Conditionals: Advanced ORDER BY

Complementary SQL: Exercises

May 11, 2025

Contents

Dataset for PostgreSQL

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

```sql
-- Drop tables if they exist to ensure a clean setup
DROP TABLE IF EXISTS employee\_projects CASCADE;
DROP TABLE IF EXISTS employees CASCADE;
DROP TABLE IF EXISTS departments CASCADE;

-- Create departments table
CREATE TABLE departments (
    department\_name VARCHAR(50) PRIMARY KEY,
    location VARCHAR(50),
    budget_allocation NUMERIC(12,2)
);

-- Populate departments table
INSERT INTO departments (department\_name, location, budget_allocation) VALUES
('Engineering', 'New York', 500000.00),
('Marketing', 'San Francisco', 300000.00),
('Sales', 'Chicago', 400000.00),
('Human Resources', 'New York', 250000.00),
('Product', 'San Francisco', 350000.00),
('Support', 'Remote', 150000.00);

-- Create employees table
CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    first\_name VARCHAR(50),
    last\_name VARCHAR(50),
    department VARCHAR(50) REFERENCES departments(department\_name),
    salary NUMERIC(10, 2),
    hire\_date DATE,
    bonus\_percentage NUMERIC(3, 2), -- Nullable
    manager_id INTEGER REFERENCES employees(id) -- Nullable
);

-- Populate employees table
INSERT INTO employees (first\_name, last\_name, department, salary, hire\_date, bonus\_percentage, manager_id) VALUES
('Alice', 'Smith', 'Engineering', 90000.00, '2020-03-15', 0.10, NULL),
('Bob', 'Johnson', 'Engineering', 95000.00, '2019-07-01', 0.12, 1),
('Charlie', 'Williams', 'Marketing', 70000.00, '2021-01-10', 0.08, NULL),
('David', 'Brown', 'Sales', 80000.00, '2020-11-05', NULL, NULL), -- Null bonus
('Eve', 'Jones', 'Engineering', 90000.00, '2021-05-20', 0.10, 1),
('Frank', 'Garcia', 'Marketing', 72000.00, '2022-02-01', NULL, 3), -- Null bonus
('Grace', 'Miller', 'Sales', 82000.00, '2019-05-20', 0.09, 4),
('Heidi', 'Davis', 'Human Resources', 65000.00, '2023-01-15', 0.05, NULL),
('Ivan', 'Rodriguez', 'Product', 110000.00, '2020-08-24', 0.15, NULL),
('Judy', 'Martinez', 'Product', 105000.00, '2021-06-10', NULL, 9), -- Null bonus
('Kevin', 'Hernandez', 'Engineering', 88000.00, '2023-03-01', 0.07, 2),
('Linda', 'Lopez', 'Marketing', 68000.00, '2023-04-10', 0.06, 3),
('Mike', 'Gonzalez', 'Sales', 78000.00, '2022-07-18', 0.11, 4),
('Nancy', 'Wilson', 'Human Resources', 67000.00, '2022-09-01', NULL, 8), -- Null bonus
('Olivia', 'Anderson', 'Engineering', 90000.00, '2020-03-15', NULL, 1), -- Null bonus, same salary/hire\_date as Alice
('Peter', 'Lee', 'Product', 100000.00, '2021-08-15', 0.12, 9),
('Zoe', 'King', 'Engineering', 92000.00, '2022-05-01', 0.11, 1),
('Yasmin', 'Scott', 'Marketing', 75000.00, '2021-11-20', NULL, 3),
('Eva', 'Taylor', 'Engineering', 90000.00, '2021-05-20', NULL, 1); -- Same salary/hire\_date as Eve, but NULL bonus

-- Create employee\_projects table
CREATE TABLE employee\_projects (
    employee_id INTEGER REFERENCES employees(id),
    project\_name VARCHAR(100),
    project\_role VARCHAR(50),
    hours\_assigned INTEGER,
    PRIMARY KEY (employee_id, project\_name)
);

-- Populate employee\_projects table
```

```sql
INSERT INTO employee\_projects (employee_id, project\_name, project\_role, hours\
    _assigned) VALUES
(1, 'Alpha Platform', 'Developer', 120),
(1, 'Beta Feature', 'Lead Developer', 80),
(2, 'Alpha Platform', 'Senior Developer', 150),
(3, 'Campaign X', 'Coordinator', 100),
(5, 'Gamma Initiative', 'Developer', 90),
(5, 'Alpha Platform', 'Tester', 40),
(6, 'Campaign Y', 'Analyst', 110),
(7, 'Client Outreach', 'Manager', 60),
(9, 'Omega Product', 'Product Owner', 180),
(10, 'Omega Product', 'UX Designer', 70),
(11, 'Gamma Initiative', 'Junior Developer', 100),
(13, 'Client Retention', 'Specialist', 50);
```

Listing 1: Dataset for Advanced ORDER BY Exercises

1 Practice Meanings, Values, Relations, and Advantages

1.1 Exercise 1.1: Ordering by Multiple Columns

List all employees, ordered primarily by their ‘department‘ alphabetically (A-Z), and secondarily by their ‘salary‘ in descending order (highest salary first within each department).

1.2 Exercise 1.2: Using `NULLS FIRST`

Display all employees, ordering them by their ‘bonus_percentage‘. Employees who do not have a ‘bonus_percentage‘ specified (which are stored as ‘NULL‘ in the ‘bonus_percentage‘ column) should appear at the top of the list. For employees with non-NULL bonus percentages, they should be sorted in ascending order of their bonus.

1.3 Exercise 1.3: Using `NULLS LAST` and Multiple Columns

List all employees from the ’Engineering’ department, ordered first by their ‘hire_date‘ in ascending order (earliest first). If multiple employees share the same ‘hire_date‘, those with a ‘NULL‘ ‘bonus_percentage‘ should be listed after those with a non-NULL ‘bonus_percentage‘. Among those with non-NULL bonus percentages on the same ‘hire_date‘, sort by ‘bonus_percentage‘ in descending order (highest bonus first).

2 Practice Disadvantages

2.1 Exercise 2.1: Disadvantage of Overly Complex Sorting (Readability/Maintainability)

An analyst needs to sort employees using multiple criteria. They constructed the following ‘ORDER BY‘ clause:

```
ORDER BY department ASC, (CASE WHEN hire_date < ’2021-01-01’ THEN 0 ELSE 1
END) ASC, salary DESC NULLS LAST, (bonus_percentage IS NULL) ASC, last_name
ASC;
```

While this clause might be functionally correct for a specific complex requirement, what is a general disadvantage of such highly intricate ‘ORDER BY‘ clauses in terms of query development and teamwork, especially when simpler, more direct ”Advanced ORDER BY” features might cover parts of the logic more clearly?

2.2 Exercise 2.2: Disadvantage of Potentially Misleading Prioritization with `NULLS FIRST/LAST`

Imagine a scenario where a report is generated to identify employees eligible for a special program, and a key sorting criterion is ‘bonus_percentage‘. If ‘ORDER BY bonus_percentage ASC NULLS FIRST‘ is used, and a significant number of employees in the ‘employees‘ table have a ‘NULL‘ ‘bonus_percentage‘ (perhaps because it’s not applicable or not yet determined), what is a potential disadvantage or misinterpretation that could arise from this sorting strategy?

3 Practice Cases of Lost Advantages

3.1 Exercise 3.1: Inefficient Simulation of `NULLS FIRST`

A developer needs to list employees, ensuring those with a ‘NULL‘ ‘bonus_percentage‘ appear first, followed by others sorted by their ‘bonus_percentage‘ in ascending order. They implemented this using a ‘CASE‘ statement in the ‘ORDER BY‘ clause:

```
ORDER BY (CASE WHEN bonus_percentage IS NULL THEN 0 ELSE 1 END) ASC, bonus_percentage
ASC;
```

Provide the more direct "Advanced ORDER BY" equivalent using ‘NULLS FIRST‘. Explain why the direct approach is generally preferred over the ‘CASE‘ statement method for this specific task of handling NULLs in sorting.

3.2 Exercise 3.2: Inefficient Custom Sort Order Implementation

A user wants to display employees with a specific ‘department‘ order: 'Sales' first, then 'Engineering', then all other departments alphabetically. Within each of these department groups, employees should be sorted by ‘salary‘ in descending order. An inefficient approach might involve fetching data for each department group separately and then trying to combine them (e.g., using ‘UNION ALL‘ with artificial sort keys). Demonstrate how a single query using ‘CASE‘ within the ‘ORDER BY‘ clause for the custom department sort, combined with multi-column sorting, is a vastly superior and more efficient solution.

4 Hardcore Problem Combining Concepts

4.1 Exercise 4.1: Comprehensive Employee Ranking and Reporting

List employees who were hired on or after January 1st, 2021, and work in departments whose ‘location‘ (from the ‘departments‘ table) is either ’New York’ or ’San Francisco’.

For these selected employees, calculate their rank within their respective ‘department‘ based on their ‘salary‘ (highest salary first). If salaries are tied within a department, the tie-breaking rules for ranking are as follows:

1. Employees with a non-NULL ‘bonus_percentage‘ should come before those with a ‘NULL‘ ‘bonus_percentage‘.

2. If the ‘bonus_percentage‘ status (NULL or not NULL) is also the same, or if both have non-NULL bonuses, further sort by ‘bonus_percentage‘ itself in descending order (higher bonus is better).

3. If ‘bonus_percentage‘ values (or their NULL status where both are NULL) are also tied, further sort by ‘hire_date‘ in ascending order (earlier hire date first).

Display the employee’s full name (concatenated ‘first_name‘ and ‘last_name‘), their ‘department‘ name, ‘salary‘, ‘hire_date‘, ‘bonus_percentage‘ (display ’N/A’ if NULL), and their calculated rank.

A crucial condition: Only include employees from departments where the total ‘hours_assigned‘ to projects for that entire department (sum of ‘hours_assigned‘ from the ‘employee_projects‘ table for all employees in that department) is greater than 200 hours. Employees who themselves have no projects should still be included in the ranking if they meet other criteria and their department meets this total hours threshold.

The final result set must be ordered by ‘department‘ name (A-Z) and then by the calculated ‘department_rank‘ (ascending).