Consider the LIBRARY relational database schema shown below. Create tables with appropriate constraints. Choose the appropriate action (reject, cascade, set to NULL, set to default) for each referential integrity constraint, both for the deletion of a referenced tuple and for the update of a primary key attribute value in a referenced tuple.

Books (book_id , title, author, published_year, genre, available_copies)

Members(member_id, name, email, age, gender)

Borrowed(borrow_id, book_id, member_id, borrow_date, return_date)

Write down SQL expressions for the following queries:

1. Get a list of books borrowed by a specific member (e.g., John Smith).

2. Find members who borrowed a specific book (e.g., “Atomic Habits”).

3. List genre wise available copies of books in library.

4. Find the genre most liked by female members of the Library.

5. Find the genre most liked by senior citizen members of the Library.

6. List the members who borrowed and returned the books before overdue (within 14 days from the borrow date).

7. Write a query to find all books that are currently borrowed and overdue (i.e., not returned within 14 days from the borrow date). Display the book titles and the names of members who borrowed them.

8. Find the most popular genre in the library (Display the genre with the highest total number of books borrowed).

9. Add a new column named fine amount to the Borrowed table. This column will store the fine amount (in Rs.) for overdue books. Set an appropriate default value for this column.

10. Write a SQL query to calculate the total fines collected from all overdue books. Use the fine amount column and consider all books that are currently overdue.

11. Find the top 5 members who have borrowed the most books. Display their names and the number of books they have borrowed.

12. Add a joint UNIQUE constraint to the book id and member id columns in the Borrowed table to prevent a member from borrowing the same book more than once simultaneously.

13. Write a query to find the books that are currently available for borrowing (i.e., books with at least one available copy). Display the book titles and the number of available copies.

14. Create a query that categorizes members based on their borrowing behaviour. Use a CASE statement to categorize them as "Frequent Borrowers" if they have borrowed more than 10

books, "Regular Borrowers" if they have borrowed between 5 and 10 books, and "Occasional Borrowers" if they have borrowed less than 5 books. (Hint: use CASE statement)

15. Find members who return books quickly. Calculate the average duration it takes for each member to return borrowed books, and then identify members whose average return time is less than 7 days.

16. Explore about Triggers in SQL. Then implement following two triggers for your Library database:

a. After each book issue, update the available copies of that book in Books table using a Trigger.

b. Similarly, after each book return update the available copies of that book in Books table using a Trigger.

Sample Data for Books Table

(Book_Id, Title, Author, Published_Year, Genre, Available_Copies)

(00551, 'The_Great_Gatsby', 'F_Scott_Fitzgerald', '19250410', 'Tragedy', '10000'),

(00552, 'ULYSSES', 'James_Joyce', '19220202', 'Modernist_Novel', '10000'),

(00553, 'Lolita', 'Vladimir_Nabokov', '19552001', 'Novel', '10000'),

(00554, 'Brave_New_World', 'Aldous_Huxley', '19320505', 'Science_Fiction_Dystopian_Fiction', '10000'),

(00555, 'The_Sound_And_The_Fury', 'William_Faulkner', '19290103', 'Southern_Gothic', '10000'),

(00556, 'Catch22', 'Joseph_Heller', '19611010', 'Dark_Comedy', '10000'),
(00557, 'The_Grapes_Of_Wrath', 'John_Steinbeck', '19391404', 'Novel', '10000'),

(00558, 'I_Claudius', 'Robert_Graves', '19340810', 'Historical', '10000'),

(00559, 'To_The_Lighthouse', 'Virginia_Woolf', '19270505', 'Modernism', '10000'),

(05510, 'Slaughterhouse_Five', 'Kurt_Vonnegut', '19693103', 'War_Novel', '10000'),

(05511, 'Invisible_Man', 'Ralph_Ellison', '19521404', 'African_American_Literature', '10000'),

(05512, 'Native_Son', 'Richard_Wright', '19400103', 'Social_Protest', '10000'),

(05513, 'USA_Trilogy', 'John_Dos_Passos', '19300405', 'Political_Fiction', '10000'),

(05514, 'A_Passage_To_India', 'E_M_Forster', '19240406', 'Novel', '10000'),

(05515, 'Tender_Is_The_Night', 'F_Scott_Fitzgerald', '19341204', 'Tragedy', '10000'),

(05516, 'Animal_Farm', 'George_Orwell', '19451708', 'Political_Satire', '10000'),

(05517, 'The_Golden_Bowl', 'Henry_James', '19041011', 'Philosophy', '10000'),

(05518, 'A_Handful_Of_Dust', 'Evelyn_Waugh', '19340603', 'Fiction', '10000'),

(05519, 'As_I_Lay_Dying', 'William_Faulkner', '19300302', 'Black_Comedy', '10000'),

(05520, 'The_Heart_Of_The_Matter', 'Graham_Greene', '19480302', 'Nove', '10000')

1. Books (book_id, title, author, published_year, genre, available_copies)

2. Members (member_id, name, email, age, gender)

3. Borrowed (borrow_id, book_id, member_id, borrow_date, return_date)

```
CREATE TABLE Books (
   book_id INT PRIMARY KEY,
   title VARCHAR(255),
   author VARCHAR(255),
   published_year YEAR,
   genre VARCHAR(100),
   available_copies INT
);


CREATE TABLE Members (
   member_id INT PRIMARY KEY,
   name VARCHAR(100),
   email VARCHAR(100) UNIQUE,
   age INT,
   gender ENUM('Male', 'Female', 'Other')
);

CREATE TABLE Borrowed (
   borrow_id INT PRIMARY KEY,
   book_id INT,
   member_id INT,
   borrow_date DATE,
   return_date DATE,
   FOREIGN KEY (book_id) REFERENCES Books(book_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
   FOREIGN KEY (member_id) REFERENCES Members(member_id)
      ON DELETE CASCADE ON UPDATE CASCADE
);
```

1. Books borrowed by a specific member (e.g., John Smith):

SELECT B.title

```
FROM Books B
JOIN Borrowed BR ON B.book_id = BR.book_id
JOIN Members M ON BR.member_id = M.member_id
WHERE M.name = 'John Smith';
```

2. Members who borrowed a specific book (e.g., “Atomic Habits”):

```
SELECT M.name
FROM Members M
JOIN Borrowed BR ON M.member_id = BR.member_id
JOIN Books B ON BR.book_id = B.book_id
WHERE B.title = 'Atomic Habits';
```

3. List genres with available copies of books:

```
SELECT DISTINCT genre
FROM Books
WHERE available_copies > 0;
```

4. Genre most liked by female members:

```
SELECT B.genre, COUNT(*) AS borrow_count
FROM Borrowed BR
JOIN Members M ON BR.member_id = M.member_id
JOIN Books B ON BR.book_id = B.book_id
WHERE M.gender = 'Female'
GROUP BY B.genre
ORDER BY borrow_count DESC
LIMIT 1;
```

5. Genre most liked by senior citizen members (age >= 60):

```
SELECT B.genre, COUNT(*) AS borrow_count
FROM Borrowed BR
JOIN Members M ON BR.member_id = M.member_id
JOIN Books B ON BR.book_id = B.book_id
WHERE M.age >= 60
GROUP BY B.genre
ORDER BY borrow_count DESC
LIMIT 1;
```

6. Members who borrowed and returned books on time (within 14 days):

```
SELECT DISTINCT M.name
FROM Borrowed BR
JOIN Members M ON BR.member_id = M.member_id
WHERE DATEDIFF(BR.return_date, BR.borrow_date) <= 14;
```

7. Overdue books not returned (borrowed > 14 days ago):

```
SELECT B.title, M.name
FROM Borrowed BR
JOIN Books B ON BR.book_id = B.book_id
JOIN Members M ON BR.member_id = M.member_id
WHERE BR.return_date IS NULL
  AND DATEDIFF(CURDATE(), BR.borrow_date) > 14;
```

8. Most popular genre:

```
SELECT B.genre, COUNT(*) AS borrow_count
FROM Borrowed BR
JOIN Books B ON BR.book_id = B.book_id
GROUP BY B.genre
ORDER BY borrow_count DESC
LIMIT 1;
```

9. Add fine amount column to Borrowed table:

```
ALTER TABLE Borrowed
ADD fine_amount DECIMAL(10,2) DEFAULT 0;
```

10. Total fines collected (for overdue books only):

```
SELECT SUM(fine_amount) AS total_fines_collected
FROM Borrowed
WHERE return_date IS NULL OR DATEDIFF(return_date, borrow_date) > 14;
```

11. Top 5 members with most borrowed books:

```
SELECT M.name, COUNT(*) AS books_borrowed
FROM Borrowed BR
JOIN Members M ON BR.member_id = M.member_id
GROUP BY M.member_id
ORDER BY books_borrowed DESC
LIMIT 5;
```

12. Add unique constraint on (book_id, member_id):

```
ALTER TABLE Borrowed
ADD CONSTRAINT unique_book_member UNIQUE (book_id, member_id);
```

13. Find currently available books (available_copies > 0):

```
SELECT title, available_copies
FROM Books
WHERE available_copies > 0;
```

14. Categorize members based on borrowing behavior:

```
SELECT M.name,
  COUNT(BR.book_id) AS books_borrowed,
  CASE
    WHEN COUNT(BR.book_id) > 10 THEN 'Frequent Borrower'
    WHEN COUNT(BR.book_id) BETWEEN 5 AND 10 THEN 'Regular Borrower'
    ELSE 'Occasional Borrower'
  END AS borrower_category
FROM Members M
JOIN Borrowed BR ON M.member_id = BR.member_id
GROUP BY M.member_id;
```

15. Average return duration and members with > 7 days:

```
SELECT M.name,
      AVG(DATEDIFF(return_date, borrow_date)) AS avg_return_days
FROM Borrowed BR
JOIN Members M ON BR.member_id = M.member_id
WHERE return_date IS NOT NULL
GROUP BY M.member_id
HAVING avg_return_days > 7;
```

16. Triggers to update available_copies after issue and return:

a. Trigger after book issue (decrease available_copies):

```
CREATE TRIGGER after_book_issue
AFTER INSERT ON Borrowed
FOR EACH ROW
BEGIN
  UPDATE Books
  SET available_copies = available_copies - 1
  WHERE book_id = NEW.book_id;
END;
```

b. Trigger after book return (increase available_copies):

```
CREATE TRIGGER after_book_return
AFTER UPDATE ON Borrowed
FOR EACH ROW
BEGIN
  IF NEW.return_date IS NOT NULL AND OLD.return_date IS NULL THEN
    UPDATE Books
    SET available_copies = available_copies + 1
    WHERE book_id = NEW.book_id;
  END IF;
END;
```