Student Management System Database Documentation

# 1. Project Overview

This project models a university student-course enrollment system. The database tracks students, departments, instructors, courses, and enrollments. It supports queries for academic reporting and analytics.

# 2. Tools Used

## SQL Environment

* Microsoft SQL

## ERD Design Tool

* Dbdiagram.io

# 3. Table Structures

## Table: students

* StudentID: Primary key, unique identifier for each student
* Name: Full name of the student
* Gender: Gender of the student (Male, Female, and Other)
* DOB: Date of Birth
* DepartmentID: Foreign key referencing departments

## Table: departments

* DepartmentID: Primary key, unique identifier for each department
* DepartmentName: Name of the department

## Table: courses

* CourseID: Primary key, unique identifier for each course
* CourseName: Name of the course
* DepartmentID: Foreign key referencing departments

## Table: enrollments

* EnrolmentID: Primary key, unique identifier for each enrollment
* CourseID: Foreign key referencing courses
* StudentID: Foreign key referencing students
* EnrollmentDate: Date of enrollment

## Table: instructors

* InstructorID: Primary key, unique identifier for each instructor
* InstructorName: Full name of the instructor
* DepartmentID: Foreign key referencing departments

## Table: course\_instructors *(junction table)*

* InstructorID: Foreign key referencing students
* CourseID: Foreign key referencing courses

|  |  |  |
| --- | --- | --- |
| **Tables** | **Columns** | **Description** |
| students | [StudentID] **[PK],**  [Name],  [Gender],  [DOB],  [DepartmentID] [FK] | * *A student can take multiple courses and many students can take a course together. Therefore, a* ***Many-to-Many*** *relationship with [courses] through Enrollment, that will serve as a ‘junction’ table to [students] and [courses]* * *Has a* ***Many-to-1*** *relationship with department (a student can only be in 1 dept.), hence the* ***[FK]*** |
| departments | [DepartmentID][PK], [DepartmentName] | * *Has a* ***1-to-Many*** *relationship with* ***students****,* ***courses****, and* ***instructors*** *Tables.* |
| courses | [CourseID] [PK],  [CourseName],  [DepartmentID] [FK] | * ***Many-to-Many*** *with students (Explained in student section.* * ***Many -to-1*** *relationship with department.* * ***Many-to-Many*** *relationship with Instructors table. (Multiple instructors can teach 1 course, and multiple courses can be taught by 1 instructor). To make this relationship possible, a new ‘junction’ table* ***[course\_instructors****] was created* |
| enrollments | [EnrolmentID] [PK],  [CourseID] [FK],  [StudentID] [FK],  [EnrollmentDate] | *Junction table for students and courses* ***Many-to-Many*** *relationship* |
| instructors | [InstructorID] [PK],  [InstructorName],  [DepartmentID][FK] | * ***Many-to-Many*** *with courses (Explained in student section.* * ***Many -to-1*** *relationship with department.* * ***Many-to-Many*** *relationship with courses (explained in courses section* |
| course\_instructors | [InstructorID],  [CourseID] | * *Junction table created just for courses and instructors* ***Many-to-Many*** *relationship* |

# 4. Table Relationships

* students.DepartmentID 🡪departments.DepartmentID (**many-to-one**)
* courses.DepartmentID 🡪 departments.DepartmentID (**many-to-one**)
* instructors.DepartmentID 🡪 departments.DepartmentID (**many-to-one**)
* enrollments.StudentID 🡪students.StudentID (**many-to-one**)
* enrollments.CourseID 🡪courses.CourseID (**many-to-one**)
* course\_instructors.CourseID 🡪courses.CourseID (**many-to-one**)
* course\_instructors.InstructorID 🡪 instructors.InstructorID (**many-to-one**)

# 5. Query Logic

## Section 1: Students & Enrollment Reports

### How many students are currently enrolled in each course?

SELECT   
 c\_o.CourseID,   
 c\_o.CourseName,  
 COUNT(e\_r.StudentID) AS no\_of\_students  
FROM courses AS c\_o  
LEFT JOIN enrollments AS e\_r  
 ON c\_o.CourseID = e\_r.CourseID  
GROUP BY  
 c\_o.CourseID,   
 c\_o.CourseName  
ORDER BY COUNT(e\_r.StudentID) DESC;

#### Logic:

* COUNT(e\_r.StudentID)
* *Counting studentsID from the enrollments table means only actual enrollments are counted.*
* **LEFT JOIN:** *Ensures* ***all courses*** *appear, including courses that NO student enrolled in, (in case there happens to be course(s) like that.)*
* Grouping by CourseID, and CourseName *ensures a clear result is obtained.*

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

WITH studentMultiCourses AS (  
-- Which students enrolled in more than 1 course  
 SELECT StudentID, COUNT(CourseID) AS no\_of\_courses  
 FROM enrollments  
 GROUP BY StudentID  
 HAVING COUNT(DISTINCT CourseID) > 1  
)  
 -- JOIN students, and course table to get students id & name, and course names respectively. [ students --> enrollments <--- courses]  
SELECT   
 s\_t.StudentID,  
 s\_t.Name,  
 c\_o.CourseName  
FROM students AS s\_t  
JOIN enrollments AS e\_r   
 ON s\_t.StudentID = e\_r.StudentID  
JOIN courses c\_o  
 ON e\_r.CourseID = c\_o.CourseID  
-- this will filter the enties to only those that enroled for 2 or more course  
WHERE s\_t.StudentID IN ( SELECT StudentID FROM studentMultiCourses);

#### Logic:

* CTE finds students that enroll for multiple courses (>1)
* Main query joins 3 tables to get student name/course name details
* DISTINCT handles duplicated course issues

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

SELECT   
 d\_p.DepartmentID,  
 d\_p.DepartmentName,  
 c\_o.CourseID,  
 c\_o.CourseName,  
 COUNT(DISTINCT s\_t.StudentID) AS no\_of\_student --handles duplicate errors  
FROM departments AS d\_p  
JOIN courses AS c\_o   
 ON d\_p.DepartmentID = c\_o.DepartmentID  
LEFT JOIN enrollments AS e\_n -- preserves department & courses with 0 enrollment  
 ON c\_o.CourseID = e\_n.CourseID  
LEFT JOIN students AS s\_t -- handles errors, or NULL entry  
 ON e\_n.StudentID = s\_t.StudentID   
GROUP BY  
 d\_p.DepartmentID,  
 d\_p.DepartmentName,  
 c\_o.CourseID,  
 c\_o.CourseName  
ORDER BY  
 no\_of\_student DESC ;

#### Logic:

* COUNT(DISTINCT s\_t.StudentID), Handles duplicate errors, and ensures accurate student count.
* Inner join for department 🡪 courses, because all courses must belong to a department.
* LEFT JOIN:
* course 🡪 enrollment preserves department and courses with 0 enrollment
* enrollment 🡪 handles NULL errors, and students with 0 enrollment.

## Section 2: Course & Instructor Analysis

### Which courses have the highest number of enrollments?

WITH topEnrolledCourses AS (  
 SELECT   
 -- rank course from the highest count of enrollment  
 c\_o.CourseID,  
 c\_o.CourseName,  
 COUNT(e\_r.StudentID) as enrollment\_count,  
 DENSE\_RANK() OVER(ORDER BY COUNT(e\_r.StudentID) DESC) AS ranking  
 FROM courses AS c\_o  
 LEFT JOIN enrollments AS e\_r  
 ON c\_o.CourseID = e\_r.CourseID  
 GROUP BY  
 c\_o.CourseID,  
 c\_o.CourseName  
)  
 SELECT CourseID, CourseName, enrollment\_count  
 FROM topEnrolledCourses  
 WHERE ranking = 1;

#### Logic:

* CTE calculates enrollment counts, and also ranks courses from highest to lowest using dense\_rank().
* DENSE\_RANK ensures that courses with the same number of enrolled students have the same rank number. This is neccessary to handle situations of a tie [e.g, 1,1,1,2,3]
* LEFT JOIN
* To include courses with 0 enrollments. For this query, Inner join will also work perfectly here (eliminates 0 enrollments).

### Which department has the least number of students?

WITH leastRankedDept AS (  
SELECT   
 d\_p.DepartmentID,  
 d\_p.DepartmentName,  
 COUNT(s\_t.StudentID) AS no\_of\_students,  
 DENSE\_RANK() OVER(ORDER BY COUNT(s\_t.StudentID) ASC) as ranking  
FROM  
 departments AS d\_p  
LEFT JOIN students AS s\_t -- presevers dept. with 0 students  
 ON d\_p.DepartmentID = s\_t.DepartmentID  
GROUP BY  
 d\_p.DepartmentID,  
 d\_p.DepartmentName  
)  
SELECT   
 DepartmentID,  
 DepartmentName  
FROM  
 leastRankedDept  
WHERE ranking = 1;

#### Logic:

* LEFT JOIN to include departments with 0 students.
* DENSE\_RANK, used to rank all departments with starting from the minimum count to the maximum. Dense\_rank is useful here to handle issues of a tie.

## Section 3: Data Integrity & Operational Insights

### Are there any students not enrolled in any course?

SELECT  
 s\_t.StudentID,  
 s\_t.Name AS [Student Name]  
  
FROM students AS s\_t   
LEFT JOIN enrollments AS e\_r  
 ON s\_t.StudentID = e\_r.StudentID  
WHERE e\_r.enrollmentID IS NULL;

#### Logic:

* LEFT JOIN + IS NULL to find students with no enrollments.
* Returns actual student records (id and Names).

### How many courses does each student take on average?

-- step 1: count number of courses enrolled by each student  
-- step 2: take the overall average (mean)  
  
WITH courseCount AS (  
 -- stp-1  
 SELECT  
 s\_t.StudentID,  
 COUNT(DISTINCT e\_n.CourseID) AS no\_of\_courses  
FROM students AS s\_t  
LEFT JOIN enrollments AS e\_n  
 ON s\_t.StudentID = e\_n.StudentID  
GROUP BY  
 s\_t.StudentID  
)  
  
 -- stp-2  
 SELECT   
 ROUND(AVG(no\_of\_courses), 2) AS no\_of\_courses\_per\_student -- 2 d.p  
 FROM courseCount;

#### Logic:

* LEFT JOIN ensures that ALL students are included.
* DISTINCT handles issues with duplicate enrollment.
* Average is formated to 2 decimal places.

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

WITH courseInstructorInfo AS (  
 SELECT   
 c\_o.CourseID, c\_o.CourseName,  
 i\_n.InstructorID, i\_n.InstructorName,  
 s\_t.Gender,  
 COUNT(\*) As student\_count  
  
 FROM students AS s\_t -- table 1  
 JOIN enrollments AS e\_n -- table 2  
 ON s\_t.StudentID = e\_n.StudentID   
   
 JOIN courses AS c\_o -- table 3  
 ON e\_n.CourseID = c\_o.CourseID  
  
 JOIN course\_instructors AS c\_i -- table 4  
 ON c\_o.CourseID = c\_i.CourseID  
  
 JOIN instructors AS i\_n  
 on c\_i.InstructorID = i\_n.InstructorID --- table 5  
  
)  
-- structure  
  
SELECT   
 CourseName,  
 InstructorName,  
 SUM(CASE when Gender = 'Male' THEN student\_count ELSE 0 END) AS Male\_students,  
 SUM(CASE WHEN Gender = 'Female' THEN student\_count ELSE 0 END) AS Female\_students,  
 SUM(CASE WHEN Gender = 'Other' THEN student\_count ELSE 0 END) AS Others,  
 SUM(student\_count) AS Total\_no\_of\_Students  
FROM courseInstructorInfo  
;

#### Logic:

* CTE collects base counts per course/instructor/gender (\*)
* Main query properly structures gender counts using CASE statements.
* JOIN connects all 5 tables correctly (3 main tables + 2 juncture tables).

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

WITH genderCount AS (  
 SELECT  
 c\_o.CourseID,  
 c\_o.CourseName,  
 s\_t.Gender,  
 COUNT(\*) AS enrollment\_count,  
 DENSE\_RANK() OVER(PARTITION BY s\_t.Gender ORDER BY COUNT(\*) DESC) AS rank\_by\_gender  
 FROM students s\_t  
 JOIN enrollments AS e\_r  
 ON s\_t.StudentID = e\_r.StudentID  
 JOIN courses AS c\_o  
 ON e\_r.CourseID = c\_o.CourseID  
 WHERE GENDER IN ('Male', 'Female') -- this excludes 'other'  
 GROUP BY  
 c\_o.CourseID,  
 c\_o.CourseName,  
 s\_t.Gender  
)  
  
SELECT  
 CourseName,  
 Gender,  
 enrollment\_count  
FROM genderCount  
WHERE rank\_by\_gender = 1;

#### Logic:

* DENSE\_RANK() finds top courses within each gender group, AND also Handles ties, in case of multiple courses being tie at the top.
* WHERE filters only Male/Female as specified.