SQL

Windows Function in SQL

https://www.linkedin.com/in/nivetha-anand/

01 →

What is a window function and what does it do?

A Windows function performs **calculations across a set of rows related to the current row**, unlike the aggregate function which group rows into a single result. Windows functions allows you to **calculate values like running totals, moving averages and ranking without grouping rows.**

https://www.linkedin.com/in/nivetha-anand/

Key features of Windows function:

- **Calculations across rows:** Window functions allow you to perform calculations that **consider the context of multiple rows, such as calculating a running total or a moving average.**

- **Maintain individual rows:** Unlike aggregate functions, **window functions do not group rows into a single output row; instead, they produce a result for each row in the result set**.

- **Use of the OVER clause:** Window functions are identified by the **OVER clause**, which defines the window frame (the set of rows to which the calculation applies).

Common use cases:

- **Running totals:** Calculate the sum of values up to the current row.
- **Moving averages:** Calculate the average of values within a specific window of rows.
- **Rankings:** Assign ranks to rows based on a specific criteria.
- **Partitioning:** Divide the data into partitions and perform calculations within each partition.

04

Syntax

```
SELECT column_name,
    FUNCTION_NAME(expression) OVER (
    PARTITION BY column_name
    ORDER BY column_name
    ROWS/RANGE frame_specification)
```

05

Syntax

🛠️ Breakdown of Syntax Components:

- FUNCTION_NAME(expression) → The aggregation or ranking function applied to the selected rows (e.g., SUM(), AVG(), RANK(), LAG(), etc.).
- OVER() → Specifies the window (or subset of rows) over which the function operates.
- PARTITION BY → (Optional) Divides the result set into smaller groups before applying the function.
- ORDER BY → (Optional) Defines the order of rows within each partition.
- ROWS/RANGE frame specification → (Optional) Defines a subset of rows within the partition for calculations like running totals or moving averages.

When and where to use some of the below commonly used functions:

1️⃣ LEAD()

- Use case: Fetch the next row’s value without using a self-join.
- Example: Finding the next month’s revenue for each month in a sales dataset.

2️⃣ LAG()

- Use case: Fetch the previous row’s value, useful for comparing trends.
- Example: Finding the previous month’s revenue to calculate month-over-month change.

3️⃣ RANK()

- Use case: Assigns a rank with gaps if there are duplicate values.
- Example: Ranking students by scores; if two students have the same score, they get the same rank, but the next rank is skipped (1,1,3...).

When and where to use some of the below commonly used functions:

4️⃣ DENSE_RANK()

- Use case: Assigns a continuous rank without gaps.
- Example: Similar to RANK(), but if two students tie, the next rank follows sequentially (1,1,2...).

5️⃣ ROW_NUMBER()

- Use case: Assigns a unique row number within each partition.
- Example: When selecting the latest transaction per customer.

Window Function Problems:

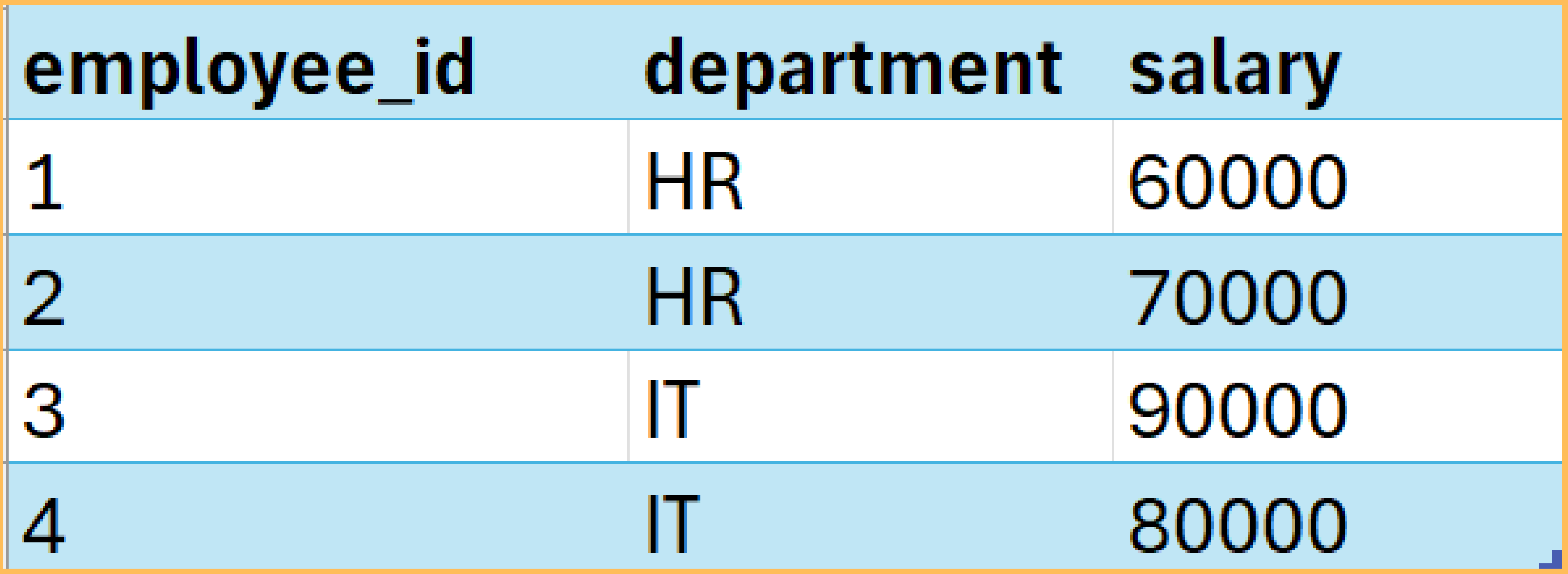
Row Number()

📌 Problem:

Assign a unique row number to employees within each department based on salary in descending order.

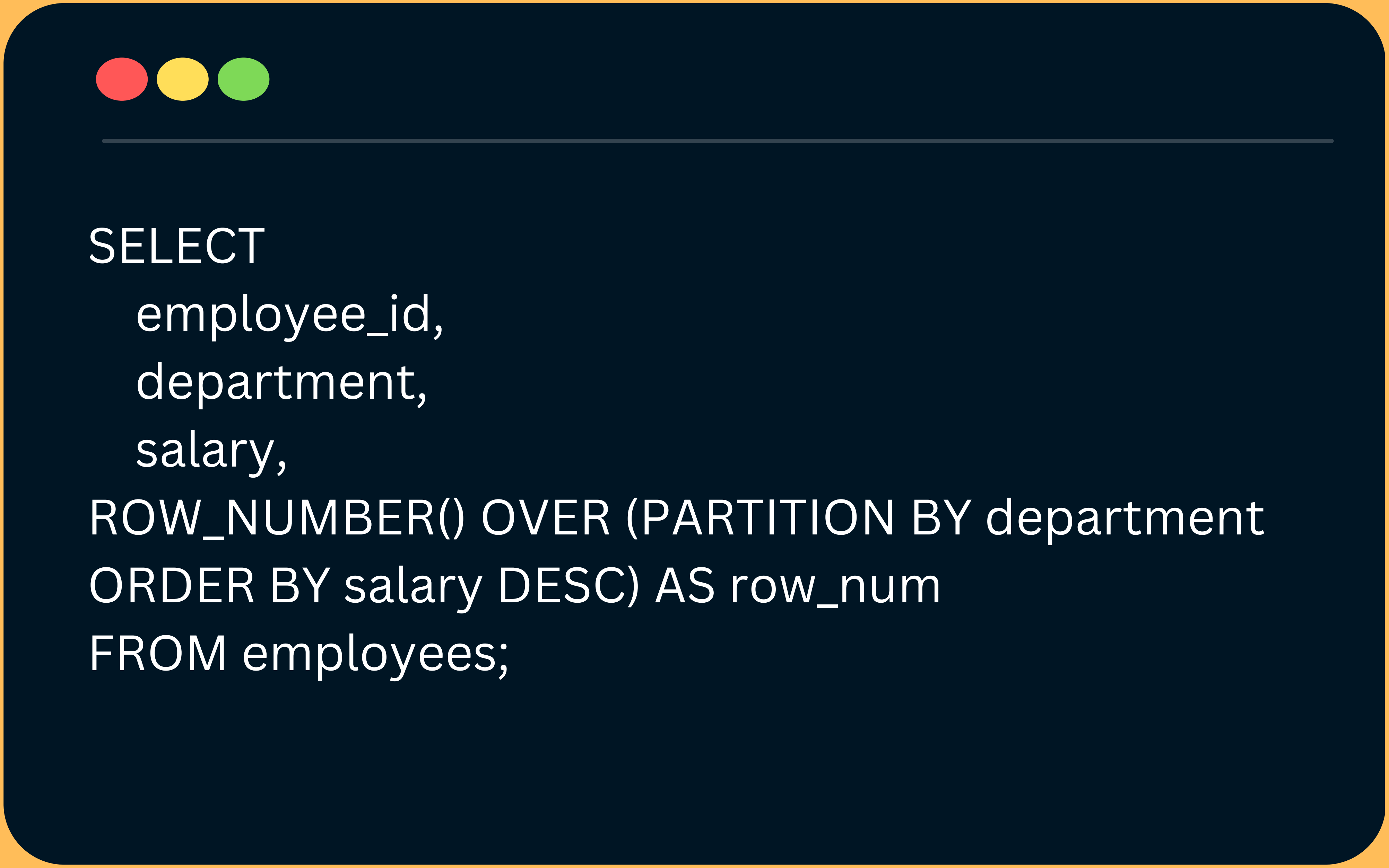
Window Function Problems:

📊 DATA SET (EMPLOYEES TABLE)



employee_id	department	salary
1	HR	60000
2	HR	70000
3	IT	90000
4	IT	80000

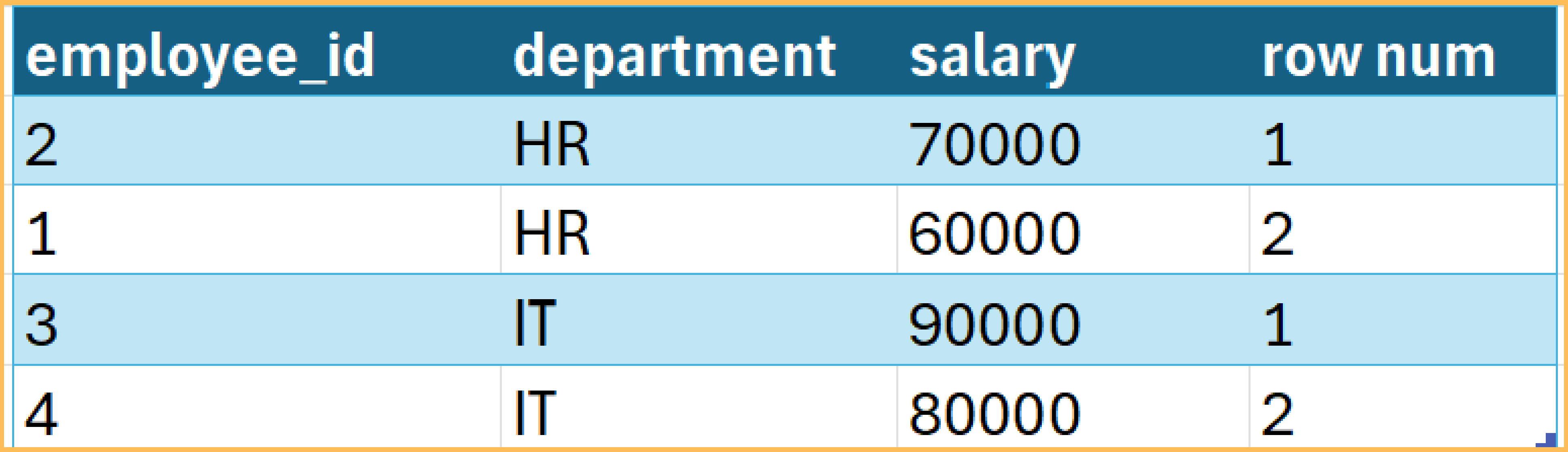
10



```
SELECT
    employee_id,
    department,
    salary,
ROW_NUMBER() OVER (PARTITION BY department
ORDER BY salary DESC) AS row_num
FROM employees;
```

Window Function Problems:

✅ EXPECTED OUTPUT



employee_id	department	salary	row num
2	HR	70000	1
1	HR	60000	2
3	IT	90000	1
4	IT	80000	2

Window Function Problems:

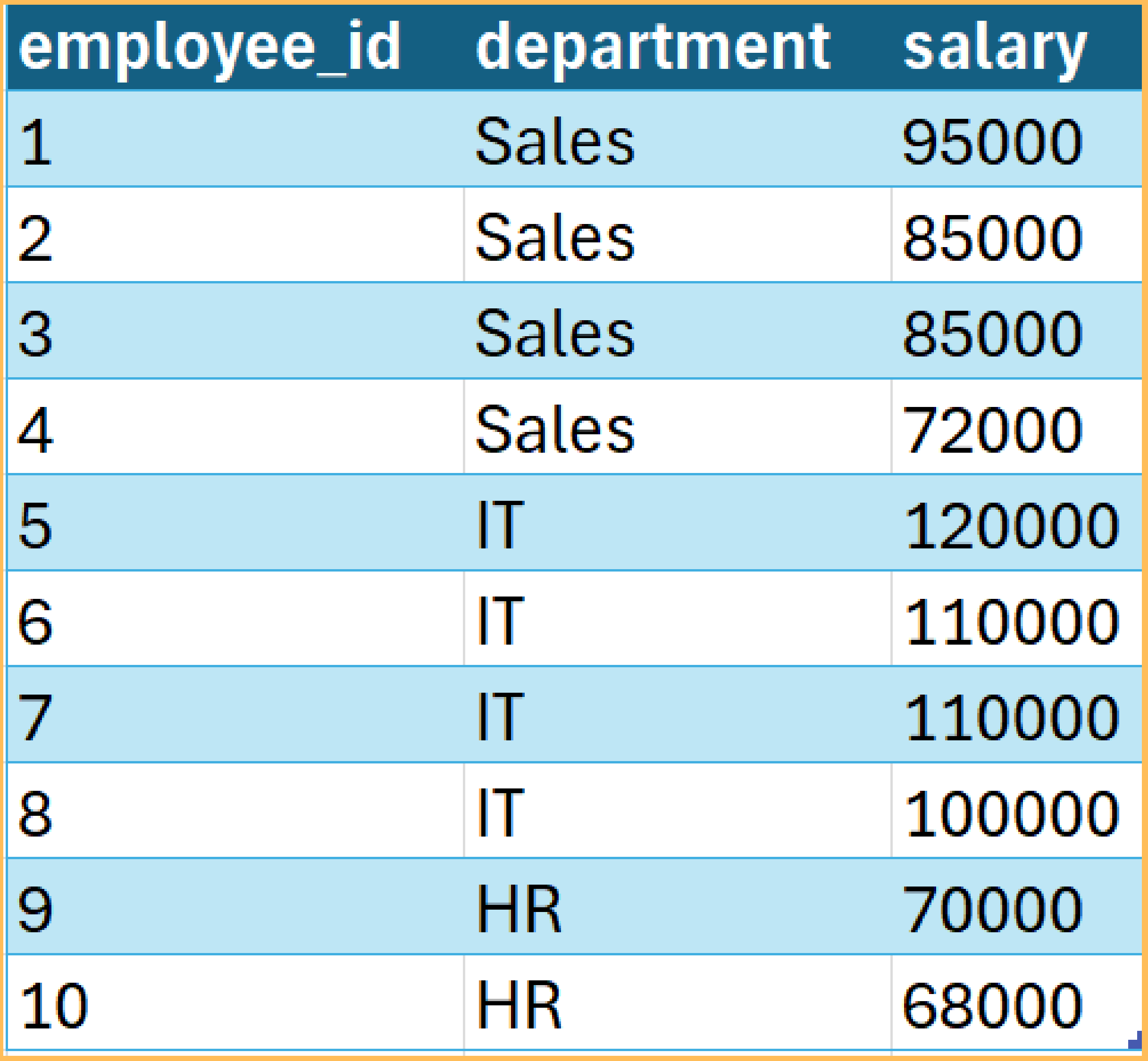
DENSE_RANK()

📌 Problem

Rank employees within each department **based on salary in descending order**. If two employees have the same salary, they should receive the same rank, and the next rank should NOT be skipped.

Window Function Problems:

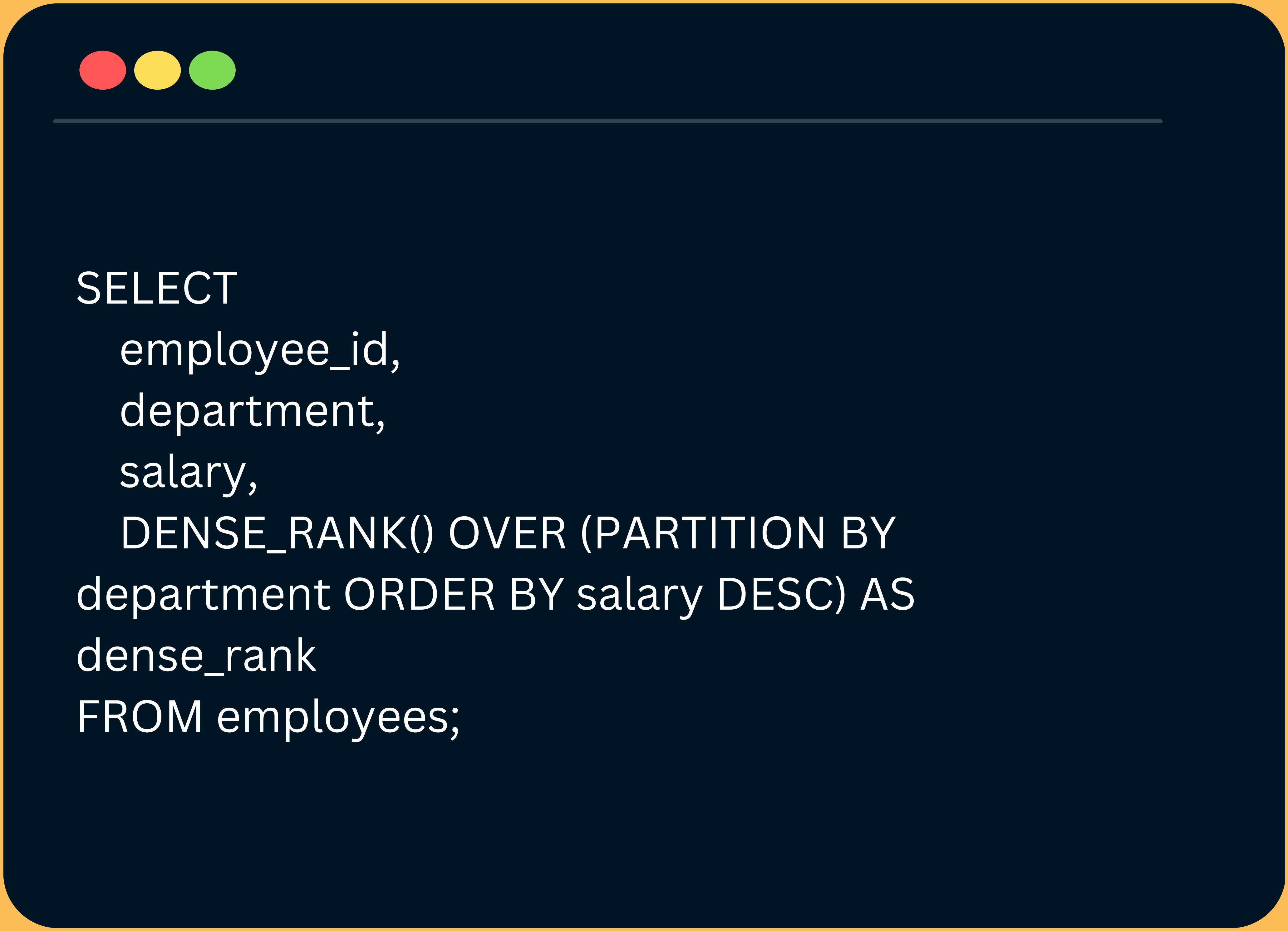
📊 DATA SET (EMPLOYEES TABLE)



employee_id	department	salary
1	Sales	95000
2	Sales	85000
3	Sales	85000
4	Sales	72000
5	IT	120000
6	IT	110000
7	IT	110000
8	IT	100000
9	HR	70000
10	HR	68000

https://www.linkedin.com/in/nivetha-anand/

06

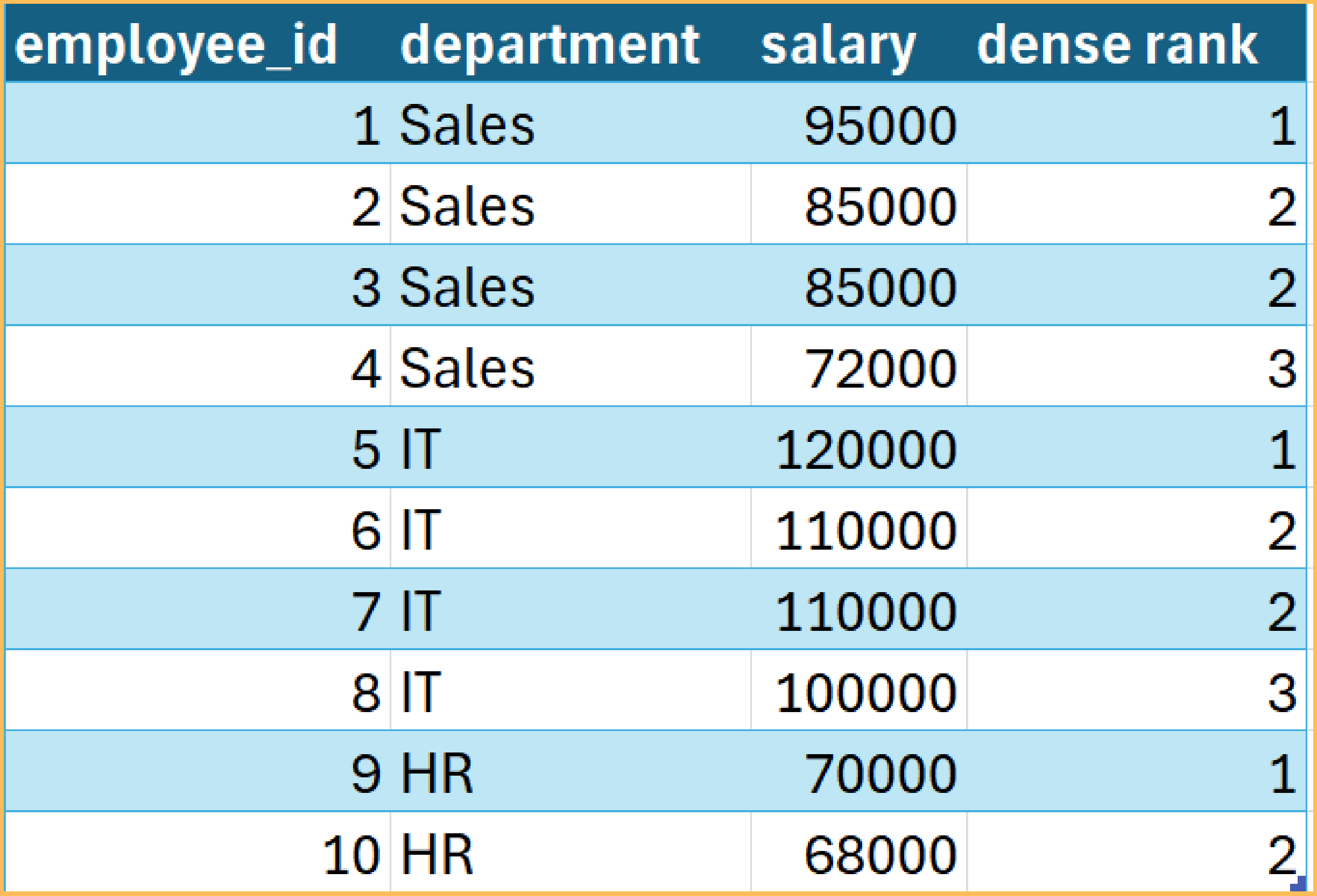


```
SELECT
    employee_id,
    department,
    salary,
    DENSE_RANK() OVER (PARTITION BY
department ORDER BY salary DESC) AS
dense_rank
FROM employees;
```

09

Window Function Problems:

✅ EXPECTED OUTPUT



employee_id	department	salary	dense rank
1	Sales	95000	1
2	Sales	85000	2
3	Sales	85000	2
4	Sales	72000	3
5	IT	120000	1
6	IT	110000	2
7	IT	110000	2
8	IT	100000	3
9	HR	70000	1
10	HR	68000	2

Explanation:

In Sales:

- The highest salary ($95,000) gets rank 1.
- Two employees have $85,000, so they share rank 2.
- The next salary ($72,000) gets rank 3 (no rank is skipped).

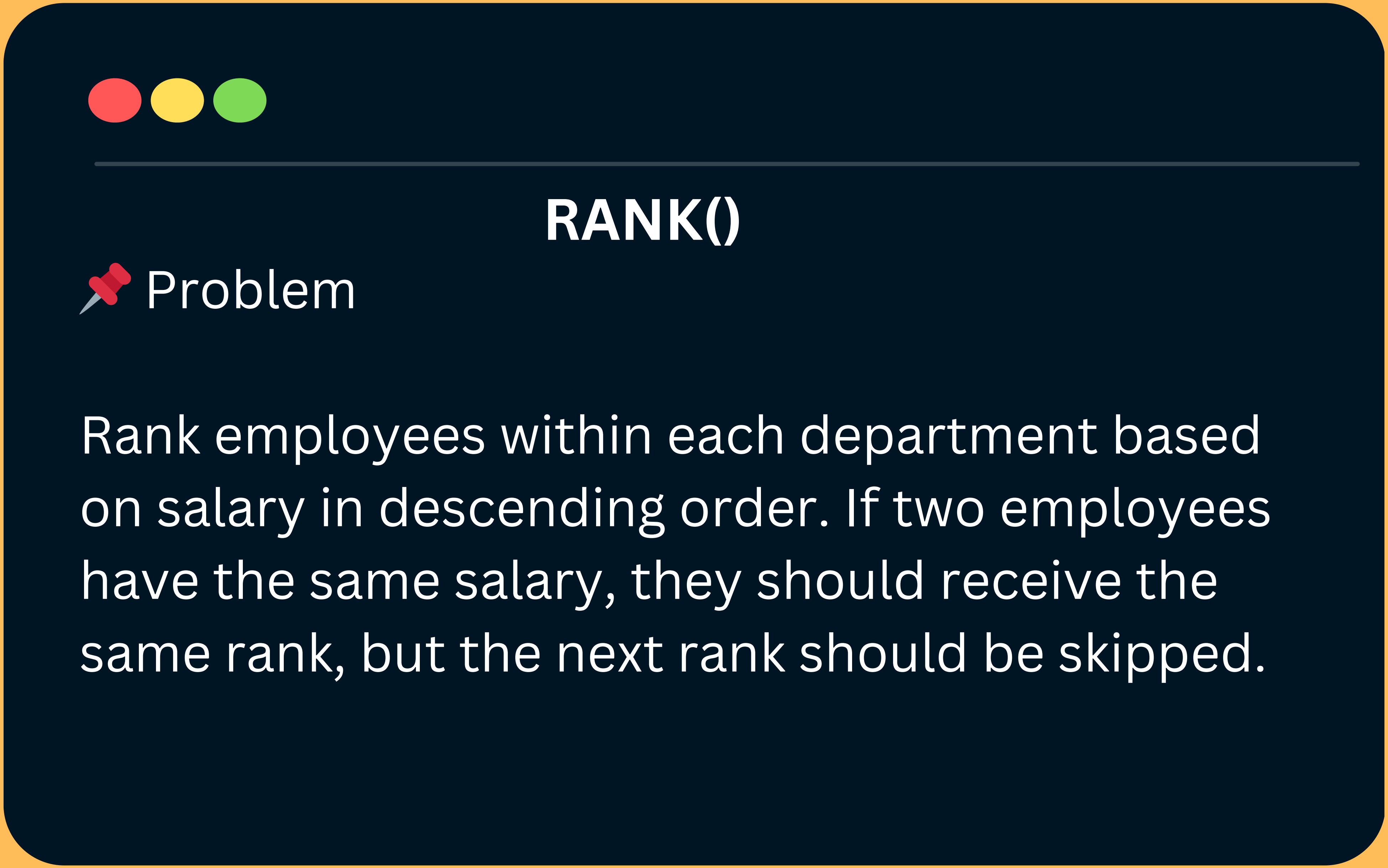
In IT:

- The highest salary ($120,000) gets rank 1.
- Two employees have $110,000, so they share rank 2.
- The next salary ($100,000) gets rank 3 (no rank is skipped).

In HR:

- The highest salary ($70,000) gets rank 1.
- The next salary ($68,000) gets rank 2.

Window Function Problems:



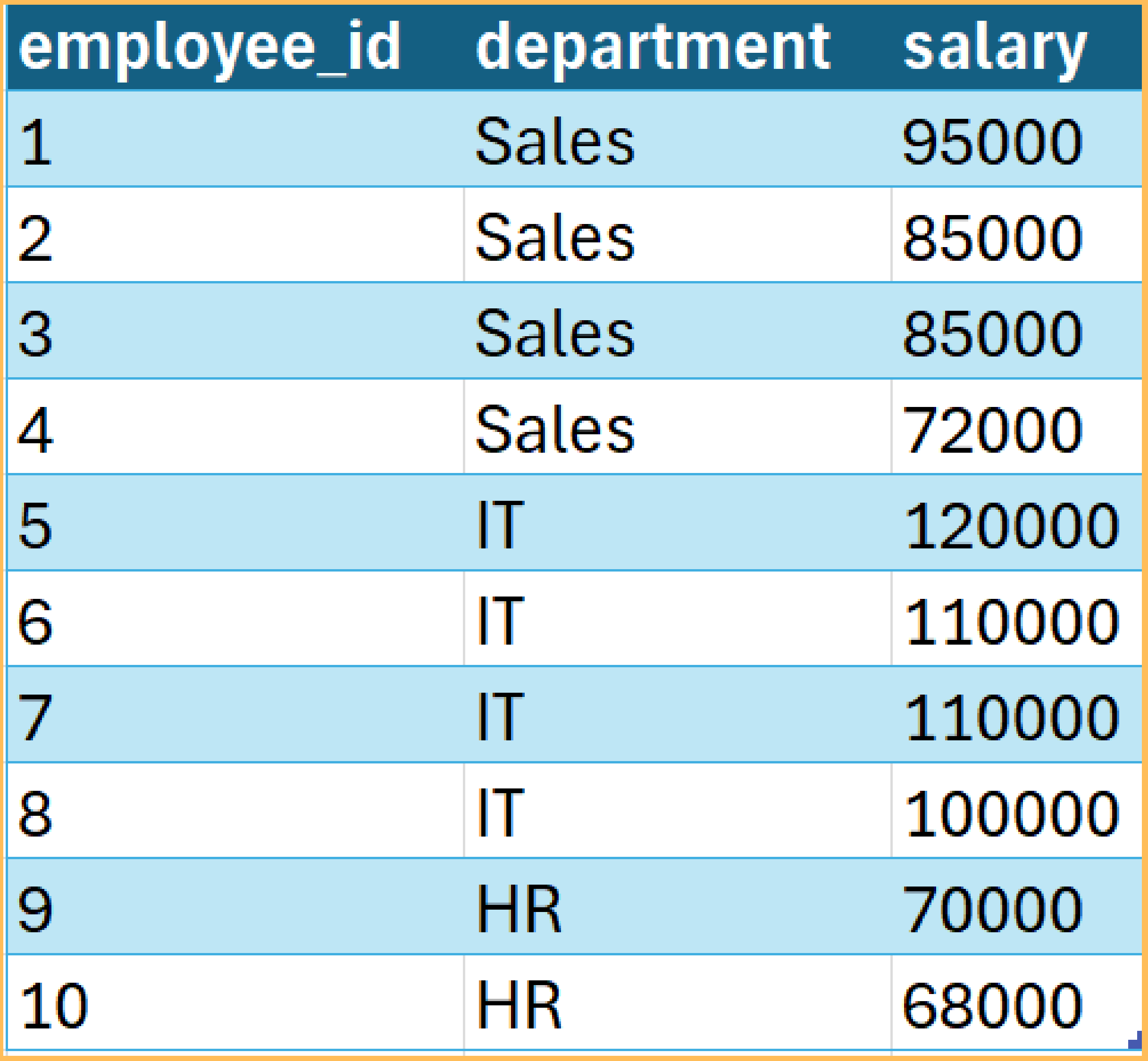
RANK()

📌 Problem

Rank employees within each department based on salary in descending order. If two employees have the same salary, they should receive the same rank, but the next rank should be skipped.

Window Function Problems:

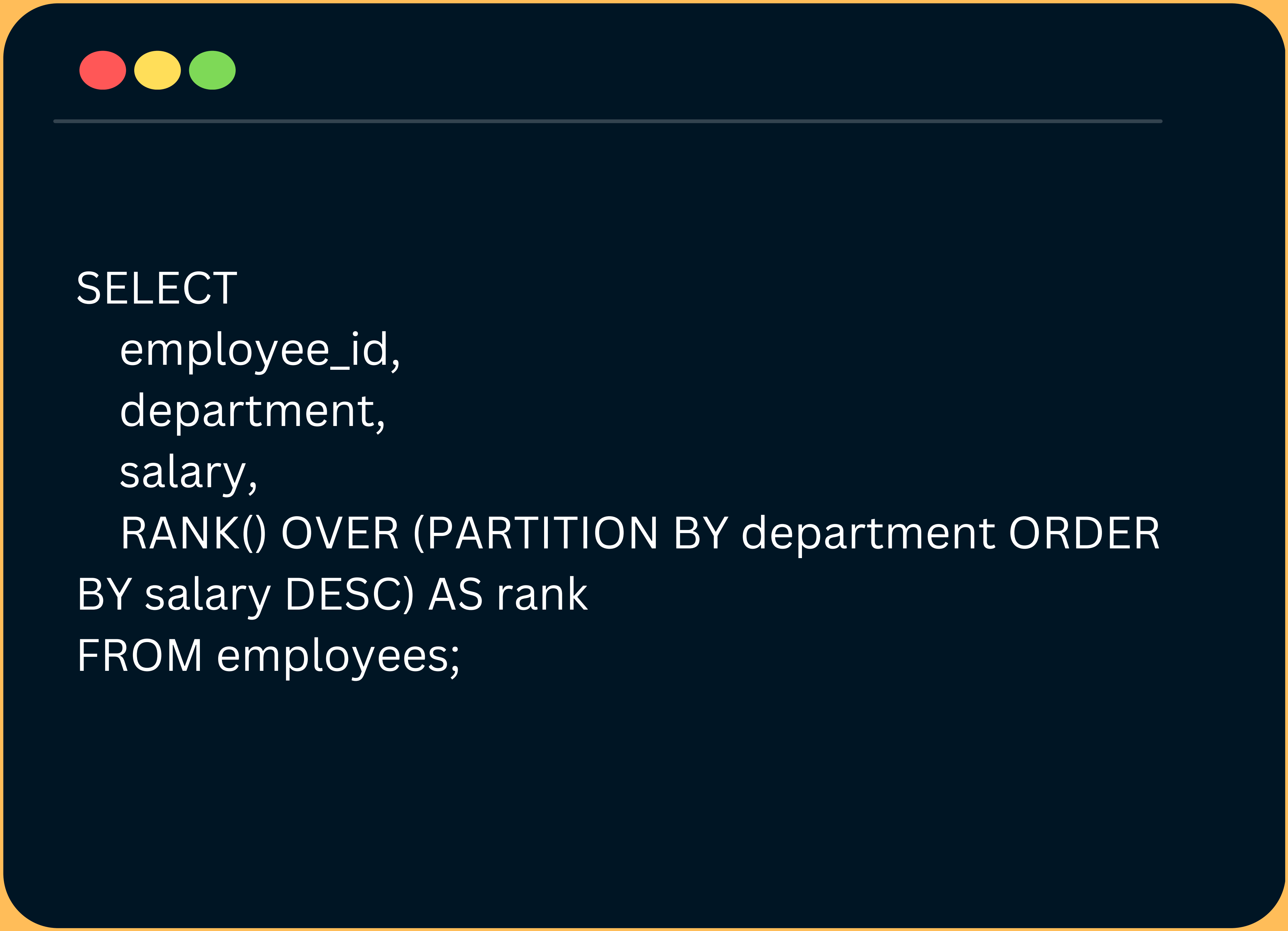
📊 DATA SET (EMPLOYEES TABLE)



employee_id	department	salary
1	Sales	95000
2	Sales	85000
3	Sales	85000
4	Sales	72000
5	IT	120000
6	IT	110000
7	IT	110000
8	IT	100000
9	HR	70000
10	HR	68000

https://www.linkedin.com/in/nivetha-anand/

10

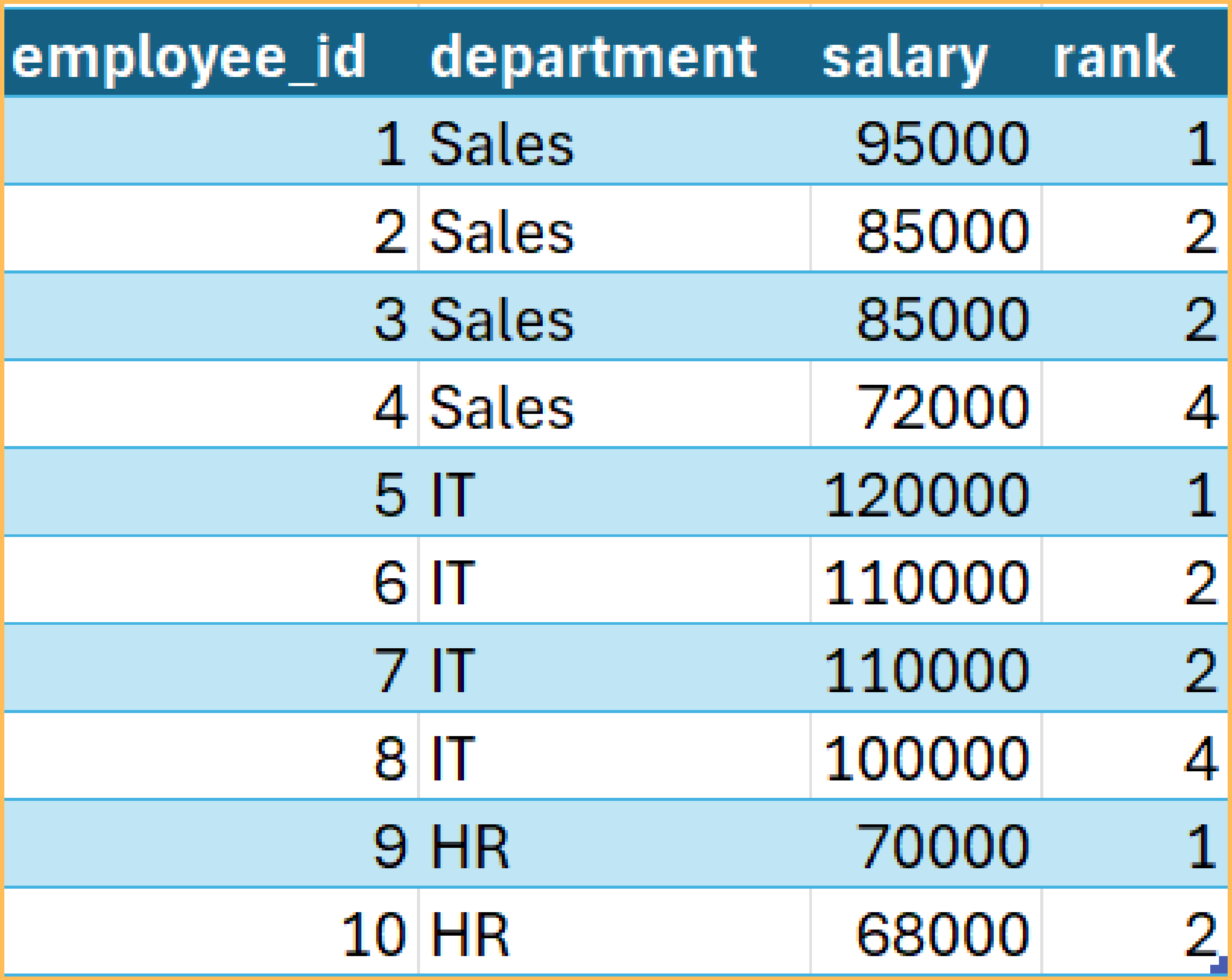


```
SELECT
    employee_id,
    department,
    salary,
    RANK() OVER (PARTITION BY department ORDER
BY salary DESC) AS rank
FROM employees;
```

11

Window Function Problems:

✅ EXPECTED OUTPUT



employee_id	department	salary	rank
1	Sales	95000	1
2	Sales	85000	2
3	Sales	85000	2
4	Sales	72000	4
5	IT	120000	1
6	IT	110000	2
7	IT	110000	2
8	IT	100000	4
9	HR	70000	1
10	HR	68000	2

https://www.linkedin.com/in/nivetha-anand/

Explanation:

In Sales:

- The highest salary ($95,000) gets rank 1.
- Two employees have $85,000, so they share rank 2.
- The next rank is 4 (rank 3 is skipped).

In IT:

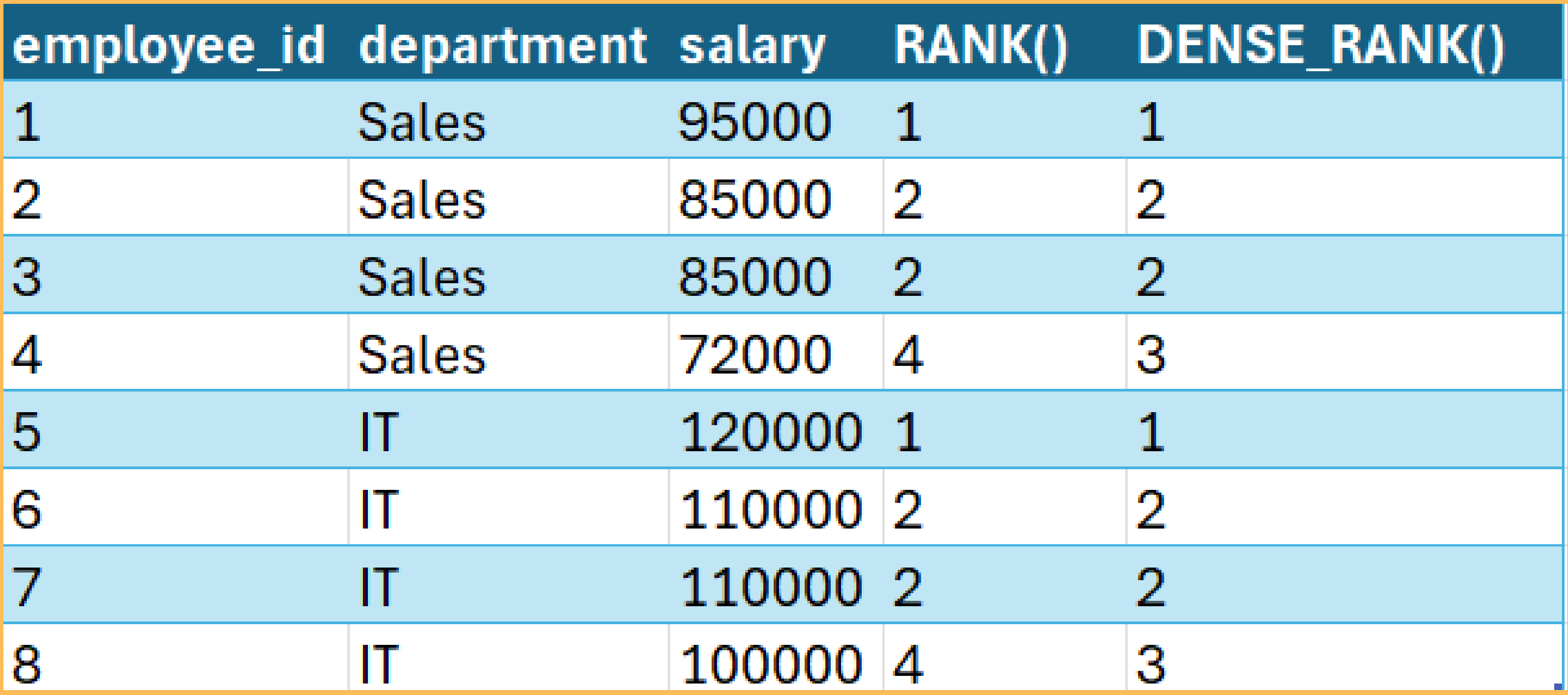
- The highest salary ($120,000) gets rank 1.
- Two employees have $110,000, so they share rank 2.
- The next salary ($100,000) gets rank 4 (rank 3 is skipped).

In HR:

- The highest salary ($70,000) gets rank 1.
- The next salary ($68,000) gets rank 2.

13

Rank() vs Dense_Rank() - Key Difference



employee_id	department	salary	RANK()	DENSE_RANK()
1	Sales	95000	1	1
2	Sales	85000	2	2
3	Sales	85000	2	2
4	Sales	72000	4	3
5	IT	120000	1	1
6	IT	110000	2	2
7	IT	110000	2	2
8	IT	100000	4	3

https://www.linkedin.com/in/nivetha-anand/

When to use which?

✅ Use RANK() if you need true ranking positions (e.g., competition scoring, where ties affect ranking).

✅ Use DENSE_RANK() if you want consecutive ranking without gaps (e.g., salary bands, grading systems).

Was this post helpful to you?

Please like, share, save, comment and repost!

👍 ➤ 🔖 💬 ♻

https://www.linkedin.com/in/nivetha-anand/ →