[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)

SRM University–AP
Neerukonda, Mangalagiri, Guntur
Andhra Pradesh – 522 240 [December, 2022]

INDEX

✞ 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.

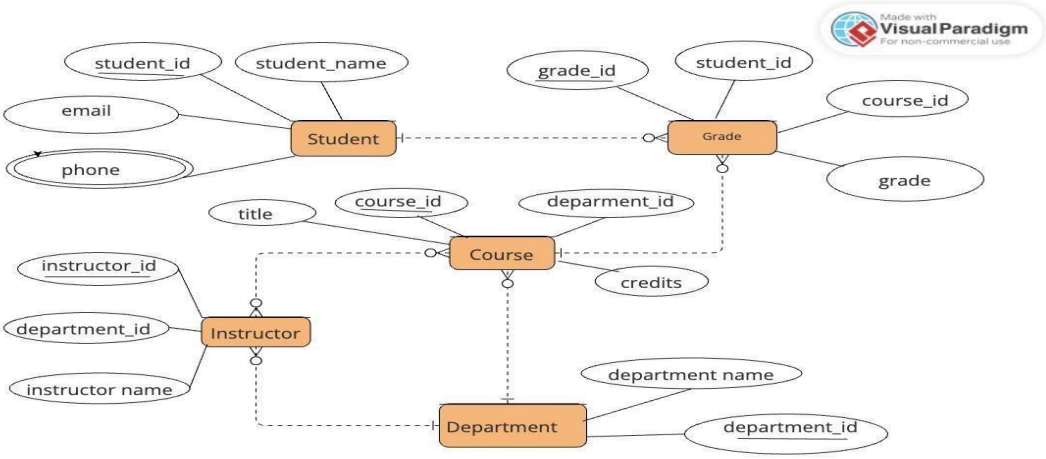
✞ Description of the Project:

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.
2. **Table Conversion:** Translating the ER diagram into relational tables.
3. **Normalization:** Ensuring tables adhere to the 3rd Normal Form.
4. **Data Population:** Adding sample data to at least 5 tables.
5. **SQL Queries:** Crafting queries for data retrieval and manipulation.
6. **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.

5. **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:

Column Name	Data Type
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:

Column Name	Data Type
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	Foreign Key (Integer)
grade	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). `email` (text string containing the student's email address). `phone` (text string storing the student's phone number, data type might vary)

2. 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).

3. 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).

4. 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).

5. 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.

2. 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.

3. 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.

4. 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)
);
```

2. 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, );
```

3. 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)  );
```

4. 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
5	David Miller	david.miller@example.com	(555) 555-7890

2. 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

3. 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

4. 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	Professor Williams	5

5. Grade Table:

```
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;
```

2. 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;
```

3. 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;
```

4. 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;
```

2. 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;
```

3. 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;
```

4. 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