Student Information System

Task 1

i.)

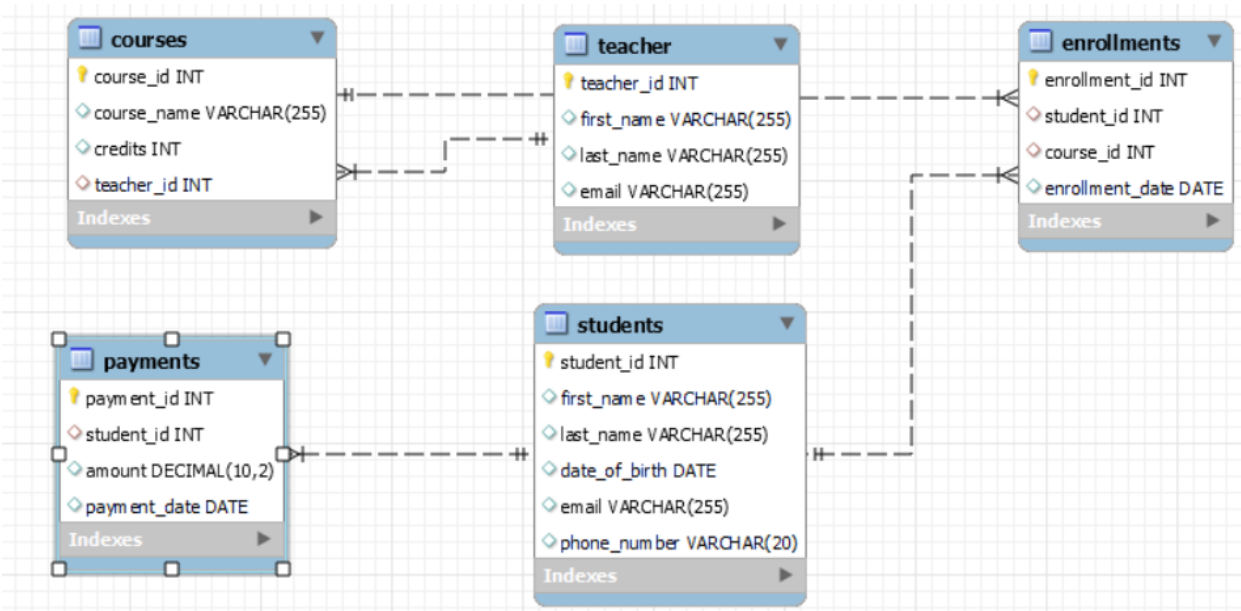
```sql
CREATE DATABASE SISDB;
USE SISDB;
```

90	20:14:10	CREATE DATABASE SISDB	1 row(s) affected
91	20:14:10	USE SISDB	0 row(s) affected

ii.) Creating the tables

```sql
CREATE TABLE Teacher (teacher_id INT PRIMARY KEY,
    first_name VARCHAR(255),
    last_name VARCHAR(255),
    email VARCHAR(255));
CREATE TABLE Students (student_id INT PRIMARY KEY,
    first_name VARCHAR(255),
    last_name VARCHAR(255),
    date_of_birth DATE,
    email VARCHAR(255),
    phone_number VARCHAR(20));
CREATE TABLE Courses (course_id INT PRIMARY KEY,
    course_name VARCHAR(255),
    credits INT,
    teacher_id INT,
    FOREIGN KEY (teacher_id) REFERENCES Teacher(teacher_id));
CREATE TABLE Enrollments (enrollment_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    enrollment_date DATE,
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id));
CREATE TABLE Payments (payment_id INT PRIMARY KEY,
    student_id INT,
    amount DECIMAL(10, 2),
    payment_date DATE,
    FOREIGN KEY (student_id) REFERENCES Students(student_id));
```

iii.) ERD Diagram for the Tables



iv.) Inserting records into the tables

```sql
INSERT INTO Teacher VALUES (1, 'Professor', 'Smith', 'prof.smith@example.com'),
(2, 'Professor', 'Johnson', 'prof.johnson@example.com'),
(3, 'Professor', 'Brown', 'prof.brown@example.com'),
(4, 'Professor', 'Taylor', 'prof.taylor@example.com'),
(5, 'Professor', 'Clark', 'prof.clark@example.com'),
(6, 'Professor', 'Baker', 'prof.baker@example.com'),
(7, 'Professor', 'Garcia', 'prof.garcia@example.com'),
(8, 'Professor', 'Smith', 'prof.smith@example.com'),
(9, 'Professor', 'Walker', 'prof.walker@example.com'),
(10, 'Professor', 'Wright', 'prof.wright@example.com');

INSERT INTO Students VALUES (1, 'John', 'Doe', '1990-01-01', 'john.doe@example.com', '123-456-7890'),
(2, 'Alice', 'Johnson', '1992-05-15', 'alice.johnson@example.com', '987-654-3210'),
(3, 'Bob', 'Smith', '1991-08-20', 'bob.smith@example.com', '456-789-0123'),
(4, 'Eva', 'Miller', '1993-04-12', 'eva.miller@example.com', '234-567-8901'),
(5, 'Daniel', 'Williams', '1992-11-30', 'daniel.williams@example.com', '789-012-3456'),
(6, 'Grace', 'Davis', '1994-07-18', 'grace.davis@example.com', '345-678-9012'),
(7, 'Michael', 'Anderson', '1991-09-25', 'michael.anderson@example.com', '567-890-1234'),
(8, 'Sophia', 'Wilson', '1993-12-08', 'sophia.wilson@example.com', '123-456-7890'),
```

```sql
(9, 'Oliver', 'Taylor', '1990-06-05', 'oliver.taylor@example.com', '456-789-0123'),
(10, 'Ava', 'Moore', '1992-03-22', 'ava.moore@example.com', '789-012-3456');

INSERT INTO Courses VALUES (101, 'Mathematics', 3, 1),
(102, 'Computer Science', 4, 2),
(103, 'History', 3, 1),
(104, 'Chemistry', 3, 3),
(105, 'Literature', 4, 2),
(106, 'Physics', 3, 3),
(107, 'Art History', 2, 1),
(108, 'Biology', 4, 4),
(109, 'Political Science', 3, 5),
(110, 'Economics', 3, 6);

INSERT INTO Enrollments VALUES (1, 1, 101, '2023-01-15'),
(2, 2, 102, '2023-01-20'),
(3, 3, 103, '2023-02-05'),
(4, 4, 104, '2023-03-15'),
(5, 5, 105, '2023-04-01'),
```

```sql
(6, 6, 106, '2023-04-10'),
(7, 7, 107, '2023-05-05'),
(8, 8, 108, '2023-06-20'),
(9, 9, 109, '2023-07-05'),
(10, 10, 110, '2023-07-15');

INSERT INTO Payments VALUES (1, 1, 500.00, '2023-02-01'),
(2, 2, 600.00, '2023-02-10'),
(3, 3, 450.00, '2023-03-01'),
(4, 4, 300.00, '2023-05-10'),
(5, 5, 550.00, '2023-05-15'),
(6, 6, 400.00, '2023-06-01'),
(7, 7, 200.00, '2023-06-15'),
(8, 8, 250.00, '2023-08-01'),
(9, 9, 350.00, '2023-08-10'),
(10, 10, 450.00, '2023-08-15');
```

98	20:19:52	CREATE TABLE Payments (payment_id INT PRIMARY KEY, student_id INT, amount DECIMAL(10, 2)...	0 row(s) affected	
99	21:08:30	INSERT INTO Teacher VALUES (1, 'Professor', 'Smith', 'prof.smith@example.com'), (2, 'Professor', 'Johnson', '...	10 row(s) affected Records: 10	Duplicates: 0 Warnings: 0
100	21:08:30	INSERT INTO Students VALUES (1, 'John', 'Doe', '1990-01-01', 'john.doe@example.com', '123-456-7890'), (2, ...	10 row(s) affected Records: 10	Duplicates: 0 Warnings: 0
101	21:08:30	INSERT INTO Courses VALUES (101, 'Mathematics', 3, 1), (102, 'Computer Science', 4, 2), (103, 'History', 3, 1...	10 row(s) affected Records: 10	Duplicates: 0 Warnings: 0
102	21:08:30	INSERT INTO Enrollments VALUES (1, 1, 101, '2023-01-15'), (2, 2, 102, '2023-01-20'), (3, 3, 103, '2023-02-05')...	10 row(s) affected Records: 10	Duplicates: 0 Warnings: 0
103	21:08:30	INSERT INTO Payments VALUES (1, 1, 500.00, '2023-02-01'), (2, 2, 600.00, '2023-02-10'), (3, 3, 450.00, '202...	10 row(s) affected Records: 10	Duplicates: 0 Warnings: 0

Task 2

i.) Insert some values into the student table

```
INSERT INTO Stundets(student_id, first_name, last_name, date_of_birth, email,
phone_number) values(11, 'John', 'Doe', '1995-08-15', '1234567890');
```

student_id	first_name	last_name	date_of_birth	email	phone_number
1	John	Doe	1995-08-15	john.doe@example.com	123-456-7890
2	Alice	Johnson	1992-05-15	alice.johnson@example.com	987-654-3210
3	Bob	Smith	1991-08-20	bob.smith@example.com	456-789-0123
4	Eva	Miller	1993-04-12	eva.miller@example.com	234-567-8901
5	Daniel	Williams	1992-11-30	daniel.williams@example.com	789-012-3456
6	Grace	Davis	1994-07-18	grace.davis@example.com	345-678-9012
7	Michael	Anderson	1991-09-25	michael.anderson@example.com	567-890-1234
8	Sophia	Wilson	1993-12-08	sophia.wilson@example.com	123-456-7890
9	Oliver	Taylor	1990-06-05	oliver.taylor@example.com	456-789-0123
10	Ava	Moore	1992-03-22	ava.moore@example.com	789-012-3456

ii.) Enroll a student in a course

```sql
INSERT INTO Enrollments(enrollment_id, student_id, course_id, enrollment_date)
                VALUES(11, 1, 102, '2023-01-15');
SELECT * FROM Enrollments;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Conten

enrollment_id	student_id	course_id	enrollment_date
1	1	101	2023-01-15
2	2	102	2023-01-20
3	3	103	2023-02-05
4	4	104	2023-03-15
5	5	105	2023-04-01
6	6	106	2023-04-10
7	7	107	2023-05-05
8	8	108	2023-06-20
9	9	109	2023-07-05
10	10	110	2023-07-15
11	1	102	2023-01-15

iii.) Change the email address of a professor.

```sql
UPDATE Teacher SET email = 'smith.prof@example.com' WHERE teacher_id=1;
SELECT * FROM Teacher;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

teacher_id	first_name	last_name	email
1	Professor	Smith	smith.prof@example.com
2	Professor	Johnson	prof.johnson@example.com
3	Professor	Brown	prof.brown@example.com
4	Professor	Taylor	prof.taylor@example.com
5	Professor	Clark	prof.clark@example.com
6	Professor	Baker	prof.baker@example.com
7	Professor	Garcia	prof.garcia@example.com
8	Professor	Smith	prof.smith@example.com
9	Professor	Walker	prof.walker@example.com
10	Professor	Wright	prof.wright@example.com

iv.) Delete an enrollment from the enrollment table and select using course id and student id.

```
DELETE FROM Enrollments WHERE student_id =1 AND course_id = 102;
SELECT * FROM Enrollments;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

enrollment_id	student_id	course_id	enrollment_date
1	1	101	2023-01-15
2	2	102	2023-01-20
3	3	103	2023-02-05
4	4	104	2023-03-15
5	5	105	2023-04-01
6	6	106	2023-04-10
7	7	107	2023-05-05
8	8	108	2023-06-20
9	9	109	2023-07-05
10	10	110	2023-07-15

v.) update courses table and assign it to a teacher.

```
UPDATE Courses SET teacher_id=7 WHERE course_id = 110;
SELECT * FROM Courses;
```

Result Grid | Filter Rows: | Edit: | Export/In

course_id	course_name	credits	teacher_id
101	Mathematics	3	1
102	Computer Science	4	2
103	History	3	1
104	Chemistry	3	3
105	Literature	4	2
106	Physics	3	3
107	Art History	2	1
108	Biology	4	4
109	Political Science	3	5
110	Economics	3	7

vi.) delete a student from the table and ensure the referential integrity is maintained.

```
DELETE FROM Enrollments WHERE student_id = 10;
DELETE FROM Payments WHERE student_id = 10;
DELETE FROM Students WHERE student_id = 10;
SELECT * FROM Students;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

student_id	first_name	last_name	date_of_birth	email	phone_number
1	John	Doe	1995-08-15	john.doe@example.com	123-456-7890
2	Alice	Johnson	1992-05-15	alice.johnson@example.com	987-654-3210
3	Bob	Smith	1991-08-20	bob.smith@example.com	456-789-0123
4	Eva	Eva	1993-04-12	eva.miller@example.com	234-567-8901
5	Daniel		1992-11-30	daniel.williams@example.com	789-012-3456
6	Grace	Davis	1994-07-18	grace.davis@example.com	345-678-9012
7	Michael	Anderson	1991-09-25	michael.anderson@example.com	567-890-1234
8	Sophia	Wilson	1993-12-08	sophia.wilson@example.com	123-456-7890
9	Oliver	Taylor	1990-06-05	oliver.taylor@example.com	456-789-0123

vii.) Update payment table

```
UPDATE Payments SET amount = 900 WHERE student_id = 1;
SELECT *  FROM Payments;
```

Result Grid | Filter Rows: | Edit: | Export/In

payment_id	student_id	amount	payment_date
1	1	900.00	2023-02-01
2	2	600.00	2023-02-10
3	3	450.00	2023-03-01
4	4	300.00	2023-05-10
5	5	550.00	2023-05-15
6	6	400.00	2023-06-01
7	7	200.00	2023-06-15
8	8	250.00	2023-08-01
9	9	350.00	2023-08-10

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.

```sql
SELECT SUM(Payments.amount) AS "Total Payments" FROM Payments JOIN Students
ON Payments.student_id = Students.student_id
WHERE Students.student_id = 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Total Payments
900.00

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.

```sql
SELECT c.course_id, c.course_name, COUNT(e.student_id) AS students_enrolled
FROM Courses c
JOIN Enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

course_id	course_name	students_enrolled
101	Mathematics	1
102	Computer Science	1
103	History	1
104	Chemistry	1
105	Literature	1
106	Physics	1
107	Art History	1
108	Biology	1
109	Political Science	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 identify students without enrollments.

```sql
SELECT s.first_name, s.last_name
FROM Students s
LEFT JOIN Enrollments e ON s.student_id = e.student_id
WHERE e.student_id IS NULL;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name

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.

```sql
SELECT s.first_name, s.last_name, c.course_name
FROM Students s
JOIN Enrollments e ON s.student_id = e.student_id
JOIN Courses c ON e.course_id = c.course_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell

first_name	last_name	course_name
John	Doe	Mathematics
Alice	Johnson	Computer Science
Bob	Smith	History
Eva	Miller	Chemistry
Daniel	Williams	Literature
Grace	Davis	Physics
Michael	Anderson	Art History
Sophia	Wilson	Biology
Oliver	Taylor	Political Science

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.first_name, t.last_name, c.course_name
FROM Teacher t
JOIN Courses c ON t.teacher_id = c.teacher_id;
```

Result Grid | Filter Rows: | Export: | Wrap Ce

first_name	last_name	course_name
Professor	Smith	Mathematics
Professor	Johnson	Computer Science
Professor	Smith	History
Professor	Brown	Chemistry
Professor	Johnson	Literature
Professor	Brown	Physics
Professor	Smith	Art History
Professor	Taylor	Biology
Professor	Clark	Political Science
Professor	Garcia	Economics

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

```sql
SELECT s.first_name, s.last_name, e.enrollment_date
FROM Students s
JOIN Enrollments e ON s.student_id = e.student_id
JOIN Courses c ON e.course_id = c.course_id
WHERE c.course_id = 101;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Co

first_name	last_name	enrollment_date
John	Doe	2023-01-15

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.

```sql
SELECT s.first_name, s.last_name
FROM Students s
LEFT JOIN Payments p ON s.student_id = p.student_id
WHERE p.student_id IS NULL;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Con

first_name	last_name

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.

```sql
SELECT c.course_id, c.course_name
FROM Courses c
LEFT JOIN Enrollments e ON c.course_id = e.course_id
WHERE e.course_id IS NULL;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

course_id	course_name
110	Economics

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.

```sql
SELECT DISTINCT e1.student_id, s.first_name, s.last_name
FROM Enrollments e1
JOIN Enrollments e2 ON e1.student_id = e2.student_id AND e1.course_id <> e2.course_id
JOIN Students s ON e1.student_id = s.student_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

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.

```sql
SELECT * FROM Teacher LEFT JOIN Courses
ON Teacher.teacher_id = Courses.teacher_id
WHERE Courses.course_id IS NULL;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

teacher_id	first_name	last_name	email	course_id	course_name	credits	teacher_id
6	Professor	Baker	prof.baker@example.com	NULL	NULL	NULL	NULL
8	Professor	Smith	prof.smith@example.com	NULL	NULL	NULL	NULL
9	Professor	Walker	prof.walker@example.com	NULL	NULL	NULL	NULL
10	Professor	Wright	prof.wright@example.com	NULL	NULL	NULL	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.

```sql
SELECT AVG(enrollment_count) AS avg_students_per_course
FROM (SELECT course_id, COUNT(DISTINCT student_id) AS enrollment_count
FROM Enrollments GROUP BY course_id) AS course_enrollment_counts;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

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.

```sql
SELECT student_id, first_name, last_name
FROM Students
WHERE student_id = (SELECT student_id FROM Payments ORDER BY amount DESC LIMIT 1);
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

student_id	first_name	last_name
1	John	Doe

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 * FROM Courses
WHERE course_id = (SELECT course_id FROM Enrollments
GROUP BY course_id ORDER BY COUNT(student_id) DESC LIMIT 1);
```

Result Grid | Filter Rows: | Edit: | Export/Import:

course_id	course_name	credits	teacher_id
101	Mathematics	3	1

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

```
SELECT teacher_id, SUM(amount) AS total_payments
FROM Courses
JOIN Enrollments ON Courses.course_id = Enrollments.course_id
JOIN Payments ON Enrollments.student_id = Payments.student_id
GROUP BY teacher_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

teacher_id	total_payments
1	1550.00
2	1150.00
3	700.00
4	250.00
5	350.00

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(DISTINCT course_id) FROM Courses) =
(SELECT COUNT(DISTINCT course_id)
FROM Enrollments WHERE Students.student_id = Enrollments.student_id);
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wr

student_id	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.

```sql
SELECT teacher_id, first_name, last_name
FROM Teacher
WHERE teacher_id NOT IN (SELECT DISTINCT teacher_id FROM Courses);
```

Result Grid | Filter Rows: | Edit: | Export/Import:

teacher_id	first_name	last_name
6	Professor	Baker
8	Professor	Smith
9	Professor	Walker
10	Professor	Wright

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

```sql
SELECT AVG(Age) AS "Average Age"
FROM (SELECT (DATEDIFF(NOW(), date_of_birth)/365) AS "Age" FROM Students)
AS Students_Age;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Average Age
31.02739726

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

```sql
SELECT course_id, course_name
FROM Courses
WHERE course_id NOT IN (SELECT DISTINCT course_id FROM Enrollments);
```

Result Grid | Filter Rows: | Edit: | Export/Import:

course_id	course_name
110	Economics

9. Calculate the total payments made by each student for each course they are enrolled in. Use subqueries and aggregate functions to sum payments.

```sql
SELECT Enrollments.student_id, Enrollments.course_id, SUM(amount) AS total_payments
FROM Enrollments
JOIN Payments ON Enrollments.student_id = Payments.student_id
AND Enrollments.course_id = Payments.course_id
GROUP BY Enrollments.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.

```sql
SELECT student_id, first_name, last_name
FROM Students
WHERE student_id IN (SELECT student_id FROM Payments GROUP BY student_id HAVING COUNT(payment_id) > 1);
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

student_id	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.

```sql
SELECT Students.student_id, first_name, last_name, SUM(amount) AS total_payments
FROM Students
JOIN Payments ON Students.student_id = Payments.student_id
GROUP BY Students.student_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

student_id	first_name	last_name	total_payments
1	John	Doe	900.00
2	Alice	Johnson	600.00
3	Bob	Smith	450.00
4	Eva	Miller	300.00
5	Daniel	Williams	550.00
6	Grace	Davis	400.00
7	Michael	Anderson	200.00
8	Sophia	Wilson	250.00
9	Oliver	Taylor	350.00

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.

```sql
SELECT Courses.course_id, course_name, COUNT(DISTINCT student_id) AS students_enrolled
FROM Courses
JOIN Enrollments ON Courses.course_id = Enrollments.course_id
GROUP BY Courses.course_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

course_id	course_name	students_enrolled
101	Mathematics	1
102	Computer Science	1
103	History	1
104	Chemistry	1
105	Literature	1
106	Physics	1
107	Art History	1
108	Biology	1
109	Political Science	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.

```sql
SELECT AVG(amount) AS "Average Payments" FROM Students
JOIN Payments ON Students.student_id = Payments.student_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Average Payments
444.444444