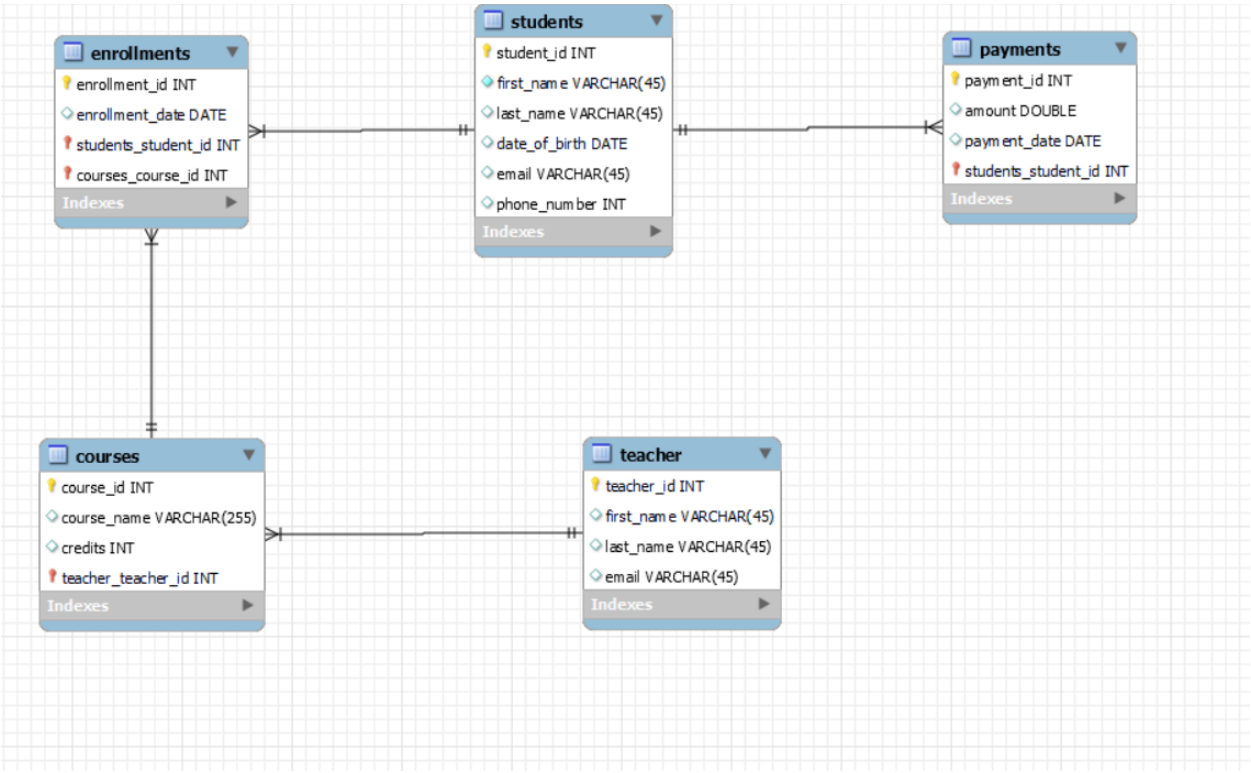
Student Information Management case study

ER DIAGRAM :



#db scripts

-- MySQL Workbench Forward Engineering

-- ---
-- Schema SISDB
-- ---

-- ---
-- Schema SISDB
-- ---
CREATE SCHEMA IF NOT EXISTS `SISDB` DEFAULT CHARACTER SET utf8 ;
USE `SISDB` ;

-- ---
-- Table `SISDB`.`students`
-- ---

```
CREATE TABLE IF NOT EXISTS `SISDB`.`students` (
  `student_id` INT NOT NULL AUTO_INCREMENT,
  `first_name` VARCHAR(45) NOT NULL,
  `last_name` VARCHAR(45) NULL,
  `date_of_birth` DATE NULL,
  `email` VARCHAR(45) NULL,
  `phone_number` INT NULL,
  PRIMARY KEY (`student_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `SISDB`.`payments`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `SISDB`.`payments` (
  `payment_id` INT NOT NULL AUTO_INCREMENT,
  `amount` DOUBLE NULL,
  `payment_date` DATE NULL,
  `students_student_id` INT NOT NULL,
  PRIMARY KEY (`payment_id`, `students_student_id`),
  INDEX `fk_payments_students_idx` (`students_student_id` ASC) ,
  CONSTRAINT `fk_payments_students`
    FOREIGN KEY (`students_student_id`)
    REFERENCES `SISDB`.`students` (`student_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `SISDB`.`teacher`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `SISDB`.`teacher` (
  `teacher_id` INT NOT NULL AUTO_INCREMENT,
  `first_name` VARCHAR(45) NULL,
  `last_name` VARCHAR(45) NULL,
  `email` VARCHAR(45) NULL,
  PRIMARY KEY (`teacher_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `SISDB`.`courses`
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `SISDB`.`courses` (
  `course_id` INT NOT NULL AUTO_INCREMENT,
  `course_name` VARCHAR(255) NULL,
  `credits` INT NULL,
  `teacher_teacher_id` INT NOT NULL,
  PRIMARY KEY (`course_id`, `teacher_teacher_id`),
  INDEX `fk_courses_teacher1_idx` (`teacher_teacher_id` ASC) ,
  CONSTRAINT `fk_courses_teacher1`
    FOREIGN KEY (`teacher_teacher_id`)
    REFERENCES `SISDB`.`teacher` (`teacher_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `SISDB`.`enrollments`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `SISDB`.`enrollments` (
  `enrollment_id` INT NOT NULL AUTO_INCREMENT,
  `enrollment_date` DATE NULL,
  `students_student_id` INT NOT NULL,
  `courses_course_id` INT NOT NULL,
  PRIMARY KEY (`enrollment_id`, `students_student_id`, `courses_course_id`),
  INDEX `fk_enrollments_students1_idx` (`students_student_id` ASC) ,
  INDEX `fk_enrollments_courses1_idx` (`courses_course_id` ASC) ,
  CONSTRAINT `fk_enrollments_students1`
    FOREIGN KEY (`students_student_id`)
    REFERENCES `SISDB`.`students` (`student_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_enrollments_courses1`
    FOREIGN KEY (`courses_course_id`)
    REFERENCES `SISDB`.`courses` (`course_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;;
```

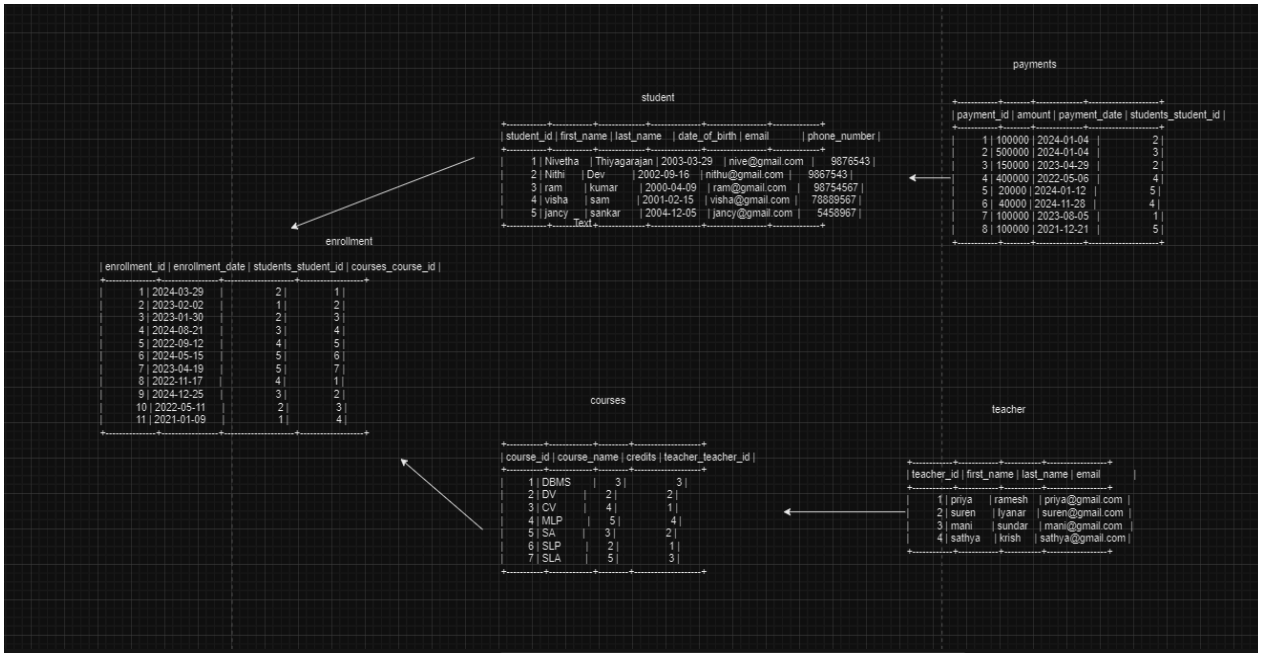
Query Insertions

```
use sisdb;
show tables;
# inssertions
```

```
insert into students (first_name , last_name, date_of_birth , email , phone_number)
values ('Nivetha ' , 'Thiyagarajan' , '2003-03-29', 'nive@gmail.com', 9876543),
('Nithi', 'Dev', '2002-09-16' , 'nithu@gmail.com ', 9867543),
('ram', 'kumar', '2000-04-09', 'ram@gmail.com' , 98754567),
('visha', 'sam', '2001-02-15', 'visha@gmail.com' , 78889567),
('jancy', 'sankar', '2004-12-05', 'jancy@gmail.com' , 5458967);

insert into payments (amount ,payment_date , students_student_id)
values(100000 , '2024-01-04',2),
(500000 , '2024-01-04',3),
(150000 , '2023-04-29',2),
(400000 , '2022-05-06',4),
(20000 , '2024-01-12',5),
(40000 , '2024-11-28',4),
(100000 , '2023-08-05',1),
(100000 , '2021-12-21',5);

insert into teacher (first_name , last_name , email)
values ('priya','ramesh','priya@gmail.com'),
('suren','Iyanar','suren@gmail.com'),
('mani','sundar','mani@gmail.com'),
('sathya','krish','sathya@gmail.com');

insert into courses(course_name , credits , teacher_teacher_id )
values ('DBMS' , 3 , 3),
('DV' , 2 , 2),
('CV' , 4 , 1),
('MLP' , 5 , 4),
('SA' , 3 , 2),
('SLP' , 2, 1),
('SLA' , 5 , 3);

insert into enrollments(enrollment_date , students_student_id , courses_course_id)
values('2024-03-29' , 2 ,1),
('2023-02-02' , 1 ,2),
('2023-01-30' , 2 ,3),
('2024-08-21' , 3 ,4),
('2022-09-12' , 4 ,5),
('2024-05-15' , 5 ,6),
('2023-04-19' , 5 ,7),
('2022-11-17' , 4 ,1),
('2024-12-25' , 3 ,2),
('2022-05-11' , 2 ,3),
```

('2021-01-09' , 1 ,4);

Reference Image:



TASK 2:

-- 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: 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(enrollment_date , students_student_id , courses_course_id)
values('2023-03-29' , 6 ,3);
-- 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='priya.r@gmail.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 students_student_id = 1 and courses_course_id = 4;

-- 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_teacher_id = 2 where course_id = 7;

-- 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 students_student_id = 1;
delete from payments where students_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 = 5000000 where payment_id =5;

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) as total_amount
from payments p join students s on s.student_id = p.students_student_id
where student_id =2;
```

-- 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_id,c.course_name , c.credits , count(e.students_student_id) as
Count_of_students
from courses c join enrollments e on c.course_id = e.courses_course_id
group by e.students_student_id;
```

-- 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.

```
insert into students (first_name , last_name, date_of_birth , email , phone_number)
```

values ('rama' , 'Devi' , '1996-08-15', 'rama@example.com', 1234567890);

```
select s.first_name , s.last_name
from students s left join enrollments e on s.student_id = e.students_student_id
where e.students_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.students_student_id join courses  c on
c.course_id = e. courses_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 , c.course_name
from courses c join teacher t on t.teacher_id = c.teacher_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 , e.enrollment_date
from students s join enrollments e  on s.student_id = e.students_student_id join courses  c on
c.course_id = e. courses_course_id
where c.course_name ='DBMS';
```

-- 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 , p.amount
from students s left join payments p on s.student_id = p.students_student_id
where p.amount 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.courses_course_id
where e.enrollment_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.

-- without self join

```
select   distinct students_student_id ,count(enrollment_id) as total_count
from enrollments
group by students_student_id
having total_count>1;
```

-- using self join

```
select distinct e1.students_student_id
from Enrollments e1
join Enrollments e2 ON e1.students_student_id = e2.students_student_id
Where e1.enrollment_id <> e2.enrollment_id;
```

-- 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
from Teacher t left join courses c on t.teacher_id = c.teacher_teacher_id
where c.course_id is null;
```

TASK 4:

-- 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(No_of_students) as average_no_of_students
from ( (select count(students_student_id) as No_of_students from enrollments group by
courses_course_id)) as subquery;
```

```
/*+----------------------+
| average_no_of_students |
+----------------------+
|                 1.5714 |
+----------------------+*/
```

-- 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 first_name
from students where student_id in (select students_student_id  from payments where 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.

```
select course_id
from courses where course_id in ( select courses_course_id from enrollments where
enrollment_id = ( select max(enrollment_id) from enrollments));
```

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

-- here course -amount is not given in the column name so i changed for student

-- 4. Calculate the total payments made to student get by each student. Use subqueries to sum payments for each student's

```
select students_student_id, total_amount
from
 (( select students_student_id , sum(amount) as total_amount from payments  group by
students_student_id)) as subquery;
```

-- 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 students_student_id
from enrollments
group by students_student_id
Having Count(courses_course_id) = (SELECT COUNT(course_id) FROM 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 first_name from teacher where teacher_id not in
( select distinct teacher_teacher_id from courses );
```

-- 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 , current_date()) as age from students)) as
subquery;
```

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

```
select course_id
from courses where course_id not in ( select distinct courses_course_id  from enrollments);
```

-- 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 e.courses_course_id , p.students_student_id, sum(amount) as total_amount
from enrollments e join payments p on e.students_student_id = p.Students_student_id
group by p.students_student_id  , e.courses_course_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 students_student_id , total_count
from (select students_student_id ,count(students_student_id) as total_count from payments
group by students_student_id) as subquery
where total_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.student_id ,sum(p.amount) as total_payment
from students s join payments p on s.student_id =p.students_student_id
group by s.student_id;
```

-- 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(*) AS count_of_students
from courses c join enrollments e on c.course_id = e.courses_course_id
group by 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.

```
select s.student_id ,avg(p.amount) as average_payments
from students s join payments p on s.student_id = p.students_student_id
group by s.student_id;
```