Group Details

- Each group must consist of **4 students**.
- **All 4 members must actively participate** in the project and final presentation.
- Each member must have a **clear and separate responsibility** within the project.
- The project work must be **properly divided and shared equally** among all group members.
- Each member is responsible for completing and understanding their assigned part of the project.
- **All members must understand the complete project**, not only their assigned part.
- During the presentation and viva, **any member may be asked questions about any part of the project**.
- Each member should be able to explain the project's **database design, ERD, tables, keys, normalization, PostgreSQL implementation, and SQL queries**.
- The contribution of all group members may be considered during the **project evaluation and viva**.

University Student Management & Course Enrollment System

1. Project Objective

Students will design and implement a **University Student Management System** in PostgreSQL.

The project should demonstrate the complete journey:

Real-World Problem ↓ Requirements ↓ Entities & Attributes ↓ ERD ↓ ERD → Relations/Tables ↓ Primary & Foreign Keys ↓ Normalization (1NF → 2NF → 3NF) ↓ PostgreSQL Database Implementation ↓ SQL Queries ↓ Views & Reports ↓ Optional: MS Access / Front-End ↓ Modern Database Concepts: NoSQL & Vector Databases

---

2. Real-World Scenario

A university wants to maintain information about:

- Departments
- Programs
- Students
- Teachers
- Courses
- Semesters
- Class sections
- Student enrollment
- Marks/results
- Attendance

The system should help the university answer questions such as:

- Which students belong to a particular department?
- Which courses are offered in a semester?
- Which teacher is teaching a course?

- Which students are enrolled in a course?
- What marks did a student obtain?
- What is a student's GPA?
- How many students are enrolled in each course?
- Which students have not passed a course?
- Which courses are being taught by a particular teacher?

---

3. Suggested Entities

Identify the entities themselves from the requirements before creating the ERD.

A suitable final design could contain:

Core Entities

1. **Department**
2. **Program**
3. **Student**
4. **Teacher**
5. **Course**
6. **Semester**
7. **Section**
8. **Enrollment**
9. **Result**
10. **Attendance**

Optional Entities

11. Classroom
12. Building
13. Prerequisite
14. Student Advisor

---

4. Important Relationships

Identify and explain relationships such as:

```
Department
    |
    | 1 : M
    ↓
Program
    |
    | 1 : M
```

```
    ↓
Student
```

and:

```
Department
    |
    | 1 : M
    ↓
Teacher
```

```
Department
    |
    | 1 : M
    ↓
Course
```

```
Course
    |
    | 1 : M
    ↓
Section
    |
    | M : 1
    ↓
Semester
```

The important **many-to-many relationship** is:

```
Student M : N Course
```

which should be resolved through:

```
Student
   |
   | 1:M
   ↓
Enrollment
   ↑
   | M:1
   |
Course
```

---

5. Suggested Database Tables

A possible final relational model:

Department

```
department
-----------
department_id PK
department_name
```

Program

```
program
-------
program_id PK
program_name
degree_level
department_id FK
```

Student

```
student
-------
student_id PK
registration_no UNIQUE
student_name
email
date_of_birth
gender
program_id FK
admission_year
```

Teacher

```
teacher
-------
teacher_id PK
teacher_name
email
designation
department_id FK
```

Course

```
course
------
course_id PK
course_code UNIQUE
course_title
credit_hours
department_id FK
```

Semester

```
semester
--------
semester_id PK
semester_name
academic_year
start_date
end_date
```

Section

```
section
-------
section_id PK
course_id FK
teacher_id FK
semester_id FK
section_name
room_no
```

Enrollment

```
enrollment
----------
enrollment_id PK
student_id FK
section_id FK
enrollment_date
status
```

Result

```
result
------
result_id PK
enrollment_id FK
mid_marks
```

```
final_marks
total_marks
grade
grade_point
```

Attendance

```
attendance
----------
attendance_id PK
enrollment_id FK
attendance_date
status
```

---

6. Important Database Concepts Must Demonstrate

A. Primary Key

Students should explain:

> A primary key uniquely identifies each record in a table.

Example:

```
student_id INTEGER PRIMARY KEY
```

B. Foreign Key

Example:

```
program_id INTEGER REFERENCES program(program_id)
```

Explain how foreign keys maintain relationships between tables.

C. Candidate Key / Unique Key

For example:

```
registration_no VARCHAR(20) UNIQUE
```

a student's registration number can uniquely identify a student even though the database uses `student_id` as the primary key.

7. Normalization Requirement

This should be an important part of the project.

Unnormalized Data

For example:

Reg No	Student	Department	Course 1	Course 2	Course 3
2024-CS-001	Ali	CS	DB	OOP	AI

Then demonstrate why this design is problematic.

1NF

Remove repeating groups and make values atomic.

2NF

Explain partial dependency and separate data appropriately.

3NF

Remove transitive dependencies.

Finally arrive at:

```
Department
Program
Student
Teacher
Course
Semester
Section
Enrollment
Result
Attendance
```

Important: Show the transformation rather than simply saying "the database is in 3NF."

8. PostgreSQL Implementation

Must actually create the database using PostgreSQL.

For example:

```
CREATE DATABASE university_db;
```

Then create tables using:

```
CREATE TABLE department (
    department_id SERIAL PRIMARY KEY,
    department_name VARCHAR(100) NOT NULL UNIQUE
);
```

Students should demonstrate:

- `CREATE DATABASE`
- `CREATE TABLE`
- Data types
- `PRIMARY KEY`
- `FOREIGN KEY`
- `UNIQUE`
- `NOT NULL`
- `CHECK`
- `DEFAULT`

---

9. Data Insertion

Should insert realistic sample data.

Minimum suggested data:

Table	Minimum Records
Department	4
Program	6
Student	30
Teacher	10
Course	15
Semester	4
Section	20
Enrollment	100
Result	80
Attendance	100+

---

10. SQL Query Requirements

Each group should prepare at least **15 SQL queries**, covering different concepts.

Basic SELECT

```sql
SELECT * FROM student;
```

Filtering

```sql
SELECT *
FROM student
WHERE admission_year = 2024;
```

Sorting

```sql
SELECT *
FROM student
ORDER BY student_name;
```

Aggregate Functions

```sql
SELECT COUNT(*)
FROM student;
```

GROUP BY

```sql
SELECT program_id, COUNT(*)
FROM student
GROUP BY program_id;
```

JOIN

```sql
SELECT s.student_name, p.program_name
FROM student s
JOIN program p
ON s.program_id = p.program_id;
```

Multiple JOINs

Demonstrate queries involving 3–5 tables.

HAVING

```
SELECT program_id, COUNT(*)
FROM student
GROUP BY program_id
HAVING COUNT(*) > 5;
```

Subquery

`IN`

`EXISTS`

`CASE`

Date functions

Aggregate calculations

---

11. Advanced PostgreSQL Component

To make this a **PostgreSQL** project rather than simply a generic SQL project, require students to demonstrate at least **3 PostgreSQL-specific features**.

For example:

1. `SERIAL` / Identity

```
student_id INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY
```

2. Views

```
CREATE VIEW student_course_view AS
SELECT ...
```

3. Functions

```
CREATE FUNCTION ...
```

4. Triggers

For example, automatically record changes to student information.

5. Indexes

```
CREATE INDEX idx_student_registration
ON student(registration_no);
```

6. PostgreSQL-specific data types

For example:

```
DATE
TIMESTAMP
BOOLEAN
NUMERIC
JSONB
```

---

12. Database Forms and Reports

Connect th database with **MS Access**, to make front-end components (Forms and Reports).

They can demonstrate:

Forms

- Student Registration Form
- Course Form
- Teacher Form
- Enrollment Form

Queries

- Student Course List
- Semester Enrollment
- Student Result
- Course Enrollment

Reports

- Student Enrollment Report
- Course-wise Student Report
- Student Result Report
- Department-wise Student Report

The important concept to explain is:

> **PostgreSQL = Database/Backend MS Access = Possible Front-End**

a database system can have separate front-end and back-end components.

---

13. Final Presentation Structure

Each group (2-3 students) can present the project in this sequence:

Part 1 — Problem

What real-world problem are we solving?

Part 2 — Requirements

What information does the university need to maintain?

Part 3 — Entities

Identify:

```
Student
Department
Program
Course
Teacher
Semester
...
```

Part 4 — ERD

Show entities, attributes and relationships.

Part 5 — Mapping

Demonstrate:

```
ERD → Relations/Tables
```

Part 6 — Keys

Explain:

```
Primary Key
Foreign Key
Candidate Key
Unique Key
Composite Key
```

Part 7 — Normalization

Show:

```
Unnormalized
      ↓
1NF
      ↓
2NF
      ↓
3NF
```

Part 8 — PostgreSQL

Demonstrate the actual database.

Part 9 — SQL

Run important queries live.

Part 10 — Reports

Show meaningful outputs.

Part 11 — Advanced Feature

Demonstrate:

```
View / Function / Trigger / Index / JSONB
```

Part 12 — Modern Databases

Briefly explain:

```
RDBMS
   ↓
NoSQL
   ↓
Vector Database
   ↓
AI / Semantic Search
```

14. Suggested Final Project Deliverables

Each group(2-3 students) should submit:

1. **Project Proposal**
2. **Requirements Document**
3. **ERD**
4. **Relational Schema**
5. **Normalization Document**

6. **PostgreSQL Database**
7. **SQL Script**
8. **Sample Data**
9. **15+ SQL Queries**
10. **3+ PostgreSQL Features**
11. **Reports**
12. **Project Presentation**
13. **Final Project Report**

---

17. Suggested Assessment

Component	Marks
Problem & Requirements	10
ERD & Relationships	15
Mapping ERD → Relations	10
Keys & Constraints	10
Normalization	15
PostgreSQL Implementation	15
SQL Queries	10
PostgreSQL Advanced Features	5
Presentation & Viva	10
Total	**100**