# Complete Guide to Subqueries in SSMS

## Table of Contents

1. [What are Subqueries?](#what-are-subqueries)
2. [Types of Subqueries](#types-of-subqueries)
3. [Subquery Locations](#subquery-locations)
4. [Single-Row Subqueries](#single-row-subqueries)
5. [Multiple-Row Subqueries](#multiple-row-subqueries)
6. [Correlated Subqueries](#correlated-subqueries)
7. [Subqueries in SELECT Clause](#subqueries-in-select-clause)
8. [Subqueries in FROM Clause (Derived Tables)](#subqueries-in-from-clause)
9. [Subqueries in WHERE Clause](#subqueries-in-where-clause)
10. [Subqueries with EXISTS and NOT EXISTS](#subqueries-with-exists-and-not-exists)
11. [Subqueries with IN and NOT IN](#subqueries-with-in-and-not-in)
12. [Subqueries with ANY and ALL](#subqueries-with-any-and-all)
13. [Nested Subqueries](#nested-subqueries)
14. [Subqueries vs JOINs](#subqueries-vs-joins)
15. [Performance Considerations](#performance-considerations)
16. [Common Use Cases](#common-use-cases)
17. [Best Practices](#best-practices)
18. [Troubleshooting Subqueries](#troubleshooting-subqueries)

## What are Subqueries?

A **Subquery** (also called an inner query or nested query) is a query nested inside another SQL statement. Subqueries can be used in SELECT, INSERT, UPDATE, and DELETE statements, providing a powerful way to perform complex data retrieval and manipulation.

### Key Characteristics:

* **Nested Execution**: Inner query executes first, result used by outer query
* **Encapsulation**: Enclosed in parentheses
* **Independence**: Can often be tested separately
* **Flexibility**: Can return single value, single row, or multiple rows
* **Scope**: Can reference outer query columns (correlated) or be independent

### Benefits:

* **Modularity**: Break complex queries into logical parts
* **Readability**: Sometimes clearer than complex JOINs
* **Dynamic Filtering**: Filter based on calculated values
* **Data Validation**: Check for existence of related data
* **Intermediate Results**: Use calculated values in further operations

## Types of Subqueries

### 1. Single-Row Subqueries

Return exactly one row with one or more columns.

### 2. Multiple-Row Subqueries

Return multiple rows (used with IN, ANY, ALL, EXISTS).

### 3. Multiple-Column Subqueries

Return multiple columns in each row.

### 4. Correlated Subqueries

Reference columns from the outer query.

### 5. Non-Correlated Subqueries

Independent of outer query, execute once.

## Sample Data Setup

-- Create comprehensive sample tables  
CREATE TABLE employees (  
 employee\_id INT IDENTITY(1,1) PRIMARY KEY,  
 first\_name VARCHAR(50),  
 last\_name VARCHAR(50),  
 email VARCHAR(100),  
 phone VARCHAR(20),  
 department\_id INT,  
 manager\_id INT,  
 salary DECIMAL(10,2),  
 hire\_date DATE,  
 job\_title VARCHAR(50),  
 commission\_pct DECIMAL(3,2),  
 is\_active BIT DEFAULT 1  
);  
  
CREATE TABLE departments (  
 department\_id INT PRIMARY KEY,  
 department\_name VARCHAR(50),  
 location VARCHAR(50),  
 budget DECIMAL(12,2),  
 manager\_id INT  
);  
  
CREATE TABLE projects (  
 project\_id INT IDENTITY(1,1) PRIMARY KEY,  
 project\_name VARCHAR(100),  
 department\_id INT,  
 start\_date DATE,  
 end\_date DATE,  
 budget DECIMAL(12,2),  
 status VARCHAR(20)  
);  
  
CREATE TABLE project\_assignments (  
 assignment\_id INT IDENTITY(1,1) PRIMARY KEY,  
 employee\_id INT,  
 project\_id INT,  
 assigned\_date DATE,  
 hours\_allocated INT,  
 role VARCHAR(50)  
);  
  
CREATE TABLE salaries\_history (  
 history\_id INT IDENTITY(1,1) PRIMARY KEY,  
 employee\_id INT,  
 salary DECIMAL(10,2),  
 effective\_date DATE,  
 end\_date DATE,  
 change\_reason VARCHAR(100)  
);  
  
-- Insert sample data  
INSERT INTO departments VALUES  
(10, 'IT', 'New York', 500000, NULL),  
(20, 'Sales', 'Chicago', 400000, NULL),  
(30, 'HR', 'Los Angeles', 250000, NULL),  
(40, 'Finance', 'Boston', 350000, NULL),  
(50, 'Marketing', 'San Francisco', 300000, NULL);  
  
INSERT INTO employees VALUES  
('John', 'Doe', 'john.doe@company.com', '555-0001', 10, NULL, 85000, '2018-01-15', 'IT Manager', NULL, 1),  
('Jane', 'Smith', 'jane.smith@company.com', '555-0002', 10, 1, 75000, '2019-03-20', 'Senior Developer', NULL, 1),  
('Bob', 'Johnson', 'bob.johnson@company.com', '555-0003', 20, NULL, 90000, '2017-06-10', 'Sales Manager', 0.10, 1),  
('Alice', 'Brown', 'alice.brown@company.com', '555-0004', 20, 3, 65000, '2020-02-14', 'Sales Rep', 0.15, 1),  
('Charlie', 'Wilson', 'charlie.wilson@company.com', '555-0005', 30, NULL, 70000, '2019-08-05', 'HR Manager', NULL, 1),  
('Diana', 'Ross', 'diana.ross@company.com', '555-0006', 10, 1, 72000, '2020-01-10', 'Developer', NULL, 1),  
('Elvis', 'King', 'elvis.king@company.com', '555-0007', 40, NULL, 95000, '2016-11-20', 'Finance Manager', NULL, 1),  
('Frank', 'Miller', 'frank.miller@company.com', '555-0008', 20, 3, 68000, '2021-04-15', 'Sales Rep', 0.12, 1),  
('Grace', 'Lee', 'grace.lee@company.com', '555-0009', 50, NULL, 80000, '2019-09-01', 'Marketing Manager', NULL, 1),  
('Henry', 'Ford', 'henry.ford@company.com', '555-0010', 10, 1, 70000, '2021-07-20', 'Developer', NULL, 1);  
  
INSERT INTO projects VALUES  
('Website Redesign', 10, '2023-01-01', '2023-06-30', 150000, 'Completed'),  
('CRM Implementation', 20, '2023-02-15', '2023-12-31', 200000, 'In Progress'),  
('HR Portal', 30, '2023-03-01', '2023-09-30', 100000, 'In Progress'),  
('Financial Audit System', 40, '2023-01-15', '2023-08-31', 180000, 'Completed'),  
('Marketing Campaign', 50, '2023-04-01', '2023-10-31', 120000, 'In Progress'),  
('Mobile App', 10, '2023-05-01', '2024-03-31', 250000, 'Planning');  
  
INSERT INTO project\_assignments VALUES  
(1, 1, '2023-01-01', 160, 'Project Lead'),  
(2, 1, '2023-01-01', 120, 'Developer'),  
(6, 1, '2023-01-01', 120, 'Developer'),  
(3, 2, '2023-02-15', 80, 'Project Lead'),  
(4, 2, '2023-02-15', 40, 'Sales Analyst'),  
(5, 3, '2023-03-01', 100, 'Project Lead'),  
(7, 4, '2023-01-15', 140, 'Project Lead'),  
(9, 5, '2023-04-01', 120, 'Project Lead'),  
(1, 6, '2023-05-01', 160, 'Project Lead'),  
(2, 6, '2023-05-01', 80, 'Developer');  
  
INSERT INTO salaries\_history VALUES  
(1, 75000, '2018-01-15', '2020-01-14', 'Initial Hire'),  
(1, 80000, '2020-01-15', '2022-01-14', 'Annual Review'),  
(1, 85000, '2022-01-15', NULL, 'Promotion'),  
(2, 65000, '2019-03-20', '2021-03-19', 'Initial Hire'),  
(2, 70000, '2021-03-20', '2023-03-19', 'Annual Review'),  
(2, 75000, '2023-03-20', NULL, 'Performance Increase');

## Single-Row Subqueries

Single-row subqueries return exactly one row and can be used with comparison operators (=, >, <, >=, <=, <>).

-- Basic single-row subquery examples  
  
-- 1. Find employees earning more than the average salary  
SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name AS employee\_name,  
 salary,  
 (SELECT AVG(salary) FROM employees) AS avg\_salary  
FROM employees  
WHERE salary > (SELECT AVG(salary) FROM employees);  
  
-- 2. Find employee with the highest salary  
SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name AS employee\_name,  
 salary  
FROM employees  
WHERE salary = (SELECT MAX(salary) FROM employees);  
  
-- 3. Find employees in the department with the largest budget  
SELECT   
 e.employee\_id,  
 e.first\_name + ' ' + e.last\_name AS employee\_name,  
 e.department\_id,  
 d.department\_name,  
 d.budget  
FROM employees e  
JOIN departments d ON e.department\_id = d.department\_id  
WHERE d.budget = (SELECT MAX(budget) FROM departments);  
  
-- 4. Compare individual salary to average in their department  
SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name AS employee\_name,  
 department\_id,  
 salary,  
 (SELECT AVG(salary)   
 FROM employees e2   
 WHERE e2.department\_id = e.department\_id) AS dept\_avg\_salary,  
 salary - (SELECT AVG(salary)   
 FROM employees e2   
 WHERE e2.department\_id = e.department\_id) AS diff\_from\_dept\_avg  
FROM employees e  
ORDER BY department\_id, salary DESC;  
  
-- 5. Find projects with budget higher than average project budget  
SELECT   
 project\_id,  
 project\_name,  
 budget,  
 (SELECT AVG(budget) FROM projects) AS avg\_project\_budget  
FROM projects  
WHERE budget > (SELECT AVG(budget) FROM projects)  
ORDER BY budget DESC;

## Multiple-Row Subqueries

Multiple-row subqueries return more than one row and must be used with operators that can handle multiple values (IN, ANY, ALL, EXISTS).

-- Multiple-row subquery examples  
  
-- 1. Find employees working in departments located in specific cities  
SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name AS employee\_name,  
 department\_id  
FROM employees  
WHERE department\_id IN (  
 SELECT department\_id   
 FROM departments   
 WHERE location IN ('New York', 'Chicago')  
);  
  
-- 2. Find employees who are managers (appear in manager\_id column)  
SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name AS employee\_name,  
 job\_title  
FROM employees  
WHERE employee\_id IN (  
 SELECT DISTINCT manager\_id   
 FROM employees   
 WHERE manager\_id IS NOT NULL  
);  
  
-- 3. Find departments that have at least one active project  
SELECT   
 department\_id,  
 department\_name,  
 location  
FROM departments  
WHERE department\_id IN (  
 SELECT DISTINCT department\_id   
 FROM projects   
 WHERE status IN ('In Progress', 'Planning')  
);  
  
-- 4. Find employees assigned to multiple projects  
SELECT   
 e.employee\_id,  
 e.first\_name + ' ' + e.last\_name AS employee\_name,  
 COUNT(pa.project\_id) AS project\_count  
FROM employees e  
WHERE employee\_id IN (  
 SELECT employee\_id   
 FROM project\_assignments   
 GROUP BY employee\_id   
 HAVING COUNT(project\_id) > 1  
)  
GROUP BY e.employee\_id, e.first\_name, e.last\_name;  
  
-- 5. Find projects in departments with more than 2 employees  
SELECT   
 project\_id,  
 project\_name,  
 department\_id,  
 status  
FROM projects  
WHERE department\_id IN (  
 SELECT department\_id   
 FROM employees   
 GROUP BY department\_id   
 HAVING COUNT(\*) > 2  
);

## Correlated Subqueries

Correlated subqueries reference columns from the outer query and execute once for each row processed by the outer query.

-- Correlated subquery examples  
  
-- 1. Find employees earning more than average in their department  
SELECT   
 e.employee\_id,  
 e.first\_name + ' ' + e.last\_name AS employee\_name,  
 e.department\_id,  
 e.salary,  
 (SELECT AVG(salary)   
 FROM employees e2   
 WHERE e2.department\_id = e.department\_id) AS dept\_avg  
FROM employees e  
WHERE e.salary > (  
 SELECT AVG(salary)   
 FROM employees e2   
 WHERE e2.department\_id = e.department\_id  
);  
  
-- 2. Find employees who earn more than their manager  
SELECT   
 e.employee\_id,  
 e.first\_name + ' ' + e.last\_name AS employee\_name,  
 e.salary AS employee\_salary,  
 e.manager\_id,  
 (SELECT salary   
 FROM employees m   
 WHERE m.employee\_id = e.manager\_id) AS manager\_salary  
FROM employees e  
WHERE e.salary > (  
 SELECT salary   
 FROM employees m   
 WHERE m.employee\_id = e.manager\_id  
);  
  
-- 3. Find departments with above-average number of employees  
SELECT   
 d.department\_id,  
 d.department\_name,  
 (SELECT COUNT(\*)   
 FROM employees e   
 WHERE e.department\_id = d.department\_id) AS employee\_count,  
 (SELECT AVG(emp\_count)   
 FROM (  
 SELECT COUNT(\*) AS emp\_count   
 FROM employees   
 GROUP BY department\_id  
 ) AS dept\_counts) AS avg\_employees\_per\_dept  
FROM departments d  
WHERE (SELECT COUNT(\*)   
 FROM employees e   
 WHERE e.department\_id = d.department\_id) >   
 (SELECT AVG(emp\_count)   
 FROM (  
 SELECT COUNT(\*) AS emp\_count   
 FROM employees   
 GROUP BY department\_id  
 ) AS dept\_counts);  
  
-- 4. Find latest salary for each employee  
SELECT   
 e.employee\_id,  
 e.first\_name + ' ' + e.last\_name AS employee\_name,  
 (SELECT TOP 1 salary   
 FROM salaries\_history sh   
 WHERE sh.employee\_id = e.employee\_id   
 ORDER BY effective\_date DESC) AS latest\_historical\_salary,  
 e.salary AS current\_salary  
FROM employees e  
WHERE EXISTS (  
 SELECT 1   
 FROM salaries\_history sh   
 WHERE sh.employee\_id = e.employee\_id  
);  
  
-- 5. Find employees with salary higher than all employees in HR department  
SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name AS employee\_name,  
 department\_id,  
 salary  
FROM employees e  
WHERE salary > ALL (  
 SELECT salary   
 FROM employees   
 WHERE department\_id = (  
 SELECT department\_id   
 FROM departments   
 WHERE department\_name = 'HR'  
 )  
);

## Subqueries in SELECT Clause

Subqueries in the SELECT clause (also called scalar subqueries) return a single value for each row of the outer query.

-- Subqueries in SELECT clause examples  
  
-- 1. Show employee info with department details  
SELECT   
 e.employee\_id,  
 e.first\_name + ' ' + e.last\_name AS employee\_name,  
 e.salary,  
 (SELECT department\_name   
 FROM departments d   
 WHERE d.department\_id = e.department\_id) AS department\_name,  
 (SELECT location   
 FROM departments d   
 WHERE d.department\_id = e.department\_id) AS department\_location,  
 (SELECT COUNT(\*)   
 FROM employees e2   
 WHERE e2.department\_id = e.department\_id) AS dept\_employee\_count  
FROM employees e;  
  
-- 2. Calculate various aggregates for each employee  
SELECT   
 e.employee\_id,  
 e.first\_name + ' ' + e.last\_name AS employee\_name,  
 e.salary,  
 (SELECT AVG(salary) FROM employees) AS overall\_avg\_salary,  
 (SELECT AVG(salary)   
 FROM employees e2   
 WHERE e2.department\_id = e.department\_id) AS dept\_avg\_salary,  
 (SELECT MAX(salary)   
 FROM employees e2   
 WHERE e2.department\_id = e.department\_id) AS dept\_max\_salary,  
 (SELECT COUNT(\*)   
 FROM project\_assignments pa   
 WHERE pa.employee\_id = e.employee\_id) AS projects\_assigned  
FROM employees e;  
  
-- 3. Show project information with related counts  
SELECT   
 p.project\_id,  
 p.project\_name,  
 p.budget,  
 (SELECT department\_name   
 FROM departments d   
 WHERE d.department\_id = p.department\_id) AS department\_name,  
 (SELECT COUNT(\*)   
 FROM project\_assignments pa   
 WHERE pa.project\_id = p.project\_id) AS team\_size,  
 (SELECT SUM(hours\_allocated)   
 FROM project\_assignments pa   
 WHERE pa.project\_id = p.project\_id) AS total\_hours\_allocated,  
 p.budget / NULLIF((SELECT COUNT(\*)   
 FROM project\_assignments pa   
 WHERE pa.project\_id = p.project\_id), 0) AS budget\_per\_team\_member  
FROM projects p;  
  
-- 4. Calculate salary percentiles  
SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name AS employee\_name,  
 salary,  
 (SELECT COUNT(\*)   
 FROM employees e2   
 WHERE e2.salary < e.salary) AS employees\_earning\_less,  
 (SELECT COUNT(\*)   
 FROM employees e2   
 WHERE e2.salary > e.salary) AS employees\_earning\_more,  
 CAST((SELECT COUNT(\*)   
 FROM employees e2   
 WHERE e2.salary < e.salary) AS FLOAT) /   
 (SELECT COUNT(\*) FROM employees) \* 100 AS salary\_percentile  
FROM employees e  
ORDER BY salary DESC;  
  
-- 5. Show manager information for each employee  
SELECT   
 e.employee\_id,  
 e.first\_name + ' ' + e.last\_name AS employee\_name,  
 e.job\_title,  
 e.salary,  
 CASE   
 WHEN e.manager\_id IS NOT NULL THEN  
 (SELECT m.first\_name + ' ' + m.last\_name   
 FROM employees m   
 WHERE m.employee\_id = e.manager\_id)  
 ELSE 'No Manager'  
 END AS manager\_name,  
 CASE   
 WHEN e.manager\_id IS NOT NULL THEN  
 (SELECT m.salary   
 FROM employees m   
 WHERE m.employee\_id = e.manager\_id)  
 ELSE NULL  
 END AS manager\_salary  
FROM employees e;

## Subqueries in FROM Clause (Derived Tables)

Subqueries in the FROM clause create temporary result sets (derived tables) that can be queried like regular tables.

-- Derived table examples  
  
-- 1. Calculate department statistics and use them in outer query  
SELECT   
 dept\_stats.department\_id,  
 dept\_stats.department\_name,  
 dept\_stats.employee\_count,  
 dept\_stats.avg\_salary,  
 dept\_stats.total\_salary,  
 dept\_stats.salary\_range,  
 CASE   
 WHEN dept\_stats.avg\_salary > overall\_stats.overall\_avg THEN 'Above Average'  
 WHEN dept\_stats.avg\_salary < overall\_stats.overall\_avg THEN 'Below Average'  
 ELSE 'Average'  
 END AS salary\_category  
FROM (  
 SELECT   
 d.department\_id,  
 d.department\_name,  
 COUNT(e.employee\_id) AS employee\_count,  
 AVG(e.salary) AS avg\_salary,  
 SUM(e.salary) AS total\_salary,  
 MAX(e.salary) - MIN(e.salary) AS salary\_range  
 FROM departments d  
 LEFT JOIN employees e ON d.department\_id = e.department\_id  
 GROUP BY d.department\_id, d.department\_name  
) AS dept\_stats  
CROSS JOIN (  
 SELECT AVG(salary) AS overall\_avg FROM employees  
) AS overall\_stats  
ORDER BY dept\_stats.avg\_salary DESC;  
  
-- 2. Rank departments by various metrics  
SELECT   
 dept\_metrics.\*,  
 ROW\_NUMBER() OVER(ORDER BY avg\_salary DESC) AS salary\_rank,  
 ROW\_NUMBER() OVER(ORDER BY employee\_count DESC) AS size\_rank,  
 ROW\_NUMBER() OVER(ORDER BY project\_count DESC) AS project\_rank  
FROM (  
 SELECT   
 d.department\_id,  
 d.department\_name,  
 COUNT(DISTINCT e.employee\_id) AS employee\_count,  
 AVG(e.salary) AS avg\_salary,  
 COUNT(DISTINCT p.project\_id) AS project\_count,  
 SUM(p.budget) AS total\_project\_budget  
 FROM departments d  
 LEFT JOIN employees e ON d.department\_id = e.department\_id  
 LEFT JOIN projects p ON d.department\_id = p.department\_id  
 GROUP BY d.department\_id, d.department\_name  
) AS dept\_metrics;  
  
-- 3. Join multiple derived tables  
SELECT   
 emp\_proj.employee\_id,  
 emp\_proj.employee\_name,  
 emp\_proj.project\_count,  
 emp\_proj.total\_hours,  
 sal\_hist.salary\_changes,  
 sal\_hist.total\_salary\_increase  
FROM (  
 SELECT   
 e.employee\_id,  
 e.first\_name + ' ' + e.last\_name AS employee\_name,  
 COUNT(pa.project\_id) AS project\_count,  
 SUM(pa.hours\_allocated) AS total\_hours  
 FROM employees e  
 LEFT JOIN project\_assignments pa ON e.employee\_id = pa.employee\_id  
 GROUP BY e.employee\_id, e.first\_name, e.last\_name  
) AS emp\_proj  
LEFT JOIN (  
 SELECT   
 employee\_id,  
 COUNT(\*) AS salary\_changes,  
 MAX(salary) - MIN(salary) AS total\_salary\_increase  
 FROM salaries\_history  
 GROUP BY employee\_id  
) AS sal\_hist ON emp\_proj.employee\_id = sal\_hist.employee\_id;  
  
-- 4. Complex aggregation with derived tables  
SELECT   
 high\_earners.department\_name,  
 high\_earners.high\_earner\_count,  
 high\_earners.high\_earner\_avg\_salary,  
 all\_emps.total\_employees,  
 all\_emps.dept\_avg\_salary,  
 CAST(high\_earners.high\_earner\_count AS FLOAT) / all\_emps.total\_employees \* 100 AS pct\_high\_earners  
FROM (  
 SELECT   
 d.department\_name,  
 COUNT(\*) AS high\_earner\_count,  
 AVG(e.salary) AS high\_earner\_avg\_salary  
 FROM employees e  
 JOIN departments d ON e.department\_id = d.department\_id  
 WHERE e.salary > (SELECT AVG(salary) FROM employees)  
 GROUP BY d.department\_name  
) AS high\_earners  
JOIN (  
 SELECT   
 d.department\_name,  
 COUNT(\*) AS total\_employees,  
 AVG(e.salary) AS dept\_avg\_salary  
 FROM employees e  
 JOIN departments d ON e.department\_id = d.department\_id  
 GROUP BY d.department\_name  
) AS all\_emps ON high\_earners.department\_name = all\_emps.department\_name;  
  
-- 5. Nested derived tables  
SELECT   
 final\_results.\*,  
 RANK() OVER(ORDER BY performance\_score DESC) AS overall\_rank  
FROM (  
 SELECT   
 dept\_performance.department\_name,  
 dept\_performance.avg\_salary,  
 dept\_performance.project\_count,  
 dept\_performance.employee\_count,  
 (dept\_performance.project\_count \* 0.4 +   
 dept\_performance.employee\_count \* 0.3 +   
 dept\_performance.avg\_salary / 1000 \* 0.3) AS performance\_score  
 FROM (  
 SELECT   
 d.department\_name,  
 AVG(e.salary) AS avg\_salary,  
 COUNT(DISTINCT p.project\_id) AS project\_count,  
 COUNT(DISTINCT e.employee\_id) AS employee\_count  
 FROM departments d  
 LEFT JOIN employees e ON d.department\_id = e.department\_id  
 LEFT JOIN projects p ON d.department\_id = p.department\_id  
 GROUP BY d.department\_name  
 ) AS dept\_performance  
) AS final\_results  
ORDER BY performance\_score DESC;

## Subqueries in WHERE Clause

Subqueries in WHERE clause are the most common type, used for filtering data based on complex conditions.

-- WHERE clause subquery examples  
  
-- 1. Find employees in departments with budget > $300,000  
SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name AS employee\_name,  
 department\_id,  
 salary  
FROM employees  
WHERE department\_id IN (  
 SELECT department\_id   
 FROM departments   
 WHERE budget > 300000  
);  
  
-- 2. Find employees who haven't been assigned to any project  
SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name AS employee\_name,  
 job\_title  
FROM employees  
WHERE employee\_id NOT IN (  
 SELECT DISTINCT employee\_id   
 FROM project\_assignments  
);  
  
-- 3. Find departments with above-average budgets  
SELECT   
 department\_id,  
 department\_name,  
 budget  
FROM departments  
WHERE budget > (SELECT AVG(budget) FROM departments)  
ORDER BY budget DESC;  
  
-- 4. Find employees hired after the average hire date  
SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name AS employee\_name,  
 hire\_date  
FROM employees  
WHERE hire\_date > (SELECT AVG(CAST(hire\_date AS FLOAT)) FROM employees)  
ORDER BY hire\_date;  
  
-- 5. Complex filtering with multiple subqueries  
SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name AS employee\_name,  
 department\_id,  
 salary  
FROM employees  
WHERE department\_id IN (  
 SELECT department\_id   
 FROM departments   
 WHERE location IN ('New York', 'Chicago')  
)  
AND salary > (SELECT AVG(salary) FROM employees)  
AND employee\_id IN (  
 SELECT employee\_id   
 FROM project\_assignments   
 GROUP BY employee\_id   
 HAVING COUNT(project\_id) >= 2  
);  
  
-- 6. Find projects in departments with more employees than average  
SELECT   
 project\_id,  
 project\_name,  
 department\_id  
FROM projects  
WHERE department\_id IN (  
 SELECT department\_id  
 FROM (  
 SELECT   
 department\_id,  
 COUNT(\*) AS emp\_count  
 FROM employees  
 GROUP BY department\_id  
 HAVING COUNT(\*) > (  
 SELECT AVG(dept\_count)  
 FROM (  
 SELECT COUNT(\*) AS dept\_count  
 FROM employees  
 GROUP BY department\_id  
 ) AS dept\_counts  
 )  
 ) AS large\_depts  
);

## Subqueries with EXISTS and NOT EXISTS

EXISTS checks for the existence of rows returned by a subquery. It’s often more efficient than IN for large datasets.

```sql – EXISTS and NOT EXISTS examples

– 1. Find employees who are assigned to at least one project SELECT e.employee\_id, e.first\_name + ’ ’ + e.last\_name AS employee\_name, e.department\_id, e.job\_title FROM employees e WHERE EXISTS ( SELECT 1 FROM project\_assignments pa WHERE pa.employee\_id = e.employee\_id );

– 2. Find employees who are NOT assigned to any project SELECT e.employee\_id, e.first\_name + ’ ’ + e.last\_name AS employee\_name, e.department\_id, e.job\_title FROM employees e WHERE NOT EXISTS ( SELECT 1 FROM project\_assignments pa WHERE pa.employee\_id = e.employee\_id );

– 3. Find departments that have at least one completed project SELECT d.department\_id, d.department\_name, d.location FROM departments d WHERE EXISTS ( SELECT 1 FROM projects p WHERE p.department\_id = d.department\_id AND p.status = ‘Completed’ );

– 4. Find employees who have received salary increases SELECT e.employee\_id, e.first\_name + ’ ’ + e.last\_name AS employee\_name, e.salary AS current\_salary FROM employees e WHERE EXISTS ( SELECT 1 FROM salaries\_history sh WHERE sh.employee\_id = e.employee\_id );

– 5. Find employees earning more than any employee in HR department SELECT e1.employee\_id, e1.first\_name + ’ ’ + e1.last\_name AS employee\_name, e1.department\_id, e1.salary FROM employees e1 WHERE EXISTS ( SELECT 1 FROM employees e2 WHERE e2.department\_id = ( SELECT department\_id FROM departments WHERE department\_name = ‘HR’ ) AND e1.salary > e2.salary );

– 6. Find projects with all team members from same department SELECT p.project\_id, p.project\_name, p.department\_id FROM projects p WHERE NOT EXISTS ( SELECT 1 FROM project\_assignments pa JOIN employees e ON pa.employee\_id = e.employee\_id WHERE pa.project\_id = p.project\_id AND e.department\_id != p.department\_id ) AND EXISTS ( SELECT 1 FROM project\_assignments pa WHERE pa.project\_id = p.project\_id );

– 7. Find managers who manage at least one high-earning employee (>$75,000) SELECT DISTINCT m.employee\_id, m.first\_name + ’ ’ + m.last\_name AS manager\_name, m.job\_title FROM employees m WHERE EXISTS ( SELECT 1 FROM employees e WHERE e.manager\_id = m.employee\_id AND e.salary > 75000 );