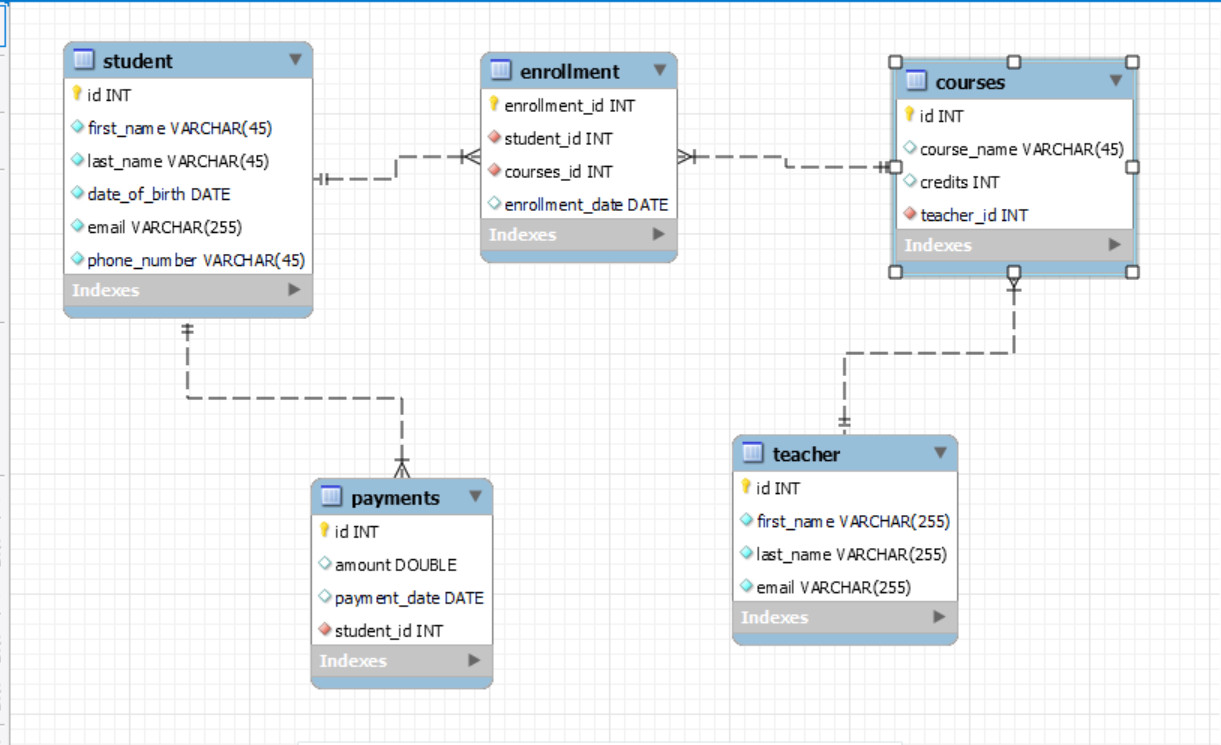
**Assignment 3 – Student Management System**

TASK 1- DATABASE DESIGN



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

USE `smsdb` ;

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

-- Table `smsdb`.`student`

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

CREATE TABLE IF NOT EXISTS `smsdb`.`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(255) NOT NULL,

`phone\_number` VARCHAR(45) NOT NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB;

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

-- Table `smsdb`.`teacher`

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

CREATE TABLE IF NOT EXISTS `smsdb`.`teacher` (

`id` INT NOT NULL AUTO\_INCREMENT,

`first\_name` VARCHAR(255) NOT NULL,

`last\_name` VARCHAR(255) NOT NULL,

`email` VARCHAR(255) NOT NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB;

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

-- Table `smsdb`.`courses`

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

CREATE TABLE IF NOT EXISTS `smsdb`.`courses` (

`id` INT NOT NULL AUTO\_INCREMENT,

`course\_name` VARCHAR(45) 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 `smsdb`.`teacher` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `smsdb`.`enrollment`

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

CREATE TABLE IF NOT EXISTS `smsdb`.`enrollment` (

`enrollment\_id` INT NOT NULL AUTO\_INCREMENT,

`student\_id` INT NOT NULL,

`courses\_id` INT NOT NULL,

`enrollment\_date` DATE NULL,

PRIMARY KEY (`enrollment\_id`),

INDEX `fk\_student\_has\_courses\_courses1\_idx` (`courses\_id` ASC) ,

INDEX `fk\_student\_has\_courses\_student\_idx` (`student\_id` ASC) ,

CONSTRAINT `fk\_student\_has\_courses\_student`

FOREIGN KEY (`student\_id`)

REFERENCES `smsdb`.`student` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_student\_has\_courses\_courses1`

FOREIGN KEY (`courses\_id`)

REFERENCES `smsdb`.`courses` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

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

-- Table `smsdb`.`payments`

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

CREATE TABLE IF NOT EXISTS `smsdb`.`payments` (

`id` INT NOT NULL AUTO\_INCREMENT,

`amount` DOUBLE NULL,

`payment\_date` DATE NULL,

`student\_id` INT NOT NULL,

PRIMARY KEY (`id`),

INDEX `fk\_payments\_student1\_idx` (`student\_id` ASC) ,

CONSTRAINT `fk\_payments\_student1`

FOREIGN KEY (`student\_id`)

REFERENCES `smsdb`.`student` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

--INSERATIONS—

insert into student (first\_name, last\_name, date\_of\_birth, email, phone\_number)

values

('harry', 'potter', '1980-07-31', 'harry@gmail.com', '45454545'),

('ronald', 'weasley', '1980-03-01', 'ron@gmail.com', '45454545'),

('hermione', 'granger', '1979-09-19', 'her@gmail.com', '45454545'),

('draco', 'malfoy', '1980-06-05', 'drac@gmail.com', '45454545'),

('ginni', 'weasley', '1981-08-11', 'ginni@gmail.com', '45454000');

insert into teacher (first\_name, last\_name, email)

values

('Severus', 'Snape', 'severus.snape@hogwarts.edu'),

('Minerva', 'McGonagall', 'minerva.mcgonagall@hogwarts.edu'),

('Filius', 'Flitwick', 'filius.flitwick@hogwarts.edu');

insert into courses (course\_name, credits, teacher\_id)

values

('Potions', 3, 1),

('Transfiguration', 4, 2),

('Charms', 3, 3);

insert into enrollment(student\_id,courses\_id,enrollment\_date)

values

(1, 1, '2000-04-08'),

(2, 2, '2000-04-09'),

(3, 3, '2000-04-10'),

(4, 1, '2000-04-11');

insert into payments(amount,payment\_date,student\_id)

values

(5000, '2000-04-08', 1),

(7500, '2000-04-09', 2),

(6000, '2000-04-10', 3),

(8000, '2000-04-11', 4);

TASK 2:

/\*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 = 'severus.snape@hogwarts.org'**

**where first\_name = 'Severus' AND last\_name = 'Snape';**

/\*

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 enrollment\_date='2000-04-11';**

/\*

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=3 where course\_name='charms';**

/\*

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 enrollment where student\_id=2;**

/\*

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=1000 where payments\_id=2;**

**TASK 3:**

/\*Write an SQL query to calculate the total payments made by a specific student.

\*/

**select s.first\_name, s.last\_name, p.amount from student s join payments p on s.id=p.student\_id where s.id=1;**

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

| first\_name | last\_name | amount |

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

| harry | potter | 5000 |

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

/\*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.student\_id) from courses c join enrollment e on c.id=e.courses\_id group by c.course\_name;**

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

| course\_name | count(e.student\_id) |

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

| Charms | 1 |

| Potions | 2 |

| Transfiguration | 1 |

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

/\*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 identifystudents without enrollments.\*/

**select s.first\_name from student s left join enrollment e on s.id=e.student\_id where s.id not in (select student\_id from enrollment);**

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

| first\_name |

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

| ginni |

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

/\*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" tableand the "Enrollments" and "Courses" tables.\*/

**select s.first\_name, s.last\_name, c.course\_name from student s join enrollment e on s.id=e.student\_id join courses c on c.id=e.courses\_id;**

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

| first\_name | last\_name | course\_name |

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

| harry | potter | Potions |

| ronald | weasley | Transfiguration |

| hermione | granger | Charms |

| draco | malfoy | Potions |

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

/\*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.last\_name, c.course\_name**

**from courses c join teacher t on**

**t.id=c.teacher\_id;**

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

| first\_name | last\_name | course\_name |

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

| Severus | Snape | Potions |

| Minerva | McGonagall | Transfiguration |

| Filius | Flitwick | Charms |

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

/\*6.Retrieve a list of students and their enrollment dates for a specific course. \*/

**select s.first\_name, e.enrollment\_date from student s join enrollment e on s.id=e.student\_id join courses c on c.id=e.courses\_id where c.course\_name='potions';**

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

| first\_name | enrollment\_date |

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

| harry | 2000-04-08 |

| draco | 2000-04-11 |

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

/\*7.Find the names of students who have not made any payments. \*/

**select s.first\_name, s.last\_name from student s left join payments p on s.id=p.student\_id where s.id not in (select student\_id from payments);**

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

| first\_name | last\_name |

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

| ginni | weasley |

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

/\*8.Write a query to identify courses that have no enrollments.\*/

**select c.course\_name from enrollment e join courses c on c.id = e.courses\_id where c.id not in (select enrollment\_id from enrollment);**

Empty set

/\*9.

Identify students who are enrolled in more than one course

\*/

**select s.first\_name,s.last\_name, count(e.enrollment\_id) as count from student s join enrollment e on s.id=e.student\_id group by s.id having count>1;**

Empty set

/\*

10. Find teachers who are not assigned to any courses. Use a LEFT JOIN between the "Teacher" table and the "Coürses" table and filter for teachers with NULL course assignments.

\*/

**select \* from teacher t left join courses c on t.id=c.teacher\_id where c.id is null;**

Empty set

TASK 4:

10. Find teachers who are not assigned to any courses. Use a LEFT JOIN between the "Teacher" table and the "Coürses" table and filter for teachers with NULL course assignments.

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 c.course\_name,avg(s.id) as average from student s join enrollment e on s.id = e.student\_id join courses c on c.id =e.courses\_id group by c.id;**

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

| course\_name | average |

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

| Potions | 2.5000 |

| Transfiguration | 2.0000 |

| Charms | 3.0000 |

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

1. 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.\*, p.amount from student s join payments p on s.id=p.student\_id order by p.amount desc limit 1;**

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

| id | first\_name | last\_name | date\_of\_birth | email | phone\_number | amount |

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

| 4 | draco | malfoy | 1980-06-05 | drac@gmail.com | 45454545 | 8000 |

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

1. 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.enrollment\_id) as number\_of\_courses from courses c join enrollment e on c.id=e.courses\_id group by c.id order by number\_of\_courses desc limit 1;**

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

| course\_name | number\_of\_courses |

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

| Potions | 2 |

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

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

**select t.\* from student s join enrollment e on s.id=e.student\_id join courses c on c.id = e.courses\_id join payments p on s.id=p.student\_id join teacher t on t.id=c.teacher\_id group by t.id;**

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

| id | first\_name | last\_name | email |

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

| 1 | Severus | Snape | severus.snape@hogwarts.org |

| 2 | Minerva | McGonagall | minerva.mcgonagall@hogwarts.edu |

| 3 | Filius | Flitwick | filius.flitwick@hogwarts.edu |

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

1. Identify students who are enrolled in all available courses. Use subqueries to compare a student's enroliments with the total number of courses.

**select s.\* from student s join enrollment e on s.id=e.student\_id join courses c on c.id = e.courses\_id where s.id =ALL(select id from courses);**

**Empty set (0.00 sec)**

1. Retrieve the names of teachers who have not been assigned to any courses. Use subqueries to find teachers with no course assignments.

**select \* from teacher where teacher.id not in (select teacher.id from courses c join teacher t on t.id=c.teacher\_id);**

Empty set (0.00 sec)

1. Calculate the average age of all students. Use subqueries to calculate the age of each student based on their date of birth.

Not possible without age

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

**select courses.id,course\_name from courses where courses.id not in (select courses\_id from enrollment);**

Empty set (0.00 sec)

1. 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.id,c.course\_name,sum(p.amount) from payments p join student s on s.id=p.student\_id join enrollment e on e.student\_id=s.id right join courses c on e.courses\_id =c.id group by e.enrollment\_id, c.course\_name;**

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

| id | course\_name | sum(p.amount) |

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

| 1 | Potions | 5000 |

| 2 | Transfiguration | 7500 |

| 3 | Charms | 6000 |

| 4 | Potions | 8000 |

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

1. 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.id , count(\*) from student s join payments p on s.id =p.student\_id group by s.id;**

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

| id | count(\*) |

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

| 1 | 1 |

| 2 | 1 |

| 3 | 1 |

| 4 | 1 |

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

1. 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.id, sum(p.amount) from student s left join payments p on s.id =p.student\_id group by s.id;**

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

| id | sum(p.amount) |

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

| 1 | 5000 |

| 2 | 7500 |

| 3 | 6000 |

| 4 | 8000 |

| 5 | NULL |

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

1. Retrieve a list of course names along with the count tof 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(s.id) from student s join enrollment e on s.id=e.student\_id join courses c on c.id = e.courses\_id group by c.course\_name;**

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

| course\_name | count(s.id) |

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

| Charms | 1 |

| Potions | 2 |

| Transfiguration | 1 |

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

1. 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.id, avg(p.amount)from student s join payments p on s.id =p.student\_id group by s.id;**

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

| id | avg(p.amount) |

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

| 1 | 5000 |

| 2 | 7500 |

| 3 | 6000 |

| 4 | 8000 |

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