EXPERIMENT – 2

Name: Kavya Rastogi
UID: 23BCC70019
Class/Sec: 23BCC-1-A

AIM

To design and implement a normalized relational database schema (up to Third Normal Form) for handling academic departments and their respective courses. The task includes populating the schema with sample data, using a subquery to retrieve departments offering more than two courses, and applying user access control via Data Control Language (DCL).

THEORY

- **Normalization (3NF):**
  Normalization is the method of organizing data within tables to eliminate redundancy and maintain integrity. A schema is in **Third Normal Form (3NF)** when:
  1. It is already in Second Normal Form (2NF).
  2. No non-key attribute is transitively dependent on any candidate key.
     This ensures that all attributes depend only on the primary key, avoiding anomalies during data manipulation.
- **Relational Model:**
  Data in relational databases is stored in **tables (relations)**. Each table represents an entity, and relationships are maintained using **keys** (primary keys and foreign keys).
  In this experiment:
  1. The **Departments** table stores information about each department uniquely.
  2. The **Courses** table stores course details, each linked to a department via a foreign key.
- **Subquery:**
  A subquery is an SQL query embedded within another query. The inner query executes first, and its output is used by the outer query to filter or retrieve results.
- **Data Control Language (DCL):**
  DCL commands are used to manage database security and access rights. For instance, the `GRANT` command allows specific users to perform actions (e.g., `SELECT`, `INSERT`), ensuring controlled access to database objects.

PROCEDURE

This experiment was executed in a PostgreSQL environment following these steps:

---

1. Schema Creation

The schema was initialized by dropping existing tables to avoid conflicts. The **Departments** table was created with a primary key (`dept_id`). The **Courses** table was created with a primary key (`course_id`) and a foreign key (`dept_id`) referencing the **Departments** table, with `ON DELETE CASCADE` to maintain referential integrity.

```
-- Drop tables if they already exist
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

Sample data was inserted into both tables. At least one department was assigned more than two courses to test the query.

```
-- Insert Departments
INSERT INTO Departments (dept_id, dept_name) VALUES
(1, 'Information Technology'),
(2, 'Electronics and Communication'),
(3, 'Mechanical Engineering'),
(4, 'Civil Engineering'),
(5, 'Chemical Engineering');

-- Insert Courses
INSERT INTO Courses (course_id, course_name, dept_id) VALUES
(201, 'Data Structures', 1),
(202, 'Web Technologies', 1),
(203, 'Computer Networks', 1),
(204, 'Digital Signal Processing', 2),
(205, 'Microprocessors', 2),
(206, 'Heat Transfer', 3),
(207, 'Manufacturing Processes', 3),
(208, 'Building Materials', 4),
(209, 'Geotechnical Engineering', 4),
```

```
(210, 'Organic Chemistry', 5);
```

---

3. Data Querying with Subquery

To identify departments offering more than two courses, a subquery was used. The subquery first identified `dept_id` values with more than two associated courses, and the outer query retrieved corresponding department names.

```
-- Find departments with 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 (DCL Implementation)

A database role named `readonly_user` was created and granted permission to **only view (SELECT)** data from the **Courses** table.

```
-- Grant read-only access on Courses table
GRANT SELECT ON TABLE Courses TO readonly_user;
```

OUTPUTS & OBSERVATIONS

- The subquery execution returned the department:

```
dept_name
---------------
Information Technology
```

This confirms that only the **Information Technology** department had more than two courses.

- The `GRANT` command successfully applied read-only privileges. Attempts by `readonly_user` to `INSERT`, `UPDATE`, or `DELETE` from the **Courses** table resulted in permission denial, confirming proper access control.

CONCLUSION

This experiment demonstrated:

1. The design and implementation of a database schema normalized to **Third Normal Form (3NF)**.

2. Proper enforcement of **referential integrity** using primary and foreign keys.
3. The effectiveness of **subqueries** in identifying aggregate conditions across tables.
4. The importance of **access control** in securing data, successfully implemented with the `GRANT` command.

Overall, the experiment illustrates a practical and efficient approach to schema modeling, querying, and database security.