Name: Swarnali Ghosh

Java Pre-Skilling Training Session

Assignment -4.1

Module-4

Mail-id: swarnalighosh666@gmail.com

ASSIGNMENT-1

Analyze a given business scenario and create an ER diagram that includes entities, relationships, attributes, and cardinality. Ensure that the diagram reflects proper normalization up to the third normal form.

SOLUTION:

Designed an ER diagram for a University Registration System, including entities for students, course offerings, instructors, and courses.

Entities:

1. Student
2. Course Offering
3. Instructor
4. Course

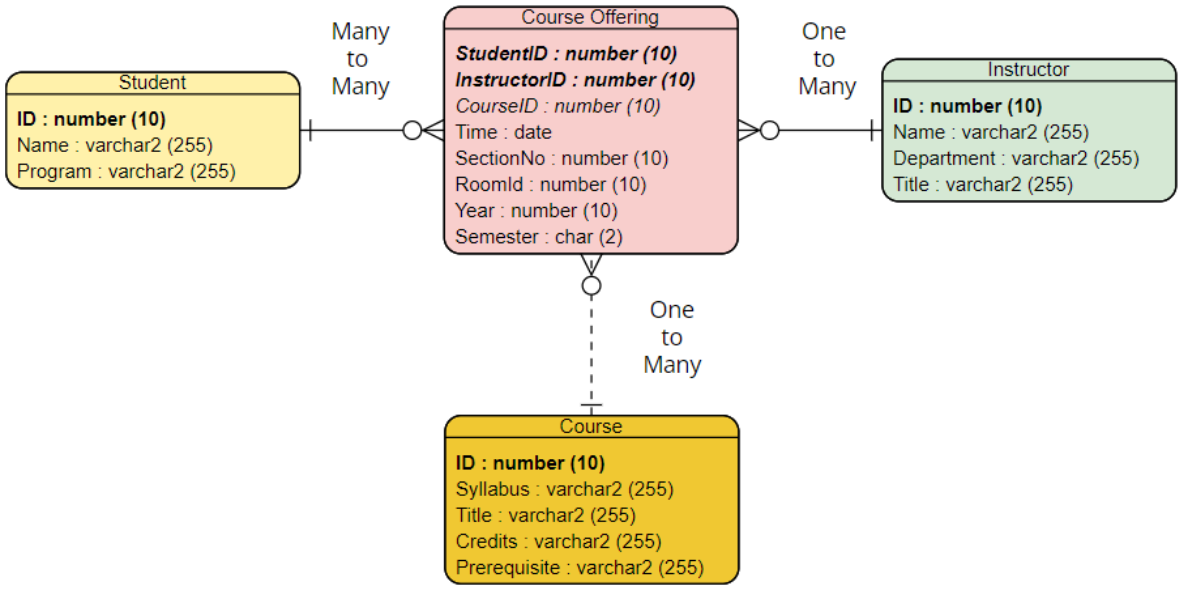
Attributes:

- Student: StudentID (PK), Name, Program
- Course Offering: StudentID (PK), CourseID (FK), InstructorID (FK), Time, SectionNo, RoomID, Semester, Year
- Instructor: ID (PK), Name, Department, Title

- Course: CourseID (PK), Syllabus,Title, Credits , Pre-requisite.

Relationships:

1. Students and Course Offerings: One student can enroll in multiple course offerings. One course offering can have multiple students enrolled. (Many-to-Many).
2. Instructors and Course Offerings: One instructor can teach multiple course offerings. One course offering has one instructor. (One-to-Many).
3. Courses and Course Offerings: One course can have multiple course offerings. One course offering belongs to one course. (One-to-Many).



ER Diagram: University Registration System

ASSIGNMENT-2

Design a database schema for a library system, including tables, fields, and constraints like NOT NULL, UNIQUE, and CHECK. Include primary and foreign keys to establish relationships between tables.

SOLUTION:

A basic database schema for a library system:

1. Table: Books
    - Fields:
        - book_id (Primary Key, Integer)
        - title (String)
        - author (String)
        - genre (String)
        - publication_year (Integer)
        - ISBN (String, UNIQUE)

2. Table: Members
    - Fields:
        - member_id (Primary Key, Integer)
        - name (String)
        - email (String, UNIQUE)
        - phone (String)

3. Table: Loans
    - Fields:
        - loan_id (Primary Key, Integer)
        - book_id (Foreign Key referencing Books.book_id)
        - member_id (Foreign Key referencing Members.member_id)
        - loan_date (Date)
        - return_date (Date, CHECK return_date >= loan_date)

4. Table: Reservations
    - Fields:
        - reservation_id (Primary Key, Integer)
        - book_id (Foreign Key referencing Books.book_id)
        - member_id (Foreign Key referencing Members.member_id)
        - reservation_date (Date)
        - status (String, CHECK status IN ('Pending', 'Completed', 'Cancelled'))

5. Table: Authors
    - Fields:
        - author_id (Primary Key, Integer)
        - author_name (String)
        - nationality (String)

6. Table: Genres
    - Fields:
        - genre_id (Primary Key, Integer)
        - genre_name (String)

7. Table: Book_Authors (Many-to-Many Relationship Table)
    - Fields:
        - book_id (Foreign Key referencing Books.book_id)
        - author_id (Foreign Key referencing Authors.author_id)

This schema establishes relationships between books and authors via a many-to-many relationship table `Book_Authors`. It also maintains information about members, loans, reservations, genres, and authors, ensuring data integrity with primary and foreign keys, as well as constraints like UNIQUE and CHECK.

Here are the MySQL commands to create the tables described in the schema:

-- Create table for Books

```
CREATE TABLE Books (
    book_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255),
    author VARCHAR(255),
    genre VARCHAR(100),
    publication_year INT,
    ISBN VARCHAR(20) UNIQUE
);
```

-- Create table for Members

```
CREATE TABLE Members (
    member_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(20)
);
```

-- Create table for Loans

```
CREATE TABLE Loans (
    loan_id INT AUTO_INCREMENT PRIMARY KEY,
    book_id INT,
    member_id INT,
    loan_date DATE,
    return_date DATE,
    FOREIGN KEY (book_id) REFERENCES Books(book_id),
    FOREIGN KEY (member_id) REFERENCES Members(member_id),
    CHECK (return_date >= loan_date)
);
```

-- Create table for Reservations

```
CREATE TABLE Reservations (
    reservation_id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    book_id INT,
    member_id INT,
    reservation_date DATE,
    status ENUM('Pending', 'Completed', 'Cancelled'),
    FOREIGN KEY (book_id) REFERENCES Books(book_id),
    FOREIGN KEY (member_id) REFERENCES Members(member_id)
);
```

-- Create table for Authors

```
CREATE TABLE Authors (
    author_id INT AUTO_INCREMENT PRIMARY KEY,
    author_name VARCHAR(255),
    nationality VARCHAR(100)
);
```

-- Create table for Genres

```
CREATE TABLE Genres (
    genre_id INT AUTO_INCREMENT PRIMARY KEY,
    genre_name VARCHAR(100)
);
```

-- Create table for Book_Authors (Many-to-Many Relationship)

```
CREATE TABLE Book_Authors (
    book_id INT,
    author_id INT,
    PRIMARY KEY (book_id, author_id),
    FOREIGN KEY (book_id) REFERENCES Books(book_id),
    FOREIGN KEY (author_id) REFERENCES Authors(author_id)
);
```

→ DESCRIBE Books;

```
+------------------+-------------+------+-----+---------+----------------+
| Field            | Type        | Null | Key | Default | Extra          |
+------------------+-------------+------+-----+---------+----------------+
| book_id          | int(11)     | NO   | PRI | NULL    | auto_increment |
| title            | varchar(255)| YES  |     | NULL    |                |
| author           | varchar(255)| YES  |     | NULL    |                |
| genre            | varchar(100)| YES  |     | NULL    |                |
| publication_year | int(11)     | YES  |     | NULL    |                |
| ISBN             | varchar(20) | YES  | UNI | NULL    |                |
+------------------+-------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

ASSIGNMENT-3

Explain the ACID properties of a transaction in your own words. Write SQL statements to simulate a transaction that includes locking and demonstrate different isolation levels to show concurrency control.

SOLUTION:

ACID Properties of a Transaction

ACID is an acronym that stands for Atomicity, Consistency, Isolation, and Durability. These are the four key properties that guarantee reliable processing of database transactions.

1. **Atomicity:** This ensures that a transaction is treated as a single unit, which either completes entirely or not at all. If any part of the transaction fails, the entire transaction is rolled back, and the database remains unchanged.
2. **Consistency:** This property ensures that a transaction takes the database from one valid state to another valid state, maintaining all predefined rules, such as integrity constraints.
3. **Isolation:** This ensures that the operations of one transaction are isolated from the operations of other transactions. This means that intermediate states of a transaction are not visible to other transactions until the transaction is committed, ensuring concurrent transactions do not affect each other.

4. Durability: This ensures that once a transaction has been committed, it will remain so, even in the event of a system failure. The results of the transaction are permanently recorded in the database.

To demonstrate a transaction that includes locking and different isolation levels, let's consider a simple scenario with two tables: accounts and transactions.

```
CREATE TABLE accounts (
    account_id INT PRIMARY KEY,
    balance DECIMAL(10, 2)
);

CREATE TABLE transactions (
    transaction_id INT PRIMARY KEY,
    account_id INT,
    amount DECIMAL(10, 2),
    FOREIGN KEY (account_id) REFERENCES accounts(account_id)
);
```

We insert some initial data:

```
INSERT INTO accounts (account_id, balance) VALUES (1, 1000.00), (2, 1500.00);

INSERT INTO transactions (transaction_id, account_id, amount) VALUES (1, 1, 200.00), (2, 2, 300.00);
```

Transaction with Locking

A transaction that transfers money from one account to another might look like this:

```
BEGIN TRANSACTION;

-- Step 1: Lock the accounts to ensure atomicity and consistency
SELECT * FROM accounts WHERE account_id = 1 FOR UPDATE;
SELECT * FROM accounts WHERE account_id = 2 FOR UPDATE;

-- Step 2: Perform the transfer
UPDATE accounts SET balance = balance - 100.00 WHERE account_id = 1;
UPDATE accounts SET balance = balance + 100.00 WHERE account_id = 2;

-- Step 3: Record the transaction
INSERT INTO transactions (transaction_id, account_id, amount) VALUES (3, 1, -100.00),
(4, 2, 100.00);

COMMIT;
```

Demonstrating Isolation Levels: -

Read Committed: This isolation level ensures that any data read is committed at the moment it is read. It prevents dirty reads.

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
BEGIN TRANSACTION;
SELECT * FROM accounts;
```

```
COMMIT;
```

Serializable: This is the highest isolation level, ensuring complete isolation from other transactions. It prevents dirty reads, non-repeatable reads, and phantom reads.

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

SELECT * FROM accounts WHERE account_id = 1 FOR UPDATE;

-- Attempt another read

SELECT * FROM accounts WHERE account_id = 2 FOR UPDATE;

COMMIT;
```

By using different isolation levels and locking mechanisms, we can control concurrency in a database to ensure data integrity and consistency even when multiple transactions are executed simultaneously.

This approach helps prevent issues like dirty reads, non-repeatable reads, and phantom reads.

ASSIGNMENT-4

Write SQL statements to CREATE a new database and tables that reflect the library schema you designed earlier. Use ALTER statements to modify the table structures and DROP statements to remove a redundant table.

SOLUTION:

--Creating all the tables reflecting library schema & describing it –

Table Books-

```
5
6    -- Create table for Books
7    CREATE TABLE Books (
8        book_id INT AUTO_INCREMENT PRIMARY KEY,
9        title VARCHAR(255) NOT NULL,
10       author VARCHAR(255) NOT NULL,
11       genre VARCHAR(100) NOT NULL,
12       publication_year INT CHECK (publication_year >= 0),
13       ISBN VARCHAR(20) UNIQUE NOT NULL
14   );
15   desc Books;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
book_id	int	NO	PRI	NULL	auto_increment
title	varchar(255)	NO		NULL	
author	varchar(255)	NO		NULL	
genre	varchar(100)	NO		NULL	
publication_year	int	YES		NULL	
ISBN	varchar(20)	NO	UNI	NULL	

Result 1

Table Members-

```
17
18   -- Create table for Members
19   CREATE TABLE Members (
20       member_id INT AUTO_INCREMENT PRIMARY KEY,
21       name VARCHAR(100) NOT NULL,
22       email VARCHAR(100) UNIQUE NOT NULL,
23       phone VARCHAR(20) NOT NULL
24   );
25   desc Members;
26
27
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
member_id	int	NO	PRI	NULL	auto_increment
name	varchar(100)	NO		NULL	
email	varchar(100)	NO	UNI	NULL	
phone	varchar(20)	NO		NULL	

Table Loans-

```sql
-- Create table for Loans
CREATE TABLE Loans (
    loan_id INT AUTO_INCREMENT PRIMARY KEY,
    book_id INT,
    member_id INT,
    loan_date DATE NOT NULL,
    return_date DATE,
    FOREIGN KEY (book_id) REFERENCES Books(book_id),
    FOREIGN KEY (member_id) REFERENCES Members(member_id),
    CHECK (return_date >= loan_date)
);

desc Loans;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
loan_id	int	NO	PRI	NULL	auto_increment
book_id	int	YES	MUL	NULL	
member_id	int	YES	MUL	NULL	
loan_date	date	NO		NULL	
return_date	date	YES		NULL	

Table Reservations-

```sql
-- Create table for Reservations
CREATE TABLE Reservations (
    reservation_id INT AUTO_INCREMENT PRIMARY KEY,
    book_id INT,
    member_id INT,
    reservation_date DATE NOT NULL,
    status ENUM('Pending', 'Completed', 'Cancelled') NOT NULL,
    FOREIGN KEY (book_id) REFERENCES Books(book_id),
    FOREIGN KEY (member_id) REFERENCES Members(member_id)
);
desc Reservations;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
reservation_id	int	NO	PRI	NULL	auto_increment
book_id	int	YES	MUL	NULL	
member_id	int	YES	MUL	NULL	
reservation_date	date	NO		NULL	
status	enum('Pending','Completed','Cancelled')	NO		NULL	

Table Authors-

```
55
56      -- Create table for Authors
57  •   CREATE TABLE Authors (
58          author_id INT AUTO_INCREMENT PRIMARY KEY,
59          author_name VARCHAR(255) NOT NULL,
60          nationality VARCHAR(100)
61      );
62  •   desc Authors;
63
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
author_id	int	NO	PRI	NULL	auto_increment
author_name	varchar(255)	NO		NULL	
nationality	varchar(100)	YES		NULL	

Table Genres-

```
65      -- Create table for Genres
66  •   CREATE TABLE Genres (
67          genre_id INT AUTO_INCREMENT PRIMARY KEY,
68          genre_name VARCHAR(100) NOT NULL
69      );
70  •   desc Genres;
71
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
genre_id	int	NO	PRI	NULL	auto_increment
genre_name	varchar(100)	NO		NULL	

Table Book_Authors-

```
72      -- Create table for Book_Authors (Many-to-Many Relationship)
73  •   CREATE TABLE Book_Authors (
74          book_id INT,
75          author_id INT,
76          PRIMARY KEY (book_id, author_id),
77          FOREIGN KEY (book_id) REFERENCES Books(book_id),
78          FOREIGN KEY (author_id) REFERENCES Authors(author_id)
79      );
80
81  •   desc Book_Authors;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
book_id	int	NO	PRI	NULL	
author_id	int	NO	PRI	NULL	

Alter Table Structures:

- **Add a new column to the Books table:**
  →'Pages' column is added now.

```
83
84  •   ALTER TABLE Books
85      ADD COLUMN pages INT;
86
87  •   select * from Books;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

book_id	title	author	genre	publication_year	ISBN	pages
NULL	NULL	NULL	NULL	NULL	NULL	NULL

- **Modify the column type of phone in Members table:**

→Firstly , we could see we have a column named phone which I wanted to change as Mobile----

```
89 •  select * from Members;
```

Result Grid | Filter Rows: | Edit:

member_id	name	email	phone
NULL	NULL	NULL	NULL

To modify we used ‘alter’: -

```
90
91 •  ALTER TABLE Members
92    CHANGE COLUMN phone mobile VARCHAR(20) NOT NULL;
93
94
95 •  select * from Members;
96
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Cont

member_id	name	email	mobile
NULL	NULL	NULL	NULL

→For displaying all the tables we have constructed under Wipro database,

```
99 •  SHOW TABLES;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Tables_in_wipro
authors
book_authors
books
customer
customers
dept
emp
employee
genres
loans
members
orders
product
reservations

→ **Dropping a Redundant Table:**

I am dropping Genres table with the help of 'drop'-

```
97 •  DROP TABLE Genres;
98
99 •  SHOW TABLES;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Tables_in_wipro
authors
book_authors
books
customer
customers
dept
emp
employee
loans
members
orders
product
reservations

ASSIGNMENT-5

Demonstrate the creation of an index on a table and discuss how it improves query performance. Use a DROP INDEX statement to remove the index and analyze the impact on query execution.

SOLUTION:

→ 1. Creating a table '`employees1`' with the following structure:

```sql
CREATE TABLE employees1 (
    id INT PRIMARY KEY,
    name VARCHAR(100),
    age INT,
    department VARCHAR(50),
    salary DECIMAL(10, 2)
);

select * from employees1;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	name	age	department	salary
NULL	NULL	NULL	NULL	NULL

Inserting some values:-

Limit to 1000 rows

```sql
INSERT INTO employees1 (id, name, age, department, salary) VALUES
(1, 'Alice', 30, 'HR', 60000),
(2, 'Bob', 25, 'Engineering', 75000),
(3, 'Charlie', 35, 'Engineering', 80000),
(4, 'Diana', 28, 'Marketing', 65000),
(5, 'Edward', 40, 'HR', 70000),
-- Add more records as needed to simulate a larger dataset
(6, 'Fay', 26, 'Engineering', 72000),
(7, 'George', 29, 'Marketing', 63000),
(8, 'Hannah', 33, 'HR', 68000);

select * from employees1;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell C

id	name	age	department	salary
1	Alice	30	HR	60000.00
2	Bob	25	Engineering	75000.00
3	Charlie	35	Engineering	80000.00
4	Diana	28	Marketing	65000.00
5	Edward	40	HR	70000.00
6	Fay	26	Engineering	72000.00
7	George	29	Marketing	63000.00
8	Hannah	33	HR	68000.00
NULL	NULL	NULL	NULL	NULL

2. **Creating an Index:** We will create an index on the `department` column to improve the performance of queries filtering by department.

When we create an index on the `department` column, the database constructs a data structure (typically a B-tree) that stores the values of the `department` column in a sorted order. This structure includes pointers to the rows in the `employees1` table where each department value appears.

```
120
121 •   select * from employees1;
122
123 •   CREATE INDEX idx_department ON employees1(department);
```

3. How Index Improves Query Performance

Query Without Index:

Before the index is created, running a query to find employees in a specific department (e.g., 'Engineering') requires the database to scan the entire table, checking each row to see if it matches the department condition.

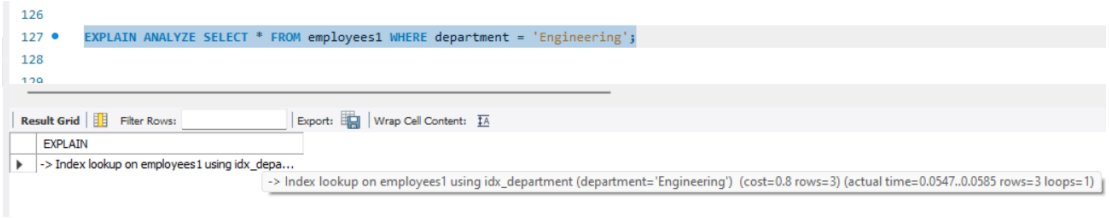
```
124
125 •   EXPLAIN ANALYZE SELECT * FROM employees1 WHERE department = 'Engineering';
126
```

Execution:

Without an index, the query plan will show a full table scan, which can be slow for large tables.

Query With Index:

After creating the index on the `department` column, running the same query allows the database to use the index to quickly locate rows where `department = 'Engineering'`.



Here, the `Index lookup` indicates that the database is using the index to quickly find the rows, significantly improving performance.

4. Dropping the Index

To revert back and see the performance impact without the index:

```
128
129 •  DROP INDEX idx_department ON employees1;
130
```

5. Query Performance After Dropping Index

Run the query again without the index:

```
130
131 •  EXPLAIN ANALYZE SELECT * FROM employees1 WHERE department = 'Engineering';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

EXPLAIN
-> Filter: (employees1.department = 'Engineeri...

Summary: -

- **Creating an Index:** Constructs a sorted data structure on the specified column, allowing faster lookups.
- **With Index:** The database uses the index to quickly locate rows matching the query condition, improving query performance.
- **Without Index:** The database performs a full table scan, which is slower, especially for large tables.
- **Dropping the Index:** Reverts the query performance to the slower full table scan, demonstrating the importance of indexes in query optimization.

Indexes significantly improve query performance for read operations by reducing the need for full table scans.

Dropping an index demonstrates the performance degradation and underscores the importance of indexes in query optimization.

ASSIGNMENT-6

Create a new database user with specific privileges using the CREATE USER and GRANT commands. Then, write a script to REVOKE certain privileges and DROP the user.

SOLUTION:

1. Creating a New User:

First, create a new database user. For this example, we'll create a user named `newuser` with a password `password123`.

→CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password123';

2. Granting Specific Privileges:

Next, grant the user specific privileges. For example, we'll grant `SELECT`, `INSERT`, `UPDATE`, and `DELETE` privileges on a specific database `testdb`.

→GRANT SELECT, INSERT, UPDATE, DELETE ON testdb.* TO 'newuser'@'localhost';

Flush privileges to ensure that they are reloaded and applied:

FLUSH PRIVILEGES;

3. Revoking Certain Privileges:

Now, let's revoke some privileges. Suppose we want to revoke the `INSERT` and `UPDATE` privileges from the user `newuser`.

→REVOKE INSERT, UPDATE ON testdb.* FROM 'newuser'@'localhost';

→FLUSH PRIVILEGES;

4. Dropping the User:

Finally, drop the user from the database.

→DROP USER 'newuser'@'localhost';

Explanation:-

- **CREATE USER**: This command creates a new user with the specified username and password.
- **GRANT**: This command assigns specific privileges to the user on the specified database.
- **FLUSH PRIVILEGES**: This command reloads the grant tables in MySQL to ensure the new privileges take effect.
- **REVOKE**: This command removes specific privileges from the user.
- DROP USER: This command deletes the user from the database.

Important Notes:

- Make sure to execute these commands with a user that has the appropriate administrative privileges, typically a user with GRANT OPTION.
- Replace 'localhost' with the appropriate host if the user will be connecting from a different machine.
- Adjust the privileges and database names as necessary to fit your specific requirements.

ASSIGNMENT-7

Prepare a series of SQL statements to INSERT new records into the library tables, UPDATE existing records with new information, and DELETE records based on specific criteria. Include BULK INSERT operations to load data from an external source.

SOLUTION:

Here are a series of SQL statements to INSERT new records, UPDATE existing records, and DELETE records based on specific criteria. Additionally, I'll include statements for BULK INSERT operations to load data from an external source.

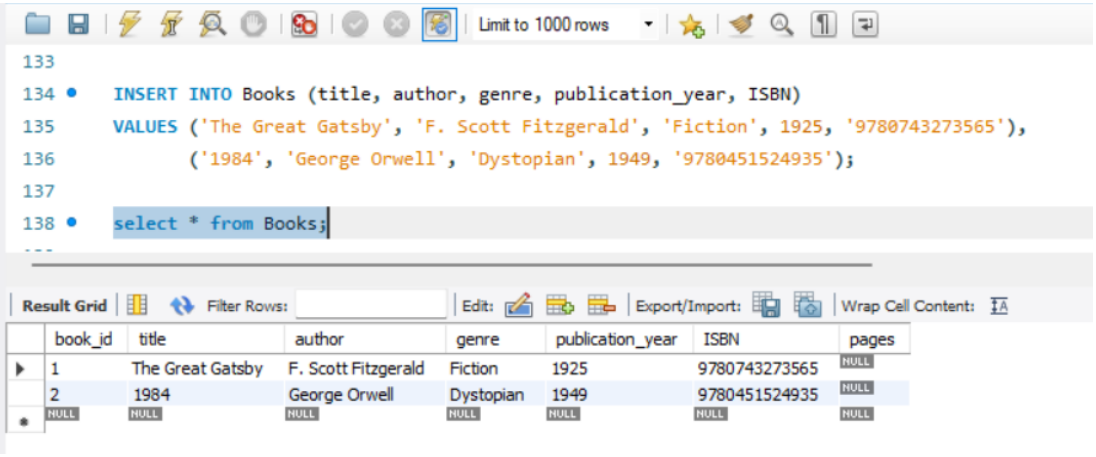
INSERT Statements

1. Insert records into the Books table:

INSERT INTO Books (title, author, genre, publication_year, ISBN)

VALUES ('The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction', 1925, '9780743273565'),

('1984', 'George Orwell', 'Dystopian', 1949, '9780451524935');

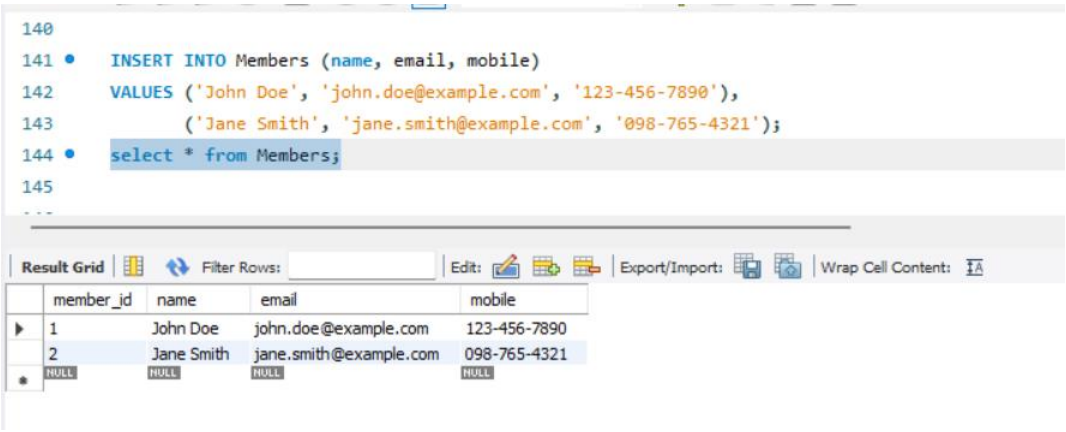


2. Insert records into the Members table:

INSERT INTO Members (name, email, phone)

VALUES ('John Doe', 'john.doe@example.com', '123-456-7890'),

('Jane Smith', 'jane.smith@example.com', '098-765-4321');



3. Insert records into the Loans table:

INSERT INTO Loans (book_id, member_id, loan_date, return_date)

VALUES (1, 1, '2024-05-01', '2024-05-15'),
(2, 2, '2024-05-10', '2024-05-24');

```
146 •  INSERT INTO Loans (book_id, member_id, loan_date, return_date)
147    VALUES (1, 1, '2024-05-01', '2024-05-15'),
148           (2, 2, '2024-05-10', '2024-05-24');
149 •  select * from Loans;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

loan_id	book_id	member_id	loan_date	return_date
1	1	1	2024-05-01	2024-05-15
2	2	2	2024-05-10	2024-05-24
NULL	NULL	NULL	NULL	NULL

UPDATE Statements

→Previously In Members table, it was,

```
144 •  select * from Members;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

member_id	name	email	mobile
1	John Doe	john.doe@example.com	123-456-7890
2	Jane Smith	jane.smith@example.com	098-765-4321
NULL	NULL	NULL	NULL

On Updating a member's email address:

UPDATE Members

SET email = 'john.doe@yahoo.com'

WHERE member_id = 1;

→ Mail id is updated for John.

```
144
145 •  select * from Members;
146 •  UPDATE Members
147    SET email = 'john.doe@yahoo.com'
148    WHERE member_id = 1;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

member_id	name	email	mobile
1	John Doe	john.doe@yahoo.com	123-456-7890
2	Jane Smith	jane.smith@example.com	098-765-4321
NULL	NULL	NULL	NULL

DELETE Statements:

Deleting a member by their email→

We have this information in Members table: -

```
147 •  select * from Members;
```

Result Grid | Filter Rows: | Edit: | Export/Im

member_id	name	email	mobile
1	John Doe	john.doe@yahoo.com	123-456-7890
2	Jane Smith	jane.smith@example.com	098-765-4321
NULL	NULL	NULL	NULL

Let's delete a member by their email -

```
152 •  select * from Members;
153
154 •  DELETE FROM Members
155    WHERE email = 'jane.smith@example.com';
156
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

member_id	name	email	mobile
1	John Doe	john.doe@yahoo.com	123-456-7890
NULL	NULL	NULL	NULL

BULK INSERT Statements

For BULK INSERT operations, you need to have a CSV file ready with the data. Assuming you have a CSV file named `books.csv` with the following structure:

→title,author,genre,publication_year,ISBN

'The Catcher in the Rye','J.D. Salinger','Fiction',1951,'9780316769488'

'To Kill a Mockingbird','Harper Lee','Fiction',1960,'9780061120084'

SQL statement to load data from the CSV file:

→Loading data into the Books table from books.csv:

```
LOAD DATA INFILE '/path/to/books.csv'
INTO TABLE Books
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(title, author, genre, publication_year, ISBN);
```

To check the file path is correctly specified and that the MySQL server has the necessary permissions to read from this path.

We may also need to adjust settings like `secure-file-priv` to allow loading files from specific directories.

Final Notes

- **Permissions:** To Ensure the MySQL server has the necessary file read permissions for BULK INSERT operations.
- **File Path:** To Adjust the file path in the `LOAD DATA INFILE` statement as needed.
- **Security:** For security reasons, to be cautious with file paths and permissions when using `LOAD DATA INFILE`.