Nested Queries (Subqueries) in SQL – Detailed Guide

A nested query (subquery) is a SQL query embedded inside another SQL query. Subqueries help retrieve data based on intermediate results from another query.

Types of Nested Queries

1. Single-Row Subquery

Returns a single value (single row & single column).

```
SELECT name, salary
FROM employees
WHERE salary > (
  SELECT AVG(salary)
  FROM employees
);.
```

Explanation:

- The subquery calculates the average salary.
- The outer query selects employees earning **above the average salary**.

2. Multiple-Row Subquery

Returns multiple values (single column, multiple rows). Use IN, ANY, or ALL.

Example:

```
SELECT name
FROM employees
WHERE department_id IN (
  SELECT department_id
  FROM departments
  WHERE location = 'New York'
```

Explanation:

- Finds departments located in New York.
- Then finds employees who belong to those departments.

3. <u>Multiple-Column Subquery</u>

Returns more than one column, used with tuple comparisons.

Example:

```
SELECT *
FROM employees
WHERE (department_id, job_id) IN (
  SELECT department_id, job_id
  FROM job_openings
);
```

Explanation:

- Returns employees whose department_id and job_id match open positions.

4. Correlated Subquery

This subquery **depends on** the outer query — it runs **once for every row** of the outer query.

Example:

```
SELECT name
FROM students s
WHERE marks > (
  SELECT AVG(marks)
  FROM students
  WHERE dept_id = s.dept_id
);
```

Explanation:

- For each student, the subquery calculates the average of **their department**.
- Selects students who scored **above the department average**.

Tip: Correlated subqueries are powerful but slower — prefer JOINs when possible.

Where Can You Use Subqueries?

1. In the WHERE clause – for filtering.
2. In FROM clause – as derived tables.
3. In the SELECT clause – for computed columns.

Example: **Subquery in FROM Clause**

```
SELECT dept_id, avg_marks
FROM (
  SELECT dept_id, AVG(marks) AS avg_marks
  FROM students
  GROUP BY dept_id
) AS dept_avg
WHERE avg_marks > 75;
```

Explanation:

- The subquery calculates average marks per department.
- Outer query filters departments with average > 75.

Example: Subquery in SELECT Clause

```
SELECT name,
    (SELECT COUNT(*) FROM enrollments WHERE student_id = s.id) AS course_count
FROM students s;
```

Explanation:

- Returns student names with the number of courses they are enrolled in.
- The subquery is executed per row (can be expensive for large data).

Subquery vs Join (When to Use)

Use Subquery:
- Filtering based on aggregates
- Simpler for nested logic

Use Join:
- Retrieving related data
- Better performance with large data and indexing

Set Operations in SQL

Set operations in SQL allow you to combine results from two or more SELECT queries. The basic set operations are UNION, INTERSECT, and MINUS (or EXCEPT, depending on the SQL dialect).

1. UNION

Purpose:

Combines the results of two SELECT queries and **removes duplicate rows**.**Example**:

```
SELECT student_id FROM course_math
UNION
SELECT student_id FROM course_science;
```

Explanation:

- Returns all **unique** student_ids from both tables.
- If a student is enrolled in both courses, they appear only once.

Note:

Columns in both SELECTs **must match** in number and data type.

1A. UNION ALL

Purpose:

Combines results from two SELECT queries without removing duplicates.

Example:

```
SELECT student_id FROM course_math
UNION ALL
SELECT student_id FROM course_science;
```

Explanation:

- Includes **all** rows from both queries.
- If duplicates exist, they are **not removed**.

Performance Tip:

UNION ALL is **faster** than UNION because it skips the duplicate check.

2. INTERSECT

Purpose:

Returns only the common rows present in both SELECT queries.

Example:

```
SELECT student_id FROM course_math
INTERSECT
SELECT student_id FROM course_science;
```

Explanation:

- Only students **enrolled in both** courses are returned.
- No duplicates.

Support Note:

- INTERSECT is **not supported in MySQL** (use JOIN instead).

- Available in Oracle, PostgreSQL, and SQL Server.

3. MINUS / EXCEPT

Purpose:

Returns rows from the first SELECT that do not exist in the second SELECT.

```
SELECT student_id FROM course_math
EXCEPT
SELECT student_id FROM course_science;
```

Explanation:

- Returns students in **math but not in science**.

Set Operation Summary Table

Operation	Returns	Removes Duplicates
UNION	All unique rows from both SELECTs	Yes
UNION ALL	All rows from both SELECTs	No
INTERSECT	Common rows from both SELECTs	Yes
MINUS	Rows from 1st query not in 2nd (Oracle)	Yes
EXCEPT	Rows from 1st query not in 2nd (PG/SQLServer)	Yes

Best Practices

- All SELECT statements in a set operation **must have:**
  - Same number of columns.
  - Compatible data types.
- Use **ORDER BY** only in the final query.
- Always verify performance for large datasets (UNION can be expensive).

Query Optimization Basics in SQL

Query optimization improves the performance of SQL queries by reducing their execution time and resource consumption. This is essential in large databases with high traffic.

1. Indexes

What is an Index?

An **index** is a data structure (usually a B-tree or bitmap) that allows the database to find rows faster, just like an index in a book.

How They Work:

Instead of scanning the entire table (full table scan), the database jumps directly to the relevant rows using the index.

Common Types of Indexes

Type	Use Case
B-tree Index	Default; great for range or equality queries
Bitmap Index	Useful for columns with few distinct values
Composite Index	Covers multiple columns in queries
Unique Index	Enforces uniqueness of column values

Example:
CREATE INDEX idx_emp_dept ON employees(department_id);

Speeds up queries filtering by department_id.

When Indexes Help:

- Queries with **WHERE**, **JOIN**, **ORDER BY**, or **GROUP BY** clauses.
- Large tables where **read speed** is critical.
- Columns are frequently used in filters or lookups.

When Indexes Hurt:

- **Too many indexes** slow down INSERT/UPDATE/DELETE operations.
- **Low-selectivity columns** (like gender or boolean).
- Unused indexes waste space and slow down writes.

2. Execution Plans

What is an Execution Plan?

An execution plan shows how the SQL engine will execute your query — the steps it will take, which indexes it will use, and the estimated cost of each step.

How to View Execution Plans

SQL Engine	Command
MySQL	EXPLAIN SELECT ...
Oracle	EXPLAIN PLAN FOR SELECT ...
SQL Server	Estimated/Actual Plan options

Example:

EXPLAIN SELECT * FROM employees WHERE department_id = 10;

Shows whether an index is being used or not.

Common Terms in Plans

Term	Meaning
Table Scan	Full scan of a table — slow for large tables
Index Scan	Reads from an index — much faster
Nested Loops	For small joins, it can be slow if the input is large.

Term	Meaning
Hash Join	Efficient for large joins without indexes
Cost	Estimated resource usage — lower is better.

3. General Query Optimization Tips

- **Use only needed columns:** Avoid SELECT *.
- **Filter rows early:** Use appropriate WHERE clauses.
- **Avoid functions on indexed columns:** e.g., avoid WHERE UPPER(name) = 'JOHN'.
- **Prefer JOINs over subqueries** when dealing with large datasets.
- **Use EXISTS** instead of IN for correlated subqueries.
- **Monitor query performance** using the execution plan.

Practice: Solve Complex SQL Queries

Platforms: Geeks for Geeks | Hacker Rank | LeetCode

Recommended Practice Platforms

1. **Geeks for Geeks** (GFG)

- **Topics Covered:** Basic to advanced SQL (joins, subqueries, views, triggers)
- **Features:**
  - Practice problem sets with explanations.
  - Beginner-friendly.
- GFG SQL Practice

2. **Hacker Rank** – SQL Track

- **Topics:** Basic SELECT, Advanced SELECT, Aggregations, JOINs, Window Functions.
- **Strengths:**
  - Real-world datasets.
  - Problems ranked by difficulty.
  - Instant feedback.
- Hacker Rank SQL Practice

3. **LeetCode** – Database Section

- **Topics:** Joins, nested queries, GROUP BY, CTEs, window functions.

- **Advanced Features:**
  - Top-level real interview questions.
  - SQL + DB schema provided.
- LeetCode SQL Problems

Suggested Practice Flow

Skill Level	Focus Areas	Example Problem Types
Beginner	SELECT, WHERE, ORDER BY	Basic filtering, sorting
Intermediate	JOINs, GROUP BY, Subqueries	Aggregations, filtering across tables
Advanced	CTEs, Window Functions, Optimization	Rankings, running totals, partitions

Pro Tips for Practice

- Use **EXPLAIN** to see query plans.
- Avoid SELECT *; select only needed columns.
- Practice writing queries without relying on GUI tools.
- Focus on **readability** and **modularity** with CTEs.