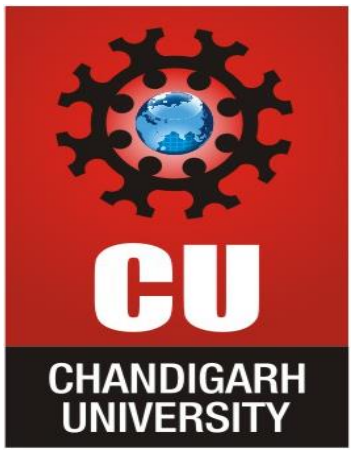


NAME : Simran Chouhan **SEMESTER:**4th

SECTION:24_AIT_KRG-G1 **UID:**24BAI70124

SUBJECT CODE:24CSH-298

FACULTY'S NAME: Mr .SHALABH BHATIA

Topic : Experiment 1

AIM:

To design and implement a **Library Management System database** using appropriate tables, primary keys, foreign keys, and constraints, and to perform **DML operations** along with **DCL commands** such as role creation, privilege granting, and revoking to ensure database security.

Software Requirements

- **Database Management System:**
    - PostgreSQL
- **Database Administration Tool:**
    - pgAdmin

Objective :

To gain practical experience in implementing Data Definition Language (DDL), Data Manipulation Language (DML), and Data Control Language (DCL) operations in a real database environment. This will also include implementing role-based privileges to secure data.

Problem Statement

1. A Library wants to develop a Library Management System database to manage information about books, members, and book issue records efficiently. The database should be designed using appropriate tables, primary keys, foreign keys, and constraints to ensure data integrity.

2. The system must support basic database operations such as inserting records, updating existing data, and deleting obsolete entries. To ensure database security.

3. To ensure database security, a database role named **`Librarian`** must be created. This role should be **password protected** and granted **SELECT, INSERT, and DELETE permissions** on the required tables. The system administrator (**pgAdmin**) should also have the ability to **revoke these permissions when required** using **role-based access control**.

CODE:

```
Create BOOKS table
CREATE TABLE BOOKS (
   BOOK_ID INT PRIMARY KEY,
   BOOK_NAME VARCHAR(50) NOT NULL,
   AUTHOR_NAME VARCHAR(30) NOT NULL,
   BOOK_COUNT INT CHECK (BOOK_COUNT > 0) NOT NULL
);

-- Insert book records
INSERT INTO BOOKS VALUES
(1, 'It Was Meant to Find You', 'Simran', 3),
(2, 'The Silent Library', 'Arjun Rao', 5),
(3, 'Echoes of Time', 'Neha Verma', 2),
(4, 'Beyond the Pages', 'Rohan Mehta', 4),
(5, 'Midnight Thoughts', 'Aditi Singh', 1);

SELECT * FROM BOOKS;
```

```
-- Create LIBRARY_VISITORS table
CREATE TABLE LIBRARY_VISITORS (
    USER_ID INT PRIMARY KEY,
    NAME VARCHAR(30) NOT NULL,
    AGE INT CHECK (AGE >= 17) NOT NULL,
    EMAIL VARCHAR(40) UNIQUE NOT NULL
);

-- Insert visitor records
INSERT INTO LIBRARY_VISITORS VALUES
(501, 'Simran', 20, 'simran@gmail.com'),
(502, 'Ansh Sharma', 21, 'ansh@gmail.com'),
(503, 'Nitin Kumar', 19, 'nitin@gmail.com');

SELECT * FROM LIBRARY_VISITORS;


-- Create BOOK_ISSUE table
CREATE TABLE BOOK_ISSUE (
    BOOK_ISSUE_ID INT PRIMARY KEY,
    USER_ID INT REFERENCES LIBRARY_VISITORS(USER_ID),
    BOOK_ID INT REFERENCES BOOKS(BOOK_ID),
    ISSUE_DATE DATE
);

-- Insert issue records
INSERT INTO BOOK_ISSUE VALUES
(101, 501, 1, '2026-01-09'),
(102, 502, 3, '2026-01-10');

SELECT * FROM BOOK_ISSUE;


-- Create role LIBRARIAN
CREATE ROLE LIBRARIAN
WITH LOGIN PASSWORD 'SIMU';

-- Check current user
SELECT CURRENT_USER;


-- Grant privileges
GRANT SELECT, INSERT, UPDATE, DELETE
ON BOOKS, LIBRARY_VISITORS, BOOK_ISSUE
```

TO LIBRARIAN;

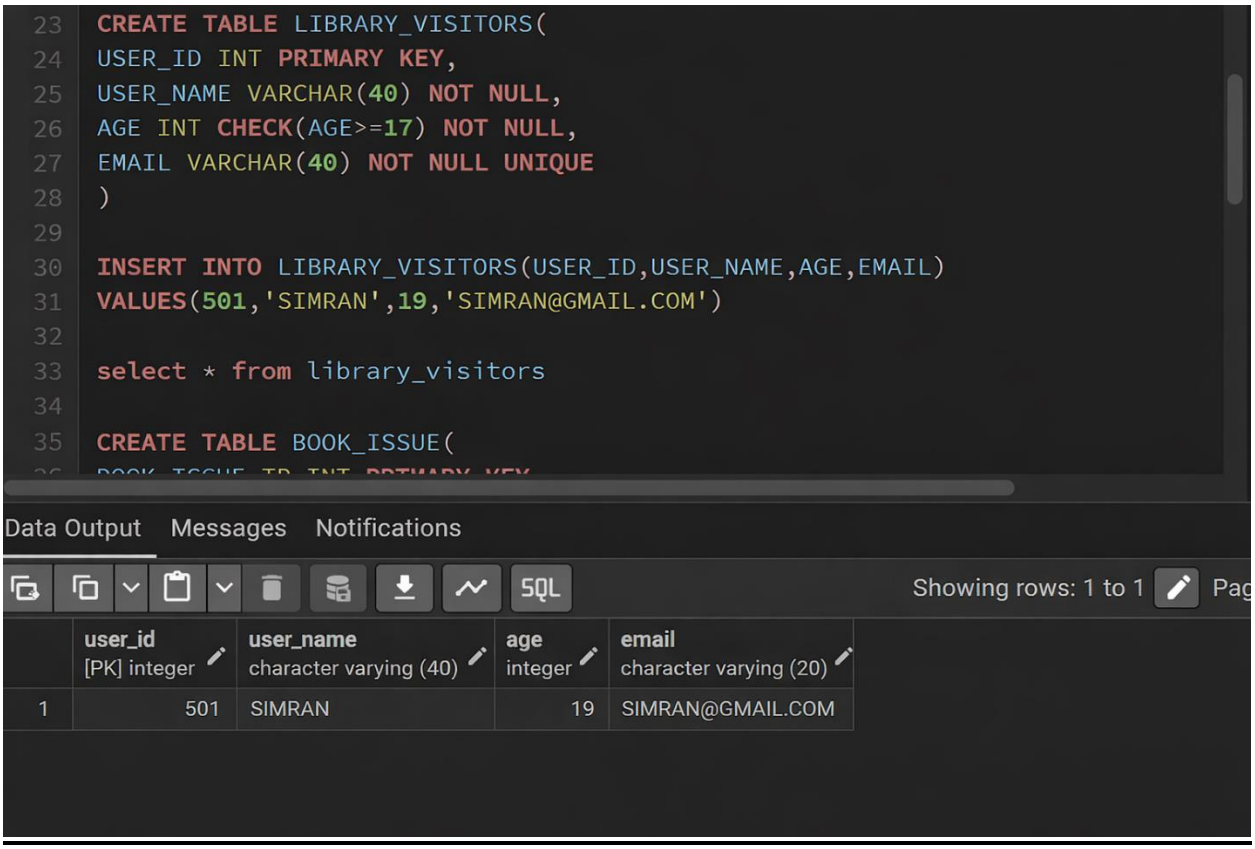
-- Revoke privileges
REVOKE UPDATE, DELETE
ON BOOKS
FROM LIBRARIAN;

-- Final data view
SELECT * FROM BOOKS;
SELECT * FROM LIBRARY_VISITORS;
SELECT * FROM BOOK_ISSUE;

OUTPUT:

	book_issue_id [PK] integer	user_id integer	book_id integer	issue_date date
1	101	501	1	2026-01-09
2	102	502	3	2026-01-10

	book_id [PK] integer	book_name character varying (50)	author_name character varying (30)	book_count integer
1	1	It Was Meant to Find Y...	Simran	3
2	2	The Silent Library	Arjun Rao	5
3	3	Echoes of Time	Neha Verma	2
4	4	Beyond the Pages	Rohan Mehta	4
5	5	Midnight Thoughts	Aditi Singh	1



Learning Outcomes:

After completing this experiment, students will be able to:

- Create database tables using **DDL commands** with appropriate constraints.
- Insert and retrieve records using **DML operations**.
- Establish relationships between tables using **PRIMARY KEY** and **FOREIGN KEY**.
- Implement **role-based access control** using **GRANT** and **REVOKE** commands