**A Report On Execution Plan Of Cube and Rollup Operator**

**Submitted By:**

Abdus Salam Islam Badhon (BSSE-1401)

Habibur Rahman Mahin (BSSE-1422)

Shaikhul Islam (BSSE-1438)

Salsabila Zaman (BSSE-1443)

Rifat Ahmed Tusher (BSSE-1451)

**Submitted To:**

Dr. Mohammad Shoyaib

Professor IIT,University of Dhaka

**Submission Date:**

29 December, 2024

**1. Introduction to Data Cube**

A Data Cube is a multidimensional representation of data, often used in online analytical processing (OLAP).

It allows users to view data from various perspectives, providing insights into patterns, trends, and relationships.

Typically, a data cube organizes data across multiple dimensions (e.g., time, location, product) and allows for various aggregations

(e.g., sum, average, count) across these dimensions.

In relational databases, the CUBE and ROLLUP operators are used to simplify complex queries involving multiple dimensions.

These operators help in aggregating data at different levels, making it easier to analyze and summarize data without needing to

write multiple SQL queries manually.

**2. Difference Between Raw SQL Query and Using CUBE and ROLLUP**

Raw SQL Query:

If you were to write a query manually to achieve similar results to what CUBE or ROLLUP would do,

you would need to write multiple GROUP BY clauses and perform union operations to get the various levels of aggregation.

For example, if you want to aggregate data at the total level, region level, and store level,

you would need separate queries and combine them using UNION:

Raw SQL query without using CUBE or ROLLUP:

SELECT

Region, Product, Category, Store, Quarter,

SUM(Revenue) AS SUM\_Revenue,

COUNT(Revenue) AS COUNT\_Revenue,

MIN(Revenue) AS MIN\_Revenue,

MAX(Revenue) AS MAX\_Revenue,

AVG(Revenue) AS AVG\_Revenue

FROM SalesTable

GROUP BY Region, Product, Category, Store, Quarter

UNION ALL

SELECT

Region, Product, Category, Store, NULL AS Quarter,

SUM(Revenue) AS SUM\_Revenue,

COUNT(Revenue) AS COUNT\_Revenue,

MIN(Revenue) AS MIN\_Revenue,

MAX(Revenue) AS MAX\_Revenue,

AVG(Revenue) AS AVG\_Revenue

FROM SalesTable

GROUP BY Region, Product, Category, Store

UNION ALL

SELECT

Region, Product, Category, NULL AS Store, NULL AS Quarter,

SUM(Revenue) AS SUM\_Revenue,

COUNT(Revenue) AS COUNT\_Revenue,

MIN(Revenue) AS MIN\_Revenue,

MAX(Revenue) AS MAX\_Revenue,

AVG(Revenue) AS AVG\_Revenue

FROM SalesTable

GROUP BY Region, Product, Category;

This query needs multiple UNION operations for every possible aggregation level, making it verbose, harder to manage, and inefficient.

Using CUBE or ROLLUP:

In contrast, using CUBE or ROLLUP simplifies the query by handling multiple aggregation levels within a single query.

- CUBE: This operator calculates all combinations of the grouping dimensions, including the total aggregates (e.g., subtotals and grand totals).

- ROLLUP: This is a variation of GROUP BY that generates subtotal values by progressively reducing the dimensions from left to right.

CUBE Example:

SELECT

Region,

Product,

Category,

Store,

Quarter,

SUM(Revenue) AS SUM\_Revenue,

COUNT(Revenue) AS COUNT\_Revenue,

MIN(Revenue) AS MIN\_Revenue,

MAX(Revenue) AS MAX\_Revenue,

AVG(Revenue) AS AVG\_Revenue

FROM SalesTable

GROUP BY CUBE (Region, Product, Category, Store, Quarter)

ORDER BY Region, Product, Category, Store, Quarter;

This query automatically calculates the aggregates for all combinations of the grouping columns,

without needing to manually specify each possible grouping combination.

**3. Efficiency of Using CUBE and ROLLUP**

The efficiency and usefulness of CUBE and ROLLUP depend on the complexity of the query and the dataset involved.

- Efficiency:

- Query Simplicity: The CUBE and ROLLUP operators significantly reduce the complexity of the SQL query. Instead of having to manually write unions for every possible combination of groupings, these operators handle the aggregation automatically.

- Performance: While CUBE can be more resource-intensive due to generating all combinations of groupings, it is generally more efficient than manually writing unions for each combination. However, if your dataset is large and you're calculating many aggregations, performance could be a concern, as the database engine needs to perform additional calculations.

- Use of Indexes: Since CUBE and ROLLUP rely on groupings, performance can improve if the database has proper indexes on the grouping columns (like Region, Product, Category, Store, Quarter). If indexes are missing, the query might become slower.

- Usefulness:

- Single Query Solution: By using CUBE or ROLLUP, you can avoid writing and maintaining multiple queries for different levels of aggregation. It helps in producing a result set with detailed breakdowns and overall totals in one go.

- Flexibility: The CUBE operator is more flexible than ROLLUP because it produces aggregates for all possible combinations of the grouped columns, which might be useful for complex analytical needs.

- Data Exploration: For users exploring data interactively, using CUBE or ROLLUP can simplify the process of quickly drilling down or rolling up the data without the need to manually create different aggregation queries.

**4. Drawbacks of Using CUBE and ROLLUP**

While CUBE and ROLLUP can significantly reduce query complexity, there are some drawbacks:

- Performance Overhead: For very large datasets, the performance overhead can be significant because the database must generate and aggregate multiple combinations. This can result in longer query execution times, especially if the cube has many dimensions.

- Complexity in Interpretation: The results from a CUBE operation can be difficult to interpret since the results contain multiple levels of aggregation. Each row in the result set corresponds to a different combination of the grouping columns.

- Handling of NULLs: Both CUBE and ROLLUP generate rows where one or more of the grouping columns will be NULL, which might require extra handling in your application or report to make sense of the output.

**5. Execution Stats:**

**\*\*\* Table Schema:**

CREATE TABLE

SalesTable (

Region VARCHAR(20),

Product VARCHAR(20),

Category VARCHAR(20),

Store VARCHAR(20),

Quarter VARCHAR(5),

Revenue INT

);

INSERT INTO

SalesTable (

Region,

Product,

Category,

Store,

Quarter,

Revenue

)

VALUES

('North', 'Apple', 'Fruit', 'Store1', 'Q1', 100),

('North', 'Banana', 'Fruit', 'Store1', 'Q1', 200),

('South', 'Apple', 'Fruit', 'Store2', 'Q1', 150),

('South', 'Banana', 'Fruit', 'Store2', 'Q1', 50),

('East', 'Apple', 'Fruit', 'Store3', 'Q1', 120),

**\*\*\* Without Cube operator:**

"Sort (cost=220.70..221.66 rows=384 width=216) (actual time=7.002..7.106 rows=1393 loops=1)"

" Sort Key: (NULL::character varying), (NULL::text), (NULL::text), (NULL::text), (NULL::text)"

" Sort Method: quicksort Memory: 156kB"

" -> Append (cost=5.18..204.22 rows=384 width=216) (actual time=0.083..3.853 rows=1393 loops=1)"

" -> Aggregate (cost=5.18..5.19 rows=1 width=216) (actual time=0.082..0.083 rows=1 loops=1)"

" -> Seq Scan on salestable (cost=0.00..3.41 rows=141 width=4) (actual time=0.024..0.043 rows=141 loops=1)"

" -> HashAggregate (cost=5.53..5.59 rows=5 width=189) (actual time=0.062..0.065 rows=5 loops=1)"

" Group Key: salestable\_1.region"

" Batches: 1 Memory Usage: 24kB"

" -> Seq Scan on salestable salestable\_1 (cost=0.00..3.41 rows=141 width=9) (actual time=0.005..0.014 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 3"" (cost=5.53..5.64 rows=5 width=216) (actual time=0.056..0.058 rows=5 loops=1)"

" -> HashAggregate (cost=5.53..5.59 rows=5 width=191) (actual time=0.055..0.057 rows=5 loops=1)"

" Group Key: salestable\_2.store"

" Batches: 1 Memory Usage: 24kB"

" -> Seq Scan on salestable salestable\_2 (cost=0.00..3.41 rows=141 width=11) (actual time=0.007..0.015 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 4"" (cost=5.53..5.77 rows=11 width=216) (actual time=0.055..0.061 rows=11 loops=1)"

" -> HashAggregate (cost=5.53..5.66 rows=11 width=191) (actual time=0.054..0.059 rows=11 loops=1)"

" Group Key: salestable\_3.product"

" Batches: 1 Memory Usage: 24kB"

" -> Seq Scan on salestable salestable\_3 (cost=0.00..3.41 rows=141 width=11) (actual time=0.003..0.011 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 5"" (cost=5.53..5.57 rows=2 width=216) (actual time=0.051..0.052 rows=2 loops=1)"

" -> HashAggregate (cost=5.53..5.55 rows=2 width=190) (actual time=0.050..0.051 rows=2 loops=1)"

" Group Key: salestable\_4.category"

" Batches: 1 Memory Usage: 24kB"

" -> Seq Scan on salestable salestable\_4 (cost=0.00..3.41 rows=141 width=10) (actual time=0.003..0.011 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 6"" (cost=5.53..5.66 rows=6 width=216) (actual time=0.064..0.069 rows=6 loops=1)"

" -> HashAggregate (cost=5.53..5.60 rows=6 width=187) (actual time=0.063..0.067 rows=6 loops=1)"

" Group Key: salestable\_5.quarter"

" Batches: 1 Memory Usage: 24kB"

" -> Seq Scan on salestable salestable\_5 (cost=0.00..3.41 rows=141 width=7) (actual time=0.008..0.017 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 7"" (cost=5.88..6.19 rows=14 width=189) (actual time=0.064..0.067 rows=5 loops=1)"

" -> HashAggregate (cost=5.88..6.05 rows=14 width=164) (actual time=0.064..0.066 rows=5 loops=1)"

" Group Key: salestable\_6.region, salestable\_6.store"

" Batches: 1 Memory Usage: 24kB"

" -> Seq Scan on salestable salestable\_6 (cost=0.00..3.41 rows=141 width=16) (actual time=0.004..0.013 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 8"" (cost=5.88..6.19 rows=14 width=189) (actual time=0.072..0.097 rows=54 loops=1)"

" -> HashAggregate (cost=5.88..6.05 rows=14 width=164) (actual time=0.071..0.090 rows=54 loops=1)"

" Group Key: salestable\_7.region, salestable\_7.product"

" Batches: 1 Memory Usage: 32kB"

" -> Seq Scan on salestable salestable\_7 (cost=0.00..3.41 rows=141 width=16) (actual time=0.005..0.013 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 9"" (cost=5.88..6.10 rows=10 width=189) (actual time=0.066..0.071 rows=10 loops=1)"

" -> HashAggregate (cost=5.88..6.00 rows=10 width=163) (actual time=0.065..0.069 rows=10 loops=1)"

" Group Key: salestable\_8.region, salestable\_8.category"

" Batches: 1 Memory Usage: 24kB"

" -> Seq Scan on salestable salestable\_8 (cost=0.00..3.41 rows=141 width=15) (actual time=0.003..0.011 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 10"" (cost=5.88..6.19 rows=14 width=189) (actual time=0.067..0.079 rows=26 loops=1)"

" -> HashAggregate (cost=5.88..6.05 rows=14 width=160) (actual time=0.066..0.076 rows=26 loops=1)"

" Group Key: salestable\_9.region, salestable\_9.quarter"

" Batches: 1 Memory Usage: 24kB"

" -> Seq Scan on salestable salestable\_9 (cost=0.00..3.41 rows=141 width=12) (actual time=0.003..0.012 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 11"" (cost=5.88..6.19 rows=14 width=216) (actual time=0.071..0.122 rows=54 loops=1)"

" -> HashAggregate (cost=5.88..6.05 rows=14 width=166) (actual time=0.071..0.115 rows=54 loops=1)"

" Group Key: salestable\_10.store, salestable\_10.product"

" Batches: 1 Memory Usage: 32kB"

" -> Seq Scan on salestable salestable\_10 (cost=0.00..3.41 rows=141 width=18) (actual time=0.005..0.013 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 12"" (cost=5.88..6.10 rows=10 width=216) (actual time=0.067..0.072 rows=10 loops=1)"

" -> HashAggregate (cost=5.88..6.00 rows=10 width=165) (actual time=0.066..0.070 rows=10 loops=1)"

" Group Key: salestable\_11.store, salestable\_11.category"

" Batches: 1 Memory Usage: 24kB"

" -> Seq Scan on salestable salestable\_11 (cost=0.00..3.41 rows=141 width=17) (actual time=0.004..0.012 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 13"" (cost=5.88..6.19 rows=14 width=216) (actual time=0.068..0.080 rows=26 loops=1)"

" -> HashAggregate (cost=5.88..6.05 rows=14 width=162) (actual time=0.067..0.076 rows=26 loops=1)"

" Group Key: salestable\_12.store, salestable\_12.quarter"

" Batches: 1 Memory Usage: 24kB"

" -> Seq Scan on salestable salestable\_12 (cost=0.00..3.41 rows=141 width=14) (actual time=0.005..0.014 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 14"" (cost=5.88..6.19 rows=14 width=216) (actual time=0.065..0.071 rows=11 loops=1)"

" -> HashAggregate (cost=5.88..6.05 rows=14 width=165) (actual time=0.064..0.068 rows=11 loops=1)"

" Group Key: salestable\_13.product, salestable\_13.category"

" Batches: 1 Memory Usage: 24kB"

" -> Seq Scan on salestable salestable\_13 (cost=0.00..3.41 rows=141 width=17) (actual time=0.003..0.011 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 15"" (cost=5.88..6.19 rows=14 width=216) (actual time=0.064..0.079 rows=32 loops=1)"

" -> HashAggregate (cost=5.88..6.05 rows=14 width=162) (actual time=0.063..0.075 rows=32 loops=1)"

" Group Key: salestable\_14.product, salestable\_14.quarter"

" Batches: 1 Memory Usage: 24kB"

" -> Seq Scan on salestable salestable\_14 (cost=0.00..3.41 rows=141 width=14) (actual time=0.003..0.011 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 16"" (cost=5.88..6.15 rows=12 width=216) (actual time=0.061..0.065 rows=8 loops=1)"

" -> HashAggregate (cost=5.88..6.03 rows=12 width=161) (actual time=0.061..0.064 rows=8 loops=1)"

" Group Key: salestable\_15.category, salestable\_15.quarter"

" Batches: 1 Memory Usage: 24kB"

" -> Seq Scan on salestable salestable\_15 (cost=0.00..3.41 rows=141 width=13) (actual time=0.005..0.013 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 17"" (cost=6.23..6.54 rows=14 width=189) (actual time=0.080..0.106 rows=54 loops=1)"

" -> HashAggregate (cost=6.23..6.41 rows=14 width=139) (actual time=0.080..0.099 rows=54 loops=1)"

" Group Key: salestable\_16.region, salestable\_16.store, salestable\_16.product"

" Batches: 1 Memory Usage: 32kB"

" -> Seq Scan on salestable salestable\_16 (cost=0.00..3.41 rows=141 width=23) (actual time=0.003..0.011 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 18"" (cost=6.23..6.54 rows=14 width=189) (actual time=0.069..0.074 rows=10 loops=1)"

" -> HashAggregate (cost=6.23..6.41 rows=14 width=138) (actual time=0.068..0.072 rows=10 loops=1)"

" Group Key: salestable\_17.region, salestable\_17.store, salestable\_17.category"

" Batches: 1 Memory Usage: 24kB"

" -> Seq Scan on salestable salestable\_17 (cost=0.00..3.41 rows=141 width=22) (actual time=0.003..0.011 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 19"" (cost=6.23..6.54 rows=14 width=189) (actual time=0.078..0.091 rows=26 loops=1)"

" -> HashAggregate (cost=6.23..6.41 rows=14 width=135) (actual time=0.078..0.087 rows=26 loops=1)"

" Group Key: salestable\_18.region, salestable\_18.store, salestable\_18.quarter"

" Batches: 1 Memory Usage: 24kB"

" -> Seq Scan on salestable salestable\_18 (cost=0.00..3.41 rows=141 width=19) (actual time=0.005..0.013 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 20"" (cost=6.23..6.54 rows=14 width=189) (actual time=0.082..0.107 rows=54 loops=1)"

" -> HashAggregate (cost=6.23..6.41 rows=14 width=138) (actual time=0.081..0.101 rows=54 loops=1)"

" Group Key: salestable\_19.region, salestable\_19.product, salestable\_19.category"

" Batches: 1 Memory Usage: 32kB"

" -> Seq Scan on salestable salestable\_19 (cost=0.00..3.41 rows=141 width=22) (actual time=0.003..0.011 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 21"" (cost=6.23..6.54 rows=14 width=189) (actual time=0.187..0.282 rows=123 loops=1)"

" -> HashAggregate (cost=6.23..6.41 rows=14 width=135) (actual time=0.186..0.262 rows=123 loops=1)"

" Group Key: salestable\_20.region, salestable\_20.product, salestable\_20.quarter"

" Batches: 1 Memory Usage: 72kB"

" -> Seq Scan on salestable salestable\_20 (cost=0.00..3.41 rows=141 width=19) (actual time=0.005..0.016 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 22"" (cost=6.23..6.54 rows=14 width=189) (actual time=0.121..0.148 rows=35 loops=1)"

" -> HashAggregate (cost=6.23..6.41 rows=14 width=134) (actual time=0.119..0.141 rows=35 loops=1)"

" Group Key: salestable\_21.region, salestable\_21.category, salestable\_21.quarter"

" Batches: 1 Memory Usage: 32kB"

" -> Seq Scan on salestable salestable\_21 (cost=0.00..3.41 rows=141 width=18) (actual time=0.006..0.019 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 23"" (cost=6.23..6.54 rows=14 width=216) (actual time=0.137..0.186 rows=54 loops=1)"

" -> HashAggregate (cost=6.23..6.41 rows=14 width=140) (actual time=0.136..0.175 rows=54 loops=1)"

" Group Key: salestable\_22.store, salestable\_22.product, salestable\_22.category"

" Batches: 1 Memory Usage: 32kB"

" -> Seq Scan on salestable salestable\_22 (cost=0.00..3.41 rows=141 width=24) (actual time=0.006..0.020 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 24"" (cost=6.23..6.54 rows=14 width=216) (actual time=0.189..0.270 rows=123 loops=1)"

" -> HashAggregate (cost=6.23..6.41 rows=14 width=137) (actual time=0.188..0.253 rows=123 loops=1)"

" Group Key: salestable\_23.store, salestable\_23.product, salestable\_23.quarter"

" Batches: 1 Memory Usage: 104kB"

" -> Seq Scan on salestable salestable\_23 (cost=0.00..3.41 rows=141 width=21) (actual time=0.012..0.026 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 25"" (cost=6.23..6.54 rows=14 width=216) (actual time=0.109..0.126 rows=35 loops=1)"

" -> HashAggregate (cost=6.23..6.41 rows=14 width=136) (actual time=0.109..0.121 rows=35 loops=1)"

" Group Key: salestable\_24.store, salestable\_24.category, salestable\_24.quarter"

" Batches: 1 Memory Usage: 32kB"

" -> Seq Scan on salestable salestable\_24 (cost=0.00..3.41 rows=141 width=20) (actual time=0.006..0.014 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 26"" (cost=6.23..6.54 rows=14 width=216) (actual time=0.084..0.099 rows=32 loops=1)"

" -> HashAggregate (cost=6.23..6.41 rows=14 width=136) (actual time=0.083..0.094 rows=32 loops=1)"

" Group Key: salestable\_25.product, salestable\_25.category, salestable\_25.quarter"

" Batches: 1 Memory Usage: 32kB"

" -> Seq Scan on salestable salestable\_25 (cost=0.00..3.41 rows=141 width=20) (actual time=0.007..0.015 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 27"" (cost=6.58..6.90 rows=14 width=189) (actual time=0.099..0.124 rows=54 loops=1)"

" -> HashAggregate (cost=6.58..6.76 rows=14 width=113) (actual time=0.098..0.117 rows=54 loops=1)"

" Group Key: salestable\_26.region, salestable\_26.store, salestable\_26.product, salestable\_26.category"

" Batches: 1 Memory Usage: 32kB"

" -> Seq Scan on salestable salestable\_26 (cost=0.00..3.41 rows=141 width=29) (actual time=0.008..0.015 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 28"" (cost=6.58..6.90 rows=14 width=189) (actual time=0.143..0.198 rows=123 loops=1)"

" -> HashAggregate (cost=6.58..6.76 rows=14 width=110) (actual time=0.142..0.184 rows=123 loops=1)"

" Group Key: salestable\_27.region, salestable\_27.store, salestable\_27.product, salestable\_27.quarter"

" Batches: 1 Memory Usage: 104kB"

" -> Seq Scan on salestable salestable\_27 (cost=0.00..3.41 rows=141 width=26) (actual time=0.003..0.010 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 29"" (cost=6.58..6.90 rows=14 width=189) (actual time=0.080..0.095 rows=35 loops=1)"

" -> HashAggregate (cost=6.58..6.76 rows=14 width=109) (actual time=0.079..0.091 rows=35 loops=1)"

" Group Key: salestable\_28.region, salestable\_28.store, salestable\_28.category, salestable\_28.quarter"

" Batches: 1 Memory Usage: 32kB"

" -> Seq Scan on salestable salestable\_28 (cost=0.00..3.41 rows=141 width=25) (actual time=0.006..0.013 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 30"" (cost=6.58..6.90 rows=14 width=189) (actual time=0.105..0.160 rows=123 loops=1)"

" -> HashAggregate (cost=6.58..6.76 rows=14 width=109) (actual time=0.105..0.146 rows=123 loops=1)"

" Group Key: salestable\_29.region, salestable\_29.product, salestable\_29.category, salestable\_29.quarter"

" Batches: 1 Memory Usage: 104kB"

" -> Seq Scan on salestable salestable\_29 (cost=0.00..3.41 rows=141 width=25) (actual time=0.003..0.011 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 31"" (cost=6.58..6.90 rows=14 width=216) (actual time=0.147..0.229 rows=123 loops=1)"

" -> HashAggregate (cost=6.58..6.76 rows=14 width=111) (actual time=0.146..0.212 rows=123 loops=1)"

" Group Key: salestable\_30.store, salestable\_30.product, salestable\_30.category, salestable\_30.quarter"

" Batches: 1 Memory Usage: 104kB"

" -> Seq Scan on salestable salestable\_30 (cost=0.00..3.41 rows=141 width=27) (actual time=0.003..0.014 rows=141 loops=1)"

" -> Subquery Scan on ""\*SELECT\* 32"" (cost=6.94..7.25 rows=14 width=189) (actual time=0.162..0.250 rows=123 loops=1)"

" -> HashAggregate (cost=6.94..7.11 rows=14 width=84) (actual time=0.161..0.232 rows=123 loops=1)"

" Group Key: salestable\_31.region, salestable\_31.store, salestable\_31.product, salestable\_31.category, salestable\_31.quarter"

" Batches: 1 Memory Usage: 104kB"

" -> Seq Scan on salestable salestable\_31 (cost=0.00..3.41 rows=141 width=32) (actual time=0.008..0.019 rows=141 loops=1)"

"Planning Time: 2.316 ms"

"Execution Time: 9.177 ms"

**\*\*\* With CUBE Operator**:

"Sort (cost=105.51..106.47 rows=384 width=84) (actual time=7.144..7.241 rows=1393 loops=1)"

" Sort Key: region, product, category, store, quarter"

" Sort Method: quicksort Memory: 156kB"

" -> MixedAggregate (cost=8.44..89.03 rows=384 width=84) (actual time=0.373..3.671 rows=1393 loops=1)"

" Hash Key: region, category"

" Hash Key: region, category, store"

" Hash Key: store, quarter"

" Hash Key: store, quarter, region"

" Hash Key: store, quarter, region, product, category"

" Hash Key: category, quarter"

" Hash Key: category, quarter, region"

" Hash Key: category, quarter, region, product"

" Hash Key: product, quarter"

" Hash Key: product, quarter, category"

" Hash Key: region, product"

" Hash Key: region, product, category"

" Hash Key: store"

" Hash Key: store, product"

" Hash Key: store, product, quarter"

" Hash Key: store, product, quarter, region"

" Hash Key: quarter"

" Hash Key: quarter, region"

" Hash Key: quarter, region, product"

" Hash Key: category"

" Hash Key: category, store"

" Hash Key: category, store, quarter"

" Hash Key: category, store, quarter, region"

" Hash Key: product"

" Hash Key: product, category"

" Hash Key: product, category, store"

" Hash Key: product, category, store, quarter"

" Group Key: region, store, product, category"

" Group Key: region, store, product"

" Group Key: region, store"

" Group Key: region"

" Group Key: ()"

" Batches: 1 Memory Usage: 984kB"

" -> Sort (cost=8.44..8.80 rows=141 width=32) (actual time=0.263..0.281 rows=141 loops=1)"

" Sort Key: region, store, product, category"

" Sort Method: quicksort Memory: 32kB"

" -> Seq Scan on salestable (cost=0.00..3.41 rows=141 width=32) (actual time=0.025..0.043 rows=141 loops=1)"

"Planning Time: 0.807 ms"

"Execution Time: 8.065 ms"

**\*\*\* Without ROllUp operator:**

"Sort (cost=39.25..39.41 rows=62 width=130) (actual time=1.252..1.269 rows=291 loops=1)"

" Sort Key: salestable.region, salestable.product, salestable.category, salestable.store, salestable.quarter"

" Sort Method: quicksort Memory: 48kB"

" -> Append (cost=6.94..37.41 rows=62 width=131) (actual time=0.230..0.745 rows=291 loops=1)"

" -> HashAggregate (cost=6.94..7.11 rows=14 width=84) (actual time=0.229..0.284 rows=123 loops=1)"

" Group Key: salestable.region, salestable.product, salestable.category, salestable.store, salestable.quarter"

" Batches: 1 Memory Usage: 104kB"

" -> Seq Scan on salestable (cost=0.00..3.41 rows=141 width=32) (actual time=0.026..0.046 rows=141 loops=1)"

" -> HashAggregate (cost=6.58..6.76 rows=14 width=113) (actual time=0.108..0.131 rows=54 loops=1)"

" Group Key: salestable\_1.region, salestable\_1.product, salestable\_1.category, salestable\_1.store"

" Batches: 1 Memory Usage: 32kB"

" -> Seq Scan on salestable salestable\_1 (cost=0.00..3.41 rows=141 width=29) (actual time=0.010..0.020 rows=141 loops=1)"

" -> HashAggregate (cost=6.23..6.41 rows=14 width=138) (actual time=0.095..0.115 rows=54 loops=1)"

" Group Key: salestable\_2.region, salestable\_2.product, salestable\_2.category"

" Batches: 1 Memory Usage: 32kB"

" -> Seq Scan on salestable salestable\_2 (cost=0.00..3.41 rows=141 width=22) (actual time=0.003..0.012 rows=141 loops=1)"

" -> HashAggregate (cost=5.88..6.05 rows=14 width=164) (actual time=0.078..0.098 rows=54 loops=1)"

" Group Key: salestable\_3.region, salestable\_3.product"

" Batches: 1 Memory Usage: 32kB"

" -> Seq Scan on salestable salestable\_3 (cost=0.00..3.41 rows=141 width=16) (actual time=0.003..0.011 rows=141 loops=1)"

" -> HashAggregate (cost=5.53..5.59 rows=5 width=189) (actual time=0.056..0.058 rows=5 loops=1)"

" Group Key: salestable\_4.region"

" Batches: 1 Memory Usage: 24kB"

" -> Seq Scan on salestable salestable\_4 (cost=0.00..3.41 rows=141 width=9) (actual time=0.006..0.013 rows=141 loops=1)"

" -> Aggregate (cost=5.18..5.19 rows=1 width=216) (actual time=0.033..0.033 rows=1 loops=1)"

" -> Seq Scan on salestable salestable\_5 (cost=0.00..3.41 rows=141 width=4) (actual time=0.003..0.011 rows=141 loops=1)"

"Planning Time: 1.394 ms"

"Execution Time: 1.774 ms"

**\*\*\* With ROLLUP Operator:**

"GroupAggregate (cost=8.44..13.10 rows=62 width=84) (actual time=0.284..0.533 rows=291 loops=1)"

" Group Key: region, product, category, store, quarter"

" Group Key: region, product, category, store"

" Group Key: region, product, category"

" Group Key: region, product"

" Group Key: region"

" Group Key: ()"

" -> Sort (cost=8.44..8.80 rows=141 width=32) (actual time=0.270..0.277 rows=141 loops=1)"

" Sort Key: region, product, category, store, quarter"

" Sort Method: quicksort Memory: 32kB"

" -> Seq Scan on salestable (cost=0.00..3.41 rows=141 width=32) (actual time=0.028..0.045 rows=141 loops=1)"

"Planning Time: 0.665 ms"

"Execution Time: 0.834 ms"

**6. Conclusion**

Using CUBE and ROLLUP operators in SQL can greatly improve query efficiency by simplifying complex aggregation tasks and reducing the need for multiple manual GROUP BY queries combined with UNION operations. These operators are especially useful when you need to explore data from multiple perspectives or want to generate reports with different levels of detail.

However, they come with performance considerations when dealing with large datasets, and results can sometimes be harder to interpret without proper post-processing. It's essential to consider the size of your data, query performance, and the specific reporting needs when deciding whether to use these operators.

In cases where flexibility and simplicity are more important than fine-tuned performance, CUBE and ROLLUP provide an excellent way to enhance query efficiency and streamline reporting efforts.