SQL Challenge: Get the Nth Highest Salary (Stored Function)

Problem Statement

You are given an `Employee` table containing employee salaries. Write a **stored function** `getNthHighestSalary(N)` that returns the **N-th highest unique salary** from the `Employee` table.

- If the N-th highest salary does not exist (e.g., not enough distinct salaries), return `NULL`.

Input Table: `Employee`

id	salary
200	...
300	...
400	...
100	...
600	...

Example

Assuming the following values:

salary
600
400
300
200
100

If `N = 3`, the result should be:

result
400

SQL Function

```
CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT
BEGIN
```

```
  RETURN (
    SELECT COALESCE(
      (
        SELECT salary
        FROM (
          SELECT DISTINCT salary,
                 DENSE_RANK() OVER (ORDER BY salary DESC) AS r
          FROM Employee
        ) t
        WHERE r = N
      ), NULL
    )
  );
END
```

Nth Highest Salary Function Explanation

Let's break down the `getNthHighestSalary` function step by step to understand how it works.

The Problem

We need to find the Nth highest **unique** salary from the Employee table. For example, if we have salaries [600, 400, 300, 200, 100] and N = 3, we should return 400.

The Function Structure

```
CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT
BEGIN
  RETURN (
    SELECT COALESCE(
      (
        SELECT salary
        FROM (
          SELECT DISTINCT salary,
                 DENSE_RANK() OVER (ORDER BY salary DESC) AS r
          FROM Employee
        ) t
        WHERE r = N
      ), NULL
    )
  );
END
```

Let's analyze each part:

Step 1: Function Declaration

```
CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT
BEGIN
```

```
  -- Function body
END
```

This creates a stored function called `getNthHighestSalary` that:

- Takes an integer parameter `N`
- Returns an integer value
- The code between `BEGIN` and `END` is executed when the function is called

Step 2: The Inner Subquery - Ranking Salaries

```
SELECT DISTINCT salary,
       DENSE_RANK() OVER (ORDER BY salary DESC) AS r
FROM Employee
```

This part:

1. Takes all salaries from the Employee table
2. Uses `DISTINCT` to ensure we only consider unique salaries
3. Assigns a rank to each unique salary using `DENSE_RANK()`
4. Orders them by salary in descending order (highest to lowest)
5. Labels this rank as `r`

The `DENSE_RANK()` window function is crucial here - it assigns rankings without gaps. For example:

salary	DENSE_RANK()
600	1
400	2
300	3
200	4
100	5

Step 3: Finding the Nth Rank

```
SELECT salary
FROM (
  -- Inner subquery from Step 2
) t
WHERE r = N
```

This part:

1. Uses the result from Step 2 as a derived table named `t`
2. Filters to find only the row where the rank `r` equals our input parameter `N`
3. Returns just the salary value from that row

Step 4: Handling No Result with COALESCE

```
SELECT COALESCE(
  (
    -- Subquery from Step 3
  ), NULL
)
```

The `COALESCE` function:

1. Takes multiple arguments and returns the first non-NULL value
2. If the subquery from Step 3 returns no result (which means NULL), it uses the second argument NULL
3. This might seem redundant, but it's a clear way to show the intent - if no Nth highest salary exists, return NULL

Example Walkthrough

Let's trace through the function with our example data and N = 3:

1. We have salaries: [600, 400, 300, 200, 100]
2. After applying DISTINCT and DENSE_RANK():

salary	r (rank)
600	1
400	2
300	3
200	4
100	5

3. We filter for r = 3, which gives us the salary 300
4. COALESCE returns 300 (since it's not NULL)
5. The function returns 300

What if N = 10?

1. We still have ranks 1 through 5 only
2. Filtering for r = 10 returns no rows
3. The subquery returns NULL
4. COALESCE returns NULL (the second argument)
5. The function returns NULL

Key Concepts to Understand

1. **DISTINCT**: Removes duplicate salaries to get unique values
2. **DENSE_RANK()**: Window function that assigns ranks without gaps
3. **Subqueries**: Allows us to use results from one query inside another
4. **COALESCE**: Provides a way to handle NULL results elegantly