**ScopeTreasure Courses**

SQL COURSES · LESSON DB.8 · CODE SCOPE SERIES

# Subqueries — Sawal Ke Andar Sawal

A question inside a question: subqueries in WHERE, in the SELECT list, and with IN — above-average students without knowing the average.

## Introduction

"Show me the students who scored **above average**." One sentence, two questions: *what is the average?* and *who is above it?* Without subqueries you would run the first query yourself, read 83 off the screen, and type it into the second query by hand — slow, and wrong the moment the data changes. A **subquery** writes the first question *inside* the second: SQL runs the inner query, takes its answer, and finishes the outer one. The question answers itself, fresh, every time.

Our five students' marks are 88, 92, 76, 95, 64 — average exactly **83**. Watch how each example uses that 83 without anyone typing it.

*(Roman Urdu: Subquery = sawal ke andar chhota sawal. Pehle andar wala chalta hai, us ka jawab bahir wale ko milta hai.)*

## Key points

- **A subquery is a SELECT inside brackets:** `(SELECT AVG(marks) FROM students)` — it runs first and hands its result to the outer query, which then behaves as if that value had been typed there.
- **In WHERE — filter by a computed value:** `WHERE marks > (SELECT AVG(marks) FROM students)`. Perfect for above/below average, bigger than the largest order, cheaper than the cheapest rival — any comparison against a fact you do not know in advance.
- **A WHERE subquery that returns one value** works with `=`, `>`, `<`. If it might return *several* values, use **IN**: `WHERE id IN (SELECT student_id FROM fees WHERE status = 'Paid')` — "id is any of these".
- **In the SELECT list — an extra computed column:** `(SELECT AVG(marks) FROM students) AS class_average` adds the same computed value to every row — each student shown *next to* the average he is compared against. Such a subquery must return exactly one value per row.

  *(Roman Urdu: SELECT wali subquery har row ke saath ek hisaab-shuda qeemat chipka deti hai.)*
- **The inner query can use another table:** that is the real power — the outer query asks about students, the inner one digs in fees. Two tables, one sentence, no JOIN syntax (DB.5's joins and subqueries often answer the same question; joins usually read better for wide reports, subqueries for filter-by-a-fact).
- **Read subqueries inside-out:** when a query confuses you, read the bracketed part first, work out what single value (or list) it produces, then read the outer query with that answer in hand.

## Code examples

### Example 1 — WHERE subquery: above the average

```
SELECT name, marks FROM students
WHERE marks > (SELECT AVG(marks) FROM students);
```

What it does: The inner query computes the average of all marks: (88 + 92 + 76 + 95 + 64) / 5 = **83**. The outer query then keeps students with marks above 83. Notice nobody typed 83 — if a sixth student joins tomorrow, the average moves and this same query still answers correctly.

**What you will see:**

```
+-----------+-------+
| name      | marks |
+-----------+-------+
| Ali Raza  |    88 |
| Sara Khan |    92 |
| Hira Shah |    95 |
+-----------+-------+
3 rows in set
```

Three students above the line — Ali, Sara, Hira. Bilal (76) and Usman (64) sit below it. Check by hand against the DB.1 table; the subquery's hidden arithmetic matches exactly.

### Example 2 — SELECT-list subquery: everyone, next to the average

```
SELECT name, marks,
  (SELECT AVG(marks) FROM students) AS class_average
FROM students;
```

What it does: Instead of filtering, this query shows *all five* students and attaches the class average (83) as an extra column on every row — a report that carries its own comparison line. MySQL prints averages with decimals; the grid rounds the display to 83.00.

**What you will see:**

```
+-------------+-------+---------------+
| name        | marks | class_average |
+-------------+-------+---------------+
| Ali Raza    |    88 |         83.00 |
| Sara Khan   |    92 |         83.00 |
| Bilal Ahmed |    76 |         83.00 |
| Hira Shah   |    95 |         83.00 |
| Usman Tariq |    64 |         83.00 |
+-------------+-------+---------------+
5 rows in set
```

Read across any row: the student's own marks beside the class's 83 — who is above and who is below is visible at a glance. Same inner query as Example 1, different job: there it *filtered*, here it *informs*.

**Example 3 — IN subquery: students whose fee is paid**

```
SELECT name FROM students
WHERE id IN (SELECT student_id FROM fees WHERE status = 'Paid');
```

What it does: The inner query digs in the `fees` table (DB.5's September data) and returns the student_ids with a Paid fee: 1, 2, 3. The outer query keeps students whose id is in that list. Two tables, one question, no JOIN written.

**What you will see:**

```
+-------------+
| name        |
+-------------+
| Ali Raza    |
| Sara Khan   |
| Bilal Ahmed |
+-------------+
3 rows in set
```

Usman (fee Pending) and Hira (no fee row at all) are both absent — the inner list simply never contained their ids. Compare with DB.5: the LEFT JOIN showed everyone *with* their status; this subquery answers the narrower question *who has paid* — pick the tool that matches the question.

## Quick recap

- A **subquery** is a bracketed SELECT whose answer feeds the outer query — computed fresh on every run.
- **In WHERE** it filters by an unknown-in-advance fact (above average); single values take `=`, `>`, `<`, lists take **IN**.
- **In the SELECT list** it attaches one computed value to every row.

- Read inside-out: inner answer first, then the outer question.

## Common mistakes

- **Hard-coding the inner answer:** typing `WHERE marks > 83` because you computed the average once. The data changes; your typed 83 does not. If the filter is a fact about the data, let a subquery fetch it.
- **Using `=` with a multi-row subquery:** `WHERE id = (SELECT student_id FROM fees)` fails — the inner query returns many values and `=` expects one. Many values means IN.

  *(Roman Urdu: Ek jawab par =, bohat jawabon par IN — ye jodi yaad rakho.)*
- **A SELECT-list subquery returning many rows:** the extra column can hold only one value per row. If the inner query can return several, the query errors — aggregate it (AVG, MAX, COUNT) or move it to WHERE with IN.
- **Burying a simple question:** not every two-part question needs a subquery — "students from Sahiwal with marks above 80" is plain WHERE with AND. Subqueries earn their brackets when the filter is itself a question about the data.

## Practice

1. Write the "below average" version of Example 1, and predict the two names before running it.
2. Using a subquery, find the name of the single student with the highest marks. (Hint: the inner query is a MAX.) Then do the same for the lowest.
3. Write the IN-subquery that lists students who have **no** fee row at all. (Hint: NOT IN with the fees table's student_id list — whose name should appear?)

## Practice quiz

**1. Which query shows students above the class average?**

- SELECT * FROM students WHERE marks > 83;
- SELECT * FROM students WHERE marks > (SELECT AVG(marks) FROM students);
- SELECT * FROM students WHERE marks = AVG(marks);
- SELECT AVG(marks) FROM students WHERE marks > AVG(marks);

**Answer:** SELECT * FROM students WHERE marks > (SELECT AVG(marks) FROM students);

**2. A subquery may return several values, like student ids from fees. Which comparison do you use?**

- =, it handles many values
- IN, as in WHERE id IN (SELECT student_id FROM fees WHERE status = 'Paid')
- >
- LIKE

**Answer:** IN, as in WHERE id IN (SELECT student_id FROM fees WHERE status = 'Paid')

**3. What must a subquery in the SELECT list return?**

- As many rows as it likes
- Exactly one value per row, like (SELECT AVG(marks) FROM students) AS class_average
- A whole table
- Nothing, it only filters

**Answer:** Exactly one value per row, like (SELECT AVG(marks) FROM students) AS class_average

---

**Watch the Code Scope channel:** https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

**Read online:** https://scopetreasure.com/notes/sql-subqueries · More free tools and notes at https://scopetreasure.com