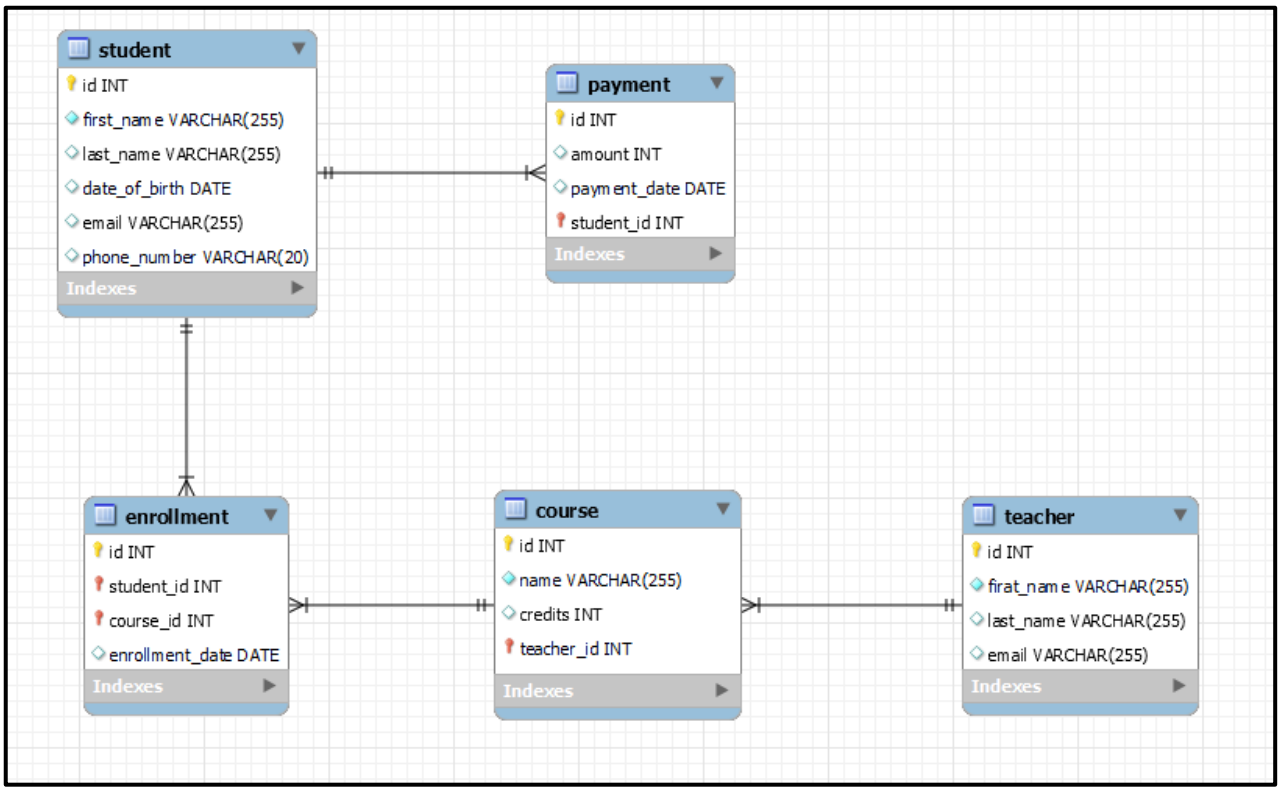
ASSIGNMENT NO : 2

Student Information System (SIS)

ER DIAGRAM:



Task 1. Database Design:

```
-- MySQL Workbench Forward Engineering


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

-- Schema student_db

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


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

-- Schema student_db

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

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

USE `student_db` ;
```

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


-- -----------------------------------------------------
-- Table `student_db`.`course`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `student_db`.`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_teacher_idx` (`teacher_id` ASC) ,
  CONSTRAINT `fk_course_teacher`
    FOREIGN KEY (`teacher_id`)
    REFERENCES `student_db`.`teacher` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `student_db`.`payment`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `student_db`.`payment` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `amount` INT 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 `student_db`.`student` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `student_db`.`enrollment`
```

```
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `student_db`.`enrollment` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `student_id` INT NOT NULL,
  `course_id` INT NOT NULL,
  `enrollment_date` DATE NULL,
  PRIMARY KEY (`id`, `student_id`, `course_id`),
  INDEX `fk_student_has_course_course1_idx` (`course_id` ASC) ,
  INDEX `fk_student_has_course_student1_idx` (`student_id` ASC) ,
  CONSTRAINT `fk_student_has_course_student1`
    FOREIGN KEY (`student_id`)
    REFERENCES `student_db`.`student` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_student_has_course_course1`
    FOREIGN KEY (`course_id`)
    REFERENCES `student_db`.`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
('dhana','lakshmi','2003-03-
30','dhana@gmail.com','9360805403'),
('dhivya','lakshmi','2002-08-
25','dhivya@gmail.com','9786205333'),
('dhivya','bharathy','2002-08-
25','bharathy@gmail.com','9787333355'),
('bhavana','ranganathan','2003-06-
12','bhavana@gmail.com','8220681348'),
```

```
('pradheesha','sivakumar','2001-07-
12','pradheesha@gmail.com','8072607205'),

('kalis','dharani','1998-08-
11','kalis@gmail.com','9443500160'),

('devibala','ponnusamy','1997-12-
12','devibala@gmail.com','8610248302'),

('dheepika','chellamuthu','2004-06-
14','dheepika@gmail.com','9787333394'),

('roshini','mani','2006-09-
20','roshini@gmail.com','9787333394'),

('hema','kannan','2005-04-11','hema@gmail.com','9787333364');
```

-- teacher insertion

```
ALTER TABLE teacher

change COLUMN firat_name first_name varchar(255);


insert into teacher(first_name,last_name,email) values

('gaja','lakshmi','gaja@gmail.com'),

('ashok','selvam','ashok@gmail.com'),

('agila','vidya','agila@gmail.com'),

('jansi','vidya','jansi@gmail.com'),

('jasmine','mary','jasmine@gmail.com'),

('josphine','mary','josphine@gmail.com'),

('selvam','suzainathan','selvam@gmail.com'),

('hema','latha','hema@gmail.com'),

('brito','lourdusamy','brito@gmail.com'),

('jalaja','marium','jalaja@gmail.com');
```

-- course insertion

```
insert into course(name,credits,teacher_id) values

('java',50,1),

('c',30,2),
```

```
('python',40,3),
('c#',20,4),
('javascript',35,5),
('ruby',15,6),
('nodejs',60,7),
('react',65,8),
('html',40,9),
('css',10,10);

-- payment insertion
insert into payment (amount,payment_date,student_id) values
(10000,'2023-03-30',1),
(1000,'2023-03-30',2),
(1000,'2023-03-30',3),
(1500,'2022-12-31',4),
(2000,'2023-11-29',5),
(1800,'2023-11-27',6),
(4000,'2023-12-30',7),
(1600,'2021-03-31',8),
(1600,'2024-03-30',9),
(10000,'2023-11-28',1);

-- enrollment insertion
insert into enrollment (student_id,course_id,enrollment_date)
values
(1,7,'2023-03-30'),
(2,2,'2023-03-30'),
(3,2,'2023-03-30'),
(1,8,'2023-11-28'),
(4,4,'2022-12-31'),
(5,5,'2023-11-28'),
```

```
(6,9,'2024-01-31'),
(7,3,'2024-01-01'),
(8,10,'2021-03-30'),
(9,10,'2024-03-03');
```

-- 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: 1995-08-15
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='lakshmi@gmail.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=8 where name='css';


/* 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.*/


insert into enrollment (student_id,course_id,enrollment_date)
values

(2,4,'2023-05-30'),(3,4,'2023-05-30');


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;
```