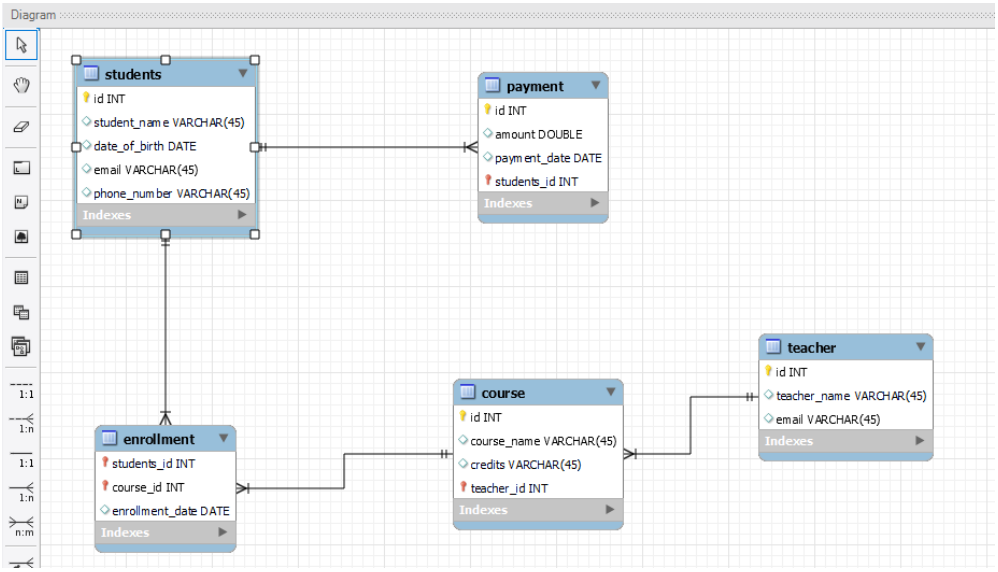
STUDENT INFORMATION SYSTEM (SIS)

ER DIAGRAM:



INSERTIONS:

insert into students (student_name, date_of_birth, email, phone_number) values

('Harsha','1999-02-26','harshakrish@gmail.com','9812345670'),

('Arohi','2000-04-09','arohiharsha@gmail.com','89124537650'),

('Ashok','1999-05-31','ashokshaka@gmail.com','8712345690'),

('Niha','2000-01-31','nihaashok@g23mail.com','9012873456'),

('Marshal','1999-10-31','marshaldev@gmail.com','8097612345'),

('Meghna','2000-09-28','megnamarsh@gmail.com','9123456790');

---

insert into teacher (teacher_name, email) values

('Aadharva','aadharvadev@gmail.com'),

('Vaishu','vaishuadev@gmail.com'),

('Rudhra','rudhrakowshik@gmail.com'),

('Apoorva','apoorvaahh@gmail.com'),

('Vijay','vijayvikranth@gmail.com'),

('Avanthika','avanthikaaaa@gmail.com');

---

insert into course (id,course_name, credits, teacher_id) values

('1','c programming','3',5),

('2','JAVA programming','4',6),

('3','DBMS','2',4),

('4','Networking','2',1),

('5','Python','3',2),

('9','CNS','2',14),

('10','DAA','4',16);

---

insert into payment (amount, payment_date, students_id) values

('10000','2024-02-23',4),

('23000','2023-11-19',1),

('19000','2024-01-06',9),

('47500','2023-07-23',10),

('50000','2022-12-14',5);

---

insert into enrollment (students_id, course_id, enrollment_date) values

(1,2,'2022-03-04'),

(1,6,'2023-06-23'),

(2,3,'2021-05-09'),

(3,6,'2023-06-22'),

(4,8,'2020-11-11'),

(6,4,'2022-03-04'),

(7,2,'2022-03-04');

Tasks 2: Select, Where, Between, AND, LIKE:

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:

d. Email: john.doe@example.com

e. Phone Number: 1234567890

```
insert into students (student_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 enrollment (students_id, course_id, enrollment_date) values
(9,5,'2023-06-02');
```

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= 'apoorvarudh@gmail.com'
where id=4;
```

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 enrollment
where students_id = 1 and course_id = 6;
```

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

```
update course
set teacher_id = 2
where course_name='Python';
```

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.

→ First delete the student 11 from enrollment txable

```
delete from enrollment
where students_id=11;
```

→Then get delete with student table

```
delete from students
where id=11;
```

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

```
update payment
set amount='23500' , payment_date='2023-11-20'
where id=2;
```

Task 3. Aggregate functions, Having, Order By, GroupBy and Joins:

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 sum(amount)
from payment
where students_id=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.students_id)
from course c
JOIN enrollment e ON c.id=e.course_id
group by c.course_name;
```

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.student_name
FROM students s
LEFT JOIN enrollment e ON S.id = e.students_id
WHERE e.students_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.student_name,c.course_name
from students s
JOIN enrollment e ON s.id = e.students_id
JOIN course c ON e.course_id=c.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.teacher_name,c.course_name
from teacher t
JOIN course c ON t.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.student_name,c.course_name,e.enrollment_date
from students s
JOIN enrollment e ON s.id=e.students_id
JOIN course c ON e.course_id=c.id;
```

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.student_name
```

FROM students s

LEFT JOIN payment p ON s.id = p.students_id

WHERE p.students_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 course c

JOIN enrollment e ON c.id=e.course_id

where e.course_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 students_id, count(course_id) as num_courses_enrolled

from enrollment

group by students_id

having count(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.id, t.teacher_name

from teacher t

left join course c on t.id = c.teacher_id

where c.teacher_id is null;

Task 4. Subquery and its type:

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 avg(students_number) as average_students_per_course

from (

```
    select course_id, count(students_id) as students_number
    from enrollment
    group by course_id );
```

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 id, student_name
from students
where id in (
    select students_id
    from payment
    where amount = (
        select max(amount)
        from payment ) );
```

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.

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 id, student_name
from students
where (
    select count(distinct course_id)
    from enrollment
) = (
    select count(distinct course_id)
    from enrollment
```

```
   where enrollment.students_id = students.id
);
```

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

```
select teacher_name
from teacher
where id NOT IN (select teacher_id from course);
```

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(age) as average_age
from (
   select timestampdiff(year, date_of_birth, curdate()) as age
   from students
) as student_ages; --> referred from internet to have knowledge about it.
```

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

```
select id, course_name
from course
where id not in (
   select distinct course_id
   from enrollment
);
```

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 student_id,course_id,sum(p.amount) as total_payment
from enrollment e
join payment p on e.students_id = p.students_id
```

```
group by student_id;
```

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 id, student_name
from students
where id in (
   select students_id
   from payment
   group by students_id
   having count(*) > 1
);
```

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.id, s.student_name, sum(p.amount) as total_payments
from students s
join payment p on s.id = p.students_id
group by s.id, s.student_name;
```

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.students_id) as enrollment_count
from course c
join enrollment e on c.id = e.course_id
group by c.id  ,c.course_name;
```

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.