Ranking Functions in SQL: Exercises

Generated Exercises

May 21, 2025

Contents

Dataset for PostgreSQL

The following SQL code creates and populates the necessary tables for the exercises.

```
DROP TABLE IF EXISTS product\_sales;
DROP TABLE IF EXISTS employees;

CREATE TABLE employees (
    employee\_id INT PRIMARY KEY,
    first\_name VARCHAR(50),
    last\_name VARCHAR(50),
    department VARCHAR(50),
    salary DECIMAL(10, 2),
    hire\_date DATE
);

INSERT INTO employees (employee\_id, first\_name, last\_name, department, salary, hire\
    _date) VALUES
(1, 'Alice', 'Smith', 'HR', 60000.00, '2020-01-15'),
(2, 'Bob', 'Johnson', 'HR', 65000.00, '2019-07-01'),
(3, 'Charlie', 'Williams', 'IT', 80000.00, '2021-03-10'),
(4, 'David', 'Brown', 'IT', 90000.00, '2018-05-20'),
(5, 'Eve', 'Jones', 'IT', 80000.00, '2022-01-05'),
(6, 'Frank', 'Garcia', 'Finance', 75000.00, '2020-11-01'),
(7, 'Grace', 'Miller', 'Finance', 75000.00, '2021-06-15'),
(8, 'Henry', 'Davis', 'Marketing', 70000.00, '2019-02-28'),
(9, 'Ivy', 'Rodriguez', 'Marketing', 72000.00, '2023-01-10'),
(10, 'Jack', 'Wilson', 'Marketing', 70000.00, '2020-08-15'),
(11, 'Karen', 'Moore', 'HR', 60000.00, '2021-09-01'),
(12, 'Liam', 'Taylor', 'IT', 95000.00, '2023-03-01');

CREATE TABLE product\_sales (
    sale\_id INT PRIMARY KEY,
    product\_name VARCHAR(100),
    category VARCHAR(50),
    sale\_date DATE,
    sale\_amount DECIMAL(10, 2),
    quantity\_sold INT
);

INSERT INTO product\_sales (sale\_id, product\_name, category, sale\_date, sale\_amount,
     quantity\_sold) VALUES
(1, 'Laptop Pro', 'Electronics', '2023-01-10', 1200.00, 5),
(2, 'Smartphone X', 'Electronics', '2023-01-12', 800.00, 10),
(3, 'Office Chair', 'Furniture', '2023-01-15', 150.00, 20),
(4, 'Desk Lamp', 'Furniture', '2023-01-18', 40.00, 30),
(5, 'Laptop Pro', 'Electronics', '2023-02-05', 1200.00, 3),
(6, 'Gaming Mouse', 'Electronics', '2023-02-10', 75.00, 50),
(7, 'Smartphone X', 'Electronics', '2023-02-15', 780.00, 8),
(8, 'Bookshelf', 'Furniture', '2023-02-20', 200.00, 10),
(9, 'Laptop Pro', 'Electronics', '2023-03-01', 1150.00, 4),
(10, 'External HDD', 'Electronics', '2023-03-05', 100.00, 25),
(11, 'Office Chair', 'Furniture', '2023-03-10', 140.00, 15),
(12, 'Desk Lamp', 'Furniture', '2023-03-15', 35.00, 40),
(13, 'Smartphone Y', 'Electronics', '2023-03-20', 900.00, 12),
(14, 'Coffee Maker', 'Appliances', '2023-01-20', 60.00, 10),
(15, 'Toaster', 'Appliances', '2023-02-25', 30.00, 15),
(16, 'Blender', 'Appliances', '2023-03-01', 50.00, 8);
```

Listing 1: Dataset for Employees and Product Sales Tables

1 Practice Meanings, Values, Relations, and Advantages

1.1 Basic ROW_NUMBER()

Problem: Assign a unique sequential number to each employee based on their hire date (oldest first).

1.2 Basic RANK()

Problem: Rank employees based on their salary in descending order. Show how ties are handled (gaps in rank).

1.3 Basic DENSE_RANK()

Problem: Rank employees based on their salary in descending order using dense ranking. Show how ties are handled differently from `RANK()` (no gaps in rank).

1.4 Comparing ROW_NUMBER(), RANK(), DENSE_RANK() within partitions

Problem: For each department, assign a unique row number, rank (with gaps), and dense rank (no gaps) to employees based on their salary in descending order.

1.5 Advantage - Top N per group

Problem: Identify the top 2 highest-paid employees in each department.

2 Practice Disadvantages of Technical Concepts

2.1 iMisinterpretation of RANK() vs DENSE_RANK() for Nth distinct value

Problem: A manager wants to identify all employees who are in the top 2 distinct salary tiers within the company. Demonstrate this by comparing results from `RANK()` and `DENSE_RANK()`. Specifically, if the requirement is "employees in the 4th highest salary tier company-wide", show how using `RANK() = 4` might yield no results, while `DENSE_RANK() = 4` would correctly identify them.

2.2 iPotential for confusion with complex ORDER BY in window definition

Problem: Employees are ranked within their department by salary (descending). For employees with the same salary, their secondary sort key is hire date (ascending, earlier hire gets better rank). Show this ranking using `ROW_NUMBER()`. Then, demonstrate how if the secondary sort key (`hire_date`) was mistakenly ordered descending, it would change the row numbers for tied-salary employees, potentially leading to incorrect "top" employee identification if the secondary sort was critical for tie-breaking. Focus on employees in departments and salary groups where ties exist.

2.3 iConceptual disadvantage - Readability with many window functions

Problem: Display each employee's salary, their salary rank within their department, their overall salary rank in the company (dense), and a row number based on alphabetical order of their full name (`last_name`, then `first_name`). The query itself will demonstrate how multiple window functions can make the `SELECT` clause dense.

3 Practice Cases of Inefficient Alternatives

3.1 iiFind the employee(s) with the highest salary in each department

Problem: List the employee(s) with the highest salary in each department. If there are ties, list all. Provide both an inefficient solution (e.g., using a subquery with `MAX` for each department and joining back) and an efficient solution using ranking functions.

3.2 iiAssign sequential numbers to records

Problem: Provide a unique sequential number for each product sale, ordered by `sale_date` then by `sale_id`. Provide both an inefficient solution (e.g., using a correlated subquery to count preceding records) and an efficient solution using ranking functions.

3.3 iiRanking based on an aggregate

Problem: Rank product categories by their total sales amount in descending order. Provide both an inefficient solution (e.g., aggregating in a subquery/CTE, then using another correlated subquery or complex logic for ranking) and an efficient solution using ranking functions on the aggregated results.

4 Practice Hardcore Combined Problem

4.1 iv.Comprehensive Employee Analysis

Problem: For each department:

1. Identify the employee(s) with the highest salary in that department.
2. For these top-paid employees, determine their salary rank across the entire company (use `RANK()` for ties).
3. Show the average salary of their department.
4. Calculate the difference between their salary and their department's average salary.
5. Identify the `employee_id` and full name (`first_name || ' ' || last_name`) of the employee who was hired immediately *after* them within the same department, along with that next hire's date. If they are the last one hired in their department (among the top-paid, or overall if simpler for this part), this should be `NULL` for the next hire's details. (Clarification: "last one hired in their department" should be interpreted as the one with latest hire date in the entire department for identifying the "next hire").

Present a single SQL query that provides all this information.