**Student Management System**

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

1. Student:
   * 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. Course:
   * Description: This table contains details about the various courses offered by the institution, including course names and descriptions.
3. Department:
   * Description: This table represents academic departments within the institution, storing data about department names and associated courses.
4. Lecturer:
   * Description: This table holds data about lecturers or professors, including their names and contact information.
5. Grade:
   * Description: This table records the grades assigned to students for specific courses by particular lecturers. It tracks the academic performance of students.
6. **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**

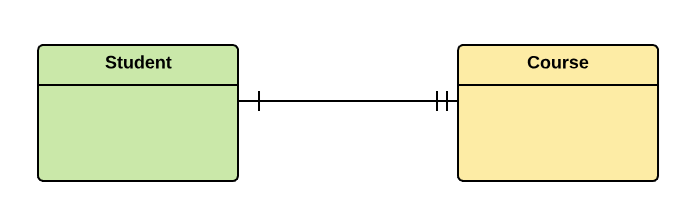
* For each entity, list the attributes (properties or fields) that describe it.
* These attributes will become columns in the corresponding database table.

a few examples of relationships:

**1.One-to-one:**

One entity from entity set X can be associated with at most one entity of entity set Y and vice versa.

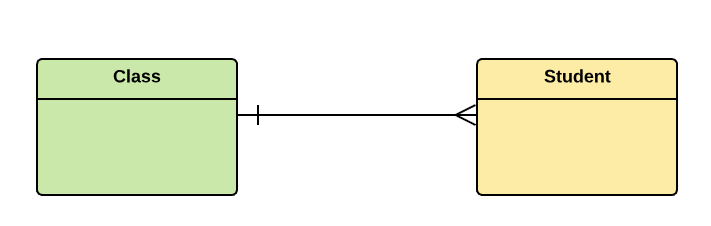
Example: One student can register for numerous courses. However, all those courses have a single line back to that one student.

[](https://www.guru99.com/images/1/100518_0621_ERDiagramTu8.png)

**2.One-to-many:**

One entity from entity set X can be associated with multiple entities of entity set Y, but an entity from entity set Y can be associated with at least one entity.

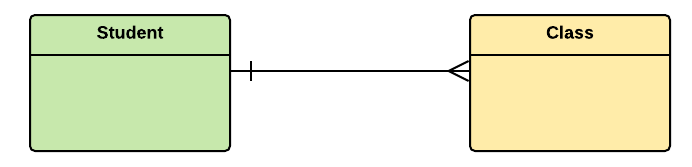
For example, one class is consisting of multiple students.

[](https://www.guru99.com/images/1/100518_0621_ERDiagramTu9.png)

**3. Many to One**

More than one entity from entity set X can be associated with at most one entity of entity set Y. However, an entity from entity set Y may or may not be associated with more than one entity from entity set X.

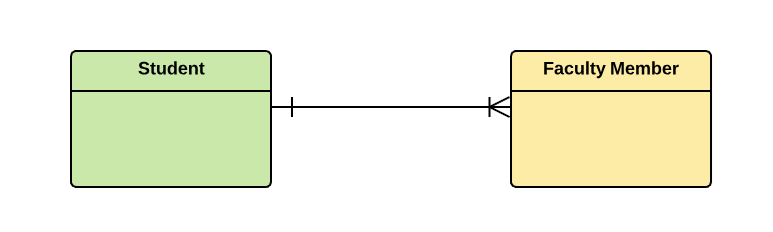
For example, many students belong to the same class.

[](https://www.guru99.com/images/1/100518_0621_ERDiagramTu10.png)

**4. Many to Many:**

One entity from X can be associated with more than one entity from Y and vice versa.

For example, Students as a group are associated with multiple faculty members, and faculty members can be associated with multiple students.

[](https://www.guru99.com/images/1/100518_0621_ERDiagramTu11.png)

* Let’s see our Student Management System

1. Entities of Student Management System
2. Student Table
3. Course Table
4. Department Table
5. Lecturer Table
6. Grade Table
7. The attributes and relationships of each entity for the Student Management System

**Student Entity:**

* Attributes:
  + StudentID (Primary Key): Unique identifier for each student.
  + FirstName: First name of the student.
  + LastName: Last name of the student.
  + Birthdate: Date of birth of the student.
  + Gender: Gender of the student (e.g., Male, Female).
* Relationships:
  + Enrollments: Many-to-Many relationship with the Course entity. A student can be enrolled in multiple courses, and a course can have multiple students enrolled.

**2. Course Entity:**

* Attributes:
  + CourseID (Primary Key): Unique identifier for each course.
  + CourseName: Name of the course.
  + Description: Description of the course.
* Relationships:
  + Enrollments: Many-to-Many relationship with the Student entity. A course can have multiple students enrolled, and a student can be enrolled in multiple courses.
  + InstructorCourseAssignments: Many-to-Many relationship with the Lecturer entity. Multiple lecturers can be assigned to teach a course, and a lecturer can be assigned to teach multiple courses.

**3. Department Entity:**

* Attributes:
  + DepartmentID (Primary Key): Unique identifier for each department.
  + DepartmentName: Name of the department.
  + DepartmentHeadID: Foreign Key referencing the Lecturer entity to identify the department head.
* Relationships:
  + Courses: One-to-Many relationship with the Course entity. A department can offer multiple courses, and a course belongs to one department.

**4. Lecturer Entity:**

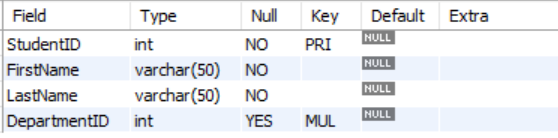
* Attributes:
  + LecturerID (Primary Key): Unique identifier for each lecturer.
  + LecturerName: Name of the lecturer.
  + Email: Email address of the lecturer.
* Relationships:
  + Department: One-to-Many relationship with the Department entity. A lecturer can be the head of one department, and a department has one department head.
  + InstructorCourseAssignments: Many-to-Many relationship with the Course entity. A lecturer can be assigned to teach multiple courses, and a course can have multiple lecturers assigned to it.

**5. Grade Entity:**

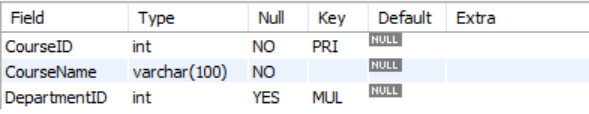
* Attributes:
  + GradeID (Primary Key): Unique identifier for each grade entry.
  + StudentID: Foreign Key referencing the Student entity to identify the student for whom the grade is recorded.
  + CourseID: Foreign Key referencing the Course entity to identify the course for which the grade is recorded.
  + LecturerID: Foreign Key referencing the Lecturer entity to identify the lecturer who assigned the grade.
  + Grade: The actual grade assigned to the student for a specific course.
* Relationships:
  + Student: One-to-Many relationship with the Student entity. A student can have multiple grade entries, each associated with a different course and lecturer.
  + Course: One-to-Many relationship with the Course entity. A course can have multiple grade entries, each associated with different students and lecturers.
  + Lecturer: One-to-Many relationship with the Lecturer entity. A lecturer can assign grades to multiple students for different courses.

Table Structure:

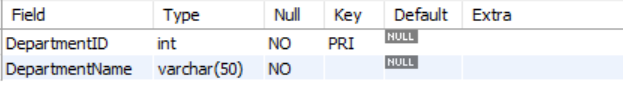
1. Student Table



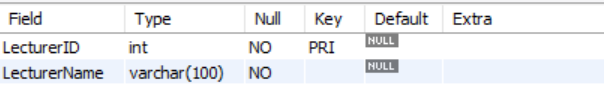
1. Course Table



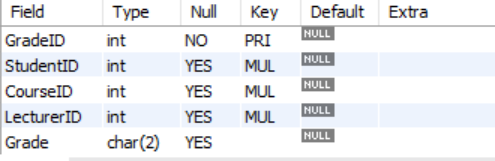
1. Department Table



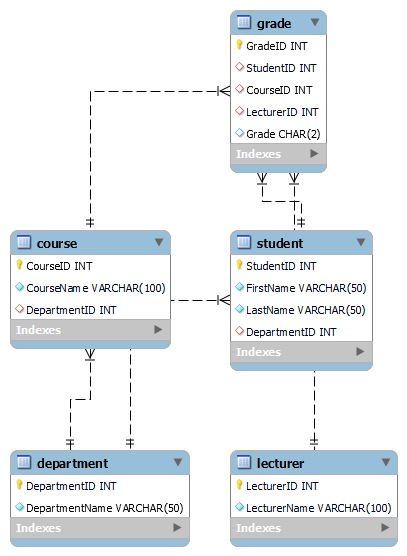
1. Lecturer Table



1. Grade Table



1. Entity-Relationship Diagram:



**In this Entity-Relationship Diagram:**

1. **Student Table**:
   * One-to-Many Relationship with the Grade Table: Each student can have multiple grade entries for different courses.
2. **Course Table**:
   * Many-to-Many Relationship with the Student Table through the Enrollment Table: Each course can have multiple students, and each student can be enrolled in multiple courses.
   * Many-to-Many Relationship with the Lecturer Table through the InstructorCourseAssignments Table: Multiple lecturers can be assigned to teach a course, and a lecturer can be assigned to teach multiple courses.
3. **Department Table**:
   * One-to-Many Relationship with the Course Table: Each department can offer multiple courses, and each course belongs to one department.
   * One-to-Many Relationship with the Lecturer Table: Each department has one department head who is a lecturer.
4. **Lecturer Table**:
   * Many-to-Many Relationship with the Course Table through the InstructorCourseAssignments Table: Lecturers can be assigned to teach multiple courses, and each course can have multiple lecturers.
   * One-to-Many Relationship with the Department Table: Each lecturer can be the head of one department, and each department has one department head.
5. **Grade Table**:
   * One-to-Many Relationship with the Student Table: Each grade entry is associated with one student, and each student can have multiple grade entries for different courses.
   * One-to-Many Relationship with the Course Table: Each grade entry is associated with one course, and each course can have multiple grade entries for different students.
   * One-to-Many Relationship with the Lecturer Table: Each grade entry is associated with one lecturer who assigned the grade.
6. **Creating a Database for Student Management System**

Query:

create database studentsdata;

Using a database

Query:

use studentsdata;

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

Query:

CREATE TABLE Department (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(50) NOT NULL

);

CREATE TABLE Course (

CourseID INT PRIMARY KEY,

CourseName VARCHAR(100) NOT NULL,

DepartmentID INT,

FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID)

);

CREATE TABLE Lecturer (

LecturerID INT PRIMARY KEY,

LecturerName VARCHAR(100) NOT NULL

);

CREATE TABLE Student (

StudentID INT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

DepartmentID INT,

FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID)

);

CREATE TABLE Grade (

GradeID INT PRIMARY KEY,

StudentID INT,

CourseID INT,

LecturerID INT,

Grade CHAR(2),

FOREIGN KEY (StudentID) REFERENCES Student(StudentID),

FOREIGN KEY (CourseID) REFERENCES Course(CourseID),

FOREIGN KEY (LecturerID) REFERENCES Lecturer(LecturerID)

);

1. **Insert Records**

-- Insert values into the Department table

INSERT INTO Department (DepartmentID, DepartmentName)

VALUES

(1, 'Computer Science'),

(2, 'Mathematics'),

(3, 'Biology'),

(4, 'History');

-- Insert values into the Course table

INSERT INTO Course (CourseID, CourseName, DepartmentID)

VALUES

(101, 'Introduction to Programming', 1),

(102, 'Calculus I', 2),

(103, 'Cell Biology', 3),

(104, 'World History', 4);

-- Insert values into the Lecturer table

INSERT INTO Lecturer (LecturerID, LecturerName)

VALUES

(201, 'Dr. Smith'),

(202, 'Prof. Johnson'),

(203, 'Dr. Lee'),

(204, 'Prof. Williams');

-- Insert values into the Student table

INSERT INTO Student (StudentID, FirstName, LastName, DepartmentID)

VALUES

(301, 'John', 'Doe', 1),

(302, 'Jane', 'Smith', 2),

(303, 'Mike', 'Johnson', 3),

(304, 'Sarah', 'Williams', 4);

-- Insert values into the Grade table

INSERT INTO Grade (GradeID, StudentID, CourseID, LecturerID, Grade)

VALUES

(401, 301, 101, 201, 'A'),

(402, 302, 102, 202, 'B+'),

(403, 303, 103, 203, 'A-'),

(404, 304, 104, 204, 'B');

1. **Select Records**
2. Retrieve a list of all students:

SELECT \* FROM Student;

1. Retrieve the details of a specific course

SELECT \* FROM Course WHERE CourseID = 101;

1. Retrieve the grades for a specific student in a specific course:

SELECT Grade

FROM Grade

WHERE StudentID = 301 AND CourseID = 101;

1. **Update Records**
2. Update a student's information

UPDATE Student

SET LastName = 'NewLastName'

WHERE StudentID = 301;

1. Modify the grade assigned to a student for a specific course

UPDATE Grade

SET Grade = 'A+'

WHERE StudentID = 301 AND CourseID = 101;

1. Change the department head for a specific department

UPDATE Department

SET DepartmentHeadID = NewInstructorID

WHERE DepartmentID = 1;

1. **Delete Records**
2. Delete a specific student by StudentID

DELETE FROM Student

WHERE StudentID = 301;

1. Delete a specific course by CourseID

DELETE FROM Course

WHERE CourseID = 101;

1. Delete all grade records for a specific course

DELETE FROM Grade

WHERE CourseID = 102;

1. Other Queries
2. List the top N students with the highest grades

SELECT StudentID, FirstName, LastName, Grade

FROM Student

INNER JOIN Grade ON Student.StudentID = Grade.StudentID

ORDER BY Grade DESC

LIMIT 5;

1. Count the number of students in each department:

SELECT Department.DepartmentName, COUNT(Student.StudentID) AS StudentCount

FROM Department

LEFT JOIN Student ON Department.DepartmentID = Student.DepartmentID

GROUP BY Department.DepartmentName;

1. Calculate the average grade for each course:

SELECT Course.CourseName, AVG(Grade) AS AverageGrade

FROM Grade

INNER JOIN Course ON Grade.CourseID = Course.CourseID

GROUP BY Course.CourseName;

1. Calculate the total number of students in the system:

SELECT COUNT(StudentID) AS TotalStudents

FROM Student;

1. Find the average grade for a specific student

SELECT AVG(CAST(Grade AS DECIMAL(5, 2))) AS AverageGrade

FROM Grade

WHERE StudentID = 301;

1. Determine the highest grade in a specific course

SELECT MAX(Grade) AS HighestGrade

FROM Grade

WHERE CourseID = 101;