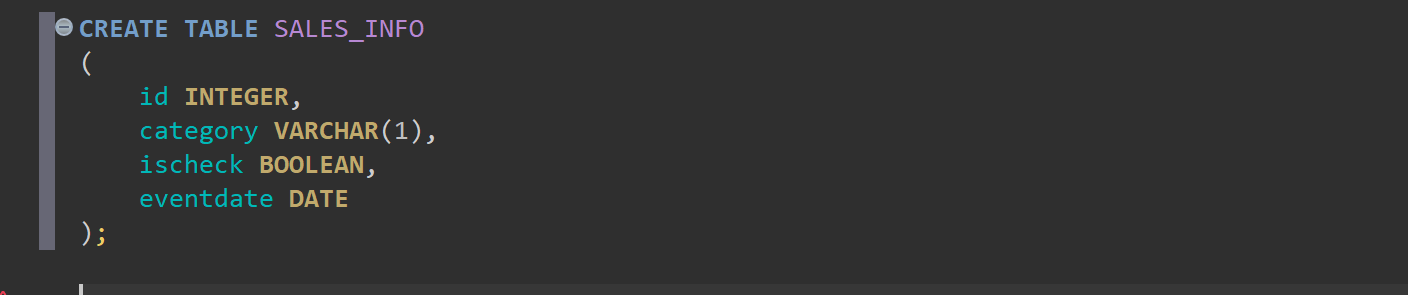
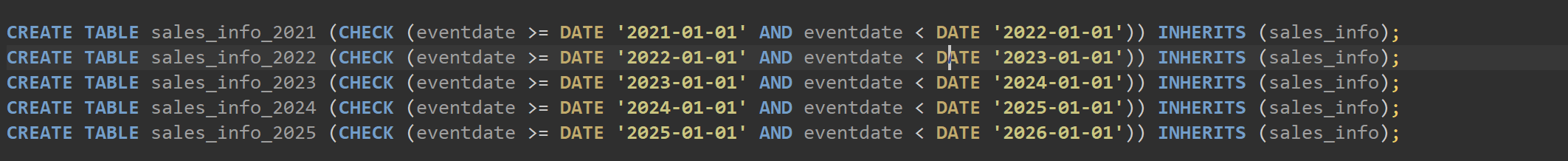
* 1. TASK 1: USE INHERITANCE

Create table:

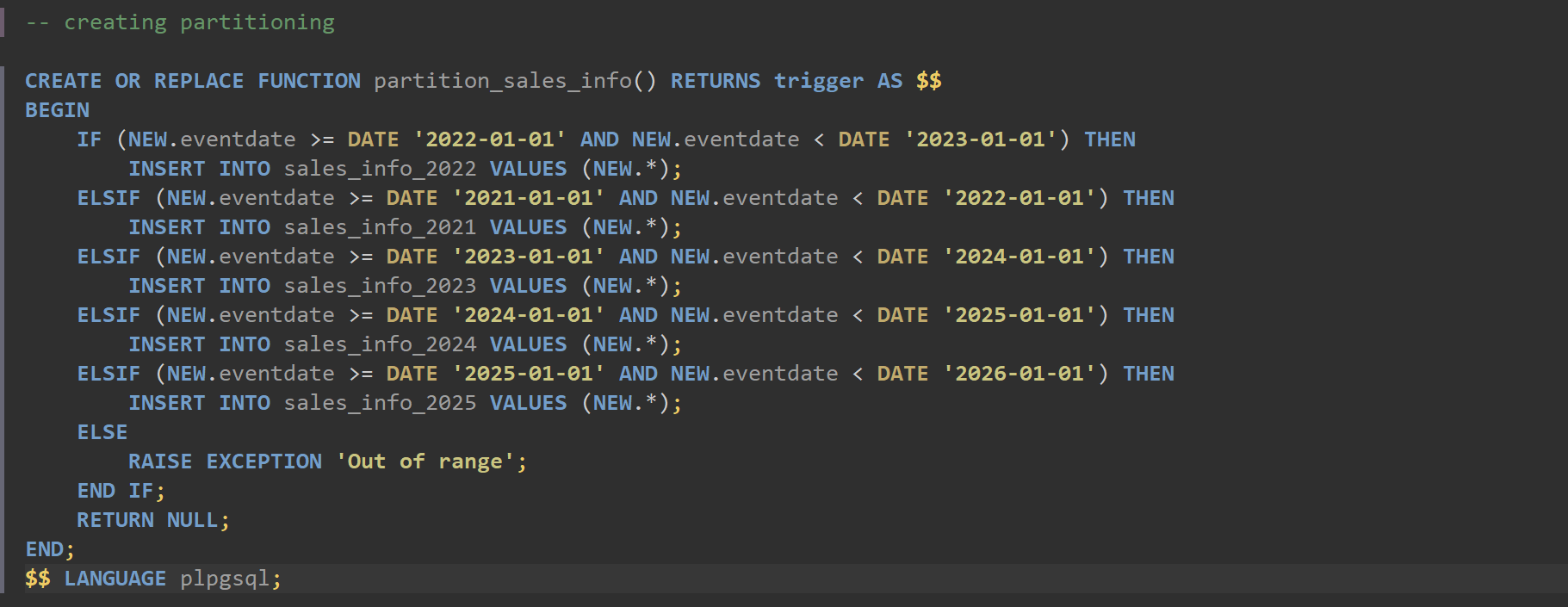


Apply partitioning by using inheritance:

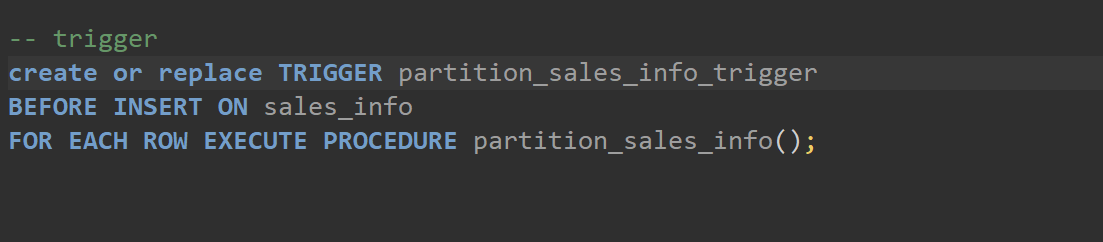
1. Create 4-5 child tables with partitioning by eventdate column. One partition is one year.



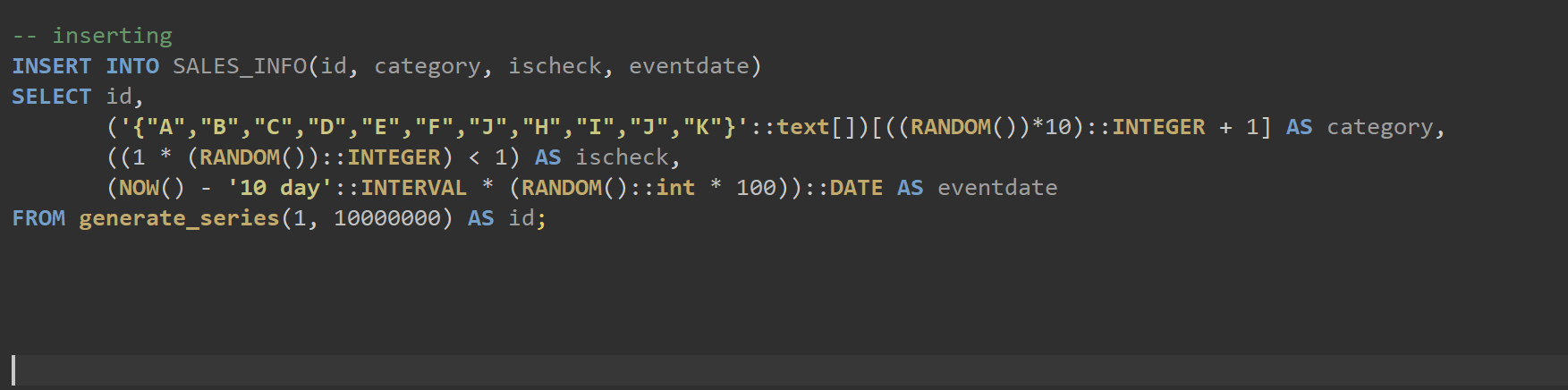
2. Create partition function for your tables. Use following as a template:



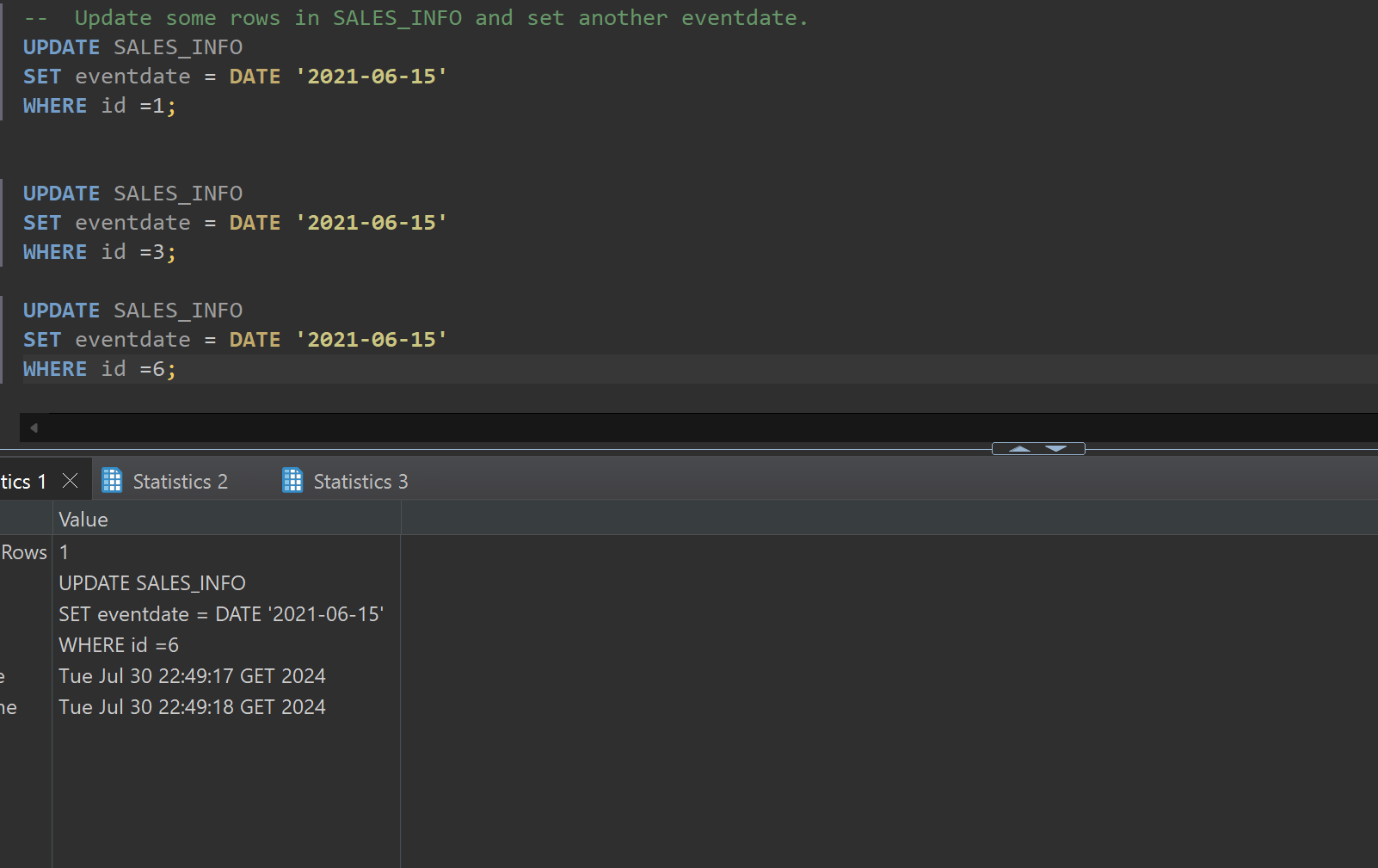
3. Create trigger for your function and tables. Use following as a template:



4. Generate test data and insert in SALES\_INFO table:



5. Update some rows in SALES\_INFO and set another eventdate.



6. Create table SALES\_INFO\_SIMPLE with the same structure as SALES\_INFO but without partitioning. Insert test data from the 5th step. Compare plans of different queries:

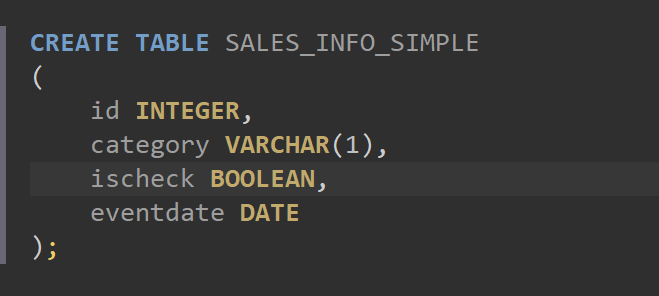
• Select all

• Select with range of dates

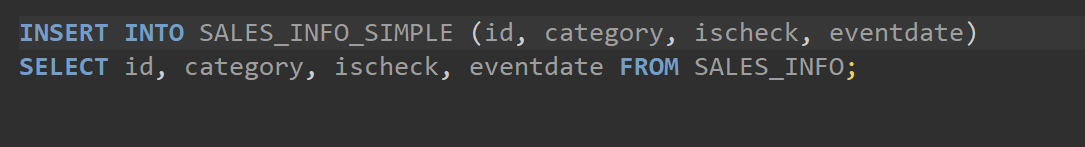
• Select exact date

• Count of all rows

• Count of rows with range of dates



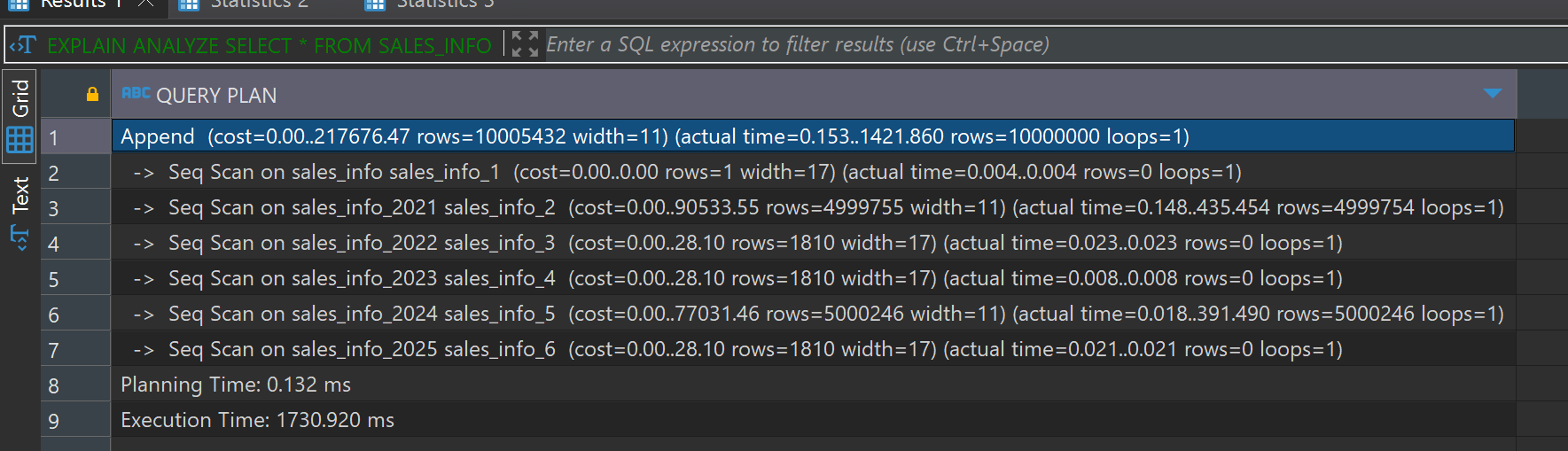
Inserting from first table into this one

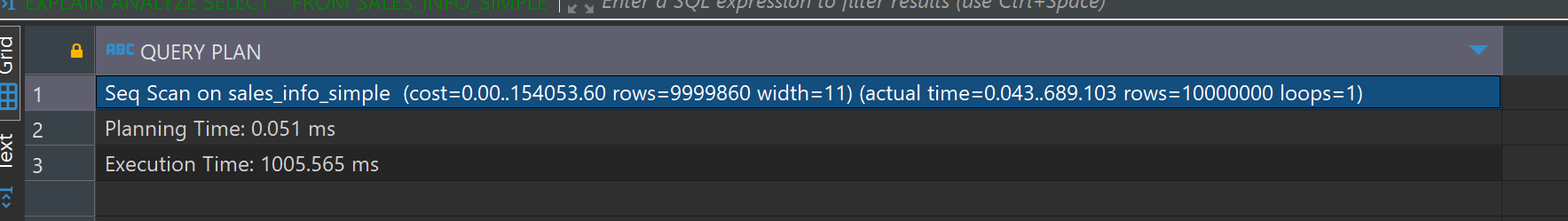


Comparing:

**EXPLAIN** **ANALYZE** **SELECT** \* **FROM** SALES\_INFO;

**EXPLAIN** **ANALYZE** **SELECT** \* **FROM** SALES\_INFO\_SIMPLE;





First of all, the non-partitioned table has a lower execution time) compared to the partitioned table for the SELECT \* query. SELECT \* query on the partitioned table has to scan multiple partitions and then combine the results, whereas the non-partitioned table performs a single scan.

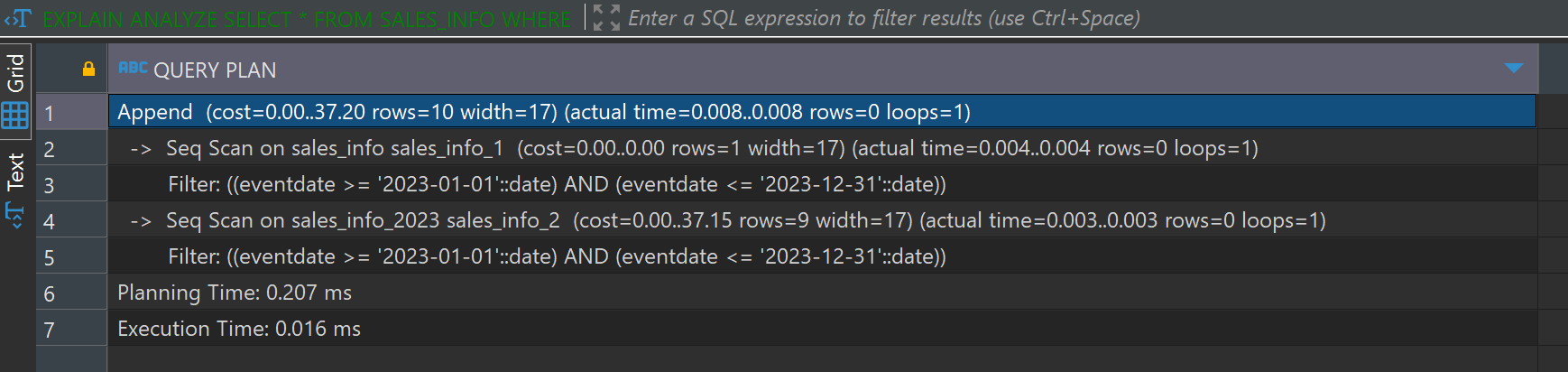
The planning time for the partitioned table is little bit higher than that of the non-partitioned table

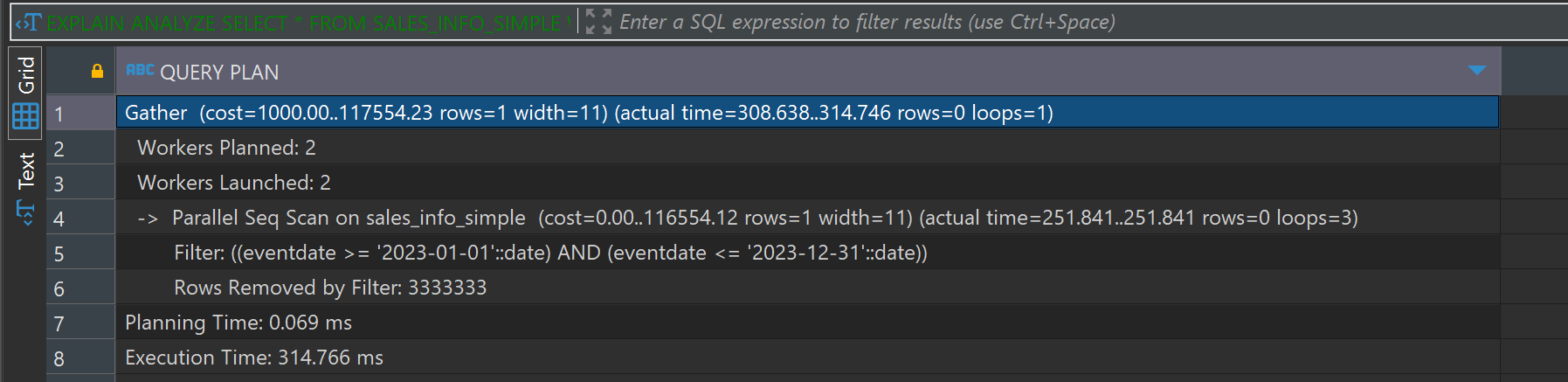
The non-partitioned table performs better in terms of execution time for full table scans, as it involves a single sequential scan.

-- select with range of dates

**EXPLAIN** **ANALYZE** **SELECT** \* **FROM** SALES\_INFO **WHERE** eventdate **BETWEEN** **'2023-01-01'** **AND** **'2023-12-31'**;

**EXPLAIN** **ANALYZE** **SELECT** \* **FROM** SALES\_INFO\_SIMPLE **WHERE** eventdate **BETWEEN** **'2023-01-01'** **AND** **'2023-12-31'**;



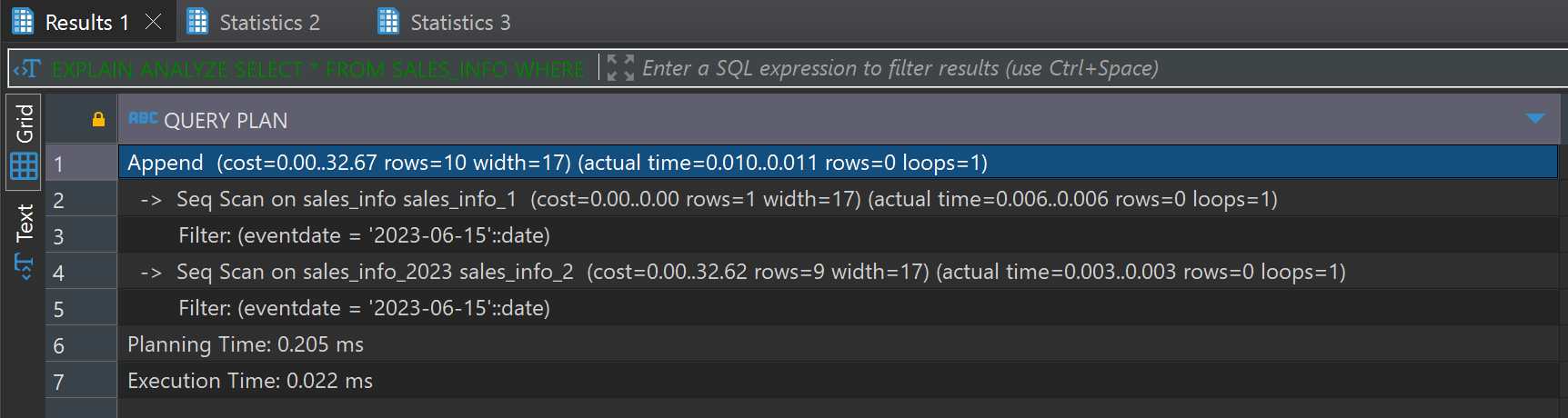


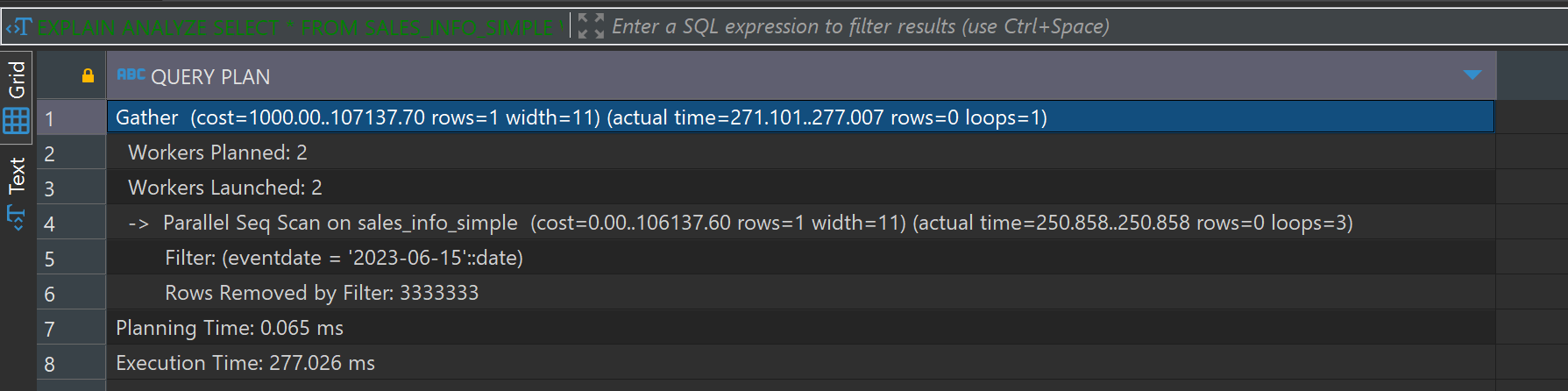
Partitioning provides significant performance benefits for range queries by limiting the scan to relevant partitions. This results in faster execution times and more efficient data retrieval. The non-partitioned table has to scan the entire table which results in higher execution times. That’s why partitioned table has a significantly lower execution time compared to the non-partitioned table for the range query.

-- Select exact date

**EXPLAIN** **ANALYZE** **SELECT** \* **FROM** SALES\_INFO **WHERE** eventdate = **'2023-06-15'**;

**EXPLAIN** **ANALYZE** **SELECT** \* **FROM** SALES\_INFO\_SIMPLE **WHERE** eventdate = **'2023-06-15'**;

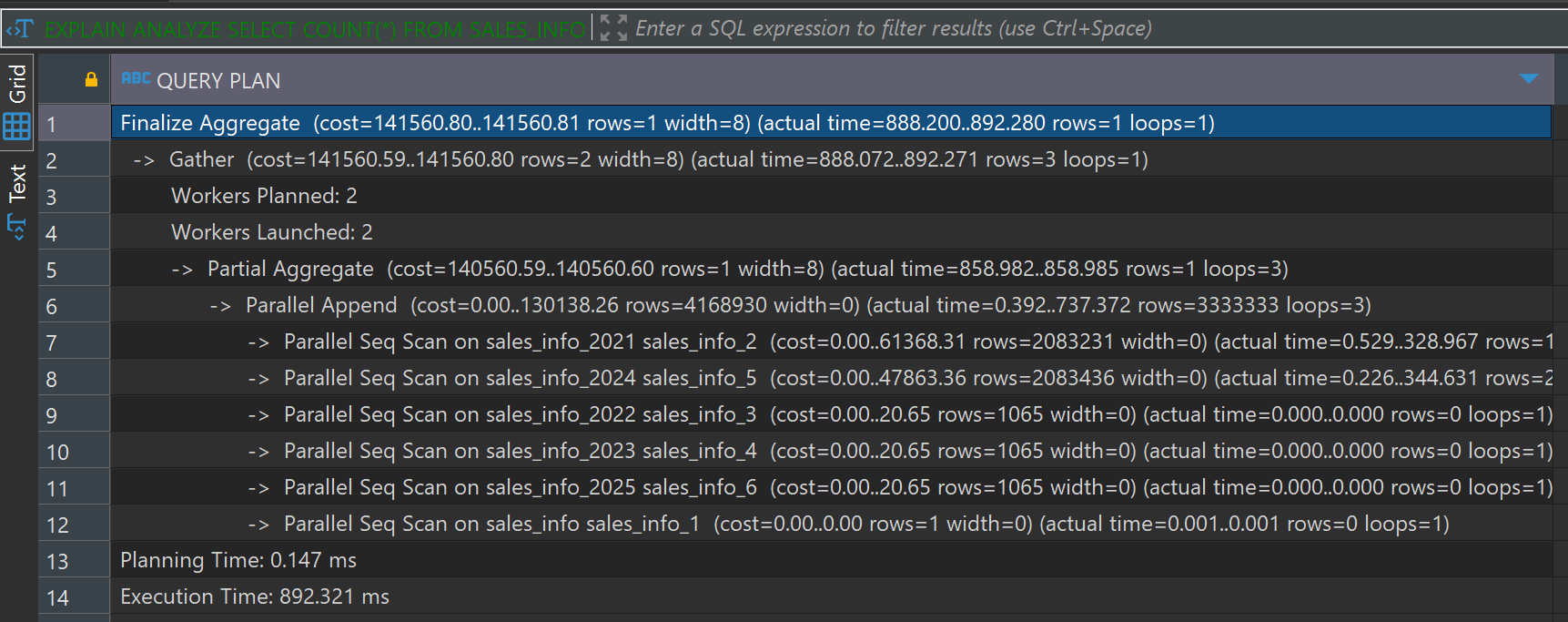


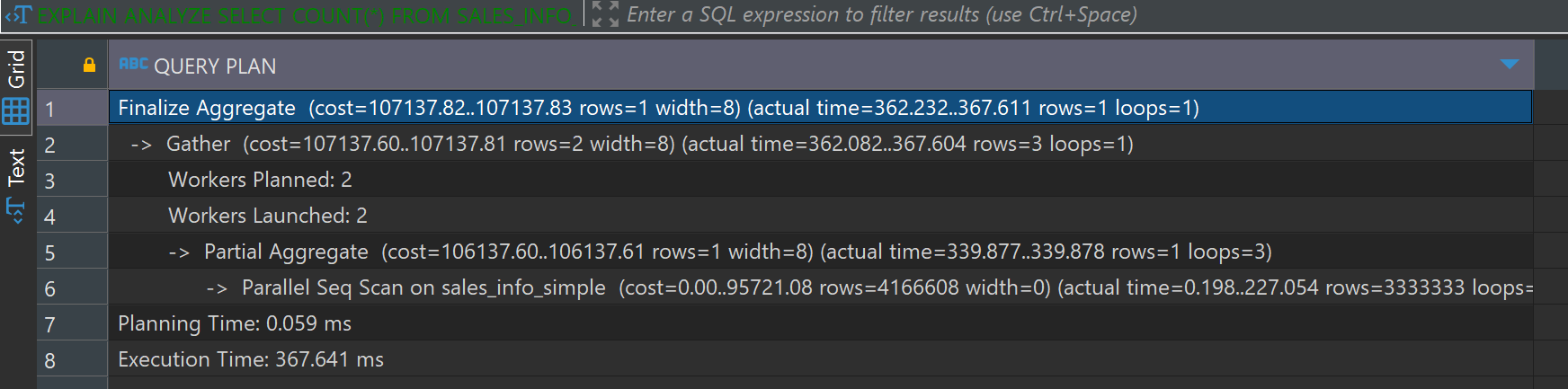


The partitioned table runs faster for the exact date query because it only looks at the needed partitions, reducing the data to scan. However, it takes a bit more time to plan the query because it has to figure out which partitions to use. The partitioned table is more efficient as it scans just the relevant partition (sales\_info\_2023). On the other hand, the non-partitioned table scans the entire table, which takes more time since it has to go through more data.

**EXPLAIN** **ANALYZE** **SELECT** **COUNT**(\*) **FROM** SALES\_INFO;

**EXPLAIN** **ANALYZE** **SELECT** **COUNT**(\*) **FROM** SALES\_INFO\_SIMPLE;





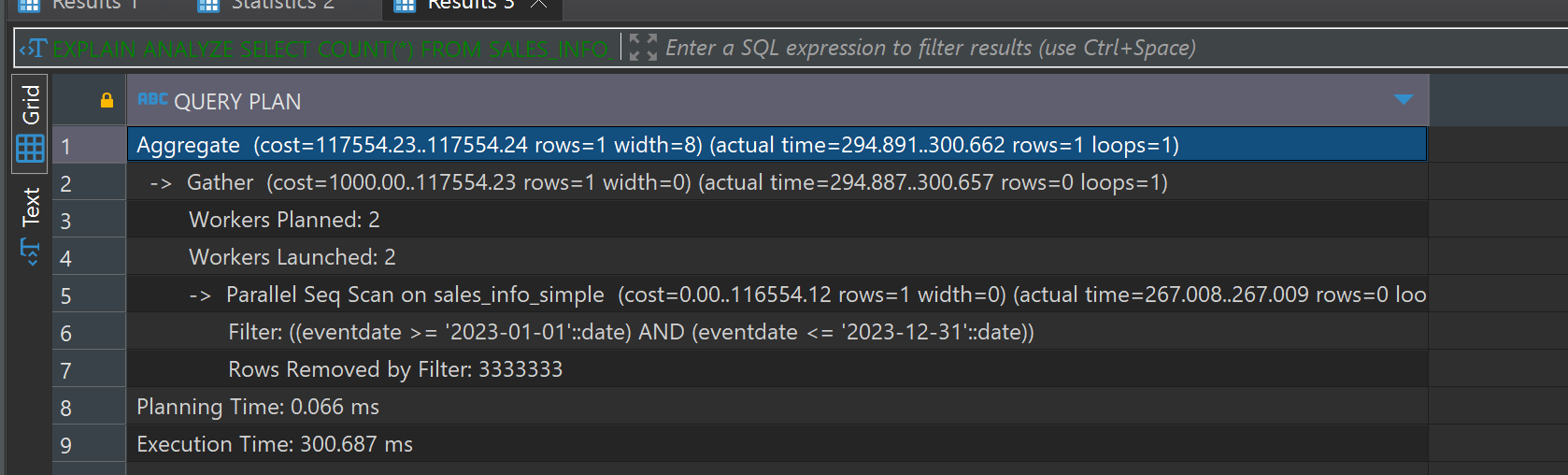
The non-partitioned table has a lower execution compared to the partitioned table. This is because the partitioned table needs to combine results from multiple partitions, which adds overhead because of it partitioned table has a higher planning time compared to the non-partitioned table. This is due to the query planner having to identify and prepare for scanning multiple partitions. the partitioned table involves multiple Parallel Seq Scan operations on each partition and that increases the total execution time. The non-partitioned table, although scanning the entire table, benefits from a more straightforward and unified scan process.

-- Count of rows with range of dates

**EXPLAIN** **ANALYZE** **SELECT** **COUNT**(\*) **FROM** SALES\_INFO **WHERE** eventdate **BETWEEN** **'2023-01-01'** **AND** **'2023-12-31'**;

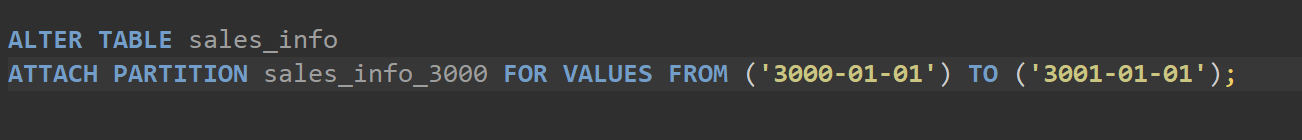
**EXPLAIN** **ANALYZE** **SELECT** **COUNT**(\*) **FROM** SALES\_INFO\_SIMPLE **WHERE** eventdate **BETWEEN** **'2023-01-01'** **AND** **'2023-12-31'**;





The SALES\_INFO has a significantly lower execution time for counting rows within a date range, as it only scans the relevant partitions. The non-partitioned table has a much higher execution time since it performs a full table scan and applies the date filter to all rows. The partitioned approach is more efficient for range queries due to targeted scans of relevant data segments.

I dropped the table sales\_info\_2021 and then created new table sales\_info\_3000



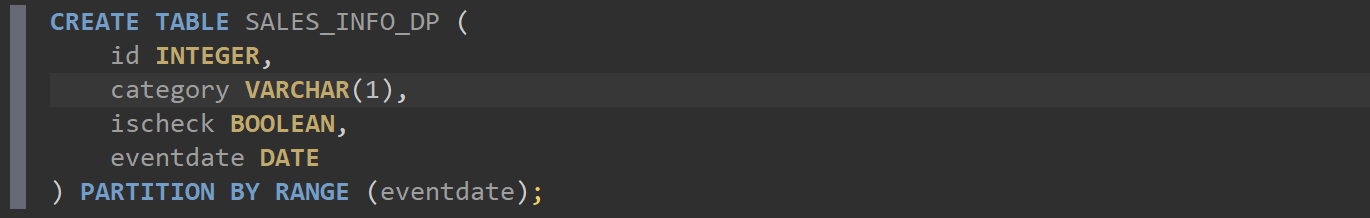
But I throws error;

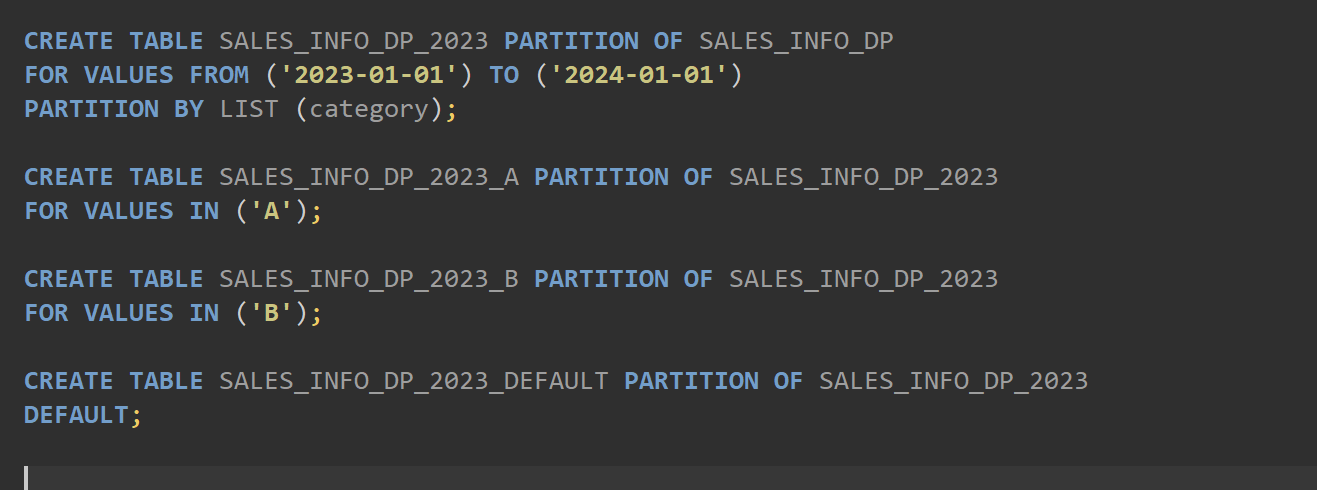
Table is not partitioned,

I partitioned by creating new table and it works

* 1. TASK2: USE DECLARATIVE PARTITIONING

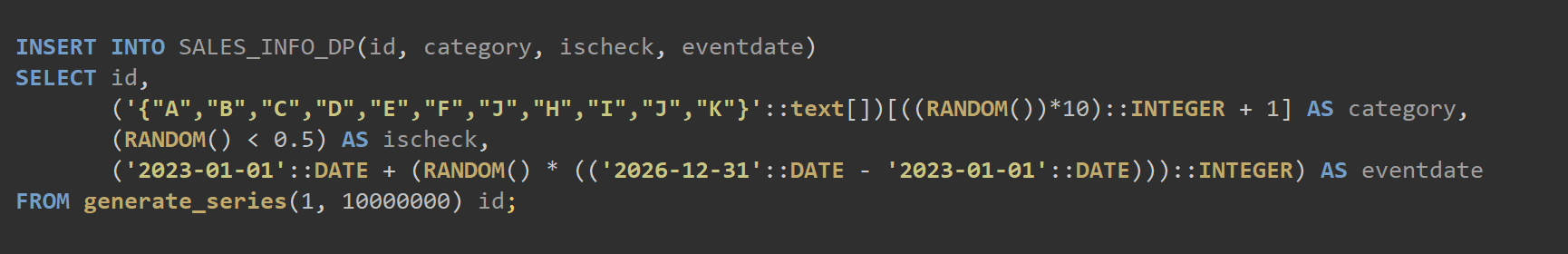
1. Create the Parent Table

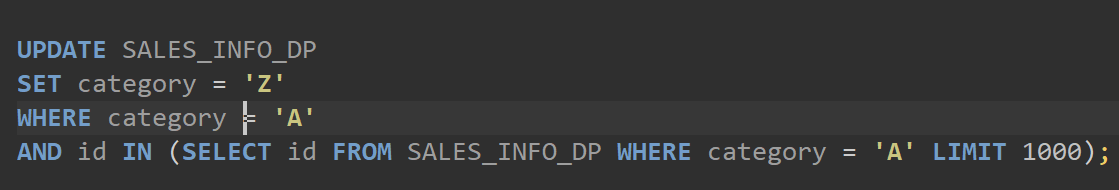


1. Create Child Tables 

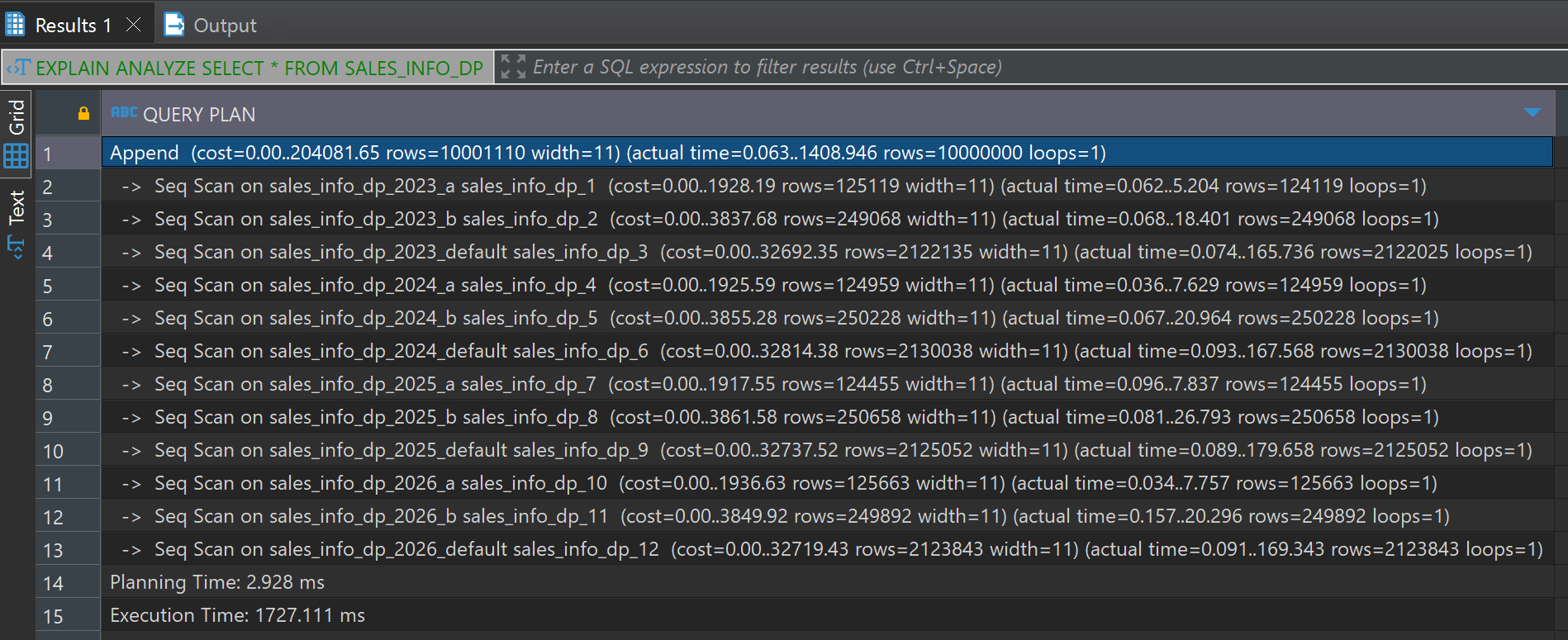
Same for 2024, 2025, 2026.

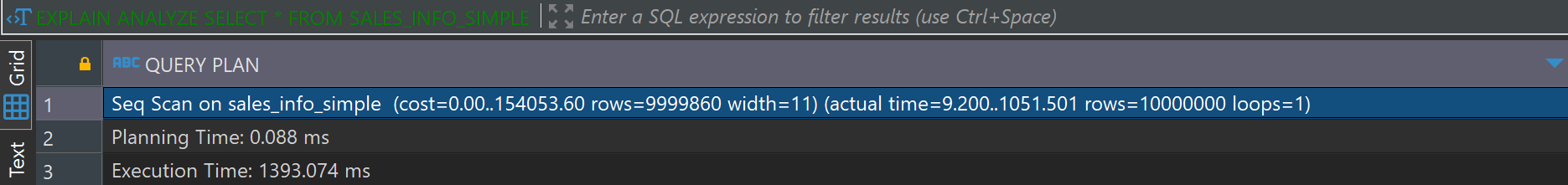
Now inserting



Updating only 1000 rows 

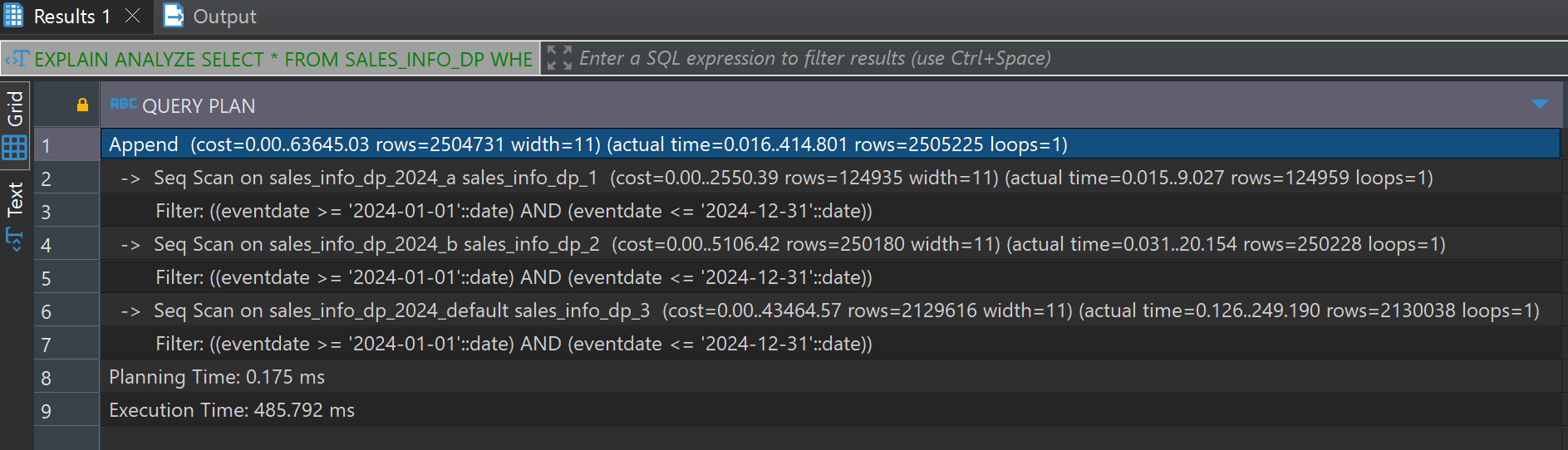
Select all for partitioning and non partitioningS

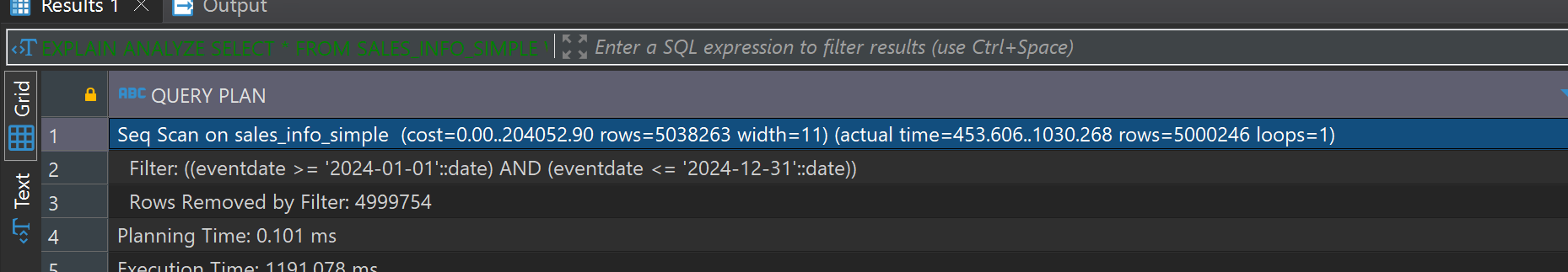




The execution plan comparison shows that the partitioned table performs sequential scans across multiple partitions, resulting in an execution time high. In contrast, the non-partitioned table performs a single sequential scan with a slightly lower execution time of approximately This indicates that while partitioning can improve query performance in certain cases, for a full table scan, the overhead of managing multiple partitions may result in longer execution times.

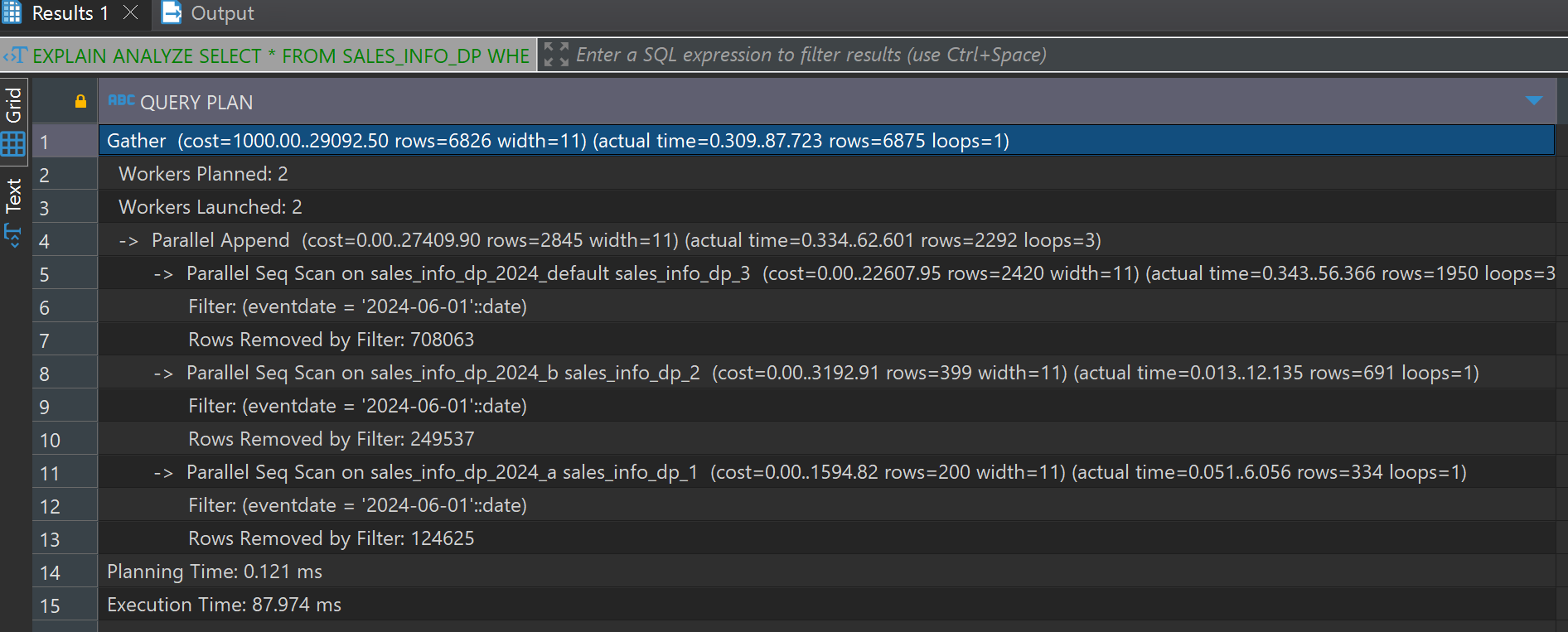
Select with Range of Dates

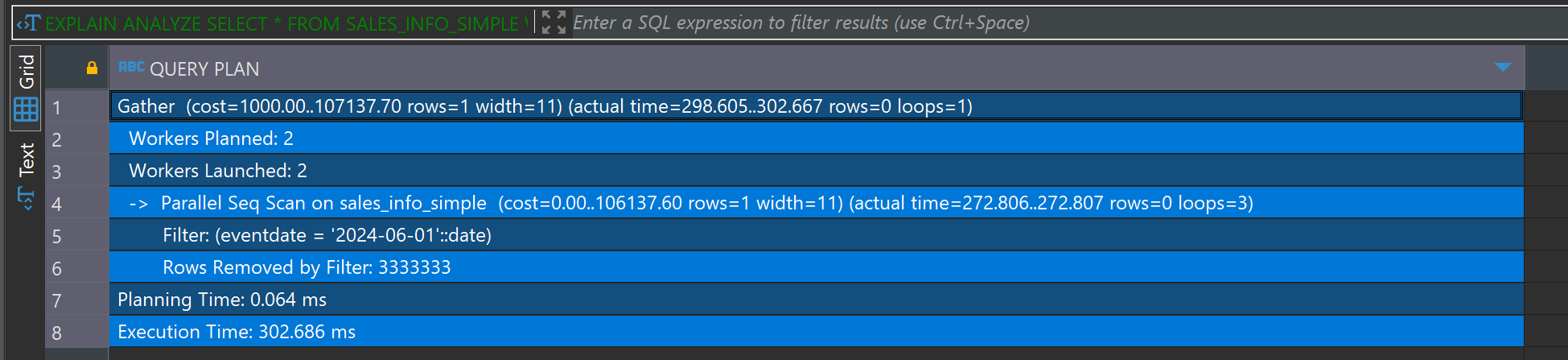




The execution plan comparison for the range date query shows that SALES\_INFO\_Dp performs sequential scans on relevant partitions, which results in lower execution time. In contrast, SALES\_INFO\_SIMPLE performs a single sequential scan filtering the rows, with a significantly longer execution time. This means the efficiency of partition pruning in the partitioned table, which reduces the amount of data scanned and improves query performance for date ranges.

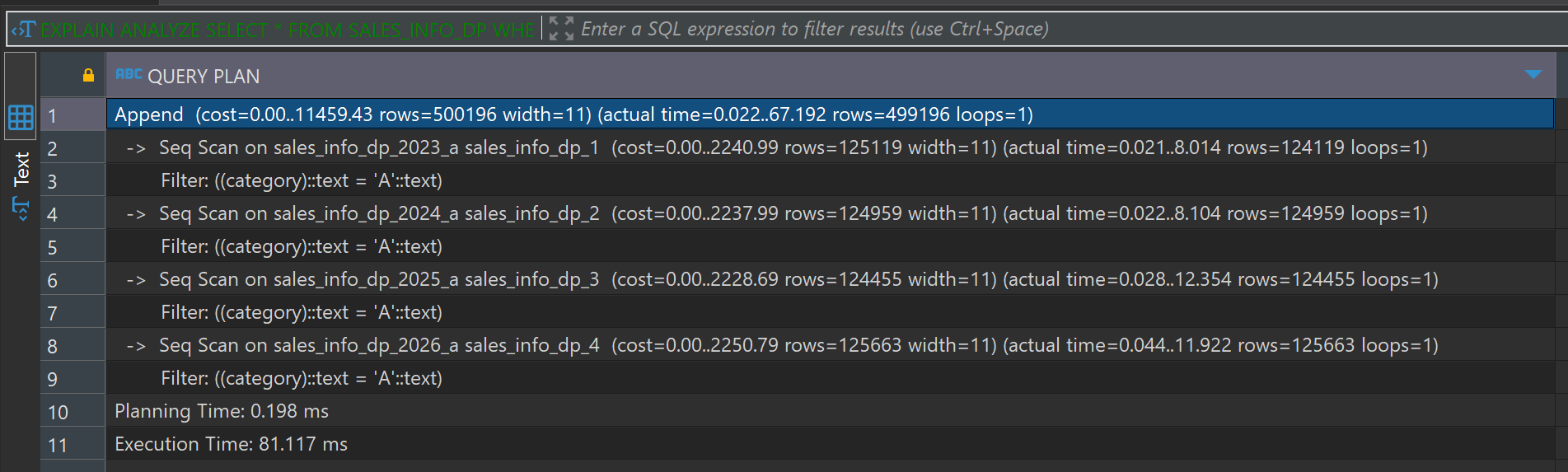
Select exact dates

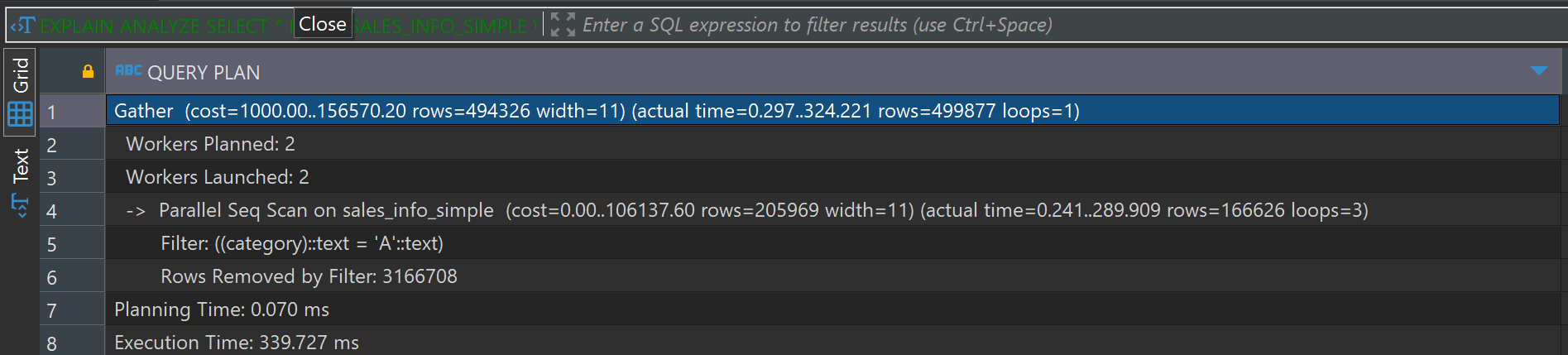




The execution plan comparison for the exact date query shows that the SALES\_INFO\_DP leverages parallel sequential scans across relevant partitions, resulting in an execution time of approximately 88 ms. In contrast, the SALES\_INFO\_SIMPLE also uses parallel sequential scans but takes significantly longer, with an execution time of approximately 303 ms. This indicates that partition pruning and parallel execution in the partitioned table provide a substantial performance advantage for queries on specific dates.

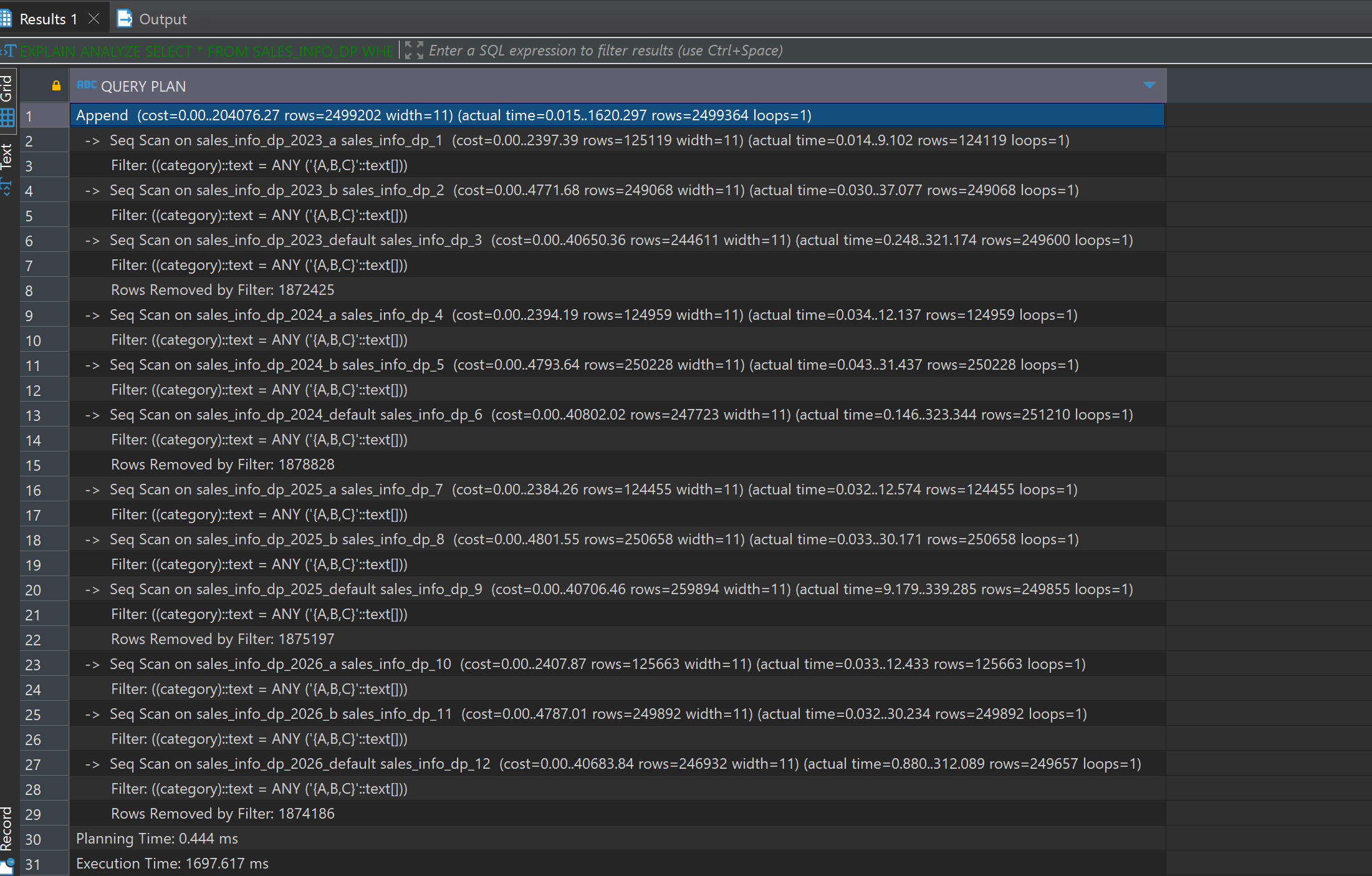
Select categories

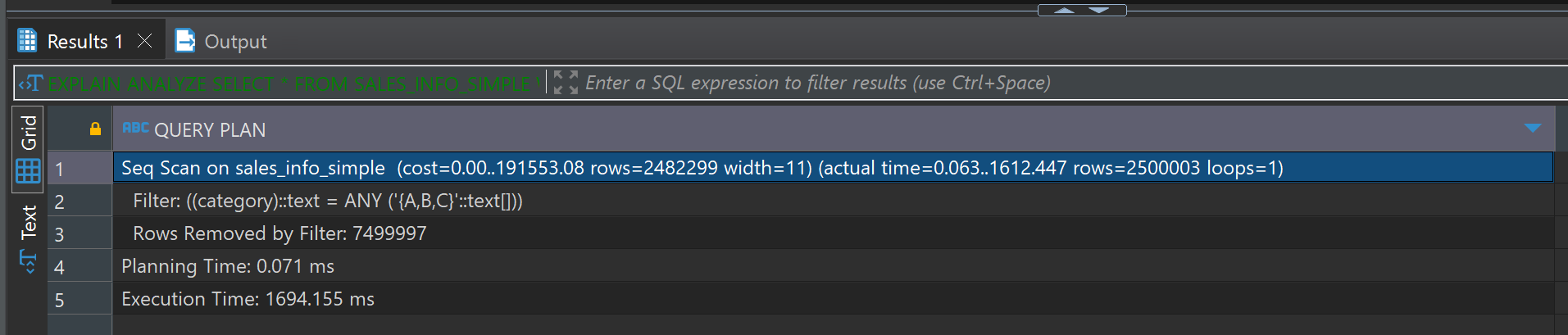




The execution plan comparison for selecting a specific category shows that SALES\_INFO\_DP performs sequential scans across relevant partitions, resulting in an execution time with low time In contrast, SALES\_INFO\_SIMPLE uses parallel sequential scans and takes significantly longer. This highlights the efficiency of partition pruning in reducing the amount of data scanned and improving query performance for category-based queries.

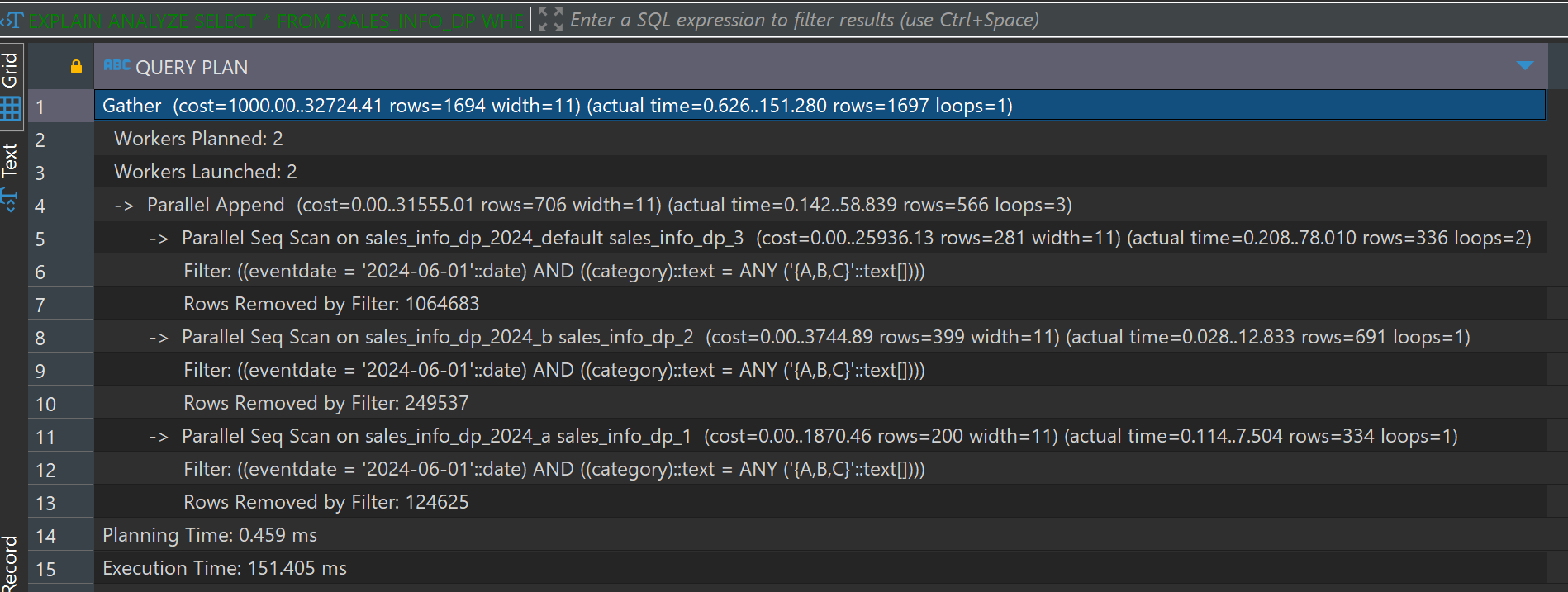
Select list of categories

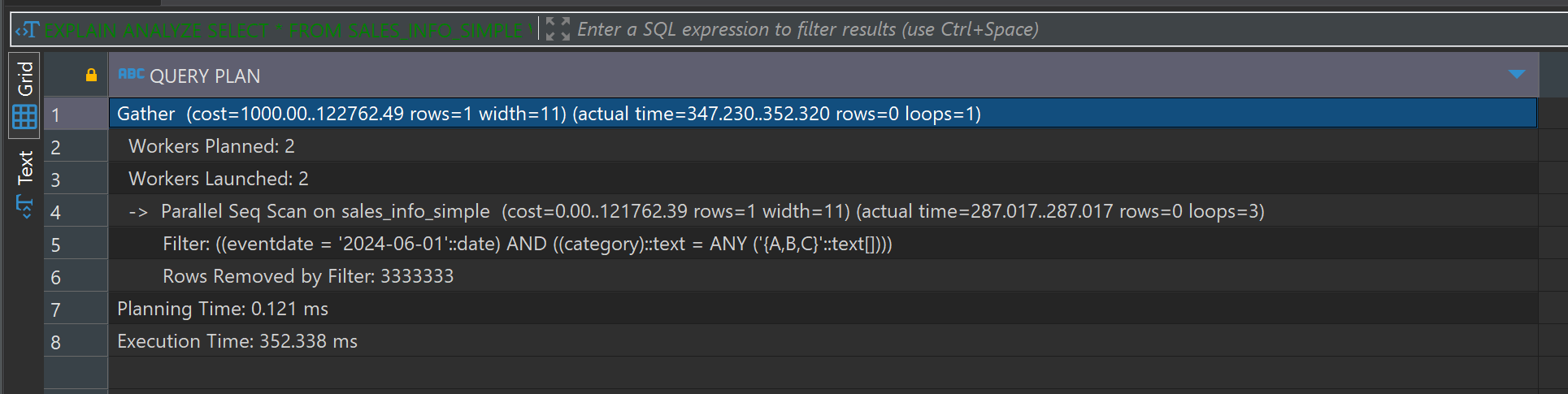




plan comparison for selecting a list of categories A, B, C shows that the partitioned table which is SALES\_INFO\_DP performs sequential scans across relevant partitions, resulting in an execution time of approximately 1697 ms. Which is a lot, In contrast, the non-partitioned table performs a single sequential scan, with a very similar execution time of approximately 1694 ms. This indicates that for queries involving multiple categories, the benefits of partition pruning may be less pronounced, and the performance can be similar between partitioned and non-partitioned tables.

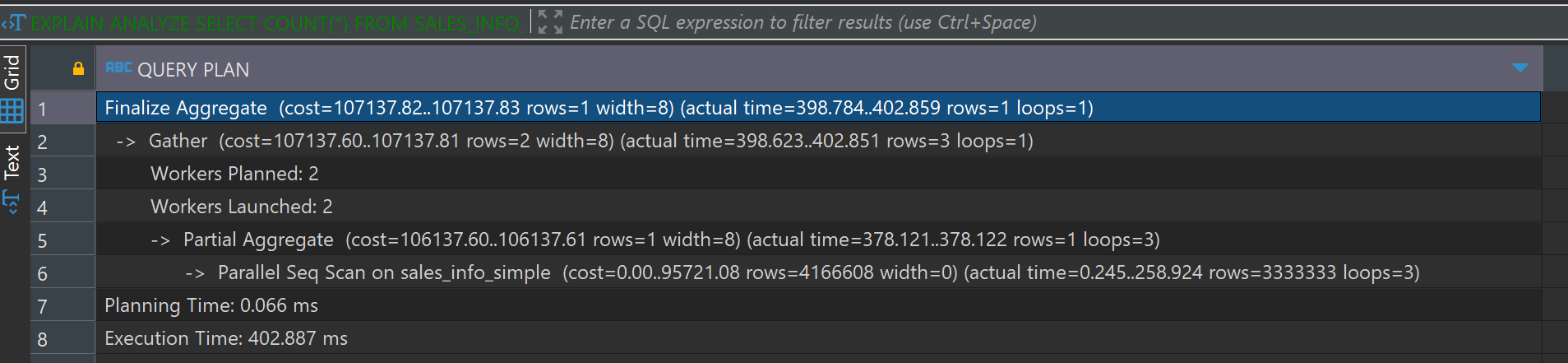
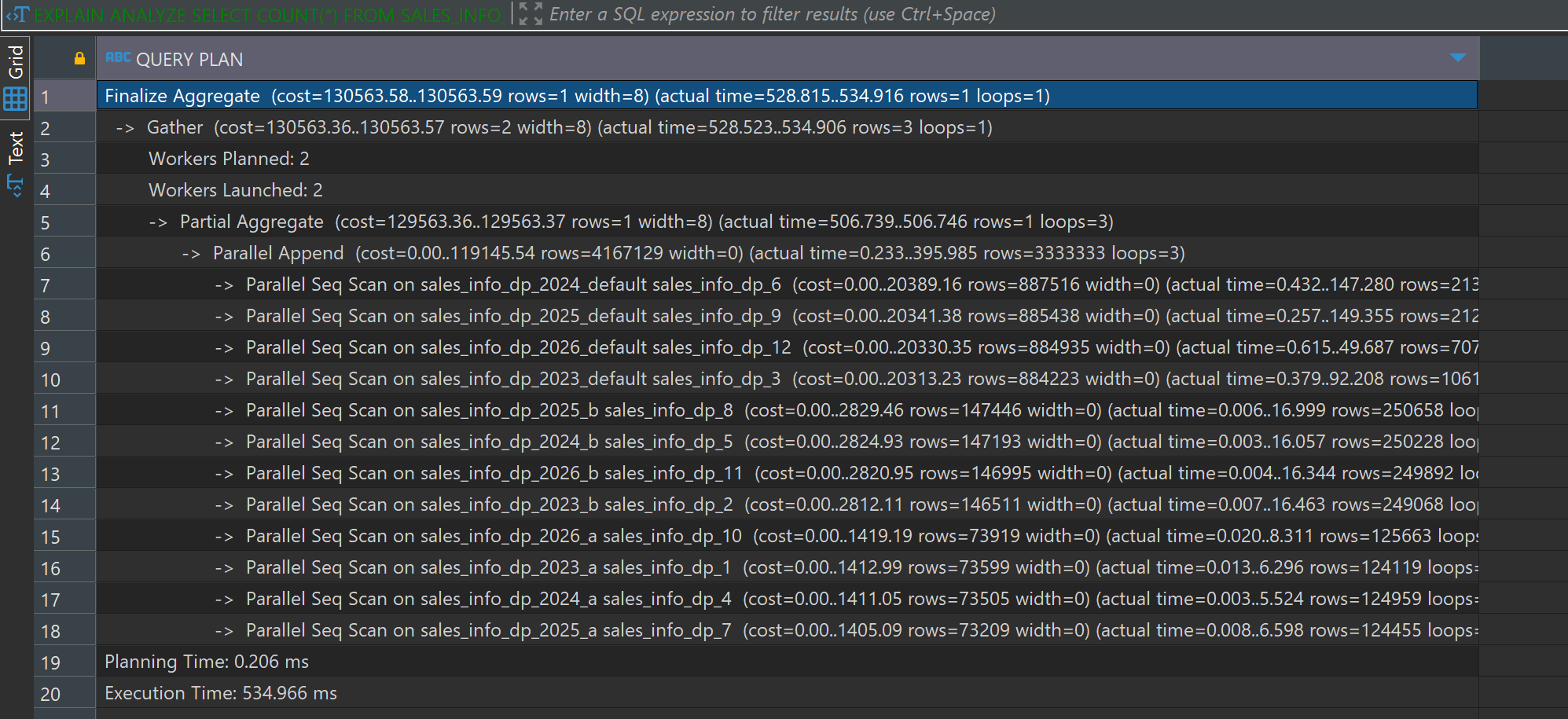
**Select a List of Categories on an Exact Date**

****

****

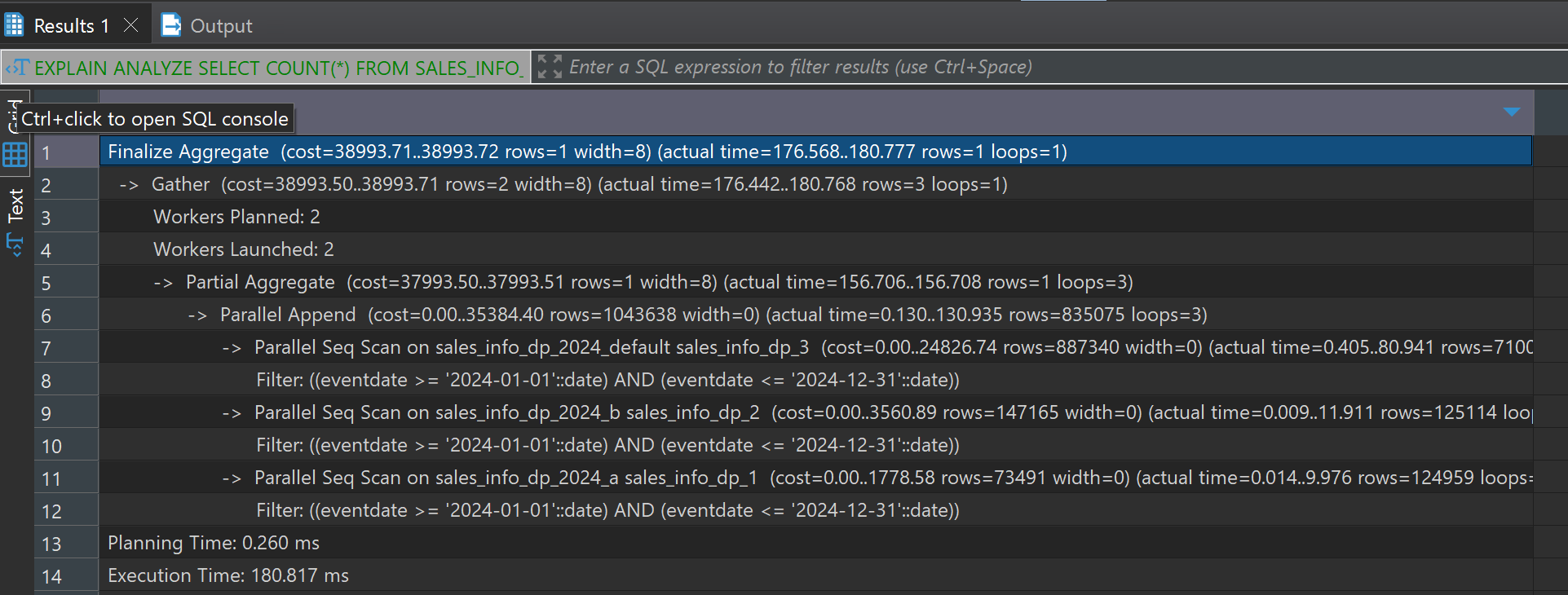
The execution plan comparison for selecting a list of categories on an exact date shows that the SALES\_INFO\_DP performs parallel sequential scans across relevant partitions, resulting in an execution time of approximately 151 ms. In contrast, the sales \_INFO\_SIMPLE also uses parallel sequential scans but takes significantly longer, with an execution time of approximately 352 ms. Shows effectiveness of partition pruning and parallel execution in the partitioned table, significantly reducing query execution time for more complex queries involving both date and category filters.

**Count of All Rows**

****

It shows that the partitioned uses parallel sequential scans across all partitions . In contrast, the non-partitioned table also performs a parallel sequential scan but takes less time. Even though partitioning, the overhead of scanning multiple partitions can lead to longer execution times compared to a single sequential scan for aggregate queries such as counting all row

**Count of Rows with Range of Dates**



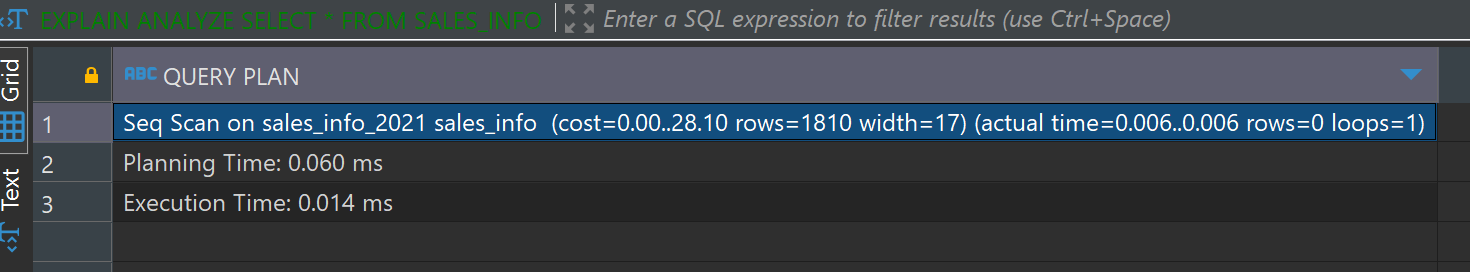


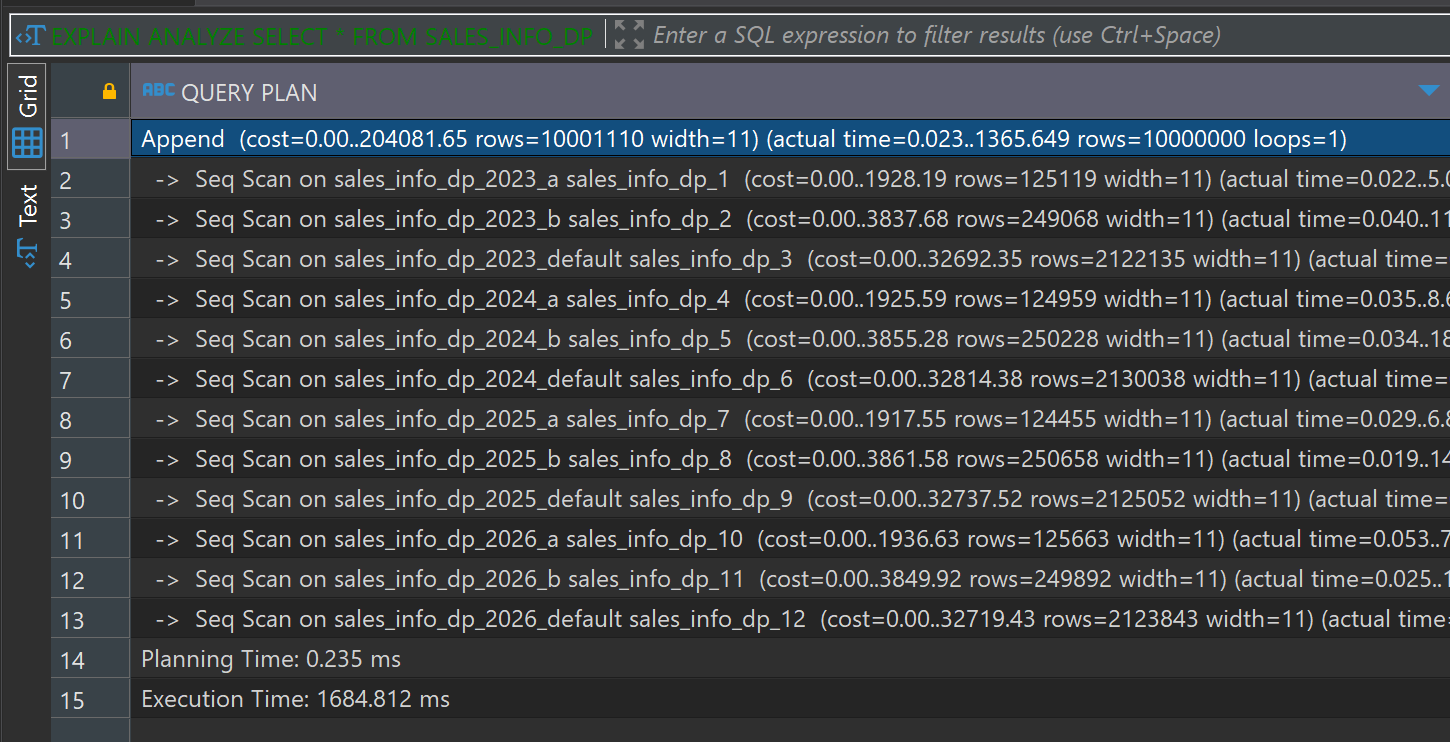
The execution plan comparison for counting rows within a date range shows that the partitioned table) uses parallel sequential scans across relevant partitions. In contrast, the non-partitioned table (also uses parallel sequential scans but takes significantly longer.

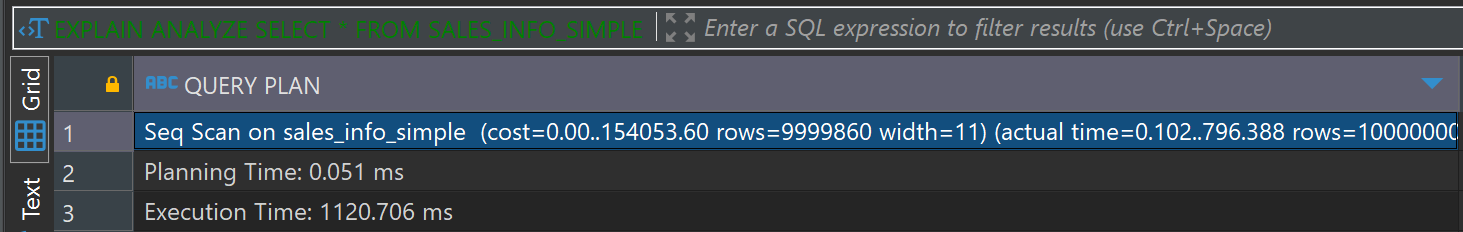
For one of the child tables with range partition by eventdate split one list partition for two. For example:

2.1 TASK 3: USE PARALLEL QUERING

a. Select all from tables

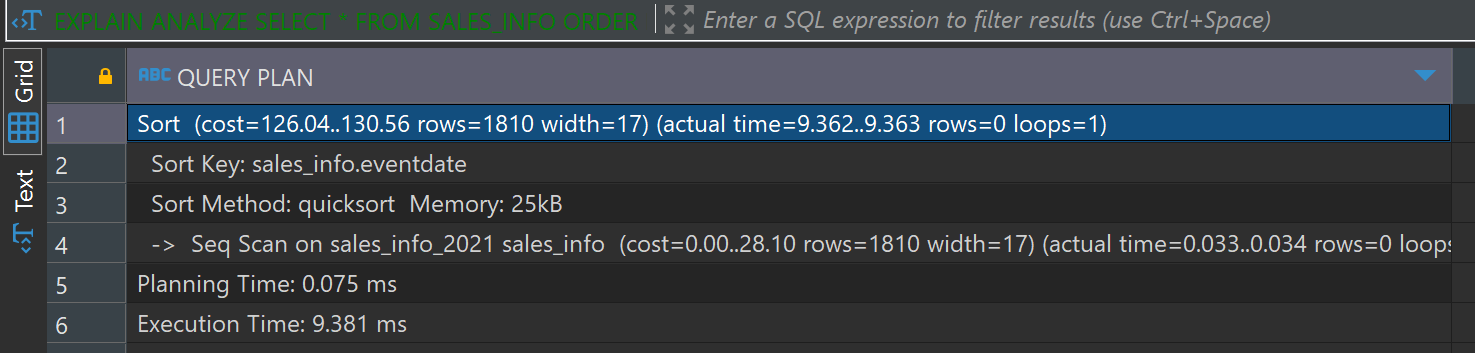


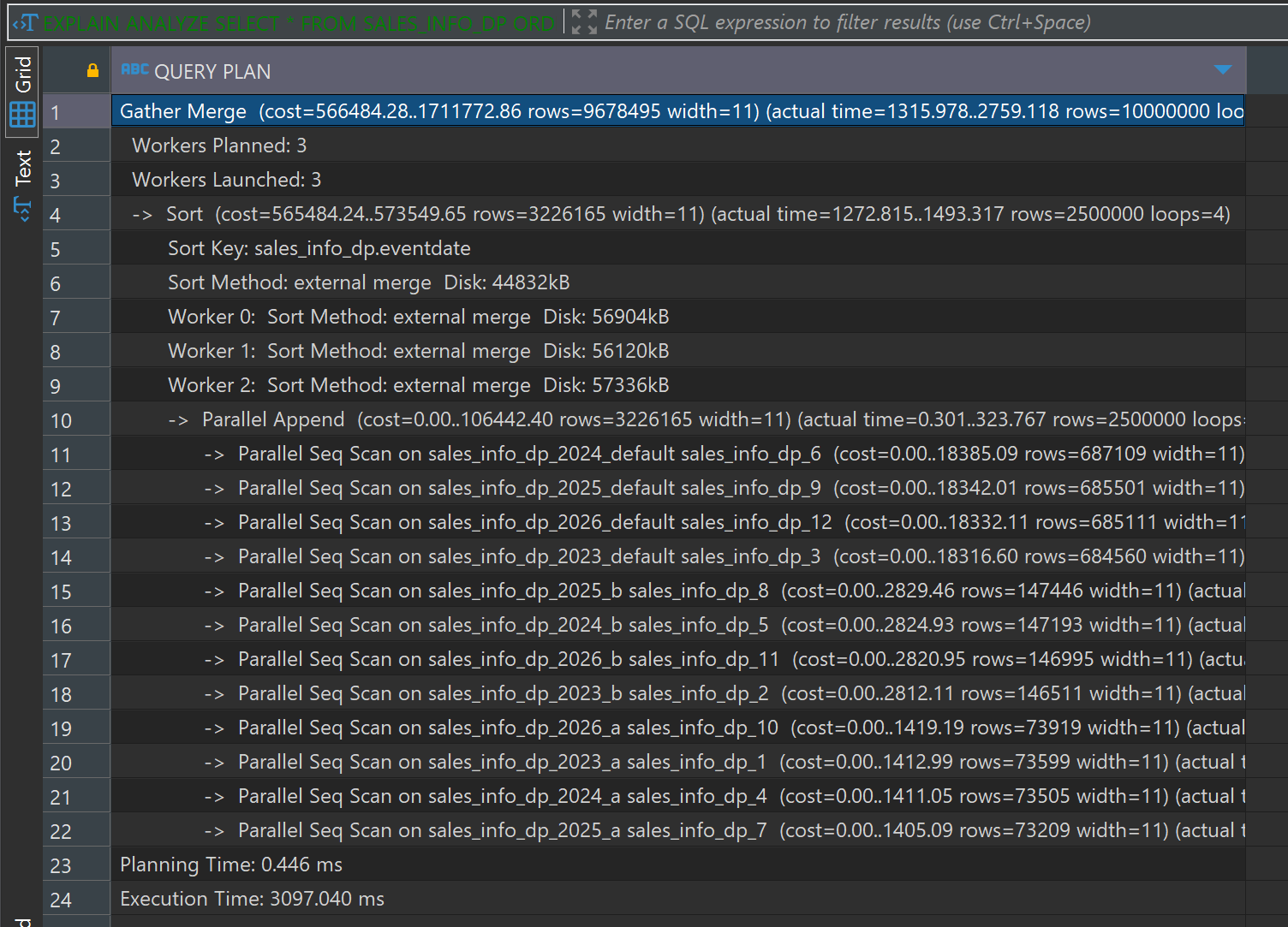


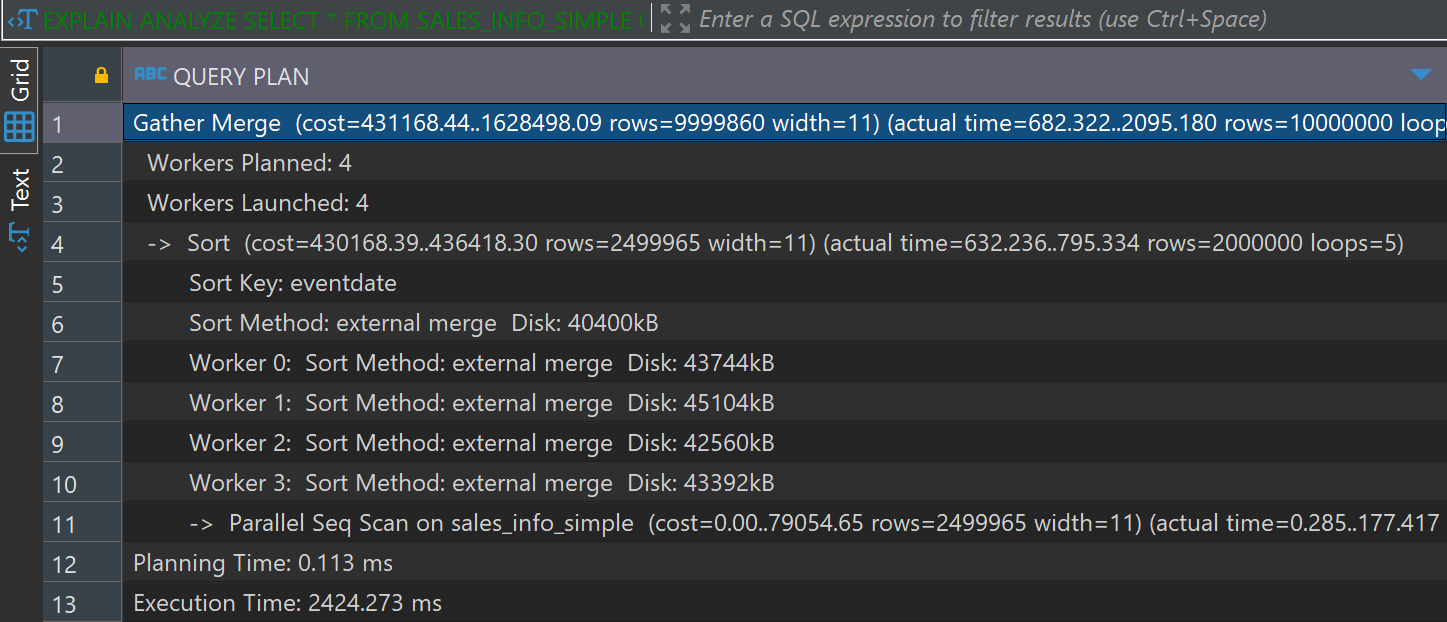


Both SALES\_INFO and SALES\_INFO\_SIMPLE perform simple sequential scans. SALES\_INFO has a very quick scan due to its small number of rows.SALES\_INFO\_SIMPLE has a larger number of rows (nearly 10 million) and takes more time for a full scan.The partitioned table (SALES\_INFO\_DP) involves an Append operation combining results from all partitions, leading to multiple sequential scans.Despite the increased complexity due to partitioning, the execution time is notably longer than the non-partitioned SALES\_INFO\_SIMPLE. SALES\_INFO is Quickest due to fewer rows.SALES\_INFO\_SIMPLE is Faster than SALES\_INFO\_DP but takes more time due to a large number of rows.

b. Add order by eventdate

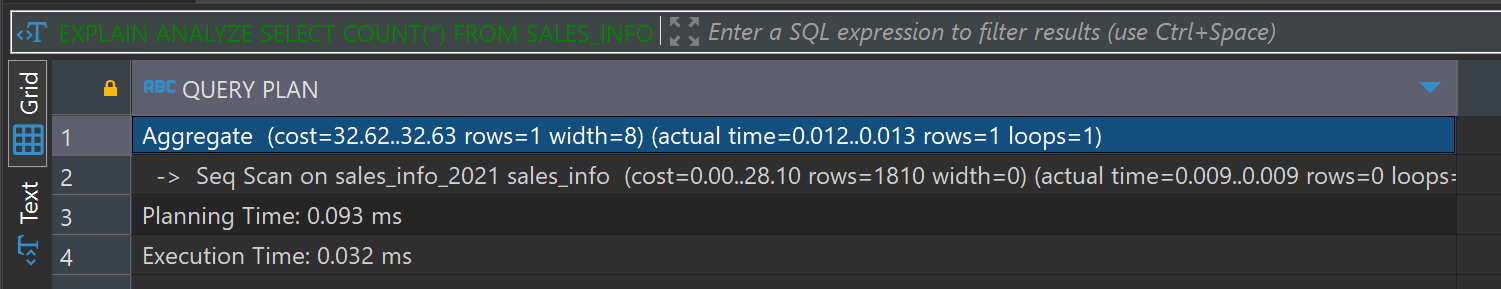


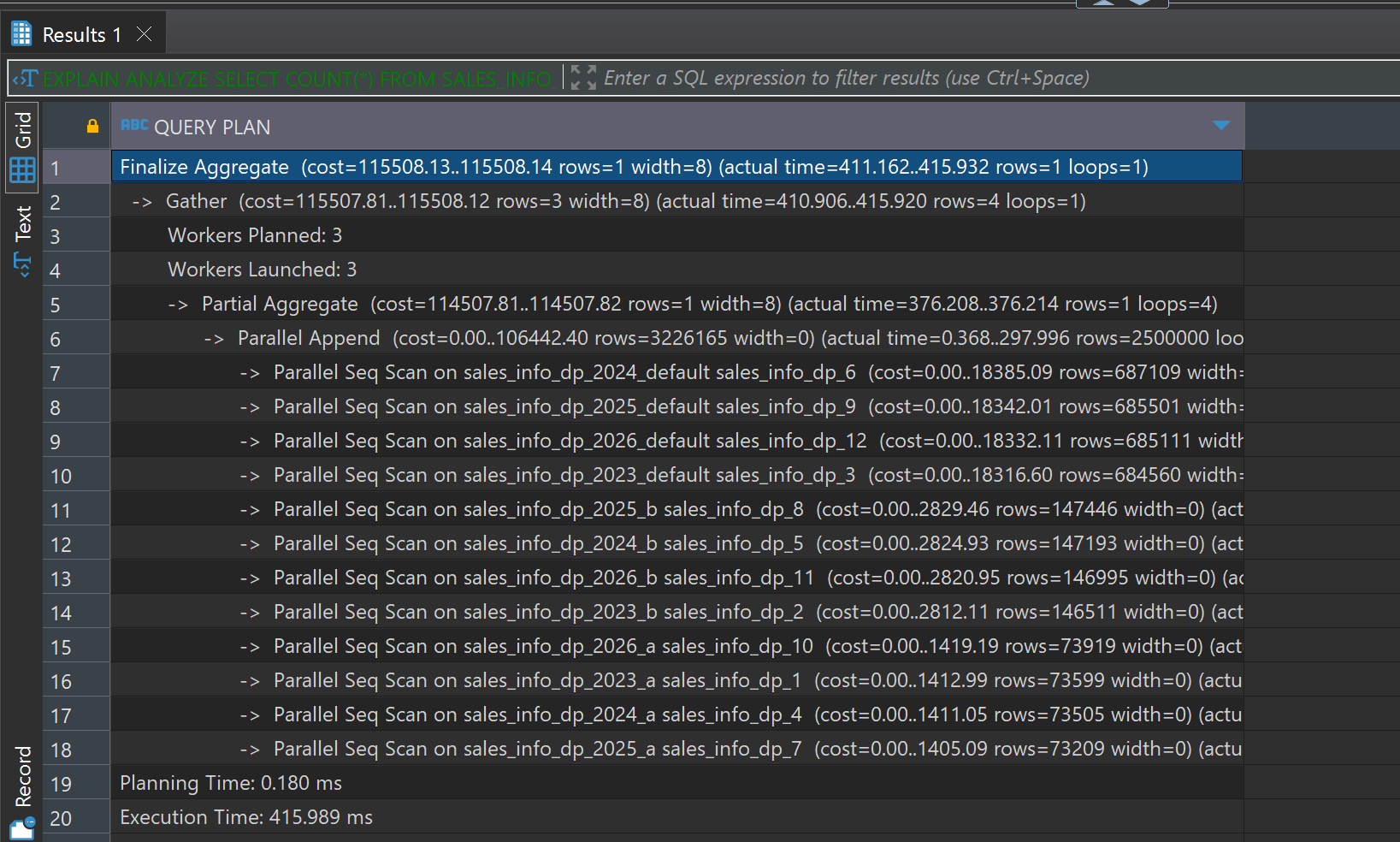


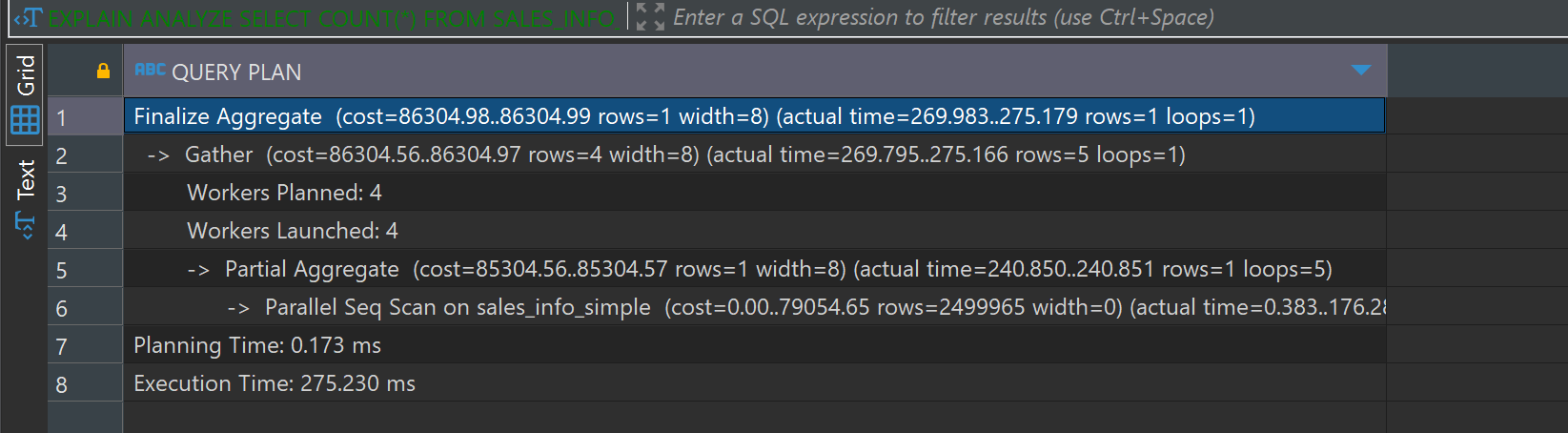


The SALES\_INFO table performs a simple sequential scan followed by sorting, resulting in relatively low execution time due to fewer rows.The SALES\_INFO\_DP table involves multiple parallel sequential scans across all partitions followed by a Gather Merge, resulting in a higher execution time. complexity of managing partitions and the overhead of combining and sorting the results from multiple partitions contribute to the increased execution time.The SALES\_INFO\_SIMPLE table utilizes parallel sequential scans and a Gather Merge for sorting, which provides better performance compared to the partitioned table but still has a higher execution time compared to the small SALES\_INFO table.This table avoids the overhead of managing multiple partitions but handles a large number of rows, leading to a significant execution time.

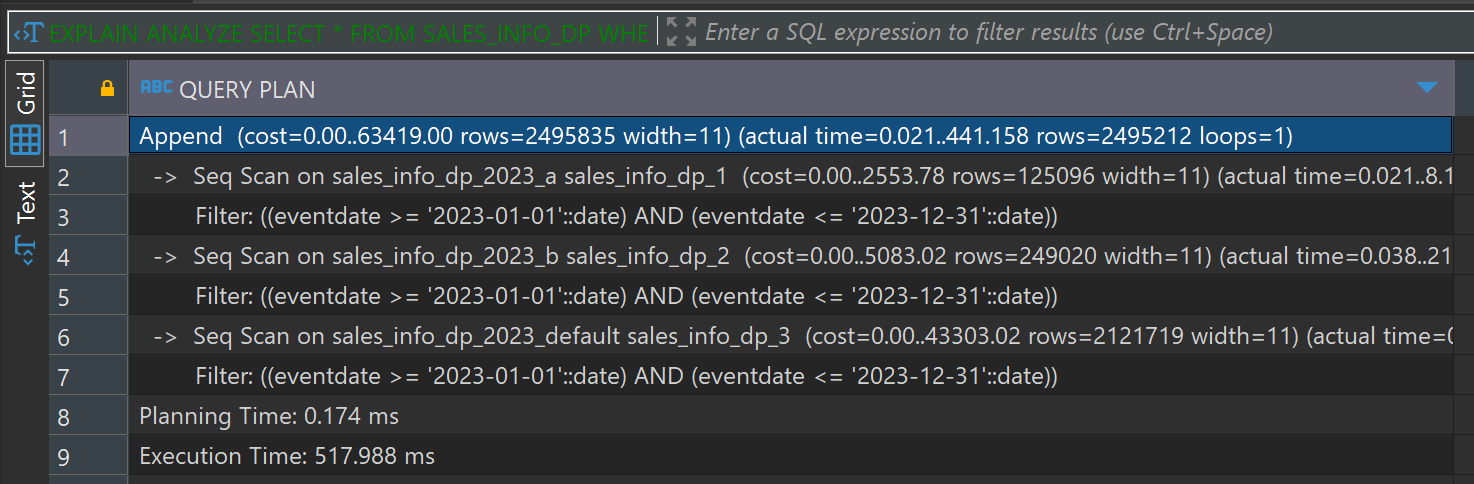
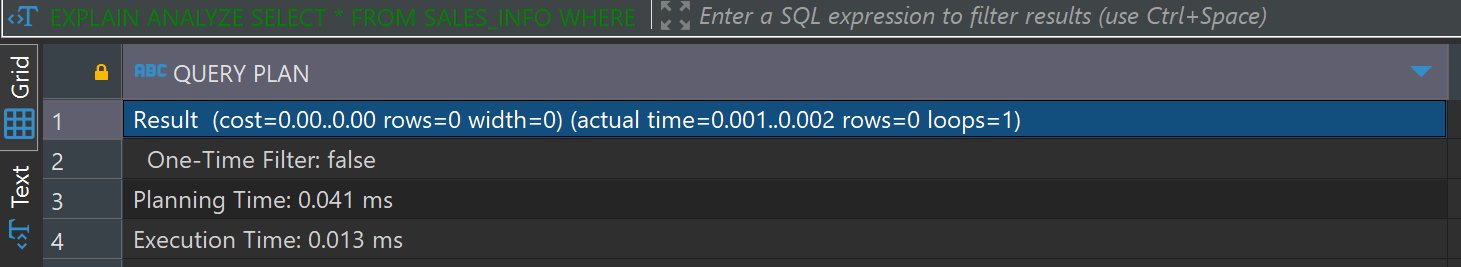
Select count of all rows

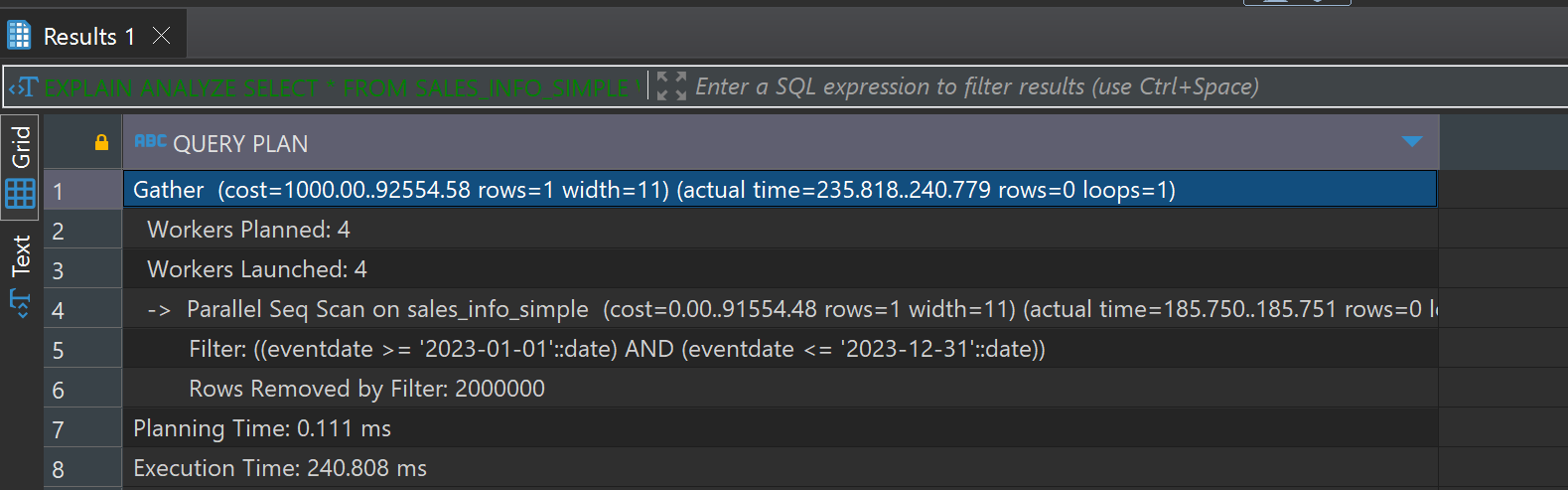




 The SALES\_INFO table performs a simple sequential scan followed by aggregation, resulting in a very low execution time due to fewer rows.The SALES\_INFO\_DP table involves parallel sequential scans across all partitions followed by aggregation, resulting in a moderate execution time. complexity of managing partitions and the overhead of combining the results from multiple partitions contribute to the increased execution time.The SALES\_INFO\_SIMPLE table utilizes parallel sequential scans followed by aggregation, providing better performance compared to the partitioned table.This table avoids the overhead of managing multiple partitions but handles a large number of rows, leading to a significant execution time.

-- d. Add range of dates



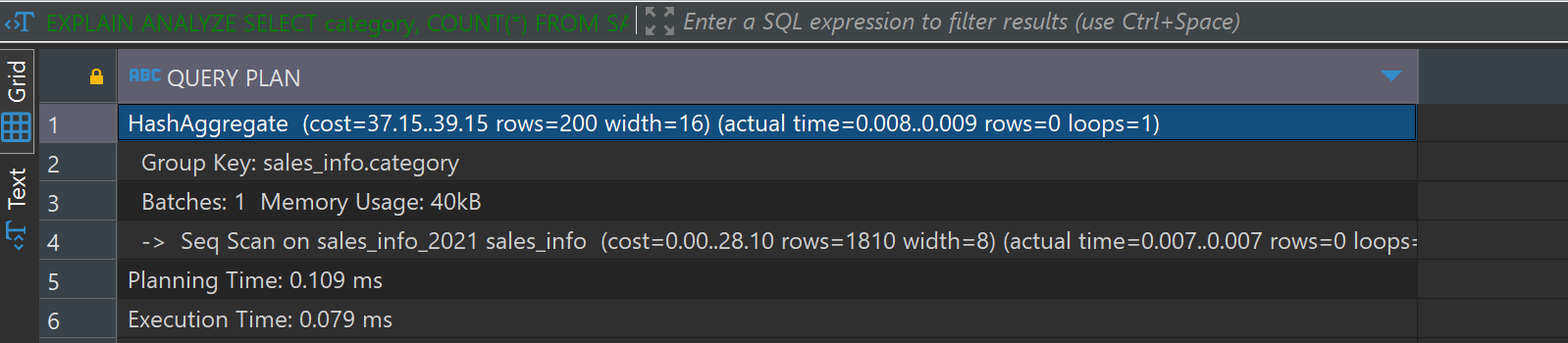


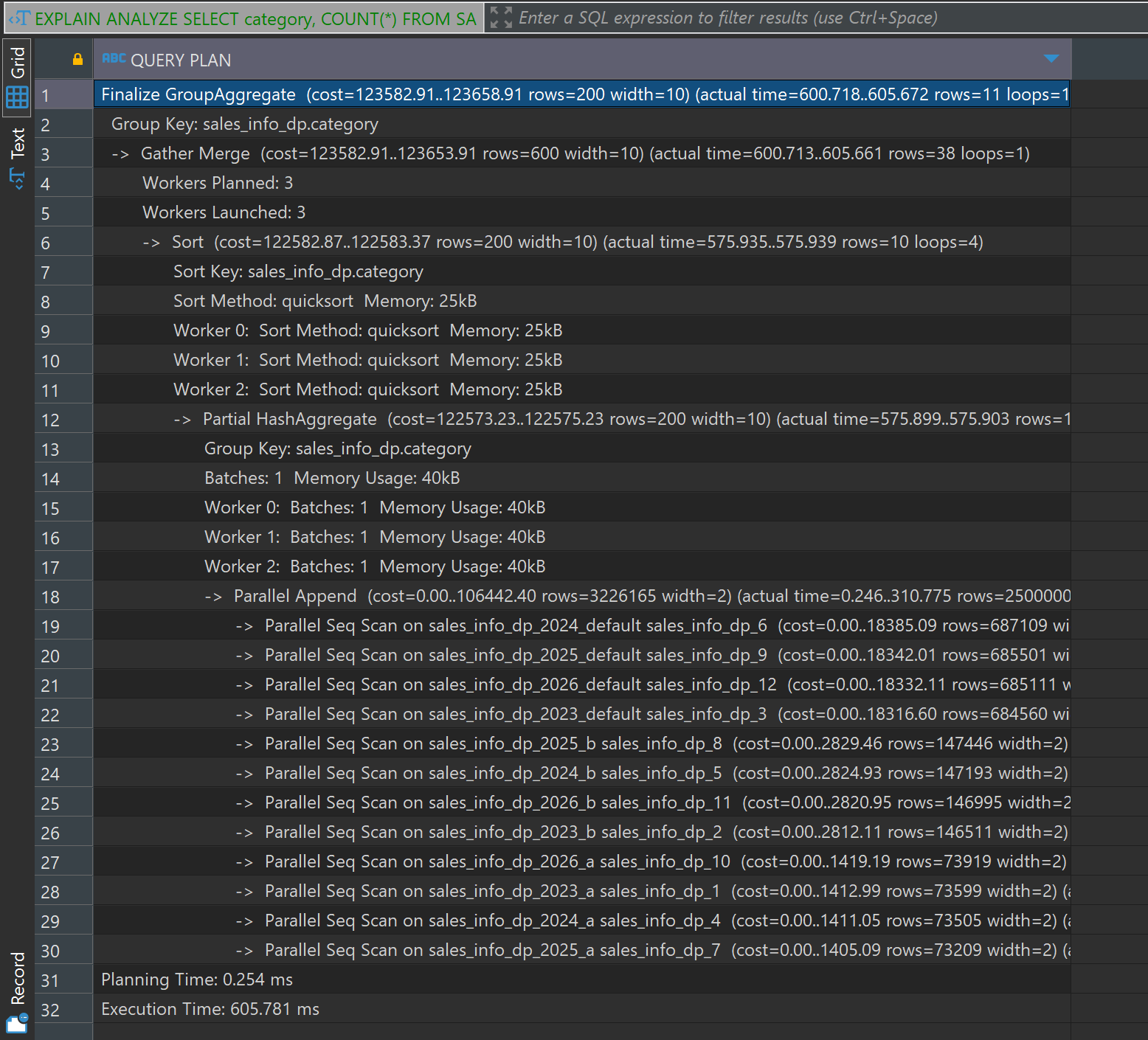
Partitioned Tableis effective for range queries due to partition pruning. The Append operation and multiple sequential scans contribute to a higher execution time.

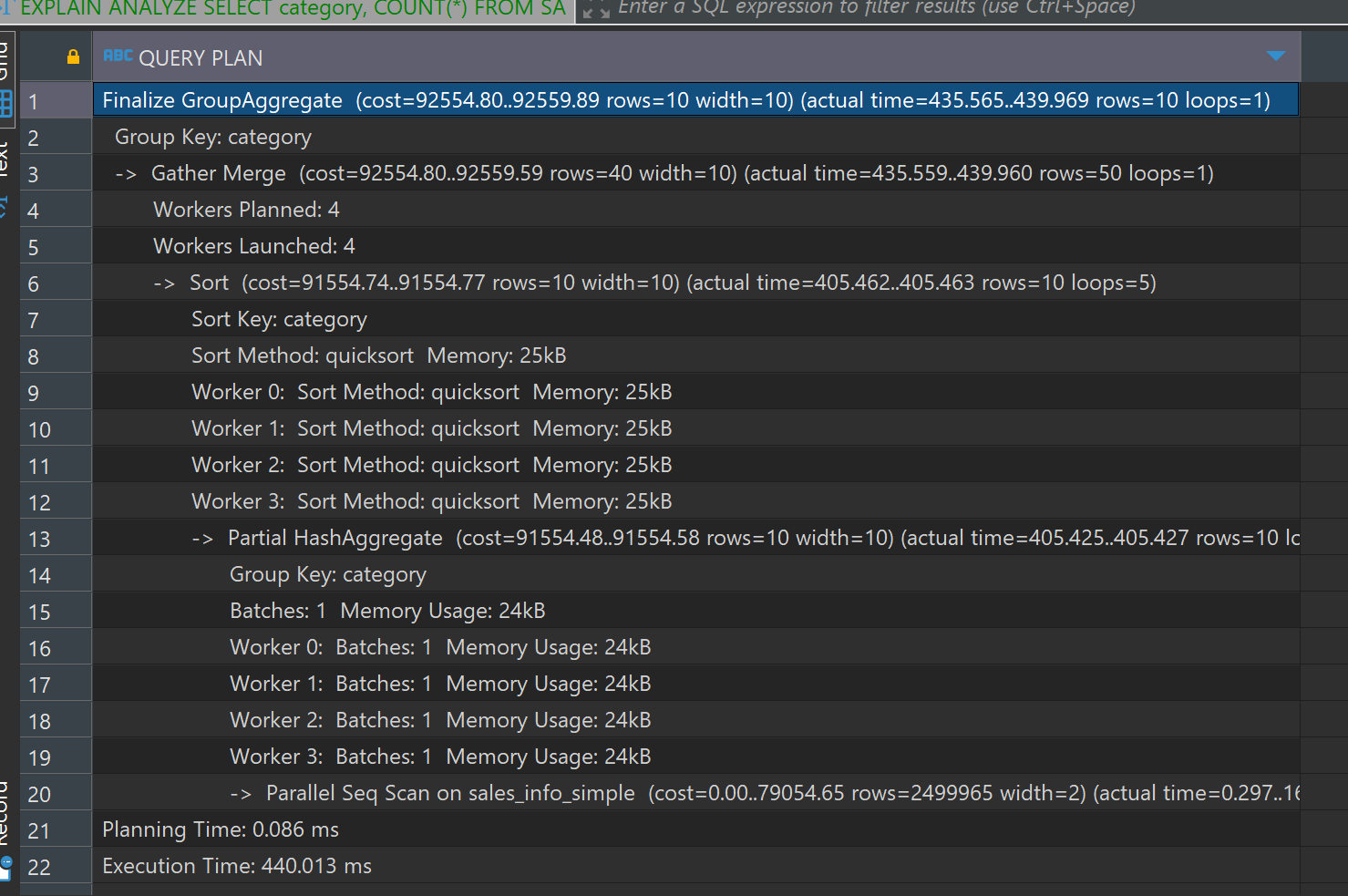
Non-Partitioned Table efficient parallel execution for range queries with lower execution time compared to the partitioned table. avoids partition management overhead, making it faster for such operations.

SALES\_INFO has quick execution with negligible time due to fewer rows and the absence of matching eventdate.

-- e. Add grouping by category







The SALES\_INFO table performs a simple sequential scan followed by a hash aggregate on category, resulting in a very low execution time due to fewer rows.

The SALES\_INFO\_DP table involves parallel sequential scans across all partitions followed by sorting and aggregation, resulting in a higher execution time.

The SALES\_INFO\_SIMPLE table utilizes parallel sequential scans followed by sorting and aggregation, providing better performance compared to the partitioned table.

This table avoids the overhead of managing multiple partitions but handles a large number of rows, leading to a moderate execution time.

- f. Join SALES\_INFO and SALES\_INFO\_DP on id and count rows on exact date

**EXPLAIN** **ANALYZE** **SELECT** **COUNT**(\*) **FROM** SALES\_INFO si

**JOIN** SALES\_INFO\_DP sidp **ON** si.id = sidp.id

**WHERE** si.eventdate = **'2021-01-01'**;



The query plan shows parallel processing with 3 workers planned and launched.

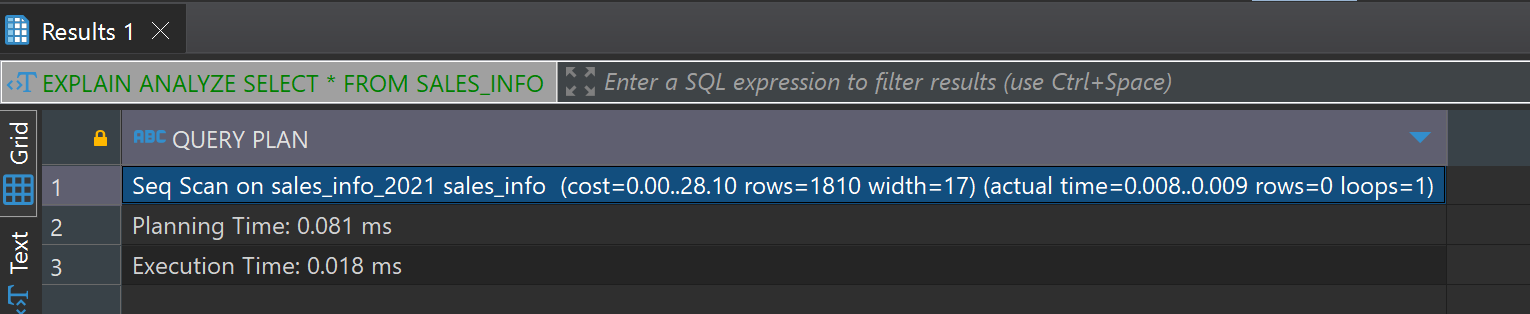
Parallel hash join and parallel sequential scans on partitions of SALES\_INFO\_DP.

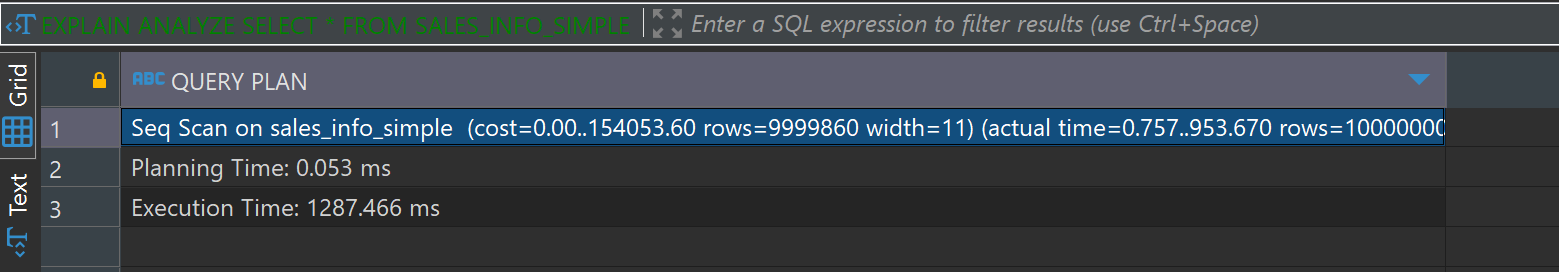
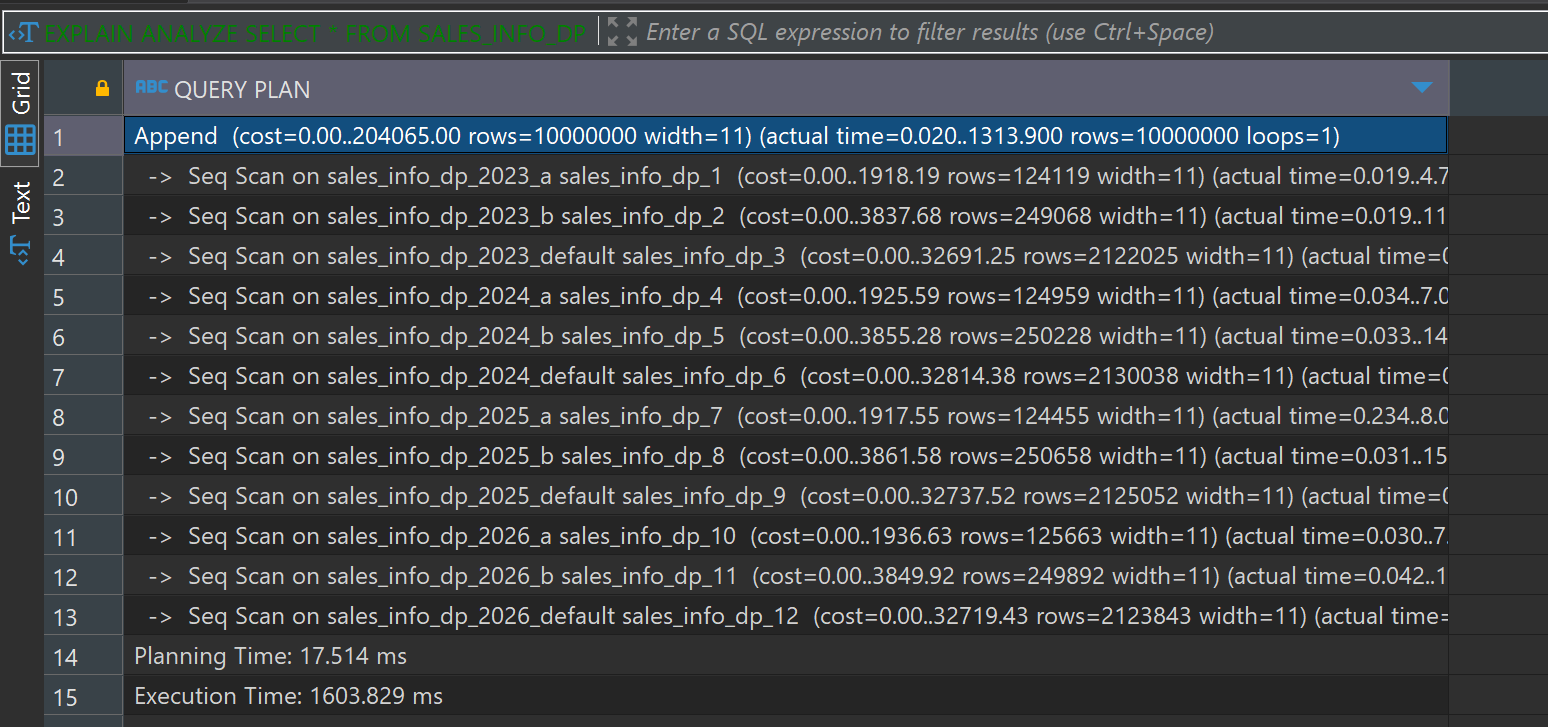
The join operation is performed using a hash join, which is efficient for large datasets.

The filter on eventdate is applied to the SALES\_INFO table, reducing the number of rows to be joined

Now creating index on id and event date on table Sales\_INFO\_DP

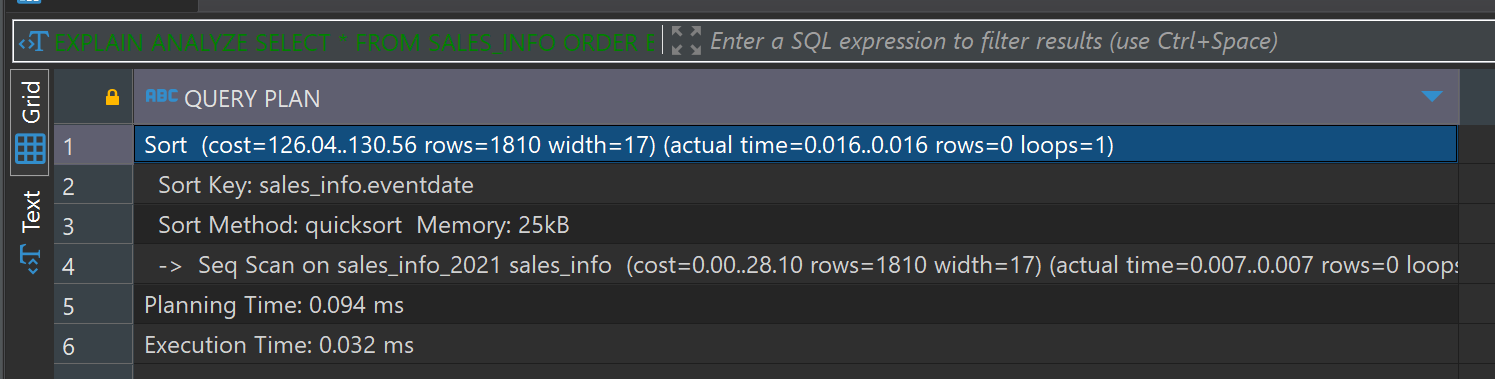
Selecting all

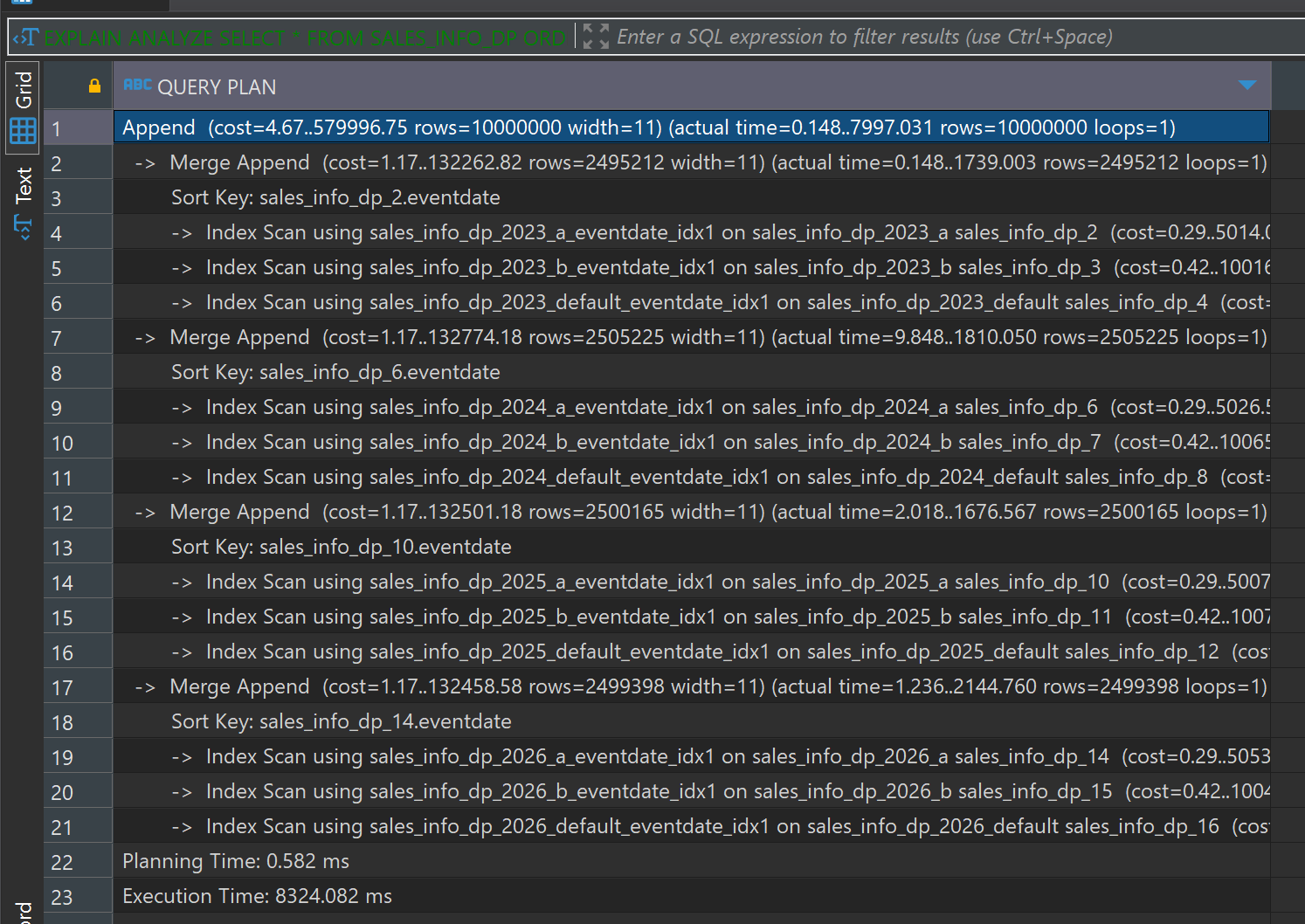


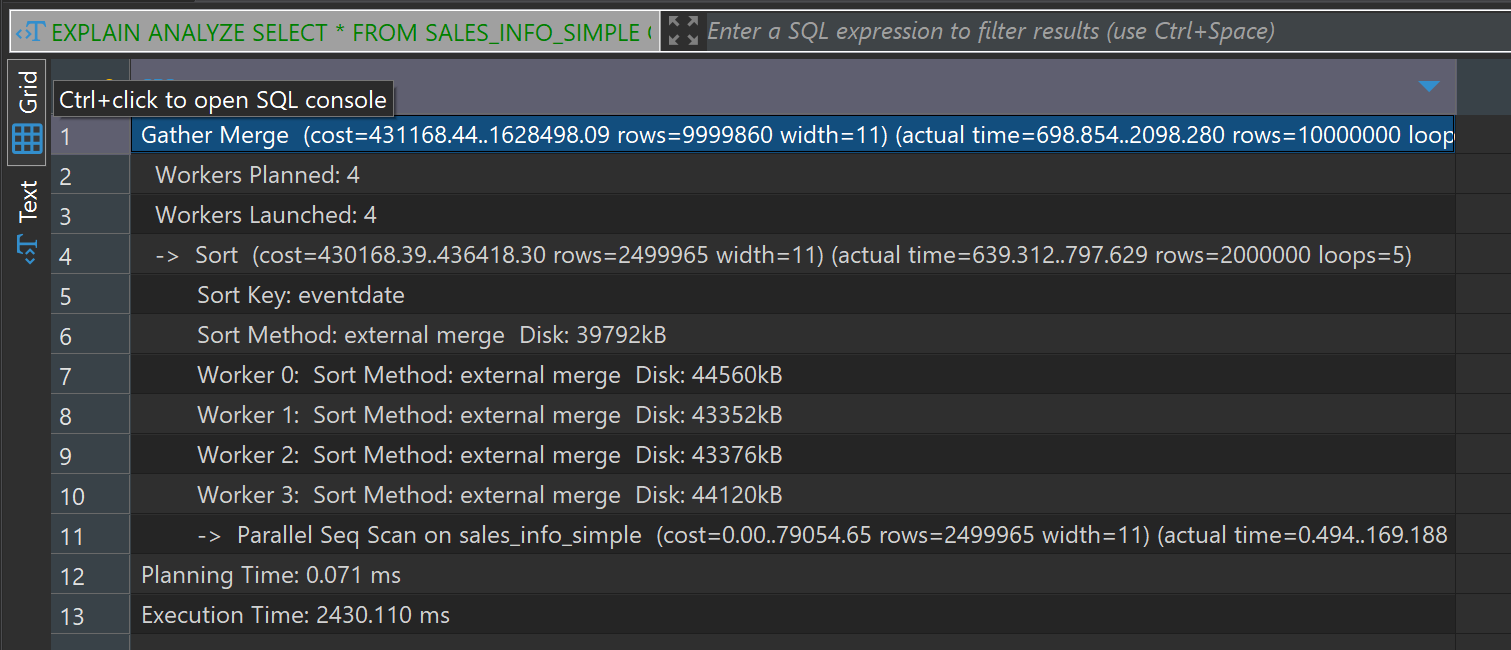


The SELECT \* queries do not show significant improvements with the added indexes, as expected. Full table scans are still required for these queries. The modest improvement in SALES\_INFO\_DP suggests some behind-the-scenes optimization, but overall, the impact is minimal.

Order by eventdate

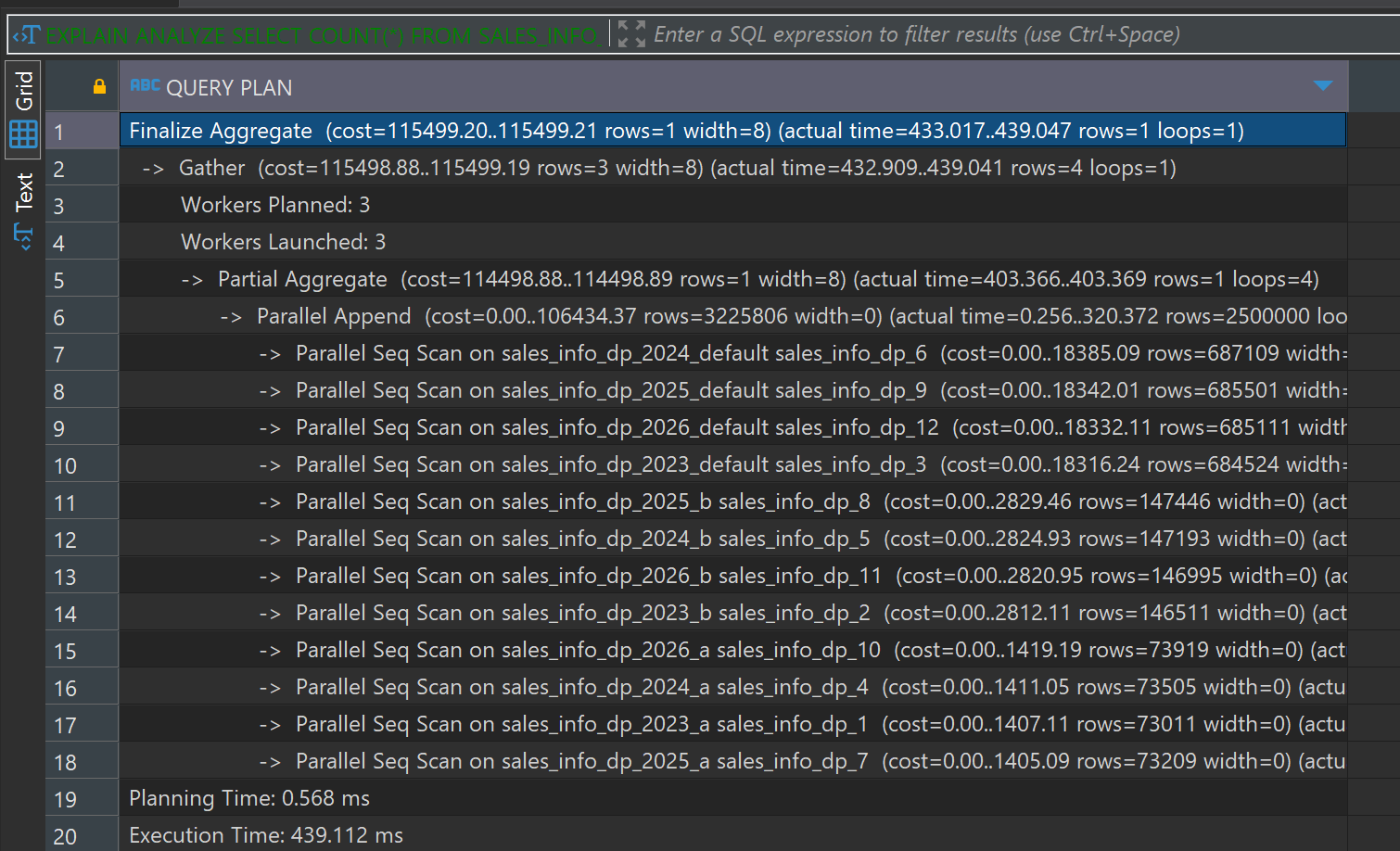
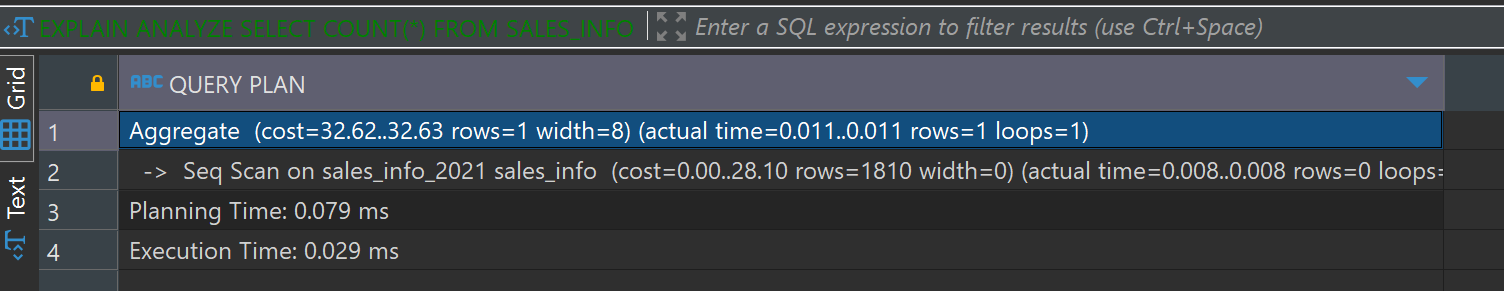


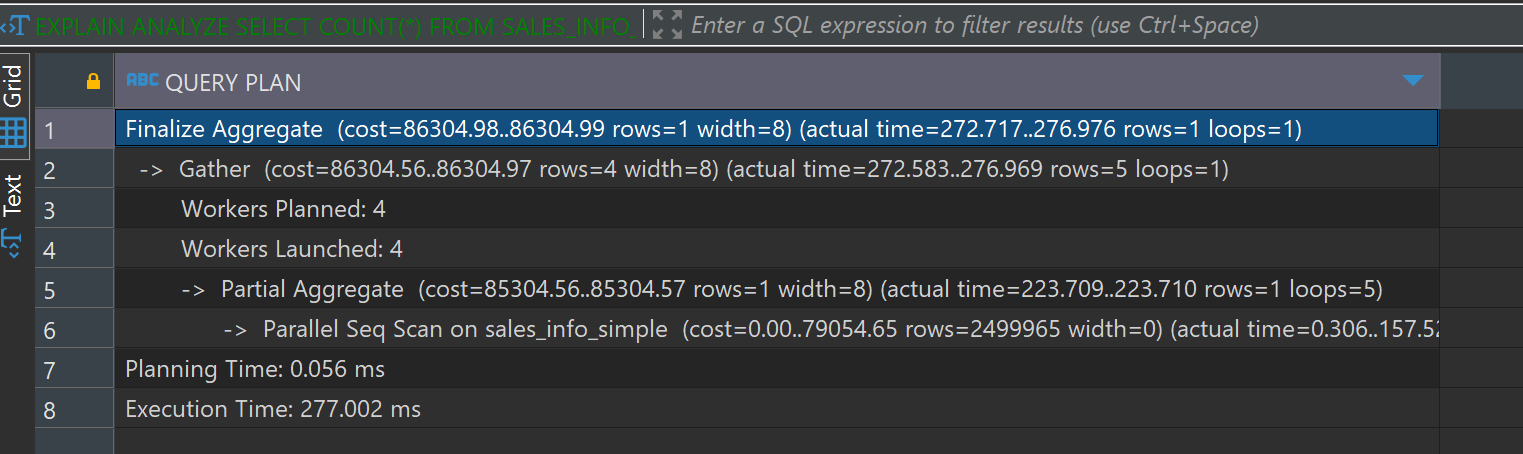




For SALES\_INFO\_DP, the execution time increased significantly due to the complexity of managing index scans and merge append operations across partitions

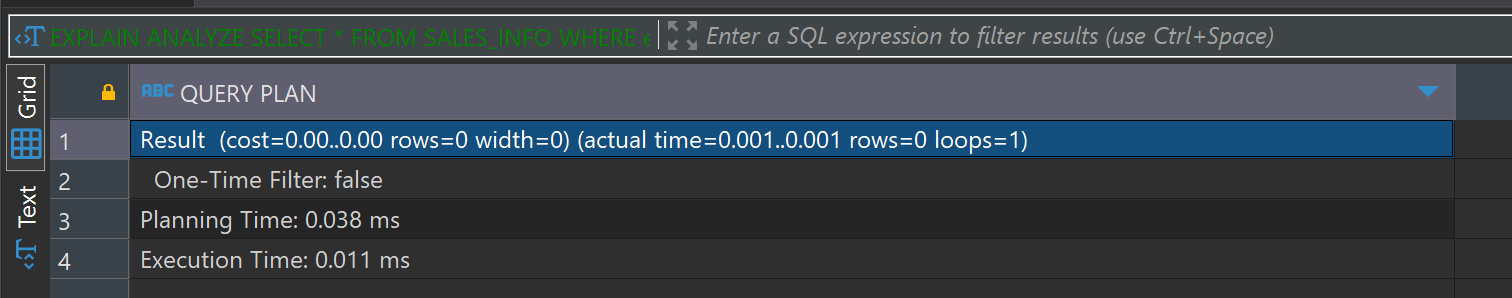
Count all rows

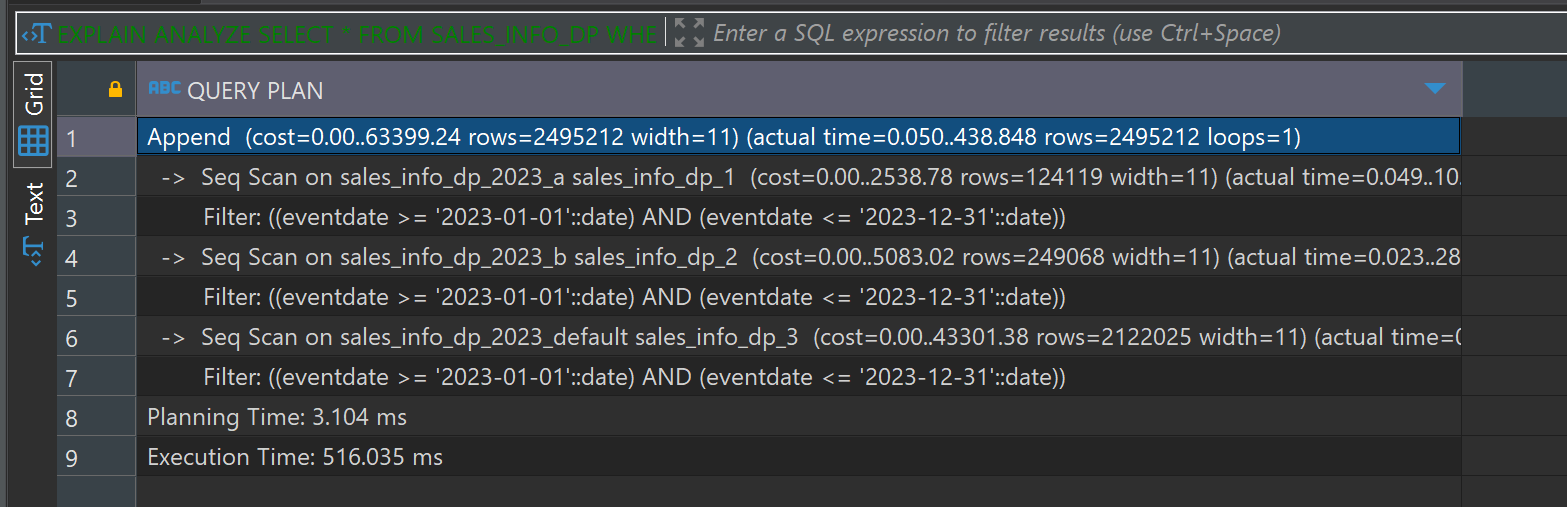


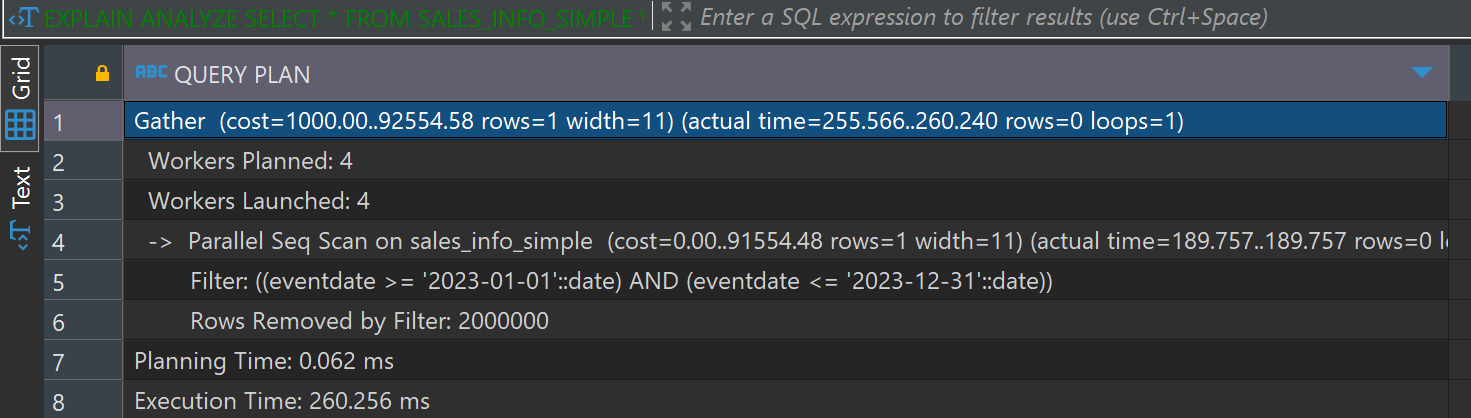


The COUNT(\*) queries do not show significant improvements with the added indexes. Full table scans are still required for these queries.The modest changes in execution time indicate that the indexes may not provide significant benefits for simple aggregation queries like COUNT(\*).

**. Add Range of Dates**

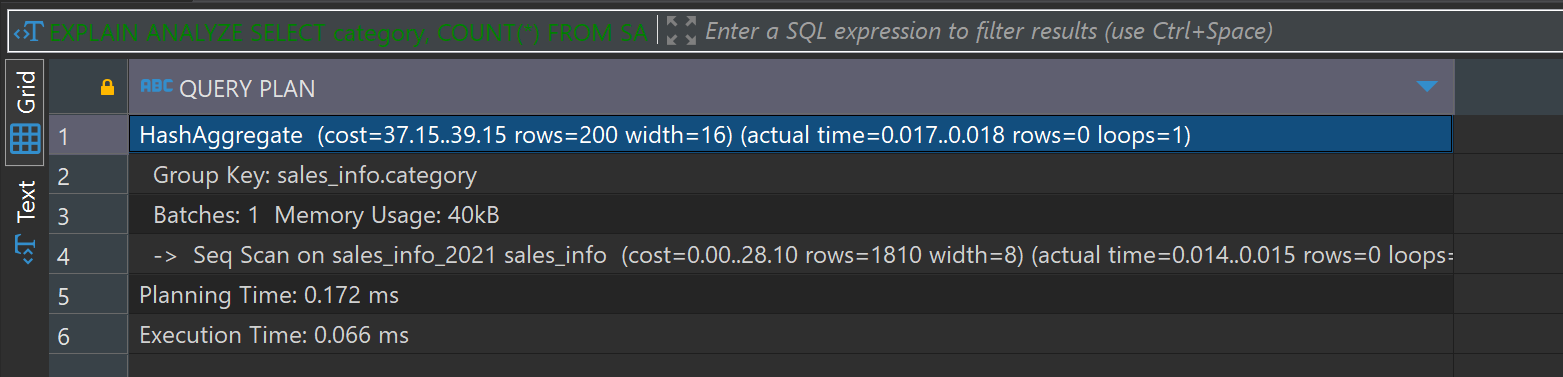


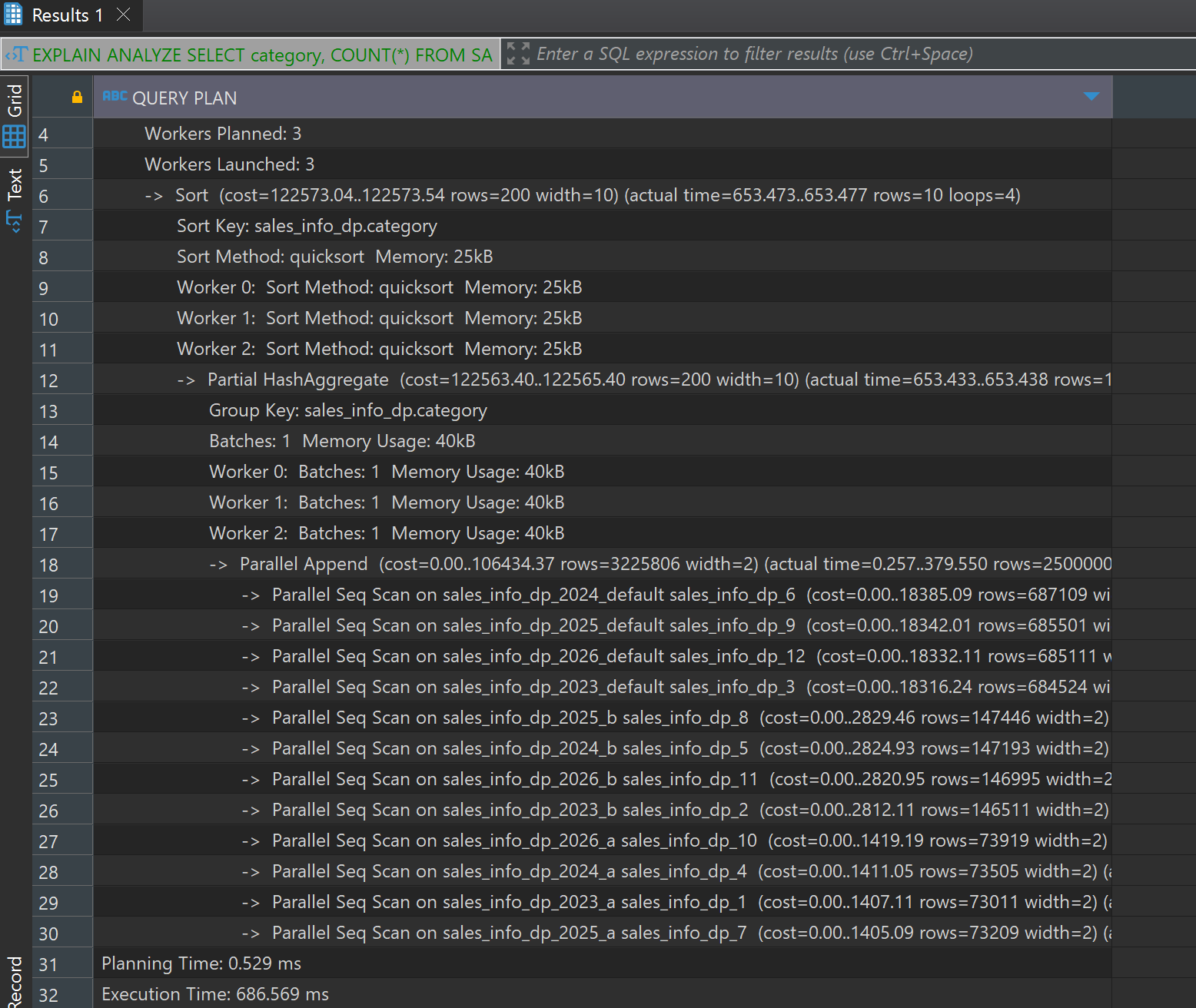


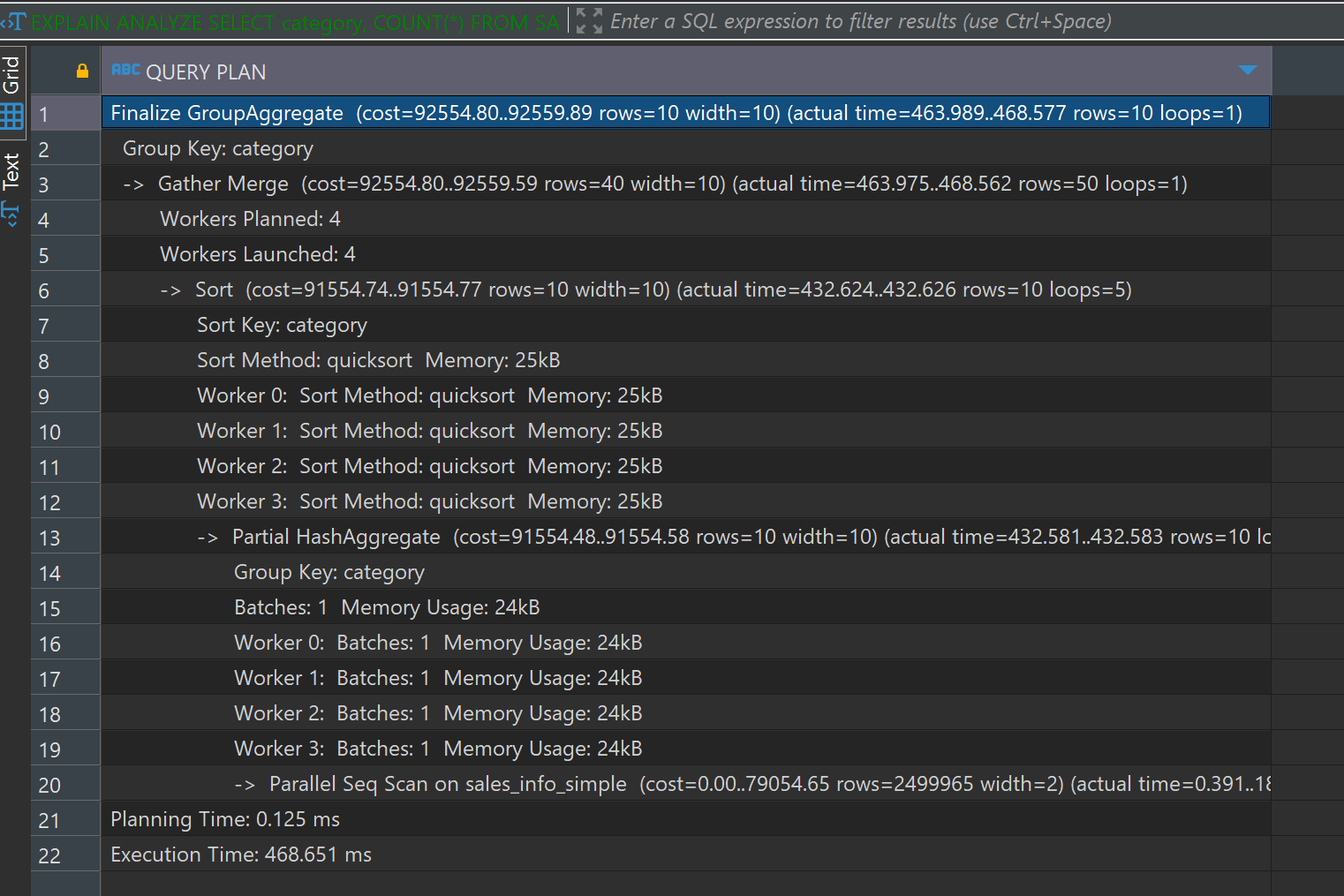


The SALES\_INFO\_DP table shows a slight improvement in execution time, indicating that the index might help with data organization but does not significantly enhance performance for range queries.

Add grouping by category







The execution time for SALES\_INFO\_DP increased slightlyafter adding the index. This indicates that the index might not help significantly with the group-by operation in this specific query, or the overhead of managing the indexes during aggregation affects performance.