SOCIAL LIBRARY DATABASE

BATOUL HAIDAR

19/06/2024

—

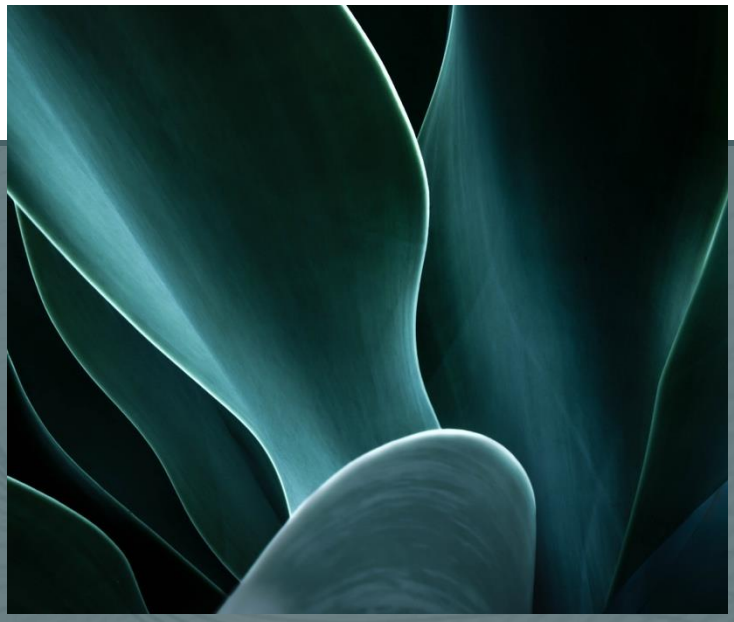
DATABASE

INSTRUCTIONS

The Social Library Database is a comprehensive and well-structured database designed to support a book-centric social networking platform. This database includes detailed records of books, user profiles, reviews, and social interactions.

- Populate the database with book information including titles, authors, genres, and publication details.
- Store user profiles with relevant information such as reading preferences and personal libraries.
- Manage user-generated content such as book reviews, ratings, and recommendations.

Utilize the Social Library Database to provide a rich and engaging experience for book enthusiasts



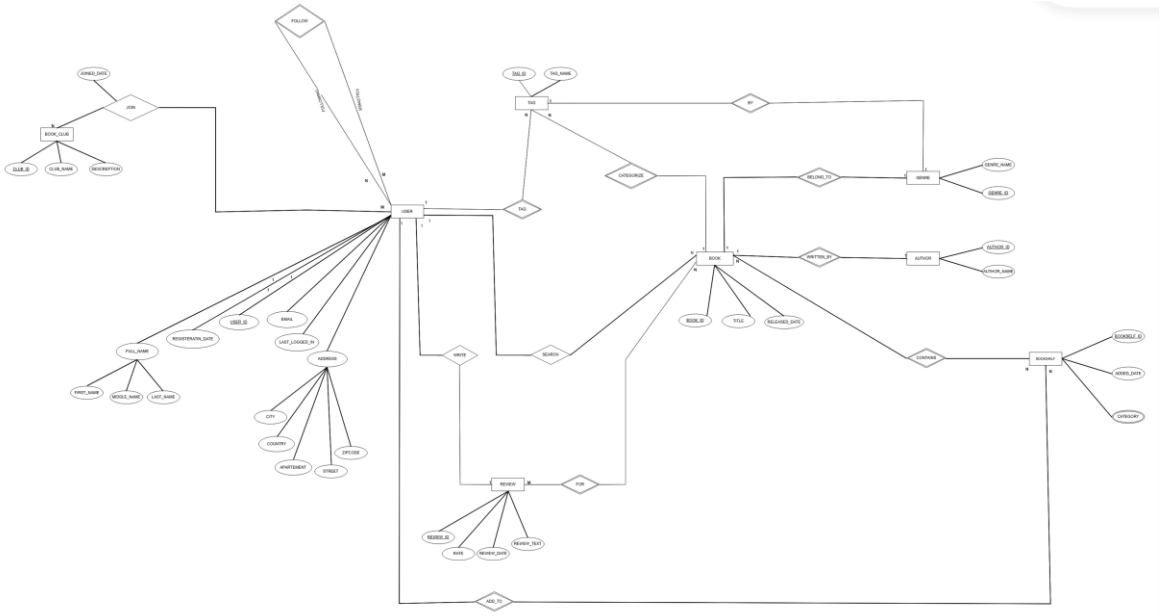
THE PROCESS

INTRODUCTION

The development of the Social Library Database involves several critical steps to ensure a robust and efficient platform. This comprehensive approach guarantees that the database will effectively support a book-centric social networking platform, providing users with a seamless and enriching experience. The process includes meticulous planning, design, and implementation phases to cover all aspects of the database requirements.

Below is a detailed description of each step in the process.

1. Design Entity-Relationship Diagram Using Chen's Notation



Entity/ Relationship Type	Attributes	Description /Requirements
USERS	• USER_ID, • FULL_NAME, • ADDRESS, EMAIL, • RIGESTERATION_DATE, • LAST_LOGIN	Users register with their full name, address, and email. Track registration date and last login date.
GENRE	• GENRE_ID • GENRE_NAME	Each book belongs to a genre. Users can filter books by genre.
AUTHOR	• AUTHOR_ID, • AUTHOR_NAME	Authors of books.
BOOK	• BOOK_ID, • GENRE_ID, • AUTHOR_ID, • TITLE, • RELEASED_DATE	Books have a title, genre, author, and release date
BOOKSHELF	• BOOKSHELF_ID, • USER_ID, • BOOK_ID, • CATEGORY_ID, • ADDED_DATE	Users add books to their virtual bookshelves with categories “Read”, “Want to Read”, “Currently Reading”
BOOK_CLUB	• CLUB_ID, • USER_ID, • CLUB_NAME, • JOINED_DATE, • DESCRIPTION	Users can join book clubs.
TAG	• TAG_ID, • USER_ID, • BOOK_ID, • GENRE_ID, • TAG_NAME	Users can tag books with descriptive tags.
REVIEW	• REVIEW_ID, • USER_ID, • BOOK_ID, RATE, • REVIEW_TEXT, • REVIEW_DATE	Users write reviews and rate books they have read.
FOLLOW	• FOLLOWER_USER_ID, • FOLLOWING_USER_ID	Users can follow other users.

Chen's ERD Notation provides a clear structure for database design, ensuring efficient data management and robust entity relationships.

2. Mapping ER Diagram into a relational model.

This section details the process of mapping the ER diagram into a relational model.

Step 1 : Mapping of Regular Entity Types:

Entity	Primary Key	Foreign Key
USERS	• USER_ID	-
GENRE	• GENRE_ID	-
AUTHOR	• AUTHOR_ID	-
BOOK	• BOOK_ID	• GENRE_ID • AUTHOR_ID
BOOKSHELF	• BOOKSHELF_ID	• USER_ID • BOOK_ID
BOOK_CLUB	• CLUB_ID	• USER_ID

Step 2 : Mapping of Weak Entity Types:
No weak Entities in my diagram.

Step 3 : Mapping of Binary 1:1 Relation Types:

Relationship	Primary Key	Foreign Key
REVIEW	• REVIEW_ID	• USER_ID • BOOK_ID

Step 4 : Mapping of Binary 1:N Relationship Types:

Relationship	Primary Key	Foreign Key
TAG	• TAG_ID	• USER_ID • BOOK_ID • GENRE_ID

Step 5 : Mapping of Binary M:N Relationship Types:

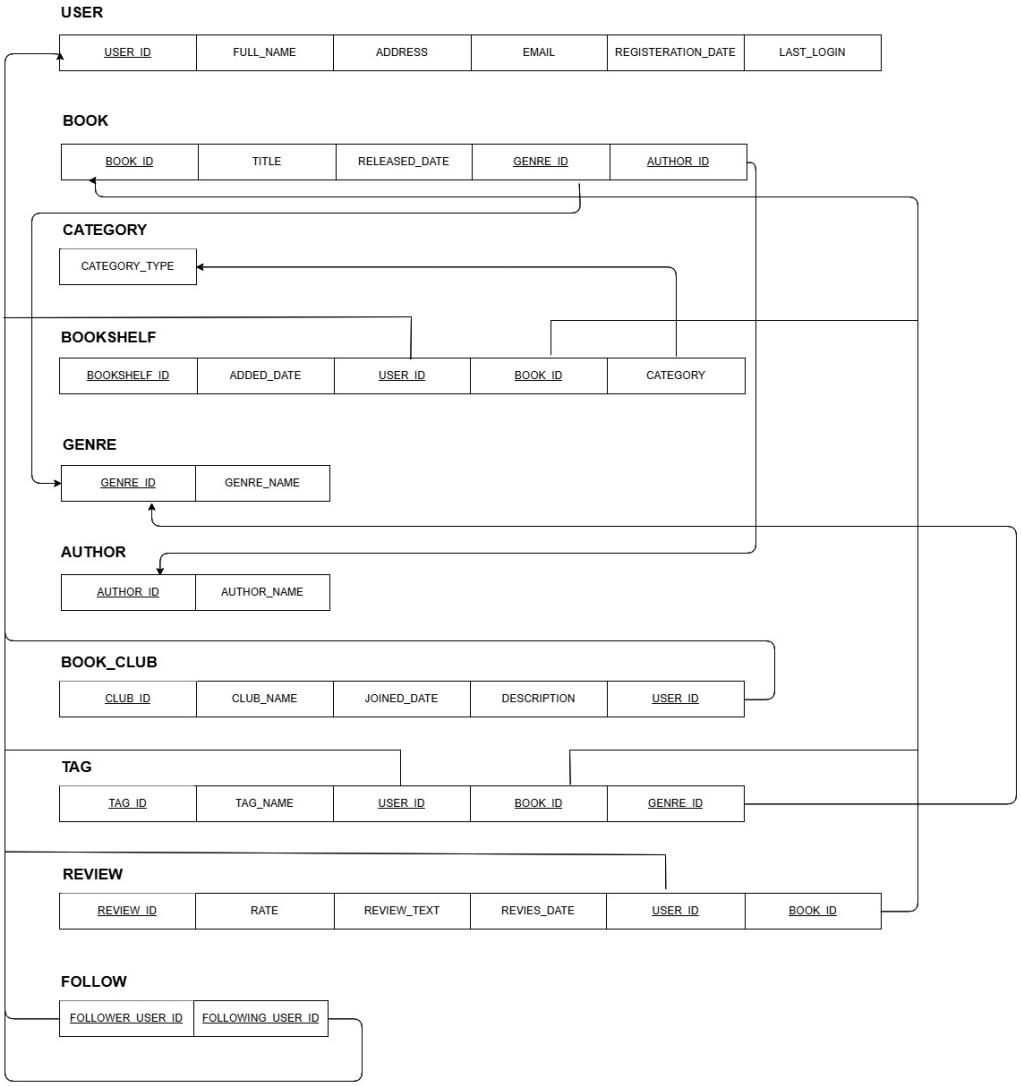
Relationship	Primary Key	Foreign Key
FOLLOW	-	• FOLLOWER_USER_ID • FOLLOWING_USER_ID

Step 6 : Mapping of Multivalued attributes:

Relationship	ATTRIBTE	Primary Key	Foreign Key
CATEGORY	• CATEGORY_TPE	• CATEGORY_ID	-

Step 7 : Mapping of N-ary Relationship Types:
No N-ary Relationship Types.

3. The Relational Model.



4. SQL SCHEMA.

```
DROP database SOCIAL_LIBRARY;
CREATE DATABASE SOCIAL_LIBRARY;
USE SOCIAL_LIBRARY;

CREATE TABLE USERS (
ID INT auto_increment,
FULL_NAME VARCHAR (100) NOT NULL,
ADDRESS VARCHAR (100) NOT NULL,
EMAIL VARCHAR (50) NOT NULL,
RIGESTERATION_DATE DATE,
LAST_LOGIN DATE,
primary key (ID),
CHECK (RIGESTERATION_DATE <= LAST_LOGIN)
);


CREATE TABLE GENRE (
ID INT auto_increment,
GENRE_NAME VARCHAR (50),
primary key (ID)
);

CREATE TABLE AUTHOR (
ID INT auto_increment,
AUTHOR_NAME VARCHAR (100),
primary key(ID)
);

CREATE TABLE CATEGORY (
ID INT auto_increment,
CATEGORY_TYPE VARCHAR (50),
primary key (ID)
);

CREATE TABLE BOOK (
ID INT auto_increment,
GENRE_ID INT NOT NULL,
AUTHOR_ID INT NOT NULL,
TITLE VARCHAR (100) NOT NULL,
RELEASED_DATE DATE,
primary key (ID),
foreign key (GENRE_ID) references GENRE (ID),
foreign key (AUTHOR_ID) references AUTHOR (ID)
);

CREATE TABLE TAG (
ID INT auto_increment,
USER_ID INT NOT NULL,
BOOK_ID INT NOT NULL,
GENRE_ID INT NOT NULL,
TAG_NAME VARCHAR (50),
primary key (ID),
foreign key (USER_ID) references USERS (ID),
foreign key (BOOK_ID) references BOOK (ID),
foreign key (GENRE_ID) references GENRE (ID)
);
```

```
CREATE TABLE REVIEW (
ID INT auto_increment,
USER_ID INT NOT NULL,
BOOK_ID INT NOT NULL,
RATE INT NOT NULL,
REVIEW_TEXT VARCHAR (1000) NOT NULL,
REVIEW_DATE DATE,
primary key (ID),
foreign key (USER_ID) references USERS (ID),
foreign key (BOOK_ID) references BOOK (ID),
CHECK (RATE >= 1 AND RATE <= 5)
);

CREATE TABLE FOLLOW (
FOLLOWER_USER_ID INT NOT NULL,
FOLLOWING_USER_ID INT NOT NULL,
foreign key (FOLLOWER_USER_ID) references USERS (ID),
foreign key (FOLLOWING_USER_ID) references USERS (ID)
);
CREATE TABLE BOOKSHELF (
ID INT auto_increment,
USER_ID INT NOT NULL,
BOOK_ID INT NOT NULL,
CATEGORY_ID INT NOT NULL,
ADDED_DATE DATE,
primary key (ID),
foreign key (USER_ID) references USERS(ID),
foreign key (BOOK_ID) references BOOK (ID),
foreign key (CATEGORY_ID) references CATEGORY (ID)
);

CREATE TABLE BOOK_CLUB (
ID INT auto_increment,
USER_ID INT NOT NULL,
CLUB_NAME VARCHAR (100) NOT NULL,
JOINED_DATE DATE,
DESCRIPTION VARCHAR (300) NOT NULL,
primary key (ID),
foreign key (USER_ID) references USERS (ID));
```

5. SQL statements to fill the database.

```
-- Insert records into USERS table
INSERT INTO USERS (FULL_NAME, ADDRESS, EMAIL, RIGESTERATION_DATE, LAST_LOGIN) VALUES
('John A. Smith', '101 Main St, Apt 1, 12345, Springfield, USA', 'john.smith@example.com', '2022-01-10', '2023-06-11'),
('Jane B. Doe', '202 Oak St, Apt 2, 67890, Shelbyville, USA', 'jane.doe@example.com', '2021-02-15', '2023-06-10'),
-- Add more users ;

-- Insert records into GENRE table
INSERT INTO GENRE (GENRE_NAME) VALUES
('Fiction'),
('Non-Fiction'),
-- Add more;

-- Insert records into AUTHOR table
INSERT INTO AUTHOR (AUTHOR_NAME) VALUES
('George Orwell'),
('J.K. Rowling'),
-- Add more;

-- Insert records into CATEGORY table
INSERT INTO CATEGORY (CATEGORY_TYPE) VALUES
('Read'),
('Currently Reading'),
('Want to Read');

-- Insert records into BOOK table
INSERT INTO BOOK (GENRE_ID, AUTHOR_ID, TITLE, RELEASED_DATE) VALUES
(1, 1, '1984', '1949-06-08'),
(4, 2, 'Harry Potter and the Philosopher\'s Stone', '1997-06-26'),
-- Add more;

-- Insert records into BOOKSHELF table
INSERT INTO BOOKSHELF (USER_ID, BOOK_ID, CATEGORY_ID, ADDED_DATE) VALUES
(1, 1, 1, '2023-01-01'),
(2, 2, 2, '2023-02-01'),
-- Add more;

-- Insert records into BOOK_CLUB table
INSERT INTO BOOK_CLUB (USER_ID, CLUB_NAME, JOINED_DATE, DESCRIPTION) VALUES
(1, 'Sci-Fi Enthusiasts', '2022-01-20', 'A club for fans of science fiction books.'),
(2, 'Fantasy Lovers', '2021-02-25', 'A club dedicated to fantasy literature.'),
-- Add more;

-- Insert records into TAG table
INSERT INTO TAG (USER_ID, BOOK_ID, GENRE_ID, TAG_NAME) VALUES
(1, 1, 1, 'Dystopia'),
(2, 2, 4, 'Magic'),
-- Add more;

-- Insert records into REVIEW table
INSERT INTO REVIEW (USER_ID, BOOK_ID, RATE, REVIEW_TEXT, REVIEW_DATE) VALUES
(1, 1, 5, 'A thought-provoking masterpiece.', '2023-01-01'),
(4, 4, 3, 'An epic journey through Middle-earth.', '2023-04-01')
-- Add more;

-- Insert records into FOLLOW table
INSERT INTO FOLLOW (FOLLOWER_USER_ID, FOLLOWING_USER_ID) VALUES
(1, 2),
(2, 3),
-- Add more;
```

6. Required Use Cases.

Procedure Name	Use Case Description Procedure
USER_REGISTRATION	Users register for the service by providing their full name, address, and e-mail. CREATE PROCEDURE `USER_REGISTRATION`(IN FULL_NAME VARCHAR(100), IN ADDRESS VARCHAR(100), IN EMAIL VARCHAR(50)) BEGIN INSERT INTO USERS (FULL_NAME, ADDRESS, EMAIL, RIGESTERATION_DATE) VALUES (FULL_NAME, ADDRESS, EMAIL, sysdate()); END
SEARCH_BOOK	Users can search for books CREATE PROCEDURE `SEARCH_BOOK`(IN BOOK_NAME VARCHAR (50)) BEGIN SELECT B.TITLE, B.RELEASED_DATE, G.GENRE_NAME, A.AUTHOR_NAME FROM BOOK B JOIN GENRE G ON B.GENRE_ID = G.ID JOIN AUTHOR A ON B.AUTHOR_ID = A.ID WHERE TITLE = BOOK_NAME; END
ADD_BOOK_TO_BOOKSHELF	Users can add book to their bookshelves. CREATE PROCEDURE `ADD_BOOK_TO_BOOKSHELF`(IN BOOK_NAME VARCHAR (100), IN USER_ID INT (50), IN CATEGORY VARCHAR (50)) BEGIN DECLARE BOOK_ID INT; DECLARE CATEGORY_ID INT; SELECT B.ID INTO BOOK_ID FROM BOOK B WHERE B.TITLE = BOOK_NAME; SELECT C.ID INTO CATEGORY_ID FROM CATEGORY C WHERE C.CATEGORY_TYPE = CATEGORY; INSERT INTO BOOKSHELF (USER_ID, BOOK_ID, CATEGORY_ID, ADDED_DATE) VALUES (USER_ID, BOOK_ID, CATEGORY_ID, sysdate()); END
WRITE_REVIEW	Users can write reviews with rating to books (1 → 5) CREATE PROCEDURE `WRITE_REVIEW`(IN USER_ID INT, IN BOOK_NAME VARCHAR (50), IN RATE INT, IN REVIEW_TEXT VARCHAR (250)) BEGIN declare BOOK_ID INT; SELECT B.ID INTO BOOK_ID FROM BOOK B WHERE B.TITLE = BOOK_NAME; INSERT INTO REVIEW (USER_ID, BOOK_ID, RATE, REVIEW_TEXT, REVIEW_DATE) VALUES (USER_ID, BOOK_ID, RATE, REVIEW_TEXT, sysdate()); END
FOLLOW	Users can follow each others CREATE PROCEDURE `FOLLOW`(IN FOLLOWER_ID INT, IN FOLLOWING_ID INT) BEGIN INSERT INTO FOLLOW (FOLLOWER_USER_ID,FOLLOWING_USER_ID) VALUES (FOLLOWER_ID,FOLLOWING_ID); END

FOLLOWING_REVIEWS	Followers can see reviews and rating for following users.
	CREATE PROCEDURE `FOLLOWING_REVIEWS`(
	IN FOLLOWER_ID INT,
	IN FOLLOWING_ID INT)
	BEGIN
	declare TEMP_COUNT INT;
	
	SELECT count(*) INTO TEMP_COUNT FROM FOLLOW F
	WHERE F.FOLLOWER_USER_ID = FOLLOWER_ID
	AND F.FOLLOWING_USER_ID = FOLLOWING_ID;
	
	IF TEMP_COUNT !=0 THEN
	SELECT U.FULL_NAME , B.TITLE, R.RATE, R.REVIEW_TEXT, R.REVIEW_DATE
	FROM REVIEW R
	JOIN USERS U ON R.USER_ID = U.ID
	JOIN BOOK B ON R.BOOK_ID = B.ID
	WHERE R.USER_ID = FOLLOWING_ID;
	END IF;
	END

JOIN_CLUB	Users can join book clubs to discuss their favourite books. CREATE PROCEDURE `JOIN_CLUB`(IN USER_ID INT, IN CLUB_NAME VARCHAR (50)) BEGIN declare CLUB_DESCRIPTION VARCHAR (100); SELECT C.DESCRIPTION INTO CLUB_DESCRIPTION FROM BOOK_CLUB C WHERE C.CLUB_NAME = CLUB_NAME; INSERT INTO BOOK_CLUB (USER_ID, CLUB_NAME,DESCRIPTION, JOINED_DATE) VALUES (USER_ID, CLUB_NAME,CLUB_DESCRIPTION, sysdate()); END
TAG_BOOK	Users can tag books with genres to help categorize them. CREATE PROCEDURE `TAG_BOOK`(IN USER_ID INT, IN BOOK_NAME VARCHAR (50), IN TAG_NAME VARCHAR (50)) BEGIN DECLARE BOOK_ID INT; DECLARE GENRE_ID INT; SELECT B.ID INTO BOOK_ID FROM BOOK B WHERE B.TITLE = BOOK_NAME; SELECT G.ID INTO GENRE_ID FROM GENRE G JOIN BOOK B ON G.ID = B.GENRE_ID WHERE B.ID = BOOK_ID; INSERT INTO TAG (USER_ID, BOOK_ID, GENRE_ID, TAG_NAME) VALUES (USER_ID, BOOK_ID, GENRE_ID, TAG_NAME); END
LIST_BOOK_BY_GENRE	Users can list books of a specific genre CREATE PROCEDURE `LIST_BOOK_BY_GENRE`(IN GENRE_NAME VARCHAR (50)) BEGIN SELECT B.TITLE, B.RELEASED_DATE, G.GENRE_NAME , A.AUTHOR_NAME FROM BOOK B JOIN GENRE G ON B.GENRE_ID = G.ID JOIN AUTHOR A ON B.AUTHOR_ID = A.ID WHERE G.GENRE_NAME = GENRE_NAME; END

LIST_BOOK_BY_GENRE_AND_RATE	Users can list books of a specific genre with rate at least a certain score
	CREATE VIEW `social_library`.`avg_rates` AS select `social_library`.`review`.`BOOK_ID` AS `BOOK_ID`, avg(`social_library`.`review`.`RATE`) AS `AVG_RATE` from `social_library`.`review` group by `social_library`.`review`.`BOOK_ID`;
	CREATE PROCEDURE `LIST_BOOK_BY_GENRE_AND_RATE`(IN GENRE_NAME VARCHAR (50), IN RATE INT) BEGIN SELECT B.TITLE, G.GENRE_NAME, A.AUTHOR_NAME, AG.AVG_RATE FROM BOOK B JOIN GENRE G ON B.GENRE_ID = G.ID JOIN AVG_RATES AG ON B.ID = AG.BOOK_ID JOIN AUTHOR A ON A.ID = B.AUTHOR_ID WHERE G.GENRE_NAME = GENRE_NAME AND AG.AVG_RATE >= RATE; END
NEW_BOOK_BY_GENRE	Users can view new book releases in a given genre
	CREATE PROCEDURE `NEW_BOOK_BY_GENRE`(IN GENRE_NAME VARCHAR (50), IN RELEASED_DATE DATE) BEGIN SELECT B.TITLE, B.RELEASED_DATE, G.GENRE_NAME, A.AUTHOR_NAME FROM BOOK B JOIN GENRE G ON G.ID = B.GENRE_ID JOIN AUTHOR A ON A.ID = B.AUTHOR_ID WHERE G.GENRE_NAME = GENRE_NAME AND B.RELEASED_DATE >= RELEASED_DATE; END
NEW_BOOK_BY_AUTHOR	Users can view new book releases in a specific author
	CREATE PROCEDURE `NEW_BOOK_BY_AUTHOR`(IN AUTHOR_NAME VARCHAR (50), IN RELEASED_DATE DATE) BEGIN SELECT B.TITLE, B.RELEASED_DATE, G.GENRE_NAME, A.AUTHOR_NAME FROM BOOK B JOIN GENRE G ON G.ID = B.GENRE_ID JOIN AUTHOR A ON A.ID = B.AUTHOR_ID WHERE A.AUTHOR_NAME = AUTHOR_NAME AND B.RELEASED_DATE >= RELEASED_DATE; END
LIST_EMAILS_BY_LOGGIN	Admins can list email addresses of users who haven't logged in for a certain period, along with their most recent bookshelf activity.
	CREATE PROCEDURE `LIST_EMAILS_BY_LOGGIN`(IN LAST_LOGGIN DATE) BEGIN SELECT U.FULL_NAME, U.EMAIL, U.LAST_LOGIN, C.CATEGORY_TYPE FROM USERS U JOIN BOOKSHELF BS ON U.ID = BS.USER_ID JOIN CATEGORY C ON C.ID = BS.CATEGORY_ID WHERE U.LAST_LOGIN <= LAST_LOGGIN; END

RECOMMEND_BOOKS	The platform can recommend books to users based on their reading history and preferences, example : the platform could suggest books that are highly rated by users with similar reading habits.
	CREATE VIEW `social_library`.`list_reading_history` AS select `bs`.`USER_ID` AS `USER_ID`, `b`.`TITLE` AS `TITLE`, `a`.`ID` AS `AUTHOR_ID`, `a`.`AUTHOR_NAME` AS `AUTHOR_NAME`, `g`.`ID` AS `GENRE_ID`, `g`.`GENRE_NAME` AS `GENRE_NAME`, `r`.`RATE` AS `AVG_RATE`, 'RECOMMENDED BY YOUR READING HISTORY' AS `RECOMMENDED BY YOUR READING HISTORY` from ((((`social_library`.`book` `b` join `social_library`.`bookshelf` `bs` on((`bs`.`BOOK_ID` = `b`.`ID`))) join `social_library`.`genre` `g` on((`g`.`ID` = `b`.`GENRE_ID`))) join `social_library`.`author` `a` on((`a`.`ID` = `b`.`AUTHOR_ID`))) join `social_library`.`review` `r` on((`r`.`USER_ID` = `bs`.`USER_ID`))) where (`bs`.`CATEGORY_ID` = 1);
	CREATE VIEW `social_library`.`list_users_preferences` AS select `r`.`USER_ID` AS `USER_ID`, `b`.`TITLE` AS `TITLE`, `a`.`ID` AS `AUTHOR_ID`, `a`.`AUTHOR_NAME` AS `AUTHOR_NAME`, `g`.`ID` AS `GENRE_ID`, `g`.`GENRE_NAME` AS `GENRE_NAME`, `r`.`RATE` AS `RATE`, 'RECOMMENDED BY YOUR PREFERENCES' AS `RECOMMENDED BY YOUR PREFERENCES` from (((`social_library`.`book` `b` join `social_library`.`review` `r` on((`r`.`BOOK_ID` = `b`.`ID`))) join `social_library`.`genre` `g` on((`g`.`ID` = `b`.`GENRE_ID`))) join `social_library`.`author` `a` on((`a`.`ID` = `b`.`AUTHOR_ID`))) where (`r`.`RATE` >= 3);
	CREATE VIEW `social_library`.`list_similar_reading_habits` AS select `bs`.`USER_ID` AS `USER_ID`, `f`.`FOLLOWING_USER_ID` AS `FOLLOWING_USER_ID`, `b`.`TITLE` AS `BOOK_TITLE_FOR_FOLLOWING_USER`, `a`.`ID` AS `AUTHOR_ID`,`a`.`AUTHOR_NAME` AS `BOOK_AUTHOR_FOR_FOLLOWING_USER`, `g`.`ID` AS `GENRE_ID`, `g`.`GENRE_NAME` AS `GENRE_FOR_FOLLOWING_USER`, 'RECOMMENDED BY OTHER USERS PREFERENCES' AS `RECOMMENDED BY OTHER USERS PREFERENCES` from ((((`social_library`.`book` `b` join `social_library`.`bookshelf` `bs` on((`bs`.`BOOK_ID` = `b`.`ID`))) join `social_library`.`genre` `g` on((`g`.`ID` = `b`.`GENRE_ID`))) join `social_library`.`author` `a` on((`a`.`ID` = `b`.`AUTHOR_ID`))) join `social_library`.`follow` `f` on((`f`.`FOLLOWING_USER_ID` = `bs`.`USER_ID`)));
	CREATE PROCEDURE `RECOMMEND_BOOKS`(IN USER_ID INT) BEGIN -- RECOMMEND BY READING_HISTORY SELECT DISTINCT(B.TITLE), LRH.GENRE_NAME, LRH.AUTHOR_NAME, `RECOMMENDED BY YOUR READING HISTORY` FROM BOOK B LEFT JOIN LIST_READING_HISTORY LRH ON B.GENRE_ID = LRH.GENRE_ID OR B.AUTHOR_ID = LRH.AUTHOR_ID WHERE LRH.USER_ID = USER_ID UNION ALL SELECT DISTINCT(B.TITLE), LUP.GENRE_NAME, LUP.AUTHOR_NAME, `RECOMMENDED BY YOUR PREFERENCES` FROM BOOK B LEFT JOIN LIST_USERS_PREFERENCES LUP ON B.GENRE_ID = LUP.GENRE_ID OR B.AUTHOR_ID = LUP.AUTHOR_ID WHERE LUP.USER_ID = USER_ID UNION ALL SELECT DISTINCT(B.TITLE), LSRH.GENRE_FOR_FOLLOWING_USER AS GENRE_NAME, LSRH.BOOK_AUTHOR_FOR_FOLLOWING_USER AS AUTHOR_NAME, `RECOMMENDED BY OTHER USERS PREFERENCES` FROM BOOK B LEFT JOIN LIST_SIMILAR_READING_HABITS LSRH ON B.GENRE_ID = LSRH.GENRE_ID OR B.AUTHOR_ID = LSRH.AUTHOR_ID WHERE LSRH.USER_ID = USER_ID; END

7. Additional Use Case.

Procedure Name	Use Case Description
	Procedure
UPDATE_CATEGORY	User can change category for a book if this book exists in his bookshelf

	CREATE PROCEDURE `UPDATE_CATEGORY`(
	IN USER_ID INT,
	IN BOOK_NAME VARCHAR (50),
	IN NEW_CTAGORY VARCHAR (50))
	
	BEGIN
	DECLARE CATEGORY_ID INT;
	DECLARE BOOK_ID INT;
	DECLARE RECORD_EXISTS INT;
	
	SELECT C.ID INTO CATEGORY_ID FROM CATEGORY C
	WHERE C.CATEGORY_TYPE = NEW_CTAGORY;
	SELECT B.ID INTO BOOK_ID FROM BOOK B WHERE B.TITLE = BOOK_NAME;
	
	SELECT COUNT(*) INTO RECORD_EXISTS FROM BOOKSHELF BS
	WHERE BS.USER_ID = USER_ID AND BS.BOOK_ID = BOOK_ID;
	
	IF RECORD_EXISTS = 0 THEN
	SIGNAL SQLSTATE '45000'
	SET MESSAGE_TEXT = 'This book is not in your bookshelf';
	
	ELSE
	UPDATE BOOKSHELF BS
	SET BS.CATEGORY_ID = CATEGORY_ID
	WHERE BS.USER_ID = USER_ID AND BS.BOOK_ID = BOOK_ID;
	END IF;
	
	END

8. Some improvements.

<table>
<tr><th rowspan="2">Trigger Name</th><th>Use Case Description</th></tr>
<tr><th>Trigger</th></tr>
<tr><td rowspan="2">CHECK_BEFORE_REVIEW</td><td>Users can review a book only for one time & can’t review book if the category isn’t “Read”</td></tr>
<tr><td>

```
CREATE TRIGGER CHECK_BEFORE_REVIEW
BEFORE INSERT ON REVIEW

FOR EACH ROW

BEGIN
DECLARE REVIEW_EXISTS INT;
DECLARE CATEGORY_ID INT;

SELECT COUNT(*) INTO REVIEW_EXISTS FROM REVIEW R
JOIN BOOKSHELF BS ON BS.BOOK_ID = R.BOOK_ID
WHERE R.USER_ID = NEW.USER_ID AND R.BOOK_ID = NEW.BOOK_ID;

IF REVIEW_EXISTS != 0 THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'Cannot review a book that you have reviewed ir
before';
END IF;


SELECT BS.CATEGORY_ID INTO CATEGORY_ID FROM BOOKSHELF BS
JOIN CATEGORY C ON C.ID = BS.CATEGORY_ID
WHERE BS.USER_ID = NEW.USER_ID;

IF CATEGORY_ID != 1 THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'Cannot review a book that you have not read it before';
END IF;

END
```

</td></tr>
<tr><td rowspan="2">CHECK_BEFORE_ADD_TO_BOOKSHELF</td><td>Users can add book to bookshelf if this book doesn’t exists in the bookshelf</td></tr>
<tr><td>

```
CREATE TRIGGER CHECK_BEFORE_ADD_TO_BOOKSHELF
BEFORE INSERT ON BOOKSHELF

FOR EACH ROW

BEGIN
DECLARE USER_EXISTS INT;

SELECT COUNT(*) INTO USER_EXISTS FROM BOOKSHELF BS
WHERE BS.USER_ID = NEW.USER_ID AND BS.BOOK_ID = NEW.BOOK_ID;

IF USER_EXISTS = 1 THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'This book is already exists in your bookshelf';
END IF;

END
```

</td></tr>
</table>

CHECK_BEFORE_FOLLOW	• User can't follow himself • Check if follower and following users exist.

```
CREATE TRIGGER CHECK_BEFORE_FOLLOW
BEFORE INSERT ON FOLLOW

FOR EACH ROW

BEGIN
DECLARE FOLLOWERS_EXISTS INT;
DECLARE FOLLOWING_USER_EXISTS INT;
DECLARE FOLLOWER_USER_EXISTS INT;

SELECT COUNT(*) INTO
FOLLOWERS_EXISTS FROM FOLLOW F
WHERE F.FOLLOWER_USER_ID = NEW.FOLLOWER_USER_ID
AND F.FOLLOWING_USER_ID = NEW.FOLLOWING_USER_ID ;

IF FOLLOWERS_EXISTS = 1 THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT =
'YOU ALREADY FOLLOW THIS USER';
END IF;

SELECT COUNT(*) INTO FOLLOWING_USER_EXISTS FROM USERS U
WHERE U.ID = NEW.FOLLOWING_USER_ID;

IF FOLLOWING_USER_EXISTS = 0 THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT =
'THE FOLLOWING USER NOT EXISTS';
END IF;


SELECT COUNT(*) INTO FOLLOWER_USER_EXISTS FROM USERS U
WHERE U.ID = NEW.FOLLOWER_USER_ID;

IF FOLLOWER_USER_EXISTS = 0 THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT =
'THE FOLLOWER USER NOT EXISTS';
END IF;



IF NEW.FOLLOWER_USER_ID = NEW.FOLLOWING_USER_ID THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT =
'ERROR';
END IF;

END
```