MAAHI POONIA
maahi.poonia01@gmail.com
SQL Proficiency

A Student Records Management System

A Student Records Management System is a database-driven application designed to manage student information efficiently. It includes tasks like adding students, updating student details, tracking academic performance, and generating reports.

Creating the Database in MySQL or PostgreSQL:

Create the Database:

```
CREATE DATABASE StudentRecordsDB;
USE StudentRecordsDB;
```

Create the Tables:

Students Table:

Stores student information.

Structure:

Column Name	Data Type	Description
student_id	INT	Unique identifier for each student (Primary Key)
name	VARCHAR(255)	Full name of the student
email	VARCHAR(255)	Email address of the student
phone_number	VARCHAR(15)	Contact number
enrollment_date	DATE	Date when the student enrolled

Code:

```
CREATE TABLE Students (
    student_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255),
    phone_number VARCHAR(15),
    enrollment_date DATE DEFAULT (CURRENT_DATE)
);
```

Teachers Table:

Stores information about the teachers.

Structure:

Column Name	Data Type	Description
teacher_id	INT	Unique identifier for each teacher (Primary Key)
name	VARCHAR(255)	Full name of the teacher
email	VARCHAR(255)	Email Address of the teacher
hire_date	DATE	Date the teacher was hired.

Code:

```
CREATE TABLE Teachers (
    teacher_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255),
    hire_date DATE DEFAULT (CURRENT_DATE)
);
```

Courses Table:

Stores information about the courses offered.

Structure:

Column Name	Data Type	Description
course_id	INT	Unique identifier for each course (Primary Key)
course_name	VARCHAR(255)	Name of the course
teacher_id	INT	Foreign Key (references Teachers)

Code:

```
CREATE TABLE Courses (
    course_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    course_name VARCHAR(255) NOT NULL,
    teacher_id INT,
    FOREIGN KEY (teacher_id) REFERENCES Teachers(teacher_id)
);
```

Enrollments Table:

Manages student enrollments in courses.

Structure:

Column Name	Data Type	Description
enrolment_id	INT	Unique identifier for each enrolment(Primary Key)
student_id	INT	Foreign Key(references Courses)
course_id	INT	Foreign Key(references Courses)
enrolment_date	DATE	Date the student enrolled in the course

Code:

```
CREATE TABLE Enrollments (
    enrollment_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    student_id INT,
    course_id INT,
    enrollment_date DATE DEFAULT (CURRENT_DATE),
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```

Grades Table:

Stores the grades students receive in courses.

Structure:

Column Name	Data Type	Description
GRADE_ID	**INT**	Unique identifier for each grade (Primary Key)
STUDENT_ID	**INT**	Foreign Key (references Courses)
COURSE_ID	**INT**	Foreign Key (references Courses)
GRADE	**CHAR(2)**	Grade received by the student

Code:

```
CREATE TABLE Grades (
    grade_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    student_id INT,
    course_id INT,
    grade CHAR(2),
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```

Inserting Data:

Add some sample data to the tables.

Inserting into the Students Table:

```
-- Insert student records into the Students table
INSERT INTO Students (name, email, phone_number) VALUES
(' Liam Gentry', 'lima@example.com', '1234567890'),
(' Lyric Glass', 'lyric@example.com', '0987654321');
```

Inserting into the Teachers Table:

```
-- Insert teacher records into the Teachers table
INSERT INTO Teachers (name, email) VALUES
(' Aiden Nichols', 'alice@example.com'),
(' Beau Booth', 'beau@example.com');
```

Inserting into the Courses Table:

```
-- Insert course records into the Courses table, associating each course with a teacher
INSERT INTO Courses (course_name, teacher_id) VALUES
('Mathematics', 1),
('Physics', 2);
```

Inserting into the Enrollments Table:

```
-- Insert enrollment records for students into courses
INSERT INTO Enrollments (student_id, course_id) VALUES
(1, 1), -- Student 1 enrolls in Course 1
(2, 2); -- Student 2 enrolls in Course 2
```

Inserting into the Grades Table:

```
-- Insert grade records for students in their respective courses
INSERT INTO Grades (student_id, course_id, grade) VALUES
(1, 1, 'A'), -- Student 1 gets grade 'A' in Course 1
(2, 2, 'B'); -- Student 2 gets grade 'B' in Course 2
```

Basic Functionalities:

- Add new students, courses, and grades.

- Track which students are enrolled in which courses.
- Generate reports on student performance.

Writing Queries for Functionality:

Query-1: Add a New Student

```
-- Insert a new student record into the Students table with name, email, and phone number

INSERT INTO Students (name, email, phone_number) VALUES
-- Values for the new student: 'Julio Stokes', 'julio@example.com', '0987654321'
('Julio Stokes', 'julio@example.com', '0987654321');
```

Explanation:

This SQL statement inserts a new record into the Students table. The student's name is Julio Stokes, with the email julio@example.com and the phone number 0987654321. The INSERT INTO statement is used to add these values to the respective columns in the table.

Output:

```
Select * from Students;

student_id      name    email   phone_number    enrollment_date
1        Liam Gentry    lima@example.com        1234567890      2024-10-18
2        Lyric Glass    lyric@example.com       0987654321      2024-10-18
3       Julio Stokes    julio@example.com       0987654321      2024-10-18
```

Query-2: Enroll a Student in a Course

```
-- Insert a new enrollment record into the Enrollments table

INSERT INTO Enrollments (student_id, course_id) VALUES
-- Enroll student 1 in course 2
(1, 2);
```

Explanation:

This SQL statement enrolls the student with student_id = 1 into the course with course_id = 2. The INSERT INTO statement is used to add a new record to the Enrollments table, linking the student and the course by their respective IDs.

Output:

```
Select * from Enrollments;
enrollment_id   student_id      course_id       enrollment_date
1       1       1       2024-10-18
2       2       2       2024-10-18
3       1       2       2024-10-18
```

Query-3: Assign a Grade to a Student

```
INSERT INTO Grades (student_id, course_id, grade) VALUES
-- Assign grade 'A' to student 1 in course 2
(1, 2, 'A');
```

Explanation:

This SQL statement assigns a grade of 'A' to the student with student_id = 1 for the course with course_id = 2. The INSERT INTO statement adds a new record to the Grades table, linking the student's ID, the course ID, and the assigned grade.

Output:

```
Select * from Grades;
grade_id        student_id      course_id       grade
1       1       1       A
2       2       2       B
3       1       2       A
```

Query-4: List All Students Enrolled in a Course

```
-- Select student names and course name for students enrolled in course 1
SELECT s.name, c.course_name
-- From the Enrollments table, join with the Students table to get student information
FROM Enrollments e
JOIN Students s ON e.student_id = s.student_id
-- Join with the Courses table to get the course name
JOIN Courses c ON e.course_id = c.course_id
-- Filter to only show students enrolled in course 1
WHERE c.course_id = 1;
```

Explanation:

This SQL query retrieves a list of all students who are enrolled in course 1 along with the course name. It selects the student's name from the Students table and the course name from the Courses table. The Enrollments table acts as a link between the Students and Courses tables, and the query filters for records where the course_id is 1, meaning it only shows students enrolled in that specific course.

Output:

```
name     course_name
 Liam Gentry     Mathematics
```

Query-5: View Student's Grades

```sql
-- Select student name, course name, and grade for student 1
SELECT s.name, c.course_name, g.grade
-- From the Grades table, join with the Students table to get student information
FROM Grades g
JOIN Students s ON g.student_id = s.student_id
-- Join with the Courses table to get course names
JOIN Courses c ON g.course_id = c.course_id
-- Filter to show grades for student 1
WHERE s.student_id = 1;
```

Explanation:

This SQL query retrieves the grades of student 1 across all the courses they are enrolled in. It selects the student's name, the course name, and the corresponding grade from the Grades table. The Students and Courses tables are joined to provide the student's details and course names, and the query filters for records where the student_id is 1, showing only the grades for that specific student.

Output:

```
name     course_name      grade
 Liam Gentry     Mathematics      A
```

Query-6: Update a Student's Contact Information

Update the phone number of student 1.

```sql
-- Update the phone number of the student with student_id 1
UPDATE Students
-- Set the new phone number to '9876543210'
SET phone_number = '9876543210'
-- Apply the update to the student with student_id 1
WHERE student_id = 1;
```

Explanation:

This SQL statement updates the phone_number of the student with student_id = 1 to '9876543210'. The UPDATE statement modifies the existing record in the Students table where the student_id matches the specified value. It ensures that only the phone number for student 1 is changed.

Output:

```
Select * from Students;
student_id      name    email   phone_number    enrollment_date
1       Liam Gentry     lima@example.com        9876543210      2024-10-18
2       Lyric Glass     lyric@example.com       0987654321      2024-10-18
```

Query-7: Remove a Student from a Course

```sql
-- Delete the enrollment record for student 1 in course 2
DELETE FROM Enrollments
-- Specify the student_id and course_id to identify the record to delete
WHERE student_id = 1 AND course_id = 2;
```

Explanation:

This SQL statement deletes the enrollment record for the student with student_id = 1 in the course with course_id = 2. The DELETE FROM statement removes the matching record from the Enrollments table where both the student and course IDs match the specified values, effectively unenrolling the student from that course.

Output:

```
Select * from Enrollments;
enrollment_id   student_id      course_id       enrollment_date
1       1       1       2024-10-18
2       2       2       2024-10-18
```

Query-8: View All Courses Taught by a Specific Teacher

```sql
-- Select the names of courses taught by the teacher with teacher_id 2
SELECT course_name
-- From the Courses table
FROM Courses
-- Filter to show only the courses where the teacher_id is 2
WHERE teacher_id = 2;
```

Explanation:

This SQL query retrieves the names of all courses taught by the teacher with teacher_id = 2. The SELECT statement is used to get the course_name from the Courses table, and the query is filtered by teacher_id, so it only returns courses that are assigned to the specific teacher.

Output:

```
course_name
Physics
```

Query-9: Count the Number of Students in a Course

```sql
-- Count the total number of students enrolled in course 1
SELECT COUNT(*) AS total_students
-- From the Enrollments table
FROM Enrollments
-- Filter to count only students enrolled in course 1
WHERE course_id = 1;
```

Explanation:

This SQL query counts the total number of students enrolled in course 1. The COUNT(*) function returns the total number of records in the Enrollments table that match the condition course_id = 1. The result is labeled as total_students to show how many students are currently enrolled in that specific course.

Output:

```
total_students
1
```

Query-10: **List Students Who Have Not Yet Been Assigned a Grade**

```sql
-- Select the names of students who do not have a grade for course 2
SELECT s.name
-- From the Students table, using an alias 's'
FROM Students s
-- Perform a LEFT JOIN with the Grades table based on student_id and course_id = 2
LEFT JOIN Grades g ON s.student_id = g.student_id AND g.course_id = 2
-- Filter to include only those students where the grade is NULL
WHERE g.grade IS NULL;
```

Explanation:

This SQL query retrieves the names of students who are not enrolled in course 2 or who have not received a grade for that course. It uses a LEFT JOIN to combine the Students table with the Grades table based on student_id and filters for records where the grade is NULL. This means it lists students who either have no corresponding entry in the Grades table for course 2, indicating they haven't received a grade or are not enrolled in the course at all.

Output:

```
name
 Liam Gentry
```

Query-11: **Calculate the Average Grade for a Course**

```sql
SELECT AVG(
    CASE
        -- Assign a numeric value for each letter grade
        WHEN grade = 'A' THEN 4
        WHEN grade = 'B' THEN 3
        WHEN grade = 'C' THEN 2
        WHEN grade = 'D' THEN 1
        -- Assign 0 for any grade that is not A, B, C, or D
        ELSE 0
    END) AS avg_grade
-- From the Grades table
FROM Grades
-- Filter to include only grades for course_id 1
WHERE course_id = 1;
```

Explanation:

This SQL query calculates the average grade point for all students enrolled in course 1. It uses a CASE statement to convert letter grades into numerical values (A = 4, B = 3, C = 2, D = 1, and any other grade is considered as 0). The AVG() function then computes the average of these numerical values. The result is labeled as avg_grade and represents the overall performance of students in the specified course based on their grades.

Output:

```
avg_grade
4.0000
```

Query-12: Find the Highest Grade Assigned in a Course

```sql
-- Retrieve the highest grade for students enrolled in course 2
SELECT MAX(grade) AS highest_grade
-- From the Grades table
FROM Grades
-- Filter to include only grades for course_id 2
WHERE course_id = 2;
```

Explanation:

This SQL query retrieves the highest grade achieved by students in course 2. It uses the MAX() function to find the maximum value in the grade column from the Grades table, specifically filtering for records where course_id = 2. The result is labeled as highest_grade, indicating the top grade received in that particular course.

Output:

```
highest_grade
B
```

Query-13: List Students with the Same Grade in a Course

```sql
SELECT s.name, g.grade
-- From the Grades table, using an alias 'g'
FROM Grades g
-- Join with the Students table based on student_id to get student names
JOIN Students s ON g.student_id = s.student_id
-- Filter to include only grades for course_id 1
WHERE g.course_id = 1
-- Group the results by grade
GROUP BY g.grade
-- Include only those grades that are assigned to more than one student
HAVING COUNT(g.grade) > 1;
```

Explanation:

This SQL query identifies the names of students and their grades for course 1, but only for those grades that have been assigned to more than one student. It joins the Grades table with the Students table based on student_id. The GROUP BY clause groups the results by grade, and the HAVING clause filters the grouped results to only include grades that appear more than once. This helps to find common grades among students in that specific course.

Output:

```
Output not generated for insufficient data
```

Query-14: Assign a Teacher to a New Course

```
INSERT INTO Courses (course_name, teacher_id)
-- Specify the values for the course name and associated teacher ID
VALUES ('Chemistry', 3);
```

Explanation:

This SQL code inserts a new record into the Courses table. It specifies that a course named "Chemistry" is being added, and it associates this course with a teacher whose ID is 3. The INSERT INTO statement is used to add new rows to a table, and the VALUES clause provides the data to be inserted into the respective columns of the table.

Output:

```
Select * from Courses;
course_id       course_name     teacher_id
1       Mathematics     1
2       Physics 2
3       Chemistry       3
```

Query-15: Find the Total Number of Courses Each Student is Enrolled In

```
SELECT s.name, COUNT(e.course_id) AS total_courses
-- From the Enrollments table, using an alias 'e'
FROM Enrollments e
-- Join with the Students table based on student_id to link enrollments with student names
JOIN Students s ON e.student_id = s.student_id
-- Group the results by student name to get total courses for each student
GROUP BY s.name;
```

Explanation:

This SQL query retrieves the names of students along with the total number of courses they are enrolled in. It joins the Enrollments table with the Students table based on the student_id to associate each enrollment with the corresponding student. The COUNT(e.course_id) function counts the number of courses for each student, and

the results are grouped by the student's name using the GROUP BY clause. This allows for a summary of course enrollments per student.

Output:

```
name    total_courses
 Liam Gentry    1
 Lyric Glass    1
```

Query-16: List All Courses a Student is Enrolled In

```sql
SELECT c.course_name
-- From the Enrollments table, using an alias 'e'
FROM Enrollments e
-- Join with the Courses table based on course_id to link enrollments with course names
JOIN Courses c ON e.course_id = c.course_id
-- Filter the results to include only the enrollments of the specified student
WHERE e.student_id = 1;
```

Explanation:

This SQL query retrieves the names of courses in which a specific student, identified by student_id = 1, is enrolled. It does so by joining the Enrollments table with the Courses table on the course_id. The join operation connects each enrollment record with its corresponding course record, allowing the query to select the course names associated with the specified student. The WHERE clause filters the results to include only the courses for the specified student.

Output:

```
course_name
Mathematics
```

Query-17: Delete a Course and All Related Data

```sql
-- Delete the course with course_id = 2 from the Courses table
DELETE FROM Courses WHERE course_id = 2;

-- Delete all enrollments associated with the course_id = 2 from the Enrollments table
DELETE FROM Enrollments WHERE course_id = 2;

-- Delete all grades associated with the course_id = 2 from the Grades table
DELETE FROM Grades WHERE course_id = 2;
```

Explanation:

The provided SQL code consists of three statements that delete data related to a specific course identified by course_id = 2. The first statement removes the course from the Courses table. The second statement deletes any enrollment records associated with that course from the Enrollments table. Finally, the third statement eliminates any grades recorded for that course from the Grades table. This cascading deletion ensures that all references to the course are removed from the database, preventing orphaned records in the related tables.

Output:

```
Select * from Courses;
Select * from Enrollments;
Select * from Grades;
course_id       course_name     teacher_id
1       Mathematics     1
enrollment_id   student_id      course_id       enrollment_date
1       1       1       2024-10-18
grade_id        student_id      course_id       grade
1       1       1       A
```