Synapse Materialized View Demo

Table of Contents

[Objective 1](#_Toc61937580)

[Prerequisites 1](#_Toc61937581)

[Setup 2](#_Toc61937582)

[Initial Timing 2](#_Toc61937583)

[Create A Materialized View for this Query 3](#_Toc61937584)

[Reset the performance 3](#_Toc61937585)

[Test with the exact same query 3](#_Toc61937586)

[Observations 4](#_Toc61937587)

[Test with Filtered/Altered Version Of Query 4](#_Toc61937588)

[Materialized Views + Resultset cache 5](#_Toc61937589)

[To setup the demo 6](#_Toc61937590)

[Run the Demo of SQL Query Stress 9](#_Toc61937591)

[Result 9](#_Toc61937592)

[Summary: 10](#_Toc61937593)

[References: 11](#_Toc61937594)

[Authors 11](#_Toc61937595)

[Reviewer 11](#_Toc61937596)

# Objective

The objective of this demo is to show that 100s to thousands of users can run queries in less time than a query would take for one user without Resultset caching!

# Prerequisites

A Synapse Dedicated Pool with Adventureworks DW

Download the SQL Query Stress tool [here](https://github.com/ErikEJ/SqlQueryStress)

# Setup

-- turn off caching in the Master database

-- Run this command when connecting to the MASTER database

ALTER DATABASE dedpoolmoss SET RESULT\_SET\_CACHING OFF;

--reset performance

DBCC DROPCLEANBUFFERS

DBCC FREEPROCCACHE

DBCC DROPRESULTSETCACHE

--drop view if exists

drop view mviewtest

# Initial Timing

Run Aggregation Query without Materialized View and get timing

SELECT [StateProvinceName],[EnglishCountryRegionName],

sum(S.[SalesAmount]) as TotalSale,

sum(S.[TotalProductCost]) as TotalProductCost,

[EnglishProductName]

FROM [dbo].[FactInternetSales] as S

JOIN dbo.dimProduct as P on P.ProductKey= S.ProductKey

JOIN [dbo].[DimCustomer] DC

on DC.[customerkey]=S.customerkey

JOIN [dbo].[DimGeography] g

on g.[GeographyKey]=dc.[GeographyKey]

GROUP BY [EnglishProductName],

[StateProvinceName],[EnglishCountryRegionName]

In my test environment, it took 19 seconds for 3,093 rows (on expanded fact table of 125 million rows).

Table

Description automatically generated

Or can use DMV to find the time of 19657 miliseconds:

--- start test: determine what the request\_id is for the query

SELECT \* FROM sys.dm\_pdw\_exec\_requests

order by end\_time desc

Graphical user interface, text, application

Description automatically generated

# Create A Materialized View for this Query

CREATE materialized view mviewtest WITH (DISTRIBUTION=HASH([EnglishProductName])) AS

SELECT [StateProvinceName],[EnglishCountryRegionName],

sum(S.[SalesAmount]) as TotalSale,

sum(S.[TotalProductCost]) as TotalProductCost,

[EnglishProductName]

FROM [dbo].[FactInternetSales] as S

JOIN dbo.dimProduct as P on P.ProductKey= S.ProductKey

JOIN [dbo].[DimCustomer] DC

on DC.[customerkey]=S.customerkey

JOIN [dbo].[DimGeography] g

on g.[GeographyKey]=dc.[GeographyKey]

GROUP BY [EnglishProductName],

[StateProvinceName],[EnglishCountryRegionName]

## Reset the performance

--reset performance

DBCC DROPCLEANBUFFERS

DBCC FREEPROCCACHE

## Test with the exact same query

as the Materialized View:

---test with exact query

SELECT [StateProvinceName],[EnglishCountryRegionName],

sum(S.[SalesAmount]) as TotalSale,

[EnglishProductName]

FROM [dbo].[FactInternetSales] as S

JOIN dbo.dimProduct as P on P.ProductKey= S.ProductKey

JOIN [dbo].[DimCustomer] DC

on DC.[customerkey]=S.customerkey

JOIN [dbo].[DimGeography] g

on g.[GeographyKey]=dc.[GeographyKey]

GROUP BY [EnglishProductName],

[StateProvinceName],[EnglishCountryRegionName]

Using the DMV, it took 515 milliseconds or a half a second. So this is roughly 40X improvement (4,000%)!

Graphical user interface, text, application, email

Description automatically generated

The estimated Execution plan shows the plan used the Materialized view:

Graphical user interface, text, application

Description automatically generated

## Observations

Test Same Query except with Filter in Where Clause: so not exact query and also it only uses one of the Sums of the totalSale amount, it doesn’t use the TotalProductCost or Sum of Product Cost.

Since this is not the exact same query, it would not initially be in the Resultset cache based on the previous query. However, the Materialized View will still work.

Also, the Materialized View wasn’t explicitly referred to, the analyst/developer would not even need to know of it, but it would still be used.

# Test with Filtered/Altered Version Of Query

/\*

Test with altered query for one state/province name only.

This would not initially be in the Resultset cache because it is not exactly the same as

the preceeding query.

\*/

SELECT [StateProvinceName],[EnglishCountryRegionName],

sum(S.[SalesAmount]) as TotalSale,

[EnglishProductName]

FROM [dbo].[FactInternetSales] as S

JOIN dbo.dimProduct as P on P.ProductKey= S.ProductKey

JOIN [dbo].[DimCustomer] DC

on DC.[customerkey]=S.customerkey

JOIN [dbo].[DimGeography] g

on g.[GeographyKey]=dc.[GeographyKey]

WHERE stateprovincename= 'British Columbia'

GROUP BY [EnglishProductName],

[StateProvinceName],[EnglishCountryRegionName]

This query took 390 miliseconds,

Graphical user interface, text, application, email

Description automatically generated

Confirm that the Materialized View was used (“mvtest”) as below in the Execution Plan.

Graphical user interface, text, application

Description automatically generated

The Materialized View is also used even when not referred to and when for a Product not a State/Province as per below, which also takes about 390 miliseconds:

--just select products, see if materialized view is used

SELECT [StateProvinceName],[EnglishCountryRegionName],

sum(S.[SalesAmount]) as TotalSale,

[EnglishProductName]

FROM [dbo].[FactInternetSales] as S

JOIN dbo.dimProduct as P on P.ProductKey= S.ProductKey

JOIN [dbo].[DimCustomer] DC

on DC.[customerkey]=S.customerkey

JOIN [dbo].[DimGeography] g

on g.[GeographyKey]=dc.[GeographyKey]

WHERE [EnglishProductName] = 'Half-Finger Gloves, M'

GROUP BY [EnglishProductName],

[StateProvinceName],[EnglishCountryRegionName]

# Materialized Views + Resultset cache

For this test, we are going to create a stored procedure that will utilize the Materialized View and then use the SQL Query Stress tool to execute it with any possible EnglishProductName.

It will show that the initial executions will be with just the materialized views, but it the same values are used again, the resultset will be cached, resulting in even better performance.

## To setup the demo

Create a stored procedure as below

Create Procedure usp\_producttest

@EnglishProductName varchar(50)

as

set nocount on

SELECT [StateProvinceName],[EnglishCountryRegionName],

sum(S.[SalesAmount]) as TotalSale,

[EnglishProductName]

FROM [dbo].[FactInternetSales] as S

JOIN dbo.dimProduct as P on P.ProductKey= S.ProductKey

JOIN [dbo].[DimCustomer] DC

on DC.[customerkey]=S.customerkey

JOIN [dbo].[DimGeography] g

on g.[GeographyKey]=dc.[GeographyKey]

WHERE [EnglishProductName] = @EnglishProductName

GROUP BY [EnglishProductName],

[StateProvinceName],[EnglishCountryRegionName]

Test it using a hard-coded value:

exec [dbo].usp\_producttest @EnglishProductName = 'Half-Finger Gloves, M'

Next, to setup the SQL Query Stress with parameters:

1. Turn off the Collect I/O and Collect Time Statistics, as these are used in Synapse Dedicated

Graphical user interface, text, application

Description automatically generated

Setup and test the database connection

Graphical user interface, text, application

Description automatically generated

Put in the stored procedure call with a parameter:

exec [dbo].usp\_producttest @EnglishProductName = @EnglishProductName

Graphical user interface

Description automatically generated with medium confidence

Then put in the Parameter Substitution screen with the following which will get all values for English Product Name to feed to the stored proc. Make sure to click Get Columns and then drop down the column until it is populated as below.

SELECT [EnglishProductName]

FROM [dbo].[DimProduct]

Graphical user interface, application

Description automatically generated

Put the Number of Threads at 100 and run it once for a baseline:

Graphical user interface, text, application, email

Description automatically generated

Now turn on Resultset cache:

ALTER DATABASE dedpoolmoss SET RESULT\_SET\_CACHING ON;

## Run the Demo of SQL Query Stress

Then run it several times and see if improvement.

## Result

After running it several times, it should start to go faster as more and more of the specific queries are cached. This shows 200 queries (simulated users with different queries) were executed in 7 seconds, vs. 9 seconds for 100 users without Resultset caching but with Materialized views, which is an over 60% improvement! (9secs x 2 (for 200 instead of 100 users) =18 seconds vs. 7 seconds.)

Graphical user interface, application

Description automatically generated

You can confirm that many different queries were cached then were used by the following query:

SELECT sum(total\_elapsed\_time) as totaltime, count(result\_cache\_hit),

command

FROM sys.dm\_pdw\_exec\_requests

WHERE result\_cache\_hit = 1

group by command

Text

Description automatically generated Graphical user interface, application

Description automatically generated

# Summary:

Materialized Views provide a major performance and scalability boost, of 40X in the tested case. They are different than Resultset Cache in that the exact query doesn’t need to be used or even known of the materialized view for it to be used.

This allows to scale to many more users without using the cost and latency to spin up additional compute.

In addition, Materialized Views and Resultset Cache can be used together to provide even better performance. The Materialized View would be used the first time a query using that view is used, providing huge benefits, then when that exact query is used subsequently it is improved substantially more, in this case more than 2.5x (250%) faster than the Materialized view alone.

Recall that the original query, without a Materialized View took about 20 seconds. We were able to have 200 simulated users run similar queries in 7 seconds, again showing the scalability to multiple users by using Materialized Views and Resultset Caching.

# References:

[Performance tuning with result set caching - Azure Synapse Analytics | Microsoft Docs](https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/performance-tuning-result-set-caching)

[ErikEJ/SqlQueryStress: SQL query stress simulator created by Adam Machanic http://dataeducation.com/sqlquerystress-the-source-code/ (github.com)](https://github.com/ErikEJ/SqlQueryStress)

[SQL Query Stress Tool (mssqltips.com)](https://www.mssqltips.com/sqlservertip/2730/sql-query-stress-tool/#:~:text=%20SQL%20Query%20Stress%20Tool%20%201%20Step,we%20are%20going%20to%20clear%20out...%20More%20)

[DBCC SHOWRESULTCACHESPACEUSED (Transact-SQL) - SQL Server | Microsoft Docs](https://docs.microsoft.com/en-us/sql/t-sql/database-console-commands/dbcc-showresultcachespaceused-transact-sql?view=azure-sqldw-latest)

# Authors

Steve Moss, Phillip Novak

# Reviewer

Saket Suman