Milestone 4

In this milestone, the goal is to improve the performance of Query 6 by appropriate precomputations. The query 6 calculated the top 20 category\_id, top 20 customer\_id, quantity sold and dollar value. The first query calculates the values without precomputing any data. It involves several intermediate tables and joins, which can be computationally expensive for large datasets. The second query addresses this issue by introducing two precomputed tables.

*WITH*

*-- top 20 product categories*

*TopCategory AS (*

*SELECT*

*pc.category\_id,*

*COALESCE(SUM(product\_list\_price\*(1-s.discount)),0) AS total\_TopCategory*

*FROM*

*Sales s*

*LEFT JOIN Products pr*

*ON pr.product\_id = s.product\_id*

*JOIN Product\_Category pc*

*ON pr.product\_id = pc.product\_id*

*GROUP BY pc.category\_id*

*ORDER BY total\_TopCategory DESC*

*LIMIT 20),*

*-- rank by category*

*rank\_category AS (*

*SELECT*

*ROW\_NUMBER() OVER (ORDER BY total\_TopCategory DESC) AS rank\_category,*

*category\_id,*

*total\_TopCategory*

*FROM*

*TopCategory),*

*-- top 20 customers*

*TopCustomers AS (*

*SELECT*

*cu.customer\_id,*

*COALESCE(SUM(product\_list\_price\*(1-s.discount)),0) AS total\_TopCustomers*

*FROM*

*Customers cu*

*LEFT JOIN Sales s*

*ON cu.customer\_id = s.customer\_id*

*LEFT JOIN Products pr*

*ON pr.product\_id = s.product\_id*

*GROUP BY cu.customer\_id*

*ORDER BY total\_TopCustomers DESC*

*LIMIT 20),*

*-- rank by customer*

*rank\_customer AS (*

*SELECT*

*ROW\_NUMBER() OVER (ORDER BY total\_TopCustomers DESC) AS rank\_customer,*

*customer\_id,*

*total\_TopCustomers*

*FROM*

*TopCustomers),*

*-- category-aware sales*

*ProductSales AS (*

*SELECT*

*s.customer\_id,*

*s.product\_id,*

*pc.category\_id,*

*s.product\_quantity,*

*s.discount,*

*pr.product\_list\_price*

*FROM*

*Sales s*

*JOIN Products pr*

*ON s.product\_id = pr.product\_id*

*JOIN Product\_Category pc*

*ON pr.product\_id = pc.product\_id)*

*SELECT*

*rank\_category.category\_id,*

*rank\_category.rank\_category,*

*rank\_customer.customer\_id,*

*rank\_customer.rank\_customer,*

*COALESCE(SUM(s.product\_quantity),0) as quantity,*

*COALESCE(SUM(s.product\_list\_price\*(1-s.discount)),0) as dollar\_value*

*FROM*

*rank\_customer*

*CROSS JOIN rank\_category*

*LEFT JOIN ProductSales s*

*ON rank\_customer.customer\_id = s.customer\_id*

*AND rank\_category.category\_id = s.category\_id*

*GROUP BY rank\_category.category\_id, rank\_category.rank\_category,*

*rank\_customer.customer\_id, rank\_customer.rank\_customer*

*ORDER BY rank\_customer.rank\_customer, rank\_category.rank\_category;*

Query complete 00:00:07.351

cost=884030.88..884044.88 rows=400 width=64

### **Adding indexes**

Based on the experiments on milestone 3, I added the following indices to the query for cost and time reduction.

* Product\_product\_id on product\_id for Products table
* Pc\_product\_id on product\_id for Product\_category table
* Sales\_customer\_id on customer\_id for Sales table

Query complete 00:00:05.505

cost=649009.44..649023.44 rows=400 width=64

# Precomputation

The original query calculates the total sales for each customer and category, then ranks them to identify the top 20. This can be computationally expensive, especially for large datasets.

## First Precomputed Table (TopCategory)

This table stores the precomputed total sales amount for each product category. It's populated periodically with data from the Sales, Products, and Product\_Category tables.

*CREATE TABLE TopCategoryPrecompute (*

*category\_id INT PRIMARY KEY,*

*total\_TopCategory DECIMAL(10,2) NOT NULL DEFAULT 0*

*);*

## Inserting data

*-- Populate the table with the logic from the original TopCategory CTE*

*INSERT INTO TopCategoryPrecompute (category\_id, total\_TopCategory)*

*SELECT*

*pc.category\_id,*

*COALESCE(SUM(product\_list\_price\*(1-s.discount)),0) AS total\_TopCategory*

*FROM*

*Sales s*

*LEFT JOIN Products pr*

*ON pr.product\_id = s.product\_id*

*JOIN Product\_Category pc*

*ON pr.product\_id = pc.product\_id*

*GROUP BY pc.category\_id;*

## Adding index

*CREATE INDEX idx\_topcategory\_precompute\_category\_id ON TopCategoryPrecompute(category\_id);*

## Second Precomputed Table (TopCustomers)

This table stores the precomputed total sales amount for each customer. Similar to TopCategoryPrecompute, it's populated periodically based on Sales, Customers, and potentially Products tables.

*CREATE TABLE TopCustomersPrecompute (*

*customer\_id INT PRIMARY KEY,*

*total\_TopCustomers DECIMAL(10,2) NOT NULL DEFAULT 0*

*);*

## Inserting data

*-- Populate the table with the logic from the original TopCustomers CTE*

*INSERT INTO TopCustomersPrecompute (customer\_id, total\_TopCustomers)*

*SELECT*

*cu.customer\_id,*

*COALESCE(SUM(product\_list\_price\*(1-s.discount)),0) AS total\_TopCustomers*

*FROM*

*Customers cu*

*LEFT JOIN Sales s*

*ON cu.customer\_id = s.customer\_id*

*LEFT JOIN Products pr*

*ON pr.product\_id = s.product\_id*

*GROUP BY cu.customer\_id*

*ORDER BY total\_TopCustomers DESC;*

## Adding index

*CREATE INDEX idx\_topcustomer\_precompute\_customer\_id ON TopCustomersPrecompute(customer\_id);*

# Main query with pre-computations

WITH

-- rank by category (**using precomputed data**)

rank\_category AS (

SELECT

ROW\_NUMBER() OVER (ORDER BY total\_TopCategory DESC) AS rank\_category,

category\_id

FROM

**TopCategoryPrecompute**

LIMIT 20),

-- rank by customer (**using precomputed data**)

rank\_customer AS (

SELECT

ROW\_NUMBER() OVER (ORDER BY total\_TopCustomers DESC) AS rank\_customer,

customer\_id

FROM

**TopCustomersPrecompute**

LIMIT 20),

-- Sales data with category join

ProductSales AS (

SELECT

s.customer\_id,

s.product\_id,

pc.category\_id,

s.product\_quantity,

s.discount,

pr.product\_list\_price

FROM

Sales s

JOIN Products pr

ON s.product\_id = pr.product\_id

JOIN Product\_Category pc

ON pr.product\_id = pc.product\_id)

SELECT

rank\_category.category\_id,

rank\_category.rank\_category,

rank\_customer.customer\_id,

rank\_customer.rank\_customer,

COALESCE(SUM(s.product\_quantity),0) as quantity,

COALESCE(SUM(s.product\_list\_price\*(1-s.discount)),0) as dollar\_value

FROM

rank\_customer

CROSS JOIN rank\_category

LEFT JOIN ProductSales s

ON rank\_customer.customer\_id = s.customer\_id

AND rank\_category.category\_id = s.category\_id

GROUP BY rank\_category.category\_id, rank\_category.rank\_category,

rank\_customer.customer\_id, rank\_customer.rank\_customer

ORDER BY rank\_customer.rank\_customer, rank\_category.rank\_category;

## Exploring the results

Query complete 00:00:00.168

cost=64814.09..64828.09 rows=400 width=64

# Summary

The main query leverages the precomputed tables directly, eliminating the need to recalculate total sales in each execution. It joins the precomputed tables with the rank\_customer and rank\_category CTEs (which use precomputed data) and the ProductSales table for category information. Finally, it aggregates quantities and dollar values (potentially using pre-calculated columns in ProductSales).

|  | | | |
| --- | --- | --- | --- |
| Experiment | Time | Cost | Modifications |
| 0 (no indices) | Query complete 00:00:07.351 | cost=884030.88..884044.88 rows=400 width=64 | - |
| 1 | Query complete 00:00:05.505 | cost=649009.44..649023.44 rows=400 width=64 | Adding indexes:  Product\_product\_id,  Pc\_product\_id,  Sales\_customer\_id |
| 2 | Query complete 00:00:00.168 | cost=64814.09..64828.09 rows=400 width=64 | Adding two precomputed tables: |

Upon closer examination, the **cost** improvement due to precomputation is estimated at approximately **89.97%**. This indicates that using precomputation makes the query nearly 90% more cost-efficient compared to the original version without precomputation.

Moreover, the **time** improvement percentage stands at approximately **97%**. This denotes that the precomputed query boasts an impressive 97% acceleration in execution time, demonstrating a substantial leap in efficiency compared to the initial query without precomputation.

* Precomputed tables significantly reduce the need to recalculate sales totals for each query execution, leading to faster query performance.
* Indexing the precomputed tables on frequently used columns (e.g., category\_id in TopCategoryPrecompute) further improves lookup speed.
* Overall, precomputing top customers and top categories significantly improved query performance where total sales calculations are a bottleneck.