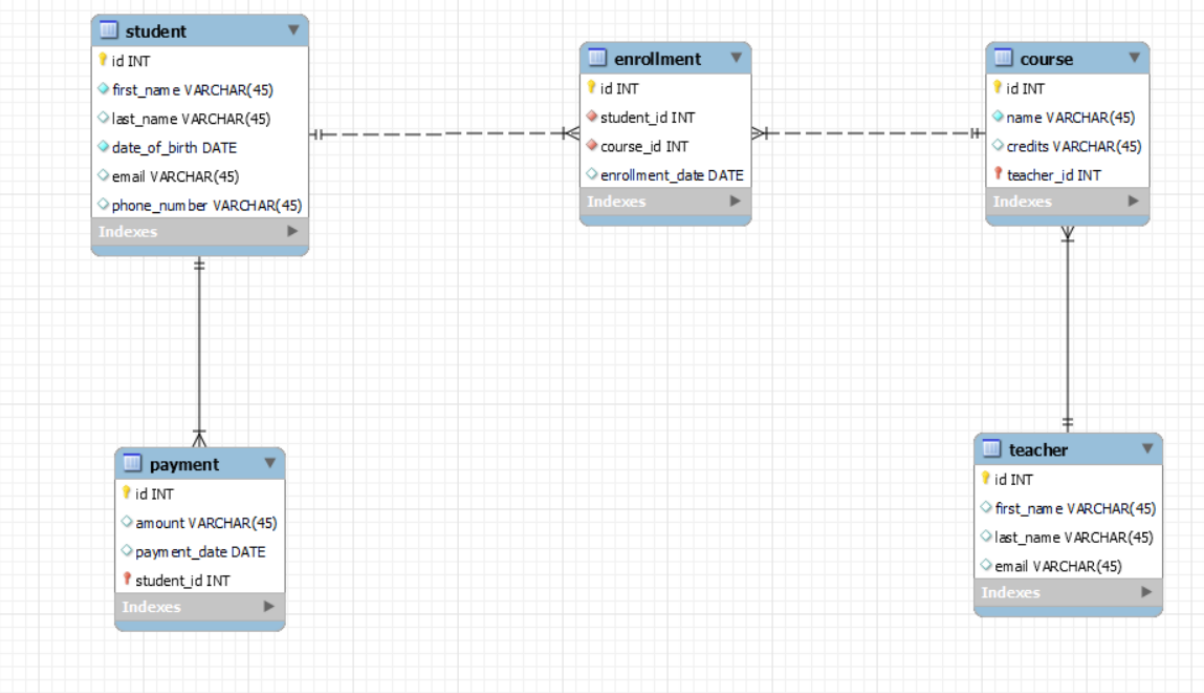
ASSIGNMENT 2

STUDENT INFORMATION SYSTEM

ER DIAGRAM:



Task 1: Database Design

-- MySQL Workbench Forward Engineering

-- ---

-- Schema assignment_student_information

-- ---

-- ---

-- Schema assignment_student_information

-- ---

CREATE SCHEMA IF NOT EXISTS `assignment_student_information` DEFAULT CHARACTER SET utf8 ;

```
USE `assignment_student_information` ;


-- -----------------------------------------------------
-- Table `assignment_student_information`.`student`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `assignment_student_information`.`student` (
  `id` INT NOT NULL,
  `first_name` VARCHAR(45) NOT NULL,
  `last_name` VARCHAR(45) NULL,
  `date_of_birth` DATE NOT NULL,
  `email` VARCHAR(45) NULL,
  `phone_number` VARCHAR(45) NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;



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



-- -----------------------------------------------------
-- Table `assignment_student_information`.`course`
```

```
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `assignment_student_information`.`course` (
  `id` INT NOT NULL,
  `name` VARCHAR(45) NOT NULL,
  `credits` VARCHAR(45) 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 `assignment_student_information`.`teacher` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `assignment_student_information`.`payment`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `assignment_student_information`.`payment` (
  `id` INT NOT NULL,
  `amount` VARCHAR(45) NULL,
  `payment_date` DATE NULL,
  `student_id` INT NOT NULL,
  PRIMARY KEY (`id`, `student_id`),
  INDEX `fk_payment_student1_idx` (`student_id` ASC),
  CONSTRAINT `fk_payment_student1`
    FOREIGN KEY (`student_id`)
    REFERENCES `assignment_student_information`.`student` (`id`)
    ON DELETE NO ACTION
```

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



-- -----------------------------------------------------
-- Table `assignment_student_information`.`enrollment`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `assignment_student_information`.`enrollment` (
  `id` INT NOT NULL,
  `student_id` INT NOT NULL,
  `course_id` INT NOT NULL,
  `enrollment_date` DATE NULL,
  INDEX `fk_student_has_course_course1_idx` (`course_id` ASC),
  INDEX `fk_student_has_course_student_idx` (`student_id` ASC),
  PRIMARY KEY (`id`),
  CONSTRAINT `fk_student_has_course_student`
    FOREIGN KEY (`student_id`)
    REFERENCES `assignment_student_information`.`student` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_student_has_course_course1`
    FOREIGN KEY (`course_id`)
    REFERENCES `assignment_student_information`.`course` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

INSERTION:

--- student insertion

```
INSERT INTO student (id,first_name, last_name, date_of_birth, email, phone_number)
VALUES
    (1,'MS', 'Dhoni', '1995-08-15', 'msd@gmail.com', '1234567890'),
    (2,'Rishab', 'Pant', '1998-03-20', 'rp@gmail.com', '9876543210'),
    (3,'Rohit', 'Sharma', '1997-12-10', 'rk@gmail.com', '5678901234'),
    (4,'Virat', 'Kohli', '1996-05-25', 'vk@gmail.com', '3456789012'),
    (5,'Jasprit', 'Bumrah', '1999-09-05', 'boom@gmail.com', '7890123456'),
    (6,'Kuldeep', 'Yadav', '1994-11-18', 'kv@gmail.com', '2345678901'),
    (7,'Ravichandran', 'Ashwin', '2000-02-08', 'ash@gmail.com', '8901234567'),
    (8,'Rinku', 'Singh', '1993-07-30', 'rs@gmail.com', '4567890123'),
    (9,'Ravindra', 'Jadeja', '1992-04-12', 'jdja@gmail.com', '6789012345'),
    (10,'Shubman', 'gill', '1991-01-05', 'sg@gmail.com', '9012345678');
```

```
mysql> select * from student;
+----+--------------+-----------+---------------+----------------+--------------+
| id | first_name   | last_name | date_of_birth | email          | phone_number |
+----+--------------+-----------+---------------+----------------+--------------+
|  1 | MS           | Dhoni     | 1995-08-15    | msd@gmail.com  | 1234567890   |
|  2 | Rishab       | Pant      | 1998-03-20    | rp@gmail.com   | 9876543210   |
|  3 | Rohit        | Sharma    | 1997-12-10    | rk@gmail.com   | 5678901234   |
|  4 | Virat        | Kohli     | 1996-05-25    | vk@gmail.com   | 3456789012   |
|  5 | Jasprit      | Bumrah    | 1999-09-05    | boom@gmail.com | 7890123456   |
|  6 | Kuldeep      | Yadav     | 1994-11-18    | kv@gmail.com   | 2345678901   |
|  7 | Ravichandran | Ashwin    | 2000-02-08    | ash@gmail.com  | 8901234567   |
|  8 | Rinku        | Singh     | 1993-07-30    | rs@gmail.com   | 4567890123   |
|  9 | Ravindra     | Jadeja    | 1992-04-12    | jdja@gmail.com | 6789012345   |
| 10 | Shubman      | gill      | 1991-01-05    | sg@gmail.com   | 9012345678   |
+----+--------------+-----------+---------------+----------------+--------------+
10 rows in set (0.02 sec)
```

--- teacher insertion

```
INSERT INTO Teacher (id,first_name, last_name, email)
VALUES
   (1,'Rahul', 'Dravid', 'rd@wall.com'),
   (2,'Ravi', 'Shastri', 'shashtri@gabba.com'),
   (3,'Kapil', 'Dev', 'kapil@wc.com');
```

```
mysql> select * from teacher;
+----+------------+-----------+--------------------+
| id | first_name | last_name | email              |
+----+------------+-----------+--------------------+
|  1 | Rahul      | Dravid    | rd@wall.com        |
|  2 | Ravi       | Shastri   | shashtri@gabba.com |
|  3 | Kapil      | Dev       | kapil@wc.com       |
+----+------------+-----------+--------------------+
3 rows in set (0.00 sec)
```

--- course insertion

```
INSERT INTO course (id,name, credits, teacher_id) VALUES
   (1,'Mathematics', 3, 1),
   (2,'History', 4, 2),
   (3,'Biology', 3, 3),
   (4,'Chemistry', 4, 1),
   (5,'Physics', 4, 2),
   (6,'English', 3, 3),
   (7,'Computer Science', 4, 1),
   (8,'Geography', 3, 2)
   (9,'Art', 2, 3),
   (10,'Music', 2, 1);
```

```
mysql> select * from course;
+----+------------------+---------+------------+
| id | name             | credits | teacher_id |
+----+------------------+---------+------------+
|  1 | Mathematics      | 3       |          1 |
|  2 | History          | 4       |          2 |
|  3 | Biology          | 3       |          3 |
|  4 | Chemistry        | 4       |          1 |
|  5 | Physics          | 4       |          2 |
|  6 | English          | 3       |          3 |
|  7 | Computer Science | 4       |          1 |
|  8 | Geography        | 3       |          2 |
|  9 | Art              | 2       |          3 |
| 10 | Music            | 2       |          1 |
+----+------------------+---------+------------+
10 rows in set (0.02 sec)
```

--- enrollment insertion

```
INSERT INTO enrollment (id,student_id, course_id, enrollment_date) VALUES
   (1,1, 1, '2023-09-01'),
   (2,2, 3, '2023-09-05'),
   (3,3, 5, '2023-09-10'),
   (4,4, 7, '2023-09-15'),
   (5,5, 9, '2023-09-20'),
   (6,6, 2, '2023-09-25'),
   (7,7, 4, '2023-09-30'),
   (8,8, 6, '2023-10-01'),
   (9,9, 8, '2023-10-05'),
   (10,10, 10, '2023-10-10');
```

```
mysql> select * from enrollment;
+----+------------+-----------+-----------------+
| id | student_id | course_id | enrollment_date |
+----+------------+-----------+-----------------+
|  1 |          1 |         1 | 2023-09-01      |
|  2 |          2 |         3 | 2023-09-05      |
|  3 |          3 |         5 | 2023-09-10      |
|  4 |          4 |         7 | 2023-09-15      |
|  5 |          5 |         9 | 2023-09-20      |
|  6 |          6 |         2 | 2023-09-25      |
|  7 |          7 |         4 | 2023-09-30      |
|  8 |          8 |         6 | 2023-10-01      |
|  9 |          9 |         8 | 2023-10-05      |
| 10 |         10 |        10 | 2023-10-10      |
+----+------------+-----------+-----------------+
10 rows in set (0.00 sec)
```

--- payment insertion

```
INSERT INTO payment (id, amount, payment_date,student_id)
VALUES
   (1, 500.00, '2023-09-01',1),
   (2, 600.00, '2023-09-05',2),
   (3, 700.00, '2023-09-10',3),
   (4, 800.00, '2023-09-15',4),
   (5, 900.00, '2023-09-20',5),
   (6, 1000.00, '2023-09-25',6),
   (7, 1100.00, '2023-09-30',7),
   (8, 1200.00, '2023-10-01',8),
   (9, 1300.00, '2023-10-05',9),
   (10, 1400.00, '2023-10-10',10);
```

```
mysql> select * from payment;
+----+---------+--------------+------------+
| id | amount  | payment_date | student_id |
+----+---------+--------------+------------+
|  1 | 500.00  | 2023-09-01   |          1 |
|  2 | 600.00  | 2023-09-05   |          2 |
|  3 | 700.00  | 2023-09-10   |          3 |
|  4 | 800.00  | 2023-09-15   |          4 |
|  5 | 900.00  | 2023-09-20   |          5 |
|  6 | 1000.00 | 2023-09-25   |          6 |
|  7 | 1100.00 | 2023-09-30   |          7 |
|  8 | 1200.00 | 2023-10-01   |          8 |
|  9 | 1300.00 | 2023-10-05   |          9 |
| 10 | 1400.00 | 2023-10-10   |         10 |
+----+---------+--------------+------------+
10 rows in set (0.00 sec)
```

-- 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','1995-08-15','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 (student_id,course_id,enrollment_date)
values (11,10,'2023-03-30');
```

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=’rd@wallofcricket.com' where 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 enrollment
where student_id=11;
```

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=1 where name='Art';
```

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.

```
alter table enrollment
add constraint fk_deletion
```

```
foreign key(student_id)
references student(id)
on delete cascade;
alter table payment
add constraint fkk_deletion
foreign key (student_id)
references student(id)
on delete cascade;
delete from student where 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 payment
set amount=2800 where id=6;
```

-- 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 concat(s.first_name,s.last_name) as name , p.amount
from student s join payment p
on s.id=p.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.name,count(e.course_id) as
number_of_students_enrolles
from course c join enrollment e
on c.id=e.course_id
group by c.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 concat(s.first_name," ",s.last_name) as
Stusents_not_enrolled_in_any_course
from student s left join enrollment 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 c.name, group_concat(concat(s.first_name,"
",s.last_name)) as students_enrolled
from student s join enrollment e on e.student_id=s.id
join course c on c.id=e.course_id
group by c.id;
select concat(s.first_name," ",s.last_name) as name,
group_concat(c.name) as courses_enrolled
from student s join enrollment e on e.student_id=s.id
join course c on c.id=e.course_id
group by s.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 concat(t.first_name," ",t.last_name) as teacher_names
,group_concat(c.name) as course
from course c join teacher t on t.id=c.teacher_id
group by t.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 c.name,concat(s.first_name," ",s.last_name) as
student_name,e.enrollment_date
from student s join enrollment e on e.student_id=s.id
join course c on c.id=e.course_id
where c.name='c';
```

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 concat(s.first_name," ",s.last_name) as student_name
from student s left join payment p on p.student_id=s.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.name from
course c left join enrollment e on e.course_id=c.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 concat(s.first_name," ",s.last_name) as student_name
from student s join enrollment e
on e.student_id=s.id
group by e.student_id
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 concat(t.first_name," ",t.last_name) as teacher_names
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 course_id,avg(student_count) as avg_enrollment
from (select course_id,count(student_id)as student_count
from enrollment
 group by course_id)as enrollment_counts
 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 concat(first_name," ",last_name) as student_name
from student where id=(select student_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.

```
select c.name ,(select count(e.student_id)
from enrollment e where e.course_id=c.id)
as count_of_students_enrolled
from course c group by c.id
order by count_of_students_enrolled desc limit 0,1;
```

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

```
select concat(t.first_name," ",t.last_name) as teacher_name,
c.name,p.amount
from teacher t join course c on t.id=c.teacher_id
join enrollment e on e.course_id=c.id
join payment p on p.student_id=e.student_id
group by t.id;
```

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 concat(s.first_name," ",s.last_name) as name
from student s join enrollment e
on e.student_id=s.id
group by s.id
having count(distinct e.course_id)=(SELECT COUNT(DISTINCT id)
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 concat(first_name," ",last_name) as 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(timestampdiff(year,date_of_birth,curdate()))
as average_age
from student;
```

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

```
select name from course
where id not in (select 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 s.first_name, s.last_name, c.name AS course_name,
SUM(p.amount) AS total_payments
FROM student s
JOIN enrollment e ON s.id = e.student_id
JOIN course c ON e.course_id = c.id
JOIN payment p ON s.id = p.student_id
GROUP BY s.id, c.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 concat(first_name," ",last_name) as name from student
where id in(select student_id from payment
group by student_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 CONCAT(s.first_name, " ", s.last_name) AS student_name,
SUM(p.amount) AS total_payments
FROM student s
JOIN payment p ON s.id = p.student_id
```

GROUP BY s.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.name, count(e.student_id) as count_of_students
from course c join enrollment e
on c.id=e.course_id
group by c.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 concat(s.first_name," ",s.last_name) as
name,avg(p.amount) as avg_payment
from student s join payment p
on p.student_id=s.id
group by s.id;
```