SQL Challenge: Minimum Number of Meeting Rooms Required

Problem Statement

You are given a `meetings` table with the start and end times of several meetings during a day. Your task is to determine the **minimum number of meeting rooms required** so that all meetings can be accommodated without overlapping.

A room can only host one meeting at a time. Meetings may partially or fully overlap, and you need to identify the maximum number of overlapping meetings at any given time — which directly translates to the number of rooms needed.

Input Table: `meetings`

id	start_time	end_time
1	02:00:00	02:59:59
2	03:00:00	03:59:59
3	04:00:00	04:59:59
4	03:00:00	04:59:59

Expected Output

solution
2

In this case, meetings `2` and `4` overlap, so at least **2 rooms** are needed.

SQL Solution

```sql
SELECT MAX(overlap) AS solution
FROM (
    SELECT COUNT(t) AS overlap
    FROM (
        SELECT *
        FROM (
            SELECT start_time AS t FROM meetings
            UNION
            SELECT end_time AS t FROM meetings
        ) AS times
        JOIN meetings AS m
            ON times.t >= m.start_time
            AND times.t < m.end_time
    ) AS all_times
```

```
    GROUP BY t
) AS overlap_counts;
```

Step 1: Understanding the Core Concept

To find the minimum number of rooms needed, we need to identify the maximum number of overlapping meetings at any point in time.

Step 2: The Query Structure

```
SELECT MAX(overlap) AS solution
FROM (
    SELECT COUNT(t) AS overlap
    FROM (
        SELECT *
        FROM (
            SELECT start_time AS t FROM meetings
            UNION
            SELECT end_time AS t FROM meetings
        ) AS times
        JOIN meetings AS m
            ON times.t >= m.start_time
            AND times.t < m.end_time
    ) AS all_times
    GROUP BY t
) AS overlap_counts;
```

This is a complex query with multiple nested subqueries. Let's understand each part:

Step 3: The Innermost Subquery - Creating Time Points

```
SELECT start_time AS t FROM meetings
UNION
SELECT end_time AS t FROM meetings
```

This creates a list of all unique time points (both start times and end times). These are the critical moments when the number of overlapping meetings might change.

For our example data:

- 02:00:00 (start of meeting 1)
- 02:59:59 (end of meeting 1)
- 03:00:00 (start of meetings 2 and 4)
- 03:59:59 (end of meeting 2)
- 04:00:00 (start of meeting 3)
- 04:59:59 (end of meetings 3 and 4)

Step 4: Joining with the Meetings Table

```
SELECT *
FROM (
```

```
    -- Previous subquery giving us time points
) AS times
JOIN meetings AS m
    ON times.t >= m.start_time
    AND times.t < m.end_time
```

For each time point, this JOIN finds all meetings that are active at that time point.

- A meeting is active if the time point is greater than or equal to the start time
- AND the time point is strictly less than the end time (note the < operator, not <=)

Step 5: Counting Overlaps at Each Time Point

```
SELECT COUNT(t) AS overlap
FROM (
    -- Previous subquery giving us time points and active meetings
) AS all_times
GROUP BY t
```

This counts how many meetings are active at each time point and gives us the "overlap" count.

For our example:

- At 02:00:00: Meeting 1 is active (count: 1)
- At 02:59:59: Meeting 1 is active (count: 1)
- At 03:00:00: Meetings 2 and 4 are active (count: 2)
- At 03:59:59: Meetings 2 and 4 are active (count: 2)
- At 04:00:00: Meetings 3 and 4 are active (count: 2)
- At 04:59:59: Meetings 3 and 4 are active (count: 2)

Step 6: Finding the Maximum Overlap

```
SELECT MAX(overlap) AS solution
FROM (
    -- Previous subquery giving us overlap counts
) AS overlap_counts;
```

Finally, we take the maximum of all these overlap counts, which gives us the minimum number of rooms needed.

In our example, the maximum overlap is 2, so we need at least 2 meeting rooms.

Important Considerations

1. Note that we're using `times.t < m.end_time` not `<=`. This is because a meeting ending at exactly the same time another is starting doesn't create a conflict.
2. The `UNION` operator removes duplicates. If we wanted to keep duplicates (not needed here), we would use `UNION ALL`.

3. The result of 2 matches the expected output because meetings 2 and 4 overlap from 3:00 to 4:00, requiring 2 rooms.