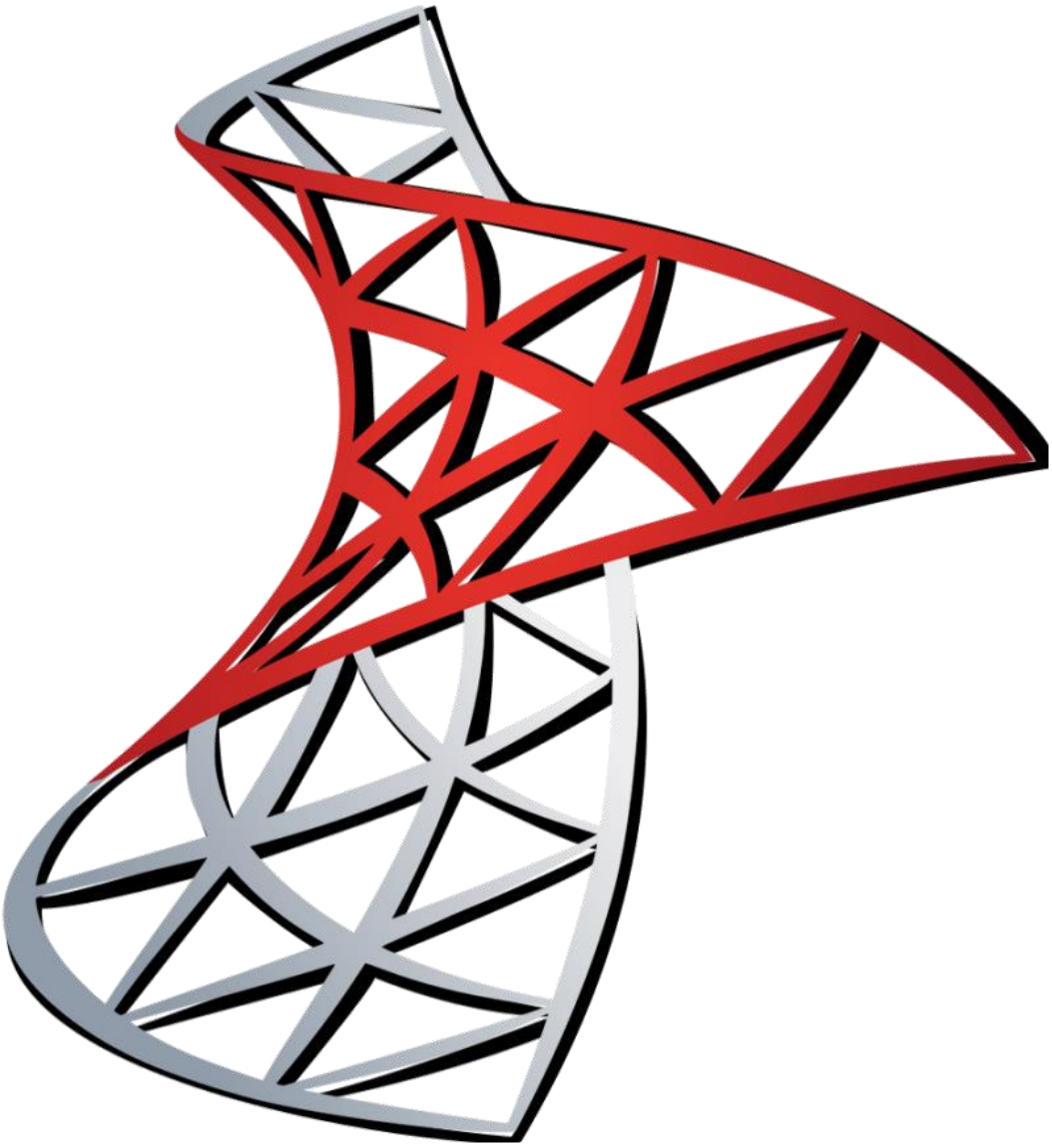


MS SQL Server

OBRADA UPITA

Đorđe Nikolić | Sistemi za upravljanje bazama podataka | april 2022.

Sadržaj

Uvod

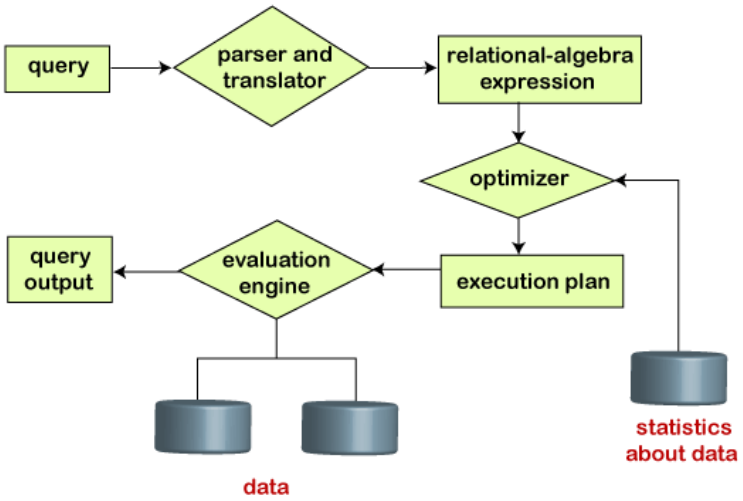
U ovom radu će biti analiziran i predstavljen proces obrade upita kod MS SQL Server sistema za upravljanje bazama podataka. Biće pomenute teorijske osnove ove aktivnosti, a zatim će biti obrađeni detalji implementacije u konkretnom sistemu MS SQL. Fokus će biti na novim tehnikama obrade i optimizacije upita koje su uvedene u verzijama sistema iz 2017. i 2019. godine.

Pojam obrade upita

Obrada upita predstavlja aktivnost prevođenja upita, pisanog u nekom upitnom jeziku, u informacije pomoću kojih sistem za upravljanje baze podataka može pronaći i vratiti odgovaraće podatke. Ovaj proces se uglavnom sastoji iz nekoliko koraka:

1. Prevođenje
2. Evaluacija
3. Optimizacija
4. Izvršenje

Detalji samih koraka zavise od datog sistema, ali u suštini, prilikom prevođenja se proverava sintaksa i ispravnost samog upita, a zatim kreira stablo iz delova upita, poznato kao „parse tree“, pomoću teorije relacione algebre. Nakon toga, se svaki deo upita koji se odnosi na neki resurs, povezuje se postojećim objektima u bazi podataka (tabele, kolone, procedure, itd.) tokom koraka evaluacije. Zatim, sistem predlaže nekoliko različitih metoda, tj. planova, izvršenja, i bira se optimalan plan izvršenja na osnovu podataka koji su dostupni sistemu i optimizacijskom procesoru. Ove odluke se uglavnom donese na osnovu statistike. Izabrani plan se zatim izvršava. (1) (2)

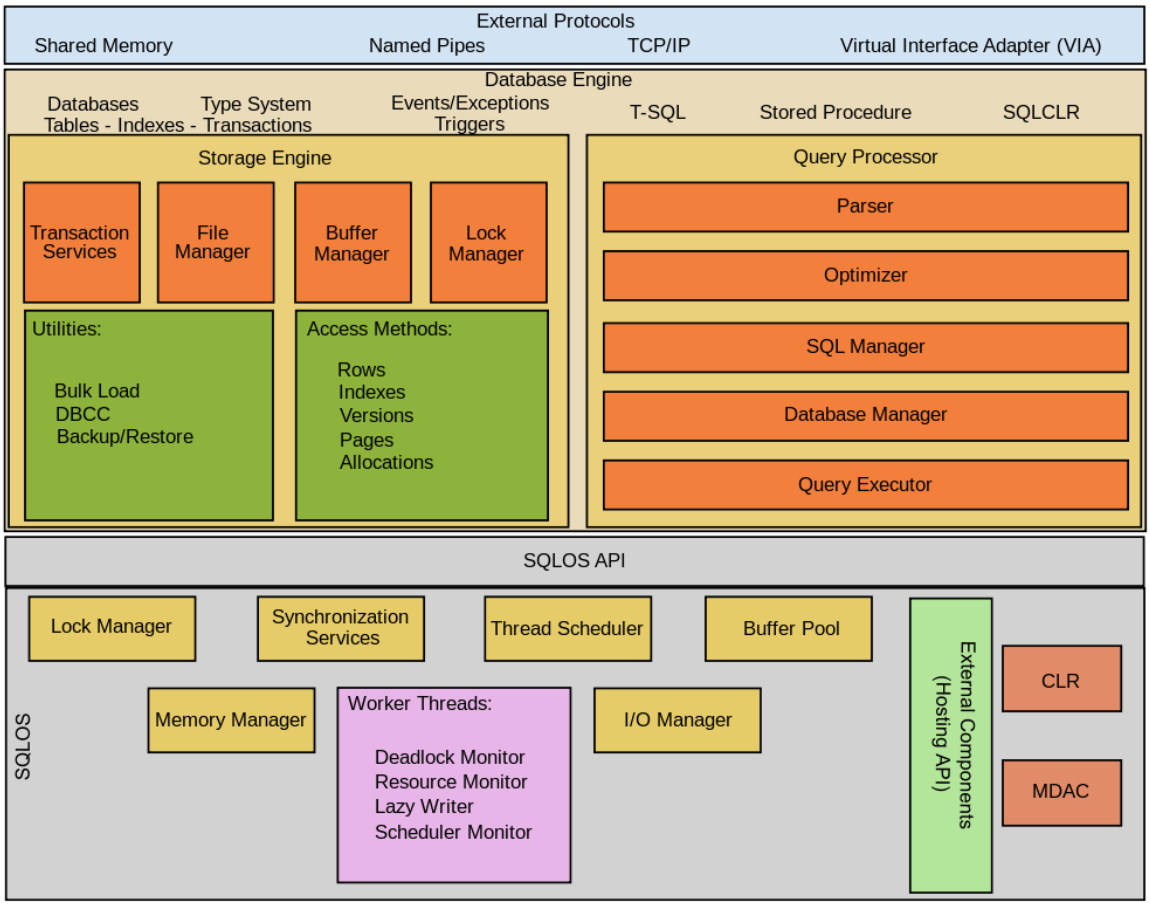


Slika 1. Prikaz opštih koraka kod obrade upita

MS SQL server

Microsoft SQL Server (skraćeno MS SQL) je relacioni sistem za upravljanje bazama podataka koji je razvila kompanije Microsoft. Sistem koristi upitni jezik pod imenom Transact SQL (skraćeno T-SQL), koji predstavlja Microsoft-ovu implementaciju SQL-a, standardnog upitnog jezika koji se koristi za relacione baze podataka. Trenutna stabilna verzija je SQL Server 2019.

GENERALNA STRUKTURA



Slika 2. Opšti prikaz arhitekture MS SQL Server-a

MS SQL se sastoji iz dva glavna dela, a to su motor baze podataka (engl. *database engine*), i SQL OS. Motor baze podataka se sastoji iz dela pod imenom motor skladištenja (engl. *storage engine*), koji upravlja samim datotekama baze podataka, stranicama, indeksima, sačuvanim procedurama, pogledima itd., i procesorom upita, koji obrađuje upit i određuje najbolji način da se on izvrši, a zatim potražuje podatke preko motora skladištenja, i obrađuje rezultate. SQLOS (engl. *SQL Server Operating System*) je sloj koji se nalazi između MS SQL-a i operativnog sistema i pruža razne servise koji spadaju pod domen operativnih sistema, kao što su upravljanje memorijom, detekcija zastoja,

upravljanje greškama, sinhronizacija itd. Shodno temi ovog rada, fokus će biti na procesoru upita.

PROCESOR UPITA

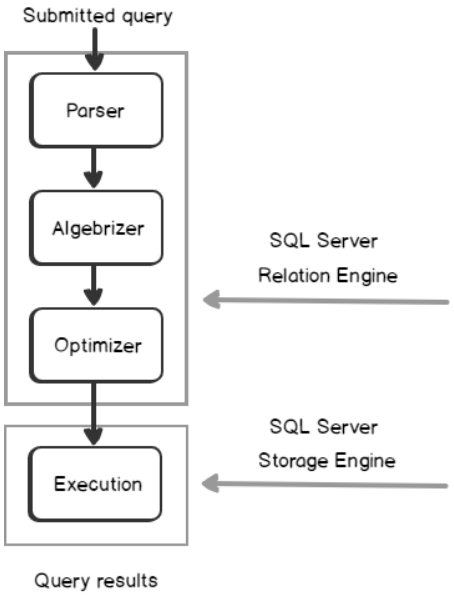
Procesor upita je najkompleksnija komponenta MS SQL Server-a, i njegove funkcionalnosti su se daleko proširile kroz verzije sistema, zaključno sa trenutno stabilnom verzijom MS SQL Server 2019. Kako bi se dobro upoznali sa njegovom funkcijom, krenućemo od njegovih najosnovnijih odgovornosti i polako preći na neke od naprednijih metoda koje su dodate u kasnijim verzijama.

Procesor upita, takođe poznat kao relacioni motor (engl. *relational engine*) je komponenta MS SQL Server-a koja preuzima upite pisane u T-SQL jeziku i prerađuje ih tako da na kraju dobije informacije pomoću koji može poslati zahteve motoru skladišta kako bi traženi rezultati mogli biti pronađeni. (3)

U osnovi, sistem prolazi kroz četiri koraka od izvornog upita do rezultate, a to su:

1. Raščlanjivanje (engl. *parsing*)
2. Algebrisanje (engl. *algebrizing*)
3. Optimizovanje (engl. *optimizing*)
4. Izvršenje (engl. *executing*)

Prva tri koraka se u potpunosti izvršavaju od strane relacionog motora, dok je izlaz trećeg koraka zapravo optimizovani plan izvršenja koji se zakazuje, i tokom kog se prave pozivi ka motoru skladišta kako bi se preuzeli podaci koji postaju rezultati originalnog upita. (4)



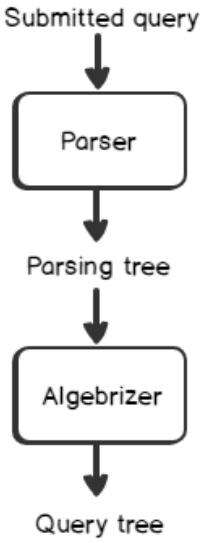
Slika 3. Grafički prikaz osnovnih koraka obrade upita

Raščlanjivanje

U okviru prvog koraka, raščlanjivanja, uzima se originalni upit i razbija na individualne delove, a zatim proverava njihova sintaktička ispravnost. Ukoliko procesor upita nađe neku grešku u upitu, on te informacije vraća nazad korisniku kroz sloj protokola. Ukoliko je upit ispravno napisan i nijedna greška nije nađena, od upita se generiše stablo članova (engl. *parse tree*), koje je zapravo interna reprezentacija samog upita koja sadrži sve korake, bolje poznate kao preemptivne operacije, koje se prate kako bi se upit izvršio.

Algebrisanje

Nakon kreiranja prethodno pomenutog stabla članova, ono se prosleđuje na dalju obradu, tj. algebrisanje. U ovoj fazi se izvršavaju nekoliko operacija, od kojih je najbitnije povezivanje imena sa već postojećim objektima u bazi podataka. Tokom ove operacije se utvrđuje da sve referencirane tabele i kolone postoje u sistemskom katalogu i da su vidljive korisniku (sa sigurnosnog gledišta). Takođe se povlače svi meta-podaci vezani za date objekte. Zatim se određuju konačni tipovi podataka za sve čvorove u stablu, kao i svaka implicitna konverzija podataka. Onda se verifikuju komande agregacije i grupisanja, kao i tačno mesto na kojima bi one trebale da se primene. Kao rezultat ovih koraka, dobija se stablo logičkih operatora (stablo upita) koji su neophodni kako bi se izvršio zadatak celokupnog upita.

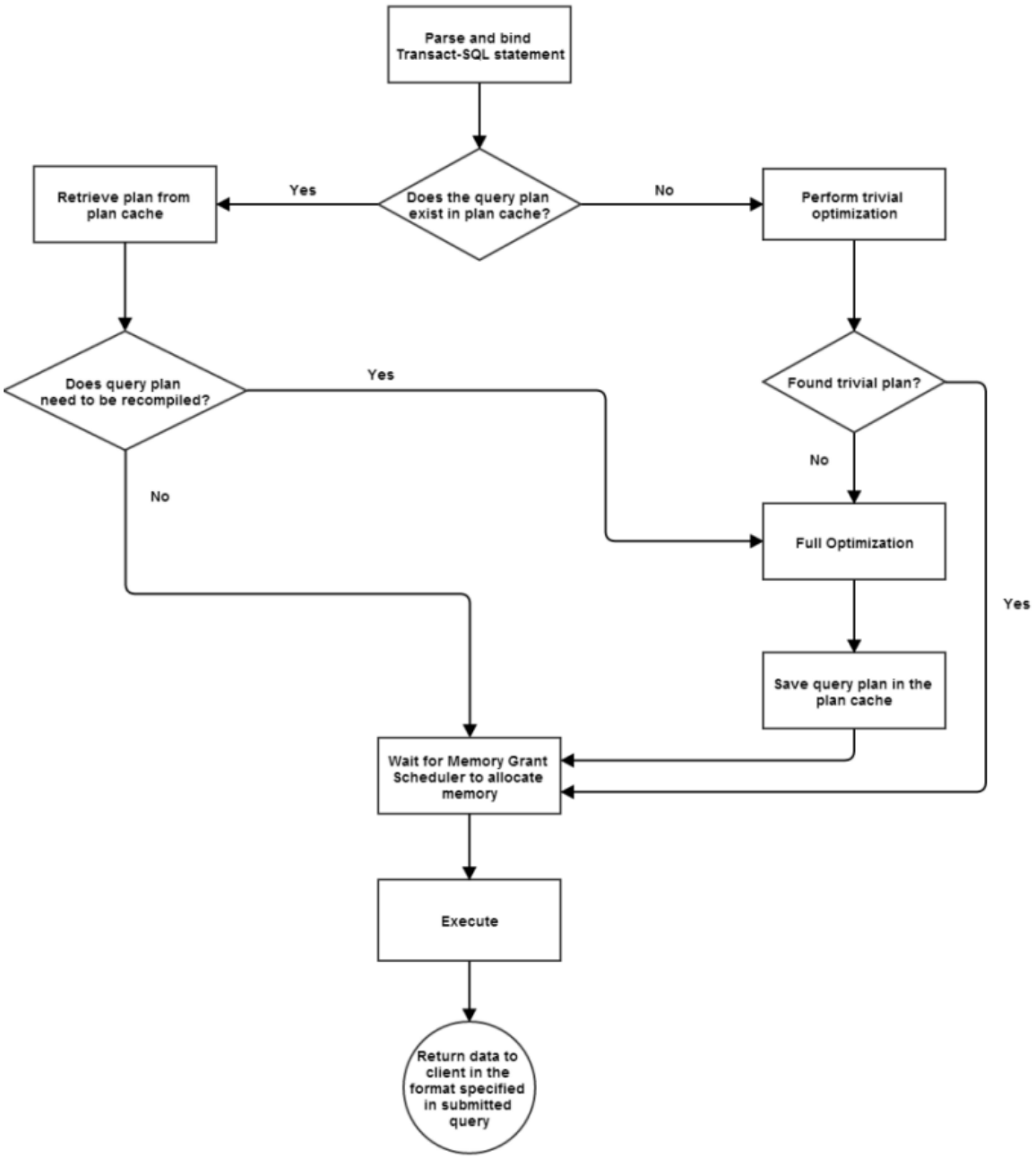


Slika 4. Grafički prikaz prva dva koraka obrade upita kao i njihovih rezultata

Optimizacija upita

Optimizacija upita je najkompleksnija faza obrade upita, čiji je cilj je dobiti optimizovani plan izvršenja koraka iz stabla upita. Optimizacija je u tome što se bira plan koji će, u teoriji, potražiti najmanju količinu resursa sistema. Takozvani plan izvršenja

zasnovan na resursnim troškovima (engl. *cost-based execution plan*). Prvi korak ove faze podrazumeva pokušaj dobijanja trivijalnog plana izvršenja upita. Trivijalni plan je onaj koji ima poznatu, konstantnu procesorsku i ulazno/izlaznu cenu. Drugim rečima, trivijalni plan je onaj koji je jedini mogući za dati T-SQL upit. Primeri upita koji bi mogli imati trivijalni plan izvršenja su SELECT naredba nad tabelom bez indeksa, SELECT naredba nad tabelom sa jednim argumentom pretrage po unikatnom ključu itd.

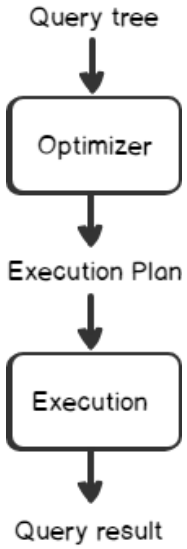


Slika 5. Grafički prikaz toka optimizacije upita

Dakle, kada prvi put do ove faze dođe neko stablo upita, optimizator prvo pokuša da odredi da li bi upit bio trivijalan. Kako bi to postigao, skupljaju se meta-podaci i istražuje se upit. Zahvaljujući ovoj trivijalnoj optimizaciji, sistem može izbeći veliku količinu posla koju je potrebno izvršiti prilikom traganja za optimalnim planom izvršenja na osnovu resursnih troškova. U slučaju da sistem nađe trivijalni plan, dalji rad nije potreban i može se odmah preći na izvršenje.

Ako trivijalni plan nije dostupan, sistem mora preuzeti sve dostupne statistike vezane za sve tabele, kolone i indekse koje su vezane za dati upit kako bi odredio najbolji plan izvršenja iz skladišta planova (engl. *plan cache*). U ovom stadijumu, sistem takođe primenjuje još neke sintaktičke transformacije nad stablom upita, kao što je preuređivanje redosleda operacija. Bitno je napomenuti da skladište planova predstavlja deo memorije koji se koristi za čuvanje prethodno kreiranih planova izvršenja. O ovome će biti više reči kasnije.

Inicijalno, sistem pokušava da nađe prost plan izvršenja u skladištu. Prosti planovi su često oni koji koriste ugnježđeno spajanje tabela (engl. *nested loop join*), i jedan indeks po tabeli. Ako takav plan za prosleđeni upit ne postoji, onda se traži kompleksan plan izvršenja, tako što se analiziraju više indeksa nad tabelama kako bi se našao dovoljno dobar plan. U situaciji u kojoj se spaja tabela bez adekvatnog indeksa za dato spajanje, optimizator upita pokušava da kreira planove uz pomoć „heš“ spojeva (engl. *hash join*).



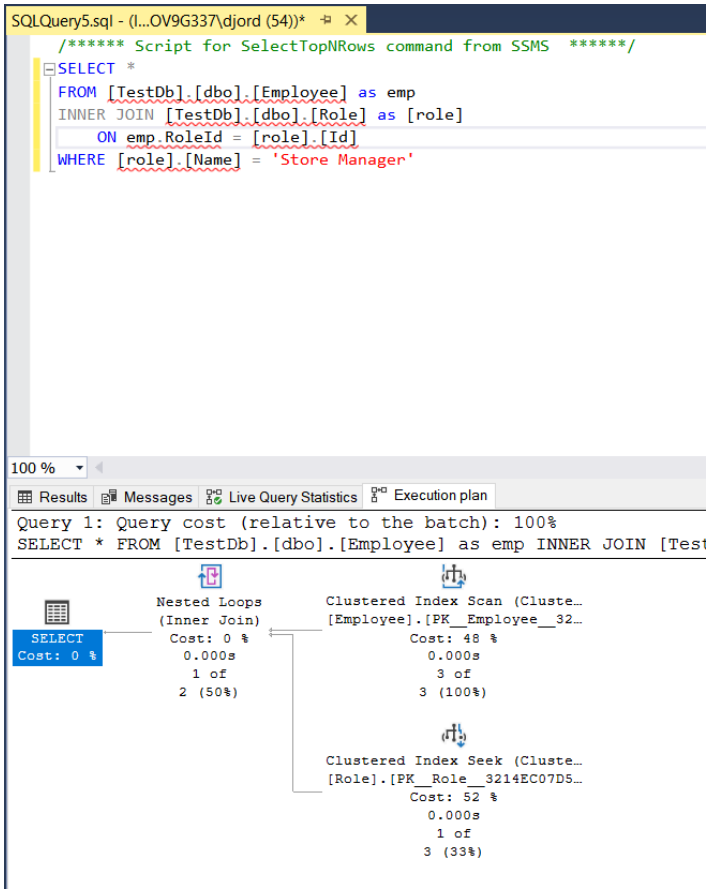
Slika 6. Grafički prikaz poslednja dva koraka obrade i izvršenja upita

U slučaju da sistem ne nađe adekvatan plan izvršenja u skladištu planova, pokreće se kompletan i temeljan proces optimizacije plana po resursnim troškovima. U ovoj fazi, MS SQL Server koristi logičko stablo upita, koje je dobio iz prethodnog koraka, kako bi generisao sve moguće načine izvršenja upita. U slučaju da računar na kome se upit izvršava poseduje više od jednog procesora, i ako su postavljena podešavanja „prag

troškova paralelizma“ (engl. *cost threshold for parallelism*) i „maksimalni stepen paralelizma“ (engl. *max degree of parallelism*), sistem će generisati plan paralelnog izvršenja. Bitno je pomenuti da sistem bira neparalelni plan izvršenja samo u slučaju da je trošak „najjeftinijeg“ paralelnog plana veći od troška „najjeftinijeg“ neparalelnog plana izvršenja.

Sistem bira plan sa najmanjim predviđenim resursnim troškovima na račun procesora računara i ulazno/izlaznih aparata i prosleđuje ga dalje na izvršavanje. Obzirom na način biranja plana, moguće je da u skladištu planova postoji „bolji“ ili brži plan, ali će optimizator upita uvek izabrati plan sa najmanjim troškovima.

Izvršenje i skladištenje plana upita



Slika 7. Primer prostog upita i primer plana izvršenja u donjoj polovini slike

Nakon što je plan izvršenja kreiran, ili pronađen u skladištu upita, sistem koristi izabrani plan i zajedno sa motorom skladištenja izvršava upit i vraća rezultate korisniku kroz sloj protokola u formatu specificiranom u T-SQL definiciji upita.

U sledećim slučajevima, optimizator upita može promeniti procenjeni plan izvršenja (ove funkcije nisu bile dostupne u starijim verzijama MS SQL Servera, a detaljnijeg razgovora o njima će biti u narednom poglavlju):

- Ako su statistike tabela ili kolona zastarele.
- Ako neparalelni plan izvršenja prekorači prag za izvršenje paralelnog plana.
- Ako se podaci u relevantnim tabelama značajno promene.

Osim toga, ako između nekoliko izvršenja dođe do značajne promene podataka u tabelama, indeksa ili statistike, plan izvršenja može biti rekompajliran. Tada se novi plan čuva u skladištu. U jednom trenutku, sistem može čuvati samo dve instance plana izvršenja u skladištu: paralelni plan izvršenja, i neparalelni plan izvršenja.

Zastarevanje planova izvršenja

Svaki plan sačuvan u skladištu planova se čuva zajedno sa starošću i faktorom troškova. Ovaj faktor predstavlja totalne troškove prilikom kompajliranja upita. Svaki put kad se ovaj plan referencira, faktor se povećava za 1. Sistem ne smanjuje ovaj faktor sve dok veličina skladišta planova ne postane 50% celog bafera. Kada se ovo desi, sledeći put kada se pristupi skladištu, sistem smanji faktor svakog skladištenog plana za 1.

MS SQL Server periodično čisti skladište planova. Ovo se dešava kada:

- Baferu servera treba više memorije za neki drugi objekat
- Faktor svakog skladištenog plana izvršenja postane 0
- Plan izvršenja ne referencira ni jedna konekcija ka bazi

ADAPTIVNO PROCESIRANJE UPITA

Pre nego što je izašla MS SQL Server 2017 verzija (5), obrada i optimizacija upita je bila relativno prosta i išla je tokom koji je opisan u prethodnim poglavljima. Desila bi se analiza upita, kreiran je plan izvršenja, i onda je plan izvršen. Samim tim, ako izabrani plan ne bude odgovarajući, procesor upita ne bi mogao da promeni plan tokom izvršenja samog upita, ili čak nakon izvršenja. Postoji više razloga zašto izabrani upit može biti neadekvatan, a ovo su neki od njih:

- Manjak odgovarajućih indeksa
- Zastarele statistike
- Neodgovarajući keširani planovi
- Loše napisani upiti

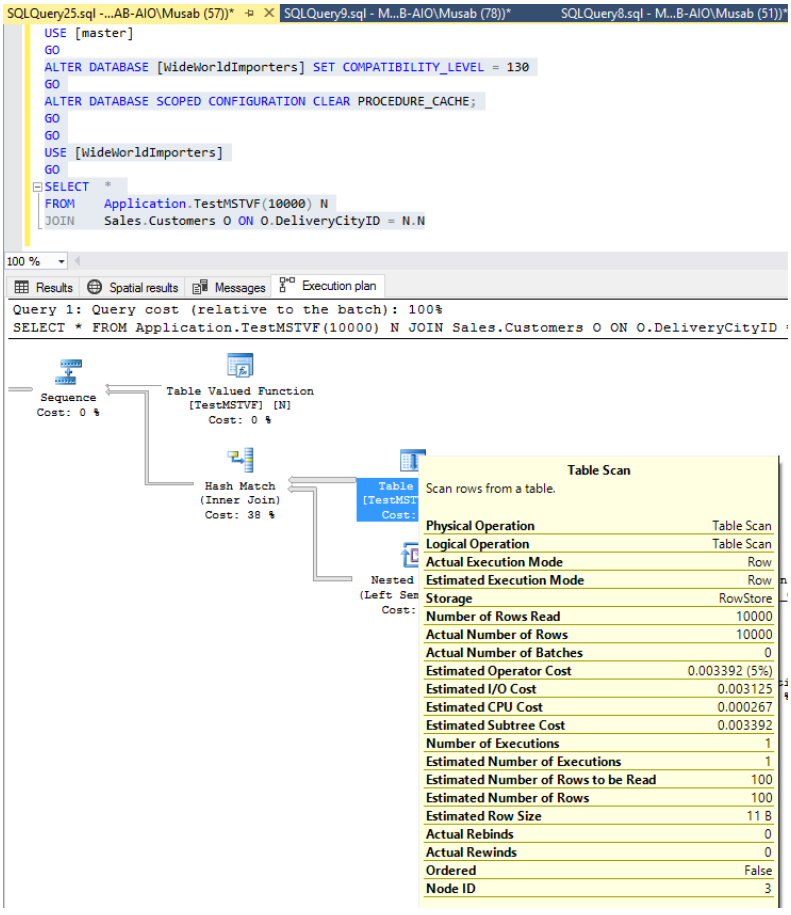
Veliki faktor u biranju odgovarajućih planova izvršenja su SQL Server statistike. Svaki put kada pošaljemo upit, procesor upita mora da proceni podatke nad kojima će se taj upit izvršavati. Kako bi to uradio, gleda se u postojeće statistike. Vrednosti koje se nalaze u sačuvanim statistikama su od ogromne važnosti za donošenje odluka o planovima izvršenja. Dakle, bitno je da se sistem postara da su statistike podataka ažurne.

U verziji ovog sistema koja je izašla 2017. godine, ubačena je funkcionalnost pod imenom „adaptivno procesiranje upita“ (engl. *adaptive query processing*). Ova funkcionalnost je premostila fazu optimizacije plana izvršenja, i fazu samog izvršenja datog plana. Sa ovom funkcionalnošću, sistem može vršiti optimizaciju dok se izvršava izabrani plan, a čak i nakon izvršenja plana kako bi poboljšao kasnija izvršenja. Postoje tri dela adaptivnog procesiranja upita:

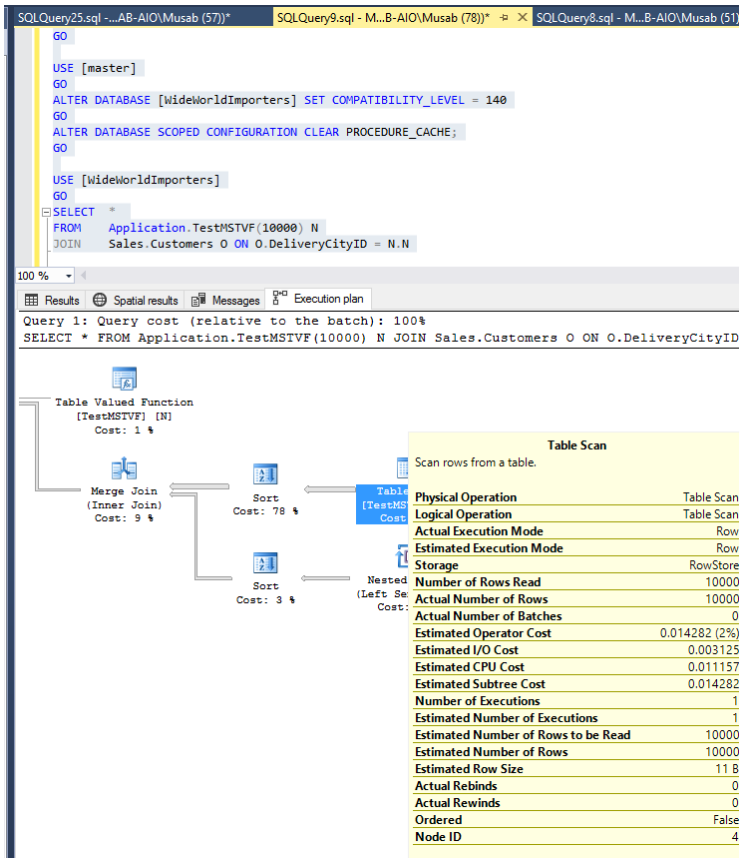
- Isprepletena izvršenja (engl. *interleaved executions*)
- Ažurno dodeljivanje memorije kod serijskog moda (engl. *batch mode memory grant feedback*)
- Adaptivno spajanje tabela kod serijskog moda (engl. *batch mode adaptive joins*)

Isprepletena izvršenja

Kod više-izraznih funkcija koje vraćaju tabele (engl. ***M**ulti **S**tatement **T**able **V**alued **F**unctions* – **MSTVF**), procene u statistici su postavljena na fiksnu vrednost. Sa funkcionalnošću isprepletenih izvršenja, optimizator upita može tokom izvršenja upita videti da je procena loša, a zatim promeniti plan izvršenja tako što prvo izvrši jedan deo plana, ponovo dizajnirati ostatak plana sa novim informacijama, i onda ga izvršiti.



Slika 8. Primer poziva proste MSTVF u staroj verziji sistema. Procena broja redova je 100, a pravi broj redova je 1000.



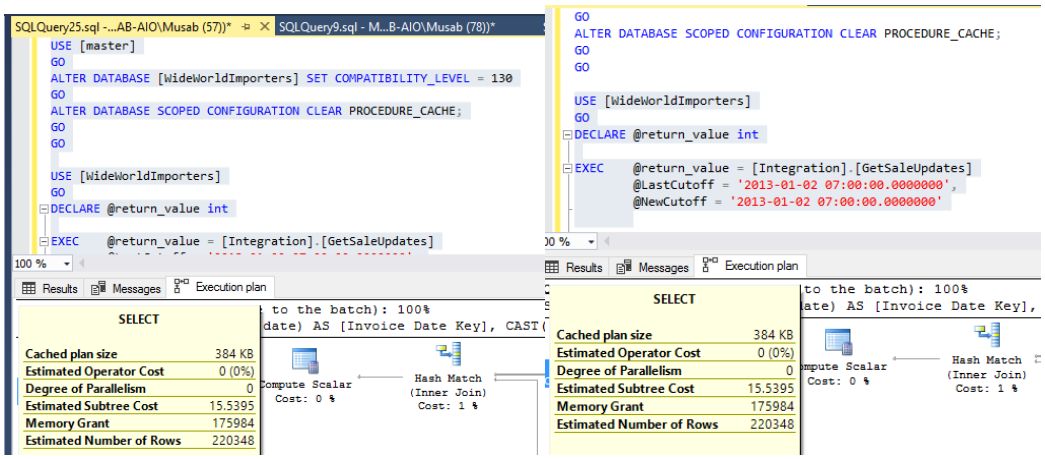
Slika 9. Primer poziva proste MSTVF u novoj verziji sistema. Procena broja redova je adekvatna.

Ažurno dodeljivanje memorije kod serijskog moda

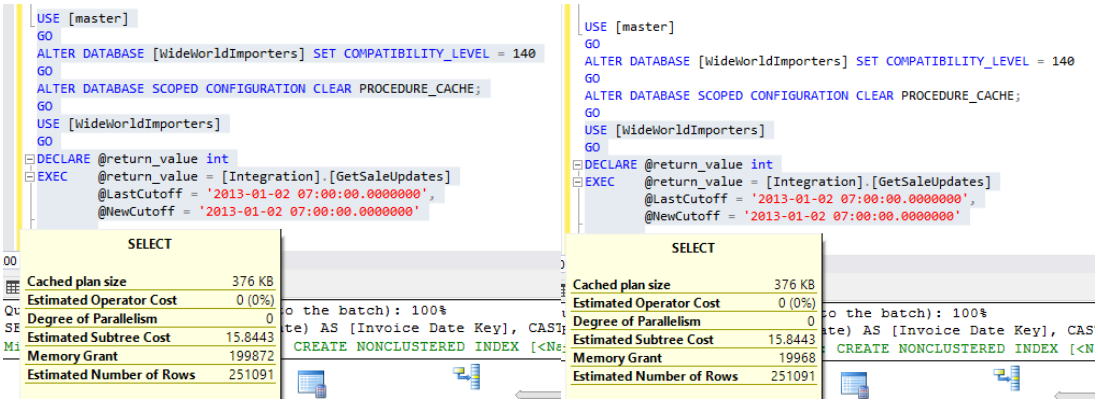
Dodeljivanje memorije je još jedan vrlo bitan deo procesa kreiranja plana izvršenja. Pre verzije sistema iz 2017. godine, sistem je keširao plan za neku skladištenu proceduru ili upit i koristio ih opet kasnije. Na primer, ako imamo skladištenu proceduru koja je prvi put izvršena sa parametrom tako da se obradi mali broj podataka, izabrani plan izvršenja koji se kešira može kasnije imati slabije performanse ako uneti parametar rezultuje većom količinom obrađenih podataka. U ovom slučaju može doći do male količine dodeljene memorije planu izvršenja, čak iako će mu trebati više memorije.

Ova funkcionalnost omogućava procesoru upita da vidi da li je dodeljena memorija adekvatna, i ako nije, onda će keširani plan biti promenjen, i količina dodeljene memorije biti povećana (ili smanjena) tako da sva sledeća izvršenja date procedure imaju dovoljno memorije.

Možemo videti ovu funkcionalnost u akciji tako što ćemo pokrenuti skladištenu proceduru za koju su zabeležene loše statistike, i prouzrokovati da procenjena dodeljena memorija bude premala.



Slika 10. Primer prvog i drugog poziva sačuvane procedure sa lošim statistikama, kod stare verzije sistema. Dodeljena memorija ostaje nepromenjena.



Slika 11. Primer prvog i drugog poziva sačuvane procedure sa lošim statistikama, kod nove verzije sistema. Dodeljena memorije je promenjena kod drugog poziva.

Adaptivno spajanje tabela kod serijskog moda

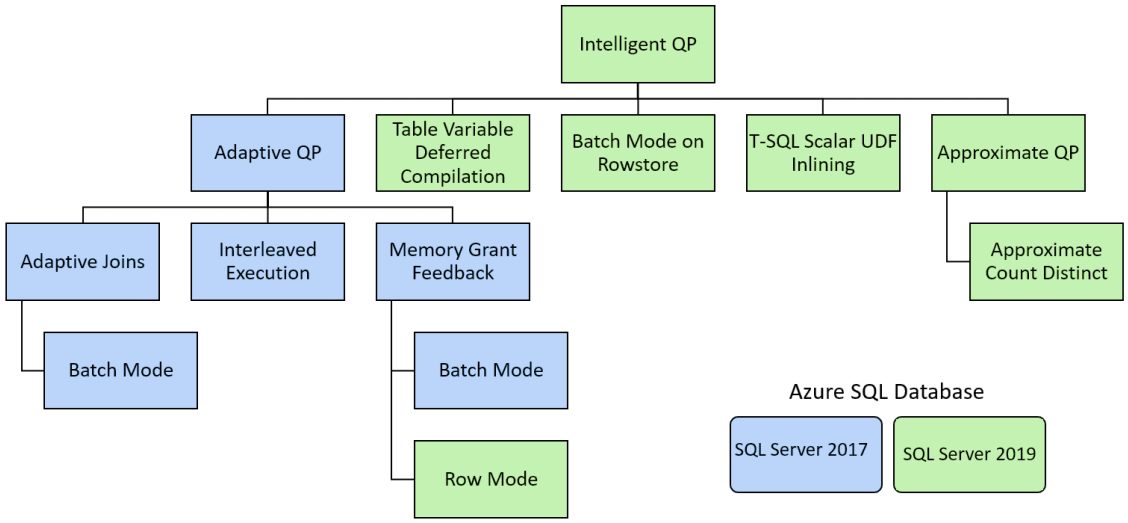
Ova funkcionalnost ima relativno prostu namenu. Motor izvršenja upita čeka da napravi izbor između ugnježdenog spoja, i „heš“ spoja, sve dok se ne izvrši prvi spoj u planu. Nakon što je taj prvi spoj izvršen, na osnovu broja redova koji su povučeni, sistem bira između ta dva načina spajanja tabela.

Postoji određeni parametar koji određuje prag odluke tipa spoja na osnovu broja redova. Poenta je da ako je broj redova mali, onda se koristi ugnježdeno spajanje tabela, a ako je broj reda veliki, koristi se „heš“ spoj.

INTELIGENTNO PROCESIRANJE UPITA

Sa novom verzijom MS SQL Servera koja je izašla 2019. godine, funkcionalnosti za premošćavanje faze obrade i optimizacije upita, i faze izvršenje plana, su proširene (6). Grupa tih funkcionalnosti je preimenovana u „Inteligentno procesiranje upita“ (engl. *intelligent query processing*) iz „Adaptivno procesiranje upita“. Većina funkcionalnosti ove grupe su od tada takođe dostupne i kroz Azure SQL bazu podataka u oblaku. Nove funkcionalnosti su:

- Ažurno dodeljivanje memorije kod moda redova (engl. *row mode memory grant feedback*)
- Kasno kompajliranje tabelarnih varijabli (engl. *table variable deferred compilation*)
- Serijsko izvršenje kod skladišta redova (engl. *batch mode execution on rowstore*)
- Aproksimiranje brojanja unikata (engl. *approximate count distinct*)
- Ugrađivanje skalarnih korisničkih funkcija (engl. *scalar **U**ser **D**efined **F**unction inlining*)



Slika 12. Grafički prikaz svih funkcionalnosti inteligentne obrade upita, kao i podela na one koje su dodate u verziji iz 2017. godine, i one koje su dodate u verziji iz 2019. godine. (7)

Ažurno dodeljivanje memorije kod moda redova

Kao proširenje ažurnom dodeljivanju memorije kod serijskog moda, sa novom verzijom sistema, dodata je ista funkcionalnost za mod redova, tj. prilikom izvršavanja nad skladištima redova (engl. *rowstore*).

Ovo funkcioniše tako što se svako prvo izvršenje pušta bez ikakvih promena. Zatim, gleda se da li se desilo takozvano „prosipanje memorije“ (engl. *memory spill*), tj.

korišćenje samog diska na računaru (jer dodeljena sistemska memorija nije bila dovoljna), i da li je izvršenje datog plana iskoristilo manje od 50% dodeljene memorije. Ako se desila neka od gorenavedenih situacija, pokreće se ažuriranje količine memorije koja će biti dodeljena sledećem izvršenju plana. U retkom slučaju da se dešava konstantna promena potrebne memorije za neki plan izvršenja, ova funkcionalnost će biti isključena za dati plan, i količina memorije fiksirana.

Kasno kompajliranje tabelarnih varijabli

Ova funkcionalnost poboljšava performanse i kvalitet planova izvršenja kod upita koji referenciraju tabelarne varijable. Tokom optimizacije i kreiranja inicijalnog plana izvršenja, ova funkcionalnost će proslediti procene kardinalnosti koje su zasnovane na pravim brojevima redova tabelarnih varijabli. Ove tačne vrednosti se zatim koriste kako bi se optimizovao plan izvršenja.

Sa kasnim kompajliranjem tabelarnih varijabli, kompajliranje izraza koji sadrži datu varijablu se odlaže do prvog izvršenja datog izraza. Ova promena dovodi do toga da se koriste prave vrednosti kardinalnosti umesto starih pretpostavki.

```
SET STATISTICS IO ON;
SET STATISTICS TIME ON;
DECLARE @Person TABLE
([BusinessEntityID] INT,
 [FirstName]        VARCHAR(30),
 [LastName]         VARCHAR(30)
);
INSERT INTO @Person
       SELECT [BusinessEntityID],
              [FirstName],
              [LastName]
       FROM [AdventureWorks].[Person].[Person];
SELECT *
FROM @Person P1
     JOIN [AdventureWorks].[Person].[Person] P2 ON P1.[BusinessEntityID] = P2.[BusinessEntityI
D];
```

Slika 13. Primer upita koji koriste tabelarnu varijablu (@Person). Kod poslednjeg **SELECT** upita, u staroj verziji sistema, koristio bi se plan izvršenja koji pretpostavlja da je @Person prazna tabela. Kod nove verzije, zapravo će se pretpostaviti adekvatan broj redova. (8)

Serijsko izvršenje kod skladišta redova

Ova funkcionalnost omogućava izvršenje serijskog moda za analitička opterećenja (engl. *analytic workloads*) bez potrebe za indeksima skladišta kolona (engl. *columnstore*).

Sa verzijom sistema koja je izašla 2012. godine, ubačena je nova funkcionalnost kako bi se poboljšale performanse analitičkih opterećenja, indeksi skladišta kolona. Sa svakom sledećom verzijom su performanse tih indeksa poboljšane. **Indeksi skladišta kolona** funkcionišu tako što analitički upiti pristupaju samo podacima u određenim kolonama koji su potrebni upitu. Pored ove funkcionalnosti, postoji još jedna koja je povezana za nju, a to je **procesiranje u serijskom modu**. Ovo procesiranje funkcioniše

tako što računarski procesor obrađuje redove podataka u serijama, umesto da prolazi kroz jedan po jedan red.

Ove funkcionalnosti su fundamentalno razdvojene, i mogu se primeniti tako da se koristi red-po-red procesiranje samo sa indeksima skladišta kolona, ili serijsko procesiranje samo sa indeksima skladišta redova. Doduše, najbolji rezultati se uglavnom dobijaju kada se ove funkcionalnosti koriste zajedno. Do verzije sistema iz 2019. godine, optimizator upita je razmatrao serijsko procesiranje samo kod upita koji se izvršava nad bar jednom tabelom sa indeksom skladišta kolona.

U nekim situacijama, moguće je dobiti bolje performanse bez korišćenja indeksa skladišta kolona, ali sa serijskim procesiranjem redova. Shodno tome, ova nova funkcionalnost omogućava da optimizator upita razmatra serijsko procesiranje redova za sve tipove upita, nevezano nad kakvim indeksima se taj upit izvršava.

Aproksimiranje brojanja unikata

Ova funkcionalnost je deo nove porodice odlika pod imenom obrada upita kroz aproksimaciju (engl. *approximate query processing*). Koristi se kod veoma velikih količina podataka gde je odziv odgovora bitniji od apsolutne preciznosti. Jedan od primera je brojanje unikatnih redova nad 10 milijardi redova, kako bi se rezultat prikaz kao informacija na tabli. U ovom slučaju, stopostotna preciznost nije od tolike važnosti, kao što je brzina odziva.

Kako bi se postiglo aproksimiranje unikatnih vrednosti ovim putem (8), kreirana je nova T-SQL funkcija, sa potpisom **APPROX_COUNT_DISTINCT (*expression*)**. Ova funkcija primenjuje specificirani izraz nad svakim redom u grupi, i vraća procenjeni broj unikatnih redova u grupi. Dizajnirana je za korišćenje nad velikim skupovima podataka i optimizovana je za situacije u kojima se izvršava nad milionima redova, i koriste se agregacije kolona sa mnogo unikatnih vrednosti.

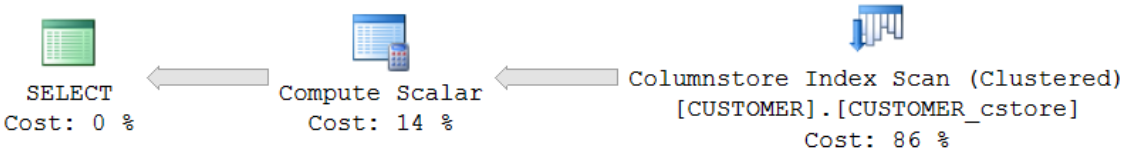
Implementacija ove funkcije garantuje maksimalnu količinu grešaka od 2% sa sigurnošću od 97%. Takođe, ova funkcija koristi manje memorije nego normalna funkcija za brojanje unikatnih redova.

Ugrađivanje skalarnih korisničkih funkcija

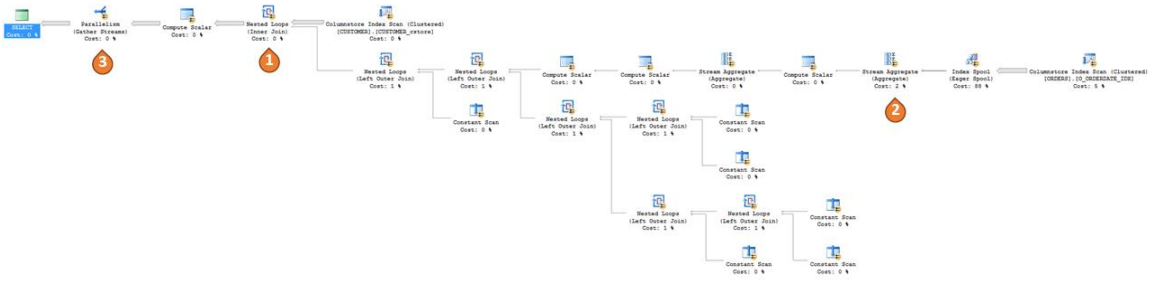
Ova funkcionalnost automatski transformiše skalarane funkcije definisane od strane korisnika u relacione izraze. Zatim, ti izrazi se ugrađuju u sam upit koji poziva datu funkciju. Ovo dovodi do toga da se sama funkcija razmatra u optimizacije zasnovanoj na troškovima koja se generalno vrši nad upitima. Ovo dovodi do velikog poboljšanja u performansama.

```
CREATE OR ALTER FUNCTION dbo.customer_category(@ckey INT)
RETURNS CHAR(10) AS
BEGIN
  DECLARE @total_price DECIMAL(18,2);
  DECLARE @category CHAR(10);

  SELECT @total_price = SUM(O_TOTALPRICE) FROM ORDERS WHERE O_CUSTKEY = @ckey;

  IF @total_price < 500000
    SET @category = 'REGULAR';
  ELSE IF @total_price < 1000000
    SET @category = 'GOLD';
  ELSE
    SET @category = 'PLATINUM';

  RETURN @category;
END;

SELECT C_NAME, dbo.customer_category(C_CUSTKEY) FROM CUSTOMER;
```

Slika 14. Primer definisanja skalarne funkcije, a zatim i njen poziv.



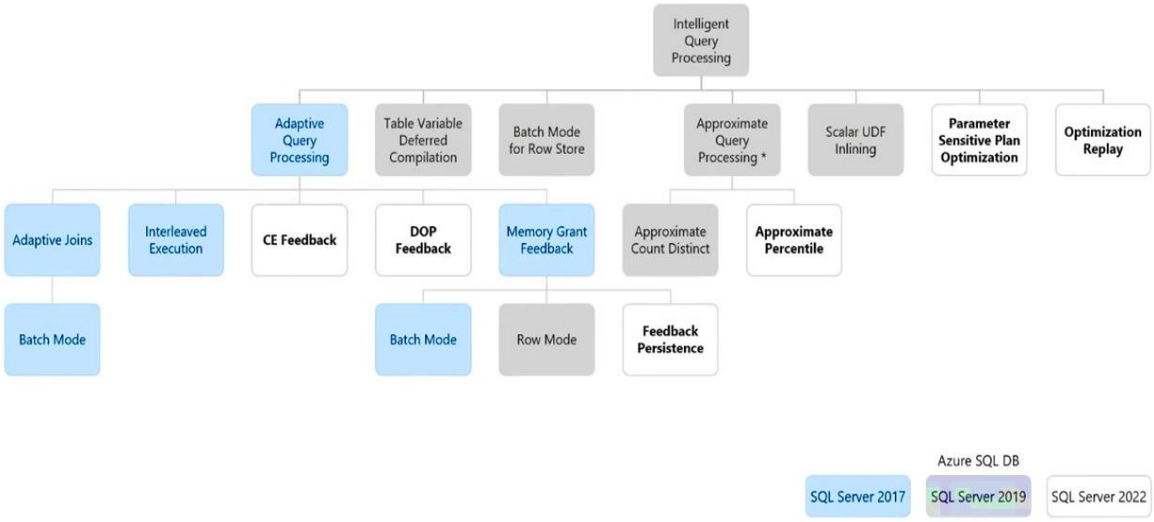
Slika 15. Plan izvršenja koji bi se generisao za dati upit u starim verzijama sistema. Prost i naivan.



Slika 16. Plan izvršenja koji bi se generisao za dati upit u novim verzijama sistema. Skalarna funkcija se analizira i ugrađuje direktno u plan.

Nove funkcionalnosti u MS SQL Server 2022 verziji sistema

Nesumnjivo je da je polje inteligentne obrade upita od velike važnosti Majkrosoftu i da će nova verzija sistema, koja je trenutno u privatnom testiranju, sadržati nove funkcionalnosti vezane za ovo polje. Nećemo zalaziti u ove funkcionalnosti, jer su i dalje podložne promenama i potpune zvanične informacije o njima nisu još uvek dostupne.



Slika 19. Grafički prikaz svih funkcionalnosti inteligentne obrade upita, uključujući i one koje će biti dostupne tek sa novom verzijom sistema. (9)

Bibliografija

1. [Na mreži] https://www.sciencedirect.com/science/article/pii/B9780128191545000187.

2. [Na mreži] https://www.javatpoint.com/query-processing-in-dbms.

3. [Na mreži] https://logicalread.com/sql-server-query-processing-w01/.

4. [Na mreži] https://www.sqlshack.com/sql-server-execution-plans-overview/.

5. [Na mreži] https://www.sqlshack.com/adaptive-query-processing-in-sql-server-2017/.

6. [Na mreži] https://subscription.packtpub.com/book/data/9781838826215/1/ch01lvl1sec09/intelligent-query-processing.

7. [Na mreži] https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing?view=sql-server-ver15.

8. [Na mreži] https://www.sqlshack.com/sql-table-variable-deferred-compilation-in-sql-server-2019/.

9. [Na mreži] https://docs.microsoft.com/en-us/sql/t-sql/functions/approx-count-distinct-transact-sql?view=sql-server-ver15.

10. [Na mreži] https://www.youtube.com/watch?v=bbXM3Pk9Ejw.