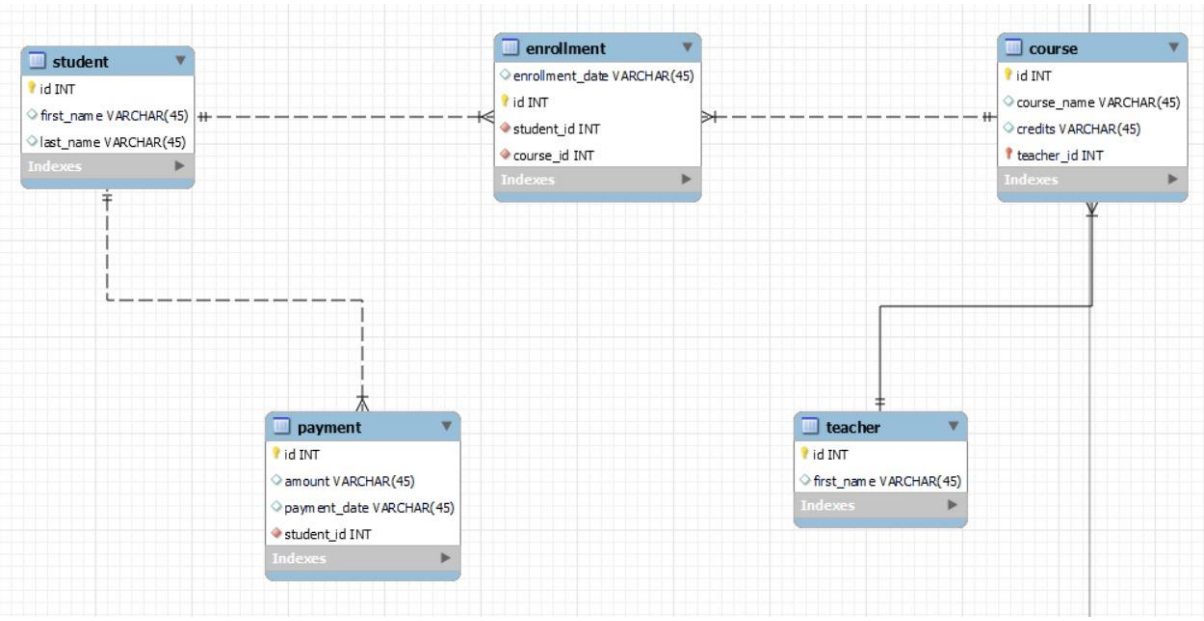
Student information Management



use SIS;

-- student

INSERT INTO student VALUES

(1, 'Aarav', 'A', '2002-03-14', 'aarav@example.com', '7548800902'),

(2, 'Aditi', 'A', '2002-10-12', 'aditi@example.com', '7548800902'),

(3, 'Divya', 'D', '2003-05-15', 'divya@example.com', '9876543210'),

(4, 'Ganesh', 'G', '2002-08-20', 'ganesh@example.com', '8765432109'),

(5, 'Isha', 'I', '2001-11-25', 'isha@example.com', '7654321098'),

(6, 'Karthik', 'K', '2003-02-10', 'karthik@example.com', '6543210987'),

(7, 'Lakshmi', 'L', '2002-04-30', 'lakshmi@example.com', '5432109876'),

(8, 'Meera', 'M', '2001-07-05', 'meera@example.com', '4321098765'),

(9, 'Neha', 'N', '2000-09-12', 'neha@example.com', '3210987654'),

(10, 'Raj', 'R', '2004-01-18', 'raj@example.com', '2109876543');

SELECT * FROM student;

-- courses

Student information Management

```
INSERT INTO courses VALUES
(101, 'Python', 5, 21),
(102, 'C++', 4, 22),
(103, 'Data Science', 1, 23),
(104, 'Web Development', 4, 24),
(105, 'Database Management', 3, 25),
(106, 'Machine Learning', 5, 26),
(107, 'Network Security', 2, 27),
(108, 'Mobile App Development', 4, 28),
(109, 'Artificial Intelligence', 5, 29),
(110, 'Software Engineering', 3, 30),
(111, 'JAVA', 2, 26),
(112, 'FSD', 2, 22),
(113, 'Block chain',3,24);

update courses set teacher_id=24 where teacher_id=26;
select * from courses;

-- enrollments

INSERT INTO enrollments VALUES
(001,2,108,'2024-02-14'),
(002,4,104,'2024-02-15'),
(003,6,107,'2024-02-14'),
(004,8,109,'2024-02-13'),
(005,10,108,'2024-02-12'),
(006,1,104,'2024-02-16'),
```

Student information Management

```
(007,3,102,'2024-02-14'),

(008,5,103,'2024-02-15'),

(009,7,105,'2024-02-13'),

(010,9,102,'2024-02-12'),

(011,9,104,'2024-02-12'),

(012,5,102,'2024-02-12'),

(013,5,101,'2024-02-12'),

(014,5,103,'2024-02-12'),

(015,5,106,'2024-02-12'),

(016,5,107,'2024-02-12'),

(017,5,108,'2024-02-12'),

(018,5,109,'2024-02-12'),

(019,5,110,'2024-02-12'),

(020,5,111,'2024-02-12'),

(021,5,112,'2024-02-12'),

(022,5,104,'2024-02-12'),

(023,5,105,'2024-02-12');


-- delete from enrollments where student_id=3;


SELECT * FROM enrollments;


-- teachers

INSERT INTO teachers VALUES

(21, 'John', 'S', 'john@gmail.com'),

(22, 'Alice', 'J', 'alice@gmail.com'),

(23, 'Michael', 'B', 'michaeln@gmail.com'),

(24, 'Emily', 'D', 'emily@gmail.com'),
```

Student information Management

(25, 'Daniel', 'T', 'daniel@gmail.com'),

(26, 'Olivia', 'A', 'olivia@gmail.com'),

(27, 'William', 'Mr', 'william@gmail.com'),

(28, 'Sophia', 'T', 'sophia@gmail.com'),

(29, 'David', 'W', 'david@gmail.com'),

(30, 'Ava', 'M', 'ava@gmail.com');

select * from teachers;

-- payments

INSERT INTO payments VALUES

(011,2,50000,'2023-02-20'),

(012,4,40000,'2023-02-21'),

(013,6,30000,'2023-02-22'),

(014,8,50000,'2023-02-23'),

(015,10,50000,'2023-02-24'),

(016,1,60000,'2023-02-25'),

(017,3,40000, '2023-02-20'),

(018,5,70000,'2023-02-22'),

(019,7,20000,'2023-02-22'),

(020,9,10000,'2023-02-20'),

(021,9,10000,'2023-02-20');

select * from payments;

-- task 2.1 - Write an SQL query to insert a new student into the "Students" table with the following details

Student information Management

```
INSERT INTO student VALUES

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


-- task 2.2 - 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 VALUES

(101,3,105,'2024-02-15');


-- task2.3 - Update the email address of a specific teacher in the "Teacher" table.

-- Choose any teacher and modify their email address.


UPDATE teachers set email='ali@gmail.com' where teacher_id=22;


-- task 2.4 - 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=3 and course_id=102;


-- task 2.5 -  Update the "Courses" table to assign a specific teacher to a course.

-- Choose any course and teacher from the respective tables.


update courses c

set c.teacher_id=23

where c.course_name='c++';
```

Student information Management

-- task 2.6 - Update the "Courses" table to assign a specific teacher to a course.

-- Choose any course and teacher from the respective tables.

```
delete from enrollments where student_id=6;

delete from payments where student_id=6;

delete from student where student_id=6;
```

-- task 2.7 - Update the payment amount for a specific payment record in the "Payments" table.

-- Choose any payment record and modify the payment amount

```
update payments p

set p.amount=25000

where p.amount=10000;
```

-- 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.student_id,sum(p.amount)

from student s

join payments p on s.student_id=p.student_id

where s.first_name='Aditi' group by s.student_id;
```

-- task 3.2 - 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)

from enrollments e
```

```
join courses c on e.course_id=c.course_id

group by e.course_id;
```

-- task 3.3 - 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

from student s

left join enrollments e on s.student_id=e.student_id

where e.student_id is null;
```

-- task 3.4 - 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 student s

join enrollments e on s.student_id=e.student_id

join courses c on e.course_id=c.course_id;
```

-- task 3.5 - 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.teacher_id,c.course_name

from teachers t

join courses c on t.teacher_id=c.teacher_id;
```

-- task 3.6 - 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,e.enrollment_date

from student s

join enrollments e on s.student_id=e.student_id
```

```
where e.course_id=104;


-- task3.7 - 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

from student s

left join payments p on s.student_id=p.student_id

where p.amount is null;


-- task 3.8 -  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_id,c.course_name,c.teacher_id

FROM courses c

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

WHERE e.course_id IS NULL;


-- task 3.9 - 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 distinct e.student_id

from enrollments e

join enrollments es on e.student_id = es.student_id

where e.course_id != es.course_id;


-- task 3.10 - 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.teacher_id

from teachers t
```

```
left join courses c on t.teacher_id=c.teacher_id

where c.teacher_id is null;


-- task 4.1 - Write an SQL query to calculate the average number of students enrolled in each course. Use aggregate functions and subqueries to achieve this.

select cname, avg(nos)

from (select c.course_name 'cname', count(e.student_id) as 'nos'

        from enrollments e

        join courses c on c.course_id = e.course_id

    group by c.course_name)

as subquery group by cname;


-- task 4.2 - 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,p.student_id

from student s

join payments p on s.student_id=p.student_id where amount = (select max(amount) from payments);


-- task 4.3 - Retrieve a list of courses with the highest number of enrollments. Use subqueries to find the course(s) with the maximum enrollment count.

select course_id,max(cenroll) as 'noeroll'

from (select count(course_id) as 'cenroll' ,course_id

        from enrollments

    group by course_id)

as subquery group by course_id;


-- task 4.4 - Calculate the total payments made to courses taught by each teacher. Use subqueries to sum payments for each teacher's courses.
```

Student information Management

```
select c.teacher_id,c.course_name,(select sum(p.amount)
                                                    from payments p
                    where p.student_id=e.student_id) as tot
from enrollments e
join courses c on e.course_id=c.course_id;
```

-- task 4.5 - Identify students who are enrolled in all available courses. Use subqueries to compare a student's enrollments with the total number of courses.

```
select student_id
from enrollments
group by student_id having count(distinct course_id ) =( select count(distinct course_id) from courses);
```

-- task 4.6 - Retrieve the names of teachers who have not been assigned to any courses. Use subqueries to find teachers with no course assignments.

```
select t.teacher_id,t.first_name
from teachers t
where not exists (select c.teacher_id
                                    from courses c
          where c.teacher_id=t.teacher_id);
```

-- task 4.7 - Calculate the average age of all students. Use subqueries to calculate the age of each student based on their date of birth.

```
select student_id,concat(first_name,' ',last_name) as 'Student Name',
timestampdiff(year,date_of_birth,current_date()) 'age',
(select avg(timestampdiff(year,date_of_birth,current_date())) from student) as average_age
from student ;
```

Student information Management

-- task 4.8 - Identify courses with no enrollments. Use subqueries to find courses without enrollment records.

```
select c.course_name

from courses c

where not exists (select e.enrollment_id

                  from enrollments e

        where c.course_id=e.course_id);
```

-- task 4.9 - Calculate the total payments made by each student for each course they are enrolled in. Use subqueries and aggregate functions to sum payments.

```
-- select s.first_name,sum(p.amount),e.course_id from student s join enrollments e on s.student_id=e.student_id join payments p on e.student_id=p.student_id group by e.course_id;

-- select s.first_name,e.course_id,(select sum(p.amount) from payments p from student s join enrollments e on s.student_id=e.student_id join payments p on e.student_id=p.student_id group by e.course_id,s.first_name;

select e.student_id,e.course_id,(select sum(p.amount)

                                 from payments as p

            where p.student_id = e.student_id) as total_payments

from enrollments as e

group by e.student_id, e.course_id;
```

-- task 4.10 - 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 distinct student_id

from payments

where (select count(p.student_id)

        from payments p
```

```
where p.student_id=payments.student_id )>1;
```

-- task4.11 - Write an SQL query to calculate the total payments made by each student. Join the "Students"table with the "Payments" table and use GROUP BY to calculate the sum of payments for each student.

```
select s.student_id,sum(amount) as total_payments
from student s
join payments p on s.student_id=p.student_id
group by s.student_id;
```

-- task4.12 - Retrieve a list of course names along with the count of students enrolled in each course. UseJOIN operations between the "Courses" table and the "Enrollments" table and GROUP BY to count enrollments.

```
select c.course_name,count(e.student_id)
from courses c
join enrollments e on c.course_id=e.course_id
group by c.course_name;
```

-- task4.13 - 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 s.first_name,s.student_id,avg(p.amount)
from payments p
join student s on p.student_id = s.student_id
group by s.student_id;
```

```
drop table student;
drop table courses;
drop table enrollments;
drop table teachers;
```

Student information Management

```
drop table payments;
```