate was generated to be in the last 20 years, and for each borrowing record, BorrowDate and ReturnDate were synthesized to mimic the realistic borrowing and return practices, where the return date is 1-30 days after the borrow date. Thus, the rigorous process of generating date contributes positively to the creation and sustenance of realistic fake data.

These methods would help create a balanced picture of the library activity, and the data would be produced to correspond to all the four types of variables: nominal, ordinal, interval, and ratio The use of Python means data would be randomized to provide sufficient variety and realism to the data in the created database.

Python Code

```
import sqlite3
import random
from datetime import datetime, timedelta


# Function to generate random dates
def random_date(start, end):
    return start + timedelta(
        seconds=random.randint(0, int((end - start).total_seconds())),
    )


# Connect to SQLite database
conn = sqlite3.connect('library_management.db')
c = conn.cursor()


# Create tables
c.execute('''
CREATE TABLE IF NOT EXISTS Authors (
    AuthorID INTEGER PRIMARY KEY,
    Name TEXT,
    BirthYear INTEGER
);
''')
```

```
c.execute('''
CREATE TABLE IF NOT EXISTS Genres (
    GenreID INTEGER PRIMARY KEY,
    GenreName TEXT
);
''')


c.execute('''
CREATE TABLE IF NOT EXISTS Books (
    BookID INTEGER PRIMARY KEY,
    Title TEXT,
    AuthorID INTEGER,
    GenreID INTEGER,
    PublishYear INTEGER,
    CopiesAvailable INTEGER,
    FOREIGN KEY (AuthorID) REFERENCES Authors (AuthorID),
    FOREIGN KEY (GenreID) REFERENCES Genres (GenreID)
);
''')


c.execute('''
CREATE TABLE IF NOT EXISTS Members (
    MemberID INTEGER PRIMARY KEY,
    Name TEXT,
    MembershipType TEXT,
    JoinDate DATE
);
''')


c.execute('''
CREATE TABLE IF NOT EXISTS Borrow (
    BorrowID INTEGER PRIMARY KEY,
    MemberID INTEGER,
    BookID INTEGER,
    BorrowDate DATE,
    ReturnDate DATE,
    FOREIGN KEY (MemberID) REFERENCES Members (MemberID),
    FOREIGN KEY (BookID) REFERENCES Books (BookID)
);
''')


# Insert random data
# Example names and genres
authors = ["Alice Munro", "James Patterson", "J.K. Rowling", "Stephen King", "Agatha Christie"]
genres = ["Fiction", "Non-fiction", "Science fiction", "Mystery", "Biography"]


# Add Authors
for i, author in enumerate(authors, 1):
        c.execute('INSERT INTO Authors (AuthorID, Name, BirthYear) VALUES (?, ?, ?)', (i, author,
random.randint(1900, 2000)))


# Add Genres
for i, genre in enumerate(genres, 1):
    c.execute('INSERT INTO Genres (GenreID, GenreName) VALUES (?, ?)', (i, genre))


# Add Books
for i in range(1, 1000):  # Generate 100 books
    c.execute('INSERT INTO Books (BookID, Title, AuthorID, GenreID, PublishYear, CopiesAvailable) VALUES (?,
?, ?, ?, ?, ?)', (
        i, f"Book Title {i}", random.randint(1, len(authors)), random.randint(1, len(genres)), random.randint(1900,
2023), random.randint(0, 20)))


# Add Members
for i in range(1, 51):  # Generate 50 members
    c.execute('INSERT INTO Members (MemberID, Name, MembershipType, JoinDate) VALUES (?, ?, ?, ?)', (
        i, f"Member Name {i}", random.choice(["Regular", "Premium", "VIP"]), random_date(datetime.now() -
timedelta(days=20*365), datetime.now()).date()))


# Add Borrow Records
for i in range(1, 151):  # Generate 150 borrow records
    borrow_date = random_date(datetime.now() - timedelta(days=365), datetime.now())
    return_date = borrow_date + timedelta(days=random.randint(1, 30))
    c.execute('INSERT INTO Borrow (BorrowID, MemberID, BookID, BorrowDate, ReturnDate) VALUES (?, ?, ?,
?, ?)', (
        i, random.randint(1, 50), random.randint(1, 100), borrow_date.date(), return_date.date()))

# Commit changes and close connection
conn.commit()
conn.close()
```

Justification for Database Schema Design: Normalization and Data Redundancy Reduction

The database schema in the Library Management Sytem was designed with great care while choosing a logical model that is properly normalized so as to minimize occurrence of redundant data and hence achieving a greater level of integrity. The reasons behind the choice of having three different tables for Authors, Genres, and Members is based on the notion of normalization where database mainly follow the first three forms of normalization.

1. Authors and Genres Tables

Authors and Genres have been stored in two tables, Authors and Genres respectively with distinct keys AuthorID and GenreID, not repeating their details in the Books table. For instance, authors and genres of books are related, which means that their similarity may be high. For each book entry made without the use of separate tables, authors' names and genre types are to be entered in full, making for essentially anechoic entries. This results in extra bytes taken up by redundancy, and for data management, changes made to the authors, or even innovated genre types would have to be duplicated across multiple records. The Author table and the Genres table also provide an more efficient storage structure where each author and genre is stored only once and are related to other tables by foreign keys. This eliminates a lot of hassles when updating records as well as standardization is achieved since changing the value in the Authors or Genres table will automatically update all entries linked to it in the Books table.

2. Members Table

The Members table is designed to handle data related to library members separately from transactional data such as borrowings (Borrow table). This separation aligns with the need to manage member information independently of their activities, such as borrowing books. By maintaining member information in a dedicated table, the library system ensures that personal details are centralized and managed securely, facilitating easy updates and queries. For example, if a member's contact information or membership status changes, the update needs to be made in only one place. Moreover, separating transaction data into the Borrow table allows the library to track borrowing history without repeatedly storing member details like names or membership types with each transaction, thereby reducing redundancy and improving query performance.

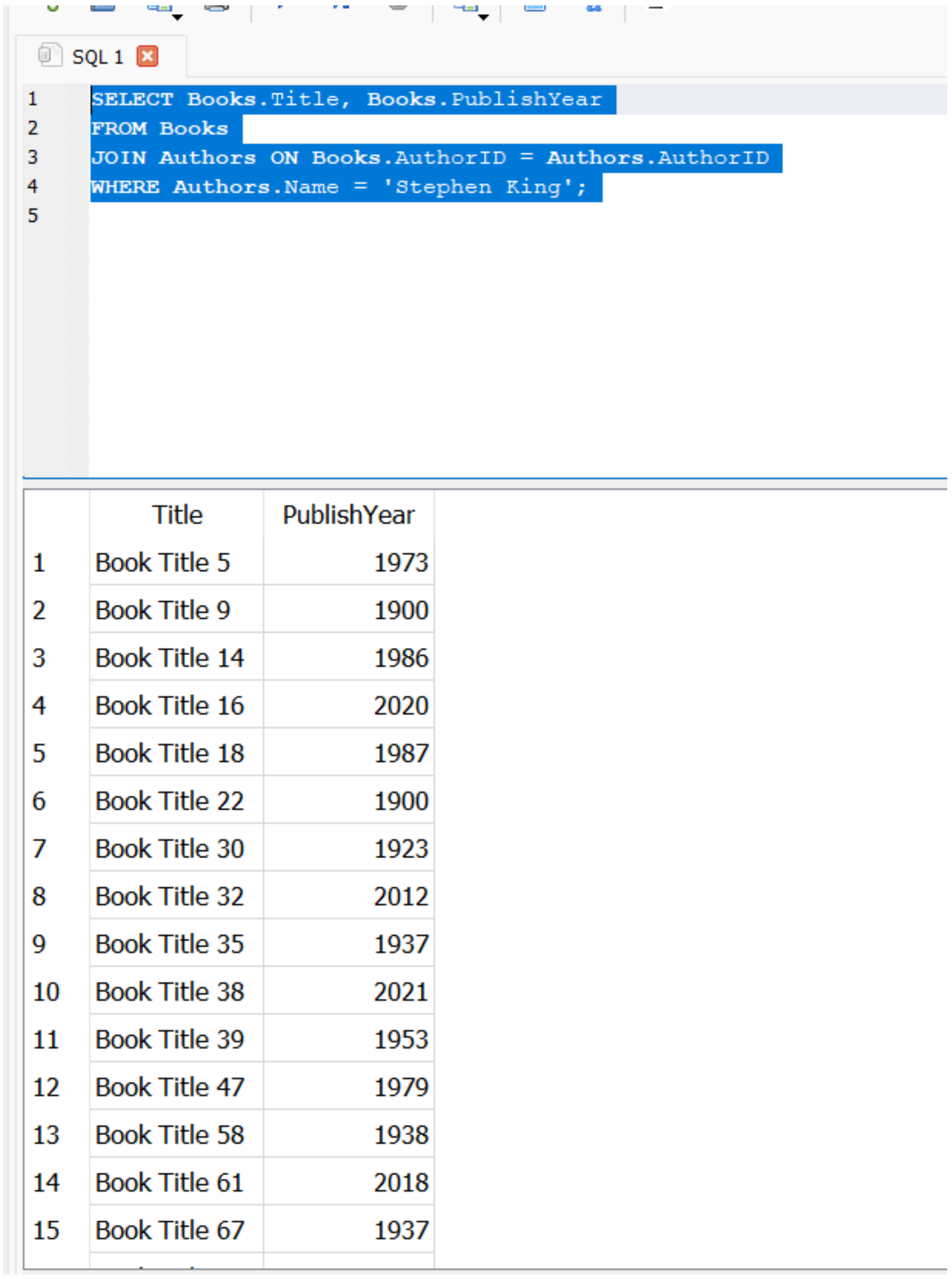
3. Reducing Update Anomalies and Improving Query Performance:
The use of different tables for Authors, Genres and Members minimize update anomalies where an update in a certain field will be a problem since the data is duplicated. For example, if there is some information about an author that must be changed, for example, the name, then it is easier to change it in one record Author than to change the same information added in several records of Books. Similarly, for the queries that seek books of a certain author or genre, the operations can be performed as joining the small normalized tables Authors and Genres with Books, rather than searching through the copies of repeated data as in the denormalized table.

Example SQL Queries for Library Management System

List All Books by a Specific Author

This query retrieves all books written by 'Stephen King', including their titles and the year they were published.

```
SELECT Books.Title, Books.PublishYear
FROM Books
JOIN Authors ON Books.AuthorID = Authors.AuthorID
WHERE Authors.Name = 'Stephen King';
```



SQL 1

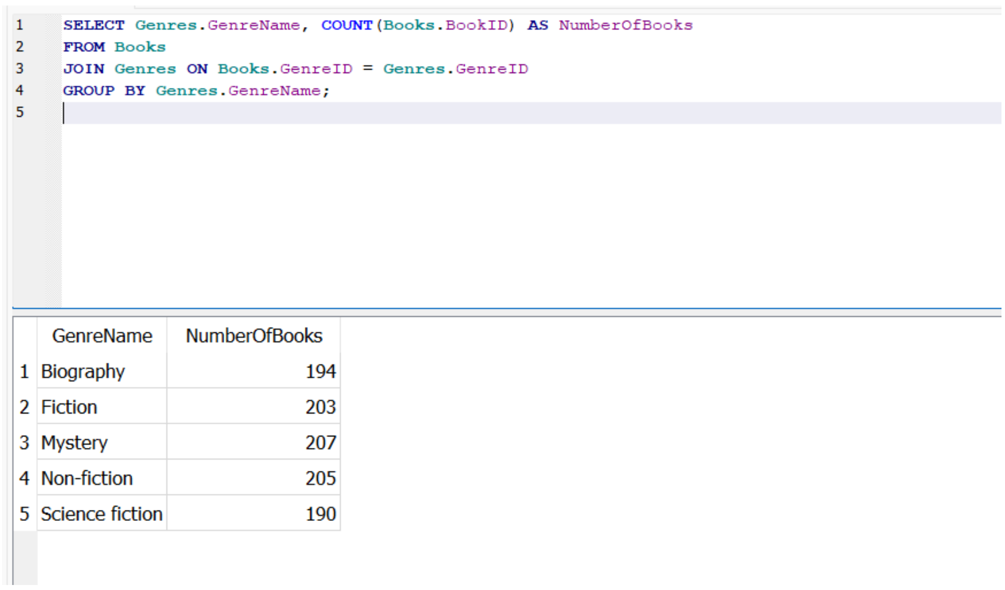
```
SELECT Books.Title, Books.PublishYear
FROM Books
JOIN Authors ON Books.AuthorID = Authors.AuthorID
WHERE Authors.Name = 'Stephen King';
```

	Title	PublishYear
1	Book Title 5	1973
2	Book Title 9	1900
3	Book Title 14	1986
4	Book Title 16	2020
5	Book Title 18	1987
6	Book Title 22	1900
7	Book Title 30	1923
8	Book Title 32	2012
9	Book Title 35	1937
10	Book Title 38	2021
11	Book Title 39	1953
12	Book Title 47	1979
13	Book Title 58	1938
14	Book Title 61	2018
15	Book Title 67	1937

Count Books in Each Genre

This query provides a summary of how many books are available in each genre, useful for inventory management and planning.

SELECT Genres.GenreName, COUNT(Books.BookID) AS NumberOfBooks
FROM Books
JOIN Genres ON Books.GenreID = Genres.GenreID
GROUP BY Genres.GenreName;



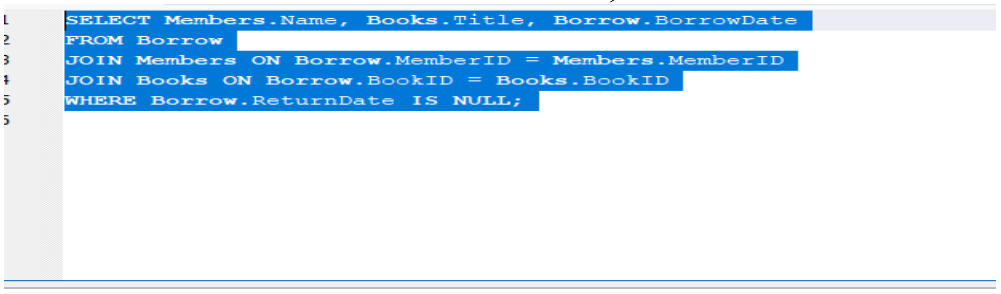
```sql
SELECT Genres.GenreName, COUNT(Books.BookID) AS NumberOfBooks
FROM Books
JOIN Genres ON Books.GenreID = Genres.GenreID
GROUP BY Genres.GenreName;
```

	GenreName	NumberOfBooks
1	Biography	194
2	Fiction	203
3	Mystery	207
4	Non-fiction	205
5	Science fiction	190

Find Members Who Have Not Returned Books

Retrieves a list of members who currently have borrowed books that have not yet been returned, along with the titles of those books and the borrow date.

SELECT Members.Name, Books.Title, Borrow.BorrowDate
FROM Borrow
JOIN Members ON Borrow.MemberID = Members.MemberID
JOIN Books ON Borrow.BookID = Books.BookID
WHERE Borrow.ReturnDate IS NULL;

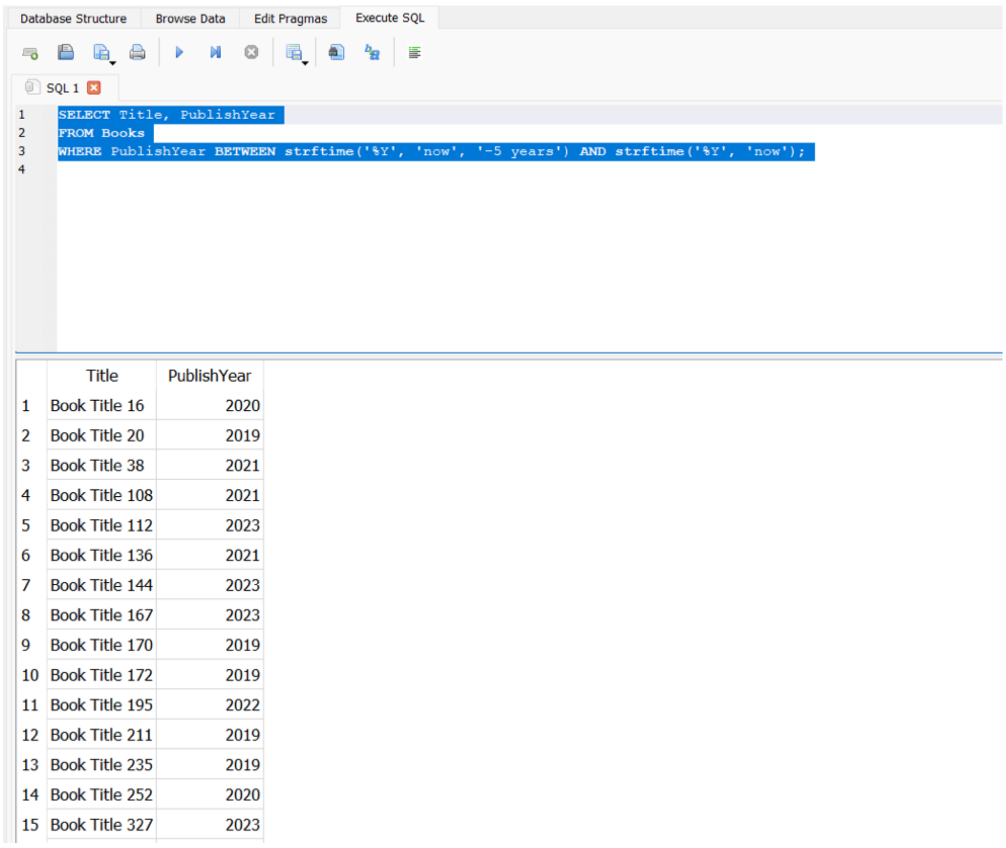


```sql
SELECT Members.Name, Books.Title, Borrow.BorrowDate
FROM Borrow
JOIN Members ON Borrow.MemberID = Members.MemberID
JOIN Books ON Borrow.BookID = Books.BookID
WHERE Borrow.ReturnDate IS NULL;
```

Books Added in the Last Five Years

Lists all books added to the library in the last five years, highlighting recent additions to the collection.

SELECT Title, PublishYear
FROM Books
WHERE PublishYear BETWEEN strftime('%Y', 'now', '-5 years') AND strftime('%Y', 'now');



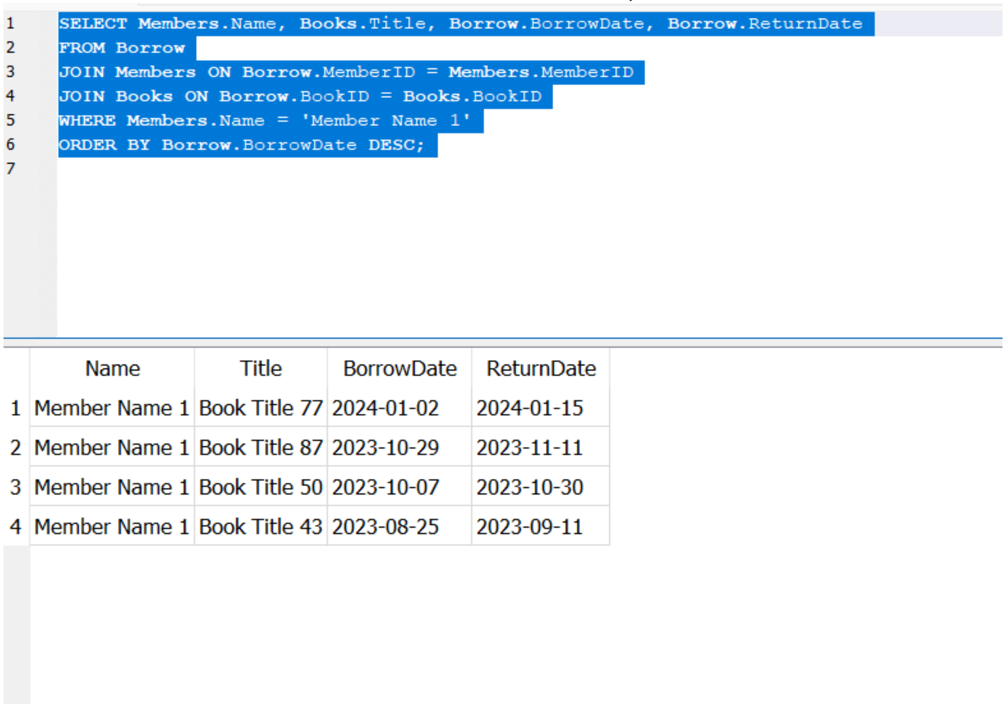
```
SELECT Title, PublishYear
FROM Books
WHERE PublishYear BETWEEN strftime('%Y', 'now', '-5 years') AND strftime('%Y', 'now');
```

	Title	PublishYear
1	Book Title 16	2020
2	Book Title 20	2019
3	Book Title 38	2021
4	Book Title 108	2021
5	Book Title 112	2023
6	Book Title 136	2021
7	Book Title 144	2023
8	Book Title 167	2023
9	Book Title 170	2019
10	Book Title 172	2019
11	Book Title 195	2022
12	Book Title 211	2019
13	Book Title 235	2019
14	Book Title 252	2020
15	Book Title 327	2023

Member Borrow History

This query provides a detailed history of all books borrowed by a specific member, sorted by borrow date. This is useful for analyzing member activity and preferences.

SELECT Members.Name, Books.Title, Borrow.BorrowDate, Borrow.ReturnDate
FROM Borrow
JOIN Members ON Borrow.MemberID = Members.MemberID
JOIN Books ON Borrow.BookID = Books.BookID
WHERE Members.Name = 'Member Name 1'
ORDER BY Borrow.BorrowDate DESC;



```
SELECT Members.Name, Books.Title, Borrow.BorrowDate, Borrow.ReturnDate
FROM Borrow
JOIN Members ON Borrow.MemberID = Members.MemberID
JOIN Books ON Borrow.BookID = Books.BookID
WHERE Members.Name = 'Member Name 1'
ORDER BY Borrow.BorrowDate DESC;
```

	Name	Title	BorrowDate	ReturnDate
1	Member Name 1	Book Title 77	2024-01-02	2024-01-15
2	Member Name 1	Book Title 87	2023-10-29	2023-11-11
3	Member Name 1	Book Title 50	2023-10-07	2023-10-30
4	Member Name 1	Book Title 43	2023-08-25	2023-09-11

Conclusion

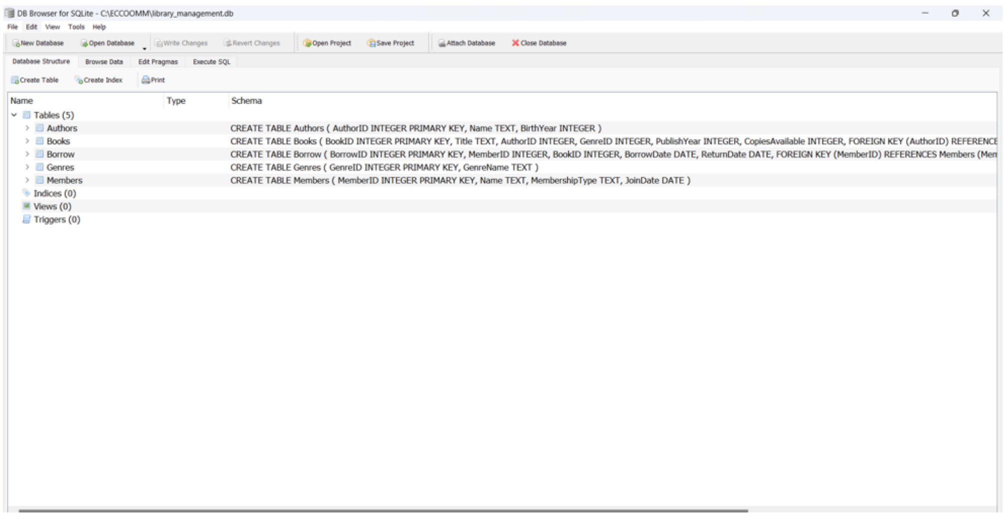
The queries illustrated in this report show that the LMS database is quite comprehensive when it comes to the functional aspect. In this way, these queries allow to identify specifics of books' management, members' activity level, and the most popular genres, which ultimately helps to improve service quality and overall organizational effectiveness. This way of database design protects data integrity and production that is vital to any efficient and user competent system. Looking at the future, the kind of database that is implemented here not only caters for the present library management operations but also holds the potential to expand as the library Tx develops more services as well as collections. It is a basic application that can greatly improve interaction between the technology and the library in order to promote a better environment in the library.

References

Connolly, T., & Begg, C. (2014). Database systems: A practical approach to design, implementation, and management (6th ed.). Pearson Education Limited.
Date, C. J. (2012). An introduction to database systems (8th ed.). Pearson Education.
Elmasri, R., & Navathe, S. B. (2015). Fundamentals of database systems (7th ed.). Pearson.
Hoffer, J. A., Venkataraman, R., & Topi, H. (2013). Modern database management (12th ed.). Pearson.
Kroenke, D., & Auer, D. (2017). Database concepts (8th ed.). Pearson.
Pratt, P. J., & Adamski, J. J. (2012). Concepts of database management (7th ed.). Cengage Learning.
Silberschatz, A., Korth, H. F., & Sudarshan, S. (2011). Database system concepts (6th ed.). McGraw-Hill.
Ullman, J. D., & Widom, J. (2014). A First Course in Database Systems (3rd ed.). Prentice Hall.
Garcia-Molina, H., Ullman, J. D., & Widom, J. (2009). Database systems: The complete book (2nd ed.). Pearson.
Ramakrishnan, R., & Gehrke, J. (2003). Database management systems (3rd ed.). McGraw-Hill Higher Education.

Appendix

Use of SQLite3 browser



Name	Type	Schema
Tables (5)		
Authors		CREATE TABLE Authors (AuthorID INTEGER PRIMARY KEY, Name TEXT, BirthYear INTEGER)
Books		CREATE TABLE Books (BookID INTEGER PRIMARY KEY, Title TEXT, AuthorID INTEGER, GenreID INTEGER, PublishYear INTEGER, CopiesAvailable INTEGER, FOREIGN KEY (AuthorID) REFERENCE
Borrow		CREATE TABLE Borrow (BorrowID INTEGER PRIMARY KEY, MemberID INTEGER, BookID INTEGER, BorrowDate DATE, ReturnDate DATE, FOREIGN KEY (MemberID) REFERENCES Members (Mem
Genres		CREATE TABLE Genres (GenreID INTEGER PRIMARY KEY, GenreName TEXT)
Members		CREATE TABLE Members (MemberID INTEGER PRIMARY KEY, Name TEXT, MembershipType TEXT, JoinDate DATE)
Indices (0)		
Views (0)		
Triggers (0)		