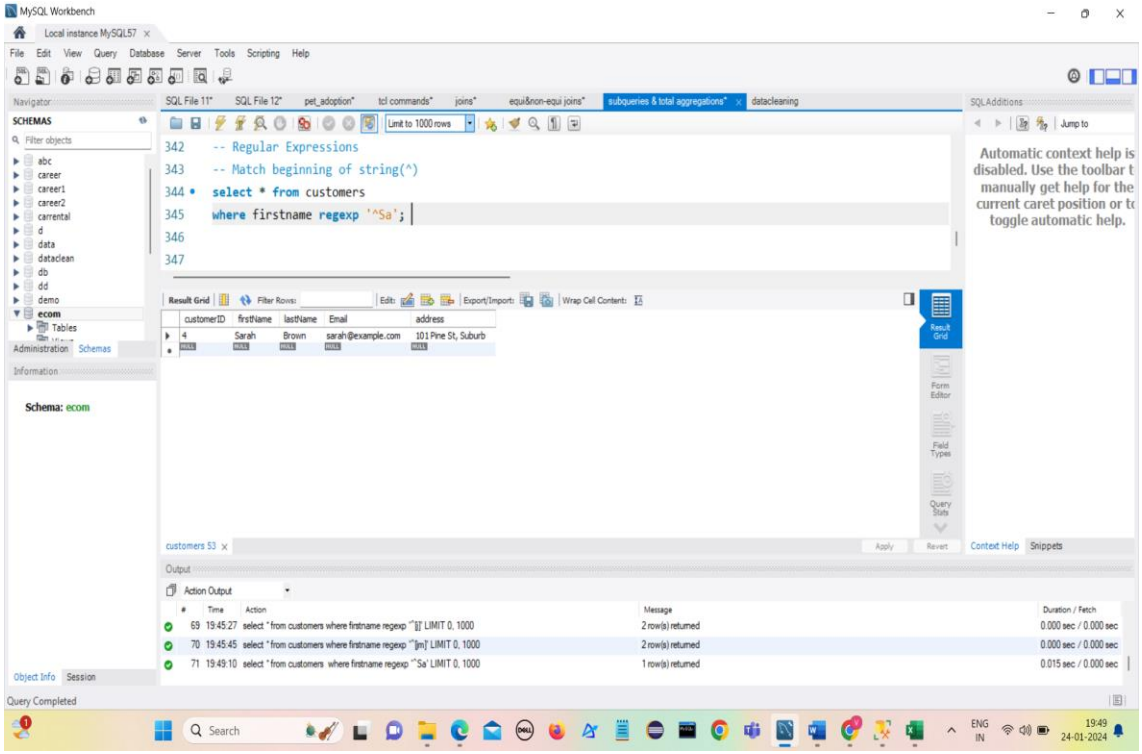
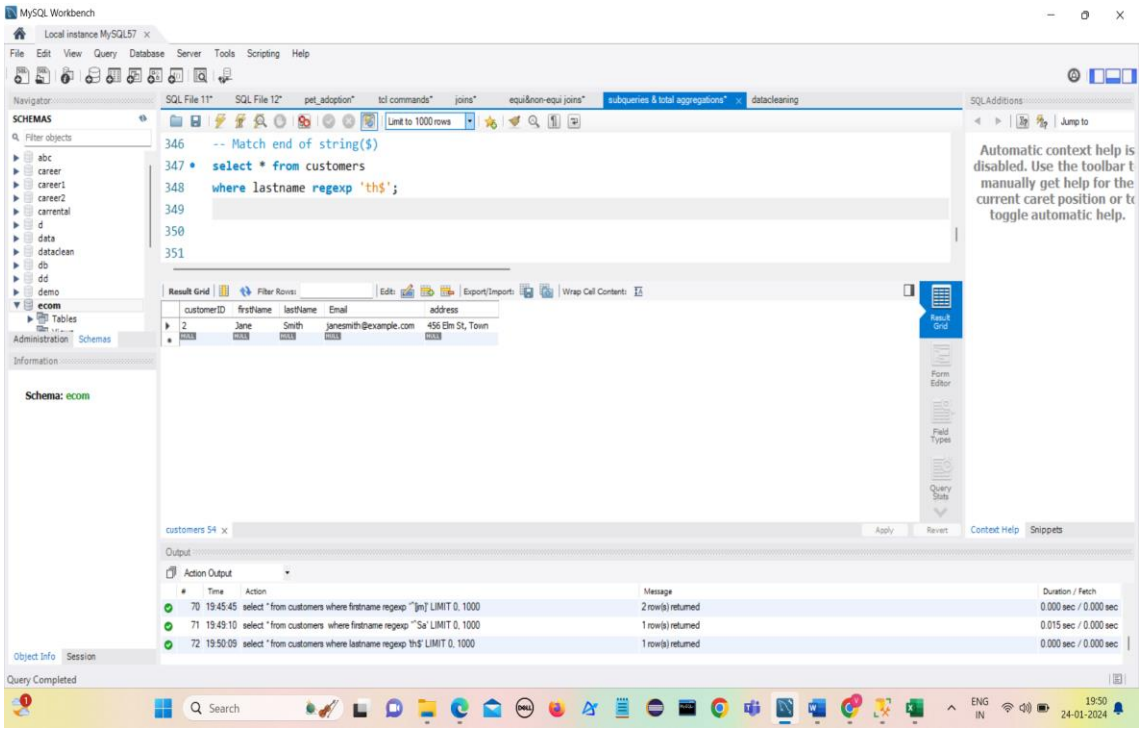
ASSIGNMENT-7

REGEXP

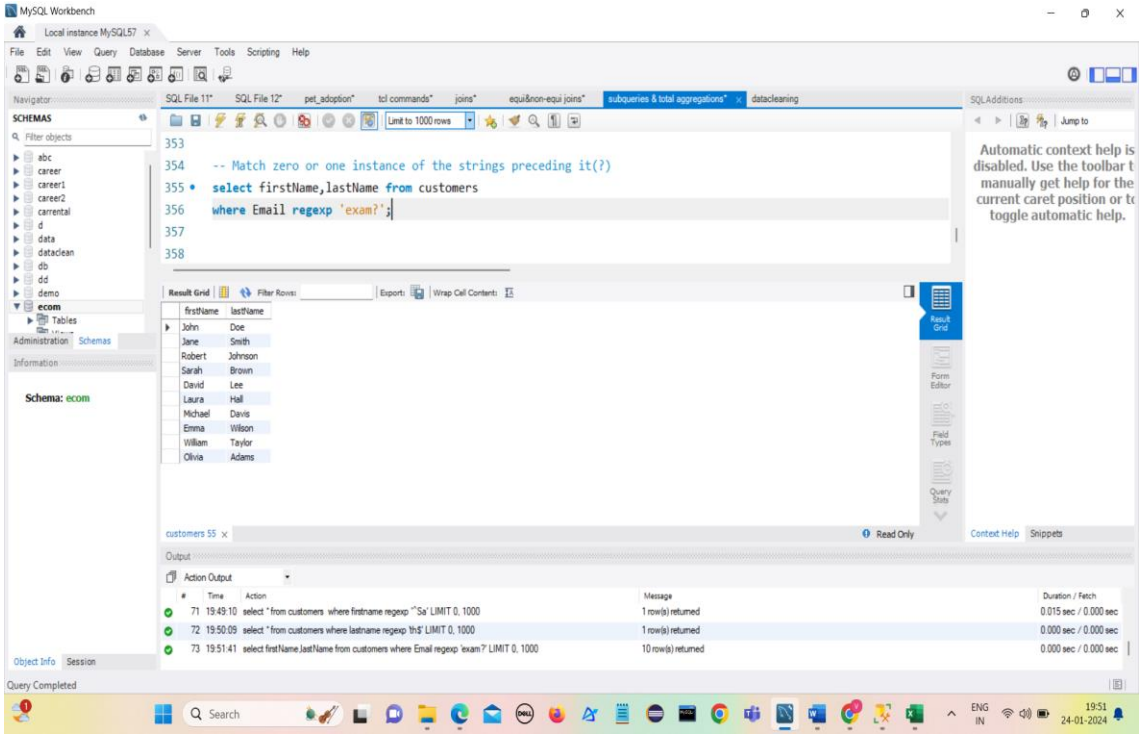
1) Match beginning of string(^)



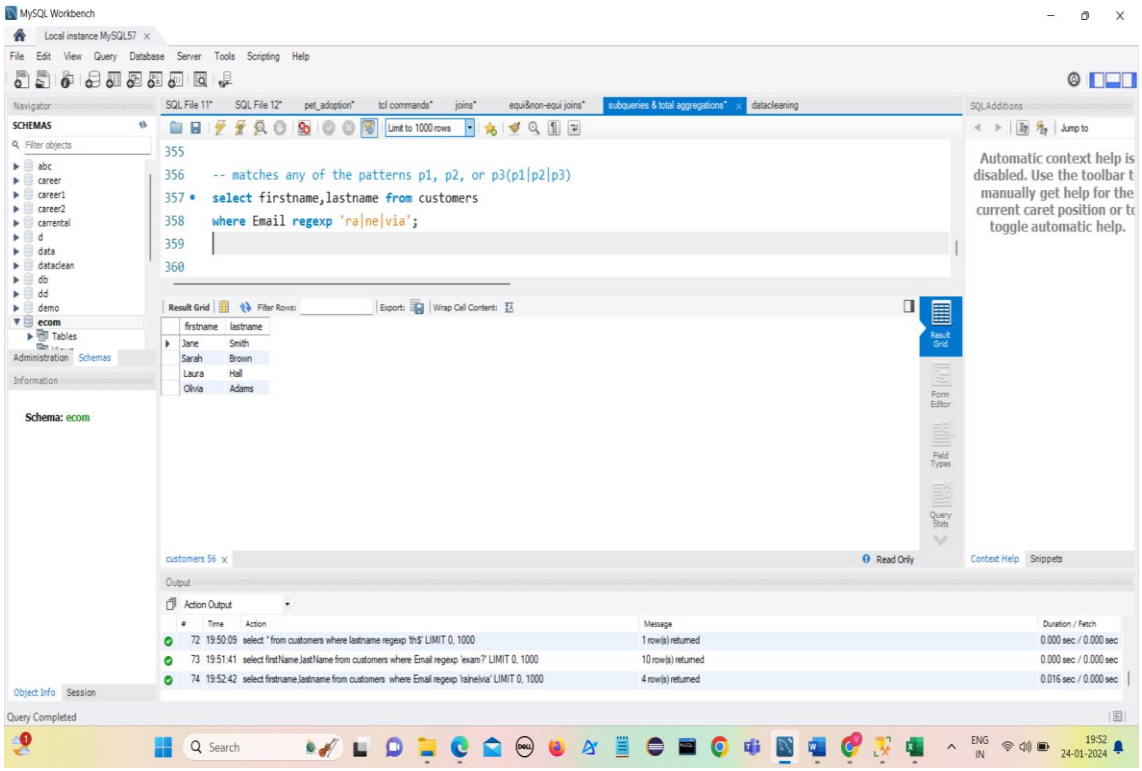
2) Match end of string($)



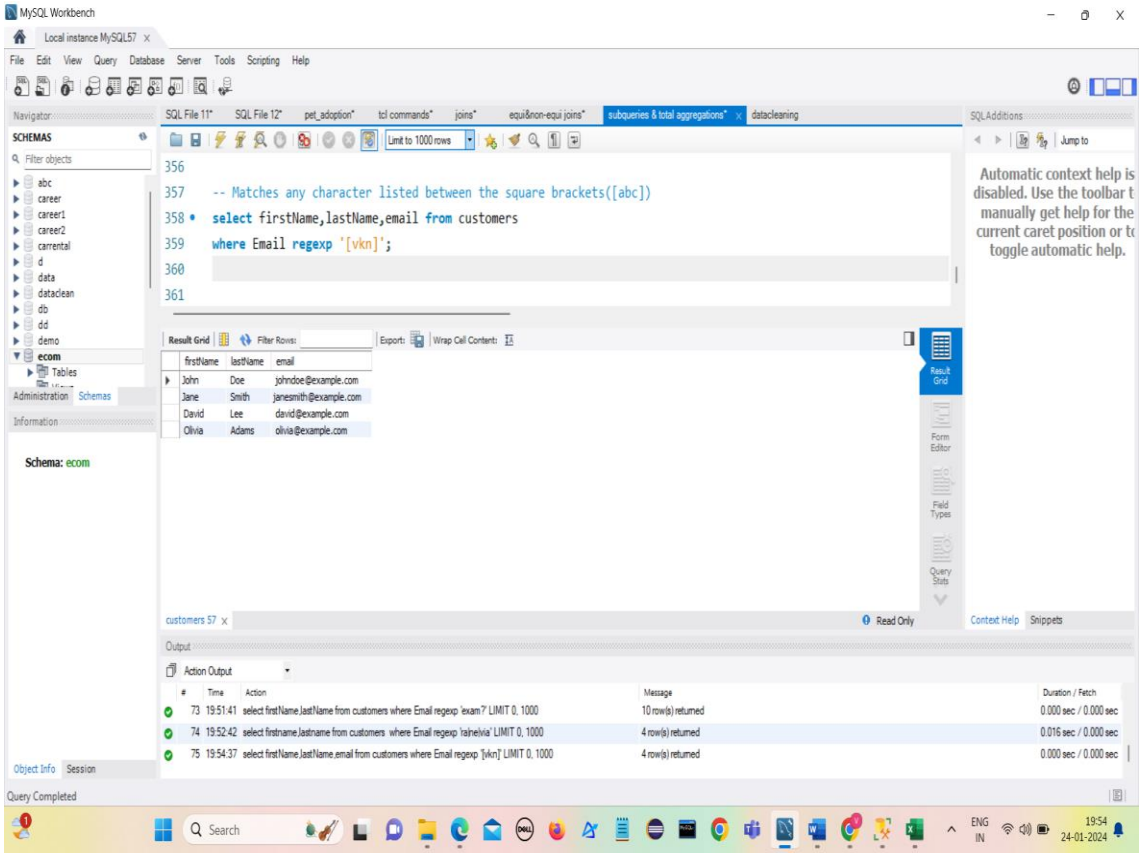
3) Match zero or one instance of the strings preceding it(?)



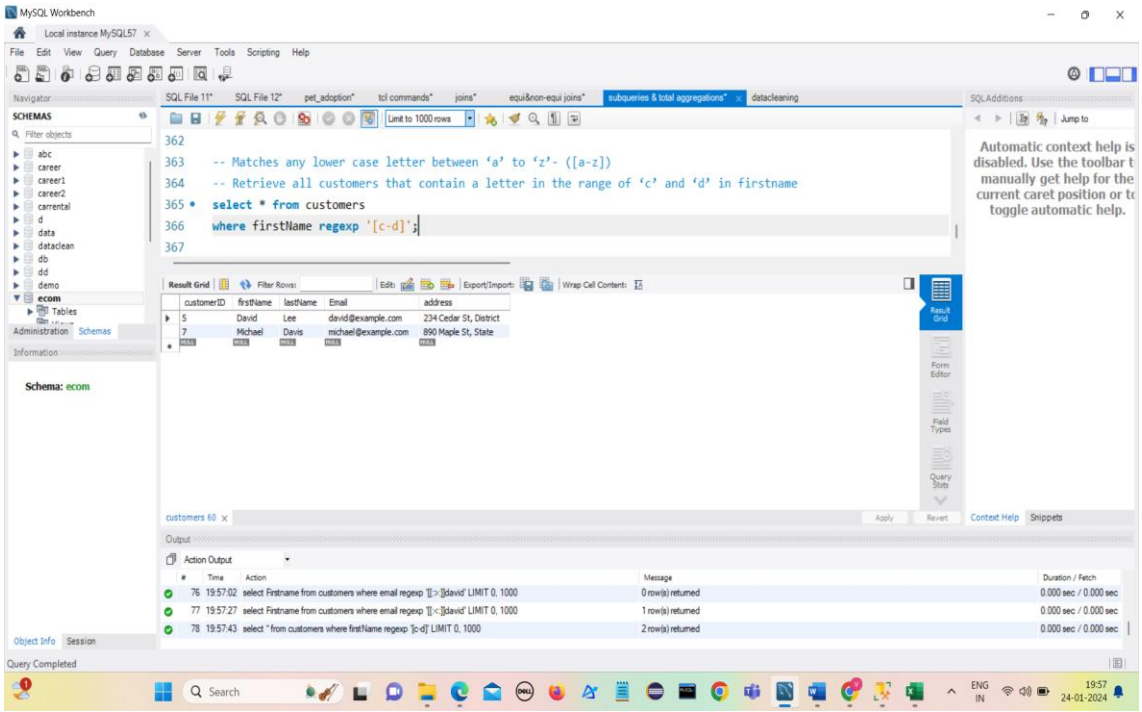
4) Matches any of the patterns p1, p2, or p3(p1|p2|p3)



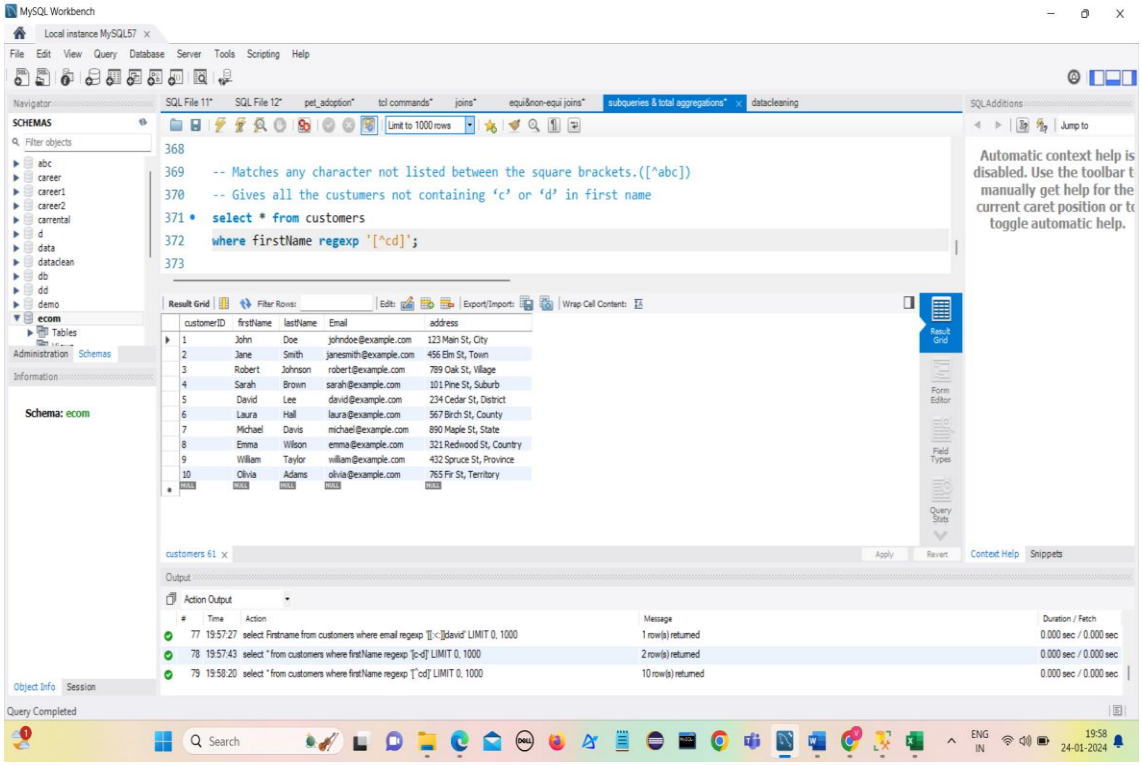
5) Matches any character listed between the square brackets([abc])



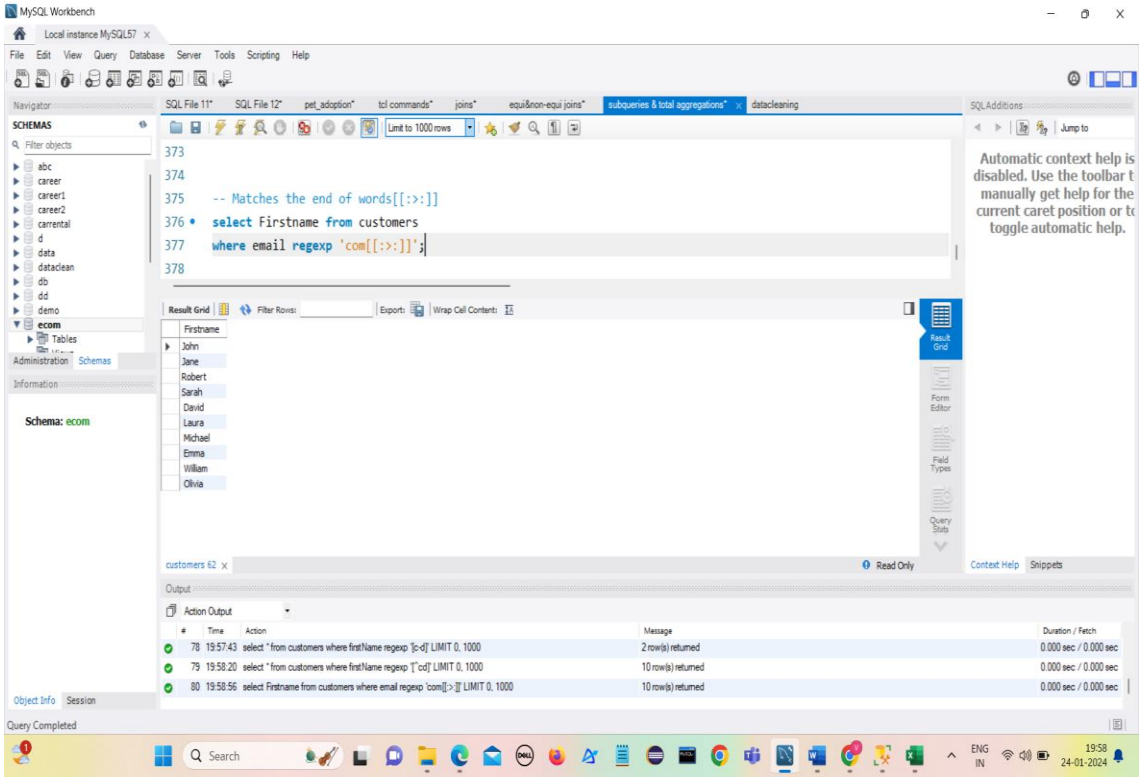
6) Matches any lower case letter between ‘a’ to ‘z’- ([a-z])
Retrieve all customers that contain a letter in the range of ‘c’ and ‘d’ in firstname.



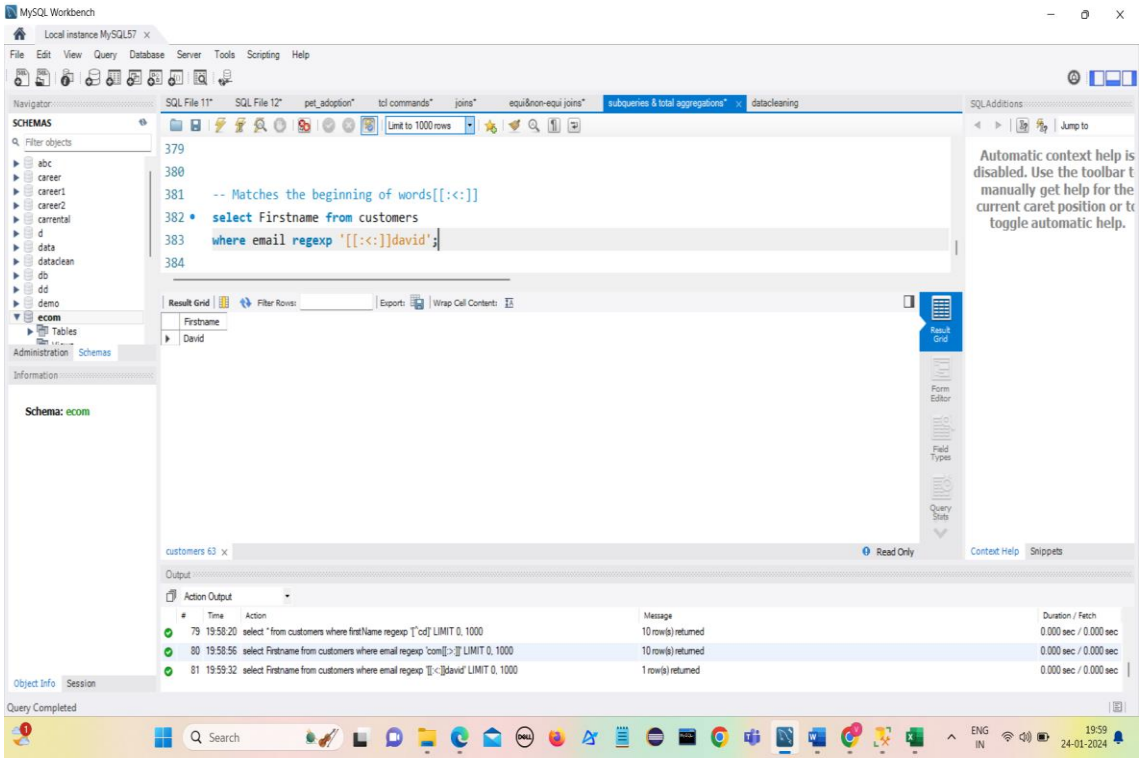
7) **Matches any character not listed between the square brackets.([^abc]) Gives all the custumers not containing ‘c’ or ‘d’ in first name**



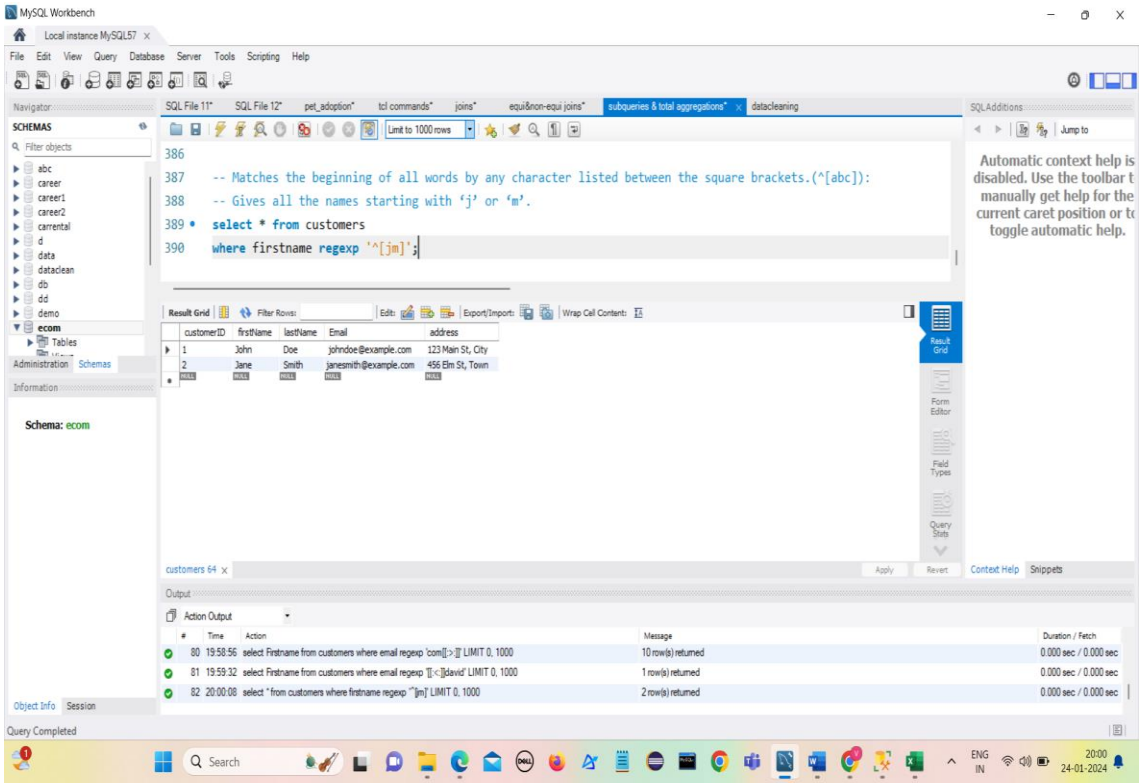
8) **Matches the end of words[[:>:]]**



9) Matches the beginning of words[[:<:]]

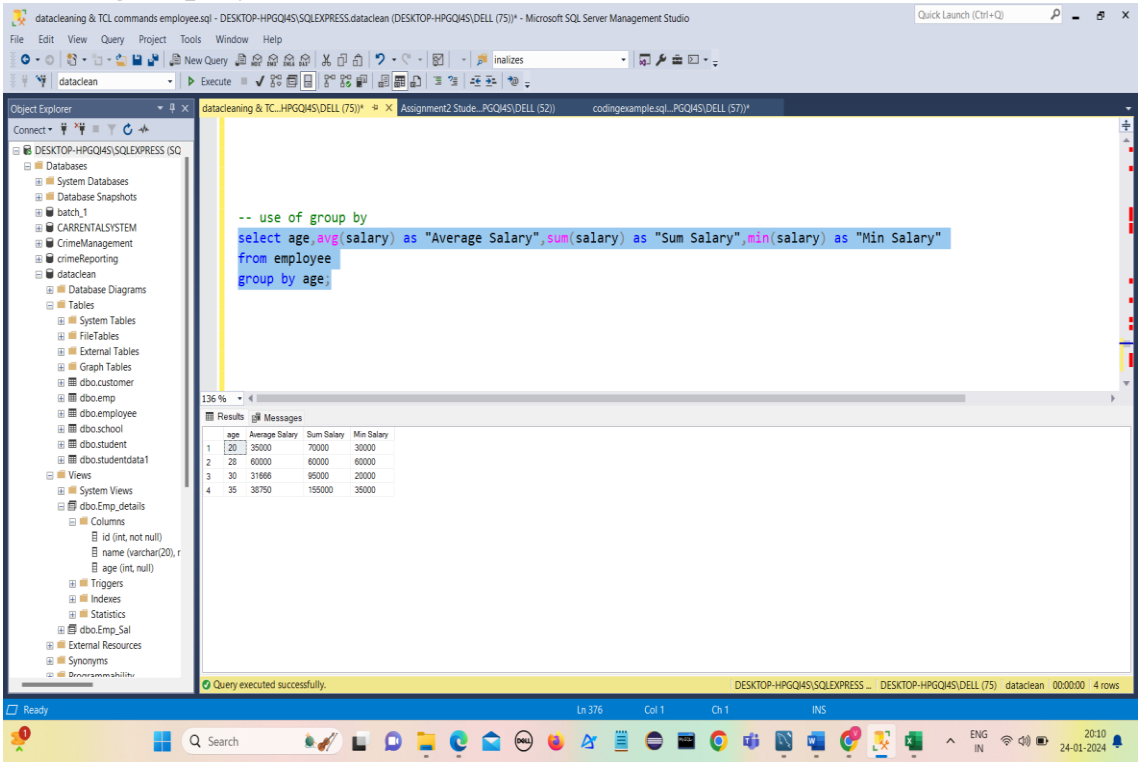


10) Matches the beginning of all words by any character listed between the square brackets.(^[abc]):
Gives all the names starting with ‘j’ or ‘m’.



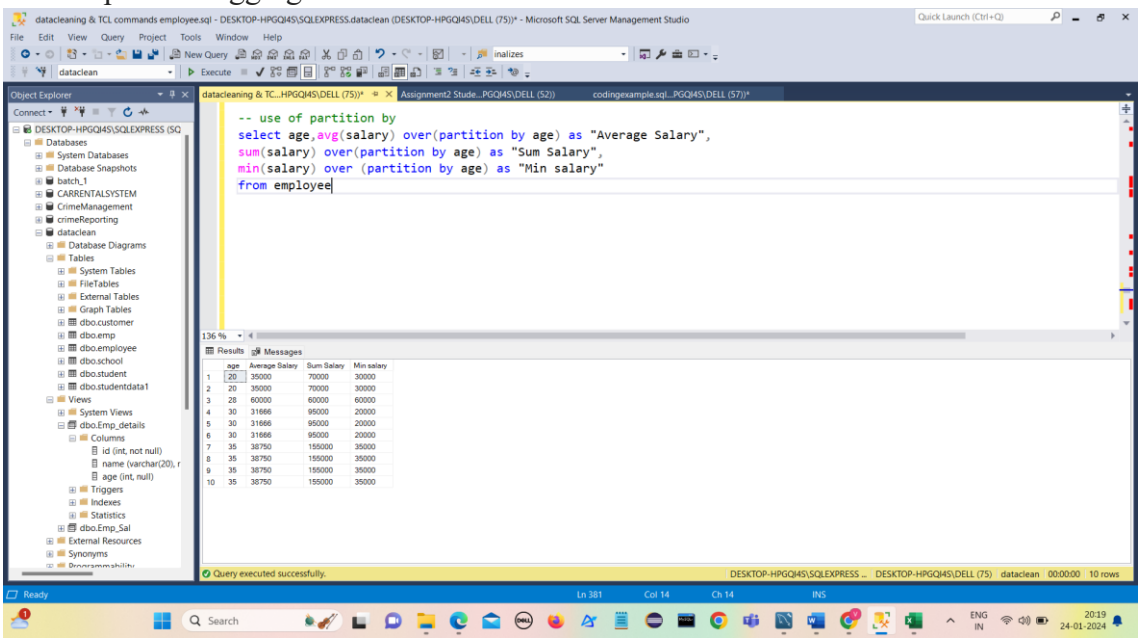
OVER AND PARTITION BY CLAUSE

1) Use of group by

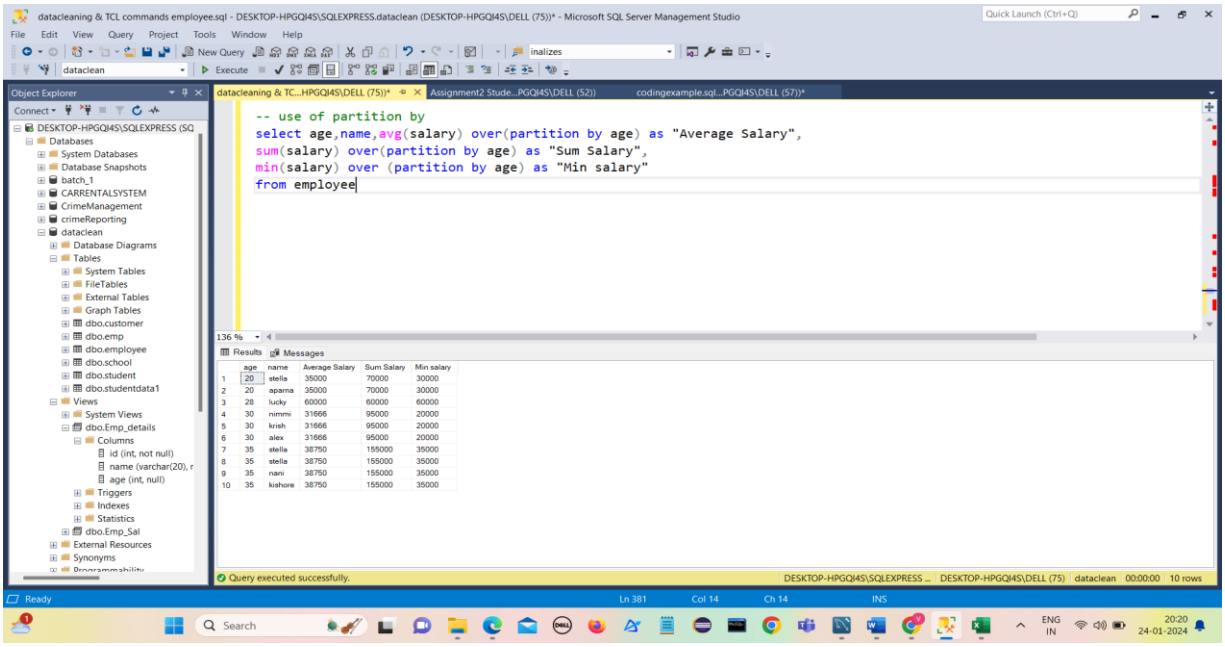


2) Partition by

SQL PARTITION BY clause with the **OVER** clause to specify the column on which we need to perform aggregation.

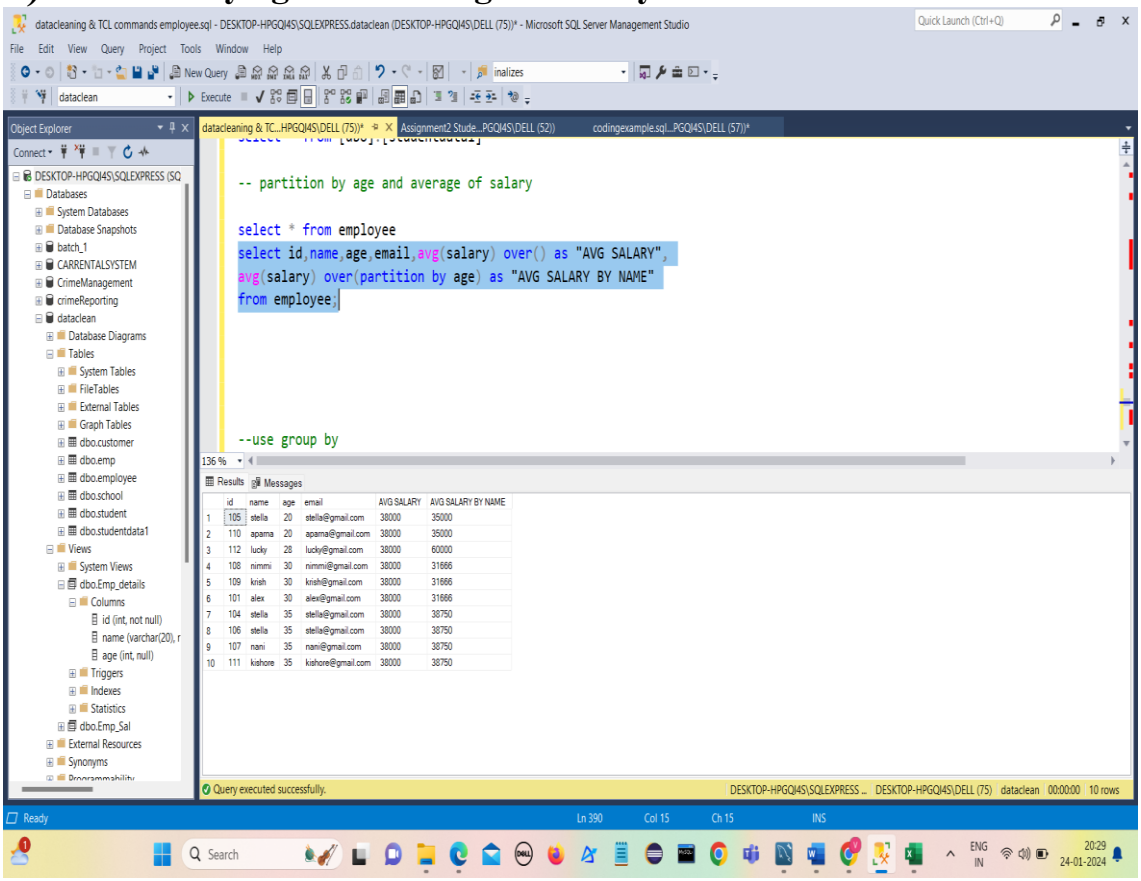


We can add name in select with partition but we cannot add in group by

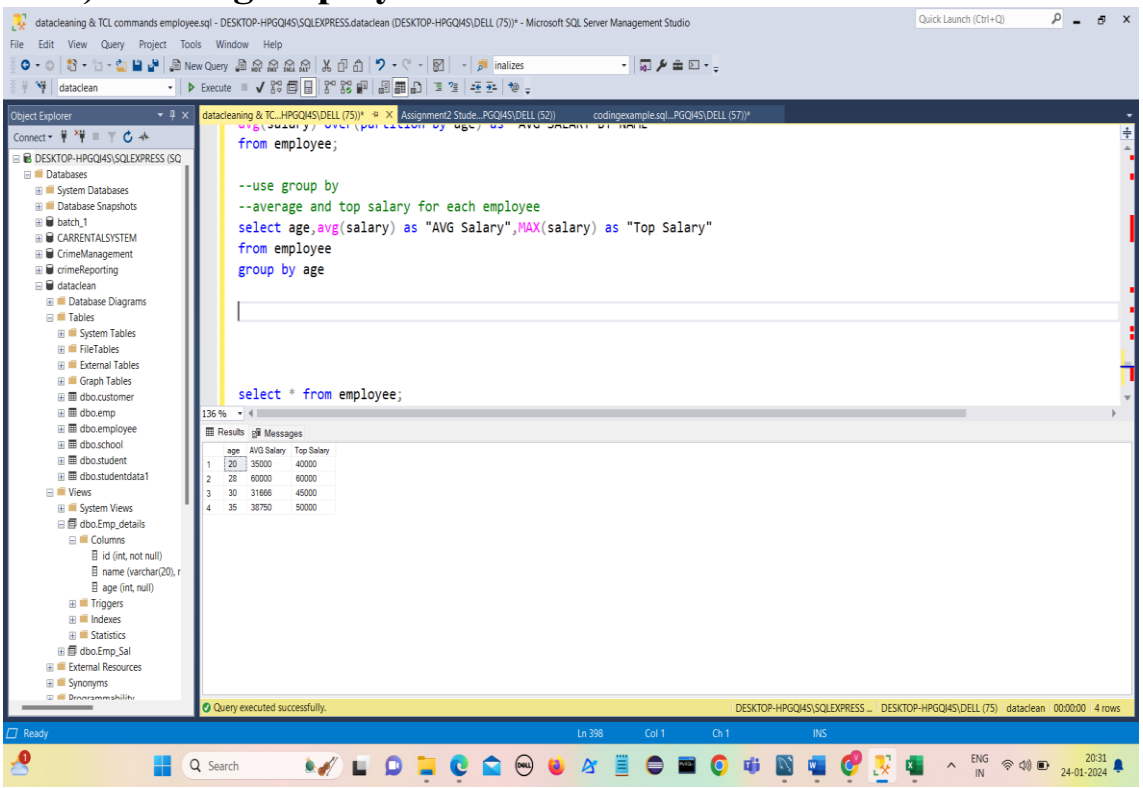


Total aggregations using over and partition clause

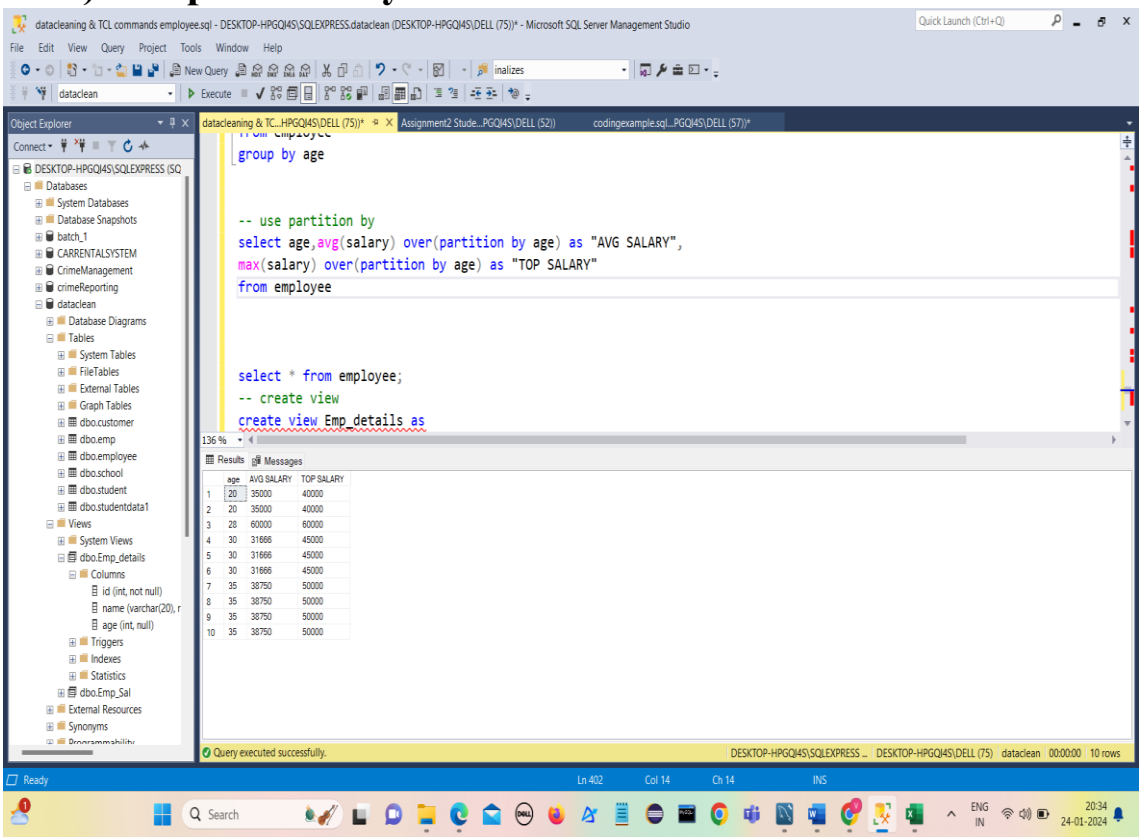
1) Partition by age and average of salary



2) Use of group by



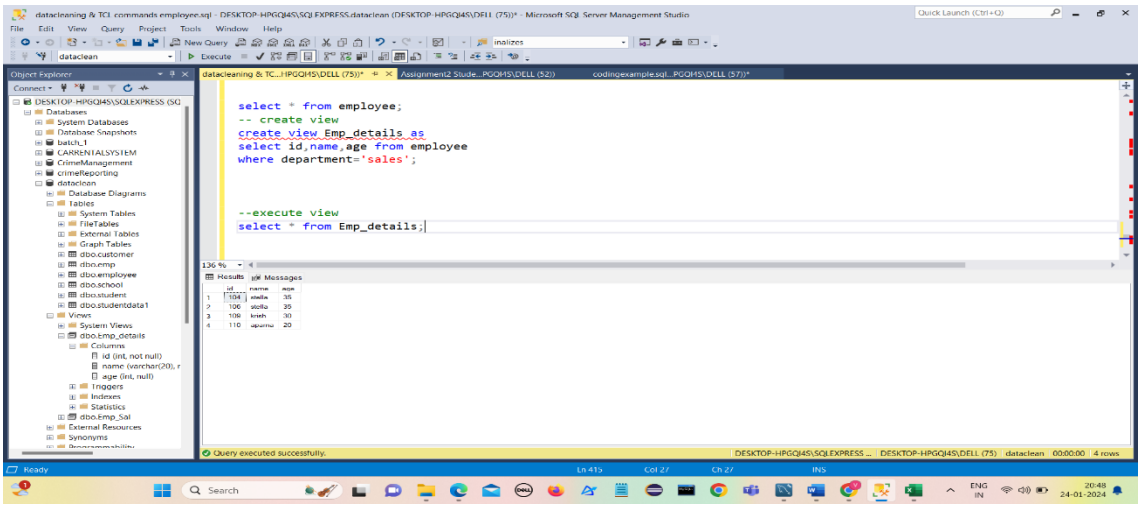
3) Use partition by



Materialized views

- A materialized view is a duplicate data table created by combining data from multiple existing tables for faster data retrieval.
- Materialized views work by precomputing and storing the results of a specific query as a physical table in the database.
- The database performs the precomputation at regular intervals, or users can trigger it by specific events.
- The data in the materialized view is precomputed and stored, meaning that the results are already available without the need to recompute the query each time the view is accessed.

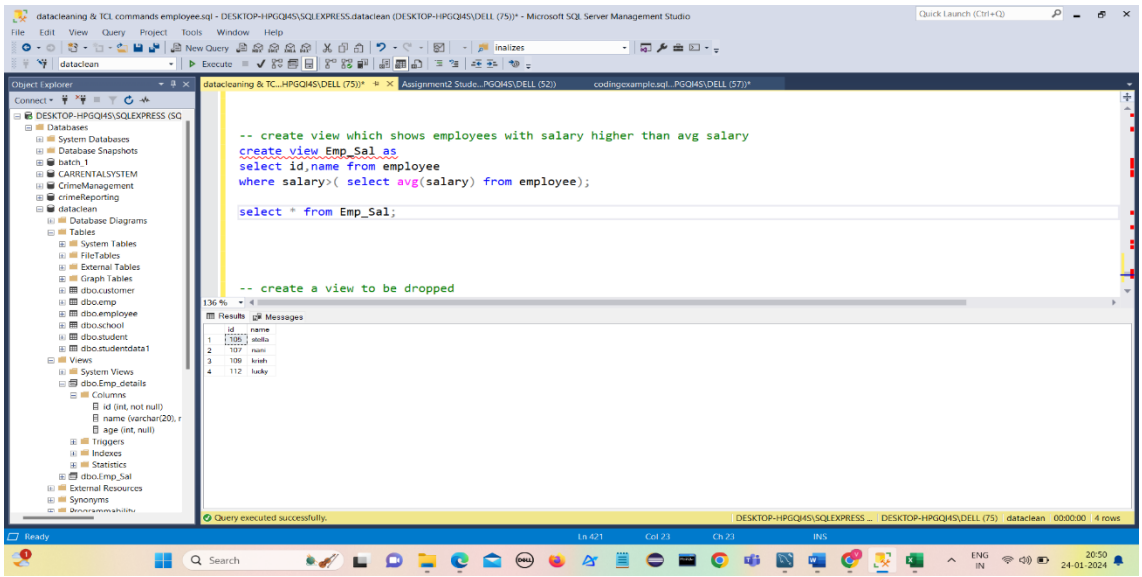
Creating a view:



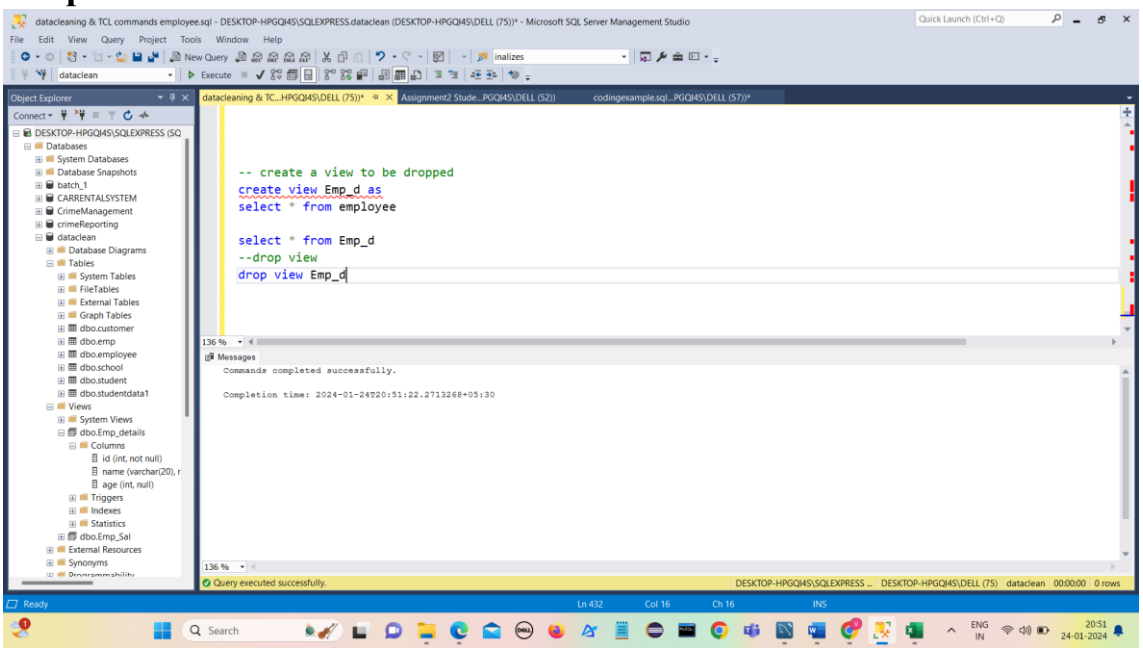
Execute view:

select * from viewname;

create view which shows employees with salary higher than avg salary



Drop view



Star Schema:

A star schema consists of a central fact table that references multiple dimension tables. Each dimension table is denormalized ("flattened") to avoid the query overhead that comes with a highly normalized schema, which can require a large number of joins to retrieve the necessary data.

The fact table contains multiple foreign keys that reference the dimension tables. The data in each dimension table is denormalized, making queries fast and efficient.

For instance, the dimTerritory table includes the TerritoryName, TerritoryCountry and TerritoryRegion columns. Together, these columns form the hierarchy TerritoryRegion > TerritoryCountry >TerritoryName.

The more rows, the lower the cardinality(which refers to the number of unique values relative to the number of rows) and the more redundant data. Although data queries are generally faster in a star schema, the schema can also be more prone to data integrity issues and require more disk space than a more normalized structure.

Snow flake schema:

In a snowflake schema, a fact is surrounded by its associated dimensions (as in a star schema), but those dimensions are further related to other dimensions, branching out into a snowflake pattern. Snowflaking normalizes the dimensions by moving attributes with low cardinality into separate dimension tables.

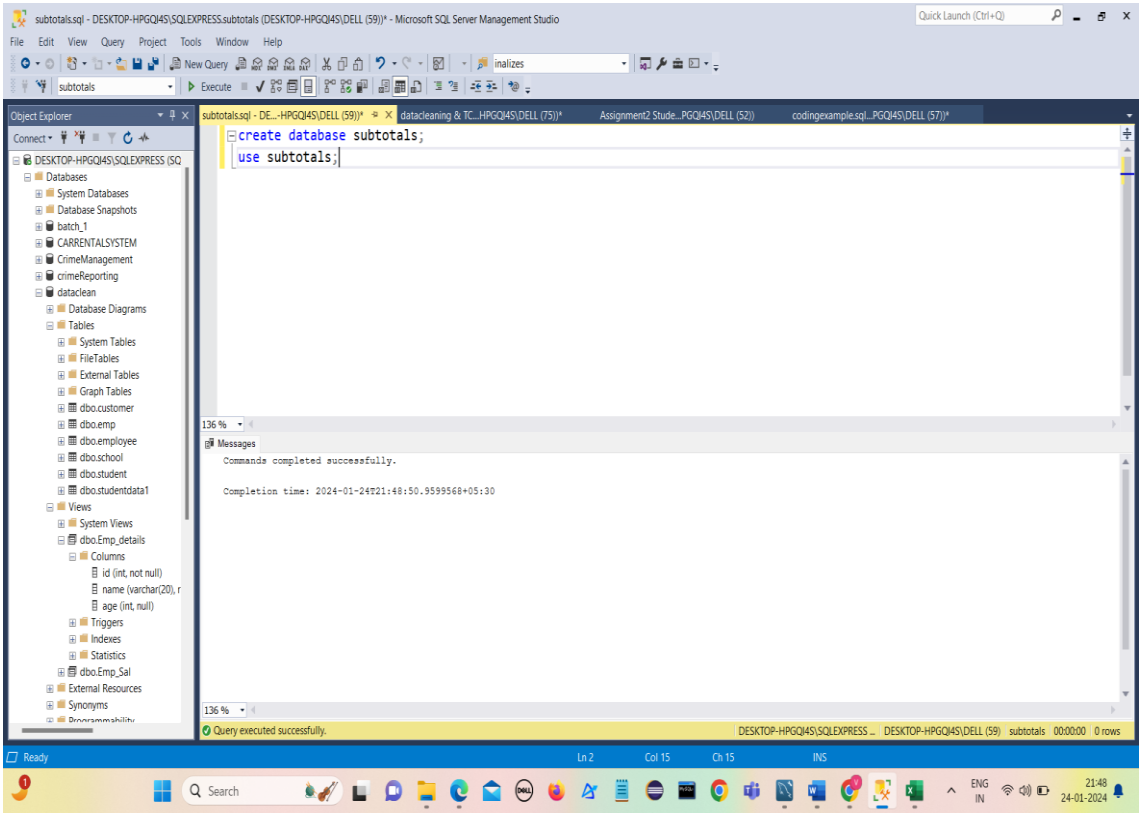
The fact table in a snowflake schema uses foreign keys to reference the core dimensions, which in turn use foreign keys to reference the dimensions at the next level.

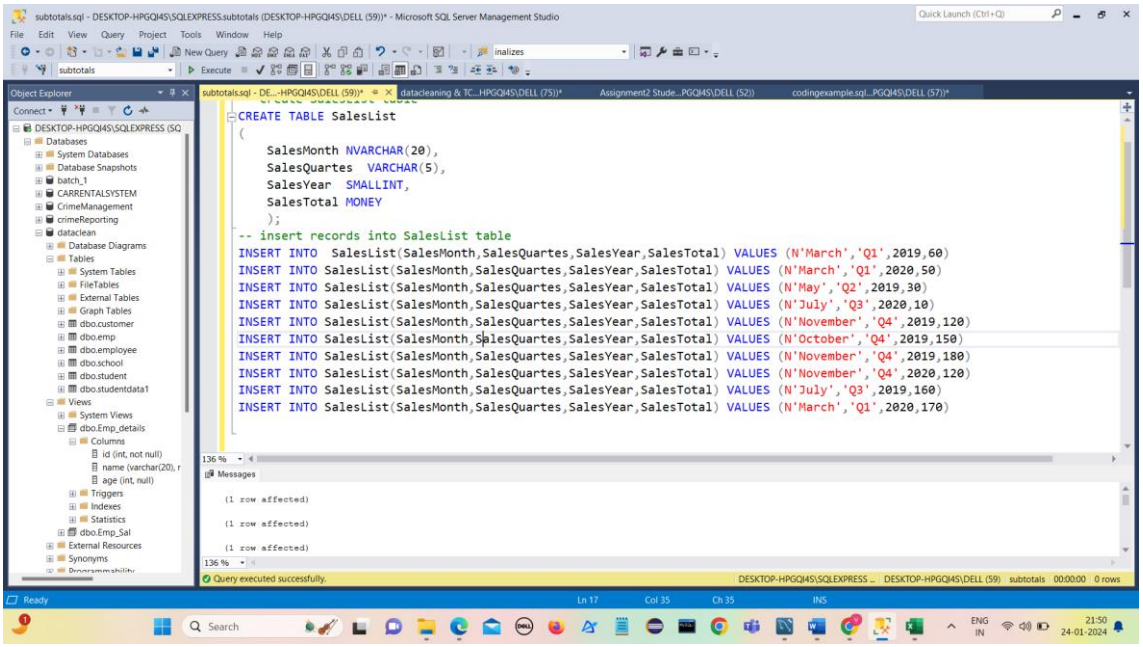
The normalized dimensions in a snowflake schema reduce the amount of redundant data, making them less susceptible to data integrity issues than a star schema.

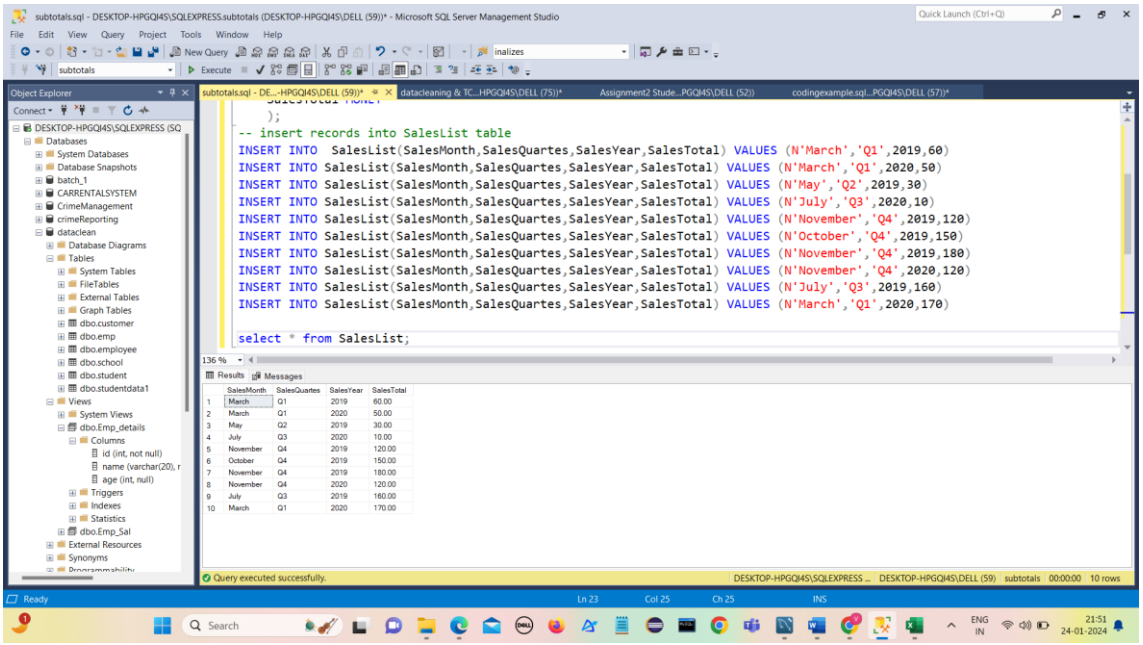
Sub Totals:

A subtotal is a figure that shows the sum of similar sets of data but it does not indicate the final total.

Calculating a subtotal in SQL query can be a bit complicated than the common aggregate queries





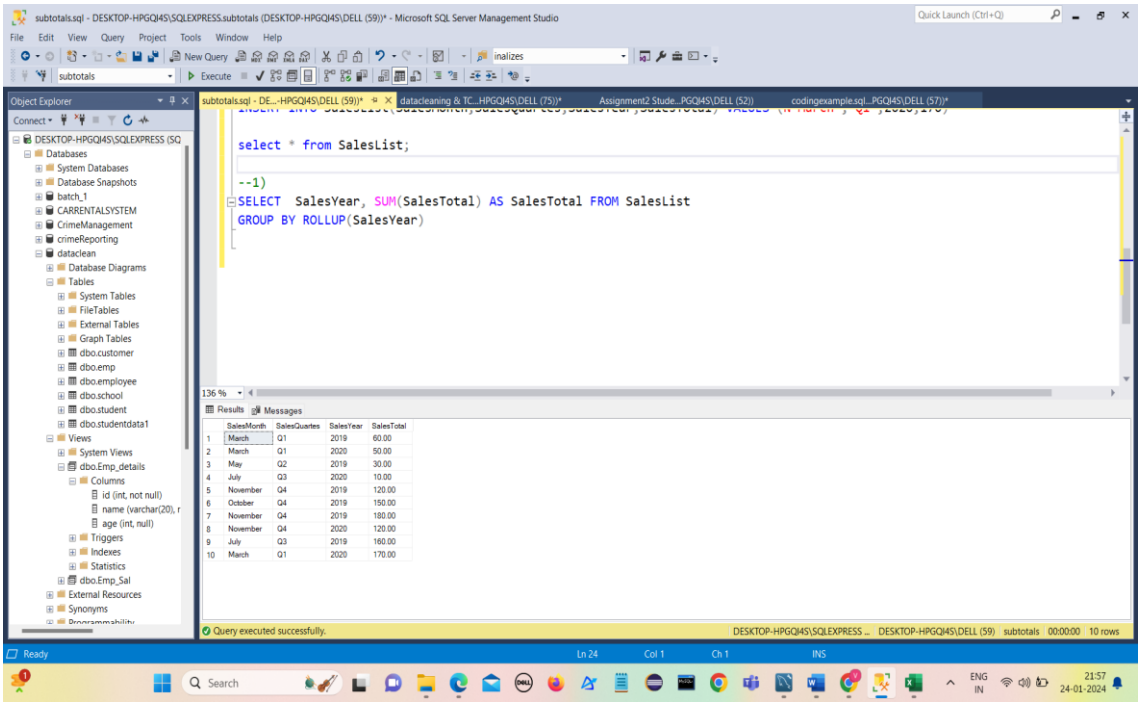


ROLLUP extension

ROLLUP, CUBE, and GROUPING SETS are extensions of the GROUP BY statement and add the extra subtotal and grand total rows to the resultset.

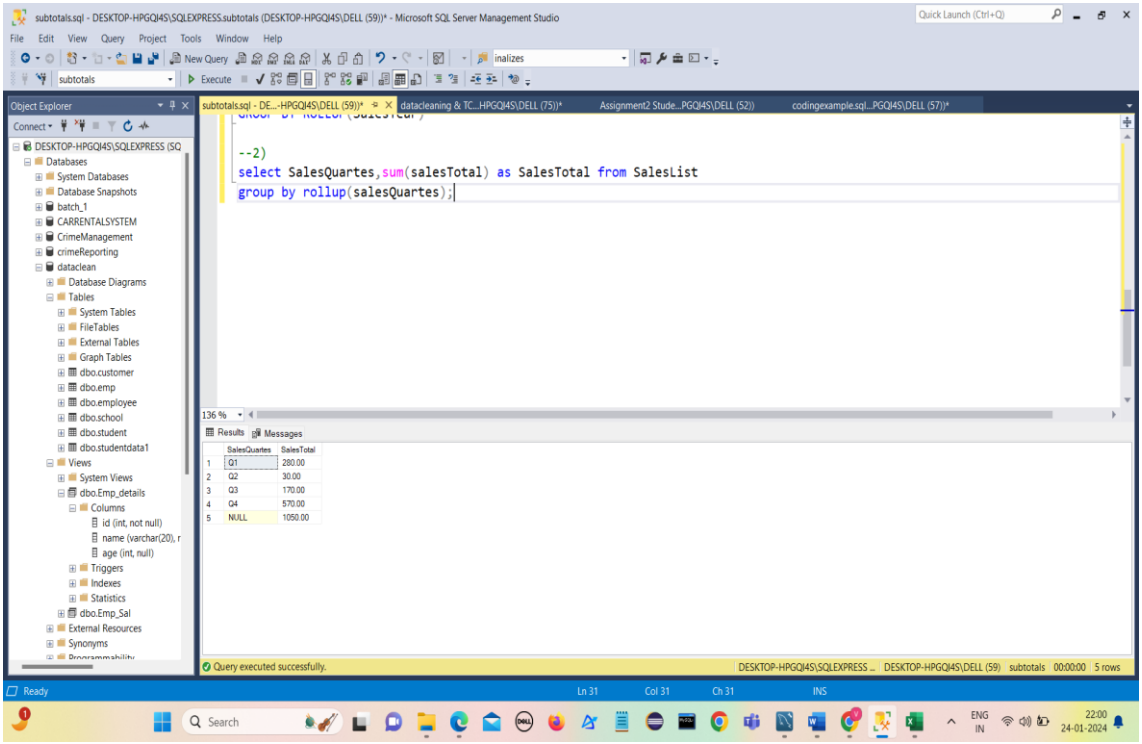
The **ROLLUP** extension allows us to generate hierarchical subtotal rows according to its input columns and it also adds a grand total row to the result set.

1)

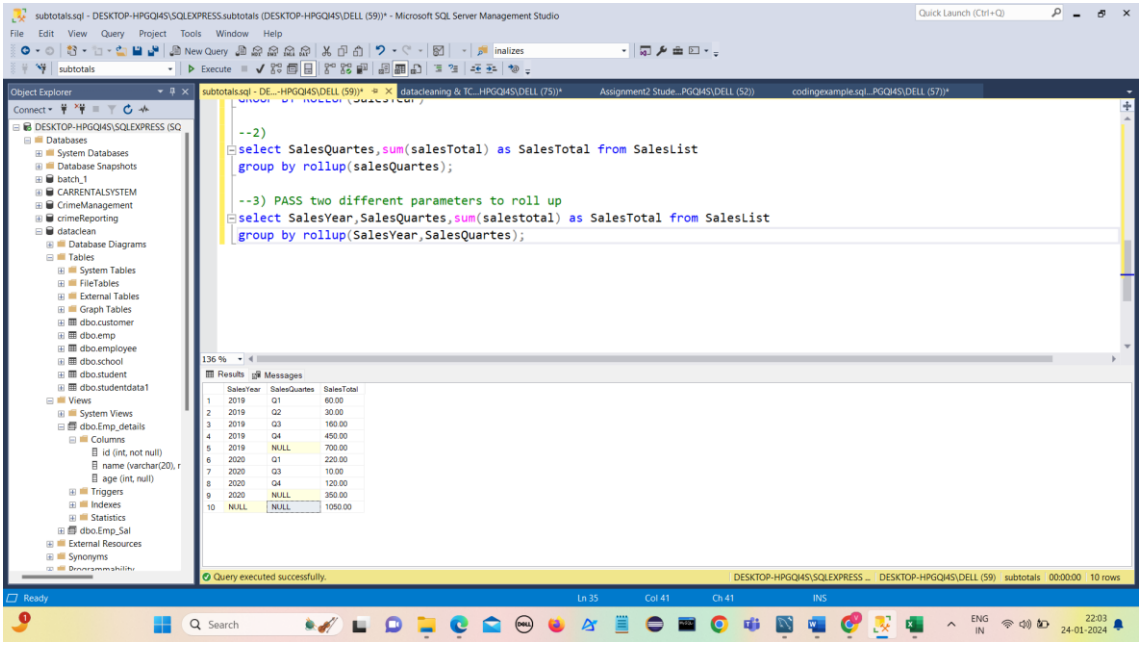


The extra row at the end of the result set, this row shows the grand total sales of the years without considering the sales month and quarters.

2)

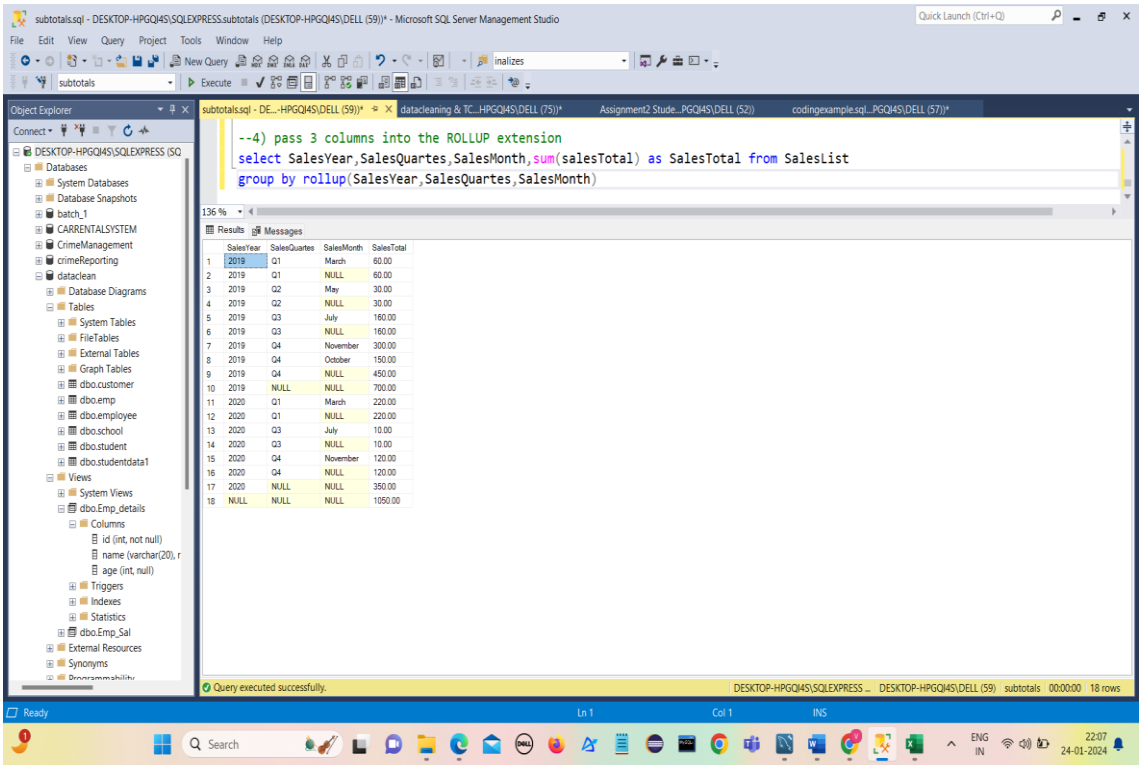


3) pass two different columns as a parameter to the ROLLUP



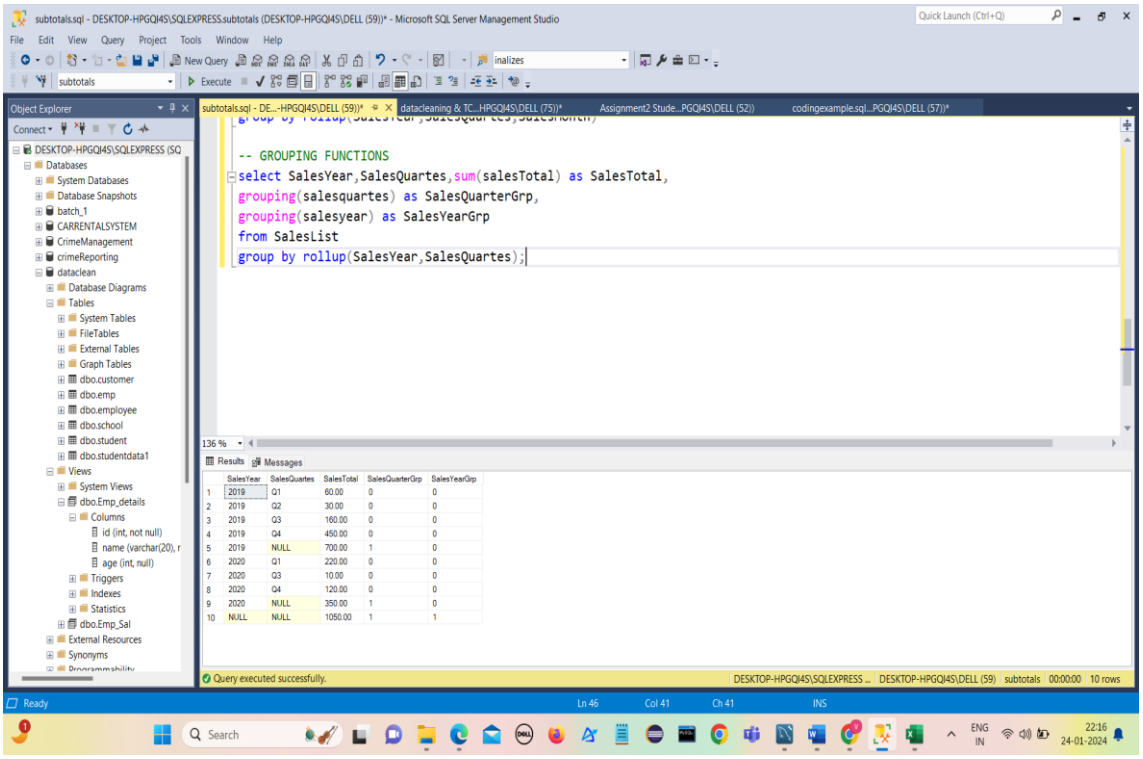
700 and 350 are sub totals by years and 1050 is the grand total.

pass 3 columns into the ROLLUP extension and then this extension will generate subtotal rows for all hierarchies.

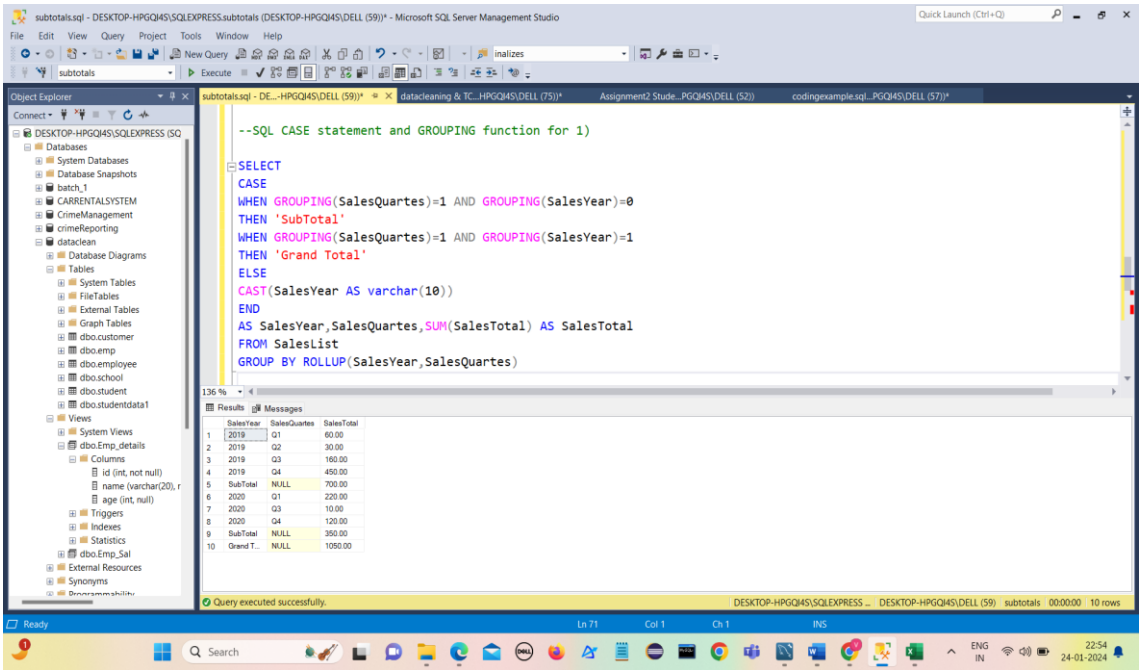


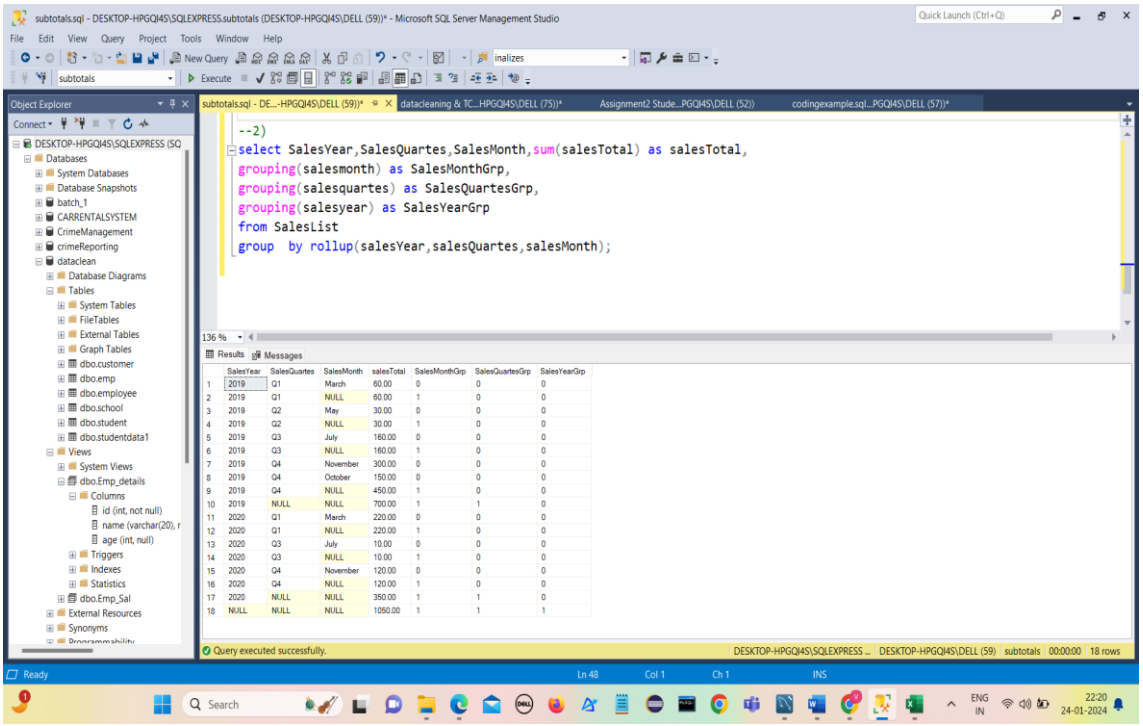
GROUPING function

The GROUPING function is used to determine whether the columns in the GROUP BY list have been aggregated.



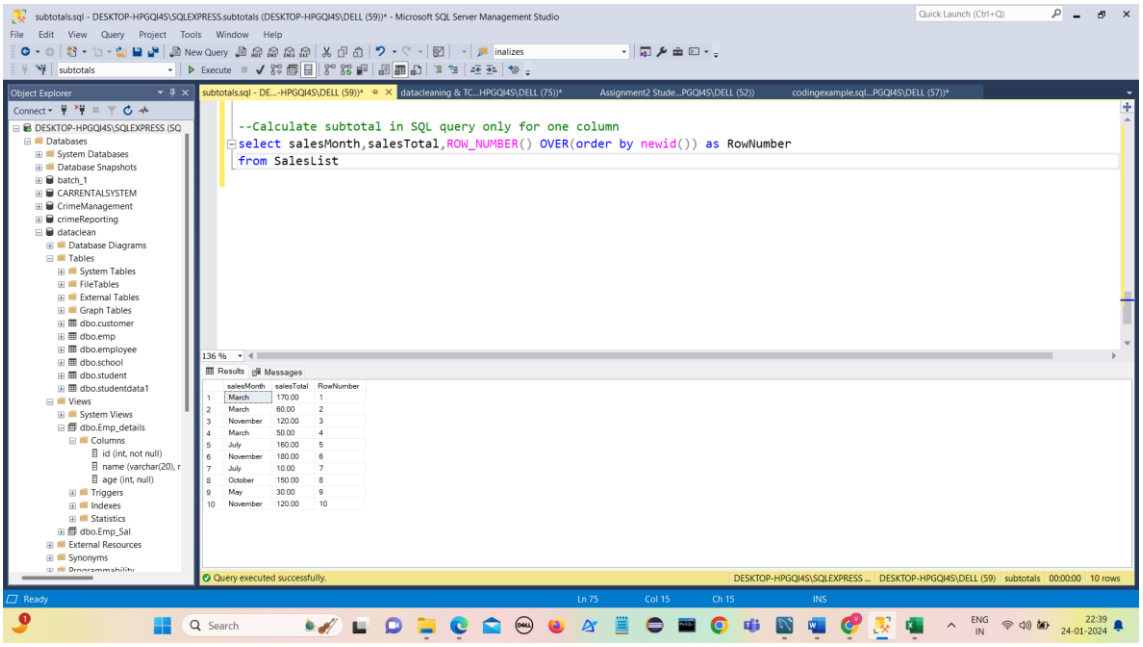
SQL CASE statement and GROUPING function together so that we will replace NULL values with more meaningful explanations.



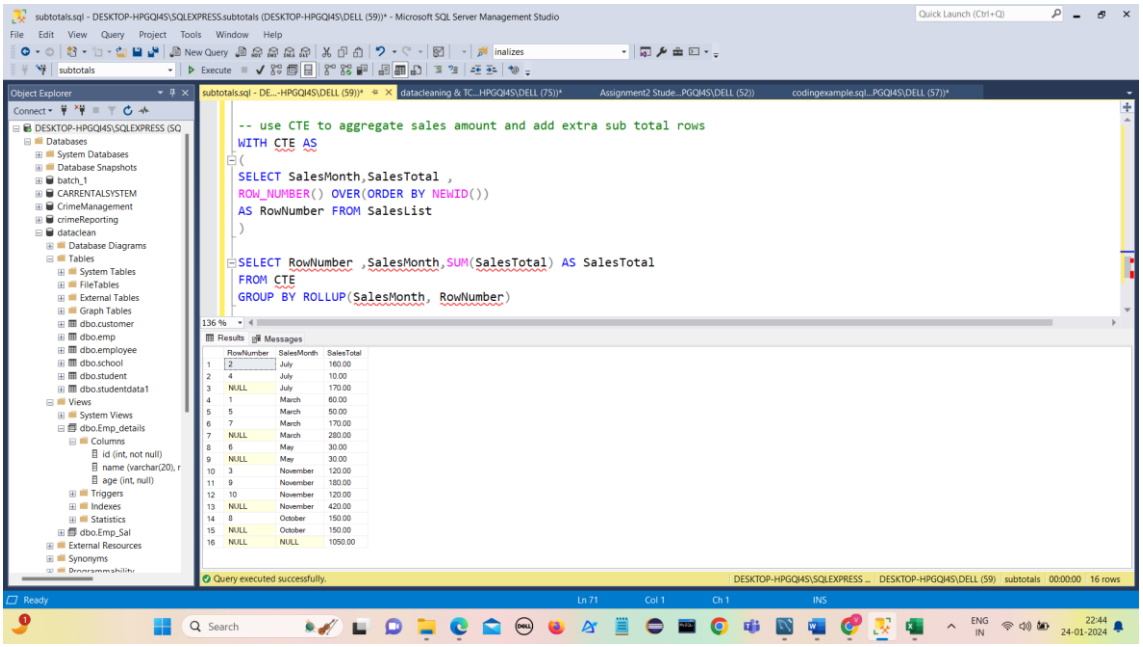


Calculate subtotal in SQL query only for one column

use the ROW_NUMBER() and NEWID() function together to add unique number and this unique number is used for grouping



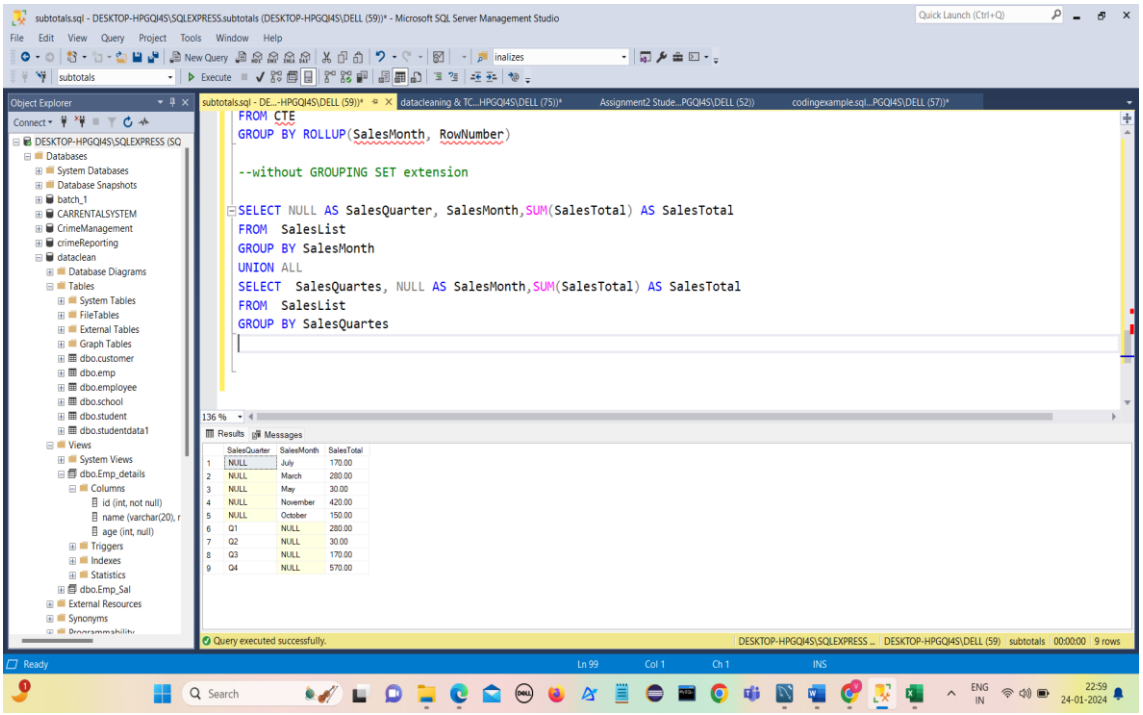
Use CTE (Common Table Expressions) to work on this result set temporarily to aggregate sales amount and adding the extra subtotals rows.



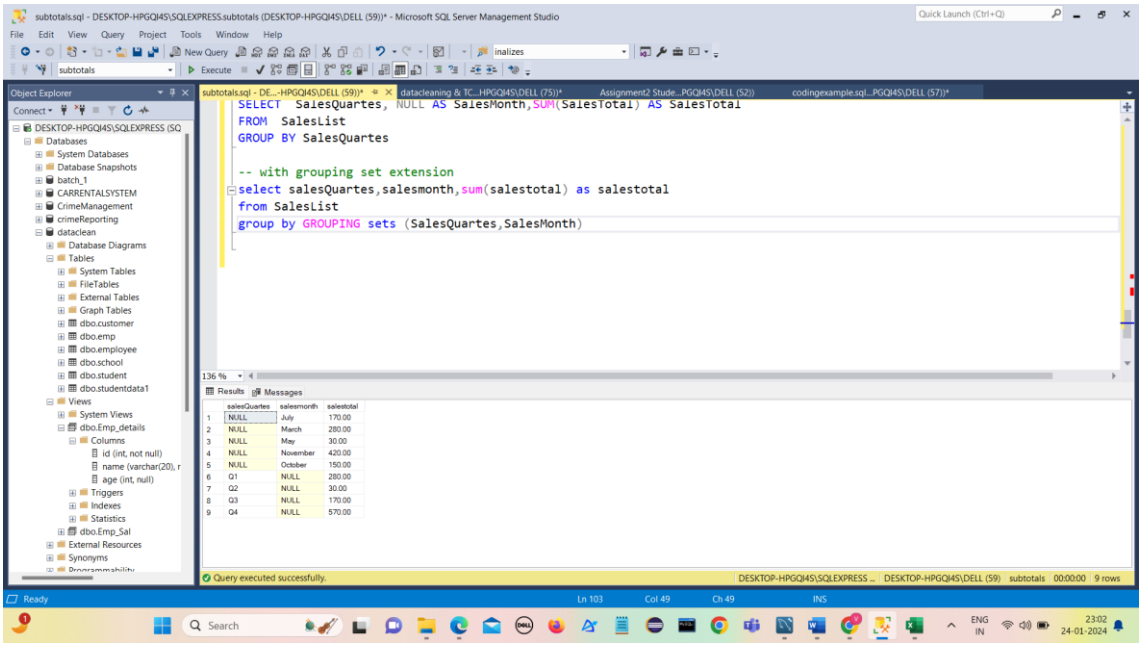
GROUPING SET extension

It is a GROUP BY extension and it allows us to display multiple grouping set in one query.

Without using group set extension



Using grouping set extension



use GROUPING SETS to add subtotal in SQL query