

CS504 Principles of Data Management and Mining

Report on
LIBRARY MANAGEMENT SYSTEM
PROJECT

Student
Chanyoung Park
G01408089

05/03/2023

Table of Contents

Introduction	3
Project Overview	3
Project Scenario	4
Requirements	4
Database Design	5
Entities and Relationships.....	5
Entity-Relationship (ER) Diagram.....	7
Database Normalization.....	8
Database Implementation	8
Creating Tables	9
Inserting Data.....	11
Querying and Manipulation	127
Testing	19
Which materials are currently available in the library?	19
Which materials are currently overdue?	19
What are the top 10 most borrowed materials in the library?	20
How many books has the author Lucas Piki written?	20
How many books were written by two or more authors?	210
What are the most popular genres in the library?	22
How many materials have been borrowed from 09/2020-10/2020?	22
How do you update the “Harry Potter and the Philosopher's Stone”?	23
How do you delete the member Emily Miller and all her related records from the database?	23
How do you add new material to the database?	23
Conclusion.....	25

Introduction

Through this semester, in CS 504 Principles of Data Management and Mining, topics like database design, implementation, and querying were covered. This project represents a practical application of these common database concepts. It is based on a real-life problem – building and maintaining a functional database management system for a public library, and it includes several important features, such as: resource management (resources consist of books, magazines, digital media, etc), members and staff management, together with tracking of borrowing and returning the resources. In terms of project goals, I will aim to minimize redundancy, ensure data integrity, and provide efficient access (with possible adding, updating, and deleting) to all the necessary database fields.

Project Overview

As mentioned above, the system in this project should help the library staff manage their printed and digital resources. Additionally, it should provide efficient access to information of the library members and facilitate borrowing and tracking of the library materials. Similarly, all available materials should be accessible at the member level. For example, all registered members should be able to browse the media catalog in detail, and easily make a choice about what to borrow next.



Project Scenario

The following list describes all required features for this project. In the next section, those features will be implemented through database entities and connected accordingly. Tables with attributes and keys will be added there. Every table will have its own primary key and foreign keys will be used for connection between entities (including relationships such as one-to-one or one-to-many).

1. Materials Management

The system should store and maintain information about all library materials, with their titles, authors, publication dates, and genres.

2. Membership Management

The system should store and manage information about library members, including their names, contact information, membership numbers, and borrowing history.

3. Borrowing

The system should allow members to pick, borrow, and return items, also to provide library staff with the necessary information to handle that complete borrow-return process. Once material is checked out, a librarian should record the borrow date, with anticipated due date. And once the material is returned, its return date should be updated.

4. Reporting and Analytics

The system should generate reports on library usage, popular materials, including other relevant statistics, enabling the library staff to make data-driven decisions for the future.

Requirements

Apart from basic knowledge on how the library system works in practice, for this assignment we need good database software, with tools that are optimized for efficient problem solving and a good user-friendly environment.

My choice is **MySQL Workbench**, which stands for a unified visual tool for database architects, developers, and DBAs. It is available for Windows, Linux, and Mac OS. It also provides data modeling, followed by comprehensive administration tools for server configuration, user administration, backup, and much more.

Using that software, I will create tables, fill them with given data, and perform a set of select, count, sum, and join queries. In the end, I will export .sql source file which contains codes for all these operations and it can be imported and executed on other computers (using Workbench again).

Database Design

Entities and Relationships

The database schema for this project contains eight entities (tables):

1. **Catalog**

This table contains information about the different catalogs in the library. Catalogs are useful in resource collection and organization. The primary key is numeric ID, there is also an attribute for the name, followed by an attribute for the location. To ensure the data integrity and consistency, I've added all three of them as mandatory (not null) fields.

2. **Genre**

This table contains information about genres of the resources in the library. Each record has a numerical primary key and two mandatory (not null) fields for the genre name and the genre description. Description is created as a longer text field (up to 255 characters).

3. **Material**

This table contains information about the actual material in the library (printed and digital books, newspaper, etc). There's a numerical material ID, followed by the title (mandatory textual field) publication date (mandatory textual field), and foreign keys for catalog and genre ID. Each material has information about the catalog and the genre too.

4. **Author**

This table contains information about the author of some material. There's a numerical author ID, author's name (mandatory textual field), birth date (mandatory date field), and nationality (mandatory text field). Overall, this table is helpful in terms of organizing and managing information about the different authors of the different materials in the library.

5. **Authorship**

This table represents a connection between authors and materials. It contains Authorship ID for the primary key, then Author and Material ID for the foreign keys. Many authors can write and publish many materials, that is the main purpose of this "helper" table.

6. **LMember**

This table contains information about the library members. They have a numerical ID, name (mandatory textual field), contact info (mandatory text field), and the date they joined the library (mandatory date field). Members are used later in the "borrowing procedure".

7. Staff

This table contains information about the different staff members of the library. They have a numerical ID, name (mandatory text field), contact info (mandatory text field), job title (mandatory text field), and hire date (mandatory date field).

8. Borrow

This table is considered a very important component of the library database, which holds all the details about the borrowing and returning of the resources. It has a numerical primary key (ID), borrow date (mandatory date field), due date (mandatory date field), return date (non-mandatory date field, some resources might not be returned yet), and there are foreign keys for material ID, member ID, and staff ID. Similar to previously mentioned foreign keys, here also I used the **on delete set null and on update cascade options**, so even if a related record is deleted or updated, these relationships will remain intact.

These entities are connected using the following relationships:

1. Table Material (**Material – Catalog, Material – Genre**)

- Foreign key **Catalog_ID** references the primary key in table **Catalog**

Relationship type: **One-to-One** (Each material is associated with one and only catalog, and each catalog can only be associated with one material).

- Foreign key **Genre_ID** references the primary key in table **Genre**

Relationship type: **Many-to-One** (Many materials can be associated with the same genre, but each material has only one genre).

2. Table Authorship (**Authorship – Author, Authorship – Material**)

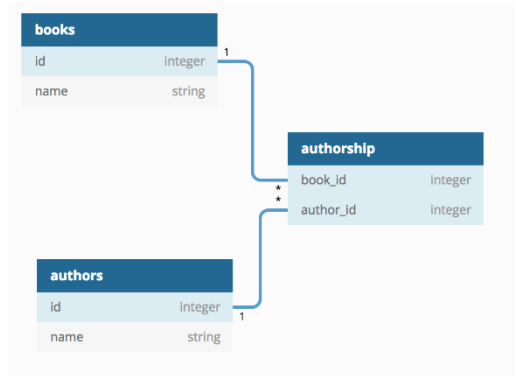
- Foreign key **Author_ID** references the primary key in table **Author**

Relationship type: **Many-to-One** (Many authorships can be associated with the same author, but each authorship has only one author).

- Foreign key **Material_ID** references the primary key in table **Material**

Relationship type: **Many-to-One** (Many authorships can be associated with the same material, but each authorship has only one material).

These two relationships are part of a **Many-to-Many** relationship, because many authors can write and publish many materials. On the next page you can see a sample image found on the internet (src: <https://community.dbdiagram.io/t/many-to-many-relationships/65>), which represents a situation very similar to this one.



3. Table Borrow (Borrow – Material, Borrow – Staff, Borrow – LMember)

- Foreign key **Material_ID** references the primary key in table **Material**

Relationship type: **Many-to-One** (Many borrowing records can be for the same material, but each borrowing record includes only one material).

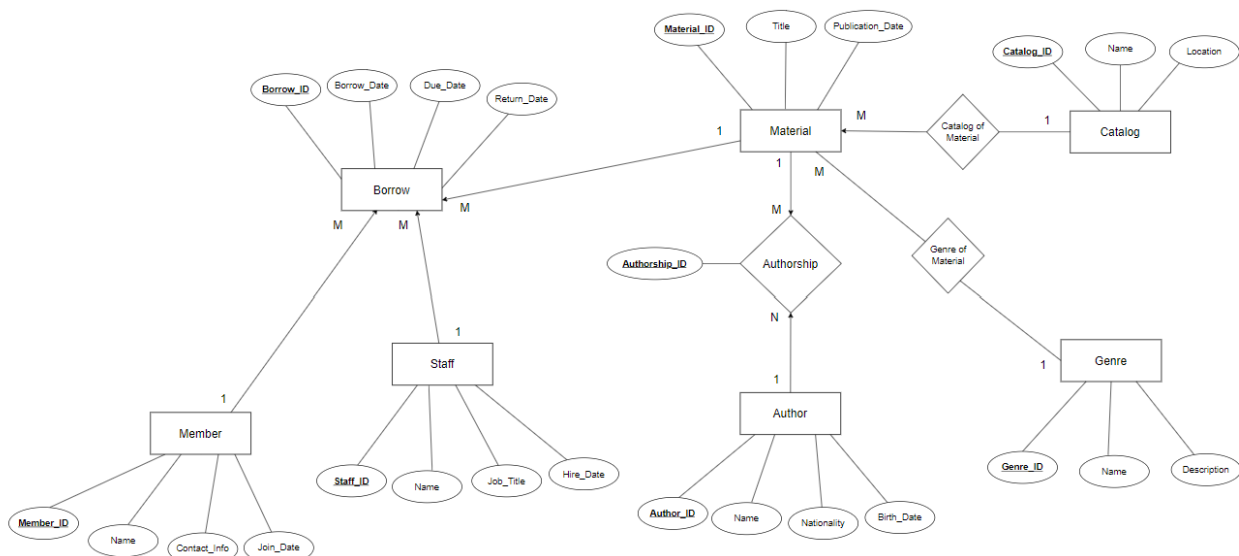
- Foreign key **Staff_ID** references the primary key in table **Staff**

Relationship type: **Many-to-One** (Many borrowing records can be processed by the same staff member, but each borrowing record has only one staff member assigned).

- Foreign key **Member_ID** references the primary key in table **LMember**

Relationship type: **Many-to-One** (Many borrowing records can belong to the same member, but each borrowing record is made by one library member only).

Entity-Relationship (ER) Diagram



Database Normalization

- **First Normal Form (1 NF)**

Each attribute must contain indivisible values to satisfy the first normal form. Likewise, there should be no repeating groups or arrays. Based on the project outline and description of the requirements, the relational schema is already in 1 NF.

- **Second Normal Form (2 NF)**

Before satisfying the 2 NF, the schema must be in 1 NF first. Any non-key attribute should be fully dependent on the primary key, which is the case with our assigned entities.

- **Third Normal Form (3 NF)**

Before satisfying the 3 NF, the schema must be in 1 NF and 2 NF. Any non-key attribute should be dependent only on the primary key, and not on any other non-key attribute.

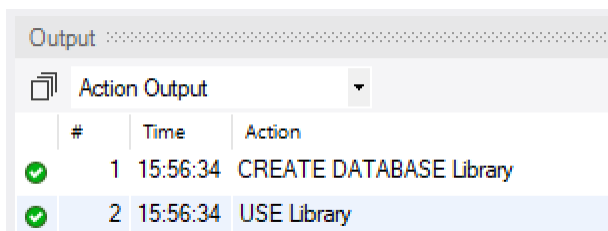
Database Implementation

As mentioned earlier, this database system is implemented using MySQL Workbench software, where the ER Diagram was also drawn. The program environment is user-friendly, and you can easily type and execute SQL codes to create tables, insert data into tables, and later perform various queries. This section contains screenshots from the program environment with descriptions and the trend continues to the next section, where more complex CRUD* queries are implemented.

In MySQL Workbench, we can type and execute SQL statements directly in the Query window. Here's how to create a new database and set it for further use. That means all upcoming changes and SQL codes will be applied to the Library database and tables that we are about to create now.

```
CREATE DATABASE Library;
```

```
USE Library;
```



The screenshot shows the 'Output' window in MySQL Workbench. It has a tab labeled 'Action Output'. Below the tab is a table with four columns: a green checkmark icon, an ID number, a timestamp, and the executed SQL command. Two rows are visible, both indicating successful execution.

	#	Time	Action
✓	1	15:56:34	CREATE DATABASE Library
✓	2	15:56:34	USE Library

* CRUD - create, read, update, and delete

Creating Tables

Table Genre for the genre of the material, apart from usual attributes, includes a primary key ID.

```
CREATE TABLE Genre (  
Genre_ID INT NOT NULL PRIMARY KEY,  
GName VARCHAR(50) NOT NULL,  
GDescription VARCHAR(255) NOT NULL  
);
```

Table Catalog where materials belong to, apart from usual attributes, includes a primary key ID.

```
CREATE TABLE Catalog (  
Catalog_ID INT NOT NULL PRIMARY KEY,  
CName VARCHAR(50) NOT NULL,  
Location VARCHAR(50) NOT NULL  
);
```

Table Author for the author of the material, apart from usual attributes, includes a primary key ID.

```
CREATE TABLE Author (  
Author_ID INT NOT NULL PRIMARY KEY,  
AName VARCHAR(50) NOT NULL,  
Birth_Date DATE,  
Nationality VARCHAR(50)  
);
```

Table LMember for the library members, apart from usual attributes, includes a primary key ID.

```
CREATE TABLE LMember (  
Member_ID INT NOT NULL PRIMARY KEY,  
MName VARCHAR(50) NOT NULL,  
Contact_Info VARCHAR(50) NOT NULL,  
Join_Date DATE NOT NULL  
);
```

Table Staff for the employees of the library, apart from usual attributes, includes a primary key ID.

```
CREATE TABLE Staff (  
  Staff_ID INT NOT NULL PRIMARY KEY,  
  SName VARCHAR(50) NOT NULL,  
  Contact_Info VARCHAR(50) NOT NULL,  
  Job_Title VARCHAR(50) NOT NULL,  
  Hire_Date DATE NOT NULL  
);
```

Table Material for the actual resources in the library. It has a primary key ID, usual attributes such as title, date, etc, but it also has foreign keys for Catalog and Genre. These fields reference their primary keys in the suitable tables. The important settings defined on creation are “ON DELETE SET NULL ON UPDATE CASCADE” so the data integrity will be maintained too. For example, we can add a Sci-Fi audio book from some catalog to this table and harmlessly delete it later.

```
CREATE TABLE Material (  
  Material_ID INT NOT NULL PRIMARY KEY,  
  Title VARCHAR(128) NOT NULL,  
  Publication_Date DATE NOT NULL,  
  Catalog_ID INT,  
  Genre_ID INT,  
  FOREIGN KEY(Catalog_ID) REFERENCES Catalog(Catalog_ID) ON DELETE SET NULL ON UPDATE CASCADE,  
  FOREIGN KEY(Genre_ID) REFERENCES Genre(Genre_ID) ON DELETE SET NULL ON UPDATE CASCADE  
);
```

Table Authorship used as a “helper” table for Authors and Materials. It has a primary key ID and two foreign keys for Author and Material that reference their primary keys in the suitable tables. The settings on creation are the same as in previous table.

```
CREATE TABLE Authorship (  
  Authorship_ID INT NOT NULL PRIMARY KEY,  
  Author_ID INT,  
  Material_ID INT,  
  FOREIGN KEY(Author_ID) REFERENCES Author(Author_ID) ON DELETE SET NULL ON UPDATE CASCADE,  
  FOREIGN KEY(Material_ID) REFERENCES Material(Material_ID) ON DELETE SET NULL ON UPDATE CASCADE  
);
```

Table Borrow which holds information about borrowing and returning materials. It has a primary key, regular fields for dates (borrow date, due data, return date), and then foreign keys for Material, Library Member, and Staff Member. These foreign keys reference their primary keys in the suitable tables. The settings on creation are the same as in previous table.

```
CREATE TABLE Borrow (  
  Borrow_ID INT NOT NULL PRIMARY KEY,  
  Material_ID INT,  
  Member_ID INT,  
  Staff_ID INT,  
  Borrow_Date DATE NOT NULL,  
  Due_Date DATE NOT NULL,  
  Return_Date DATE,  
  FOREIGN KEY(Material_ID) REFERENCES Material(Material_ID) ON DELETE SET NULL ON UPDATE CASCADE,  
  FOREIGN KEY(Member_ID) REFERENCES LMember(Member_ID) ON DELETE SET NULL ON UPDATE CASCADE,  
  FOREIGN KEY(Staff_ID) REFERENCES Staff(Staff_ID) ON DELETE SET NULL ON UPDATE CASCADE  
);
```

Inserting Data

Inserting data into database tables represents a crucial part of database management. When adding new database records, it is important that the data matches the database schema, and all required fields are included (no attributes are missed or written in a format different than the column type).

Together with project prompt and instructions, I received some sample data from my professor. I used that data in building and testing this assignment. There are around 20 library members, 4 staff members, 20 authors, 30 resources, 8 different genres, 10 catalogues, and 40 borrow records.

SQL INSERT statement is used for inserting new records in all tables. This statement contains the name of the table, followed by the column names, and then there are lines with the actual values. Foreign keys are also added carefully because they must exist in the referenced table, and they must match the data type with the connected attribute (primary key) in the original table.

The following code snippets show the process of inserting sample data into all tables, fetched from Workbench. After that, a series of SQL **SELECT** queries can be executed to ensure that all necessary records were added successfully. If you get to open the exported .sql file with codes for all create and insert queries, you can simply execute that in Workbench and have a whole database created, with all these records inserted. Of course, you can always your own records, just make sure to follow the types and length of the fields, and don't forget about the relationships and keys.

```
INSERT INTO Author(Author_ID, AName, Birth_Date, Nationality)
```

```
VALUES
```

```
(1, "Jane Austen", "1775-12-16", "British"),  
(2, "Ernest Hemingway", "1899-07-21", "American"),  
(3, "George Orwell", "1903-06-25", "British"),  
(4, "Scott Fitzgerald", "1896-09-24", "American"),  
(5, "J.K. Rowling", "1965-07-31", "British"),  
(6, "Mark Twain", "1835-11-30", "American"),  
(7, "Leo Tolstoy", "1828-09-09", "Russian"),  
(8, "Virginia Woolf", "1882-01-25", "British"),  
(9, "Gabriel Márquez", "1927-03-06", "Colombian"),  
(10, "Charles Dickens", "1812-02-07", "British"),  
(11, "Harper Lee", "1926-04-28", "American"),  
(12, "Oscar Wilde", "1854-10-16", "Irish"),  
(13, "William Shakespeare", "1564-04-26", "British"),  
(14, "Franz Kafka", "1883-07-03", "Czech"),  
(15, "James Joyce", "1882-02-02", "Irish"),  
(16, "J.R.R. Tolkien", "1892-01-03", "British"),  
(17, "Emily Brontë", "1818-07-30", "British"),  
(18, "Toni Morrison", "1931-02-18", "American"),  
(19, "Fyodor Dostoevsky", "1821-11-11", "Russian"),  
(20, "Lucas Piki", "1847-10-16", "British");
```

```
INSERT INTO Catalog(Catalog_ID, CName, Location)
```

```
VALUES
```

```
(1, "Books", "A1.1"),  
(2, "Magazines", "B2.1"),  
(3, "E-Books", "C3.1"),  
(4, "Audiobooks", "D4.1"),  
(5, "Journals", "E5.1"),
```

```
(6, "Newspaper", "F6.1"),
(7, "Maps", "G7.1"),
(8, "Novels", "H8.1"),
(9, "Sheet Music", "I9.1"),
(10, "Educational", "J10.1");
```

```
INSERT INTO Genre(Genre_ID, GName, GDescription)
```

```
VALUES
```

```
(1, "General Fiction", "Literary works with a focus on character and plot development,
exploring various themes and human experiences."),
(2, "Mystery & Thriller", "Suspenseful stories centered around crime, investigation,
orespionage with an emphasis on tension and excitement."),
(3, "Science Fiction & Fantasy", "Imaginative works that explore alternate realities,
futuristic concepts, and magical or supernatural elements."),
(4, "Horror & Suspense", "Stories designed to evoke fear, unease, or dread, often
featuring supernatural or psychological elements."),
(5, "Dystopian & Apocalyptic", "Depictions of societies in decline or collapse, often
exploring themes of political and social oppression or environmental disaster."),
(6, "Classics", "Enduring works of literature that have stood the test of time, often
featuring rich language and complex themes."),
(7, "Historical Fiction", "Fictional stories set in the past, often based on real
historical events or figures, and exploring the customs and experiences of that time."),
(8, "Epic Poetry & Mythology", "Ancient or traditional stories and poems, often
featuring heroes, gods, and mythical creatures, and exploring cultural values and
beliefs.");
```

```
INSERT INTO Staff(Staff_ID, SName, Contact_Info, Job_Title, Hire_Date)
```

```
VALUES
```

```
(1, "Amy Green ", "amy.green@email.com ", "Librarian ", "2017-06-01"),
(2, "Brian Taylor", "brian.taylor@email.com", "Library Assistant", "2018-11-15"),
(3, "Christine King", "chris.king@email.com", "Library Assistant", "2019-05-20"),
(4, "Daniel Wright", "dan.wright@email.com", "Library Technician", "2020-02-01");
```

```

INSERT INTO Material(Material_ID, Title, Publication_Date, Catalog_ID, Genre_ID)
VALUES
(1, "The Catcher in the Rye", "1951-07-16", 1, 1),
(2, "To Kill a Mockingbird", "1960-07-11", 2, 1),
(3, "The Da Vinci Code", "2003-04-01", 3, 2),
(4, "The Hobbit", "1937-09-21", 4, 3),
(5, "The Shining", "1977-01-28", 5, 4),
(6, "Pride and Prejudice", "1813-01-28", 1, 1),
(7, "The Great Gatsby", "1925-04-10", 2, 1),
(8, "Moby Dick", "1851-10-18", 3, 1),
(9, "Crime and Punishment", "1866-01-01", 4, 1),
(10, "The Hitchhiker's Guide to the Galaxy", "1979-10-12", 5, 3),
(11, "1984", "1949-06-08", 1, 5),
(12, "Animal Farm", "1945-08-17", 2, 5),
(13, "The Haunting of Hill House", "1959-10-17", 3, 4),
(14, "Brave New World", "1932-08-01", 4, 5),
(15, "The Chronicles of Narnia: The Lion, the Witch and the Wardrobe", "1950-10-16",
5, 3),
(16, "The Adventures of Huckleberry Finn", "1884-12-10", 6, 1),
(17, "The Catch-22", "1961-10-11", 7, 1),
(18, "The Picture of Dorian Gray", "1890-07-01", 8, 1),
(19, "The Call of Cthulhu", "1928-02-01", 9, 4),
(20, "Harry Potter and the Philosopher's Stone", "1997-06-26", 10, 3),
(21, "Frankenstein", "1818-01-01", 6, 4),
(22, "A Tale of Two Cities", "1859-04-30", 7, 1),
(23, "The Iliad", "1750-01-01", 8, 6),
(24, "The Odyssey", "1725-01-01", 9, 6),
(25, "The Brothers Karamazov", "1880-01-01", 10, 1),
(26, "The Divine Comedy", "1320-01-01", 6, 6),
(27, "The Grapes of Wrath", "1939-04-14", 7, 1),
(28, "The Old Man and the Sea", "1952-09-01", 8, 1),

```

```
(29, "The Count of Monte Cristo", "1844-01-01", 9, 1),
(30, "A Midsummer Night's Dream", "1596-01-01", 10, 7),
(31, "The Tricky Book", "1888-01-01", 10, 7);
```

```
INSERT INTO Authorship(Authorship_ID, Author_ID, Material_ID)
```

```
VALUES
```

```
(1, 1, 1), (2, 2, 2),
(3, 3, 3), (4, 4, 4),
(5, 5, 5), (6, 6, 6),
(7, 7, 7), (8, 8, 8),
(9, 9, 9), (10, 10, 10),
(11, 11, 11), (12, 12, 12),
(13, 13, 13), (14, 14, 14),
(15, 15, 15), (16, 16, 16),
(17, 17, 17), (18, 18, 18),
(19, 19, 19), (20, 20, 20),
(21, 1, 21), (22, 2, 22),
(23, 3, 23),
(24, 4, 24), (25, 5, 25),
(26, 6, 26), (27, 7, 27),
(28, 8, 28), (29, 19, 28),
(30, 9, 29), (31, 10, 30),
(32, 8, 30), (33, 2, 29);
```

```
INSERT INTO Borrow(Borrow_ID, Material_ID, Member_ID, Staff_ID, Borrow_Date,
Due_Date, Return_Date)
```

```
VALUES
```

```
(1, 1, 1, 1, "2018-09-12", "2018-10-03", "2018-09-30"),
(2, 2, 2, 1, "2018-10-15", "2018-11-05", "2018-10-29"),
(3, 3, 3, 1, "2018-12-20", "2019-01-10", "2019-01-08"),
(4, 4, 4, 1, "2019-03-11", "2019-04-01", "2019-03-27"),
```

(5, 5, 5, 1, "2019-04-20", "2019-05-11", "2019-05-05"),
(6, 6, 6, 1, "2019-07-05", "2019-07-26", "2019-07-21"),
(7, 7, 7, 1, "2019-09-10", "2019-10-01", "2019-09-25"),
(8, 8, 8, 1, "2019-11-08", "2019-11-29", "2019-11-20"),
(9, 9, 9, 1, "2020-01-15", "2020-02-05", "2020-02-03"),
(10, 10, 10, 1, "2020-03-12", "2020-04-02", "2020-03-28"),
(11, 1, 11, 2, "2020-05-14", "2020-06-04", "2020-05-28"),
(12, 2, 12, 2, "2020-07-21", "2020-08-11", "2020-08-02"),
(13, 3, 13, 2, "2020-09-25", "2020-10-16", "2020-10-15"),
(14, 4, 1, 2, "2020-11-08", "2020-11-29", "2020-11-24"),
(15, 5, 2, 2, "2021-01-03", "2021-01-24", "2021-01-19"),
(16, 6, 3, 2, "2021-02-18", "2021-03-11", "2021-03-12"),
(17, 17, 4, 2, "2021-04-27", "2021-05-18", "2021-05-20"),
(18, 18, 5, 2, "2021-06-13", "2021-07-04", "2021-06-28"),
(19, 19, 6, 2, "2021-08-15", "2021-09-05", "2021-09-03"),
(20, 20, 7, 2, "2021-10-21", "2021-11-11", "2021-11-05"),
(21, 21, 1, 3, "2021-11-29", "2021-12-20", null),
(22, 22, 2, 3, "2022-01-10", "2022-01-31", "2022-01-25"),
(23, 23, 3, 3, "2022-02-07", "2022-02-28", "2022-02-23"),
(24, 24, 4, 3, "2022-03-11", "2022-04-01", "2022-03-28"),
(25, 25, 5, 3, "2022-04-28", "2022-05-19", "2022-05-18"),
(26, 26, 6, 3, "2022-06-22", "2022-07-13", "2022-07-08"),
(27, 27, 7, 3, "2022-08-04", "2022-08-25", "2022-08-23"),
(28, 28, 8, 3, "2022-09-13", "2022-10-04", "2022-09-28"),
(29, 29, 9, 3, "2022-10-16", "2022-11-06", "2022-11-05"),
(30, 30, 8, 3, "2022-11-21", "2022-12-12", "2022-12-05"),
(31, 1, 9, 4, "2022-12-28", "2023-01-18", null),
(32, 2, 1, 4, "2023-01-23", "2023-02-13", null),
(33, 3, 10, 4, "2023-02-02", "2023-02-23", "2023-02-17"),
(34, 4, 11, 4, "2023-03-01", "2023-03-22", null),


```
(35, 5, 12, 4, "2023-03-10", "2023-03-31", null),
(36, 6, 13, 4, "2023-03-15", "2023-04-05", null),
(37, 7, 17, 4, "2023-03-25", "2023-04-15", null),
(38, 8, 8, 4, "2023-03-30", "2023-04-20", null),
(39, 9, 9, 4, "2023-03-26", "2023-04-16", null),
(40, 10, 20, 4, "2023-03-28", "2023-04-18", null);
```

```
INSERT INTO LMember(Member_ID, MName, Contact_Info, Join_Date)
```

```
VALUES
```

```
(1, "Alice Johnson", "alice.johnson@email.com", "2018-01-10"),
(2, "Bob Smith", "bob.smith@email.com", "2018-03-15"),
(3, "Carol Brown", "carol.brown@email.com", "2018-06-20"),
(4, "David Williams", "david.williams@email.com", "2018-09-18"),
(5, "Emily Miller", "emily.miller@email.com", "2019-02-12"),
(6, "Frank Davis", "frank.davis@email.com", "2019-05-25"),
(7, "Grace Wilson", "grace.wilson@email.com", "2019-08-15"),
(8, "Harry Garcia", "harry.garcia@email.com", "2019-11-27"),
(9, "Isla Thomas", "isla.thomas@email.com", "2020-03-04"),
(10, "Jack Martinez", "jack.martinez@email.com", "2020-07-01"),
(11, "Kate Anderson", "kate.anderson@email.com", "2020-09-30"),
(12, "Luke Jackson", "luke.jackson@email.com", "2021-01-18"),
(13, "Mia White", "mia.white@email.com", "2021-04-27"),
(14, "Noah Harris", "noah.harris@email.com", "2021-07-13"),
(15, "Olivia Clark", "olivia.clark@email.com", "2021-10-05"),
(16, "Peter Lewis", "peter.lewis@email.com", "2021-12-01"),
(17, "Quinn Hall", "quinn.hall@email.com", "2022-02-28"),
(18, "Rachel Young", "rachel.young@email.com", "2022-06-17"),
(19, "Sam Walker", "sam.walker@email.com", "2022-09-25"),
(20, "Tiffany Allen", "tiffany.allen@email.com", "2022-12-10");
```

Querying and Manipulation

Basic queries for checking the contents of tables after adding some records...

List some recent borrowings with member and material names from March 2023

```
SELECT Borrow_ID, Borrow_Date, Due_Date, LMember.MName AS "Member_Name",  
Material.Title AS "Material_Title"  
  
FROM Borrow  
  
JOIN LMember ON LMember.Member_ID = Borrow.Member_ID  
  
JOIN Material ON Material.Material_ID = Borrow.Material_ID  
  
WHERE Borrow_Date > "2023-03-01"  
  
ORDER BY Borrow_Date
```

	Borrow_ID	Borrow_Date	Due_Date	Member_Name	Material_Title
▶	35	2023-03-10	2023-03-31	Luke Jackson	The Shining
	36	2023-03-15	2023-04-05	Mia White	Pride and Prejudice
	37	2023-03-25	2023-04-15	Quinn Hall	The Great Gatsby
	39	2023-03-26	2023-04-16	Isla Thomas	Crime and Punishment
	40	2023-03-28	2023-04-18	Tiffany Allen	The Hitchhiker's Guide to the Galaxy
	38	2023-03-30	2023-04-20	Harry Garcia	Moby Dick

Two JOIN statements, borrow date criteria in WHERE statement, ordering done by borrow date.

List all British or American authors from 19th century

```
SELECT * FROM Author  
  
WHERE Birth_Date > "1800-01-01" AND Birth_Date < "1899-12-31"  
  
AND Nationality IN ("British", "American")  
  
ORDER BY Birth_Date
```

	Author_ID	AName	Birth_Date	Nationality
▶	10	Charles Dickens	1812-02-07	British
	17	Emily Brontë	1818-07-30	British
	6	Mark Twain	1835-11-30	American
	20	Luas Piki	1847-10-16	British
	8	Virginia Woolf	1882-01-25	British
	16	J.R.R. Tolkien	1892-01-03	British
	4	Scott Fitzgerald	1896-09-24	American
	2	Ernest Hemingway	1899-07-21	American
*	NULL	NULL	NULL	NULL

Birthdate range and Nationality set written as criteria in WHERE statement, ordering by birthdate.

Testing

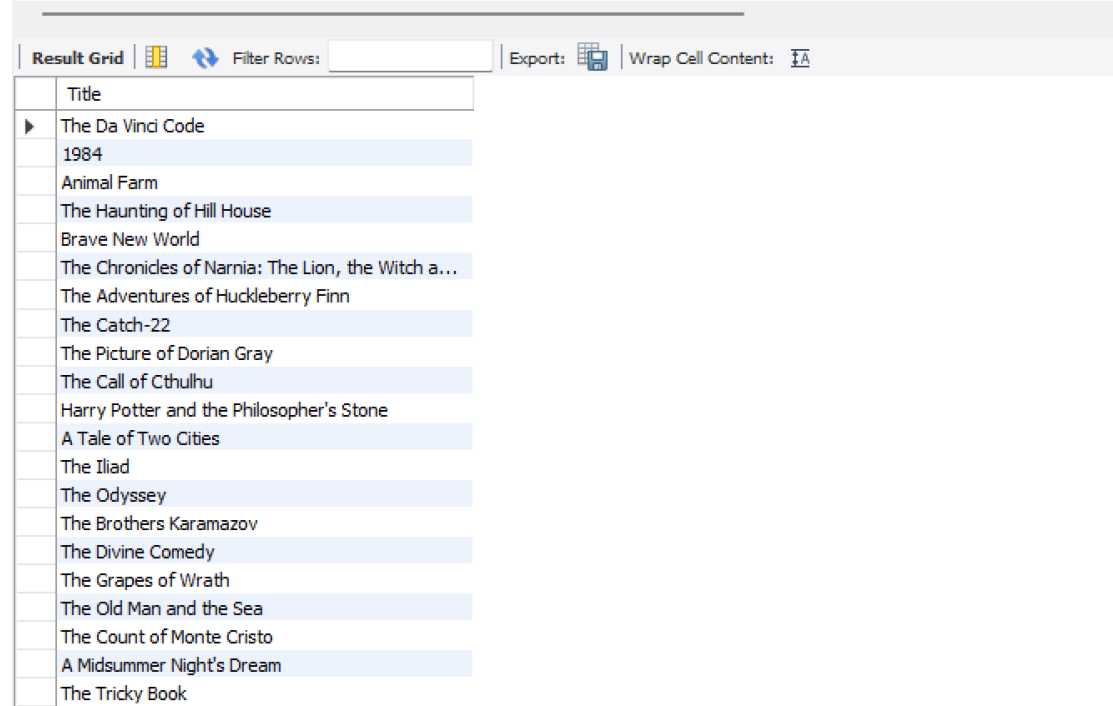
These ten queries were implemented to demonstrate the use of different querying techniques, such as joins, aggregation, and subqueries.

For each query, a screenshot from MySQL Workbench is attached, together with the code or a suitable description where it is necessary.

After that, some suggestions for database extension are mentioned and described in short terms.

Which materials are currently available in the library?

```
254 • SELECT Title
255 FROM Material
256 WHERE Material_ID NOT IN(SELECT Material_ID FROM Borrow WHERE Return_Date IS NULL);
```



The screenshot displays the MySQL Workbench interface. At the top, the SQL query is entered in the editor. Below the editor, the 'Result Grid' tab is active, showing the results of the query. The results are presented as a table with two columns: 'Title' and an empty column. The table contains 20 rows of book titles.

Title	
The Da Vinci Code	
1984	
Animal Farm	
The Haunting of Hill House	
Brave New World	
The Chronicles of Narnia: The Lion, the Witch a...	
The Adventures of Huckleberry Finn	
The Catch-22	
The Picture of Dorian Gray	
The Call of Cthulhu	
Harry Potter and the Philosopher's Stone	
A Tale of Two Cities	
The Iliad	
The Odyssey	
The Brothers Karamazov	
The Divine Comedy	
The Grapes of Wrath	
The Old Man and the Sea	
The Count of Monte Cristo	
A Midsummer Night's Dream	
The Tricky Book	

Using SELECT statement, we are picking Titles from the Materials table, but in WHERE condition, we are using a subquery to filter out only those materials that are not currently rented. The statement known as NOT IN is helpful here, because we want to display titles of materials whose return date is not null (they are not currently under any rental).

Material_ID plays an important role here, first as a primary key in Material table, then as a foreign key in Borrow table.

Which materials are currently overdue?

Suppose today is 04/01/2023 and show the borrow date and due date of each material.

```
20 • SELECT Material.Title, Borrow.Borrow_Date, Borrow.Due_Date
21 FROM Material
22 JOIN Borrow ON Material.Material_ID = Borrow.Material_ID
23 WHERE Borrow.Due_Date < "2023-04-01" AND Borrow.Return_Date IS NULL;
24
```

	Title	Borrow_Date	Due_Date
▶	Frankenstein	2021-11-29	2021-12-20
	The Catcher in the Rye	2022-12-28	2023-01-18
	To Kill a Mockingbird	2023-01-23	2023-02-13
	The Hobbit	2023-03-01	2023-03-22
	The Shining	2023-03-10	2023-03-31

In this query, we are selecting suitable titles of materials, along with their borrow and due dates. Material title is selected from the Material table, while Borrow information is included using INNER JOIN (written as only JOIN in SQL). Two tables are connected using Material_ID field, because each borrow record contains information about the borrowed material. In terms of filtering, in WHERE statement, the due date should be less than April 1st, 2023, also the return date for that borrowing should be null. Additionally, these records can be ordered by dates or titles.

What are the top 10 most borrowed materials in the library?

```
263 • SELECT Material.Title, COUNT(*) N_Borrowed
264 FROM Material
265 JOIN Borrow ON Material.Material_ID = Borrow.Material_ID
266 GROUP BY Material.Material_ID
267 ORDER BY N_Borrowed DESC
268 LIMIT 10;
```

	Title	N_Borrowed
▶	The Catcher in the Rye	3
	To Kill a Mockingbird	3
	The Da Vinci Code	3
	The Hobbit	3
	The Shining	3
	Pride and Prejudice	3
	The Great Gatsby	2
	Moby Dick	2
	Crime and Punishment	2
	The Hitchhiker's Guide to the Galaxy	2

Using SELECT statement, titles of materials are included, together with the suitable counter of the rows. There's an INNER JOIN between Material and Borrow tables (again on Material_ID field), just in this case records are grouped by Material_ID and ordered by the counter (N_Borrowed) in descending order. As the very last statement, we are limiting the result set to the first 10 records only. That is how only the top 10 most borrowed materials are displayed in the attached screenshot.

How many books has the author Lucas Piki written?

```
SELECT COUNT(*) AS N_Books
FROM Authorship
JOIN Material ON Authorship.Material_ID = Material.Material_ID
JOIN Author ON Authorship.Author_ID = Author.Author_ID
JOIN Catalog ON Material.Catalog_ID = Catalog.Catalog_ID
WHERE Author.AName = "Lucas Piki" AND Catalog.CName = "book"
```

In this query, we are counting the number of rows that match the given criteria (materials are books and they are written by Lucas Piki). For condition about the author, first we need to connect Material and Authorship tables (Material_ID field is used in INNER JOIN). Then we need to join Authorship and Author tables using Author_ID field in INNER JOIN. After that, we can connect Catalog and Material using Catalog_ID in INNER JOIN, and then we can put the desired criteria under WHERE statement.

The current query result is 0, because there aren't any books from this author yet, but we might add some later.

Result Grid	
	N_Books
▶	0

How many books were written by two or more authors?

```
1 • SELECT Title
2 FROM Material
3 JOIN Authorship ON Material.Material_ID = Authorship.Material_ID
4 JOIN Catalog ON Material.Catalog_ID = Catalog.Catalog_ID
5 WHERE Catalog.CName = "Book"
6 GROUP BY Material.Title
7 HAVING COUNT(DISTINCT Authorship.Author_ID) >= 2;
```

In this query, we are selecting titles of books that are written by two or more authors. INNER JOIN is necessary between Authorship and Material (Material_ID is used in the connection), and between Catalog and Material (Catalog_ID is used in the connection), because we are looking for books only. Grouping is done by titles of materials where the result set is filtered out using the appropriate HAVING statement in the end. The count of distinct authors on these books should be at least two and that is put in the HAVING statement.

What are the most popular genres in the library?

```
270 • SELECT GName, COUNT(*) AS Count_Most
271 FROM Genre
272 JOIN Material ON Genre.Genre_ID = Material.Genre_ID
273 GROUP BY GName
274 ORDER BY Count_Most DESC
```

Result Grid			Filter Rows:	Export:	Wrap Cell Cont
	GName	Count_Most			
▶	General Fiction	14			
	Science Fiction & Fantasy	4			
	Horror & Suspense	4			
	Dystopian & Apocalyptic	3			
	Classics	3			
	Historical Fiction	2			
	Mystery & Thriller	1			

For this query, we are using SELECT and COUNT statements, where we are selecting genre names and counting their occurrences in the genre table. One INNER JOIN is necessary, to connect Material and Genre (Genre_ID is the key used in this relationship). Grouping is done by genre name and the ordering is done by counter of occurrences in descending order, so the most popular genre is displayed first, while the least popular is displayed last in the result set.

How many materials have been borrowed from 09/2020-10/2020?

```
25 • SELECT Title, COUNT(*) AS N_Borrowed
26 FROM Material
27 JOIN Borrow ON Material.Material_ID = Borrow.Material_ID
28 WHERE Borrow_Date >= "2020-09-01" AND Borrow_Date <= "2020-10-31"
29 GROUP BY Title
```

Result Grid			Filter Rows:	Export:	Wrap Cell Content:
	Title	N_Borrowed			
▶	The Da Vinci Code	1			

Like in previous query, here we are also using SELECT and COUNT statements, just we are counting the number of borrowed materials within some period now. One INNER JOIN is necessary, to connect Borrow and Material tables (Material_ID is the key used in this relationship). Grouping is done by material title and a specific date range (Sept 1, 2020 - Oct 31, 2020) is put in WHERE statement as a filtering condition. There is only one result that matches the criteria.

How do you update the “Harry Potter and the Philosopher's Stone”?

Based on assignment instructions, it is returned 04/01/2023.

```
UPDATE Borrow
SET Return_Date = "2023-04-01"
WHERE Material_ID IN (SELECT Material_ID FROM Material
                      WHERE Title = "Harry Potter and the Philosopher's Stone")
AND Return_Date IS NULL;
```

Using UPDATE statement we are setting the return date to the given April 1, 2023, but only for records that match the given criteria – the material must be with specific title (Harry Potter and the Philosopher;s Stone) and it must be borrowed already (return date is null). Since we need the material ID, we are using a subquery to select the material ID of this Harry Potter publication and after that we are also adding a condition for the date.

How do you delete the member Emily Miller and all her related records from the database?

```
DELETE FROM Borrow WHERE Member_ID =
(SELECT Member_ID FROM LMember WHERE MName = "Emily Miller");

DELETE FROM LMember WHERE MName = "Emily Miller";
```

Using DELETE statement, first we are removing all borrowing records of Emily Miller, and after that we are removing her from the table of members. That will ensure that all related records to Emily Miller are be removed from the database.

For the successful removal of borrowing records, we need a subquery that will select the member ID where the name is Emily Miller, so we can identify her and use her ID as the criteria for deleting from the Borrow table. After that, we are simply removing her from the list of library members.

How do you add new material to the database?

Title: New book Date: 2020-08-01

Catalog: E-Books

Genre: Mystery & Thriller

Author: Lucas Pipi

```

INSERT INTO Author(Author_ID, AName)
VALUES
(21, "Lucas Pipi");

INSERT INTO Material(Material_ID, Title, Publication_Date, Catalog_ID, Genre_ID)
VALUES
(32, "New book", "2020-08-01",
~ (SELECT Catalog_ID FROM Catalog WHERE CName = "E-Books"), (SELECT Genre_ID FROM Genre WHERE GName = "Mystery and Thriller"));

INSERT INTO Authorship(Authorship_ID, Material_ID, Author_ID)
VALUES
(34, (SELECT Material_ID FROM Material WHERE Title = "New Book"), (SELECT Author_ID FROM Author WHERE AName = "Lucas Pipi"));

```

To insert a new material into the database, where the author is also completely new (we don't have it in the database yet), we must add an author first, then add a material, then create a suitable record in the authorship table. Author has only ID and name, Material has ID, Title, Publication date, Catalog ID, (we must find an ID of the e-book, that is why we are performing a subquery with SELECT statement) and Genre ID (we must also find an ID of the Mystery and Thriller genre, that is why we are performing one more subquery with SELECT statement). In the end we insert a new record in the Authorship table, which consists of Authorship_ID, Material_ID and Author_ID.

Potential extension of the database

Alert staff about overdue materials on a daily-basis?

I find this a very useful feature, which we can implement through a database trigger. The trigger can automatically increase the value in the column for overdue occurrences that each member should have. So, once a member misses the due date and fails to return the material, a value in his column for overdue occurrences will be increased and staff members will be notified about this activity. They can email him or eventually call/text or put a penalty if this is not the first time that a specific member fails to return the material. There are many possibilities for this approach, but surely a database trigger would be handy to control the overdue materials and member behavior.

Automatically deactivate the membership based on the member's overdue occurrence (\geq three times). And reactivate the membership once the member pays the overdue fee.

As mentioned above, a penalty must be applied for members that are not returning materials on time. This is a good idea to deactivate their membership if values in overdue occurrence columns reach 3. Reactivating after paying the fee is also okay. We can add one more column to the Member table, which will be used for the status of the membership. For example, if there are 3 overdue rentals in a row, the trigger automatically sets the member status to "DEACTIVATED", and that member cannot log in or make new rentals at all. Once a payment is made (something around \$15 for 3 late rentals), the overdue occurrences get cleared (value is 0 in the table), so the trigger automatically changes the status to "ACTIVE" and that member can proceed with browsing, borrowing, and all other available activities in the library.

Conclusion

The given Library project significantly helped me with summarizing and better understanding of various database concepts that we previously covered in class. While I was working on the project, I got a chance to research a bit, test different approaches, analyze the given data, make conclusions, and individually work on something more complex than regular lab or homework assignments.

Defining relationships was really an interesting challenge, where I had to consider all aspects and find the best possible solution. I must admit it took me a while to determine all those 1:1 and 1:M relationships and make sure that the initial database idea won't be interrupted with my recently implemented entities.

I truly hope I will get a chance to work on a similar project again, maybe during some company internship, or through a freelance project.