**[Student Record Management System)]**

Project submitted to the SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

**Bachelor of Technology**

In

**Computer Science and Engineering**

**School of Engineering and Sciences**

Submitted by

Thalakola Nikhil Sireesh (AP22110010009)

Sugreevu Aswartha Harshitha (AP22110010012)

Kothapalli Jayadeep (AP22110010064)

Gudimetla Sarayu (AP22110010020)



**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240** **[December,**

**2022]**

# INDEX

* Project Background

* Description of the Project

* ER Diagram Creation (use any online tools to draw ER diagram)

* Description of ER diagram

* Conversion of ER diagram into Tables

* Description of Tables

* Normalization of tables up to 3-NF

* Creation of Data in the tables (at least 5 tables)

* Few sql queries on the created tables (your choice)

* Creation of 5 views using the tables

##  Project Background

The Student Record Management System aims to streamline the process of managing student data within an educational institution. It involves storing, retrieving, updating, and deleting student records efficiently using a welldesigned database system.

 **Description of the Project:**

The project involves designing and implementing a database system to manage student records. This includes creating an Entity-Relationship (ER) diagram, converting it into tables, normalizing the tables up to 3rd Normal Form (3-NF), populating the tables with sample data, writing SQL queries to retrieve information, and creating views for easy access to specific information.

Creating a database system to store, retrieve, update, and delete student records. **Steps:**

1. **ER Diagram Creation:** Visualizing entities, relationships, and attributes.

1. **Table Conversion:** Translating the ER diagram into relational tables.

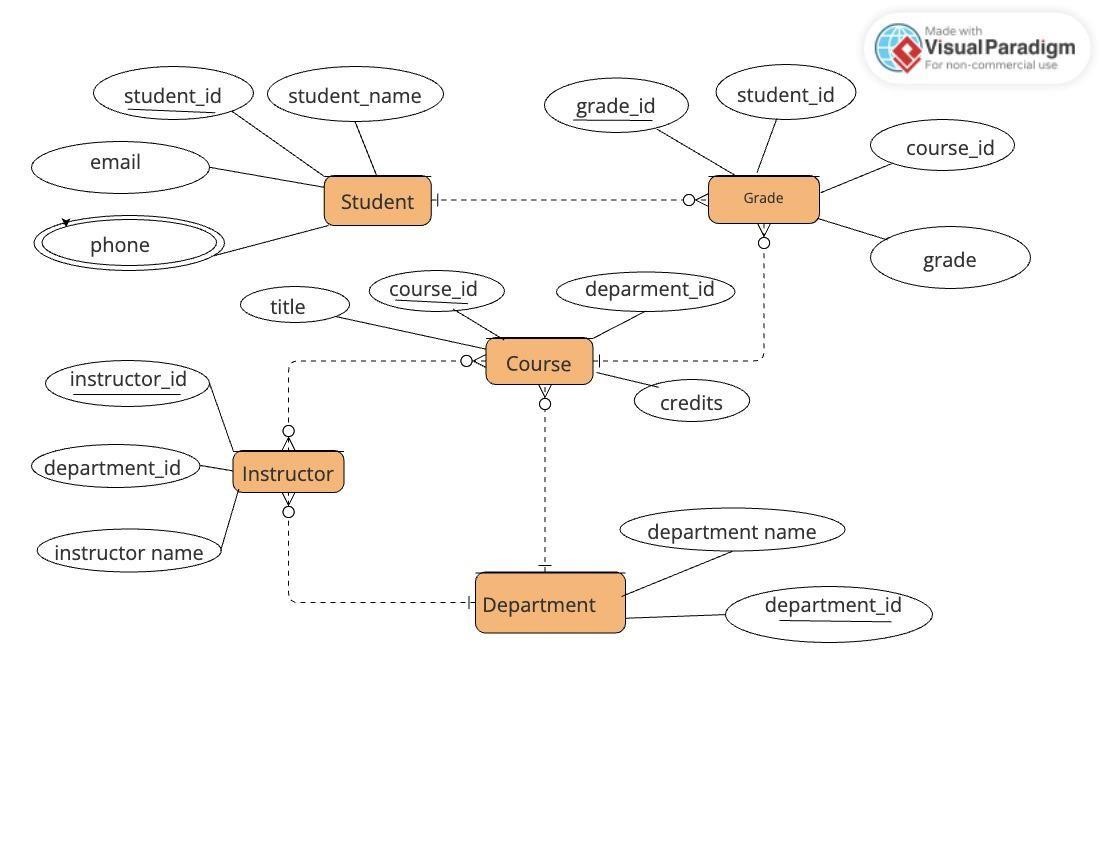
1. **Normalization:** Ensuring tables adhere to the 3rd Normal Form.

1. **Data Population:** Adding sample data to at least 5 tables.

1. **SQL Queries:** Crafting queries for data retrieval and manipulation.

1. **View Creation:** Generating 5 views for customized data perspectives.

* + ER Diagram



* + Description ER Diagram:

**Entities:**

1. **Student**: This entity represents individual students enrolled in the educational institution. Each student is uniquely identified by a StudentID. Students have attributes such as Name, Email, and Phone, which provide their personal information.
2. **Grade**: The Grade entity stores information about the grades or scores achieved by students in their courses. Each grade entry is uniquely identified by a GradeID. It includes the StudentID, linking it to the respective student, and the CourseID, linking it to the corresponding course. The Grade attribute stores the actual grade obtained by the student in the course.
3. **Course**: Courses offered by the institution are represented by this entity. Each course is uniquely identified by a CourseID. It includes attributes like Title (the name of the course) and Credits (the credit hours associated with the course). Additionally, it has a DepartmentID attribute, which is a foreign key referencing the Department entity, indicating the department responsible for offering the course.
4. **Instructor**: Instructors or teachers who conduct courses are represented in this entity. Each instructor is uniquely identified by an InstructorID. Instructors have attributes such

as Name and are associated with a specific department through the DepartmentID attribute, which serves as a foreign key referencing the Department entity.

1. **Department**: Represents the academic departments or faculties within the educational institution. Each department is identified by a DepartmentID. Departments have a Name attribute, indicating the name of the department, and possibly additional attributes such as Location.

**Attributes:**

* **Student**: StudentID (PK), Name, Email, Phone
* **Grade**: GradeID (PK), StudentID (FK), CourseID (FK), Grade
* **Course**: CourseID (PK), Title, Credits, DepartmentID (FK)
* **Instructor**: InstructorID (PK), Name, DepartmentID (FK)
* **Department**: DepartmentID (PK), Name

**Relationships:**

1. **Student - Grade (One-to-Many)**: One student can have multiple grades, as they can enroll in multiple courses and obtain grades in each course. This relationship allows for the tracking of a student's academic performance across various courses.
2. **Course - Grade (One-to-Many)**: Each course can have multiple grades associated with it, as multiple students enroll in and complete the course. This relationship facilitates the recording of grades for each student in a specific course.
3. **Instructor - Course (Many-to-Many)**: Many instructors can teach multiple courses, and each course can be taught by multiple instructors. This many-to-many relationship allows for flexible assignment of instructors to courses and vice versa.
4. **Course - Department (Many-to-One)**: Many courses belong to one department, indicating the department responsible for offering the course. This relationship helps in organizing courses under specific academic departments.
5. **Instructor - Department (Many-to-One)**: Many instructors belong to one department, indicating their affiliation with a particular academic unit. This relationship establishes the association between instructors and the departments they are part of.

Overall, this Student Record Management System captures the essential entities, attributes, and relationships required to manage student records, course offerings, instructor assignments, and departmental affiliations within an educational institution. It provides a structured framework for organizing and accessing information related to students, grades, courses, instructors, and departments.

 **Conversion of ER diagram into Tables:**

**Student Query:**

**CREATE TABLE Student ( student\_name VARCHAR(255) NOT NULL, student\_id INT PRIMARY KEY AUTO\_INCREMENT, email**

**VARCHAR(255) NOT NULL, phone VARCHAR(20)**

**);**

**Table:**

|  |  |
| --- | --- |
|  | Data Type |
| Column Name |
|  |

|  |  |
| --- | --- |
| student\_id | Primary Key  (Integer) |
| student\_name | Text |
| email | Text |
| phone | Text |

**Course**

**Query:**

CREATE TABLE Course ( course\_id INT PRIMARY

KEY AUTO\_INCREMENT, title VARCHAR(255)

NOT NULL, credits INT NOT NULL, department\_id

INT NOT NULL,

FOREIGN KEY (department\_id) REFERENCES Department (department\_id), description TEXT

);

Table:

|  |  |
| --- | --- |
|  | Data Type |
| Column Name |
|  |  |

course\_id Primary Key (Integer)

title Text

credits Integer

department\_id Foreign Key (Integer)

**Instructor Query:**

CREATE TABLE Instructor ( instructor\_id INT

PRIMARY KEY AUTO\_INCREMENT, instructor\_name

VARCHAR(255) NOT NULL, department\_id INT NOT NULL,

FOREIGN KEY (department\_id) REFERENCES Department (department\_id)

);

**Table:**

|  |  |
| --- | --- |
|  |  |
| Column Name | Data Type |

instructor\_id Primary Key (Integer)

instructor\_name Text

department\_id Foreign Key (Integer)

**Department Query:**

CREATE TABLE Department ( department\_id INT PRIMARY KEY AUTO\_INCREMENT, department\_name VARCHAR(255) NOT NULL ); **Table:**

|  |  |
| --- | --- |
| Column Name | Data Type |
|  |  |

department\_id Primary Key (Integer)

department\_name Text

**Grade**

**CREATE TABLE Grade ( student\_id INT NOT NULL, course\_id INT NOT NULL, grade TEXT NOT NULL, grade\_id INT,**

**PRIMARY KEY (student\_id, course\_id),**

**FOREIGN KEY (student\_id) REFERENCES Student (student\_id),**

**FOREIGN KEY (course\_id) REFERENCES Course (cCourseourse\_id)**

**);**

|  |  |
| --- | --- |
| Column Name | Data Type |
| grade\_id | | Primary Key (Integer) | |
| student\_id | | Foreign Key (Integer) | |
| course\_id  grade | | Foreign Key (Integer)  Text | |

 **Description of the Tables**

1. **Student Table:**
   * **Purpose:** Stores information about students.

* + **Primary Key:** student\_id (unique identifier for each student, typically an autoincrementing integer).

* + **Attributes:**
    - student\_name (text string holding the student's full name). o email (text string containing the student's email address). o phone (text string storing the student's phone number, data type might vary)

1. **Course Table:** 
   * **Purpose:** Stores information about courses offered.

* + **Primary Key:** course\_id (unique identifier for each course, typically an autoincrementing integer).

* + **Attributes:**
    - title (text string containing the course title). o credits (integer representing the number of credits the course awards).
    - department\_id (foreign key referencing the department\_id in the Department table, establishing a relationship between courses and their departments).

1. **Instructor Table:** 
   * **Purpose:** Stores information about instructors.

* + **Primary Key:** instructor\_id (unique identifier for each instructor, typically an autoincrementing integer).

* + **Attributes:**
    - instructor\_name (text string containing the instructor's full name).
    - department\_id (foreign key referencing the department\_id in the Department table, establishing a relationship between instructors and their departments).

1. **Department Table:**
   * **Purpose:** Stores information about academic departments.

* + **Primary Key:** department\_id (unique identifier for each department, typically an autoincrementing integer).

* + **Attributes:** o department\_name (text string containing the full name of the department).

1. **Grade Table** 
   * **Purpose:** Stores information about student grades in specific courses (if needed).
   * **Primary Key:** grade\_id (unique identifier for each grade record, typically an autoincrementing integer).
   * **Attributes:**
     + student\_id (foreign key referencing the student\_id in the Student table, establishing a relationship between grades and students).
     + course\_id (foreign key referencing the course\_id in the Course table, establishing a relationship between grades and courses).
     + grade (text string typically storing the letter grade earned by the student

 Normalizing the table up to 3NF

1. **Student Table:**
   * **Purpose:** Stores information about individual students.
   * **Primary Key:** student\_id (unique identifier for each student, typically an autoincrementing integer).
   * **Attributes:**
     + student\_name (text string holding the student's full name). o email (text string containing the student's email address). o phone (text string storing the student's phone number, data type might vary).

**Normalization Analysis:**

* + This table is in 1NF, 2NF, and 3NF. All attributes depend solely on the primary key (student\_id), and there are no transitive dependencies.

1. **Course Table:**
   * **Purpose:** Stores information about courses offered.
   * **Primary Key:** course\_id (unique identifier for each course, typically an autoincrementing integer).
   * **Attributes:**
     + title (text string containing the course title). o credits (integer representing the number of credits the course awards).
     + department\_id (foreign key referencing the department\_id in the Department table, establishing a relationship between courses and their departments).

**Normalization Analysis:**

* + This table is in 1NF, 2NF, and 3NF. All attributes depend solely on the primary key (course\_id), and there are no transitive dependencies.

1. **Instructor Table:**
   * **Purpose:** Stores information about instructors.
   * **Primary Key:** instructor\_id (unique identifier for each instructor, typically an autoincrementing integer).
   * **Attributes:**
     + instructor\_name (text string containing the instructor's full name).
     + department\_id (foreign key referencing the department\_id in the Department table, establishing a relationship between instructors and their departments).

**Normalization Analysis:**

* + This table is in 1NF, 2NF, and 3NF. All attributes depend solely on the primary key

(instructor\_id), and there are no transitive dependencies.

1. **Department Table:**
   * **Purpose:** Stores information about academic departments.
   * **Primary Key:** department\_id (unique identifier for each department, typically an autoincrementing integer).
   * **Attributes:** o department\_name (text string containing the full name of the department).

**Normalization Analysis:**

* + This table inherently satisfies 1NF, 2NF, and 3NF as it has only one attribute and no possibility of dependencies.

**Grade Table**

* + **Purpose:** Stores information about student grades in specific courses (if needed).
  + **Primary Key:** Composite key consisting of student\_id (foreign key referencing Student.student\_id) and course\_id (foreign key referencing Course.course\_id).
  + **Attributes:** o grade (text string typically storing the letter grade earned by the

**Normalization Analysis:**

* + **1NF:** The table is in 1NF because there are no repeating groups within a single cell.

* + **2NF:** The table satisfies 2NF because all attributes depend on the composite primary key (student\_id and course\_id) and not on any subset of it.

* + **3NF:** There might be a potential argument for a transitive dependency here. While grade directly depends on the composite key, it could be argued that it indirectly depends on department\_id through course\_id.

Grade Normalization to 3NF with Tables:

Here's the breakdown of Grade table normalization to 3NF, along with the corresponding table structures:

**Strict 3NF (Separate Enrollment Table)**

1. **Student Table:**

CREATE TABLE Student ( student\_id INT PRIMARY KEY AUTO\_INCREMENT, student\_name

VARCHAR(255) NOT NULL, email VARCHAR(255) NOT NULL, phone VARCHAR(20),

PRIMARY KEY (student\_id)

);

1. **Course Table:**

CREATE TABLE Course ( course\_id INT

PRIMARY KEY AUTO\_INCREMENT, title

VARCHAR(255) NOT NULL, credits INT NOT

NULL, department\_id INT NOT NULL,

FOREIGN KEY (department\_id) REFERENCES Department(department\_id), description TEXT, );

1. **Enrollment Table:**

CREATE TABLE Enrollment ( enrollment\_id INT

PRIMARY KEY AUTO\_INCREMENT, student\_id INT NOT NULL, course\_id INT NOT NULL,

-- Optional attributes (semester, term, etc.)

FOREIGN KEY (student\_id) REFERENCES Student(student\_id),

FOREIGN KEY (course\_id) REFERENCES Course(course\_id) );

1. **Grade Table:**

CREATE TABLE Grade ( student\_id INT NOT NULL, course\_id INT NOT NULL, grade TEXT NOT NULL, grade\_id INT, -- Optional foreign key to Grade Details table

PRIMARY KEY (student\_id, course\_id),

FOREIGN KEY (student\_id) REFERENCES Student(student\_id),

FOREIGN KEY (course\_id) REFERENCES Course(course\_id),

FOREIGN KEY (grade\_id) REFERENCES Grade\_Details(grade\_id) );

**Explanation:**

* This approach separates enrollment information from grades, allowing for a flexible and data-integrity-focused structure.
* The Enrollment table acts as a bridge between Student and Course, ensuring a student can only be enrolled in a single course instance at a time (based on the enrollment\_id primary key).
* The Grade table uses a composite primary key of (student\_id, course\_id) to uniquely identify a grade for a specific student in a specific course.

 Creation of data in a table

1. **Student Table:**

INSERT INTO Student (student\_id, student\_name, email,phone)

VALUES (1, 'John Doe', 'john.doe@example.com', '(555) 555-1234'),

(2, 'Jane Smith', 'jane.smith@example.com', '(555) 555-5678'), (3, 'Michael Brown', 'michael.brown@example.com', '(555)

5559012'),

(4, 'Amanda Johnson', 'amanda.johnson@example.com', '(555) 555- 3456'),

(5, 'David Miller', 'david.miller@example.com', '(555) 555-7890');

|  |  |  |  |
| --- | --- | --- | --- |
| student\_Id | student\_name | email | phone |
|  |  |  |  |

* 1. John Doe john.doe@example.com (555) 555-1234
  2. Jane Smith jane.smith@example.com (555) 555-5678
  3. Michael Brown michael.brown@example.com (555) 555-9012
  4. Amanda Johnson amanda.johnson@example.com (555) 555-

3456

* 1. David Miller david.miller@example.com (555) 555-7890

1. **Course Table:**

INSERT INTO Course (course\_id, Title, credits, department\_id)

VALUES (1, 'Introduction to Computer Science', 3, 1),

(2, 'Calculus I', 4, 2),

(3, 'Introduction to Literature', 3, 3),

(4, 'Biology I', 4, 4),

(5, 'History of Western Civilization', 3, 5);

|  |  |  |  |
| --- | --- | --- | --- |
| course\_id | title | credits | department\_Id |
|  |  |  |  |

* 1. Introduction to Computer Science 3 1
  2. Calculus I 4 2
  3. Introduction to Literature 3 3
  4. Biology I 4 4
  5. History of Western Civilization 3 5

1. **Department Table:**

INSERT INTO Department (department\_id, department\_name)

VALUES (1, 'Computer Science'),

(2, 'Mathematics'),

(3, 'English Literature'),

(4, 'Biology'),

(5, 'History');

|  |  |
| --- | --- |
| department\_id | department\_name |

* + 1. Computer Science
    2. Mathematics
    3. English Literature
    4. Biology
    5. History

1. **Instructor Table:**

INSERT INTO Instructor (instructor\_name, department\_id, instructor\_id)

VALUES ('Professor Jones', 1, 'IJONES123'),

('Dr. Miller', 2, 'DMILLER456'),

('Ms. Garcia', 3, 'MGARCIA789'),

('Dr. Chen', 4, 'DCHEN012'),

('Professor Williams', 5, 'PWILLIAMS345');

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | |  | | |
| instructor\_id | instructor\_name | | department\_id | | |
|  |  | |  | | |
| I JONES123 | | | Professor Jones | | 1 |
| D MILLER456 | | | Dr. Miller | | 2 |
| M GARCIA789 | | | Ms. Garcia | | 3 |
| D CHEN012 | | | Dr. Chen | | 4 |
| P WILLIAMS345    **5. Grade Table:** | | | Professor Williams | | 5 |

INSERT INTO Grade (student\_id, course\_id, grade, grade\_id)

VALUES (1, 1, 'A', 101),

(2, 2, 'B', 102),

(3, 3, 'C', 103),

(4, 4, 'A-', 104),

(5, 5, 'B+', 105);

student\_id course\_id grade grade\_id

1. 1 A 101
2. 2 B 102
3. 3 C 103
4. 4 A- 104
5. 5 B+ 105

###  Few SQL queries on the created tables

1. **List all instructors and their departments:**

SELECT i.instructor\_name, d.department\_name

FROM Instructor i

INNER JOIN Department d ON i.department\_id = d.department\_id;

1. **Find all students enrolled in a specific course (course ID = 101):**

SELECT s.student\_name

FROM Student s

INNER JOIN Enrollment e ON s.student\_id = e.student\_id

WHERE e.course\_id = 101;

1. **Get the average grade for a particular course (course ID = 101):**

SELECT AVG(g.grade) AS average\_grade

FROM Grade g

INNER JOIN Enrollment e ON g.student\_id = e.student\_id

WHERE e.course\_id = 101;

1. **List all students with a grade of 'A' and their instructors:**

SELECT s.student\_name, i.instructor\_name

FROM Student s

INNER JOIN Enrollment e ON s.student\_id = e.student\_id

INNER JOIN Grade g ON e.student\_id = g.student\_id AND e.course\_id = g.course\_id

INNER JOIN Instructor i ON e.course\_id = i.course\_id -- Assuming instructors teach courses they are enrolled in WHERE g.grade = 'A';

 Creation of 5 views using the tables

1. **View for Instructor Information with Department Details:**

CREATE VIEW InstructorDetails AS

SELECT i.instructor\_name, d.department\_name

FROM Instructor i

INNER JOIN Department d ON i.department\_id = d.department\_id;

1. **View for Enrolled Students with Course Details:**

CREATE VIEW EnrolledStudents AS

SELECT s.student\_name, c.title, c.credits

FROM Student s

INNER JOIN Enrollment e ON s.student\_id = e.student\_id

INNER JOIN Course c ON e.course\_id = c.course\_id;

1. **View for Student Grades with Course Information:**

CREATE VIEW StudentGrades AS

SELECT s.student\_name, c.title, g.grade

FROM Student s

INNER JOIN Enrollment e ON s.student\_id = e.student\_id

INNER JOIN Grade g ON e.student\_id = g.student\_id AND e.course\_id = g.course\_id

INNER JOIN Course c ON e.course\_id = c.course\_id;

1. **View for Average Grades per Course:**

CREATE VIEW AverageGradesPerCourse AS

SELECT c.title, AVG(g.grade) AS average\_grade

FROM Course c

INNER JOIN Enrollment e ON c.course\_id = e.course\_id

INNER JOIN Grade g ON e.student\_id = g.student\_id AND e.course\_id = g.course\_id

GROUP BY c.course\_id;

**5.View for Students with Low Grades and Instructor Emails:**

CREATE VIEW StudentsWithLowGrades

AS

SELECT s.student\_name, c.title, g.grade, i.email

FROM Student s

INNER JOIN Enrollment e ON s.student\_id = e.student\_id

INNER JOIN Grade g ON e.student\_id = g.student\_id AND e.course\_id = g.course\_id

INNER JOIN Course c ON e.course\_id = c.course\_id

INNER JOIN Instructor i ON e.course\_id = i.course\_id

WHERE g.grade < 'C'

AND i.email IS NOT NULL;

## THANK YOU