STUDENT MANAGEMENT SYSTEM DATABASE DOCUMENTATION

1. Table Structure

Departments Table

Column Name	Datatype	Feature	Description
DepartmentID	VARCHAR(30)	Primary Key	Unique ID for each department
DepartmentName	VARCHAR(50)	Not Null	Name of the department

Students Table

Column Name	Datatype	Feature	Description
StudentID	VARCHAR(30)	Primary Key	Unique ID for each student
FirstName	NVARCHAR(30)	Not Null	Student's first name
LastName	NVARCHAR(30)	Not Null	Student's last name
Gender	NVARCHAR(10)	Not Null	Gender of the student
Date_of_Birth	Date		Student's birth date
DepartmentID	VARCHAR(30)	Foreign Key	References Department(DepartmentID)

Courses Table

Column Name	Datatype	Feature	Description
CourseID	VARCHAR(30)	Primary Key	Unique ID for each course
CourseName	VARCHAR(100)	Not Null	Name of the course
DepartmentID	VARCHAR(30)	Foreign Key	References Department(DepartmentID)

Enrollments Table

Column Name	Datatype	Feature	Description
EnrollmentID	VARCHAR(30)	Primary Key	Unique ID for each enrollment
StudentID	VARCHAR(30)	Foreign Key	References Students(StudentID)
CourseID	VARCHAR(30)	Foreign Key	Reference Courses(CourseID)
EnrollmentDate	Date		Date of Enrollment

Instructors Table

Column Name	Datatype	Feature	Description
InstructorID	VARCHAR(30)	Primary Key	Unique ID for each instructor
Title	VARCHAR(10)		Instructor's title
FirstName	NVARCHAR(30)	Not Null	Instructor's first name
LastName	NVARCHAR(30)	Not Null	Instructor's last name
Gender	NVARCHAR(10)	Not Null	Instructor's gender
DepartmentID	VARCHAR(30)	Foreign Key	References Departments(DepartmentID)

CourseInstructors Table

Column Name	Datatype	Feature	Description
CourseID	VARCHAR(30)	Foreign Key	References Courses(CourseID)
InstructorID	VARCHAR(30)	Foreign Key	References Instructors(InstructorID)
(CourseID, InstructorID)		Composite Primary Key	Act as unique ID

2. Table Relationship

Departments ↔ Students

One-to-Many Relationship

- A department can have many students
- Each student is enrolled in one department

Departments ↔ Courses

One-to-Many Relationship

- A department can offer many courses
- Each course belongs to one department

Departments ↔ Instructors

One-to-Many Relationship

- A department can have many instructors
- Each instructor belongs to one department

Students ↔ Enrollments

One-to-Many Relation

- Each student can enroll in many courses

Courses ↔ Enrollments

One-to-Many Relation

- Each course can have many students enrolled

Courses ↔ CourseInstructors

One-to-Many Relationship

- Each course can be taught by multiple instructors

Instructors ↔ CourseInstructors

One-to-Many Relationship

- CourseInstructors enables a many-to-many relationship between Courses and Instructors.
- One course may be taught by multiple instructors, and one instructor may teach multiple courses.

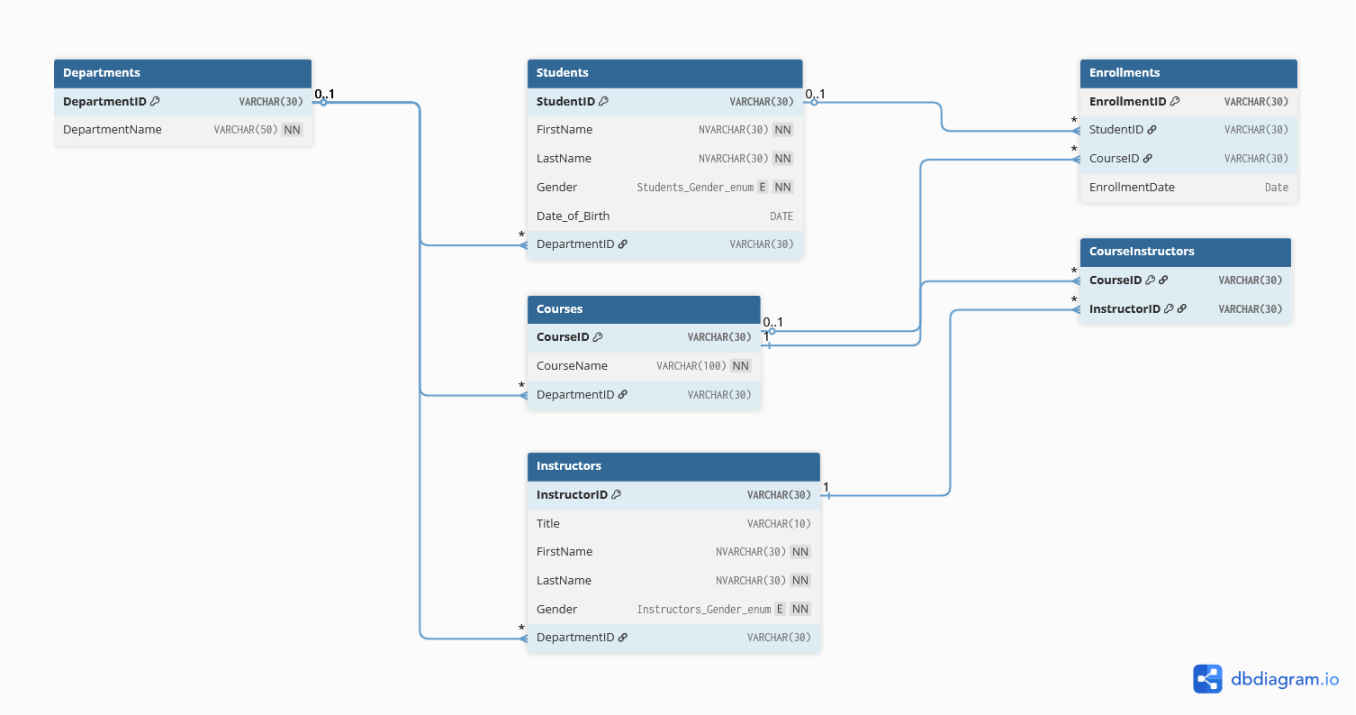

Figure 1: Entity Relationship Diagram

3. Query Logic

I. How many students are currently enrolled in each course?

```
SELECT c.CourseName, c.CourseID, COUNT(e.StudentID) AS No_of_Students_Enrolled
FROM Courses c
JOIN Enrollments e
ON c.CourseID = e.CourseID
GROUP BY c.CourseName, c.CourseID
ORDER BY No_of_Students_Enrolled DESC;
```

Logic:

- Joins Enrollments and Courses to link each enrollment to its corresponding course name.
- COUNT(e.StudentID) counts the number of students per course.
- GROUP BY c.CourseName, c.CourseID organizes the data by course.
- ORDER BY No_of_Students_Enrolled DESC lists courses starting with the most enrolled.

II. Which students are enrolled in multiple courses, and which courses are they taking?

```
SELECT s.FirstName + ' ' + s.LastName AS FullName, s.StudentID, c.CourseID,
c.CourseName
FROM Enrollments e
JOIN Students s
ON s.StudentID = e.StudentID
JOIN Courses c
ON c.CourseID = e.CourseID
WHERE e.StudentID IN (
        SELECT StudentID
        FROM Enrollments
        GROUP BY StudentID
        HAVING COUNT(EnrollmentID) > 1)
ORDER BY FullName, CourseID;
```

Logic:

- Joins Enrollments with Students and Courses to show which students are enrolled in which courses.
- Filters to only include students enrolled in more than one course using a subquery with HAVING COUNT > 1.

- Concatenates first and last names as FullName.
- Orders the results alphabetically by student name and then by course.

III. What is the total number of students per department across all courses?

```
SELECT d.DepartmentID, d.DepartmentName, COUNT(DISTINCT s.StudentID) AS
No_of_Students
FROM Departments d
JOIN Students s
ON d.DepartmentID = s.DepartmentID
GROUP BY d.DepartmentID, d.DepartmentName
ORDER BY No_of_Students DESC;
```

Logic:

- Joins Departments with Students to associate each student with their department.
- COUNT(DISTINCT s.StudentID) calculates the number of unique students in each department.
- GROUP BY d.DepartmentID, d.DepartmentName groups the data by department.
- ORDER BY No_of_Students DESC ranks departments by student count, from highest to lowest.

IV. Which courses have the highest number of enrollments?

```
SELECT TOP 3 c.CourseID, c.CourseName, COUNT(e.EnrollmentID) AS
No_of_Enrollments
FROM Enrollments e
JOIN Courses c
ON c.CourseID = e.CourseID
GROUP BY c.CourseID, c.CourseName
ORDER BY No_of_Enrollments DESC;
```

Logic:

- Joins Enrollments with Courses to match each enrollment to its course.
- COUNT(e.EnrollmentID) calculates the total number of enrollments per course.
- GROUP BY c.CourseID, c.CourseName groups the results by course.

- ORDER BY No_of_Enrollments DESC ranks courses by popularity.
- SELECT TOP 3 returns only the three most enrolled courses.

V. Which department has the least number of students?

```
SELECT TOP 3 d.DepartmentID, d.DepartmentName, COUNT(s.StudentID) AS
No_of_Students
FROM Departments d
JOIN Students s
ON s.DepartmentID = d.DepartmentID
GROUP BY d.DepartmentID, d.DepartmentName
ORDER BY No_of_Students ASC;
```

Logic:

- Joins Departments with Students to link each student to their department.
- COUNT(s.StudentID) counts how many students are in each department.
- GROUP BY d.DepartmentID, d.DepartmentName aggregates the count per department.
- ORDER BY No_of_Students ASC ranks departments from fewest to most students.
- SELECT TOP 3 returns the three departments with the lowest student count.

VI. Are there any students not enrolled in any course?

```
SELECT s.StudentID, s.FirstName + ' ' + s.LastName AS FullName
FROM Students s
LEFT JOIN Enrollments e
ON e.StudentID = s.StudentID
WHERE e.EnrollmentID IS NULL;
```

Logic:

- LEFT JOIN to include all students, even those without enrollments.
- Joins Students with Enrollments on StudentID.
- Filters with WHERE e.EnrollmentID IS NULL to return only students not enrolled in any course.

- Concatenates first and last name as FullName.

VII. How many courses does each student take on average?

```
SELECT AVG(Course_Count) AS Avg_Course_Count_Per_Student
FROM (
        SELECT StudentID, COUNT(CourseID) AS Course_Count
        FROM Enrollments
        GROUP BY StudentID
        ) sub
;
```

Logic:

- The subquery groups Enrollments by StudentID and counts how many courses each student is enrolled in.
- The outer query calculates the average number of courses per student using AVG(Course_Count).
- The result shows the average course load per student across all students.

VIII. What is the gender distribution of students across courses and instructors?

Across Courses

```
SELECT c.CourseID, c.CourseName, s.Gender, COUNT(s.StudentID) AS
No_of_Students
FROM Students s
JOIN Enrollments e
ON e.StudentID = s.StudentID
JOIN Courses c
ON c.CourseID = e.CourseID
GROUP BY c.CourseID, c.CourseName, s.Gender
ORDER BY c.CourseID;
```

Logic:

- Joins Students, Enrollments, and Courses to connect students with the courses they're enrolled in.
- Groups the data by CourseID, CourseName, and Gender to break down enrollment by gender per course.
- COUNT(s.StudentID) counts how many students of each gender are enrolled in each course.

- Orders the results by CourseID.

Across Instructors

```
SELECT ci.InstructorID, i.InstructorName, c.CourseID, c.CourseName, s.Gender,
COUNT(s.StudentID) AS No_of_Students
FROM (SELECT FirstName + ' ' + LastName AS InstructorName, InstructorID
    FROM Instructors) i
JOIN CourseInstructors ci
ON ci.InstructorID = i.InstructorID
JOIN Courses c
ON c.CourseID = ci.CourseID
JOIN Enrollments e
ON e.CourseID = c.CourseID
JOIN Students s
ON s.StudentID = e.StudentID
GROUP BY ci.InstructorID, i.InstructorName, c.CourseID, c.CourseName, s.Gender
ORDER BY ci.InstructorID, c.CourseID;
```

Logic:

- Creates a derived table (i) that combines instructors' first and last names as InstructorName.

- Joins CourseInstructors, Courses, Enrollments, and Students to connect instructors to their courses and enrolled students.

- Groups by instructor, course, and student gender to show how many students of each gender are taught by each instructor per course.

- COUNT(s.StudentID) gives the number of students per gender in each course taught by each instructor.

- Orders the results by InstructorID and CourseID for structured output.

IX. Which course has the highest number of male or female students enrolled?

```
SELECT c.CourseID, c.CourseName, s.Gender, COUNT(e.EnrollmentID) as
No_of_Students_Enrolled
FROM Courses c
JOIN Enrollments e
ON c.CourseID = e.CourseID
JOIN Students s
ON s.StudentID = e.StudentID
GROUP BY c.CourseID, c.CourseName, s.Gender
ORDER BY No_of_Students_Enrolled DESC;
```

Logic:

- Joins Courses, Enrollments, and Students to link each course with enrolled students and their gender.
- Groups by CourseID, CourseName, and Gender to break down enrollment by gender for each course.
- COUNT(e.EnrollmentID) counts how many students of each gender are enrolled per course.
- Orders results by the number of students enrolled, from highest to lowest.