Navigation Functions in SQL: Exercises

Generated Exercises

May 9, 2025

Contents

Dataset for PostgreSQL

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

```sql
-- Dataset for Navigation Functions Exercises
-- This dataset will be used for all exercises below.
-- It represents fictional employee performance metrics over time.

CREATE TABLE EmployeePerformance (
    perf_id SERIAL PRIMARY KEY,
    employee_id INT,
    employee_name VARCHAR(50),
    department VARCHAR(50),
    metric_date DATE,
    sales_amount DECIMAL(10, 2),
    tasks_completed INT
);

INSERT INTO EmployeePerformance (employee_id, employee_name, department, metric_date,
    sales_amount, tasks_completed) VALUES
-- Alice Smith (Sales)
(101, 'Alice Smith', 'Sales', '2023-01-05', 1500.00, 5),
(101, 'Alice Smith', 'Sales', '2023-01-12', 1700.00, 7),
(101, 'Alice Smith', 'Sales', '2023-01-19', 1600.00, 6),
(101, 'Alice Smith', 'Sales', '2023-02-03', 1800.00, 8),
(101, 'Alice Smith', 'Sales', '2023-02-10', 1750.00, 5),
(101, 'Alice Smith', 'Sales', '2023-03-05', 2000.00, 9),

-- Bob Johnson (Sales)
(102, 'Bob Johnson', 'Sales', '2023-01-08', 1200.00, 4),
(102, 'Bob Johnson', 'Sales', '2023-01-15', 1300.00, 6),
(102, 'Bob Johnson', 'Sales', '2023-02-05', 1100.00, 3),
(102, 'Bob Johnson', 'Sales', '2023-02-12', 1400.00, 7),
(102, 'Bob Johnson', 'Sales', '2023-03-10', 1500.00, 5),

-- Carol Davis (Engineering)
(201, 'Carol Davis', 'Engineering', '2023-01-10', 50.00, 10), -- Assuming minor sales
    for cross-functional tasks or internal transfers
(201, 'Carol Davis', 'Engineering', '2023-01-17', 70.00, 12),
(201, 'Carol Davis', 'Engineering', '2023-01-24', 60.00, 8),
(201, 'Carol Davis', 'Engineering', '2023-02-07', 80.00, 11),
(201, 'Carol Davis', 'Engineering', '2023-02-14', 75.00, 13),
(201, 'Carol Davis', 'Engineering', '2023-03-08', 90.00, 9),

-- David Wilson (Engineering)
(202, 'David Wilson', 'Engineering', '2023-01-05', 40.00, 7),
(202, 'David Wilson', 'Engineering', '2023-01-12', 60.00, 9),
(202, 'David Wilson', 'Engineering', '2023-02-03', 30.00, 6),
(202, 'David Wilson', 'Engineering', '2023-02-10', 65.00, 10),
(202, 'David Wilson', 'Engineering', '2023-03-05', 55.00, 8),

-- Eva Brown (Marketing)
(301, 'Eva Brown', 'Marketing', '2023-01-15', 500.00, 3),
(301, 'Eva Brown', 'Marketing', '2023-02-20', 600.00, 4),
(301, 'Eva Brown', 'Marketing', '2023-03-25', 550.00, 2);
```

Listing 1: Dataset for Employee Performance Exercises

1 Practice Meanings, Values, Relations, and Advantages

1.1 Exercise 1.1: Next Sales Amount Per Employee

Problem: For each performance record, display the employee's name, metric date, current sales amount, and the sales amount from their immediate next performance record. Order results by employee name and then by metric date.

1.2 Exercise 1.2: Previous Tasks Completed Per Employee within Department with Default

Problem: For each performance record, display the department, employee name, metric date, current tasks completed, and the tasks completed from their immediate previous performance record within the same department. If there is no previous record for that employee in that department, display 0 for previous tasks. Order results by department, employee name, and metric date.

1.3 Exercise 1.3: Sales Lookback and Lookahead for a Specific Employee

Problem: For 'Alice Smith', display her metric date, current sales amount, the sales amount from two performance records prior, and the sales amount from two performance records ahead. If such prior or ahead records do not exist, their values should be NULL. Order by metric date.

1.4 Exercise 1.4: Date of Next Performance Entry

Problem: For each performance record, display the employee's name, current metric date, and the date of their next performance entry. If there is no next entry, display NULL. Order by employee name and then current metric date.

2 Practice Disadvantages of Technical Concepts

2.1 Exercise 2.1: Handling NULLs from LAG at Partition Boundaries

Problem: For each performance record, show the employee name, metric date, current sales amount, and the sales amount from the previous record. Calculate the difference (current sales - previous sales). Observe the NULLs for the first record of each employee and how it affects the difference calculation. Order by employee name, then metric date.

2.2 Exercise 2.2: Impact of Incorrect ORDER BY in OVER() Clause

Problem: Display 'Alice Smith's performance records showing her metric date, tasks completed, and `next_tasks_correct_order` (tasks from the next chronological record). Then, show `next_tasks_incorrect_order` by mistakenly using `ORDER BY metric_date DESC` in the `LEAD` function's `OVER()` clause. Observe how `next_tasks_incorrect_order` now represents the tasks from the *previous* chronological record.

2.3 Exercise 2.3: Impact of Omitting PARTITION BY

Problem: For 'Bob Johnson', retrieve his `metric_date`, `sales_amount`, and the `previous_sales_amoun` (using `LAG` partitioned by `employee_id`). Also retrieve `previous_sales_amount_unpartitioned` (using `LAG` *without* `PARTITION BY employee_id`, but still ordered by `employee_id, metric_date` globally to ensure some row comes before Bob if not partitioned). Compare the `previous_sales_amount_unpartitioned` for Bob's first record ('2023-01-08') with `previous_sales_amount_partitioned`.

3 Practice Cases of Inefficient Alternatives

3.1 Exercise 3.1: Efficiently Finding Previous Sales Amount

Problem: For each employee performance record, find the `sales_amount` from their immediately preceding record. Using `LAG` is efficient. An inefficient alternative might involve a correlated subquery to find the `MAX(metric_date)` less than the current `metric_date` for the same employee, and then another subquery or join to retrieve the `sales_amount` for that found date, which is more complex and typically slower. Display employee name, metric date, current sales, and previous sales using the efficient `LAG` function.

3.2 Exercise 3.2: Efficiently Finding the Date of the Next Record

Problem: For each employee performance record, find the `metric_date` of their next performance record. Using `LEAD` is efficient. An inefficient alternative could be a correlated subquery like `(SELECT MIN(ep2.metric_date) FROM EmployeePerformance ep2 WHERE ep2.employee_id = ep1.employee_id AND ep2.metric_date > ep1.metric_date)`. Display employee name, current metric date, and the next metric date using the efficient `LEAD` function.

3.3 Exercise 3.3: Identifying Sales Increases Efficiently

Problem: Identify all performance records where an employee's `sales_amount` was greater than their `sales_amount` in the immediately preceding record for that same employee. Using `LAG` within a Common Table Expression (CTE) or subquery, followed by a `WHERE` clause, is efficient. Inefficient methods could involve complex self-joins and date logic to identify and compare with the correct previous record. Display the employee name, metric date, current sales, previous sales, and mark if it's an increase.

4 Hardcore Problem Combining Concepts

4.1 Exercise 4.1: Employee Sales Streak and Monthly Comparison Analysis

Problem: For each employee in the 'Sales' department:

1. Retrieve `employee_name`, `metric_date`, current `sales_amount`.
2. Show `previous_sales` (using `LAG`) and `next_sales` (using `LEAD`). Default previous sales to 0 for calculation.
3. Determine if the current `sales_amount` is greater than the `previous_sales` (`is_increase` - boolean).
4. Assign a `streak_group_id`. A new streak of increases starts if `is_increase` is true and the previous record was not an increase (or it's the first record and it's an increase over 0). This ID should increment for each new streak for an employee. (Hint: sum a marker that is 1 when a streak starts).
5. Calculate the `running_sales_in_streak`: the cumulative `sales_amount` within the current `streak_group_id` for that employee.
6. For each record, show the employee's average `sales_amount` for the calendar month of that `metric_date` (`avg_monthly_sales_for_employee`).
7. Assign a `sales_rank_overall` to each employee based on their highest single `sales_amount` record using `DENSE_RANK()`. This rank should appear on all records for that employee.

Order results by `employee_name`, then `metric_date`.

4.2 Exercise 4.2: Departmental Task Performance Analysis

Problem: For each department:

1. Calculate `total_tasks_monthly` per employee per month (use `DATE_TRUNC` for month).
2. For each employee's `total_tasks_monthly`, show `prev_month_tasks` and `next_month_tasks` for that employee. Default to 0 if no data for previous/next month.
3. Calculate `mom_task_change_pct` (month-over-month percentage change in tasks). Handle NULLs or zero previous month tasks appropriately (e.g., output NULL or 100% if previous was 0 and current is ¿0).
4. Assign `feb_task_rank_in_dept`: a row number to each employee *within their department* based on their total tasks completed in February 2023 (month starting '2023-02-01'), ordered highest to lowest. This rank should only appear for February data.
5. Identify employees who had at least one month where their `total_tasks_monthly` were 20% higher than their department's average tasks completed for that same month (`dept_avg_tasks_monthly`). List `employee_name`, `month_start_date`, their `total_tasks_monthly`, and `dept_avg_tasks_monthly` for these instances.

Order the final result for point 5 by department, employee name, and month.

4.3 Exercise 4.3: Cross-Metric Anomaly Detection and Trend Analysis

Problem:

1. For each performance record, calculate `sales_to_tasks_ratio` (`sales_amount / tasks_completed`). If `tasks_completed` is 0, the ratio should be NULL.
2. Show `prev_ratio` (from the previous record for the same employee) using `LAG`.
3. Show `next_ratio_plus_1` (from the next record) and `next_ratio_plus_2` (from two records ahead) for the same employee, defaulting to NULL.
4. Identify `is_significant_ratio_dip`: true if current `sales_to_tasks_ratio` is less than 50% of `prev_ratio`, and `prev_ratio` was not NULL and not zero.
5. Calculate `rolling_avg_3rec_sales`: employee's rolling average of `sales_amount` over their last 3 performance records (current and 2 preceding).
6. Assign `jan_sales_rank_in_dept`. This ranks employees within their department by their total `sales_amount` in January 2023 (records where `metric_date` is between '2023-01-01' and '2023-01-31'). Only include employees with at least 2 performance records in January for this ranking. The rank should appear on all January records for qualifying employees.
7. List `employee_name`, `department`, `metric_date`, `sales_to_tasks_ratio`, `prev_ratio`, `next_ratio_plus_1`, `next_ratio_plus_2`, `is_significant_ratio_dip`, `rolling_avg_3rec_sales`, and applicable `jan_sales_rank_in_dept`.

Order results by employee name and metric date.