PARIJAT PAL | 23BCC70037 | 23BCC1-A |ADBMS

EXPERIMENT 2.1

Title: Create Department and Course Tables with Normalization (up to 3NF)

Description:
You are designing an academic schema to manage departments and the courses they offer. Normalize the design into 3NF using two tables:

`Departments`
 and
`Courses`
. Ensure each course belongs to exactly one department, and department names are not duplicated.

Input Format:

- Table **Departments** with columns:
  - `dept_id`

    (INT, Primary Key)
  - `dept_name`

    (VARCHAR(50))
- Table **Courses** with columns:
  - `course_id`

    (INT, Primary Key)
  - `course_name`

    (VARCHAR(100))
  - `dept_id`

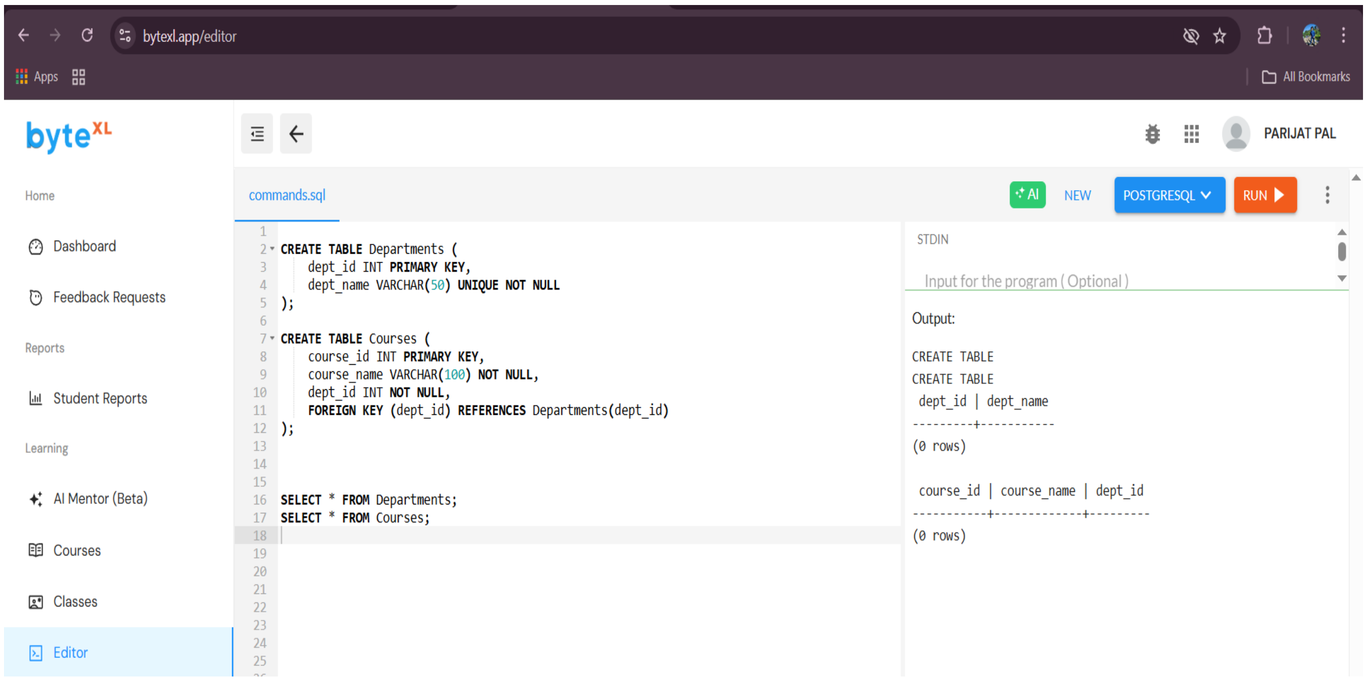
    (INT, Foreign Key referencing Departments)

Constraints:
- Each course must be linked to a valid department.
- Department names must not repeat.
- All data should be in 3NF.

Query:

```
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 NOT NULL,
    FOREIGN KEY (dept_id) REFERENCES Departments(dept_id));
SELECT * FROM Departments;
SELECT * FROM Courses;
```

Output:



```
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 NOT NULL,
    FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
);


SELECT * FROM Departments;
SELECT * FROM Courses;
```

Output:

```
CREATE TABLE
CREATE TABLE
 dept_id | dept_name
---------+-----------
(0 rows)

 course_id | course_name | dept_id
-----------+-------------+---------
(0 rows)
```

Learning Outcome:

Design tables with primary and foreign keys.

Maintain data integrity by enforcing constraints like PRIMARY KEY, FOREIGN KEY, and UNIQUE.

PARIJAT PAL | 23BCC70037 | 23BCC1-A |ADBMS

EXPERIMENT 2.2

Title: Insert Sample Data into Department and Course Tables

Description:

After defining the schema, your task is to populate the

`Departments`
and
`Courses`
tables with at least 5 departments and 10 courses. Ensure that multiple courses are associated with each department.

Input Format:

Pre-existing **Departments** and **Courses** table structures from Problem 2A.

Output Format:

No output — just successful insertion of sample data.

Constraints:

- Use meaningful department names like "Computer Science", "Electrical", "Mechanical", etc.
- Use course names like “DBMS”, “Circuits”, “Thermodynamics” etc.
- Use valid foreign keys linking courses to department.

Sample Input:

Departments

dept_id	dept_name
1	Computer Science
2	Electrical
3	Mechanical
4	Civil
5	Electronics

Courses

course_id	course_name	dept_id
101	DBMS	1
102	Operating Systems	1
103	Power Systems	2
104	Digital Circuits	2
105	Thermodynamics	3
106	Fluid Mechanics	3
107	Structural Engineering	4
108	Surveying	4
109	Embedded Systems	5
110	VLSI Design	5

Query:

```
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, 'Power Systems', 2),
(104, 'Digital Circuits', 2),
(105, 'Thermodynamics', 3),
```

(106, 'Fluid Mechanics', 3),

(107, 'Structural Engineering', 4),

(108, 'Surveying', 4),

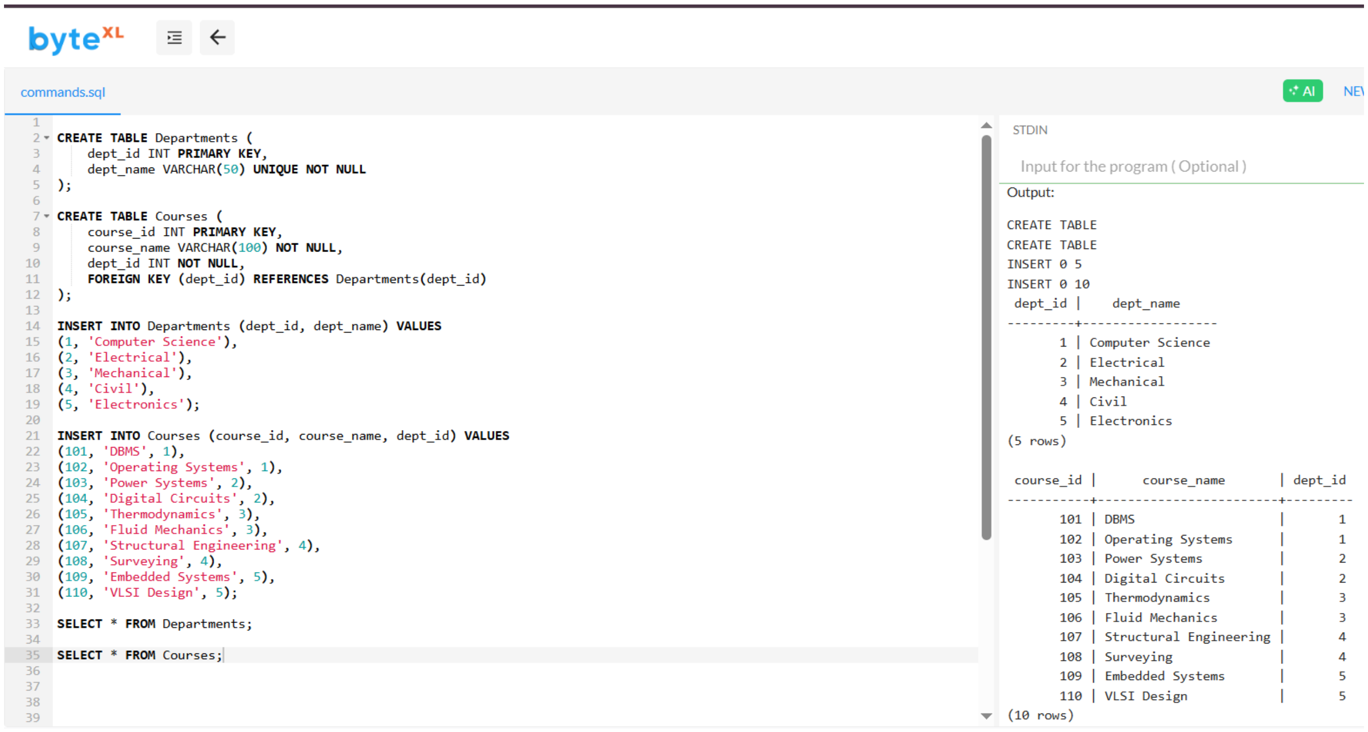
(109, 'Embedded Systems', 5),

(110, 'VLSI Design', 5);

SELECT * FROM Departments;

SELECT * FROM Courses;

Output:



```sql
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 NOT NULL,
    FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
);

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, 'Power Systems', 2),
(104, 'Digital Circuits', 2),
(105, 'Thermodynamics', 3),
(106, 'Fluid Mechanics', 3),
(107, 'Structural Engineering', 4),
(108, 'Surveying', 4),
(109, 'Embedded Systems', 5),
(110, 'VLSI Design', 5);

SELECT * FROM Departments;

SELECT * FROM Courses;
```

Output:

```
CREATE TABLE
CREATE TABLE
INSERT 0 5
INSERT 0 10
 dept_id |    dept_name
---------+------------------
       1 | Computer Science
       2 | Electrical
       3 | Mechanical
       4 | Civil
       5 | Electronics
(5 rows)

 course_id |      course_name       | dept_id
-----------+------------------------+---------
       101 | DBMS                   |       1
       102 | Operating Systems      |       1
       103 | Power Systems          |       2
       104 | Digital Circuits       |       2
       105 | Thermodynamics         |       3
       106 | Fluid Mechanics        |       3
       107 | Structural Engineering |       4
       108 | Surveying              |       4
       109 | Embedded Systems       |       5
       110 | VLSI Design            |       5
(10 rows)
```

Learning Outcome:

Understand how to associate courses with departments using foreign keys.

Learn to write and run INSERT and SELECT SQL statements.

PARIJAT PAL | 23BCC70037 | 23BCC1-A |ADBMS

EXPERIMENT 2.3

Title: Retrieve Departments Offering More Than Two Courses Using Subquery

Description:

Given the

`Departments`
 and
`Courses`
 tables, write a subquery to find the names of departments that offer more than **two courses**.

Input Format:

- Table **Departments** with columns:
  - `dept_id`

    (INT, Primary Key)
  - `dept_name`

    (VARCHAR(50))
- Table **Courses** with columns:
  - `course_id`

    (INT, Primary Key)
  - `course_name`

    (VARCHAR(100))
  - `dept_id`

    (INT, Foreign Key referencing Departments)

Output Format:

A list of department names (

`dept_name`
) that offer more than two courses.

Constraints:

- A department must be present in both tables.
- Each course must belong to one department only.

Sample Output:

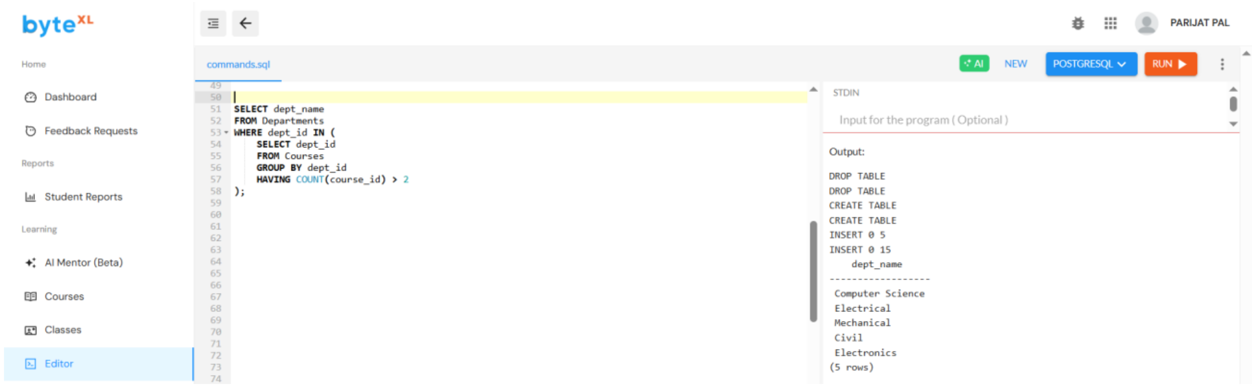
dept_name
Computer Science
Electrical
Mechanical
Civil
Electronics

(If a department had only 1 or 2 courses, it wouldn't appear in the result.)

Query:

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

Output:



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

Output:

```
DROP TABLE
DROP TABLE
CREATE TABLE
CREATE TABLE
INSERT 0 5
INSERT 0 15
    dept_name
------------------
 Computer Science
 Electrical
 Mechanical
 Civil
 Electronics
(5 rows)
```

Learning Outcome:

We learned how to use a subquery inside a WHERE ... IN clause to filter results based on another table.

We understood how subqueries return intermediate results that feed into a main query.

PARIJAT PAL | 23BCC70037 | 23BCC1-A |ADBMS

EXPERIMENT 2.4

Title: Grant SELECT Access on Courses Table Using DCL

Description:

You are required to allow a user named

`viewer_user`
to only read the data from the
`Courses`
table. Use a DCL command to grant this access.

Input Format:

- Table **Courses** already exists.
- User **viewer_user** exists.

Output Format:

No output — just successful execution of a DCL command.

Constraints:

- Only SELECT access must be granted.
- Access should be restricted to the

  `Courses`

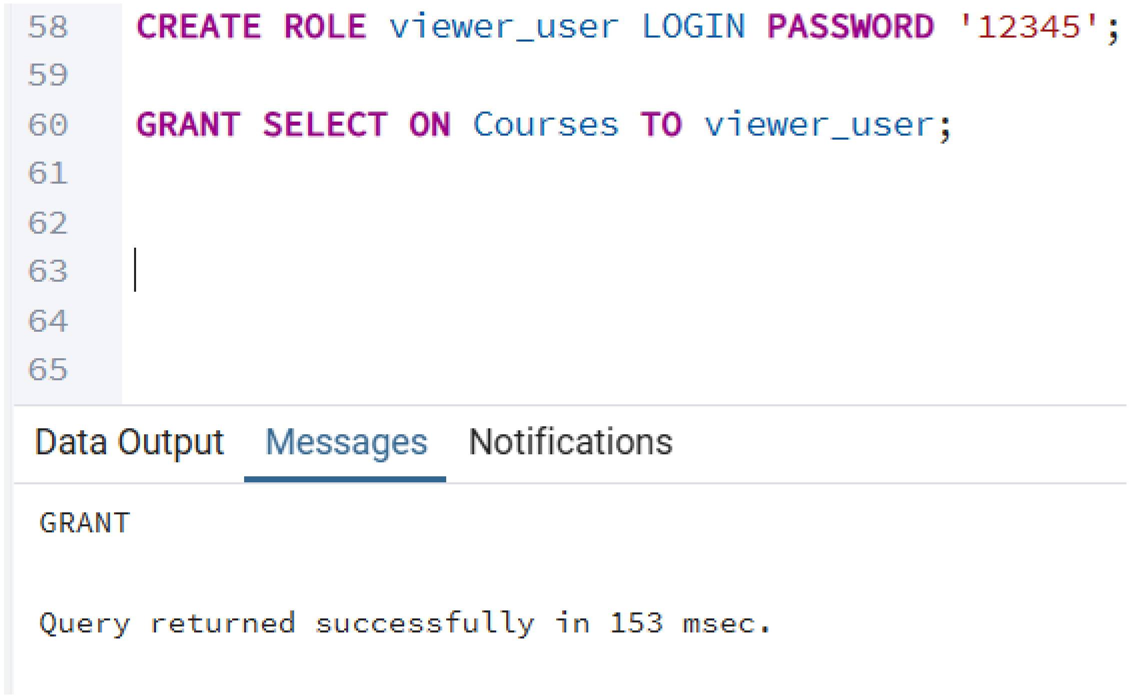
  table only.

Query:

```
CREATE ROLE viewer_user LOGIN PASSWORD '12345';


GRANT SELECT ON Courses TO viewer_user;
```

Output:



```
CREATE ROLE viewer_user LOGIN PASSWORD '12345';

GRANT SELECT ON Courses TO viewer_user;
```

```
GRANT

Query returned successfully in 153 msec.
```

Learning Outcome:

Learned to use the GRANT command to give read-only (SELECT) access on specific tables.

Understood that users must exist before granting permissions.

Recognized the need for proper privileges to create users.