Asmita Porwal
Data Engineering
Batch-1
25/1/24

Coding Challenge -1 Question-1

1 a. Execute OVER and PARTITION BY Clause in SQL Queries

Over

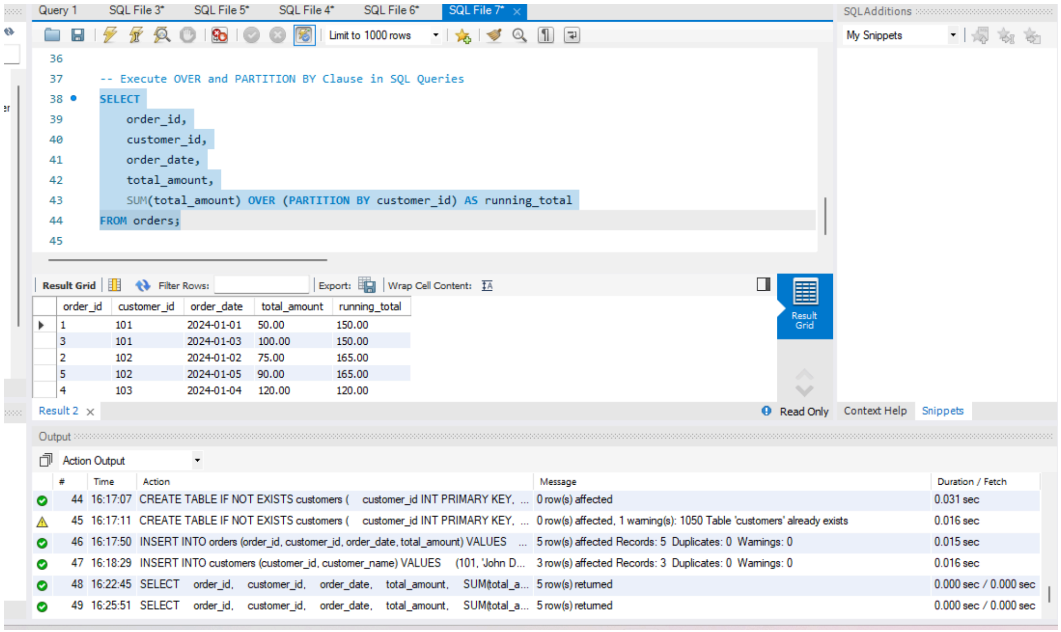
The Over clause defines the window of rows over which a window function operates. It allows you to define a window of rows related to the current row. This can include all rows in the result set, or a subset defined by the Partition by.

Partition By

The Partition By clause divides the result set into partitions to which the window function is applied.
It operates independently within each partition. It's used to group rows based on one or more columns, and the window function is applied separately to each partition.

SUM(total_amount): This is a window function that calculates the sum of the "total_amount" column.

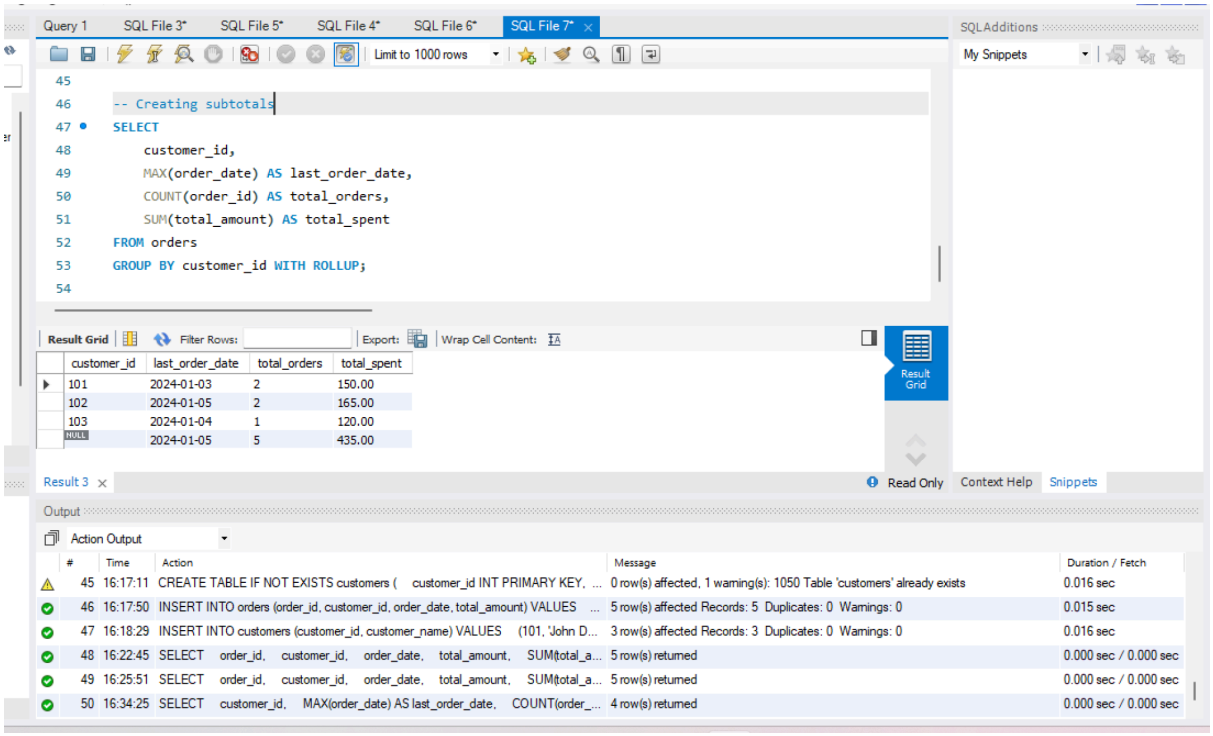
OVER (PARTITION BY customer_id): This part of the query specifies that the summation should be done separately for each unique value in the "customer_id" column. It creates a partition for each customer.



1 b. creating subtotals

ROLLUP: Adds extra rows to the result set to represent subtotals.
The extra rows will have NULL values in the columns used in the GROUP BY clause.

This query calculates the last order date, total number of orders, and total amount spent for each customer, along with subtotals for each unique customer and a grand total for the entire result set.



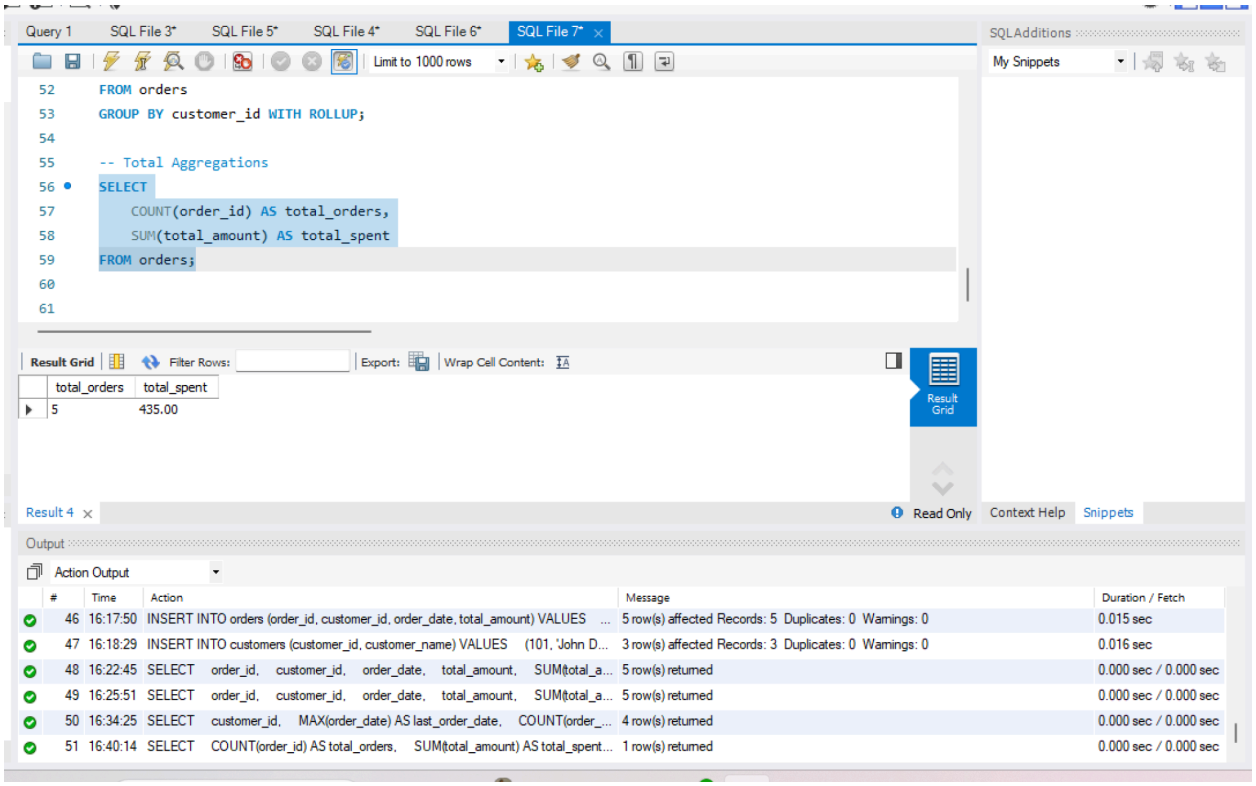
1 c. Total Aggregations using SQL Queries.

These are some aggregate functions:

- **COUNT :** counts how many rows are in a particular column.
- **SUM :** adds together all the values in a particular column**.**
- **MIN and MAX :** return the lowest and highest values in a particular column, respectively.
- **AVG :** calculates the average of a group of selected values.

I have used **count()** for calculating total orders

And **Sum()** for total money spent on all orders.


Query 1
SQL File 3*
SQL File 5*
SQL File 4*
SQL File 6*
SQL File 7*
Limit to 1000 rows
SQLAdditions
My Snippets
52 FROM orders
53 GROUP BY customer_id WITH ROLLUP;
54
55 -- Total Aggregations
56 SELECT
57 COUNT(order_id) AS total_orders,
58 SUM(total_amount) AS total_spent
59 FROM orders;
60
61
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
total_orders
total_spent
5
435.00
Result Grid
Result 4
Read Only
Context Help
Snippets
Output
Action Output
Time Action Message Duration / Fetch
46 16:17:50 INSERT INTO orders (order_id, customer_id, order_date, total_amount) VALUES ... 5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0 0.015 sec
47 16:18:29 INSERT INTO customers (customer_id, customer_name) VALUES (101, 'John D... 3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0 0.016 sec
48 16:22:45 SELECT order_id, customer_id, order_date, total_amount, SUM(total_a... 5 row(s) returned 0.000 sec / 0.000 sec
49 16:25:51 SELECT order_id, customer_id, order_date, total_amount, SUM(total_a... 5 row(s) returned 0.000 sec / 0.000 sec
50 16:34:25 SELECT customer_id, MAX(order_date) AS last_order_date, COUNT(order_... 4 row(s) returned 0.000 sec / 0.000 sec
51 16:40:14 SELECT COUNT(order_id) AS total_orders, SUM(total_amount) AS total_spent... 1 row(s) returned 0.000 sec / 0.000 sec