

Spring -2024

CS504 – Project Report

**Library Management System: Design, Implementation, and
Querying**

G01445921

Akhil Arekatika

Introduction:

In the era of digital transformation, effective data management is essential across various sectors, including libraries. This research paper outlines the design and implementation of a database management system (DBMS) customized for a public library environment. The project is undertaken as part of the coursework for CS504 Principles of Data Management and Mining. The project's objective is to develop a functional database system facilitating the efficient management of library resources like books, magazines, digital media, and other materials. Furthermore, the system aims to streamline membership management, borrowing processes, and provide analytical insights to assist library staff in decision-making. Utilizing MySQL Workbench as the chosen Database Management System, this paper provides a comprehensive overview of the database design, implementation, querying, and manipulation techniques used to address the specific requirements of the library domain. Additionally, the project explores extending the database system to include advanced features such as automated alerts for overdue materials and membership management based on overdue occurrences.

Overview:

This research paper presents the design and implementation of a database management system (DBMS) tailored for a public library environment. The project is conducted as part of the coursework for CS504 Principles of Data Management and Mining. The project aims to develop a functional database system enabling efficient management of library resources, membership, borrowing processes, and generation of analytical insights for data-driven decision-making by library staff. The paper outlines the database design, implementation, querying, and manipulation techniques used, focusing on MySQL Workbench as the chosen Database Management System. Additionally, the paper explores extending the database system to incorporate advanced features such as automated alerts for overdue materials and membership management based on overdue occurrences.

Technological Resources:

1. MySQL Workbench: MySQL Workbench serves as the primary Database Management System for implementing the library database. It is selected for its robust features, scalability, and support for relational data modeling.

2. CSV Files: Eight independent CSV files are provided, containing sample data required for the project:

- Borrow.csv
- Genre.csv
- Catalog.csv

- Authorship.csv
- Author.csv
- Material.csv
- Staff.csv
- Member.csv

3. Draw.io: Draw.io is used for creating Entity-Relationship (ER) diagrams as part of the database design process. ER diagrams offer a visual representation of the database schema, including entities, attributes, and relationships.

By utilizing these materials and tools, the research paper aims to provide a comprehensive overview of the database design and implementation process, along with insights into the practical application of data management and mining principles in a real-world scenario.

Procedure for Executing the Code:

1. Database Creation and Selection:

Commence by setting up a database environment suitable for executing the code.

Use the appropriate command or interface to create a new database instance.

Ensure that the newly created database is selected as the active environment for subsequent operations.

2. Table Creation:

Employ the provided SQL queries to create all necessary tables within the designated database.

Execute each query sequentially to establish the required table structures.

Verify the successful creation of tables by reviewing the database schema or utilizing relevant commands.

3. Data Insertion:

Populate the created tables with the requisite values by inserting data into each table.

Utilize manual insertion methods or import data from external sources such as CSV files.

Ensure that the data insertion process aligns with the defined table schemas and constraints.

4. Query Execution:

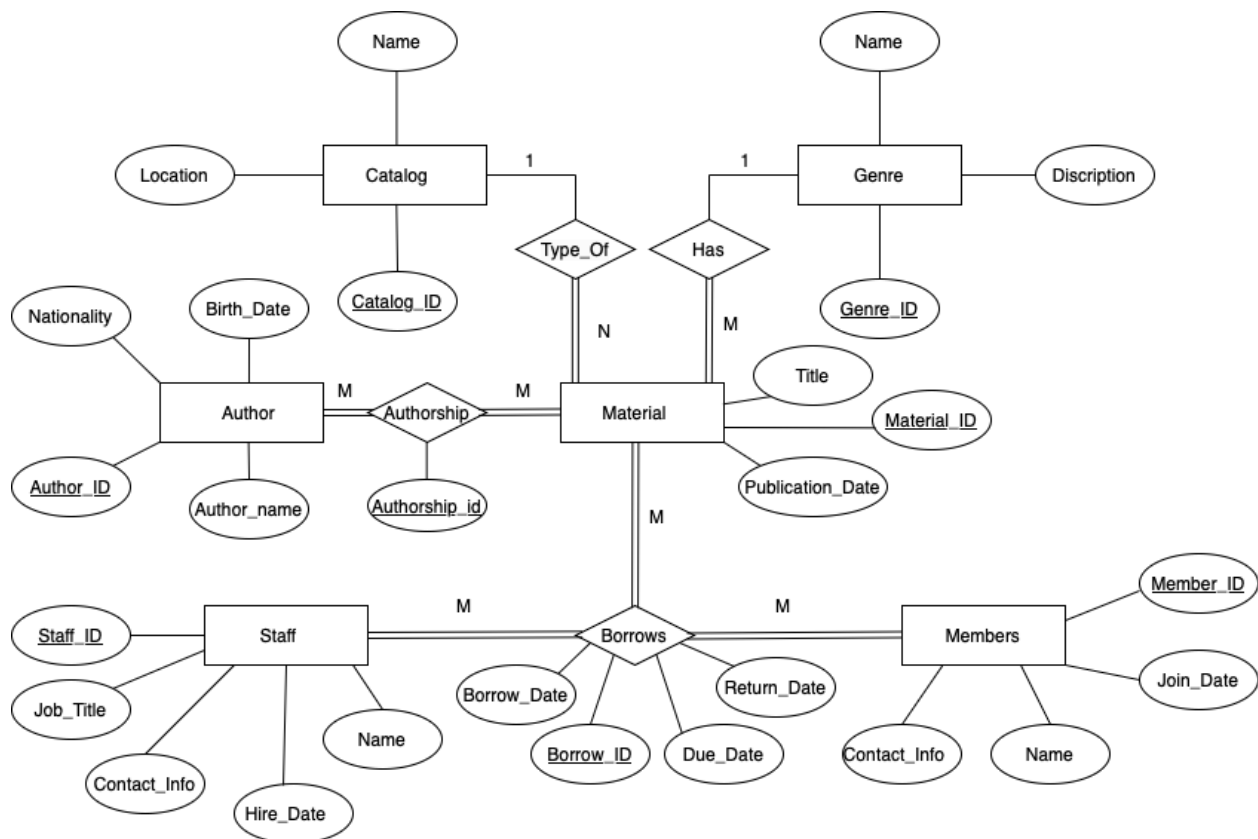
Execute the predefined SQL queries crafted to address specific questions or tasks.

Utilize the appropriate command or interface to execute each query against the populated database. Review the query results to ensure correctness and completeness in accordance with the specified requirements.

By adhering to these detailed steps, users can successfully execute the provided code, create the necessary database structures, populate them with relevant data, and execute queries to obtain desired results.

Entity-Relationship Diagram:

Utilizing the foundational Entities and Relationships outlined as the framework for this project, I have constructed the following Entity-Relationship Diagram.



Within the depicted ER Diagram, the Entities and Relationships are delineated as follows:

Entities:

- Catalog Entity:** Within the schema, Catalog is represented as a regular entity containing attributes like Catalog_ID, Name, and Location. Catalog_ID serves as the primary identifier for each catalog entry.

●**Genre Entity:** Genre is another standard entity featuring attributes such as Name, Description, and Genre_ID. Genre_ID is designated as the primary identifier for unique identification purposes.

●**Author Entity:** Author is depicted as a regular entity, encompassing attributes like Author_ID, Name, Birth_Date, and Nationality. Author_ID is utilized as the primary identifier for each author.

●**Material Entity:** Material is portrayed as a standard entity, housing attributes like Title, Material_ID, Publication_Date, Catalog_ID, Genre_ID, and Borrow_ID. Material_ID is assigned as the primary identifier, while Catalog_ID and Genre_ID serve as foreign keys referencing the Catalog and Genre tables, respectively. Additionally, Borrow_ID acts as a foreign key referencing the Borrows table within the schema.

●**Staff Entity:** Staff Entity is characterized by attributes including Staff_ID, Job_Title, Contact_Info, Hire_Date, and Name. Staff_ID is utilized as the primary identifier for staff members.

●**Member Entity:** Member Entity consists of attributes like Member_ID, Name, Contact_Info, and Join_Date. Member_ID serves as the primary identifier for members.

Relationships:

●**Belongs_To:** The association between Material and Catalog Entities is facilitated through the Belongs_To relationship. It exhibits a cardinality ratio of 1:N, indicating that Material entities can be linked to a specific Catalog category, while Catalog entities possess optional participation.

●**Type_Of:** Genre Entity is linked to the Material Entity via the Type_Of relationship. This relationship demonstrates a cardinality ratio of 1:N, signifying that a material can be associated with a particular genre. Material entities demonstrate complete participation, while Genre entities exhibit single participation.

●**Authorship:** Authorship represents the relationship between Author and Material tables. This relationship showcases a cardinality ratio of N:M, indicating that multiple materials can have multiple authors. Material entities exhibit complete participation in this relationship. Authorship is treated as a separate table with Authorship_ID as the primary identifier. Material_ID and Author_ID act as foreign keys referencing the Material and Author tables, respectively.

●**Borrows:** Within the Library Schema, Borrows symbolizes a ternary relationship involving Material, Staff, and Member entities. All three entities exhibit N:M relationships, signifying that multiple staff members can issue materials and multiple members can borrow materials. Borrows features its own attributes like Borrow_ID, Borrow_Date, Return_Date, Due_Date, Staff_ID, Member_ID, and Material_ID. Borrow_ID is utilized as the primary identifier, while Staff_ID, Member_ID, and Material_ID serve as foreign keys referencing the Staff, Member, and Material tables, respectively.

Implementation of Database:

We see that the given Entities and Relationships strictly follow the 3NF Normal form and need not be normalized further.

Based on the Entity-Relationship Diagram written above, we implement the database in MySQL Workspace.

- **Creation of GENRE Table:**

SQL Query:

```
create table Genre(  
Genre_id int not null primary key,  
Genre_name varchar(30),  
description varchar(500));  
Select *from Genre;
```

Here we insert the data into the GENRE table using the postgres import function directly from the .csv files.

Output:

Result Grid			
Filter Rows:			
Edit: Export/Import:			
	Genre_id	Genre_name	description
▶	1	General Fiction	Literary works with a focus on character and pl...
	2	Mystery & Thriller	Suspenseful stories centered around crime, inv...
	3	Science Fiction & Fantasy	Imaginative works that explore alternate realiti...
	4	Horror & Suspense	Stories designed to evoke fear, unease, or dre...
	5	Dystopian & Apocalyptic	Depictions of societies in decline or collapse, oft...
	6	Classics	Enduring works of literature that have stood th...
	7	Historical Fiction	Fictional stories set in the past, often based on ...
	8	Epic Poetry & Mythology	Ancient or traditional stories and poems, often f...
★	NULL	NULL	NULL

- Creation of CATALOG Table:

SQL Query:

```
create table catalog(  
catalog_id int not null primary key,  
catalog_name varchar(30),  
location varchar(30));  
Select *from CATALOG;
```

Output:

Result Grid			
Filter Rows:			
	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	Newspaper	F6.1
	7	Maps	G7.1
	8	Novels	H8.1
	9	Sheet Music	I9.1
	10	Educational	J10.1
★	NULL	NULL	NULL

- **Creation of AUTHOR Table:**

SQL Query:

```
create table author(
author_id int not null primary key,
author_name varchar(30),
author_birthdate date,
nationality varchar(30));
Select *from AUTHOR;
```

Output:







Result Grid				
Filter Rows: <input type="text"/>				
Edit:   				
	author_id	author_name	author_birthdate	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 MÃairquez	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 BrontÃ«	1818-07-30	British

- **Creation of MATERIAL Table:**

SQL Query:

```
create table material(
material_id int not null primary key,
material_title varchar(100),
publication_date date,
catalog_id int,
genre_id int,
foreign key(catalog_id) references catalog(catalog_id),
foreign key(genre_id) references genre(genre_id));
Select *from MATERIAL;
```

Output:

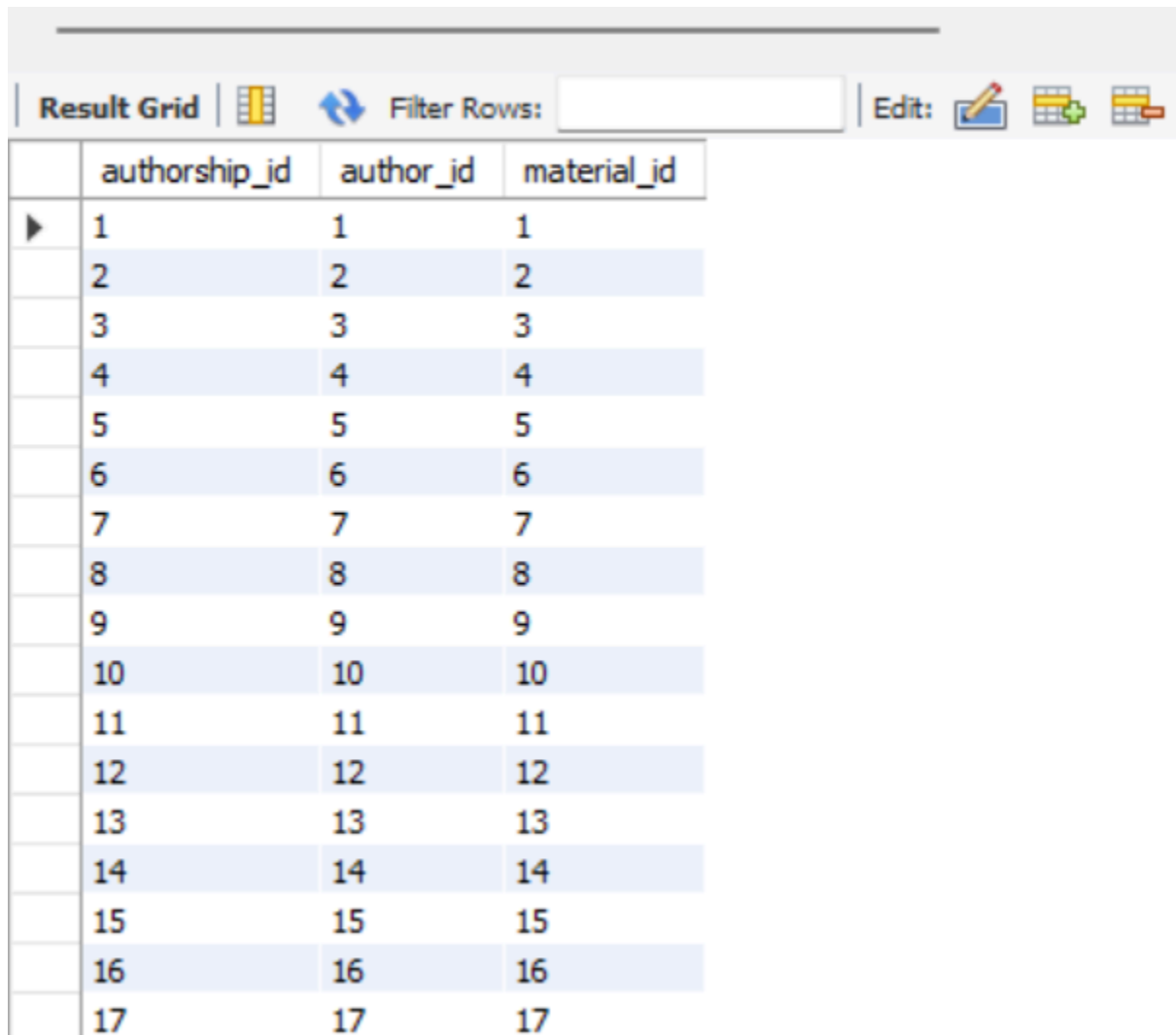
Result Grid   Filter Rows: <input type="text"/> Edit:    Export/Import: 					
	material_id	material_title	publication_date	catalog_id	genre_id
▶	1	The Catcher in the Rye	1951-07-16	1	1
	2	To Kill a Mockingbird	1960-07-11	2	1
	3	The Da Vinci Code	2003-04-01	3	2
	4	The Hobbit	1937-09-21	4	3
	5	The Shining	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 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 ...	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

- **Creation of AUTHOURSHIP Table:**

SQL Query:

```
create table authorship(  
authorship_id int not null primary key,  
author_id int,  
material_id int,  
foreign key(author_id) references author(auth_id),  
foreign key(material_id) references material(material_id));  
Select *from AUTHORSHIP;
```

Output:






	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

- **Creation of MEMBER Table:**

SQL Query:

```
create table member(
mem_id int not null primary key,
mem_name varchar(30),
mem_contact varchar(30),
mem_joindate date);
Select *from MEMBER;
```

Output:








Result Grid				
Filter Rows: <input type="text"/>				
Edit:   				
	mem_id	mem_name	mem_contact	mem_joindate
▶	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

- **Creation of STAFF Table:**

SQL Query:

```
create table staff(
staff_id int not null primary key,
staff_name varchar(30),
staff_contact varchar(30),
job_title varchar(30),
staff_hiredate date);
Select *from STAFF;
```

Output:

Result Grid   Filter Rows: <input type="text"/> Edit:    Export/Import:  					
	staff_id	staff_name	staff_contact	job_title	staff_hiredate
▶	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
✱	NULL	NULL	NULL	NULL	NULL

- **Creation of BORROW Table:**

SQL Query:

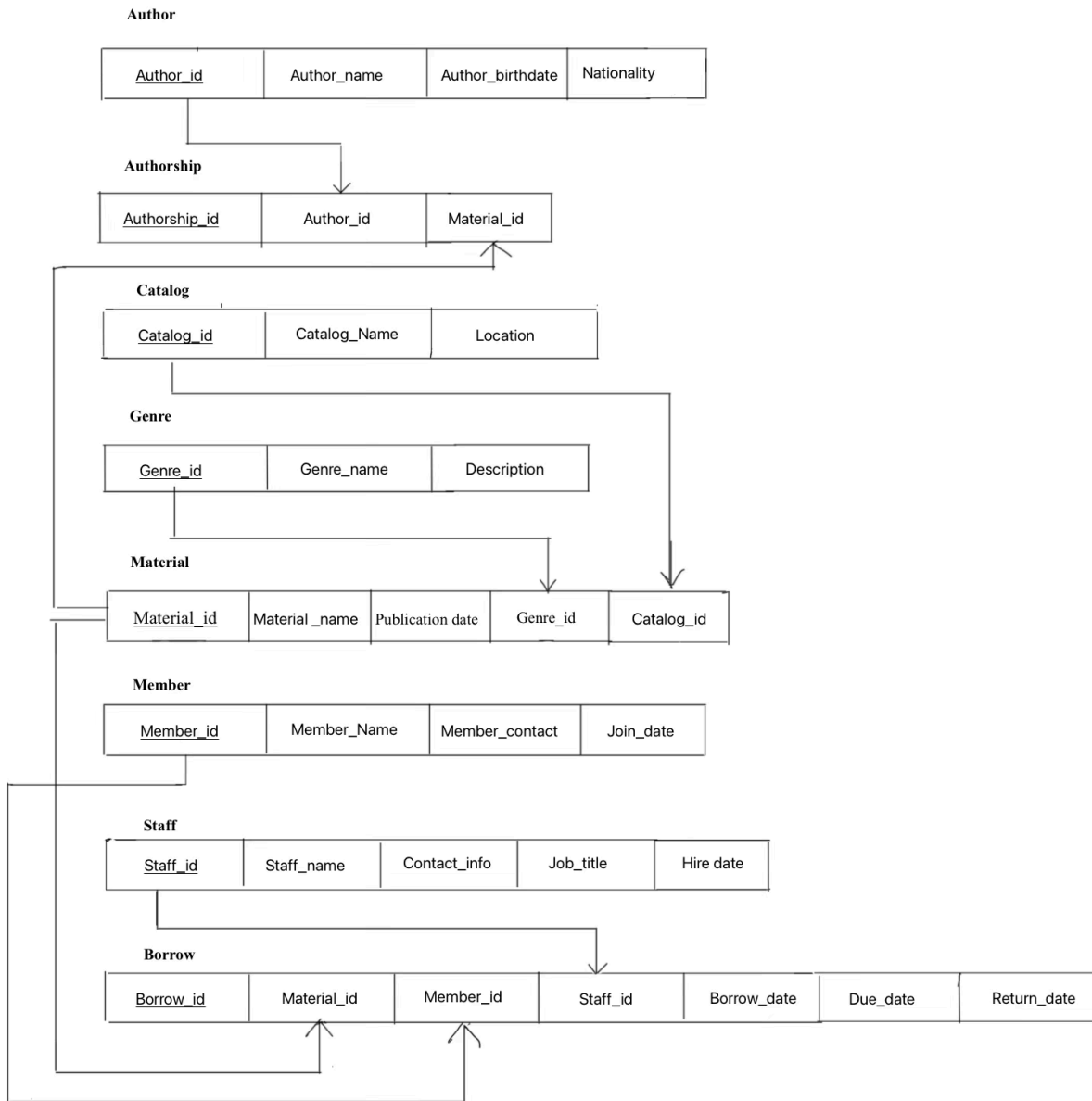
```
create table borrow(
borrow_id int not null primary key,
material_id int,
mem_id int,
staff_id int,
borrow_date date,
due_date date,
return_date date,
foreign key(material_id) references material(material_id),
foreign key(mem_id) references member(mem_id),
foreign key(staff_id) references staff(staff_id));
Select *from BORROW;
```

Output:

Result Grid		Filter Rows:		Edit:		Export/Import:	
	borrow_id	material_id	mem_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
	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

After the creation of the tables and the relationships we depict the same on an Relational Schema Diagram based on the Library Schema:

Relational Schema Diagram of Library Schema:



Queries and their Output:

1. Which materials are currently available in the library? If a material is borrowed and not returned, it's not considered as available.

SQL Query:

```
select material_id, material_title from material
```

```
where material_id not in(
```

```
select material_id from borrow
```

```
where return_date is null);
```

Output:





Result Grid			Filter Rows:	Edit:
	material_id	material_title		
▶	3	The Da Vinci Code		
	11	1984		
	12	Animal Farm		
	13	The Haunting of Hill House		
	14	Brave New World		
	15	The Chronicles of Narnia: The Lion the Witch an...		
	16	The Adventures of Huckleberry Finn		
	17	The Catch-22		
	18	The Picture of Dorian Gray		
	19	The Call of Cthulhu		
	22	A Tale of Two Cities		
	23	The Iliad		
	24	The Odyssey		
	25	The Brothers Karamazov		
	26	The Divine Comedy		
	27	The Grapes of Wrath		
	28	The Old Man and the Sea		

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

SQL Query:

```
select material.material_title, borrow.due_date, borrow.borrow_date from material , borrow
where borrow.material_id = material.material_id and borrow.due_date < '2023-04-01'
and borrow.return_date is null;
```

Output:

Result Grid   Filter Rows: <input type="text"/> Export:  Wrap Cell Content: 			
	material_title	due_date	borrow_date
▶	Harry Potter and the Philosopher's Stone	2020-11-11	2021-10-21
	Frankenstein	2021-12-20	2021-11-29
	The Catcher in the Rye	2023-01-18	2022-12-28
	To Kill a Mockingbird	2023-02-13	2023-01-23
	The Hobbit	2023-03-22	2023-03-01
	The Shining	2023-03-31	2023-03-10

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

SQL Query:

```
select material.material_title, count(borrow_id) as most_borrowed
from material
join borrow on material.material_id = borrow.material_id
group by material_title
order by most_borrowed desc
limit 10;
```


Output:

Result Grid			Filter Rows:	Export:
	material_title	most_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		

4. How many materials has the author Lucas Piki written?

SQL Query:

```
select count(authorship.Authorship_ID) as LucasPiki_materials,  
material.material_title as book_name  
from Author  
join Authorship on author.author_ID = authorship.author_ID  
join Material on authorship.Material_ID = material.Material_ID  
where author.author_name = 'Lucas Piki'  
group by material.material_title;
```

Output: I intend to display the name of the book as well as the count was 1

Result Grid			Filter Rows:	Export:
	LucasPiki_materials	book_name		
▶	1	Harry Potter and the Philosopher's Stone		

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

SQL Query:

```
select count(*) as count_of_two_and_more_authors
from
( select material_id
from authorship
group by material_id
having count(*) >= 2)
as authors;
```

Output:



The screenshot shows a database interface with a 'Result Grid' tab. The grid contains one row with the column header 'count_of_two_and_more_authors' and the value '4'. Above the grid, there are controls for 'Filter Rows' and 'Export'.

count_of_two_and_more_authors
4

6. What are the most popular genres in the library ranked by the total number of borrowed times of each genre?

SQL Query:

```
SELECT Genre.Genre_ID, Genre.genre_name,
COUNT(Borrow.Borrow_ID) AS Total_Borrowed_Times
FROM Genre
LEFT JOIN Material ON Genre.Genre_ID = Material.Genre_ID
LEFT JOIN Borrow ON Material.Material_ID = Borrow.Material_ID
GROUP BY Genre.Genre_ID, Genre.genre_name
ORDER BY Total_Borrowed_Times DESC;
```

Output:

	Genre_ID	genre_name	Total_Borrowed_Times
▶	1	General Fiction	22
	3	Science Fiction & Fantasy	6
	4	Horror & Suspense	5
	2	Mystery & Thriller	3
	6	Classics	3
	7	Historical Fiction	1
	5	Dystopian & Apocalyptic	0
	8	Epic Poetry & Mythology	0

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

SQL Query:

```
select material.material_title, count(*) as count
from borrow, material
where material.material_id = borrow.material_id
and borrow.borrow_date between '2020-09-01' and '2020-10-01'
group by material_title;
```

Output:

	material_title	count
▶	The Da Vinci Code	1

8. How do you update the “Harry Potter and the Philosopher's Stone” when it is returned on

04/01/2023?

SQL Query:

update borrow

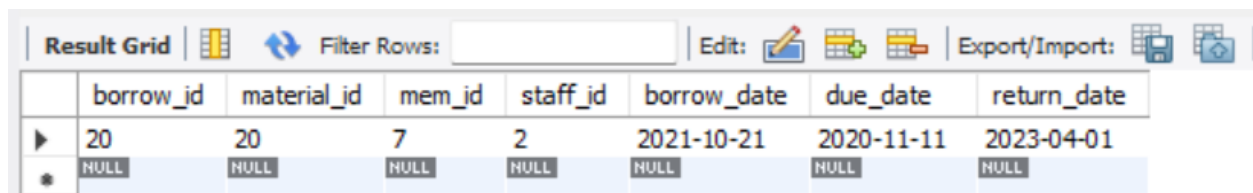
set return_date = '2023-04-01'

where material_id = (select material_id from material where material_title = 'Harry Potter and the Philosopher's Stone');

*select * from borrow*

where material_id = (select material_id from material where material_title = 'Harry Potter and the Philosopher's Stone');

Output:



The screenshot shows a database interface with a 'Result Grid' tab. The grid contains one row of data with the following values: borrow_id: 20, material_id: 20, mem_id: 7, staff_id: 2, borrow_date: 2021-10-21, due_date: 2020-11-11, and return_date: 2023-04-01. Below this row is a row of NULL values. The interface includes a 'Filter Rows' field, an 'Edit' button, and an 'Export/Import' button.

	borrow_id	material_id	mem_id	staff_id	borrow_date	due_date	return_date
▶	20	20	7	2	2021-10-21	2020-11-11	2023-04-01
•	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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

SQL Query:

delete from borrow where




mem_id = (select mem_id from member where mem_name = 'Emily Miller');

SET SQL_SAFE_UPDATES=OFF;

DELETE FROM member WHERE mem_name = 'Emily Miller';

*select * from member where mem_name = 'Emily Miller';*

Output:

Result Grid				
Filter Rows: <input type="text"/>				
Edit:   				
	mem_id	mem_name	mem_contact	mem_joindate
*	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 Luke

SQL Query:

```
insert into Author (author_ID,author_name)
```

```
values('21','Lucas Luke');
```

```
select * from Author where author_id = 21 and author_name = 'Lucas Luke';
```

```
insert into material(material_id, material_title, publication_date,catalog_id, genre_id)
```

```
values(32, 'New Book', '2020-08-01', (select catalog_id from catalog where catalog_name = 'E-Books'),  
(select genre_id from genre where genre_name = 'Mystery & Thriller'));
```

```
select * from material where material_title = 'New Book';
```

```
insert into authorship( authorship_id, author_id, material_id)
```

values(35, (select author_id from author where author_name = 'Lucas Luke'), (select material_id from material where material_title = 'New Book'));

*select * from authorship where authorship_id = 35;*

Output:

After inserting Author related records in the AUTHOR table:

	author_id	author_name	author_birthdate	nationality
▶	21	Lucas Luke	NULL	NULL
✱	NULL	NULL	NULL	NULL

After inserting Material related records into the MATERIAL table:

	material_id	material_title	publication_date	catalog_id	genre_id
▶	32	New Book	2020-08-01	3	2
✱	NULL	NULL	NULL	NULL	NULL

After Inserting the values of the new book in the AUTHORSHIP table:

	authorship_id	author_id	material_id
▶	35	21	32
✱	NULL	NULL	NULL

Future Work and Design:

- **Alert staff about overdue materials on a daily-basis?**

Here, we can build trigger to alert the librarians on the members who have the not returned the books although if it's overdue. The following procedure needs to be run daily so the librarians would be notified on the members who might have the overdue hold on their accounts and prevent further transactions.

SQL-Query:

```
CREATE FUNCTION overdue_alert()
RETURNS TRIGGER AS A
DECLARE
    member_name VARCHAR(255);
BEGIN
    SELECT M.member_name INTO member_name
    FROM "cs_project".Member AS M
    WHERE M.Member_ID = NEW.Member_ID;
    IF NEW.due_date < CURRENT_DATE then
        RAISE NOTICE 'ALERT: Member % has overdue books!', member_name;
    END IF;
    RETURN NEW;
END;

CREATE TRIGGER checking_duedate
AFTER INSERT OR UPDATE ON Borrow
FOR EACH ROW EXECUTE FUNCTION overdue_alert();
```

In This code we define a trigger function named ``overdue_alert`` and a corresponding trigger named ``check_due_date``. The trigger is designed to execute the ``alert_librarians`` function after each insertion or update operation on the "CS_Project".Borrow table. Inside the function, the code retrieves the name of the member associated with the newly inserted or updated row in the "CS_Project".Borrow table. It then checks if the due date for the borrowed item is overdue by comparing it to the current date. If overdue, a notice is raised (for testing purposes) indicating that the member has overdue books. In a real-world scenario, the ``RAISE NOTICE`` line would be replaced with an actual alert mechanism such as sending an email to the librarians. The trigger helps automate the process of alerting librarians about overdue books whenever new borrow records are added or existing ones are updated.

- **Automatically deactivate the membership based on the member's overdue occurrence (>= three times). And reactivate the membership once the member pays the overdue fee.**

Here we define a trigger to deactivate the ID of the members who have overdue more than three times in a month. The following trigger runs monthly and based on the overdue check, the member's account would be rendered deactivated until the overdue fee is paid.

SQL-Query:

```
CREATE FUNCTION membership_status(member_id_argument INT)
RETURNS void AS A
DECLARE
    overdue_count INT;
    last_month DATE;
BEGIN
    last_month = CURRENT_DATE - INTERVAL '1 month';
    SELECT COUNT(*)
    INTO overdue_count
    FROM cs_project.Borrow
    WHERE Member_ID = member_id_argument
    AND Return_Date IS NULL
    AND due_Date < CURRENT_DATE
    AND due_Date >= last_month;
    IF overdue_count >= 3 THEN
        UPDATE cs_project.Member
        SET Membership_Status = 'Inactive'
        WHERE Member_ID = member_id_argument;
        RAISE NOTICE 'Membership for member % deactivated due to overdue books.', member_id_arg;
    ELSE
        UPDATE cs_project.Member
        SET Membership_Status = 'Active'
        WHERE Member_ID = member_id_arg
        AND Overdue_Fees = 0
        RAISE NOTICE 'Membership for member % reactivated.', member_id_arg;
    END IF;
END;
```

In This SQL function, named `membership_status`, takes a member ID as an argument and is designed to manage the membership status of a library member. The function calculates the last month, then counts the number of overdue borrow occurrences for the specified member within that month. If the count is greater than or equal to 3, indicating multiple overdue instances, the membership status is set to 'Inactive' for that member, and a notice is raised. If the overdue count is less than 3, the function checks if the member has paid all overdue fees; if so, the membership status is set to 'Active,' and another notice is raised. This function provides a mechanism for automating the management of membership status based on the member's borrowing history, allowing for both deactivation due to excessive overdue instances and reactivation upon payment of overdue fee

Conclusion:

In conclusion, I have successfully developed a Library Management System using MySQL Workbench, incorporating essential features of Data Definition Language (DDL) and Data Manipulation Language (DML). Through meticulous database structuring, including primary and foreign keys, data integrity and relational consistency have been ensured. Leveraging MySQL Workbench's capabilities, robust mechanisms for managing library materials and member

interactions have been implemented. The elimination of data redundancy and optimization of storage efficiency contribute to the system's reliability and functionality. Overall, this project demonstrates the effective utilization of MySQL Workbench in creating a comprehensive solution for library resource management.