Common Table Expressions - CTEs

Advanced Query Techniques: Solutions

May 17, 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
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): Solutions for Meanings, Values, Relations, and Advantages

1.1 Exercise 1: Basic CTE for Readability

```sql
WITH TechDepartment AS (
    SELECT departmentId
    FROM DepartmentsI
    WHERE departmentName = 'Technology'
)
SELECT e.employeeName, e.salary, d.departmentName
FROM EmployeesI e
JOIN TechDepartment td ON e.departmentId = td.departmentId
JOIN DepartmentsI d ON e.departmentId = d.departmentId
WHERE e.salary > 90000.00;
```

Listing 2: Solution for Exercise 1.1

1.2 Exercise 2: CTE Referenced Multiple Times

```sql
WITH DepartmentAvgSalary AS (
    SELECT
        departmentId,
        AVG(salary) AS avgSalaryForDept
    FROM EmployeesI
    WHERE departmentId IS NOT NULL
    GROUP BY departmentId
)
SELECT
    e.employeeName,
    e.salary,
    d.departmentName,
    das.avgSalaryForDept
FROM EmployeesI e
JOIN DepartmentsI d ON e.departmentId = d.departmentId
JOIN DepartmentAvgSalary das ON e.departmentId = das.departmentId
WHERE e.salary > das.avgSalaryForDept
ORDER BY d.departmentName, e.employeeName;
```

Listing 3: Solution for Exercise 1.2

1.3 Exercise 3: Nested CTEs

```sql
WITH RelevantDepartments AS (
    SELECT departmentId, departmentName, locationCity
    FROM DepartmentsI
    WHERE locationCity IN ('New York', 'London')
), FilteredEmployees AS (
    SELECT employeeName, salary, departmentId, hireDate
    FROM EmployeesI
    WHERE EXTRACT(YEAR FROM hireDate) > 2019
)
SELECT
    fe.employeeName,
    fe.salary,
    rd.departmentName,
    rd.locationCity,
    fe.hireDate
FROM FilteredEmployees fe
JOIN RelevantDepartments rd ON fe.departmentId = rd.departmentId
ORDER BY rd.departmentName, fe.employeeName;
```

Listing 4: Solution for Exercise 1.3

1.4 Exercise 4: Recursive CTE for Hierarchical Data

```sql
WITH RECURSIVE EmployeeHierarchy AS (
    SELECT
        employeeId,
        employeeName,
        managerId,
        0 AS level
    FROM EmployeesI
    WHERE employeeId = 103 -- Starting employee (Charlie Brown)

    UNION ALL

    SELECT
        e.employeeId,
        e.employeeName,
        e.managerId,
        eh.level + 1
    FROM EmployeesI e
    JOIN EmployeeHierarchy eh ON e.employeeId = eh.managerId
)
SELECT employeeId, employeeName, managerId, level
FROM EmployeeHierarchy
ORDER BY level DESC;
```

Listing 5: Solution for Exercise 1.4

2 Category (ii): Solutions for Disadvantages

2.1 Exercise 1: Potential Performance Issue (Optimization Fence / Materialization)

```
WITH ProductRevenue AS (
    SELECT
        st.productId,
        SUM(p.basePrice * st.quantitySold * (1 - st.discount)) AS totalRevenue
    FROM SalesTransactionsII st
    JOIN ProductsII p ON st.productId = p.productId
    GROUP BY st.productId -- This computes revenue for ALL products
)
SELECT
    p.productName,
    pr.totalRevenue
FROM ProductRevenue pr
JOIN ProductsII p ON pr.productId = p.productId
JOIN ProductCategoriesII pc ON p.categoryId = pc.categoryId
WHERE pc.categoryName = ’Electronics’;
-- Note: The disadvantage is that the CTE ‘ProductRevenue‘ might be fully computed
-- before the ‘WHERE pc.categoryName = ’Electronics’‘ filter is applied, especially
-- if the CTE is complex or the optimizer chooses to materialize it.
```

Listing 6: Solution for Exercise 2.1

2.2 Exercise 2: No Indexing on CTE Results

```
-- The dataset setup ensures some sales are in the previous month.
-- Example: If current month is February, this looks for January sales.
WITH PreviousMonthSalesProducts AS (
    SELECT DISTINCT productId
    FROM SalesTransactionsII
    WHERE DATE_TRUNC(’month’, saleDate) = DATE_TRUNC(’month’, CURRENT_DATE - INTERVAL ’1 month’)
)
-- Simulating multiple uses of the intermediate "Previous Month Sales Products" concept:
-- Use 1: List product names
SELECT p.productName
FROM ProductsII p
WHERE p.productId IN (SELECT productId FROM PreviousMonthSalesProducts)
UNION ALL
-- Use 2: Count these products (conceptually another query part)
SELECT CONCAT(’Total Previous Month Sales Products: ’, COUNT(productId)::TEXT)
FROM PreviousMonthSalesProducts;
-- Note: The disadvantage is that ‘PreviousMonthSalesProducts‘ logic is either re-run or its temporary,
-- unindexed result is scanned. If this CTE were used many times in a very complex query,
-- this could be inefficient compared to a (materialized and indexed) temporary table
-- in procedural contexts.
```

Listing 7: Solution for Exercise 2.2

2.3 Exercise 3: CTE Scope Limitation

```
-- Query 1: Calculate total sales for Books (This works)
WITH BookSales AS (
    SELECT SUM(p.basePrice * st.quantitySold * (1 - st.discount)) AS totalBookRevenue
    FROM SalesTransactionsII st
    JOIN ProductsII p ON st.productId = p.productId
    JOIN ProductCategoriesII pc ON p.categoryId = pc.categoryId
    WHERE pc.categoryName = ’Books’
)
SELECT totalBookRevenue FROM BookSales;

-- Query 2: Attempt to use BookSales CTE (this will fail, illustrating scope)
-- SELECT totalBookRevenue * 0.1 AS tenPercentOfBookRevenue FROM BookSales;
-- The above query would cause an error: "ERROR: relation "booksales" does not exist"
-- This demonstrates the scope limitation.

-- To achieve the desired outcome, the CTE must be re-declared if used in a separate statement:
WITH BookSales AS (
    SELECT SUM(p.basePrice * st.quantitySold * (1 - st.discount)) AS totalBookRevenue
    FROM SalesTransactionsII st
    JOIN ProductsII p ON st.productId = p.productId
    JOIN ProductCategoriesII pc ON p.categoryId = pc.categoryId
    WHERE pc.categoryName = ’Books’
)
SELECT totalBookRevenue * 0.1 AS approximateProfit FROM BookSales;
-- Note: The point is that ‘BookSales‘ had to be defined again.
-- If it were a very complex CTE, this is redundant.
```

Listing 8: Solution for Exercise 2.3

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

3.1 Exercise 1: Replacing Repeated Subqueries

```sql
WITH Orders2022 AS (
    SELECT DISTINCT customerId
    FROM OrdersIII
    WHERE EXTRACT(YEAR FROM orderDate) = 2022
), Orders2023 AS (
    SELECT DISTINCT customerId
    FROM OrdersIII
    WHERE EXTRACT(YEAR FROM orderDate) = 2023
)
SELECT c.customerName, c.city
FROM CustomersIII c
WHERE c.customerId IN (SELECT customerId FROM Orders2022)
  AND c.customerId IN (SELECT customerId FROM Orders2023);
```

Listing 9: Solution for Exercise 3.1

Alternatively, a slightly different CTE structure could also work:

```sql
WITH CustomerYearlyOrders AS (
    SELECT customerId, EXTRACT(YEAR FROM orderDate) AS orderYear
    FROM OrdersIII
    WHERE EXTRACT(YEAR FROM orderDate) IN (2022, 2023)
    GROUP BY customerId, EXTRACT(YEAR FROM orderDate)
)
SELECT c.customerName, c.city
FROM CustomersIII c
WHERE EXISTS (SELECT 1 FROM CustomerYearlyOrders cyo WHERE cyo.
   customerId = c.customerId AND cyo.orderYear = 2022)
  AND EXISTS (SELECT 1 FROM CustomerYearlyOrders cyo WHERE cyo.
   customerId = c.customerId AND cyo.orderYear = 2023);
```

Listing 10: Alternative Solution for Exercise 3.1

3.2 Exercise 2: Simplifying Complex Joins and Filters

```sql
WITH OrderTotals AS (
    SELECT
        orderId,
        SUM(quantity * pricePerUnit) AS totalValue
    FROM OrderItemsIII
    GROUP BY orderId
), NorthAmericaHighValueOrders AS (
    SELECT o.orderId
    FROM OrdersIII o
    JOIN OrderTotals ot ON o.orderId = ot.orderId
    WHERE o.shipmentRegion = ’North America’ AND ot.totalValue > 600.00
)
SELECT DISTINCT p.productName, p.category
FROM ProductsMasterIII p
JOIN OrderItemsIII oi ON p.productId = oi.productId
WHERE oi.orderId IN (SELECT orderId FROM NorthAmericaHighValueOrders)
```

```
ORDER BY p.productName;
```

Listing 11: Solution for Exercise 3.2

3.3 Exercise 3: Avoiding Temporary Tables for Single-Query Scope

```
WITH OrderValues AS (
    SELECT
        o.orderId,
        o.shipmentRegion,
        SUM(oi.quantity * oi.pricePerUnit) AS orderTotal
    FROM OrdersIII o
    JOIN OrderItemsIII oi ON o.orderId = oi.orderId
    GROUP BY o.orderId, o.shipmentRegion
), RegionalAverageOrderValue AS (
    SELECT
        shipmentRegion,
        AVG(orderTotal) AS avgRegionalValue
    FROM OrderValues
    GROUP BY shipmentRegion
), OverallAverageOrderValue AS (
    SELECT AVG(orderTotal) AS overallAvgValue
    FROM OrderValues
)
SELECT
    raov.shipmentRegion,
    raov.avgRegionalValue
FROM RegionalAverageOrderValue raov, OverallAverageOrderValue oaov
WHERE raov.avgRegionalValue > oaov.overallAvgValue
ORDER BY raov.avgRegionalValue DESC;
```

Listing 12: Solution for Exercise 3.3

3.4 Exercise 4: Step-by-Step Multi-Level Aggregations (No Window Functions)

```
WITH MonthlyCategorySales AS (
    SELECT
        p.category,
        TO_CHAR(o.orderDate, ’YYYY-MM’) AS saleMonth,
        SUM(oi.quantity) AS totalQuantity
    FROM OrderItemsIII oi
    JOIN ProductsMasterIII p ON oi.productId = p.productId
    JOIN OrdersIII o ON oi.orderId = o.orderId
    GROUP BY p.category, TO_CHAR(o.orderDate, ’YYYY-MM’)
),
MaxQuantityPerCategory AS (
    SELECT
        category,
        MAX(totalQuantity) AS maxTotalQuantity
    FROM MonthlyCategorySales
    GROUP BY category
)
SELECT
```

```sql
    mcs.category,
    mcs.saleMonth,
    mcs.totalQuantity
FROM MonthlyCategorySales mcs
JOIN MaxQuantityPerCategory mqpc ON mcs.category = mqpc.category AND mcs.totalQuantity = mqpc.maxTotalQuantity
ORDER BY mcs.category, mcs.saleMonth;
```

Listing 13: Solution for Exercise 3.4

4 Category (iv): Solution for Hardcore Combined Problem

```sql
WITH CriticalProjects AS (
    SELECT projectId
    FROM ProjectsIV
    WHERE budget > 150000.00
),
EmployeeLoggedTimeOnCriticalTasks AS ( -- Renamed for clarity and corrected logic
    SELECT DISTINCT tl.employeeId
    FROM TimeLogsIV tl
    JOIN TasksIV t ON tl.taskId = t.taskId
    WHERE t.projectId IN (SELECT projectId FROM CriticalProjects)
),
SeniorEmployees AS (
    SELECT
        e.employeeId,
        e.employeeName,
        e.departmentId,
        e.salary,
        e.hireDate,
        d.departmentName
    FROM EmployeesIV e
    JOIN DepartmentsIV d ON e.departmentId = d.departmentId
    WHERE e.salary > 70000.00
      AND e.hireDate >= '2020-01-01'
      AND e.employeeId IN (SELECT employeeId FROM EmployeeLoggedTimeOnCriticalTasks)
),
DepartmentSeniorStats AS (
    SELECT
        s.departmentId,
        s.departmentName,
        SUM(s.salary) AS totalSeniorSalary,
        COUNT(s.employeeId) AS countSeniorEmployees
    FROM SeniorEmployees s
    GROUP BY s.departmentId, s.departmentName
    HAVING COUNT(s.employeeId) >= 2
),
TopDepartments AS (
    SELECT
        departmentId,
        departmentName,
        totalSeniorSalary,
        countSeniorEmployees
    FROM DepartmentSeniorStats
    ORDER BY totalSeniorSalary DESC
    FETCH FIRST 2 ROWS ONLY
),
EmployeeProjectCounts AS (
    SELECT
        tl.employeeId,
        e.departmentId, -- DepartmentId of the employee logging time
        COUNT(DISTINCT t.projectId) AS distinctProjectCount
    FROM TimeLogsIV tl
    JOIN TasksIV t ON tl.taskId = t.taskId
    JOIN EmployeesIV e ON tl.employeeId = e.employeeId -- Join to get employee's department
    GROUP BY tl.employeeId, e.departmentId
)
SELECT
    td.departmentName,
    td.totalSeniorSalary,
    td.countSeniorEmployees,
    empDetails.employeeName AS mostActiveEmployeeName,
    empDetails.distinctProjectCount AS mostActiveEmployeeProjectCount
FROM TopDepartments td
LEFT JOIN LATERAL (
    SELECT
        epc.employeeId,
        eiv.employeeName,
        epc.distinctProjectCount
    FROM EmployeeProjectCounts epc
    JOIN EmployeesIV eiv ON epc.employeeId = eiv.employeeId
    WHERE epc.departmentId = td.departmentId -- Match employee's department with the top department
    ORDER BY epc.distinctProjectCount DESC, epc.employeeId ASC
    LIMIT 1
) empDetails ON TRUE
ORDER BY td.totalSeniorSalary DESC;
```

Listing 14: Solution for Hardcore Problem (Part 1: Top Departments Info)

```sql
WITH CriticalProjects AS ( -- Re-declare or ensure CTEs are available if running separately
    SELECT projectId
    FROM ProjectsIV
    WHERE budget > 150000.00
),
EmployeeLoggedTimeOnCriticalTasks AS (
    SELECT DISTINCT tl.employeeId
    FROM TimeLogsIV tl
    JOIN TasksIV t ON tl.taskId = t.taskId
    WHERE t.projectId IN (SELECT projectId FROM CriticalProjects)
),
SeniorEmployees AS (
    SELECT
        e.employeeId,
        e.departmentId,
        e.salary,
        e.hireDate
    FROM EmployeesIV e
    WHERE e.salary > 70000.00
      AND e.hireDate >= '2020-01-01'
      AND e.employeeId IN (SELECT employeeId FROM EmployeeLoggedTimeOnCriticalTasks)
),
DepartmentSeniorStats AS (
    SELECT
        s.departmentId,
        SUM(s.salary) AS totalSeniorSalary
    FROM SeniorEmployees s
    GROUP BY s.departmentId
    HAVING COUNT(s.employeeId) >= 2
),
TopDepartmentForHierarchy AS ( -- Select only the #1 department
    SELECT
        ds.departmentId,
        d.departmentName,
        d.headEmployeeId
    FROM DepartmentSeniorStats ds
    JOIN DepartmentsIV d ON ds.departmentId = d.departmentId
    ORDER BY ds.totalSeniorSalary DESC
    FETCH FIRST 1 ROWS ONLY
),
EmployeeHierarchy AS (
    SELECT
        e.employeeId,
        e.employeeName,
        e.managerId,
        0 AS level
    FROM EmployeesIV e
    WHERE e.employeeId = (SELECT headEmployeeId FROM TopDepartmentForHierarchy) -- Start with head of #1 dept

    UNION ALL

    SELECT
        e_mgr.employeeId,
        e_mgr.employeeName,
        e_mgr.managerId,
        eh.level + 1
    FROM EmployeesIV e_mgr
    JOIN EmployeeHierarchy eh ON e_mgr.employeeId = eh.managerId
    WHERE eh.managerId IS NOT NULL -- Stop when managerId is NULL (top of hierarchy)
)
SELECT
    (SELECT 'Hierarchy for Head of Department: ' || tdH.departmentName FROM TopDepartmentForHierarchy tdH) AS context,
    eh.employeeId,
    eh.employeeName,
    eh.managerId,
    eh.level
FROM EmployeeHierarchy eh
ORDER BY eh.level DESC;
```

Listing 15: Solution for Hardcore Problem (Part 2: Hierarchy for #1 Department Head)