**Task 1. Database Design:**

1. Create the database named "SISDB"

CREATE DATABASE sisdb;

2. Define the schema for the Students, Courses, Enrollments, Teacher, and Payments tables based

on the provided schema. Write SQL scripts to create the mentioned tables with appropriate data

types, constraints, and relationships.

a. Students

b. Courses

c. Enrollments

d. Teacher

e. Payments

4. Create appropriate Primary Key and Foreign Key constraints for referential integrity.

USE sisdb;

CREATE TABLE student(

student\_id VARCHAR (10)NOT NULL,

first\_name VARCHAR(20) NOT NULL,

last\_name VARCHAR(20) NOT NULL,

date\_of\_birth VARCHAR(20) NOT NULL,

email VARCHAR(20) NOT NULL,

phone\_number INT NOT NULL,

PRIMARY KEY (student\_id)

);

**OUTPUT:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| student\_id | first\_name | last\_name | date\_of\_birth | email | phone\_number |
|  |  |  |  |  |  |

USE sisdb;

CREATE TABLE teacher(

teacher\_id VARCHAR (10)NOT NULL,

first\_name VARCHAR(20) NOT NULL,

last\_name VARCHAR(20) NOT NULL,

email VARCHAR(20) NOT NULL,

PRIMARY KEY (teacher\_id)

);

**OUTPUT:**

|  |  |  |  |
| --- | --- | --- | --- |
| teacher\_id | first\_name | last\_name | email |
|  |  |  |  |

USE sisdb;

CREATE TABLE course(

course\_id VARCHAR (10) NOT NULL,

course\_name VARCHAR(20) NOT NULL,

credits INT NOT NULL,

teacher\_id VARCHAR(10) NOT NULL,

PRIMARY KEY (course\_id),

FOREIGN KEY (teacher\_id) REFERENCES teacher(teacher\_id)

);

**OUTPUT:**

|  |  |  |  |
| --- | --- | --- | --- |
| course\_id | course\_name | credits | teacher\_id |
|  |  |  |  |

USE sisdb;

CREATE TABLE enrollment(

enrollment\_id VARCHAR (10)NOT NULL,

student\_id VARCHAR(20) NOT NULL,

course\_id VARCHAR(20) NOT NULL,

enrollment\_date VARCHAR(20) NOT NULL,

PRIMARY KEY (enrollment\_id),

FOREIGN KEY (student\_id) REFERENCES student(student\_id),

FOREIGN KEY (course\_id) REFERENCES course(course\_id)

);

**OUTPUT:**

|  |  |  |  |
| --- | --- | --- | --- |
| enrollment\_id | student\_id | course\_id | enrollment\_date |
|  |  |  |  |

USE sisdb;

CREATE TABLE payment(

payment\_id VARCHAR (10)NOT NULL,

student\_id VARCHAR(20) NOT NULL,

amount VARCHAR(20) NOT NULL,

payment\_date VARCHAR(20) NOT NULL,

PRIMARY KEY (payment\_id),

FOREIGN KEY (student\_id) REFERENCES student(student\_id)

);

**OUTPUT:**

|  |  |  |  |
| --- | --- | --- | --- |
| payment\_id | student\_id | amount | payment\_date |
|  |  |  |  |

3.Create an ERD (Entity Relationship Diagram) for the database.

payment\_id

payment\_id

payment\_id

payment\_id

payments

M

makes payment

enrollment\_id

\_id

student\_id

\_id

student\_id

1

first\_name

course\_id

enrolls-in

1 M

enrollments

last\_name

student

enrollment\_date

date\_of\_birth

M

has

email

phone\_numbere

teacher\_id

course\_id

M

taught by

M 1

first\_name

teacher

Credits

course

last\_name

teacher\_id

course\_name

email

5. Insert at least 10 sample records into each of the following tables.

i. Students

ii. Courses

iii. Enrollments

iv. Teacher

v. Payments

INSERT INTO student(student\_id, first\_name, last\_name, date\_of\_birth, email, phone\_number)

VALUES

('s1','John','Moxley','2001-06-20','mox@gmail.com',24340),

('s2','Rowan','Atkinson','2001-05-13','rowan@gmail.com',54673),

('s3','Eddie','Gurrero','2001-01-03','eddie@gmail.com',36453),

('s4','David','Beckham','2001-02-20','david@gmail.com',73863),

('s5','Robert','Junior','2001-07-10','robert@gmail.com',23256),

('s6','Carl','Anderson','2001-10-06','carl@gmail.com',35467),

('s7','Michael','Cole','2001-08-11','michael@gmail.com',54767),

('s8','Roy','Mathew','2001-07-05','roy@gmail.com',75678),

('s9','Chelsea','Green','2001-05-05','chelsea@gmail.com',63453),

[('s10','Jack','Morison','2001-11-20','jack@gmail.com',26470);](mailto:('s10','Jack','Morison','2001-11-20','jack@gmail.com',26470);)

**OUTPUT:**

|  | **student\_id** | **first\_name** | **last\_name** | **date\_of\_birth** | **email** | **phone\_number** |
| --- | --- | --- | --- | --- | --- | --- |
|  | s1 | John | Moxley | 2001-06-20 | mox@gmail.com | 9476824340 |
|  | s10 | Jack | Morison | 2001-11-20 | jack@gmail.com | 9876526470 |
|  | s2 | Rowan | Atkinson | 2001-05-13 | rowan@gmail.com | 9342354673 |
|  | s3 | Eddie | Gurrero | 2001-01-03 | eddie@gmail.com | 9677836453 |
|  | s4 | David | Beckham | 2001-02-20 | david@gmail.com | 9423173863 |
|  | s5 | Robert | Junior | 2001-07-10 | robert@gmail.com | 9534423256 |
|  | s6 | Carl | Anderson | 2001-10-06 | carl@gmail.com | 9768035467 |
|  | s7 | Michael | Cole | 2001-08-11 | michael@gmail.com | 9087654767 |
|  | s8 | Roy | Mathew | 2001-07-05 | roy@gmail.com | 9875675678 |
|  | s9 | Chelsea | Green | 2001-05-05 | chelsea@gmail.com | 9076563453 |
|  |  |  |  |  |  |  |

INSERT INTO course(course\_id,course\_name,credits,teacher\_id)

VALUES

('c1','python', 09,'t1'),

('c2','java', 07,'t2'),

('c3','sql', 08,'t3'),

('c4','web development', 07,'t4'),

('c5','cybersecurity', 08,'t5'),

('c6','networking', 07,'t6'),

('c7','DevOps', 09,'t7'),

('c8','Cloud computing', 08,'t8'),

('c9','Database Management', 09,'t9'),

('c10','Javascript', 08,'t10');

**OUTPUT:**

|  | **course\_id** | **course\_name** | **credits** | **teacher\_id** |
| --- | --- | --- | --- | --- |
|  | c1 | python | 9 | t1 |
|  | c10 | Javascript | 8 | t10 |
|  | c2 | java | 7 | t2 |
|  | c3 | sql | 8 | t3 |
|  | c4 | web development | 7 | t4 |
|  | c5 | cybersecurity | 8 | t5 |
|  | c6 | networking | 7 | t6 |
|  | c7 | DevOps | 9 | t7 |
|  | c8 | Cloud computing | 8 | t8 |
|  | c9 | Database Management | 9 | t9 |
|  |  |  |  |  |

INSERT INTO enrollment(enrollment\_id,student\_id,course\_id,enrollment\_date)

VALUES

('e1','s1','c1','2023-06-20'),

('e2','s2','c2','2023-07-17'),

('e3','s3','c3','2023-06-11'),

('e4','s4','c4','2023-05-21'),

('e5','s5','c5','2023-06-11'),

('e6','s6','c6','2023-07-06'),

('e7','s7','c7','2023-05-03'),

('e8','s8','c8','2023-06-05'),

('e9','s9','c9','2023-07-13'),

('e10','s10','c10','2023-05-10');

**OUTPUT:**

|  | **enrollment\_id** | **student\_id** | **course\_id** | **enrollment\_date** |
| --- | --- | --- | --- | --- |
|  | e1 | s1 | c1 | 2023-06-20 |
|  | e10 | s10 | c10 | 2023-05-10 |
|  | e2 | s2 | c2 | 2023-07-17 |
|  | e3 | s3 | c3 | 2023-06-11 |
|  | e4 | s4 | c4 | 2023-05-21 |
|  | e5 | s5 | c5 | 2023-06-11 |
|  | e6 | s6 | c6 | 2023-07-06 |
|  | e7 | s7 | c7 | 2023-05-03 |
|  | e8 | s8 | c8 | 2023-06-05 |
|  | e9 | s9 | c9 | 2023-07-13 |
|  |  |  |  |  |

INSERT INTO teacher(teacher\_id, first\_name, last\_name, email)

VALUES

('t1','Leana','Brooke','leana@gmail.com'),

('t2','Teena','Paul','teena@gmail.com'),

('t3','Megan','Knox','megan@gmail.com'),

('t4','laula','George','laula@gmail.com'),

('t5','Mich','Luke','mich@gmail.com'),

('t6','Jacob','Downey','jacob@gmail.com'),

('t7','Laura','Grace','laura@gmail.com'),

('t8','Baron','Corbin','baron@gmail.com'),

('t9','Cayla','Ruke','cayla@gmail.com'),

[('t10','Bruce','Baner','bruce@gmail.com');](mailto:('t10','Bruce','Baner','bruce@gmail.com');)

**OUTPUT:**

|  | **teacher\_id** | **first\_name** | **last\_name** | **email** |
| --- | --- | --- | --- | --- |
|  | t1 | Leana | Brooke | leana@gmail.com |
|  | t10 | Bruce | Baner | bruce@gmail.com |
|  | t2 | Teena | Paul | teena@gmail.com |
|  | t3 | Megan | Knox | megan@gmail.com |
|  | t4 | laula | George | laula@gmail.com |
|  | t5 | Mich | Luke | mich@gmail.com |
|  | t6 | Jacob | Downey | jacob@gmail.com |
|  | t7 | Laura | Grace | laura@gmail.com |
|  | t8 | Baron | Corbin | baron@gmail.com |
|  | t9 | Cayla | Ruke | cayla@gmail.com |
|  |  |  |  |  |

INSERT INTO payment(payment\_id,student\_id,amount,payment\_date)

VALUES

('p1','s1','10,000','2023-06-30'),

('p2','s2','9,000','2023-07-27'),

('p3','s3','15,000','2023-06-21'),

('p4','s4','12,000','2023-06-01'),

('p5','s5','18,000','2023-06-21'),

('p6','s6','25,000','2023-07-16'),

('p7','s7','18,000','2023-05-13'),

('p8','s8','30,000','2023-06-15'),

('p9','s9','11,000','2023-07-23'),

('p10','s10','8,000','2023-05-20');

**OUTPUT:**

|  | **payment\_id** | **student\_id** | **amount** | **payment\_date** |
| --- | --- | --- | --- | --- |
|  | p1 | s1 | 10,000 | 2023-06-30 |
|  | p10 | s10 | 8,000 | 2023-05-20 |
|  | p2 | s2 | 9,000 | 2023-07-27 |
|  | p3 | s3 | 15,000 | 2023-06-21 |
|  | p4 | s4 | 12,000 | 2023-06-01 |
|  | p5 | s5 | 18,000 | 2023-06-21 |
|  | p6 | s6 | 25,000 | 2023-07-16 |
|  | p7 | s7 | 18,000 | 2023-05-13 |
|  | p8 | s8 | 30,000 | 2023-06-15 |
|  | p9 | s9 | 11,000 | 2023-07-23 |
|  |  |  |  |  |

**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 student(student\_id, first\_name, last\_name,date\_of\_birth, email,phone\_number)

VALUES('s11','John','Doe','1995-08-15','john.doe@example.com','1234567890');

**OUTPUT:**

|  | **student\_id** | **first\_name** | **last\_name** | **date\_of\_birth** | **email** | **phone\_number** |
| --- | --- | --- | --- | --- | --- | --- |
|  | s1 | John | Moxley | 2001-06-20 | mox@gmail.com | 9476824340 |
|  | s10 | Jack | Morison | 2001-11-20 | jack@gmail.com | 9876526470 |
|  | s11 | John | Doe | 1995-08-15 | john.doe@example.com | 1234567890 |
|  | s2 | Rowan | Atkinson | 2001-05-13 | rowan@gmail.com | 9342354673 |
|  | s3 | Eddie | Gurrero | 2001-01-03 | eddie@gmail.com | 9677836453 |
|  | s4 | David | Beckham | 2001-02-20 | david@gmail.com | 9423173863 |
|  | s5 | Robert | Junior | 2001-07-10 | robert@gmail.com | 9534423256 |
|  | s6 | Carl | Anderson | 2001-10-06 | carl@gmail.com | 9768035467 |
|  | s7 | Michael | Cole | 2001-08-11 | michael@gmail.com | 9087654767 |
|  | s8 | Roy | Mathew | 2001-07-05 | roy@gmail.com | 9875675678 |
|  | s9 | Chelsea | Green | 2001-05-05 | chelsea@gmail.com | 9076563453 |
|  |  |  |  |  |  |  |

1. 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(enrollment\_id,student\_id, course\_id,enrollment\_date)

VALUES('e11','s11','c10','2023-07-10');

**OUTPUT:**

|  | **enrollment\_id** | **student\_id** | **course\_id** | **enrollment\_date** |
| --- | --- | --- | --- | --- |
|  | e1 | s1 | c1 | 2023-06-20 |
|  | e10 | s10 | c10 | 2023-05-10 |
|  | e11 | s11 | c10 | 2023-07-10 |
|  | e2 | s2 | c2 | 2023-07-17 |
|  | e3 | s3 | c3 | 2023-06-11 |
|  | e4 | s4 | c4 | 2023-05-21 |
|  | e5 | s5 | c5 | 2023-06-11 |
|  | e6 | s6 | c6 | 2023-07-06 |
|  | e7 | s7 | c7 | 2023-05-03 |
|  | e8 | s8 | c8 | 2023-06-05 |
|  | e9 | s9 | c9 | 2023-07-13 |
|  |  |  |  |  |

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

UPDATE teacher

SET email= 'teena.paul@gmail.com'

WHERE teacher\_id='t2';

**OUTPUT:**

|  | **teacher\_id** | **first\_name** | **last\_name** | **email** |
| --- | --- | --- | --- | --- |
|  | t1 | Leana | Brooke | leana@gmail.com |
|  | t10 | Bruce | Baner | bruce@gmail.com |
|  | t2 | Teena | Paul | teena.paul@gmail.com |
|  | t3 | Megan | Knox | megan@gmail.com |
|  | t4 | laula | George | laula@gmail.com |
|  | t5 | Mich | Luke | mich@gmail.com |
|  | t6 | Jacob | Downey | jacob@gmail.com |
|  | t7 | Laura | Grace | laura@gmail.com |
|  | t8 | Baron | Corbin | baron@gmail.com |
|  | t9 | Cayla | Ruke | cayla@gmail.com |
|  |  |  |  |  |

1. 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\_id = 'e11';

**OUTPUT:**

|  | **enrollment\_id** | **student\_id** | **course\_id** | **enrollment\_date** |
| --- | --- | --- | --- | --- |
|  | e1 | s1 | c1 | 2023-06-20 |
|  | e10 | s10 | c10 | 2023-05-10 |
|  | e2 | s2 | c2 | 2023-07-17 |
|  | e3 | s3 | c3 | 2023-06-11 |
|  | e4 | s4 | c4 | 2023-05-21 |
|  | e5 | s5 | c5 | 2023-06-11 |
|  | e6 | s6 | c6 | 2023-07-06 |
|  | e7 | s7 | c7 | 2023-05-03 |
|  | e8 | s8 | c8 | 2023-06-05 |
|  | e9 | s9 | c9 | 2023-07-13 |
|  |  |  |  |  |

1. Update the "Courses" table to assign a specific teacher to a course. Choose any course and teacher from the respective tables.

INSERT INTO course(course\_id,course\_name,credits,teacher\_id)

VALUES('c11','c++','08','t9');

**OUTPUT:**

|  | **course\_id** | **course\_name** | **credits** | **teacher\_id** |
| --- | --- | --- | --- | --- |
|  | c1 | python | 9 | t1 |
|  | c10 | Javascript | 8 | t10 |
|  | c11 | c++ | 8 | t9 |
|  | c2 | java | 7 | t2 |
|  | c3 | sql | 8 | t3 |
|  | c4 | web development | 7 | t4 |
|  | c5 | cybersecurity | 8 | t5 |
|  | c6 | networking | 7 | t6 |
|  | c7 | DevOps | 9 | t7 |
|  | c8 | Cloud computing | 8 | t8 |
|  | c9 | Database Management | 9 | t9 |
|  |  |  |  |  |

1. 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 = 's8';

DELETE FROM payment

WHERE student\_id = 's8';

DELETE FROM student

WHERE student\_id = 's8';

**OUTPUT:**

|  | **enrollment\_id** | **student\_id** | **course\_id** | **enrollment\_date** |
| --- | --- | --- | --- | --- |
|  | e1 | s1 | c1 | 2023-06-20 |
|  | e10 | s10 | c10 | 2023-05-10 |
|  | e2 | s2 | c2 | 2023-07-17 |
|  | e3 | s3 | c3 | 2023-06-11 |
|  | e4 | s4 | c4 | 2023-05-21 |
|  | e5 | s5 | c5 | 2023-06-11 |
|  | e6 | s6 | c6 | 2023-07-06 |
|  | e7 | s7 | c7 | 2023-05-03 |
|  | e9 | s9 | c9 | 2023-07-13 |
|  |  |  |  |  |

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=23000

WHERE payment\_id= 'p6';

**OUTPUT:**

|  | **payment\_id** | **student\_id** | **amount** | **payment\_date** |
| --- | --- | --- | --- | --- |
|  | p1 | s1 | 10,000 | 2023-06-30 |
|  | p10 | s10 | 8,000 | 2023-05-20 |
|  | p2 | s2 | 9,000 | 2023-07-27 |
|  | p3 | s3 | 15,000 | 2023-06-21 |
|  | p4 | s4 | 12,000 | 2023-06-01 |
|  | p5 | s5 | 18,000 | 2023-06-21 |
|  | p6 | s6 | 23000 | 2023-07-16 |
|  | p7 | s7 | 18,000 | 2023-05-13 |
|  | p9 | s9 | 11,000 | 2023-07-23 |
|  |  |  |  |  |

**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 student.first\_name, payment.amount

FROM student INNER JOIN payment

ON student.student\_id=payment.student\_id;

**OUTPUT:**

|  | **first\_name** | **amount** |
| --- | --- | --- |
|  | John | 10,000 |
|  | Jack | 8,000 |
|  | Rowan | 9,000 |
|  | Eddie | 15,000 |
|  | David | 12,000 |
|  | Robert | 18,000 |
|  | Carl | 23000 |
|  | Michael | 18,000 |
|  | Chelsea | 11,000 |

1. 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 course.course\_id,course.course\_name, COUNT(enrollment.student\_id) AS enrolled\_students

FROM course

LEFT JOIN enrollment ON course.course\_id=enrollment.course\_id

GROUP BY course.course\_id,course.course\_name

ORDER BY enrolled\_students DESC;

**OUTPUT:**

|  | **course\_id** | **course\_name** | **enrolled\_students** |
| --- | --- | --- | --- |
|  | c1 | python | 1 |
|  | c10 | Javascript | 1 |
|  | c2 | java | 1 |
|  | c3 | sql | 1 |
|  | c4 | web development | 1 |
|  | c5 | cybersecurity | 1 |
|  | c6 | networking | 1 |
|  | c7 | DevOps | 1 |
|  | c9 | Database Management | 1 |
|  | c11 | c++ | 0 |
|  | c8 | Cloud computing | 0 |

1. 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 course.course\_id,course.course\_name, COUNT(student.student\_id) AS enrolled\_students

FROM course

LEFT JOIN enrollment AS enrollment ON course.course\_id=enrollment.course\_id

LEFT JOIN student AS student ON enrollment.student\_id=student.student\_id

GROUP BY course.course\_id,course.course\_name

HAVING COUNT(student.student\_id)=0;

**OUTPUT:**

|  |  |  |
| --- | --- | --- |
| c11 | c++ | 0 |
| c8 | Cloud computing | 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 student.first\_name,student.last\_name,course.course\_name

FROM student INNER JOIN enrollment ON student.student\_id=enrollment.student\_id

INNER JOIN course ON enrollment.course\_id=course.course\_id;

**OUTPUT:**

|  | **first\_name** | **last\_name** | **course\_name** |
| --- | --- | --- | --- |
|  | John | Moxley | python |
|  | Jack | Morison | Javascript |
|  | Rowan | Atkinson | java |
|  | Eddie | Gurrero | sql |
|  | David | Beckham | web development |
|  | Robert | Junior | cybersecurity |
|  | Carl | Anderson | networking |
|  | Michael | Cole | DevOps |
|  | Chelsea | Green | Database Management |

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 teacher.first\_name, teacher.last\_name,course.course\_name

FROM teacher INNER JOIN course

ON teacher.teacher\_id=course.teacher\_id;

**OUTPUT:**

|  | **first\_name** | **last\_name** | **course\_name** |
| --- | --- | --- | --- |
|  | Leana | Brooke | python |
|  | Bruce | Baner | Javascript |
|  | Teena | Paul | java |
|  | Megan | Knox | sql |
|  | laula | George | web development |
|  | Mich | Luke | cybersecurity |
|  | Jacob | Downey | networking |
|  | Laura | Grace | DevOps |
|  | Baron | Corbin | Cloud computing |
|  | Cayla | Ruke | c++ |
|  | Cayla | Ruke | Database Management |

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 student.first\_name,student.last\_name, enrollment.enrollment\_date,course.course\_name

FROM student INNER JOIN enrollment ON student.student\_id=enrollment.student\_id

INNER JOIN course ON enrollment.course\_id=course.course\_id;

**OUTPUT:**

|  | **first\_name** | **last\_name** | **enrollment\_date** | **course\_name** |
| --- | --- | --- | --- | --- |
|  | John | Moxley | 2023-06-20 | python |
|  | Jack | Morison | 2023-05-10 | Javascript |
|  | Rowan | Atkinson | 2023-07-17 | java |
|  | Eddie | Gurrero | 2023-06-11 | sql |
|  | David | Beckham | 2023-05-21 | web development |
|  | Robert | Junior | 2023-06-11 | cybersecurity |
|  | Carl | Anderson | 2023-07-06 | networking |
|  | Michael | Cole | 2023-05-03 | DevOps |
|  | Chelsea | Green | 2023-07-13 | Database Management |

1. 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 student.first\_name

FROM student

LEFT JOIN payment ON student.student\_id=payment.student\_id

WHERE payment.payment\_id IS NULL;

**OUTPUT:**

|  | **first\_name** |
| --- | --- |
|  | John |

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 course.course\_name

FROM course

LEFT JOIN enrollment ON enrollment.course\_id=course.course\_id

WHERE enrollment.enrollment\_id IS NULL;

**OUTPUT:**

|  | **course\_name** |
| --- | --- |
|  | c++ |
|  | Cloud computing |

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 DISTINCT E1.student\_id,student.first\_name,student.last\_name

FROM enrollment E1

JOIN enrollment E2 ON E1.student\_id=E2.student\_id

AND E1.course\_id<>E2.course\_id

JOIN student ON E1.student\_id=student.student\_id;

**OUTPUT:**

|  |  |  |
| --- | --- | --- |
| student\_id | first\_name | last\_name |
|  |  |  |

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 teacher.teacher\_id,teacher.first\_name

FROM teacher

LEFT JOIN course ON teacher.teacher\_id=course.teacher\_id

WHERE course\_id IS NULL

**OUTPUT:**

|  |  |
| --- | --- |
| teacher\_id | first\_name |
|  |  |

**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.course\_id,course.course\_name, AVG (enrollment.student\_id) AS average\_students

FROM course

INNER JOIN (

SELECT student\_id, COUNT(\*) AS enrolled\_courses

FROM enrollment

GROUP BY student\_id

) AS enrollment ON course.course\_id= enrollment.enrolled\_courses

GROUP BY course.course\_id,course.course\_name;

**OUTPUT:**

|  |  |  |
| --- | --- | --- |
| course\_id | course\_name | average\_students |
|  |  |  |

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.first\_name, payment.amount

FROM student

INNER JOIN payment ON student.student\_id=payment.student\_id

WHERE payment.amount=(

SELECT MAX(amount)

FROM payment

);

**OUTPUT:**

|  | **first\_name** | **amount** |
| --- | --- | --- |
|  | Carl | 23,000 |

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.course\_id, course.course\_name, COUNT () AS enrolled\_students

FROM course

INNER JOIN enrollment ON course.course\_id=enrollment.course\_id

GROUP BY course.course\_id, course.course\_name

HAVING COUNT(\*) =(

SELECT MAX(enrolled\_count)

FROM(

SELECT course.course\_id, COUNT(\*) AS enrolled\_count

FROM course

INNER JOIN enrollment

GROUP BY course.course\_id

) AS sub\_query

);

**OUTPUT:**

|  |  |  |
| --- | --- | --- |
| course\_id | course\_name | enrolled\_students |
|  |  |  |

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

SELECT teacher.teacher\_id, teacher.first\_name,SUM (payment.amount) AS total\_amount

FROM teacher

INNER JOIN course ON teacher.teacher\_id= course.teacher\_id

INNER JOIN enrollment ON course.course\_id=enrollment.course\_id

INNER JOIN payment ON enrollment.student\_id=payment.student\_id

GROUP BY teacher.teacher\_id, teacher.first\_name;

**OUTPUT:**

|  | **teacher\_id** | **first\_name** | **total\_amount** |
| --- | --- | --- | --- |
|  | t1 | Leana | 10 |
|  | t10 | Bruce | 8 |
|  | t2 | Teena | 9 |
|  | t3 | Megan | 15 |
|  | t4 | laula | 12 |
|  | t5 | Mich | 18 |
|  | t6 | Jacob | 23000 |
|  | t7 | Laura | 18 |
|  | t9 | Cayla | 11 |

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 student.student\_id, student.first\_name

FROM student

INNER JOIN (

SELECT COUNT(\*) AS total\_courses

FROM course

)AS course ON(

SELECT COUNT(\*) AS enrolled\_courses

FROM enrollment

WHERE enrollment.student\_id=student.student\_id

)=course.total\_courses;

**OUTPUT:**

|  |  |
| --- | --- |
| student\_id | first\_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 teacher.first\_name

FROM teacher

WHERE NOT EXISTS(

SELECT 1

FROM course

WHERE course.teacher\_id=teacher.teacher\_id

);

**OUTPUT:**

|  |
| --- |
| first\_name |
|  |

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 (age) AS average\_age

FROM(

SELECT student\_id, datediff(CURDATE(),date\_of\_birth)/365.25 AS age

FROM student

) AS student\_ages

**OUTPUT:**

|  | **average\_age** |
| --- | --- |
|  | 23.06776181 |

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

records.

SELECT course.course\_name

FROM course

WHERE NOT EXISTS(

SELECT 1

FROM enrollment

WHERE enrollment.course\_id=course.course\_id

);

**OUTPUT:**

|  | **course\_name** |
| --- | --- |
|  | c++ |
|  | Cloud computing |

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.student\_id, student.first\_name,course.course\_id,course.course\_name,

SUM(payment.amount) AS total\_payment

FROM student

INNER JOIN enrollment ON student.student\_id=enrollment.student\_id

INNER JOIN course ON course.course\_id=enrollment.course\_id

INNER JOIN payment ON payment.student\_id=enrollment.student\_id

GROUP BY student.student\_id,student.first\_name,course.course\_id,course.course\_name;

**OUTPUT:**

|  | **student\_id** | **first\_name** | **course\_id** | **course\_name** | **total\_payment** |
| --- | --- | --- | --- | --- | --- |
|  | s1 | John | c1 | python | 10 |
|  | s10 | Jack | c10 | Javascript | 8 |
|  | s2 | Rowan | c2 | java | 9 |
|  | s3 | Eddie | c3 | sql | 15 |
|  | s4 | David | c4 | web development | 12 |
|  | s5 | Robert | c5 | cybersecurity | 18 |
|  | s6 | Carl | c6 | networking | 23000 |
|  | s7 | Michael | c7 | DevOps | 18 |
|  | s9 | Chelsea | c9 | Database Management | 11 |

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.student\_id,student.first\_name, COUNT(\*) AS total\_payments

FROM student

INNER JOIN payment ON student.student\_id=payment.student\_id

GROUP BY student.student\_id, student.first\_name

HAVING COUNT(\*)>1;

**OUTPUT:**

|  |  |  |
| --- | --- | --- |
| student\_id | first\_name | total\_payments |
|  |  |  |

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 student.student\_id,student.first\_name, SUM(payment.amount) AS total\_payments

FROM student

INNER JOIN payment ON student.student\_id=payment.student\_id

GROUP BY student.student\_id, student.first\_name;

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 course.course\_name, COUNT(\*) AS enrolled\_students

FROM course

INNER JOIN enrollment ON course.course\_id=enrollment.course\_id

GROUP BY course.course\_name

ORDER BY enrolled\_students DESC;

**OUTPUT:**

|  | **student\_id** | **first\_name** | **total\_payments** |
| --- | --- | --- | --- |
|  | s1 | John | 10 |
|  | s10 | Jack | 8 |
|  | s2 | Rowan | 9 |
|  | s3 | Eddie | 15 |
|  | s4 | David | 12 |
|  | s5 | Robert | 18 |
|  | s6 | Carl | 23000 |
|  | s7 | Michael | 18 |
|  | s9 | Chelsea | 11 |

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(payment.amount) AS average\_payment

FROM student

INNER JOIN payment ON student.student\_id=payment.student\_id;

**OUTPUT:**

|  | **average\_payment** |
| --- | --- |
|  | 2566.777777777778 |