Conditionals: Advanced WHERE

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 projects CASCADE;
DROP TABLE IF EXISTS employees CASCADE;
DROP TABLE IF EXISTS departments CASCADE;

-- Create Departments Table
CREATE TABLE departments (
    dept_id SERIAL PRIMARY KEY,
    dept_name VARCHAR(50) NOT NULL,
    location VARCHAR(50),
    monthly_budget NUMERIC(10,2) NULL
);

-- Create Employees Table
CREATE TABLE employees (
    emp_id SERIAL PRIMARY KEY,
    emp_name VARCHAR(100) NOT NULL,
    dept_id INTEGER REFERENCES departments(dept_id),
    salary NUMERIC(10,2) NOT NULL,
    manager_id INTEGER REFERENCES employees(emp_id) NULL,
    performance_rating INTEGER NULL CHECK (performance_rating IS NULL OR
    performance_rating BETWEEN 1 AND 5),
    last_bonus NUMERIC(8,2) NULL,
    hire_date DATE NOT NULL
);

-- Create Projects Table
CREATE TABLE projects (
    proj_id SERIAL PRIMARY KEY,
    proj_name VARCHAR(100) NOT NULL,
    lead_emp_id INTEGER REFERENCES employees(emp_id) NULL,
    budget NUMERIC(12,2),
    start_date DATE NOT NULL,
    end_date DATE NULL
);

-- Create Employee_Projects Junction Table
CREATE TABLE employee_projects (
    emp_id INTEGER REFERENCES employees(emp_id),
    proj_id INTEGER REFERENCES projects(proj_id),
    role VARCHAR(50),
    hours_assigned INTEGER NULL,
    PRIMARY KEY (emp_id, proj_id)
);

-- Populate Departments
INSERT INTO departments (dept_name, location, monthly_budget) VALUES
('Human Resources', 'New York', 50000.00),
('Technology', 'San Francisco', 75000.00),
('Sales', 'Chicago', 60000.00),
('Support', 'Austin', 40000.00),
('Research', 'Boston', NULL), -- Budget is NULL
('Operations', 'New York', 50000.00);

-- Populate Employees
-- Top Managers (no manager_id)
INSERT INTO employees (emp_name, dept_id, salary, manager_id, performance_rating,
    last_bonus, hire_date) VALUES
('Alice Wonderland', 1, 90000.00, NULL, 5, 10000.00, '2010-03-15'),
('Bob The Builder', 2, 95000.00, NULL, 4, 8000.00, '2008-07-01');

-- Other Employees
INSERT INTO employees (emp_name, dept_id, salary, manager_id, performance_rating,
    last_bonus, hire_date) VALUES
('Charlie Brown', 1, 60000.00, 1, 3, 3000.00, '2012-05-20'), -- HR
('Diana Prince', 2, 75000.00, 2, 5, 7000.00, '2015-11-01'), -- Tech
('Edward Scissorhands', 2, 70000.00, 2, 2, NULL, '2016-02-10'), -- Tech, NULL bonus, low
     rating
```

```sql
('Fiona Apple', 3, 65000.00, NULL, 4, 5000.00, '2018-08-01'), -- Sales, no manager_id in
    this context
('George Jetson', 3, 55000.00, 6, 3, 2500.00, '2019-01-15'), -- Sales
('Hannah Montana', 4, 50000.00, 1, NULL, 1500.00, '2020-06-01'), -- Support, NULL rating
('Ivan Drago', 4, 48000.00, 8, 2, 1000.00, '2021-03-10'), -- Support
('Julia Child', 5, 80000.00, NULL, 5, NULL, '2011-09-05'), -- Research, NULL bonus
('Kevin McCallister', 1, 58000.00, 1, 4, 2000.00, '2013-07-22'), -- HR
('Laura Palmer', 2, 82000.00, 2, 3, 4000.00, '2014-01-30'), -- Tech
('Michael Knight', 3, 68000.00, 6, 5, 6000.00, '2017-04-11'), -- Sales
('Nancy Drew', 4, 52000.00, 8, 4, NULL, '2019-10-01'), -- Support, NULL bonus
('Oscar Wilde', 5, 78000.00, 10, NULL, 7500.00, '2022-01-20'); -- Research, NULL rating

UPDATE employees SET manager_id = 1 WHERE emp_name = 'Charlie Brown';
UPDATE employees SET manager_id = 1 WHERE emp_name = 'Kevin McCallister';
UPDATE employees SET manager_id = 2 WHERE emp_name = 'Diana Prince';
UPDATE employees SET manager_id = 2 WHERE emp_name = 'Edward Scissorhands';
UPDATE employees SET manager_id = 2 WHERE emp_name = 'Laura Palmer';
UPDATE employees SET manager_id = 6 WHERE emp_name = 'George Jetson';
UPDATE employees SET manager_id = 6 WHERE emp_name = 'Michael Knight';
INSERT INTO employees (emp_name, dept_id, salary, manager_id, performance_rating,
    last_bonus, hire_date) VALUES
('Peter Pan', NULL, 30000.00, NULL, 3, NULL, '2023-01-01'); -- No department, NULL bonus

-- Populate Projects
INSERT INTO projects (proj_name, lead_emp_id, budget, start_date, end_date) VALUES
('Alpha Launch', 4, 150000.00, '2023-01-01', '2023-12-31'), -- Lead: Diana Prince (Tech)
('Beta Test', 5, 80000.00, '2023-03-01', '2023-09-30'), -- Lead: Edward Scissorhands (
    Tech)
('Gamma Initiative', 1, 200000.00, '2022-06-01', NULL), -- Lead: Alice Wonderland (HR)
('Delta Rollout', 13, 120000.00, '2024-02-01', NULL), -- Lead: Michael Knight (Sales)
('Epsilon Research', 10, 90000.00, '2023-05-01', '2024-05-01'), -- Lead: Julia Child (
    Research)
('NoLead Project', NULL, 50000.00, '2023-07-01', NULL); -- NULL lead_emp_id

-- Populate Employee_Projects
INSERT INTO employee_projects (emp_id, proj_id, role, hours_assigned) VALUES
(4, 1, 'Developer', 160), -- Diana on Alpha
(5, 1, 'QA Engineer', 120), -- Edward on Alpha
(12, 1, 'UI Designer', 100), -- Laura on Alpha
(5, 2, 'Lead Tester', 150), -- Edward on Beta
(9, 2, 'Tester', 80), -- Ivan on Beta
(1, 3, 'Project Manager', 200), -- Alice on Gamma
(3, 3, 'Coordinator', NULL), -- Charlie on Gamma, NULL hours
(11, 3, 'Analyst', 100), -- Kevin on Gamma
(13, 4, 'Sales Lead', 180), -- Michael on Delta
(7, 4, 'Sales Rep', 140), -- George on Delta
(10, 5, 'Lead Researcher', 190), -- Julia on Epsilon
(15, 5, 'Researcher', NULL); -- Oscar on Epsilon, NULL hours

-- Add an employee in a department that will be used for NOT IN examples
INSERT INTO departments (dept_name, location, monthly_budget) VALUES ('Intern Pool', '
    Remote', 10000.00);
INSERT INTO employees (emp_name, dept_id, salary, manager_id, performance_rating,
    last_bonus, hire_date) VALUES
('Intern Zero', (SELECT dept_id FROM departments WHERE dept_name = 'Intern Pool'),
    20000.00, NULL, NULL, NULL, '2024-06-01');

-- Add a department with no employees for Exercise III.4
INSERT INTO departments (dept_name, location, monthly_budget) VALUES ('Empty Department'
    , 'Nowhere', 1000.00);
```

Listing 1: Dataset for Advanced WHERE Conditions Exercises

1 Practice Meanings, Values, Relations, Advantages, and Unique Uses

1.1 Subquery with `IN`

Problem: List the names and salaries of all employees who work in departments located in 'New York'.

1.2 Subquery with `EXISTS`

Problem: Find the names of departments that have at least one employee with a salary greater than $85,000.

1.3 Subquery with `ANY`

Problem: List employees whose salary is greater than *any* salary in the 'Support' department. (This means their salary is greater than the minimum salary in the 'Support' department).

1.4 Subquery with `ALL`

Problem: Find employees in the 'Sales' department whose salary is less than *all* salaries in the 'Technology' department.

1.5 `IS DISTINCT FROM`

Problem: List employees whose `performance_rating` is different from 3. This list should include employees who have a NULL `performance_rating` (as NULL is distinct from 3).

1.6 `IS NOT DISTINCT FROM`

Problem: Find pairs of employees (display their names) who have the exact same `manager_id`, including cases where both employees have no manager (i.e., their `manager_id` is NULL). Avoid listing an employee paired with themselves.

2 Practice Disadvantages of All Its Technical Concepts

2.1 `NOT IN` with Subquery Returning NULL

Problem: Attempt to list employees who are NOT leads on any project using the condition `emp_id NOT IN (SELECT lead_emp_id FROM projects)`. Observe the (potentially unexpected) result given that `projects.lead_emp_id` can be NULL (e.g., 'NoLead Project'). Explain why this happens.

2.2 `!= ANY` Misinterpretation

Problem: Find employees whose salary is not equal to *any* salary found in the 'Intern Pool' department. The 'Intern Pool' department currently has one employee ('Intern Zero') with a salary of $20,000. Consider what happens if the 'Intern Pool' department had multiple distinct salaries (e.g., $20,000, $22,000). Explain the logical evaluation of `salary != ANY (subquery_salaries)` in such a scenario.

2.3 Performance of `IS DISTINCT FROM` vs. Standard Operators (Conceptual)

Problem: Consider finding employees where `performance_rating` is 3.

- Compare conceptually querying this using `performance_rating = 3` versus `performance_rating IS NOT DISTINCT FROM 3`.
- When might the `IS NOT DISTINCT FROM` approach be slightly less optimal if `performance_rating` is indexed and guaranteed `NOT NULL`? Discuss potential minor overheads or familiarity issues for developers.

2.4 Readability of `EXISTS` vs. `IN` for Simple Cases

Problem: Retrieve employees who are in departments listed in a small, explicit list of department IDs (e.g., department IDs 1 and 2).

- Write a query fragment using `IN` with a literal list.
- Write a query fragment using `EXISTS` with a subquery that provides these values.
- Compare the readability and conciseness of these two approaches for this specific simple case.

3 Practice Inefficient/Incorrect Alternatives vs. Advanced WHERE Conditions

3.1 Inefficient `COUNT(*)` vs. `EXISTS` for Existence Check

Problem: List department names that have at least one project associated with them (i.e., a project where `lead_emp_id` belongs to an employee in that department).

- **Task:** Write a query using an inefficient approach: a correlated subquery with `COUNT(*)` in the `WHERE` clause, checking if `COUNT(*) > 0`.
- **Consideration:** Why is using `EXISTS` generally more efficient for this type of "is there at least one?" check?

3.2 Verbose/Incorrect NULL Handling vs. `IS DISTINCT FROM`

Problem: Find employees whose `last_bonus` is not $5000.00. This list should include employees whose `last_bonus` is NULL, as NULL is considered different from $5000.00.

- **Task:** Write a query using a verbose approach: `(last_bonus <> 5000.00 OR last_bonus IS NULL)`.
- **Consideration:** How does `last_bonus IS DISTINCT FROM 5000.00` offer a more concise and less error-prone solution?

3.3 Complex NULL-aware Equality vs. `IS NOT DISTINCT FROM`

Problem: Find all employees whose `manager_id` is the same as Peter Pan's `manager_id`. (Peter Pan, emp_id 16, has a NULL `manager_id`). Do not include Peter Pan himself in the results.

- **Task:** Write a query using a complex approach: explicitly check for equality of `manager_id` with Peter Pan's `manager_id`, AND explicitly check if both the employee's `manager_id` and Peter Pan's `manager_id` are NULL.
- **Consideration:** How does using `IS NOT DISTINCT FROM` simplify this NULL-aware equality check and make the query more robust and readable?

3.4 Using `LEFT JOIN` and checking for NULL vs. `NOT EXISTS`

Problem: Find departments that have no employees. (Note: The dataset includes an 'Empty Department' specifically for this exercise).

- **Task:** Write a query using a common approach: `LEFT JOIN` the `departments` table with the `employees` table and then filter for departments where the employee's primary key (or any non-nullable employee column from the join) is NULL. Another variant could use `GROUP BY` and `HAVING COUNT(e.emp_id) = 0`.
- **Consideration:** For the specific task of checking non-existence, how does `NOT EXISTS` compare in terms of directness and potential efficiency?

4 Hardcore Problem Combining Previous Concepts

Problem Statement:

Identify "Key Departments" based on the following criteria:

1. The department name must contain either 'Tech' or 'HR' (case-sensitive as per standard SQL `LIKE`).
2. The department's `monthly_budget` IS NULL, OR its `monthly_budget` = 50000.00.
3. The department must have at least one "veteran" employee associated with it. This is determined by checking if there `EXISTS` such an employee in the department. A "veteran" employee is defined as someone:
   - Hired more than 8 years ago from `CURRENT_DATE`.
   - Whose `performance_rating IS DISTINCT FROM` 1 (i.e., their rating is not 1, or their rating is NULL).

For these "Key Departments", calculate the total `hours_assigned` to all their employees for projects that started on or after '2023-01-01'. If a department has no such employees or projects meeting this date criterion, their total hours should be displayed as 0.

Display the `dept_name` and the calculated `total_project_hours`. Order the results in descending order of `total_project_hours`, then by `dept_name` alphabetically. Limit the result to the top 3 departments.

Note for execution: To get more illustrative results for this specific hardcore problem, you might consider temporarily updating the Technology department's budget before running your solution:

```sql
-- UPDATE departments SET monthly_budget = 50000.00 WHERE dept_name = 'Technology';
```

Listing 2: Optional temporary data modification for Hardcore Problem

Remember to revert this change if you want to work with the original dataset for other exercises.

Previous Concepts to Integrate:

- **Advanced WHERE:** `EXISTS`, `IS DISTINCT FROM`.
- **Basic WHERE:** `LIKE`, `OR`, date comparisons (`>=`), arithmetic with dates (`CURRENT_DATE - INTERVAL`).
- **Intermediate SQL:**
  - Joins: `INNER JOIN`, `LEFT JOIN`.
  - Aggregators: `SUM()`, `GROUP BY`.
  - Null Space: `COALESCE`.
  - Date Functions: `CURRENT_DATE`, `INTERVAL`.
- **Ordering & Limiting:** `ORDER BY`, `LIMIT`.
- Consider using Common Table Expressions (CTEs) for clarity.