Common Table Expressions - CTEs

Advanced Query Techniques: Exercises

May 18, 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
-- Dataset for Category (i)
CREATE TABLE DepartmentsI (
    departmentId INTEGER PRIMARY KEY,
    departmentName VARCHAR(100) NOT NULL,
    locationCity VARCHAR(50)
);

CREATE TABLE EmployeesI (
    employeeId INTEGER PRIMARY KEY,
    employeeName VARCHAR(100) NOT NULL,
    departmentId INTEGER REFERENCES DepartmentsI(departmentId),
    managerId INTEGER REFERENCES EmployeesI(employeeId), -- Self-reference for hierarchy
    salary DECIMAL(10, 2) NOT NULL,
    hireDate DATE NOT NULL
);

INSERT INTO DepartmentsI (departmentId, departmentName, locationCity) VALUES
(1, 'Technology', 'New York'),
(2, 'Human Resources', 'London'),
(3, 'Sales', 'Tokyo'),
(4, 'Marketing', 'Paris');

INSERT INTO EmployeesI (employeeId, employeeName, departmentId, managerId, salary, hireDate) VALUES
(101, 'Alice Wonderland', 1, NULL, 120000.00, '2018-03-15'),
(102, 'Bob The Builder', 1, 101, 90000.00, '2019-07-01'),
(103, 'Charlie Brown', 1, 102, 80000.00, '2020-01-10'),
(104, 'Diana Prince', 2, NULL, 110000.00, '2017-05-20'),
(105, 'Eve Harrington', 2, 104, 75000.00, '2021-02-28'),
(106, 'Frankenstein Monster', 3, NULL, 130000.00, '2018-11-01'),
(107, 'Grace OMalley', 3, 106, 85000.00, '2019-05-15'),
(108, 'Henry Jekyll', 3, 106, 82000.00, '2022-08-20'),
(109, 'Ivy Pepper', 1, 101, 95000.00, '2020-06-01'),
(110, 'John Doe', NULL, 101, 60000.00, '2023-01-15');

-- Dataset for Category (ii)
CREATE TABLE ProductCategoriesII (
    categoryId SERIAL PRIMARY KEY,
    categoryName VARCHAR(50) UNIQUE NOT NULL
);

CREATE TABLE ProductsII (
    productId SERIAL PRIMARY KEY,
    productName VARCHAR(100) NOT NULL,
    categoryId INTEGER REFERENCES ProductCategoriesII(categoryId),
    basePrice DECIMAL(10,2)
);

CREATE TABLE SalesTransactionsII (
    transactionId SERIAL PRIMARY KEY,
    productId INTEGER REFERENCES ProductsII(productId),
    saleDate TIMESTAMP NOT NULL,
    quantitySold INTEGER NOT NULL,
    discount DECIMAL(3,2) DEFAULT 0.00
);

INSERT INTO ProductCategoriesII (categoryName) VALUES ('Electronics'), ('Books'), ('Home Goods');
INSERT INTO ProductsII (productName, categoryId, basePrice) VALUES
('Laptop Pro', 1, 1200.00), ('Quantum Physics Primer', 2, 25.00), ('Smart LED Bulb', 3, 15.00),
('Desktop Gamer', 1, 1800.00), ('History of Time', 2, 20.00), ('Robotic Vacuum', 3, 300.00);

DO $$
DECLARE
    i INT;
    pId INT;
    sDate TIMESTAMP;
    qty INT;
BEGIN
    FOR i IN 1..10000 LOOP
        pId := (MOD(i, 6)) + 1;
        sDate := CURRENT_TIMESTAMP - (MOD(i,365) || ’ days’)::INTERVAL - (MOD(i,24) || ’ hours’)::INTERVAL;
        qty := (MOD(i, 5)) + 1;
        INSERT INTO SalesTransactionsII (productId, saleDate, quantitySold, discount)
        VALUES (pId, sDate, qty, CASE WHEN MOD(i,10) = 0 THEN 0.05 ELSE 0.00 END);
    END LOOP;
END $$;

UPDATE SalesTransactionsII
SET saleDate = CURRENT_DATE - INTERVAL ’1 month’ + (MOD(transactionId, 30) || ’ days’)::INTERVAL
WHERE MOD(productId, 2) = 0; -- Update some products to have recent sales

-- Dataset for Category (iii)
CREATE TABLE CustomersIII (
    customerId SERIAL PRIMARY KEY,
    customerName VARCHAR(100) NOT NULL,
    registrationDate DATE,
    city VARCHAR(50)
);

CREATE TABLE ProductsMasterIII (
    productId SERIAL PRIMARY KEY,
    productName VARCHAR(100),
    category VARCHAR(50)
);

CREATE TABLE OrdersIII (
    orderId SERIAL PRIMARY KEY,
    customerId INTEGER REFERENCES CustomersIII(customerId),
    orderDate DATE,
    shipmentRegion VARCHAR(50)
);

CREATE TABLE OrderItemsIII (
    orderItemId SERIAL PRIMARY KEY,
    orderId INTEGER REFERENCES OrdersIII(orderId),
    productId INTEGER REFERENCES ProductsMasterIII(productId),
    quantity INTEGER,
    pricePerUnit DECIMAL(10,2)
);

INSERT INTO CustomersIII (customerName, registrationDate, city) VALUES
(’Global Corp’, ’2020-01-15’, ’New York’), (’Local Biz’, ’2021-06-01’, ’London’),
(’Alpha Inc’, ’2019-11-20’, ’Tokyo’), (’Beta LLC’, ’2022-03-10’, ’New York’);

INSERT INTO ProductsMasterIII (productName, category) VALUES
(’Widget A’, ’Gadgets’), (’Widget B’, ’Gizmos’), (’Service C’, ’Services’), (’Tool D’, ’Tools’);

INSERT INTO OrdersIII (customerId, orderDate, shipmentRegion) VALUES
(1, ’2022-02-10’, ’North America’), (2, ’2022-03-15’, ’Europe’),
(1, ’2023-04-20’, ’North America’), (3, ’2023-05-05’, ’Asia’),
(2, ’2023-06-10’, ’Europe’), (4, ’2022-07-01’, ’North America’);

INSERT INTO OrderItemsIII (orderId, productId, quantity, pricePerUnit) VALUES
(1, 1, 10, 50.00), (1, 2, 5, 100.00), (2, 3, 1, 200.00),
(3, 1, 20, 45.00), (3, 4, 2, 150.00), (4, 2, 8, 95.00),
(5, 3, 2, 190.00), (6, 4, 3, 140.00);

-- Dataset for Category (iv)
CREATE TABLE DepartmentsIV (
    departmentId SERIAL PRIMARY KEY,
    departmentName VARCHAR(100) NOT NULL,
    headEmployeeId INTEGER -- Nullable, to be cross-referenced with EmployeesIV
);

```

```sql
CREATE TABLE EmployeesIV (
    employeeId SERIAL PRIMARY KEY,
    employeeName VARCHAR(100) NOT NULL,
    departmentId INTEGER REFERENCES DepartmentsIV(departmentId),
    managerId INTEGER REFERENCES EmployeesIV(employeeId), -- For hierarchy
    salary DECIMAL(10, 2) NOT NULL,
    hireDate DATE NOT NULL
);

ALTER TABLE DepartmentsIV ADD CONSTRAINT fkHeadEmployee FOREIGN KEY (headEmployeeId) REFERENCES EmployeesIV(employeeId) DEFERRABLE INITIALLY DEFERRED;

INSERT INTO DepartmentsIV (departmentId, departmentName) VALUES
(1, 'Engineering'), (2, 'Product Management'), (3, 'Research & Development'), (4, 'Operations');

INSERT INTO EmployeesIV (employeeId, employeeName, departmentId, managerId, salary, hireDate) VALUES
(1, 'Ava CEO', 1, NULL, 250000, '2015-01-01'),
(2, 'Brian Lead', 1, 1, 150000, '2018-06-01'),
(3, 'Chloe SeniorDev', 1, 2, 110000, '2020-03-15'),
(4, 'David JuniorDev', 1, 3, 75000, '2022-07-01'),
(5, 'Eli PMHead', 2, 1, 160000, '2017-09-01'),
(6, 'Fiona SeniorPM', 2, 5, 120000, '2020-11-01'),
(7, 'George PM', 2, 6, 85000, '2021-05-10'),
(8, 'Hannah RDHead', 3, 1, 170000, '2016-04-12'),
(9, 'Ian SeniorScientist', 3, 8, 130000, '2021-01-20'),
(10, 'Julia Scientist', 3, 9, 90000, '2022-08-01'),
(11, 'Kevin OpsLead', 4, 1, 140000, '2019-02-10'),
(12, 'Liam OpsSpecialist', 4, 11, 95000, '2021-10-05'),
(13, 'Mike AnotherDev', 1, 2, 105000, '2021-02-01');

UPDATE DepartmentsIV SET headEmployeeId = 2 WHERE departmentName = 'Engineering';
UPDATE DepartmentsIV SET headEmployeeId = 5 WHERE departmentName = 'Product Management';
UPDATE DepartmentsIV SET headEmployeeId = 8 WHERE departmentName = 'Research & Development';
UPDATE DepartmentsIV SET headEmployeeId = 11 WHERE departmentName = 'Operations';

CREATE TABLE ProjectsIV (
    projectId SERIAL PRIMARY KEY,
    projectName VARCHAR(150) NOT NULL,
    startDate DATE,
    endDate DATE,
    budget DECIMAL(12, 2)
);

CREATE TABLE TasksIV (
    taskId SERIAL PRIMARY KEY,
    projectId INTEGER REFERENCES ProjectsIV(projectId),
    taskName VARCHAR(200),
    assignedToEmployeeId INTEGER REFERENCES EmployeesIV(employeeId),
    estimatedHours INTEGER,
    actualHours INTEGER,
    status VARCHAR(20)
);

CREATE TABLE TimeLogsIV (
    logId SERIAL PRIMARY KEY,
    taskId INTEGER REFERENCES TasksIV(taskId),
    employeeId INTEGER REFERENCES EmployeesIV(employeeId),
    logDate DATE NOT NULL,
    hoursWorked DECIMAL(5,2) NOT NULL,
    notes TEXT
);

INSERT INTO ProjectsIV (projectName, startDate, endDate, budget) VALUES
('Alpha Core System', '2022-01-01', '2023-12-31', 200000.00),
('Beta Mobile App', '2023-03-01', '2024-02-28', 80000.00),
('Gamma Research Initiative', '2021-06-15', '2023-05-30', 160000.00),
('Delta Operations Upgrade', '2023-07-01', NULL, 120000.00);

INSERT INTO TasksIV (projectId, taskName, assignedToEmployeeId, estimatedHours, actualHours, status) VALUES
```

```sql
(1, 'Design Alpha Architecture', 3, 100, 90, 'Completed'),
(1, 'Develop Alpha Module 1', 3, 150, 160, 'In Progress'),
(2, 'Beta UI/UX Design', 6, 80, 70, 'Completed'),
(2, 'Beta Backend Dev', 7, 120, 50, 'In Progress'),
(3, 'Gamma Initial Research', 9, 200, 180, 'Completed'),
(3, 'Gamma Experiment Setup', 9, 100, 110, 'Overdue'),
(4, 'Delta Process Analysis', 12, 60, 40, 'In Progress'),
(1, 'Alpha Documentation', 13, 80, 0, 'Pending');
-- The next task inserted will have taskId = 9 (due to SERIAL on previous 8 inserts)
INSERT INTO TasksIV (projectId, taskName, assignedToEmployeeId, estimatedHours, actualHours, status) VALUES
(2, 'Cross-project review for Alpha', 3, 20, 0, 'Pending');

INSERT INTO TimeLogsIV (logId, taskId, employeeId, logDate, hoursWorked, notes) VALUES
(DEFAULT, 1, 3, '2022-03-01', 8.0, 'Initial design'), (DEFAULT, 1, 3, '2022-03-02', 8.0, 'Refinement'),
(DEFAULT, 2, 3, '2022-04-01', 8.0, 'Dev start'), (DEFAULT, 2, 3, '2022-04-02', 8.0, 'Core logic'),
(DEFAULT, 3, 6, '2023-03-10', 7.0, 'UX flows'),
(DEFAULT, 5, 9, '2021-07-01', 6.0, 'Literature review'), (DEFAULT, 5, 9, '2021-07-02', 8.0, 'Planning'),
(DEFAULT, 6, 9, '2021-09-01', 8.0, 'Setup phase 1'), (DEFAULT, 6, 9, '2021-09-02', 5.0, 'Troubleshooting setup'),
(DEFAULT, 7, 12, '2023-07-15', 8.0, 'Mapping current state'),
(DEFAULT, 8, 13, '2022-05-01', 4.0, 'Doc outline');
-- logId values will be 1 to 11 after these inserts
INSERT INTO TimeLogsIV (logId, taskId, employeeId, logDate, hoursWorked, notes) VALUES
(DEFAULT, 2, 3, '2022-04-03', 8.0, 'Task 2 for emp 3'), -- emp 3 (Chloe) on task 2 (project 1), logId 12
(DEFAULT, 4, 7, '2023-08-01', 5.0, 'Task 4 for emp 7'), -- emp 7 (George) on task 4 (project 2), logId 13
(DEFAULT, 9, 3, '2023-09-01', 3.0, 'Time for task 9, project 2'); -- emp 3 works on task 9 (project 2), logId 14
```

Listing 1: Global Dataset for Exercises

1 Category (i): Practice Meanings, Values, Relations, and Advantages

These exercises focus on understanding the fundamental meanings, values, and relational aspects of Common Table Expressions (CTEs). They demonstrate unique uses and advantages, building upon concepts from Basic and Intermediate SQL.

1. **Exercise 1: Basic CTE for Readability**
   Problem: List all employees in the 'Technology' department who earn more than $90,000. Show how a CTE can simplify selecting the department first.

2. **Exercise 2: CTE Referenced Multiple Times**
   Problem: Find all employees whose salary is above the average salary of their respective department. Also, show the department's average salary. This requires calculating departmental average salary and then using it for comparison.

3. **Exercise 3: Nested CTEs**
   Problem: List employees from 'New York' or 'London' who were hired after 2019. First, create a CTE for relevant departments. Then, a CTE for employees in those departments hired after 2019.

4. **Exercise 4: Recursive CTE for Hierarchical Data**
   Problem: Display the organizational hierarchy for 'Charlie Brown' (employeeId 103), showing his reporting line up to the top manager. List employee ID, name, manager ID, and level in hierarchy.

2 Category (ii): Practice Disadvantages

These exercises explore potential disadvantages or limitations associated with CTEs, such as performance considerations and scope.

1. **Exercise 1: Potential Performance Issue (Optimization Fence / Materialization)**
   Problem: Calculate the total revenue (price * quantity * (1-discount)) for each product using tables from dataset part II (‘ProductsII‘, ‘SalesTransactionsII‘, ‘ProductCategoriesII‘). Then, retrieve this information ONLY for products in the ’Electronics’ category. A CTE might calculate revenue for ALL products first, then filter. (This exercise highlights a *potential* disadvantage; actual performance depends on the DBMS optimizer).

2. **Exercise 2: No Indexing on CTE Results**
   Problem: Using tables from dataset part II, identify products that had sales in the month immediately preceding the current month (e.g., if today is Feb 15, 2024, identify sales in Jan 2024). Simulate multiple conceptual uses of this intermediate result: first list the product names, then provide a count of these distinct products. The disadvantage illustrated is that if the CTE result was large and queried multiple times, it’s re-evaluated or its unindexed materialized result is scanned.

3. **Exercise 3: CTE Scope Limitation**
   Problem: You need to calculate total sales revenue for the ’Books’ category (using dataset part II tables) and use this total in two *separate subsequent independent queries* (e.g., one to show the total, another to show 10% of this total). Show conceptually or by attempting that a CTE defined in one query is not available in the next, illustrating its scope. Then, demonstrate how you would achieve this by re-declaring the CTE if needed.

3 Category (iii): Practice Cases Avoiding Inefficient Basic Solutions

These exercises demonstrate scenarios where CTEs offer significant advantages in readability, maintainability, and sometimes performance over more basic or convoluted approaches that don't leverage CTEs effectively. Use tables from dataset part III ('CustomersIII', 'ProductsMasterIII', 'OrdersIII', 'OrderItemsIII').

1. **Exercise 1: Replacing Repeated Subqueries**
   Problem: Find customers who placed orders in both 2022 and 2023. List their names and city. Illustrate how CTEs can avoid repeating subquery logic that might scan 'OrdersIII' multiple times.

2. **Exercise 2: Simplifying Complex Joins and Filters**
   Problem: List products (name and category) that were part of orders shipped to 'North America' and had a total order value (sum of quantity * pricePerUnit for all items in that order) greater than $600. Show how CTEs can break down this logic compared to a single, very long query.

3. **Exercise 3: Avoiding Temporary Tables for Single-Query Scope**
   Problem: Calculate the average total order value for each 'shipmentRegion'. Then, list regions whose average order value is greater than the overall average order value across all regions. Demonstrate how CTEs provide a cleaner, single-statement solution compared to potentially using temporary tables.

4. **Exercise 4: Step-by-Step Multi-Level Aggregations (Revised)**
   Problem: For each product category, find the month (e.g., '2023-04') with the highest total sales quantity for that category. Display category, the best month, and total quantity for that month. Solve this using CTEs for structured aggregation, without using window functions.

4 Category (iv): Hardcore Combined Problem

This problem requires combining various SQL concepts learned prior to and including Common Table Expressions, focusing on their application in a complex scenario. Use tables from dataset part IV (‘DepartmentsIV‘, ‘EmployeesIV‘, ‘ProjectsIV‘, ‘TasksIV‘, ‘TimeLogsIV‘). Window functions (like ‘RANK()‘, ‘ROW_NUMBER() OVER()‘) should NOT be used as they are covered later in the course sequence.

1. **Hardcore Problem**
   Problem: Identify the top 2 departments by the total salary of their ’Senior’ employees. A ’Senior’ employee is defined as someone with a salary > $70,000 AND hired on or after ’2020-01-01’ AND has logged time (in ‘TimeLogsIV‘) on at least one task belonging to a ’Critical’ project. A ’Critical’ project is any project from ‘ProjectsIV‘ with a budget > $150,000.

   For these top 2 departments, display:

   a. Department Name.
   b. Total salary of these qualified ’Senior’ employees in that department.
   c. The count of such ’Senior’ employees in the department.
   d. Using a LATERAL join, for each of these top 2 departments, find the employee (can be any employee in that department, not necessarily senior) who has logged time against the highest number of distinct projects (based on ‘TimeLogsIV‘ and ‘TasksIV‘). If there’s a tie in distinct project count, pick the one with the lower ‘employeeId‘. Show this employee’s name and their distinct project count.

   Additionally, for the single department (from the top 2 identified above) that has the absolute highest total senior salary:

   e. Display the organizational hierarchy for its department head (employee specified in ‘DepartmentsIV.headEmployeeId‘), showing the reporting line upwards to the CEO (employee with ‘managerId IS NULL‘). List employee ID, name, manager ID, and level in hierarchy (0 for the department head, increasing for their managers).

   Constraints:

   - Departments must have at least 2 qualified ’Senior’ employees to be considered for the top 2.
   - Use ‘FETCH FIRST ... ROWS ONLY‘ to get the top 2 departments based on total senior salary (descending).
   - The final list of top 2 departments should be ordered by their total senior salary in descending order.
   - The hierarchy should be for the head of the #1 department from this list.
   - All parts of the problem should be solved within a single SQL statement where possible (the hierarchy query might be separate if needed for clarity, but aim to use CTEs effectively if combining). Ideally, two main ‘SELECT‘ statements: one for the top 2 departments’ details, and one for the hierarchy of the #1 department’s head.