**Univerzitet u Nišu, Elektronski fakultet**

**Katedra za računarstvo**



Obrada upita kod SQL Server baze podataka

Mentor: Doc. dr Aleksandar Stanimirović

Student: Ana Milenković 1524

Univerzitet u Nišu

Elektronski Fakultet Niš

Sadržaj

Table of Contents

[1.1 Šta je obrada upita? 4](#_Toc132570036)

[2.1 Režimi izvršenja 5](#_Toc132570037)

[2.1.1 Row režim 5](#_Toc132570038)

[2.1.2 Batch režim 5](#_Toc132570039)

[2.2 Obrada SQL izraza 6](#_Toc132570040)

[2.2.2 Optimizacija SELECT izraza 7](#_Toc132570041)

[2.2.3 Procesiranje SELECT izraza 10](#_Toc132570042)

[2.3 Skraćivanje konstanti i procena izraza 10](#_Toc132570043)

[2.3.1 Izrazi koji se mogu skratiti 11](#_Toc132570044)

[2.3.2 Izrazi koji se ne mogu skratiti 11](#_Toc132570045)

[2.3.3 Primeri konstantnih izraza koji se mogu i ne mogu skratiti 12](#_Toc132570046)

[2.3.4 Procena izraza 12](#_Toc132570047)

[2.3.5 Primeri procene izraza u vreme kompajliranja 13](#_Toc132570048)

[2.3.6 Obrada drugih izraza 14](#_Toc132570049)

[2.3.7 Obrada pogleda 14](#_Toc132570050)

[2.3.8 Obrada distribuiranih particionisanih pogleda 16](#_Toc132570051)

3. Keširanje i ponovno korišćenje plana izvršenja…………………………………………………….18

[3.1 Keširanje više planova za isti upit 20](#_Toc132570052)

[3.2 Uklanjanje planova izvršenja iz keša planova 23](#_Toc132570053)

[3.3 Rekompajliranje planova izvršenja 24](#_Toc132570054)

[3.4 Parametri i ponovno korišćenje planova izvršenja 25](#_Toc132570055)

[3.5 Jednostavna parametrizacija 27](#_Toc132570056)

[3.6 Forsirana parametrizacija 28](#_Toc132570057)

[3.7 Tipovi podataka parametara 29](#_Toc132570058)

[4.1 Stepen paralelizma 32](#_Toc132570059)

[4.1.1 Pregaziti stepen paralelizma 33](#_Toc132570060)

5. Primeri na realnoj bazi ……………………………………………………………………………..34  
6. Zaključak ……………………………………………….…………………………………………..42

Uvod

SQL server je sistem za upravljanje relacionom bazom podataka razvijen od strane Microsoft-a. To je moćna i fleksibilna platforma koja je široko rasprostranjena u poslovanju i organizacijama svih obima. Neke od glavnih odlika SQL servera su čuvanje i upravljanje podacima, skalabilnost, optimizacija performansi, sigurnost, poslovna inteligencija i analitika, razvojni alati.  
 SQL server pruža otpornu platformu za čuvanje i upravljanje podacima, koja uključuje podršku za tabele, poglede (*eng. views*), procedure, okidače (*triggers*) i mnoge druge objekte u bazi. Podržava skaliranje sa manjih okruženja sa jednim serverom, na velike multi-server konfiguracije. Ovo se postiže tehnologijama poput klasterovanja i replikacije. SQL server takođe uključuje i brojne alate za optimizaciju performansi baze podataka, gde spadaju planovi izvršenja upita, indeksiranje i statistike. Zaštita podataka se ogleda u robusnim bezbednosnim funkcionalnostima, kao što su enkripcija, upravljanje pristupom zasnovanom na ulogama, revizija. SQL server implementira i podršku za poslovnu analitiku kroz alate kao što su servisi SQL servera za analizu (*SQL Server Analysis Services -SSAS*), servisi za izveštaje (*SQL Server Reporting Services - SSRS*) i integracioni servisi (*SQL Server Integration Services - SSIS)*. Od razvojnih alata SQL server nudi moćne alate i API-je razvojnim timovima, kao što su SQL Server Menadžment Studio (SSMS), SQL Server Alat za podatke (*SQL Server Data Tools - SSDT*), i podršku za popularne programske jezike T-SQL, C# i Java.  
 SQL server dostupan je u više edicija, kao što su *Enterprise, Standard* i *Express.* Enterprise nudi najveći skup mogućnosti i alata, dok je Express edicija besplatna i lakša verzija dizajnirana za razvojna okruženja manjih obima. SQL server je moćna i svestrana platforma koja pomaže organizacijama svih veličina u efikasnoj analizi podataka.

1. Obrada upita

# 1.1 Šta je obrada upita?

Obrada upita je ključna komponenta sistema za upravljanje bazama podataka, koji obuhvata prevođenje i izvršavanje korisničkih upita nad bazom. Primarni cilj obrade upita je transformacija upita visokog nivoa pisanih na deklarativnom jeziku (na primer, SQL) u plan izvršenja niskog nivoa. Ovaj plan sastoji se od sekvence operacija koje se mogu efikasno izvršiti nad bazom.  
Obrada upita uglavnom uključuje sledeće korake:

1. Parsiranje – ulazni upit se parsira da bi se obezbedila njegova skladnost sa sintaksom jezika upita
2. Semantička analiza – upit se analizira da bi se osiguralo da je semantički ispravan i smislen. Ovaj korak vrši provere kao što su ispravnost imena tabela i kolona i pravilna upotreba operatora.
3. Optimizacija – optimizator upita generiše različite planove izvršenja za isti upit i bira najefikasniji u pogledu vremena izvršenja i iskorišćenih resursa. Optimizacione tehnike uključuju optimizaciju baziranu na ceni, optimizaciju baziranu na pravilima i heurističku optimizaciju.
4. Izvršenje - odabrani plan izvršenja se pokreće nad bazom, pritom generišući rezultate samog upita.

2. Arhitektura obrade upita – SQL Server

SQL Server endžin obrađuje upite nad raznim arhitekturama za čuvanje podataka, poput lokalnih tabela, particionisanih tabela, i tabela distribuiranim na više servera. U ovom odeljku su obrađene teme režimi izvršenja, obrada SQL izraza, izvršenje procedura, keširanje i ponovno korišćenje planova izvršenja, paralelna obrada upita.

# 2.1 Režimi izvršenja

SQL Server može obraditi Transact-SQL izraze korišćenjem dva različita režima obrade:

* Režim reda (*Row mode execution,* u tekstu nadalje Row režim)
* Režim gomile (*Batch mode execution ,* u tekstu nadalje Batch režim)

## 2.1.1 Row režim

Row režim je način obrade upita koji se koristi kod tradicionalnih RDBMS tabela, gde se podaci čuvaju u formatu redova. Kada se upit izvrši i pristupa podacima čuvanim u redovima tabela, operatori stabla izvršenja čitaju svaki potreban red, preko svih kolona specificiranih u šemi tabele. Iz svakog pročitanog reda, SQL Server potom pribavlja kolone koje se traže u skupu rezultata, onako kako su referencirane u SELECT izrazu, JOIN predikatu, ili filteru.  
Ovaj režim je veoma efikasan za OLTP scenario *(Online Transaction Processing) –* tip obrade koji izvršava određen broj transakcija koje se izvršavaju uporedo. Ali zato može biti manje efikasan kada se skeniraju velike količine podataka, na primer u slučaju *Data Warehouse*-a.

## 2.1.2 Batch režim

Batch režim je način za obradu upita koji se koristi za procesiranje više redova zajedno. Svaka kolona u gomili se čuva kao vektor u odvojenoj oblasti memorije, pa je zbog toga batch režim baziran na vektorima. Ovaj režim koristi algoritme optimizovane za procesore sa više jezgara i povećanim memorijskim izlazom koji su rasprostranjeni kod modernog hardvera.

Kada je ovaj režim prvi put uveden, bio je blisko integrisan i optimizovan za format čuvanja podataka po kolonama. Ipak, sa SQL Serverom 2019 (15.x) i u Azure SQL bazi, batch režim više ne zahteva indekse po kolonama.

Batch režim funkcioniše nad kompresovanim podacima kada je to moguće, i eliminiše operator razmene koji koristi Row režim. Rezultat ovoga je bolja paralelizacija i brže performanse.  
  
Kada se upit izvrši u ovom režimu, i pristupa podacima u indeksima kolona, operatori stabla izvršenja čitaju više redova zajedno u segmentima kolona. SQL Server čita samo kolone potrebne za rezultat, onako kako su referencirane u SELECT izrazu, JOIN predikatu, ili filteru. Ovaj režim je veoma efikasan u slučaju korišćenja Data Warehouse-a, gde se velike količine podataka čitaju i grupišu.

# 2.2 Obrada SQL izraza

Obrada jednog Transact-SQL izraza je najosnovniji način na koji SQL Server izvršava Transact-SQL izraze. Koraci koji se koriste u obradi jednog SELECT izraza koji referenciraju samo tabele u lokalnoj bazi (bez pogleda ili udaljenih tabela) ilustruju ovaj osnovni proces.  
  
2.2.1 Prvenstvo logičkih operatora   
Kada se u SELECT izrazu koristi više od jednog logičkog operatora, NOT se obrađuje prvo, zatim AND i na kraju OR. Aritmetički i operatori nad bitovima se obrađuju pre logičkih operatora.  
U sledećem primeru, uslov za boju odnosi se na model proizvoda 21, a ne na model 20, jer AND ima prednost u odnosu na OR:

SELECT ProductID, ProductModelID

FROM Production.Product

WHERE ProductModelID = 20 OR ProductModelID = 21

AND Color = 'Red';

Značenje upita se može promeniti dodavanjem zagrada da bi se forsirala obrada OR operatora pre AND. Sledeći upit pronalazi proizvode modela 20 i 21 koji su crveni.

SELECT ProductID, ProductModelID

FROM Production.Product

WHERE (ProductModelID = 20 OR ProductModelID = 21)

AND Color = 'Red';

Korišćenje zagrada, čak i kada one nisu obavezne, može poboljšati čitljivost upita, i smanjiti šanse za greške nastale zbog prvenstva operatora. Ne postoji veći uticaj korišćenja zagrada na performanse. Sledeći primer je čitljiviji od originalnog primera, iako su sintaksno jednaki:

SELECT ProductID, ProductModelID

FROM Production.Product

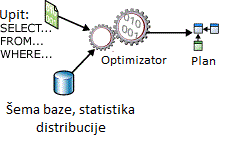
WHERE ProductModelID = 20 OR

(ProductModelID = 21 AND Color = 'Red');

## 2.2.2 Optimizacija SELECT izraza

SELECT izraz je ne-proceduralni; ne zadaje konkretne korake koje server treba da iskoristi da bi nabavio tražene podatke. Ovo znači da server mora da analizira izraz da bi odlučio koji je najefikasniji način za izvlačenje potrebnih podataka. Ovo se naziva optimizacija SELECT izraza. Komponenta koja radi ovo naziva se optimizator upita (*Query optimizer*). Ulaz na ovu komponentu sastoji se od upita, šeme baze (definicije tabela i indeksa), i statistike baze. Izlaz iz optimizatora je plan izvršenja upita, koji se ponekad naziva i plan upita, ili plan izvršenja.

Ulazi i izlazi optimizatora za vreme optimizacije jednog SELECT izraza prikazani su na sledećem dijagramu:

  
Slika 2.1 Optimizacija SELECT izraza

SELECT izraz definiše samo sledeće:

* Format skupa rezultata. Ovo je uglavnom specificirano u select listi. Ipak, druge klauzule poput ORDER BY i GROUP BY takođe utiču na krajnji izgled skupa rezultata.
* Tabele koje sadrže izvorne podatke. Ovo je specificirano u FORM klauzuli.
* Kako su tabele logički povezane u svrhe SELECT izraza. Ovo je definisano specifikacijama spoja, koji se može naći u WHERE klauzuli ili u ON klauzuli koja prati FROM.
* Uslove koje redovi u izvornoj tabeli moraju da zadovolje da bi se kvalifikovali za SELECT izraz. Ovi uslovi su specificirani u WHERE i HAVING klauzulama.

Plan izvršenja upita definiše sledeće:

* **Sekvencu u kojoj se pristupa izvornim tabelama.**  
  Tipično, postoje mnoge sekvence u kojima server može pristupiti tabelama u bazi da bi izgradio skup rezultata. Na primer, ako SELECT izraz referencira 3 tabele, server bi prvo mogao da pristupi Tabeli A, iskoristi podatke iz Tabele A da izvuče redove koji se poklapaju sa redovima iz Tabele B, a potom iskoristi podatke iz B da izvuče podatke iz tabele C. Druge sekvence u kojima se isto može izvršiti su:  
  Tabela C, Tabela B, Tabela A, ili   
  Tabela B, Tabela A, Tabela C, ili   
  Tabela B, Tabela C, Tabela A, ili   
  Tabela C, Tabela A, Tabela B
* **Metode koje se koriste za izvlačenje podataka iz svake tabele**  
  Generalno, postoje različite metode za pristupanje podacima iz svake tabele. Ako su potrebni samo par redova sa specifičnim vrednostima ključeva, server može iskoristiti indeks. Ako su potrebni svi redovi u tabeli, baza može ignorisati indekse i izvršiti skeniranje tabele. Ako su potrebni svi redovi u tabeli, ali postoji indeks čije su ključne kolone u ORDER BY klauzuli, izvršenje skeniranja indeksa umesto skeniranja tabele može uštedeti resurse. Ako je tabela veoma mala, skeniranje tabele može biti najefikasniji metod za skoro svaki pristup toj tabeli.
* **Metode koje se koriste za izračunavanja, kao i filtriranje, agregaciju i sortiranje podataka iz svake tabele.**    
  Postoji više načina da se izvrše kalkulacije nad podacima kao što su izračunavanje skalarnih vrednosti, agregacija i sortiranje podataka definisanih u tekstu upita, na primer kada se koristi GROUP BY ili ORDER BY klauzula, kao i za filtriranje podataka, na primer kada se koriste WHERE ili HAVING klauzule.

Proces biranja jednog plana izvršavanja od mnogih potencijalnih planova se naziva optimizacija. Optimizator upita je jedna od možda najbitnijih komponenti u endžinu baze. Na primer, ako se dvema konstrukcijskim kompanijama da isti nacrt za kuću, i prva kompanija prvih par dana provede planirajući kako da sagradi kuću, a druga odmah krene da gradi, kompanija koja je prethodno isplanirala će verovatno brže završiti.

SQL Server optimizator je baziran na ceni. Svaki mogući plan izvršenja ima svoju cenu po pitanju potrebnih resursa. Optimizator analizira sve planove i bira onaj sa najmanjom cenom. Neki SELECT izrazi imaju hiljade mogućih planova. U takvim slučajevima, optimizator koristi kompleksne algoritme za pronalazak plana čija je cena najbliža minimalnoj mogućoj.  
Optimizator ne bira samo plan sa najnižom cenom resursa; bira plan sa najrazumnijom cenom koji će najbrže vratiti rezultate korisniku. Na primer, izvršenje upita paralelno uglavnom troši više resursa, ali brže završi upit. Paralelno izvršenje se koristi ako server nije preopterećen.

SQL optimizator je važna komponenta jer omogućava serveru da se dinamički prilagodi na promene uslova u bazi bez potrebe za intervencijom programera ili administratora baze. Ovo dopušta programerima da se fokusiraju na opisivanje krajnjeg rezultata upita. Mogu da veruju da će optimizator kreirati efikasan plan izvršenja za trenutno stanje baze svaki put kada se izraz pokrene.

SQL Server Managment Studio ima tri opcija za prikaz ovih planova izvršenja:

* Procenjeni plan izvršenja ( *Estimated Execution Plan* ), koji je kompajlirani plan, kreiran od strane optimizatora upita.
* Konkretni plan izvršenja, koji je isti kao kompajlirani, ali sadrži i kontekst njegovog izvršenja. Ovo uključuje informacije za vreme pokretanja dostupne nakon što se izvrši, kao što su upozorenja, ili u novijim verzijama, proteklo i CPU vreme potrošeno u toku izvršenja.
* Statistike upita uživo, koje su iste kao kompajlirani plan, ali takođe sadrži i kontekst izvršenja. Ovaj kontekst se ažurira svake sekunde. Uključuje informacije o tačnom broju redova koji prolaze kroz operatore.

2.2.3 Procesiranje SELECT izraza

Kada se prethodno opisani procesi spoje, zajedno čine sledeću sekvencu koraka:

1. Parser skenira SELECT izraz i deli ga na logičke jedinice kao što su ključne reči, izrazi, operatori i identifikatori.
2. Stablo upita, ili sekvencijalno stablo kako se ponekad naziva, se kreira, opisujući logičke korake koje je potrebno izvršiti da bi se podaci transformisali u format zahtevan od strane skupa rezultata.
3. Optimizator upita analizira načine na koje se može pristupiti izvornom stablu. Onda bira seriju koraka koji će najbrže vratiti rezultate uz korišćenje što manje resursa. Stablo upita se ažurira da čuva ovu seriju koraka. Krajnji, optimizovani plan se naziva plan izvršenja upita.
4. Relacioni endžin počinje sa izvršenjem plana. Kako se koraci obrađuju, tako relacioni endžin traži od endžina za čuvanje podataka da prosledi podatke iz skupa redova koje je relacioni endžin tražio.
5. Relacioni endžin obrađuje podatke vraćene od endžina za čuvanje podataka u formatu definisanom u skupu rezultata i vraća krajnji rezultat korisniku.

# 2.3 Skraćivanje konstanti i procena izraza

SQL Server vrši procenu nekih izraza sa konstantama ranije, da bi poboljšao performanse upita. Ovo se naziva skraćivanje konstanti (*constant folding*). Konstanta je Transact-SQL literal, kao što su 3, ' ABC ', '2005-12-31', 1.0e3 ili 0x12345678.

## 2.3.1 Izrazi koji se mogu skratiti

SQL Server može primeniti skraćivanje konstanti na sledećim tipovima izraza:

* Aritmetički izrazi, poput 1+1 i 5 / 3 \* 2 , koji sadrže samo konstante.
* Logički izrazi, kao što su 1 = 1 i 1 > 2 AND 3 > 4 , koji sadrže samo konstante.
* Ugrađene funkcije koje se smatraju moguće za skraćivanje od strane SQL Servera, uključujući CAST i CONVERT. Generalno, unutrašnju funkciju je moguće skratiti ako je to funkcija svojih ulaza, a ne samo kontekstnih informacija, poput SET opcija, podešavanja jezika, opcija za bazu, i ključevi za šifriranje. Nedeterminističke funkcije se ne mogu skratiti. Determinističke ugrađene funkcije mogu, sa nekim izuzecima.
* Determinističke CLR (*Common Language Runtime*) metode korisnički definisanih tipova i determinističke CLR korisnički definisane funkcije.

Izuzetak su veliki tipovi objekata. Ako je tip izlaza procesa skraćivanja veliki objekat (text, ntext, image, nvarchar(max), varchar(max), varbinary(max), XML), onda SQL Server ne skraćuje taj izraz.

## 2.3.2 Izrazi koji se ne mogu skratiti

Svi ostali izrazi se ne mogu skratiti. Konkretno, sledeći izrazi se ne skraćuju:

* Izrazi koji nisu konstante, kao što su izrazi čiji rezultati zavise od vrednosti kolone.
* Izrazi čije vrednosti zavise od lokalnih promenljivih ili parametara, kao što su @x.
* Nedeterminističke funkcije.
* Korisnički definisane Transact-SQL funkcije.
* Izrazi čiji rezultati zavise od podešavanja jezika.
* Izrazi čiji rezultati zavise od SET opcija.
* Izrazi čiji rezultati zavise od konfiguracije servera

## 2.3.3 Primeri konstantnih izraza koji se mogu i ne mogu skratiti

Razmotriti sledeći upit:

SELECT \*

FROM Sales.SalesOrderHeader AS s

INNER JOIN Sales.SalesOrderDetail AS d

ON s.SalesOrderID = d.SalesOrderID

WHERE TotalDue > 117.00 + 1000.00;

Ako PARAMETERIZATION opcija nije postavljena na FORCED za ovaj upit, onda će izraz 117.00 + 1000.00 biti procenjen i zamenjen rezultatom , 1117.00, pre nego što se upit kompajlira. Koristi ovog skraćivanja uključuju sledeće:

* Izraz se ne mora stalno ponovo procenjivati za vreme izvršenja.
* Vrednosti izraza nakon što je izračunat se koristi od strane optimizatora upita za procenu veličine skupa rezultata za deo upita TotalDue > 117.00 + 1000.00.

Sa druge strane, ako je dbo.f skalarna korisnički definisana funkcija, izraz dbo.f(100) se ne skraćuje, jer SQL server ne skraćuje izraze koji uključuju korisnički definisane funkcije, čak i ako su determinističke.

## 2.3.4 Procena izraza

Dodatno, neki izrazi koji se ne skraćuju kao konstante ali čiji argumenti su poznati u vreme kompajliranja, bilo da su argumenti parametri ili konstante, se procenjuju od strane estimatora kardinalnosti koji je deo optimizatora, za vreme optimizacije.

Konkretno, sledeće ugrađene funkcije i specijalni operatori su procenjeni u vreme kompajliranja ako su svi njihovi ulazi poznati: UPPER, LOWER, RTRIM, DATEPART (samo godina), GETDATE, CAST, CONVERT. Sledeći operatori se takođe procenjuju u vreme kompajliranja ako su im ulazi poznati:

* Aritmetički operatori: + , - , \* , / , unarni -
* Logički operatori: AND, OR, NOT
* Operatori poređenja: <, >, <=, >=, < >, LIKE, IS NULL, IS NOT NULL

## 2.3.5 Primeri procene izraza u vreme kompajliranja

Razmotriti sledeću proceduru:

CREATE PROCEDURE MyProc( @d datetime )

AS SELECT COUNT(\*)

FROM Sales.SalesOrderHeader

WHERE OrderDate > @d+1;

Za vreme optimizacije SELECT upita u proceduri, optimizator pokušava da proceni očekivanu kardinalnost rezultata za uslov OrderDate > @d+1. Ovaj izraz nije skraćen, jer je @d parametar. Ipak, u vreme optimizacije, vrednost parametra je poznata. Ovo omogućava optimizatoru da precizno proceni veličinu skupa rezultata, što pomaže u kreiranju dobrog plana.

Sada razmotriti primer sličan prethodnom, osim što lokalna promenljiva d2 menja d+1 u upitu i izraz se procenjuje u SET naredbi umesto u upitu.

CREATE PROCEDURE MyProc2( @d datetime )

AS BEGIN

DECLARE @d2 datetime

SET @d2 = @d+1

SELECT COUNT(\*)

FROM Sales.SalesOrderHeader

WHERE OrderDate > @d2

END;

Kada se SELECT izraz u MyProc2 optimizuje, vrednosti d2 nije poznata. Stoga, optimizator koristi drugačiju procenu za selektivnost izraza OrderDate > @d2.

## 2.3.6 Obrada drugih izraza

Osnovni koraci opisani za obradu SELECT upita primenjuju se i na ostale Transact-SQL izraze poput INSERT, UPDATE, DELETE. UPDATE i DELETE izrazi moraju da ciljaju skup redova koje je potrebno modifikovati odnosno obrisati. Proces identifikacije ovih redova je isti kao proces koji određuje izvorne redove koji se koriste za kreiranje rezultata kod SELECT naredbe. UPDATE i INSERT naredbe mogu imati ugnježdeni SELECT koji dostavlja podatke koje treba ubaciti ili ažuritati.

Čak i naredbe jezika definicije podataka (*Data Definition Language - DDL*), kao što su CREATE PROCEDURE ili ALTER TABLE, se ultimativno svode na seriju relacionih operacija nad sistemskim katalogom tabela i ponekad (kao ALTER TABLE ADD COLUMN) i nad tabelama sa podacima.

## 2.3.7 Obrada pogleda

SQL Server procesor upita tretira indeksirane i neindeksirane poglede drugačije:

* Redovi indeksiranih pogleda se čuvaju u bazi u istom formatu kao i tabela. Ako optimizator odluči da koristi indeksirani pogled u planu izvršenja, taj pogled će biti tretiran isto kao i osnovna tabela.
* Čuva se samo definicija neindeksiranog pogleda, ne i redovi tog pogleda. Optimizator upita integriše logiku iz definicije pogleda u plan izvršenja koji gradi za neki Transact SQL izraz koji referencira neindeksirani pogled.

Logika koju koristi optimizator da odluči kada da koristi indeksirani pogled je slična logici korišćenoj za odlučivanje kada koristiti indeks nad tabelom. Ako podaci u indeksiranom pogledu pokrivaju ceo ili deo izraza, i optimizator odluči da je indeks nad pogledom putanja niske cene, optimizator će odabrati indeks nezavisno od toga da li je pogled referenciran imenom u upitu.

Kada izraz referencira neindeksirani pogled, parser i optimizator analiziraju izvor izraza i pogleda, a zatim ih preslikavaju u krajnji plan izvršenja. Ne postoji jedan plan za izraz, a drugi odvojeni plan za pogled.

Na primer, razmotriti sledeći pogled:

CREATE VIEW EmployeeName AS

SELECT h.BusinessEntityID, p.LastName, p.FirstName

FROM HumanResources.Employee AS h

JOIN Person.Person AS p ON h.BusinessEntityID = p.BusinessEntityID;

Na osnovu ovog pogleda, oba sledeća Transact SQL izraza rade iste operacije nad osnovnim tabelama i kreiraju iste rezultate:

/\* SELECT koji referencira EmployeeName pogled \*/

SELECT LastName AS EmployeeLastName, SalesOrderID, OrderDate

FROM Sales.SalesOrderHeader AS soh

JOIN dbo.EmployeeName AS EmpN

ON (soh.SalesPersonID = EmpN.BusinessEntityID)

WHERE OrderDate > '20020531';

/\* SELECT koji referencira Person i Employee tabele direktno \*/

SELECT LastName AS EmployeeLastName, SalesOrderID, OrderDate

FROM HumanResources.Employee AS e

JOIN Sales.SalesOrderHeader AS soh

ON soh.SalesPersonID = e.BusinessEntityID

JOIN Person.Person AS p

ON e.BusinessEntityID =p.BusinessEntityID

WHERE OrderDate > '20020531';

SQL Server menadžment studio prikazuje da endžin kreira isti plan izvršenja za oba ova SELECT izraza.

## 2.3.8 Obrada distribuiranih particionisanih pogleda

Procesor upita poboljšava performanse distribuiranih particionisanih pogleda. Najbitniji aspekt performansi ovakvih pogleda je minimizacija količine podataka koji se prenose između servera.

SQL Server kreira pametne, dinamičke planove koji efikasno koriste distribuirane upite da pristupe podacima iz udaljenih tabela:

* Procesor upita prvo koristi OLE DB (*Object Linking and Embredding*) da pribavi uslove za proveru ograničenja iz svake tabele. Ovo omogućava procesoru da mapira distribuciju ključeva preko svih tabela.
* Procesor upita potom poredi raspone ključeva specificiranih u izrazu, u okviru WHERE naredbe, sa mapom koja prikazuje kako su redovi distribuirani u tabelama. Procesor potom kreira plan izvršenja koji koristi distribuirane upite da pribavi samo one udaljene redove koji su potrebni za izvršenje izraza. Plan je takođe kreiran tako da bilo koji pristup udaljenim tabelama, bilo za podatke ili meta podatke, je odložen do samog trenutka kada su podaci potrebni.

Na primer, razmotriti sistem gde je Customers tabela particionisana na Server1 (CustomerID od 1 do 3299999), Server 2 (od 3300000 do 6599999) i Server 3 (od 6600000 do 9999999).

Sledeći upit se izvršava na Serveru 1:

SELECT \*

FROM CompanyData.dbo.Customers

WHERE CustomerID BETWEEN 3200000 AND 3400000;

Plan izvršenja povlači redove sa identifikatorima od 3200000 do 3299999 iz lokalne tabele, dok za redove od 3300000 do 3400000 izdaje distribuirani upit iz Servera 2.

Procesor takođe ugrađuje dinamičku logiku u planove izvršenja za izraze u kojima vrednosti ključeva nisu poznate kada je plan potrebno izgraditi. Pogledati sledeću proceduru:

CREATE PROCEDURE GetCustomer @CustomerIDParameter INT

AS SELECT \*

FROM CompanyData.dbo.Customers

WHERE CustomerID = @CustomerIDParameter;

SQL Server ne može predvideti koja vrednost ključa će biti prosleđena kao @CustomerIDParameter svaki put kada se procedura izvršava. Zbog toga, procesor takođe ne može predvideti kojoj distribuiranoj tabeli treba pristupiti. Da bi se obradio ovakav slučaj, SQL Server kreira plan koji ima uslovnu logiku, nazvanu dinamički filteri, koji kontrolišu kojoj tabeli će se pristupiti, na osnovu ulaznog parametra. Pod pretpostavkom da se ova procedura izvršava na Serveru 1, logika plana izvršenja se može predstaviti na sledeći način:

IF @CustomerIDParameter BETWEEN 1 and 3299999

Retrieve row from local table CustomerData.dbo.Customer\_33

ELSE IF @CustomerIDParameter BETWEEN 3300000 and 6599999

Retrieve row from linked table Server2.CustomerData.dbo.Customer\_66

ELSE IF @CustomerIDParameter BETWEEN 6600000 and 9999999

Retrieve row from linked table Server3.CustomerData.dbo.Customer\_99

SQL Sever ponekad kreira ove tipove dinamičkih planova čak i za upite koji nisu parametrizovani. Optimizator može parametrizovati upit da bi se plan izvršenja mogao ponovo iskoristiti. Ako optimizator parametrizuje upit koji referencira particionisani pogled, ne može više predvideti koji redovi će doći iz koje distribuirane tabele. Iskoristiće dinamičke filtere u planu izvršenja.

3. Keširanje i ponovno korišćenje plana izvršenja

SQL Server poseduje memoriju posvećenu čuvanju planova izvršenja i baferovanju podataka. Procenat memorije alociran bilo planovima, bilo baferima, se dinamički menja, u zavisnosti od stanja sistema. Deo memorije koji se koristi za čuvanje planova izvršenja se naziva keš planova.

Keš planova ima dva skladišta za čuvanje svih kompajliranih planova:

* Skladište za objektne planove, koje čuva planove koji su vezani za objekte koji se perzistiraju (procedure, funkcije, trigeri).
* Skladište za SQL planove, koje čuva planove vezane za autoparametrizovane, dinamičke, ili pripremljene upite.

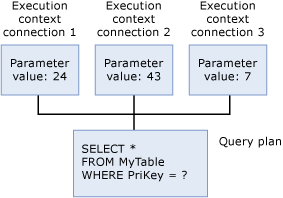
Sledeći upit daje informacije o iskorišćenoj memoriji za ova dva skladišta keša:

SELECT \* FROM sys.dm\_os\_memory\_clerks

WHERE name LIKE '%plans%';

SQL Server planovi izvršenja imaju sledeće glavne komponente:

* **Kompajlirani plan (ili plan upita)**  
  Plan upita se kreira u procesu kompajliranja, i predstavlja read-only strukturu podataka koju mogu koristiti veći broj korisnika. Čuva podatke o:
  + Fizičkim operatorima koji implementiraju operaciju opisanu logičkim operatorima.
  + Redosled operatora koji implementiraju operaciju opisanu logičkim operatorima.
  + Broj procenjenih redova koji prolaze kroz operatore.
  + Koje podržavajuće objekte treba kreirati u tempdb. Ne čuvaju se kontekst, niti informacije u vreme pokretanja u planu upita. Ne postoji nikada više od jedne ili dve kopije plana izvršenja u memoriji: jedna za sva serijska izvršenja, i druga za sva paralelna. Paralelna kopija pokriva sva paralelna izvršenja, nezavisno od stepena paralelizma.
* **Kontekst izvršenja**   
  Svaki korisnik koji trenutno izvršava upit poseduje strukturu podataka koja čuva podatke specifične za njihovo izvršenje, kao što su vrednosti parametara. Ova struktura se naziva kontekst izvršenja. Strukture podataka konteksta se koriste više puta, ali njihov sadržaj, odnosno sami podaci, ne. Ako drugi korisnik izvrši isti upit, strukture podataka se ponovo inicijalizuju sa kontekstom novog korisnika.

  
Slika 3.1 Ponovno korišćenje strukture konteksta izvršenja

Kada bilo koji izraz biva izvršen, endžin prvo pregleda keš planova da proveri da li već postoji plan za taj isti izraz. Izraz se smatra postojećim ako u potpunosti odgovara prethodno izvršenom SQL izrazu sa keširanim planom, karakter po karakter. SQL Server ponovo koristi bilo koji plan koji nađe. Ukoliko takav plan ne postoji, Server generiše novi plan za upit.

SQL Server ima efikasan algoritam za pronalazak bilo kojih postojećih planova izvršenja za bilo koji specifični SQL izraz. U većini sistema, minimalni resursi potrebni za ovu pretragu su manji od resursa koji se čuvaju time što se postojeći planovi ponovo koriste umesto da se ponovo kompajliraju.

Algoritmi koji porede novi SQL izraz sa postojećim, neiskorišćenim planovima u kešu, zahtevaju da sve reference na objekte budu u celosti kvalifikovane. Na primer, pretpostaviti da je Person osnovna šema za korisnika koji izvršava date SQL upite. Dok u ovom primeru se ne zahteva da Person tabela bude u celosti kvalifikovana, u drugom izrazu se neće pronaći poklapanje sa postojećim planom izvršenja, dok će treći SQL izraz imati poklapanje:

SELECT \* FROM Person; GO   
SELECT \* FROM Person.Person; GO   
SELECT \* FROM Person.Person; GO

Promena bilo kojih od sledećih SET opcija za dati plan izvršenja će uticati na sposobnost ponovnog korišćenja plana, jer endžin vrši skraćivanje konstanti i ove opcije će uticati na rezultate takvih izraza, jer se upiti proveravaju slovo po slovo:

ANSI\_NULL\_DFLT\_OFF, FORCEPLAN, ARITHABORT, DATEFIRST, ANSI\_PADDING, NUMERIC\_ROUNDABORT, ANSI\_NULL\_DFLT\_ON, LANGUAGE, CONCAT\_NULL\_YIELDS\_NULL, DATEFORMAT, ANSI\_WARNINGS,QUOTED\_IDENTIFIER, ANSI\_NULLS, NO\_BROWSETABLE, ANSI\_DEFAULTS

# 3.1 Keširanje više planova za isti upit

Upiti i planovi izvršenja se jedinstveno identifikuju u endžinu baze, slično kao i otisak prsta:

* Heš plana upita je binarna heš vrednost izračunata nad planom za dati upit, i koristi se da jedinstveno identifikuje slične planove.
* Heš upita je binarna heš vrednost izračunata nad Transact-SQL tekstom upita, i koristi se da jedinstveno odredi upite.

Kompajlirani plan se može pribaviti iz keša korišćenjem ručke za plan (*Plan Handle,* hendler na dalje u tektsu), koji je tranzijentni identifikator koji ostaje konstanta samo dok plan ostaje u kešu. Plan hendler je heš vrednost izvedena iz kompajliranog plana celog batch-a. Plan hendler ostaje isti za dati kompajlirani plan, čak i ako se jedan ili više izraza rekompajlira.

Ako je plan kompajliran za batch umesto za pojedinačan izraz, plan za svaki izraz u batch-u se može nabaviti korišćenjem hendlera plana i pomeraja izraza. DMV (*Dynamic Manage-ment Views)* sys.dm\_exec\_requests sadrži statement\_start\_offset i statement\_end\_offset kolone za svaki red, koji se odnose na izraz koji se trenutno izvršava u okviru batch-a koji se takođe trenutno izvršava.

Tekst batch-a se čuva u odvojenom memorijskom prostoru od keša plana, koij se zove SQL Menadžer keš. Tekst za kompajlirani plan se može pribaviti korišćenjem SQL hendlera, koji je tranzijentni identifikator koji ostaje konstanta dok je barem jedan plan koji referencira i dalje u kešu plana. SQL hendler je heš vrednost izvedena iz celokupnog batch teksta i zagarantovano je jedinstven za svaki batch.

Sledeći upit pribavlja informacije o korišćenju memorije za menadžer keš:

SELECT \* FROM sys.dm\_os\_memory\_objects WHERE type = 'MEMOBJ\_SQLMGR';

Postoji 1:N veza između sql hendlera i hendlera plana. Takav slučaj se javlja kada je ključ keša za kompajlirani plan drugačiji. Ovo se može desiti usled promene SET opcija između dva izvršenja u istom batch-u.

Pogledati sledeću proceduru:

CREATE PROCEDURE usp\_SalesByCustomer @CID int

AS SELECT \* FROM Sales.Customers

WHERE CustomerID = @CID

GO

SET ANSI\_DEFAULTS ON

GO

EXEC usp\_SalesByCustomer 10

GO

Verifikovati šta se može pronaći u kešu plana sledećim upitom:

SELECT cp.memory\_object\_address, cp.objtype, refcounts, usecounts, qs.query\_plan\_hash, qs.query\_hash, qs.plan\_handle, qs.sql\_handle

FROM sys.dm\_exec\_cached\_plans AS cp

CROSS APPLY sys.dm\_exec\_sql\_text (cp.plan\_handle)

CROSS APPLY sys.dm\_exec\_query\_plan (cp.plan\_handle)

INNER JOIN sys.dm\_exec\_query\_stats AS qs ON qs.plan\_handle = cp.plan\_handle

WHERE text LIKE '%usp\_SalesByCustomer%'

GO

Rezultat je sledeći:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| memory\_object\_address | objtype | refcounts | usecounts | query\_plan\_hash | query\_hash |
| 0x000001CC6C534060 | Proc | 2 | 1 | 0x3B4303441A1D7E6D | 0xA05D5197DA1EAC2D |

plan\_handle

0x0500130095555D02D022F111CD01000001000000000000000000000000000000000000000000000000000000

sql\_handle

0x0300130095555D02C864C10061AB000001000000000000000000000000000000000000000000000000000000

Sada je potrebno izvršiti istu proceduru sa drugim parametrom, ali nijednom drugom promenom konteksta izvršenja:

EXEC usp\_SalesByCustomer 8 GO

Verifikovati ponovo šta se može naći kešu planova:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| memory\_object\_address | objtype | refcounts | usecounts | query\_plan\_hash | query\_hash |
| 0x000001CC6C534060 | Proc | 2 | 2 | 0x3B4303441A1D7E6D | 0xA05D5197DA1EAC2D |

plan\_handle 0x0500130095555D02D022F111CD01000001000000000000000000000000000000000000000000000000000000

sql\_handle 0x0300130095555D02C864C10061AB000001000000000000000000000000000000000000000000000000000000

Primetiti da se usercounts povećalo na 2, što znači da se isti keširani plan ponovo iskoristio, jer se batch struktura kontekstnih podataka takođe ponovo koristila. Sada promeniti SET ANSI\_DEFAULTS opciju i izvršiti proceduru opet.

SET ANSI\_DEFAULTS OFF GO

EXEC usp\_SalesByCustomer 8 GO

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| memory\_object\_address | objtype | refcounts | usecounts | query\_plan\_hash | query\_hash |
| 0x000001CD01DEC060 | Proc | 2 | 1 | 0x3B4303441A1D7E6D | 0xA05D5197DA1EAC2D |
| 0x000001CC6C534060 | Proc | 2 | 2 | 0x3B4303441A1D7E6D | 0xA05D5197DA1EAC2D |

plan\_handle 0x0500130095555D02B031F111CD01000001000000000000000000000000000000000000000000000000000000 0x0500130095555D02D022F111CD01000001000000000000000000000000000000000000000000000000000000 sql\_handle 0x0300130095555D02C864C10061AB000001000000000000000000000000000000000000000000000000000000 0x0300130095555D02C864C10061AB000001000000000000000000000000000000000000000000000000000000

Primetiti kako sada postoje dva reda u sys.dm\_exec\_cached\_plans izlazu:

* Usercounts kolona pokazuje vrednost 1 za prvi red, koji predstavlja plan izvršen nakon što je promenjena SET opcija.
* Usercounts kolona pokazuje vrednost 2 u drugom redu, pre izmene SET opcije.
* Različita memorijska adresa odnosi se na različiti plan u kešu planova. Ipak, vrednost sql hendlera je ista za oba reda jer se odnose na isti batch.

Ovo efektivno znači da postoje dva reda plana u kešu koja odgovaraju istom batch-u, i to pokazuje koliko je važno obezbediti da su planovi u kešu pre i posle promene SET opcija isti, kako bi se omogućilo njihovo ponovno korišćenje.

# 3.2 Uklanjanje planova izvršenja iz keša planova

Planovi izvršenja ostaju u kešu planova dokle god postoji dovoljno memorije da ih čuva. Kada se vrši pritisak nad memorijom, SQL Server endžin koristi pristup baziran na ceni da odluči koje planove izvršenja da ukloni iz keša. Da bi doneo ovakvu odluku, endžin povećava i smanjuje trenutne cene promenljivih za svaki plan, poštujući sledeće faktore.

Kada korisnički proces ubaci plan u keš, taj proces postavlja trenutnu cenu jednaku originalnoj ceni kompajliranog upita; za momentalno izvršenje planova, korisnički proces postavlja trenutnu cenu na nulu.

Posle toga, svaki put kada korisnički proces referencira plan, resetuje trenutnu cenu na originalnu cenu; za momentalno izvršenje plana, proces povećava trenutnu cenu.

Kada se javi pritisak na memoriju, endžin reaguje uklanjanjem planova iz keša. Konstantno skenira stanje svakog plana i uklanja planove čije je trenutna cena nula. Dok vrši skeniranje, endžin gura cene ka nuli tako što smanjuje trenutnu cenu, ako upit trenutno ne koristi taj plan. Ovaj postupak se nastavlja sve dok se ne ukloni dovoljno planova da se zadovolje potrebe za memorijom. Dok postoji pritisak nad memorijom, cene planova se mogu više puta smanjivati i povećavati. Kada nema više pritiska, endžin prestaje da smanjenjem trenutne cene nekorišćenih planova i svi planovi ostaju u kešu, čak iako im je cena nula. Izbacuju se samo za vreme skeniranja.

Sledeći primer ilustruje kako se uklanjaju planovi iz keša planova:

* Plan izvršenja se često referencira tako da njegova cena nikada ne padne na nulu. Plan ostaje u kešu i ne uklanja se, osim ako ne postoji memorijski pritisak i trenutna cena je nula.
* Ubačen je plan koji se momentalno izvršava i ne referencira se ponovo, pre nego što postoji memorijski pritisak. Pošto se ovakvi planovi inicijalizuju sa trenutnom cenom nula, endžin će ukloniti ovaj plan dok postoji memorijski pritisak. Ovaj plan sa cenom nula ostaje u kešu sve dok nema memorijskog pritiska.

# 3.3 Rekompajliranje planova izvršenja

Određene promene u bazi mogu izazvati da plan izvršenja postane neefikasan ili nevažeći, na osnovu novog stanja baze. SQL Server prepoznaje promene koje su plan učinile nevažećim i obeležava plan. Novi plan se mora rekompajlirati za sledeću konekciju koja izvršava upit. Uslovi koji invalidiraju plan su sledeći:

* Promene nad tabelom ili pogledom koji se referenciraju u upitu (ALTER TABLE i ALTER VIEW).
* Promene nad procedurom, koje bi odbacile sve planove za tu proceduru iz keša (ALTER PROCEDURE).
* Promene nad bilo kojim indeksima koje plan koristi.
* Ažuriranje statistika korišćenih od strane plana izvršenja, generisano eksplicitno u naredbi, kao što je UPDATE STATISTICS, ili generisano automatski.
* Brisanje indeksa korišćenog od strane plana.
* Eksplicitni poziv sp\_recompile
* Veliki broj promena ključeva (generisane od strane INSERT ili DELETE izraza drugih korisnika koji menjaju tabelu referenciranu u upitu).
* Za tabele sa trigerima, ako broj redova u ubačenoj ili obrisanoj tabeli značajno poraste.
* Izvršenje procedure korišćenjem WITH RECOMPILE opcije.

Većina rekompajliranja su obavezni bilo za ispravnost izraza, ili za potencijalno brže izvršenje planova.

Kod SQL Server verzija pre 2005, kad god bi izraz iz batch-a izazvao rekompajliranje, ceo batch bi bio rekompajliran. Sa početkom verzije SQL Server 2005 (9.x), samo izraz koji izazove rekompajliranje se zaista i rekompajlira. Rekompajliranje na nivou izraza poboljšava performanse jer, u većini slučajeva, mali broj izraza izaziva ponovno kompajliranje.

# 3.4 Parametri i ponovno korišćenje planova izvršenja

Korišćenje parametara, uključujući markere parametara u ADO, OLE DB i ODBC aplikacijama, može povećati višestruku upotrebljivost planova izvršenja.

Korišćenje parametara za čuvanje vrednosti koje se tipiziraju od strane krajnjeg korisnika je takođe mnogo bezbednije nego nadovezivanje vrednosti u string, koji bi se potom izvršio upotrebom API metode za pristup podacima, EXECUTE izrazom, ili sp\_executesql procedurom.

Jedina razlika između sledeća dva SELECT izraza su vrednosti koje se porede u WHERE klauzuli:

SELECT \* FROM AdventureWorks2014.Production.Product

WHERE ProductSubcategoryID = 1;

SELECT \* FROM AdventureWorks2014.Production.Product

WHERE ProductSubcategoryID = 4;

Jedina razlika između planova izvršenja ova dva upita je vrednost koja se čuva za poređenje sa ProductSubcategoryID kolonom. Dok je cilj SQL Servera da uvek prepozna kada izrazi generišu iste planove, nekada ne može detektovati ovo u kompleksnim Transact-SQL izrazima.

Odvajanje konstanti iz SQL izraza korišćenjem parametara pomaže relacionom endžinu da prepozna duplikate planova. Parametri se mogu iskoristiti na sledeći način:

* Kod Transact-SQL-a, koristiti sp\_executesql:  
  DECLARE @MyIntParm INT   
  SET @MyIntParm = 1 EXEC sp\_executesql   
  N'SELECT \* FROM AdventureWorks2014.Production.Product   
  WHERE ProductSubcategoryID = @Parm',   
  N'@Parm INT',   
  @MyIntParm  
  Ovaj metod je preporučen za skripte, procedure ili trigere koji generišu SQL izraze dinamički.
* ADO, OLE DB i ODBC koriste markere parametara. Markeri su znakovi pitanja koji zamenjuju konstantu u izrazu i vezuju se za promenljivu. Na primer, u ODBC aplikaciji:  
  - Koristiti SQLBindParameter za vezivanje celobrojne promenljive za prvi marker u sql izrazu  
  - Postaviti celobrojnu vrednost u promenljivu  
  - Izvršiti izraz, specificirajući marker parametra (?):  
   SQLExecDirect(hstmt,   
  "SELECT \* FROM AdventureWorks2014.Production.Product  
   WHERE ProductSubcategoryID = ?",   
  SQL\_NTS);
* Za dizajniranje procedura, koje koriste parametere po dizajnu

Ako se parametri eksplicitno ne ubace u dizajn aplikacije, može se računati na optimizator upita za automatsku parametrizaciju određenih upita uz pomoć mehanizma jednostavne parametrizacije. Alternativno, može se forsirati optimizator tako da razmotri parametrizaciju svih upita u bazi postavljanjem opcije PARAMETRIZATION na FORCED.

# 3.5 Jednostavna parametrizacija

U SQL Serveru, korišćenje parametara ili markera u SQL izrazima povećava mogućnost relacionog endžina da prepozna nove SQL izraze kao već postojeće, prethodno kompajlirane planove.

Ako se SQL izraz izvrši bez parametara, SQL Server parametrizuje izraz interno, da bi povećao šanse za prepoznavanje izraza. Ovaj proces se naziva jednostavna parametrizacija. U SQL Server verzijama pre 2005, ovaj proces se nazivao auto-parametrizacija.

Razmotriti sledeći izraz:

SELECT \* FROM AdventureWorks2014.Production.Product

WHERE ProductSubcategoryID = 1;

Vrednost 1 na kraju izraza se može specificirati kao parametar. Relacioni endžin kreira plan za ovaj batch kao da je parametar specificiran umesto vrednosti 1. Zahvaljujući ovoj jednostavnoj parametrizaciji, SQL Server prepoznaje da prethodni izrazi generišu u osnovi isti plan i koristi prvi plan i za drugi izraz:

SELECT \* FROM AdventureWorks2014.Production.Product WHERE ProductSubcategoryID = 1;

SELECT \* FROM AdventureWorks2014.Production.Product WHERE ProductSubcategoryID = 4;

Kada endžin obrađuje kompleksne SQL izraze, može imati poteškoća u određivanju koji izrazi se mogu parametrizovati. Da bi se poboljšale mogućnosti poklapanja postojećih izraza sa nekorišćenih planovima, eksplicitno treba specificirati parametre korišćenjem sp\_executesql ili markerima. Jedna napomena, kada se koriste operatori poput -,+, \*, /, % za implicitnu ili eksplicitnu konverziju konstantnih vrednosti tipa int, smallint, tinzint, bigint u float, real, decimal ili numeric, SQL SERVER primenjuje specifična pravila za računanje tipa i preciznosti rezultata. Ipak, ova pravila se razlikuju u zavisnosti od toga da li je upit parametrizovan ili ne. Stoga, neki slični izrazi mogu ponekad da kreiraju različite rezultate.

Kada se koristi objektno relacioni maper kao što je Entity Framework, aplikacioni upiti poput LINQ stabala upita ili određeni SQL upiti ne mogu biti parametrizovani, što utiče na ponovno korišćenje planova i mogućnost praćenja upita u bazi upita.

# 3.6 Forsirana parametrizacija

Osnovno ponašanje prilikom parametrizacije se može izmeniti specificiranjem da sve SELECT, INSERT, UPDATE i DELETE naredbe u bazi budu parametrizovane, uz neka ograničenja. Forsirana parametrizacija se uključuje postavljanjem opcije PARAMETRIZATION na FORCED. Ovo može poboljšati performanse određenih baza tako što smanjuje frekvenciju kompajliranja i rekompajliranja upita. Baze koje mogu imati koristi od forsirane parametrizacije su uglavnom one koje imaju visok nivo konkurentnih upita iz izvora kao što su aplikacije o trenucima prodaje.

Kada parametrizacija postaje forsirana, bilo koja vrednost literala koja se pojavljuje u SELECT, INSERT, UPDATE ili DELETE izrazu, se konvertuje u parametar za vreme kompajliranja upita. Izuzeci su literali koji se pojavljuju u sledećim konstrukcijama upita:

* INSERT....EXECUTE izrazima
* Izrazima unutar tela procedura, trigera ili korisnički definisanih funkcija. SQL Server već koristi postojeće planove za te rutine.
* Pipremljene izraze koji su već parametrizovani od strane klijentske aplikacije.
* Izraze koji sadrže XQuery pozive metoda, gde se metoda pojavljuje u kontekstu gde su njeni argumenti uglavnom parametrizovani, kao što je WHERE klauzula. Ako se metod pojavljuje u kontekstu gde se njegovi argumenti ne bi parametrizovali, ostatak izraza se parametrizuje.
* Izrazi unutar SQL kursora. (SELECT izrazi unutar API kursora se parametrizuju.)
* Zastarele konstrukcije upita.
* Bilo koji izraz koji je pokrenut u kontekstu ANSI\_PADDING ili ANSI\_NULLS kada se postavlja na OFF.
* Izrazi koji sadrže više od 2097 literala koje je inače moguće parametrizovati.
* Izrazi koji referenciraju promenljive, kao što su WHERE t.col2 >= @bb
* Izrazi koji sadrže RECOMPILE hint za upit.
* Izrazi koji sadrže COMPUTE klauzulu.
* Izrazi koji sadrže WHERE CURRENT OF klauzulu.

Parametrizacija se odigrava na nivou individualnog izraza, što znači da se u okviru batch-a parametrizuju pojedinačni izrazi. Nakon kompajliranja, parametrizovani upit se izvršava u kontekstu batch-a u kojem je originalno primljen.

Ako je plan izvršenja keširan, može se odrediti da li je upit parametrizovan referenviranjem sql kolone sys.syscacheobjects dinamičkog pogleda. Ako jeste parametrizovan, imena i tipovi podataka parametara dolaze pre teksta predatog batch-a u ovoj koloni, na primer (@1 tinyint).  
Imena parametara su nasumična. Korisnici ili aplikacije ne treba da se oslanjaju na određeni redosled imenovanja. Takođe, imenovanje se može menjati iz verzije u verziju SQL Servera.

# 3.7 Tipovi podataka parametara

Kada SQL Server parametrizuje literale, parametri se konvertuju u sledeće tipove podataka:

* Celobrojni literali čija bi se veličina inače uklapala u int tip podataka parametrizuju se na int. Veći celobrojni literali koji su delovi predikata koji uključuju bilo koji operator poređenja (uključuje <, <=, =, !=, >, >=, !<, !>, <>, ALL, ANY, SOME, BETWEEN i IN) parametrizuju se u numeric(38,0). Veći literali koji nisu delovi predikata koji uključuju operatore poređenja parametrizuju se u numeric čija je preciznost dovoljno velika da podrži njegovu veličinu i čija je skala 0.
* Numerički literali sa fiksnim zarezom koji su delovi predikata koji uključuju operatore poređenja parametrizuju se na numeric čija je preciznost 38 i čija je skala dovoljno velika da podrži njegovu veličinu. Numerički literali fiksnog zareza koji nisu delovi predikata koji uključuju operatore poređenja parametrizuju se na numeričke vrednosti čija su preciznost i razmera dovoljno veliki da podrže njegovu veličinu.
* Numerički literali sa pokretnim zarezom parametrizuju se u float(53).
* Literali stringova koji nisu Unicode parametrizuju se na varchar(8000) ako se literal uklapa u 8.000 znakova i na varchar(max) ako je veći od 8.000 znakova.
* Unicode string literali se parametrizuju na nvarchar(4000) ako se literal uklapa u 4.000 Unicode znakova i na nvarchar(max) ako je literal veći od 4.000 znakova.
* Binarni literali se parametrizuju na varbinari(8000) ako se literal uklapa u 8.000 bajtova. Ako je veći od 8.000 bajtova, konvertuje se u varbinary(max).
* Literali tipa money se parametrizuju na money.

4. Obrada paralelnih upita

SQL Server obezbeđuje paralelne upite kako bi optimizovao izvršenje upita i indeksne operacije za računare koji imaju više od jednog mikroprocesora (CPU). Operacija se može završiti brzo i efikasno jer SQL Server može paralelno izvršavati upite ili indeksne operacije korišćenjem više niti operativnog sistema.

Za vreme optimizacije upita, SQL Server pretražuje upite ili indeksne operacije koje mogu imati koristi od paralelnog izvršenja. Za ove upite, ubacuje operatore razmene u plan izvršenja upita da bi ga pripremio za paralelno izvršenje. Operator razmene je operator u planu koji obezbeđuje upravljanje procesima, raspodelu podataka i upravljanje i kontrolu toka. Operator razmene uključuje Distribute Streams, Repartition Streams, i Gather Streams logičke operatore kao pod-tipove, od kojih se jedan ili više mogu pojaviti u izlazu plana za izvršenje paralelnog upita.

Neke konstrukcije u okviru SQL Servera nasleđuju mogućnost da upravljaju paralelizmom na nivou celog plana izvršenja ili nekog njegovog dela. Takve konstrukcije su:

* Skalarne korisnički definisane funkcije  
  Od verzije SQL Server 2019, endžin ima mogućnost da ove funkcije pretvori u inline, i da otključa paralelizam za vreme obrade upita.
* Udaljeni upiti
* Dinamički kursori
* Rekurzivni upiti
* Više-izrazne funckije na nivou tabela
* TOP ključna reč

Plan izvršenja može sadržati atribut NonParallerPlanReason u QueryPlan elementu, koji opisuje zašto se paralelizam nije koristio. Neki od primera vrednosti ovog atributa su:

|  |  |
| --- | --- |
| **NonParallelPlanReason Vrednost** | **Opis** |
| MaxDOPSetToOne | Maksimalni nivo paralelizma postavljen na 1. |
| EstimatedDOPIsOne | Procenjeni nivo paralelizma je 1. |
| NoParallelWithRemoteQuery | Paralelizam nije podržan za odaljene upite. |
| NoParallelDynamicCursor | Paralelni planovi nisu podržani za dinamičke kursore. |
| NoParallelFastForwardCursor | Paralelni planovi nisu podržani za fast forward kursore. |
| NoParallelCursorFetchByBookmark | Paralelni planovi nisu podržani za kursore koji se pribavljaju putem obeleživača. |
| NoParallelCreateIndexInNonEnterpriseEdition | Kreiranje paralelnih indeksa nije podržano za edicije koje nisu Enterprise. |
| NoParallelPlansInDesktopOrExpressEdition | Paralelni planovi nisu podržani za Desktop i Express edicije. |
| NonParallelizableIntrinsicFunction | Upit referencira funkciju koja se ne može paralelizovati. |
| CouldNotGenerateValidParallelPlan | Verifikacija paralelnog plana je neuspela, vrši se povratak na serijalni. |

Nakon ubacivanja operatora razmene, rezultat je plan izvršenja paralelnog upita. Ovaj plan može koristiti više od jedne niti. Serijski plan izvršenja, koji koristi serijski upit, koristi samo jednu nit prilikom izvršenja. Konkretan broj niti koje koriste paralelni upit se utvrđuje u vreme inicijalizacije plana i ustanovljava na osnovu kompleksnosti plana i stepena paralelizma (*Degree of Parallelism* – DOP).

Stepen paralelizma (DOP) određuje maksimalni broj CPU-ova koji se koriste; to nije jednako broju niti koje se koriste. DOP granica se postavlja na nivou task-a, ne po zahtevu ili po ograničenju upita. Ovo znači da za vreme izvršenja paralelnog upita, jedan zahtev može kreirati više task-ova koji se dodeljuju raspoređivačima (*scheduler*). Više procesora od specificiranih od strane MAXDOP se mogu koristiti konkurentno u bilo kom trenutku izvršenja upita, kada se različiti task-ovi izvršavaju uporedo.

SQL Server optimizator ne koristi paralelni plan za upit ako je jedan od sledećih uslova tačan:

* Serijski plan je trivijalan, ili ne prelazi granicu cene za uključivanje paralelizma.
* Serijski plan ima manju ukupnu procenjenu cenu od paralelnog plana koji je otkrio optimizator.
* Upit sadrži skalarne ili relacione operatore koji se ne mogu pokrenuti u paraleli. Određeni operatori mogu izazvati da se deo plana izvrši serijski, ili da se ceo plan izvrši u serijskom modu.

4.1 Stepen paralelizma

SQL Server automatski detektuje najbolji stepen paralelizma za svaku instancu paralelnog upita. Ovo radi na osnovu sledećih kriterijuma:

1. Da li je SQL Server pokrenut na računaru koji ima više od jednog mikroprocesora ili CPU, kao što su simetrični multiprocesorski računari. Samo računari sa više od jednog CPU mogu koristiti paralelne upite.
2. Da li je dostupan dovoljan broj niti. Svaki upit ili indeksna operacija zahtevaju određen broj niti za izvršenje. Izvršenje paralelnog plana zahteva više niti od serijskog, i broj potrebnih niti se povećava sa stepenom paralelizma. Kada zahtev za određenim brojem niti za plan izvršenja ne može biti zadovoljen, SQL Server smanjuje stepen paralelizma automatski ili u potpunosti napušta paralelni plan u specificiranom kontekstu. Potom izvršava serijski plan.
3. Tipa upita ili indeksne operacije koju treba izvršiti. Indekse operacije koje kreiraju ili rekreiraju indekse, ili brišu klasterovani indeks, i upiti koji mnogo koriste CPU cikluse su najbolji kandidati za paralelni plan. Na primer, JOIN više velikih tabela, velike agregacije, sortiranje velikih skupova rezultata su dobri kandidati. Jednostavni upiti, koji se često nalaze u transakcionim aplikacijama zahtevaju više koordinacije da bi se izvršili paralelno i ti zahtevi su često skuplji od koristi koje bi se ostvarile paralelnim izvršenjem. SQL Server poredi cenu izvršenja upita sa graničnom cenom paralelizma. Ova vrednost predstavlja granicu, koja ako bude prevaziđena, označava da se paralelno izvršenje više isplati. Korisnici mogu menjati ovu vrednost sa početne vrednosti 5, korišćenjem sp\_configure, ukoliko je testiranje pokazalo da je neka druga vrednost bolja.
4. Da li ima dovoljno redova za obradu. Ako optimizator upita odluči da je broj redova previše mali, neće uvesti operatore razmene za distribuciju redova. Stoga, operatori se izvršavaju serijski, Izvršenje operatora u serijskom planu izbegava scenarije gde pokretanje, distribucija i koordinacija koštaju više od dobiti koje se ostvaruju paralelnim izvršenjem.
5. Da li su dostupne trenutne statistike o distribuciji. Ako je najviši stepen paralelizma nemoguće dostići, niži stepeni se razmatraju pre nego što se paralelni plan odbaci. Na primer, prilikom kreiranja klasterovanog indeksa nad pogledom, distribuciona statistika se ne može proceniti, jer klasterovani indeks još ne postoji.

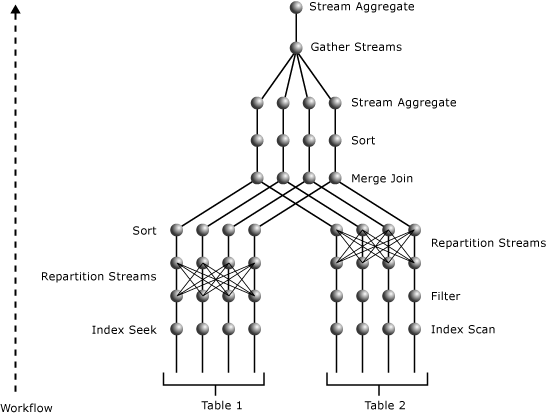
U vreme izvršenja, endžin odlučuje da li trenutno opterećenje sistema i prethodno opisane informacije o konfiguraciji sistema omogućavaju paralelno izvšrenje. Ako je paralelizam odobren, endžin određuje optimalni broj niti i raspoređuje izvršenje paralelnog plana na te niti. Isti broj niti se koristi od početka do kraja izvršenja. Endžin proverava optimalni broj niti svaki put kada se plan pribavi iz keša. Na primer, izvršenje plana prvi put može rezultovati serijskim izvršenjem, a kasnije izvršenje može biti paralelno sa tri niti, dok sledeće izvršenje može biti sa četiri niti.

Update i Delete operatori u paralelnom izvršenju se vrše serijski, ali WHERE klauzula u UPDATE i DELETE izrazu može biti izvršena paralelno. Izmene nad podacima se potom serijski primenjuju nad bazom.

## 4.1.1 Pregaziti stepen paralelizma

Stepen paralelizma postavlja broj procesora koji se koriste u izvršenju paralelnog plana. Ova konfiguracija se može postaviti na više nivoa:

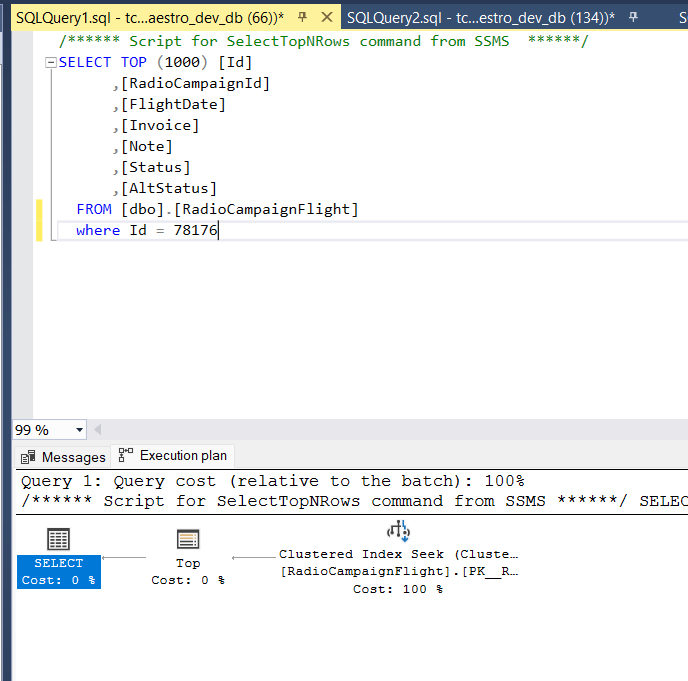
1. Na nivou servera korišćenjem MAXDOP opcije konfiguracije servera.
2. Na nivou radnog opterećenja, korišćenjem MAX\_DOP opcije konfiguracije upravljača resursima.
3. Na nivou baze, korišćenjem MAXDOP opcije u opsegu baze.
4. Na nivou upita, korišćenjem MAXDOP hint-a upita.



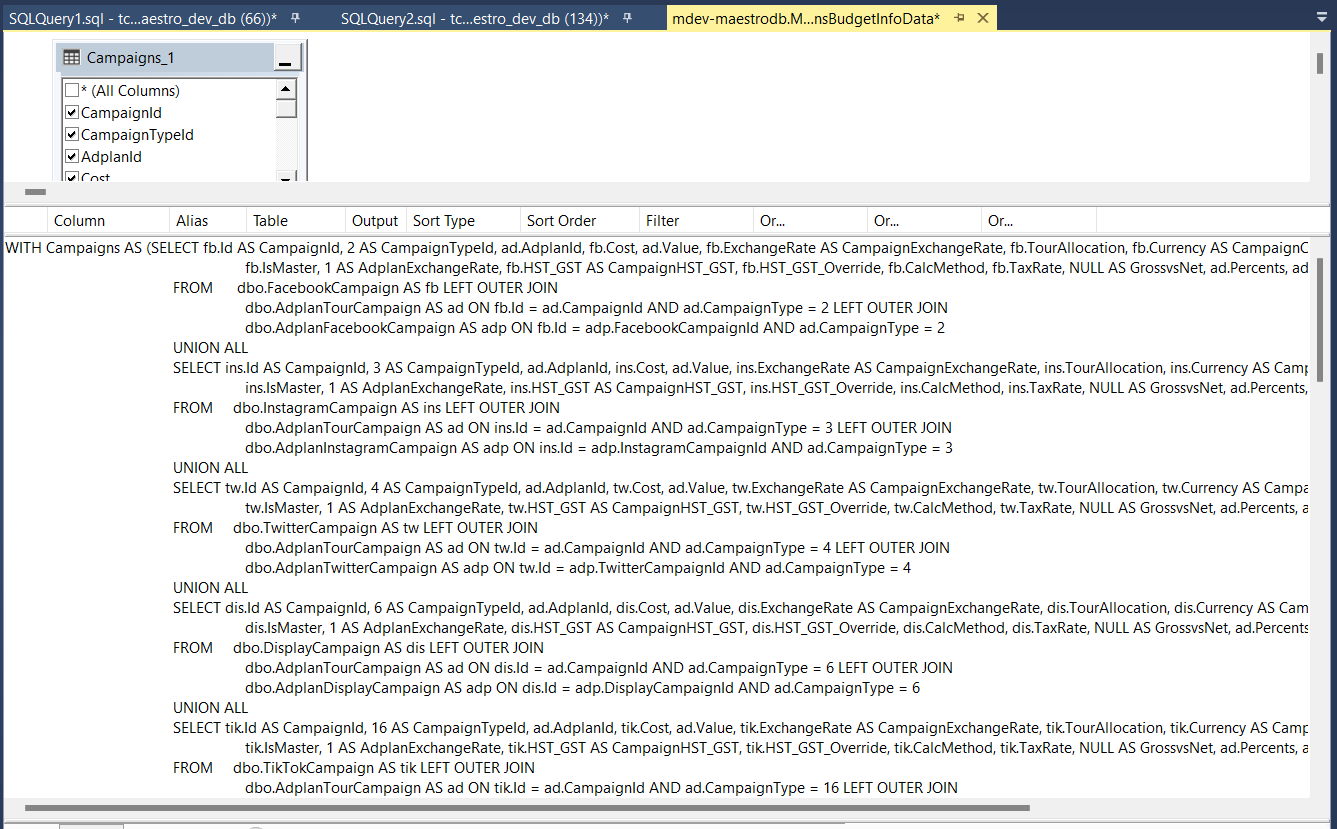
4.1 Ilustracija plana sa stepenom paralelizma 4

5. Primeri na realnoj bazi

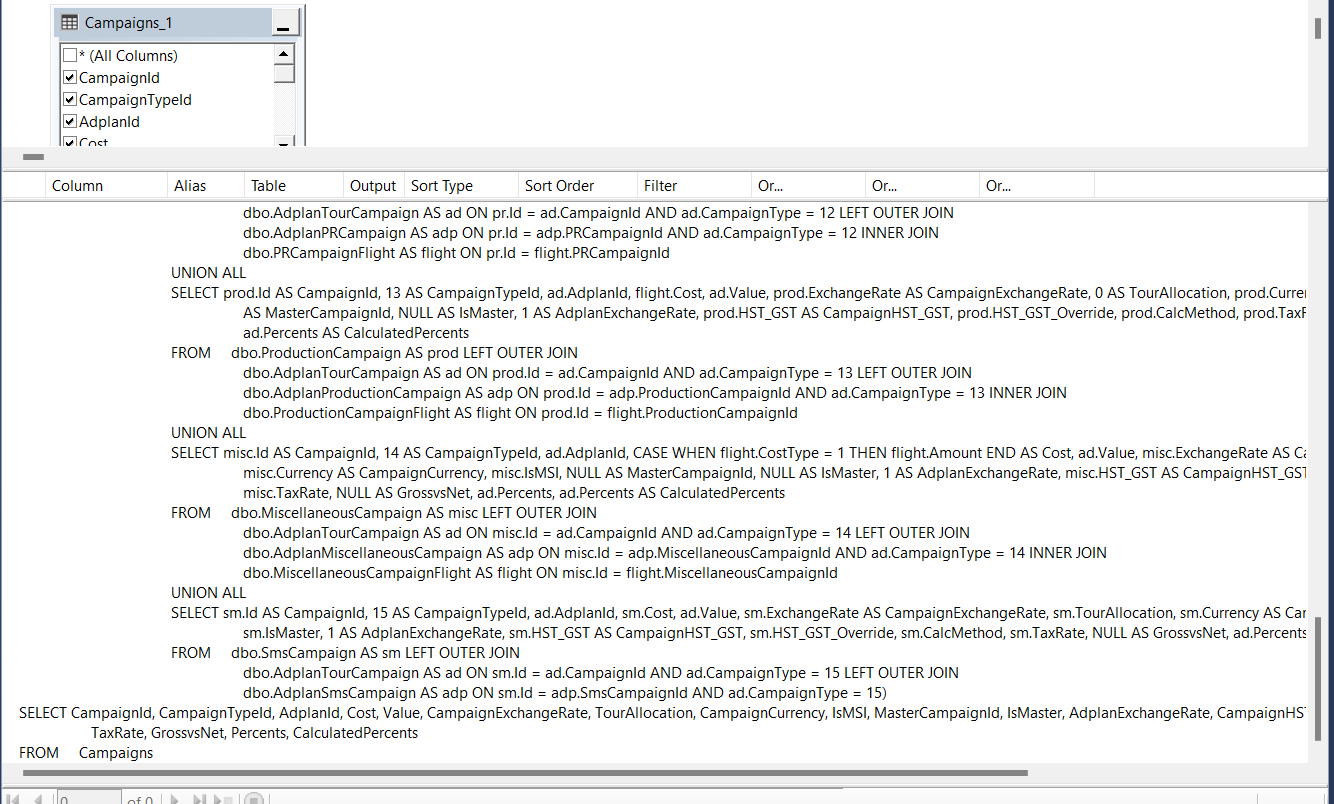
Primer kako izgleda Execution plan u MSSQL:



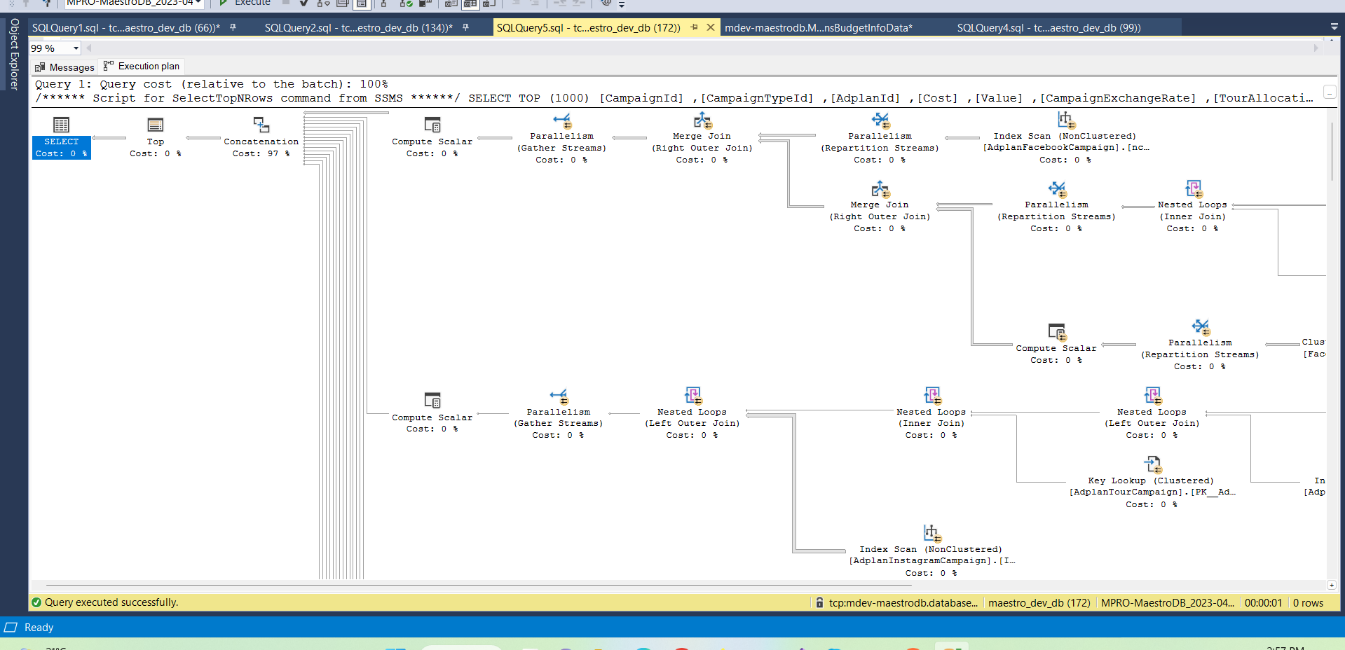
Primer pogleda i njegovog plana izvršenja:



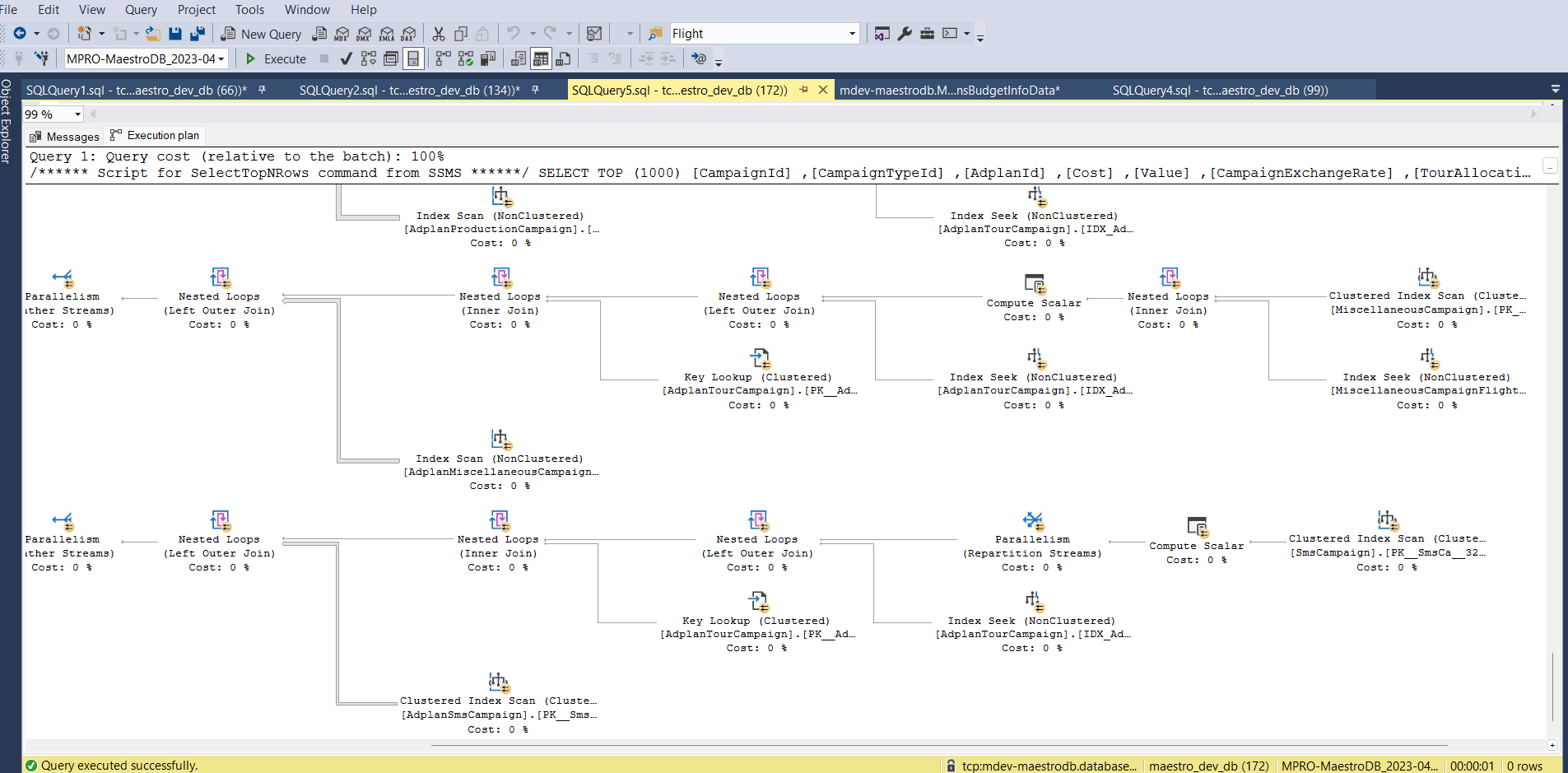
Nastavak View-a



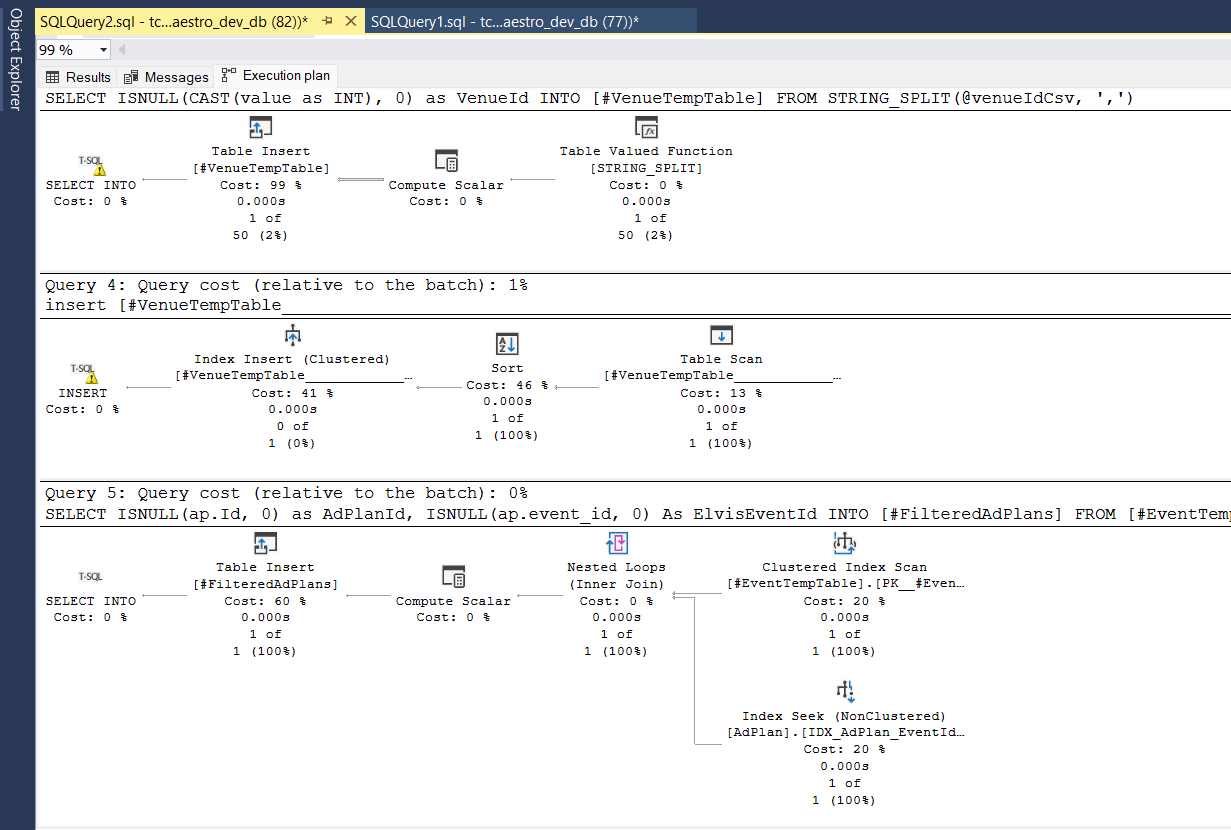
Execution plan prethodnog view-a u Execution Plan Preview:



Nastavak plana

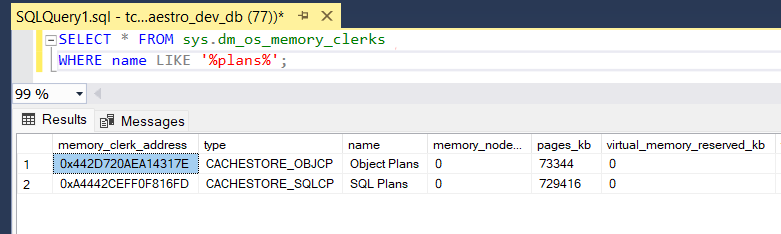


Primer kompleksnog plana izvršenja procedure



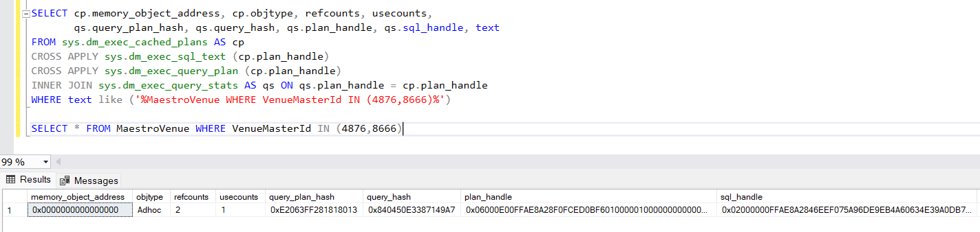
Primer izvršenja sledećeg upita koji prikazuje dva tipa keša planova:

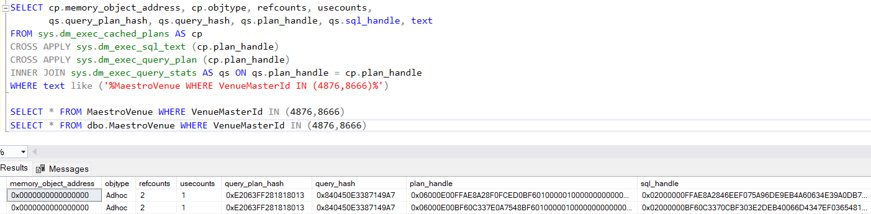
SELECT \* FROM sys.dm\_os\_memory\_clerks WHERE name LIKE '%plans%';

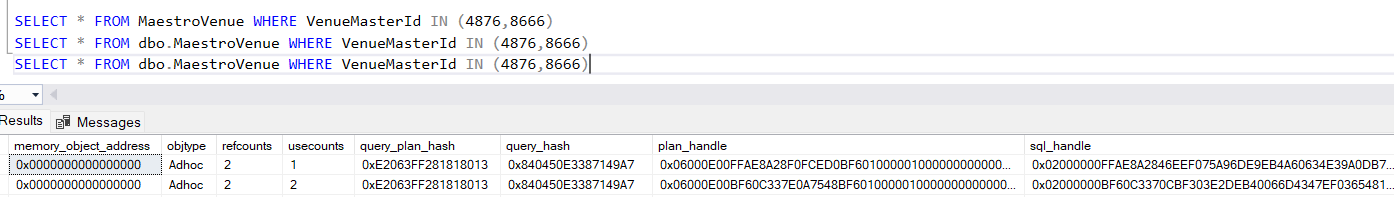


Keš planova izvršenja:  
  
SELECT cp.memory\_object\_address, cp.objtype, refcounts, usecounts, qs.query\_plan\_hash, qs.query\_hash, qs.plan\_handle, qs.sql\_handle FROM sys.dm\_exec\_cached\_plans AS cp CROSS APPLY sys.dm\_exec\_sql\_text (cp.plan\_handle) CROSS APPLY sys.dm\_exec\_query\_plan (cp.plan\_handle) INNER JOIN sys.dm\_exec\_query\_stats AS qs ON qs.plan\_handle = cp.plan\_handle

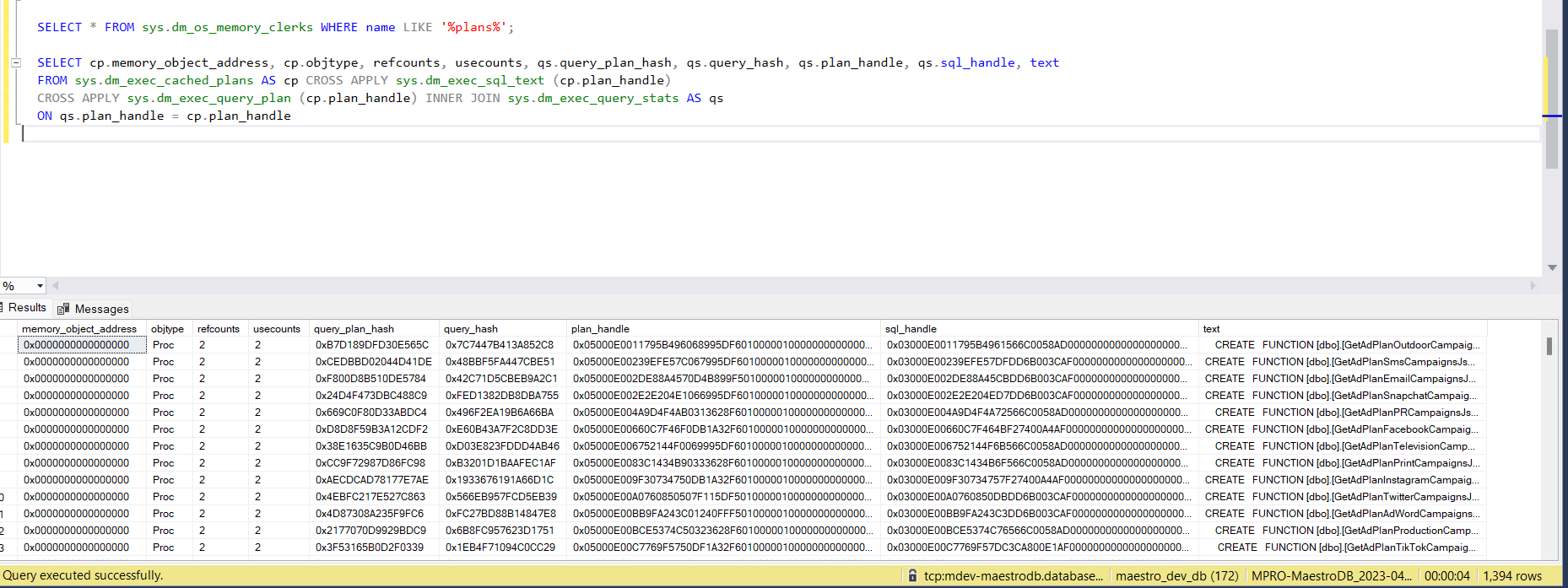
Izvršenje upita sa različitim načinima referenciranja tabela (pun naziv i parcijalni):

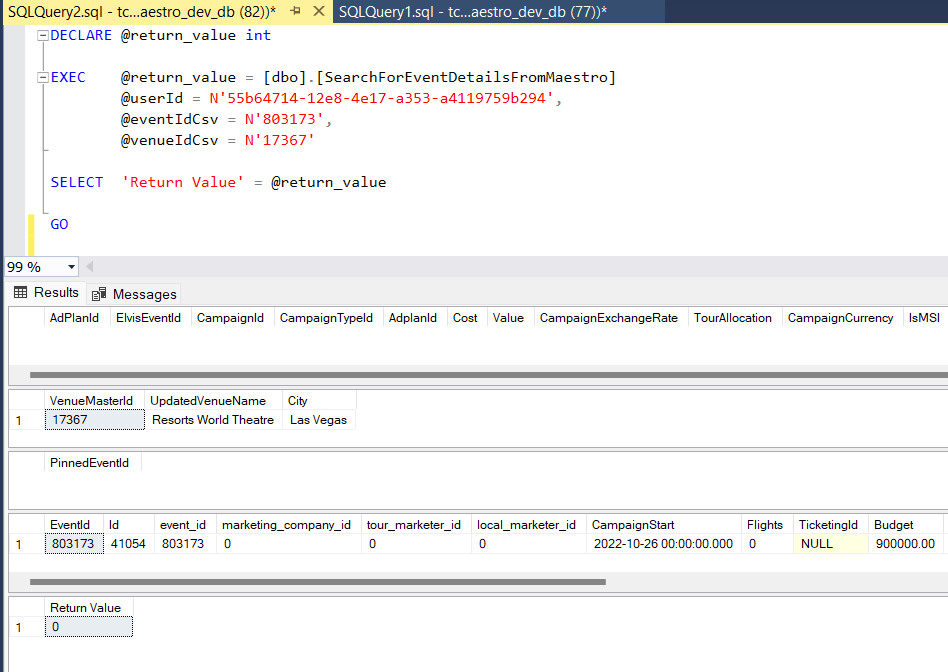


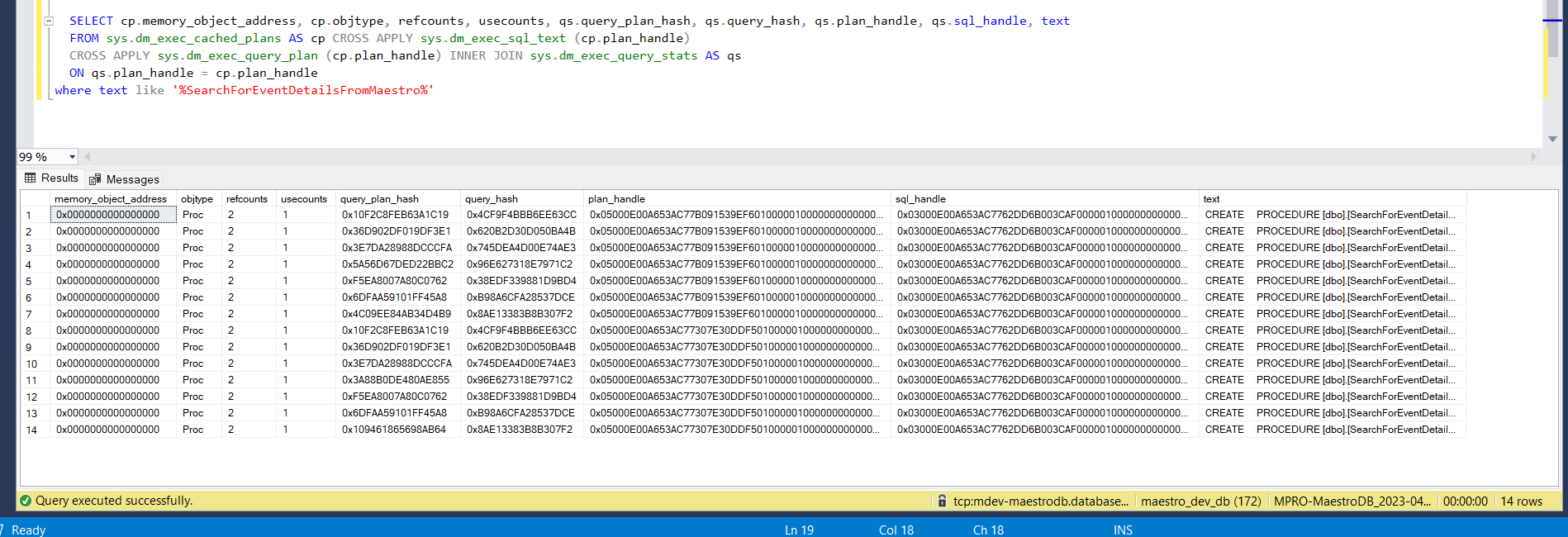


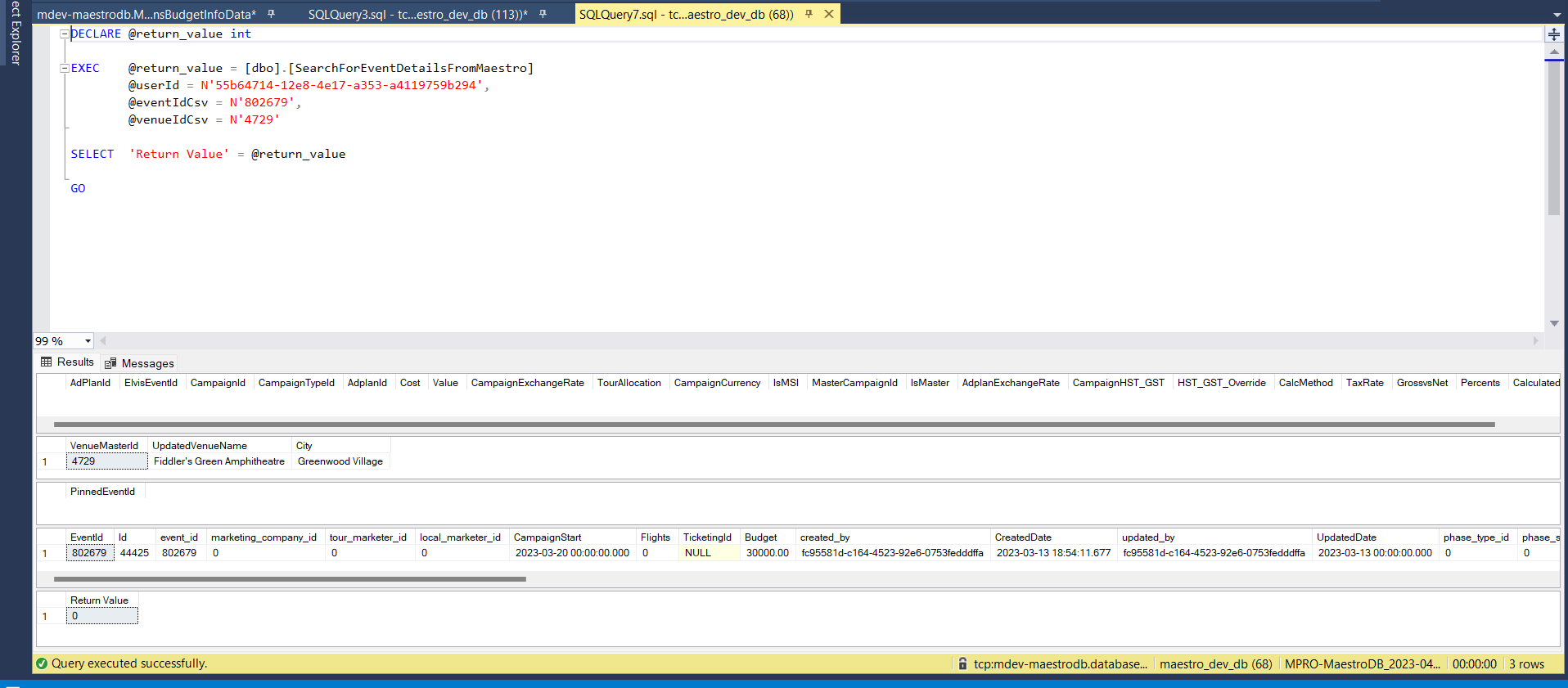


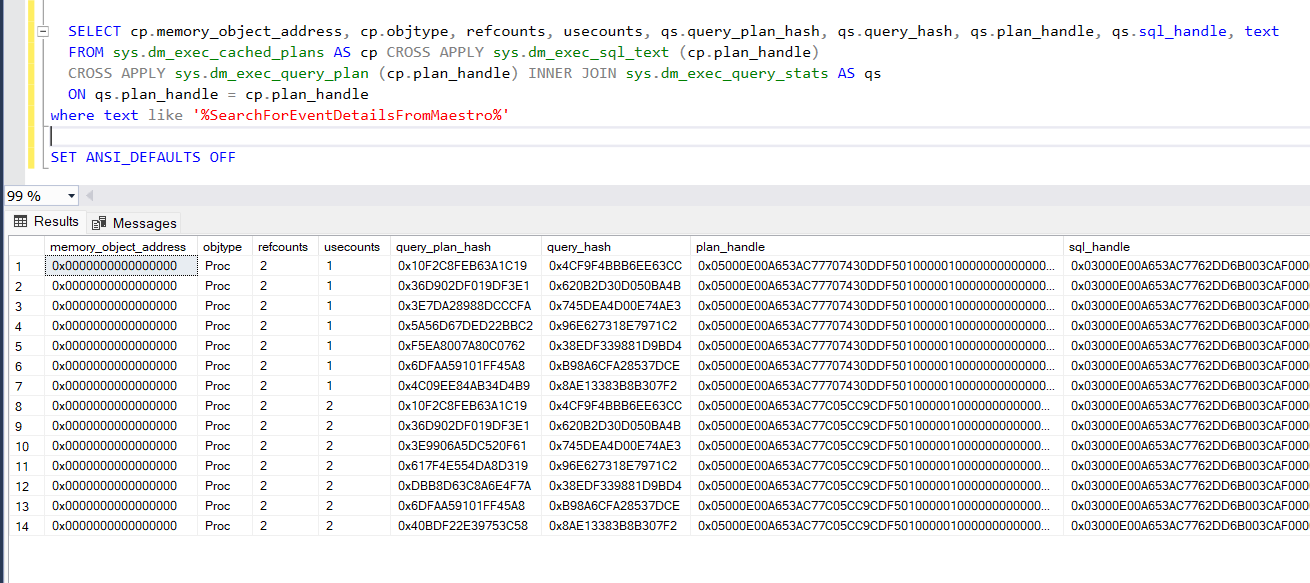
Prvi plan se izvršio samo jednom jer se plan novog upita zbog drugačijeg referenciranja nije poklopio sa već postojećim planom u kešu. Drugi plan se izvršio dva puta, jer je plan trećeg upita identičan planu drugog, i pronađeno je poklapanje u kešu.

Veza izmedju plan i plan handle:

Izvršenje procedure sa jednim setom parametara  


Nakon izvršenja sa tim setom: (usecounts svugde 1, zoom na 210% za bolji pregled)  


Nakon izvršenja sa novim parametrima:  


  
usecounts se povećalo na 2, što znači da se iskoristio isti plan.

6. Zaključak

SQL Server puno pažnje posvećuje obradi upita. Od skraćivanja konstanti, režima izvršenja, procene cena izraza i planova, preko keširanja planova, njihovog ponovnog korišćenja, brisanja iz keša. Obrada upita je sveobuhvatna i uključuje i paralelizaciju upita, kao i obradu distribuiranih upita i pogleda. Algoritmi za brisanje planova iz keša, kao i za paralelizaciju su odrađeni tako da pokrivaju sve moguće scenarije i da uvek donesu najoptimalnije moguće odluke, bilo da je u pitanju stepen paralelizacije ili koji plan sledeći obrisati iz keša prilikom nedostatka memorije.

Relacioni endžin koji je zadužen za obradu upita je komponenta od velikog značaja za celokupan rad i obradu upita. Obrada upita kao proces omogućava optimizatoru da odabere najbolji plan izvršenja, kao i da maksimalno optimizuje izvršenje upita, procedura, funkcija i svih ostalih struktura. Kao takav, proces obrade upita je nezamenjiva komponenta u celokupnom funkcionisanju SQL Servera.

Literatura

<https://learn.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide?view=sql-server-ver16>

<https://learn.microsoft.com/en-us/sql/relational-databases/showplan-logical-and-physical-operators-reference?view=sql-server-ver16>

<https://learn.microsoft.com/en-us/sql/t-sql/language-elements/operator-precedence-transact-sql?view=sql-server-ver16>

<https://learn.microsoft.com/en-us/sql/relational-databases/performance/execution-plans?view=sql-server-ver16>