CS504_003

Principles of Data Management and Mining

Final Project

Snehita Moturu (#G01388464)

Professor – Binqian Yin

George Mason University

30 April 2023

Contents

1 Database Design

1.1 Scope of the project

The aim of this project is to develop and deploy a database management system to aid in the organization and management of a public library. This system will help the library staff to manage their library resources, including books, magazines, digital media, and other materials. Moreover, it will provide a streamlined approach to accessing library member's information and facilitate the borrowing and tracking of library materials. The library management system will prioritize data integrity and aim to reduce redundancy for both the library staff and members. (*CS504 Final Project.docx*, n.d.)

1.2 Entities and Relationships

An entity is typically represented as a table in a relational database, with each row in the table representing a specific instance of the entity. For the given data, the entities along with their relationships are as follows:

1. **Material**

Represents individual items available in the library, such as books, magazines, e-books, and audiobooks.

Attributes:

- Material_ID: A unique identifier for each material.
- Title: The title of the material.
- Publication_Date: The date of publication of the material.
- Catalog_ID: A reference to the catalog entry for the material.
- Genre_ID: A reference to the genre of the material.

2. **Catalog**

Represents a record of library materials with information on their availability and location.

Attributes:

- Catalog_ID: A unique identifier for each catalog entry.
- Name: The name of the catalog.
- Location: The location of the material within the library.

3. **Genre**

Represents the various genres or categories of library materials.

Attributes:

- Genre_ID: A unique identifier for each genre.
- Name: The name of the genre.
- Description: The brief introduction of the genre.

4. **Borrow**
   Represents the borrowing activity of library materials by members.

   Attributes:

   - Borrow_ID: A unique identifier for each borrowing transaction.
   - Material_ID: A reference to the borrowed material.
   - Member_ID: A reference to the member who borrowed the material.
   - Staff_ID: A reference to the staff who processed the transaction.
   - Borrow_Date: The date the material was borrowed.
   - Due_Date: The date the material is due.
     - Return_Date: The date the material is returned.

5. **Author**

   Represents authors who have created library materials.

   Attributes:

   - Author_ID: A unique identifier for each author.
   - Name: The name of the author.
   - Birth_Date: The birth date of the author.
   - Nationality: The nationality of the author.

6. **Authorship**
   Represents the relationship between authors and the materials they have created.

   Attributes:

   - Authorship_ID: A unique identifier for each authorship record.
   - Author_ID: A reference to the author.
   - Material_ID: A reference to the material authored.

7. **Member**
   Represents library members who can borrow and reserve materials.

   Attributes:

   - Member_ID: A unique identifier for each member.
   - Name: The name of the member.
   - Contact_Info: Email address (or phone number) of the member.
   - Join_Date: The date the member joined the library.

8. **Staff**
   Represents library staff who manage library resources and assist members.

   Attributes:

   - Staff_ID: A unique identifier for each staff member.
   - Name: The name of the staff member.
   - Contact_Info: Email address (or phone number) of the member.
   - Job_Title: The job title of the staff member (e.g., librarian, assistant librarian).
   - Hire_Date: The date the staff member was hired by the library.

The following explains the relationships between each entity of the given schema;

- **Catalog to Material:** A catalog can have many materials, but a material can only belong to one catalog. This is a one-to-many relationship, shown as (1) on the Catalog side and (N) on the Material side.

- **Genre to Material:** A genre can have many materials, but a material can only have one genre. This is a one-to-many relationship, shown as (1) on the Genre side and (N) on the Material side.

- **Material to Authorship**: A material can have multiple Authorship and one an Authorship is associated with only one Material. This is a one-to-many relationship, shown as (1) in the Material side and (N) on the Authorship side.

- **Author to Authorship**: An author can own multiple Authorship, and an authorship can only have one author. This is a one-to-many relationship, shown as (1) on the Author side and (N) on the Authorship side.

- **Material to Borrow:** A material can be borrowed many times, but a borrow can only be associated with one material. This is a one-to-many relationship, shown as a line between Material and Borrow with an arrow pointing from Material to Borrow.

- **Member to Borrow:** A member can borrow many materials, but a material can only be borrowed by one member. This is a one-to-many relationship, shown as a line between Member and Borrow with an arrow pointing from Member to Borrow.

- **Staff to Borrow:** A staff member can handle many borrow transactions, but a borrow transaction can only be handled by one staff member. This is a one-to-many relationship, shown as a line between Staff and Borrow with an arrow pointing from Staff to Borrow.

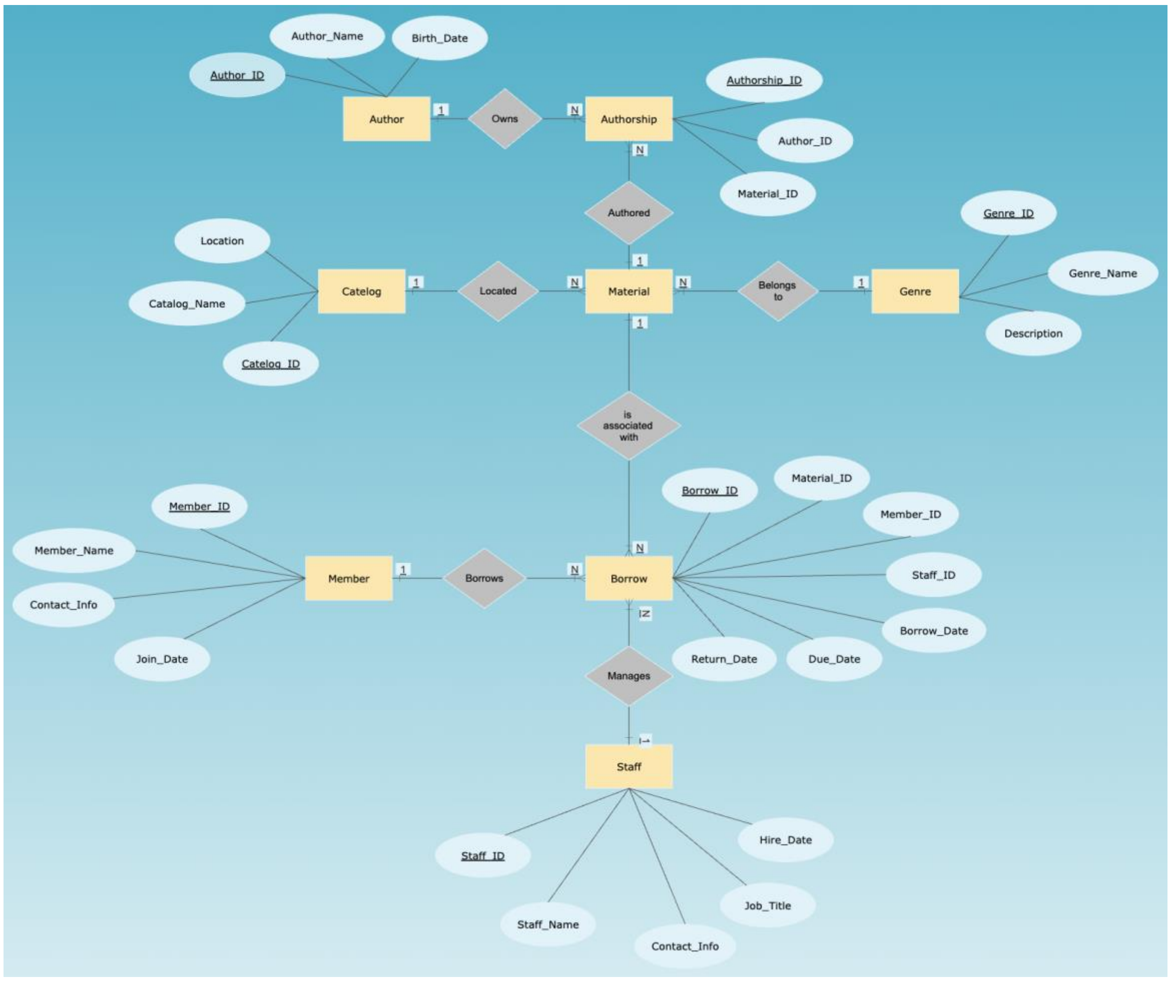


Fig1: Entity-Relationship (ER) diagram to represent the database schema.

1.3 Normalization

Normalization is a way to design a database that makes sure data is organized logically and reduces repetition of information. By breaking up large tables into smaller ones and linking them together, normalization prevents problems like data inconsistencies and errors that can occur when information is duplicated across multiple tables. Essentially, normalization ensures that data is stored efficiently and makes it easier to manage and maintain the database over time.

Normalization has not been applied to code, but the following could be a normalized version of the given schema that eliminates redundancy and ensures data integrity:

```
CREATE TABLE Catalog (
Catalog_ID INTEGER PRIMARY KEY,
Catalog_Name VARCHAR(30) NOT NULL,
Location VARCHAR(30) NOT NULL
);

CREATE TABLE Genre (
Genre_ID INTEGER PRIMARY KEY,
Genre_Name VARCHAR(300) NOT NULL,
Description VARCHAR(300)
);

CREATE TABLE Material (
Material_ID INTEGER PRIMARY KEY,
Title VARCHAR(300) NOT NULL,
Publication_Date DATE,
Catalog_ID INTEGER NOT NULL,
Genre_ID INTEGER NOT NULL,
CONSTRAINT CatalogMaterialFK FOREIGN KEY (Catalog_ID)
REFERENCES Catalog(Catalog_ID)
ON DELETE CASCADE,
CONSTRAINT GenreMaterialFK FOREIGN KEY (Genre_ID)
REFERENCES Genre(Genre_ID)
ON DELETE CASCADE
);

CREATE TABLE Contact (
Contact_ID INTEGER PRIMARY KEY,
Contact_Name VARCHAR(300) NOT NULL,
Contact_Info VARCHAR(300) NOT NULL
);

CREATE TABLE Member (
Member_ID INTEGER PRIMARY KEY,
Contact_ID INTEGER NOT NULL,
Join_Date DATE NOT NULL,
CONSTRAINT MemberContactFK FOREIGN KEY (Contact_ID)
REFERENCES Contact_Info(Contact_ID)
);

CREATE TABLE Staff (
Staff_ID INTEGER PRIMARY KEY,
Job_Title VARCHAR(300) NOT NULL,
Contact_ID INTEGER NOT NULL,
Hire_Date DATE NOT NULL,
CONSTRAINT StaffContactFK FOREIGN KEY (Contact _ID)
REFERENCES Contact_Info(Contact_ID)
);

CREATE TABLE Borrow (
Borrow_ID INTEGER PRIMARY KEY,
Material_ID INTEGER NOT NULL,
Member_ID INTEGER NOT NULL,
Staff_ID INTEGER NOT NULL,
Borrow_Date DATE NOT NULL,
Due_Date DATE NOT NULL,
Return_Date DATE DEFAULT NULL,
CONSTRAINT BorrowMaterialFK FOREIGN KEY (Material_ID)
REFERENCES Material(Material_ID)
```

```
ON DELETE CASCADE,
CONSTRAINT BorrowMemberFK FOREIGN KEY (Member_ID)
REFERENCES Member(Member_ID)
ON DELETE CASCADE,
CONSTRAINT BorrowStaffFK FOREIGN KEY (Staff_ID)
REFERENCES Staff(Staff_ID)
ON DELETE CASCADE
);

CREATE TABLE Author (
Author_ID INTEGER PRIMARY KEY,
Author_Name VARCHAR(300) NOT NULL,
Birth_Date DATE,
Nationality VARCHAR(300)
);

CREATE TABLE Authorship (
Authorship_ID INTEGER PRIMARY KEY,
Author_ID INTEGER NOT NULL,
Material_ID INTEGER NOT NULL,
CONSTRAINT AuthorAuthorshipFK FOREIGN KEY (Author_ID)
REFERENCES Author(Author_ID)
ON DELETE CASCADE
ON UPDATE CASCADE,
CONSTRAINT MaterialAuthorshipFK FOREIGN KEY (Material_ID)
REFERENCES Material(Material_ID)
ON DELETE CASCADE
);
```

In this normalized version, Member and Staff table has a common attribute Contact_Info in both the entities. Here a new entity 'Contact' can be created with attributes Contact_ID, Contact_Name and Contact_Info. This replaces the Contact_Info attribute in both the tables with Contact_ID attribute, which acts as a foreign key for both the tables. This helps reducing the data redundancy, if in case a staff is also a member in the same library.

It appears that the database schema is in third normal form (3NF). To elaborate, it can be seen that in the provided schema there are no repeated groups or duplicated data and that each table only includes atomic values. The Material table has foreign keys to both the Catalog and Genre tables, ensuring that they are fully functionally dependent on the primary key.

Overall, this level of normalization helps to ensure that the data in the database is consistent, accurate, and easy to maintain.

2. Database Implementation

For this project, I have chosen to use MySQL Workbench as the appropriate DBMS for the database implementation. MySQL Workbench is a powerful and user-friendly tool that allows for efficient design, development, and administration of MySQL databases. It provides a visual interface for creating database models, designing, and managing schemas, and performing database queries and maintenance tasks. Additionally, it offers advanced features such as reverse engineering of existing databases, data synchronization, and database backup and restore functionality.

2.1 Implementation of Database Schema using MySQL

Below displays the implementation of database and its respective sample output for the given sample data.

#DATABASE CREATION

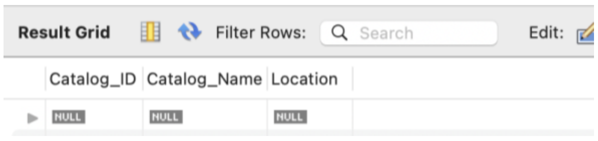
```
CREATE DATABASE CS_504_FINAL_PROJECT;
USE CS_504_FINAL_PROJECT;
```

#TABLE CREATION

```
CREATE TABLE Catalog (
   Catalog_ID INTEGER PRIMARY KEY,
   Catalog_Name VARCHAR(30) NOT NULL,
   Location VARCHAR(30) NOT NULL
);
```

Sample Output:

```
SELECT * FROM Catalog;
```

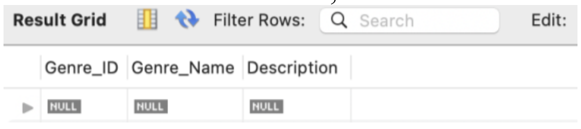


```
Catalog_ID  Catalog_Name  Location
NULL        NULL          NULL
```

```
CREATE TABLE Genre (
   Genre_ID INTEGER PRIMARY KEY,
   Genre_Name VARCHAR(300) NOT NULL,
   Description VARCHAR(300)
);
```

Sample Output:

```
SELECT * FROM Genre;
```

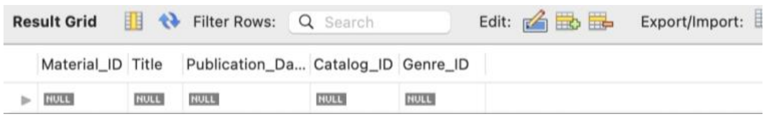


```
Genre_ID  Genre_Name  Description
NULL      NULL        NULL
```

```
CREATE TABLE Material (
   Material_ID INTEGER PRIMARY KEY,
   Title VARCHAR(300) NOT NULL,
   Publication_Date DATE,
   Catalog_ID INTEGER NOT NULL,
   Genre_ID INTEGER NOT NULL,
   CONSTRAINT CatalogMaterialFK FOREIGN KEY (Catalog_ID)
       REFERENCES Catalog(Catalog_ID)
       ON DELETE CASCADE,
   CONSTRAINT GenreMaterialFK FOREIGN KEY (Genre_ID)
       REFERENCES Genre(Genre_ID)
       ON DELETE CASCADE
);
```

Sample Output:

SELECT* FROM Material;

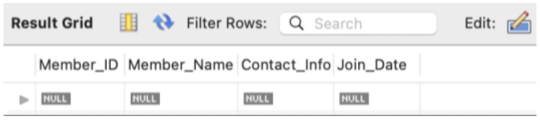


Material_ID	Title	Publication_Da...	Catalog_ID	Genre_ID
NULL	NULL	NULL	NULL	NULL

```
CREATE TABLE Member (
    Member_ID INTEGER PRIMARY KEY,
    Member_Name VARCHAR(300) NOT NULL,
    Contact_Info VARCHAR(300) NOT NULL,
    Join_Date DATE NOT NULL
);
```

Sample Output:

SELECT * FROM Member;

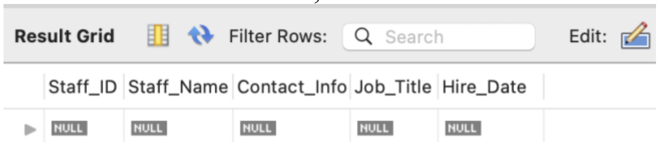


Member_ID	Member_Name	Contact_Info	Join_Date
NULL	NULL	NULL	NULL

```
CREATE TABLE Staff (
    Staff_ID INTEGER PRIMARY KEY,
    Staff_Name VARCHAR(300) NOT NULL,
    Contact_Info VARCHAR(300) NOT NULL,
    Job_Title VARCHAR(300) NOT NULL,
    Hire_Date DATE NOT NULL
);
```

Sample Output:

SELECT * FROM Staff;

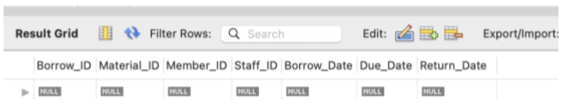


Staff_ID	Staff_Name	Contact_Info	Job_Title	Hire_Date
NULL	NULL	NULL	NULL	NULL

```
CREATE TABLE Borrow (
    Borrow_ID INTEGER PRIMARY KEY,
    Material_ID INTEGER NOT NULL,
    Member_ID INTEGER NOT NULL,
    Staff_ID INTEGER NOT NULL,
    Borrow_Date DATE NOT NULL,
    Due_Date DATE NOT NULL,
    Return_Date DATE DEFAULT NULL,
    CONSTRAINT StaffMaterialFK FOREIGN KEY (Material_ID)
        REFERENCES Material(Material_ID)
        ON DELETE CASCADE,
            CONSTRAINT StaffMemberFK FOREIGN KEY (Member_ID)
        REFERENCES Member(Member_ID)
        ON DELETE CASCADE,
            CONSTRAINT StaffBorrowFK FOREIGN KEY (Staff_ID)
        REFERENCES Staff(Staff_ID)
        ON DELETE CASCADE
);
```

Sample Output:

SELECT * FROM Borrow;

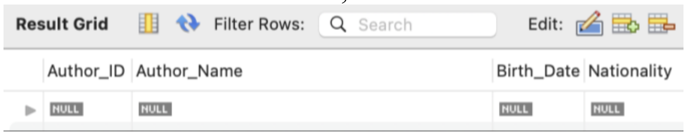


Borrow_ID	Material_ID	Member_ID	Staff_ID	Borrow_Date	Due_Date	Return_Date
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
CREATE TABLE Author (
    Author_ID INTEGER PRIMARY KEY,
    Author_Name VARCHAR(300) NOT NULL,
    Birth_Date DATE,
    Nationality VARCHAR(300)
);
```

Sample Output:

SELECT * FROM Author;

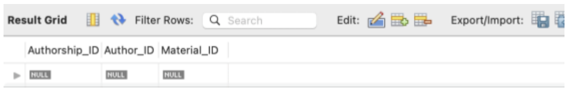


Author_ID	Author_Name	Birth_Date	Nationality
NULL	NULL	NULL	NULL

```
CREATE TABLE Authorship (
    Authorship_ID INTEGER PRIMARY KEY,
    Author_ID INTEGER NOT NULL,
    Material_ID INTEGER NOT NULL,
    CONSTRAINT AuthorAuthorshipFK FOREIGN KEY (Author_ID)
        REFERENCES Author(Author_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
            CONSTRAINT MaterialAuthorshipFK FOREIGN KEY (Material_ID)
        REFERENCES Material(Material_ID)
        ON DELETE CASCADE
);
```

Sample Output:

SELECT * FROM Authorship;


Result Grid   Filter Rows: Edit: Export/Import:

```
Authorship_ID Author_ID Material_ID
NULL          NULL      NULL
```

#Interpretation:

The ON UPDATE CASCADE and ON DELETE CASCADE are used here to make sure that the Authorship table is connected to the Author and Material tables in a proper way. ON DELETE CASCADE means that if a record is deleted from the Author or Material table, any records in the Authorship table that reference that record will also be deleted. Whereas, ON UPDATE CASCADE means that if the primary key of a record in the Author or Material table is updated, the corresponding foreign key in the Authorship table will also be updated to keep everything connected. This helps to keep the data in the database accurate and prevents errors.

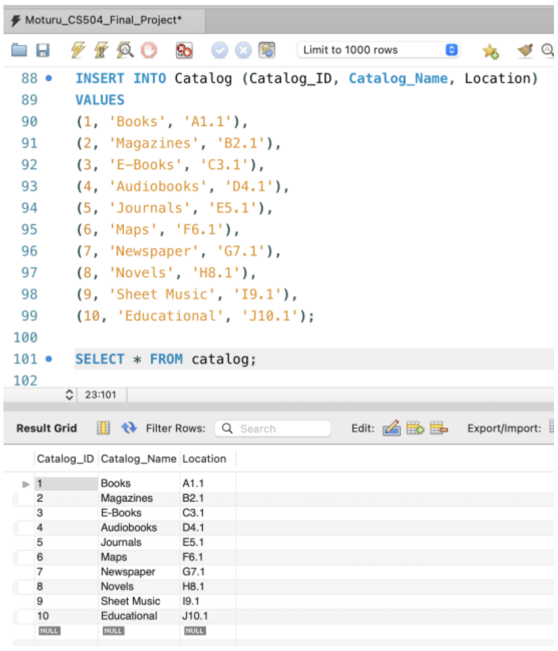
2.2 Population of Database

Below displays and describes the population of database with the given sample data and its respective sample output.

```
#DATA INSERTION

INSERT INTO Catalog (Catalog_ID, Catalog_Name, Location)
VALUES
(1, 'Books', 'A1.1'),
(2, 'Magazines', 'B2.1'),
(3, 'E-Books', 'C3.1'),
(4, 'Audiobooks', 'D4.1'),
(5, 'Journals', 'E5.1'),
(6, 'Maps', 'F6.1'),
(7, 'Newspaper', 'G7.1'),
(8, 'Novels', 'H8.1'),
(9, 'Sheet Music', 'I9.1'),
(10, 'Educational', 'J10.1');
```

Sample Output:

SELECT * FROM catalog;



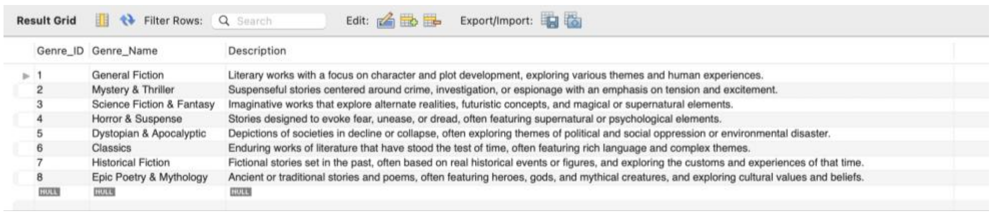
```sql
INSERT INTO Catalog (Catalog_ID, Catalog_Name, Location)
VALUES
(1, 'Books', 'A1.1'),
(2, 'Magazines', 'B2.1'),
(3, 'E-Books', 'C3.1'),
(4, 'Audiobooks', 'D4.1'),
(5, 'Journals', 'E5.1'),
(6, 'Maps', 'F6.1'),
(7, 'Newspaper', 'G7.1'),
(8, 'Novels', 'H8.1'),
(9, 'Sheet Music', 'I9.1'),
(10, 'Educational', 'J10.1');

SELECT * FROM catalog;
```

Catalog_ID	Catalog_Name	Location
1	Books	A1.1
2	Magazines	B2.1
3	E-Books	C3.1
4	Audiobooks	D4.1
5	Journals	E5.1
6	Maps	F6.1
7	Newspaper	G7.1
8	Novels	H8.1
9	Sheet Music	I9.1
10	Educational	J10.1
NULL	NULL	NULL

INSERT INTO Genre (Genre_ID, Genre_Name, Description)
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, or espionage 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.');

Sample Output:

SELECT * FROM Genre;



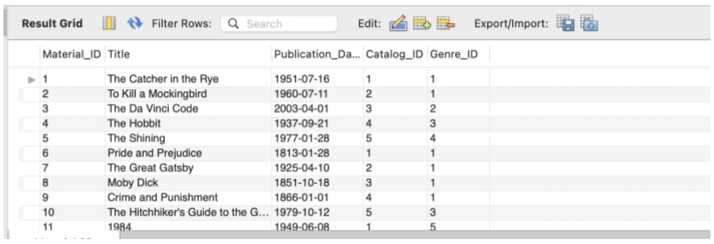
Genre_ID	Genre_Name	Description
1	General Fiction	Literary works with a focus on character and plot development, exploring various themes and human experiences.
2	Mystery & Thriller	Suspenseful stories centered around crime, investigation, or espionage with an emphasis on tension and excitement.
3	Science 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.
NULL	NULL	NULL

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, '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 Nights Dream','1596-01-01',10,7),
    (31, 'The Tricky Book','1888-01-01',10,7);
```

Sample Output:

SELECT * FROM Material;

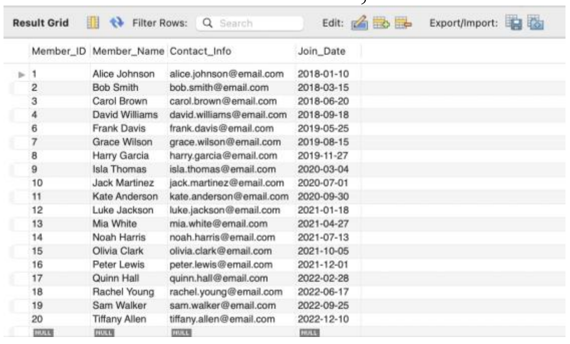


Material_ID	Title	Publication_Da...	Catalog_ID	Genre_ID
1	The Catcher in the Rye	1951-07-16	1	1
2	To Kill a Mockingbird	1960-07-11	2	1
3	The Da Vinci Code	2003-04-01	3	2
4	The Hobbit	1937-09-21	4	3
5	The Shining	1977-01-28	5	4
6	Pride and Prejudice	1813-01-28	1	1
7	The Great Gatsby	1925-04-10	2	1
8	Moby Dick	1851-10-18	3	1
9	Crime and Punishment	1866-01-01	4	1
10	The Hitchhiker's Guide to the G...	1979-10-12	5	3
11	1984	1949-06-08	1	5

```
INSERT INTO Member (Member_ID, Member_Name, 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');
```

Sample Output:

SELECT * FROM Member;

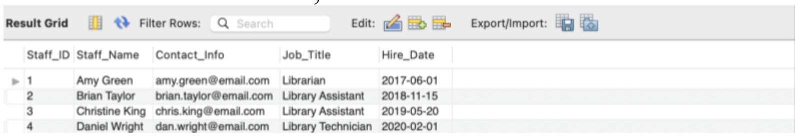


Member_ID	Member_Name	Contact_Info	Join_Date
1	Alice Johnson	alice.johnson@email.com	2018-01-10
2	Bob Smith	bob.smith@email.com	2018-03-15
3	Carol Brown	carol.brown@email.com	2018-06-20
4	David Williams	david.williams@email.com	2018-09-18
6	Frank Davis	frank.davis@email.com	2019-05-25
7	Grace Wilson	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
NULL	NULL	NULL	NULL

```
INSERT INTO Staff (Staff_ID, Staff_Name, 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');
```

Sample Output:

SELECT * FROM Staff;

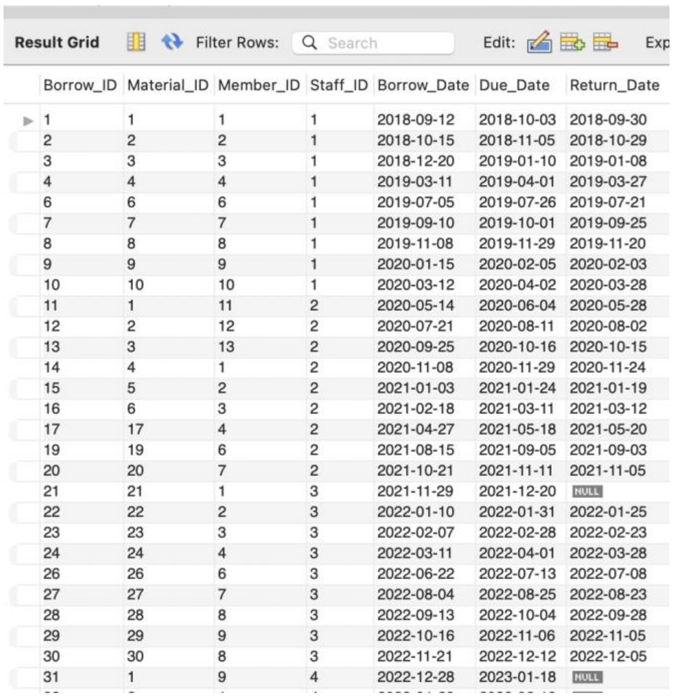


Staff_ID	Staff_Name	Contact_Info	Job_Title	Hire_Date
1	Amy Green	amy.green@email.com	Librarian	2017-06-01
2	Brian Taylor	brian.taylor@email.com	Library Assistant	2018-11-15
3	Christine King	chris.king@email.com	Library Assistant	2019-05-20
4	Daniel Wright	dan.wright@email.com	Library Technician	2020-02-01

```
INSERT INTO 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);
```

Sample Output:

```
SELECT * FROM Borrow;
```

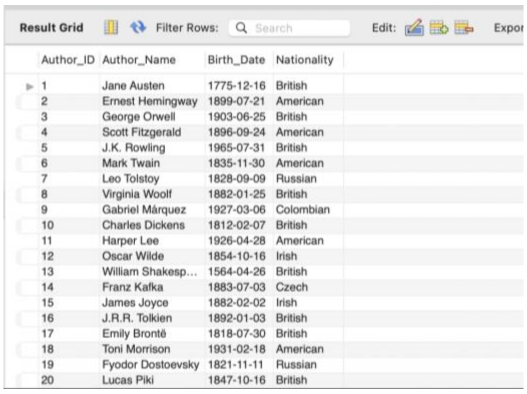


Borrow_ID	Material_ID	Member_ID	Staff_ID	Borrow_Date	Due_Date	Return_Date
1	1	1	1	2018-09-12	2018-10-03	2018-09-30
2	2	2	1	2018-10-15	2018-11-05	2018-10-29
3	3	3	1	2018-12-20	2019-01-10	2019-01-08
4	4	4	1	2019-03-11	2019-04-01	2019-03-27
6	6	6	1	2019-07-05	2019-07-26	2019-07-21
7	7	7	1	2019-09-10	2019-10-01	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
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
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

```
INSERT INTO Author (Author_ID, Author_Name, 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');
```

Sample Output:

```
SELECT * FROM Author;
```

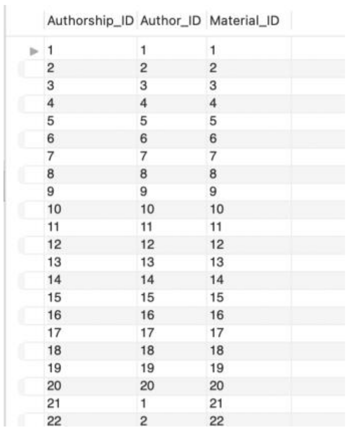


Author_ID	Author_Name	Birth_Date	Nationality
1	Jane Austen	1775-12-16	British
2	Ernest Hemingway	1899-07-21	American
3	George Orwell	1903-06-25	British
4	Scott Fitzgerald	1896-09-24	American
5	J.K. Rowling	1965-07-31	British
6	Mark Twain	1835-11-30	American
7	Leo Tolstoy	1828-09-09	Russian
8	Virginia 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 Shakesp...	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 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);
```

Sample Output:

```
SELECT * FROM Authorship;
```



Authorship_ID	Author_ID	Material_ID
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

3. Querying and Manipulation

A set of SQL queries is used to perform common tasks such as searching, updating, inserting, and deleting, in order to answer the given questions. These queries also include advanced techniques like joins, aggregation, and subqueries to accomplish these tasks.

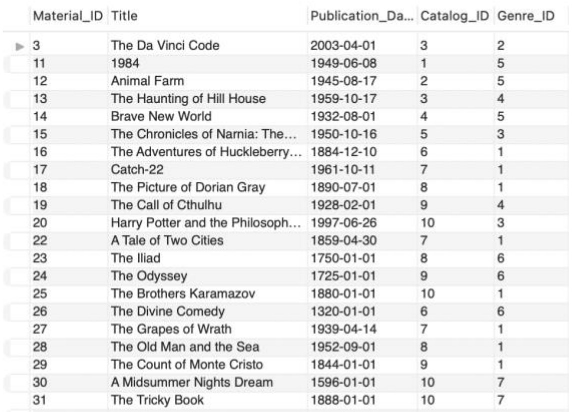
#Queries/Updates

#1. Which materials are currently available in the library?

The below SQL query selects all materials that are currently available in the library by checking which materials are not borrowed and not yet returned.

```
SELECT *
FROM Material
WHERE Material_ID not IN (
   SELECT Material_ID
   FROM Borrow
   WHERE Return_Date IS NULL
);
```

Sample Output:



Material_ID	Title	Publication_Da...	Catalog_ID	Genre_ID
3	The Da Vinci Code	2003-04-01	3	2
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...	1950-10-16	5	3
16	The Adventures of Huckleberry...	1884-12-10	6	1
17	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 Philosoph...	1997-06-26	10	3
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 Nights Dream	1596-01-01	10	7
31	The Tricky Book	1888-01-01	10	7

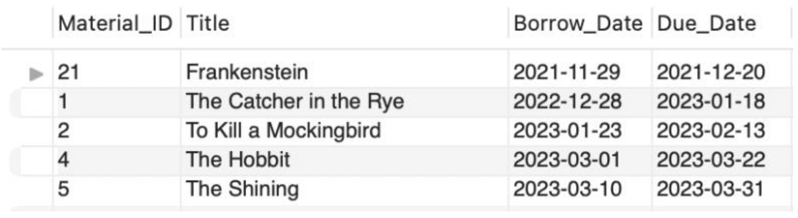
#2. Which materials are currently overdue? Suppose today is 04/01/2023, and show the borrow date and due date of each material.

This SQL query retrieves a list of materials that are currently overdue along with their corresponding borrow date and due date. It is achieved by joining the Material and Borrow tables and by checking for borrow records with a due date before the current date and no return date.

The Borrow_Date and Due_Date and Material_ID are displayed in the output so that it helps to verify the records. Ideally, to answer the given question the Title results alone will suffice.

```
SELECT M.Material_ID, M.Title, B.Borrow_Date, B.Due_Date
FROM Material AS M
INNER JOIN Borrow AS B
ON M.Material_ID = B.Material_ID
WHERE B.Return_Date IS NULL AND B.Due_Date < '2023-04-1';
```

Sample Output:



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

#3.What are the top 10 most borrowed materials in the library? Show the title of each material and order them based on their available counts.

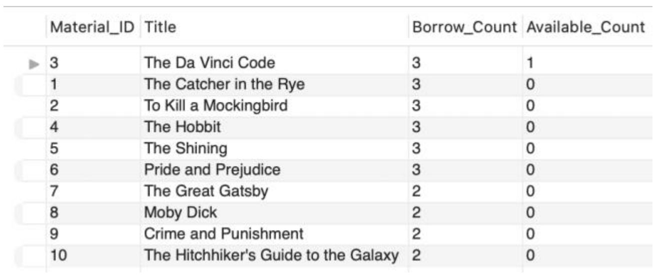
#Considering the number of materials currently available.

The below SQL query selects the top 10 most borrowed materials in the library based on their available counts, showing the title of each material. This is done by joining the Material table with a subquery that counts the number of times each material has been borrowed and orders them in descending order, and then orders the results by available count in descending order as well.

```
SELECT M.Material_ID, M.Title, B.Borrow_Count, IF (M.Material_ID IN (SELECT
Material_ID

                                            FROM Borrow

                                            WHERE Return_Date IS NULL), 0, 1) AS
Available_Count
FROM Material AS M, (SELECT Material_ID, COUNT(Material_ID) AS Borrow_Count
                                                    FROM Borrow
                                                    GROUP BY Material_ID
                                                    ORDER BY Borrow_Count DESC
                                                    LIMIT 10) AS B
```

```
WHERE M.Material_ID= B.Material_ID
ORDER BY Available_Count DESC;
```

Sample Output:



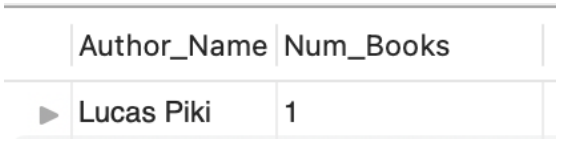
Material_ID	Title	Borrow_Count	Available_Count
3	The Da Vinci Code	3	1
1	The Catcher in the Rye	3	0
2	To Kill a Mockingbird	3	0
4	The Hobbit	3	0
5	The Shining	3	0
6	Pride and Prejudice	3	0
7	The Great Gatsby	2	0
8	Moby Dick	2	0
9	Crime and Punishment	2	0
10	The Hitchhiker's Guide to the Galaxy	2	0

#4. How many books has the author Lucas Piki written?

This SQL query retrieves the number of books written by the author "Lucas Piki" by joining the Authorship and Author tables using subquery and counting the number of Material_IDs associated with the Author's ID.

```
SELECT A2.Author_Name, COUNT(A1.Material_ID) AS Num_Books
FROM Authorship AS A1, (SELECT Author_ID, Author_Name
                                                    FROM Author WHERE Author_Name =
'Lucas Piki') AS A2
WHERE A1.Author_ID = A2.Author_ID;
```

Sample Output:



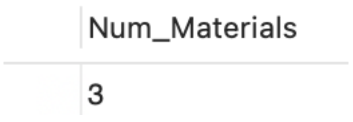
Author_Name	Num_Books
Lucas Piki	1

#5.How many books were written by two or more authors?

The below SQL query determines the total number of books with two or more authors. This is accomplished by counting the number of distinct author IDs for each material and grouping the authorship table by Material_ID. The total number of resources that meet this requirement is then calculated by only choosing those with two or more authors.

```
SELECT COUNT(*) AS Num_Materials
FROM (SELECT Material_ID, COUNT(Author_ID) AS Num_Authors FROM Authorship
              GROUP BY Material_ID
              HAVING Num_Authors >= 2) AS A;
```

Sample Output:



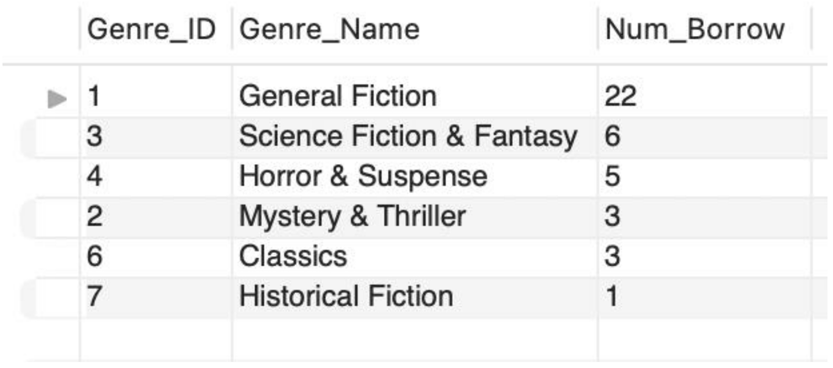
Num_Materials
3

#6. What are the most popular genres in the library?

The SQL query retrieves a list of the most popular genres in the library by joining the Genre and Material tables and grouping them by genre using subquery. The number of times each material has been borrowed is calculated and sums up the number of borrows for each genre. The results are ordered by the number of borrows in descending order.

```
SELECT MB.Genre_ID, G.Genre_Name, SUM(MB.Num_Borrow) AS Num_Borrow
FROM Genre AS G, (SELECT B.Material_ID, M.Genre_ID, COUNT(*) AS Num_Borrow
                  FROM Borrow AS B, Material AS M
                  WHERE B.Material_ID = M.Material_ID
                  GROUP BY Material_ID
                  ORDER BY Num_Borrow DESC) AS MB
WHERE G.Genre_ID = MB.Genre_ID
GROUP BY Genre_ID
ORDER BY Num_Borrow DESC;
```

Sample Output:



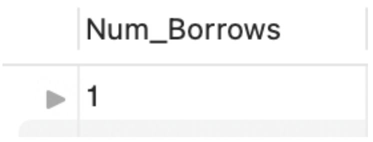
Genre_ID	Genre_Name	Num_Borrow
1	General Fiction	22
3	Science Fiction & Fantasy	6
4	Horror & Suspense	5
2	Mystery & Thriller	3
6	Classics	3
7	Historical Fiction	1

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

This SQL query selects all borrow records having a borrow date within that date range and counts them using the COUNT() function to determine how many items have been borrowed between the dates of September 2020 to October 2020.

```
SELECT COUNT(*) AS Num_Borrows
FROM Borrow
WHERE Borrow_Date BETWEEN '2020-09-01' AND '2020-10-31';
```

Sample Output:



Num_Borrows
1

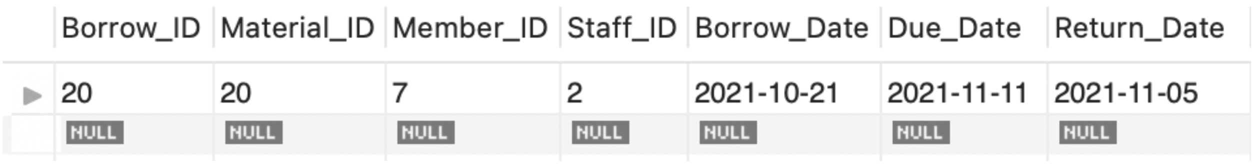
#8. How do you update the “Harry Potter and the Philosopher's Stone” when it is returned on 04/01/2023?

The given record of the material "Harry Potter and the Philosopher's Stone" is updated to 04/01/2023 using the below SQL by finding its corresponding Material_ID and setting the Return_Date to that value.

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

Sample Output:

```
SELECT * FROM Borrow WHERE Material_ID = (SELECT Material_ID FROM Material
WHERE Title = 'Harry Potter and the Philosopher''s Stone');
```



	Borrow_ID	Material_ID	Member_ID	Staff_ID	Borrow_Date	Due_Date	Return_Date
▶	20	20	7	2	2021-10-21	2021-11-11	2021-11-05
	NULL	NULL	NULL	NULL	NULL	NULL	NULL

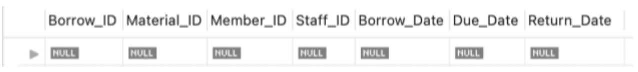
#9: How do you delete the member Emily Miller and all her related records from the database?

This SQL statement deletes the member record for Emily Miller from the Member table as well as all of the member's linked records from the Borrow table. Since ‘member' is a MySQL keyword, SQL_SAFE_UPDATES must be set to 0, which enables safe mode and allows the delete statement to be run.

```
DELETE FROM Borrow WHERE Member_ID IN (SELECT Member_ID FROM Member
WHERE Member_Name = 'Emily Miller');

SET SQL_SAFE_UPDATES = 0; # As member is a keyword in mysql hence the where
cluase is being ignored. Therefore, inorder to execute this query we need to disable the SQL
safe mode.
DELETE FROM Member WHERE Member_Name = 'Emily Miller';
```

Sample Output:

```
SELECT * FROM Borrow WHERE Member_ID IN (SELECT Member_ID FROM Member
WHERE Member_Name = 'Emily Miller');
```



	Borrow_ID	Material_ID	Member_ID	Staff_ID	Borrow_Date	Due_Date	Return_Date
▶	NULL	NULL	NULL	NULL	NULL	NULL	NULL

#10. How do you add the following material to the database? Title: New book, Date: 2020-08-01, Catalog: E-Books, Genre: Mystery & Thriller, Author: Lucas Pipi

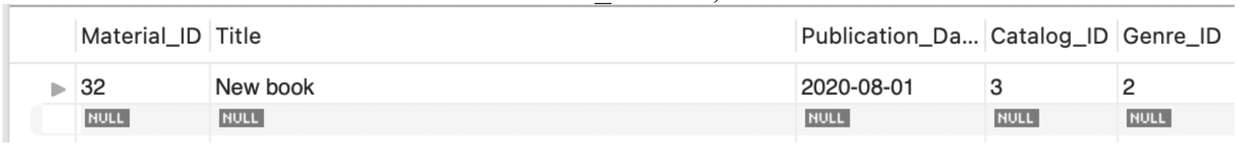
This SQL code adds a new material to the database with the title 'New book', publication date of '2020-08-01', catalog ID of 3 and genre ID of 2. It also adds a new author 'Lucas Pipi' with

ID 21 and creates a relationship between this author and the newly added material with ID 32 in the Authorship table.

```
INSERT INTO Material VALUES (32,'New book', '2020-08-01', 3, 2);
INSERT INTO Author Values (21, 'Lucas Pipi', NULL, NULL);
INSERT INTO Authorship VALUES (34, 21, 32);
```

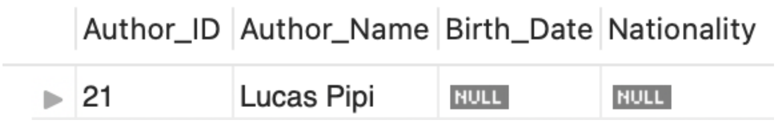
Sample Outputs:

SELECT * FROM Material WHERE Material_ID = 32;



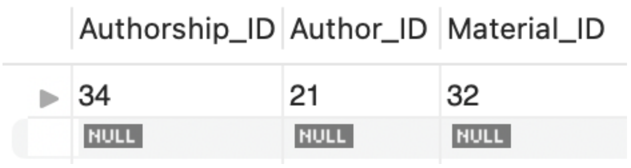
	Material_ID	Title	Publication_Da...	Catalog_ID	Genre_ID
▶	32	New book	2020-08-01	3	2
	NULL	NULL	NULL	NULL	NULL

SELECT * FROM Author WHERE Author_ID = 21;



	Author_ID	Author_Name	Birth_Date	Nationality
▶	21	Lucas Pipi	NULL	NULL

SELECT * FROM Authorship WHERE Authorship_ID = 34;



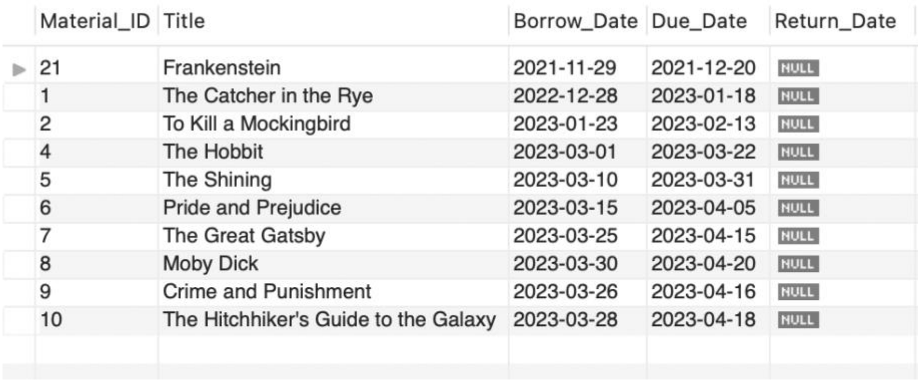
	Authorship_ID	Author_ID	Material_ID
▶	34	21	32
	NULL	NULL	NULL

4. Design

1) The following shows the due Materials considering the current date.

```
SELECT M.Material_ID, M.Title, B.Borrow_Date, B.Due_Date, B.Return_Date
FROM Material AS M
INNER JOIN Borrow AS B
ON M.Material_ID = B.Material_ID
WHERE B.Return_Date IS NULL AND B.Due_Date < CURDATE();
```

Sample Output:



	Material_ID	Title	Borrow_Date	Due_Date	Return_Date
▶	21	Frankenstein	2021-11-29	2021-12-20	NULL
	1	The Catcher in the Rye	2022-12-28	2023-01-18	NULL
	2	To Kill a Mockingbird	2023-01-23	2023-02-13	NULL
	4	The Hobbit	2023-03-01	2023-03-22	NULL
	5	The Shining	2023-03-10	2023-03-31	NULL
	6	Pride and Prejudice	2023-03-15	2023-04-05	NULL
	7	The Great Gatsby	2023-03-25	2023-04-15	NULL
	8	Moby Dick	2023-03-30	2023-04-20	NULL
	9	Crime and Punishment	2023-03-26	2023-04-16	NULL
	10	The Hitchhiker's Guide to the Galaxy	2023-03-28	2023-04-18	NULL

The above SQL query retrieves a list of materials from a library that are currently borrowed and their due date has already passed based on the current date. The inner join function is used to match the Material_ID in the Material table with the same column in the Borrow table, filtering the results based on the Borrow table's return date being null and the due date being less than the current date. The retrieved columns include the Material_ID, Title, Borrow _Date, Due_Date, and Return_Date.

2) To incorporate the above feature, we can modify the table design as follows:

   The following attributes are to be added to the member entity.

   I. Member_Status
   II. Overdue_Fee
   III. Num_Overdue

Considering the following assumptions:

- Member_Status can have the values of 0 and 1 which represents inactive member and active member respectively.
- Overdue_Fee can be updated manually depending on the overdue occurrences.
- Num_Overdue is also updated manually depending on the overdue occurrences.

Trigger function can be used in order to Automatically deactivate the membership based on the member’s overdue occurrence (>= three times). And reactivate the membership once the member pays the overdue fee.

```
CREATE TRIGGER Deactivate_Membership

AFTER UPDATE ON Member

FOR EACH ROW

BEGIN

   -- Deactivate membership if overdue count is greater than or equal to 3

   IF New.Num_Overdue >= 3 AND Old.Num_Overdue <3 THEN

      UPDATE Member SET Member_Status = 0 WHERE Member_ID =
New.Member_ID;

   END IF;

   -- Reactivate membership if overdue count is less than 3 and the Overdue_Fee is 0.

   IF New.Overdue_Fee = 0 AND Old.Overdue_Fee > 0 THEN
```

```
        UPDATE Member SET Member_Status = 1 WHERE Member_ID =
New.Member_ID;

      END IF;

   IF New.Overdue_Fee = 0 THEN

        UPDATE Member SET Num_Overdue = 0 WHERE Member_ID =
New.Member_ID;

     END IF;

END;
```

5. Conclusion

This project has been an extensive exercise that involved the design, implementation, and querying of a database, serving as a practical application of the concepts taught throughout the course. The main objective of the project was to create a database management system for a public library, with the ability to efficiently store and manage information about library materials and members, facilitate borrowing and tracking of library materials, and generate reports on library usage and related statistics.

An Entity-Relationship diagram has been included which illustrates the entities and relationships involved in managing a library. These entities included Material, Catalog, Genre, Borrow, Author, Authorship, Member, and Staff. To meet the project objectives, a set of requirements were established, which included defining the project scope, designing the database schema, optionally normalizing the schema, implementing the schema using an appropriate DBMS (MySQL), and developing SQL queries or stored procedures to perform common tasks and advanced querying techniques.

To conclude, this project offered a chance to put into practice the principles and techniques of database management to solve a real-world problem. By creating and implementing an effective database system for a public library, the project demonstrated the ability to apply the learned concepts to practical situations.

6. References

[1] (*CS504 Final Project.docx*, n.d.)