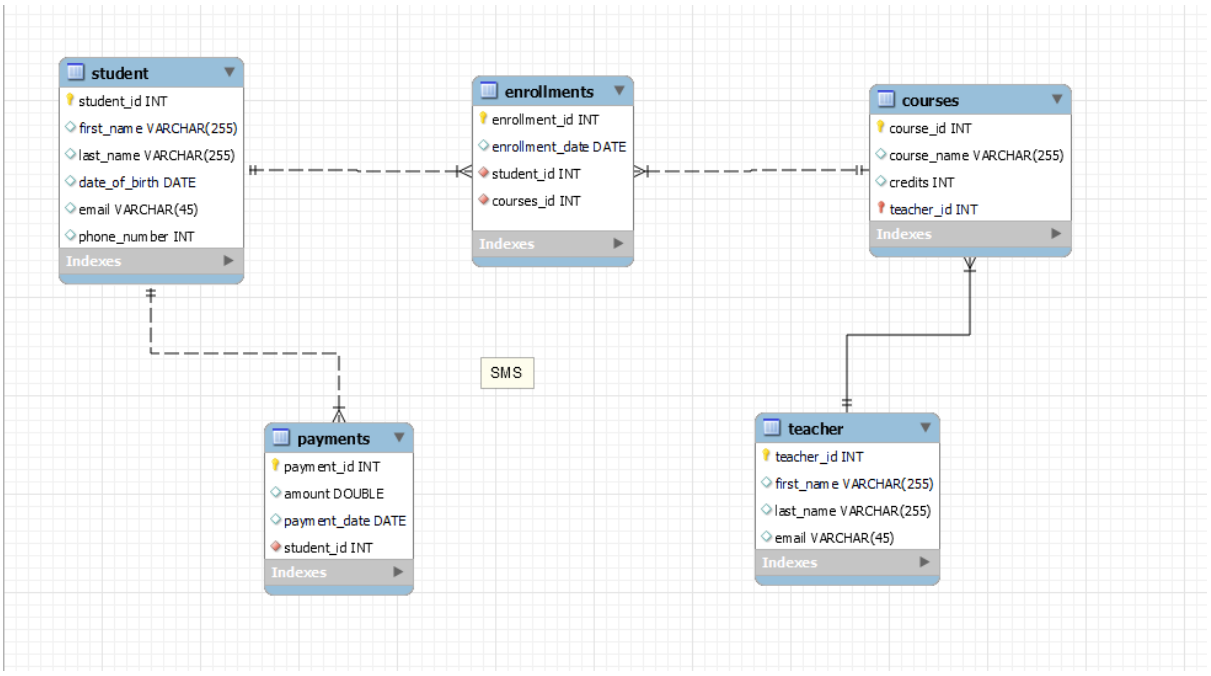
Student Management System



```
CREATE DATABASE SMS;

Use sms;


CREATE TABLE Students (

  student_id INT PRIMARY KEY AUTO_INCREMENT,

  first_name VARCHAR(50) NOT NULL,

  last_name VARCHAR(50) NOT NULL,

  date_of_birth DATE NOT NULL,

  email VARCHAR(100) UNIQUE NOT NULL,

  phone_number VARCHAR(20)

);


CREATE TABLE Courses (

  course_id INT PRIMARY KEY AUTO_INCREMENT,

  course_name VARCHAR(100) NOT NULL,

  credits INT NOT NULL,

  teacher_id INT NOT NULL,
  FOREIGN KEY (teacher_id) REFERENCES Teacher(teacher_id)
);

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

CREATE TABLE Teacher (
  teacher_id INT PRIMARY KEY AUTO_INCREMENT,
  first_name VARCHAR(50) NOT NULL,
  last_name VARCHAR(50) NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL
);

CREATE TABLE Payments (
  payment_id INT PRIMARY KEY AUTO_INCREMENT,
  student_id INT NOT NULL,
  amount DECIMAL(10,2) NOT NULL,
  payment_date DATE NOT NULL,
  FOREIGN KEY (student_id) REFERENCES Students(student_id)
);

INSERT INTO Students (first_name, last_name, date_of_birth, email, phone_number)
VALUES
```

```
  ('Arthur', 'Morgan', '2003-01-12', 'arthur.morgan@rdr2.com', '555-123-4567'),
  ('John', 'Marston', '2001-04-21', 'john.marston@rdr2.com', '555-789-0123'),
  ('Dutch', 'van der Linde', '2002-09-07', 'dutch.linde@rdr2.com', '555-456-7890'),
  ('Micah', 'Bell', '2000-11-05', 'micah.bell@rdr2.com', '555-246-8135'),
  ('Sadie', 'Adler', '2004-05-28', 'sadie.adler@rdr2.com', '555-987-6543');

INSERT INTO Teacher (first_name, last_name, email)
VALUES
  ('Jill', 'Valentine', 'jill.valentine@re.com'),
  ('Chris', 'Redfield', 'chris.redfield@re.com'),
  ('Leon', 'S. Kennedy', 'leon.kennedy@re.com'),
  ('Claire', 'Redfield', 'claire.redfield@re.com'),
  ('Ada', 'Wong', 'ada.wong@re.com');

INSERT INTO Courses (course_name, credits, teacher_id)
VALUES
  ('Introduction to Programming', 3, 1),
  ('Data Structures and Algorithms', 3, 2),
  ('Computer Architecture and Organization', 3, 3),
  ('Database Management Systems', 3, 4),
  ('Software Engineering', 3, 5);

INSERT INTO Enrollments (student_id, course_id, enrollment_date)
VALUES
  (1, 2, '2012-02-14'),
  (2, 4, '2011-10-26'),
  (3, 1, '2010-09-01'),
  (4, 3, '2013-01-07'),
  (5, 5, '2012-05-15');
```

```
INSERT INTO Payments (student_id, amount, payment_date)

VALUES

  (1, 250.75, '2018-03-12'),

  (2, 178.50, '2017-07-19'),

  (3, 421.00, '2016-02-05'),

  (4, 399.99, '2019-11-23'),

  (5, 600.00, '2015-12-24');


  -- TASK 2 --

  /*1. Write an SQL query to insert a new student into the "Students" table with the
following details:

 a. First Name: John

 b. Last Name: Doe

 c. Date of Birth: 1995-08-15

 d. Email: john.doe@example.com

 e. Phone Number: 1234567890*/

INSERT INTO Students (first_name, last_name, date_of_birth, email, phone_number)

VALUES ('John', 'Doe', '1995-08-15', 'john.doe@example.com', '1234567890');


-- 2. Write an SQL query to enroll a student in a course. Choose an existing student and
course and insert a record into the "Enrollments" table with the enrollment date.

INSERT INTO Enrollments (student_id, course_id, enrollment_date)

VALUES (1, 3, '2024-03-08');


-- 3. Update the email address of a specific teacher in the "Teacher" table. Choose any
teacher and modify their email address.

UPDATE Teacher

SET email = 'jill.valentine.new@re.com'

WHERE teacher_id = 1;
```

-- 4. Write an SQL query to delete a specific enrollment record from the "Enrollments" table. Select an enrollment record based on the student and course.

```
DELETE FROM Enrollments

WHERE student_id = 1 AND course_id = 3;
```

-- 5. Update the "Courses" table to assign a specific teacher to a course. Choose any course and teacher from the respective tables.

```
UPDATE Courses

SET teacher_id = 3

WHERE course_id = 2;
```

-- 6. Delete a specific student from the "Students" table and remove all their enrollment records from the "Enrollments" table. Be sure to maintain referential integrity.

```
DELETE FROM Enrollments

WHERE student_id = 1;

DELETE FROM Students

WHERE student_id = 1;
```

-- 7. Update the payment amount for a specific payment record in the "Payments" table. Choose any payment record and modify the payment amount.

```
UPDATE Payments

SET amount = 500.00

WHERE payment_id = 1;
```

-- TASK 3 --

-- 1. Write an SQL query to calculate the total payments made by a specific student. You will need to join the "Payments" table with the "Students" table based on the student's ID.

```
SELECT S.first_name, S.last_name, SUM(P.amount) AS total_payment

FROM Students s

JOIN Payments p ON S.student_id = P.student_id
```

```
GROUP BY S.student_id, S.first_name, S.last_name;
```

-- 2. Write an SQL query to retrieve a list of courses along with the count of students enrolled in each course. Use a JOIN operation between the "Courses" table and the "Enrollments" table.

```
SELECT C.course_name, COUNT(E.student_id) AS student_count

FROM Courses c

JOIN Enrollments e ON C.course_id = E.course_id

GROUP BY C.course_name, C.course_id

ORDER BY student_count DESC;
```

-- 3. Write an SQL query to find the names of students who have not enrolled in any course. Use a LEFT JOIN between the "Students" table and the "Enrollments" table to identify students without enrollments.

```
SELECT S.first_name, s.last_name

FROM Students s

LEFT JOIN Enrollments e ON s.student_id = E.student_id

WHERE E.student_id IS NULL;
```

-- 4. Write an SQL query to retrieve the first name, last name of students, and the names of the courses they are enrolled in. Use JOIN operations between the "Students" table and the "Enrollments" and "Courses" tables.

```
SELECT S.first_name, s.last_name, C.course_name

FROM Students s

JOIN Enrollments e ON s.student_id = e.student_id

JOIN Courses c ON c.course_id = e.course_id;
```

-- 5. Create a query to list the names of teachers and the courses they are assigned to. Join the "Teacher" table with the "Courses" table

```
SELECT t.first_name,t.last_name, c.course_name

FROM Teacher t

JOIN Courses c ON t.teacher_id = c.teacher_id;
```

-- 6. Retrieve a list of students and their enrollment dates for a specific course. You'll need to join the "Students" table with the "Enrollments" and "Courses" tables.

```
SELECT s.first_name, s.last_name, e.enrollment_date

FROM Students s

JOIN Enrollments e ON s.student_id = e.student_id

JOIN Courses c ON c.course_id = e.course_id

WHERE c.course_name = 'Software engineering';
```

-- 7. Find the names of students who have not made any payments. Use a LEFT JOIN between the "Students" table and the "Payments" table and filter for students with NULL payment records.

```
SELECT s.first_name,s.last_name

FROM Students s

LEFT JOIN Payments p ON s.student_id = p.student_id

WHERE p.payment_id IS NULL;
```

-- 8. Write a query to identify courses that have no enrollments. You'll need to use a LEFT JOIN between the "Courses" table and the "Enrollments" table and filter for courses with NULL enrollment records.

```
SELECT c.course_name

FROM Courses c

LEFT JOIN Enrollments e ON c.course_id = e.course_id

WHERE e.student_id IS NULL;
```

-- 9. Identify students who are enrolled in more than one course. Use a self-join on the "Enrollments" table to find students with multiple enrollment records.

```
SELECT s.first_name, s.last_name

FROM Students s

JOIN Enrollments AS e1 ON s.student_id = e1.student_id

JOIN Enrollments AS e2 ON s.student_id = e2.student_id
```

```
WHERE e1.course_id != e2.course_id

GROUP BY s.student_id, S.first_name, S.last_name

HAVING COUNT(DISTINCT e1.course_id) > 1;


-- 10. Find teachers who are not assigned to any courses. Use a LEFT JOIN between the
"Teacher" table and the "Courses" table and filter for teachers with NULL course
assignments.

SELECT t.first_name, t.last_name

FROM Teacher t

LEFT JOIN Courses c ON t.teacher_id = c.teacher_id

WHERE c.course_id IS NULL;


-- TASK 4 : SUBQUERY AND ITS TYPES

-- 1. Write an SQL query to calculate the average number of students enrolled in each
course. Use aggregate functions and subqueries to achieve this.

SELECT c.course_name, AVG(student_count) AS average_enrollment

FROM Courses c

LEFT JOIN (

  SELECT course_id, COUNT(*) AS student_count

  FROM Enrollments

  GROUP BY course_id

) AS enrolled_students ON c.course_id = enrolled_students.course_id

GROUP BY c.course_name;


-- 2. Identify the student(s) who made the highest payment. Use a subquery to find the
maximum payment amount and then retrieve the student(s) associated with that amount.

SELECT s.first_name, s.last_name

FROM Students s

JOIN Payments p ON s.student_id = p.student_id

WHERE p.amount = (

  SELECT MAX(amount)
  FROM Payments
);

-- 3. Retrieve a list of courses with the highest number of enrollments. Use subqueries to
find the course(s) with the maximum enrollment count.


-- 4 Calculate the total payments made to courses taught by each teacher. Use subqueries to
sum payments for each teacher's courses.

SELECT t.first_name, t.last_name, SUM(p.amount) AS total_payments

FROM Teacher t

LEFT JOIN Courses AS c ON t.teacher_id = c.teacher_id

LEFT JOIN Enrollments AS e ON c.course_id = e.course_id

LEFT JOIN Payments AS p ON e.student_id = p.student_id

GROUP BY t.teacher_id, t.first_name, t.last_name;


-- 5. Identify students who are enrolled in all available courses. Use subqueries to compare a
student's enrollments with the total number of courses.

-- 6 Retrieve the names of teachers who have not been assigned to any courses. Use
subqueries to find teachers with no course assignments.

SELECT t.first_name, t.last_name

FROM Teacher AS t

LEFT JOIN Courses AS c ON t.teacher_id = c.teacher_id

WHERE c.course_id IS NULL;


-- 7. Calculate the average age of all students. Use subqueries to calculate the age of each
student based on their date of birth.

SELECT AVG(YEAR(CURDATE()) - YEAR(date_of_birth)) AS average_age

FROM Students;
```

-- 8. Identify courses with no enrollments. Use subqueries to find courses without enrollment records.

```
SELECT c.course_name

FROM Courses AS c

LEFT JOIN Enrollments AS e ON c.course_id = e.course_id

WHERE e.student_id IS NULL;
```

-- 9

-- 10. Identify students who have made more than one payment. Use subqueries and aggregate functions to count payments per student and filter for those with counts greater than one.

```
SELECT s.first_name, s.last_name

FROM Students AS s

WHERE (SELECT COUNT(*) FROM Payments AS p2

    WHERE p2.student_id = s.student_id) > 1;
```

-- 12. Retrieve a list of course names along with the count of students enrolled in each course. Use JOIN operations between the "Courses" table and the "Enrollments" table and GROUP BY to count enrollments.

```
SELECT c.course_name, COUNT(e.student_id) AS student_count

FROM Courses c

JOIN Enrollments e ON c.course_id = e.course_id

GROUP BY c.course_name, c.course_id

ORDER BY student_count DESC;
```

-- 13. Calculate the average payment amount made by students. Use JOIN operations between the "Students" table and the "Payments" table and GROUP BY to calculate the average.

```
SELECT AVG(p.amount) AS average_payment

FROM Students s

JOIN Payments p ON s.student_id = p.student_id;
```