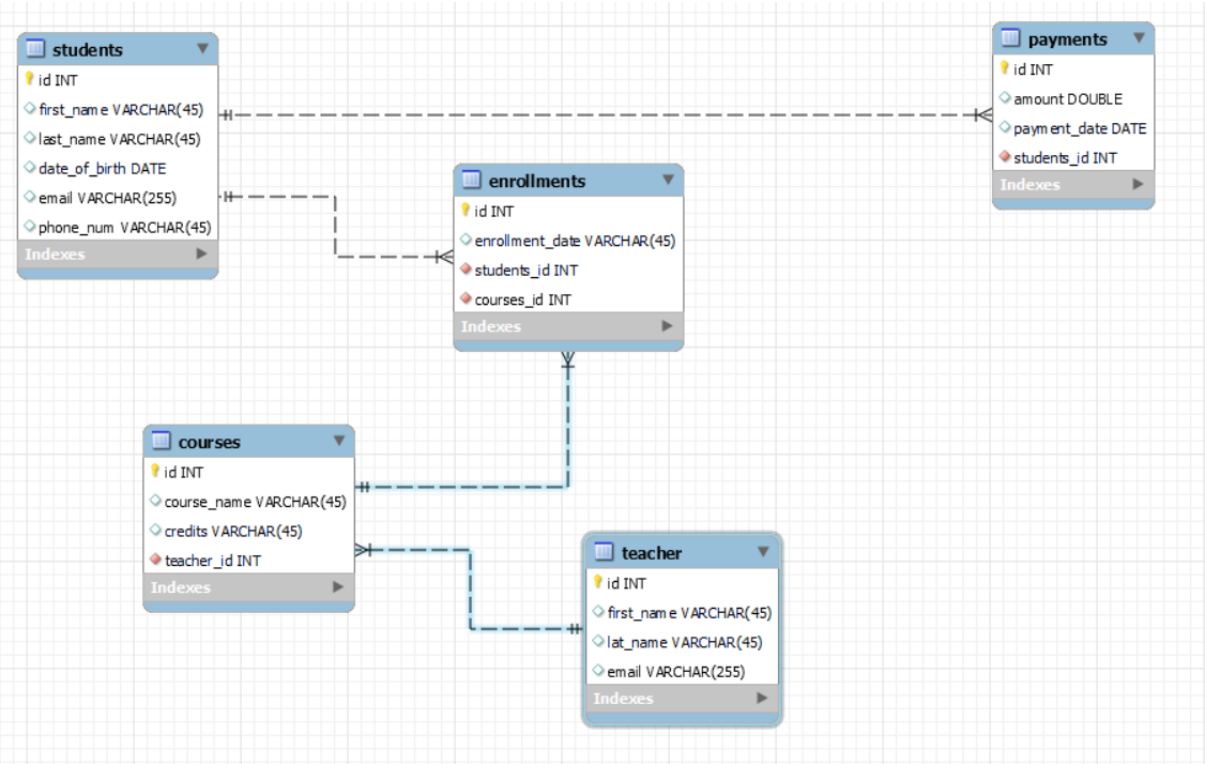
Student Information System



-- ---

-- Schema student_information_system

-- ---

-- ---

-- Schema student_information_system

-- ---

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

USE `student_information_system` ;

```
-- -----------------------------------------------------
-- Table `student_information_system`.`students`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `student_information_system`.`students` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `first_name` VARCHAR(45) NULL,
  `last_name` VARCHAR(45) NULL,
  `date_of_birth` DATE NULL,
  `email` VARCHAR(255) NULL,
  `phone_num` VARCHAR(45) NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `student_information_system`.`teacher`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `student_information_system`.`teacher` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `first_name` VARCHAR(45) NULL,
  `lat_name` VARCHAR(45) NULL,
  `email` VARCHAR(255) NULL,
  PRIMARY KEY (`id`))
```

```
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `student_information_system`.`courses`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `student_information_system`.`courses` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `course_name` VARCHAR(45) NULL,
  `credits` VARCHAR(45) NULL,
  `teacher_id` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_courses_teacher_idx` (`teacher_id` ASC) ,
  CONSTRAINT `fk_courses_teacher`
    FOREIGN KEY (`teacher_id`)
    REFERENCES `student_information_system`.`teacher` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `student_information_system`.`enrollments`
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `student_information_system`.`enrollments` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `enrollment_date` VARCHAR(45) NULL,
  `students_id` INT NOT NULL,
  `courses_id` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_enrollments_students1_idx` (`students_id` ASC) ,
  INDEX `fk_enrollments_courses1_idx` (`courses_id` ASC) ,
  CONSTRAINT `fk_enrollments_students1`
    FOREIGN KEY (`students_id`)
    REFERENCES `student_information_system`.`students` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_enrollments_courses1`
    FOREIGN KEY (`courses_id`)
    REFERENCES `student_information_system`.`courses` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `student_information_system`.`payments`
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `student_information_system`.`payments` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `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_information_system`.`students` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
--Insertions
insert into students(first_name,last_name,date_of_birth,email,phone_num) values
('nivetha','R','12.11.2002','nive@gmail.com','98765'),
('nirupama','S','10.1.2002','niru@gmail.com','12345'),
('hari','A','7.8.2002','hari@gmail.com','10002'),
('anu','S','3.12.2002','anu@gmail.com','13579'),
('nithi','B','25.5.2003','nithi@gmail.com','24680'),
('sangeetha','S','15.11.2002','sangee@gmail.com','50000');
select * from students;
+----+------------+-----------+---------------+---------------------+------------+
| id | first_name | last_name | date_of_birth | email               | phone_num  |
```

```
+----+------------+-----------+---------------+---------------------+------------+
|  1 | nivetha    | R         | 2012-11-20    | nive@gmail.com      | 98765      |
|  2 | nirupama   | S         | 2010-01-20    | niru@gmail.com      | 12345      |
|  3 | hari       | A         | 2007-08-20    | hari@gmail.com      | 10002      |
|  4 | anu        | S         | 2003-12-20    | anu@gmail.com       | 13579      |
|  5 | nithi      | B         | 2025-05-20    | nithi@gmail.com     | 24680      |
|  6 | sangeetha  | S         | 2015-11-20    | sangee@gmail.com    | 50000      |
```

```
insert into teacher(first_name,lat_name,email) values
('jayanthi','S','jainthi@gmail.com'),
('shanthi','M','shanthi@gmail.com'),
('janai','R','janani@gmail.com'),
('sajeetha','B','sajee@gmail.com'),
('priya','E','priya@gmail.com');
Select * from teacher;
+----+------------+----------+------------------+
| id | first_name | lat_name | email            |
+----+------------+----------+------------------+
|  1 | jayanthi   | S        | jainthi@gmail.com |
|  2 | shanthi    | M        | shanthi@gmail.com |
|  3 | janai      | R        | janani@gmail.com  |
|  4 | sajeetha   | B        | sajee@gmail.com   |
|  5 | priya      | E        | priya@gmail.com   |
+----+------------+----------+------------------+
```

```
insert into courses(course_name,credits,teacher_id) values
('java','A','1'),
('python','B','2'),
('c#','C','3'),
('jva script','D','4'),
('SQL','E','5');
Select * from courses;
+----+-------------+---------+------------+
| id | course_name | credits | teacher_id |
+----+-------------+---------+------------+
|  1 | java        | A       |          1 |
|  2 | python      | B       |          2 |
|  3 | c#          | C       |          3 |
|  4 | jva script  | D       |          4 |
|  5 | SQL         | E       |          5 |
+----+-------------+---------+------------+
```

```
INSERT INTO enrollments(enrollment_date, students_id, courses_id)
VALUES
('2024-01-01', 1, 1),
('2024-01-02', 2, 2),
('2024-01-03', 3, 3),
('2024-01-04', 4, 4),
('2024-01-05', 5, 5);
```

Select * from enrollments;

```
+----+-----------------+-------------+------------+
| id | enrollment_date | students_id | courses_id |
+----+-----------------+-------------+------------+
|  1 | 2024-01-01      |           1 |          1 |
|  2 | 2024-01-02      |           2 |          2 |
|  3 | 2024-01-03      |           3 |          3 |
|  4 | 2024-01-04      |           4 |          4 |
|  5 | 2024-01-05      |           5 |          5 |
+----+-----------------+-------------+------------+
```

INSERT INTO payments(id, amount, payment_date, students_id)

VALUES

(1, 1000, '2024-01-01', 1),

(2, 1500, '2024-01-02', 2),

(3, 800, '2024-01-03', 3),

(4, 1200, '2024-01-04', 4),

(5, 1600, '2024-01-05', 5);

Select * from payments;

```
+----+--------+--------------+-------------+
| id | amount | payment_date | students_id |
+----+--------+--------------+-------------+
|  1 |   1000 | 2024-01-01   |           1 |
|  2 |   1500 | 2024-01-02   |           2 |
```

```
|  3 |    800 | 2024-01-03   |          3 |
|  4 |   1200 | 2024-01-04   |          4 |
|  5 |   1600 | 2024-01-05   |          5 |
+----+--------+--------------+------------+
```

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: 123456789

```
insert into students (first_name, last_name, date_of_birth, email, phone_num)
values ('john', 'doe', '1995-08-15', 'john.doe@example.com', '1234567890');
+----+------------+-----------+---------------+---------------------+------------+
| id | first_name | last_name | date_of_birth | email               | phone_num  |
+----+------------+-----------+---------------+---------------------+------------+
|  1 | nivetha    | r         | 2012-11-20    | nive@gmail.com      | 98765      |
|  2 | nirupama   | s         | 2010-01-20    | niru@gmail.com      | 12345      |
|  3 | hari       | a         | 2007-08-20    | hari@gmail.com      | 10002      |
|  4 | anu        | s         | 2003-12-20    | anu@gmail.com       | 13579      |
|  5 | nithi      | b         | 2025-05-20    | nithi@gmail.com     | 24680      |
|  6 | sangeetha  | s         | 2015-11-20    | sangee@gmail.com    | 50000      |
|  7 | 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.

---

insert into enrollments (enrollment_date, students_id, courses_id)

values ('2024-03-06', 3, 3);

select * from enrollments;

+----+----------+----------+----------+

| id | enrollment_date | students_id | courses_id |

+----+----------+----------+----------+

| 1 | 2024-01-01 | 1 | 1 |

| 2 | 2024-01-02 | 2 | 2 |

| 3 | 2024-01-03 | 3 | 3 |

| 4 | 2024-01-04 | 4 | 4 |

| 5 | 2024-01-05 | 5 | 5 |

| 6 | 2024-03-06 | 3 | 3 |

---

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 = 'teacher@gmail.com'

where id = 4;

select * from teacher;

+----+----------+----------+----------+

| id | first_name | lat_name | email |

+----+----------+----------+----------+

| 1 | jayanthi | s | jainthi@gmail.com |

| 2 | shanthi | m | shanthi@gmail.com |

| 3 | janai | r | janani@gmail.com |

| 4 | sajeetha | b | teacher@gmail.com |

| 5 | priya | e | priya@gmail.com |

+----+----------+----------+----------+

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 students_id = 1 and courses_id = 1;

+----+----------+----------+----------+

| id | enrollment_date | students_id | courses_id |

+----+----------+----------+----------+

| 2 | 2024-01-02 | 2 | 2 |

| 3 | 2024-01-03 | 3 | 3 |

| 4 | 2024-01-04 | 4 | 4 |

| 5 | 2024-01-05 | 5 | 5 |

| 6 | 2024-03-06 | 3 | 3 |

+----------+----------+-----+----------+

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 = 1

where id = 2;

+----+----------+---------+----------+

| id | course_name | credits | teacher_id |

```
+----+-------------+---------+------------+
|  1 | java        | a       |          1 |
|  2 | python      | b       |          1 |
|  3 | c#          | c       |          3 |
|  4 | jva script  | d       |          4 |
|  5 | sql         | e       |          5 |
+----+-------------+---------+------------+
```

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.

```
set student_id = 1;
delete from enrollments where students_id = 1;
delete from students where students_id=1;
+----+-----------------+-------------+------------+
| id | enrollment_date | students_id | courses_id |
+----+-----------------+-------------+------------+
|  4 | 2024-01-04      |           4 |          4 |
|  5 | 2024-01-05      |           5 |          5 |
+----+-----------------+-------------+------------+
+----+------------+-----------+---------------+---------------------+------------+
| id | first_name | last_name | date_of_birth | email               | phone_num  |
+----+------------+-----------+---------------+---------------------+------------+
|  2 | nirupama   | s         | 2010-01-20    | niru@gmail.com      | 12345      |
|  3 | hari       | a         | 2007-08-20    | hari@gmail.com      | 10002      |
|  4 | anu        | s         | 2003-12-20    | anu@gmail.com       | 13579      |
|  5 | nithi      | b         | 2025-05-20    | nithi@gmail.com     | 24680      |
```

| 6 | sangeetha | s | 2015-11-20 | sangee@gmail.com | 50000 |

| 7 | john | doe | 1995-08-15 | john.doe@example.com | 1234567890 |

+----+----------+----------+----------+----------+----------+

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 = 2000

where id = 1;

+----+--------+----------+----------+

| id | amount | payment_date | students_id |

+----+--------+----------+----------+

| 1 | 2000 | 2024-01-01 | 1 |

| 2 | 1500 | 2024-01-02 | 2 |

| 3 | 800 | 2024-01-03 | 3 |

| 4 | 1200 | 2024-01-04 | 4 |

| 5 | 1600 | 2024-01-05 | 5 |

+----+--------+----------+----------+

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 s.first_name, s.last_name, sum(p.amount) as total_payments

from students s

join payments p on s.id = p.students_id

where s.id = 3;

+----------+----------+----------+

| first_name | last_name | total_payments |

+----------+----------+----------+

| hari | a | 800 |

+----------+----------+----------+

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_name, count(e.students_id) as student_count
from courses c
left join enrollments e on c.id = e.courses_id
group by c.id;
```

Result Grid | Filter Rows:

course_name	student_count
java	0
python	0
c#	0
jva script	1
SQL	1

Result 5

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.first_name, s.last_name
from students s
left join enrollments e on s.id = e.students_id
where e.students_id is null;
```

Result Grid | Filter R

first_name	last_name
nirupama	S
hari	A
sangeetha	S
John	Doe

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.course_name
from students s
join enrollments e on s.id = e.students_id
```

join courses c on e.courses_id = c.id;

	first_name	last_name	course_name
▶	anu	S	jva script
	nithi	B	SQL

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.lat_name, c.course_name

from teacher t

join courses c on t.id = c.teacher_id

	first_name	lat_name	course_name
▶	jayanthi	S	java
	jayanthi	S	python
	janai	R	c#
	sajeetha	B	jva script
	priya	E	SQL

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" table

select s.first_name, s.last_name, e.enrollment_date

from students s

join enrollments e on s.id = e.students_id

join courses c on e.courses_id = c.id

where c.course_name = 'sql';

	first_name	last_name	enrollment_date
▶	nithi	B	2024-01-05

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.first_name, s.last_name

from students s

left join payments p on s.id = p.students_id

where p.students_id is null;

Result Grid | Filter Rows:

first_name	last_name
sangeetha	S
John	Doe

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_name
from courses c
left join enrollments e on c.id = e.courses_id
where e.courses_id is null;
```

Result Grid

course_name
java
python
c#

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.first_name, s.last_name, count(e.courses_id) as course_count
from students s
join enrollments e on s.id = e.students_id
group by s.id
having count(e.courses_id) > 1;
```

first_name	last_name	course_count

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.first_name, t.lat_name
from teacher t
left join courses c on t.id = c.teacher_id
where c.teacher_id is null;
```

Result Grid | Filter Rows:

first_name	lat_name
shanthi	M

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 avg_students_per_course
from (
    select count(*) as student_count
    from enrollments
    group by courses_id
) as course_counts;
```

	avg_students_per_course
▶	1.0000

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.first_name, s.last_name
from students s
where s.id in (select students_id from payments where amount = (select
max(amount) from payments));
```

Result Grid | Filter Row

	first_name	last_name

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.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
having count(e.students_id) = (select max(enrollment_count) from (select
count(*) as enrollment_count from enrollments group by courses_id) as counts);
```

	course_name	enrollment_count
▶	jva script	1
	SQL	1

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

```
select t.first_name, t.lat_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;
```

Result Grid | Filter Rows:

	first_name	lat_name	total_payments
▶	sajeetha	B	1200
	priya	E	1600

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.first_name, s.last_name
from students s
where (select count(distinct courses_id) from enrollments where students_id =
s.id) = (select count(*) from courses);
```

	first_name	last_name

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.first_name, t.last_name
from teacher t
where t.id not in (select distinct teacher_id from courses);
```

	first_name	lat_name
▶	shanthi	M

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(datediff(curdate(), date_of_birth) / 365) as average_age
from students;

	average_age
▶	14.43105023

---

8. identify courses with no enrollments. use subqueries to find courses without enrollment records.

---

select c.course_name

from courses c

where c.id not in (select distinct courses_id from enrollments);

	course_name
▶	java
	python
	c#

---

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.course_name, sum(p.amount) as total_payments
from students s
join enrollments e on s.id = e.students_id
join courses c on e.courses_id = c.id
join payments p on e.students_id = p.students_id
group by s.id, c.id;

Result Grid | Filter Rows: | Export:

	first_name	last_name	course_name	total_payments
▶	anu	S	jva script	1200
	nithi	S	SQL	1600

nithi

---

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.first_name, s.last_name

from students s

where s.id in (select students_id from payments group by students_id having count(*) > 1);

	first_name	last_name

---

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.first_name, s.last_name, sum(p.amount) as total_payments

from students s

join payments p on s.id = p.students_id

group by s.id;

Result Grid | Filter Rows:

	first_name	last_name	total_payments
▶	nirupama	S	1500
	hari	A	800
	anu	S	1200
	nithi	B	1600

---

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_name, (select count(*) from enrollments where courses_id = c.id) as student_count

from courses c;

	course_name	student_count
▶	java	0
	python	0
	c#	0
	jva script	1
	SQL	1

---

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(amount) as average_payment_amount
from payments;

	average_payment_amount
▶	1420
