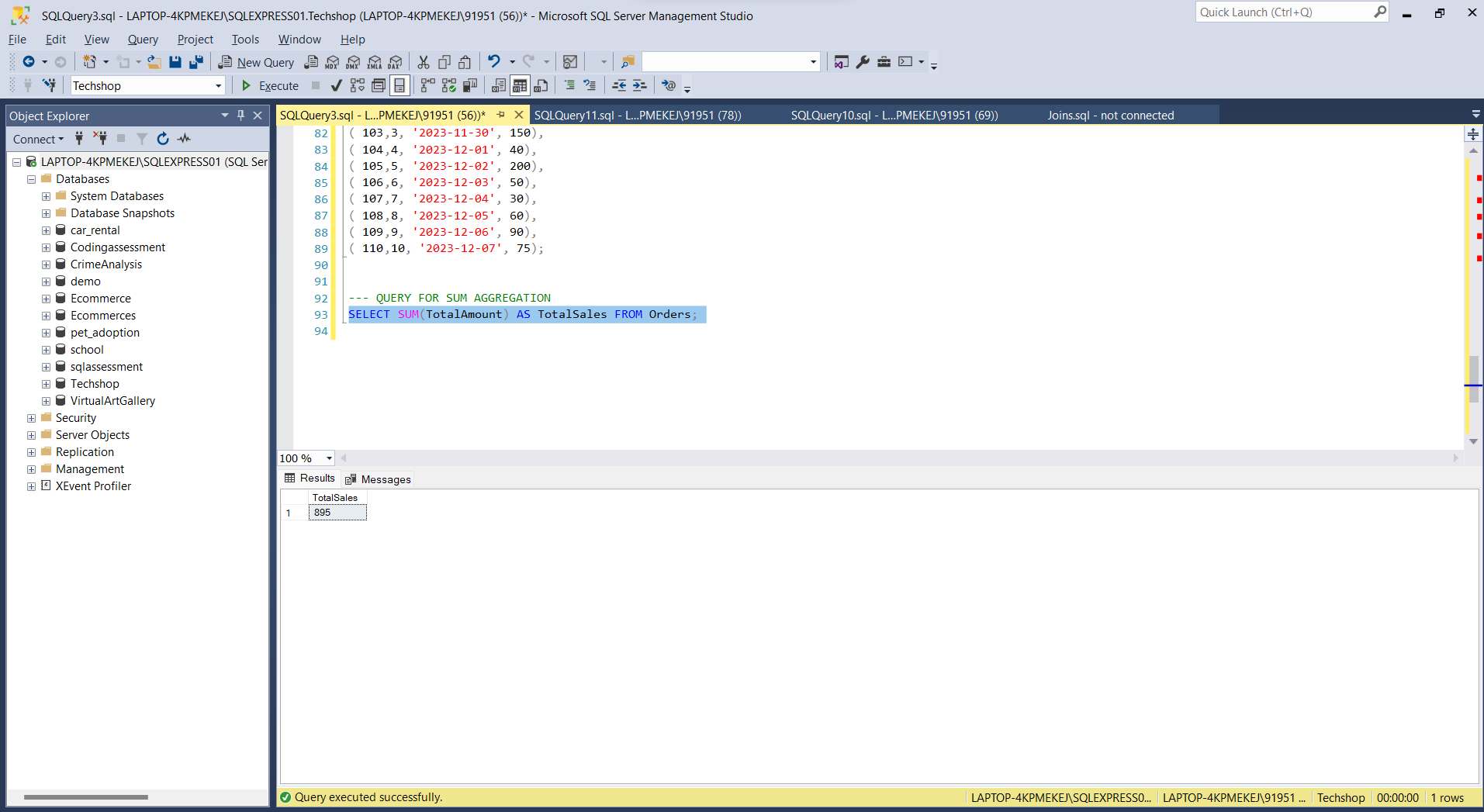
**TOTAL AGGRERGATIONS USING SQL QUERIES**

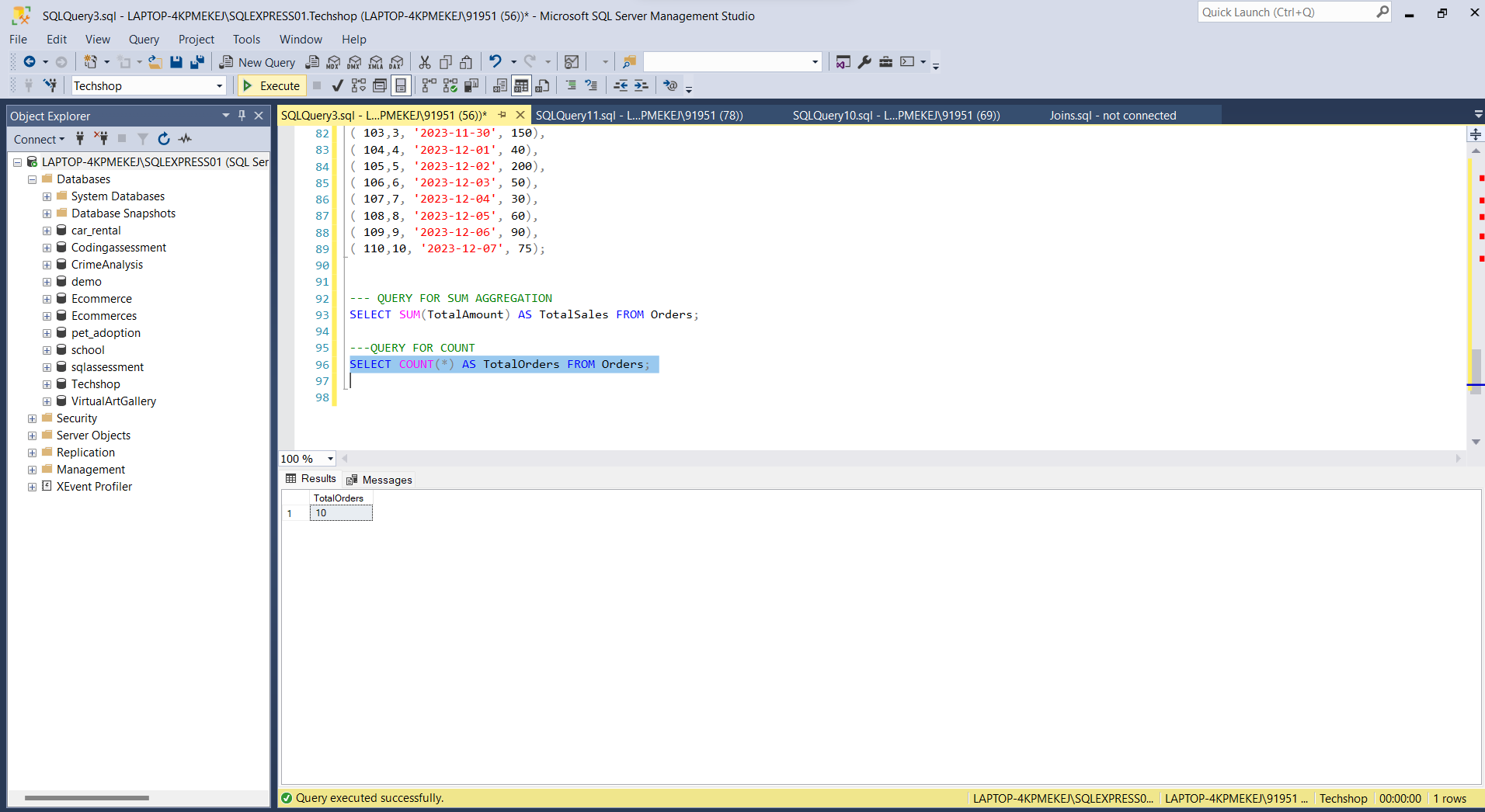
SQL aggregation involves consolidating multiple values into a single result using aggregate functions. These functions include:

* SUM()
* COUNT()
* AVG()
* GROUP BY
* MIN() and MAX()

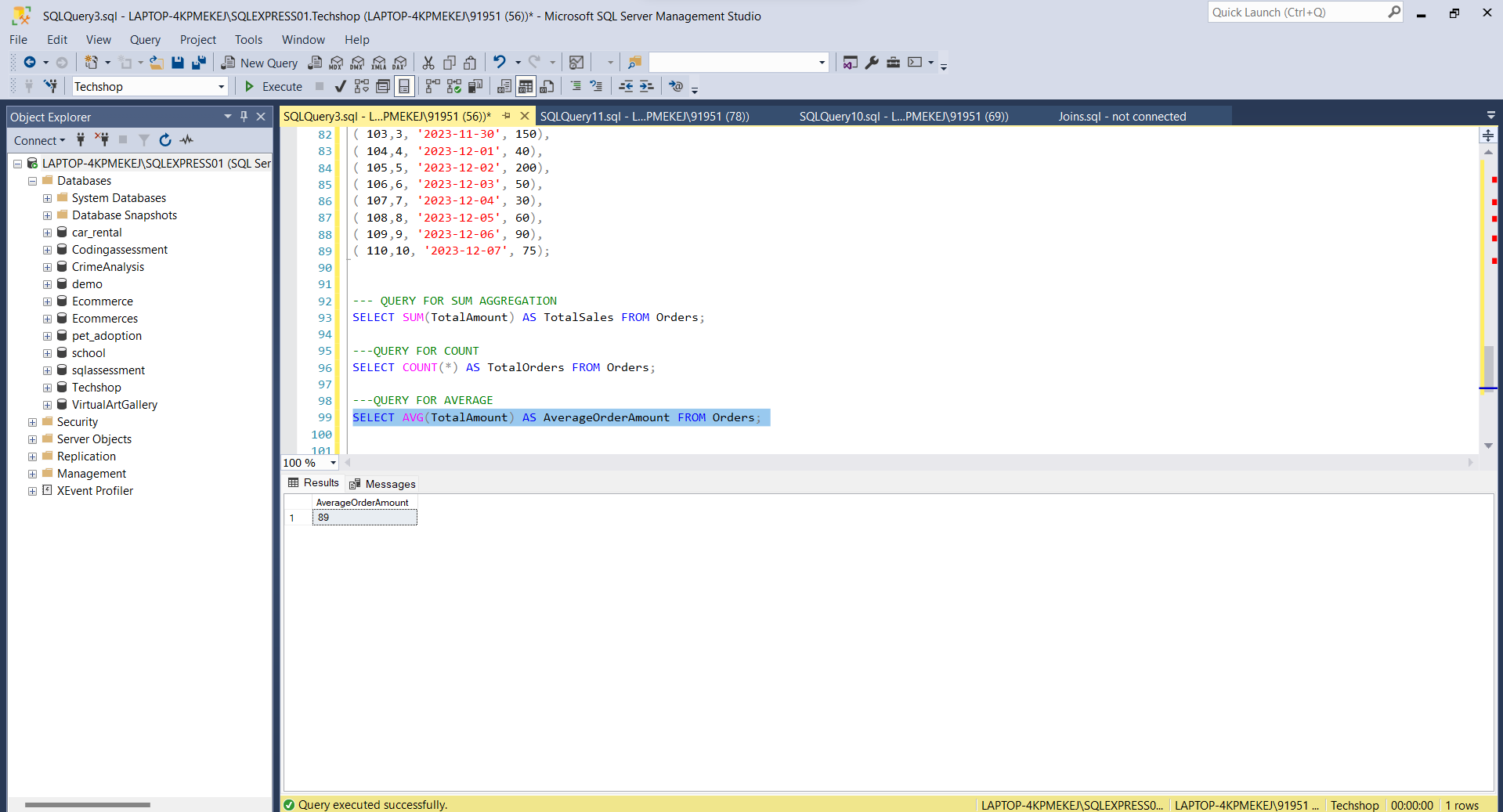
**SUM():** The SUM() function in SQL is an aggregate function used to calculate the total sum of numeric values within a specified column or expression in a database table.



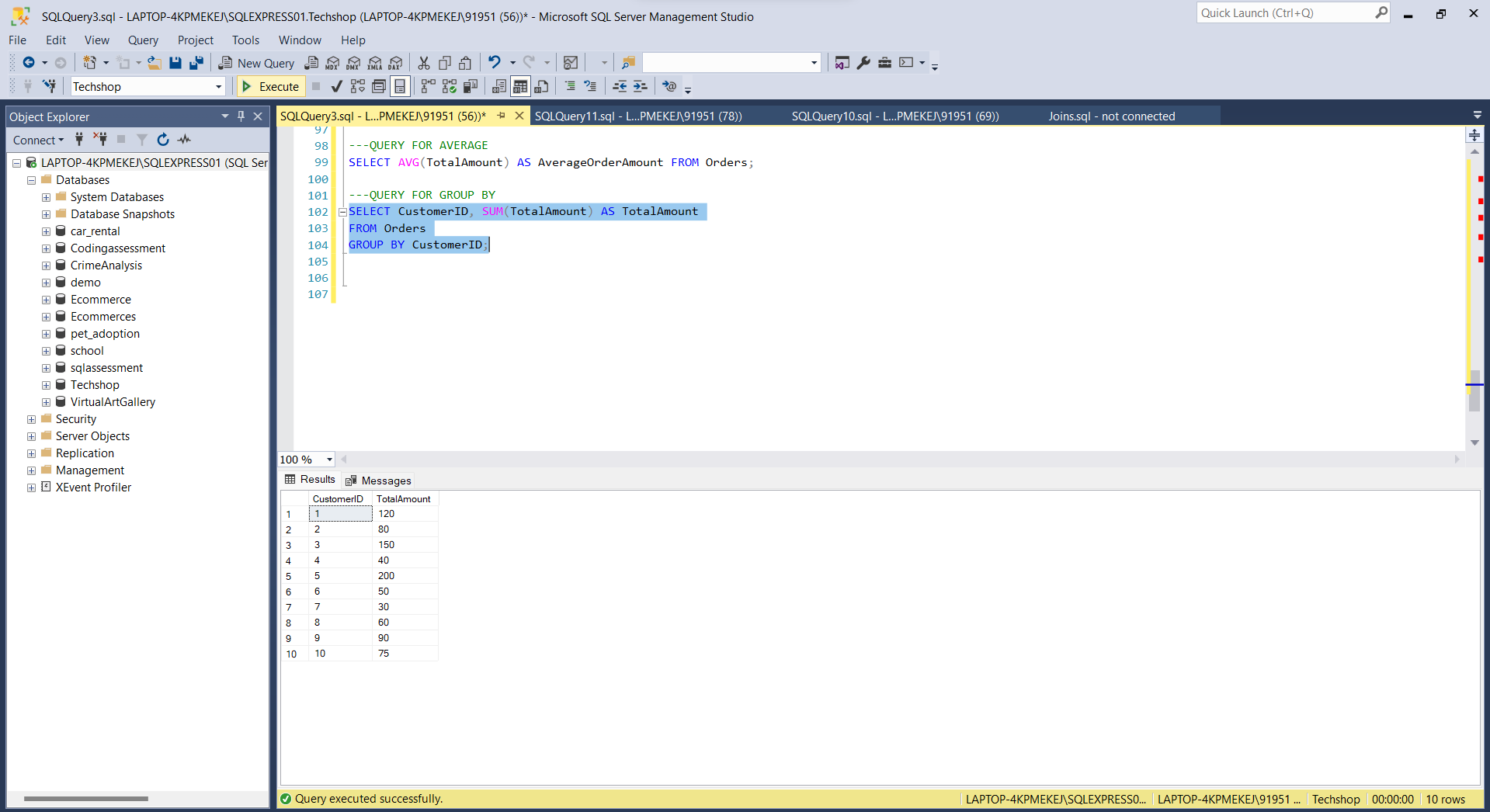
**COUNT():** The COUNT() is an SQL function used to calculate the number of rows that meet a specified condi tion in a database table.



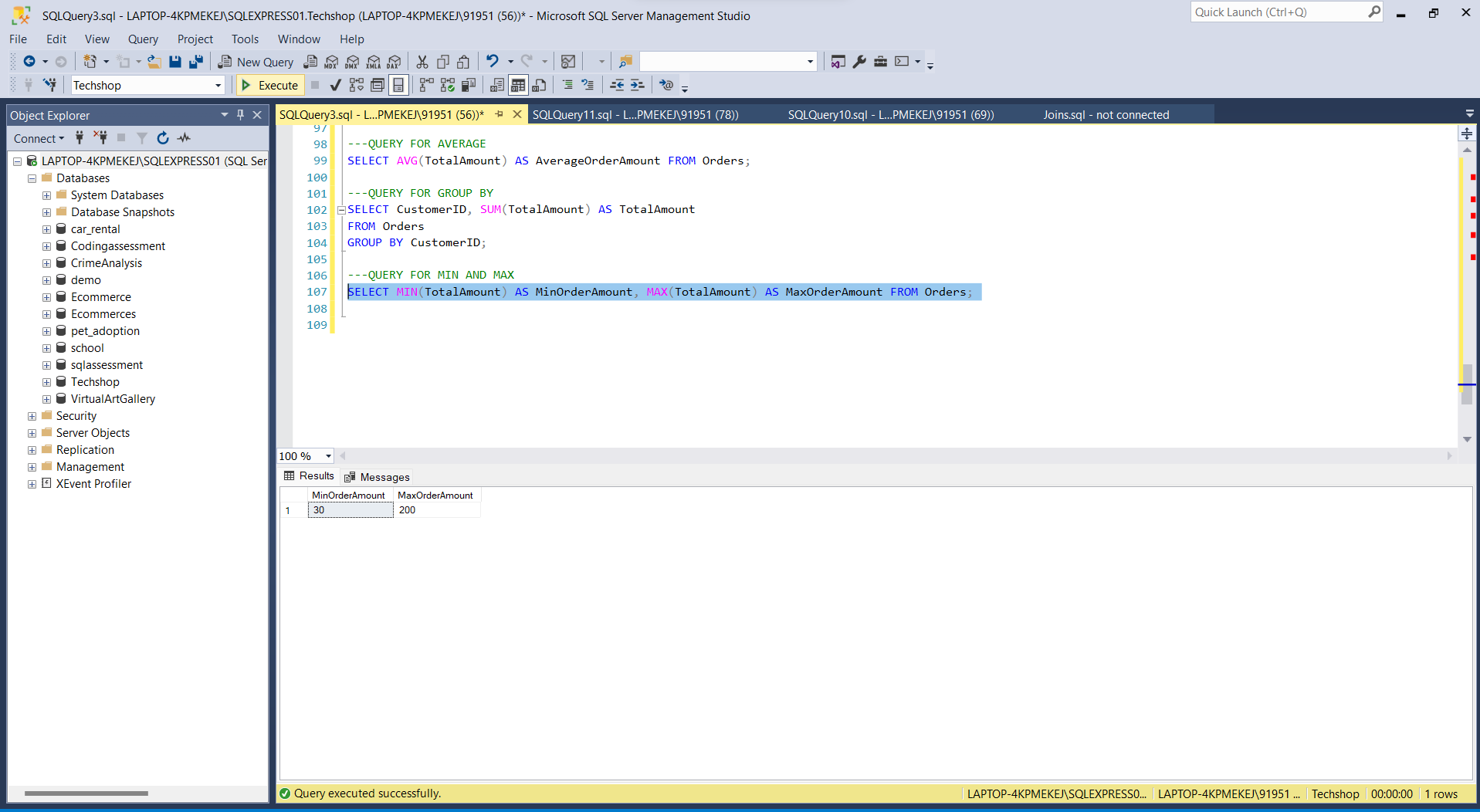
**AVG():**The AVG() is a SQL aggregate function used to calculate the average value of a set of numerical values within a specified column in a database table.



**GROUP BY :** GROUP BY is a clause in SQL used to organize rows that have the same values into summary rows, typically to perform aggregate functions like COUNT, SUM, AVG, etc., on those grouped rows.



**MIN() AND MAX():**MIN() and MAX() are SQL aggregate functions used to find the smallest (minimum) and largest (maximum) values, respectively, within a specified column of a table.

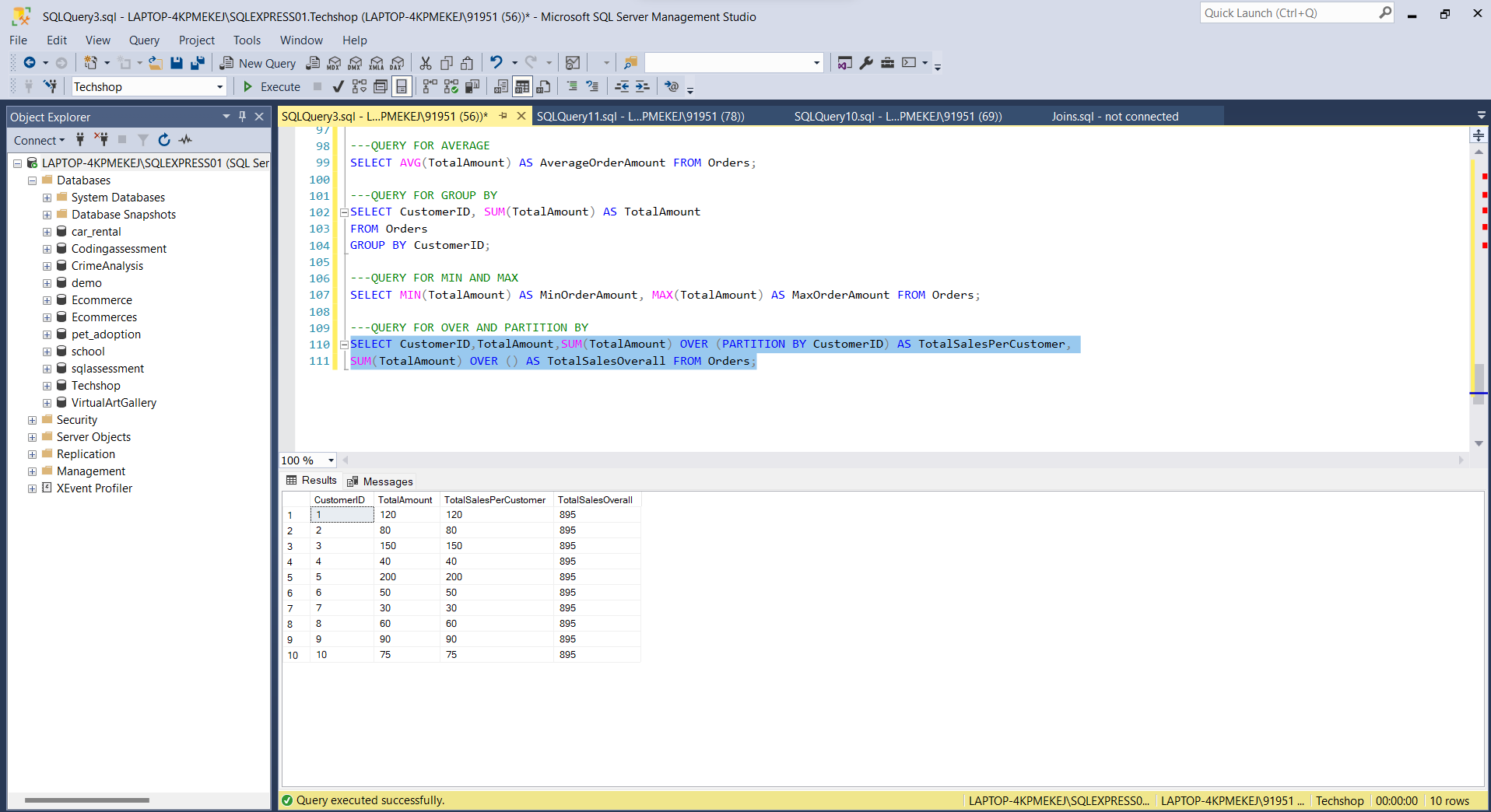


OVER And PARTITION BY clauses in SQL :

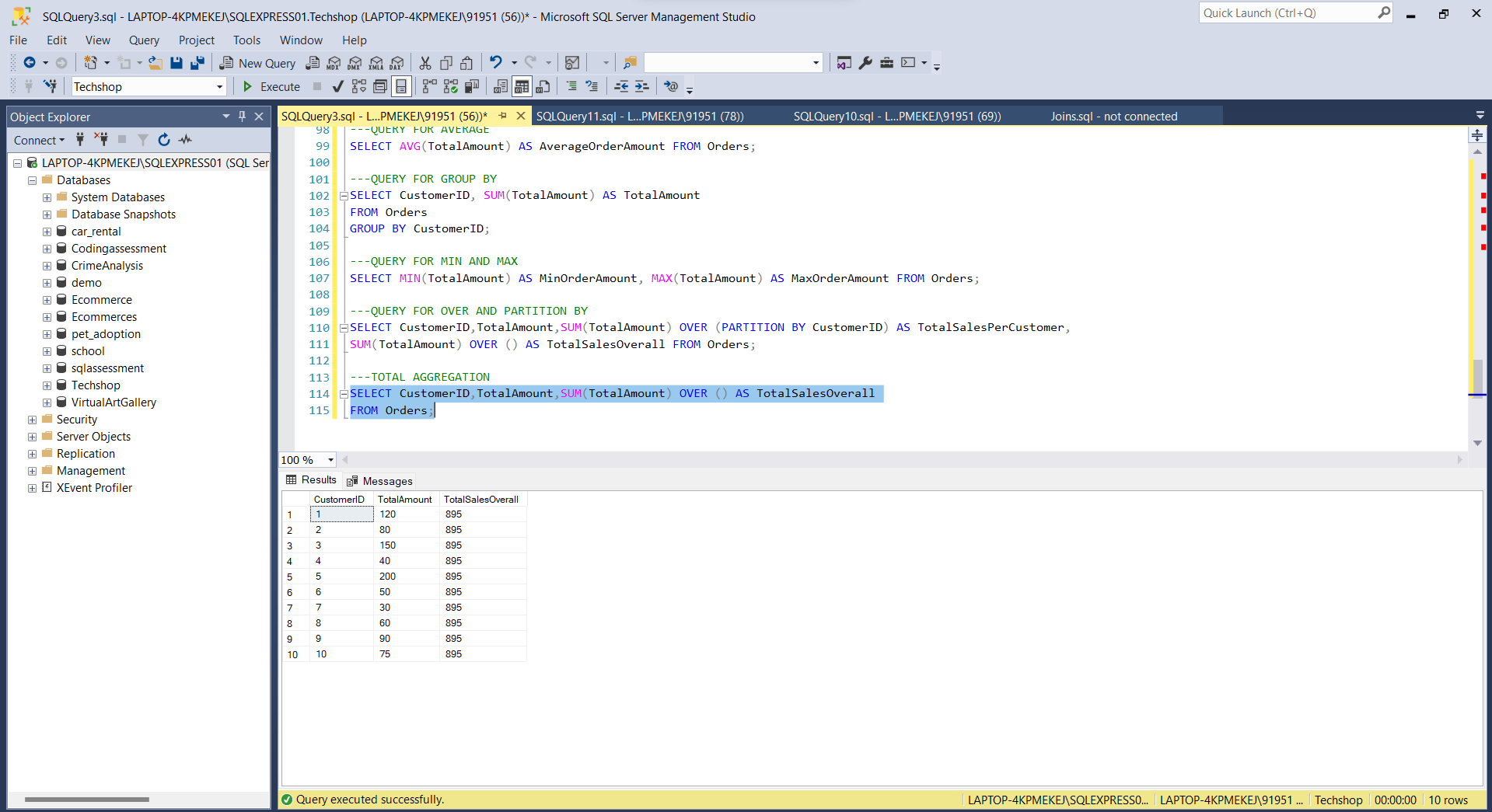
In SQL, the OVER and PARTITION BY clauses team up with window functions to crunch numbers across specific sets of rows relative to the current one. They're essential in analytical queries, where you want to whip up aggregated outcomes or crunch numbers within a defined window of rows.

**OVER:** The OVER clause accompanies window functions, providing parameters for the window or row set upon which the function acts. It enables you to define the partitioning, ordering, and framing of the window.

**PARTITION BY:** The PARTITION BY clause, nested within the OVER clause, divides a large dataset into smaller, manageable segments. It's particularly handy for conducting calculations on specific rows within a group, leveraging data from within that same group.



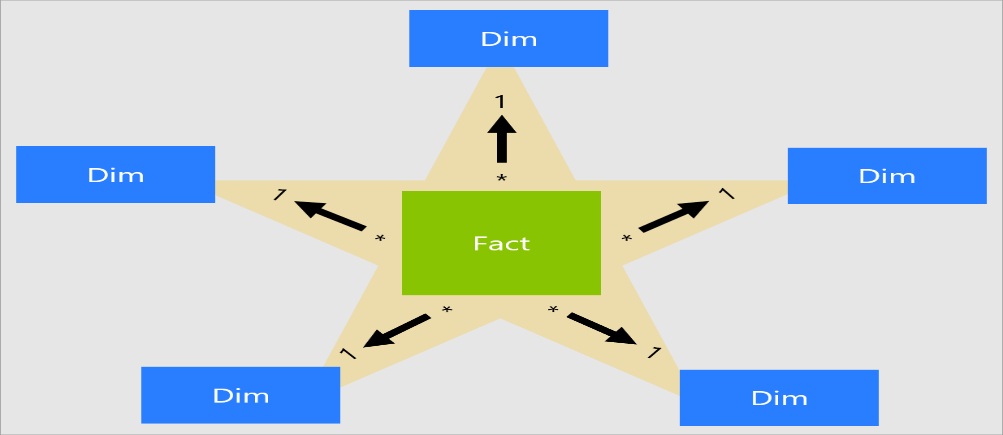
TOTAL AGGREGATION USING OVER AND PARTITION BY



**Star schemas and Snowflaking schemas:**

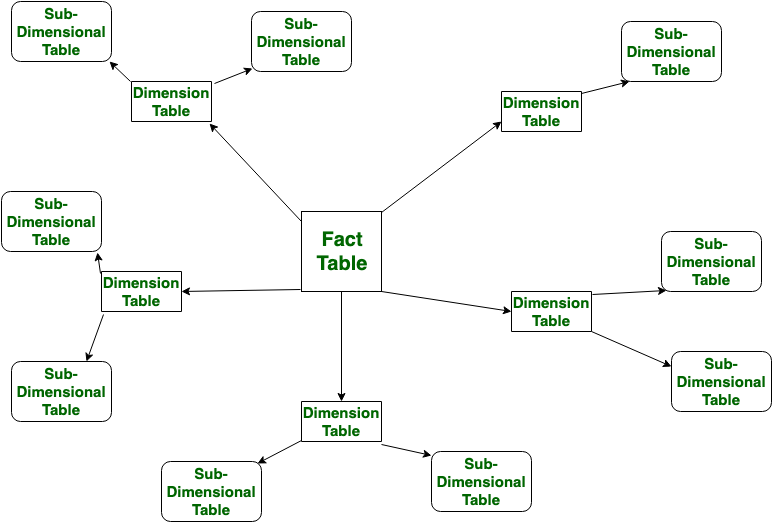
Star Schemas:

In a star schema, the fact table serves as the central hub, linking to multiple dimension tables. This design tends to increase data redundancy, consuming more disk space due to the denormalized structure where tables are organized without the typical normalization practices.



**Snowflake Schema:**

In a snowflake schema, dimension tables extend beyond a single layer, connecting to additional dimensions. This design minimizes data redundancy, resulting in a more efficient use of disk space compared to the star schema. The structure of the tables is partially denormalized, contributing to a more organized and streamlined data model.



**Rules and Restrictions to Group and Filter Data in SQL queries :**

In SQL queries, grouping and filtering data are common operations that help organize and extract specific information from a dataset. Here are some rules and restrictions to consider when using these operations:

**Grouping Data:**

* Aggregation Functions with GROUP BY
* Non-Aggregated Columns

**FILTERING DATA :**

* **WHERE**
* **LOGICAL OPERATORS**
* **IN AND OUT IN**
* **NULL VALUES**
* **Comparison Operators**
* **LIKE operator**

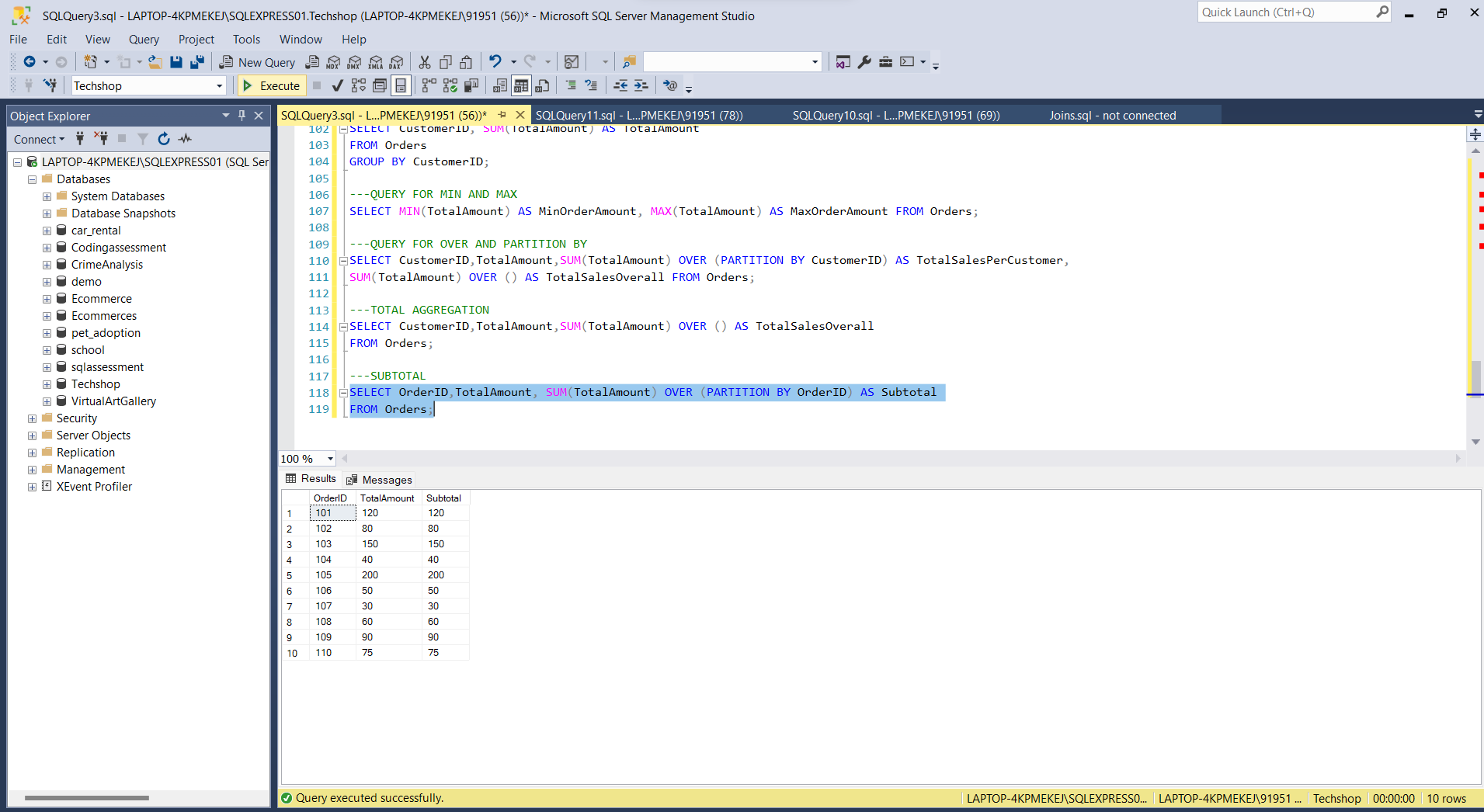
**Order of Execution of SQL Queries :**

The order of execution of SQL queries is crucial to understanding how the database processes and retrieves data. SQL queries typically go through several logical processing phases, and understanding this order can be beneficial for optimizing queries and troubleshooting performance issues. The order of execution for a SQL query is:

* FROM
* WHERE
* GROUP BY
* HAVING
* SELECT
* DISTINCT
* ORDER BY
* LIMIT

**How to calculate Subtotals in SQL :**

Calculating subtotals in SQL queries involves using aggregate functions such as SUM(), AVG(), COUNT(), etc., along with GROUP BY clauses to group the data based on specific criteria. By grouping the data appropriately, these aggregate functions can compute subtotals for each group, providing summarized results for subsets of the dataset.



**Differences Between UNION EXCEPT and INTERSECT Operators in SQL**

In SQL Server, the UNION, EXCEPT, and INTERSECT operators are used to combine or compare the

results of two or more queries. Here are the key differences between them:

**UNION Operator:**

Combines the results of two or more SELECT statements into a single result set.

Removes duplicate rows from the combined result set.

**EXCEPT Operator:**

Returns the rows that are unique to the first SELECT statement and not present in the second SELECT

statement.

Removes duplicate rows from the result set.

**INTERSECT Operator:**

Returns the common rows between two SELECT statements.

Removes duplicate rows from the result set.