

ASSIGNMENT STUDENT INFORMATION SYSTEM

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
FROM (
    SELECT course_id, COUNT(student_id) AS student_count
    FROM Enrollments
    GROUP BY course_id
) AS CourseEnrollment;
```

The screenshot shows a database query editor with the following SQL query:

```
254 • SELECT AVG(student_count) AS avg_students
255 FROM (
256     SELECT course_id, COUNT(student_id) AS student_count
257     FROM Enrollments
258     GROUP BY course_id
259 ) AS CourseEnrollment;
260
```

The result grid below the query shows the following data:

avg_students
1.0000

The interface includes a toolbar at the top with icons for file operations, a 'Limit to 1000 rows' dropdown, and a 'Filter Rows' section. The bottom of the window shows a tab labeled 'Result 1' and a 'Read Only' status.

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_id, amount
FROM Payments
WHERE amount = (SELECT MAX(amount) FROM Payments);
```

The screenshot shows a database query editor with the following SQL query:

```
261 • SELECT student_id, amount
262 FROM Payments
263 WHERE amount = (SELECT MAX(amount) FROM Payments);
264
265
```

The result grid below the query shows the following data:

student_id	amount
10	5400.00

The interface includes a toolbar at the top with icons for file operations, a 'Filter Rows' section, and a 'Wrap Cell Content' dropdown. The bottom of the window shows a tab labeled 'Payments 2' and a 'Read Only' status.

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_id, COUNT(student_id) AS enrollment_count
FROM Enrollments
GROUP BY course_id
HAVING COUNT(student_id) = (
    SELECT MAX(enrollment_count)
    FROM (
        SELECT course_id, COUNT(student_id) AS enrollment_count
        FROM Enrollments
        GROUP BY course_id
    ) AS CourseCounts
);
```

course_id	enrollment_count
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1

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

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

teacher_id	total_payments
2	4500.00
5	9400.00
4	500.00
6	4900.00
7	5100.00
8	5200.00
9	5300.00
10	5400.00

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

```
SELECT student_id
FROM Enrollments
GROUP BY student_id
HAVING COUNT(DISTINCT course_id) = (SELECT COUNT(*) FROM Courses);
```

The screenshot shows a SQL query editor with the following query:

```
283 • SELECT student_id
284 FROM Enrollments
285 GROUP BY student_id
286 HAVING COUNT(DISTINCT course_id) = (SELECT COUNT(*) FROM Courses);
287
288
```

Below the query editor, there is a toolbar with options: Result Grid, Filter Rows, Export, and Wrap Cell Content. The Result Grid is active, showing a table with one column: student_id. The table is empty.

On the right side, there is a sidebar with icons for Result Grid, Form Editor, and Field Types. At the bottom, there is a status bar showing "Enrollments 5" and a "Read Only" indicator.

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_ID, CONCAT(FIRST_NAME, ' ', LAST_NAME) AS FULL_NAME
FROM TEACHERS
WHERE TEACHER_ID NOT IN (SELECT DISTINCT TEACHER_ID FROM COURSES);
```

The screenshot shows a SQL query editor with the following query:

```
288 • SELECT TEACHER_ID, CONCAT(FIRST_NAME, ' ', LAST_NAME) AS FULL_NAME
289 FROM TEACHERS
290 WHERE TEACHER_ID NOT IN (SELECT DISTINCT TEACHER_ID FROM COURSES);
291
292
```

Below the query editor, there is a toolbar with options: Result Grid, Filter Rows, Export, and Wrap Cell Content. The Result Grid is active, showing a table with two columns: TEACHER_ID and FULL_NAME. The table has one row with the values 3 and MARTIN RAJ.

On the right side, there is a sidebar with icons for Result Grid, Form Editor, and Field Types. At the bottom, there is a status bar showing "Result 6" and a "Read Only" indicator.

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 avg_age  
FROM Students;
```

The screenshot shows a database query editor with the following SQL query:

```
292 • SELECT AVG(DATEDIFF(CURDATE(), date_of_birth) / 365) AS avg_age  
293 FROM Students;  
294  
295
```

Below the query editor, the 'Result Grid' is displayed, showing the results of the query:

avg_age
23.92136986

The interface includes a toolbar with options like 'Filter Rows', 'Export', and 'Wrap Cell Content'. A sidebar on the right contains icons for 'Result Grid', 'Form Editor', and 'Field Types'. The bottom status bar indicates 'Result 7' and 'Read Only'.

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

```
SELECT COURSE_ID, COURSE_NAME  
FROM COURSES  
WHERE COURSE_ID NOT IN (SELECT DISTINCT COURSE_ID FROM ENROLLMENTS);
```

The screenshot shows a database query editor with the following SQL query:

```
296 • SELECT COURSE_ID, COURSE_NAME  
297 FROM COURSES  
298 WHERE COURSE_ID NOT IN (SELECT DISTINCT COURSE_ID FROM ENROLLMENTS);  
299
```

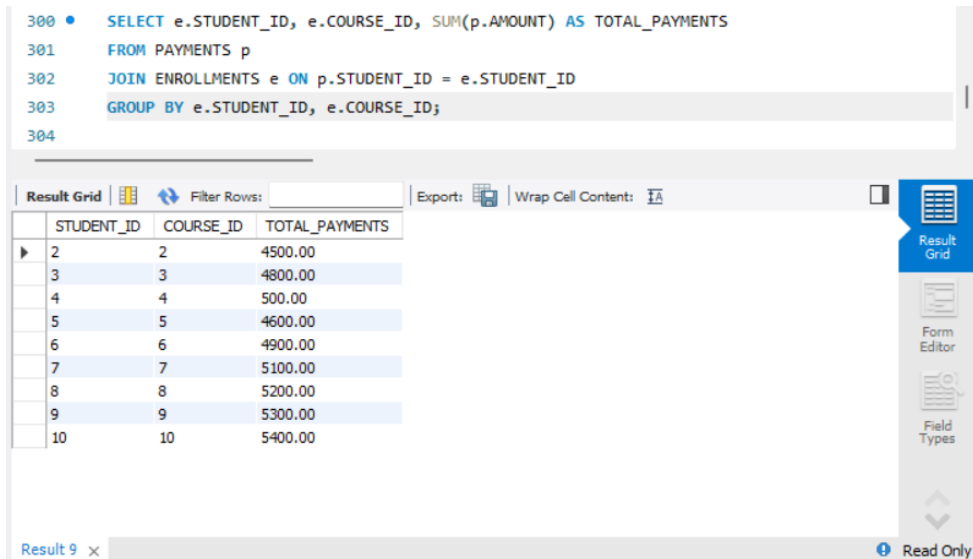
Below the query editor, the 'Result Grid' is displayed, showing the results of the query:

COURSE_ID	COURSE_NAME
1	JAVA
NULL	NULL

The interface includes a toolbar with options like 'Filter Rows', 'Edit', 'Export/Import', and 'Wrap Cell Content'. A sidebar on the right contains icons for 'Result Grid', 'Form Editor', and 'Field Types'. The bottom status bar indicates 'COURSES 8' and 'Apply'.

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 e.STUDENT_ID, e.COURSE_ID, SUM(p.AMOUNT) AS TOTAL_PAYMENTS
FROM PAYMENTS p
JOIN ENROLLMENTS e ON p.STUDENT_ID = e.STUDENT_ID
GROUP BY e.STUDENT_ID, e.COURSE_ID;
```



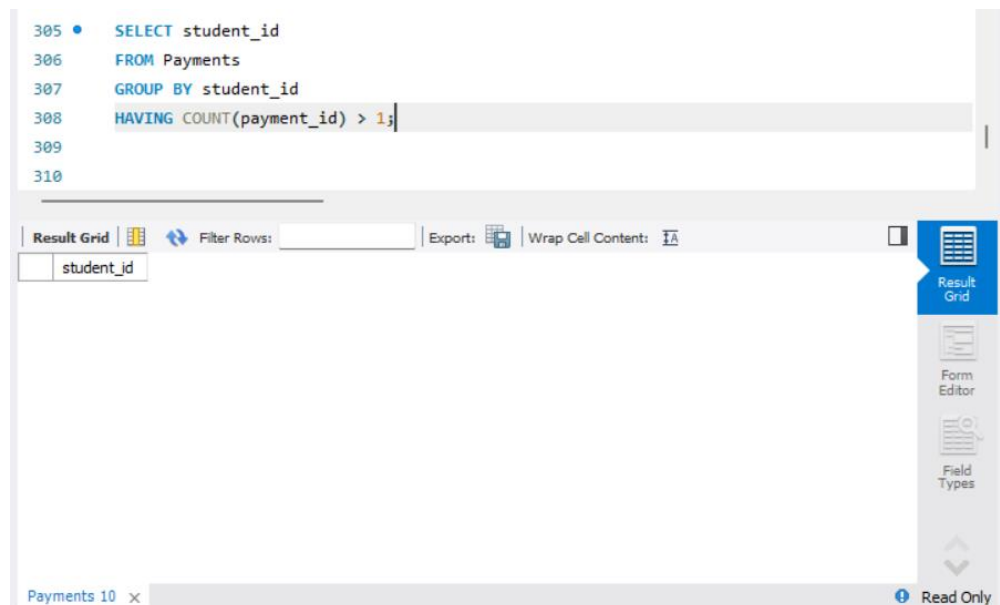
```
300 • SELECT e.STUDENT_ID, e.COURSE_ID, SUM(p.AMOUNT) AS TOTAL_PAYMENTS
301 FROM PAYMENTS p
302 JOIN ENROLLMENTS e ON p.STUDENT_ID = e.STUDENT_ID
303 GROUP BY e.STUDENT_ID, e.COURSE_ID;
304
```

STUDENT_ID	COURSE_ID	TOTAL_PAYMENTS
2	2	4500.00
3	3	4800.00
4	4	500.00
5	5	4600.00
6	6	4900.00
7	7	5100.00
8	8	5200.00
9	9	5300.00
10	10	5400.00

Result 9 x Read Only

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_id
FROM Payments
GROUP BY student_id
HAVING COUNT(payment_id) > 1;
```



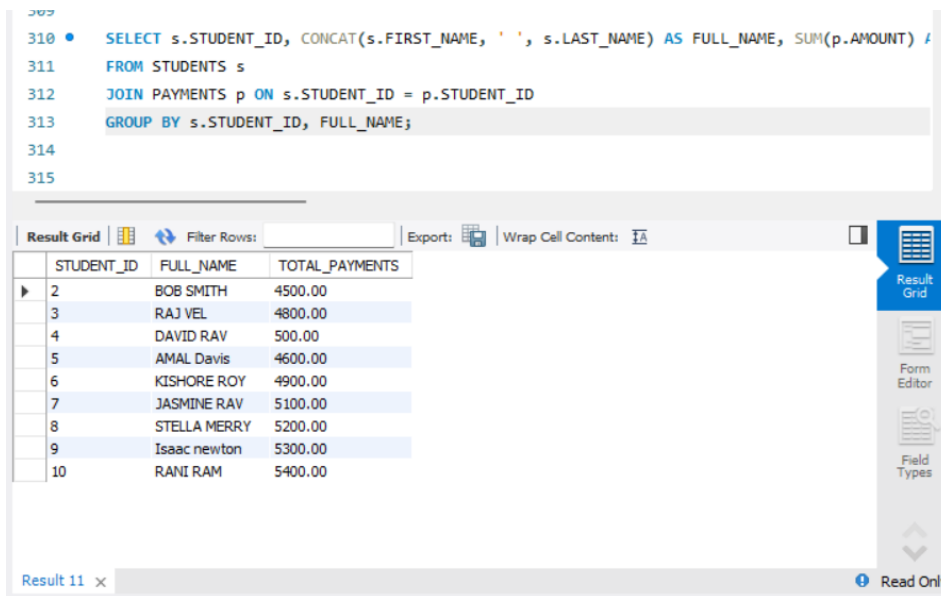
```
305 • SELECT student_id
306 FROM Payments
307 GROUP BY student_id
308 HAVING COUNT(payment_id) > 1;
309
310
```

student_id

Payments 10 x Read Only

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.STUDENT_ID, CONCAT(s.FIRST_NAME, ' ', s.LAST_NAME) AS FULL_NAME,
SUM(p.AMOUNT) AS TOTAL_PAYMENTS
FROM STUDENTS s
JOIN PAYMENTS p ON s.STUDENT_ID = p.STUDENT_ID
GROUP BY s.STUDENT_ID, FULL_NAME;
```

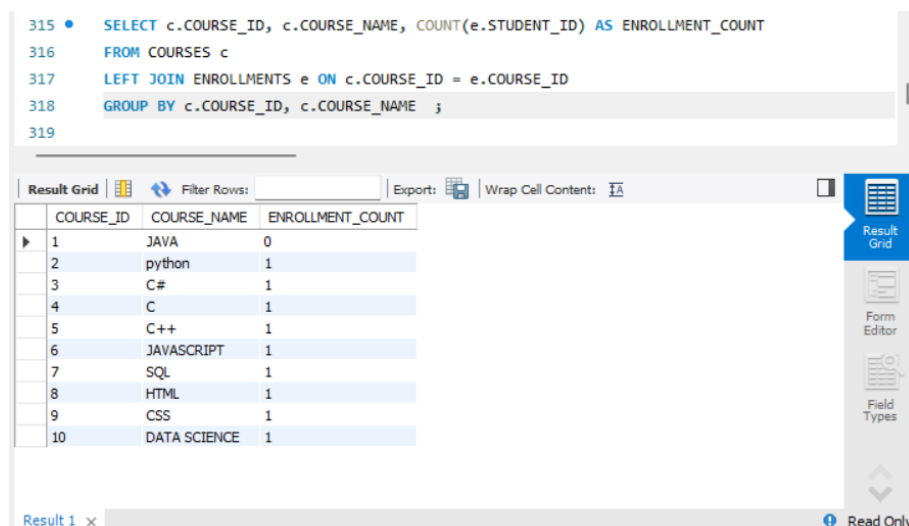


The screenshot shows a SQL IDE interface. The top pane contains the SQL query for question 11. The bottom pane displays the results in a table format. The table has three columns: STUDENT_ID, FULL_NAME, and TOTAL_PAYMENTS. The results are as follows:

STUDENT_ID	FULL_NAME	TOTAL_PAYMENTS
2	BOB SMITH	4500.00
3	RAJ VEL	4800.00
4	DAVID RAV	500.00
5	AMAL Davis	4600.00
6	KISHORE ROY	4900.00
7	JASMINE RAV	5100.00
8	STELLA MERRY	5200.00
9	Isaac newton	5300.00
10	RANI RAM	5400.00

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.

```
SELECT c.COURSE_ID, c.COURSE_NAME, COUNT(e.STUDENT_ID) AS ENROLLMENT_COUNT
FROM COURSES c
LEFT JOIN ENROLLMENTS e ON c.COURSE_ID = e.COURSE_ID
GROUP BY c.COURSE_ID, c.COURSE_NAME;
```

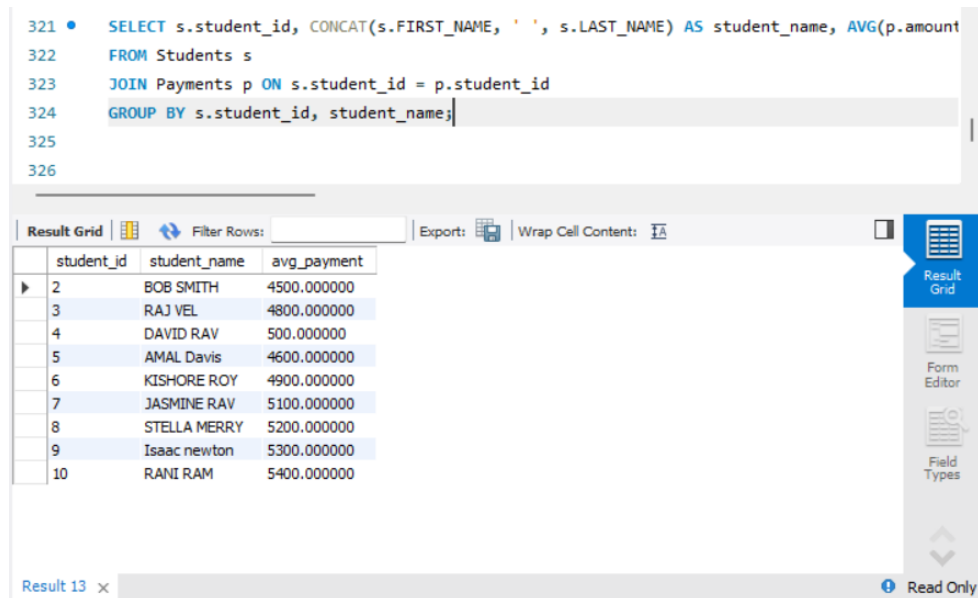


The screenshot shows a SQL IDE interface. The top pane contains the SQL query for question 12. The bottom pane displays the results in a table format. The table has three columns: COURSE_ID, COURSE_NAME, and ENROLLMENT_COUNT. The results are as follows:

COURSE_ID	COURSE_NAME	ENROLLMENT_COUNT
1	JAVA	0
2	python	1
3	C#	1
4	C	1
5	C++	1
6	JAVASCRIPT	1
7	SQL	1
8	HTML	1
9	CSS	1
10	DATA 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.

```
SELECT s.student_id, CONCAT(s.FIRST_NAME, ' ', s.LAST_NAME) AS student_name,
AVG(p.amount) AS avg_payment
FROM Students s
JOIN Payments p ON s.student_id = p.student_id
GROUP BY s.student_id, student_name;
```



The screenshot shows a database query editor with a SQL query in the top pane and its results in a grid below. The query calculates the average payment amount for each student by joining the 'Students' and 'Payments' tables and grouping by student ID and name. The results grid displays 10 rows of data, including student IDs, names, and average payment amounts.

student_id	student_name	avg_payment
2	BOB SMITH	4500.000000
3	RAJ VEL	4800.000000
4	DAVID RAV	500.000000
5	AMAL Davis	4600.000000
6	KISHORE ROY	4900.000000
7	JASMINE RAV	5100.000000
8	STELLA MERRY	5200.000000
9	Isaac newton	5300.000000
10	RANI RAM	5400.000000