Set Operations and Subqueries

Complementary SQL Exercises

May 15, 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 EmployeeProjects CASCADE;
DROP TABLE IF EXISTS Sales CASCADE;
DROP TABLE IF EXISTS Products CASCADE;
DROP TABLE IF EXISTS OldEmployees CASCADE;
DROP TABLE IF EXISTS CandidateEmployees CASCADE;
DROP TABLE IF EXISTS OnLeaveEmployees CASCADE;
DROP TABLE IF EXISTS Employees CASCADE;
DROP TABLE IF EXISTS Departments CASCADE;
DROP TABLE IF EXISTS Projects CASCADE;

-- Create Tables
CREATE TABLE Departments (
    departmentId INT PRIMARY KEY,
    departmentName VARCHAR(100) NOT NULL UNIQUE,
    locationCity VARCHAR(50)
);

CREATE TABLE Employees (
    employeeId INT PRIMARY KEY,
    firstName VARCHAR(50) NOT NULL,
    lastName VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    phoneNumber VARCHAR(20),
    hireDate DATE NOT NULL,
    jobId VARCHAR(20),
    salary NUMERIC(10, 2) NOT NULL CHECK (salary > 0),
    commissionPct NUMERIC(4, 2) CHECK (commissionPct >= 0 AND commissionPct <= 1),
    managerId INT REFERENCES Employees(employeeId),
    departmentId INT REFERENCES Departments(departmentId)
);

CREATE TABLE Projects (
    projectId INT PRIMARY KEY,
    projectName VARCHAR(100) NOT NULL UNIQUE,
    departmentId INT REFERENCES Departments(departmentId),
    startDate DATE,
    endDate DATE,
    budget NUMERIC(12, 2) CHECK (budget >= 0)
);

CREATE TABLE EmployeeProjects (
    employeeId INT REFERENCES Employees(employeeId),
    projectId INT REFERENCES Projects(projectId),
    assignedRole VARCHAR(50),
    hoursWorked INT CHECK (hoursWorked >= 0),
    PRIMARY KEY (employeeId, projectId)
);

CREATE TABLE OldEmployees (
    employeeId INT PRIMARY KEY,
    firstName VARCHAR(50),
    lastName VARCHAR(50),
    lastDepartmentId INT,
    terminationDate DATE NOT NULL,
    finalSalary NUMERIC(10, 2),
    reasonForLeaving VARCHAR(255)
);

CREATE TABLE CandidateEmployees (
    candidateId INT PRIMARY KEY,
    firstName VARCHAR(50),
    lastName VARCHAR(50),
    appliedPosition VARCHAR(100),
    expectedSalary NUMERIC(10, 2),
    applicationDate DATE
);
```

```sql

CREATE TABLE OnLeaveEmployees (
    employeeId INT PRIMARY KEY REFERENCES Employees(employeeId),
    leaveStartDate DATE,
    leaveEndDate DATE,
    leaveReason VARCHAR(100)
);

CREATE TABLE Products (
    productId INT PRIMARY KEY,
    productName VARCHAR(100) NOT NULL,
    productCategory VARCHAR(50),
    unitPrice NUMERIC(10, 2) CHECK (unitPrice > 0)
);

CREATE TABLE Sales (
    saleId INT PRIMARY KEY,
    employeeId INT REFERENCES Employees(employeeId),
    productId INT REFERENCES Products(productId),
    saleDate TIMESTAMP NOT NULL,
    quantitySold INT CHECK (quantitySold > 0),
    saleAmount NUMERIC(10, 2)
);

-- Populate Tables
INSERT INTO Departments (departmentId, departmentName, locationCity) VALUES
(1, 'Human Resources', 'New York'),
(2, 'Engineering', 'San Francisco'),
(3, 'Sales', 'Chicago'),
(4, 'Marketing', 'New York'),
(5, 'Finance', 'London'),
(6, 'Research', 'San Francisco'),
(7, 'Customer Support', 'Austin');

INSERT INTO Employees (employeeId, firstName, lastName, email, phoneNumber, hireDate,
    jobId, salary, commissionPct, managerId, departmentId) VALUES
(101, 'John', 'Smith', 'john.smith@example.com', '555-1234', '2018-06-01', 'HR_REP',
    60000, NULL, NULL, 1),
(102, 'Alice', 'Johnson', 'alice.j@example.com', '555-5678', '2019-03-15', 'ENG_LEAD',
    90000, NULL, NULL, 2),
(103, 'Bob', 'Williams', 'bob.w@example.com', '555-8765', '2019-07-20', 'SALES_MGR',
    75000, 0.10, NULL, 3),
(104, 'Eva', 'Brown', 'eva.b@example.com', '555-4321', '2020-01-10', 'MKT_SPEC', 65000,
    NULL, 101, 4),
(105, 'Charlie', 'Davis', 'charlie.d@example.com', '555-1122', '2018-11-05', 'ENG_SR',
    85000, NULL, 102, 2),
(106, 'Diana', 'Miller', 'diana.m@example.com', '555-6543', '2021-05-25', 'SALES_REP',
    55000, 0.05, 103, 3),
(107, 'Frank', 'Wilson', 'frank.w@example.com', '555-7890', '2022-08-01', 'ENG_JR',
    70000, NULL, 102, 2),
(108, 'Grace', 'Moore', 'grace.m@example.com', '555-2109', '2019-09-01', 'FIN_ANALYST',
    72000, NULL, NULL, 5),
(109, 'Henry', 'Taylor', 'henry.t@example.com', '555-1098', '2023-02-15', 'ENG_JR',
    68000, NULL, 105, 2),
(110, 'Ivy', 'Anderson', 'ivy.a@example.com', '555-8076', '2020-11-30', 'MKT_MGR',
    80000, NULL, 101, 4),
(111, 'Jack', 'Thomas', 'jack.t@example.com', '555-7654', '2017-07-14', 'RES_SCI',
    95000, NULL, NULL, 6),
(112, 'Karen', 'Jackson', 'karen.j@example.com', '555-6547', '2021-10-01', 'HR_ASSIST',
    50000, NULL, 101, 1),
(113, 'Leo', 'White', 'leo.w@example.com', '555-5438', '2023-05-20', 'SALES_REP', 58000,
     0.06, 103, 3),
(114, 'Mia', 'Harris', 'mia.h@example.com', '555-4329', '2019-02-18', 'FIN_MGR', 92000,
    NULL, 108, 5),
(115, 'Noah', 'Martin', 'noah.m@example.com', '555-3210', '2022-06-10', 'RES_ASSIST',
    60000, NULL, 111, 6),
(116, 'Olivia', 'Garcia', 'olivia.g@example.com', '555-1987', '2018-09-01', 'ENG_SR',
    88000, NULL, 102, 2),
(117, 'Paul', 'Martinez', 'paul.m@example.com', '555-8760', '2023-01-05', 'SALES_INTERN'
    , 40000, 0.02, 106, 3),
(118, 'Quinn', 'Robinson', 'quinn.r@example.com', '555-7651', '2020-07-07', 'MKT_INTERN'
    , 42000, NULL, 104, 4),
```

```
(119, 'Ruby', 'Clark', 'ruby.c@example.com', '555-6542', '2022-03-03', 'HR_SPEC', 62000, NULL, 101, 1),
(120, 'Sam', 'Rodriguez', 'sam.r@example.com', '555-5433', '2021-11-11', 'ENG_TECH', 72000, NULL, 105, 2),
(121, 'Tom', 'Lee', 'tom.lee@example.com', '555-1122', '2023-08-15', 'FIN_ANALYST', 73000, NULL, 114, 5),
(122, 'Ursula', 'Walker', 'ursula.w@example.com', '555-2233', '2019-01-20', 'RES_HEAD', 120000, NULL, NULL, 6),
(123, 'Victor', 'Hall', 'victor.h@example.com', '555-3344', '2020-05-10', 'ENG_LEAD', 95000, NULL, NULL, 2),
(124, 'Wendy', 'Allen', 'wendy.a@example.com', '555-4455', '2021-09-01', 'MKT_COORD', 63000, NULL, 110, 4),
(125, 'Xavier', 'Young', 'xavier.y@example.com', '555-5566', '2022-12-12', 'SALES_LEAD', 78000, 0.08, 103, 3),
(126, 'Yara', 'King', 'yara.k@example.com', '555-6677', '2018-04-04', 'HR_MGR', 85000, NULL, NULL, 1),
(127, 'Zack', 'Wright', 'zack.w@example.com', '555-7788', '2023-07-01', 'ENG_INTERN', 45000, NULL, 107, 2),
(128, 'Laura', 'Palmer', 'laura.p@example.com', '555-1111', '2023-01-15', 'SUPPORT_REP', 52000, NULL, NULL, 7),
(129, 'Dale', 'Cooper', 'dale.c@example.com', '555-2222', '2023-02-20', 'SUPPORT_LEAD', 65000, NULL, NULL, 7);

UPDATE Employees SET managerId = 102 WHERE employeeId IN (105, 107, 116, 120);
UPDATE Employees SET managerId = 123 WHERE employeeId IN (109, 127);
UPDATE Employees SET managerId = 103 WHERE employeeId IN (106, 113, 117, 125);
UPDATE Employees SET managerId = 110 WHERE employeeId IN (104, 118, 124);
UPDATE Employees SET managerId = 101 WHERE employeeId IN (112, 119);
UPDATE Employees SET managerId = 126 WHERE employeeId = 101;
UPDATE Employees SET managerId = 111 WHERE employeeId = 115;
UPDATE Employees SET managerId = 122 WHERE employeeId = 111;
UPDATE Employees SET managerId = 114 WHERE employeeId IN (108, 121);
UPDATE Employees SET managerId = 129 WHERE employeeId = 128;

INSERT INTO Projects (projectId, projectName, departmentId, startDate, endDate, budget) VALUES
(1, 'Alpha Launch', 4, '2023-01-01', '2023-06-30', 150000.00),
(2, 'Beta Platform', 2, '2022-09-01', '2024-03-31', 500000.00),
(3, 'Gamma Sales Drive', 3, '2023-03-01', '2023-09-30', 80000.00),
(4, 'Delta HR System', 1, '2023-02-01', '2023-12-31', 120000.00),
(5, 'Epsilon Research', 6, '2022-05-01', '2024-05-30', 300000.00),
(6, 'Zeta Finance Tool', 5, '2023-07-01', '2024-06-30', 200000.00),
(7, 'Omega Security Update', 2, '2023-10-01', '2024-01-31', 250000.00),
(8, 'Sigma Marketing Campaign', 4, '2024-01-15', '2024-07-15', 180000.00),
(9, 'Kappa Efficiency Audit', 5, '2022-11-01', '2023-04-30', 75000.00),
(10, 'New Support Portal', 7, '2023-05-01', '2023-11-30', 90000.00);

INSERT INTO EmployeeProjects (employeeId, projectId, assignedRole, hoursWorked) VALUES
(102, 2, 'Project Lead', 500), (105, 2, 'Senior Developer', 600), (107, 2, 'Junior Developer', 450), (116, 2, 'Senior Developer', 550), (120, 2, 'Technician', 400), (123, 7, 'Project Lead', 200),
(104, 1, 'Marketing Specialist', 300), (110, 1, 'Campaign Manager', 250), (124, 1, 'Coordinator', 320),
(106, 3, 'Sales Representative', 400), (113, 3, 'Sales Representative', 380), (125, 3, 'Lead Sales', 350),
(101, 4, 'HR Lead', 200), (112, 4, 'HR Assistant', 300), (119, 4, 'HR Specialist', 280),
(111, 5, 'Lead Scientist', 700), (115, 5, 'Research Assistant', 650), (122, 5, 'Principal Investigator', 500),
(108, 6, 'Financial Analyst', 300), (114, 6, 'Finance Lead', 250), (121, 6, 'Analyst', 320),
(105, 7, 'Security Consultant', 150), (107, 7, 'Developer', 180),
(108, 9, 'Auditor', 200), (114, 9, 'Audit Lead', 150),
(128, 10, 'Support Analyst', 350), (129, 10, 'Project Manager', 280);

INSERT INTO OldEmployees (employeeId, firstName, lastName, lastDepartmentId, terminationDate, finalSalary, reasonForLeaving) VALUES
(201, 'Gary', 'Oldman', 2, '2022-12-31', 82000, 'Retired'),
(202, 'Helen', 'Hunt', 3, '2023-03-15', 70000, 'New Opportunity'),
(203, 'Mike', 'Myers', 2, '2021-08-20', 90000, 'Relocation'),
(204, 'Olivia', 'Garcia', 2, '2023-11-01', 88000, 'New Opportunity');

INSERT INTO CandidateEmployees (candidateId, firstName, lastName, appliedPosition, expectedSalary, applicationDate) VALUES
```

```
(301, 'Peter', 'Pan', 'ENG_JR', 65000, '2023-10-01'),
(302, 'Wendy', 'Darling', 'MKT_SPEC', 68000, '2023-09-15'),
(303, 'John', 'Smith', 'HR_REP', 60000, '2023-11-01'),
(304, 'Alice', 'Wonder', 'ENG_LEAD', 92000, '2023-08-20'),
(305, 'Bruce', 'Wayne', 'FIN_ANALYST', 75000, '2023-11-05');

INSERT INTO OnLeaveEmployees (employeeId, leaveStartDate, leaveEndDate, leaveReason)
    VALUES
(104, '2023-11-01', '2024-02-01', 'Maternity Leave'),
(111, '2023-09-15', '2023-12-15', 'Sabbatical');

INSERT INTO Products (productId, productName, productCategory, unitPrice) VALUES
(1, 'AlphaWidget', 'Electronics', 49.99),
(2, 'BetaGear', 'Software', 199.00),
(3, 'GammaCore', 'Hardware', 120.50),
(4, 'DeltaService', 'Services', 75.00),
(5, 'EpsilonPlus', 'Electronics', 89.90);

INSERT INTO Sales (saleId, employeeId, productId, saleDate, quantitySold, saleAmount)
    VALUES
(1, 106, 1, '2023-04-10 10:30:00', 2, 99.98),
(2, 106, 3, '2023-04-12 14:00:00', 1, 120.50),
(3, 113, 2, '2023-05-05 11:15:00', 1, 199.00),
(4, 106, 1, '2023-05-20 16:45:00', 3, 149.97),
(5, 117, 4, '2023-06-01 09:00:00', 10, 750.00),
(6, 125, 5, '2023-07-10 12:30:00', 5, 449.50),
(7, 113, 1, '2023-07-15 15:00:00', 2, 99.98),
(8, 103, 2, '2023-08-01 10:00:00', 2, 398.00),
(9, 106, 3, '2023-08-18 13:20:00', 1, 120.50),
(10, 125, 2, '2023-09-05 17:00:00', 1, 199.00),
(11, 113, 5, '2023-11-10 09:30:00', 3, 269.70),
(12, 106, 4, '2023-11-15 11:45:00', 5, 375.00),
(13, 117, 1, '2023-11-20 14:15:00', 1, 49.99),
(14, 125, 3, '2023-12-01 10:00:00', 2, 241.00),
(15, 103, 5, '2023-12-05 16:30:00', 4, 359.60),
(16, 128, 2, '2023-06-15 10:00:00', 1, 199.00),
(17, 105, 3, '2023-07-20 11:00:00', 1, 120.50);
```

Listing 1: Global Dataset for Exercises

1 Set Operations

1.1 Type (i): Meaning, values, relations, advantages

SO-1.1: Unified List of All Current and Potential Engineering Staff
Problem: The company wants a single, unique list of full names (firstName lastName) of all current employees in the 'Engineering' department and all candidates who applied for an 'ENG_LEAD' or 'ENG_JR' position. The list should be ordered by full name. This demonstrates `UNION` for combining different sources and ensuring uniqueness.

SO-1.2: Log of All Employee-Project Assignments and Recent Terminations
Problem: Create a comprehensive log showing employee IDs. Include all current employee-project assignments (employeeId and projectId as 'activityIdentifier') and employee IDs of those who left the company in 2023 (employeeId and terminationDate as 'activityIdentifier'). Include duplicates if an employee is on multiple projects. This demonstrates `UNION ALL` where duplicates are meaningful.

SO-1.3: Employees in Sales and Marketing Departments
Problem: Identify employees (employeeId, firstName, lastName) who are part of *both* the 'Sales' department team for project 'Gamma Sales Drive' *and* also have skills recognized by the 'Marketing' department (assume this means they are in the marketing department currently). This demonstrates `INTERSECT` for finding commonalities.

SO-1.4: Active Engineers Not Assigned to 'Omega Security Update' Project
Problem: List the employeeId, firstName, and lastName of all current employees in the 'Engineering' department who are *not* assigned to the 'Omega Security Update' project. This demonstrates `EXCEPT` for finding differences.

1.2 Type (ii): Disadvantages

SO-2.1: Mismatched Column Data Types in Union
Problem: Try to create a unified list showing employeeId and their salary, and candidateId and their expectedSalary. Intentionally try to select `hireDate` for employees instead of salary to demonstrate a data type mismatch error that `UNION` would typically cause if not handled. Then, show the corrected version by casting. This highlights the column compatibility disadvantage.

SO-2.2: Performance of UNION vs UNION ALL with Large Datasets (Conceptual)
Problem: Explain a scenario where using `UNION` instead of `UNION ALL` could significantly degrade performance. Retrieve all first names from `Employees` and `CandidateEmployees`. First with `UNION ALL`, then with `UNION`. Imagine these tables have millions of rows.

1.3 Type (iii): Inefficient alternatives

SO-3.1: Simulating EXCEPT with NOT IN (and its NULL pitfall)
Problem: Find all employees from the 'Engineering' department (employeeId) who are not listed in the `OnLeaveEmployees` table. First, attempt this using `NOT IN` with a subquery, and then show the more robust `EXCEPT` (or `NOT EXISTS`) solution, highlighting the NULL issue with `NOT IN` if `OnLeaveEmployees.employeeId` could be NULL.

SO-3.2: Simulating INTERSECT with Multiple Joins/WHERE conditions
Problem: Find employees (employeeId, firstName) who are in the 'Engineering' department AND are working on the 'Beta Platform' project. Show how this can be done with `INTERSECT` and then with a more traditional JOIN approach. Discuss when `INTERSECT` might be clearer.

1.4 Type (iv): Hardcore problem

SO-4.1: Consolidated List of High-Value Personnel Not On Leave, Ranked
Problem: Create a consolidated list of personnel who are either:

a. Current employees with a salary greater than $70,000.

b. Former employees (from `OldEmployees`) whose final salary was greater than $75,000 and left for 'New Opportunity' or 'Retired'.

Exclude any personnel from this consolidated list if their `employeeId` appears in the `OnLeaveEmployees` table. For the resulting list, display `employeeId`, `firstName`, `lastName`, their relevant `salary` (current or final), a `personnelType` ('Current Employee' or 'Former Employee'), and rank them within their `personnelType` based on their salary in descending order.

2 Subqueries

2.1 Type (i): Meaning, values, relations, advantages

SQ-1.1: Employees Earning More Than Average (Scalar Subquery)
Problem: List the `employeeId`, `firstName`, `lastName`, and `salary` of all employees who earn more than the average salary of all employees. This demonstrates a scalar subquery in the `WHERE` clause.

SQ-1.2: Departments with Higher Than Average Project Budgets (Subquery in FROM)
Problem: List department names and their average project budget, but only for departments where their average project budget is greater than the overall average budget of all projects. This demonstrates a subquery in the `FROM` clause (derived table).

SQ-1.3: Employee's Project Count (Subquery in SELECT - Correlated)
Problem: For each employee, display their `employeeId`, `firstName`, `lastName`, and the total number of projects they are currently assigned to. This demonstrates a correlated subquery in the `SELECT` clause.

SQ-1.4: Employees in Departments Located in 'New York' (Subquery in WHERE with IN)
Problem: List the `employeeId`, `firstName`, and `lastName` of all employees who work in departments located in 'New York'. Use a subquery with `IN`.

SQ-1.5: Departments with at Least One Project (Subquery in WHERE with EXISTS)
Problem: List the `departmentId` and `departmentName` of all departments that have at least one project associated with them. Use a subquery with `EXISTS`.

SQ-1.6: Employees Earning More Than Any Sales Intern (Subquery in WHERE with ANY/SOME)
Problem: Find all employees whose salary is greater than *any* salary of an employee with `jobId` = 'SALES_INTERN'.

SQ-1.7: Employees Earning More Than All Sales Interns (Subquery in WHERE with ALL)
Problem: Find all employees whose salary is greater than *all* salaries of employees with `jobId` = 'SALES_INTERN'.

2.2 Type (ii): Disadvantages

SQ-2.1: Performance of Correlated Subquery in SELECT
Problem: Retrieve each employee's `employeeId`, `firstName`, and the name of the most expensive project their department is leading. Highlight that a correlated subquery in `SELECT` might be re-executed for each employee, potentially leading to performance issues. (An alternative join-based approach might be more efficient).

SQ-2.2: Scalar Subquery Returning Multiple Rows (Error Scenario)
Problem: Attempt to find employees whose salary is equal to a salary from the 'Sales' department. Intentionally write a scalar subquery that *could* return multiple rows to demonstrate the error. Then show a corrected version using `IN` or `ANY`.

2.3 Type (iii): Inefficient alternatives

SQ-3.1: Finding Max Salary Without Scalar Subquery (Inefficient Application Logic)
Problem: List employees who earn the maximum salary in the company. An inefficient alternative would be to first query the max salary, then use that value in a second query. Show the efficient SQL way using a scalar subquery.

SQ-3.2: Checking Existence of Sales by Engineers (Inefficient: Fetching All Sales Data)
Problem: Determine if any employee from the 'Engineering' department has ever made a sale. An inefficient approach would be to fetch all sales records and then join/filter in application code or with complex client-side logic. Show the efficient `EXISTS` subquery approach.

2.4 Type (iv): Hardcore problem

SQ-4.1: Departmental High Earners Analysis
Problem: Identify departments where the average salary of its employees is greater than the overall average salary of all employees hired since '2020-01-01'. For these identified departments:

1. List the department name.
2. List the top 2 highest-paid employees (firstName, lastName, salary) within each of these departments.
3. For these top employees, show the difference between their salary and their department's average salary.
4. Also, display the department's total budget from all its projects.

Order results by department name, then by employee salary descending.