SQL 2 Assignment 2
Christofer Lindholm DE25

Uppgift

Ditt skript skall hämta information från databasen AdventureWorksDW2019, specifikt från tabellerna DimCustomer och FactInternetSales. Målet är att hämta minst kundens för- och efternamn för den kund som har köpt mest. Det är valfritt att även presentera det totala spenderade beloppet.

Skriptet bör innehålla flera sätt att hitta toppkunden så att effektiviteten kan jämföras. För varje metod skall Execution Plan, statistik samt I/O-, minnes- och CPU-användning visas. Valet av det mest effektiva skriptet skall även motiveras.

Utförande

Tre olika queries implementerades för att identifiera den kund som har spenderat mest i databasen AdventureWorksDW2019. Samtliga queries returnerar kundens för- och efternamn, medan Query 1 och Query 3 även returnerar det totala spenderade beloppet. Skillnaderna mellan queries ligger främst i hur aggregering, sortering och filtrering genomförs, vilket möjliggör en jämförelse av olika exekveringsstrategier och resursanvändning.
[SQL Kod 2.x: SQL_2_Assignment_2_CL.sql]

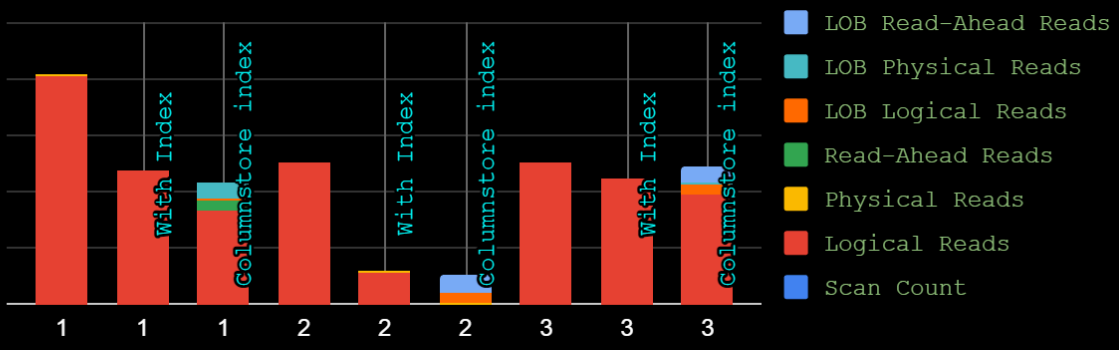
Inledande tester mot tabellen FactInternetSales visade att alla tre queries hade liknande exekveringstid. Däremot uppvisade Query 2 en betydligt lägre minnesförbrukning. Analys av execution plans visar att Query 2 hade en Memory Granted på 3016 kB, jämfört med 5648 kB för Query 1 och Query 3. Denna skillnad beror på att Query 2 inte returnerar det aggregerade försäljningsbeloppet, vilket minskar behovet av minne under exekveringen.
[Execution Plan: Query1, Query2, Query3.sqlplan]
[Mätvärden: SQL_2_Assignment_2_CL.xlsx]

För att erhålla mer tillförlitliga prestandamätningar skapades en större testtabell, RawFactInternetSalesBig, genom att duplicera innehållet i FactInternetSales hundra gånger. Varje query kördes därefter hundra gånger. För att minska påverkan av extremvärden exkluderades de fem snabbaste och fem långsammaste körningarna innan medelvärden för CPU-tid och total exekveringstid beräknades.
[SQL Kod 3.x: SQL_2_Assignment_2_CL.sql]

Resultaten visar att samtliga tre queries presterar mycket likvärdigt när de körs upprepade gånger mot den större tabellen. Detta indikerar att SQL Servers optimerare i stor utsträckning genererar liknande exekveringsplaner för dessa logiskt närliggande lösningar, och att skillnader i syntax har begränsad påverkan på den totala exekveringstiden.
[SQL Kod 4.x: SQL_2_Assignment_2_CL.sql]
[Execution Plan: Query1, Query2, Query3_BigTable.sqlplan]
[Mätvärden: SQL_2_Assignment_2_CL.xlsx]

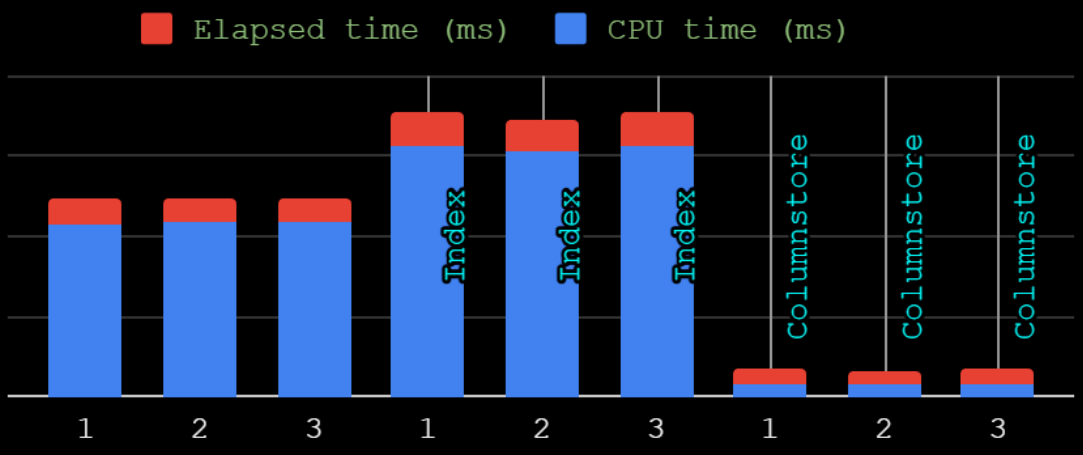
Indexering och Optimering

För att vidare analysera prestandan testades olika indexeringsstrategier. Först skapades ett nonclustered index på CustomerKey med SalesAmount inkluderad, följt av ett nonclustered columnstore index på samma kolumner.
[SQL Kod 5.x: SQL_2_Assignment_2_CL.sql]
[Execution Plan: Query1, Query2, Query3_With_Index.sqlplan]
[Execution Plan: Query1, Query2, Query3_BigTable_With_Index.sqlplan]
[Execution Plan: Query1, Query2, Query3_With_Columnstore_Index.sqlplan]
[Mätvärden: SQL_2_Assignment_2_CL.xlsx]



Nonclustered indexet reducerade antalet logical reads avsevärt för samtliga queries. Query 2 minskade sina logical reads från 1252 till 288, vilket indikerar effektivare dataåtkomst. Trots detta ökade CPU-tiden något. Detta pekar nog på att färre logical reads inte alltid leder till lägre CPU-användning.

Columnstoreindexet gav den tydligaste prestandaförbättringen vid aggregering över stora datamängder. Logical reads för FactInternetSales reducerades till nära noll, och den genomsnittliga exekveringstiden minskade markant när queries kördes hundra gånger. Nackdelen är försämrad skrivprestanda, vilket gör denna indexering är mindre lämplig för Transaktionsintensiva system.



Analys av Execution Plans

Genomgång av execution plans visar att samtliga queries använder snarlika operatorer när samma index är tillgängliga. Vid körning mot den större tabellen utnyttjade SQL Server parallelism, vilket bidrog till kortare exekveringstid. Detta är ett förväntat beteende vid frågor mot stora tabeller.

En tydligare skillnad observerades i minnesanvändningen. Query 2 krävde konsekvent minst tilldelat minne, vilket kan kopplas till dess enklare resultatuppsättning utan aggregerat försäljningsbelopp. Detta gör Query 2 till det mest minneseffektiva alternativet.

Slutsats

Samtliga tre queries uppvisar likvärdig exekveringstid och använder jämförbara execution plans när relevanta index är aktiverade. Query 2 framstår som det mest resurseffektiva alternativet ur ett minnesperspektiv, med cirka hälften så stor Memory Granted jämfört med Query 1 och Query 3, samtidigt som prestandan i övrigt är jämförbar. Nackdelen är att den inte returnerar det totala spenderade beloppet.

För scenarier där minimal exekveringstid är prioriterad och arbetslasten huvudsakligen är läsintensiv, är columnstore-index den mest effektiva lösningen. Detta sker dock på bekostnad av skrivprestanda.

Sammanfattningsvis erbjuder Query 2 den bästa balansen mellan prestanda och resursförbrukning, medan Query 1 och Query 3 ger mer komplett affärsinformation till priset av högre minnesanvändning.

Query 2 använder minst minne, har överlägset lägst logical reads och är lika snabb som de andra alternativen, men visar inte det totala spenderade beloppet.