**SCHOOL DATABASE MANAGEMENT SYSTEM**

**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.

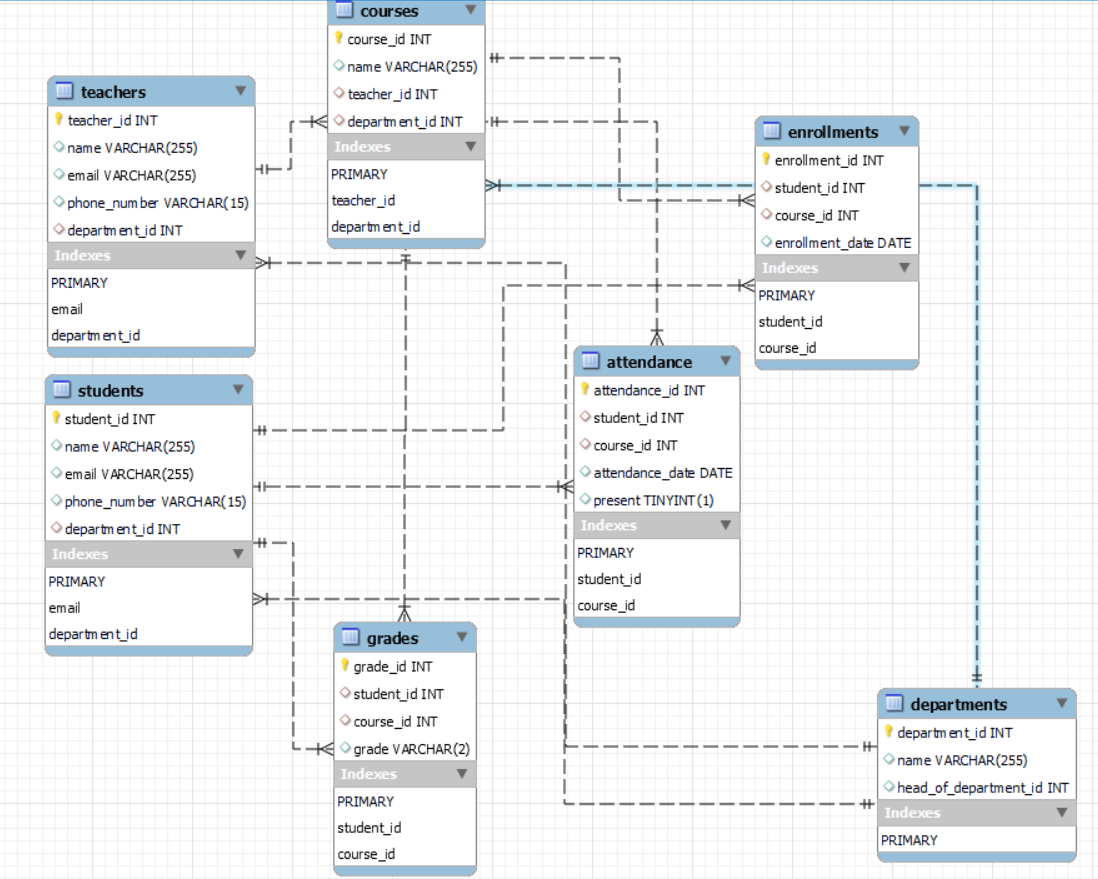
**Description of the Project**

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**

**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)

);

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.
* 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.

1. **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.
* 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.

1. **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.

1. **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.

1. **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.

1. **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.

1. **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

1. Teachers Table (Teachers)

* TeacherID (Primary Key)
* FirstName
* LastName
* DepartmentID (Foreign Key)

1. Students Table (Students)

* StudentID (Primary Key)
* FirstName
* LastName
* DepartmentID (Foreign Key)

1. Courses Table (Courses)

* CourseID (Primary Key)
* CourseName
* DepartmentID (Foreign Key)
* TeacherID (Foreign Key)

1. Enrollments Table (Enrollments)

* EnrollmentID (Primary Key)
* StudentID (Foreign Key)
* CourseID (Foreign Key)
* EnrollmentDate

1. Attendance Table (Attendance)

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

1. 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 each 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.

1. 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.)

1. 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.

**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 |

**Submitted by**

|  |  |
| --- | --- |
| **Name** | **Registration\_Number** |
| G.Harshit | AP22110010076 |
| P. Vishnu | AP22110010094 |
| V. Revanth | AP22110010109 |