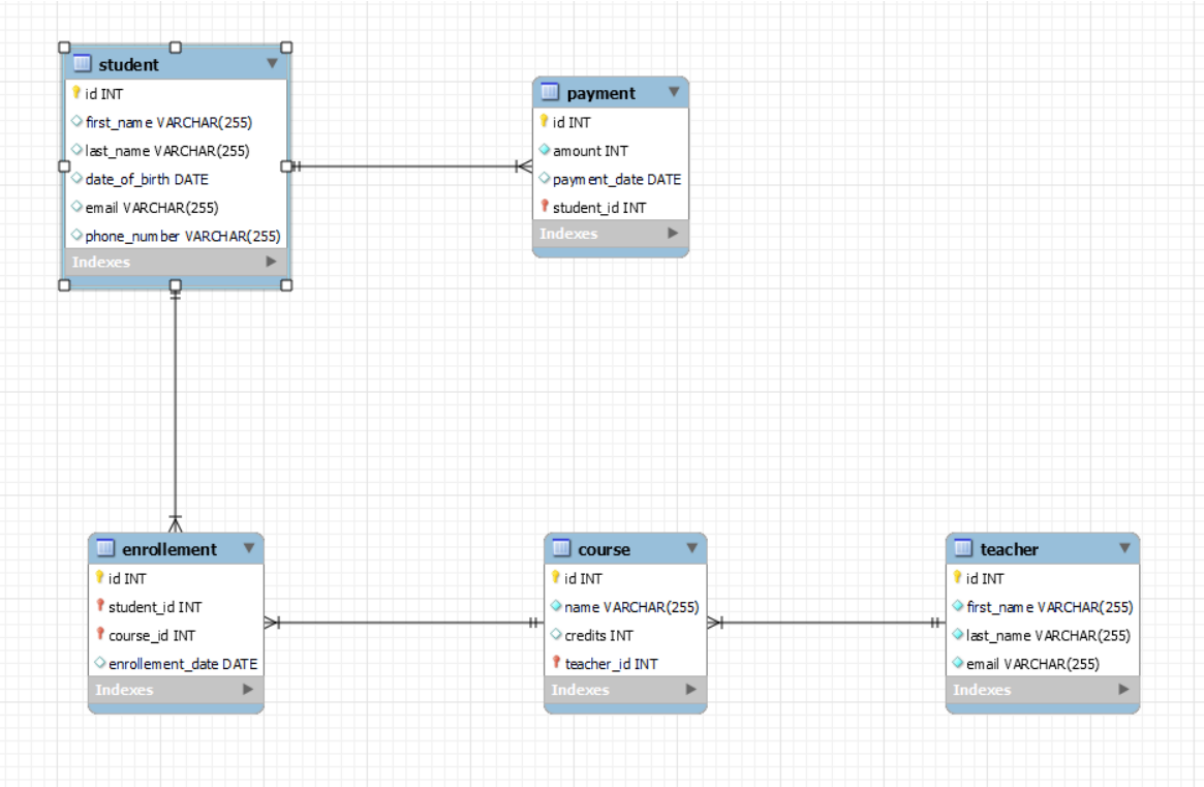
ASSIGNMENT NO : 4

STUDENT INFORMATION SYSTEM

ER DIAGRAM:



Task:1. Database Design:

-- MySQL Workbench Forward Engineering

-- ---

-- Schema sms

-- ---

-- ---

-- Schema sms

-- ---

```
CREATE SCHEMA IF NOT EXISTS `sms` DEFAULT CHARACTER SET utf8 ;
USE `sms` ;

-- -----------------------------------------------------
-- Table `sms`.`student`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `sms`.`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 (`id`),
  UNIQUE INDEX `phone_number_UNIQUE` (`phone_number` ASC) )
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `sms`.`payment`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `sms`.`payment` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `amount` INT NOT NULL,
  `payment_date` DATE NULL,
  `student_id` INT NOT NULL,
  PRIMARY KEY (`id`, `student_id`),
  INDEX `fk_payment_student_idx` (`student_id` ASC) ,
  CONSTRAINT `fk_payment_student`
    FOREIGN KEY (`student_id`)
    REFERENCES `sms`.`student` (`id`)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `sms`.`teacher`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `sms`.`teacher` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `first_name` VARCHAR(255) NOT NULL,
  `last_name` VARCHAR(255) NOT NULL,
  `email` VARCHAR(255) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `sms`.`course`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `sms`.`course` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(255) NOT NULL,
  `credits` INT NULL,
  `teacher_id` INT NOT NULL,
  PRIMARY KEY (`id`, `teacher_id`),
  INDEX `fk_course_teacher1_idx` (`teacher_id` ASC) ,
  CONSTRAINT `fk_course_teacher1`
    FOREIGN KEY (`teacher_id`)
    REFERENCES `sms`.`teacher` (`id`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------

-- Table `sms`.`enrollement`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `sms`.`enrollement` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `student_id` INT NOT NULL,
  `course_id` INT NOT NULL,
  `enrollement_date` DATE NULL,
  PRIMARY KEY (`id`, `student_id`, `course_id`),
  INDEX `fk_enrollement_student1_idx` (`student_id` ASC) ,
  INDEX `fk_enrollement_course1_idx` (`course_id` ASC) ,
  CONSTRAINT `fk_enrollement_student1`
    FOREIGN KEY (`student_id`)
    REFERENCES `sms`.`student` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_enrollement_course1`
    FOREIGN KEY (`course_id`)
    REFERENCES `sms`.`course` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

INSERTION:

-- student insertion

```
insert into student(first_name,last_name,date_of_birth,email,phone_number)values
 ('prasanna','prabakaran','2003-03-28','prasanna@gmail.com','9360805403'),
 ('pradeep','munusamy','2003-10-09','pradeep@gmail.com','9360805405'),
 ('niranjan','kumar','2003-05-17','niranjan@gmail.com','9360805497'),
 ('naveen','kumar','2003-06-17','naveen@gmail.com','9360805433'),
 ('mani','bharathi','2002-11-13','mani@gmail.com','9360805444'),
 ('gokul','kumar','2002-03-09','gokul@gmail.com','9360805422'),
 ('sedhu','prakash','2002-07-07','sedhu@gmail.com','9360805400'),
 ('selva','vignesh','2003-07-06','selva@gmail.com','9360809864'),
 ('ragul','balaji','2003-05-23','ragul@gmail.com','9360803334'),
 ('vinay','kumar','2003-04-04','vinay@gmail.com','93608035642');
```

```
ysql> select * from student;
+----+------------+------------+---------------+--------------------+--------------+
| id | first_name | last_name  | date_of_birth | email              | phone_number |
+----+------------+------------+---------------+--------------------+--------------+
|  1 | prasanna   | prabakaran | 2003-03-28    | prasanna@gmail.com | 9360805403   |
|  2 | pradeep    | munusamy   | 2003-10-09    | pradeep@gmail.com  | 9360805405   |
|  3 | niranjan   | kumar      | 2003-05-17    | niranjan@gmail.com | 9360805497   |
|  4 | naveen     | kumar      | 2003-06-17    | naveen@gmail.com   | 9360805433   |
|  5 | mani       | bharathi   | 2002-11-13    | mani@gmail.com     | 9360805444   |
|  6 | gokul      | kumar      | 2002-03-09    | gokul@gmail.com    | 9360805422   |
|  7 | sedhu      | prakash    | 2002-07-07    | sedhu@gmail.com    | 9360805400   |
|  8 | selva      | vignesh    | 2003-07-06    | selva@gmail.com    | 9360809864   |
|  9 | ragul      | balaji     | 2003-05-23    | ragul@gmail.com    | 9360803334   |
| 10 | vinay      | kumar      | 2003-04-04    | vinay@gmail.com    | 93608035642  |
+----+------------+------------+---------------+--------------------+--------------+
```

-- payment insertion

```
insert into payment(amount,payment_date,student_id) values
  (10000,'2024-09-18',1),
  (12200,'2025-01-12',1),
  (11000,'2024-08-07',2),
```

(20000,'2024-05-04',2),

(4000,'2024-03-19',2),

(5000,'2024-07-07',3),

(5500,'2024-03-03',5),

(5600,'2024-12-14',6),

(3400,'2025-11-11',7),

(9900,'2024-05-12',8),

(8000,'2024-08-12',9),

(9900,'2024-08-08',10);

```
mysql> select * from payment;
+----+--------+--------------+------------+
| id | amount | payment_date | student_id |
+----+--------+--------------+------------+
|  1 |  10000 | 2024-09-18   |          1 |
|  2 |  12200 | 2025-01-12   |          1 |
|  3 |  11000 | 2024-08-07   |          2 |
|  4 |  20000 | 2024-05-04   |          2 |
|  5 |   4000 | 2024-03-19   |          2 |
|  6 |   5000 | 2024-07-07   |          3 |
|  7 |   5500 | 2024-03-03   |          5 |
|  8 |   5600 | 2024-12-14   |          6 |
|  9 |   3400 | 2025-11-11   |          7 |
| 10 |   9900 | 2024-05-12   |          8 |
| 11 |   8000 | 2024-08-12   |          9 |
| 12 |   9900 | 2024-08-08   |         10 |
+----+--------+--------------+------------+
```

-- teacher insertion

insert into teacher(first_name,last_name,email)values

('rajesh','kumar','rajesh@gmail.com'),

('ravi','kumar','ravi@gmail.com'),

('vimal','doss','vimal@gmail.com'),

('sam','ben','sam@gmail.com'),

('lizaad','williams','lizaad@gmail.com'),

('ben','stokes','ben@gmail.com'),

('tim','seifert','tim@gmail.com'),

('ben','duckett','duckett@gmail.com'),

('zak','crawley','zak@gmail.com'),

('ollie','pope','ollie@gmail.com');

```
mysql> select * from teacher;
+----+------------+-----------+--------------------+
| id | first_name | last_name | email              |
+----+------------+-----------+--------------------+
|  1 | rajesh     | kumar     | rajesh@gmail.com   |
|  2 | ravi       | kumar     | ravi@gmail.com     |
|  3 | vimal      | doss      | vimal@gmail.com    |
|  4 | sam        | ben       | sam@gmail.com      |
|  5 | lizaad     | williams  | lizaad@gmail.com   |
|  6 | ben        | stokes    | ben@gmail.com      |
|  7 | tim        | seifert   | tim@gmail.com      |
|  8 | ben        | duckett   | duckett@gmail.com  |
|  9 | zak        | crawley   | zak@gmail.com      |
| 10 | ollie      | pope      | ollie@gmail.com    |
+----+------------+-----------+--------------------+
```

-- course insertion

INSERT INTO course(name, credits, teacher_id) VALUES

('c', 10, 1),

('c++', 10, 2),

('python', 10, 3),

('java', 10, 4), -- Corrected this line

('c#', 10, 5),

('php', 10, 6),

('dbms', 10, 7),

('ai', 10, 8),

('cloud computing', 10, 9),

('html', 10, 10);

```
mysql> select * from course;
+----+-----------------+---------+------------+
| id | name            | credits | teacher_id |
+----+-----------------+---------+------------+
|  1 | c               |      10 |          1 |
|  2 | c++             |      10 |          2 |
|  3 | python          |      10 |          3 |
|  4 | java            |      10 |          4 |
|  5 | c#              |      10 |          5 |
|  6 | php             |      10 |          6 |
|  7 | dbms            |      10 |          7 |
|  8 | ai              |      10 |          8 |
|  9 | cloud computing |      10 |          9 |
| 10 | html            |      10 |         10 |
+----+-----------------+---------+------------+
```

-- enrollement insertion

insert into enrollement(student_id,course_id,enrollement_date)values

(1,2,'2023-04-04'),

(2,1,'2022-05-06'),

(3,5,'2023-01-02'),

(4,3,'2023-09-11'),

(5,4,'2023-05-29'),

(6,7,'2023-10-12'),

(7,6,'2023-02-14'),

(8,9,'2023-11-13'),

(9,10,'2023-07-14'),

(10,8,'2023-04-16');

```
mysql> select * from enrollement;
+----+------------+-----------+-----------------+
| id | student_id | course_id | enrollement_date |
+----+------------+-----------+-----------------+
|  1 |          1 |         2 | 2023-04-04      |
|  2 |          2 |         1 | 2022-05-06      |
|  3 |          3 |         5 | 2023-01-02      |
|  4 |          4 |         3 | 2023-09-11      |
|  5 |          5 |         4 | 2023-05-29      |
|  6 |          6 |         7 | 2023-10-12      |
|  7 |          7 |         6 | 2023-02-14      |
|  8 |          8 |         9 | 2023-11-13      |
|  9 |          9 |        10 | 2023-07-14      |
| 10 |         10 |         8 | 2023-04-16      |
+----+------------+-----------+-----------------+
```

--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 student(first_name,last_name,date_of_birth,email,phone_number) values
('John','Doe','1995-08-15','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 enrollement(student_id,course_id,enrollement_date)values
(1,3,'2024-02-03');
```

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='doss@gmail.com' where id=3;
```

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 enrollement WHERE student_id = 3 AND course_id = 5;
```

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 name='javascript' where 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 enrollement WHERE student_id = 4;

Delete from Student WHERE student_id = 4;
```

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=1000 where id=1;
```

Task 3. Aggregate functions, Having, Order By, Group By 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 s.id, s.first_name, SUM(p.amount) AS total_payments FROM student s
JOIN payment p ON s.id = p.student_id WHERE s.id =your_student_id
GROUP BY s.id, s.first_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.id AS course_id,  c.name AS course_name, COUNT(e.student_id) AS
number_of_students FROM course AS c LEFT JOIN sms.enrollement AS e ON c.id =
e.course_id GROUP BY c.id, c.name ORDER BY number_of_students 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.id, s.first_name, s.last_name FROM student AS s LEFT JOIN enrollement AS
e ON s.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.name AS course_name FROM student AS s
JOIN enrollement AS e ON s.id = e.student_id JOIN course AS c ON e.course_id = c.id
ORDER BY s.first_name, s.last_name, c.name;
```

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.name AS course_name FROM teacher AS t
JOIN course AS c ON t.id = c.teacher_id ORDER BY t.first_name, t.last_name, c.name;
```

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.enrollement_date FROM student AS s
JOIN enrollement AS e ON s.id = e.student_id JOIN course AS c ON e.course_id = c.id
WHERE c.name = 'java';
```

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.id, s.first_name, s.last_name FROM student AS s LEFT JOIN payment AS p
ON s.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.id, c.name FROM course AS c LEFT JOIN enrollement AS 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 s.id, s.first_name, s.last_name, COUNT(e.student_id) AS course_count
FROM student AS s JOIN enrollement AS e ON s.id = e.student_id
GROUP BY s.id, s.first_name, s.last_name HAVING COUNT(e.student_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.first_name, t.last_name FROM teacher AS t LEFT JOIN course AS 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(student_count) AS average_students_per_course FROM (SELECT
course_id, COUNT(student_id) AS student_count FROM enrollement
GROUP BY course_id) AS course_enrollments;
```

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.id AS student_id, s.first_name, s.last_name, s.email, s.phone_number,
p.amount AS payment_amount FROM student s JOIN payment p ON s.id =
p.student_id WHERE p.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.

```
SELECT c.id AS course_id, c.name AS course_name, c.credits, c.teacher_id,
COUNT(e.student_id) AS enrollment_count FROM course c JOIN enrollement e ON
c.id = e.course_id GROUP BY c.id, c.name, c.credits, c.teacher_id HAVING
COUNT(e.student_id) = ( SELECT MAX(enrollment_count)
FROM (SELECT course_id, COUNT(student_id) AS enrollment_count FROM
enrollement GROUP BY course_id) AS course_enrollments);
```

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

```
SELECT t.id AS teacher_id, t.first_name, t.last_name, t.email, SUM(p.amount) AS
total_payments FROM teacher t JOIN course c ON t.id = c.teacher_id
JOIN enrollement e ON c.id = e.course_id JOIN payment p ON e.student_id =
p.student_id GROUP BY t.id, t.first_name, t.last_name, t.email;
```

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 s.id AS student_id, s.first_name, s.last_name, s.email FROM student s
WHERE (SELECT COUNT(DISTINCT course_id) FROM enrollement e
WHERE e.student_id = s.id) = (SELECT COUNT(*) FROM course)
```

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.id AS teacher_id, t.first_name, t.last_name, t.email FROM teacher t

WHERE NOT EXISTS ( SELECT * FROM course c WHERE c.teacher_id = t.id);
```

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 student) AS student_ages;
```

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

```
SELECT c.id AS course_id, c.name AS course_name, c.credits, c.teacher_id
FROM course c WHERE NOT EXISTS (SELECT * FROM enrollement e
WHERE e.course_id = c.id);
```

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.student_id, e.course_id, s.first_name, s.last_name, c.name AS
course_name, SUM(p.amount) AS total_payments FROM enrollement e

JOIN student s ON e.student_id = s.id JOIN course c ON e.course_id = c.id

JOIN payment p ON e.student_id = p.student_id GROUP BY e.student_id,
e.course_id, s.first_name, s.last_name, c.name;
```

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.id AS student_id, s.first_name, s.last_name, s.email, COUNT(p.id) AS
number_of_payments FROM student s JOIN payment p ON s.id = p.student_id
GROUP BY s.id, s.first_name, s.last_name, s.email HAVING COUNT(p.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 s.id AS student_id, s.first_name, s.last_name, s.email, SUM(p.amount) AS
total_payments FROM student s JOIN payment p ON s.id = p.student_id
GROUP BY s.id, s.first_name, s.last_name, s.email;
```

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.name AS course_name, COUNT(e.student_id) AS number_of_students
FROM course c JOIN enrollement e ON c.id = e.course_id GROUP BY c.id, c.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 AVG(p.amount) AS average_payment_amount FROM student s
JOIN payment p ON s.id = p.student_id;
```