Date Functions, Cases, and Null Space

Complementary SQL Exercises

May 13, 2025

Contents

Global Dataset for PostgreSQL

The following SQL code creates and populates the necessary tables for the exercises. Execute this script in your PostgreSQL environment before attempting the exercises.

```sql
-- Drop tables if they exist to ensure a clean setup
DROP TABLE IF EXISTS leave_requests CASCADE;
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;

-- Departments Table
CREATE TABLE departments (
    dept_id SERIAL PRIMARY KEY,
    dept_name VARCHAR(100) NOT NULL UNIQUE,
    creation_date DATE NOT NULL,
    location VARCHAR(50)
);

-- Employees Table
CREATE TABLE employees (
    emp_id SERIAL PRIMARY KEY,
    emp_name VARCHAR(100) NOT NULL,
    hire_date DATE NOT NULL,
    salary NUMERIC(10, 2),
    dept_id INT REFERENCES departments(dept_id),
    manager_id INT REFERENCES employees(emp_id), -- Self-reference for manager
    termination_date DATE, -- NULL if currently employed
    email VARCHAR(100) UNIQUE,
    performance_rating INT CHECK (performance_rating BETWEEN 1 AND 5 OR
    performance_rating IS NULL)
);

-- Projects Table
CREATE TABLE projects (
    project_id SERIAL PRIMARY KEY,
    project_name VARCHAR(100) NOT NULL UNIQUE,
    start_date DATE NOT NULL,
    planned_end_date DATE NOT NULL,
    actual_end_date DATE, -- NULL if not completed
    budget NUMERIC(12, 2),
    lead_emp_id INT REFERENCES employees(emp_id),
    CONSTRAINT check_project_dates CHECK (planned_end_date >= start_date)
);

-- Employee_Projects Table (Junction table)
CREATE TABLE employee_projects (
    emp_project_id SERIAL PRIMARY KEY,
    emp_id INT REFERENCES employees(emp_id) ON DELETE CASCADE,
    project_id INT REFERENCES projects(project_id) ON DELETE CASCADE,
    assigned_date DATE NOT NULL,
    role VARCHAR(50),
    hours_billed NUMERIC(6, 2) DEFAULT 0.00,
    billing_rate NUMERIC(8, 2),
    completion_date DATE, -- Date employee completed their part
    UNIQUE(emp_id, project_id) -- An employee has one role per project
);

-- Leave_Requests Table
CREATE TABLE leave_requests (
    leave_id SERIAL PRIMARY KEY,
    emp_id INT REFERENCES employees(emp_id) ON DELETE CASCADE,
    request_date TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    leave_start_date DATE NOT NULL,
    leave_end_date DATE NOT NULL,
    status VARCHAR(20) CHECK (status IN ('Pending', 'Approved', 'Rejected', 'Cancelled')
    ),
    approved_by_manager_id INT REFERENCES employees(emp_id),
    reason TEXT,
    CONSTRAINT check_leave_dates CHECK (leave_end_date >= leave_start_date)
);
```

```sql

-- Populate Departments
INSERT INTO departments (dept_name, creation_date, location) VALUES
('Human Resources', '2010-01-15', 'New York'),
('Engineering', '2010-03-01', 'San Francisco'),
('Sales', '2010-02-01', 'Chicago'),
('Marketing', '2011-05-20', 'New York'),
('Finance', '2010-01-20', 'New York');

-- Populate Employees (managers first, then their reports)
INSERT INTO employees (emp_name, hire_date, salary, dept_id, manager_id,
    termination_date, email, performance_rating) VALUES
('Alice Wonderland', '2015-06-01', 90000.00, 2, NULL, NULL, 'alice@example.com', 5), --
    Eng Manager
('David Copperfield', '2015-03-10', 120000.00, 3, NULL, NULL, 'david@example.com', 4),
    -- Sales Manager
('Frankenstein Monster', '2019-11-01', 95000.00, 1, NULL, NULL, 'frank@example.com', 3),
     -- HR Manager
('Ivy Poison', '2022-09-15', 110000.00, 5, NULL, NULL, 'ivy@example.com', 4); -- Finance
     Head

INSERT INTO employees (emp_name, hire_date, salary, dept_id, manager_id,
    termination_date, email, performance_rating) VALUES
('Bob The Builder', '2016-08-15', 75000.00, 2, (SELECT emp_id from employees WHERE email
    ='alice@example.com'), NULL, 'bob@example.com', 4),
('Carol Danvers', '2017-01-20', 80000.00, 2, (SELECT emp_id from employees WHERE email='
    alice@example.com'), NULL, 'carol@example.com', 5),
('Eve Harrington', '2018-07-01', 65000.00, 3, (SELECT emp_id from employees WHERE email=
    'david@example.com'), '2023-12-31', 'eve@example.com', 2),
('Grace Hopper', '2020-02-10', 70000.00, 1, (SELECT emp_id from employees WHERE email='
    frank@example.com'), NULL, 'grace@example.com', NULL),
('Henry Jekyll', '2021-05-01', 50000.00, 4, (SELECT emp_id from employees WHERE email='
    david@example.com'), NULL, 'henry@example.com', 3), -- Marketing, reports to Sales
    head for now
('Jack Sparrow', '2023-01-20', 60000.00, 5, (SELECT emp_id from employees WHERE email='
    ivy@example.com'), '2023-08-15', 'jack@example.com', 1),
('Kevin McCallister', '2018-09-01', 72000.00, 2, (SELECT emp_id from employees WHERE
    email='alice@example.com'), NULL, 'kevin@example.com', NULL),
('Laura Croft', '2019-04-15', 85000.00, 4, (SELECT emp_id from employees WHERE email='
    david@example.com'), NULL, 'laura@example.com', 5); -- Marketing, reports to Sales
    head

-- Populate Projects
INSERT INTO projects (project_name, start_date, planned_end_date, actual_end_date,
    budget, lead_emp_id) VALUES
('Project Alpha', '2023-01-01', '2023-06-30', '2023-07-15', 100000.00, (SELECT emp_id
    from employees WHERE email='alice@example.com')),
('Project Beta', '2023-03-01', '2023-09-30', NULL, 150000.00, (SELECT emp_id from
    employees WHERE email='bob@example.com')),
('Project Gamma', '2022-09-01', '2023-03-31', '2023-03-20', 80000.00, (SELECT emp_id
    from employees WHERE email='alice@example.com')),
('Project Delta', '2023-08-01', '2024-02-29', NULL, 200000.00, (SELECT emp_id from
    employees WHERE email='david@example.com')),
('Project Epsilon', '2023-05-01', '2023-08-31', '2023-09-05', 60000.00, NULL),
('Project Zeta', '2024-01-01', '2024-01-30', NULL, 120000.00, (SELECT emp_id from
    employees WHERE email='bob@example.com')), -- Ends Jan 30, 2024
('Project Omega', '2023-10-01', '2023-12-20', '2023-12-20', 50000.00, (SELECT emp_id
    from employees WHERE email='carol@example.com')),
('Critical Eng Proj 1', '2024-01-02', '2024-01-25', NULL, 5000, (SELECT emp_id from
    employees WHERE email='alice@example.com')), -- Critical for report
('Critical Eng Proj 2', '2024-01-05', '2024-02-10', NULL, 5000, (SELECT emp_id from
    employees WHERE email='bob@example.com')); -- Critical for report

-- Populate Employee_Projects
INSERT INTO employee_projects (emp_id, project_id, assigned_date, role, hours_billed,
    billing_rate, completion_date) VALUES
((SELECT emp_id from employees WHERE email='alice@example.com'), (SELECT project_id from
     projects WHERE project_name='Project Alpha'), '2023-01-01', 'Project Manager', 200,
     150.00, '2023-07-15'),
((SELECT emp_id from employees WHERE email='alice@example.com'), (SELECT project_id from
     projects WHERE project_name='Project Gamma'), '2022-09-01', 'Project Manager', 180,
     150.00, '2023-03-20'),
```

```
((SELECT emp_id from employees WHERE email='bob@example.com'), (SELECT project_id from projects WHERE project_name='Project Alpha'), '2023-01-05', 'Developer', 300, 120.00, '2023-07-10'),
((SELECT emp_id from employees WHERE email='bob@example.com'), (SELECT project_id from projects WHERE project_name='Project Beta'), '2023-03-01', 'Lead Developer', 250, 130.00, NULL),
((SELECT emp_id from employees WHERE email='bob@example.com'), (SELECT project_id from projects WHERE project_name='Project Zeta'), '2024-01-01', 'Lead Developer', 50, 135.00, NULL),
((SELECT emp_id from employees WHERE email='carol@example.com'), (SELECT project_id from projects WHERE project_name='Project Beta'), '2023-03-05', 'Developer', 220, 120.00, NULL),
((SELECT emp_id from employees WHERE email='carol@example.com'), (SELECT project_id from projects WHERE project_name='Project Epsilon'), '2023-05-01', 'Consultant', 100, 200.00, '2023-09-05'),
((SELECT emp_id from employees WHERE email='carol@example.com'), (SELECT project_id from projects WHERE project_name='Project Omega'), '2023-10-01', 'Developer', 80, 125.00, '2023-12-20'),
((SELECT emp_id from employees WHERE email='eve@example.com'), (SELECT project_id from projects WHERE project_name='Project Delta'), '2023-08-01', 'Sales Rep', 150, 100.00, '2023-12-31'),
((SELECT emp_id from employees WHERE email='grace@example.com'), (SELECT project_id from projects WHERE project_name='Project Alpha'), '2023-02-01', 'HR Coordinator', 80, NULL, '2023-07-15'),
((SELECT emp_id from employees WHERE email='kevin@example.com'), (SELECT project_id from projects WHERE project_name='Project Beta'), '2023-03-15', 'QA Engineer', 180, 110.00, NULL),
((SELECT emp_id from employees WHERE email='laura@example.com'), (SELECT project_id from projects WHERE project_name='Project Delta'), '2023-08-05', 'Marketing Lead', 200, 140.00, NULL);

-- Populate Leave_Requests
INSERT INTO leave_requests (emp_id, request_date, leave_start_date, leave_end_date, status, approved_by_manager_id, reason) VALUES
((SELECT emp_id from employees WHERE email='bob@example.com'), '2023-04-01 10:00:00', '2023-04-10', '2023-04-12', 'Approved', (SELECT emp_id from employees WHERE email='alice@example.com'), 'Vacation'),
((SELECT emp_id from employees WHERE email='carol@example.com'), '2023-05-10 14:30:00', '2023-06-01', '2023-06-05', 'Approved', (SELECT emp_id from employees WHERE email='alice@example.com'), 'Personal Leave'),
((SELECT emp_id from employees WHERE email='eve@example.com'), '2023-11-01 09:00:00', '2023-11-10', '2023-11-15', 'Pending', (SELECT emp_id from employees WHERE email='david@example.com'), 'Sick Leave'),
((SELECT emp_id from employees WHERE email='grace@example.com'), '2023-06-15 11:00:00', '2023-07-01', '2023-07-03', 'Rejected', (SELECT emp_id from employees WHERE email='frank@example.com'), NULL),
((SELECT emp_id from employees WHERE email='bob@example.com'), '2023-08-01 16:00:00', '2023-08-20', '2023-08-25', 'Approved', (SELECT emp_id from employees WHERE email='alice@example.com'), 'Family event'),
((SELECT emp_id from employees WHERE email='kevin@example.com'), '2023-09-01 08:00:00', '2023-09-10', '2023-09-11', 'Pending', (SELECT emp_id from employees WHERE email='alice@example.com'), ''),
((SELECT emp_id from employees WHERE email='laura@example.com'), '2024-01-10 10:00:00', '2024-02-01', '2024-02-05', 'Pending', (SELECT emp_id from employees WHERE email='david@example.com'), 'Conference'),
((SELECT emp_id from employees WHERE email='alice@example.com'), '2023-12-01 09:00:00', '2023-12-20', '2023-12-28', 'Approved', NULL, 'Holiday Season'),
((SELECT emp_id from employees WHERE email='bob@example.com'), '2024-01-10 10:00:00', '2024-01-14', '2024-01-16', 'Approved', (SELECT emp_id from employees WHERE email='alice@example.com'), 'Short break'); -- Bob on leave on Jan 15 2024
```

Listing 1: Global Dataset for Exercises

1 Date Functions (Complementary SQL)

Concepts: *Date arithmetic, OVERLAPS operator.*

(i) Meaning, values, relations, advantages of unique usage

Exercise 1.1: Project Timeline Extension and Next Month Check
Problem: For all projects, calculate their planned end date if extended by 2 months and 15 days. Also, determine if the original `planned_end_date` falls within February 2024. Display project name, original planned end date, extended planned end date, and a boolean indicating if it's planned for Feb 2024.

Exercise 1.2: Identifying Concurrent Project Assignments for Employees
Problem: Identify employees who are assigned to multiple projects whose active periods on those projects overlap. An employee's active period on a project is from `assigned_date` to `COALESCE(completion_date, 'infinity')` (for ongoing assignments). List the employee's name and the names of the two overlapping projects along with their assignment and completion dates.

(ii) Disadvantages of all its technical concepts

Exercise 1.3: OVERLAPS with identical start/end points
Problem: The `OVERLAPS` operator checks if two time periods overlap. The standard definition is `(S1, E1) OVERLAPS (S2, E2)` is true if `S1 < E2 AND S2 < E1`. What happens if `E1` is the same as `S1` (a zero-duration interval)? Consider checking if a project's start day `(start_date, start_date)` overlaps with an approved leave period. Explain the behavior and potential misinterpretation in PostgreSQL. Show a query that demonstrates this behavior and contrast it with a correct way to check if a single date point falls within a range.

Exercise 1.4: Time Zone Issues in Date Arithmetic without Explicit Time Zone Handling
Problem: If `CURRENT_TIMESTAMP` is used in date arithmetic (e.g., `CURRENT_TIMESTAMP + INTERVAL '1 day'`) in a system with users or data from different time zones, and without explicit time zone conversion (e.g., `AT TIME ZONE`), discuss the potential disadvantages or inconsistencies. Provide conceptual SQL examples to illustrate potential issues with both `TIMESTAMP WITHOUT TIME ZONE` and `TIMESTAMP WITH TIME ZONE` across DST transitions or in global applications.

(iii) Practice entirely cases where people in general does not use these approaches losing their advantages, relations and values because of the easier, basic, common or easily understandable but highly inefficient solutions

Exercise 1.5: Inefficiently Finding Projects Active During a Specific Period
Problem: Find all projects that were active (i.e., their period from `start_date` to `COALESCE(actual_end_date, planned_end_date)`) at any point during Q1 2023 (Jan 1, 2023 to Mar 31, 2023). An inefficient way involves multiple `OR` conditions checking if

project start is in Q1, end is in Q1, or project spans Q1. Show this inefficient method, then provide the efficient `OVERLAPS` solution.

2 Cases (Complementary SQL)

Concepts: `Searched CASE expressions`, `CASE in ORDER BY`, `CASE in GROUP BY`.

(i) Meaning, values, relations, advantages of unique usage

Exercise 2.1: Project Status Categorization
Problem: Categorize projects into: 'Upcoming' (starts after '2024-01-15'), 'Ongoing' (started on or before '2024-01-15' and `actual_end_date` is NULL or after '2024-01-15'), 'Completed Early/On-Time' (`actual_end_date <= planned_end_date`), 'Completed Late' (`actual_end_date > planned_end_date`). Use a searched `CASE` expression. Display project name and its status.

Exercise 2.2: Sorting Employees by Custom Priority
Problem: List all employees. Sort them by: first, managers (those who are `manager_id` for someone or have no `manager_id` themselves), then non-managers. Within managers, sort by salary descending. Within non-managers, sort by hire_date ascending. Use `CASE in ORDER BY`.

Exercise 2.3: Grouping Projects by Budget Ranges
Problem: Group projects by budget: 'Low' (`<= 50000`), 'Medium' (`50001 - 150000`), 'High' (`> 150000`), 'Undefined' (`budget IS NULL`). Count projects and sum their budgets in each category. Use `CASE in GROUP BY`.

(ii) Disadvantages of all its technical concepts

Exercise 2.4: Overly Nested CASE Expressions for Readability
Problem: Create a complex employee "profile string" using a `CASE` expression. If salary ¿ 100k, profile starts "High Earner". If also in Eng dept (ID 2), append ", Key Engineer". If also rating 5, append ", Top Performer". Otherwise, profile is "Standard". Construct this query and discuss the readability impact of such deeply nested or numerous `WHEN` conditions.

Exercise 2.5: CASE in GROUP BY Causing Performance Issues with Non-SARGable Conditions
Problem: Group employees by a category derived from their email address: 'Internal' (ends with '`@example.com`'), 'External' (otherwise). Write the query using `CASE in GROUP BY`. If the `CASE` expression uses a function like `SUBSTRING` or `LIKE '%pattern'` on the email column, and that column is not indexed appropriately for such operations, discuss potential performance disadvantages.

(iii) Practice entirely cases where people in general does not use these approaches losing their advantages, relations and values because of the easier, basic, common or easily understandable but highly inefficient solutions

Exercise 2.6: Multiple UNION ALL Queries vs. CASE in GROUP BY for Segmented Counts

Problem: Count employees in 'Engineering' (dept_id=2) and 'Sales' (dept_id=3) separately, and all other employees as 'Other'. A common but verbose way is to use `UNION ALL` with three separate `SELECT COUNT(*)` queries. Show this inefficient method, then solve efficiently using `CASE in GROUP BY`.

3 Null Space (Complementary SQL)

Concepts: `NULLIF`, `NULL handling in aggregations`, `NULL handling in sorting`.

(i) Meaning, values, relations, advantages of unique usage

Exercise 3.1: Safe Bonus Calculation Using NULLIF
Problem: Calculate a bonus as `salary * 0.10 / performance_rating`. If `performance_rating` is 0 (hypothetically, though our check is 1-5 or NULL), this would cause a division by zero. Use `NULLIF` to make `performance_rating` `NULL` if it's 0 (for our data, adapt this to use a rating of 1 as the value to nullify for this example, as '0' is not a valid rating per check constraint). Calculate `salary * 0.10 / NULLIF(performance_rating, 1)`.

Exercise 3.2: Average Billing Rate Excluding Internal/Non-Billable Roles
Problem: The `employee_projects` table has a `billing_rate` which can be `NULL` (e.g., for internal roles like 'HR Coordinator'). Calculate the average `billing_rate` across all project assignments. Show the query and discuss how `AVG()` handles the `NULL` billing rates and why this is usually correct.

Exercise 3.3: Listing Projects by Actual End Date, Undefined Last
Problem: List all projects, showing their name and `actual_end_date`. Sort them by `actual_end_date` ascending, but projects that are not yet completed (NULL `actual_end_date`) should appear last.

(ii) Disadvantages of all its technical concepts

Exercise 3.4: NULLIF with Unintended Type Coercion or Comparison Issues
Problem: Suppose `performance_rating` was a VARCHAR column and could contain 'N/A' or be `NULL`. A user tries `NULLIF(performance_rating, 0)` (comparing string to integer). Discuss potential issues like type coercion errors or unexpected behavior if the database attempts implicit conversion. Provide a small hypothetical example if necessary to illustrate.

Exercise 3.5: Aggregates over Mostly NULL Data Yielding Misleading Results
Problem: If a department has 10 employees, but only 1 has a `performance_rating` (e.g., 5), and the rest are `NULL`. `AVG(performance_rating)` would be 5. While mathematically correct by definition (ignores NULLs), this could be misleading if presented without context of how many were rated. Write a query for a department (e.g., 'Human Resources') showing average rating, total employees, and rated employees, and discuss this "disadvantage" (it's more a data interpretation issue).

(iii) Practice entirely cases where people in general does not use these approaches losing their advantages, relations and values because of the easier, basic, common or easily understandable but highly inefficient solutions

Exercise 3.6: Using `CASE WHEN expr = val THEN NULL ELSE expr END` instead of `NULLIF(expr, val)`

Problem: A user wants to convert `reason` text in `leave_requests` to `NULL` if it's an empty string `''`. They write `SELECT CASE WHEN reason = '' THEN NULL ELSE reason END ...`. Show this verbose `CASE` method for the employee 'Kevin McCallister' (who has an empty reason) and then show how `NULLIF` is more concise for this.

4 Hardcore Problem

Exercise 4.1: Comprehensive Departmental Project Health and Employee Engagement Report

Problem: Generate a report as of '2024-01-15' ('current_report_date') that assesses project health and employee engagement for each department. The report should include for each department:

- `dept_name`.
- `total_active_employees`: Count of employees in the department not terminated as of `current_report_date`.
- `avg_employee_tenure_years`: Average tenure (in years, rounded to 2 decimal places) of active employees in the department as of `current_report_date`. Tenure is `(current_report_date - hire_date) / 365.25`. If a department has no active employees, this should be `NULL`.
- `projects_info`: A string categorizing department's project involvement:
  - 'High Overlap Risk': If any two distinct active projects led by *active managers from this department* have overlapping timeframes (`start_date` to `planned_end_date`). An active project is one with no `actual_end_date` or `actual_end_date` ¿ `current_report_date` An active manager is one not terminated and is either a `manager_id` for someone or has no `manager_id` themselves (top-level).
  - 'Multiple Critical Deadlines': (If not 'High Overlap Risk') If the department has more than one active project (led by its active managers) with `planned_end_date` within 30 days from `current_report_date` (inclusive of `current_report_date`, up to `current_report_date + 30 days`).
  - 'Normal Load': Otherwise.
- `avg_rating_adjusted`: Average performance rating of active employees. If `performance_rating` is `NULL`, it's treated as '2' for this calculation. Round to 2 decimal places. If no rated employees (even after `COALESCE`), show `NULL` (or 0 if preferred for display).
- `employees_on_leave_percentage`: Percentage of active employees in the department currently on 'Approved' leave on `current_report_date`. Calculate as `(distinct_employees_ / total_active_employees) * 100.0`. Handle division by zero with `NULLIF`, resulting in `NULL`. Round to 2 decimal places.

The final report should be ordered by:

- `projects_info` ('High Overlap Risk' first, then 'Multiple Critical Deadlines', then 'Normal Load').
- Within these categories, by `dept_name` alphabetically.
- Limit the result to the top 5 departments based on this combined order.