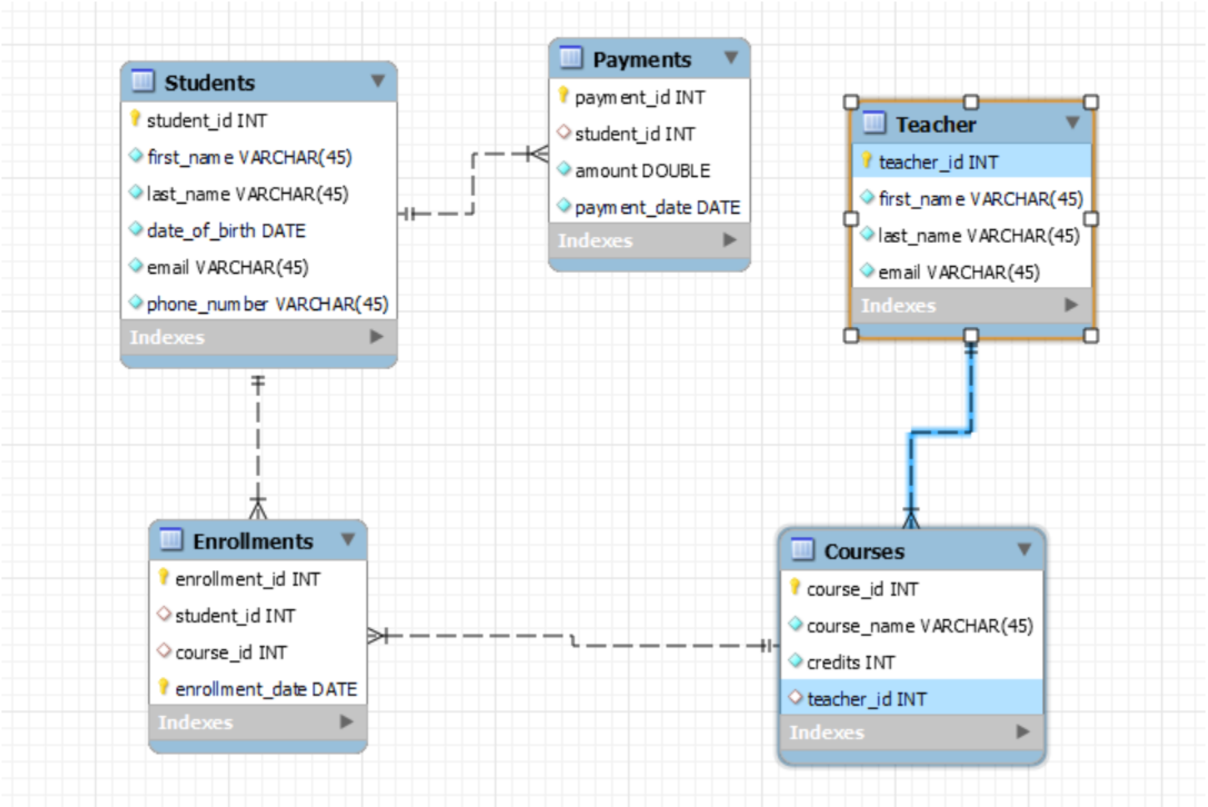

Students
student_id INT
first_name VARCHAR(45)
last_name VARCHAR(45)
date_of_birth DATE
email VARCHAR(45)
phone_number VARCHAR(45)

Payments
payment_id INT
student_id INT
amount DOUBLE
payment_date DATE

Teacher
teacher_id INT
first_name VARCHAR(45)
last_name VARCHAR(45)
email VARCHAR(45)

Enrollments
enrollment_id INT
student_id INT
course_id INT
enrollment_date DATE

Courses
course_id INT
course_name VARCHAR(45)
credits INT
teacher_id INT

```
-- MySQL Script generated by MySQL Workbench

-- Tue Mar  5 08:12:54 2024

-- Model: New Model    Version: 1.0

-- MySQL Workbench Forward Engineering


SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;

SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;

SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ER
ROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';


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

-- Schema StudentInformationSystem

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



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

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


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

```
-- -----------------------------------------------------
-- Table `StudentInformationSystem`.`Courses`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `StudentInformationSystem`.`Courses` (
  `course_id` INT NOT NULL AUTO_INCREMENT,
  `course_name` VARCHAR(45) NOT NULL,
  `credits` INT NOT NULL,
  `teacher_id` INT NULL,
  PRIMARY KEY (`course_id`),
  INDEX `teacher_id_idx` (`teacher_id` ASC),
  CONSTRAINT `teacher_id`
    FOREIGN KEY (`teacher_id`)
    REFERENCES `StudentInformationSystem`.`Teacher` (`teacher_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `StudentInformationSystem`.`Enrollments`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `StudentInformationSystem`.`Enrollments` (
  `enrollment_id` INT NOT NULL AUTO_INCREMENT,
  `student_id` INT NULL,
  `course_id` INT NULL,
  `enrollment_date` DATE NOT NULL,
  PRIMARY KEY (`enrollment_id`, `enrollment_date`),
  INDEX `student_id_idx` (`student_id` ASC),
  INDEX `course_id_idx` (`course_id` ASC),
  CONSTRAINT `student_id`
    FOREIGN KEY (`student_id`)
    REFERENCES `StudentInformationSystem`.`Students` (`student_id`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `course_id`
    FOREIGN KEY (`course_id`)
    REFERENCES `StudentInformationSystem`.`Courses` (`course_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `StudentInformationSystem`.`Payments`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `StudentInformationSystem`.`Payments` (
  `payment_id` INT NOT NULL AUTO_INCREMENT,
  `student_id` INT NULL,
  `amount` DOUBLE NOT NULL,
  `payment_date` DATE NOT NULL,
  PRIMARY KEY (`payment_id`),
  INDEX `student_id_paid_idx` (`student_id` ASC),
  CONSTRAINT `student_id_paid`
    FOREIGN KEY (`student_id`)
    REFERENCES `StudentInformationSystem`.`Students` (`student_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

describe students;

insert into Students(first_name,last_name,date_of_birth,email,phone_number)
 values('kane','williamson','2002-03-01','kane@gmail.com','9704070043'),
 ('virat','kohli','2002-05-10','virat@gmail.com','9966728862'),
 ('mahendra singh','dhoni','2002-03-01','dhoni@gmail.com','9704070043'),
 ('rohit','sharma','2004-03-01','rohit@gmail.com','8245076345'),
 ('peter','williamson','2002-03-01','peter@gmail.com','8245076345');

 select * from students;


 describe teacher;

 insert into teacher(first_name,last_name,email)
 values('mike','andrason','mike@gmail.com'),
 ('jack','sparrow','jack@gmail.com'),
 ('captian','america','jacksparrow@gmail.com'),
 ('iron','man','ironman@gmail.com'),
 ('black','panthar','blackpanthar@gmail.com');

 select * from teacher;

 describe courses;

 insert into courses(course_name,credits,teacher_id)
 values('AI-ML',10,1),
 ('data science',15,1),
 ('java fullstack',10,4),
 ('blockchain',5,3),
 ('networking',8,2);

  select * from courses;

 describe enrollments;

 insert into enrollments(student_id,course_id,enrollment_date)
 values(1,2,'2024-03-01'),
 (2,1,'2024-03-05'),
 (1,3,'2024-03-07'),
 (3,4,'2024-02-01'),
 (4,5,'2024-01-10');

 insert into enrollments(student_id,course_id,enrollment_date)
 values(2,2,'2024-03-01'),
 (5,1,'2024-03-05'),
 (2,3,'2024-03-07'),
 (1,4,'2024-02-01'),
 (3,5,'2024-01-10');

 select * from enrollments;

 describe payments;

 insert into payments(student_id,amount,payment_date)
 values(1,5000,'2024-03-01'),
 (2,4000,'2024-03-10'),
 (1,7000,'2024-03-15'),
 (3,5000,'2024-03-01'),
 (4,10000,'2024-02-01');
```

```sql
 select * from payments;


-- 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.


    select * from students;
    select * from courses;
    select * from enrollments;


    insert into enrollments(student_id,course_id,enrollment_date)
 values(16,5,'2024-03-01');


-- 3. Update the email address of a specific teacher in the "Teacher" table. Choose any teacher and modify their email address.


  select * from teacher;
  update teacher set email='mikeandrason@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 student_id=1 and course_id=2;
```

-- 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=5 where course_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 students where student_id=1; -- if you give onaction=cascade in the referential integrity then the child records automatically deleted
```

-- 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=80 where payment_id=2;
```

-- 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 p.student_id,p.payment_id,p.amount,p.payment_date
from students s join payments p
on s.student_id=p.student_id
where s.student_id=1;
```

```sql
-- 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,count(e.student_id) as num_of_students
from enrollments e right join courses c
on c.course_id=e.course_id
group by c.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.student_id,s.first_name
from enrollments e right join students s
on s.student_id=e.student_id
group by s.student_id
having count(e.student_id)=0;

-- 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.student_id,s.first_name,s.last_name,c.course_name
from enrollments e  join students s
on s.student_id=e.student_id
join  courses c
on c.course_id=e.course_id
order by s.student_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.
```

```sql
select t.teacher_id,t.first_name,c.course_name
from teacher t join courses c
on t.teacher_id=c.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.student_id,s.first_name,e.enrollment_date
from enrollments e  join students s
on s.student_id=e.student_id
join courses c
on e.course_id=c.course_id
where c.course_id=1
order by s.student_id;

-- 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.student_id,s.first_name
from students s left join payments p
on s.student_id=p.student_id
group by s.student_id
having count(p.payment_id)=0;

-- 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_id,c.course_name
from courses c left join enrollments e
```

```sql
on c.course_id=e.course_id

group by c.course_id

having count(e.enrollment_id)=0;


-- 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 student_id

from  enrollments

group by student_id

having count(student_id)>1

order by 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.teacher_id,t.first_name ,t.last_name

from teacher t left join courses c

on t.teacher_id=c.teacher_id

group by t.teacher_id

having count(c.course_id)=0;


-- 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_id)

from enrollments

where student_id in (select student_id from students)

group by course_id;
```

```sql
-- 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 student_id,first_name,last_name
from students where student_id in (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,course_name from courses
where course_id in (select course_id from enrollments group by course_id
                    having count(course_id)=(select count(course_id) from enrollments group by course_id order by count(course_id) 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 c.course_name,t.first_name,t.last_name ,p.amount
from courses c join teacher t
on c.teacher_id=t.teacher_id
join enrollments e
on e.course_id=c.course_id
join payments p
on e.student_id=p.student_id
group by c.course_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.
```

```sql
select student_id,first_name,last_name
from students
where (select count(student_id) from enrollments)=(select count(*) 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 teacher_id,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 ( select sum(year(now())-year(date_of_birth)) from students)/(select count(*) from students) as average_age
from students limit 0,1;

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

select course_id,course_name
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.

select student_id,sum(amount) as total_payment
from payments
```

```sql
where student_id in (select student_id from students)

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(student_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.student_id,s.first_name,s.last_name,sum(amount) as total_amount

from students s join payments p

on s.student_id=p.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_id,c.course_name,count(e.student_id) as count

from courses c join enrollments e

on c.course_id=e.course_id

group by c.course_id;
```

```sql
-- 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_amount
from students s join payments p
on s.student_id=p.student_id
group by s.student_id;
```