

## Aim

The aim of this lab is to design, implement, and manage a relational database schema for an academic institution. This involves applying principles of database normalization, manipulating data, and performing complex queries and access control.

## Objectives

1. **Schema Design:** Create a normalized database schema in 3NF for managing department and course information.
2. **Data Population:** Insert a substantial amount of sample data into the created tables to simulate a real-world scenario.
3. **Advanced Querying:** Formulate a subquery to retrieve specific data by joining and filtering information from multiple tables.
4. **Access Control:** Use Data Control Language (DCL) commands to manage user permissions on the database tables.

## Theory

This lab is built upon several core concepts of relational database management:

- **Database Normalization (3NF):** Normalization is a process of organizing data in a database to reduce redundancy and improve data integrity. Third Normal Form (3NF) requires that a table is in 2NF and that all its columns are non-transitively dependent on the primary key. In this lab, creating separate Departments and Courses tables linked by a foreign key is a key step to achieve 3NF.
- **SQL (Structured Query Language):** The standard language for interacting with relational databases.
  - **DDL (Data Definition Language):** Commands like CREATE TABLE are used to define the database schema.
  - **DML (Data Manipulation Language):** Commands like INSERT INTO are used to add, update, or delete data.
  - **DQL (Data Query Language):** Commands like SELECT are used to retrieve data. This lab uses advanced DQL features like GROUP BY, HAVING, and subqueries.
  - **DCL (Data Control Language):** Commands like GRANT are used to control who has access to the data and what actions they can perform.

## Procedure

1. **Part A - Table Creation:** Execute the CREATE TABLE statements for both

Departments and Courses. Ensure the primary and foreign key constraints are correctly defined.

2. **Part B - Data Insertion:** Run the INSERT INTO statements to populate the tables with at least 5 departments and 10 courses, ensuring that the foreign key constraint is satisfied by linking courses to existing departments.
3. **Part C - Subquery Execution:** Write and execute a SELECT statement that uses a subquery to count the number of courses per department and then filters the results to display only the departments with more than two courses.
4. **Part D - Access Control:** Execute the GRANT statement to assign SELECT permissions on the Courses table to the specified user viewer\_user.

### Code

-- Part A: Create Department and Course Tables with Normalization (3NF)

```
CREATE TABLE Departments (  
    dept_id INT PRIMARY KEY,  
    dept_name VARCHAR(50) UNIQUE NOT NULL  
);
```

```
CREATE TABLE Courses (  
    course_id INT PRIMARY KEY,  
    course_name VARCHAR(100) NOT NULL,  
    dept_id INT,  
    FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)  
);
```

-- Part B: Insert Sample Data into Department and Course Tables

```
INSERT INTO Departments (dept_id, dept_name) VALUES  
    (1, 'Computer Science'),  
    (2, 'Electrical'),  
    (3, 'Mechanical'),  
    (4, 'Civil'),  
    (5, 'Electronics');
```

```
INSERT INTO Courses (course_id, course_name, dept_id) VALUES  
    (101, 'DBMS', 1),  
    (102, 'Operating Systems', 1),  
    (103, 'Algorithms', 1),  
    (104, 'Power Systems', 2),  
    (105, 'Digital Circuits', 2),
```

```
(106, 'Control Systems', 2),
(107, 'Thermodynamics', 3),
(108, 'Fluid Mechanics', 3),
(109, 'Structural Engineering', 4),
(110, 'Surveying', 4),
(111, 'Embedded Systems', 5),
(112, 'VLSI Design', 5);
```

-- Part C: Retrieve Departments Offering More Than Two Courses Using Subquery

```
SELECT dept_name
FROM Departments
WHERE dept_id IN (
    SELECT dept_id
    FROM Courses
    GROUP BY dept_id
    HAVING COUNT(course_id) > 2
);
```

-- Part D: Grant SELECT Access on Courses Table Using DCL

```
GRANT SELECT ON Courses TO viewer_user;
```

## Output

### Part A



The screenshot shows a PostgreSQL command-line interface. The top bar includes a menu icon, a back arrow, a user profile icon for 'ARYAN SAXENA', and buttons for 'AI', 'NEW', 'POSTGRESQL', and 'RUN'. The main area is divided into two panes. The left pane, titled 'commands.sql', contains the following SQL code:

```
1 CREATE TABLE Departments (
2     dept_id INT PRIMARY KEY,
3     dept_name VARCHAR(50) UNIQUE NOT NULL
4 );
5
6 CREATE TABLE Courses (
7     course_id INT PRIMARY KEY,
8     course_name VARCHAR(100),
9     dept_id INT,
10    FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
11 );
```

The right pane shows the output of the commands. It has a section for 'STDIN' with a text input field labeled 'Input for the program ( Optional )'. Below this is an 'Output:' section containing the text:

```
CREATE TABLE
CREATE TABLE
```

## Part B

commands.sql

1 CREATE TABLE Departments (  
2 dept\_id INT PRIMARY KEY,  
3 dept\_name VARCHAR(50) UNIQUE NOT NULL  
4 );  
5  
6 CREATE TABLE Courses (  
7 course\_id INT PRIMARY KEY,  
8 course\_name VARCHAR(100),  
9 dept\_id INT,  
10 FOREIGN KEY (dept\_id) REFERENCES Departments(dept\_id)  
11 );  
12  
13 INSERT INTO Departments (dept\_id, dept\_name) VALUES  
14 (1, 'Computer Science'),  
15 (2, 'Electrical'),  
16 (3, 'Mechanical'),  
17 (4, 'Civil'),  
18 (5, 'Electronics');  
19  
20 INSERT INTO Courses (course\_id, course\_name, dept\_id) VALUES  
21 (101, 'DBMS', 1),  
22 (102, 'Operating Systems', 1),  
23 (103, 'Data Structures', 1),  
24 (104, 'Power Systems', 2),  
25 (105, 'Digital Circuits', 2),  
26 (106, 'Thermodynamics', 3),  
27 (107, 'Fluid Mechanics', 3),  
28 (108, 'Structural Engineering', 4),  
29 (109, 'Surveying', 4),  
30 (110, 'Embedded Systems', 5),  
31 (111, 'VLSI Design', 5);

STDIN  
Input for the program ( Optional)  
Output:  
CREATE TABLE  
CREATE TABLE  
INSERT 0 5  
INSERT 0 12

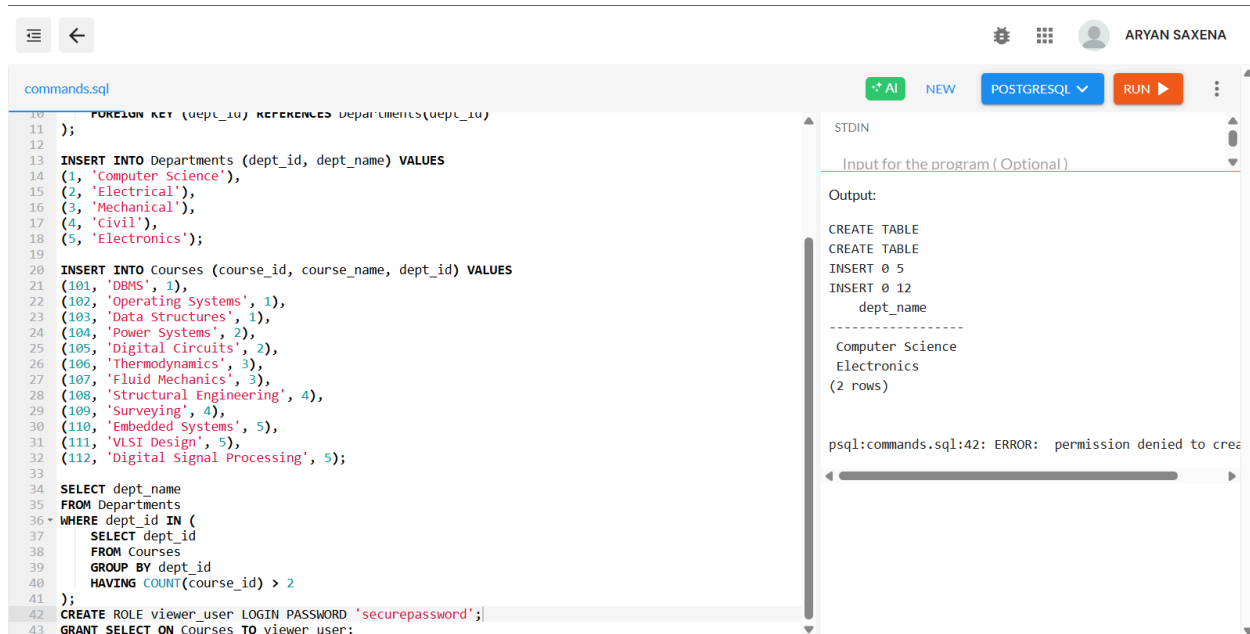
## Part C

commands.sql

8  
9 course\_name VARCHAR(100),  
10 dept\_id INT,  
11 FOREIGN KEY (dept\_id) REFERENCES Departments(dept\_id)  
12 );  
13  
14 INSERT INTO Departments (dept\_id, dept\_name) VALUES  
15 (1, 'Computer Science'),  
16 (2, 'Electrical'),  
17 (3, 'Mechanical'),  
18 (4, 'Civil'),  
19 (5, 'Electronics');  
20  
21 INSERT INTO Courses (course\_id, course\_name, dept\_id) VALUES  
22 (101, 'DBMS', 1),  
23 (102, 'Operating Systems', 1),  
24 (103, 'Data Structures', 1),  
25 (104, 'Power Systems', 2),  
26 (105, 'Digital Circuits', 2),  
27 (106, 'Thermodynamics', 3),  
28 (107, 'Fluid Mechanics', 3),  
29 (108, 'Structural Engineering', 4),  
30 (109, 'Surveying', 4),  
31 (110, 'Embedded Systems', 5),  
32 (111, 'VLSI Design', 5),  
33 (112, 'Digital Signal Processing', 5);  
34  
35 SELECT dept\_name  
36 FROM Departments  
37 WHERE dept\_id IN (  
38 SELECT dept\_id  
39 FROM Courses  
40 GROUP BY dept\_id  
41 HAVING COUNT(course\_id) > 2  
42 );

STDIN  
Input for the program ( Optional)  
Output:  
CREATE TABLE  
CREATE TABLE  
INSERT 0 5  
INSERT 0 12  
dept\_name  
-----  
Computer Science  
Electronics  
(2 rows)

## Part D



The screenshot shows a PostgreSQL command-line interface with a user named ARYAN SAXENA. The interface has a top bar with a menu icon, a back arrow, and the user's name. Below the bar, there are buttons for 'AI', 'NEW', 'POSTGRESQL', and 'RUN'. The main area is divided into two panes. The left pane, titled 'commands.sql', contains the following SQL code:

```
10 FOREIGN KEY (dept_id) REFERENCES departments(dept_id)
11 );
12
13 INSERT INTO Departments (dept_id, dept_name) VALUES
14 (1, 'Computer Science'),
15 (2, 'Electrical'),
16 (3, 'Mechanical'),
17 (4, 'Civil'),
18 (5, 'Electronics');
19
20 INSERT INTO Courses (course_id, course_name, dept_id) VALUES
21 (101, 'DBMS', 1),
22 (102, 'Operating Systems', 1),
23 (103, 'Data Structures', 1),
24 (104, 'Power Systems', 2),
25 (105, 'Digital Circuits', 2),
26 (106, 'Thermodynamics', 3),
27 (107, 'Fluid Mechanics', 3),
28 (108, 'Structural Engineering', 4),
29 (109, 'Surveying', 4),
30 (110, 'Embedded Systems', 5),
31 (111, 'VLSI Design', 5),
32 (112, 'Digital Signal Processing', 5);
33
34 SELECT dept_name
35 FROM Departments
36 WHERE dept_id IN (
37     SELECT dept_id
38     FROM Courses
39     GROUP BY dept_id
40     HAVING COUNT(course_id) > 2
41 );
42 CREATE ROLE viewer_user LOGIN PASSWORD 'securepassword';
43 GRANT SELECT ON Courses TO viewer_user;
```

The right pane shows the output of the commands. It starts with 'STDIN' and 'Input for the program (Optional)'. The output is as follows:

```
Output:
CREATE TABLE
CREATE TABLE
INSERT 0 5
INSERT 0 12
      dept_name
-----
Computer Science
Electronics
(2 rows)

psql:commands.sql:42: ERROR: permission denied to create
```

## Learning Outcomes (3)

Upon completion of this lab, students will be able to:

1. **Database Design:** Apply normalization principles to design and create a relational database schema with correct primary and foreign key constraints.
2. **Data Management:** Utilize SQL DML commands to effectively insert and manage data while respecting table relationships.
3. **Data Retrieval and Security:** Write complex queries using subqueries and GROUP BY clauses, and use DCL commands to control user access and permissions.