Conditionals: Advanced WHERE Statements

Complementary SQL: Solutions

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'),
('Beta Test', 5, 80000.00, '2023-03-01', '2023-09-30'),
('Gamma Initiative', 1, 200000.00, '2022-06-01', NULL),
('Delta Rollout', 13, 120000.00, '2024-02-01', NULL),
('Epsilon Research', 10, 90000.00, '2023-05-01', '2024-05-01'),
('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 section III, exercise 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

Exercise 1: Subquery with `IN`

```sql
SELECT emp_name, salary
FROM employees
WHERE dept_id IN (SELECT dept_id FROM departments WHERE location = 'New York');
```

Listing 2: Solution for I.1: Subquery with IN

Exercise 2: Subquery with `EXISTS`

```sql
SELECT d.dept_name
FROM departments d
WHERE EXISTS (
    SELECT 1
    FROM employees e
    WHERE e.dept_id = d.dept_id AND e.salary > 85000.00
);
```

Listing 3: Solution for I.2: Subquery with EXISTS

Exercise 3: Subquery with `ANY`

```sql
SELECT emp_name, salary
FROM employees
WHERE salary > ANY (
    SELECT e.salary
    FROM employees e
    JOIN departments d ON e.dept_id = d.dept_id
    WHERE d.dept_name = 'Support'
);
```

Listing 4: Solution for I.3: Subquery with ANY

Exercise 4: Subquery with `ALL` (Revised)

```sql
SELECT e.emp_name, e.salary, d_emp.dept_name
FROM employees e
JOIN departments d_emp ON e.dept_id = d_emp.dept_id
WHERE d_emp.dept_name = 'Sales' AND e.salary < ALL (
    SELECT e_tech.salary
    FROM employees e_tech
    JOIN departments d_tech ON e_tech.dept_id = d_tech.dept_id
    WHERE d_tech.dept_name = 'Technology'
);
```

Listing 5: Solution for I.4: Subquery with ALL (Revised)

Exercise 5: `IS DISTINCT FROM`

```sql
SELECT emp_name, performance_rating
FROM employees
WHERE performance_rating IS DISTINCT FROM 3;
```

Listing 6: Solution for I.5: IS DISTINCT FROM

Exercise 6: `IS NOT DISTINCT FROM`

```sql
SELECT e1.emp_name AS employee1, e2.emp_name AS employee2, e1.manager_id
FROM employees e1
JOIN employees e2 ON e1.emp_id < e2.emp_id -- Avoid self-join and duplicate pairs
WHERE e1.manager_id IS NOT DISTINCT FROM e2.manager_id
ORDER BY e1.manager_id, e1.emp_name, e2.emp_name;
```

Listing 7: Solution for I.6: IS NOT DISTINCT FROM

2 Practice Disadvantages of All Its Technical Concepts

Exercise 1: NOT IN with Subquery Returning NULL

```sql
-- This query might yield unexpected results (likely empty or fewer than expected)
-- because projects.lead_emp_id includes NULL (from ’NoLead Project’).
SELECT emp_name
FROM employees
WHERE emp_id NOT IN (SELECT DISTINCT lead_emp_id FROM projects);

-- To see the lead_emp_id values including NULL:
-- SELECT DISTINCT lead_emp_id FROM projects;

-- Corrected approaches:
-- 1. Filter NULLs from subquery:
-- SELECT emp_name
-- FROM employees
-- WHERE emp_id NOT IN (SELECT lead_emp_id FROM projects WHERE lead_emp_id IS NOT NULL);
-- 2. Use NOT EXISTS:
-- SELECT e.emp_name
-- FROM employees e
-- WHERE NOT EXISTS (
--     SELECT 1
--     FROM projects p
--     WHERE p.lead_emp_id = e.emp_id
-- );
```

Listing 8: Solution for II.1: Demonstrating NOT IN with NULL issue

Exercise 2: != ANY Misinterpretation

```sql
-- ’Intern Pool’ currently has one salary (20000 for ’Intern Zero’)
SELECT emp_name, salary
FROM employees
WHERE salary != ANY (
    SELECT e.salary
    FROM employees e
    JOIN departments d ON e.dept_id = d.dept_id
    WHERE d.dept_name = ’Intern Pool’
);
-- This lists everyone EXCEPT ’Intern Zero’.
-- This is because ‘salary != ANY (20000)‘ becomes ‘salary != 20000‘.

-- If ’Intern Pool’ had salaries (20000, 22000), then:
-- ‘salary != ANY (20000, 22000)‘ translates to ‘salary != 20000 OR salary != 22000‘.
-- This condition is TRUE for any salary value.
-- E.g., If salary = 20000: (20000 != 20000 [FALSE]) OR (20000 != 22000 [TRUE]) -> TRUE
-- E.g., If salary = 22000: (22000 != 20000 [TRUE]) OR (22000 != 22000 [FALSE]) -> TRUE
-- E.g., If salary = 21000: (21000 != 20000 [TRUE]) OR (21000 != 22000 [TRUE]) -> TRUE
-- This operator is often confused with ‘NOT IN (value1, value2)‘ or ‘column != ALL (...)‘.
```

Listing 9: Solution for II.2: Demonstrating != ANY behavior

Exercise 3: Performance of `IS DISTINCT FROM` vs. Standard Operators

```sql
-- Standard equality, potentially more direct for indexed, NOT NULL columns:
-- SELECT emp_name FROM employees WHERE performance_rating = 3;

-- IS NOT DISTINCT FROM handles NULLs as equal, standard ’=’ treats NULL = NULL as UNKNOWN.
SELECT emp_name FROM employees WHERE performance_rating IS NOT DISTINCT FROM 3;

-- The "disadvantage" is minor potential overhead IF the column is NOT NULL
-- AND indexed, where ’=’ might be slightly more optimized.
-- The main advantage of IS [NOT] DISTINCT FROM is correct NULL handling,
-- which often outweighs slight performance considerations.
```

Listing 10: Solution for II.3: Comparing IS DISTINCT FROM

Exercise 4: Readability of `EXISTS` vs. `IN` for Simple Cases

```sql
-- Using IN (more direct and often more readable for simple, static lists)
SELECT emp_name
FROM employees
WHERE dept_id IN (1, 2);

-- Using EXISTS (more verbose for this specific simple case)
SELECT e.emp_name
FROM employees e
WHERE EXISTS (
    SELECT 1
    FROM (VALUES (1), (2)) AS d_list(dept_id_val) -- Subquery to provide values
    WHERE e.dept_id = d_list.dept_id_val
);
-- While EXISTS is powerful, for a short, static list of values, IN is usually clearer.
```

Listing 11: Solution for II.4: Comparing IN vs. EXISTS for simple lists

3 Practice Inefficient/Incorrect Alternatives vs. Advanced WHERE Conditions

Exercise 1: Inefficient `COUNT(*)` vs. `EXISTS` for Existence Check

```sql
-- Inefficient Approach (using COUNT)
SELECT d.dept_name
FROM departments d
WHERE (
    SELECT COUNT(*)
    FROM projects p
    JOIN employees e ON p.lead_emp_id = e.emp_id
    WHERE e.dept_id = d.dept_id
) > 0;
```

Listing 12: Inefficient Approach: Exercise III.1

```sql
-- Efficient Approach (using EXISTS)
SELECT d.dept_name
FROM departments d
WHERE EXISTS (
    SELECT 1
    FROM projects p
    JOIN employees e ON p.lead_emp_id = e.emp_id
    WHERE e.dept_id = d.dept_id
);
```

Listing 13: Efficient Approach (using EXISTS): Exercise III.1

Exercise 2: Verbose/Incorrect NULL Handling vs. `IS DISTINCT FROM`

```sql
-- Inefficient/Verbose Approach
SELECT emp_name, last_bonus
FROM employees
WHERE (last_bonus <> 5000.00 OR last_bonus IS NULL);
```

Listing 14: Inefficient/Verbose Approach: Exercise III.2

```sql
-- Efficient/Clear Approach
SELECT emp_name, last_bonus
FROM employees
WHERE last_bonus IS DISTINCT FROM 5000.00;
```

Listing 15: Efficient/Clear Approach (using IS DISTINCT FROM): Exercise III.2

Exercise 3: Complex NULL-aware Equality vs. `IS NOT DISTINCT FROM`

```sql
-- Complex Approach (Example: Compare with Peter Pan's manager_id, which is NULL)
-- (Peter Pan emp_id = 16, manager_id IS NULL)
SELECT emp_name, manager_id
FROM employees
```

```sql
WHERE emp_id != 16 AND ( -- Exclude Peter Pan himself
    (manager_id = (SELECT manager_id FROM employees WHERE emp_id = 16)) OR
    (manager_id IS NULL AND (SELECT manager_id FROM employees WHERE emp_id = 16) IS NULL)
);
```

Listing 16: Complex Approach for NULL-aware equality: Exercise III.3

```sql
-- Clear Approach (using IS NOT DISTINCT FROM Peter Pan's manager_id)
-- (Peter Pan emp_id = 16, manager_id IS NULL)
SELECT e.emp_name, e.manager_id
FROM employees e
WHERE e.emp_id != 16 AND e.manager_id IS NOT DISTINCT FROM (
    SELECT m.manager_id FROM employees m WHERE m.emp_id = 16
);
```

Listing 17: Clear Approach (using IS NOT DISTINCT FROM): Exercise III.3

Exercise 4: Using LEFT JOIN / IS NULL vs. NOT EXISTS

```sql
-- Common Approach (LEFT JOIN / IS NULL)
-- This version correctly identifies departments with no employees directly
SELECT d.dept_name
FROM departments d
LEFT JOIN employees e ON d.dept_id = e.dept_id
WHERE e.emp_id IS NULL;

-- Alternative common approach (LEFT JOIN / GROUP BY / HAVING COUNT = 0)
-- SELECT d.dept_name
-- FROM departments d
-- LEFT JOIN employees e ON d.dept_id = e.dept_id
-- GROUP BY d.dept_id, d.dept_name
-- HAVING COUNT(e.emp_id) = 0;
```

Listing 18: Common Approach (LEFT JOIN / IS NULL or COUNT): Exercise III.4

```sql
-- Efficient/Clear Approach (NOT EXISTS)
SELECT d.dept_name
FROM departments d
WHERE NOT EXISTS (
    SELECT 1
    FROM employees e
    WHERE e.dept_id = d.dept_id
);
```

Listing 19: Efficient/Clear Approach (using NOT EXISTS): Exercise III.4

4 Hardcore Problem Combining Previous Concepts

For a more illustrative result set from the hardcore problem, consider temporarily updating the ’Technology’ department’s budget:

```sql
-- Run this before the main query to see 'Technology' potentially in results:
-- UPDATE departments SET monthly_budget = 50000.00 WHERE dept_name = 'Technology';

-- Remember to revert if needed:
-- UPDATE departments SET monthly_budget = 75000.00 WHERE dept_name = 'Technology';
```

Listing 20: Optional: Temporary Data Update for Hardcore Problem

```sql
WITH VeteranDepartments AS (
    SELECT DISTINCT d.dept_id
    FROM departments d
    JOIN employees e ON d.dept_id = e.dept_id
    WHERE e.hire_date < (CURRENT_DATE - INTERVAL '8 years')
      AND e.performance_rating IS DISTINCT FROM 1
),
KeyDepartments AS (
    SELECT d.dept_id, d.dept_name
    FROM departments d
    WHERE (d.dept_name LIKE '%Tech%' OR d.dept_name LIKE '%HR%')
      AND (d.monthly_budget IS NULL OR d.monthly_budget = 50000.00)
      AND EXISTS (
          SELECT 1
          FROM VeteranDepartments vd
          WHERE vd.dept_id = d.dept_id
      )
)
SELECT
    kd.dept_name,
    COALESCE(SUM(ep.hours_assigned), 0) AS total_project_hours
FROM KeyDepartments kd
LEFT JOIN employees e ON kd.dept_id = e.dept_id
LEFT JOIN employee_projects ep ON e.emp_id = ep.emp_id
LEFT JOIN projects p ON ep.proj_id = p.proj_id AND p.start_date >= '2023-01-01'
GROUP BY kd.dept_id, kd.dept_name
ORDER BY total_project_hours DESC, kd.dept_name ASC
LIMIT 3;
```

Listing 21: Solution for Hardcore Problem (IV)