# [Student Management System (DBMS)]

Project submitted to the

SRM University – AP, Andhra Pradesh

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

### Bachelor of Technology

In

### Computer Science and Engineering School of Engineering and Sciences

Submitted by

### AP22110010008 AP22110010041 AP22110010048 AP22110010051



**SRM University–AP Neerukonda, Mangalagiri, Guntur**

### Andhra Pradesh – 522 240

**[December, 2022]**

INDEX

Abstract, Aim, Attributes, Entity… 3-4

Relations 5

ER-Model Diagram 6

[Relational Diagram 7](#_TOC_250002)

[Normalization… 8-9](#_TOC_250001)

[Entity records 10-16](#_TOC_250000)

QUERIES AND VIEWS 17-20

**Abstract**:

This database management system (DBMS) project focuses on managing student information within an educational institution. The system is designed to handle various aspects of student data, including contact details, enrollment in courses, academic performance, and departmental information.

The database consists of several key components.The database structure facilitates efficient management of student data and enables users to retrieve relevant information easily. With this system in place, administrators, faculty members, and students can access and utilize accurate and organized information to support various academic and administrative tasks within the institution.

**Aim**: to efficiently manage student information and academic data within an educational institution

**Entity:** a “thing” or “object” in the enterprise that is distinguishable from other objects

* Described by a set of attributes List of Entities in the database :

1. Tables:
   * Contacts: Stores contact information such as email, phone numbers, and addresses for individuals associated with the institution.
   * Students: Contains data about enrolled students, including their names and contact information.
   * Courses: Stores information about the courses offered by the institution, including course names and credit values.
   * Grades: Records the grades achieved by students in various courses.
   * Departments: Stores information about different academic departments, including department names, heads of departments, and locations.
2. Views:
   * StudentGrades: Provides a view of student names along with the courses they are enrolled in and their corresponding grades.
   * EnrolledStudents: Presents a count of students enrolled in each course.
   * StudentContactInfo: Offers a view of student names along with their contact email and phone numbers.
   * HighGrades: Displays students with high grades (A or A+) along with the corresponding course and grade.
   * LowGrades: Presents students with low grades (D or F) along with the corresponding course and grade.

**Attributes** — characteristics of an entity, and has an oval symbol. There are different types of attributes :

* ***Key attribute:*** An attribute uniquely distinguishes the entity in an entity set.
* ***Simple attribute:*** An attribute that cannot be further subdivided into components.
* ***Composite attribute:*** An attribute that can be split into components.
* ***Single-valued attribute:*** The attribute which takes up only a single value for each entity instance.
* ***Multi-valued attribute:*** The attribute which takes up more than a single value for

each entity instance.

* ***Stored attribute:*** An attribute that stores the data which can be used to get the derived attribute.
* ***Derived attribute:*** An attribute that can be derived from other attributes. here are the attributes of this project:

1. ContactID: Unique identifier for each contact record.
2. Email: Email address of the contact person.
3. Phone: Phone number of the contact person.
4. Address: Physical address of the contact person.
5. StudentID: Unique identifier for each student.
6. Name: Name of the student.
7. CourseID: Unique identifier for each course.
8. Name (in Courses): Name of the course.
9. Credits: Number of credits associated with the course.
10. GradeID: Unique identifier for each grade record.
11. Grade: Grade achieved by the student in a course. 12.DepartmentID: Unique identifier for each department. 13.DepartmentName: Name of the department.
12. HeadOfDepartment: Name of the head of the department.
13. Location: Location of the department within the institution.

A relationship is an association among several entities Relationships:

1. Contacts:
   * ContactID is related to Email, Phone, and Address.
2. Students:
   * StudentID is related to Name and ContactID.
3. Courses:
   * CourseID is related to Name and Credits.
4. Grades:
   * GradeID is related to StudentID, CourseID, and Grade.
5. Departments:
   * DepartmentID is related to DepartmentName, HeadOfDepartment, and Location.

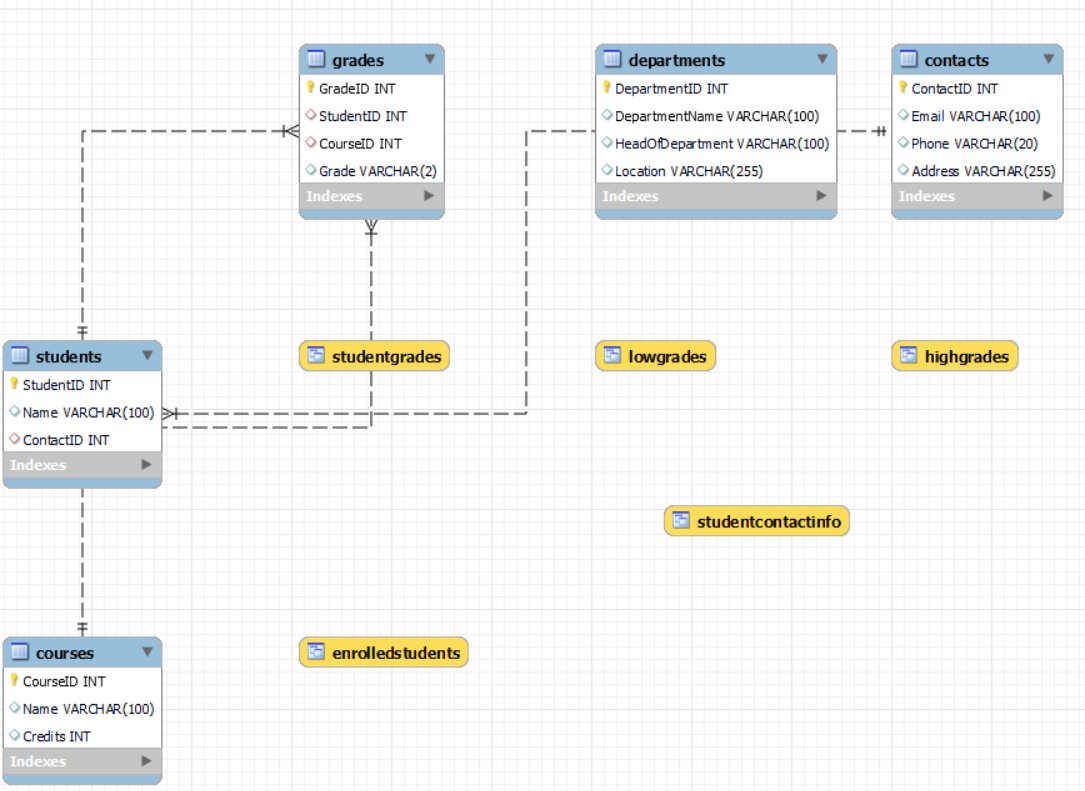
# E-R Model:

ER model stands for an Entity-Relationship model.

It is a high-level data model. This model is used to define the data elements and relationships for a specified system.

It develops a conceptual design for the database. It also develops a very simple and easy-to- design view of data.

ER diagram here



### Relational Diagram:

Converting ER model to tables/relations, commonly used, flexible.

Each and every column header is called an attribute. The row header is called a

tuple.

Relational diagram here

Contacts(ContactID, Email, Phone, Address)

|

| Students(StudentID, Name, ContactID)

|

| StudentContactInfo (View:Students.Name,Contacts.Email, Contacts.Phone)

Courses(CourseID, Name, Credits)

|

| Grades(GradeID, StudentID, CourseID, Grade)

|

| StudentGrades (View: Students.Name, Courses.Name AS Course, Grades.Grade)

|

| EnrolledStudents (View: Courses.Name AS Course, COUNT(\*) AS EnrolledStudents)

Departments(DepartmentID, DepartmentName, HeadOfDepartment, Location)

**Normalization:**

Normalization is used to minimize the redundancy from a relation or set of relations.

1. First Normal Form ( 1NF ):

A relation is said to be in its First Normal form if it has got no non-atomic attribute.

(Non-atomic attribute means the attribute which can’t be subdivided).

The database schema, as represented by the provided code, is in the First Normal Form (1NF), with each table having a unique primary key and each column containing atomic values, ensuring basic data organization and integrity.

1. Second Normal Form (2NF):

A relation that is in 1NF is said to have a second normal form if it satisfies any one of the following conditions.

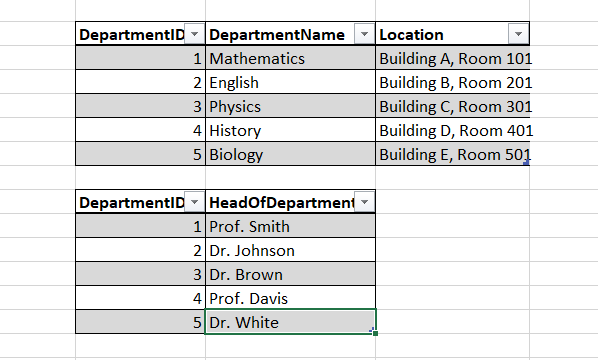
* 1. The primary key contains only one attribute.
  2. There exist no non-key attributes.
  3. Every non-key attribute present in the relation should functionally depend upon a full set of the primary key.

The database schema, as represented by the provided code, is in the Second Normal Form (2NF), with each table having a unique primary key and each column containing atomic values, ensuring basic data organization and integrity.

1. Third Normal Form (3NF).

The relation in 2Nf is said to be 3NF if there exists no transitive dependency of any non-key attribute on the set of the primary key.

In the Departments table, converted into Third Normal Form (3NF), each attribute is functionally dependent only on the candidate keys, ensuring minimal data redundancy and maintaining data integrity.



## ENTITY RECORDS

1. Table Table: Contact

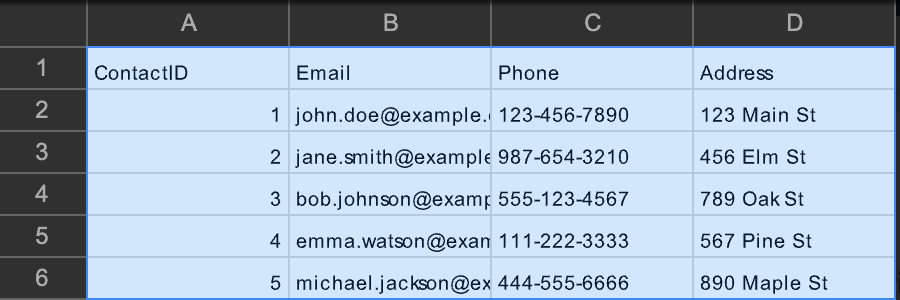
Below is the SQL command to Create Contact Table

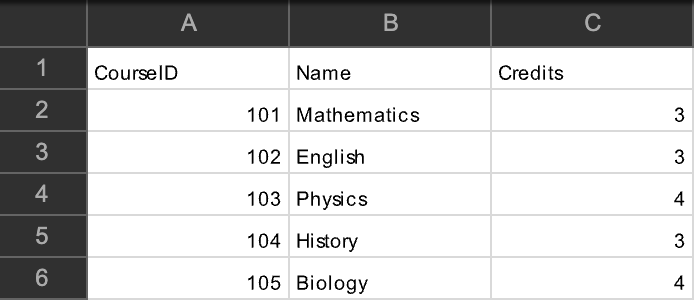
CREATE TABLE Contacts ( ContactID INT PRIMARY KEY, Email VARCHAR(100),

Phone VARCHAR(20),

Address VARCHAR(255)

);





1. Table: Courses

Below is the SQL command to Create Courses Table CREATE TABLE Courses (

CourseID INT PRIMARY KEY,

Name VARCHAR(100),

Credits INT

);

INSERT INTO Courses (CourseID, Name, Credits) VALUES (101, 'Mathematics', 3),

(102, 'English', 3),

(103, 'Physics', 4),

(104, 'History', 3),

(105, 'Biology', 4);

1. Table: Department

Below is the SQL command to Create Department Table

CREATE TABLE Departments (

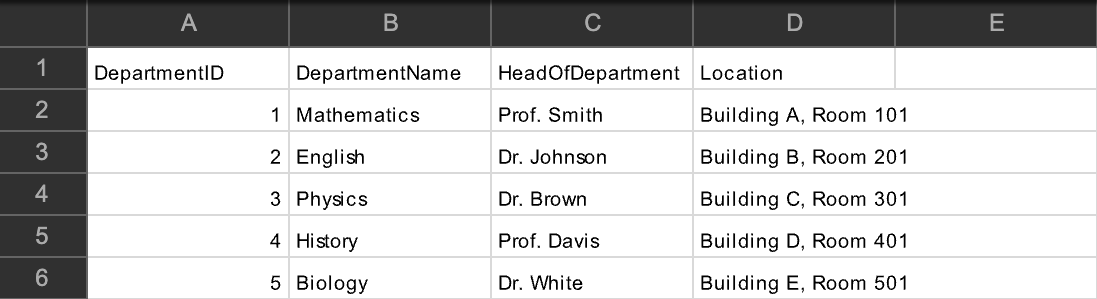
DepartmentID INT PRIMARY KEY AUTO\_INCREMENT,

DepartmentName VARCHAR(100) UNIQUE,

HeadOfDepartment VARCHAR(100),

Location VARCHAR(255)

);



1. Table : Grade

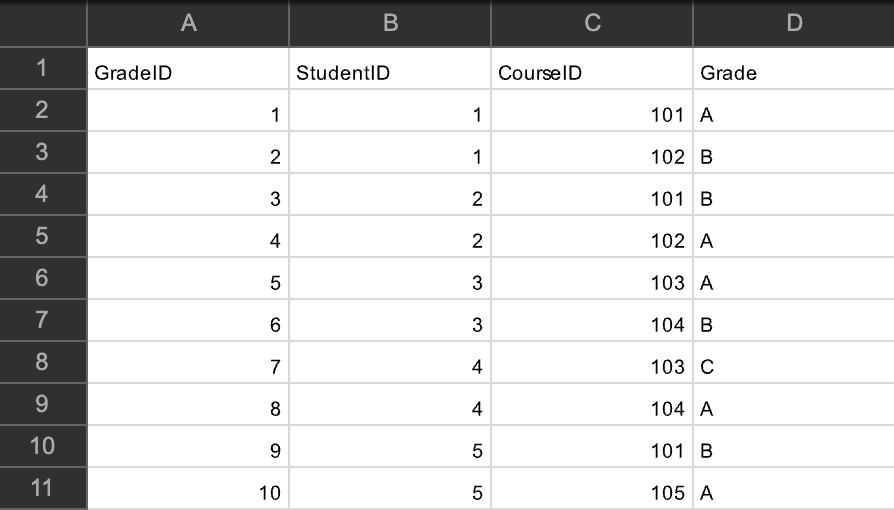
Below is the SQL command to Create Grade Table CREATE TABLE Grades (

GradeID INT PRIMARY KEY,

StudentID INT, CourseID INT, Grade VARCHAR(2),

FOREIGN KEY (StudentID) REFERENCES Students(StudentID), FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)

);



1. Table : Student contact info

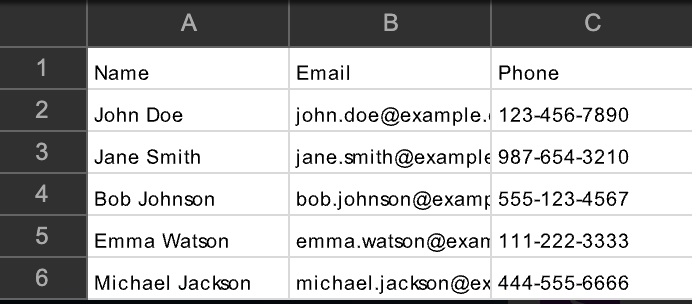
Below is the SQL command to Create Student contact info Table

CREATE TABLE StudentContactInfo ( StudentID INT PRIMARY KEY, Name VARCHAR(100),

Email VARCHAR(100), Phone VARCHAR(20),

Address VARCHAR(255)

);

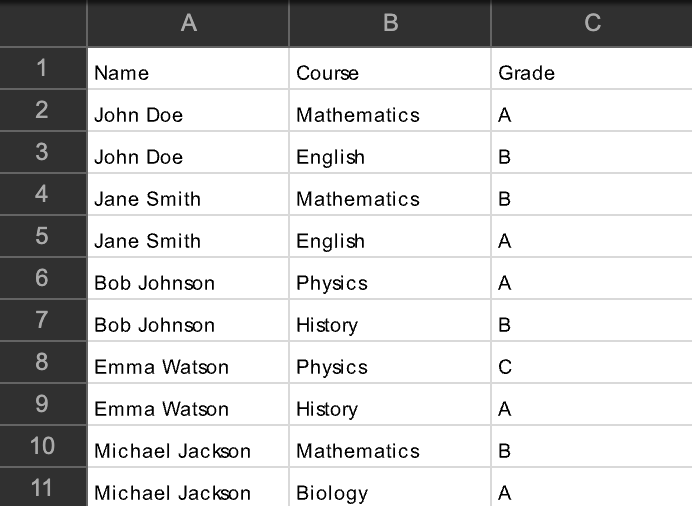


1. Table: Student Grades

Below is the SQL command to Create Student Grades Table

CREATE TABLE StudentGrades ( StudentName VARCHAR(100), CourseName VARCHAR(100), Grade VARCHAR(2)

);



1. Table: Students

Below is the SQL command to Create Students Table CREATE TABLE Students (

StudentID INT PRIMARY KEY, Name VARCHAR(100),

ContactID INT,

FOREIGN KEY (ContactID) REFERENCES Contacts(ContactID)

);

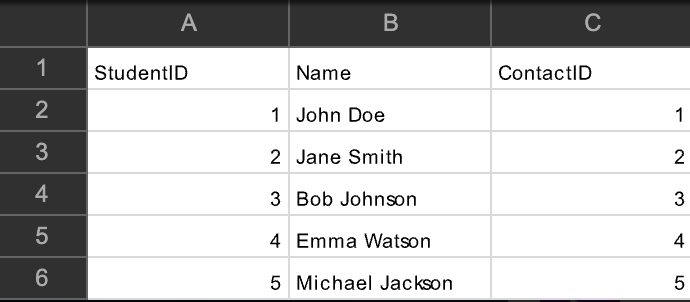
INSERT INTO Students (StudentID, Name, ContactID) VALUES (1, 'John Doe', 1),

(2, 'Jane Smith', 2),

(3, 'Bob Johnson', 3),

(4, 'Emma Watson', 4),

(5, 'Michael Jackson', 5);



### Query 1:

**\*\*Update a student's email address:\*\***

### ```sql

**UPDATE Contacts**

### SET Email = 'new\_email@example.com'

**WHERE ContactID = (SELECT ContactID FROM Students WHERE Name = 'John Doe');**

### ``` Query 2.

**\*\*Insert a new course into the Courses table:\*\***

### ```sql

**INSERT INTO Courses (CourseID, Name, Credits) VALUES (106, 'Chemistry', 4);**

### ``` Query 3.

**\*\*Delete a course from the Courses table:\*\***

### ```sql

**DELETE FROM Courses WHERE CourseID = 105;**

### Query 4.

**\*\*Update a student's grade in a particular course:\*\***

### ```sql

**UPDATE Grades SET Grade = 'B+'**

### WHERE StudentID = (SELECT StudentID FROM Students WHERE Name = 'John Doe')

**AND CourseID = (SELECT CourseID FROM Courses WHERE Name = ‘Mathematics');**

### Query 5.

**\*\*Insert a new student record into the Students table along with their contact information:\*\***

### ```sql

**INSERT INTO Contacts (ContactID, Email, Phone, Address) VALUES (6, 'new\_student@example.com', '999-888-7777', '123 Oak St');**

### INSERT INTO Students (StudentID, Name, ContactID) VALUES (6, 'New Student', 6);

**Query 6.**

### \*\*Delete a student record from the Students table along with their contact information:\*\*

**```sql**

### DELETE FROM Students

**WHERE Name = 'Michael Jackson';**

### DELETE FROM Contacts

**WHERE ContactID = (SELECT ContactID FROM Students WHERE Name = 'Michael Jackson');**

# VIEWS:

1. CREATE VIEW StudentGrades AS

SELECT Students.Name, Courses.Name AS Course, Grades.Grade FROM Students

JOIN Grades ON Students.StudentID = Grades.StudentID JOIN Courses ON Grades.CourseID = Courses.CourseID**;**

1. CREATE VIEW EnrolledStudents AS

SELECT Courses.Name AS Course, COUNT(\*) AS EnrolledStudents

FROM Courses

JOIN Grades ON Courses.CourseID = Grades.CourseID GROUP BY Courses.Name;

1. CREATE VIEW StudentContactInfo AS

SELECT Students.Name, Contacts.Email, Contacts.Phone FROM Students

JOIN Contacts ON Students.ContactID = Contacts.ContactID;

1. CREATE VIEW LowGrades AS

SELECT Students.Name, Courses.Name AS Course, Grades.Grade

FROM Students

JOIN Grades ON Students.StudentID = Grades.StudentID JOIN Courses ON Grades.CourseID = Courses.CourseID WHERE Grades.Grade IN ('D', ‘F');

1. CREATE VIEW HighGrades AS

SELECT Students.Name, Courses.Name AS Course, Grades.Grade

FROM Students

JOIN Grades ON Students.StudentID = Grades.StudentID JOIN Courses ON Grades.CourseID = Courses.CourseID WHERE Grades.Grade IN ('A', 'A+');