Name: Madhipati Saran Sri Datha

UID: 23BCC70007

Class/Sec: 23BCC-1-A

EXPERIMENT-1

AIM

To design and implement a normalized database schema (up to **Third Normal Form**) for managing academic departments and courses. The experiment involves populating the schema, querying it using a subquery to identify departments with more than two courses, and implementing user access control using Data Control Language (DCL).

THEORY

- **Normalization (3NF):** The process of organizing columns and tables in a relational database to minimize data redundancy. A relation is in **Third Normal Form (3NF)** if it is in Second Normal Form (2NF) and has no transitive functional dependencies. This means every non-prime attribute is non-transitively dependent on every candidate key, ensuring data integrity.
- **Relational Model:** A data model where data is stored in tables (relations). Our design consists of:
    - A `Departments` table for unique departmental data.
    - A `Courses` table where each course is linked to a single department via a **foreign key**, enforcing referential integrity.
- **Subquery:** A query nested inside another SQL query (e.g., within a `WHERE` or `FROM` clause). Subqueries are executed first, and their result is used by the outer query to filter or evaluate data.
- **Data Control Language (DCL):** A subset of SQL used to control access to data in the database. The `GRANT` command is used to give specific user privileges (e.g., `SELECT`, `INSERT`) on database objects.

PROCEDURE

The experiment was conducted by executing a series of SQL statements in a PostgreSQL environment.

1. Schema Creation: The initial step was to create the database schema. To ensure a clean execution, any pre-existing tables were dropped. The `Departments` table was created with a primary key `dept_id`. The `Courses` table was then created with its own primary key and a foreign key, `dept_id`, which references the `Departments` table. The `ON DELETE CASCADE` constraint was included to automatically remove a department's courses if the department itself is deleted.

SQL

```
-- Drop if exists for clean re-execution
DROP TABLE IF EXISTS Courses;
DROP TABLE IF EXISTS Departments;

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

-- Create Courses table
CREATE TABLE Courses (
 course_id INT PRIMARY KEY,
 course_name VARCHAR(100) NOT NULL,
 dept_id INT NOT NULL,
 FOREIGN KEY (dept_id) REFERENCES Departments(dept_id) ON
DELETE CASCADE
);
```

2. Data Population: The tables were populated with meaningful sample data to simulate a real-world academic environment.

SQL

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

-- Insert Courses, ensuring one department has more than
two
```

```
INSERT INTO Courses (course_id, course_name, dept_id)
VALUES
(101, 'Database Management Systems', 1),
(102, 'Operating Systems', 1),
(103, 'Advanced Programming', 1),
(104, 'Power Systems', 2),
(105, 'Digital Circuits', 2),
(106, 'Thermodynamics', 3),
(107, 'Fluid Mechanics', 3),
(108, 'Structural Engineering', 4),
(109, 'Surveying', 4),
(110, 'Embedded Systems', 5);
```

3. Data Querying with a Subquery: A query was executed to find departments offering more than two courses. A subquery was used to first identify the `dept_ids` from the `Courses` table that satisfy this condition (`COUNT(*) > 2`). The outer query then used this list of IDs to retrieve the corresponding department names from the `Departments` table.

SQL

```
-- Query to find departments offering more than two
courses
SELECT dept_name
FROM Departments
WHERE dept_id IN (
  SELECT dept_id
  FROM Courses
  GROUP BY dept_id
  HAVING COUNT(*) > 2
);
```

4. Access Control Implementation: Finally, a DCL command was executed to manage user permissions. A user role named `viewer_user` was granted read-only (`SELECT`) access to the `Courses` table, preventing this user from modifying the table's data.

SQL

```
-- Grant SELECT access to a user
GRANT SELECT ON TABLE Courses TO viewer_user;
```

OUTPUTS & OBSERVATIONS

The execution of the subquery in Step 3 yielded the following result, correctly identifying the 'Computer Science' department as the only one offering more than two courses based on the sample data.

dept_name
Computer Science

Export to Sheets

The `GRANT` command in Step 4 executed successfully, confirming that privileges were applied. Any subsequent attempts by `viewer_user` to `INSERT`, `UPDATE`, or `DELETE` data from the `Courses` table would result in a permission denied error, as expected.

CONCLUSION

This experiment successfully demonstrated the design and implementation of a database schema normalized to **3NF**. The use of **primary and foreign keys** effectively maintained referential integrity between the `Departments` and `Courses` tables. The experiment also validated the utility of **subqueries** with `GROUP BY` and `HAVING` clauses for performing complex data analysis across related tables. Finally, the successful implementation of access control using the `GRANT` statement highlighted a fundamental aspect of database security. The overall procedure serves as a practical model for efficient schema modeling and data management.