## SVKM’s NMIMS

**School of Technology Management & Engineering, Chandigarh**

A.Y. 2023 - 24

**Course: Database Management Systems Project Report**

|  |  |  |
| --- | --- | --- |
| **Program:** | Btech | |
| **Semester** | IV | |
| **Name of the Project:** | Book Review Management Project | |
|  | | |
| **Details of Project Members** |  |  |
| **Batch** | **Roll No.** | **Name** |
| **B2** | B068 | Vidhi Damani |
| **B2** | B074 | Samridhi Raj Sinha |
| **B2** | B082 | Asmi Parikh |
|  | | |

**Note:**

1. Create a readme file if you have multiple files
2. All files must be properly named (Example:R004\_DBMSProject)
3. Submit all relevant files of your work ( Report, all SQL files, Any other files)

## Plagiarism is highly discouraged (Your report will be checked for plagiarism Rubrics for the Project evaluation:

10 marks

First phase of evaluation:

|  |  |
| --- | --- |
| Innovative Ideas (5 Marks)  Design and Partial implementation (5 Marks) |  |
| Final phase of evaluation Implementation, presentation and viva,  Self-Learning and Learning Beyond classroom | 10 marks |

**Project Report**

**Book Management System**

**Course: DBMS AY: 2023-24**

## Table of Contents

|  |  |
| --- | --- |
| **Sr no.** | **Topic** |
| **1** | Storyline |
| **2** | Components of Database Design |
| **3** | Entity Relationship Diagram |
| **4** | Relational Model |
| **5** | Normalization |
| **6** | SQL Queries |
| **7** | Learning from the Project |
| **8** | Project Demonstration |
| **9** | Self-learning beyond classroom |
| **10** | Learning from the project |
| **8** | Challenges faced |
| **9** | Conclusion |

* 1. Storyline

**Book Review/Management System**

Our project aims to develop a sophisticated Book Review and Management System akin to Goodreads. The journey begins with establishing a robust database named "books\_db," meticulously designed to store author profiles, genre classifications, book details, user accounts, and interactions.

Authors, genres, and books are meticulously cataloged, each with unique identifiers and comprehensive metadata. Users can create personalized accounts, ensuring privacy and security with encrypted passwords.

The platform offers a rich user experience, allowing users to explore, discover, and engage with a vast library of literature. From classic works to contemporary bestsellers, every facet of the literary world is accessible, complete with book descriptions, reviews, and critical analyses.

With advanced features such as search functionality, personalized recommendations, and interactive discussion forums, our platform aims to revolutionize the way readers interact with literature online.

# ​Components of Database Design

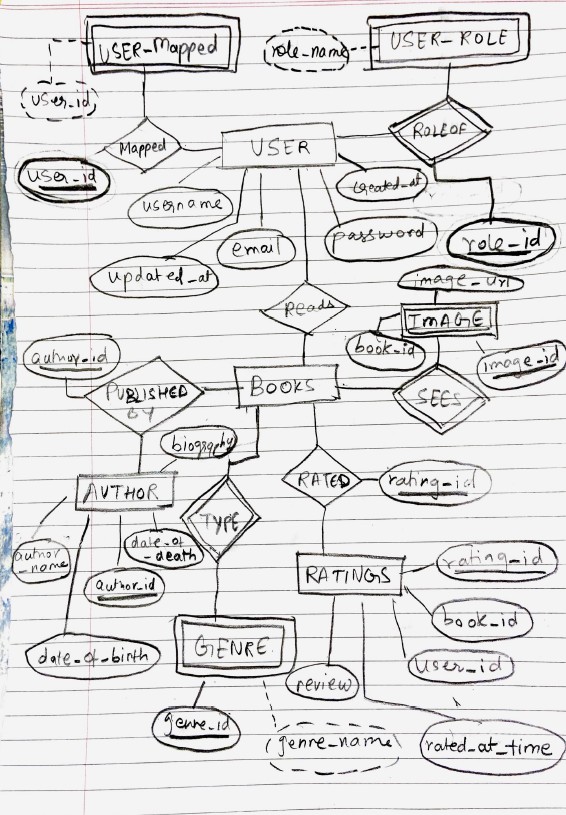
II. Components of Database Design Entities and Attributes:

1. Authors:
   * Attributes:
     + author\_id (Primary Key)
     + author\_name
     + biography
     + date\_of\_birth
     + date\_of\_death
   * Relationships:
     + One author can write multiple books (One-to-Many relationship with Books entity).
2. Genres:
   * Attributes:
     + genre\_id (Primary Key)
     + genre\_name
   * Relationships:
     + Each book can belong to one genre, and a genre can have multiple books (One-to-Many relationship with Books entity).
3. Images:
   * Attributes:
     + image\_id (Primary Key)
     + book\_id (Foreign Key referencing Books)
     + image\_url
   * Relationships:
     + Each book can have multiple images (One-to-Many relationship with Books entity).
4. Books:
   * Attributes:
     + book\_id (Primary Key)
     + title
     + author\_id (Foreign Key referencing Authors)
     + publishing\_date
     + isbn
     + genre\_id (Foreign Key referencing Genres)
     + language
     + page\_count
     + description
     + image\_id (Foreign Key referencing Images)
     + created\_at
     + updated\_at
   * Relationships:
     + Each book is written by one author (Many-to-One relationship with Authors).
     + Each book belongs to one genre (Many-to-One relationship with Genres).
     + Each book can have multiple ratings (One-to-Many relationship with Ratings entity).
5. Ratings:
   * Attributes:
     + rating\_id (Primary Key)
     + book\_id (Foreign Key referencing Books)
     + user\_id (Foreign Key referencing Users)
     + rating
     + review
     + rated\_at
   * Relationships:
     + Each rating is given by one user for one book (Many-to-One relationship with Books and Users entities).
6. Users:
   * Attributes:
     + user\_id (Primary Key)
     + username
     + email
     + password
     + created\_at
     + updated\_at
   * Relationships:
     + Each user can rate multiple books (One-to-Many relationship with Ratings entity).
     + Each user can have one or more roles (Many-to-Many relationship with User Roles entity).
7. User Roles:
   * Attributes:
     + role\_id (Primary Key)
     + role\_name
   * Relationships:
     + Each user can have one or more roles, and each role can be assigned to multiple users (Many-to-Many relationship with Users entity).

Cardinality and Participation:

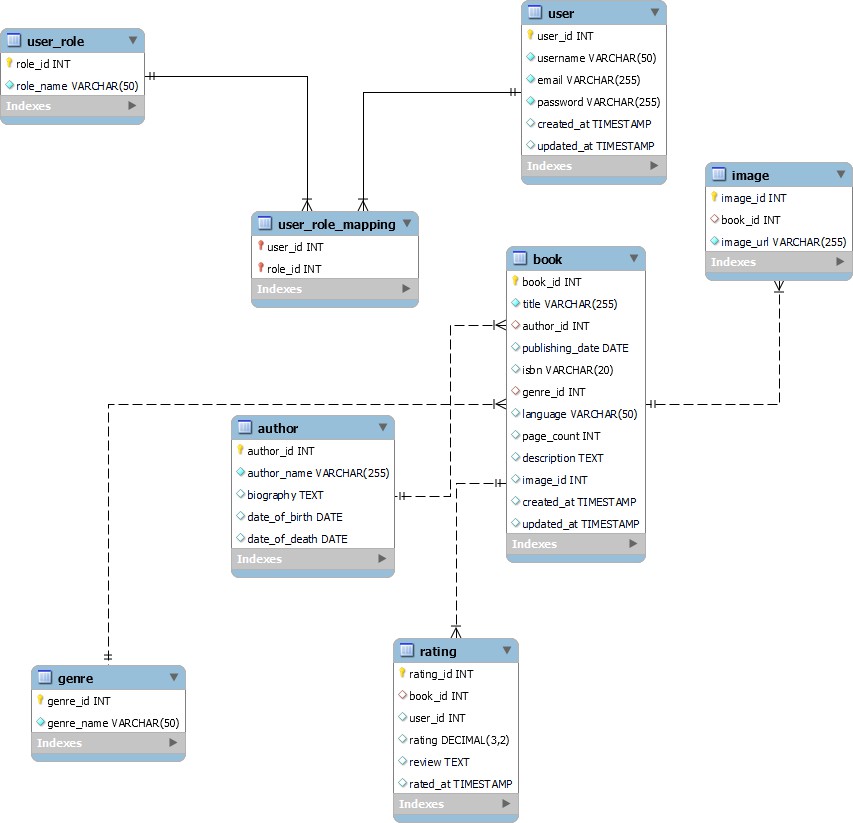
* Authors to Books: One-to-Many, Mandatory on the Books side.
* Genres to Books: One-to-Many, Mandatory on the Books side.
* Images to Books: One-to-Many, Mandatory on the Books side.
* Ratings to Books: One-to-Many, Mandatory on the Books side.
* Ratings to Users: Many-to-One, Mandatory on the Ratings side.
* Users to User Roles: Many-to-Many.

# ​Entity Relationship Diagram



* 1. **​Relational Model**

Convert the ER diagram to the relational model using the concepts learned in the class. List the various tables obtained.



# ​Normalization

1. First Normal Form (1NF):

All tables already have atomic values in each cell, so they satisfy 1NF.

1. Second Normal Form (2NF):

To achieve 2NF, there should be no partial dependencies.

All tables have primary keys, and there are no partial dependencies, so they satisfy 2NF.

1. Third Normal Form (3NF):

To achieve 3NF, there should be no transitive dependencies. We can identify transitive dependencies in the Books table:

author\_id depends on author\_name, which is not the primary key.

To resolve this, we need to create a separate Authors table and reference it using author\_id. This ensures that author\_id directly depends on the primary key of the Authors table.

Modified Tables:

Authors Table:

author\_id (Primary Key) author\_name

biography date\_of\_birth date\_of\_death Books Table:

book\_id (Primary Key) title

author\_id (Foreign Key) publishing\_date

isbn

genre\_id (Foreign Key) language

page\_count description

image\_id (Foreign Key) created\_at

updated\_at

1. Boyce-Codd Normal Form (BCNF):

All functional dependencies in the revised tables are already fully functional dependent on the primary key. Therefore, they satisfy BCNF.

The tables are now normalized up to BCNF.

# ​SQL Queries

Using a DBMS software (SQLite3 or MySQL or any other of your choice):

## Create the tables

-- Table for authors

CREATE TABLE IF NOT EXISTS author (

author\_id INT AUTO\_INCREMENT PRIMARY KEY,

author\_name VARCHAR(255) NOT NULL, biography TEXT,

date\_of\_birth DATE, date\_of\_death DATE,

UNIQUE KEY(author\_name) -- Ensures author names are unique

);

-- Table for genres

CREATE TABLE IF NOT EXISTS genre (

genre\_id INT AUTO\_INCREMENT PRIMARY KEY,

genre\_name VARCHAR(50) NOT NULL UNIQUE -- Ensures genre names are unique

);

-- Table for books

CREATE TABLE IF NOT EXISTS book (

book\_id INT AUTO\_INCREMENT PRIMARY KEY, title VARCHAR(255) NOT NULL,

author\_id INT, publishing\_date DATE, isbn VARCHAR(20),

genre\_id INT,

language VARCHAR(50), page\_count INT, description TEXT, image\_id INT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

FOREIGN KEY (author\_id) REFERENCES author(author\_id), FOREIGN KEY (genre\_id) REFERENCES genre(genre\_id)

);

-- Table for images

CREATE TABLE IF NOT EXISTS image (

image\_id INT AUTO\_INCREMENT PRIMARY KEY,

book\_id INT,

image\_url VARCHAR(255) NOT NULL,

FOREIGN KEY (book\_id) REFERENCES book(book\_id)

);

-- Table for ratings

CREATE TABLE IF NOT EXISTS rating (

rating\_id INT AUTO\_INCREMENT PRIMARY KEY,

book\_id INT, user\_id INT,

rating DECIMAL(3,2), review TEXT,

rated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (book\_id) REFERENCES book(book\_id)

);

-- Table for users

CREATE TABLE IF NOT EXISTS user (

user\_id INT AUTO\_INCREMENT PRIMARY KEY, username VARCHAR(50) NOT NULL,

email VARCHAR(255) NOT NULL UNIQUE, -- Ensures email addresses are unique password VARCHAR(255) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

);

-- Table for user roles

CREATE TABLE IF NOT EXISTS user\_role (

role\_id INT AUTO\_INCREMENT PRIMARY KEY,

role\_name VARCHAR(50) NOT NULL UNIQUE -- Ensures role names are unique

);

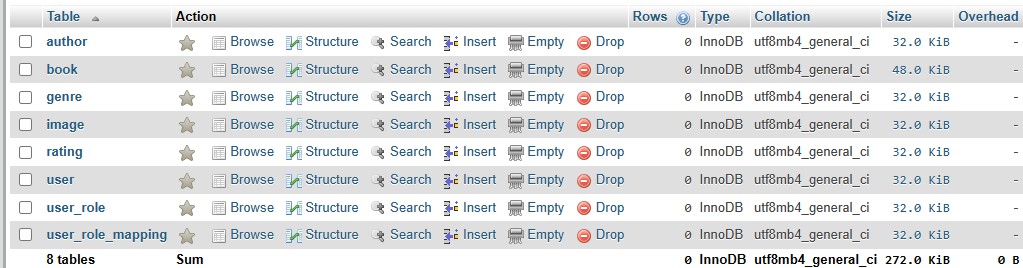
-- Table for mapping users to roles

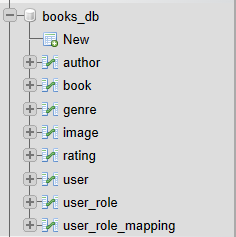
CREATE TABLE IF NOT EXISTS user\_role\_mapping ( user\_id INT,

role\_id INT,

FOREIGN KEY (user\_id) REFERENCES user(user\_id), FOREIGN KEY (role\_id) REFERENCES user\_role(role\_id), PRIMARY KEY (user\_id, role\_id)

);





## Populate the tables (insert some meaningful data, at least 10 tuples for each relation)

INSERT INTO author (author\_name, biography, date\_of\_birth, date\_of\_death) VALUES

('J.K. Rowling', 'British author known for the Harry Potter series', '19G5-07-31', NULL),

('Stephen King', 'American author known for horror and supernatural 1ction', '1947-09-21', NULL), ('Agatha Christie', 'English writer known for her detective novels', '1890-09-15', '197G-01-12'),

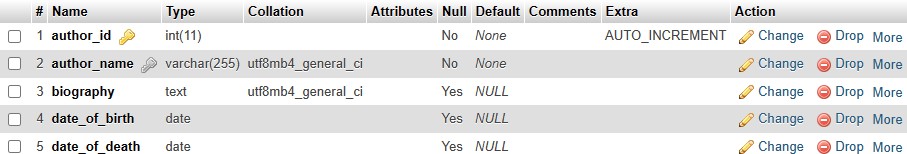
('George Orwell', 'English novelist and essayist known for Animal Farm and 1984', '1903-0G-25', '1950-01-21'), ('Haruki Murakami', 'Japanese author known for his surreal and imaginative storytelling', '1949-01-12', NULL), ('Jane Austen', 'English novelist known for her romantic 1ction', '1775-12-1G', '1817-07-18'),

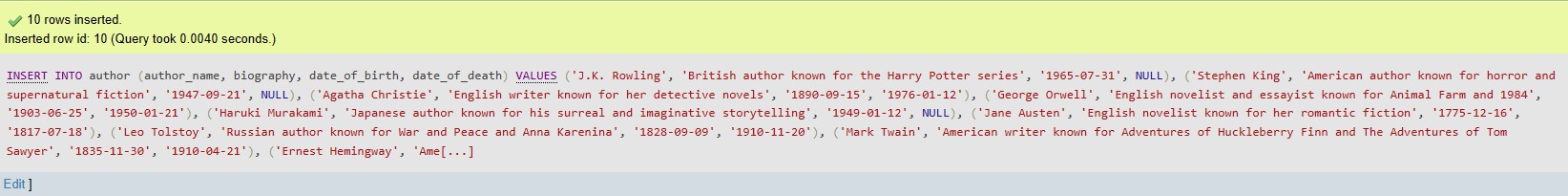
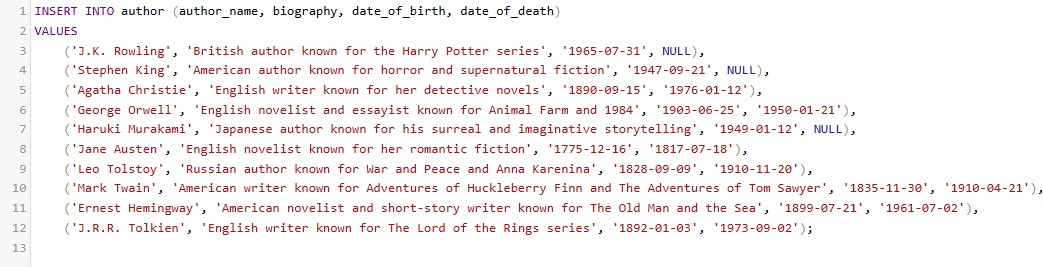
('Leo Tolstoy', 'Russian author known for War and Peace and Anna Karenina', '1828-09-09', '1910-11-20'),

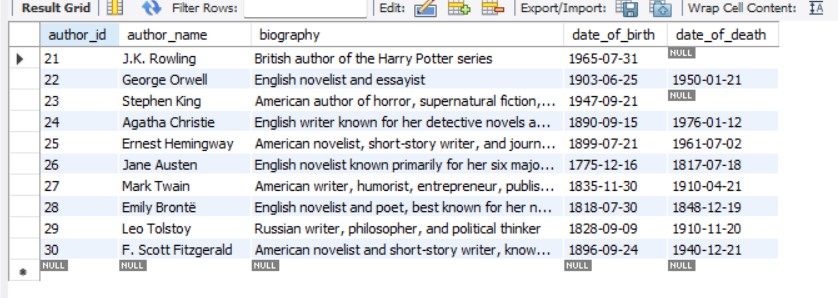
('Mark Twain', 'American writer known for Adventures of Huckleberry Finn and The Adventures of Tom Sawyer', '1835-11-30', '1910-04-21'),

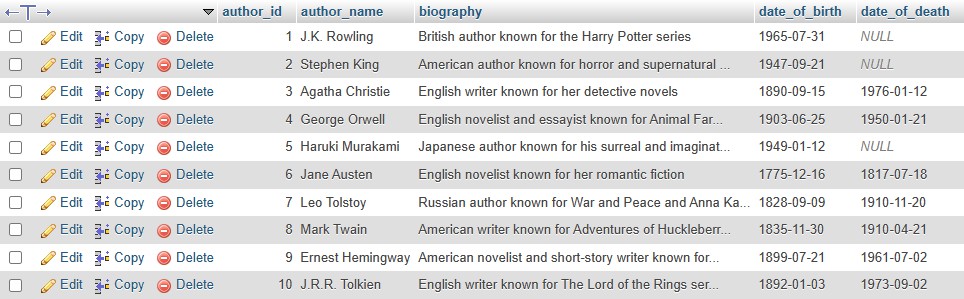
('Ernest Hemingway', 'American novelist and short-story writer known for The Old Man and the Sea', '1899-07-21', '19G1-07-02'),

('J.R.R. Tolkien', 'English writer known for The Lord of the Rings series', '1892-01-03', '1973-09-02');









INSERT INTO genre (genre\_name) VALUES

('Fantasy'),

('Horror'),

('Mystery'), ('Science Fiction'), ('Romance'),

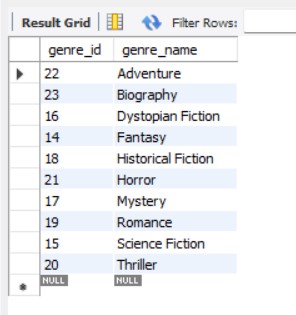
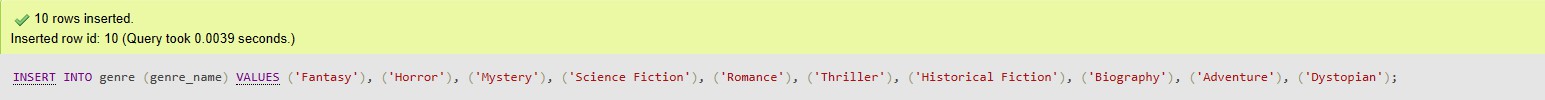
('Thriller'), ('Historical Fiction'), ('Biography'),

('Adventure'),

('Dystopian');







INSERT INTO book (title, author\_id, publishing\_date, isbn, genre\_id, language, page\_count, description, image\_id)

VALUES

('Harry Potter and the Philosopher''s Stone', 1, '1997-06-26', '978-0747532743', 1, 'English', 320, 'First book in the Harry Potter series', NULL),

('The Shining', 2, '1977-01-28', '978-0385121675', 2, 'English', 447, 'Horror novel about a haunted hotel', NULL),

('Murder on the Orient Express', 3, '1934-01-01', '978-0062693662', 3, 'English', 347, 'Classic detective fiction', NULL),

('1984', 4, '1949-06-08', '978-0451524935', 4, 'English', 328, 'Dystopian novel by George Orwell', NULL),

('Norwegian Wood', 5, '1987-08-20', '978-0375704024', 5, 'Japanese', 296, 'Novel set in 1960s Tokyo', NULL),

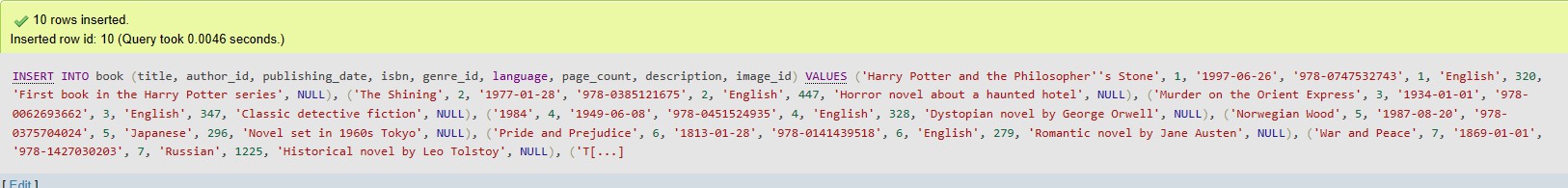
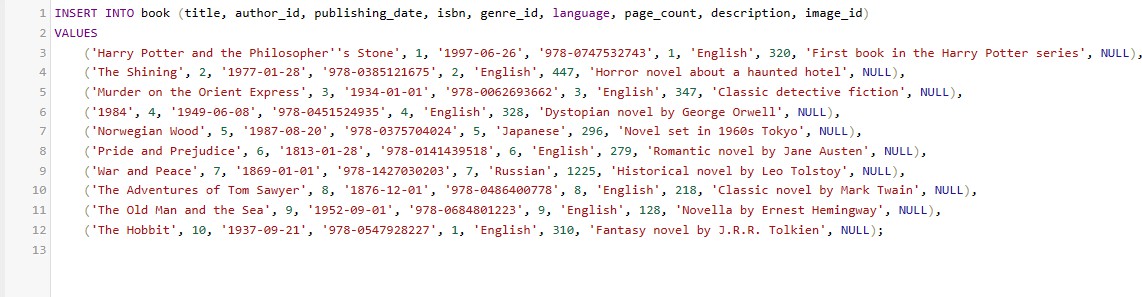
('Pride and Prejudice', 6, '1813-01-28', '978-0141439518', 6, 'English', 279, 'Romantic novel by Jane Austen', NULL),

('War and Peace', 7, '1869-01-01', '978-1427030203', 7, 'Russian', 1225, 'Historical novel by Leo Tolstoy', NULL),

('The Adventures of Tom Sawyer', 8, '1876-12-01', '978-0486400778', 8, 'English', 218, 'Classic novel by Mark Twain', NULL),

('The Old Man and the Sea', 9, '1952-09-01', '978-0684801223', 9, 'English', 128, 'Novella by Ernest Hemingway', NULL),

('The Hobbit', 10, '1937-09-21', '978-0547928227', 1, 'English', 310, 'Fantasy novel by J.R.R. Tolkien', NULL);



INSERT INTO image (book\_id, image\_url) VALUES

(1, 'https://example.com/harry\_potter.jpg'),

(2, 'https://example.com/the\_shining.jpg'),

(3, 'https://example.com/murder\_orient\_express.jpg'),

(4, 'https://example.com/1984.jpg'),

(5, 'https://example.com/norwegian\_wood.jpg'),

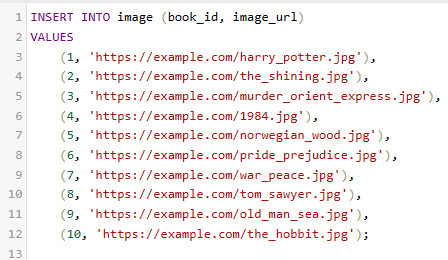
(6, 'https://example.com/pride\_prejudice.jpg'),

(7, 'https://example.com/war\_peace.jpg'),

(8, 'https://example.com/tom\_sawyer.jpg'),

(9, 'https://example.com/old\_man\_sea.jpg'),

(10, 'https://example.com/the\_hobbit.jpg');



-- Assuming user\_id and rating values are available INSERT INTO rating (book\_id, user\_id, rating, review) VALUES

(1, 1, 4.5, 'Amazing book, loved the characters and the magical world!'), (2, 2, 4.0, 'A thrilling and suspenseful read.'),

(3, 3, 4.8, 'Classic mystery with a brilliant twist.'),

(4, 4, 4.7, 'Terrifyingly relevant dystopian novel.'),

(5, 5, 4.2, 'Beautifully written story with deep themes.'),

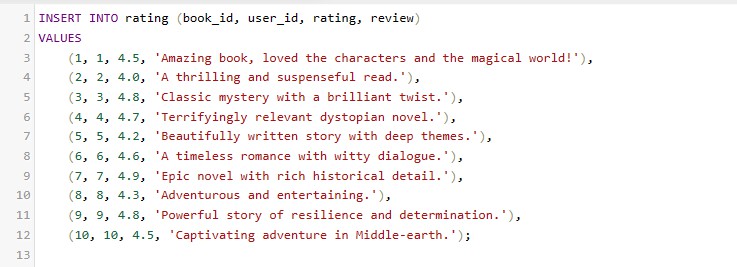
(6, 6, 4.6, 'A timeless romance with witty dialogue.'),

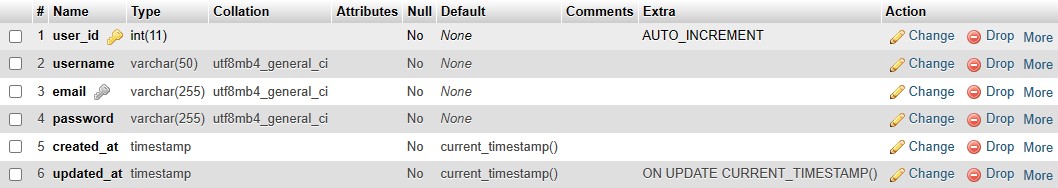
(7, 7, 4.9, 'Epic novel with rich historical detail.'),

(8, 8, 4.3, 'Adventurous and entertaining.'),

(9, 9, 4.8, 'Powerful story of resilience and determination.'),

(10, 10, 4.5, 'Captivating adventure in Middle-earth.');



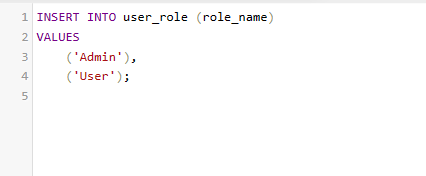


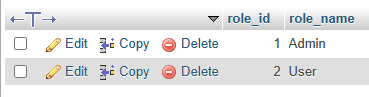




INSERT INTO user\_role (role\_name) VALUES ('Admin'),

('User');





INSERT INTO user\_role\_mapping (user\_id, role\_id) VALUES

(1, 1),

(2, 2),

(3, 2),

(4, 2),

(5, 2),

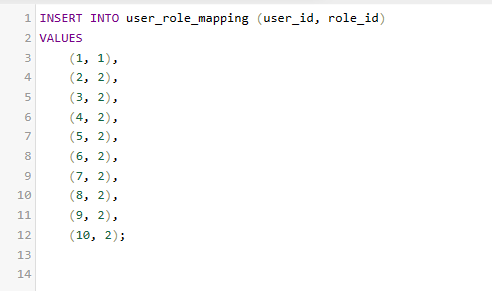
(6, 2),

(7, 2),

(8, 2),

(9, 2),

(10, 2);





## Run SQL queries (minimum 20) covering all concepts learned in the class

This section should contain the question, SQL code, and the output snapshot for each query.

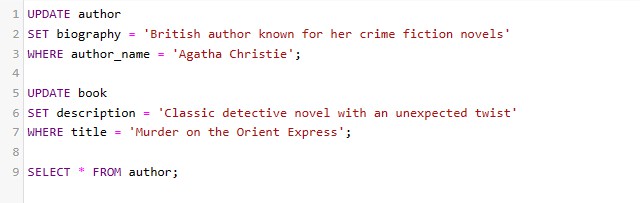
## DML-UPDATE DATA

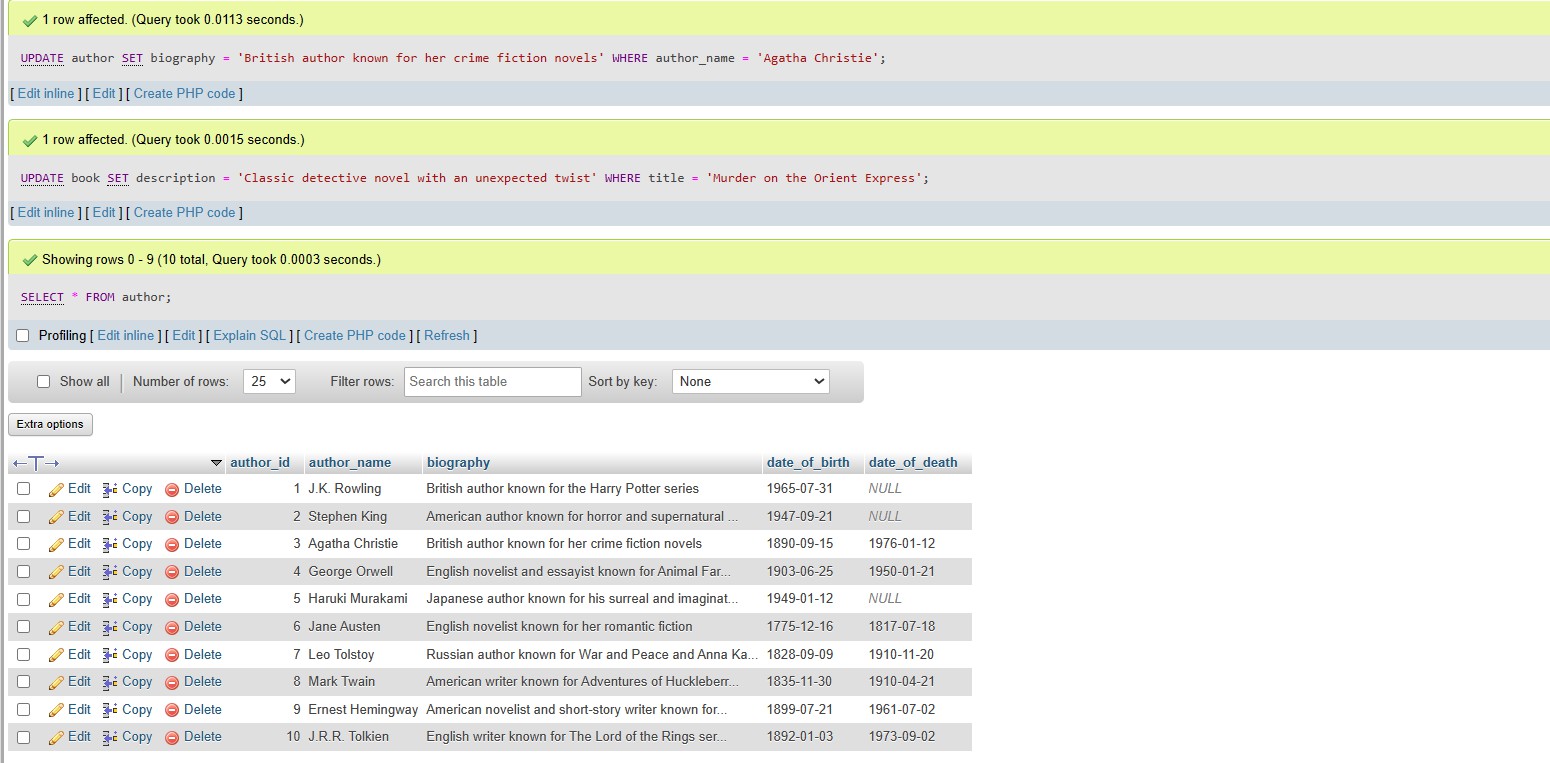
UPDATE author

SET biography = 'British author known for her crime fiction novels' WHERE author\_name = 'Agatha Christie';

UPDATE book

SET description = 'Classic detective novel with an unexpected twist' WHERE title = 'Murder on the Orient Express';

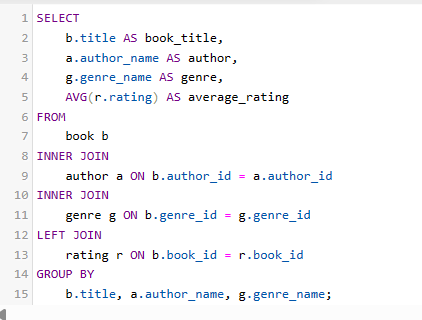


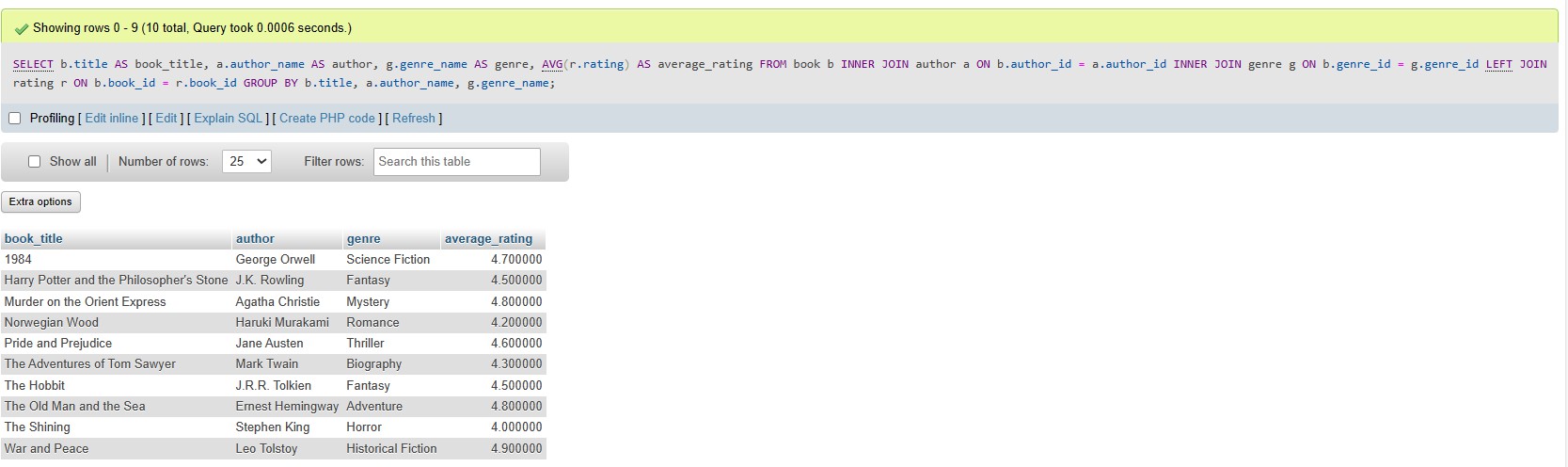


## Find books with their authors, genres, and average ratings:

SELECT b.title, a.author\_name, g.genre\_name, AVG(r.rating) AS average\_rating FROM book b

JOIN author a ON b.author\_id = a.author\_id JOIN genre g ON b.genre\_id = g.genre\_id LEFT JOIN rating r ON b.book\_id = r.book\_id GROUP BY b.book\_id;

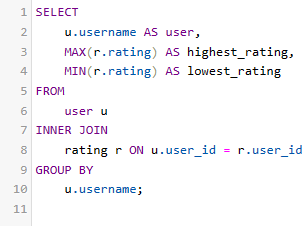


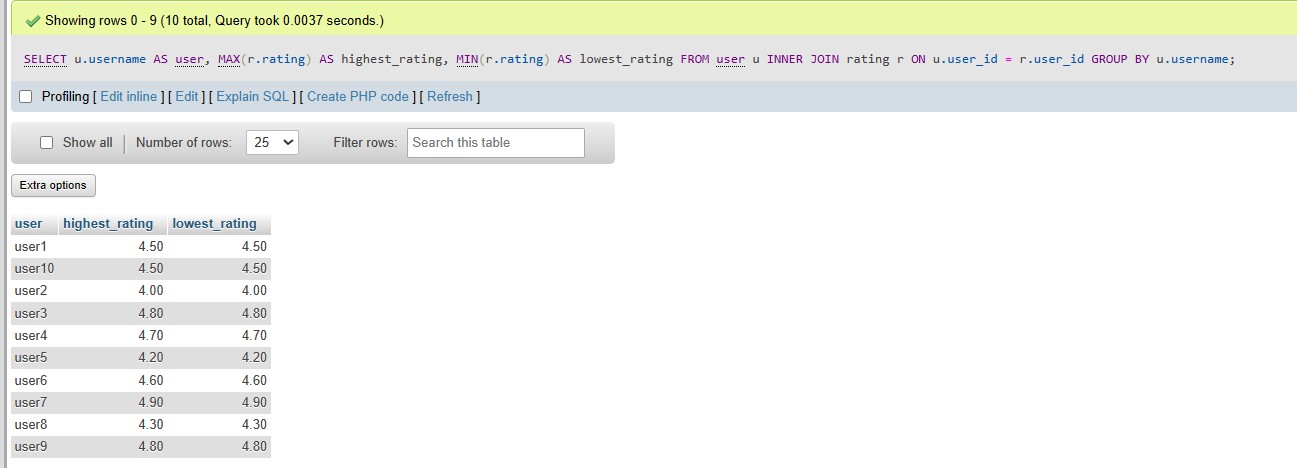


## Find users who have rated books with their highest and lowest ratings:

SELECT u.username, MAX(r.rating) AS highest\_rating, MIN(r.rating) AS lowest\_rating FROM user u

JOIN rating r ON u.user\_id = r.user\_id GROUP BY u.user\_id;





## To retrieve the user with the highest and with the lowest rating

SELECT

u.username AS user, MAX(r.rating) AS highest\_rating, MIN(r.rating) AS lowest\_rating

FROM

user u INNER JOIN

rating r ON u.user\_id = r.user\_id GROUP BY

u.username HAVING

MAX(r.rating) = (

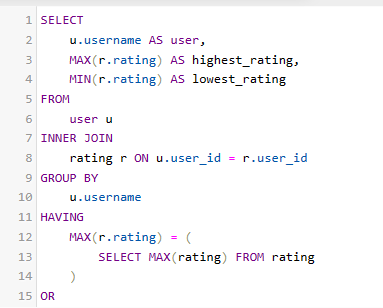
SELECT MAX(rating) FROM rating

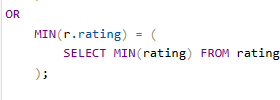
) OR

MIN(r.rating) = (

SELECT MIN(rating) FROM rating

);

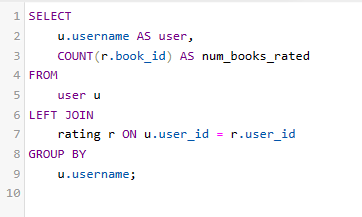




## Count the number of books each user has rated:

SELECT u.username, COUNT(r.book\_id) AS rated\_books\_count FROM user u

LEFT JOIN rating r ON u.user\_id = r.user\_id GROUP BY u.user\_id;





## Find books with no ratings yet:

SELECT

b.title AS book\_title, b.author\_id AS author\_id, a.author\_name AS author\_name, b.genre\_id AS genre\_id, g.genre\_name AS genre\_name

FROM

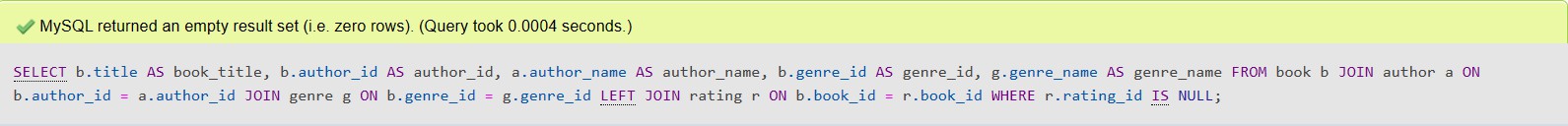
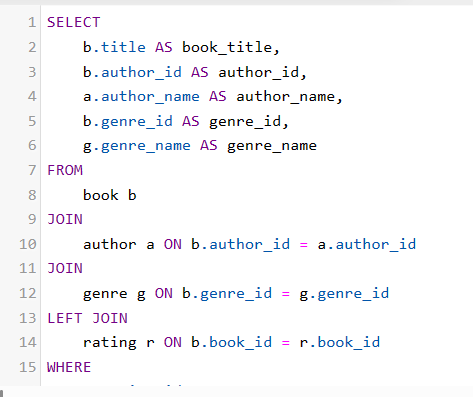
book b JOIN

author a ON b.author\_id = a.author\_id JOIN

genre g ON b.genre\_id = g.genre\_id LEFT JOIN

rating r ON b.book\_id = r.book\_id WHERE

r.rating\_id IS NULL;



## Find the top-rated book in each genre:

SELECT

b.title AS book\_title, a.author\_name AS author\_name, g.genre\_name AS genre\_name, r.rating AS max\_rating

FROM

book b JOIN

author a ON b.author\_id = a.author\_id JOIN

genre g ON b.genre\_id = g.genre\_id JOIN

(

SELECT

genre\_id,

MAX(rating) AS max\_rating FROM

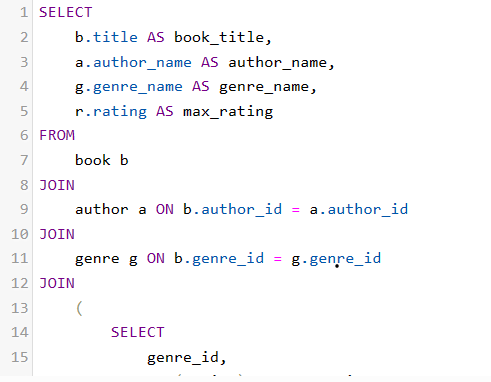
book JOIN

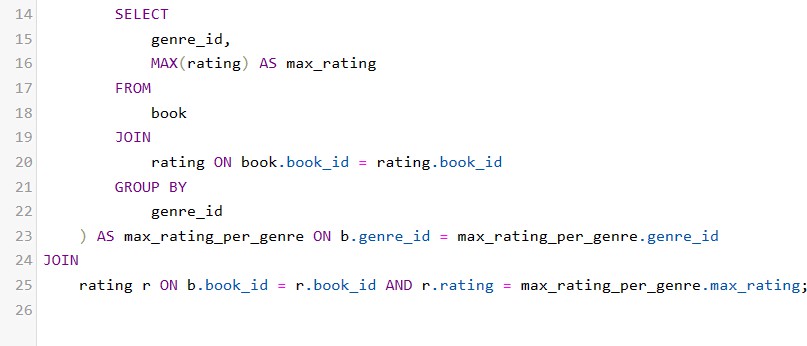
rating ON book.book\_id = rating.book\_id GROUP BY

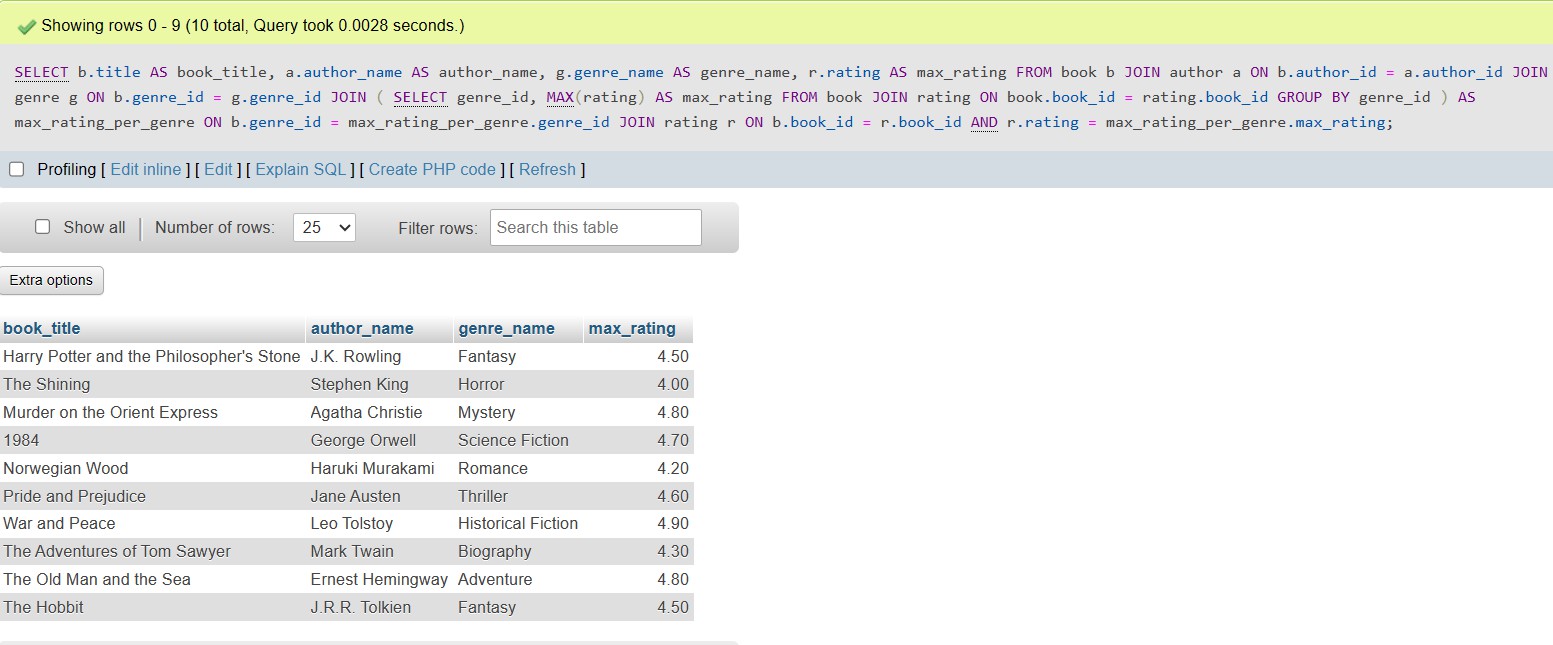
genre\_id

) AS max\_rating\_per\_genre ON b.genre\_id = max\_rating\_per\_genre.genre\_id JOIN

rating r ON b.book\_id = r.book\_id AND r.rating = max\_rating\_per\_genre.max\_rating;







## Find books with similar genres based on a keyword search:

SELECT

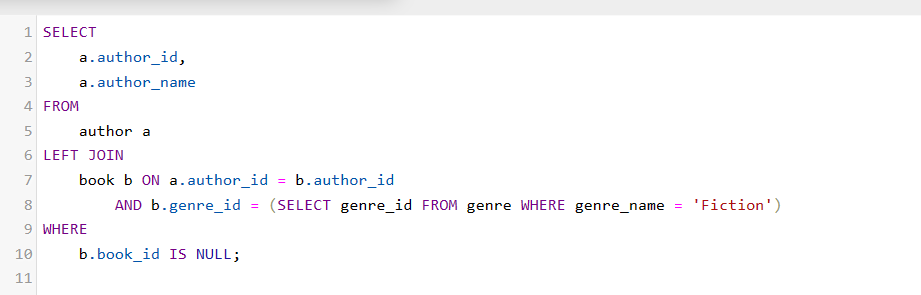
a.author\_id, a.author\_name

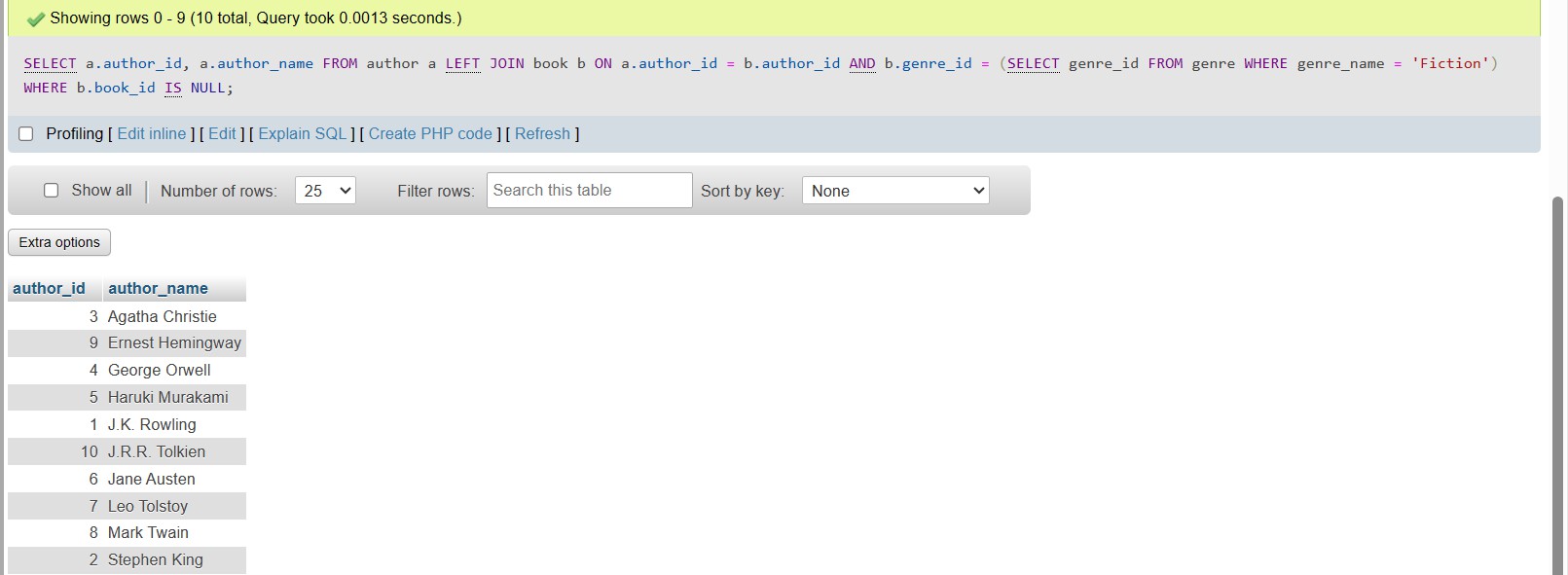
FROM

author a LEFT JOIN

book b ON a.author\_id = b.author\_id

AND b.genre\_id = (SELECT genre\_id FROM genre WHERE genre\_name = 'Fiction') WHERE

b.book\_id IS NULL;



## Find books and their total ratings count ordered by the count in descending order:

SELECT

b.title AS Book\_Title,

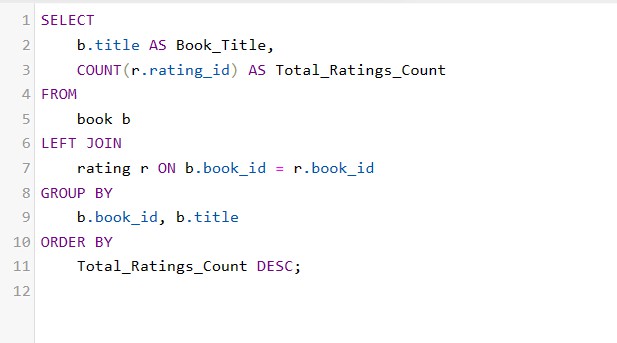
COUNT(r.rating\_id) AS Total\_Ratings\_Count FROM

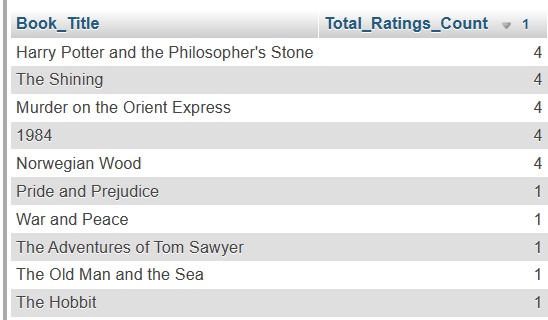
book b LEFT JOIN

rating r ON b.book\_id = r.book\_id GROUP BY

b.book\_id, b.title ORDER BY

Total\_Ratings\_Count DESC;

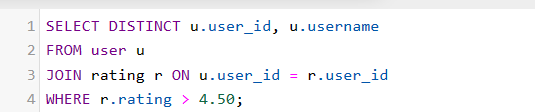


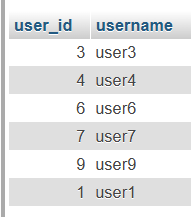
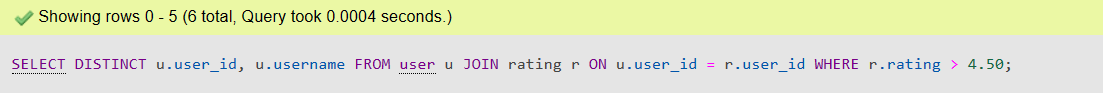


## Find users who have rated books with ratings greater than a specific value:

SELECT DISTINCT u.user\_id, u.username FROM user u

JOIN rating r ON u.user\_id = r.user\_id WHERE r.rating > 4.50;





## Insert a new book and its associated image within a transaction:

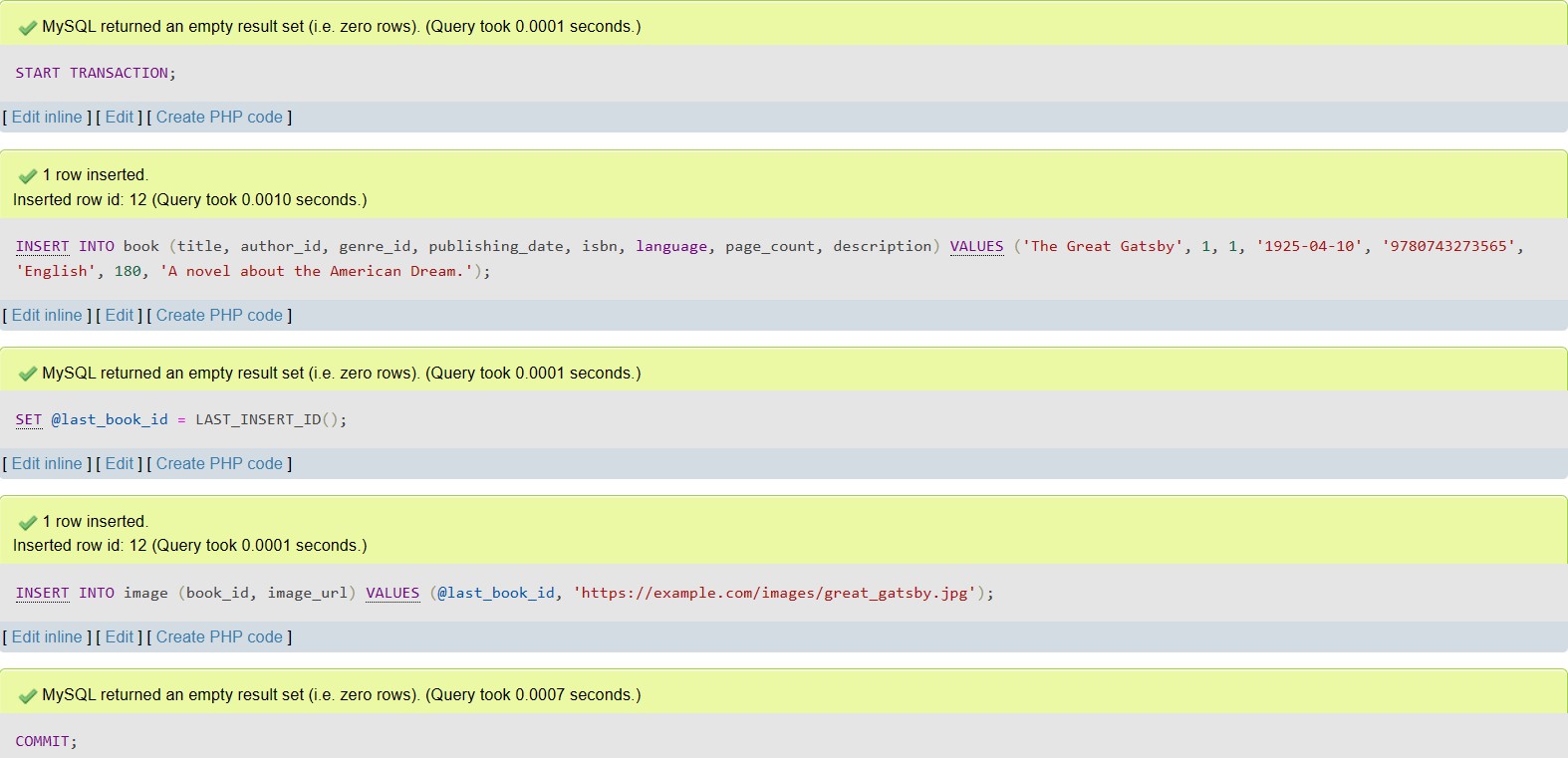
START TRANSACTION;

INSERT INTO book (title, author\_id, genre\_id, publishing\_date, isbn, language, page\_count, description) VALUES ('The Great Gatsby', 1, 1, '1925-04-10', '9780743273565', 'English', 180, 'A novel about the American Dream.');

SET @last\_book\_id = LAST\_INSERT\_ID(); INSERT INTO image (book\_id, image\_url)

VALUES (@last\_book\_id, 'https://example.com/images/great\_gatsby.jpg'); COMMIT;





## Transaction to update author's biography and book's publishing date with a commit

START TRANSACTION;

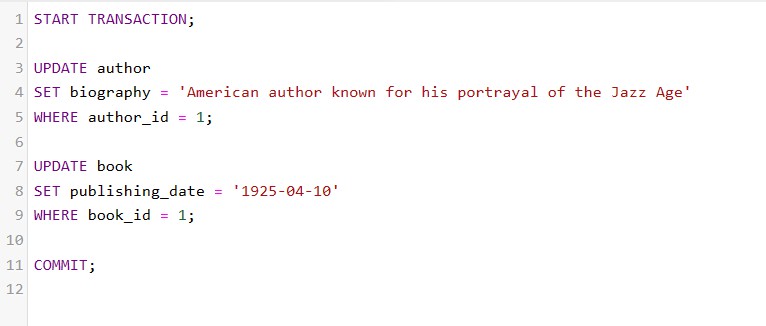
UPDATE author

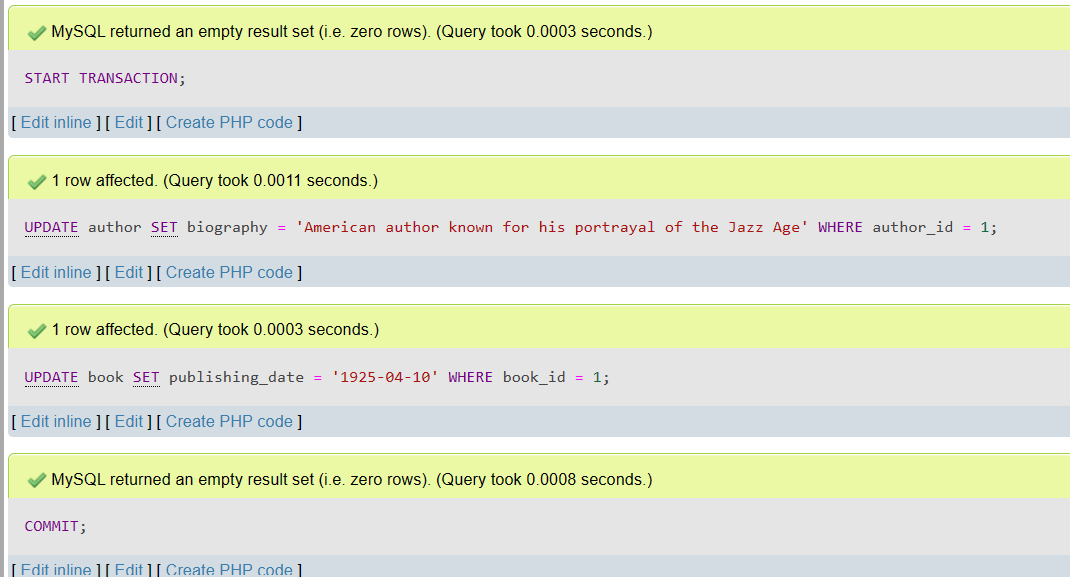
SET biography = 'American author known for his portrayal of the Jazz Age' WHERE author\_id = 1;

UPDATE book

SET publishing\_date = '1925-04-10' WHERE book\_id = 1;

COMMIT;





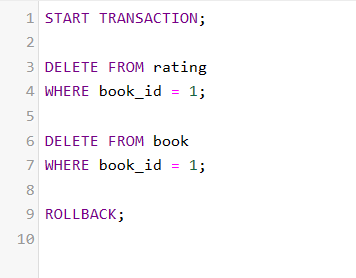
## Transaction to delete a book and its associated ratings with a rollback:

START TRANSACTION;

DELETE FROM rating WHERE book\_id = 1;

DELETE FROM book

WHERE book\_id = 1; ROLLBACK;



**Find books with ratings higher than the average rating for their genre:** SELECT b.title, b.author\_id, a.author\_name, b.genre\_id, g.genre\_name, r.rating FROM book b

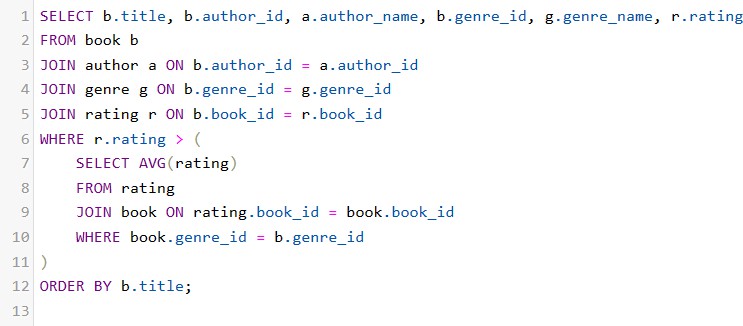
JOIN author a ON b.author\_id = a.author\_id JOIN genre g ON b.genre\_id = g.genre\_id JOIN rating r ON b.book\_id = r.book\_id WHERE r.rating > (

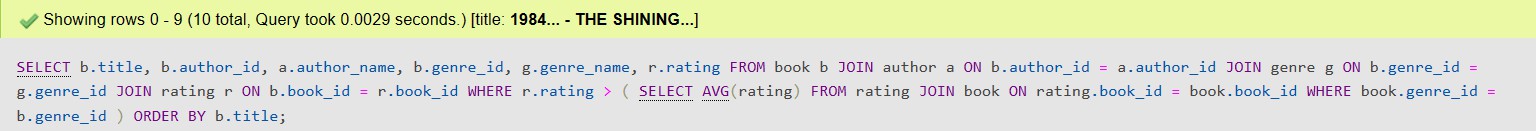
SELECT AVG(rating) FROM rating

JOIN book ON rating.book\_id = book.book\_id WHERE book.genre\_id = b.genre\_id

)

ORDER BY b.title;





## Find books with the highest rating in each genre:

SELECT b.title, b.author\_id, a.author\_name, b.genre\_id, g.genre\_name, r.rating FROM (

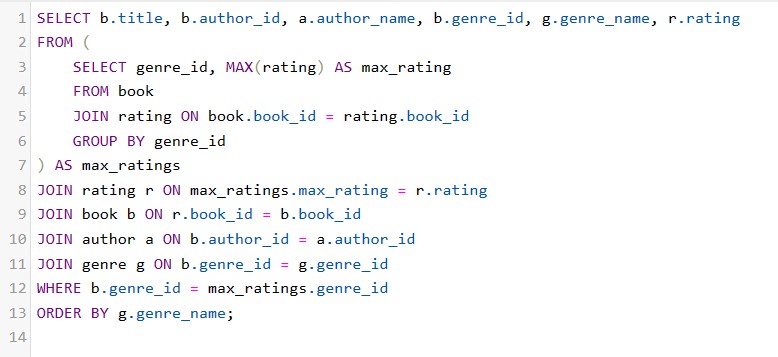
SELECT genre\_id, MAX(rating) AS max\_rating FROM book

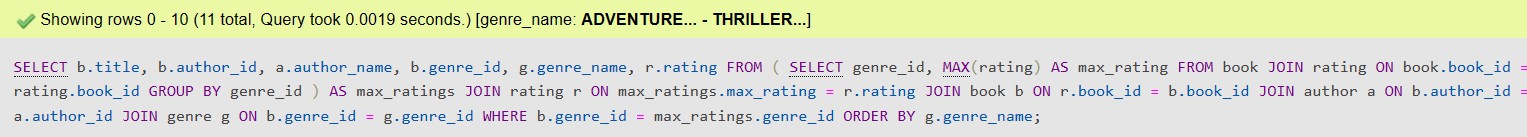
JOIN rating ON book.book\_id = rating.book\_id GROUP BY genre\_id

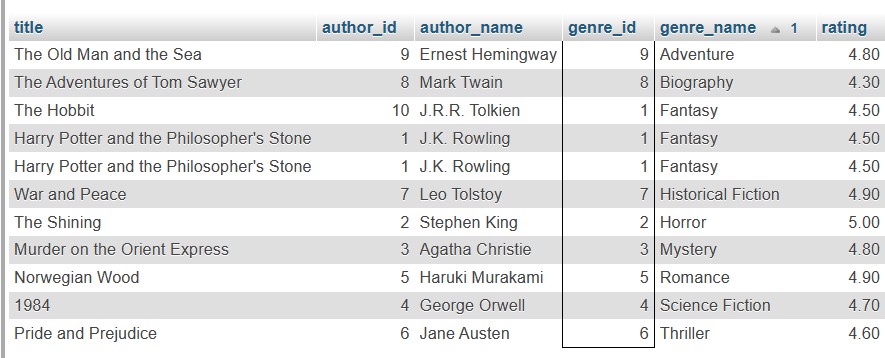
) AS max\_ratings

JOIN rating r ON max\_ratings.max\_rating = r.rating JOIN book b ON r.book\_id = b.book\_id

JOIN author a ON b.author\_id = a.author\_id JOIN genre g ON b.genre\_id = g.genre\_id WHERE b.genre\_id = max\_ratings.genre\_id ORDER BY g.genre\_name;







## DDL Commands

ALTER TABLE book

ADD COLUMN IF NOT EXISTS new\_page\_count INT;

UPDATE book

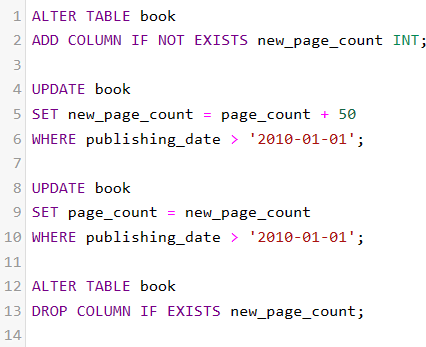
SET new\_page\_count = page\_count + 50 WHERE publishing\_date > '2010-01-01';

UPDATE book

SET page\_count = new\_page\_count WHERE publishing\_date > '2010-01-01';

ALTER TABLE book

DROP COLUMN IF EXISTS new\_page\_count;





## PROJECT SQL Queries

**private void LoginBtnActionPerformed(java.awt.event.ActionEvent evt) {**

String Email, Password, query, passDb=null, fname=null; String SUrl, SUser, SPass;

SUrl="jdbc:MySQL://localhost:3306/java\_user\_database"; SUser="root";

SPass="";

int notFound = 0; try{

Class.forName("com.mysql.cj.jdbc.Driver");

Connection con = DriverManager.getConnection(SUrl, SUser, SPass); Statement st = con.createStatement();

if("".equals(email.getText())){

JOptionPane.showMessageDialog(new JFrame(), "Email ID is required", "Error", JOptionPane.ERROR\_MESSAGE);

}else if("".equals(password.getText())){

JOptionPane.showMessageDialog(new JFrame(), "Password is required", "Error", JOptionPane.ERROR\_MESSAGE);

}else{

Email = email.getText(); Password = password.getText();

**query = "SELECT \* FROM user WHERE email='"+Email+"'"; java.sql.ResultSet rs = st.executeQuery(query);**

while(rs.next()){

passDb = rs.getString("password"); fname = rs.getString("full\_name"); notFound=1;

}

**private void LoginBtnActionPerformed(java.awt.event.ActionEvent evt) {**

// TODO add your handling code here:

String Email, Password, query, passDb=null, fname=null; String SUrl, SUser, SPass;

SUrl="jdbc:MySQL://localhost:3306/java\_user\_database"; SUser="root";

SPass="";

int notFound = 0; try{

Class.forName("com.mysql.cj.jdbc.Driver");

Connection con = DriverManager.getConnection(SUrl, SUser, SPass); Statement st = con.createStatement();

if("".equals(email.getText())){

JOptionPane.showMessageDialog(new JFrame(), "Email ID is required", "Error", JOptionPane.ERROR\_MESSAGE);

}else if("".equals(password.getText())){

JOptionPane.showMessageDialog(new JFrame(), "Password is required", "Error", JOptionPane.ERROR\_MESSAGE);

}else{

Email = email.getText(); Password = password.getText();

**query = "SELECT \* FROM user WHERE email='"+Email+"'";**

java.sql.ResultSet rs = st.executeQuery(query); while(rs.next()){

passDb = rs.getString("password"); fname = rs.getString("full\_name"); notFound=1;

}

**private void addActionPerformed(java.awt.event.ActionEvent evt) {**

String bn, aut, rev,query;

//connect to database try{

Class .forName("com.mysql.cj.jdbc.Driver");

String url = "jdbc:MySQL://localhost:3306/java\_user\_database"; String user = "root";

String pass = "";

Connection con = DriverManager.getConnection(url, user, pass); Statement st = con.createStatement();

if("".equals(bookname.getText())){

JOptionPane.showMessageDialog(new JFrame(), "Book name is required", "dialog", JOptionPane.ERROR\_MESSAGE);

}else if("".equals(author.getText())){

JOptionPane.showMessageDialog(new JFrame(), "Author name is required", "dialog", JOptionPane.ERROR\_MESSAGE);

}else if("".equals(review.getText())){

JOptionPane.showMessageDialog(new JFrame(), "Review is required", "dialog", JOptionPane.ERROR\_MESSAGE);

}else{

bn = bookname.getText(); aut = author.getText(); rev = review.getText();

query = "INSERT INTO review (book\_name, author, review)" + " VALUES ('"+bn+"','"+aut+"','"+rev+"')";

st.executeUpdate(query); bookname.setText(""); author.setText("");

review.setText("");

showMessageDialog(null, "Successfully Registered"); loadData();

con.close();

}

}catch(Exception e){ System.out.println("Error" + e.getMessage());

}

**private void updateActionPerformed(java.awt.event.ActionEvent evt) {**

String ID;

int notFound = 0;

String bn, aut = null, rev; try{

Class .forName("com.mysql.cj.jdbc.Driver");

String url = "jdbc:MySQL://localhost:3306/java\_user\_database"; String user = "root";

String pass = "";

Connection con = DriverManager.getConnection(url, user, pass); Statement st = con.createStatement();

ID = searchData.getText();

if("".equals(ID)){

JOptionPane.showMessageDialog(new JFrame(), "ID is required", "dialog", JOptionPane.ERROR\_MESSAGE);

}else{

**String sql = "SELECT \* FROM review WHERE id="+ID;**

ResultSet rs = st.executeQuery(sql); while(rs.next()){

notFound=1;

bn = bookname.getText(); aut = author.getText(); rev = review.getText();

**String sql2 = "UPDATE review SET book\_name='"+bn+"', author='"+aut+"', review='"+rev+"' WHERE id="+ID;**

st.executeUpdate(sql2); loadData(); con.close();

}

if(notFound==0){

JOptionPane.showMessageDialog(new JFrame(), "invalid ID", "dialog", JOptionPane.ERROR\_MESSAGE);

}

}

}catch(Exception e){

System.out.println("Error" + e.getMessage());

}

}

# ​Self -Learning beyond classroom

:

VII. Self-Learning beyond the classroom:

During the project, we collectively explored various aspects beyond the classroom curriculum to enhance our skills and understanding. Some of the new aspects we learned on our own include:

Advanced SQL queries and database design: We delved into more complex SQL queries and learned about database normalization techniques to ensure efficient database design.

Foreign key constraints: We gained a deeper understanding of foreign key constraints in databases and how they enforce referential integrity between tables.

Data modeling: We learned about conceptualizing and designing database schemas to represent real-world entities and their relationships effectively.

Error handling: We explored methods for handling errors and troubleshooting issues that arise during database operations.

# ​Learning from the Project

This project provided us with valuable learning experiences that contributed to our overall growth and understanding. Some of the key learnings from the project include:

Practical application of database concepts: We gained hands-on experience in applying database concepts learned in class to a real-world project scenario.

Collaboration and teamwork: Working on the project in a team environment helped us improve our collaboration skills and learn from our peers' perspectives.

Problem-solving skills: We encountered various challenges during the project, which required us to think critically and devise solutions to overcome them.

Project management: Managing the project timeline, tasks, and deliverables helped us develop organizational and time management skills.

# ​Challenges Faced

IX. Challenges Faced:

Throughout the project, we faced several challenges that tested our problem-solving abilities and resilience. Some of the challenges we encountered include:

Database design complexities: Designing a comprehensive database schema that adequately represented all project requirements while maintaining efficiency posed a significant challenge. Foreign key constraints errors: Resolving errors related to foreign key constraints and ensuring proper referencing between tables required careful attention to detail.

Time constraints: Balancing project tasks with other academic and personal commitments was challenging at times, necessitating effective time management strategies.

Communication issues: Coordinating with team members and ensuring clear communication regarding project tasks and updates occasionally presented challenges.

# ​Conclusion

Overall, this project was a valuable learning experience that allowed us to apply theoretical concepts to real-world scenarios and develop practical skills. Through overcoming challenges and working collaboratively as a team, we gained insights into database management, teamwork, and problem-solving. The project helped us solidify our understanding of database design principles, SQL queries, and project management practices, which will be beneficial in our academic and professional endeavors.