

Joins

Question 1 Inner join:

Retrieve the list of students and their enrolled courses.

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure, including schemas, tables, and views. The main window is the Query Editor, showing a SQL query for an inner join between the 'students' and 'enrollments' tables, joined to the 'courses' table. The query is executed, and the results are displayed in the Data Output pane.

```
1 --Question 1 Inner join: Retrieve the list of students and their enrolled courses.
2 SELECT s.student_name, c.course_name
3 FROM students s
4 JOIN enrollments e ON e.student_id = s.student_id
5 JOIN courses c ON c.course_id = e.course_id
6
7 --Question 2 Left Join: List all students and their enrolled courses, including those who haven't enrolled in any course.
8 --Question 3 Right Join: Display all courses and the students enrolled in each course, including courses with no enrolled students.
9 --Question 4 Self Join: Find pairs of students who are enrolled in at least one common course.
10 --Question 5 Complex Join: Retrieve students who are enrolled in 'Introduction to CS' but not in 'Data Structures'.
```

| student_name | course_name |
|--------------|--------------------|
| Alice | Introduction to CS |
| Bob | Biology Basics |
| Charlie | World History |
| Alice | Data Structures |
| Diana | Calculus I |
| Bob | Introduction to CS |
| Charlie | Data Structures |
| Diana | Introduction to CS |
| Alice | Calculus I |
| Bob | Calculus I |

Successfully run. Total query runtime: 152 msec. 10 rows affected.

Question 2 Left Join:

List all students and their enrolled courses, including those who haven't enrolled in any course.

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure. The main window is the Query Editor, showing a SQL query for a left join between the 'students' and 'enrollments' tables, joined to the 'courses' table. The query is executed, and the results are displayed in the Data Output pane.

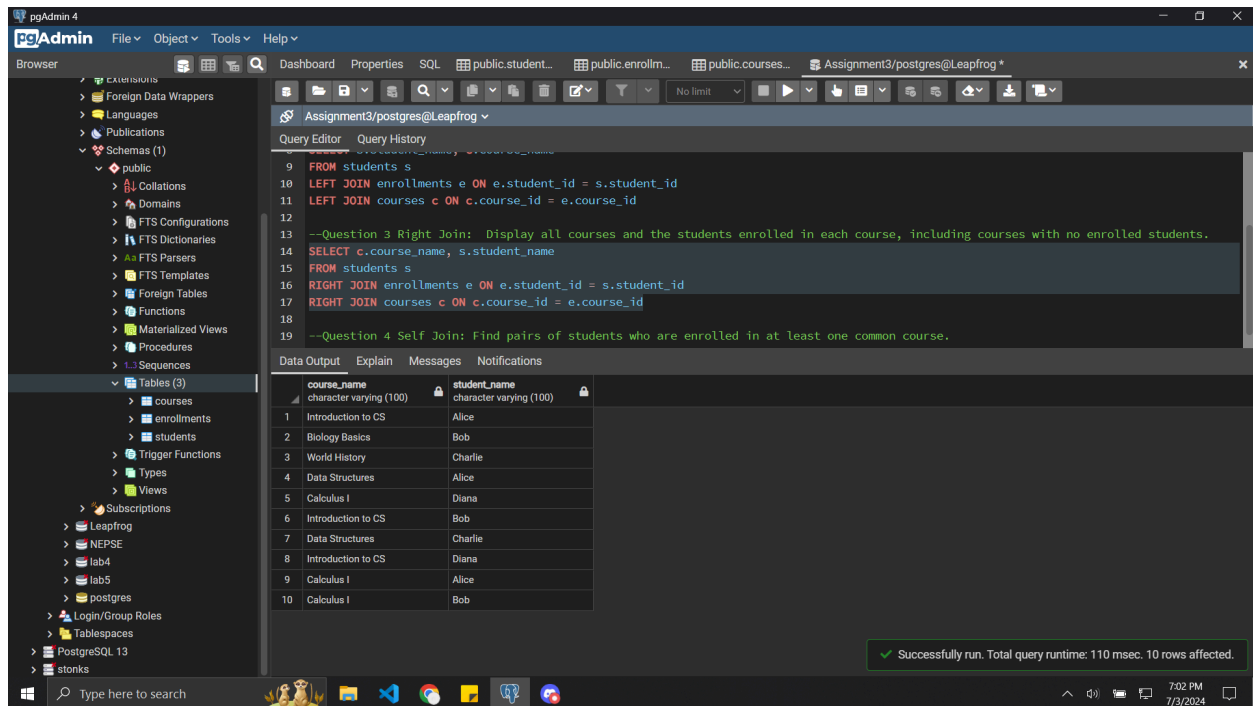
```
4 JOIN enrollments e ON e.student_id = s.student_id
5 JOIN courses c ON c.course_id = e.course_id
6
7 --Question 2 Left Join: List all students and their enrolled courses, including those who haven't enrolled in any course.
8 SELECT s.student_name, c.course_name
9 FROM students s
10 LEFT JOIN enrollments e ON e.student_id = s.student_id
11 LEFT JOIN courses c ON c.course_id = e.course_id
12
13 --Question 3 Right Join: Display all courses and the students enrolled in each course, including courses with no enrolled students.
14 --Question 4 Self Join: Find pairs of students who are enrolled in at least one common course.
```

| student_name | course_name |
|--------------|--------------------|
| Alice | Introduction to CS |
| Bob | Biology Basics |
| Charlie | World History |
| Alice | Data Structures |
| Diana | Calculus I |
| Bob | Introduction to CS |
| Charlie | Data Structures |
| Diana | Introduction to CS |
| Alice | Calculus I |
| Bob | Calculus I |

Successfully run. Total query runtime: 140 msec. 10 rows affected.

Question 3 Right Join:

Display all courses and the students enrolled in each course, including courses with no enrolled students.



The screenshot shows the pgAdmin 4 interface with a SQL query editor. The query is as follows:

```
9 FROM students s
10 LEFT JOIN enrollments e ON e.student_id = s.student_id
11 LEFT JOIN courses c ON c.course_id = e.course_id
12
13 --Question 3 Right Join: Display all courses and the students enrolled in each course, including courses with no enrolled students.
14 SELECT c.course_name, s.student_name
15 FROM students s
16 RIGHT JOIN enrollments e ON e.student_id = s.student_id
17 RIGHT JOIN courses c ON c.course_id = e.course_id
18
19 --Question 4 Self Join: Find pairs of students who are enrolled in at least one common course.
```

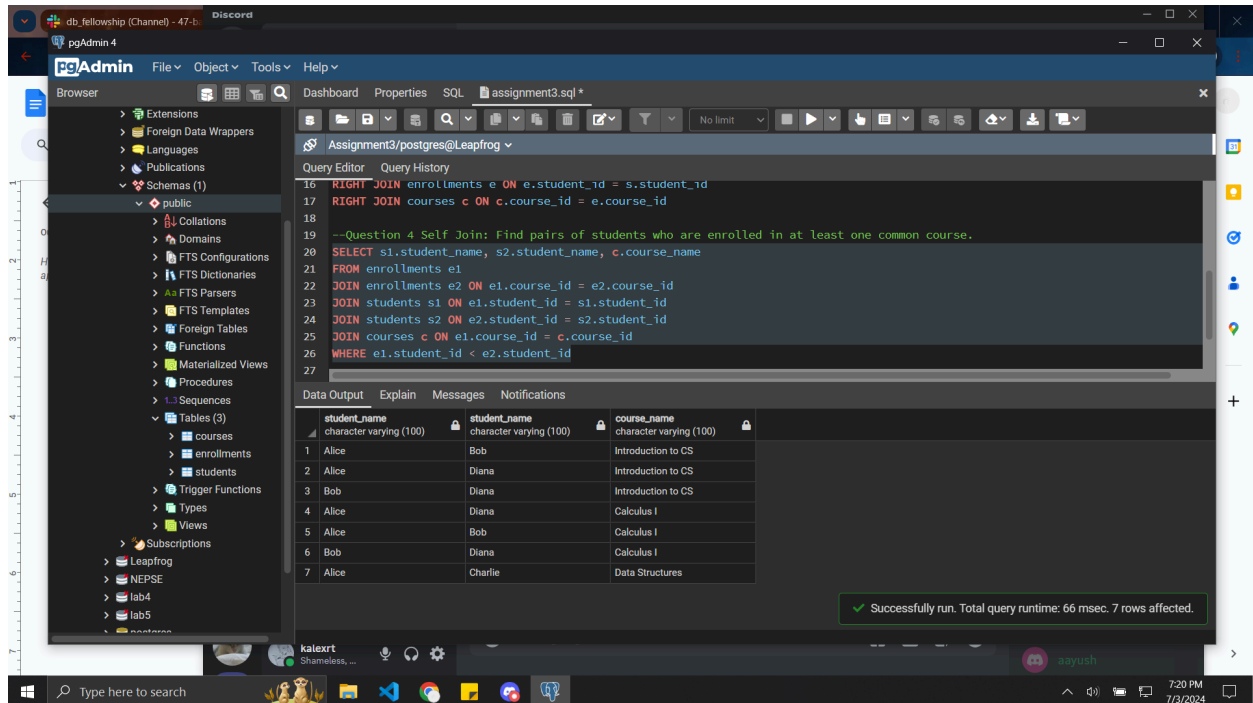
The Data Output tab shows the following results:

| course_name | student_name |
|--------------------|--------------|
| Introduction to CS | Alice |
| Biology Basics | Bob |
| World History | Charlie |
| Data Structures | Alice |
| Calculus I | Diana |
| Introduction to CS | Bob |
| Data Structures | Charlie |
| Introduction to CS | Diana |
| Calculus I | Alice |
| Calculus I | Bob |

Successfully run. Total query runtime: 110 msec. 10 rows affected.

Question 4 Self Join:

Find pairs of students who are enrolled in at least one common course.



The screenshot shows the pgAdmin 4 interface with a SQL query editor. The query is as follows:

```
16 RIGHT JOIN enrollments e ON e.student_id = s.student_id
17 RIGHT JOIN courses c ON c.course_id = e.course_id
18
19 --Question 4 Self Join: Find pairs of students who are enrolled in at least one common course.
20 SELECT s1.student_name, s2.student_name, c.course_name
21 FROM enrollments e1
22 JOIN enrollments e2 ON e1.course_id = e2.course_id
23 JOIN students s1 ON e1.student_id = s1.student_id
24 JOIN students s2 ON e2.student_id = s2.student_id
25 JOIN courses c ON e1.course_id = c.course_id
26 WHERE e1.student_id < e2.student_id
27
```

The Data Output tab shows the following results:

| student_name | student_name | course_name |
|--------------|--------------|--------------------|
| Alice | Bob | Introduction to CS |
| Alice | Diana | Introduction to CS |
| Bob | Diana | Introduction to CS |
| Alice | Diana | Calculus I |
| Alice | Bob | Calculus I |
| Bob | Diana | Calculus I |
| Alice | Charlie | Data Structures |

Successfully run. Total query runtime: 66 msec. 7 rows affected.

Question 5 Complex Join:

Retrieve students who are enrolled in 'Introduction to CS' but not in 'Data Structures'.

The screenshot shows the pgAdmin 4 interface with a SQL query editor. The query is a complex join designed to find students enrolled in 'Introduction to CS' but not in 'Data Structures'. The query uses multiple joins between the students, enrollments, and courses tables, and includes a subquery to exclude students enrolled in 'Data Structures'.

```
21 FROM enrollments e1
22 JOIN enrollments e2 ON e1.course_id = e2.course_id
23 JOIN students s1 ON e1.student_id = s1.student_id
24 JOIN students s2 ON e2.student_id = s2.student_id
25 JOIN courses c ON e1.course_id = c.course_id
26 WHERE e1.student_id < e2.student_id
27
28 --Question 5 Complex Join: Retrieve students who are enrolled in 'Introduction to CS' but not in 'Data Structures'.
29 SELECT s.student_name
30 FROM students s
31 JOIN enrollments e ON e.student_id = s.student_id
32 JOIN courses c ON c.course_id = e.course_id
33 WHERE c.course_name = 'Introduction to CS'
34 AND s.student_name NOT IN(
35     SELECT s.student_name
36     FROM students s
37     JOIN enrollments e ON e.student_id = s.student_id
38     JOIN courses c ON c.course_id = e.course_id
39     WHERE c.course_name = 'Data Structures')
40
```

The Data Output tab shows the results of the query:

| student_name |
|--------------|
| 1 Bob |
| 2 Diana |

Successfully run. Total query runtime: 99 msec. 2 rows affected.

Window Functions

1. Using ROW_NUMBER():

Question: List all students along with a row number based on their enrollment date in ascending order.

The screenshot shows the pgAdmin 4 interface with a SQL query editor. The query uses the ROW_NUMBER() window function to assign a unique row number to each student based on their enrollment date in ascending order.

```
39 WHERE c.course_name = 'Data Structures'
40 )
41 -- List all students along with a row number based on their enrollment date in ascending order.
42 SELECT s.student_name, e.enrollment_date,
43 ROW_NUMBER() OVER (ORDER BY e.enrollment_date) AS row_num
44 FROM Students s
45 JOIN Enrollments e ON s.student_id = e.student_id
46 ORDER BY e.enrollment_date;
47
```

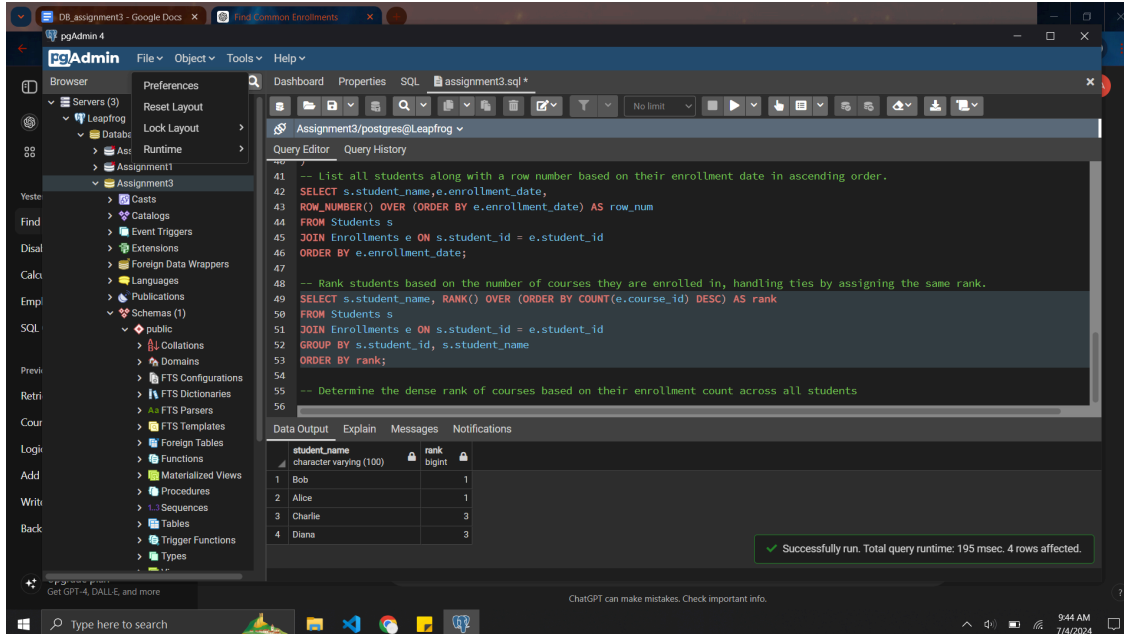
The Data Output tab shows the results of the query:

| student_name | enrollment_date | row_num |
|--------------|-----------------|---------|
| 1 Alice | 2023-01-15 | 1 |
| 2 Bob | 2023-01-20 | 2 |
| 3 Charlie | 2023-02-01 | 3 |
| 4 Alice | 2023-02-05 | 4 |
| 5 Diana | 2023-02-10 | 5 |
| 6 Bob | 2023-02-12 | 6 |
| 7 Charlie | 2023-02-15 | 7 |
| 8 Diana | 2023-02-20 | 8 |
| 9 Alice | 2023-03-01 | 9 |
| 10 Bob | 2023-03-05 | 10 |

Successfully run. Total query runtime: 63 msec. 10 rows affected.

2. Using **RANK()**:

Question: Rank students based on the number of courses they are enrolled in, handling ties by assigning the same rank.



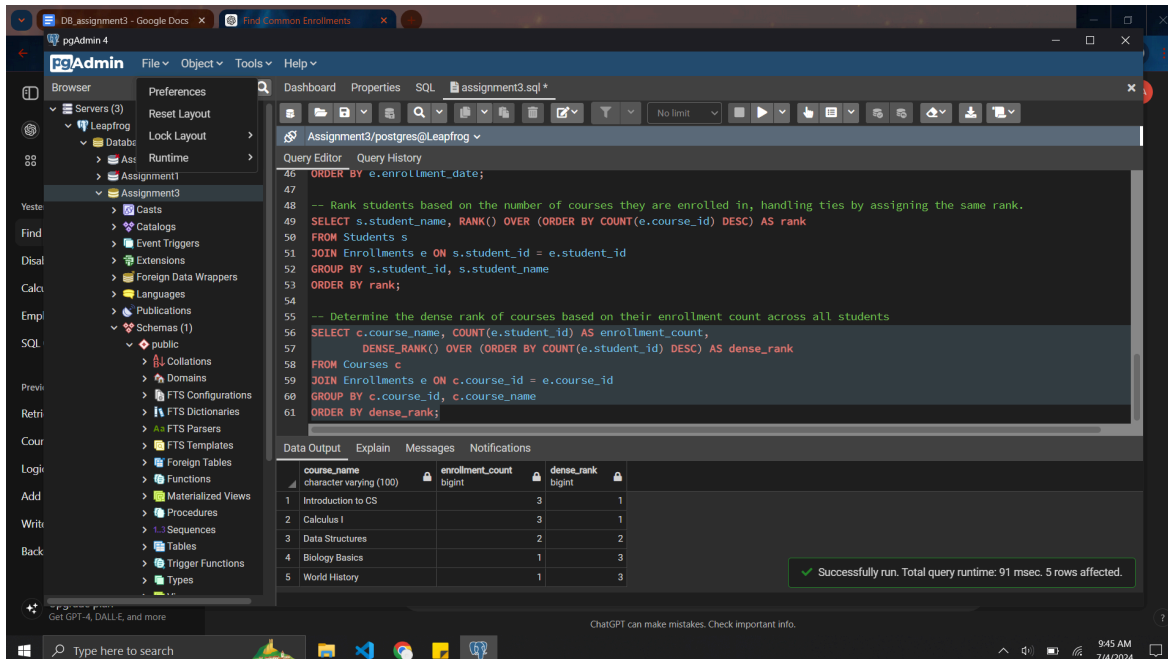
```
41 -- List all students along with a row number based on their enrollment date in ascending order.
42 SELECT s.student_name, e.enrollment_date,
43 ROW_NUMBER() OVER (ORDER BY e.enrollment_date) AS row_num
44 FROM Students s
45 JOIN Enrollments e ON s.student_id = e.student_id
46 ORDER BY e.enrollment_date;
47
48 -- Rank students based on the number of courses they are enrolled in, handling ties by assigning the same rank.
49 SELECT s.student_name, RANK() OVER (ORDER BY COUNT(e.course_id) DESC) AS rank
50 FROM Students s
51 JOIN Enrollments e ON s.student_id = e.student_id
52 GROUP BY s.student_id, s.student_name
53 ORDER BY rank;
54
55 -- Determine the dense rank of courses based on their enrollment count across all students
56
```

| student_name | rank |
|--------------|------|
| Bob | 1 |
| Alice | 1 |
| Charlie | 3 |
| Diana | 3 |

Successfully run. Total query runtime: 195 msec. 4 rows affected.

3. Using **DENSE_RANK()**:

Question: Determine the dense rank of courses based on their enrollment count across all students



```
46 ORDER BY e.enrollment_date;
47
48 -- Rank students based on the number of courses they are enrolled in, handling ties by assigning the same rank.
49 SELECT s.student_name, RANK() OVER (ORDER BY COUNT(e.course_id) DESC) AS rank
50 FROM Students s
51 JOIN Enrollments e ON s.student_id = e.student_id
52 GROUP BY s.student_id, s.student_name
53 ORDER BY rank;
54
55 -- Determine the dense rank of courses based on their enrollment count across all students
56 SELECT c.course_name, COUNT(e.student_id) AS enrollment_count,
57 DENSE_RANK() OVER (ORDER BY COUNT(e.student_id) DESC) AS dense_rank
58 FROM Courses c
59 JOIN Enrollments e ON c.course_id = e.course_id
60 GROUP BY c.course_id, c.course_name
61 ORDER BY dense_rank;
```

| course_name | enrollment_count | dense_rank |
|--------------------|------------------|------------|
| Introduction to CS | 3 | 1 |
| Calculus I | 3 | 1 |
| Data Structures | 2 | 2 |
| Biology Basics | 1 | 3 |
| World History | 1 | 3 |

Successfully run. Total query runtime: 91 msec. 5 rows affected.