Univerzitet u Nišu,

Elektronski fakultet Niš

Logo

Description automatically generated

*Seminarski rad*

Sistemi za upravljanje bazama podataka

**OPTIMIZACIJA UPITA KOD MICROSOFT SQL SERVER-A**

Mentor: Student:

Aleksandar Stanimirović Ivan Milojković 1687

Niš, septembar 2024. Godine

**Sadržaj**

# 

[Uvod 3](#_Toc38129635)

[1. Optimizator upita 4](#_Toc38129636)

[2. Optimizacije upita 5](#_Toc38129637)

[3. Planovi izvršenja 8](#_Toc38129644)

[4. Cena izvršenja plana 10](#_Toc38129646)

[5. Selekcija indeksa 17](#_Toc38129655)

[6. Materijalizovani pogledi 26](#_Toc38129660)

7.[Zaključak 29](#_Toc38129664)

8.[Literatura 30](#_Toc38129665)

# **Uvod**

SQL je jedan od najmoćnijih alata za rad sa podacima, ali loše optimizovani upiti mogu izazvati prekomernu potrošnju resursa, što može negativno uticati na performanse i dovesti do prekida usluga za druge korisnike, pogotovo ako upiti sadrže greške. Poboljšanje neefikasnih upita i rešavanje problema sa performansama zahteva mnogo truda, ali je ključno za uspeh aplikacija i njihov optimalan rad. Ovaj proces obuhvata analizu i testiranje različitih rešenja, uključujući pregled planova izvršenja, statistika, proširenih događaja i drugih parametara. Prvi korak je identifikacija upita koji najviše opterećuju sistem, kako bi se radilo na njihovoj optimizaciji. To se može postići korišćenjem SQL Profiler-a za otkrivanje upita koji zauzimaju najviše vremena ili troše najviše resursa.

Optimizacija upita je složen aspekt sistema za upravljanje bazama podataka. Ona podrazumeva razmatranje više mogućih planova izvršenja koji mogu biti korišćeni za određeni upit. Na osnovu procenjene cene svakog mogućeg plana, sistem odabira onaj plan koji ima najnižu procenjenu cenu kako bi se postigla optimalna izvedba upita.

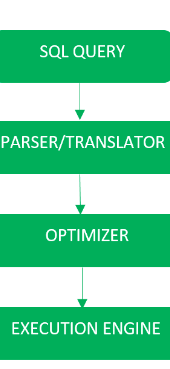
Ovaj rad istražuje funkcionalnost optimizatora upita, uključujući način na koji se procenjuju troškovi različitih planova izvršenja i faktore koji utiču na tu procenu. Takođe, obrađuje se izbor indeksa i metode za otkrivanje nedostajućih i neiskorišćenih indeksa, što može značajno doprineti poboljšanju performansi sistema. Materijalizovani pogledi su predstavljeni kao strategija za poboljšanje upita, uz analizu njihove upotrebe i razmatranje njihovih prednosti i nedostataka. Detaljno su opisane faze kroz koje se prolazi tokom optimizacije upita. Microsoft SQL Server DBMS je korišćen kao platforma, dok je za praktične primere korišćena baza podataka AdventureWorks 2017, dostupna kroz Microsoft dokumentaciju.

# 

# **Optimizator upita**

SQL Server se sastoji od dve ključne komponente: sistema za skladištenje podataka i procesora upita. Sistem za skladištenje podataka upravlja premeštanjem podataka između diska i memorije, pri čemu nastoji da optimizuje konkurentnost i očuva integritet podataka. Procesor upita, s druge strane, prima sve upite koji se šalju na SQL Server, analizira ih kako bi izabrao najefikasniji plan izvršenja, implementira taj plan i vraća potrebne rezultate.

Za kreiranje upita koristi se SQL deklarativni jezik, kao i T-SQL, koji je proširenje SQL jezika. Pošto je SQL jezik visokog nivoa, omogućava korisnicima da definišu koje podatke žele da preuzmu iz baze, ali ne i specifične korake ili algoritme potrebne za procesuiranje tih podataka. Kada upit stigne u SQL Server, prvi korak je parsiranje. **Parser** je komponenta koja analizira sintaksu SQL upita, proverava njegovu ispravnost i strukturu prema pravilima jezika. Ukoliko postoje greške u pisanju upita (kao što su nedostajući zarezi, pogrešni nazivi tabela ili kolona), parser ih detektuje i prijavljuje. Nakon što parser potvrdi da je upit ispravan, dolazi do faze **prevođenja**. Upit se prevodi iz SQL jezika u interni reprezentativni format, koji je razumljiv sistemu za upravljanje bazama podataka. Ovaj interni format je prilagođen za dalju obradu u sledećim fazama procesa. Optimizator je ključna komponenta u obradi upita. Njegov zadatak je da pronađe najbolji plan za izvršenje upita, tj. način koji će omogućiti najefikasniji pristup i preuzimanje podataka iz baze. Optimizator razmatra različite strategije (ili planove) za izvršenje upita, kao što su korišćenje indeksa, redosled pridruživanja tabela i strategije filtriranja podataka. Za svaku opciju, on procenjuje koliko će resursa biti potrebno (vreme, memorija, procesorska snaga) i bira plan sa najmanjim troškovima (najefikasniji). Nakon što optimizator odabere najbolji plan, upit prelazi u fazu **izvršenja**. Plan koji je kreirao optimizator se prosleđuje **sistemu za izvršenje upita**. Ova komponenta prati korake definisane u planu, pristupa podacima na diskovima ili u memoriji, i izvršava potrebne operacije kako bi vratio tražene rezultate korisniku. Plan može biti sačuvan u memoriji ili u kešu. Keširanjem se štedi na vremenu jer se onda prethodno navedeni koraci ne moraju obavljati. Kako bi došao do optimalnog plana, optimizator prolazi kroz nekoliko faza, koje su prikazane na Slici 1.



Slika 1 - Koraci pri procesiranju SQL upita

# **Optimizacije upita**

Optimizator upita ima nekoliko faza optimizacije koje su razvijene na taj nacin da pokušavaju da optimizuju upite što brže i jednostavnije, izbegavajuci da koriste skuplje osim ako to nije neophodno. Faze optimizacije su:

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

Sem, u fazi simplifikacije, plan izvršenja može biti generisan u bilo kojoj od ostalih faza.

## **Faze optimizacije upita**

## 

## **Simplifikacija**

Tokom faze simplifikacije, upit se preoblikuje, odnosno stablo upita se prepravlja tako da se smanji na jednostavniju strukturu. Cilj je da se proces optimizacije učini lakšim. U ovom koraku dolazi do pojednostavljivanja strukture upita.

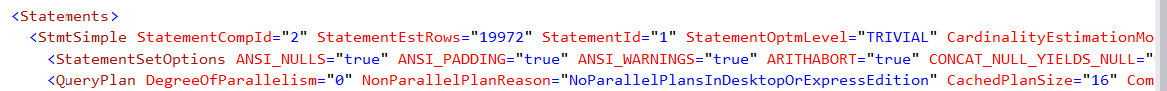
* Ugnježdeni upiti se konvertuju u spojeve, ali pošto nije uvek moguće prevesti ugnježdeni upit u unutrašnji spoj, spoljašnji spoj i operacije grupisanja se mogu dodati po potrebi.
* Redundantni unutrašnji i spoljni spojevi se eliminišu. Tipičan primer je eliminacija *Foreign Key Join*-a koja se javlja kada SQL Server detektuje da neki spoj nije potreban, zato što postoji ograničenje stranog ključa i samo kolone referencirajuće tabele su zahtevane, ili se koristi logički izraz koji uvek daje isti rezultat (poput WHERE 1 = 1), on se uklanja jer ne doprinosi krajnjem rezultatu.
* Filteri u WHERE klauzuli se guraju niz stablo upita da bi se omogućilo rano filtriranje podataka, i potencijalno bolje poklapanje sa indeksima.
* Uklanjanje suvišnih upita, ukoliko postoji situacija u kojoj se podaci više puta filtriraju ili sortira već sortirani set podataka, simplifikacija će eliminisati takve redundantne korake. Na primer, ako su podaci već poredani prema nekoj koloni, a upit uključuje novo sortiranje po istoj koloni, ta naredba se može ignorisati.

## **Trivijalni plan**

Sam proces optimizacije može biti zahtevniji za pokretanje i izvršavanje jednostavnih upita koji ne traže procenu troškova. Da bi se izbegla ova skupa operacija kod jednostavnih upita, SQL Server primenjuje optimizaciju trivijalnih planova (kao što je prikazano na Slici 3). Ako postoji samo jedan moguć način za izvršenje upita, na osnovu strukture upita i dostupnih metapodataka, značajan deo posla može biti zaobiđen.



Slika 2. Upit za selektovanje podataka iz tabele Person



Slika 3. Prikaz dela trivijalnog plana za izvršenje jednostavnog upita

Osim trivijalnog plana, postoji i druga mogućnost optimizacije nazvana FULL. Ova opcija podrazumeva da je izvršena potpuna optimizacija upita jer se on nije kvalifikovao za trivijalni plan. Potpuna optimizacija je primenjena na složenije upite ili one koji uključuju kompleksne operacije, gde je neophodno uporediti različite planove izvršenja kako bi se izabrao najefikasniji.

## **Puna optimizacija**

Ako se upit ne kvalifikuje za trivijalni plan, SQL Server pokreće optimizaciju zasnovanu na ceni, koja koristi skup transformacionih pravila za kreiranje alternativnih planova. Ovi alternativni planovi se smeštaju u strukturu zvanu memo, a na osnovu procene troškova bira se najoptimalniji plan. Potpuna optimizacija se odvija kroz tri faze, pri čemu svaka faza primenjuje različita pravila transformacije. Zbog mogućnosti generisanja velikog broja alternativnih planova, SQL Server koristi heuristički pristup kako bi smanjio broj opcija. Proces može završiti odmah ako se pronađe dovoljno efikasan plan, u skladu s unutrašnjim pragom poznatim optimizatoru. Ako je plan na kraju svake faze previše skup, optimizator prelazi na sledeću fazu, gde se primenjuju složenija pravila transformacije. Faze su:

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

U prvoj fazi, slično konceptu trivijalnih planova, cilj je što brže pronaći odgovarajući plan, bez primene naprednih transformacija. Ova faza, poznata i kao **Search 0** ili **Transaction Processing Phase**, idealna je za manje upite tipične za transakcione sisteme i primenjuje se na upite sa najmanje tri tabele. Pre nego što započne proces optimizacije, optimizator stvara početni niz spojeva koristeći heuristiku. Heuristički pristup počinje tako što se prvo spajaju najmanje tabele ili tabele koje imaju najvišu stopu filtriranja zasnovanu na selektivnosti. Tokom ove faze, razmatraju se isključivo takvi redosledi spojeva. Na kraju faze, optimizator upoređuje najefikasniji plan sa internim pragom troškova, a ukoliko plan i dalje bude ocenjen kao preskup, SQL Server prelazi na sledeću fazu optimizacije..

Faza **Search 1**, poznata i kao **Quick Plan**, primenjuje dodatna transformaciona pravila i ograničeno menja redosled spojeva, što je pogodno za složenije upite. Na kraju ove faze, SQL Server upoređuje troškove najjeftinijeg plana sa internim pragom troškova, i ako je plan dovoljno ekonomičan, bira se taj plan za izvršenje. Ako je upit i dalje ocenjen kao skup, a sistem podržava paralelno izvršavanje, faza se ponavlja kako bi se pronašao optimalan paralelni plan, ali u ovoj fazi nijedan plan još nije izabran za izvršenje. Nakon toga, upoređuju se troškovi najboljih serijskih i paralelnih planova, i najisplativiji plan se koristi u narednoj fazi, **Search 2**.

Završna faza, poznata kao **Search 2**, naziva se i **puna optimizacija** i koristi se za upite koji variraju od složenih do izrazito kompleksnih. U ovoj fazi razmatra se širi spektar transformacionih pravila, uključujući paralelne operatore i napredne strategije optimizacije. S obzirom na to da je ovo poslednja faza, mora se pronaći odgovarajući plan za izvršenje. Proces optimizacije u ovoj fazi uključuje i koncept timeout-a, gde tajmer nije unapred određen, već se izračunava na osnovu broja izvršenih transformacija i proteklog vremena. Kada optimizator dostigne timeout, prekida proces i vraća najjeftiniji plan koji je pronađen do tog trenutka.

## 

# 

# **Planovi izvršenja**

Kao što je ranije navedeno, svrha optimizatora upita je da pronađe najučinkovitiji plan za izvršenje datog upita. Čak i jednostavni upiti mogu imati mnogo različitih pristupa za dobijanje željenih podataka i ostvarivanje istih rezultata. Cilj je izabrati najbolji plan među svim mogućim opcijama.

Za svaki upit, formira se prostor traženja koji sadrži različite potencijalne planove. Teoretski, da bi se pronašao najbolji mogući plan, trebalo bi generisati sve moguće varijante. Međutim, to nije izvodljivo zbog performansi, jer složeniji upiti mogu rezultirati velikim brojem potencijalnih planova, što može značajno usporiti sistem. Na primer, ako optimizator već pronađe adekvatan plan, nije razumno trošiti dodatno vreme tražeći savršeno rešenje, jer bi to moglo ponovo pokrenuti problem sa početka. Bitno je pronaći dovoljno dobar plan umesto savršenog.

Za istraživanje prostora traženja, optimizator upita koristi pravila transformacije i heurističke pristupe. Heuristika pomaže u ograničavanju broja potencijalnih rešenja kako bi se postiglo optimalno vreme izvršenja upita.

Plan izvršenja upita predstavlja skup sledećih stvari:

1. **Sekvenca pristupa tabelama** - Obično, za dobijanje rezultata, server baze podataka može pristupati tabelama na različite načine. Na primer, ako **SELECT** upit uključuje tri tabele, **TableA**, **TableB** i **TableC**, server može koristiti različite redose za pristup tim tabelama, a da pritom dobije iste rezultate.
2. **Metode koje se koriste za izdvajanje podataka iz svake tabele**: Postoje različiti načini za pristup podacima u tabelama. Ako je potrebno samo nekoliko redova sa specifičnim ključem, server baze podataka može koristiti indeks za brzo pretraživanje. Kada su potrebni svi redovi iz tabele, server može zanemariti indeks i izvršiti skeniranje cele tabele. U slučajevima kada su svi zapisi iz tabele potrebni, ali postoji indeks koji uključuje kolone korišćene u **ORDER BY** klauzuli, skeniranje indeksa može biti efikasnije nego skeniranje tabele, jer može očuvati redosled rezultata. Kada je tabela vrlo mala, skeniranje tabele je obično najefikasnija metoda za pristup podacima.
3. **Metode koje se koriste za izvršenje računanja i filtriranja, agregiranje i sortiranje podataka iz svake tabele**: Postoje različite tehnike za obradu podataka, uključujući izračunavanje skalarnih vrednosti, agregiranje i sortiranje podataka prema upitu koristeći klauzule kao što su **GROUP BY** i **ORDER BY**, kao i filtriranje podataka pomoću **WHERE** ili **HAVING** klauzula.

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

1. **Estimirani plan izvršenja**: Prikazuje plan izvršenja koji SQL Server procenjuje pre nego što stvarno izvrši upit. Ovo omogućava uvid u plan koji bi SQL Server koristio bez pokretanja upita.
2. **Stvarni plan izvršenja**: Prikazuje plan izvršenja koji je stvarno korišćen tokom izvršavanja upita. Ovo uključuje detalje o stvarnom toku izvršenja i sve resurse koji su korišćeni.
3. **Statistika upita**: Ona uključuje i *runtime* informacije tokom samog procesa izvršenja, kao što je broj redova koji prolaze kroz operatore.

## **Dekompozicija upita**

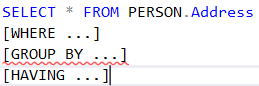
Dekompozicija je proces razlaganja upita na manje komponente ili delove, poznate kao blokovi. Ovaj proces zamenjuje jednu relaciju sa više manjih relacija. Ključne karakteristike dekompozicije uključuju:

1) **Bez gubitaka** - Dekompozicija mora očuvati sve informacije iz originalne relacije, što znači da nijedna informacija ne sme biti izgubljena tokom razbijanja na manje relacije.

2) **Očuvanje zavisnosti** - Važno je da sve zavisnosti u bazi podataka budu zadovoljene, bez narušavanja ograničenja koja su bila postavljena na originalnoj relaciji.

3) **Bez redundancije** - Dekompozicija ne sme dovesti do nepotrebnih duplikata podataka. Da bi se izbegla redundancija i problemi sa podacima, primenjuje se proces normalizacije.

Optimizator upita obrađuje jedan blok po jednom. Blok se definiše kao jednostavan upit sa jednom **SELECT** i jednom **FROM** klauzulom, uz mogućnost da sadrži samo jednu **WHERE**, **GROUP BY** ili **HAVING** klauzulu. Na slici 4. je prikazan primer kako izgleda jedan takav SQL upit blok, pri čemu su elementi u uglastim zagradama opcioni.



Slika 4. Primer bloka SQL upita

# **Cena izvršenja plana**

Pretraga i kreiranje kandidatskih planova je samo deo procesa optimizacije. Nakon što se planovi generišu, optimizator upita mora proceniti troškove svakog plana i izabrati onaj sa najmanjim troškom. Da bi izvršio procenu troškova, optimizator analizira cenu svakog fizičkog operatora unutar plana, koristeći formule koje uzimaju u obzir korišćenje resursa poput I/O-a, CPU-a i memorije. Procena troškova zavisi od algoritma koji koristi fizički operator, kao i od procenjenog broja zapisa koji će biti obrađeni. Ova procena broja zapisa se naziva procena kardinalnosti.

Da bi se procena kardinalnosti obavila, SQL Server koristi i održava statistiku optimizatora, koja sadrži informacije o distribuciji vrednosti u jednoj ili više kolona tabele. Nakon što se cene svih operatora procene, optimizator sabira sve troškove kako bi dobio ukupnu procenu troška za ceo plan.

## 

## **Veličina rezultata**

Izlaz jednog operatora često služi kao ulaz za drugi operator, a trošak jednog operatora zavisi od veličine njegovih ulaznih podataka. Da bi se procenila veličina rezultata, koristi se redukcioni faktor dodeljen svakom uslovu u **WHERE** klauzuli.

Redukcioni faktor se odnosi na odnos između veličine ulaznih podataka i veličine rezultata, uzimajući u obzir samo selektivnost određenog uslova. Veličina ukupnog rezultata može se izračunati kao proizvod redukcionih faktora za svaki pojedinačni uslov. Postoji nekoliko metoda za izračunavanje ovih redukcionih faktora, koje se oslanjaju na podatke dostupne u sistemskom katalogu. Evo primera metoda i situacija za izračunavanje redukcionih faktora.

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.

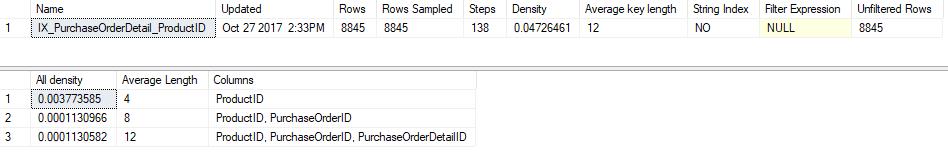
## **Gustina**

Gustina se određuje kao recipročna vrednost broja jedinstvenih vrednosti. Ove informacije o gustini mogu poboljšati procene za operacije **GROUP BY** i za predikate jednakosti kada su vrednosti nepoznate, kao što je slučaj sa lokalnim promenljivama. Gustina se računa za svaki par kolona, stvarajući time prefiks za te kolone u statističkom objektu. Na slici 3 je prikazana komanda koja je korišćena za pregled statistike određenog indeksa



Slika 5. Prikaz statistike navedenog indeksa

Pomenuta komanda pruža uvid u gustinu vektora, prikazujući gustine za kolone u `Columns` koloni. To uključuje `ProductID`, kao i kombinacije kolona kao što su `ProductID`, `PurchaseOrderID` i `ProductID`, `PurchaseOrderID`, `PurchaseOrderDetailID` (vidi Slika 4). Takođe, budući da se sve navedene kolone čuvaju kao `int` vrednosti, prosečna dužina za svaku od vrednosti za gustinu iznosi 4, 8 i 12 bajtova.



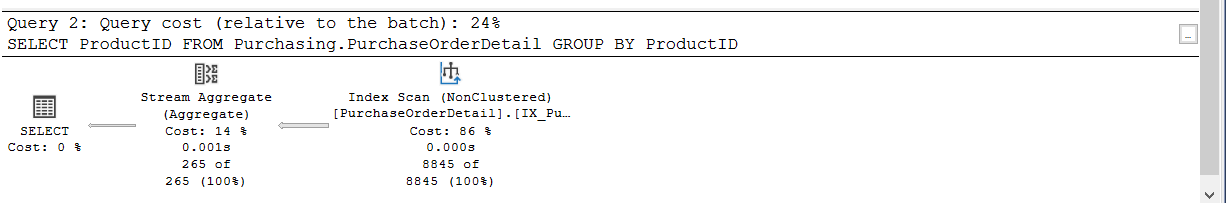
Slika 6. Rezultat komande za prikaz statistike navedenog indeksa

Optimizator upita može iskoristiti informacije o gustini za procenu kardinalnosti `GROUP BY` upita. Kada je dostupna informacija o gustini, procenjeni broj jedinstvenih vrednosti može se dobiti izračunavanjem recipročne vrednosti gustine, tj. 1\ Gustina.

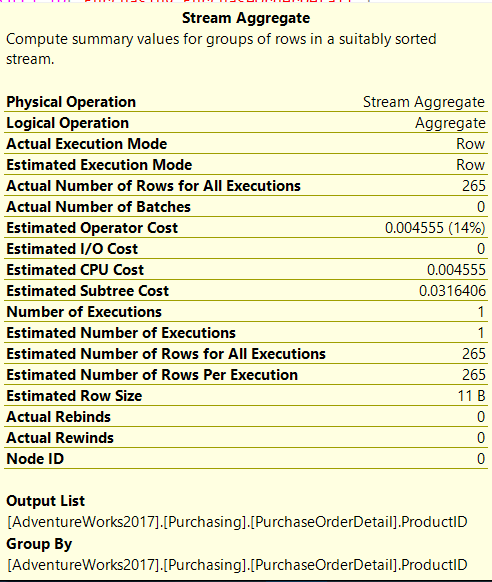
Za izračunavanje kardinalnosti upita prikazanog na Slici 7., koji koristi `GROUP BY` za `ProductID`, potrebno je izračunati recipročnu vrednost gustine za `ProductID`. U ovom slučaju, 1/0.003773585 daje broj 265, što predstavlja procenjeni broj redova. Ovaj broj je prikazan u planu izvršenja na Slici 6..



Slika 7. Upit koji sadrži GROUP BY klauzulu



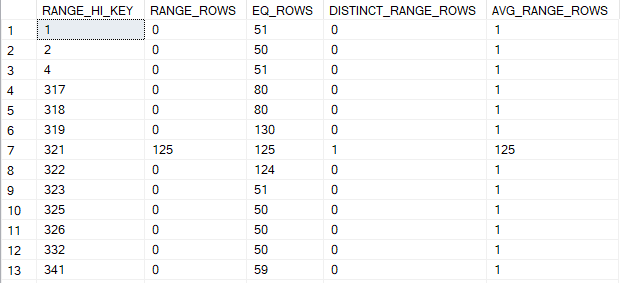
Slika 8.



Slike 8.,9. Prikaz izračunatog procenjenog broja redova na osnovu gustine

## **Histogram**

Mnogi sistemi za upravljanje bazama podataka koriste napredne statističke metode, poput histograma, za procenu redukcionih faktora. U SQL Serveru, histogrami su primarno korišćeni za prvu kolonu statističkog objekta i pružaju informacije o raspodeli vrednosti u toj koloni, raspodeljenoj u delove koji se nazivaju koraci ili korpe. Maksimalan broj koraka u histogramu je 200. Da bi se kreirao histogram, SQL Server identifikuje jedinstvene vrednosti u koloni i beleži najčešće korišćene vrednosti koristeći algoritam maxdiff. Ovaj algoritam pomaže u distribuciji vrednosti u relacionim bazama podataka i pruža ključne informacije za optimizaciju upita. Na slici 10. prikazan je deo rezultata koji se dobija upitom iz Slike 5.



Slika 10. Deo rezultata koji predstavljaju histogram

U histogramskim statistikama u SQL Serveru, nekoliko ključnih polja igra značajnu ulogu u proceni kardinalnosti upita i optimizaciji upita. Evo šta svako od tih polja predstavlja:

- **RANGE\_HI\_KEY** označava gornju granicu koraka histograma. Svaki zapis u histogramu definiše opseg vrednosti između prethodne vrednosti ovog polja i trenutne vrednosti u tom redu. Na primer, korak 4 može sadržavati vrednosti koje se kreću između 5 i 317.

- **RANGE\_ROWS** predstavlja procenjeni broj redova čije vrednosti kolona spadaju u opseg koraka, isključujući gornju granicu. Ovo je korisno za procenu koliko redova upit može obuhvatiti unutar određenog opsega.

- **EQ\_ROWS** označava procenjeni broj redova čije vrednosti kolona su tačno jednake vrednosti `RANGE\_HI\_KEY`. Ova vrednost pokazuje stvarni broj postojećih zapisa za određenu vrednost, kao što je `ProductId` sa vrednošću 321.

- **DISTINCT\_RANGE\_ROWS** označava procenjeni broj redova sa jedinstvenim vrednostima kolona unutar opsega, isključujući gornju granicu. Ova metrika pomaže u proceni koliko različitih vrednosti postoji u određenom opsegu.

- **AVG\_RANGE\_ROWS** predstavlja prosečan broj redova po jedinstvenoj vrednosti, isključujući gornju granicu. Računa se kao količnik `RANGE\_ROWS` i `DISTINCT\_RANGE\_ROWS`. Ova vrednost pomaže u proceni gustine podataka unutar opsega koraka histograma.

Kombinovanjem ovih statistika, optimizator upita može bolje proceniti raspodelu podataka i, shodno tome, odabrati najefikasniji plan izvršenja za upit.

Histogramske statistike u bazama podataka koriste se za poboljšanje tačnosti procene kardinalnosti i optimizaciju upita. Postoje dve osnovne vrste histograma koje se koriste za ovo:

**Equiwidth histogram**: Ovaj tip histograma deli opseg vrednosti na pod-opsege jednake širine. Svaka korpa histograma pokriva isti interval vrednosti. Na primer, ako opseg vrednosti ide od 0 do 100, a histogram koristi deset korpi, svaki opseg će pokriti interval od 10 jedinica (0-10, 10-20, itd.). Ovaj pristup može biti jednostavan za implementaciju, ali može biti manje precizan u slučajevima kada distribucija podataka nije ravnomerna.

**Equidepth histogram**: Ovaj tip histograma deli opseg vrednosti na pod-opsege tako da svaka korpa sadrži otprilike isti broj redova ili torki. Na primer, ako imamo 1.000 redova i koristimo deset korpi, svaka korpa će sadržati približno 100 redova. Ovaj pristup omogućava bolju statistiku jer uzima u obzir raspodelu podataka, a ne samo opseg vrednosti. Equidepth histogrami često pružaju preciznije informacije o raspodeli podataka u poređenju sa equiwidth histogramima, zbog čega su češće u upotrebi u modernim sistemima za upravljanje bazama podataka (DBMS-ovima).

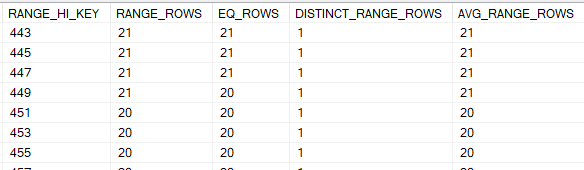
Generalno, equidepth histogrami nude bolje rezultate za procenu kardinalnosti i često se preferiraju zbog njihove sposobnosti da bolje odraze stvarnu raspodelu podataka.

Primer upotrebe kako bi se prikazala korisnost informacija koje su otkrivene pomoću histograma prikazan je pomoću upita na Slici 11.

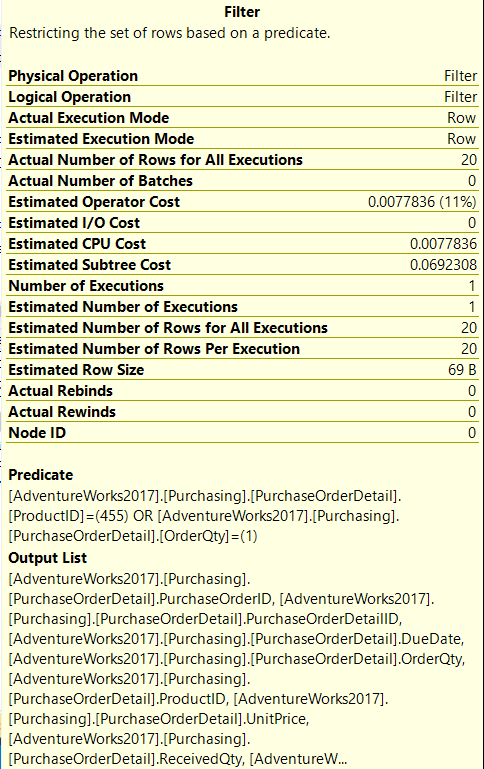


Slika 11. Upit za prikaz detalja narudžbine sa parametrom ProductID=455

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=455 + broj estimiranih redova za OrderQty = 1, ali ako postoje duplikati oni se uzimaju u obzir samo jednom. Estimirani broj redova za uslov ProductID=455 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 11.



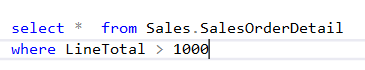
Slika 12.,13. - Prikaz broja estimiranih redova za upit sa Slike 11

## **Estimiranje cene upita**

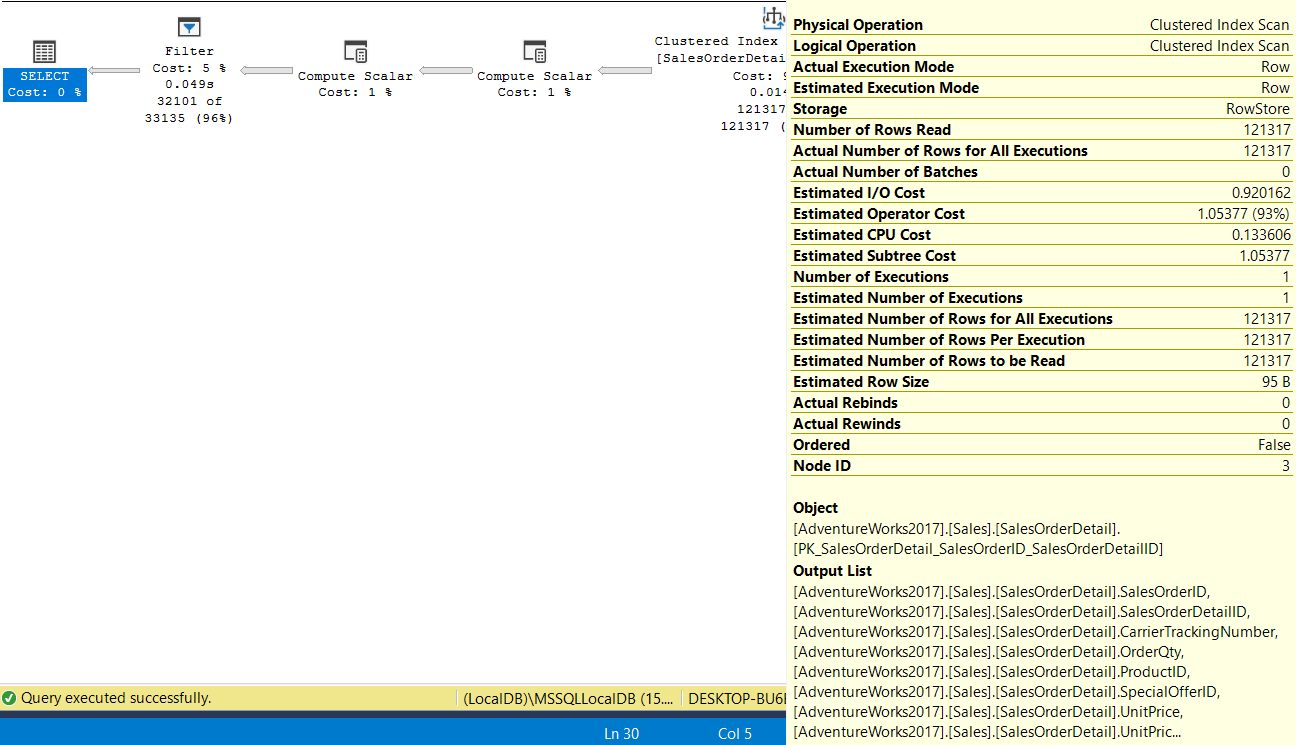
Tačnost plana izvršenja koju generiše optimizator upita značajno zavisi od preciznosti njegovih procena. Čak i kada optimizator izabere plan sa niskim troškovima, greške u procenama mogu dovesti do odabira manje efikasnih planova, što negativno utiče na performanse baze podataka. Procena treba biti precizna jer se koristi više puta tokom optimizacije.

Trošak se računa za svaki operator posebno, dok je ukupni trošak zbir troškova svih operatora u planu. Trošak svakog operatora zavisi od korišćenog algoritma i procenjenog broja rezultantnih zapisa.

Procena CPU troška se vrši za svakog operatora, dok neki operatori imaju i I/O troškove. Na primer, Clustered Index Scan uključuje oba tipa troškova, dok Stream Aggregate uzima u obzir samo CPU trošak. Ove procene nisu javno dostupne, budući da su zaštićene od strane kompanije Microsoft. Izvršavanjem upita prikazanih na slici 9 moguće je analizirati rezultate procenjenih I/O i CPU troškova.



Slika 14.



Slike 14.,15. - Upit za dobijanje detalja o narudžbini kada je ukupan broj linija veci od 1000

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.

# **Odabir indeksa**

Odabir indeksa predstavlja ključnu tehniku u optimizaciji upita u SQL Serveru. Pravilno odabran indeks može značajno poboljšati brzinu izvršavanja upita, što se direktno odražava na performanse aplikacije. Indeksi omogućavaju brže pronalaženje zapisa, čime se izbegava potreba za skeniranjem cele tabele i štede resursi i vreme.

Da bi optimizator upita mogao efikasno da koristi indeks, potrebno je da utvrdi njegovu korisnost u evaluaciji predikata. Ovo uključuje upoređivanje ključeva indeksa sa konstantama ili promenljivama iz upita. Takođe, važno je proveriti da li indeks pokriva sve kolone navedene u upitu, jer neklasterizovani indeksi često obuhvataju samo deo kolona iz tabele.

Kada jedan indeks ne pokriva sve potrebne kolone, moguće je koristiti kombinaciju više indeksa kako bi se obuhvatile sve tražene kolone. Ako kombinovanje indeksa nije moguće, može biti neophodan pristup osnovnoj tabeli, bilo putem klasterizovanog indeksa ili heap-a. Ova operacija, poznata kao Key Lookup, može biti skupa, naročito u slučaju nasumičnog I/O.

Iako SQL Server može koristiti jedan ili više indeksa, odluka o njihovom uključivanju u plan izvršenja zavisi od procene cene. Stoga, nakon kreiranja indeksa, važno je proveriti da li je on zaista korišćen i da li poboljšava performanse upita. Neiskorišćeni indeksi mogu zauzimati dragocen prostor na disku i usporavati operacije ažuriranja, ne pružajući stvarnu korist. Takođe, indeksi koji su nekada bili korisni mogu postati neefikasni ili neupotrebljivi zbog promena u upitima ili strukturi baze podataka.

## 

## **Načini selekcije indeksa**

Tokom operacije pretraživanja, SQL Server koristi B-stablo indeksa kako bi brzo pronašao željene zapise, bez potrebe za potpunim skeniranjem indeksa ili tabele. Ovaj proces može se uporediti sa sadržajem na kraju knjige koji omogućava brz pronalazak relevantne informacije bez čitanja cele knjige. Kada se pronađe prvi zapis, SQL Server može koristiti skeniranje na nivou listova indeksa, kretanjem unapred ili unazad, kako bi identifikovao dodatne zapise.

Operacije pretraživanja indeksa koriste različite predikate za filtriranje podataka. Ovi predikati uključuju:

- **Operatori jednakosti**: `=`, koji se koristi za precizno pronalaženje zapisa koji tačno odgovaraju vrednosti.

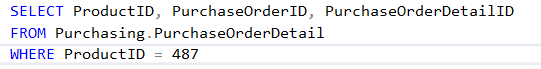
- **Operatori nejednakosti**: `<`, `>`, `<=`, `>=`, `<>`, `!=`, `!<`, `!>`, koji omogućavaju pronalaženje zapisa koji zadovoljavaju određene uslove odnosa između vrednosti.

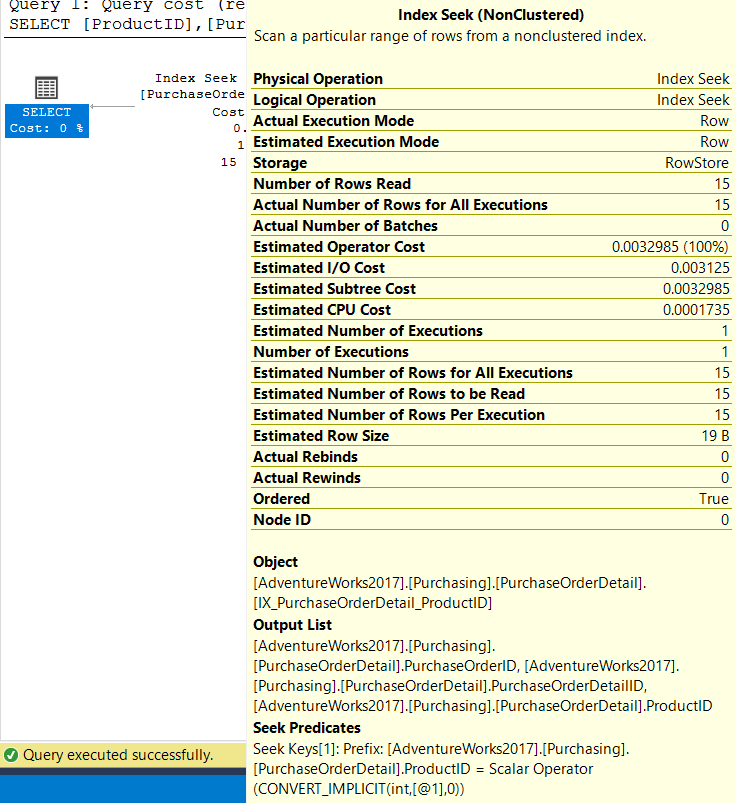
- **Rasponi i skupovi**: `BETWEEN` i `IN`, koji omogućavaju pronalaženje zapisa unutar određenog opsega ili u skupu specifičnih vrednosti.

Ovi predikati se primenjuju kada je kolona koja se koristi za pretraživanje navedena kao vodeći ključ indeksa. U takvim slučajevima, SQL Server može efikasno mapirati pretragu na indekse, čime se značajno poboljšava brzina izvršavanja upita i smanjuje potreba za skeniranjem celokupne tabele.

* ProductId=350
* UnitPrice<4.563
* FirstName=’Syed’
* LastName LIKE ‘Abbas’

Sledeći upit koristi operator pretraživanja indeksa:

  
Slika 16.



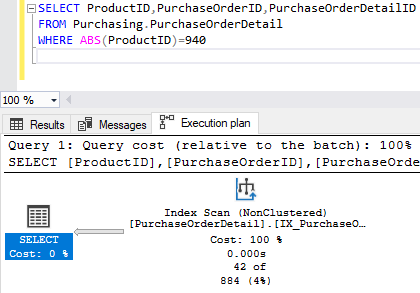
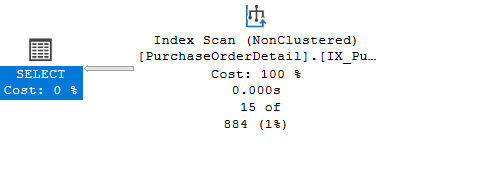
Slike 16.,17. - Prikaz predikata za *Index Seek* navedenog upita

*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)=487
* UnitPrice+1 < 1000.00
* LastName like ‘%Abbas’

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 18.

Slika 18. Plan izvršenja kada se kao predikat koristi ABS funkcija

U slučajevima kada indeks uključuje više kolona, SQL Server može koristiti taj indeks za pretragu prema drugoj koloni, pod određenim uslovima. Ako postoji predikat jednakosti nad prvom kolonom indeksa, SQL Server može efikasno koristiti indeks za pretragu i po drugoj koloni.

Pretpostavljajući da postoji indeks sa više kolona nad obe kolone u redosledu kojim su prikazane, SQL Server može primeniti ovaj indeks u sledećim scenarijima:

1) **Jednakost nad prvom kolonom i predikati nad drugom kolonom**: Ako je predikat jednakosti (`=`) primenjen na prvu kolonu indeksa, SQL Server može koristiti taj indeks za pretragu i nad drugom kolonom, čak i ako je predikat za drugu kolonu različit. Na primer, za indeks na kolone `(A, B)`, ako je predikat `A = 10` i `B > 20`, SQL Server može koristiti ovaj indeks za brzo pronalaženje zapisa koji zadovoljavaju oba uslova.

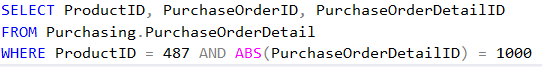
2) **Predikati nad prvim i drugim kolonom**: Ako su predikati primenjeni na obe kolone indeksa, SQL Server može koristiti indeks za obavljanje pretrage. Na primer, za indeks na kolone `(A, B)`, ako su predikati `A = 10` i `B = 20`, SQL Server može direktno koristiti ovaj indeks da pronađe tačno odgovarajuće zapise.

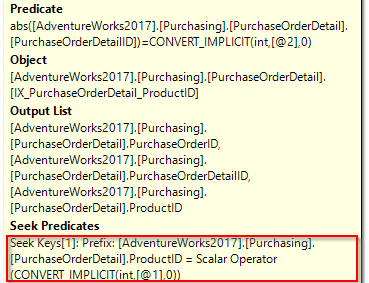
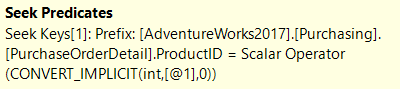
3) **Rasponi nad prvom kolonom i predikati nad drugom kolonom**: Kada je primenjen raspon predikata nad prvom kolonom (`BETWEEN`, `>=`, `<=`), a predikati nad drugom kolonom su različiti, SQL Server može koristiti indeks za pretragu u okviru tog raspona. Na primer, za indeks na kolone `(A, B)`, ako je predikat `A BETWEEN 10 AND 20` i `B = 30`, SQL Server može efikasno pronaći zapise koji zadovoljavaju oba uslova.

Korišćenje indeksa sa više kolona omogućava SQL Serveru da optimizuje pretragu i smanji vreme izvršavanja upita, tako što omogućava brzu navigaciju kroz vrednosti u indeksom.

* ProductID=487 AND PurchaseOrderID > 1000
* LastName=’Abbas’AND FirstName=’Syad’

Na Slici 19. 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 19 - Prikaz situacije kada je moguća pretraga indeksa samo po prvoj koloni u indeksu koji se sastoji iz više kolona

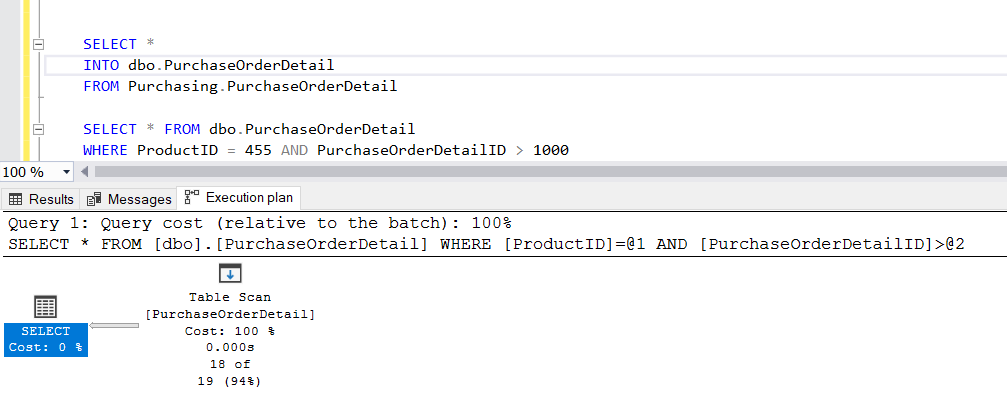
## 

## **Pronalazak nedostajućih indeksa**

SQL Server nudi mogućnost identifikacije korisnih nedostajućih indeksa za postojeće upite. Tokom optimizacije, optimizator upita može preporučiti indekse koji bi poboljšali performanse upita, a ako takvi indeksi nisu prisutni, obavestiće korisnika o tome. Ove preporuke se temelje na analizi trenutnih upita i njihovih planova izvršenja.

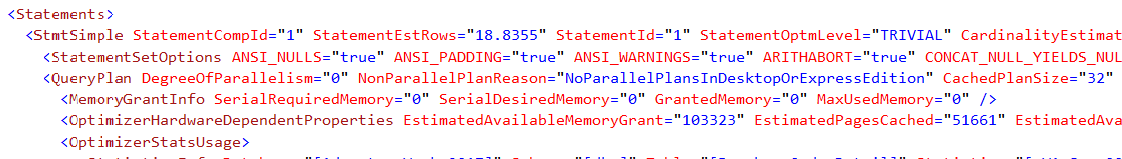
SQL Server prikuplja podatke o potrebnim indeksima od trenutka kada je instanca pokrenuta i čuva ih u `sys.dm\_db\_missing\_index` dinamičkom prikazu (DMV). Ovaj prikaz omogućava pregled informacija o indeksima koji se preporučuju za poboljšanje performansi. Korišćenjem ovih informacija, SQL Server upozorava na moguće nedostatke u trenutnim planovima izvršenja i sugeriše koji indeksi mogu doprineti poboljšanju 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 20.



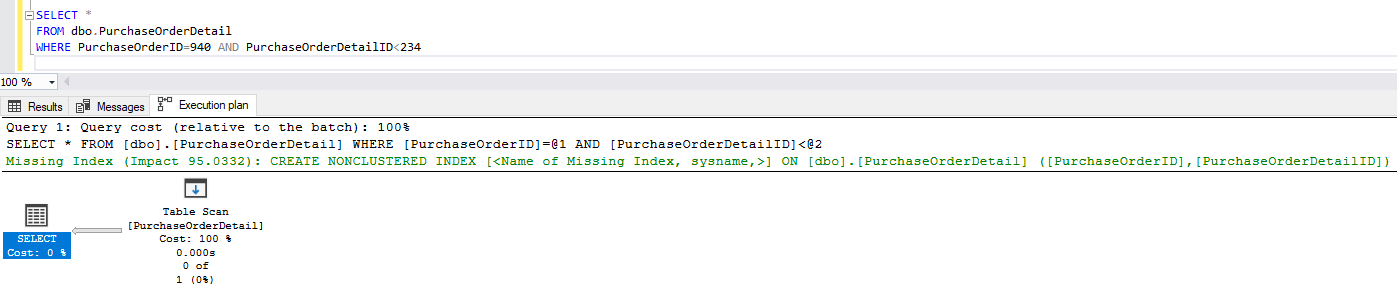
Slika 20. Kreiranje tabele *dbo.PurchaseOrderDetail* i izvršavanje upita nad kreiranom tabelom

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 21.

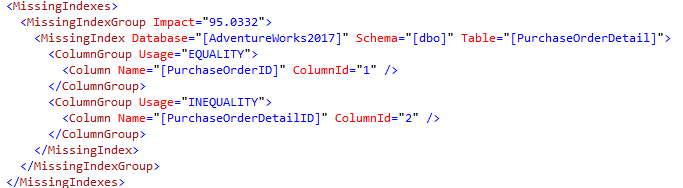


Slika 21. Prikaz dela XML plana izvršenja upita prikazanog na Slici 13

U okviru XML plana izvršenja i zapisa *Missing Indexes* mogu se uočiti tri grupe: : *equality*, *inequality*, i *included.* Na Slici 22. prikazano je obaveštenje o nedostajućem indeksu za zadati upit, a na Slici 23. prikazane su grupe nedostajućeg indeksa.



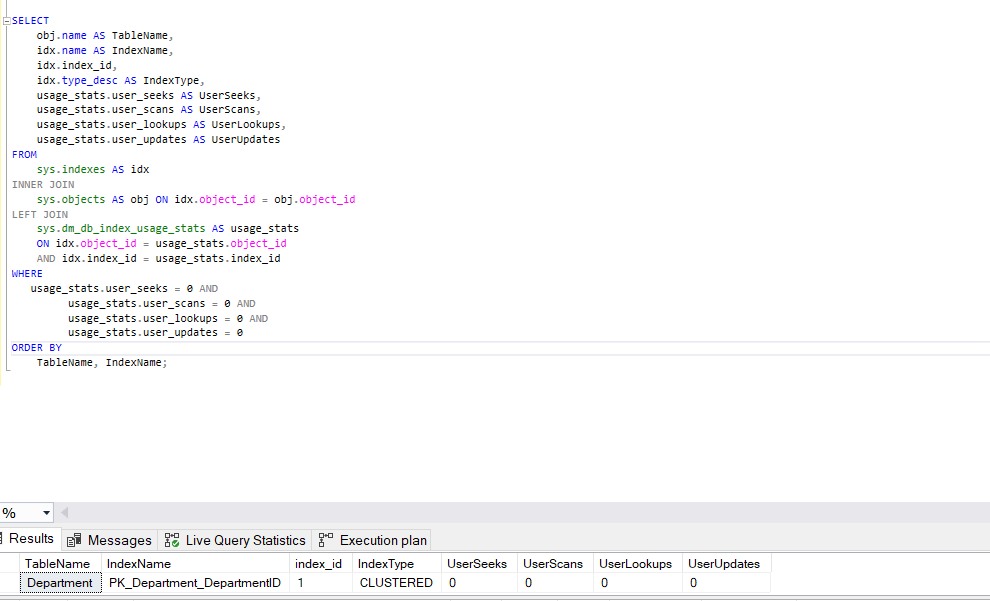
Slika 22. Prikaz informacije o nedostajućem indeksu za prikazani upit



Slika 23. Prikaz nedostajućih grupa indeksa

## **Pronalazak nekorišćenih indeksa**

Indeksi u SQL Serveru igraju ključnu ulogu u poboljšanju performansi SELECT upita, međutim, njihova prisutnost može imati negativan uticaj na operacije ažuriranja podataka. Svaki put kada se izvrši INSERT, UPDATE ili DELETE operacija, indeksi se moraju ažurirati. Ovaj proces može dovesti do produženog trajanja transakcija i produženog vremena izvršenja upita, što može rezultirati zaključavanjima, blokiranjima i čestim timeout-ima. Kod velikih baza podataka ili tabela, prisustvo suvišnih indeksa može takođe zauzeti dragoceni prostor na disku.

Kako bi se optimizovala baza podataka, važno je ne samo dodavati potrebne indekse, već i identifikovati i ukloniti suvišne indekse koji se ne koriste. Informacije o upotrebi indeksa čuvaju se u dinamičkom prikazu `sys.dm\_db\_index\_usage\_stats`. Ovaj prikaz beleži broj pretraživanja, skeniranja, pregledanja i ažuriranja od strane korisničkih i sistemskih upita, kao i vreme poslednje aktivnosti. Važno je napomenuti da se podaci u ovom prikazu brišu nakon ponovnog pokretanja SQL Servera. Upit za identifikaciju neaktivnih indeksa može pomoći u prepoznavanju i uklanjanju suvišnih indeksa, kao što je prikazano na slici.  


Slika 24. 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 25. prikazan je deo rezultata, indeks *PK\_Department\_DepartmentID* koji nije skoro korišćen.



Slika 25. Prikaz dela rezultata nekorišćenih indeksa

## **Uklanjanje nekorišćenjih indeksa**

Kod identifikacije indeksa koji se ne koriste, važno je doneti odluku o njihovom uklanjanju uzimajući u obzir nekoliko faktora:

1. **Korišćenje indeksa**: Ako se indeks ne koristi u izvršenju upita, to može biti znak da je suvišan. U sys.dm\_db\_index\_usage\_stats se mogu pronaći podaci o pretraživanjima, skeniranjima i drugim operacijama. Ako indeks ima nisku ili nulu vrednost za ove statistike, to može ukazivati na to da se ne koristi.
2. **Učinak na performanse**: Ako uklanjanje indeksa ne utiče na performanse upita, to je dobar znak da je indeks zaista suvišan. Pre nego što obrišete indeks, dobar korak bi bio njegovo testiranje gde bi se videlo kako bi njegovo uklanjanje uticalo na performanse upita.
3. **Troškovi održavanja**: Indeksi koji se ne koriste povećavaju troškove održavanja baze podataka. Svaki dodatni indeks dodaje overhead prilikom izvršavanja operacija ažuriranja, što može usporiti transakcije. Ako su indeksi skupi za održavanje i ne donose korist u pogledu poboljšanja performansi upita, oni bi trebali biti uklonjeni.
4. **Istorijski podaci**: Razmatranje da li je indeks možda bio koristan u prošlosti, ali više nije potreban zbog promene u obrascima upita ili aplikacijama. Ako se očekuje da će se obrasci upita promeniti ili da će se novi upiti koristiti u budućnosti, možda će biti korisno sačuvati neki indeks koji se trenutno ne koristi.
5. **Sistem i aplikacija**: Razmišljanje o uticaju na aplikaciju i sistem. Ako je indeks deo neke specifične aplikacije ili sistema koji je trenutačno van upotrebe, ali se očekuje da će se vratiti, možda je bolje zadržati indeks.

.

# 

# 

# **Materijalizovani pogledi**

Materijalizovani pogledi predstavljaju naprednu funkcionalnost u SQL bazama podataka koja omogućava skladištenje rezultata upita u fizičkom obliku, što može značajno poboljšati performanse u određenim scenarijima.

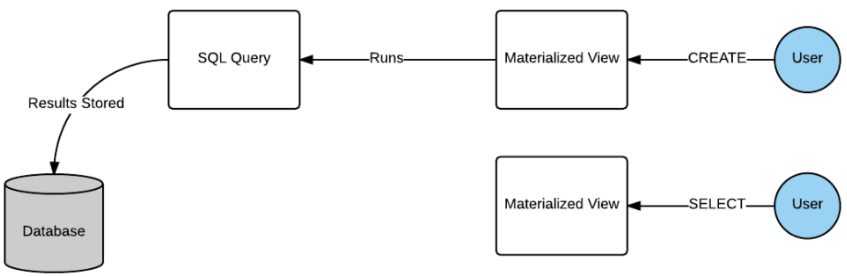
Materijalizovani pogled je SQL objekat koji skladišti rezultate upita kao fizičke podatke. Za razliku od običnih pogleda koji prilikom svakog pristupa izvršavaju upit, materijalizovani pogled čuva rezultate upita, omogućavajući brži pristup tim podacima. Ovaj pristup je sličan tabelama, ali sa prednostima koje nudi unapred definisani rezultat upita.

Materijalizovani pogled se kreira definisanjem upita čiji rezultati se žele sačuvati. Kada se kreira, materijalizovani pogled popunjava podatke na osnovu izvršenja upita. Ovi podaci mogu biti ažurirani ili obnovljeni u skladu sa potrebama, što može biti automatski ili manualno, zavisno od konfiguracije. Održavanje podataka može se vršiti na različite načine, uključujući periodična osvežavanja (refresh) kako bi se podaci ažurirali u skladu sa izvorom.

Zbog prethodnog skladištenja rezultata, materijalizovani pogledi mogu značajno smanjiti vreme potrebno za izvršavanje složenih upita koji uključuju velike količine podataka i kompleksne operacije kao što su spajanja i agregacije. Ovo takođe smanjuje opterećenje na izvornim tabelama jer se skupi upiti ne izvršavaju u stvarnom vremenu, već koriste unapred izračunate rezultate. Pomaže i u optimizaciji performansi upita koji se često koriste, jer omogućava brzi pristup već pripremljenim rezultatima.

Međutim, materijalizovani pogledi imaju i svoje nedostatke. Zahtevaju dodatni prostor za skladištenje rezultata upita, što može biti značajno kod velikih i kompleksnih upita. Takođe, potrebno je redovno osvežavanje materijalizovanih pogleda da bi se podaci uskladili sa izvorom, što može dodatno opteretiti sistem, posebno kod velikih baza podataka ili čestih promena podataka. Implementacija materijalizovanih pogleda zahteva pažljivo planiranje i upravljanje kako bi se osiguralo da su uvek ažurirani i relevantni za korisničke potrebe.

Materijalizovani pogledi su naročito korisni u scenarijima sa velikim upitima koji uključuju kompleksne operacije i velike količine podataka, kao što su analitičke baze podataka i sistemi za izveštavanje. Oni omogućavaju unapred definisanje rezultata za često korišćene upite, čime se poboljšava brzina odziva i efikasnost sistema.

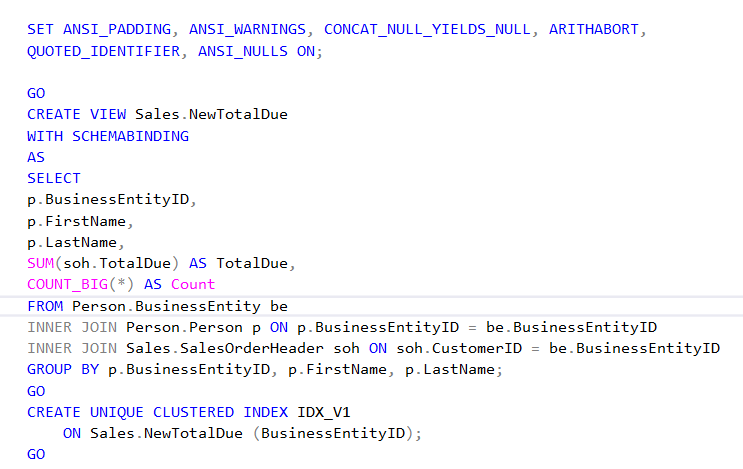


Slika 26. Način kreiranja materijalizovanog pogleda

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

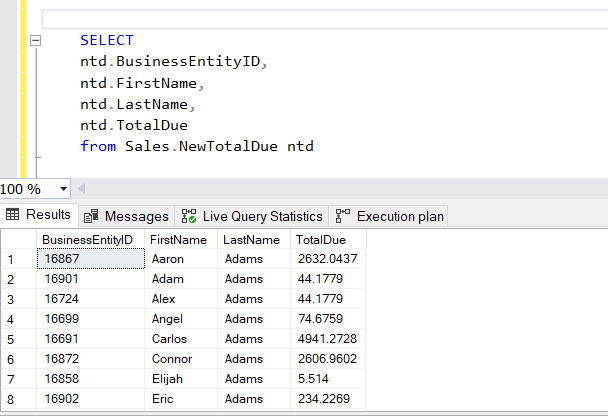
* Kreirati regularni pogled
* Kreirati klasterizovani indeks nad tim pogledom

Kreiranje indeksiranog pogleda prikazano je na Slici 27. 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 27. Kreiranje indeksiranog pogleda

Nakon kreiranja indeksiranog pogleda, podacima je moguće pristupiti na sledeći način (Slika 28):



Slika 28. 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**

Uloga optimizatora upita u sistemima za upravljanje bazama podataka (DBMS) je od ključne važnosti za postizanje visokih performansi i efikasnosti. Ovaj rad istražuje osnovne komponente i rad optimizatora upita, uključujući faze obrade kao što su parsiranje, vezivanje, optimizacija, izvršenje i keširanje planova. Posebnu pažnju posvećujemo analiziranju procesa optimizacije upita, uključujući simplifikaciju, evaluaciju trivijalnih planova i punu optimizaciju.

Fokus rada je na analizi metoda koje SQL Server koristi za odabir najefikasnijeg plana izvršenja. Analizom kandidata i procenom njihove cene, optimizator upita može identifikovati najpovoljniji plan. Detaljno su objašnjene tehnike formiranja planova, uključujući korišćenje informacija iz sistemskog kataloga, histogramima, proceni gustine i estimaciji veličine rezultata. Računanje cene svakog operatora i sumiranje ukupne cene plana pruža uvid u efikasnost izvršenja.

Indeksi predstavljaju ključne komponente za ubrzanje upita, omogućavajući brzo pretraživanje bez skeniranja celokupnih tabela. U radu su detaljno analizirani metodi za pronalaženje nedostajućih i suvišnih indeksa, kao i njihovo uticaj na performanse sistema. Takođe, materijalizovani pogledi su razmatrani kao dodatna tehnika optimizacije koja omogućava direktan pristup prethodno obrađenim podacima.

Efikasnost sistema zavisi od sposobnosti optimizacije upita, što predstavlja suštinski aspekt svakog DBMS-a. Analizom tehnika i alata dostupnih u MS SQL Serveru, ovaj rad pruža uvid u napredne metode za optimizaciju i obrada upita, naglašavajući njihov značaj za postizanje optimalnih performansi.

# **Literatura**

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

# <https://aiven.io/developer/sql-query-optimization-guide>

<https://byjus.com/gate/decomposition-in-dbms/>

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

<https://learn.microsoft.com/en-us/sql/relational-databases/performance/display-an-actual-execution-plan?view=sql-server-ver16>  
  
<https://learn.microsoft.com/en-us/sql/relational-databases/views/views?view=sql-server-ver16>  
  
<https://aws.amazon.com/what-is/materialized-view/>

<https://www.amazon.com/Inside-SQL-Server-Query-Optimizer/dp/1906434603>

<https://freecomputerbooks.com/Inside-the-SQL-Server-Query-Optimizer.html>