Student Management System Database Documentation

---

1. Table Structures

👩‍🎓 `Students`

Stores student information.

Column Name	Data Type	Description
`StudentID`	INT	Unique ID for each student *(Primary Key)*
`Name`	VARCHAR()	Student's full name
`Gender`	CHAR(1)	Gender of the student (M/F)
`DOB`	DATE	Date of Birth
`DepartmentID`	INT	Reference to `Departments`

✅ **Foreign Key**: `DepartmentID → Departments.DepartmentID`

🏢 `Departments`

Stores academic department details.

Column Name	Data Type	Description
`DepartmentID`	INT	Unique department ID *(Primary Key)*
`DepartmentName`	VARCHAR(50)	Name of the department

📚 Courses

Contains course offerings.

Column Name	Data Type	Description
`CourseID`	INT	Unique course ID *(Primary Key)*
`CourseName`	VARCHAR(50)	Name of the course
`DepartmentID`	INT	Reference to `Departments`

✅ **Foreign Key**: `DepartmentID → Departments.DepartmentID`

📝 Enrollments

Tracks which students are enrolled in which courses.

Column Name	Data Type	Description
`EnrollmentID`	INT	Unique enrollment ID *(Primary Key)*
`StudentID`	INT	ID of student *(Foreign Key to Students)*
`CourseID`	INT	ID of course *(Foreign Key to Courses)*
`EnrollmentDate`	DATE	Date of enrollment

✅ **Foreign Keys**:

- `StudentID → Students.StudentID`
- `CourseID → Courses.CourseID`

👨‍🏫 Instructors

Details of faculty members.

Column Name	Data Type	Description
`InstructorID`	INT	Unique instructor ID *(Primary Key)*
`Name`	VARCHAR(100)	Instructor name
`Gender`	CHAR(1)	Gender of the instructor (M/F)
`DOB`	DATE	Date of Birth
`DepartmentID`	INT	Department reference

✅ **Foreign Key**: `DepartmentID → Departments.DepartmentID`

🔄 3. Table Relationships

Departments ---< Students
Departments ---< Courses
Departments ---< Instructors
Students ---< Enrollments >--- Courses

- One department can have many students, instructors, and courses.
- A student can enroll in many courses (via Enrollments).
- A course can have many students enrolled in it.

---

4. Common Query Logic

Q1: How many students are enrolled in each course?

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

Logic:

- Joins `Enrollments` with `Courses` to associate each enrollment with its course name.
- `COUNT(e.StudentID)` counts the number of enrollments per course.
- `GROUP BY c.CourseName` groups the data by course to show counts per course

Q2: Students enrolled in multiple courses

```
SELECT s.StudentID, s.Name, c.CourseName
FROM Enrollments e
JOIN Students s ON e.StudentID = s.StudentID
JOIN Courses c ON e.CourseID = c.CourseID
WHERE e.StudentID IN (
   SELECT StudentID
   FROM Enrollments
   GROUP BY StudentID
   HAVING COUNT(DISTINCT CourseID) > 1
)
ORDER BY s.StudentID;
```

Logic:

- Inner query finds `StudentID`s enrolled in **more than one distinct course**.
- Outer query retrieves those students' details and the **courses they are taking**.
- `JOIN` with `Courses` and `Students` brings full names and course names into the result.

Q3: Total students per department

```
SELECT d.DepartmentName, COUNT(DISTINCT s.StudentID) AS Total_Students
FROM Students s
JOIN Departments d ON s.DepartmentID = d.DepartmentID
```

```
GROUP BY d.DepartmentName;
```

Logic:

- Links `Students` to `Departments` to identify which department each student belongs to.
- `COUNT(DISTINCT s.StudentID)` ensures unique students are counted.
- `GROUP BY` department to calculate total per department.

Q4: Courses with the highest enrollments

```
SELECT c.CourseName, COUNT(e.StudentID) AS Total_Enrolled
FROM Enrollments e
JOIN Courses c ON e.CourseID = c.CourseID
GROUP BY c.CourseName
ORDER BY Total_Enrolled DESC
LIMIT 1;
```

Logic:

- Same logic as Q1, but now sorted in **descending order**.
- `LIMIT 1` ensures only the **top course** by enrollment is shown.

Q5: Departments with the least number of students

```
SELECT d.DepartmentName, COUNT(s.StudentID) AS Total_Students
FROM Departments d
LEFT JOIN Students s ON d.DepartmentID = s.DepartmentID
GROUP BY d.DepartmentName
ORDER BY Total_Students ASC
LIMIT 1;
```

Logic:

- `LEFT JOIN` includes departments **even if they have no students**.
- `GROUP BY` and `COUNT` gives a student count per department.
- `ORDER BY ASC` shows the one with the fewest students first.

Q6: Students not enrolled in any course

```
SELECT s.StudentID, s.Name
FROM Students s
LEFT JOIN Enrollments e ON s.StudentID = e.StudentID
WHERE e.EnrollmentID IS NULL;
```

Logic:

- `LEFT JOIN` gets all students, even if they have **no match** in `Enrollments`.
- `WHERE e.EnrollmentID IS NULL` filters only students **not enrolled** in any course.

Q7: Average number of courses per student

```
SELECT AVG(course_count) AS Avg_Courses_Per_Student
FROM (
   SELECT COUNT(DISTINCT CourseID) AS course_count
   FROM Enrollments
   GROUP BY StudentID
) AS sub;
```

Logic:

- The inner query gets the number **of courses each student** is enrolled in.
- Outer query takes the `AVG()` of these counts to get the **average**.

Q8: Gender distribution across courses

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

Logic:

- Joins `Enrollments`, `Students`, and `Courses` to get course-wise and gender-wise details.
- `GROUP BY` both `CourseName` and `Gender` to break the counts into **segments**.

Q9: Course with highest number of male or female students

```
SELECT c.CourseName, s.Gender, COUNT(*) AS Total
FROM Enrollments e
JOIN Students s ON e.StudentID = s.StudentID
JOIN Courses c ON e.CourseID = c.CourseID
GROUP BY c.CourseName, s.Gender
ORDER BY Total DESC
LIMIT 1;
```

Logic:

- Same structure as Q8 but sorted to find the **single largest gender count** per course.
- `ORDER BY Total DESC LIMIT 1` gives the top result.