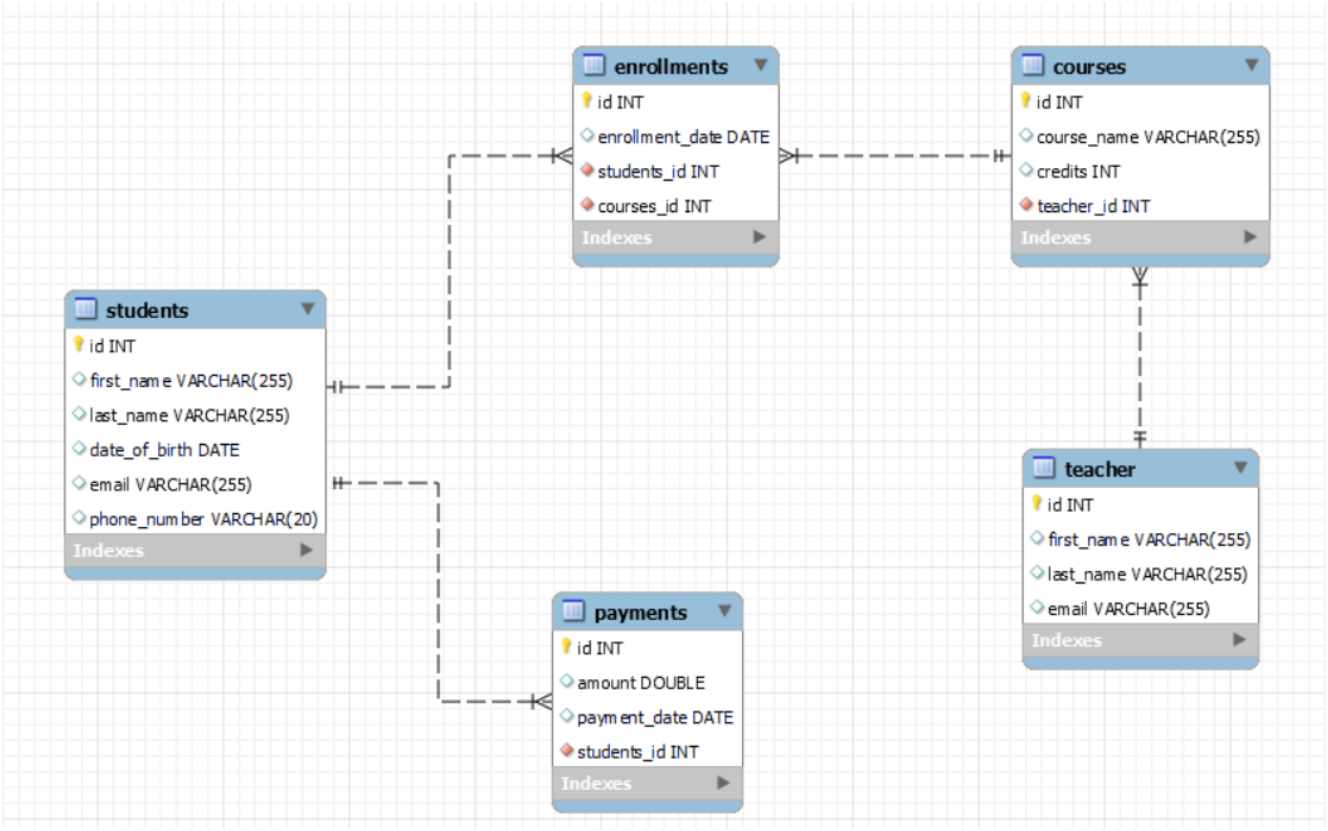
ASSIGNMENT 4 – STUDENT INFORMATION SYSTEM

ER DIAGRAM:



Queries:

```
-- MySQL Workbench Forward Engineering
-- -----------------------------------------------------
-- Schema student_info_system
-- -----------------------------------------------------
-- -----------------------------------------------------
-- Schema student_info_system
-- -----------------------------------------------------

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

-- -----------------------------------------------------
-- Table `student_info_system`.`students`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `student_info_system`.`students` (
  `id` INT NOT NULL,
  `first_name` VARCHAR(255) NULL,
  `last_name` VARCHAR(255) NULL,
  `date_of_birth` DATE NULL,
  `email` VARCHAR(255) NULL,
  `phone_number` VARCHAR(20) NULL,
```

```
  PRIMARY KEY (`id`))
ENGINE = InnoDB;

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

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

-- -----------------------------------------------------
-- Table `student_info_system`.`enrollments`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `student_info_system`.`enrollments` (
  `id` INT NOT NULL,
  `enrollment_date` DATE NULL,
  `students_id` INT NOT NULL,
  `courses_id` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_enrollments_students_idx` (`students_id` ASC) ,
  INDEX `fk_enrollments_courses1_idx` (`courses_id` ASC) ,
  CONSTRAINT `fk_enrollments_students`
    FOREIGN KEY (`students_id`)
    REFERENCES `student_info_system`.`students` (`id`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_enrollments_courses1`
    FOREIGN KEY (`courses_id`)
    REFERENCES `student_info_system`.`courses` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `student_info_system`.`payments`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `student_info_system`.`payments` (
  `id` INT NOT NULL,
  `amount` DOUBLE NULL,
  `payment_date` DATE NULL,
  `students_id` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_payments_students1_idx` (`students_id` ASC) ,
  CONSTRAINT `fk_payments_students1`
    FOREIGN KEY (`students_id`)
    REFERENCES `student_info_system`.`students` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- insertion of values

insert into students (id, first_name, last_name, date_of_birth, email, phone_number) values
(1, 'Harry', 'Potter', '1980-07-31', 'harry.potter@email.com', '1234567890'),
(2, 'Hermione', 'Granger', '1979-09-19', 'hermione.granger@email.com', '9876543210'),
(3, 'Ronald', 'Weasley', '1980-03-01', 'ron.weasley@email.com', '5551234567'),
(4, 'Ginny', 'Weasley', '1981-08-11', 'ginny.weasley@email.com', '4447890123'),
(5, 'Luna', 'Lovegood', '1981-02-13', 'luna.lovegood@email.com', '7774561230'),
(6, 'Neville', 'Longbottom', '1980-07-30', 'neville.longbottom@email.com', '3339876542'),
(7, 'Draco', 'Malfoy', '1980-06-05', 'draco.malfoy@email.com', '1112223333'),
(8, 'Lavender', 'Brown', '1981-01-21', 'lavender.brown@email.com', '9990001111'),
(9, 'Seamus', 'Finnigan', '1981-11-27', 'seamus.finnigan@email.com', '2223334444'),
(10, 'Cho', 'Chang', '1980-04-10', 'cho.chang@email.com', '6667778888');

insert into teacher (id, first_name, last_name, email) values
(101, 'Albus', 'Dumbledore', 'albus.dumbledore@email.com'),
(102, 'Minerva', 'McGonagall', 'minerva.mcgonagall@email.com'),
(103, 'Severus', 'Snape', 'severus.snape@email.com'),
(104, 'Sybill', 'Trelawney', 'sybill.trelawney@email.com'),
(105, 'Remus', 'Lupin', 'remus.lupin@email.com'),
(106, 'Gilderoy', 'Lockhart', 'gilderoy.lockhart@email.com'),
```

```
(107, 'Filius', 'Flitwick', 'filius.flitwick@email.com'),
(108, 'Pomona', 'Sprout', 'pomona.sprout@email.com'),
(109, 'Rubeus', 'Hagrid', 'rubeus.hagrid@email.com'),
(110, 'Dolores', 'Umbridge', 'dolores.umbridge@email.com');

insert into courses (id, course_name, credits, teacher_id) values
(1001, 'Data Structures', 3, 103),
(1002, 'Databases', 4, 102),
(1003, 'Programming in c', 3, 108),
(1004, 'Programming in java', 4, 105),
(1005, 'Programming in python', 3, 107),
(1006, 'Cyber security', 2, 109),
(1007, 'Artificial Intelligence', 2, 104),
(1008, 'Machine Learning', 3, 101),
(1009, 'Devops', 2, 110),
(1010, 'Software Engineering', 3, 106);

insert into enrollments (id, enrollment_date, students_id, courses_id) values
(11, '2023-09-01', 1, 1004),
(22, '2023-05-15', 2, 1007),
(33, '2023-08-20', 3, 1001),
(44, '2024-02-10', 4, 1005),
(55, '2023-12-05', 1, 1001),
(66, '2023-04-30', 6, 1003),
(77, '2023-10-12', 7, 1006),
(88, '2023-07-18', 3, 1002),
(99, '2023-11-25', 9, 1007),
(1111, '2024-03-08', 10, 1010);

insert into payments (id, amount, payment_date, students_id) values
(201, 500.00, '2023-09-15', 1),
(202, 600.00, '2023-06-20', 2),
(203, 450.00, '2023-09-25', 3),
(204, 700.00, '2024-03-15', 4),
(205, 550.00, '2023-12-10', 1),
(206, 800.00, '2023-05-02', 6),
(207, 350.00, '2023-11-15', 7),
(208, 900.00, '2023-08-22', 3),
(209, 600.00, '2023-12-01', 9),
(210, 750.00, '2024-03-20', 10);

-- Task 2: .....
/* Q1. 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 (id, first_name, last_name, date_of_birth, email, phone_number) values
(121, 'John', 'Doe', '1995-08-15', 'john.doe@example.com', '1234567890');

-- Q2. 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 (id, enrollment_date, students_id, courses_id) values
(31, '2023-09-02', 4, 1006);
select * from students;

-- Q3. Update the email address of a specific teacher in the "Teacher" table. Choose any teacher and modify their email address.
update teacher
    set email='albus101@gmail.com'
    where id=101;

-- Q4. 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_id=2 and courses_id=1007;

-- Q5. 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=104
    where course_name='Databases';

-- Q6. 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 payments
    where students_id=1;
delete from enrollments
    where students_id=1;
delete from students
    where id=1;

-- Q7. 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=1000
    where id=202;
```

-- Task 3:....

-- Q1. 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.first_name as Name, sum(p.amount) as Total_payment
    from payments p join students s
    on s.id=p.students_id
    group by p.students_id;
```

-- Q2. 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.id) as count_of_students
    from courses c join enrollments e
    on c.id=e.courses_id
    group by c.course_name;
```

-- Q3. 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
    from students s join enrollments e
    on s.id=e.students_id
    where e.students_id is null;
```

-- Q4. 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.id=e.students_id join courses c on
    c.id=e.courses_id;
```

-- Q5. 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 name_of_teacher, c.course_name
    from teacher t join courses c
    on t.id=c.teacher_id;
```

-- Q6. 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 concat(s.first_name,' ',s.last_name) as name_of_student,
    e.enrollment_date, c.course_name
    from students s join enrollments e
    on s.id=e.students_id join courses c
    on c.id=e.courses_id
    where c.course_name='Databases';
```

```
-- Q7. 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 name_of_student
    from students s left join payments p
    on s.id=p.students_id
    where p.students_id is null;

-- Q8. 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.id=e.courses_id
    where e.courses_id is null;

-- Q9. 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.first_name as student_name
    from students s join enrollments e
    on s.id=e.students_id
    group by e.students_id
    having count(courses_id) >1;

-- Q10. 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 as name_of_teacher
    from teacher t left join courses c
    on t.id=c.teacher_id
    where c.teacher_id is null;

-- Task 4: .....
-- Q2. 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.first_name, p.amount
    from students s , payments p
    where p.amount=(select max(amount) from payments) and s.id=p.students_id;

-- Q3. 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_name, enrollment_count
from (select c.course_name, count(e.students_id) as enrollment_count
      from courses c left join enrollments e on c.id = e.courses_id
      group by c.id, c.course_name ) as course_enrollments
where enrollment_count = ( select max(enrollment_count)
      from ( select courses_id, count(distinct students_id) as enrollment_count from enrollments
                  group by courses_id ) as max_enrollments );
```

```
-- Q4. 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 as first_name,
   t.last_name as last_name,
   sum(p.amount) as total_payments
from teacher t
join courses c on t.id = c.teacher_id
join enrollments e ON c.id = e.courses_id
join payments p ON e.students_id = p.students_id
group by t.id;

-- Q5. 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.first_name as student_name
    from students s join enrollments e
    on s.id=e.students_id join courses c
    on c.id=e.courses_id
    group by e.students_id
    having count(distinct c.id) = (select count(id) from courses);

-- Q6. Retrieve the names of teachers who have not been assigned to any courses. Use subqueries to find teachers with no course assignments.
select concat(t.first_name,' ',t.last_name) as teacher_name
    from teacher t
    where t.id not in (select teacher_id from courses);

-- Q7. 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 avg_age_of_students
    from students;

-- Q8. Identify courses with no enrollments. Use subqueries to find courses without enrollment records.
select c.course_name
    from courses c left join enrollments e
    on c.id=e.courses_id
    where e.courses_id is null;

-- Q9. 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, c.course_name,sum(p.amount) as total_payment
    from students s join payments p
    on s.id=p.students_id
    group by p.students_id;
```

```
-- Q10. 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.first_name , count(p.id) as no_of_payments
    from students s join payments p
    on s.id = p.students_id
    group by p.students_id
    having no_of_payments >1;

-- Q11. 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.first_name, sum(p.amount) as total_payment
    from students s join payments p
    on s.id=p.students_id
    group by p.students_id;

-- Q12. 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 count_of_students
    from courses c join enrollments e
    on c.id = e.courses_id
    group by e.students_id;

-- Q13. 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.first_name, avg(p.amount) as avg_amount
    from payments p join students s
    on s.id=p.students_id
    group by p.students_id;
```