[School Data Base Management System (SDBMS)]

The 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
G. Harshit – AP22110010076
P. Vishnu – AP22110010094
V. Revanth – AP22110010109

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

INDEX

- Abstract, Aim, Project Background …..3-4
- Description of Project……..3-4
- Schema Representation…..5
- ER-Model Diagram……..6
- Description of ER Diagram……..7-9
- Conversion of ER Diagram into Tables…9-10
- Description of Tables……..10-13
- Normalization of Tables……..14-16
- Creation of Data in Tables…..17-18
- SQL Queries…..19-20
- Creation of Views…..20-22

Abstract:
The School Database Management System (SDMS) is designed to facilitate the efficient management and retrieval of information in educational institutions. This system addresses the comprehensive needs of school administration and aims to enhance the interaction between students, teachers, and administrative staff through a well-organized database.

Aim:
The aim of a school database management system (DBMS) is multifaceted, addressing various needs of educational institutions through efficient, structured handling and storage of data.

Project Background

The School Database Management System (SDMS) is designed to streamline and optimize the management of educational institutions by digitizing student records, faculty information, and administrative processes. The primary goal is to provide a centralized repository that facilitates easy access to information and enhances the efficiency of administrative tasks, thereby facilitating smoother administrative operations and supporting the academic journey of students. In today's educational landscape, managing vast amounts of data efficiently and accurately is crucial for educational institutions to operate smoothly. A School Database Management System (DBMS) provides a centralized platform for storing, managing, and accessing various types of data related to students, teachers, courses, grades, attendance, and administrative tasks. This project aims to develop a comprehensive School DBMS to streamline administrative processes, enhance communication between stakeholders, and improve overall efficiency within the educational institution.

Description of the Project

The School Database Management System (SDMS) is an essential tool for modern educational institutions, designed to efficiently manage and organize vast amounts of information across different departments. At its core, the SDMS maintains detailed records on every student, including personal information, academic history, ID numbers, and health data, ensuring that all student-related data is secure and easily accessible

by authorized personnel. Additionally, the system comprehensively handles faculty and staff details, from employment histories and qualifications to payroll and scheduling, streamlining administrative tasks and improving operational efficiency. By integrating these functionalities into a single platform, SDMS ensures that the management of educational data is both effective and efficient, supporting the school's goals of delivering quality education and enhancing student outcomes.

The School DBMS will cover the following key functionalities:

1) Student Management: This module will handle student enrollment, personal information, academic records, attendance, and disciplinary records.

2) Teacher Management: This module will manage teacher profiles, qualifications, teaching assignments, and performance evaluations.

3) Course Management: This module will catalog all courses offered by the school, including course details, schedules, prerequisites, and faculty assignments.

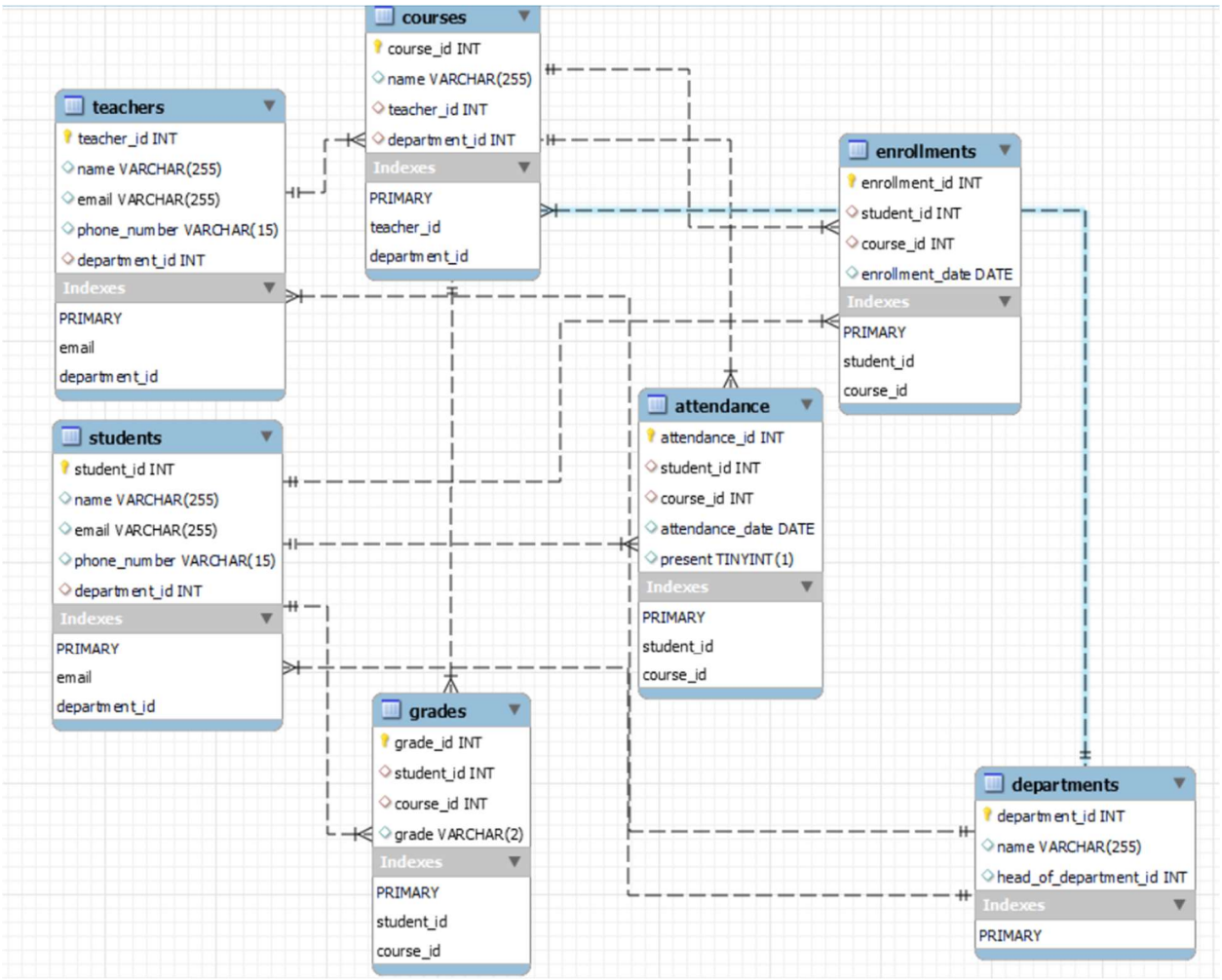
4) Attendance Tracking: This module will automate attendance tracking for both students and teachers, providing real-time data on attendance status.

5) Grade Management: This module will facilitate the recording and calculation of student grades for individual assignments, exams, and overall course performance.

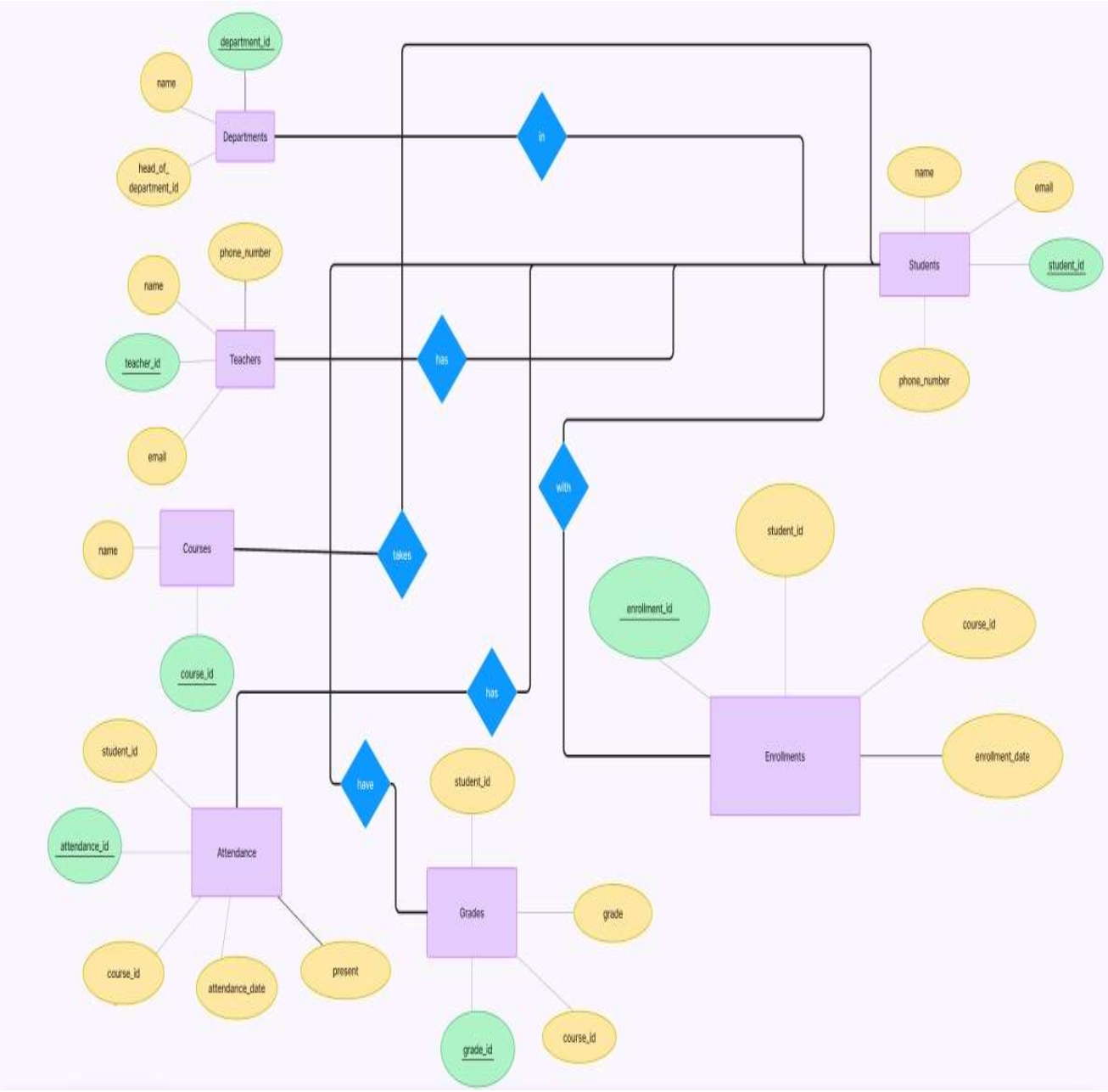
6) Administrative Tasks: This module will include functionalities for managing administrative tasks such as scheduling, resource allocation, fee management, and reporting.

7) Communication Platform: The system will provide a platform for communication between stakeholders, including students, teachers, parents, and administrative staff, through announcements, notifications, and messaging features.

Schematic Representation



ER Diagram:


department_id
name
Departments
head_of_department_id
in
name
email
Students
student_id
phone_number
phone_number
name
teacher_id
Teachers
email
has
with
name
Courses
course_id
takes
student_id
enrollment_id
course_id
has
Enrollments
enrollment_date
student_id
attendance_id
Attendance
have
student_id
Grades
grade
course_id
attendance_date
present
grade_id
course_id

Description of ER diagram

1. **Entities**:
Departments: Represents various departments within the school, each identified uniquely by a `department_id`. Each department has a name (`name`) and optionally may have a head of department, identified by `head_of_department_id`.

Teachers: Represents the teaching staff of the school, identified uniquely by a `teacher_id`. Each teacher has a name, email, and phone number. They are associated with a department through the `department_id` attribute.

Students: Represents the enrolled students, each identified by a unique `student_id`. Students have a name, email, and phone number, and they belong to a specific department as denoted by the `department_id`.

Courses: Represents the various courses offered by the school, identified by a unique `course_id`. Each course has a name and is associated with a teacher (identified by `teacher_id`) and a department (`department_id`).

Enrollments: Represents the enrollment of students in courses. Each enrollment is uniquely identified by an `enrollment_id` and is associated with a student (`student_id`) and a course (`course_id`). It also stores the date of enrollment (`enrollment_date`).

Attendance: Represents the attendance records of students in courses. Each attendance record is uniquely identified by an `attendance_id` and is associated with a student (`student_id`) and a course (`course_id`). It also stores the date of attendance (`attendance_date`) and whether the student was present (`present`).

Grades: Represents the grades assigned to students for their performance in courses. Each grade record is uniquely identified by a `grade_id` and is associated with a student (`student_id`) and a course (`course_id`). It stores the actual grade assigned (`grade`), which could be represented as a letter grade (e.g., 'A', 'B+', 'C') or a numerical grade (e.g., 90, 85.5).

2. **Relationships**:
Department-Teacher Relationship: There is a **one-to-many** relationship between Departments and Teachers, where each department can have multiple teachers but each teacher belongs to only one department. This relationship is established through the `department_id` attribute in both tables.

Department-Head Relationship: There is a **one-to-one (or zero)** relationship between Departments and Teachers for the head of department. This relationship is established through

the `head_of_department_id` attribute in the Departments table, which references the `teacher_id` in the Teachers table.
Department-Student Relationship: Similar to the Department-Teacher relationship, there is a **one-to-many** relationship between Departments and Students, where each department can have multiple students but each student belongs to only one department. This relationship is established through the `department_id` attribute in both tables.

Course-Teacher Relationship: There is **a one-to-many** relationship between Courses and Teachers, where each course is taught by one teacher, but a teacher can teach multiple courses. This relationship is established through the `teacher_id` attribute in the Courses table, referencing the `teacher_id` in the Teachers table.

Course-Department Relationship: Similar to the Department-Teacher relationship, there is a **one-to-many** relationship between Courses and Departments, where each course belongs to one department, but a department can offer multiple courses. This relationship is established through the `department_id` attribute in both tables.

Enrollment-Student Relationship: There is a **one-to-many** relationship between Enrollments and Students, where each enrollment is associated with one student, but a student can have multiple enrollments. This relationship is established through the `student_id` attribute in the Enrollments table, referencing the `student_id` in the Students table.

Enrollment-Course Relationship: Similar to the Enrollment-Student relationship, there is a **one-to-many** relationship between Enrollments and Courses, where each enrollment is associated with one course, but a course can have multiple enrollments. This relationship is established through the `course_id` attribute in the Enrollments table, referencing the `course_id` in the Courses table.

Course-Grades Relationship: There is a **one-to-many** relationship between Courses and Grades, where each course can have multiple grades assigned to different students, but a grade is associated with only one course. This relationship is established through the course_id attribute in the Grades table, referencing the course_id in the Courses table.

Student-Grades Relationship: There is a **one-to-many** relationship between Students and Grades, where each student can have multiple grades assigned for different courses, but a grade is associated with only one student. This relationship is established through the student_id attribute in the Grades table, referencing the student_id in the Students table.

Course-Attendance Relationship: There is a **one-to-many** relationship between Courses and Attendance, where each course can have multiple attendance records for different dates, but an attendance record is associated with only one course. This relationship is established through the course_id attribute in the Attendance table, referencing the course_id in the Courses table.

Student-Attendance Relationship: There is a **one-to-many** relationship between Students and Attendance, where each student can have multiple attendance records for different courses and dates, but an attendance record is associated with only one student. This relationship is established through the student_id attribute in the Attendance table, referencing the student_id in the Students table.

This ER diagram represents the structure of the school database and the relationships between its entities, facilitating a better understanding of how data is organized and connected within the system.

Conversion of ER diagram into Tables

```
create database harshit;
use harshit;
CREATE TABLE Departments (
    department_id INT PRIMARY KEY,
    name VARCHAR(255),
    head_of_department_id INT
);
CREATE TABLE Teachers (
    teacher_id INT PRIMARY KEY,
    name VARCHAR(255),
    email VARCHAR(255) UNIQUE,
    phone_number VARCHAR(15),
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES Departments(department_id)
);
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    name VARCHAR(255),
    email VARCHAR(255) UNIQUE,
    phone_number VARCHAR(15),
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES Departments(department_id)
);
CREATE TABLE Courses (
    course_id INT PRIMARY KEY,
    name VARCHAR(255),
    teacher_id INT,
    department_id INT,
    FOREIGN KEY (teacher_id) REFERENCES Teachers(teacher_id),
    FOREIGN KEY (department_id) REFERENCES Departments(department_id)
);
```

```sql
CREATE TABLE Enrollments (
    enrollment_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    enrollment_date DATE,
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);

CREATE TABLE Grades (
    grade_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    grade VARCHAR(2),
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
CREATE TABLE Attendance (
    attendance_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    attendance_date DATE,
    present BOOLEAN,
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```

Description of Tables

1) **Departments Table**:

❖ Attributes:

- department_id: A unique identifier for each department, serving as the primary key of the table.
- name: The name of the department, stored as a variable-length string.
- head_of_department_id: An optional reference to the head of the department, linking to the teacher_id in the Teachers table.

o Description: This table stores information about the departments within the school. Each department has a unique identifier (department_id) and a name (name). Additionally, it optionally records the teacher who serves as the head of the department (head_of_department_id). The foreign key constraint ensures that the head_of_department_id references a valid teacher from the Teachers table.

2) **Teachers Table**:

❖ Attributes:

- teacher_id: A unique identifier for each teacher, serving as the primary key of the table.
- name: The name of the teacher, stored as a variable-length string.
- email: The email address of the teacher, stored as a unique string.
- phone_number: The phone number of the teacher, stored as a string.
- department_id: The department to which the teacher belongs, referencing the department_id in the Departments table.

o Description: This table contains information about the teachers employed by the school. Each teacher is uniquely identified by their teacher_Id, and their details such as name, email, and phone number are recorded. The department_id attribute establishes a relationship between teachers and the department they are associated with via a foreign key constraint.

3) **Students Table**:

❖ Attributes:

- student_id: A unique identifier for each student, serving as the primary key of the table.
- name: The name of the student, stored as a variable-length string.

- email: The email address of the student, stored as a unique string.
- phone_number: The phone number of the student, stored as a string.
- department_id: The department in which the student is enrolled, referencing the department_id in the Departments table.

- Description: This table stores information about the students enrolled in the school. Each student has a unique identifier (student_id) and personal details such as name, email, and phone number. The department_id attribute establishes a relationship between students and the department in which they are enrolled via a foreign key constraint.

4) **Courses Table**:

❖ Attributes:
- course_id: A unique identifier for each course, serving as the primary key of the table.
- name: The name of the course, stored as a variable-length string.
- teacher_id: The teacher responsible for teaching the course, referencing the teacher_id in the Teachers table.
- department_id: The department to which the course belongs, referencing the department_id in the Departments table.

- Description: This table contains information about the courses offered by the school. Each course has a unique identifier (course_id) and a name (name). The teacher_id attribute establishes a relationship between courses and the teacher responsible for teaching the course, while the department_id attribute indicates the department to which the course belongs. Foreign key constraints ensure that valid references are made to teachers and departments.

5) **Enrollments Table**:

❖ Attributes:
- enrollment_id: A unique identifier for each enrollment record, serving as the primary key of the table.
- student_id: The student enrolled in the course, referencing the student_id in the Students table.
- course_id: The course in which the student is enrolled, referencing the course_id in the Courses table.
- enrollment_date: The date on which the student enrolled in the course, stored as a date.

- Description: This table records the enrollments of students in courses. Each enrollment is uniquely identified by an enrollment_id. The student_id attribute establishes a relationship between enrollments and the students enrolled, while the course_id attribute indicates the course in which the student is enrolled. Additionally, the enrollment_date attribute stores the date of enrollment. Foreign key constraints ensure that valid references are made to students and courses.

6) Attendance Table

Attributes:

- attendance_id: A unique identifier for each attendance record, serving as the primary key of the table.
- student_id: The ID of the student associated with the attendance record, linking to the student_id in the Students table.
- course_id: The ID of the course for which the attendance is recorded, linking to the course_id in the Courses table.
- attendance_date: The date on which the attendance was taken, stored as a date data type.
- present: A boolean value indicating whether the student was present on the specified date for the given course.

Description:
The Attendance table stores records of student attendance for various courses. Each attendance record is uniquely identified by an attendance_id. It tracks which students attended particular courses on specific dates. The student_id attribute references the ID of the student who attended, while the course_id attribute links to the course for which the attendance was taken. The attendance_date field indicates the date on which the attendance was recorded, and the present field specifies whether the student was present (true) or absent (false) for that particular session.

7) Grades Table:

Attributes:

- grade_id: A unique identifier for each grade record, serving as the primary key of the table.
- student_id: The ID of the student associated with the grade, linking to the student_id in the Students table.
- course_id: The ID of the course for which the grade is assigned, linking to the course_id in the Courses table.
- grade: The grade assigned to the student for the specified course, stored as a string or numerical value.

Description:
The Grades table maintains records of grades assigned to students for their performance in various courses. Each grade record is uniquely identified by a grade_id. It tracks which student received which grade for a specific course. The student_id attribute references the ID of the student who received the grade, while the course_id attribute links to the course for which the grade was assigned. The grade field stores the actual grade assigned to the student, which could be represented as a letter grade (e.g., 'A', 'B+') or a numerical grade (e.g., 90, 85.5), depending on the grading system used by the school.

Normalization of tables up to 3-NF

Designing a normalized relational database involves organizing the data to minimize redundancy and dependency by dividing large tables into smaller ones and defining relationships between them. Here's a design for your school DBMS up to 3rd Normal Form (3NF):

Tables:

1) Departments Table (Departments)
- DepartmentID (Primary Key)
- DepartmentName

2) Teachers Table (Teachers)
- TeacherID (Primary Key)
- FirstName
- LastName
- DepartmentID (Foreign Key)

3) Students Table (Students)
- StudentID (Primary Key)
- FirstName
- LastName
- DepartmentID (Foreign Key)

4) Courses Table (Courses)
- CourseID (Primary Key)
- CourseName
- DepartmentID (Foreign Key)
- TeacherID (Foreign Key)

5) Enrollments Table (Enrollments)
- EnrollmentID (Primary Key)
- StudentID (Foreign Key)
- CourseID (Foreign Key)
- EnrollmentDate

6) Attendance Table (Attendance)

- AttendanceID (Primary Key)
- EnrollmentID (Foreign Key)
- Date
- Status (Present/Absent)

7) Grades Table (Grades)

- GradeID (Primary Key)
- EnrollmentID (Foreign Key)
- Marks
- Grade

Relationships:

❖ One-to-many relationship between Departments and Teachers:

- One department can have multiple teachers but each teacher belongs to one department.

❖ One-to-many relationship between Departments and Students:

- One department can have multiple students but each student belongs to one department.

❖ One-to-many relationship between Departments and Courses:

- One department can offer multiple courses but each course belongs to one department.

❖ One-to-many relationship between Teachers and Courses:

- One teacher can teach multiple courses but eac.h course is taught by one teacher.

❖ One-to-Many Relationship between Students and Enrollments:

- One student can enroll in multiple courses but each enrollment is for one student.

❖ One-to-Many Relationship between Courses and Enrollments:
- One course can have multiple enrollments but each enrollment is for one course.

❖ One-to-Many Relationship between Enrollments and Attendance:
- One enrollment can have multiple attendance records but each attendance record is for one enrollment.

❖ One-to-Many Relationship between Enrollments and Grades:
- One enrollment can have multiple grades (exams) but each grade is for one enrollment.

Normalization:

1) 1NF (First Normal Form):
- Each table has a primary key.
- All columns contain atomic (indivisible) values.
- Eliminate repeating groups or arrays.

2) 2NF (Second Normal Form):
- All requirements for 1NF are met.
- Eliminate partial dependencies of non-prime attributes on the primary key. (In this design, there aren't any partial dependencies.)

3) 3NF (Third Normal Form):
- All requirements for 2NF are met.
- Eliminate transitive dependencies of non-prime attributes on the primary key. (In this design, there aren't any transitive dependencies.)

This design should help maintain data integrity and minimize redundancy. You can further refine the design based on the specific requirements and constraints of your DBMS.

1NF Tables:

Departments Table:

DepartmentID	DepartmentName
1	Mathematics
2	Physics
3	Literature

Teachers Table:

TeacherID	FirstName	LastName	DepartmentID
1	John	Doe	1
2	Jane	Smith	2
3	Alice	Johnson	3

Students Table:

| StudentID	FirstName	LastName	DepartmentID
1	Mike	Brown	1
2	Emily	Dravis	2
3	Sarah	Wilson	3

Courses Table:

CourseID	CourseName	DepartmentID	TeacherID
1	Calculus	1	1
2	Quantum Mech	2	2
3	Poetry	3	3

Enrollments Table:

EnrollmentID	StudentID	CourseID	EnrollmentDate
1	1	1	2024-01-15
2	2	2	2024-01-15
3	3	3	2024-01-16

Attendance Table:

AttendanceID	| EnrollmentID	Date	Status
1	1	2024-02-01	Present
2	2	2024-02-01	Absent
3	3	2024-02-02	Present

Grades Table:

GradeID	EnrollmentID	Marks	Grade
1	1	85	B
2	2	90	A
3	3	88	A

2NF Tables: Remain the same as 1NF because there are no partial dependencies.
3NF Tables: Remain the same as 1NF because there are no transitive dependencies. These tables represent the database schema in the first, second, and third normal forms, ensuring data integrity and minimizing redundancy.

Creation of Data in the tables

Insert data into Departments table

```
Insert data into Departments table
INSERT INTO Departments (department_id, name, head_of_department_id)
VALUES
    (1, 'Mathematics', 2),
    (2, 'Science', 3),
    (3, 'History', NULL);
```

department_id	name	head_of_department_id
1	Mathematics	2
2	Science	3
3	History	NULL

Insert data into Teachers table

```
INSERT INTO Teachers (teacher_id, name, email, phone_number, department_id)
VALUES
    (1, 'John Doe', 'john.doe@gmail.com', '123456789', 1),
    (2, 'Jane Smith', 'jane.smith@gmail.com', '987654321', 1),
    (3, 'Michael Johnson', 'michael.@gmail.com', 456789123', 2);
```

teacher_id	name	email	Phone_number	Department_id
1	John Doe	john.doe@gmail.com	123456789	1
2	Jane Smith	jane.smith@gmail.com	987654321	1
3	Michael	michael@gmail.com	456789123	2

Insert data into Students table

```
INSERT INTO Students (student_id, name, email, phone_number, department_id)
VALUES
    (1, 'Alice Johnson', 'alice.johnson@gmail.com', '111222333', 1),
    (2, 'Bob Williams', 'bob.williams@gmail.com', '444555666', 2);
```

student_id	Name	email	phone_number	department_id
1	Alice Johnson	alice.johnson@gmail.com	111222333	1
2	Bob Williams	bob.williams@gmail.com	444555666	2

Insert data into Courses table

```
INSERT INTO Courses (course_id, name, teacher_id, department_id)
VALUES
    (1, 'Algebra', 1, 1),
    (2, 'Biology', 3, 2),
    (3, 'History', NULL, 3);
```

course_id	name	teacher_id	department_id
1	Algebra	1	1
2	Biology	3	2
3	History	NULL	3

Insert data into Enrollments table

```
INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date)
VALUES
    (1, 1, 1, '2024-04-01'),
    (2, 2, 2, '2024-04-02');
```

enrollment_id	student_id	course_id	enrolment_date
1	1	1	2024-04-01
2	2	2	2024-04-02

Inserting data into the Grades table

```
INSERT INTO Grades (student_id, course_id, grade)
VALUES
    (1, 1, 'A'),
    (2, 2, 'B+'),
    (1, 2, 'C');
```

student_id	course_id	grade
1	1	A
2	2	B+
1	2	C

Inserting data into the Attendance table

```
INSERT INTO Attendance (student_id, course_id, attendance_date, present)
VALUES
    (1, 1, '2024-04-01', true),
    (2, 2, '2024-04-02', true),
    (1, 2, '2024-04-02', false);
```

Student_id	Course_id	Attendance_date	present
1	1	2024-04-01	true
2	2	2024-04-02	True
1	2	2024-04-02	False

Few sql queries on the created tables

1. **SELECT Queries**:

Select all records from a table:

```
SELECT * FROM Departments;
SELECT * FROM Teachers;
SELECT * FROM Students;
SELECT * FROM Courses;
SELECT * FROM Enrollments;
SELECT * FROM ATTENDACE;
SELECT * FROM GRADES;
```

Select specific columns from a table:

```
SELECT department_id, name FROM Departments;
SELECT teacher_id, name, email FROM Teachers;
SELECT student_id, name, email FROM Students;
SELECT course_id, name FROM Courses;
SELECT enrollment_id, student_id, course_id FROM Enrollments;
```

Select records based on conditions:

```
SELECT * FROM Students WHERE department_id = 1;
SELECT * FROM Teachers WHERE name LIKE 'J%';
SELECT * FROM Courses WHERE department_id IS NULL;
```

2. **INSERT Queries**:

Insert a single record into a table:

```
INSERT INTO Departments (department_id, name) VALUES (4, 'Literature');
INSERT INTO Teachers (teacher_id, name, email, department_id) VALUES (4, 'Emily
Brown', 'emily.brown@gmail.com', 4);
```

Insert multiple records into a table:

```
INSERT INTO Students (student_id, name, email, department_id) VALUES
(3, 'Charlie Green', 'charlie.green@gmail.com', 1),
(4, 'Ella Davis', 'ella.davis@gmail.com', 2);
```

3. **UPDATE Queries**:

Update existing records:

```
UPDATE Teachers SET phone_number = '555666777' WHERE teacher_id = 2;
UPDATE Students SET department_id = 3 WHERE student_id = 1;
```

4. **DELETE Queries**:

Delete records from a table:

```
DELETE FROM Departments WHERE department_id = 4;
DELETE FROM Teachers WHERE teacher_id = 4;
```

5. **JOIN Queries**:

Inner Join:

```
SELECT Students.name, Departments.name
FROM Students
INNER JOIN Departments ON Students.department_id =Departments.department_id;
```

Left Join:

```
SELECT Courses.name, Teachers.name
FROM Courses
LEFT JOIN Teachers ON Courses.teacher_id = Teachers.teacher_id;
```

6. **Aggregate Functions**:

Count:

```
SELECT COUNT(*) FROM Students;
```

Average:

```
SELECT AVG(enrollment_date) FROM Enrollments;
```

Creation of views using the tables

View 1: List of Departments with Head of Department's Name:

```
CREATE VIEW Department_Head_View AS
SELECT d.department_id, d.name AS department_name, d.head_of_department_id,
t.name AS head_of_department_name
FROM Departments d
LEFT JOIN Teachers t ON d.head_of_department_id = t.teacher_id;
```

Department_Head_View:

department_id	department_name	head_of_department_id	head_of_department_name
1	Mathematics	2	Jane Smith
2	Science	3	Michael
3	History	NULL	NULL

View 2: List of Courses with Teachers' Names:

```
CREATE VIEW Course_Teacher_View AS
SELECT c.course_id, c.name AS course_name, c.teacher_id, t.name AS teacher_name
FROM Courses c
LEFT JOIN Teachers t ON c.teacher_id = t.teacher_id;
```

Course_Teacher_View:

Course_id	Course_name	Teacher_id	Teacher_name
1	Algebra	1	John Doe
2	Biology	3	Michael
3	History	NULL	NULL

View 3: List of Students with Department Names:

```
CREATE VIEW Student_Department_View AS
SELECT s.student_id, s.name AS student_name, s.department_id, d.name AS
department_name
FROM Students s
LEFT JOIN Departments d ON s.department_id = d.department_id;
```

Student_Department_view:

Student_id	Student_name	Department_id	Department_name
1	Alice Johnson	1	Mathematics
2	Bb Williams	2	Science

View 4: List of Enrollments with Student and Course Details:

```
CREATE VIEW Enrollment_Details_View AS
SELECT e.enrollment_id, e.student_id, s.name AS student_name, e.course_id, c.name
AS course_name, e.enrollment_date
FROM Enrollments e
JOIN Students s ON e.student_id = s.student_id
JOIN Courses c ON e.course_id = c.course_id;
```

Enrollment_Details_View:

Enrolment_id	Student_id	Student_name	Course_id	Course_name	Enrolment_date
1	1	Alice Johnson	1	Algebra	2024-04-01
2	2	Bob Williams	2	Biology	2024-04-02

View 5: List of Teachers with their Courses:

```
CREATE VIEW Teacher_Course_View AS
SELECT t.teacher_id, t.name AS teacher_name, c.course_id, c.name AS course_name
FROM Teachers t
JOIN Courses c ON t.teacher_id = c.teacher_id;
```

Teacher_Course_View:

Teacher_id	Teacher_name	Course_id	Course_name
1	John Doe	1	Algebra
3	Michael	2	Biology

THANK YOU