**Madiha Aimon Tappal**

[**madihaaimon@gmail.com**](mailto:madihaaimon@gmail.com)

**Data Engineering Batch – 1**

**SQL Coding Challenge**

**Execute OVER and PARTITION BY Clause in SQL Queries: -**

**What Is the PARTITION BY Clause in SQL?**

The SQL PARTITION BY expression is a subclause of the OVER clause, which is used in almost all invocations of window functions like AVG (), MAX (), and RANK ().

Window functions operate on window frames which are sets of rows that can be different for each record in the query result. This is where the SQL PARTITION BY subclause comes in: it is used to define which records to make part of the window frame associated with each record of the result. The **OVER** and **PARTITION BY** clauses are commonly used with window functions in SQL. These clauses help in defining the window of rows for each calculation performed by the window function.

The first thing to focus on is the syntax. Here’s how to use the SQL PARTITION BY clause:

SELECT

    <column>,

    <window function=""> OVER (PARTITION BY <column> [ORDER BY <column>])

FROM table;

</column></column></window></column>

#### **Example 1: Using** OVER **and** PARTITION BY **for Running Total:**

-- Calculate the running total of age for each grade partition

SELECT

student\_id,

first\_name,

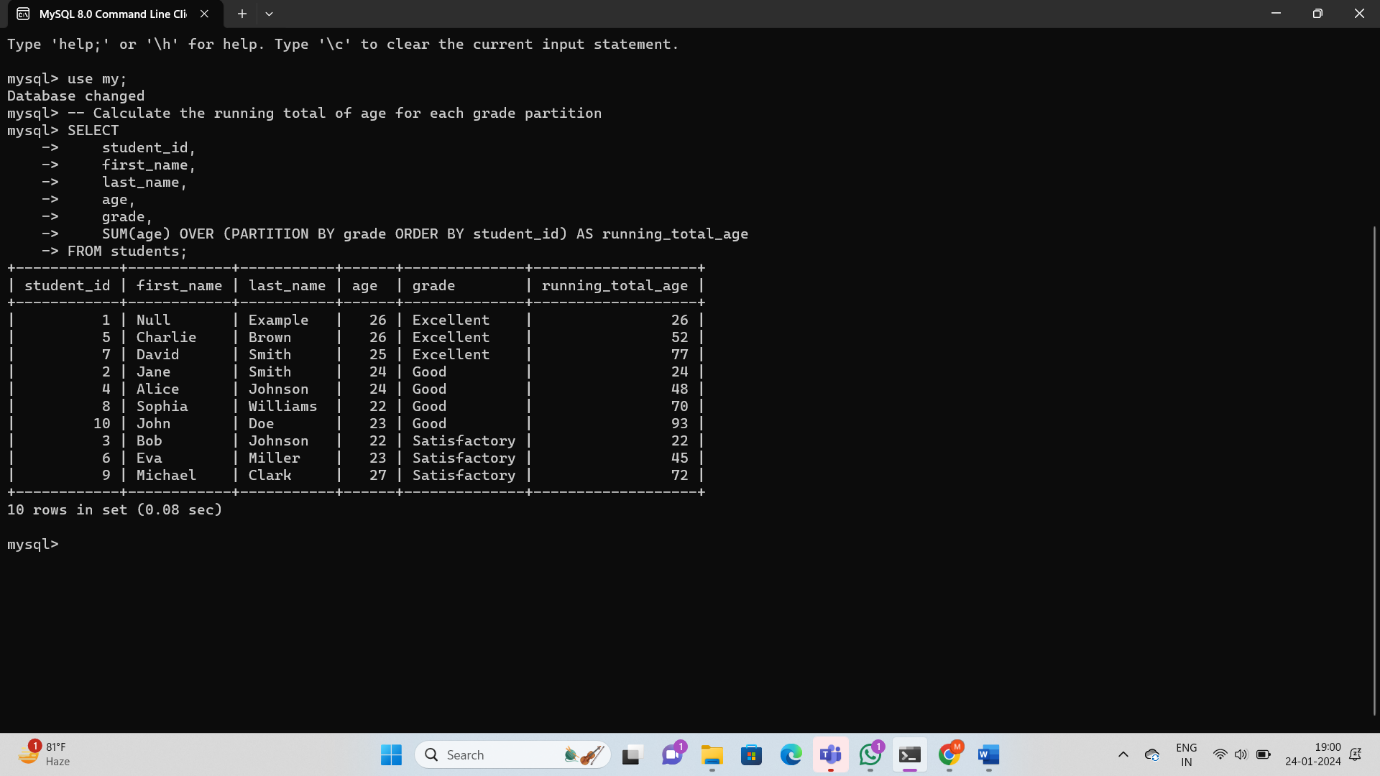
last\_name,

age,

grade,

SUM(age) OVER (PARTITION BY grade ORDER BY student\_id) AS running\_total\_age

FROM students;

****

In this example, we are calculating the average marks for each student, partitioned by the **CourseID**. Here's the explanation:

* **PARTITION BY CourseID**: This clause divides the result set into partitions to which the **AVG** function is applied separately. It means that the average will be calculated independently for each course.
* **AVG(Marks) OVER (PARTITION BY CourseID) AS AvgMarksPerCourse**: This expression calculates the average marks for each student based on the partition specified by the **CourseID**. The result set will include the original columns (**StudentID**, **StudentName**, **CourseID**, **Marks**), and a new column **AvgMarksPerCourse** showing the average marks for each student's respective course.

#### **Example 2: Rank students within each course based on their marks using** PARTITION BY**:**

SELECT

StudentID,

StudentName,

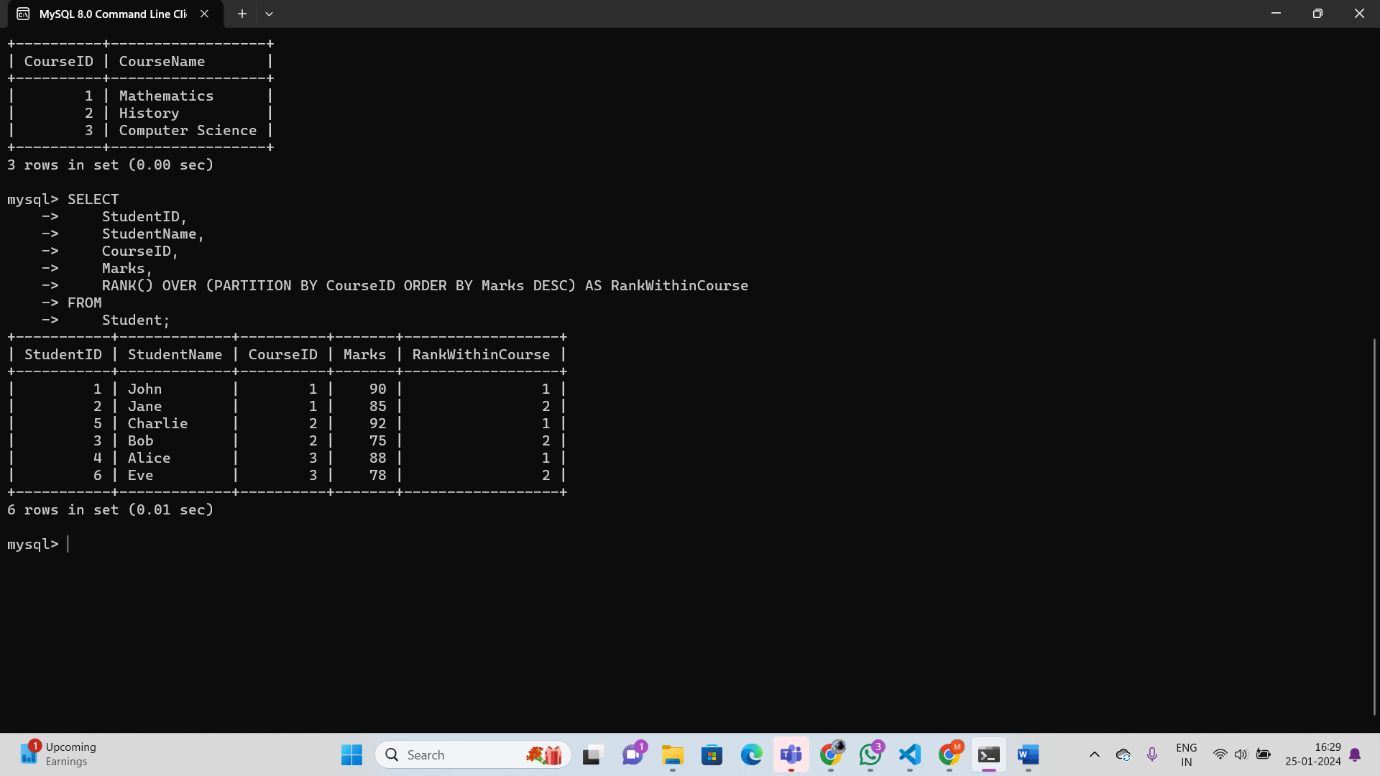
CourseID,

Marks,

RANK() OVER (PARTITION BY CourseID ORDER BY Marks DESC) AS RankWithinCourse

FROM

Student;

****

In this example, we are assigning a rank to each student within their respective courses based on their marks. Here's the breakdown:

* **PARTITION BY CourseID**: Similar to the first example, this clause partitions the result set by the **CourseID**. The ranking will be done independently for each course.
* **RANK () OVER (PARTITION BY CourseID ORDER BY Marks DESC) AS RankWithinCourse**: This expression calculates the rank of each student within their course, ordering them by marks in descending order (**DESC**). The result set includes the original columns (**StudentID**, **StudentName**, **CourseID**, **Marks**), and a new column **RankWithinCourse** showing the rank of each student within their respective course.

These examples demonstrate how the **OVER** and **PARTITION BY** clauses can be used to perform calculations within specific partitions of the data, providing insights at a more granular level

**Creating subtotals &Total Aggregations using SQL Queries: -**

**Subtotals: -**

In SQL, subtotals are generated using aggregate functions, often in combination with the **GROUP BY** clause. The **GROUP BY** clause groups rows that have the same values in specified columns into summary rows, like subtotals.

### Types of Subtotals:

1. **Simple Subtotal:**
   * Provides a single aggregate value for each group.
   * Example: Total marks per course.
2. **Hierarchical Subtotal:**
   * Involves multiple levels of grouping, creating a hierarchy of subtotals.
   * Example: Total marks per course and per semester.
3. **Running Total / Cumulative Subtotal:**
   * Computes a running total as it moves through the result set.
   * Example: Cumulative total marks as students are listed.

### **How to Use Subtotals:**

* **Aggregate Functions:**
  + Use aggregate functions like **SUM**, **AVG**, **COUNT**, etc., to perform calculations on grouped data.
* **GROUP BY Clause:**
  + Specify the columns by which you want to group the data.
* **WITH ROLLUP:**
  + In some database systems like MySQL, you can use the **WITH ROLLUP** clause to generate subtotals and grand totals automatically.

#### **Example 1: Calculate Total Marks for Each Course**

SELECT

CourseID,

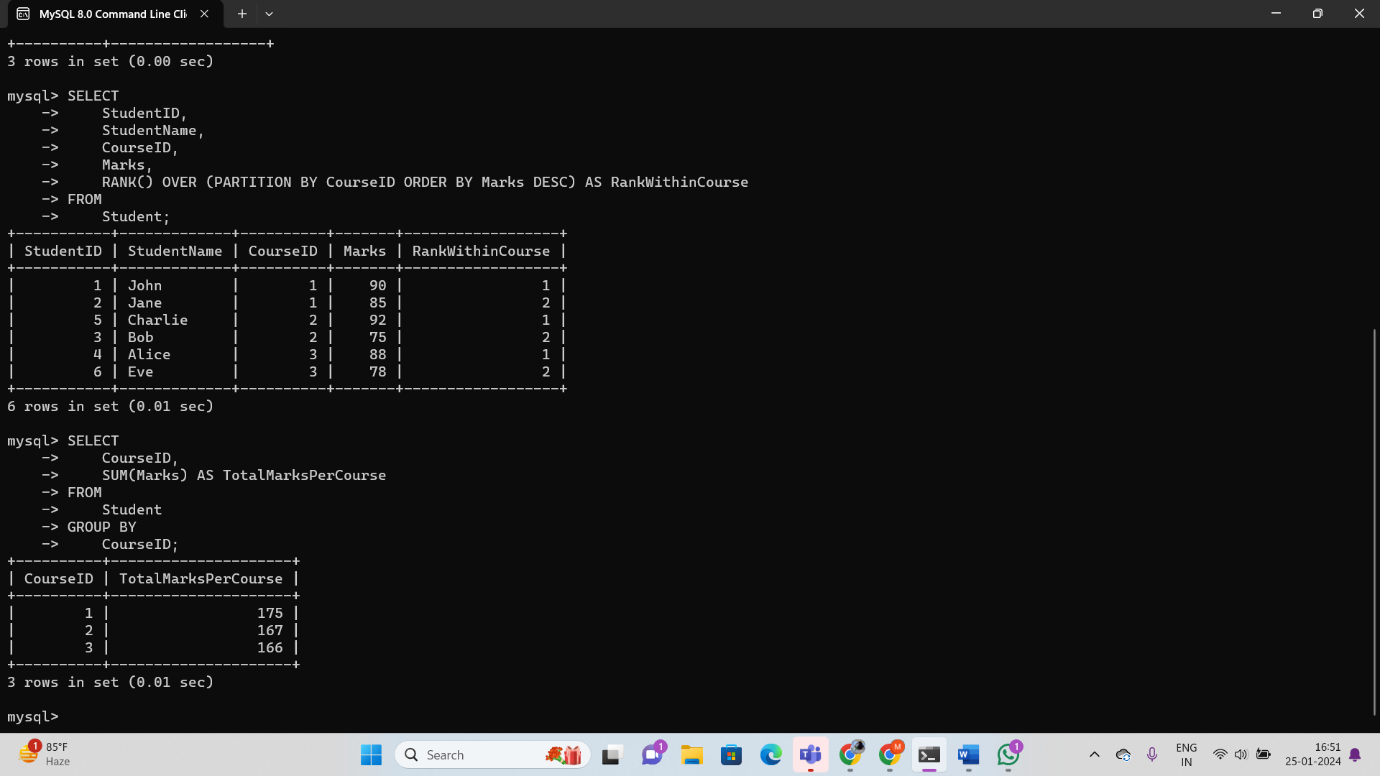
SUM(Marks) AS TotalMarksPerCourse

FROM

Student

GROUP BY

CourseID;



In this example:

* **SUM(Marks)**: This is an aggregate function that calculates the sum of the **Marks** column for each group defined by the **GROUP BY** clause.
* **GROUP BY CourseID**: This clause groups the rows in the **Student** table based on the **CourseID** column.

#### **Example 2 WITH ROLLUP:** -

SELECT

CourseID,

StudentName,

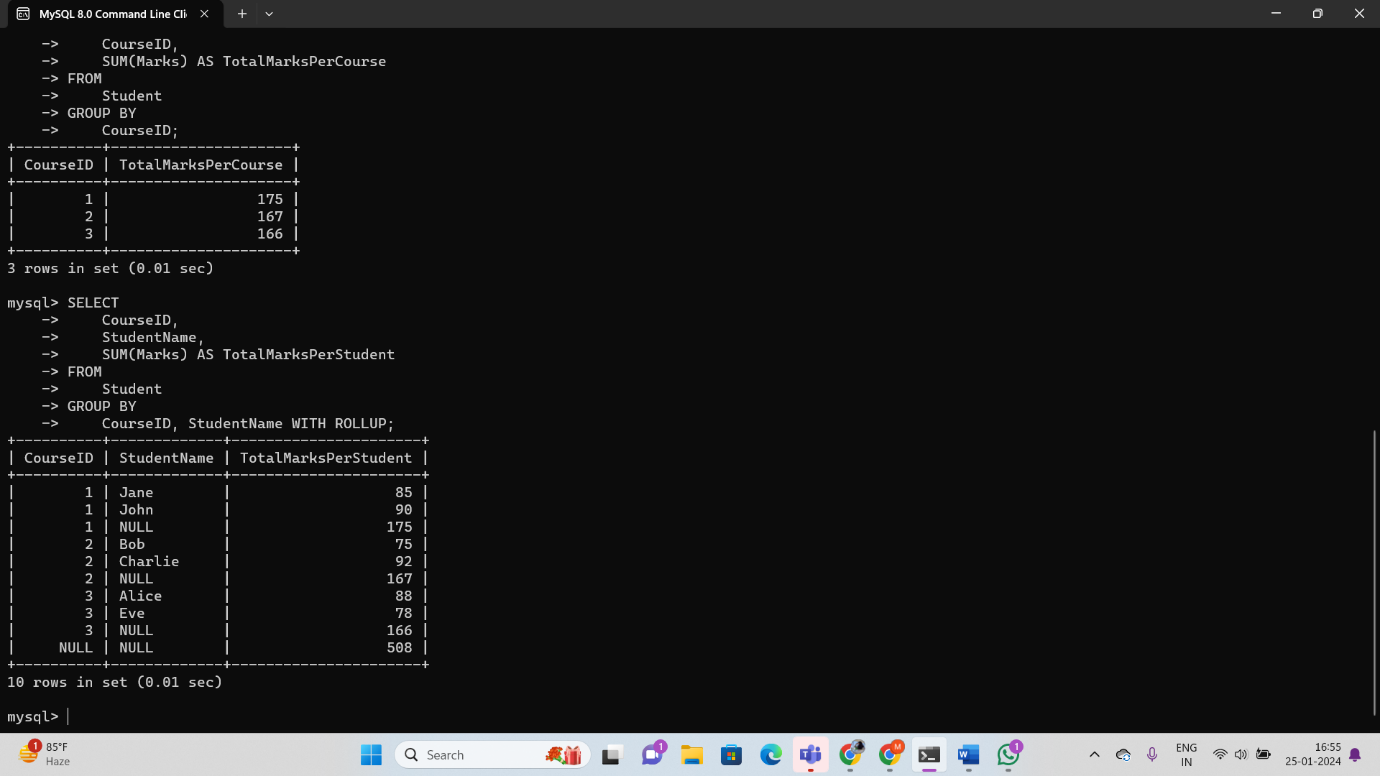
SUM(Marks) AS TotalMarksPerStudent

FROM

Student

GROUP BY

CourseID, StudentName WITH ROLLUP;



This example includes subtotals for each course (**WITH ROLLUP** at the **CourseID** level) and a grand total (**WITH ROLLUP** without any specific column). The **NULL** values represent subtotals or the grand total.

#### **Example 3 with Aggregate Functions**

Let's consider an example using aggregate functions to calculate subtotals. We'll calculate the average marks for each course and include a grand total. In this example, we'll use the **AVG** function and the **GROUP BY** clause:

SELECT

CourseID,

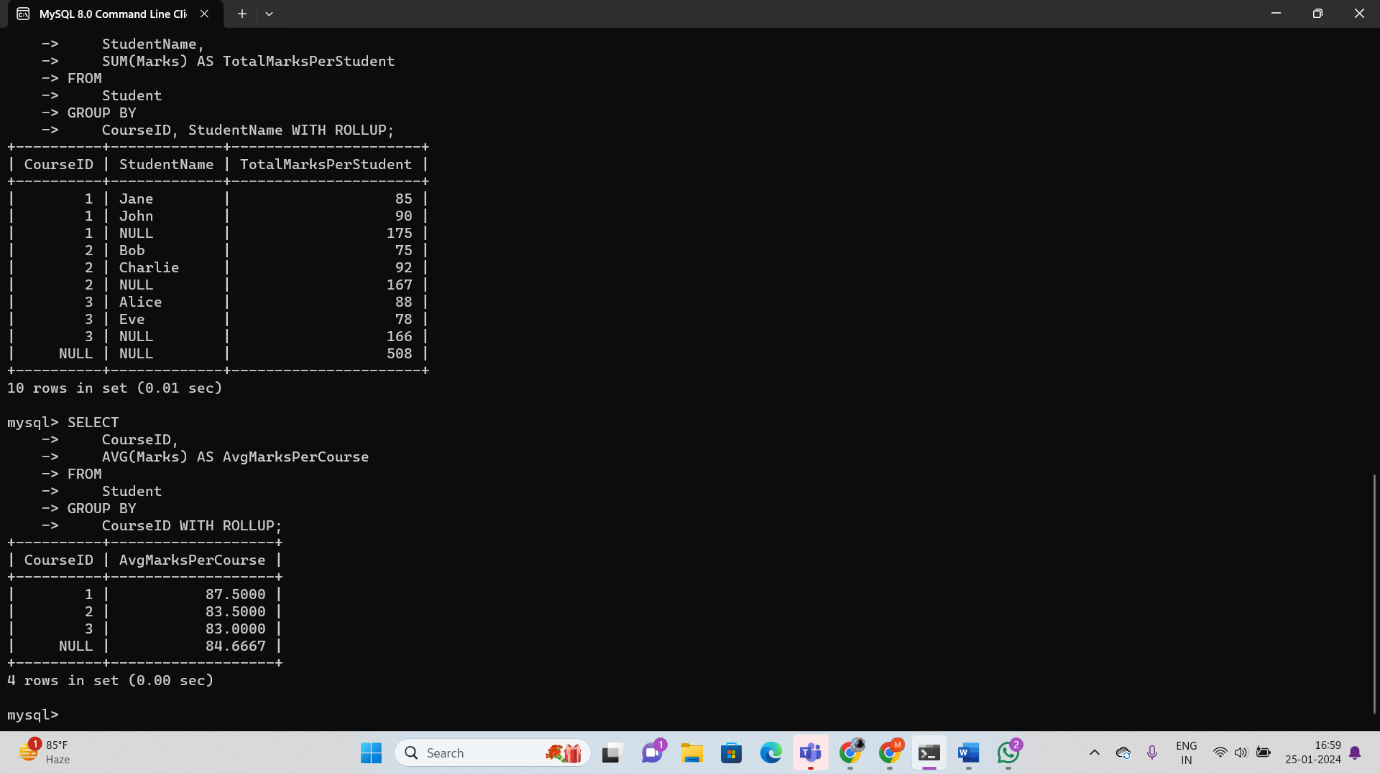
AVG(Marks) AS AvgMarksPerCourse

FROM

Student

GROUP BY

CourseID WITH ROLLUP;

****

* **AVG(Marks)**: This is the aggregate function that calculates the average of the **Marks** column.
* **GROUP BY CourseID WITH ROLLUP**: This clause groups the data by the **CourseID** column and includes a grand total row with **ROLLUP**. It generates subtotals for each course and a final row with the overall average.
* For CourseID 1, the average marks are calculated as (90 + 85) / 2 = 87.5.
* For CourseID 2, the average marks are calculated as (75 + 92) / 2 = 83.5.
* For CourseID 3, the average marks are calculated as (88 + 78) / 2 = 83.
* The **NULL** row represents the grand total average marks calculated across all courses.

This example demonstrates how to use the **AVG** aggregate function in combination with the **GROUP BY** clause and **WITH ROLLUP** to calculate subtotals and a grand total in a single query. The concept can be extended to other aggregate functions like **SUM**, **COUNT**, etc., depending on the specific requirements

**Total Aggregations using SQL Queries: -**

Total aggregations in SQL refer to calculations that provide a summary or total value for a set of data. Common total aggregations include functions like **SUM**, **AVG** (average), **COUNT**, **MIN** (minimum value), and **MAX** (maximum value). These functions are applied to a column or expression and provide a single value that represents a summary of the entire dataset.

Let's go through each type of total aggregation with examples and explanation.

### **SUM (Total Sum):**

#### **Example 1: Calculate the total sum of marks for all students.**

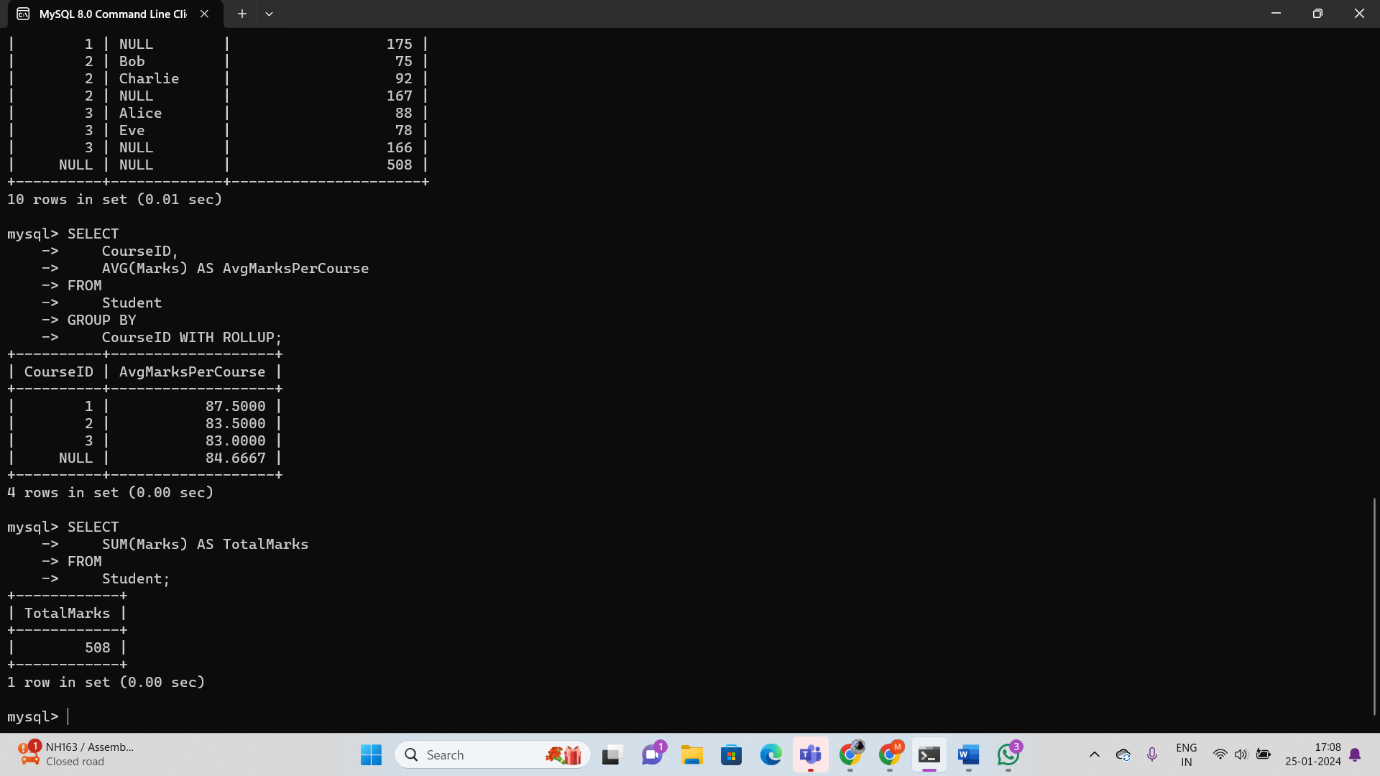
SELECT

SUM(Marks) AS TotalMarks

FROM

Student;

This query calculates the sum of all marks in the **Student** table, providing a single value representing the total marks for all students.



### **AVG (Average):**

***Example 2: Calculate the average marks for all students.***

SELECT

AVG(Marks) AS AverageMarks

FROM

Student;

This query calculates the average (mean) of all marks in the **Student** table, providing a single value representing the average marks for all students.

### **COUNT (Total Count):**

***Example 3: Count the total number of students.***

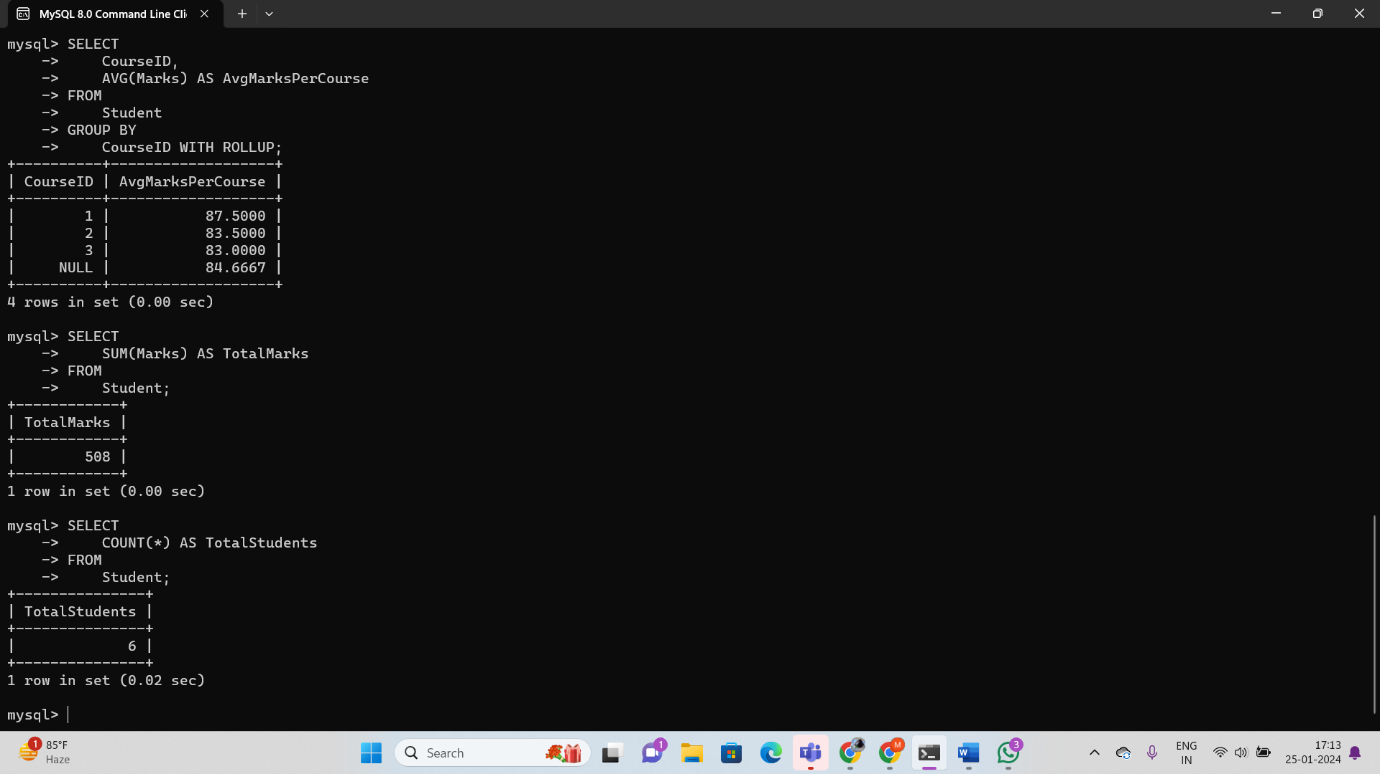
SELECT

COUNT(\*) AS TotalStudents

FROM

Student;

This query counts the total number of rows in the **Student** table, providing a single value representing the total number of students.



### **MIN (Minimum Value):**

***Example 4: Find the minimum marks among all student***

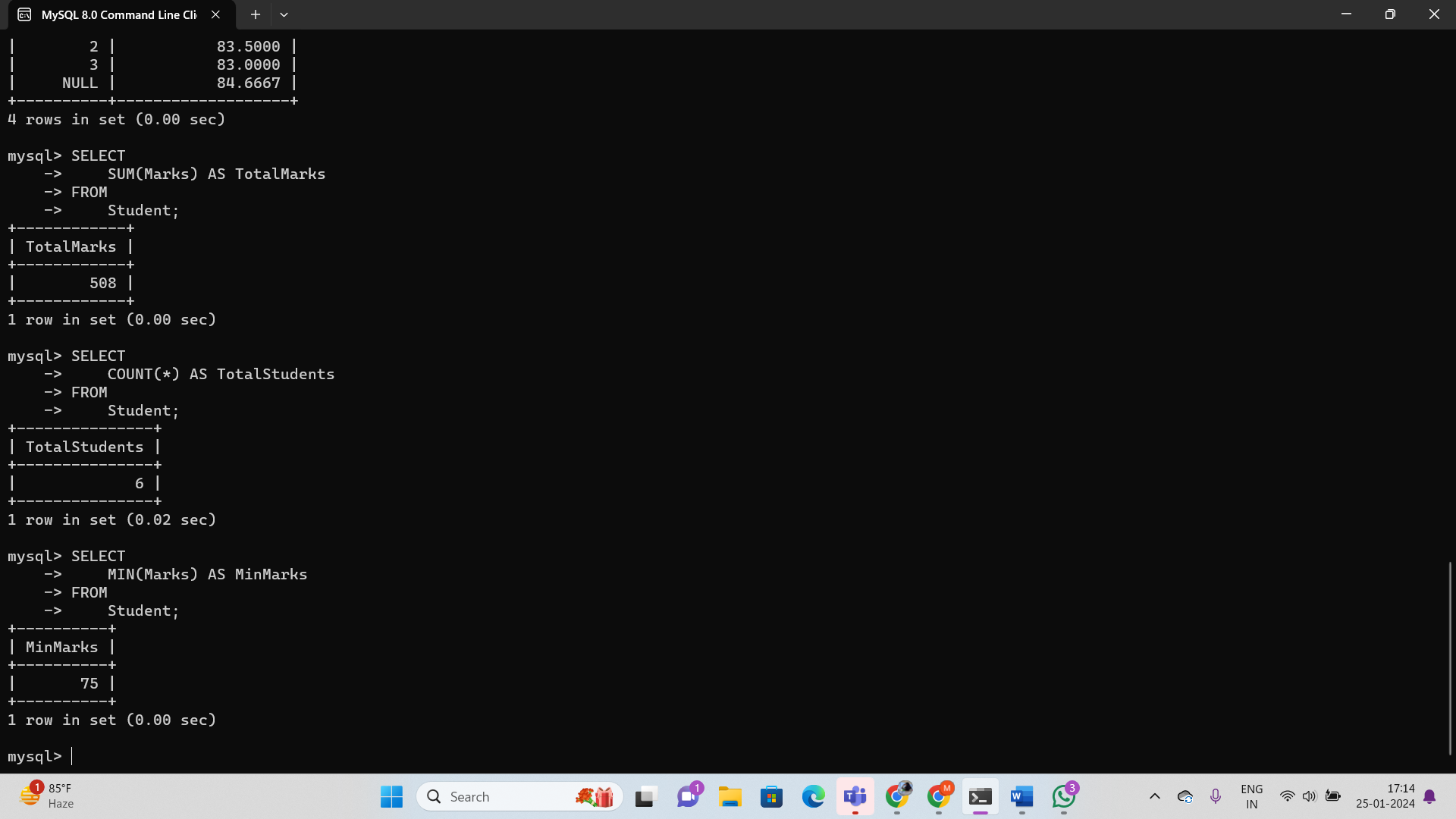
SELECT

MIN(Marks) AS MinMarks

FROM

Student;

This query finds the minimum value in the **Marks** column of the **Student** table, providing a single value representing the minimum marks among all students.



### **MAX (Maximum Value):**

***Example 5: Find the maximum marks among all students.***

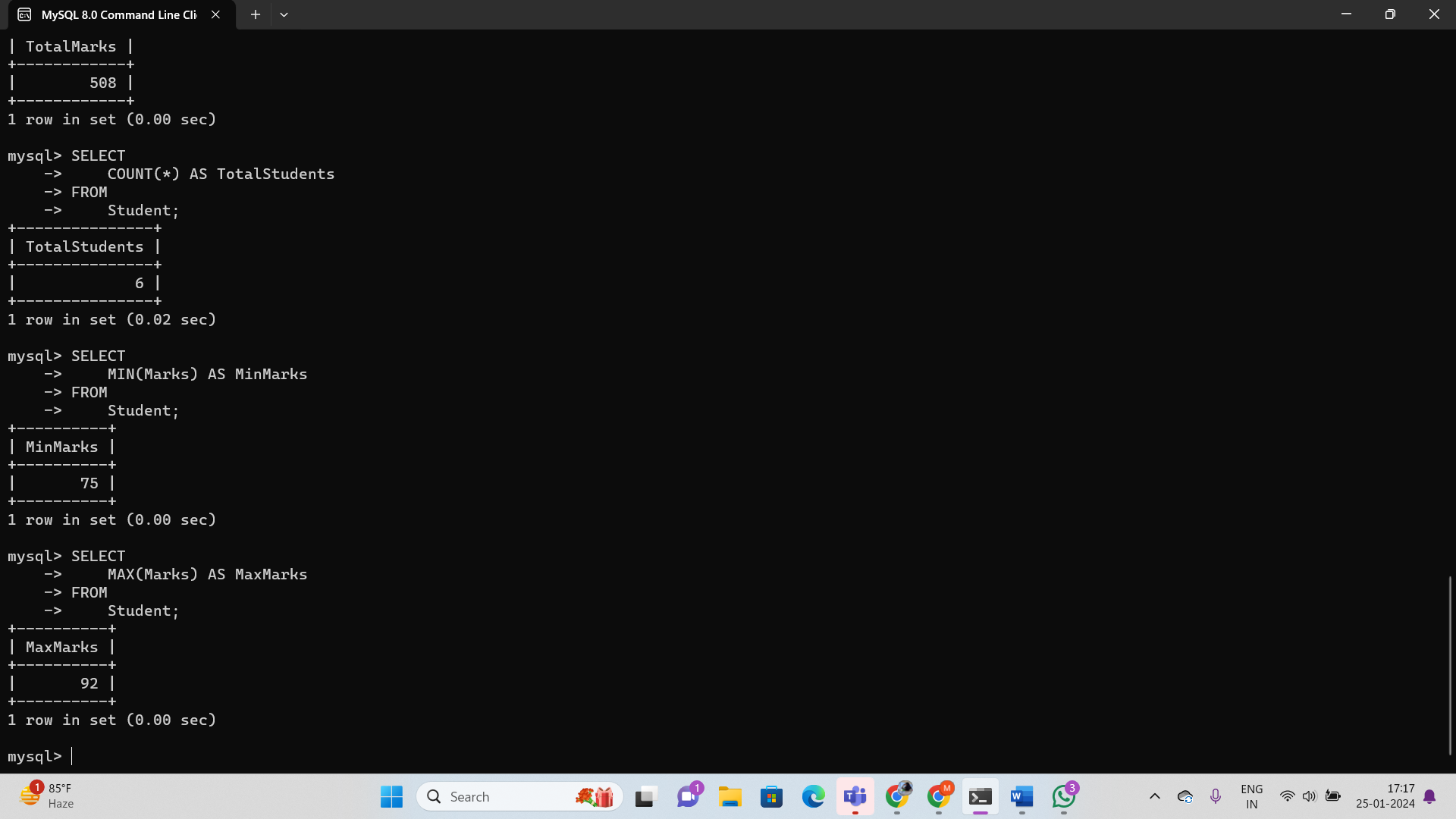
SELECT

MAX(Marks) AS MaxMarks

FROM

Student;

This query finds the maximum value in the **Marks** column of the **Student** table, providing a single value representing the maximum marks among all students.



These total aggregations are useful when you want to understand the overall summary or characteristics of a dataset. They can be applied to various scenarios, such as analyzing exam scores, calculating average sales, or determining the total count of records. The choice of which aggregation to use depends on the specific information you want to extract from your data.