

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:

The screenshot shows the byteXL editor interface. The left sidebar contains a navigation menu with options: Home, Dashboard, Feedback Requests, Reports, Student Reports, Learning, AI Mentor (Beta), Courses, Classes, and Editor (selected). The main editor area displays a file named 'commands.sql' with the following SQL code:

```
1  
2 - CREATE TABLE Departments (  
3     dept_id INT PRIMARY KEY,  
4     dept_name VARCHAR(50) UNIQUE NOT NULL  
5 );  
6  
7 - CREATE TABLE Courses (  
8     course_id INT PRIMARY KEY,  
9     course_name VARCHAR(100) NOT NULL,  
10    dept_id INT NOT NULL,  
11    FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)  
12 );  
13  
14  
15  
16 SELECT * FROM Departments;  
17 SELECT * FROM Courses;  
18  
19  
20  
21  
22  
23  
24  
25
```

On the right side of the editor, there is a 'STDOUT' panel. It shows the output of the executed queries:

```
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.

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:

commands.sql

```

1
2 CREATE TABLE Departments (
3     dept_id INT PRIMARY KEY,
4     dept_name VARCHAR(50) UNIQUE NOT NULL
5 );
6
7 CREATE TABLE Courses (
8     course_id INT PRIMARY KEY,
9     course_name VARCHAR(100) NOT NULL,
10    dept_id INT NOT NULL,
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, 'Power Systems', 2),
25 (104, 'Digital Circuits', 2),
26 (105, 'Thermodynamics', 3),
27 (106, 'Fluid Mechanics', 3),
28 (107, 'Structural Engineering', 4),
29 (108, 'Surveying', 4),
30 (109, 'Embedded Systems', 5),
31 (110, 'VLSI Design', 5);
32
33 SELECT * FROM Departments;
34
35 SELECT * FROM Courses;
36
37
38
39

```

STDIN

Input for the program (Optional)

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.

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:

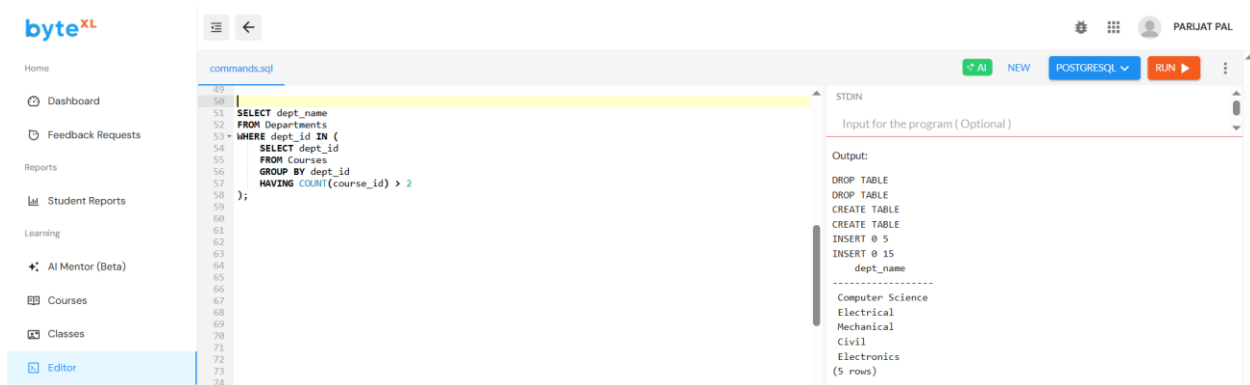
dept_name
Computer Science
Electrical
Mechanical
Civil
Electronics

(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:



The screenshot displays the byteXL SQL editor interface. On the left is a sidebar with navigation links: Home, Dashboard, Feedback Requests, Reports, Student Reports, Learning, AI Mentor (Beta), Courses, Classes, and Editor (highlighted). The main editor area shows a SQL query in the 'commands.sql' file:

```
49  
50  
51 SELECT dept_name  
52 FROM Departments  
53 WHERE dept_id IN (  
54     SELECT dept_id  
55     FROM Courses  
56     GROUP BY dept_id  
57     HAVING COUNT(course_id) > 2  
58 );  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74
```

On the right, the 'Output' pane shows the results of the query:

```
STDIN  
Input for the program ( Optional )  
  
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.

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:

```
58 CREATE ROLE viewer_user LOGIN PASSWORD '12345';
59
60 GRANT SELECT ON Courses TO viewer_user;
61
62
63 |
64
65
```

Data Output	Messages	Notifications
-------------	----------	---------------

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.