**Student Management System**

The Student Management System database will assist in managing student details, department information, courses, enrollments, and academic performance records. Here's the modified description:

The Student Management System (SMS) database will consist of the following tables:

1. **Students**:
   * Description: This table stores information about students, including their names, birthdates, and genders. It serves as a repository for student data within the educational institution.
2. **Courses**:
   * Description: This table contains details about the various courses offered by the institution. It includes information such as course names and descriptions.
3. **Instructors**:
   * Description: This table holds data about instructors who teach at the institution. It includes information like their names and contact details, such as email addresses.
4. **Departments**:
   * Description: This table represents the academic departments within the institution. It stores data about department names and identifies the department head using the **DepartmentHeadID** field, which relates to an instructor.
5. **Enrollment**:
   * Description: This table establishes a relationship between students and the courses they are enrolled in. It includes information about the enrollment process, such as the **EnrollmentID**, **StudentID**, and **CourseID**.
6. **InstructorCourseAssignments**:
   * Description: This table defines the relationship between instructors and the courses they are assigned to teach. It records the assignments of instructors to specific courses.
7. **Introduction to MySQL Workbench**

What is MySQL?

MySQL is an open-source Relational Database Management System (RDBMS) developed by Oracle Corporation, Sun Microsystems, that uses [Structured Query language(SQL)](https://www.simplilearn.com/tutorials/sql-tutorial/what-is-sql) to interact with databases. You can use MySQL to store, retrieve, manipulate and process data that is in the form of tables.

Why Use MySQL?

There are various relational database management systems present in the tech world today, such as Microsoft SQL Server, Microsoft Access, Oracle, DB2, etc.

Here are some reasons why people use MySQL over other Database Management Systems.

* Multiple Storage Engines

MySQL adheres to multiple storage engines, and each one of the storage engines possesses unique features, while other databases like SQL Server only support a single storage engine.

* InnoDB: It is the default storage engine fabricated with MySQL since version 5.5. It supports ACID-based transactions.
* MyISAM: Former to version 5.5, MyISAM was the default storage engine used by MySQL. It does not support ACID-based transactions.
* High Performance

MySQL has reported high performance compared to other database management systems because of its simplicity in design, and adherence to multiple storage engines.

* Cost-Effective

The community edition of MySQL is free of cost, and the commercial edition has a licensing fee, which is cost-effective compared to other products available in the market.

* Cross-Platform

MySQL runs on Windows, Linux, and macOS because of its cross-platform property.

With this, it is clear why MySQL is used. Now, you will see what MySQL Workbench exactly is and how one can use it.

What is MySQL Workbench?

MySQL Workbench is a unified cross-platform, open-source relational database design tool that adds functionality and ease to MySQL and SQL development work. MySQL Workbench provides data modeling, SQL development, and various administration tools for configuration. It also offers a graphical interface to work with the databases in a structured way.

* You can create a Graphical Model using MySQL Workbench
* MySQL Workbench provides reverse engineering for live databases to models
* MySQL Workbench offers a forward engineering model to a script/live database

1. **Installation of MySQL Workbench**

The installation process is similar to other operating systems.

1. Open the MySQL website on a browser. Click on the following link: [MySQL Downloads](https://www.mysql.com/downloads/).
2. Select the Downloads option.
3. Select MySQL Installer for Windows.
4. Choose the desired installer and click on download.
5. After it downloads the installer, open it.
6. It will ask for permission; when it does, click Yes. The installer will then open. Now, it will ask to choose the setup type, here, select Custom.
7. Click on Next. With this, you will be able to install MySQL server, MySQL Workbench, and MySQL shell.
8. Open MySQL Servers, select the server you want to install, and move it to the  Products/Features to be installed window section. Now, expand Applications, choose MySQL Workbench and MySQL shell. Move both of them to ‘Products/Features to be installed’.
9. Click on the Next button. Now, click on the Execute button to download and install the MySQL server, MySQL Workbench, and the MySQL shell.
10. Once the product is ready to configure, click on Next. Under Type and Networking, go with the default settings and select Next.
11. For authentication, use the recommended strong password encryption.
12. Set your MySQL Root password and click on next.
13. Go for the default windows service settings and under apply configuration, click on execute. Once the configuration is complete, click on finish.
14. Complete the installation. This will now launch the MySQL Workbench and the MySQL Shell.

Once MySQL Workbench is installed, select the Local instance and enter the password.

**Identify Entities**

* Start by identifying the main entities in your system. These are the objects or concepts about which you want to store data.
* Each entity should correspond to a table in your database.

**Define Attributes**

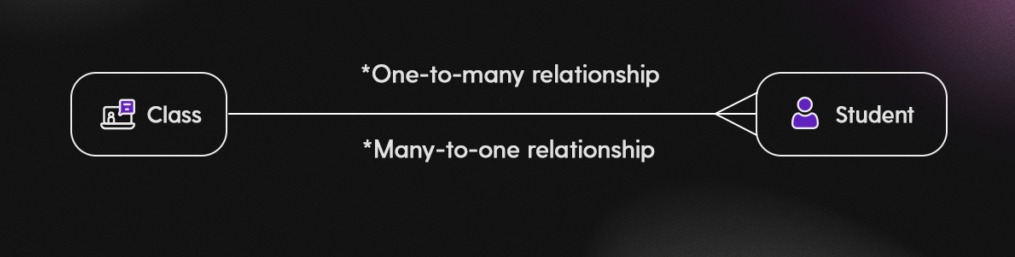
* For each entity, list the attributes (properties or fields) that describe it.
* These attributes will become columns in the corresponding database table.

a few examples of relationships:

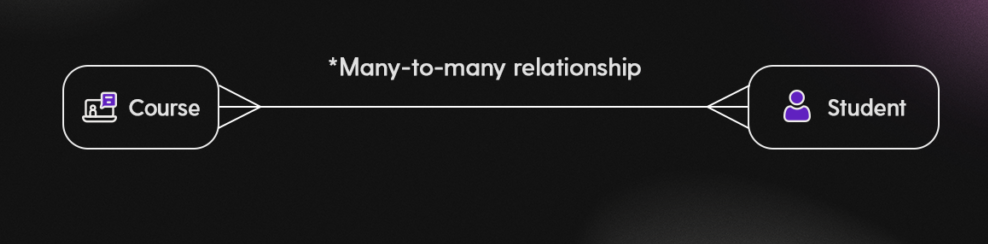
1. **One-to-One:** When each entity in each entity set can take part only once in the relationship, the cardinality is one-to-one. Let us assume that a male can marry one female and a female can marry one male. So the relationship will be one-to-one.



1. **One-to-Many:** In one-to-many mapping as well where each entity can be related to more than one relationship and the total number of tables that can be used in this is 2. Let us assume that one surgeon deparment can accomodate many doctors. So the Cardinality will be 1 to M. It means one deparment has many Doctors.
2. **Many-to-One:** When entities in one entity set can take part only once in the relationship set and entities in other entity sets can take part more than once in the relationship set, cardinality is many to one. Let us assume that a student can take only one course but one course can be taken by many students.



**4. Many-to-Many:**When entities in all entity sets can take part more than once in the relationship cardinality is many to many.



**Cardinality Notation**

Cardinality is a mathematical term. It translates into the number of elements in a set. In databases, cardinality refers to the relationships between the data in two database tables. Cardinality defines how many instances of one entity are related to instances of another entity.

* Let’s see our Student Management System
* Entities of Student Management System

1. Students
2. Courses
3. Instructors
4. Departments
5. Enrollment
6. InstructorCourseAssignments

* The attributes and relationships of each entity for the Student Management System

**Students Table**:

* Description: This table stores information about students.
* Attributes:
  + StudentID (Primary Key): Unique identifier for each student.
  + FirstName: The first name of the student.
  + LastName: The last name of the student.
  + BirthDate: The date of birth of the student.
  + Gender: The gender of the student (e.g., Male, Female).
* Relationships:
  + Many-to-Many relationship with Courses via the Enrollment table (StudentID).

**Courses Table**:

* Description: This table contains details about the various courses offered by the institution.
* Attributes:
  + CourseID (Primary Key): Unique identifier for each course.
  + CourseName: The name of the course.
  + CourseDescription: A textual description of the course.
* Relationships:
  + Many-to-Many relationship with Students via the Enrollment table (CourseID).
  + One-to-Many relationship with Instructors via the InstructorCourseAssignments table (CourseID).

**Instructors Table**:

* Description: This table holds data about instructors who teach at the institution.
* Attributes:
  + InstructorID (Primary Key): Unique identifier for each instructor.
  + FirstName: The first name of the instructor.
  + LastName: The last name of the instructor.
  + Email: Email address of the instructor.
* Relationships:
  + One-to-Many relationship with Departments via the Departments table (DepartmentHeadID).
  + One-to-Many relationship with Courses via the InstructorCourseAssignments table (InstructorID).

**Departments Table**:

* Description: This table represents the academic departments within the institution.
* Attributes:
  + DepartmentID (Primary Key): Unique identifier for each department.
  + DepartmentName: The name of the department.
  + DepartmentHeadID: Identifies the department head.
* Relationships:
  + One-to-Many relationship with Instructors via the Instructors table (DepartmentHeadID).

**Enrollment Table**:

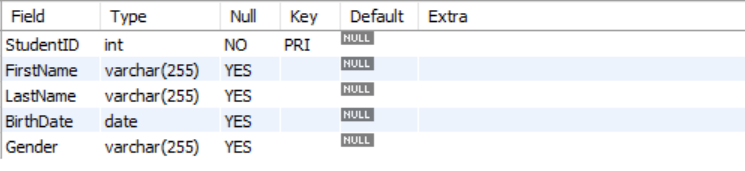
* Description: This table establishes a relationship between students and the courses they are enrolled in.
* Attributes:
  + EnrollmentID (Primary Key): Unique identifier for each enrollment.
  + StudentID: References the StudentID from the Students table.
  + CourseID: References the CourseID from the Courses table.
* Relationships:
  + Many-to-One relationship with Students (StudentID).
  + Many-to-One relationship with Courses (CourseID).

**InstructorCourseAssignments Table**:

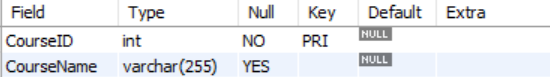
* Description: This table defines the relationship between instructors and the courses they are assigned to teach.
* Attributes:
  + AssignmentID (Primary Key): Unique identifier for each assignment.
  + InstructorID: References the InstructorID from the Instructors table.
  + CourseID: References the CourseID from the Courses table.
* Relationships:
  + Many-to-One relationship with Instructors (InstructorID).
  + Many-to-One relationship with Courses (CourseID).

Table Structure

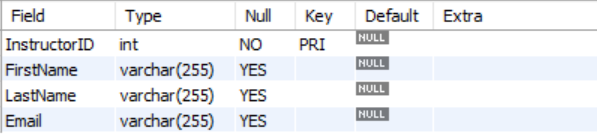
1. Student:



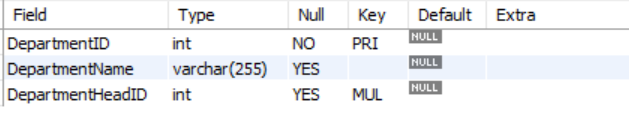
1. Courses:



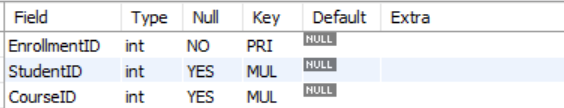
3. Instructors



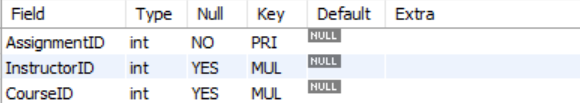
4. Departments



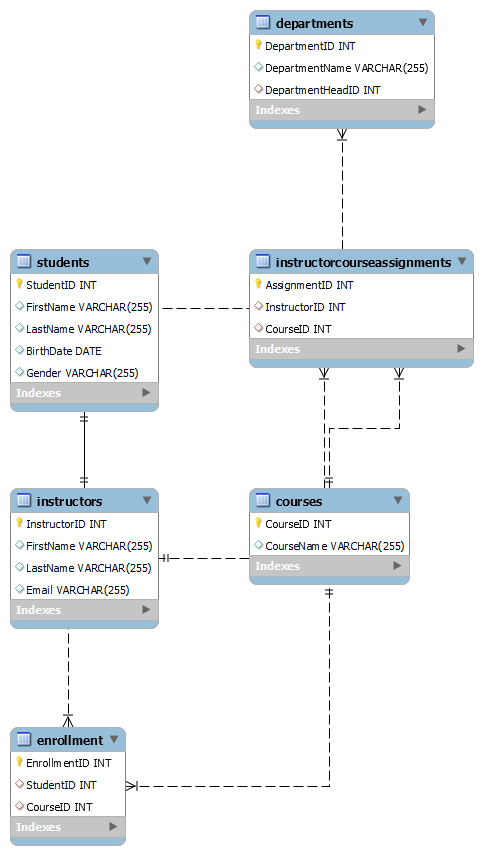
5. Enrollment



6. InstructorCourseAssignments



* Entity-Relationship Diagram:



**In this Entity-Relationship Diagram:**

1. **Students to Enrollments**:
   * Relationship Type: One-to-Many
   * Description: One student can be enrolled in many courses, but each enrollment record is associated with only one student.
   * Representation: In an ERD, you can depict this with a straight line connecting "Students" (one side) to "Enrollments" (many side).
2. **Courses to Enrollments**:
   * Relationship Type: One-to-Many
   * Description: One course can have many enrolled students, but each enrollment record corresponds to only one course.
   * Representation: In an ERD, you can show this with a straight line connecting "Courses" (one side) to "Enrollments" (many side).
3. **Instructors to InstructorCourseAssignments**:
   * Relationship Type: One-to-Many
   * Description: Each instructor can be assigned to teach multiple courses, but each course assignment is associated with only one instructor.
   * Representation: In an ERD, you can represent this with a straight line connecting "Instructors" (one side) to "InstructorCourseAssignments" (many side).
4. **Departments to Instructors**:
   * Relationship Type: One-to-Many
   * Description: Each department can have multiple instructors, but each instructor is associated with only one department.
   * Representation: In an ERD, you can depict this with a straight line connecting "Departments" (one side) to "Instructors" (many side).
5. **Instructors to Departments**:
   * Relationship Type: One-to-One
   * Description: Each department has only one department head (an instructor), and each instructor heads only one department.
   * Representation: In an ERD, you can show this with a straight line connecting "Instructors" (one side) to "Departments" (one side), often labeled as "Heads."
6. **Creating a Database for Student Management System**

Query:

create database studentsdb;

Using a database

Query:

use studentsdb;

1. **Creating the Tables for each entity**

Query:

-- Create a table for students

CREATE TABLE Students (

StudentID INT PRIMARY KEY,

FirstName VARCHAR(255),

LastName VARCHAR(255),

BirthDate DATE,

Gender VARCHAR(255)

);

-- Create a table for courses

CREATE TABLE Courses (

CourseID INT PRIMARY KEY,

CourseName VARCHAR(255),

CourseDescription TEXT

);

-- Create a table for instructors

CREATE TABLE Instructors (

InstructorID INT PRIMARY KEY,

FirstName VARCHAR(255),

LastName VARCHAR(255),

Email VARCHAR(100)

);

-- Create a table for departments

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(255),

DepartmentHeadID INT,

FOREIGN KEY (DepartmentHeadID) REFERENCES Instructors(InstructorID)

);

-- Create a table for student enrollment

CREATE TABLE Enrollment (

EnrollmentID INT PRIMARY KEY,

StudentID INT,

CourseID INT,

FOREIGN KEY (StudentID) REFERENCES Students(StudentID),

FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)

);

-- Create a table for instructor-course assignments

CREATE TABLE InstructorCourseAssignments (

AssignmentID INT PRIMARY KEY,

InstructorID INT,

CourseID INT,

FOREIGN KEY (InstructorID) REFERENCES Instructors(InstructorID),

FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)

);

1. **Insert Records**

INSERT INTO Students (StudentID, FirstName, LastName, BirthDate, Gender)

VALUES

(1, 'Shruti', 'roy', '1990-05-15', 'Feale'),

(2, 'Jerold', 'nadar', '1992-08-21', 'male'),

(3, 'Tiya', 'Johnson', '1995-03-10', 'Female');

INSERT INTO Courses (CourseID, CourseName)

VALUES

(101, 'Math 101'),

(102, 'History 101'),

(103, 'Programming 101');

INSERT INTO Instructors (InstructorID, FirstName, LastName, Email)

VALUES

(1, 'Professor', 'Saniya', 'prof.saniya@example.com'),

(2, 'Professor', 'Jay', 'prof.jay@example.com');

INSERT INTO Departments (DepartmentID, DepartmentName, DepartmentHeadID)

VALUES

(1, 'Mathematics', 1),

(2, 'History', 2);

INSERT INTO Enrollment (EnrollmentID, StudentID, CourseID)

VALUES

(1, 1, 101),

(2, 2, 102),

(3, 3, 103);

INSERT INTO InstructorCourseAssignments (AssignmentID, InstructorID, CourseID)

VALUES

(1, 1, 101),

(2, 2, 102);

1. **Select Records**
2. **Select all students**:

SELECT \* FROM Students;

1. **Select students enrolled in a specific course (e.g., CourseID = 101)**:

SELECT Students.StudentID, Students.FirstName, Students.LastName

FROM Students

JOIN Enrollment ON Students.StudentID = Enrollment.StudentID

WHERE Enrollment.CourseID = 101;

1. **Select courses taught by a specific instructor (e.g., InstructorID = 1)**:

SELECT Courses.CourseID, Courses.CourseName

FROM Courses

JOIN InstructorCourseAssignments ON Courses.CourseID = InstructorCourseAssignments.CourseID

WHERE InstructorCourseAssignments.InstructorID = 1;

1. **Update Records**
2. **Update a Student's Information**:

UPDATE Students

SET LastName = 'NewLastName'

WHERE StudentID = 1;

1. **Change the Course Description**:

UPDATE Courses

SET CourseDescription = 'Updated description'

WHERE CourseID = 101;

1. **Assign an Instructor to a Different Course**:

UPDATE InstructorCourseAssignments

SET CourseID = 102

WHERE InstructorID = 2;

1. **Delete Records**
2. **Delete a Student**:

DELETE FROM Students

WHERE StudentID = 1;

1. **Remove an Enrollment Record**:

DELETE FROM Enrollment

WHERE EnrollmentID = 2;

1. **Delete an Instructor's Assignment**:

DELETE FROM InstructorCourseAssignments

WHERE InstructorID = 2 AND CourseID = 102;

1. **Other Queries**
2. **Subquery for Finding Courses with Maximum Enrollments**:

SELECT CourseName, CourseDescription

FROM Courses

WHERE CourseID = (

SELECT CourseID

FROM Enrollment

GROUP BY CourseID

ORDER BY COUNT(StudentID) DESC

LIMIT 1

);

1. **Calculating Average Enrollment per Course**:

SELECT AVG(StudentCount) AS AverageEnrollment

FROM (SELECT CourseID, COUNT(StudentID) AS StudentCount FROM Enrollment GROUP BY CourseID) AS CourseEnrollments;

1. **List Courses Not Assigned to Instructors**:

SELECT CourseID, CourseName

FROM Courses

WHERE CourseID NOT IN (

SELECT CourseID

FROM InstructorCourseAssignments

);

1. **Calculate Total Enrollments in Each Course**:

SELECT Courses.CourseName, COUNT(Enrollment.StudentID) AS TotalEnrollments

FROM Courses

LEFT JOIN Enrollment ON Courses.CourseID = Enrollment.CourseID

GROUP BY Courses.CourseName;

1. **Calculate Average Student Age**:

SELECT AVG(YEAR(NOW()) - YEAR(BirthDate)) AS AverageAge

FROM Students;

1. **Find the Maximum Number of Enrollments**:

SELECT Courses.CourseName, MAX(EnrollmentCount) AS MaxEnrollments

FROM (SELECT CourseID, COUNT(StudentID) AS EnrollmentCount FROM Enrollment GROUP BY CourseID) AS CourseEnrollments;

1. **Count Male and Female Students**:

SELECT Gender, COUNT(\*) AS StudentCount

FROM Students

GROUP BY Gender;

1. **students whose last name contains the substring**

SELECT FirstName, LastName

FROM Students

WHERE LastName LIKE '%Shruti%';