Navigation Functions: Solutions

Sequential SQL

May 21, 2025

Contents

1 Solutions: Meanings, Values, Relations, and Advantages **3**
- 1.1 Solution 1.1: Next Sales Amount Per Employee . . . 3
- 1.2 Solution 1.2: Previous Tasks Completed Per Employee within Department with Default . . . 3
- 1.3 Solution 1.3: Sales Lookback and Lookahead for a Specific Employee . . 3
- 1.4 Solution 1.4: Date of Next Performance Entry . . . 4

2 Solutions: Disadvantages of Technical Concepts **5**
- 2.1 Solution 2.1: Handling NULLs from LAG at Partition Boundaries . . . 5
- 2.2 Solution 2.2: Impact of Incorrect ORDER BY in OVER() Clause . . . 5
- 2.3 Solution 2.3: Impact of Omitting PARTITION BY . . . 5

3 Solutions: Cases of Inefficient Alternatives **7**
- 3.1 Solution 3.1: Efficiently Finding Previous Sales Amount . . . 7
- 3.2 Solution 3.2: Efficiently Finding the Date of the Next Record . . . 7
- 3.3 Solution 3.3: Identifying Sales Increases Efficiently . . . 7

4 Solutions: Hardcore Problem Combining Concepts **9**
- 4.1 Solution 4.1: Employee Sales Streak and Monthly Comparison Analysis . 9
- 4.2 Solution 4.2: Departmental Task Performance Analysis . . . 10
- 4.3 Solution 4.3: Cross-Metric Anomaly Detection and Trend Analysis . . . 12

Dataset for PostgreSQL

The following SQL code creates and populates the necessary tables for the exercises. (Same as in exercises document).

```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 (identical to exercises document)

1 Solutions: Meanings, Values, Relations, and Advantages

1.1 Solution 1.1: Next Sales Amount Per Employee

```sql
SELECT
    employee_name,
    metric_date,
    sales_amount AS current_sales,
    LEAD(sales_amount, 1) OVER (PARTITION BY employee_id ORDER BY
   metric_date) AS next_sales_amount
FROM
    EmployeePerformance
ORDER BY
    employee_name, metric_date;
```

Listing 2: Solution for Exercise 1.1

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

```sql
SELECT
    department,
    employee_name,
    metric_date,
    tasks_completed AS current_tasks,
    LAG(tasks_completed, 1, 0) OVER (PARTITION BY department,
   employee_id ORDER BY metric_date) AS previous_tasks_completed
FROM
    EmployeePerformance
ORDER BY
    department, employee_name, metric_date;
```

Listing 3: Solution for Exercise 1.2

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

```sql
SELECT
    metric_date,
    sales_amount AS current_sales,
    LAG(sales_amount, 2) OVER (ORDER BY metric_date) AS
   sales_2_records_prior,
    LEAD(sales_amount, 2) OVER (ORDER BY metric_date) AS
   sales_2_records_ahead
FROM
    EmployeePerformance
WHERE
    employee_name = 'Alice Smith'
ORDER BY
    metric_date;
```

Listing 4: Solution for Exercise 1.3

1.4 Solution 1.4: Date of Next Performance Entry

```sql
SELECT
    employee_name,
    metric_date AS current_metric_date,
    LEAD(metric_date, 1) OVER (PARTITION BY employee_id ORDER BY
   metric_date) AS next_metric_date
FROM
    EmployeePerformance
ORDER BY
    employee_name, current_metric_date;
```

Listing 5: Solution for Exercise 1.4

2 Solutions: Disadvantages of Technical Concepts

2.1 Solution 2.1: Handling NULLs from LAG at Partition Boundaries

```sql
SELECT
    employee_name,
    metric_date,
    sales_amount AS current_sales,
    LAG(sales_amount, 1) OVER (PARTITION BY employee_id ORDER BY metric_date) AS previous_sales,
    sales_amount - LAG(sales_amount, 1) OVER (PARTITION BY employee_id ORDER BY metric_date) AS sales_difference
FROM
    EmployeePerformance
ORDER BY
    employee_name, metric_date;
```

Listing 6: Solution for Exercise 2.1

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

```sql
SELECT
    employee_name,
    metric_date,
    tasks_completed,
    LEAD(tasks_completed, 1) OVER (PARTITION BY employee_id ORDER BY metric_date ASC) AS next_tasks_correct_order,
    LEAD(tasks_completed, 1) OVER (PARTITION BY employee_id ORDER BY metric_date DESC) AS next_tasks_incorrect_order
FROM
    EmployeePerformance
WHERE
    employee_name = 'Alice Smith'
ORDER BY
    metric_date;
```

Listing 7: Solution for Exercise 2.2

2.3 Solution 2.3: Impact of Omitting PARTITION BY

```sql
-- To ensure proper comparison, we'll query all data for the unpartitioned case's ordering context
-- then filter for Bob to see the effect specifically on him.
WITH UnpartitionedLag AS (
    SELECT
        perf_id, -- Added perf_id to join back
        employee_id,
        employee_name,
        metric_date,
        sales_amount,
        LAG(sales_amount, 1) OVER (ORDER BY employee_id, metric_date) AS previous_sales_amount_unpartitioned_global
```

```
    FROM
        EmployeePerformance
)
SELECT
    ep.employee_name,
    ep.metric_date,
    ep.sales_amount,
    LAG(ep.sales_amount, 1) OVER (PARTITION BY ep.employee_id ORDER BY ep.metric_date) AS previous_sales_amount_partitioned,
    ul.previous_sales_amount_unpartitioned_global AS previous_sales_amount_unpartitioned_contextual
FROM
    EmployeePerformance ep
JOIN
    UnpartitionedLag ul ON ep.perf_id = ul.perf_id
WHERE
    ep.employee_name = 'Bob Johnson'
ORDER BY
    ep.metric_date;
-- Note: The unpartitioned LAG for Bob's first record will likely pick up Alice Smith's last record if ordered by employee_id then date,
-- or another employee's record, demonstrating data leakage if partitioning is missed.
```

Listing 8: Solution for Exercise 2.3

3 Solutions: Cases of Inefficient Alternatives

3.1 Solution 3.1: Efficiently Finding Previous Sales Amount

```sql
SELECT
    employee_name,
    metric_date,
    sales_amount AS current_sales,
    LAG(sales_amount, 1, 0.00) OVER (PARTITION BY employee_id ORDER BY
   metric_date) AS previous_sales_efficient
FROM
    EmployeePerformance
ORDER BY
    employee_name, metric_date;
```

Listing 9: Solution for Exercise 3.1

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

```sql
SELECT
    employee_name,
    metric_date AS current_metric_date,
    LEAD(metric_date, 1) OVER (PARTITION BY employee_id ORDER BY
   metric_date) AS next_metric_date_efficient
FROM
    EmployeePerformance
ORDER BY
    employee_name, metric_date;
```

Listing 10: Solution for Exercise 3.2

3.3 Solution 3.3: Identifying Sales Increases Efficiently

```sql
WITH SalesWithPrevious AS (
    SELECT
        employee_name,
        metric_date,
        sales_amount AS current_sales,
        LAG(sales_amount, 1) OVER (PARTITION BY employee_id ORDER BY
   metric_date) AS previous_sales
    FROM
        EmployeePerformance
)
SELECT
    employee_name,
    metric_date,
    current_sales,
    previous_sales,
    (current_sales > previous_sales) AS is_increase
FROM
    SalesWithPrevious
WHERE
    previous_sales IS NOT NULL AND current_sales > previous_sales -- Or
    just (current_sales > previous_sales) if NULLs handled by boolean
   logic as desired
```

```
ORDER BY
    employee_name, metric_date;
```

Listing 11: Solution for Exercise 3.3

4 Solutions: Hardcore Problem Combining Concepts

4.1 Solution 4.1: Employee Sales Streak and Monthly Comparison Analysis

```
WITH EmployeeSalesData AS (
    SELECT
        perf_id,
        employee_id,
        employee_name,
        department,
        metric_date,
        sales_amount,
        LAG(sales_amount, 1, 0.00) OVER (PARTITION BY employee_id ORDER BY metric_date) AS previous_sales,
        LEAD(sales_amount, 1) OVER (PARTITION BY employee_id ORDER BY metric_date) AS next_sales
    FROM
        EmployeePerformance
    WHERE department = 'Sales'
),
StreakCalculation AS (
    SELECT
        *,
        (sales_amount > previous_sales) AS is_increase,
        LAG(sales_amount > previous_sales, 1, FALSE) OVER (PARTITION BY employee_id ORDER BY metric_date) AS prev_is_increase
    FROM
        EmployeeSalesData
),
StreakGroup AS (
    SELECT
        *,
        SUM(CASE WHEN is_increase AND NOT prev_is_increase THEN 1 ELSE 0 END) OVER (PARTITION BY employee_id ORDER BY metric_date) AS streak_group_id_core
    FROM StreakCalculation
),
FinalCalcs AS (
    SELECT
        sg.employee_id,
        sg.employee_name,
        sg.metric_date,
        sg.sales_amount,
        sg.previous_sales,
        sg.next_sales,
        sg.is_increase,
        sg.streak_group_id_core + CASE WHEN sg.is_increase THEN 1 ELSE 0 END AS streak_group_id,
        SUM(CASE WHEN sg.is_increase THEN sg.sales_amount ELSE 0 END) OVER (PARTITION BY sg.employee_id, sg.streak_group_id_core + CASE WHEN sg.is_increase THEN 1 ELSE 0 END ORDER BY sg.metric_date) AS running_sales_in_streak,
        AVG(sg.sales_amount) OVER (PARTITION BY sg.employee_id, DATE_TRUNC('month', sg.metric_date)) AS avg_monthly_sales_for_employee,
        DENSE_RANK() OVER (ORDER BY MAX(sg.sales_amount) OVER (
```

```
    PARTITION BY sg.employee_id) DESC) AS
    employee_max_sale_based_rank_value
    FROM
        StreakGroup sg
),
EmployeeMaxSale AS (
    SELECT
        employee_id,
        MAX(sales_amount) as max_employee_sale
    FROM EmployeePerformance
    WHERE department = 'Sales'
    GROUP BY employee_id
),
EmployeeRankedByMaxSale AS (
    SELECT
        employee_id,
        DENSE_RANK() OVER (ORDER BY max_employee_sale DESC) as
    sales_rank_overall
    FROM EmployeeMaxSale
)
SELECT
    fc.employee_name,
    fc.metric_date,
    fc.sales_amount,
    fc.previous_sales,
    fc.next_sales,
    fc.is_increase,
    fc.streak_group_id,
    CASE WHEN fc.is_increase THEN fc.running_sales_in_streak ELSE 0 END
    as running_sales_in_streak,
    ROUND(fc.avg_monthly_sales_for_employee, 2) as
    avg_monthly_sales_for_employee,
    er.sales_rank_overall
FROM
    FinalCalcs fc
JOIN
    EmployeeRankedByMaxSale er ON fc.employee_id = er.employee_id
ORDER BY
    fc.employee_name, fc.metric_date;
```

Listing 12: Solution for Exercise 4.1

4.2 Solution 4.2: Departmental Task Performance Analysis

```
WITH MonthlyTasks AS (
    SELECT
        employee_id,
        employee_name,
        department,
        DATE_TRUNC('month', metric_date) AS month_start_date,
        SUM(tasks_completed) AS total_tasks_monthly
    FROM
        EmployeePerformance
    GROUP BY
        employee_id, employee_name, department, DATE_TRUNC('month',
    metric_date)
),
```

```sql
MonthlyTasksWithLagLead AS (
    SELECT
        *,
        LAG(total_tasks_monthly, 1, 0) OVER (PARTITION BY employee_id ORDER BY month_start_date) AS prev_month_tasks,
        LEAD(total_tasks_monthly, 1, 0) OVER (PARTITION BY employee_id ORDER BY month_start_date) AS next_month_tasks
    FROM
        MonthlyTasks
),
MonthlyAnalysis AS (
    SELECT
        *,
        CASE
            WHEN prev_month_tasks = 0 AND total_tasks_monthly > 0 THEN 100.0 -- Or NULL, depends on definition
            WHEN prev_month_tasks = 0 AND total_tasks_monthly = 0 THEN 0.0
            WHEN prev_month_tasks IS NULL THEN NULL -- Or prev_month_tasks = 0 handled above
            ELSE ROUND(((total_tasks_monthly - prev_month_tasks) * 100.0 / prev_month_tasks), 2)
        END AS mom_task_change_pct,
        AVG(total_tasks_monthly) OVER (PARTITION BY department, month_start_date) AS dept_avg_tasks_monthly,
        CASE
            WHEN month_start_date = '2023-02-01' THEN
                ROW_NUMBER() OVER (PARTITION BY department, month_start_date ORDER BY total_tasks_monthly DESC)
            ELSE NULL
        END AS feb_task_rank_in_dept
    FROM
        MonthlyTasksWithLagLead
)
-- For point 5:
SELECT
    department,
    employee_name,
    month_start_date,
    total_tasks_monthly,
    ROUND(dept_avg_tasks_monthly, 2) AS dept_avg_tasks_monthly
FROM
    MonthlyAnalysis
WHERE
    total_tasks_monthly > (dept_avg_tasks_monthly * 1.20)
ORDER BY
    department, employee_name, month_start_date;

-- To see all calculated fields from MonthlyAnalysis for verification (optional, not part of the direct answer to point 5):
-- SELECT * FROM MonthlyAnalysis ORDER BY department, employee_name, month_start_date;
```

Listing 13: Solution for Exercise 4.2

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

```sql
WITH RatioData AS (
    SELECT
        perf_id,
        employee_id,
        employee_name,
        department,
        metric_date,
        sales_amount,
        tasks_completed,
        CASE
            WHEN tasks_completed IS NULL OR tasks_completed = 0 THEN NULL
            ELSE sales_amount / tasks_completed
        END AS sales_to_tasks_ratio
    FROM
        EmployeePerformance
),
NavigationAndRollingAvg AS (
    SELECT
        *,
        LAG(sales_to_tasks_ratio, 1) OVER (PARTITION BY employee_id ORDER BY metric_date) AS prev_ratio,
        LEAD(sales_to_tasks_ratio, 1) OVER (PARTITION BY employee_id ORDER BY metric_date) AS next_ratio_plus_1,
        LEAD(sales_to_tasks_ratio, 2) OVER (PARTITION BY employee_id ORDER BY metric_date) AS next_ratio_plus_2,
        AVG(sales_amount) OVER (PARTITION BY employee_id ORDER BY metric_date ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS rolling_avg_3rec_sales
    FROM
        RatioData
),
DipDetection AS (
    SELECT
        *,
        CASE
            WHEN prev_ratio IS NOT NULL AND prev_ratio <> 0 AND sales_to_tasks_ratio < (0.5 * prev_ratio) THEN TRUE
            ELSE FALSE
        END AS is_significant_ratio_dip
    FROM
        NavigationAndRollingAvg
),
JanSalesTotals AS (
    SELECT
        employee_id,
        department,
        SUM(sales_amount) AS total_jan_sales,
        COUNT(*) AS jan_record_count
    FROM
        EmployeePerformance
    WHERE
        metric_date BETWEEN '2023-01-01' AND '2023-01-31'
    GROUP BY
```

```
        employee_id, department
    HAVING
        COUNT(*) >= 2
),
JanSalesRanking AS (
    SELECT
        employee_id,
        department,
        DENSE_RANK() OVER (PARTITION BY department ORDER BY total_jan_sales DESC) AS jan_rank
    FROM
        JanSalesTotals
)
SELECT
    dd.employee_name,
    dd.department,
    dd.metric_date,
    ROUND(dd.sales_to_tasks_ratio, 2) AS sales_to_tasks_ratio,
    ROUND(dd.prev_ratio, 2) AS prev_ratio,
    ROUND(dd.next_ratio_plus_1, 2) AS next_ratio_plus_1,
    ROUND(dd.next_ratio_plus_2, 2) AS next_ratio_plus_2,
    dd.is_significant_ratio_dip,
    ROUND(dd.rolling_avg_3rec_sales, 2) AS rolling_avg_3rec_sales,
    CASE
        WHEN dd.metric_date BETWEEN '2023-01-01' AND '2023-01-31' THEN jsr.jan_rank
        ELSE NULL
    END AS jan_sales_rank_in_dept
FROM
    DipDetection dd
LEFT JOIN
    JanSalesRanking jsr ON dd.employee_id = jsr.employee_id AND dd.department = jsr.department
ORDER BY
    dd.employee_name, dd.metric_date;
```

Listing 14: Solution for Exercise 4.3