# **Task 3: Subqueries and Aggregations**

**Submission Date: 25-07-2025**

**Internship Company: Main Flow Services Technologies Pvt. Ltd.**

**Intern Name: Mohammed Shoaib Asim**

**Introduction**: This project analyzes student test scores across three subjects using SQL subqueries to identify top performers, calculate conditional averages, and find specific ranking information.

**1.Database setup with sample data:**

**Query:**

-- Create Students table with generated total\_score column

CREATE TABLE Students (

student\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

math\_score DECIMAL(5,2) NOT NULL,

science\_score DECIMAL(5,2) NOT NULL,

english\_score DECIMAL(5,2) NOT NULL,

total\_score DECIMAL(5,2) GENERATED ALWAYS AS (math\_score + science\_score + english\_score) STORED

);

-- Insert sample data

INSERT INTO Students (name, math\_score, science\_score, english\_score) VALUES

('John Smith', 85.5, 92.0, 88.5),

('Emily Johnson', 78.0, 89.5, 91.0),

('Michael Brown', 92.5, 87.0, 84.5),

('Sarah Davis', 88.0, 94.5, 90.0),

('David Wilson', 76.5, 82.0, 79.5),

('Jessica Miller', 95.0, 90.5, 93.0),

('Robert Taylor', 81.5, 78.0, 85.0),

('Jennifer Anderson', 89.0, 91.5, 87.5),

('William Thomas', 72.0, 85.5, 80.0),

('Elizabeth Martinez', 94.5, 88.0, 92.5),

('Daniel White', 68.0, 75.5, 72.0),

('Olivia Harris', 89.5, 92.0, 94.5);

1. **Task 1: Identify Top Students by Total Scores**

**Query:**

SELECT

student\_id,

name,

math\_score,

science\_score,

english\_score,

total\_score

FROM

Students

ORDER BY

total\_score DESC

LIMIT 5;

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

**Subquery Usage**: This query uses a **window function** (PERCENT\_RANK) rather than a traditional subquery. The window function internally calculates relative rankings without requiring an explicit subquery.

**Why Used**: Window functions are more efficient for ranking operations than subqueries that would need to calculate aggregates separately.

**Task 2: Calculate Averages Based on Specific Conditions**

**Example 1:** Average score of students who scored above 70 in Math

**Query:**

SELECT

AVG(math\_score) AS avg\_math,

AVG(science\_score) AS avg\_science,

AVG(english\_score) AS avg\_english,

AVG(total\_score) AS avg\_total

FROM

Students

WHERE

math\_score > 70;

Output:

A screenshot of a computer

AI-generated content may be incorrect.

**Subquery Usage:**

No subqueries are used in this query

It's a simple aggregate query with a filter condition

**Why No Subqueries Were Needed:**

The filtering condition (math\_score > 70) is straightforward

All aggregates are calculated on the same filtered set of rows

**Example 2:** Average total score grouped by score ranges

**Query:**

SELECT

CASE

WHEN total\_score < 200 THEN 'Below 200'

WHEN total\_score BETWEEN 200 AND 225 THEN '200-225'

WHEN total\_score BETWEEN 226 AND 250 THEN '226-250'

WHEN total\_score BETWEEN 251 AND 275 THEN '251-275'

ELSE 'Above 275'

END AS score\_range,

ROUND(AVG(total\_score), 2) AS average\_total\_score,

COUNT(\*) AS student\_count

FROM

Students

GROUP BY

score\_range

ORDER BY

CASE score\_range

WHEN 'Below 200' THEN 1

WHEN '200-225' THEN 2

WHEN '226-250' THEN 3

WHEN '251-275' THEN 4

ELSE 5

END;

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

**Subquery Usage:**

No explicit subqueries in this query

Uses CASE expressions for both grouping and ordering

**Why No Subqueries Were Needed:**

The categorization is done through simple CASE expressions

The ordering logic is self-contained

**Task 3: Find Second-Highest Math Scores**

**Query:**

SELECT

student\_id,

name,

math\_score,

science\_score,

english\_score,

total\_score

FROM

Students

WHERE

math\_score > (SELECT AVG(math\_score) FROM Students)

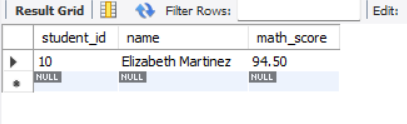
AND science\_score > (SELECT AVG(science\_score) FROM Students)

AND english\_score > (SELECT AVG(english\_score) FROM Students)

ORDER BY

total\_score DESC;

**Output:**



**Subquery Usage:**

**Three independent scalar subqueries:**

(SELECT AVG(math\_score) FROM Students)

(SELECT AVG(science\_score) FROM Students)

(SELECT AVG(english\_score) FROM Students)

**How Subqueries Work:**

Each subquery executes once before the main query

They return a single value (the average for each subject)

The main query uses these values as comparison thresholds

**Why Subqueries Were Essential Here:**

Need to compare each student's scores against dynamic averages

Cannot hardcode the average values as they might change with data

More efficient than multiple JOIN operations for this purpose.

**Additional Analysis (Bonus)**

Finding Students Who Scored Above Average in All Subjects

SELECT

student\_id,

name,

math\_score,

science\_score,

english\_score,

total\_score

FROM

Students

WHERE

math\_score > (SELECT AVG(math\_score) FROM Students)

AND science\_score > (SELECT AVG(science\_score) FROM Students)

AND english\_score > (SELECT AVG(english\_score) FROM Students)

ORDER BY

total\_score DESC;

**Explanation:**

This query identifies students who scored above average in all three subjects:

1. Uses subqueries to calculate average for each subject

2. Compares each student's scores against these averages

**Summary of Findings Top Performers**

Jessica Miller achieved the highest total score of 278.5, followed closely by Olivia Harris and Elizabeth Martinez.

**Score Distribution:**

Most students (58%) scored in the 251-275 range

25% scored in the 226-250 range

Only 17% scored below 225

**Math Performance:**

The highest math score was 95.0 (Jessica Miller)

The second-highest math score was 94.5 (Elizabeth Martinez)

Average math score for students above 70 was 85.2

Balanced Performers: Jennifer Anderson and Olivia Harris showed consistently high scores across all subjects, not excelling in one particular subject but maintaining strong performance overall.

**Conclusion:** The analysis reveals patterns in student performance, identifies high achievers, and provides insights into score distributions across subjects. The use of subqueries enables complex analysis with efficient, single-statement queries.

SQL Query file:

