UNIVERZITET U NIŠU

ELEKTRONSKI FAKULTET

Katedra za računarstvo

**Optimizacija upita u MS SQL Serveru**

**Sistemi za upravljanje bazama podataka**

**Student:** Minja Ignjatović 1000

# 

[Uvod 3](#_Toc38129635)

[1. Način rada optimizatora upita 4](#_Toc38129636)

[2. Proces optimizacije upita 6](#_Toc38129637)

[2.1. Faze optimizacije upita 6](#_Toc38129638)

[2.1.1. Simplifikacija 6](#_Toc38129639)

[2.1.2. Trivijalni plan 6](#_Toc38129640)

[2.1.3. Puna optimizacija 7](#_Toc38129641)

[2.2. Optimizacija upita ažuriranja 8](#_Toc38129642)

[2.3. *Parameter Sniffing* 8](#_Toc38129643)

[3. Generisanje planova izvršenja 9](#_Toc38129644)

[3.1. Dekompozicija upita 10](#_Toc38129645)

[4. Procena cene izvršenja plana 11](#_Toc38129646)

[4.1. Estimacija veličine rezultata 11](#_Toc38129647)

[4.2. Gustina 11](#_Toc38129648)

[4.3. Upotreba histograma 13](#_Toc38129649)

[4.3. Estimacija cene upita 15](#_Toc38129650)

[4.4. Upiti nad jednom relacijom 16](#_Toc38129651)

[4.4.1. Cena planova bez indeksa 16](#_Toc38129652)

[4.4.2. Cena planova koji koriste indekse 16](#_Toc38129653)

[4.5. Upiti nad više relacija 17](#_Toc38129654)

[5. Selekcija indeksa 18](#_Toc38129655)

[5.1. Mehanizmi selekcije indeksa 18](#_Toc38129656)

[5.2. Pronalaženje nedostajućih indeksa 21](#_Toc38129657)

[5.3. Pronalaženje nekorišćenih indeksa 23](#_Toc38129658)

[5.3.1. Koje nekorišćene indekse treba ukloniti? 24](#_Toc38129659)

[6. Materijalizovani pogledi 25](#_Toc38129660)

[6.1. Prednosti korišćenja materijalizovanih pogleda 25](#_Toc38129661)

[6.2. Nedostaci korišćenja materijalizovanih pogleda 26](#_Toc38129662)

[6.3. Kreiranje materijalizovanih pogleda 26](#_Toc38129663)

[Zaključak 28](#_Toc38129664)

[Reference 29](#_Toc38129665)

# **Uvod**

SQL je jedan od najmoćnijih alata za manipulaciju podacima. Neefikasni upiti mogu dovesti do toga da se resursi potroše i da to utiče na slabe performanse pa čak i gubitak servisa za druge korisnike, ukoliko upit sadrži određene greške. Unapređivanje loše napisanih upita i razrešavanje problema sa performansama može biti veoma naporan posao koji je od ključnog značaja za uspeh i dobre performanse aplikacija. To obuhvata kako istraživanje tako i testiranje različitih rešenja, istraživanje planova izvršenja, statistika, proširenih događaja i slično. Pre svega je potrebno utvrditi koji upiti narušavaju performanse sistema i raditi na njihovom poboljšanju. To je moguće korišćenjem SQL Profiler-a, utvrđivanjem upita koji troše najviše vremena ili koriste najveći deo resursa.

Optimizacija upita predstavlja kompleksnu komponentu DBMS-a. Optimizacija obuhvata analizu određenog broja planova izvršenja koji predstavljaju kandidate za određeni upit. Na osnovu estimirane cene svakog od potencijalnih planova, vrši se izbor plana sa najmanjom cenom.

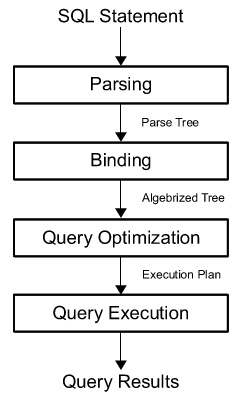
U radu je opisan način rada optimizatora upita, na koji način se vrši procena cene izvršenja plana i faktori koji utiču na tu procenu. Pored toga, prikazan je način selekcije indeksa i mehanizmi za pronalaženje nedostajućih i nekorišćenih indeksa, što može biti veoma korisno za performanse sistema. Kao jedan od načina optimizacije upita, predstavljeni su materijalizovani pogledi, način njihove upotrebe kao i prednosti i mane koji oni pružaju. Prikazane su i faze kroz koje se prolazi prilikom optimizacije samih upita. U radu je korišćen Microsoft SQL Server DBMS. Za prikaz praktične upotrebe tehnika optimizacije korišćena je *AdventureWorks 2017* baza podataka čija kopija je javno dostupna u okviru Microsoft dokumentacije.

# 

# **1. Način rada optimizatora upita**

Srž SQL Servera čine dve glavne komponente, to su: sistem za skladištenje podataka i procesor upita. Sistem za skladištenje podataka je zadužen za čitanje podataka između diska i memorija tako da optimizuje konkurentnost i održi integritet podataka. Sa druge strane, procesor upita prihvata sve upite koji se izvršavaju na SQL Serveru, analizira plan za optimalno izvršenje upita, izvršava plan i pruža zahtevane rezultate.

Za pisanje upita koristi se SQL deklarativni jezik ili T-SQL koji predstavlja ekstenziju nad SQL jezikom. Budući da je SQL jezik visokog nivoa, njime je moguće definisati koje podatke je potrebno preuzeti iz baze podataka, ali ne i definisati korake koji su zahtevani da bi se do tih podataka došlo, niti navoditi algoritme koji bi se mogli koristiti za procesiranje zahteva. Za svaki upit koji pristigne, optimizator upita treba da napravi plan, što je brže moguće. Plan opisuje najbolji mogući način za izvršenje upita, odnosno najefikasniji način. Nakon što se definše plan, potrebno je izvršiti upit na osnovu izabranog plana. Svaki od navedenih zadataka optimizatora upita se delegira posebnoj komponenti optimizatora upita. Sam plan generiše optimizator upita, dok izvršenje plana prosleđuje sistemu za izvršenje upita. Da bi stigao do onoga za šta se smatra da je najbolji plan izvršenja, optimizator upita treba da izvrši nekoliko različitih koraka. Ti koraci su prikazani na Slici 1.



Slika 1 - Koraci pri procesiranju SQL upita

Koraci pri procesiranju SQL upita od strane optimizatora upita su:

1. **Parsiranje i vezivanje** - Upit se parsuje, proverava se njegova sintaksna i spravnost i generiše se inicijalna verzija logičkog stabla, gde svaki čvor u stablu predstavlja logičku operaciju koju upit treba da izvrši, kao što je čitanje podataka iz određene tabele ili izvođenje unutrašnjeg spoja. Cilj formiranja logičkog stabla je njegovo korišćenje u koraku optimizacije upita.
2. **Optimizacija upita** - Optimizacija se sastoji iz:

* Generisanja mogućih planova izvršenja - Korišćenjem logičkog stabla, optimizator upita generiše nekoliko mogućih načina za izvršenje upita. Plan izvršenja upita je skup fizičkih operacija kao što su ugnježdeni spojevi, pretraga indeksa i sl., koji mogu biti izvršeni kako bi se dobio željeni rezultat koji je opisan kroz logičko stablo.
* Procena cene izvršenja svakog od planova - Optimizator upita ne generiše svaki mogući plan izvršenja, on procenjuje skup resursa i vremena svakog plana koji generiše. Plan koji će biti izabran je onaj koji ima najmanju procenjenu cenu izvršenja i taj plan se dalje šalje sistemu za izvršenje upita.

1. **Izvršenje upita i keširanje plana izvršenja** - Upit se izvršava od strane sistema za izvršenje upita, na osnovu izabranog plana. Plan može biti sačuvan u memoriji ili u kešu. Keširanjem se štedi na vremenu jer se onda prethodno navedeni koraci ne moraju obavljati.

# 

# 

# **2. Proces optimizacije upita**

Optimizator upita ima nekoliko faza optimizacije koje su dizajnirane tako da pokušavaju da optimizuju upite što brže i jednostavnije, ne koristeći skupe opcije osim ako to nije neophodno. Faze optimizacije su:

1. **Simplifikacija**
2. **Optimizacija trivijalnih planova**
3. **Puna optimizacija**

Plan izvršenja može biti generisan u bilo kojoj od ovih faza, osim u fazi simplifikacije

## **2.1. Faze optimizacije upita**

## 

## **2.1.1. Simplifikacija**

U fazi simplifikacije vrši se prepisivanje upita, tj prepisivanje stabla kako bi se stablo upita redukovalo u jednostavniji oblik da bi proces optimizacije bio olakšan. Simplifikacija uključuje:

* Ugnježdeni upiti se konvertuju u spojeve, ali pošto nije uvek moguće prevesti ugnježdeni upit u unutrašnji spoj, spoljašnji spoj i operacije grupisanja se mogu dodati po potrebi.
* Redundantni unutrašnji i spoljni spojevi se eliminišu. Tipičan primer je eliminacija *Foreign Key Join*-a koja se javlja kada SQL Server detektuje da neki spoj nije potreban, zato što postoji ograničenje stranog ključa i samo kolone referencirajuće tabele su zahtevane.
* Filteri u WHERE klauzuli se guraju niz stablo upita da bi se omogućilo rano filtriranje podataka, i potencijalno bolje poklapanje sa indeksima.
* Kontradikcije se otkrivaju i uklanjaju.

## **2.1.2. Trivijalni plan**

Sam proces optimizacije može biti skup za inicijalizaciju i izvršenje jednostavnih upita, koji ne zahtevaju bilo kakvu estimaciju cene. Da bi izbegao ovu skupu operaciju za jednostavne upite, SQL Server koristi optimizaciju trivijalnih planova (Slika 22). Ukoliko postoji samo jedan način da se izvrši dati upit, zavisno od definicije upita i dostupnih metapodataka, dosta posla se može izbeći.





Slika 22 - Prikaz dela trivijalnog plana za izvršenje jednostavnog upita

Pored trivijalnog plana, druga opcija za nivo optimizacije je *FULL,* što znači da je izvršen proces pune optimizacije i da se dati upit nije kvalifikovao za trivijalni plan. Puna optimizacija se koristi za kompleksnije upite ili upite koji koriste kompleksne operacije, koji zahtevaju poređenja cena planova kandidata kako bi se donela konačna odluka.

## **2.1.3. Puna optimizacija**

Ukoliko se upit ne kvalifikuje za trivijalni plan, SQL Server pokreće proces optimizacije, baziran na ceni, koji koristi transformaciona pravila kako bi generisao alternativne planove. Alternativni planovi se čuvaju u memo strukturi i koristi se estimacija cene kako bi se izabrao najbolji plan. Sam proces pune optimizacije se izvršava u tri faze, gde se u svakoj koriste drugačija pravila transformacije. Budući da može postojati veliki broj alternativnih planova, SQL Server koristi heuristiku kako bi se ograničio broj generisanih alternativa. Takođe, proces se može odmah završiti ukoliko se pronađe dovoljno dobar plan-relativno internom pragu koji je poznat optimizatoru. Ukoliko je plan na kraju svake od faza previše skup, optimizator izvršava narednu etapu, gde koristi skup kompleksnijih transformacionih pravila. Faze u okviru pune optimizacije su:

1. Search 0
2. Search 1
3. Search 2

**Search 0 - *Transaction Processing Phase***

Prva faza, slično konceptu trivijalnih planova, ima za cilj da pronađe plan što je moguće brže, bez pokušavanja sofisticiranih transformacija. Search 0, koja se naziva još i *Transaction Processing Phase* je idealan za male upite koji su tipični za sisteme koji procesiraju transakcije i koristi se za upite sa najmanje tri tabele. Pre nego što se proces optimizacije pokrene, optimizator upita generiše inicijalni skup redosleda spojeva na osnovu heuristike. Ova heuristika počinje tako što se prvo vrši spoj najmanje tabele, ili tabele koja dostigne najveću stopu filtriranja na osnovu selektivnosti. To su jedini redosledi spojeva koji se razmatraju u Search 0. Na kraju ove faze, optimizator upita poredi najbolji generisani plan sa internim pragom cene, i ako je plan i dalje previše skup, SQL Server će izvršiti sledeću fazu.

**Search 1 - *Quick Plan***

Search 1, koji se još zove i *Quick Plan*, koristi dodatna transformaciona pravila, ograničeno preuređenje redosleda spojeva, i prikladan je za kompleksnije upite. Na kraju ove faze, SQL Server poredi cenu najjeftinijeg plana sa drugim internim pragom cene, i ako je najbolji plan dovoljno jeftin, onda se bira taj plan. Ako je upit i dalje skup i sistem može da koristi parelelne upite, ova faza se izvršava ponovo da se pronađe dobar paralelni plan, ali se ni jedan plan ne bira za izvršenje nakon ovog koraka. Na kraju ove faze, porede se cene najbolje serijskog i paralelnog plana, i najjeftinij plan se koristi u narednoj fazi, Search 2.

**Search 2 - *Full Optimization***

Poslednja faza, Search 2, se zove još i puna optimizacija, i koristi se samo za upite koji variraju od kompleksnih do jako kompleksnih. Veći skup potencijalnih transformacionih pravila, paralelni operatori i druge napredne optimizacione strategije se razmatraju u ovoj fazi i pošto je ovo poslednja faza u njoj se mora pronaći plan izvršenja. Proces optimizacije takođe uključuje koncept *timeout-a*, pri čemu ova vrednost nije fiksna, već se računa zavisno od broja transformacija koje su primenje zajedno sa proteklim vremenom. Kada se pronađe *timeout*, optimizator upita prekida proces optimizacije i vraća najjeftiniji plan koji je pronađen do tog trenutka.

## **2.2. Optimizacija upita ažuriranja**

Planovi koji se generišu za operacije ažuriranja mogu biti veoma komplikovani, budući da ažuriranje podrazumeva i ažuriranje indeksa. Operacije ažuriranja se izvršavaju u dva koraka, koji se mogu svesti na čitanje i upis.

* Prvi korak obezbeđuje detalje promena koje treba primeniti i torke koje treba ažurirati. Kada je reč o INSERT operacijama, to podrazumeva vrednosti koje je potrebno upisati. Kod DELETE operacije, detalji uključuju pronalaženje ključeva torki koje će biti obrisane koje mogu biti i ključevi klasterizovanih indeksa. Detalji UPDATE operacije se tiču kombinacije ključeva torki koje će biti ažurirane i podataka koji će biti upisani.
* Drugi korak obuhvata izvršavanje operacija ažuriranja, uključujući ažuriranje indeksa, validacije ograničenja i postojećih trigera.

## **2.3. *Parameter Sniffing***

SQL Server može da koristi statistiku da estimira kardinalnost upita, i da onda pokuša da upotrebi ovu informaciju da generiše optimalan plan izvršenja. Optimizator upita ovo radi tako što prvo pregleda vrednosti parametara upita. Ovo ponašanje se naziva *Parameter sniffing* i generalno je dobra stvar: dobija se plan izvršenja koji je specifično optimizovan za trenutne parametre. Takođe, SQL Server može da kešira planove izvršenja tako da mogu biti iskorišćeni više puta kada je potrebno izvršiti identičan upit. Ovo štedi optimizaciono vreme i CPU resurse, jer se upit ne mora optimizovati ponovo. Sa druge strane, iako ovo ponašanje u većini slučajeva funkcioniše jako dobro, nekada može dovesti do problema u performansama. Obzirom da optimizator upita može da generiše raziličite planove izvršenja zavisno od vrednosti parametara upita, keširanje i ponovno korišćenje nekog od ovih planova može dovesti do problema u performansama za alternativne instance datog upita kojima bi bolje odgovarao drugi plan izvršenja. Ovo je poznati problem kada se koristi eksplicitna parametrizacija upita, kao što je slučaj sa store procedurama.

# 

# **3. Generisanje planova izvršenja**

Kao što je ranije pomenuto, cilj optimizatora upita je pronalaženje efikasnog plana izvršenja za navedeni upit. Relativno jednostavni upiti mogu dati veliki broj različitih načina za pristup željenim podacima i za dobijanje istog rezultata. Cilj je odabrati najbolji mogući plan od svih planova koji predstavljaju kandidate.

Za zadati upit generiše se prostor traženja kao skup potencijalnih planova. Teoretski, da bi se izabrao najbolji plan upita, potrebno je generisati sve moguće planove. Sa strane performansi ovo nikako nije dobro, iz razloga što kompleksniji upiti mogu imati ogroman broj potencijalnih puteva izvršenja, što bi oslabilo performanse sistema. Na primer, ukoliko optimizator upita nađe dovoljno dobro rešenje, nema smisla trošiti dodatno vreme na traženje savršenog rešenja, jer se na taj način problem vraća na početak. Poenta nije nalaženje savršenog plana izvršenja, već nalaženje dovoljno dobrog plana.

Za istraživanje prostora traženja optimizator upita koristi pravila za transformaciju i heuristiku. Heuristika se koristi za ograničavanje brojeva potencijalnih rešenja kako bi se postiglo optimalno vreme izvršenja upita.

Plan izvršenja upita predstavlja skup sledećih stvari:

1. **Sekvenca pristupa tabelama** - Tipično, da bi se dobio rezultat, server baze podataka može na različite načine pristupati tabelama. Ukoliko npr SELECT referencira tri tabele, TableA, TableB i TableC, server baze podataka može pristupiti ovim tabelama u različitom redosledu i pri tom dobiti iste rezultate.
2. **Metode koje se koriste za izdvajanje podataka iz svake tabele**: Postoje različiti načini za pristup podacima u svakoj tabeli. Ukoliko se zahteva samo nekoliko redova sa specifičnim ključem, server baze podataka može koristiti indeks. Ako se zahtevaju svi redovi tabele, server može ignorisati indeks i izvršiti skeniranje celokupne tabele. U situacijama kada se zahtevaju svi zapisi iz tabele ali postoji indeks čije kolone ključevi se nalaze u okviru ORDER BY klauzule, korišćenje skeniranja indeksa umesto skeniranja tabele može sačuvati zasebnu vrstu skupa rezultata. Ukoliko je tabela veoma mala, skeniranje tabele je obično najefikasniji metod za skoro sve pristupe tabeli.
3. **Metode koje se koriste za izvršenje računanja i filtriranja, agregiranje i sortiranje podataka iz svake tabele**: Postoje različite metode za izvršavanje računanja nad podacima, kao što je računanje skalarnih vrednosti i agregiranje i sortiranje podataka na način koji je naveden u samom upitu korišćenjem npr klauzula kao što su GROUP BY i ORDER BY, i filtriranje podataka korišćenjem WHERE ili HAVING klauzula.

*SQL Server Management Studio* nudi tri opcije za prikaz planova izvršenja i to su:

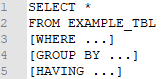
1. **Estimirani plan izvršenja**: On predstavlja kompajlirani plan, koji je proizveden od strane optimizatora upita i zasniva se na estimacijama.
2. **Stvarni plan izvršenja**: Ovaj plan je isti kao i kompajlirani plan, sa dodatkom konteksta izvršenja. On uključuje informacije koje su dostupne nakon izvršenja upita, kao što su upozorenja generisana prilikom izvršenja, vreme izvršenja, vreme korišćenja CPU-a tokom izvršenja upita i sl.
3. **Statistika upita**: Statistika obuhvata kompajlirani plan i kontekst izvršenja. Ona uključuje i *runtime* informacije tokom samog procesa izvršenja, kao što je broj redova koji prolaze kroz operatore.

## **3.1. Dekompozicija upita**

Dekompozicija je proces razdvajanja upita u manje jedinice ili elemente koji se još nazivaju i blokovi. Dekompozicija vrši zamenu relacije sa određenim brojem manjih relacija. Karakteristike dekompozicije su:

1. **Nema gubitaka** - Dekompozicija mora biti bez gubitaka, što znači da informacije ne smeju biti izgubljene, posmatrajući relaciju koja je razbijena na više manjih relacija.
2. **Očuvanje zavisnost**i - Zavisnost nad bazom podataka je bitno ograničenje. Svako navedeno ograničenje mora biti zadovoljeno.
3. **Nedostatak redukcije podataka** - Ova karakteristika je poznata i kao ponavljanje informacija. Pravilna dekompozicija ne sme da pretrpi suvišne podatke. Nepažljivo dekomponovanje može dovesti do problema sa podacima. Nedostatak svojstva redundantnosti može se postići procesom normalizacije.

Optimizator upita vrši optimizaciju jednog bloka u jednom trenutku. Jedan blok predstavlja jedan upit bez ugnježdenja i sa tačno jednom SELECT i jednom FROM klauzulom i najviše jednom WHERE, GROUP BY ili HAVING klauzulom. Na Slici 2 je prikazan izgled jednog bloka SQL upita, pri čemu su izrazi u uglastim zagradama opcioni.



Slika 2 - Primer bloka SQL upita

# **4. Procena cene izvršenja plana**

Pretraga i generisanje planova kandidata je samo jedan deo procesa optimizacije. Optimizator upita treba da izvrši procenu troškova svakog od planova i da izabere onaj sa najmanjom cenom. Kako bi se izvršila procena cene, optimizator procenjuje cenu svakog fizičkog operatora u planu, koristeći formule koje uzimaju u obzir upotrebu resursa kao što su I/O, CPU i memorija. Procena cene najviše zavisi od algoritma koji je korišćen od strane fizičkog operatora, kao i od estimiranog broja zapisa koji treba da budu obrađeni. Ovakva estimacija brojeva zapisa se naziva procena kardinalnosti.

Kako bi procena kardinalnosti bila moguća, SQL Server koristi i održava statistiku optimizatora koja sadrži statističke informacije koje opisuju distribuciju vrednosti u jednoj ili više kolona određene tabele. Kada je cena svakog od operatora estimirana, optimizator dodaje sve cene kako bi estimirao cenu celokupnog plana.

## **4.1. Estimacija veličine rezultata**

Izlaz jednog operatora može biti ulaz za drugi operator. Cena jednog operatora zavisi od veličine njegovih ulaznih vrednosti. Jedan od načina za estimaciju veličine rezultata je da se svakom terminu u WHERE klauzuli dodeli redukcioni faktor.

Redukcioni faktor predstavlja odnos veličine rezultata i veličine ulaznog podatka, uzimajući u obzir samo selekciju nad tim terminom. Veličina celokupnog rezultata se može izračunati kao proizvod pojedinačnih redukcionih faktora. Postoji više načina za računanje redukcionih faktora, korišćenjem informacija koje su dostupne u sistemskom katalogu. Neke od situacija i načini računanja redukcionih faktora u datim situacijama su:

1. **Kolona = Vrednost**: redukcioni faktor se aproksimira kao 1/𝑁𝐾𝑒𝑦𝑠(𝐼) ako postoji indeks I nad kolonom za relaciju o kojoj je reč. Ako ne postoji indeks nad kolonom, uzima se neka određena vrednost kao što je 1/10 .
2. **Kolona1 = Kolona2**: redukcioni faktor se aproksimira kao 1/𝑀𝐴𝑋(𝑁𝐾𝑒𝑦𝑠(𝐼1),𝑁𝐾𝑒𝑦𝑠(𝐼2)) ako postoje indeksi I1 i I2 nad kolonama Kolona1 i Kolona2, respektivno.
3. **Kolona > Vrednost**: redukcioni faktor se aproksimira kao (𝐻𝑖𝑔ℎ(𝐼) − 𝑣𝑟𝑒𝑑𝑛𝑜𝑠𝑡)/ (𝐻𝑖𝑔ℎ(𝐼) − 𝐿𝑜𝑤(𝐼)).
4. **Kolona IN (lista vrednosti)**: redukcioni faktor se uzima kao da se radi o kolona = vrednost slučaju pomnoženo sa brojem vrednosti u listi.

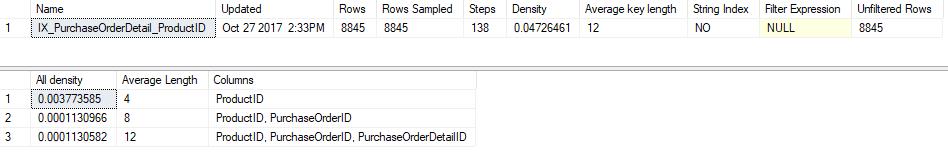
## **4.2. Gustina**

Gustina se izračunava kao 1/Broj jedinstvenih vrednosti. Informacije o gustini mogu se upotrebiti kako bi se unapredile estimacije za GROUP BY operacije i za jednakost predikata, gde je vrednost nepoznata, kao što je slučaj sa lokalnim promenljivama. Računa se za svaki par kolona, formirajući na taj način prefiks za te kolone u statističkom objektu. Na Slici 3 prikazana je komanda koja je izvršena kako bi se dobio uvid u statistiku određenog indeksa.



Slika 3 - Komanda za prikaz statistike navedenog indeksa

Pomenuta komanda daje informacije o gustini vektora, prikazujući gustine za kolone u Columns koloni i to za: *ProductID*, kao i kombinacija kolona *ProductID, PurchaseOrderID* i *ProductID, PurchaseOrderID, PurchaseOrderDetailID* (Slika 4). Pored te informacije, budući da se sve pomenute kolone čuvaju kao int vrednosti, prosečna dužina za svaku od vrednosti za gustinu će biti 4, 8 i 12 bajtova.



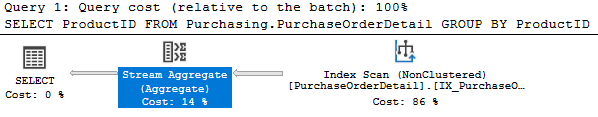
Slika 4 - Rezultat komande za prikaz statistike navedenog indeksa

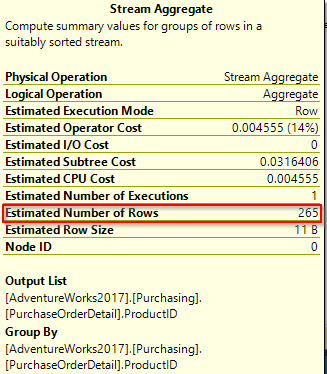
Optimizator upita može koristiti informaciju o gustini kako bi procenio kardinalnost GROUP BY upita. Ukoliko postoji informacija o gustini, sve što je potrebno uraditi da bi se dobio procenjeni broj jedinstvenih vrednosti je računanje recipročne vrednosti gustine tj 1/Gustina.

Da bi se izračunala kardinalnost upita na Slici 5, koji koristi GROUP BY *ProductID*, možemo izračunati recipročnu vrednost *ProductID* gustine. U ovom slučaju, 1/0.003773585 daje nam broj od 265, što predstavlja estimirani broj redova koji su i prikazani u planu izvršenja na Slici 6.



Slika 5 - Upit koji sadrži GROUP BY klauzulu

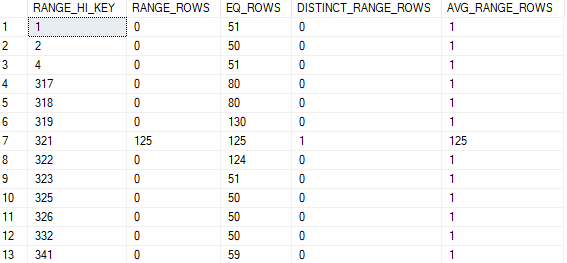




Slika 6 - Prikaz izračunatog procenjenog broja redova na osnovu gustine

## **4.3. Upotreba histograma**

Veći broj DBMS-ova koristi neku vrstu napredne statistike u vidu histograma za određivanje redukcionih faktora. U SQL Serveru, histogrami se koriste samo za prvu kolonu statističkog objekta i sadrže samo informacije o raspodeli vrednosti u tim kolonama za podelu informacija na podskupove koji se nazivaju koraci (eng. *steps*) ili korpe (eng. *buckets*). Maksimalni broj koraka u histogramu je 200. Kako bi se histogram napravio, SQL Server nalazi jedinstvene vrednosti u koloni i pokušava da zabeleži najčešće korišćene vrednosti koristeći *maxdiff* algoritam. Korišćenjem ovog algoritma pruža se najznačajnija informacija i njegova svrha je distribucija vrednosti u relacionim bazama podataka. Na Slici 7 prikazan je deo redova koji su rezultat upita na Slici 3.



Slika 7 - Deo rezultata koji predstavljaju histogram

* RANGE\_HI\_KEY predstavlja gornju granicu koraka histograma. Sledeći zapis određuje opseg od prethodne vrednosti ovog polja do trenutne vrednosti u tom redu. Npr korak 4 može sadržati samo vrednosti između 5 i 317.
* RANGE\_ROWS je estimirani broj redova čije vrednosti kolona ulaze u opseg koraka, isključujući gornju granicu.
* EQ\_ROWS je estimirani broj redova čije vrednosti kolona su jednake vrednosti RANGE\_HI\_KEY. Ova vrednost predstavlja stvarni broj postojećih zapisa za vrednost polja ProductId od npr 321 (korak 7).
* DISTINCT\_RANGE\_ROWS je procenjeni broj redova sa jedinstvenom vrednosti kolone unutar opsega, isključujući gornju granicu.
* AVG\_RANGE\_ROWS je prosečan broj redova po jedinstvenoj vrednosti, isključujući gornju granicu. Računa se kao RANGE\_ROWS\DISTINCT\_RANGE\_ROWS.

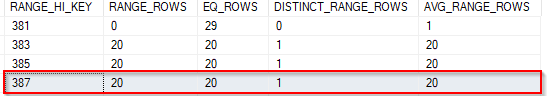
Postoje dve različite vrste histograma: *equiwidth* i *equidepth.* Opseg vrednosti se deli na pod-opsege i za svaki pod-opseg vrednosti broje se torke sa vrednošću za koju se vodi statistika. Kod *equiwidth* histograma, opseg se deli u pod-opsege podjednakih veličina. Sa druge strane, *equidepth* histogram opseg vrednosti deli na pod-opsege tako da svaka korpa sadrži podjednak broj torki. Generalno, *equidepth* histogrami pružaju bolju statistiku od *equiwidth* histograma i zato su češće u upotrebi u većini modernih DBMS-ova.

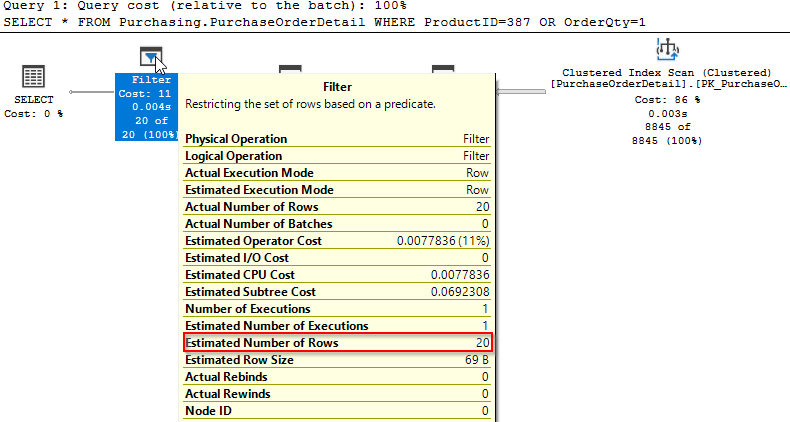
Primer upotrebe kako bi se prikazala korisnost informacija koje su otkrivene pomoću histograma prikazan je pomoću upita na Slici 8.



Slika 8 - Upit za prikaz detalja narudžbine sa parametrom ProductID=387

Po definiciji, predikat OR predstavlja uniju redova obe klauzule, bez duplikata. To znači da bi estimirani broj redova trebao biti: broj estimiranih redova za ProductID=870 + broj estimiranih redova za OrderQty=1, ali ako postoje duplikati oni se uzimaju u obzir samo jednom. Estimirani broj redova za uslov ProductID=387 je 20 a za uslov OrderQty=1 je 1. Budući da je broj zapisa kojima pripadaju oba uslova 1, sledi da će ukupan broj estimiranog broja redova biti 20+1-1=20, što se i može videti na sledećoj slici.





Slika 8 - Prikaz broja estimiranih redova za upit sa Slike 7

## **4.3. Estimacija cene upita**

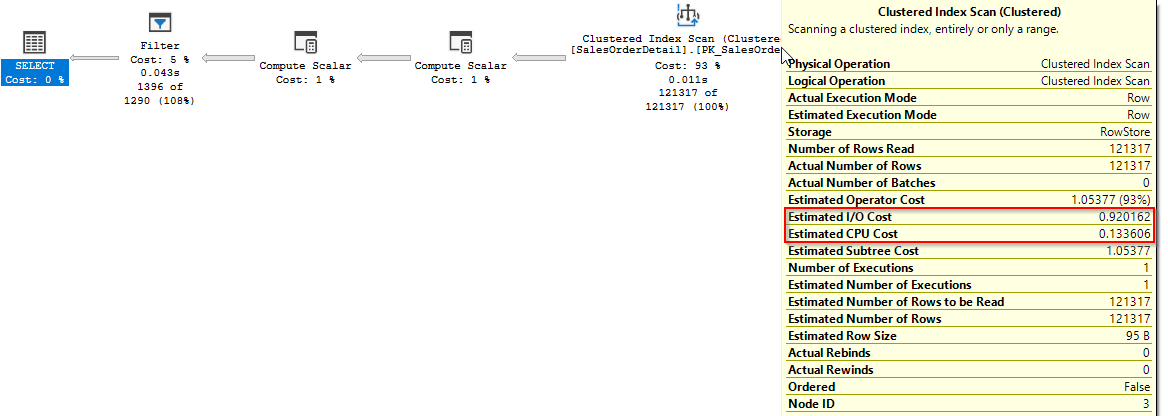
Kvalitet plana izvršenja koje generiše optimizator upita je direktno povezan sa tačnošću njegovih estimacija. Čak i u situacijama kada optimizator upita pronađe plan izvršenja sa niskom cenom, greška u proceni može rezultovati odabirom neefikasnih planova koji negativno utiču na performanse baze podataka. Estimacija mora biti efikasna, budući da se ona koristi nekoliko puta tokom procesa optimizacije.

Računanje cene se obavlja za svaki operator zasebno, a celokupna cena je suma cena svih operatora u okviru plana. Cena svakog operatora zavisi od algoritma i procenjenog broja zapisa koje on vraća. Neki operatori, kao što su Sort ili Hash Join, uzimaju u obzir i dostupnu memoriju u sistemu.

Za svaki od operatora se vrši procena CPU cene, a neki od njih imaju i I/O cenu. Operator kao što je *Clustered Index Scan* ima obe pomenute cene, dok *Stram Aggregate* ima samo CPU cenu. Računanja cene nisu javno dostupna, zaštićena su od strane kompanije Microsoft.

Izvršavanjem upita sa Slike 9 mogu se analizirati dobijeni rezultati za I/O i CPU estimirane cene.





Slika 9 - Upit za dobijanje detalja o narudžbini kada je ukupan broj linija 35

Za *Clustered Index Scan* operator, CPU cena je 0.0001581 za prvi zapis, a za svaki naredni 0.0000011. Budući da je u ovom slučaju estimirano 121 317 zapisa, za računanje estimirane CPU cene može se koristiti izraz 0.0001581+0.0000011 \* (121317 – 1), čiji rezultat je 0.133606. Ova vrednost je prikazana kao *Estimated CPU Cost* na Slici 9.

Na sličan način, minimum I/O cena je 0.003125 za prvu stranicu, a zatim raste u inkrementima od po 0.00074074 za svaku narednu stranicu. Budući da operator koji se koristi skenira celokupnu tabelu, i broj stranica 1 234, izraz 0.003125 + 0.00074074 \* (1234 – 1) daje ukupnu procenjenu vrednost I/O cene.

## **4.4. Upiti nad jednom relacijom**

Ukoliko upit sadrži samo jednu relaciju u FROM klauzuli, kao operacije se mogu javiti samo projekcija, grupisanje i agregacija, bez spojeva.

## **4.4.1. Cena planova bez indeksa**

Ukoliko ne postoje odgovarajući indeksi za obradu upita, vrši se skeniranje relacije i primenjuju se selekcija i projekcija na svaku pročitanu torku, na osnovu algebarskog izraza koji se obrađuje. Rezultat se sortira na osnovu GROUP BY klauzule a filtriranje na osnovu HAVING klauzule. Ukupna cena ovakve obrade se sastoji iz zbira cena sledećih koraka:

* Skeniranje datoteke radi čitanja svih torki, primena selekcije i projekcije
* Ispisivanje torki nakon projekcije i selekcije
* Sortiranje torki da bi se implementirala GROUP BY klauzula

## **4.4.2. Cena planova koji koriste indekse**

## 

Postoji više planova koji se mogu kreirati korišćenjem indeksa:

1. **Putanja pristupa sa jednim indeksom** - Ukoliko više indeksa odgovara uslovu selekcije u WHERE klauzuli, svaki indeks pruža alternativnu putanju pristupa. Optimizator bira putanju za koju proceni da će rezultovati sa najmanje pročitanih stranica.
2. **Putanja pristupa sa više indeksa** - Ukoliko nekoliko indeksa odgovara uslovu selekcije, svaki od njih se može koristiti za čitanje skupova rid-ova. Nakon toga se vrši presek skupova, nakon čega se mogu pročitati torke koje zadovoljavaju termine u selekciji svih odgovarajućih indeksa.
3. **Putanja pristupa sa sortiranim indeksom** - Ako lista atributa po kojima se vrši grupisanje pripada prefiksu nekog stablo indeksa, indeks se može iskoristiti za pronalaženje torki u redosledu koji zahteva GROUP BY klauzula. Svi uslovi selekcije se potom mogu primeniti nad pročitanim torkama. Ova strategija je povoljna za klasterizovane indekse.
4. **Putanja pristupa samo sa indeksima** - Ako se svi atributi koji se javljaju u upitu nalaze u ključu traženja nekog gustog indeksa nad relacijom u FROM klauzuli, može se obaviti skeniranje samo indeksa (eng. *index only scan*) da se izračunaju odgovori. Dobra stvar je što nikada ne moraju da se pročitaju stvarne torke, već se mogu pročitati samo podaci iz indeksa.

## **4.5. Upiti nad više relacija**

Blokovi upita koji sadrže dve ili više relacija u WHERE klauzuli zahtevaju spojeve. Bitno je pronaći dobar plan izvršenja jer oni mogu biti veoma skupi. Veličina konačnog rezultata se može estimirati proizvodom veličina relacija u FROM klauzuli i redukcionih faktora za termine u WHERE klauzuli. Međutim, u zavisnosti od redosleda kojim se obavljaju spojevi između relacija, mogu se kreirati relacije različitih veličina za čuvanje među rezultata, što može dovesti do kreiranja različitih planova.

Enumeracija *left-deep*  planova obuhvata enumeraciju stabla u kome je svako desno dete u čvoru spoja neka od osnovnih relacija. Selekcije i projekcije se razmatraju što je ranije moguće u većini slučajeva.

# **5. Selekcija indeksa**

Selekcija indeksa je jedna od najbitnih tehnika koja se koristi za optimizaciju upita. Korišćenjem pravih indeksa, SQL Server može ubrzati upite i drastično povećati performanse same aplikacije. SQL Server koristi indekse kako bi izvršavao operacije traženja i skeniranja. Indeksi se koriste kako bi se ubrzalo izvršavanje upita brzim nalaženjem zapisa bez izvršavanja skeniranja tabele. Ovo je omogućeno na taj način što se pristupa kolonama koje su zahtevane upitom, bez potrebe za pristupanjem celoj tabeli.

Deo optimizatora upita je da utvrdi da li se indeks može koristiti da evaluira predikate u upitu. To je zapravo upoređivanje ključa indeksa i konstante ili promenljive. Pored toga, optimizator upita treba da utvrdi da li indeks pokriva upit, odnosno da li indeks sadrži sve kolone koje su zahtevane upitom. To je potrebno utvrditi jer neklasterizovani indeksi najčešće sadrže samo podskup kolona iz tabele.

SQL Server može korisititi i više od jednog indeksa i njihovim spajanjem moguće je pokriti sve kolone zahtevane od strane upita. U slučaju da nije moguće pokriti sve kolone, moguće je da je pristup osnovnoj tabeli neophodan, što može biti klasterizovan indeks ili heap. Ova operacija se naziva *Key Lookup* i može biti veoma skupa u slučaju da se zahteva nasumičan I/O.

Postoje situacije u kojima je moguće korišćenje jednog ili više indeksa, ali na kraju oni ne budu odabrani u planu izvršenja, budući da se konačna odluka donosi na osnovu cene. Zbog toga je potrebno, nakon kreiranja indeksa, utvrditi da se indeks zapravo koristi u planovima i da upit ima bolje performanse. Indeks koji se ne koristi može zauzimati dragocen prostor na disku i može negativno uticati na performanse operacija ažuriranja, ne pružajući nikakav boljitak. Takođe može se dogoditi da se indeks, koji je prvobitno bio koristan, više ne koristi.

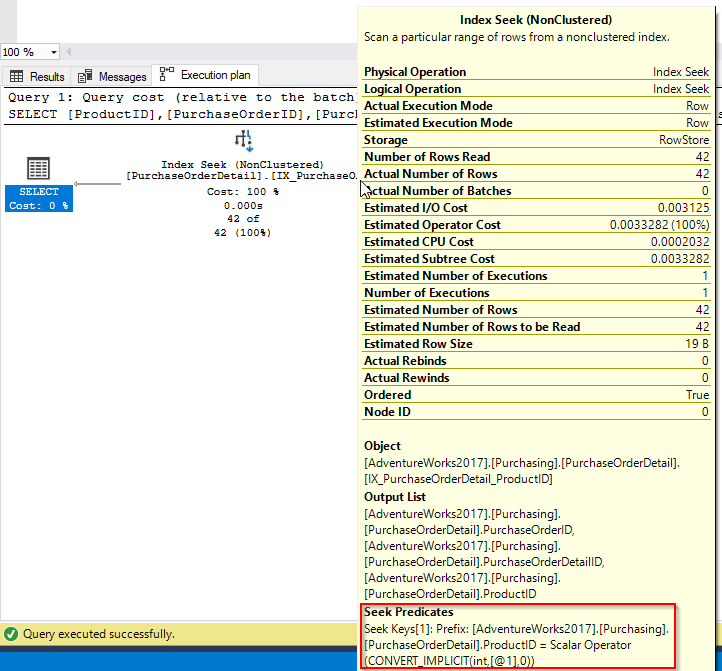
## **5.1. Mehanizmi selekcije indeksa**

Prilikom operacije pretraživanja SQL Server vrši navigaciju kroz indeks B-stabla kako bi brzo pronašao željene zapise, bez potrebe za skeniranjem indeksa ili tabele. Analogija ovom ponašanju može biti sadržaj na kraju knjige kako bismo brzo pronašli celinu koja nas zanima bez potrebe da zbog toga pročitamo čitavu knjigu. Kada se pronađe prvi zapis, SQL Server može skenirati indeks na nivou listova unapred ili unazad, kako bi pronašao dodatne zapise. Operatori jednakosti i nejednakosti se mogu koristiti u predikatu, kao npr =, <, >, <=, >=, <>, !=, !<, !>, BETWEEN i IN. Navedeni predikati mogu se koristiti za mapiranje nad operacijom pretraživanja indeksa, ukoliko je posmatrana kolona navedena kao vodeći ključ indeksa:

* ProductId=350
* UnitPrice<4.563
* FirstName=’Allen’
* LastName LIKE ‘Brown%’

Sledeći upit koristi operator pretraživanja indeksa:





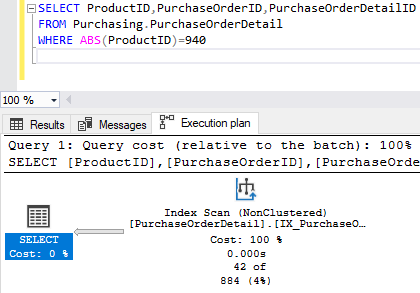
Slika 10 - Prikaz predikata za *Index Seek* navedenog upita

*PurchaseOrderDetail* tabela sadrži indeks koji se sastoji iz više kolona, sa *ProductID* vodećom kolonom. Svojstva operatora *Index Seek* sadrže *Seek* predikat nad *ProductID* kolonom (Slika 10), što ukazuje na to da je SQL Server mogao efikasno da koristi pretraživanje indeksa nad navedenom kolonom.

Sa druge strane, indeks se ne može koristiti za kompleksnije izraze, izraze koji koriste funkcije ili stringove koji na početku imaju *wildcard* karakter, kao što su:

* ABS(ProductID)=940
* UnitPrice+1 < 45.12
* LastName like ‘%Allen’

Korišćenjem neke od pomenutih funkcija SQL Server više nije u mogućnosti da koristi pretraživanje indeksa, već umesto toga koristi skeniranje indeksa, što je i prikazano na Slici 11.



Slika 11 - Plan izvršenja kada se kao predikat koristi ABS funkcija

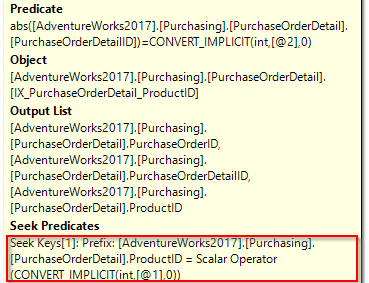
U slučaju indeksa koji se sastoje iz više kolona, SQL Server može koristiti indeks za pretragu po drugoj koloni ukoliko postoji jednakost predikata nad prvom kolonom. Pod pretpostavkom da postoje indeksi sa više kolona nad obe kolone u predstavljenom redosledu, SQL Server može koristiti indeks sa više kolona za pretragu nad obe kolone u sledećim slučajevima:

* ProductID=940 AND PurchaseOrderID > 350
* LastName=’Smith’AND FirstName=’John’

Ukoliko jednakost nad prvom kolonom ne postoji, ili predikat ne može biti evaluiran nad drugom kolonom (u slučaju kompleksnijih izraza), SQL Server će biti u mogućnosti da vrši pretragu indeksa samo nad prvom kolonom. U situaciji kada SQL Server nije u mogućnosti da vrši pretragu indeksa nad prvom navedenom kolonom, pretraga nad indeksom koji sadrži više kolona neće biti moguća.

Na Slici 12 prikazana je situacija kada je pretraga nad indeksom moguća samo po prvo navedenoj koloni, koloni *ProductId* a pretraga po *PurchaseOrderDetailID* koloni nije moguća budući da se koristi ABS funkcija.



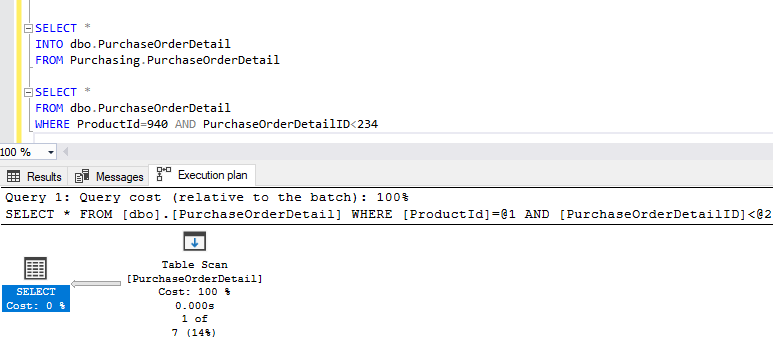


Slika 12 - Prikaz situacije kada je moguća pretraga indeksa samo po prvoj koloni u indeksu koji se sastoji iz više kolona

## **5.2. Pronalaženje nedostajućih indeksa**

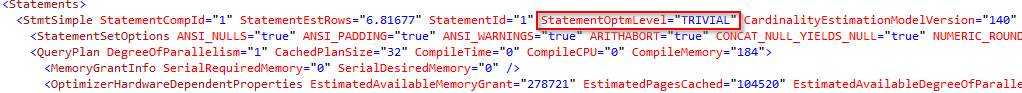
SQL Server pruža funkcionalnost pronalaženja korisnih nedostajućih indeksa za postojeće upite. Tokom same optimizacije, optimizator upita definiše najbolje indekse za upite i, ako oni ne postoje, pruža informaciju o tome za posmatrani plan. Takođe, SQL Server agregira prikupljene podatke, od kako je instanca startovana i skladišti ih u *sys.dm\_db\_missing\_index* DMV-u (*Database Management View*). Prikazivanjem ovakve informacije, SQL Server obaveštava da potencijalno ne koristi optimalan plan izvršenja i pored toga pokazuje koji bi indeksi mogli dovesti do poboljšanja performansi upita.

Nakon upita koji kreira novu tabelu *dbo.PurchaseOrderDetail* i popunjava je podacima iz tabele *Purchasing.PurchaseOrderDetail,* izvršen je upit koji je prikazan na Slici 13.



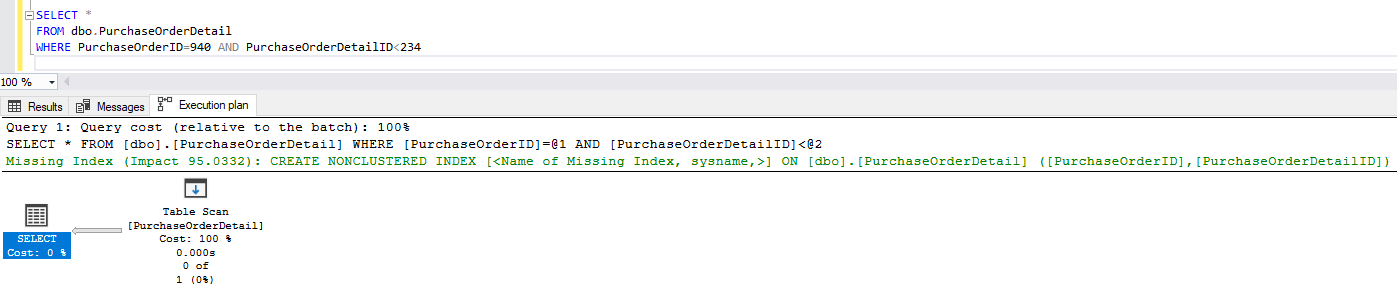
Slika 13 - Kreiranje tabele *dbo.PurchaseOrderDetail* i izvršavanje upita nad kreiranom tabelom

Prikazani upit bi mogao imati koristi od indeksa nad *ProductId* i *PurchaseOrderDetailID* ali SQL Server ne prikazuje nikakve informacije o nedostajućem indeksu. Razlog tome je taj što postoji ograničenje koje se tiče funkcionalnosti nedostajućih indeksa, a to je da funkcionalnost nema primenu nad trivijalnim planovima optimizacije. Ukoliko pogledamo XML format plana, polje *StatementOptmLevel* ima vrednost TRIVIAL, što je prikazano na Slici 14.

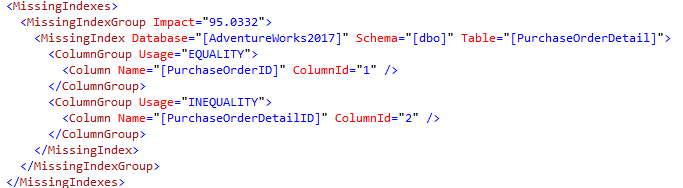


Slika 14 - Prikaz dela XML plana izvršenja upita prikazanog na Slici 13

U okviru XML plana izvršenja i zapisa *Missing Indexes* mogu se uočiti tri grupe: : *equality*, *inequality*, i *included.* Na Slici 15 prikazano je obaveštenje o nedostajućem indeksu za zadati upit, a na Slici 16 prikazane su grupe nedostajućeg indeksa.



Slika 15 - Prikaz informacije o nedostajućem indeksu za prikazani upit

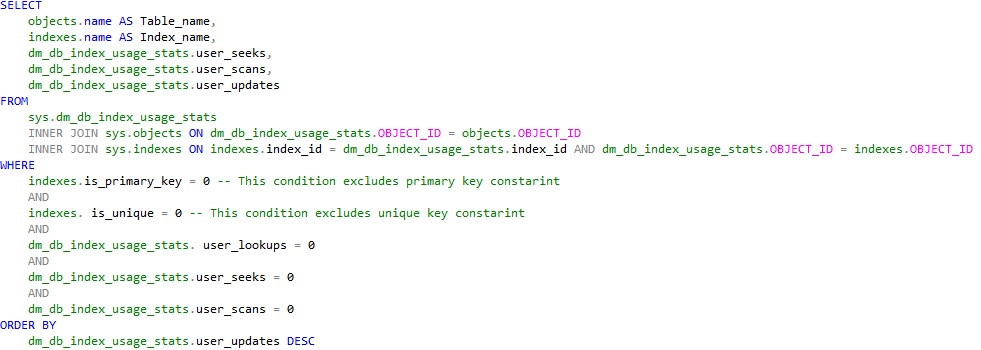


Slika 16 - Prikaz nedostajućih grupa indeksa

## **5.3. Pronalaženje nekorišćenih indeksa**

Indeksi u okviru SQL Servera su moćan alat za poboljšanje performansi SELECT upita, međutim, istovremeno oni imaju negativan efekat na ažuriranje podataka. INSERT, UPDATE i DELETE operacije uzrokuju ažuriranje indeksa i samim tim dupliranje podataka koji već postoje u okviru tabele. Kao rezultat, to uzrokuje da trajanje transakcija i izvršenje samih upita bude duže, što može dovesti do zaključavanja, blokiranja i veoma često do *timeout*-a. Kod velikih baza podataka ili tabela, suvišni indeksi utiču na slobodan prostor u skladištu. Pored kreiranja nedostajućih indeksa, bitna stvar za optimizaciju baze podataka jeste uklanjanje suvišnih indeksa, indeksa koji se ne koriste.

U okviru *sys.dm\_db\_index\_usage\_stats,* *Dynamic Management View*-a, skladištene su informacije o broju pretraživanja, skeniranja, pregledanja i ažuriranja od strane kako korisničkih, tako i sistemskih upita, uključujući poslednji put kada je svaka od pomenutih operacija bila izvedena. Nakon što se SQL Server restartuje, podaci se brišu. Na Slici 17 nalazi se upit pomoću kojeg je moguće utvrditi koji indeksi se ne koriste.



Slika 17 - Upit za pronalaženje indeksa koji se ne koriste

Kolona *user\_updates* broji kada je indeks bio ažuriran u situacijama kada je aplikacija unosila promene nad podacima, tako da je i indeks trebao biti ažuriran. Kolone *user\_seeks* i *user\_scans* čuvaju brojeve pretraživanja i skeniranja posmatranog indeksa, a od interesa su situacije u kojima su vrednosti ovih kolona jednake 0. Na Slici 18 prikazan je deo rezultata, indeks *IX\_Customer\_Username* koji nije skoro korišćen.



Slika 18 - Prikaz dela rezultata nekorišćenih indeksa

## **5.3.1. Koje nekorišćene indekse treba ukloniti?**

**Jedinstvena ograničenja** - Prilikom uklanjanja indeksa, potrebno je biti oprezan jer postoje situacije kada je indeks naveden kao nekorišćen, ali može nametati jedinstveno ograničenje. Prema tome, postoji mogućnost da optimizator koristi taj indeks i uklanjanje jedinstvenog indeksa ili ograničenja mora se obaviti sa najvećom pažnjom.

**Korišćenje statistike** - Treba biti oprezan i sa situacijama u kojim optimizator upita koristi statistiku koja je povezana sa izdvojenim nekorišćenim indeksom, čak i u situacijama kada konačni plan izvršenja ne uključuje pristup tom indeksu. Estimacija kardinalnosti, učitavanje kandidata za statistiku i konačno kreiranje plana izvršenja upita su potpuno nezavisne akcije.

Navedeni su samo neki od problema koji mogu nastati prilikom uklanjanja indeksa. Uzimajući navedene situacije u obzir, pre svakog uklanjanja indeksa potrebno je isplanirati adekvatno testiranje sa planom oporavka u slučaju da nešto pođe po zlu. Takođe, broj nekorišćenih indeksa ne mora isključivo ukazivati na problem, ali ukoliko taj broj raste s vremenom ili kada postoji nagli rast broja nekorišćenih indeksa, to može ukazivati na problem i zahtevati detaljnije analize a nakon toga i uklanjanje nekorišćenih indeksa.

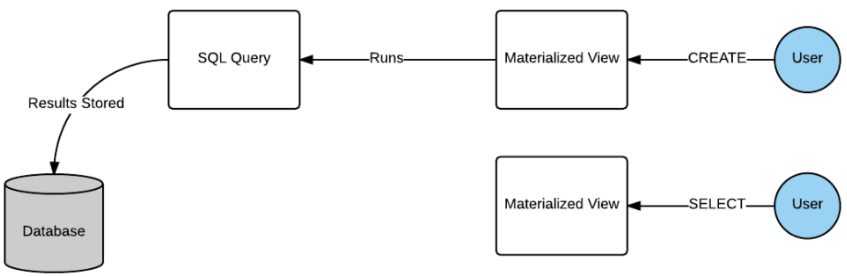
# 

# 

# **6. Materijalizovani pogledi**

Pogled (eng. *view*) predstavlja SQL upite skladištene u bazi podataka. Podaci se ne skladište zajedno sa upitom. Svaki put kada se pristupa pogledu, pokreće se skladišteni upit nad navedenim tabelama. Glavna prednost upotrebe pogleda je jednostavnost napisanih upita. Kada imamo komplikovani upit ili logiku koja se koristi na više mesta, pogled se može koristiti za skladištenje te logike. Sa druge strane, postoje situacije u kojima je korisno imati podatke skladištene zajedno sa pogledom. Situacije u kojima je to pogodno su kada postoje upiti koji vrše spajanje više tabela i izvode funkcije za agregiranje i filtriranje podataka. Upiti mogu postati skupi i trošiti dosta vremena na izvršenje.

Materijalizovani pogledi skladište podatke koji su dostupni kada se pristupi materijalizovanom pogledu. Materijalizovani pogled ne izvršava upit kojim je napravljen, već vraća podatke pa je zato veoma sličan samim tabelama. Podaci u okviru materijalizovanog pogleda se popunjavaju prilikom njegovog kreiranja (Slika 19).



Slika 19 - Način kreiranja materijalizovanog pogleda

## **6.1. Prednosti korišćenja materijalizovanih pogleda**

Postoji više prednosti korišćenja materijalizovanih pogleda u okviru baze podataka. Neki od njih su:

1. **Poboljšana efikasnost upita** - Glavna prednost korišćenja materijalizovanih pogleda je poboljšavanje performansi upita. Ukoliko se određeni upit izvršava duže nego što je prihvatljivo, uzrok tome može biti to što zahteva dosta transformacija, pod-upita i funkcija za dobijanje željenih podataka. Svaki korisnik, ili aplikacija, koja zahteva te podatke, može pristupiti materijalizovanom pogledu, budući da su tu svi podaci u okviru jedne tabele, bez potrebe za izvršavanjem skupih upita.
2. **Računanja u okviru materijalizovanog pogleda -** Računanja takođe mogu biti deo materijalizovanih pogleda za bilo koja polja za koja je to potrebno. To štedi vreme i pruža mogućnost za dobijanje informacija koje najčešće nisu skladištene u samoj bazi podataka.
3. **Jednostavniji upiti** - Može se koristiti za pojednostavljanje upita. Ukoliko upit koristi dosta logike, njegova upotreba može ukloniti deo logike i skladištiti je u materijalizovanom pogledu.

## **6.2. Nedostaci korišćenja materijalizovanih pogleda**

Pored navedenih prednosti, materijalizovani pogledi imaju i određene nedostatke. Neki od nedostataka su:

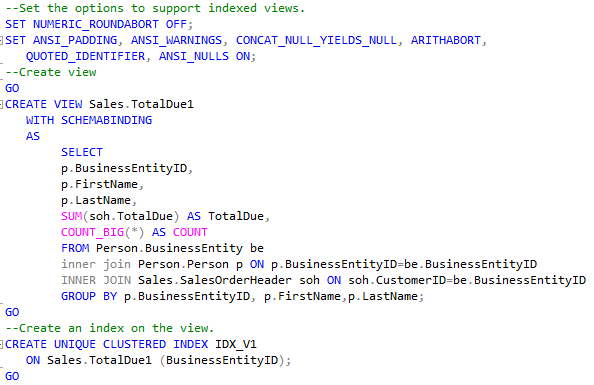
1. **Ažuriranje podataka mora biti definisano**- Glavna mana korišćenja materijalizovanih pogleda je ta što se skladišteni podaci povremeno moraju ažurirati. Budući da se tabele, na osnovu kojih je materijalizovani pogled popunjen, ažuriraju brisanjem, dodavanjem i ažuriranjem sadržaja, podatke u materijalizovanom pogledu takođe treba ažurirati. Preporučen način za ažuriranje jeste automatizovati sam proces.
2. **Nekonzistentnost podataka** - Podaci mogu biti nekonzistentni sa podacima koji se nalaze u tabelama iz kojih su vrednosti preuzete kako bi se materijalizovani pogled popunio. U situacijama kao što su mesečni ili drugi izveštaji iz prošlosti, ova situacija najčešće ne predstavlja problem.

## **6.3. Kreiranje materijalizovanih pogleda**

U okviru MS SQL Server baze podataka, materijalizovani pogledi se nazivaju indeksirani pogledi. Da bi bilo moguće kreirati indeksirani pogled, potrebno je:

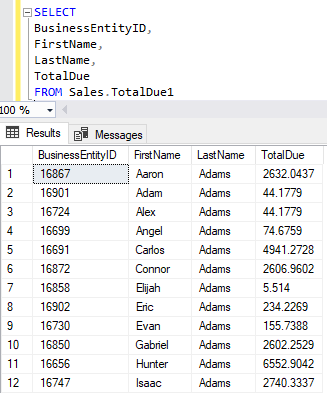
* Kreirati regularni pogled
* Kreirati klasterizovani indeks nad tim pogledom

Kreiranje indeksiranog pogleda prikazano je na Slici 20. Najpre je izvršeno kreiranje pogleda izdvajanjem podataka kao što su *BusinessEntityID, FirstName, LastName* i suma ukupnog duga, *TotalDue.* Nakon toga izvršeno je kreiranje klasterizovanog indeksa nad kolonom *BusinessEntityID.* Navedeni pogled se koristi za prikaz ukupne sume određenog naloga za prodaju, zajedno sa osnovnim informacijama o kupcu.



Slika 20 - Kreiranje indeksiranog pogleda

Nakon kreiranja indeksiranog pogleda, podacima je moguće pristupiti na sledeći način (Slika 21):



Slika 21 - Prikaz dela podataka u okviru kreiranog indeksiranog pogleda

Kreirani indeksirani pogled može se koristiti u situacijama kada postoji potreba za učestalim pristupom statistici koja sadrži imena kupaca i sumu cene naloga za prodaju.

# Zaključak

U ovom radu je najpre predstavljen način rada optimizatora i njegove glavne komponente. Detaljnije su opisani koraci pri samom procesiranju SQL upita a to su: parsiranje i vezivanje, optimizacija upita, izvršenje upita i keširanje planova. Takođe, prikazane su faze optimizacije upita: simplifikacija, optimizacija trivijalnih planova i puna optimizacija. Budući da se SQL Server optimizator upita bazira na ceni, predstavljen je način na koji se vrši odabir najboljeg plana. SQL Server optimizator najpre vrši analizu planova kandidata i upoređuje njihove cene. Nakon što nađe najpovoljniju cenu, vrši odabir tog plana. Za samo formiranje planova izvršenja, koriste se informacije koje su dostupne u okviru sistemskog kataloga. U radu su predstavljeni histogrami i formiranje gustine, kao i procena cene izvršenja plana korišćenjem estimacije veličine rezultata. Estimacija cene upita zasniva se na računanju cene za svaki operator zasebno, nakon čega se vrši sabiranje cena svih operatora i konačan rezultat predstavlja ukupnu cenu. Indeksi predstavljaju jedan od bitnijih faktora za optimizaciju upita. Oni ubrzavaju izvršenje upita brzim nalaženjem zapisa, bez potrebe za skeniranjem celokupne tabele. U radu je kroz primere prikazan način pronalaženja nedostajućih indeksa, ali i način pronalaženja suvišnih indeksa koji mogu negativno uticati na performanse. Pored indeksa, bitan faktor optimizacije su materijalizovani pogledi koji uz SQL upit kojim su formirani, skladište i podatke, tako da je moguće direktno pristupiti podacima koji su rezultat spajanja više tabela, bez potrebe za izvršavanjem upita. Takođe, iznete su prednosti i mane kreiranja materijalizovanih pogleda kao i praktičan prikaz kreiranja.

Efikasnost sistema u mnogome zavisi od performansi koje sistem pruža. Optimizacija upita predstavlja jako bitan aspekt svakog DBMS-a. Kroz rad je prikazan način rada optimizatora upita u MS SQL Serveru i može se zaključiti da on pruža dosta moćnih tehnika i alata za optimizaciju i obradu upita.

# Reference

[1] Benjamin Navarez - The SQL Server Query Optimizer <https://www.red-gate.com/simple-talk/sql/sql-training/the-sql-server-query-optimizer/>

[2] Tutorial Ride - Database Decomposition

<https://www.tutorialride.com/dbms/database-decomposition.htm>

[3] Benjamin Navarez - Inside the SQL Server Query Optimizer

<http://assets.red-gate.com/community/books/inside-the-sql-server-query-optimizer.pdf>

[4] Microsoft Documentation - Execution Plans

<https://docs.microsoft.com/en-us/sql/relational-databases/performance/execution-plans?view=sql-server-2017>

[5] Nikola Dimitrijević - How to identify and monitor unused indexes in SQL Server

<https://www.sqlshack.com/how-to-identify-and-monitor-unused-indexes-in-sql-server/>

[6] Ben Brumm - SQL Views and Materialized Views: The Complete Guide

<https://www.databasestar.com/sql-views/>

[7] Solar Winds - How the SQL Server Query Optimizer Works

<https://logicalread.com/2015/10/30/sql-server-query-optimizer-mc11/#.XpNas8gzaUn>