SQL Query Challenge: Count Passengers per Bus

Problem Statement

You are given two tables: `buses` and `passengers`. Each row in the `buses` table represents a scheduled bus, and each row in the `passengers` table represents a passenger arriving at the station.

Your task is to determine the number of passengers who can board **each bus**, based on the following conditions:

- A passenger can only board a bus if:
  - Their **origin** and **destination** match the bus's origin and destination.
  - Their **arrival time** is **greater than** the departure time of the **previous bus** on the same route or there is no previous bus.
  - Their **arrival time** is **less than or equal to** the departure time of the current bus.
  - If **only one** bus exists for a route, all passengers arriving **on or before** that bus can be considered.
- If a bus has no valid passengers, the count should be zero.

Sample Tables

`buses` Table

id	origin	destination	time
1	berlin	paris	10:12
2	paris	berlin	10:15
3	berlin	paris	10:18
4	milan	paris	10:20

`passengers` Table

id	origin	destination	time
1	berlin	paris	10:11
2	paris	berlin	10:14
3	berlin	paris	10:12
4	milan	paris	10:21

Expected Result

id	num_of_passengers
1	2

2	1
3	0
4	0

SQL Solution

```sql
SELECT
    b.id,
    COUNT(p.id) AS num_passengers
FROM (
    SELECT
        b.*,
        LAG(time) OVER (PARTITION BY origin, destination ORDER BY time) AS prev_time
    FROM buses b
) b
LEFT JOIN passengers p
    ON p.origin = b.origin AND p.destination = b.destination
    AND (p.time > b.prev_time OR b.prev_time IS NULL)
    AND (p.time <= b.time)
GROUP BY b.id, b.origin, b.destination, b.time
ORDER BY b.id ASC;
```

Understanding the Problem

We need to:

1. Count passengers for each bus
2. A passenger can board a bus if:
    - Their origin and destination match the bus's route
    - They arrive after the previous bus on the same route left (or there's no previous bus)
    - They arrive before or exactly when the current bus departs

Breaking Down the Solution

Step 1: Finding Previous Bus Departure Times

The first part of the query finds the previous bus departure time for each route:

```sql
SELECT
    b.*,
    LAG(time) OVER (PARTITION BY origin, destination ORDER BY time) AS prev_time
FROM buses b
```

This uses the `LAG()` window function to look at the previous row in a sorted partition. Let's see what this produces:

id	origin	destination	time	prev_time
1	berlin	paris	10:12	NULL

3	berlin	paris	10:18	10:12
4	milan	paris	10:20	NULL
2	paris	berlin	10:15	NULL

Notice:

- Bus #3 has prev_time=10:12 (from Bus #1) because they share the same route (berlin→paris)
- Bus #1, #4, and #2 have NULL for prev_time because they're the first buses on their respective routes

Step 2: Joining with Passengers Table

Next, we join this enhanced buses table with passengers to find valid matches:

```
LEFT JOIN passengers p
    ON p.origin = b.origin AND p.destination = b.destination
    AND (p.time > b.prev_time OR b.prev_time IS NULL)
    AND (p.time <= b.time)
```

This join has several conditions:

1. `p.origin = b.origin AND p.destination = b.destination` - Match routes
2. `(p.time > b.prev_time OR b.prev_time IS NULL)` - Passenger arrived after previous bus or no previous bus exists
3. `(p.time <= b.time)` - Passenger arrived before or exactly when current bus departs

Let's analyze what happens with our sample data:

For Bus #1 (berlin→paris, 10:12):

- prev_time is NULL (first bus on route)
- Passenger #1 (berlin→paris, 10:11): Matches (arrived before bus departs)
- Passenger #3 (berlin→paris, 10:12): Matches (arrived exactly when bus departs)

For Bus #2 (paris→berlin, 10:15):

- prev_time is NULL (first bus on route)
- Passenger #2 (paris→berlin, 10:14): Matches (arrived before bus departs)

For Bus #3 (berlin→paris, 10:18):

- prev_time is 10:12 (previous bus #1)
- Passenger #1 (berlin→paris, 10:11): Doesn't match (arrived before previous bus departed)
- Passenger #3 (berlin→paris, 10:12): Doesn't match (arrived exactly when previous bus departed)
- No other matching passengers

For Bus #4 (milan→paris, 10:20):

- prev_time is NULL (first bus on route)
- Passenger #4 (milan→paris, 10:21): Doesn't match (arrived after bus departed)

Step 3: Counting and Grouping

Finally, we count passengers per bus and group the results:

```
SELECT
    b.id,
    COUNT(p.id) AS num_passengers
FROM (...) b
LEFT JOIN passengers p ...
GROUP BY b.id, b.origin, b.destination, b.time
ORDER BY b.id ASC;
```

This counts the number of passenger records that match each bus after applying all conditions. We use `LEFT JOIN` to ensure that buses with no passengers still appear in the results with a count of `0`.

The result: | id | num_of_passengers | |----|-------------------| | 1 | 2 | | 2 | 1 | | 3 | 0 | | 4 | 0 |

Why This Works (Detailed Explanation)

1. **LAG() Window Function**: This is crucial for comparing with the previous bus. It gives us access to the departure time of the previous bus on the same route.
2. **LEFT JOIN**: Ensures all buses appear in the results, even if they have no matching passengers.
3. **Complex JOIN Conditions**: The multiple conditions in the JOIN ensure we only match passengers who:
    - Are traveling on the same route as the bus
    - Arrived after the previous bus on that route departed (or there was no previous bus)
    - Arrived on or before the current bus departs
4. **COUNT(p.id)**: When using LEFT JOIN, this counts only non-NULL values. If no passengers match, this returns 0.
5. **Grouping**: By grouping on bus attributes, we ensure one row per bus in the final output.

This is a practical example of window functions and complex JOIN conditions working together to solve a real-world scheduling problem.