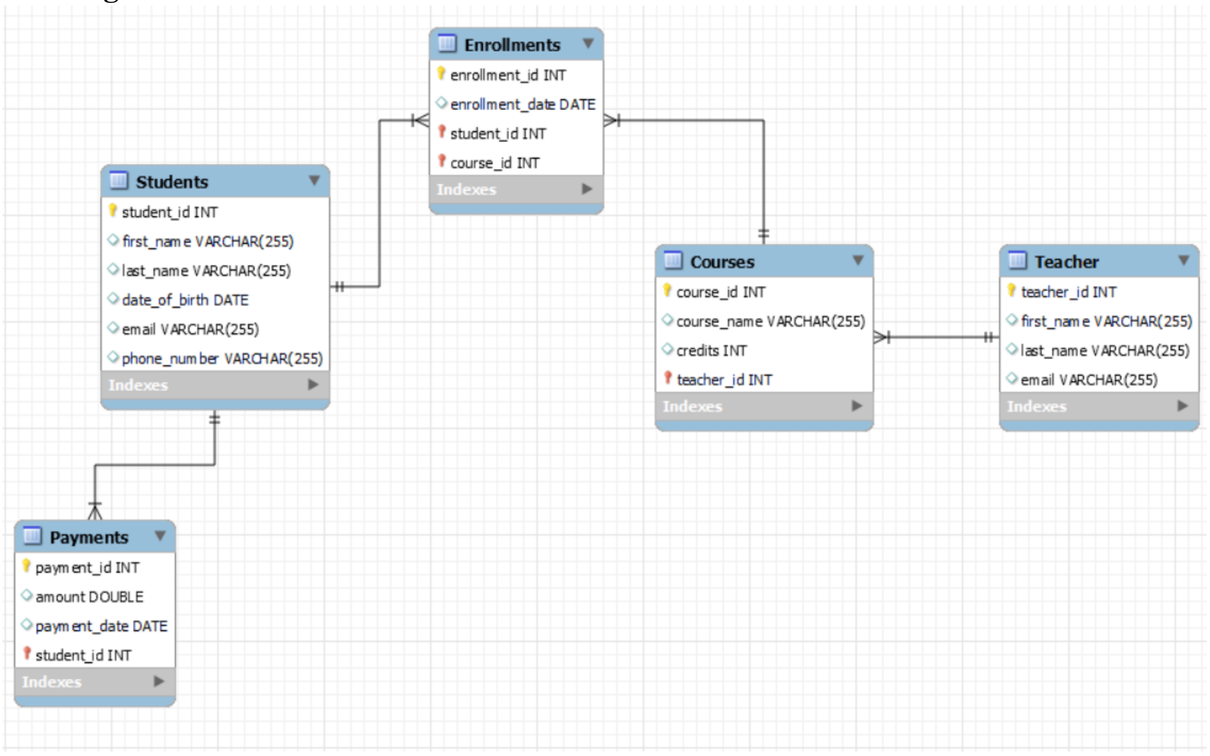
Student Information System

ER Diagram



TASK-1:

```
-- 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(255) NULL,
  `last_name` VARCHAR(255) NULL,
  `date_of_birth` DATE NULL,
  `email` VARCHAR(255) NULL,
  `phone_number` VARCHAR(255) NULL,
  PRIMARY KEY (`student_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `SISDB`.`Teacher`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `SISDB`.`Teacher` (
  `teacher_id` INT NOT NULL AUTO_INCREMENT,
  `first_name` VARCHAR(255) NULL,
  `last_name` VARCHAR(255) NULL,
  `email` VARCHAR(255) 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_id` INT NOT NULL,
  PRIMARY KEY (`course_id`, `teacher_id`),
  INDEX `fk_Courses_Teacher1_idx` (`teacher_id` ASC),
  CONSTRAINT `fk_Courses_Teacher1`
    FOREIGN KEY (`teacher_id`)
    REFERENCES `SISDB`.`Teacher` (`teacher_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
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,
  `student_id` INT NOT NULL,
  PRIMARY KEY (`payment_id`, `student_id`),
  INDEX `fk_Payments_Students_idx` (`student_id` ASC),
  CONSTRAINT `fk_Payments_Students`
    FOREIGN KEY (`student_id`)
    REFERENCES `SISDB`.`Students` (`student_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,
  `student_id` INT NOT NULL,
  `course_id` INT NOT NULL,
  PRIMARY KEY (`enrollment_id`, `student_id`, `course_id`),
  INDEX `fk_Enrollments_Students1_idx` (`student_id` ASC),
  INDEX `fk_Enrollments_Courses1_idx` (`course_id` ASC),
  CONSTRAINT `fk_Enrollments_Students1`
    FOREIGN KEY (`student_id`)
    REFERENCES `SISDB`.`Students` (`student_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Enrollments_Courses1`
    FOREIGN KEY (`course_id`)
    REFERENCES `SISDB`.`Courses` (`course_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

TASK-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', '1995-03-15', '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, 1, '2024-02-15');
```

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='lasith.malinga@yahoo.com'
where teacher_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 student_id=2 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=1
where course_id=4;
```

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 = (select student_id FROM Students where first_name = 'John' AND
last_name = 'Doe');
delete from Students where first_name = 'John' AND last_name = 'Doe';
```

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=650
where student_id=1;
```

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 student_id,sum(amount) from payments
group by student_id;
```

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(enrollment_id) as Total_Students_enrolled
from courses c join enrollments e on c.course_id=e.course_id
group by e.course_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.

```
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 e.course_id=c.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 c.teacher_id=t.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.* ,e.enrollment_date from

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

where e.course_id=1;

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

where p.student_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.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 distinct e1.student_id, s.first_name, s.last_name

from enrollments e1

join enrollments e2 on e1.student_id = e2.student_id and e1.enrollment_id = e2.enrollment_id

join students s on e1.student_id = s.student_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_id

where c.teacher_id is not 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 course_id, course_name, avg(enrollment_count) as average_students_enrolled

from (

   select c.course_id, c.course_name, count(e.student_id) as enrollment_count

   from courses c

   left join enrollments e on c.course_id = e.course_id

   group by c.course_id

) as subQuery

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 first_name,last_name from students

where student_id = (select 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 ,max(total_enrollment) as max_num_of_enrollments from (select
course_id,count(enrollment_id) as total_enrollment

from enrollments

group by course_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 student_id, first_name, last_name

from students

where student_id in (

   select e.student_id
```

```
    from enrollments e
    group by e.student_id
    having count(distinct e.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,last_name from teacher
where teacher_id not in (select 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, CURDATE()) as age
from Students) as sq;
```

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

```
select * from courses
where course_id not in (select 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. // doubt

```
select student_id,sum(amount) as total_payment
from payments
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 student_id,first_name,last_name from students
where student_id in (select student_id from payments group by student_id having count(payment_id)>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 student_id,sum(amount) as total_payments from payments
group by 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(e.enrollment_id) from

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

group by e.course_id;
```

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,s.first_name,s.last_name,avg(amount) as Average_payment from payments p

join students s on p.student_id=s.student_id

group by s.student_id;
```