|  |  |  |
| --- | --- | --- |
| Unigrb | Univerzitet u Nišu ELEKTRONSKI FAKULTET | logo_1960_4 |

**Optimizacija upita kod MS SQL baza podataka**

SEMINARSKI RAD

|  |  |
| --- | --- |
| Mentor: | Student: |
|  |  |
| Prof. dr Aleksandar Stanimirović | Stefan Mladenović1694 |

Niš, jun 2024. God.

Sadržaj

[1. Uvod 2](#_Toc387516732)

[2. Razumevanje arhitekture izvršavanja upita u MS SQL Serveru 3](#_Toc76756144)

[2.1. Pregled arhitekture izvršavanja upita 3](#_Toc208694367)

[2.2. Plan izvršavanja upita i njegov značaj 4](#_Toc2102928589)

[3. Plan izvršenja upita 4](#_Toc1861741426)

[3.1. Strukutra plana izvršenja 6](#_Toc2139527839)

[3.2. Vrste operatora u planu izvršenja 6](#_Toc209959150)

[3.3. Čitanje MSSQL plana izvršenja 8](#_Toc1181115819)

[3.4. Identifikacija uskih grla u planu izvršavanja 8](#_Toc1905791120)

[4. Tehnike optimizacije upita 9](#_Toc787914399)

[4.1. Indeksi 9](#_Toc755291817)

[4.2. Klasterisani indeksi 10](#_Toc1203502507)

[4.2.1. Primer 11](#_Toc368807898)

[4.3. Neklasterisani indeksi 13](#_Toc900205681)

[4.3.1. Primer 14](#_Toc1718333235)

[4.4. Particionisanje tabela za poboljšanje performansi 16](#_Toc2098930315)

[4.5. Korišćenje CTE (Common Table Expressions) i privremenih tabela 18](#_Toc300619088)

[4.5.1. Korišćenje CTE (Common Table Expressions) 18](#_Toc85927954)

[4.5.2. Korišćenje privremenih tabela (temporary tables) 20](#_Toc1559039942)

[5. Napredne tehnike optimizacije upita 21](#_Toc1796873264)

[5.1. Heuristička optimizacija 21](#_Toc1387918757)

[5.2. Statistika 22](#_Toc171421370)

[5.2.1. Histogram 23](#_Toc1650506763)

[5.2.2. Density vector 24](#_Toc713035769)

[5.2.3. Kreiranje statistike 24](#_Toc381775042)

[5.3. Inteligentna obrada upita (Intelligent Query Processing - IQP) 25](#_Toc1302010903)

[5.4. Automatsko podešavanje (Automatic Tuning) 27](#_Toc646474849)

[6. Alati za poboljšanje performansi izvršenja upita 28](#_Toc1899040654)

[6.1. SQL Server Profiler 28](#_Toc607912501)

[6.1.1. Primer korišćenja 29](#_Toc642652735)

[6.2. Database Engine Tuning Advisor 31](#_Toc405653323)

[6.2.1. Primer korišćenja 31](#_Toc1199199068)

[7. Zaključak 34](#_Toc1250030157)

[8. Literatura 35](#_Toc1117478207)

# 1. Uvod

Optimizacija upita u Microsoft SQL Serveru je jedna od ključnih stvari za postizanje efikasnosti u radu sa velikim količinama podataka. Sa porastom složenosti aplikacija, broj izvršenih upita raste, a time i potreba za bržom i preciznijom obradom podataka. SQL Server koristi niz naprednih tehnika za izvršavanje upita, koje, ukoliko se pravilno primene, mogu značajno poboljšati performanse sistema. Razumevanje kako SQL Server obrađuje upite, od trenutka kada se unesu do konačnog izvršenja, ključno je za dizajniranje optimalnih rešenja u realnim sistemima baza podataka.

Arhitektura izvršavanja upita u SQL Serveru sastoji se od nekoliko komponenti koje rade zajedno na generisanju i optimizaciji plana izvršavanja. Ovi planovi sadrže detaljne instrukcije o tome kako će se podacima pristupati i manipulisati, uključujući upotrebu indeksa, skeniranja tabela i sortiranja. Razumevanje ovog procesa omogućava nam da identifikujemo potencijalna uska grla i unapredimo efikasnost izvršenja.

Jedan od ključnih elemenata optimizacije upita su indeksi, klasterisani i neklasterisani indeksi, koji omogućavaju brži pristup podacima i smanjuju opterećenje na resurse sistema. Pored indeksa, particionisanje tabela i upotreba privremenih struktura, kao što su *Common Table Expressions* (CTE) i privremene tabele, dodatno doprinose boljoj organizaciji i pristupu podacima, posebno kod kompleksnih i često izvršavanih upita.

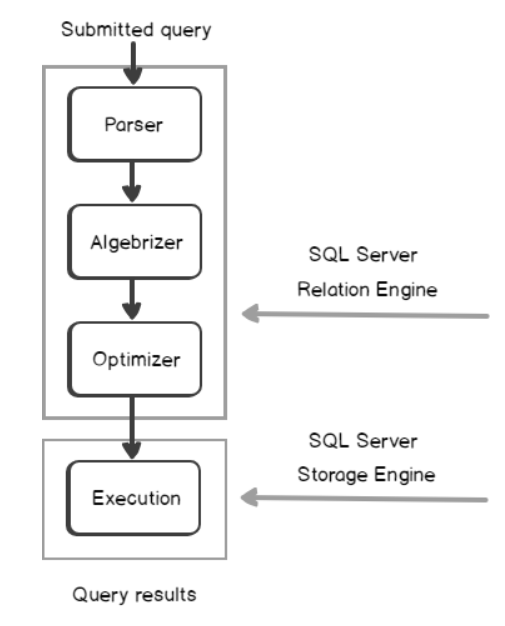
Napredne tehnike kao što je *Intelligent Query Processing* (IQP) pružaju automatizovana rešenja za optimizaciju, oslanjajući se na učenje iz prethodnih izvršenja upita. Ove tehnike omogućavaju dinamičko prilagođavanje plana izvršavanja u realnom vremenu, čime se dodatno poboljšavaju performanse sistema, bez potrebe za ručnom intervencijom. U daljem tekstu, razmotrićemo detalje ovih tehnika i kako se mogu koristiti za optimizaciju samih performansi.

# 2. Razumevanje arhitekture izvršavanja upita u MS SQL Serveru

Da bismo još dublje razumeli proces optimizacije upita, bitno je da se detaljnije upoznamo sa arhitekturom izvršavanja upita u MS SQL Serveru. Arhitektura izvršavanja upita je složen proces koji uključuje više komponenti, a razumevanje ovog procesa ključno je za efikasnu optimizaciju upita.

## 2.1. Pregled arhitekture izvršavanja upita

Kada šaljemo T-SQL upit SQL serveru, samim tim informišemo SQL Server Engine o tome šta želimo, ali ne i o načinu na koji to tačno treba da se izvede. Između trenutka slanja upita serveru i dobijanja rezultata, SQL Server Engine će proći kroz četri interne operacije obrade upita kako bi pretvorio taj upit u format koji je najpogdniji serveru za samo izvršenje datog upita.



Slika 2.1. Arhitektura obrade upita [1]

Prve tri operacije parsiranje, algebrarizacija i optimizacija se obavalja od strane SQL Server Relation Engine-a. Parsiranje je proces u kojem se proverava sintaksa upita, dok algebrarizacija pronalazi i vrši proveru objekata u bazi podataka koje upit koristi. Nakon toga, optimizacija traži najefikasniji način izvršenja samog upita. Ako je upit naredba koja ne manipuliše podacima, poput CREATE TABLE ili ALTER TABLE, neće biti potrebno optimizovati taj upit. Nakon kreiranja plana izvršenja, izabrani plan se čuva u memoriji za buduću upotrebu.

Poslednji korak, izvršenje upita, obavlja se od strane SQL Server Storage Engine-a. To je faza u kojoj se izabrani plan izvršenja primenjuje na bazu podataka kako bi se dobio traženi rezultat. Ova faza obuhvata procese kao što su čitanje, upisivanje ili ažuriranje podataka, u zavisnosti od zahteva upita.

Svaka od ovih komponenti igra ključnu ulogu u procesu izvršavanja upita, i razumevanje njihove interakcije omogućava nam da identifikujemo potencijalne tačke za optimizaciju i poboljšanje performansi sistema. [1]

## 2.2. Plan izvršavanja upita i njegov značaj

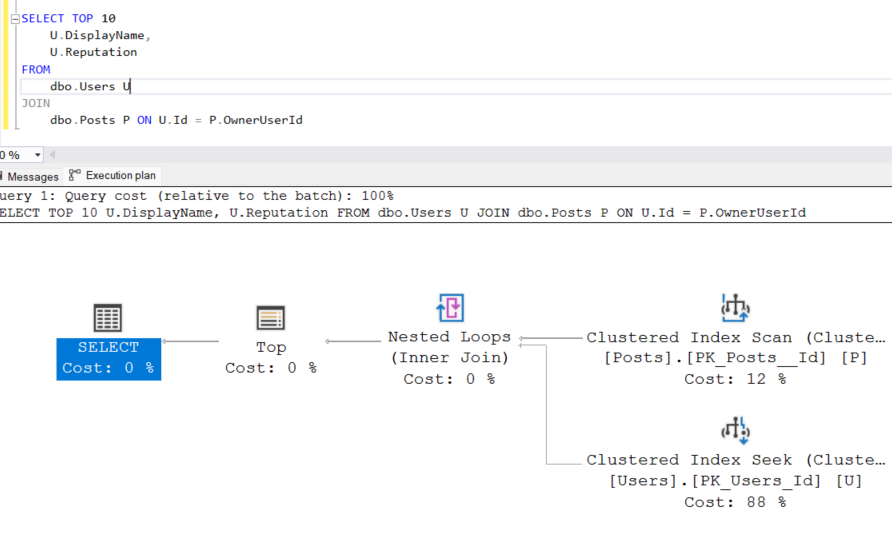
SQL Server Execution Plan je ključni deo procesa optimizacije. To je binarna reprezentacija koraka koje će SQL Server Engine pratiti kako bi izvršio sam upit na najefikasniji način. Plan izvršenja je rezultat rada SQL Server Query Optimizer-a, koji generiši najbolji plan izvršenja za zadati upit.

Za optimizaciju upita, razumevanje plana izvršenja je od velikog značaja. Analizom plana izvršenja možemo identifikovati delove upita koji možda zahtevaju optimizaciju, kao što su skup podataka koji se često pristupa ili operacije koje troše previše resursa. Takođe, plan izvršenja omogućava nam da predvidimo performanse izvršenja upita i da prilagodimo indekse, statistike ili strukturu upita kako bismo poboljšali performanse sistema.

# 3. Plan izvršenja upita

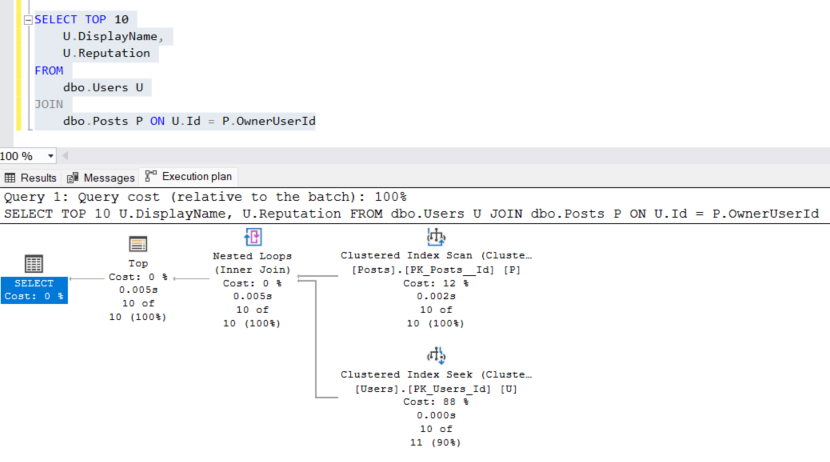
U Microsoft SQL Server Management Studio (SSMS), postoji nekoliko načina na koje možemo generisati plan izvršenja:

1. **Estimated Execution Plan**: Plan procene izvršenja daje prognozu kako SQL Server planira da izvrši upit, ali bez stvarnog izvršenja. Ovaj plan se generiše na osnovu procene optimizatora upita, koji uzima u obzir statistiku podataka kako bi odredio najefikasniji način izvršenja upita. Plan procene izvršenja može ukazati na vreme na probleme sa performansama i optimizacijom upita pre nego što bude upit izvršen, i na taj način štedi vreme.

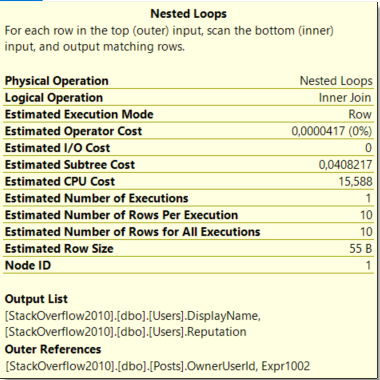
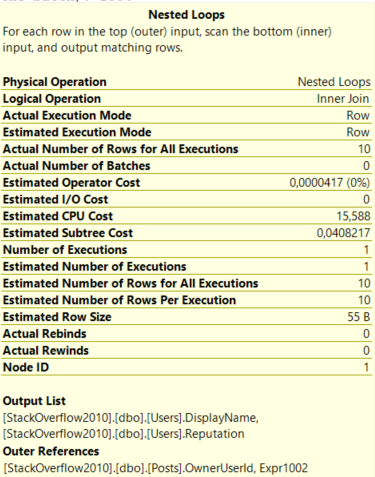


Slika 3.1. Plan procene izvršenja

1. **Actual Execution Plan**: Za razliku od plana procene izvršenja, stvarni plan izvršenja se generiše nakon izvršenja upita. Stvarni plan izvršenja daje više informacija o operacijama koje je SQL Server izvršio tokom izvršenja upita, kao što su broj obrađenih redova, korišćeni resursi i troškove svake operacije. Ovaj plan za razliku od prethodnog daje značajno detaljnije informacije za dijagnostikovanje problema sa performansama koji nisu bili očigledni kod plana procene izvršenja, jer prikazuje realni tok izvršenja.



Slika 3.2. Stvarni plan izvršenja

Slika 3.3. Prikaz informacija kod stvarnog i procenjenog plana izvršenja

1. **Live Query Statistics**: Statistike izvršenja upita uživo daju realan prikaz izvršenja upita, pri čemu omogućavaju da korisnici vide kako napreduje svaka operacija u planu izvršenja dok se upit izvršava. Ova funkcija je posebno korisna za praćenje izvršavanja upita koji predugo traju. Ove statistike uživo omogućavaju da se identifikuju uska grla i problemi sa performansama dok se sam upit izvršava i na taj način skraćuju vreme čekanja da se sam upit izvrši. Ovakav vid statistika je veoma koristan kod upita koji traju više časova.

Svaka vrsta plana izvršenja u SSMS-u služi za različite svrhe, od planiranja i optimizacije upita pre izvršenja do dijagnostikovanja i rešavanja problema nakon ili tokom izvršenja upita. Koriščenjem ovih alata, možemo dobiti detaljan uvid u proces izvršenja upita, na osnovu čega možemo primeniti odgovarajuće korake kako bi dobili što bolje performanse. [2]

## 3.1. Strukutra plana izvršenja

Plan izvršenja ima dobro dizajniranu strukturu kako bi mogao da pruži vizuelno predstavljanje svih koraka koje SQL Server preduzima tokom izvršenja jednog upita. Razumevanje ove strukture je ključno za analizu i samu optimizaciju upita. Planovi izvršenja mogu se predstaviti u TXT, XML formatu ili grafičkim prikazom koji je čitljiviji samom korisniku.

MSSQL plan izvršenja sastoji se od međusobno povezanih čvorova, pri čemu svaki čvor predstavlja određenu operaciju ili korak izvršenja upita. Čvorovi se drugačije nazivaju operatori i organizovani su u strukturu stabla. Na taj način se prikazuje tok podataka od jedne do druge operacije. Struktura plana se može podeliti u tri grupe:

1. **Operatori:** Predstavljaju osnovni element plana izvršenja. Svaki operator predstavlja jednu fizičku operaciju kao što su skeniranje tabele, traženje indeksa, spajanje tabele ili sortiranje. Operatori se prikazuju u obliku ikonice u planu izvršenja i svaki operator ima svoju ikonicu.
2. **Strelice:** Strelice između operatora predstavljaju tok podataka od jednog operatora do drugog. Debljina strelice zavisi od količine podataka koja se prenosi između operatora, čime se pruža vizuelna informacija o količini podataka koja se obrađuje u tom koraku.
3. **Svojstva:** Svaki operator sadrži set informacija koji daju detaljan opis operacije, kao što su procenjeni broj redova, ukupni trošak operacije u odnosu na ukupni trošak upita, korišćeni indeksi itd. Svojstva pružaju detaljan uvid performansa svake operacije.

## 3.2. Vrste operatora u planu izvršenja

Svaki operator obavlja specifičnu funkciju nad skupovima podataka, kao što su pretraga, filtriranje, spajanje, ili sortiranje. Osnovni operatori u planu izvršenja mogu se podeliti u nekoliko glavnih kategorija, od kojih svaki obavlja specifičnu ulogu u obradi podataka. Ovi operatori su ključni za razumevanje kako SQL Server izvršava upite i kako se mogu optimizovati za bolje performanse.  
  
 **Skeniranje** - Ovi operatori obavljaju pretragu nad celim tabelama ili indeksima. A to su:

* *Table Scan*: Sekvencijalno pretraga cele tabele bez korišćenja indeksa, što je najneefikasniji način pretrage za velike tabele.
* *Index Scan*: Pretraga podataka preko indeksa, omogućava brže izvršenje upita u poređenju sa Table Scan, ali i dalje može biti neefikasno kod neoptimalnih upita.
* *Index Seek*: Umesto pretrage celog indeksa, ovaj operator koristi selektivnu pretragu kroz indeks, čime značajno ubrzava pronalaženje podataka. Koristi se kada SQL Server može efikasno da iskoristi indeks za tražene vrednosti.

**Spajanje** - Operator spajanja se koristi kada je potrebno kombinovati podatke iz dve ili više tabela. Tri korišćena operatora spajanja su:

* *Nested Loops Join*: Efikasan za manje setove podataka, ali može biti vrlo neefikasan za veće tabele. Spajanje funkcioniše tako što se jedan po jedan red iz jedne tabele upoređuje sa svim redovima druge tabele.
* *Merge Join*: Najefikasniji kada su ulazne tabele unapred sortirane. Radi tako što upoređuje redove na osnovu sortiranih ključeva i kombinuje podatke.
* *Hash Match Join*: Koristi se kada podaci nisu unapred sortirani. Ovaj operator kreira heš tabelu iz jedne tabele, a zatim koristi heš vrednosti da efikasno pronađe odgovarajuće redove u drugoj tabeli.

**Filtriranje** - Ovi operatori se koriste za primenu filtera nad podacima.

* *Filter*: Ovaj operator koristi WHERE klauzulu da bi eliminisao redove koji ne zadovoljavaju zadati uslov. Efikasnost zavisi od dostupnosti odgovarajućih indeksa.
* *Predicate*: Slično *Filter* operatoru, *Predicate* se koristi za evaluaciju logičkih uslova nad redovima tokom procesa izvršenja.

**Sortiranje -** Kada je potrebno sortirati rezultate pre nego što se proslede dalje operacijama poput agregacije ili spajanja, koristi se operator sortiranja.

* *Sort*: Sortira rezultate prema specifičnim kolonama definisanim u SQL upitu, što može biti resursno zahtevno za velike setove podataka ako ne postoje odgovarajući indeksi.

**Agregacija -** Ovi operatori grupišu i sabiraju podatke, često koristeći funkcije kao što su SUM, COUNT, AVG, i slično.

* **Stream Aggregate**: Ovaj operator koristi prethodno sortirane podatke da efikasno grupiše redove bez kreiranja dodatnih struktura podataka.
* **Hash Aggregate**: Koristi se kada podaci nisu unapred sortirani, i koristi heš tabelu za grupisanje podataka. Efikasan je za neuređene skupove podataka, ali može zahtevati više memorijskih resursa.

**Bookmark Lookup** - Ovaj operator koristi se u kombinaciji sa indeksima kako bi se pristupilo celim redovima podataka na osnovu vrednosti pronađenih u indeksu. Ovaj operator može značajno usporiti izvršenje upita ako se često koristi u neefikasnim upitima.

* *Key Lookup*: Koristi indeksirani ključ za pronalaženje i povlačenje ostatka podataka iz tabele.
* *RID Lookup*: Koristi jedinstveni identifikator (Row ID) da bi pronašao određeni red u tabeli.

## 3.3. Čitanje MSSQL plana izvršenja

Pod čitanjem MSSQL plana izvršenja, podrazumeva se analiza operatora, tok podataka i svojstva svake operacije kako bi smo razumeli kako će SQL server izvršiti upit. Plan izvršenja se čita sa desna na levo i odozgo na dole. Prvi operator na desnoj strani je obično prva operacija koju SQL Server izvršava, a tok podataka se kreće na levo.

Sledeći korak jeste da se prepoznaju sami operatori. Operatori su označeni ikonicama i svaki operator ima svoju ikonicu. Vrlo je bitno poznavati česte operatore kao što su skeniranje tabela, traženja indeksa i join-ova. Razumevanje svakog operatora je ključno prilikom same analize plana.

Nakon toga, potrebno je analizirati sam tok podataka. Strelice u planu označavaju kako se podaci kreću, dok debele strelice predstavljaju veliku količinu podataka koja se obrađuje. To može biti znak neefikasnosti u slučaju da se podaci značajno smanje u narednim operacijama.

Prelaskom miša preko operatora, ili desnim klikom na operator mogu se pregledati svojstva operatora. Tu se mogu videti svojstva poput procenjeni broj redova, procenjeni I/O trošak i procenjeni CPU trošak koja su jako bitna kod procene same efikasnosti operacije.

SQL Server često daje upozorenja na šta treba obratiti pažnju. Upozorenja su označena malom ikonicom na samom operatoru i ukazaju na potencijalne probleme sa performansama, poput nedostatka indeksa ili implicitnih konverzija tipova. Ova upozorenja mogu biti vrlo korisna kod optimizacije upita.

Kod svakog operatora u planu imamo dat njegov trošak, koji predstavlja potrošnju resursa u odnosu na ukupni trošak upita. Operatori sa visokim troškovima treba da budu potencijalni kandidati za samu optimizaciju. [3]

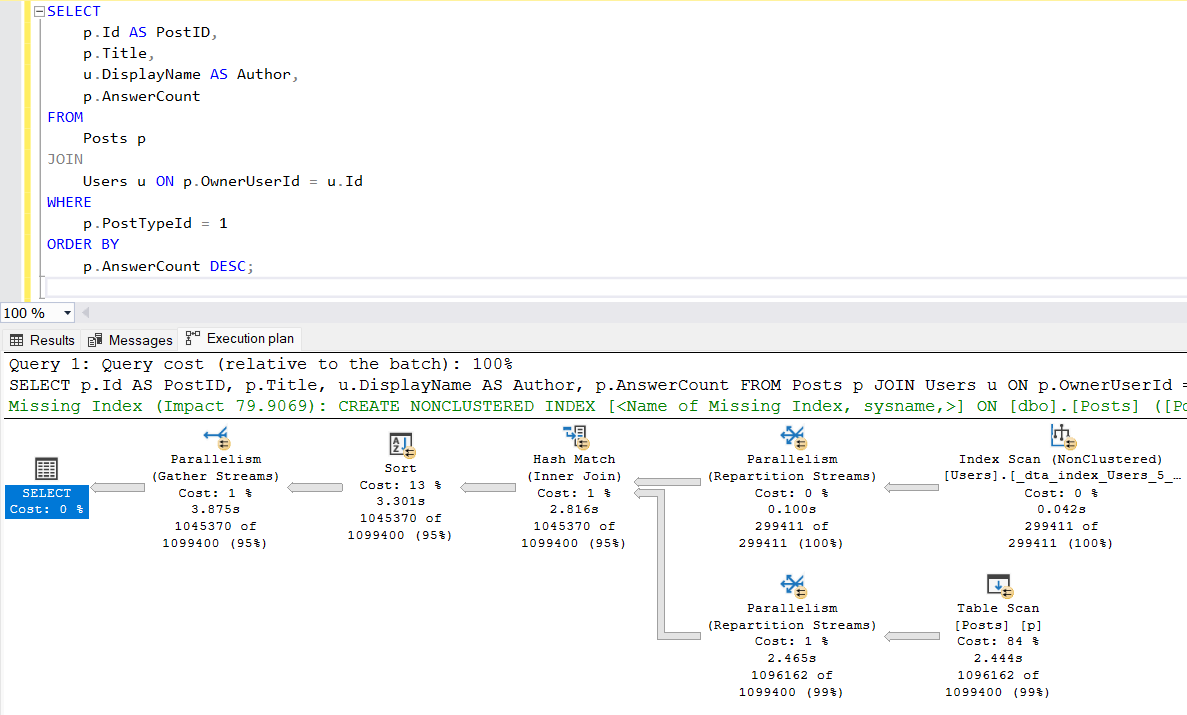
Ovo su glavni koraci kod analize plana koji mogu pomoći prilikom optimizacije performansi upita.

## 3.4. Identifikacija uskih grla u planu izvršavanja

U planu izvršavanja, SQL Server detaljno prikazuje način na koji će se upit izvršiti, što uključuje operacije kao što su pretrage indeksa, spajanja tabela, sortiranja, i slično. U tom procesu, moguće je identifikovati uska grla, odnosno operacije koje najviše doprinose ukupnom vremenu izvršavanja upita.

Navešćemo operacije koje ukazuju na uska grla u planu izvršenja:

1. Operacije sa visokim troškom: U planu izvršavanja, SQL Server procenjuje koliko svaka operacija doprinosi ukupnom vremenu izvršavanja. Operacije sa visokim procentom troška obično signaliziraju potencijalna uska grla.
2. Skeniranje tabele: Kada SQL Server ne može da koristi indeks, može izvršiti skeniranje cele tabele, što može značajno usporiti upit, posebno na velikim tabelama.
3. Spajanja: Ako se spajanja vrše bez adekvatnih indeksa, SQL Server može koristiti neefikasne metode kao što su Nested Loops, što može uzrokovati velika kašnjenja.
4. Sortiranje: Operacije sortiranja, posebno nad velikim skupovima podataka, mogu biti zahtevne u pogledu memorije i CPU resursa.
5. Izvršavanje operacija na disku: Kada SQL Server nema dovoljno memorije da obradi određene operacije (poput sortiranja ili hash spajanja), deo podataka može biti prebačen na disk, što značajno usporava izvršavanje.



Slika 3.4. Prikaz neoptimizovanog plana izvršenja.

# 4. Tehnike optimizacije upita

## 4.1. Indeksi

Indeksi igraju jednu od glavnih uloga u aspektu poboljšanja efikasnosti operacija povlačenja podataka. Pružaju struktuiran način pristupa podacima koji može značajno smanjiti vreme i resurse potrebne za upite u velikim bazama podataka. Održavanjem sortiranog redosleda podataka na osnovu kolona ključa, indeksi omogućavaju SQL Server Engine-u brzo lociranje i povlačenje podataka bez skeniranja cele tabele.

Indeks sadrži ključeve napravljene od jedne ili više kolona u tabeli ili pogledu. Ključevi se čuvaju u strukturi B+ stabla koja omogućava SQL Server-u da brzo i efikasno pronađe red ili redove povezane sa datim ključevima.

MSSQL podržava dva osnovna tipa indeksa i to su klasterisani i neklasterisani, pri čemu svaki od njih ima svoju ulogu.

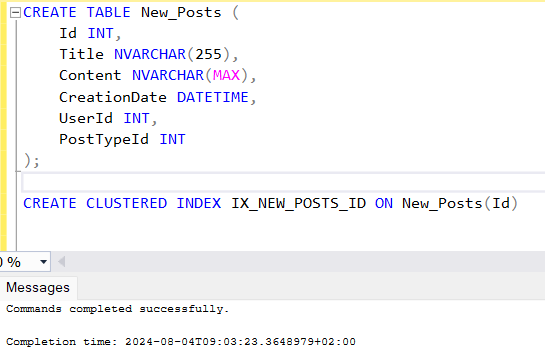
## 4.2. Klasterisani indeksi

Klasterisani indeksi određuju fizički redosled podataka u tabeli. Oni sortiraju i čuvaju redove podataka tabele ili pogleda na osnovu ključa indeksa. Može postojati samo jedan klasterisani indeks po tabeli, jer se redovi u tabeli mogu čuvati samo u jednom redosledu. Jedini slučaj kada su redovi podataka u tabeli smešteni u sortiranom redosledu jeste kada tabela sadrži klasterisani indeks. Ovakvo fizičko uređenje ubrzava operacije pribavljanja podataka. Ako tabela nema klasterisani indeks, njeni redovi podataka se čuvaju u nesortiranoj strukturi (*heap*).

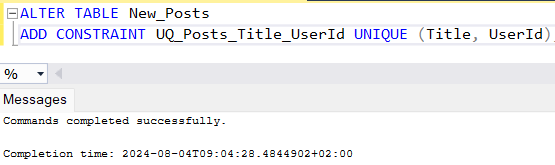
Kada kreiramo primarni ključ, baza automatski kreira jedinstveni klasterisani indeks na toj koloni ili kolonama ako klasterisani indeks već ne postoji na tabeli. Kolona sa primarnim ključem ne može sadržati NULL vrednosti, što znači da svaka vrednost u toj koloni mora biti jedinstvena i ne sme biti prazna.

Klasterisani indeks će biti kreiran automatski prilikom kreiranja jedinstvenog ograničenja (*unique constraint*), ali samo ako na zadatoj tabeli već ne postoji klasterisani indeks.

Ako klasterisani indeks nije eksplicitno definisan kao jedinstven, SQL Server automatski održava njegovu jedinstvenost dodavanjem jedinstvenog identifikatora dodatnim atributom (*uniquifier*). [4]



Slika 4.1. Primer kreiranja klasterisanog indeksa



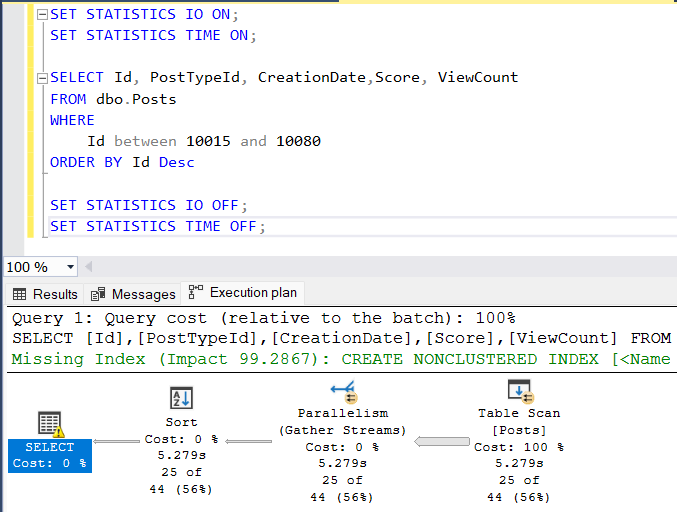
Slika 4.2. Primer kreiranja jedinstvenog ograničenja

Jedinstveno ograničenje će izazvati kreiranje klasterisanog indeksa samo ako on već ne postoji nad istom tabelom.

Klasterisani indeksi značajno poboljšavaju performanse upita koji pretražuju podatke po vrednostima koje su deo ključa indeksa, posebno kada upiti zahtevaju opsežno pretraživanje ili sortiranje. Ali dodavanje, ažuriranje ili brisanje redova može biti sporije u tabelama sa klasterisanim indeksom, jer SQL Server mora da održava fizički redosled podataka. Takođe, fragmentacija može nastati tokom vremena, što može zahtevati dodatno održavanje, poput rekonstrukcije ili reorganizacije indeksa.

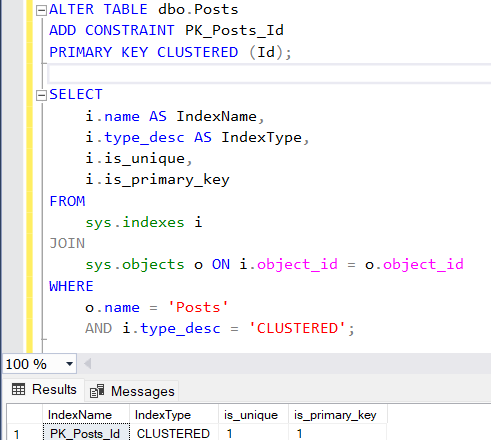
### 4.2.1. Primer

Na sledećem primeru je data demonstracija izvršenja istog upita prilikom nepostojanja i postojanja klasterisanog indeksa.



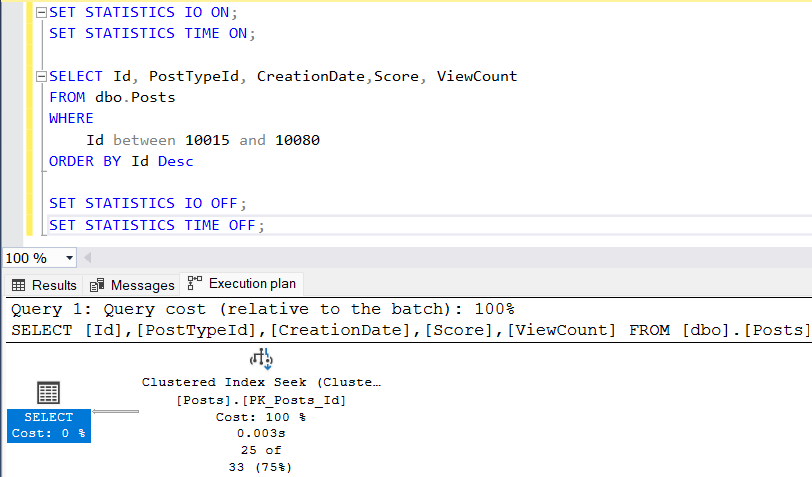
Slika 4.3. Primer izvršenja upita bez klasterisanog indeksa.

Nakon toga smo izvršili dodavanja primarnog ključa nad kolonom Id, što je automatski uzrokovalo dodavanje klasterisanog indeksa.



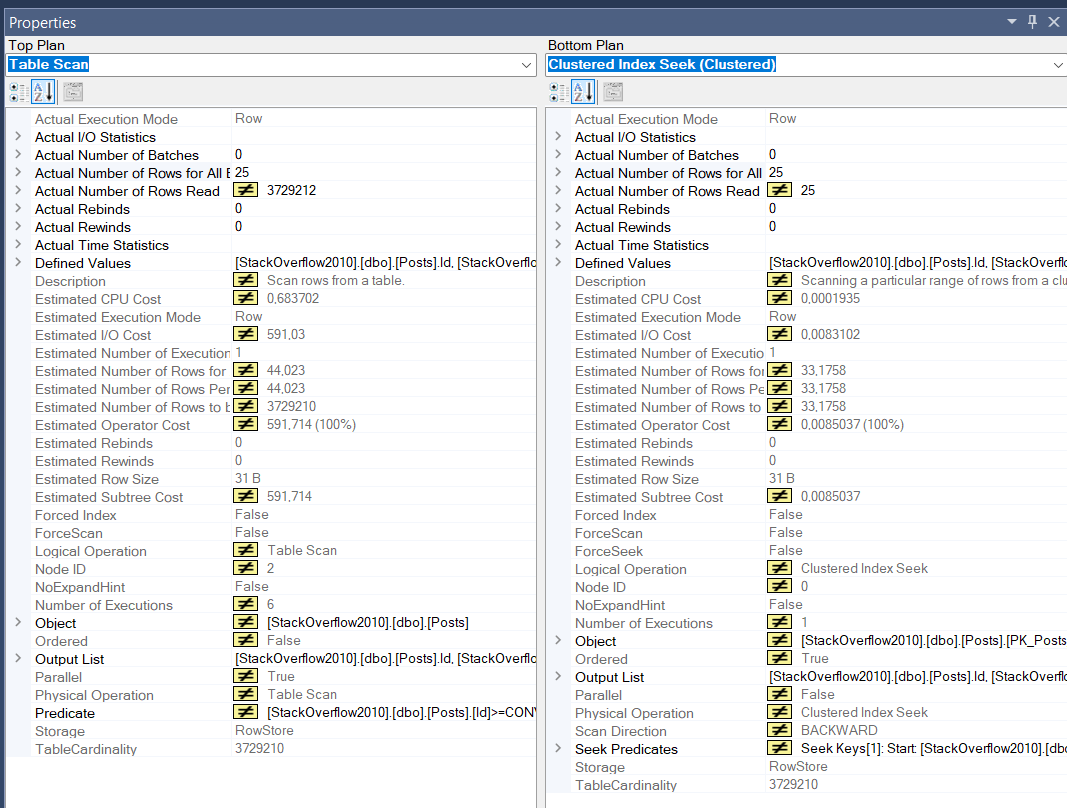
Slika 4.4. Dodavanje primarnog ključa

Nakon toga ponavljamo izvršenje istog upita samo sa postojanjem klasterisanog indeksa.



Slika 4.5. Primer izvršenja upita sa klasterisanim indeksom.

Ključne razlike između izvršenja upita sa i bez klasterisanog indeksa na osnovu planova izvršenja:



Slika 4.6. Poređenje *table scan* i *clustered index seek* operacije.

Prvi upit je prouzrokovao čitanje svih 3729212 redova u tabeli, iako je upit vratio samo 25 redova koji zadovoljavaju uslov *Id BETWEEN 10015 AND 10080*. Ukupni procenjeni trošak (*I/O cost*) je 591,03 što je značajno visoko zbog činjenice da mora pročitati svaki red u tabeli. Ukupni procenjeni CPU trošak je takođe visok (0,683702), što ukazuje na neefikasnost u pogledu potrošnje resursa.

Nakon dodavanja primarnog ključa i klasterisanog indeksa, upit je pročitao samo 25 redova, što značajno smanjuje (*I/O cost*) na vrlo nisku vrednost od 0,0083102. Ukupan procenjeni CPU trošak je takođe vrlo nizak (0,0001935), što ukazuje na mnogo efikasniji pristup u pogledu potrošnje resursa.

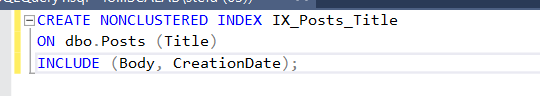
Klasterisani indeks omogućava brz pristup podacima putem direktnog skeniranja indeksa, dok se bez klasterisanog indeksa mora izvršiti celokupno skeniranje tabele i sortiranje, što značajno povećava troškove. Korišćenje klasterisanog indeksa značajno poboljšava performanse upita, smanjuje vreme izvršavanja, smanjuje opterećenje na I/O sistem i povećava efikasnost. Bez klasterisanog indeksa, upit je znatno sporiji, zahteva više memorije, koristi paralelizam i opterećuje sistem. Kod plana izvršenja bez klasterisanog indeksa, dobijamo preporuku od SQL Servera za kreiranje klasterisanog indeksa kako bi se poboljšale performanse.

## 4.3. Neklasterisani indeksi

Kod neklasterisanih indeksa ne menja se fizički redosled podataka, kao što je to slučaj kod klasterisanih indeksa. Umesto toga, neklasterisani indeksi stvaraju zasebnu strukturu koja se odnosi na redove podataka. Ova struktura sadrži vrednosti neklasterisanog ključa indeksa i pokazivače (pointere) na redove podataka koji sadrže te vrednosti ključa.

Ovi indeksi se čuvaju odvojeno od podataka tabele, obično u strukturi B+ stabla, koja sadrži ključeve indeksa i pokazivače na stvarne redove podataka. Takođe, tabela može imati više neklasterisanih indeksa, jer oni ne utiču na fizičko skladištenje redova podataka.

SQL Server omogućava neklasterisanim indeksima da uključuju kolone koje nisu deo ključa, što može poboljšati performanse upita, omogućavajući pokrivanje više scenarija upita bez potrebe za dodatnim pretragama. [4]



Slika 4.7. Primer uključivanja dodatnih kolona u indeks

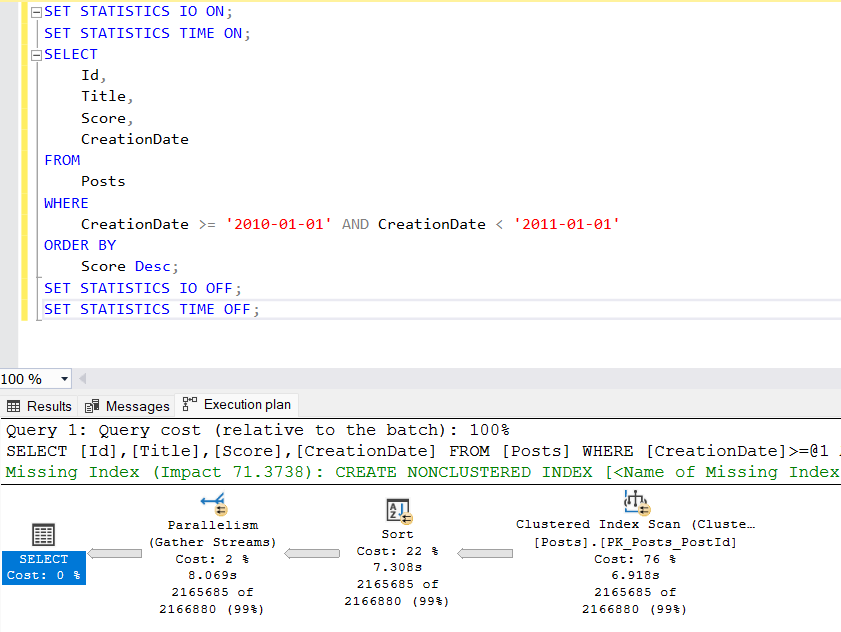
I klasterisani i neklasterisani indeksi mogu biti jedinstveni (unique).

Iako neklasterisani indeksi mogu značajno poboljšati performanse, oni dolaze sa određenim troškovima. Svaki indeks zauzima dodatni prostor na disku i može povećati vreme potrebno za dodavanje, brisanje i ažuriranje podataka, jer se indeksne strukture moraju ažurirati kako se podaci u tabeli menjaju. Zbog toga je važno pažljivo odabrati koje kolone će biti indeksirane, kako bi se postigla ravnoteža između performansi upita i troškova održavanja indeksa.

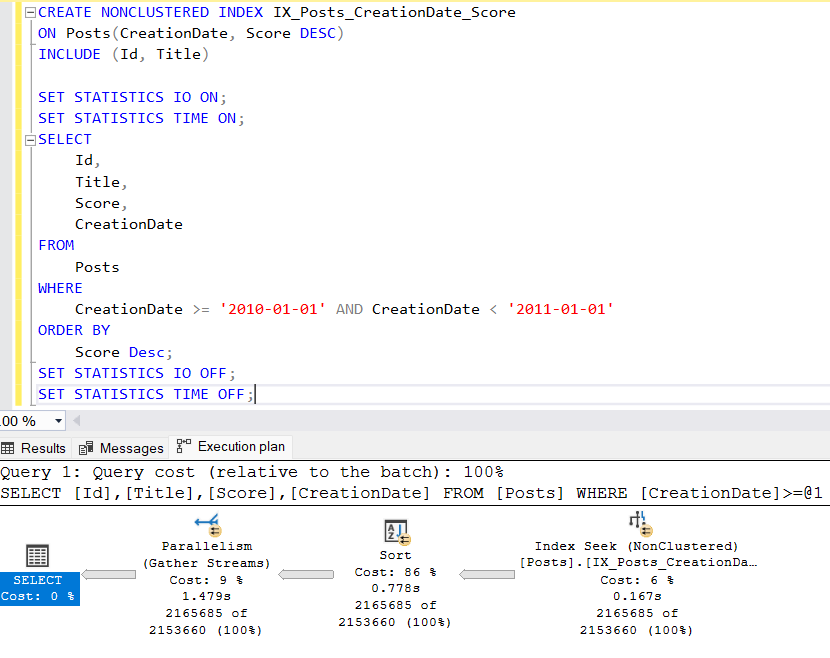
Takođe, previše neklasterisanih indeksa na tabeli može dovesti do smanjenja performansi tokom operacija ažuriranja, jer se svaki indeks mora ažurirati kako se podaci menjaju. Iz tog razloga, preporučuje se optimizacija indeksa na osnovu konkretnih potreba upita koji se najčešće izvode na bazi podataka.

### 4.3.1. Primer

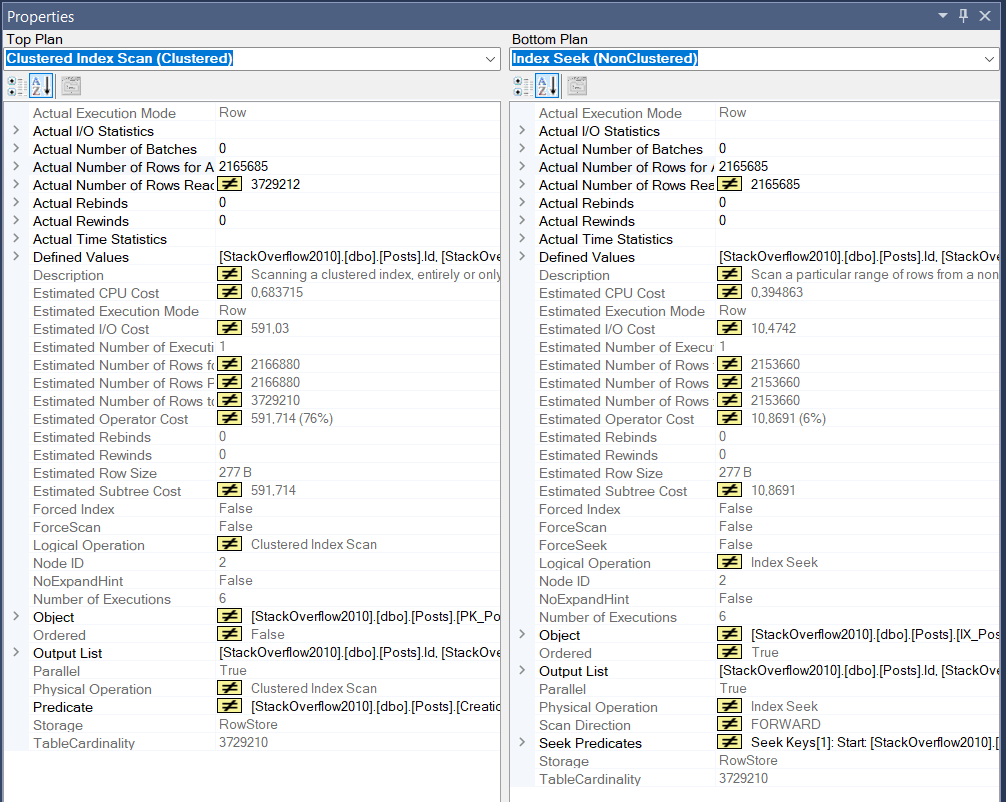
Na sledećem primeru je data demonstracija izvršenja istog upita prilikom nepostojanja i postojanja neklasterisanog indeksa.



Slika 4.8. Primer izvršenja upita bez neklasterisanog indeksa.



Slika 4.9. Primer izvršenja upita sa postojanjem neklasterisanog indeksa.



Slika 4.10. Poređenje *clustered index seek* i non*clustered index seek* operacije.

Na osnovu plana izvršenja možemo uočiti da za izvršavanje prvog upita SQL Server je morao da skenira celu tabelu ili veliki deo tabele jer koristi klasterisani indeks. U ovom slučaju, taj indeks je na primarnom ključu. Na slici možemo videti da je broj skeniranih redova 2165685 i broj pročitanih redova 3729212 što znači da je moralo biti pročitan veliki broj redova, čak i više nego što je zapravo bilo potrebno za izvršenje upita.  
Troškovi I/O i CPU su mnogo veći (*Estimated I/O Cost* 591.03, *Estimated CPU Cost 0.683715*), što ukazuje da je ovaj upit bio teži za izvršenje.

Drugi upit korist*i Nonclustered Index Seek* operaciju, što je znatno efikasniji način pretrage, jer SQL Server koristi specifičan indeks koji smo kreirali na kolonama *CreationDate* i *Score*. U ovom slučaju, SQL Server može direktno da potraži odgovarajuće redove u indeksu bez skeniranja cele tabele, što znatno smanjuje broj pročitanih redova. Vidimo da je broj pročitanih redova sada 2165685, što je mnogo manje u poređenju sa prethodnim upitom.   
Takođe, Troškovi I/O i CPU su značajno manji (*Estimated I/O Cost* 10.47, *Estimated CPU Cost* 0.394863), što pokazuje bolju optimizaciju i brže izvršenje.

## 4.4. Particionisanje tabela za poboljšanje performansi

Particionisanje je proces u bazi podataka gde se veoma velike tabele dele na više manjih delova. Razdvajanjem velike tabele na manje, pojedinačne tabele, upiti koji pristupaju samo delu podataka mogu se izvršavati brže jer ima manje podataka za pretragu. Glavni cilj particionisanja je da pomogne u održavanju većih tabela i da smanji ukupno vreme odziva za čitanje i učitavanje podataka za određene SQL operacije. Postoje dva tipa particionisanja, a to su horizontalno i vertikalno particionisanje.

Horizontalno particionisanje je proces u bazi podataka gde se velike tabele dele horizontalno na manje delove, odnosno redovi tabele se dele u manje particije radi lakšeg upravljanja. Svaka particija sadrži određeni broj redova. Ovo je korisno za skaliranje baze podataka i povećanje performansi upita, posebno kada se radi sa tabelama sa velikim brojem redova.

Vertikalno particionisanje se odnosi na deljenje tabele po kolonama, umesto po redovima kao što je slučaj kod horizontalnog particionisanja. Ovaj proces je koristan kada postoje kolone koje se retko koriste ili imaju veliku veličinu, jer omogućava da se ove kolone izdvoje u posebne tabele. Na ovaj način se smanjuje potreba za skeniranjem celokupne tabele prilikom izvršavanja upita.

Obe tehnike mogu doprineti efikasnijem upravljanjem velikim količinama podataka. Horizontalno omogućava bolje skaliranje, dok vertikalno particionisanje omogućava efikasnije koriščenje resursa sistema.

MSSQL nudi tehniku eliminacije particija koja nisu od značaja za sam upit. Naime, query optimizer pristupa samo relevantnim particijama kako bi zadovoljio filtere upita. Na ovaj način performanse mogu ostati iste ako se pristupa velikim brojem particija, ali ako se pristupa manjem broju to će značajno poboljšati performanse. Na primer, ako imamo tabelu koja je podeljena na 100 particija, i drugom slučaju na 500 particija na osnovu kolone A. I ako imamo upit koji ima where klauzulu i filtrira po koloni A, taj upit može se izvršiti brže u drugom slučaju ako se u oba slučaja pristupa samo jednoj particiji, zato što u tom slučaju particija ima manje redova. Takođe, ako bi se filtriranje radilo po nekoj drugoj koloni onda bi se moralo vršiti skeniranje svih particija.

Upiti koji uključuju operatore TOP ili MIN/MAX nad kolonama koje nisu kolone za particionisanje mogu dati lošije performanse sa particionisanjem jer moraju da se evaluiraju sve particije. [5]

Slika na kojoj se nalazi tekst, snimak ekrana, Font, broj

Opis je automatski generisan  
Slika 4.11. Primer vertikalnog particionisanja

Slika na kojoj se nalazi tekst, snimak ekrana, Font, broj

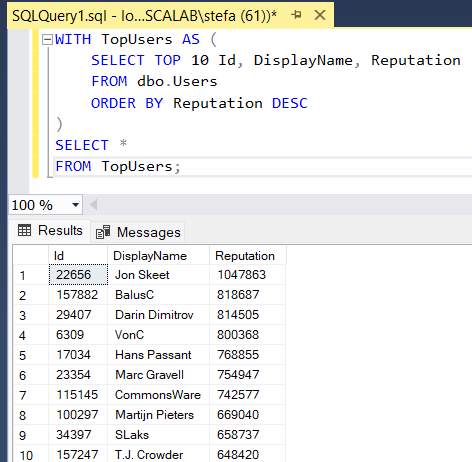
Opis je automatski generisan

Slika 4.12. Primer vertikalnog particionisanja

## 4.5. Korišćenje CTE (Common Table Expressions) i privremenih tabela

### 4.5.1. Korišćenje CTE (Common Table Expressions)

CTE izrazi u Microsoft SQL Serveru su specijalne funkcije koje se koriste da pojednostave kompleksne SQL upite omogućavajući bolju organizaciju, čitljivost i ponovnu upotrebljivost. CTE može se koristiti u okviru SELECT, INSERT, UPDATE ili DELETE naredbi, kao i u definisanju pogleda. Na taj način, izolovanjem specifičnih delova upita, CTE-ovi mogu pomoći u poboljšanju performansi. SQL Server može optimizovati izvršenje upita kroz keširanje rezultata CTE-a, smanjujući potrebu za ponovnim izvršavanjem istog podupita.

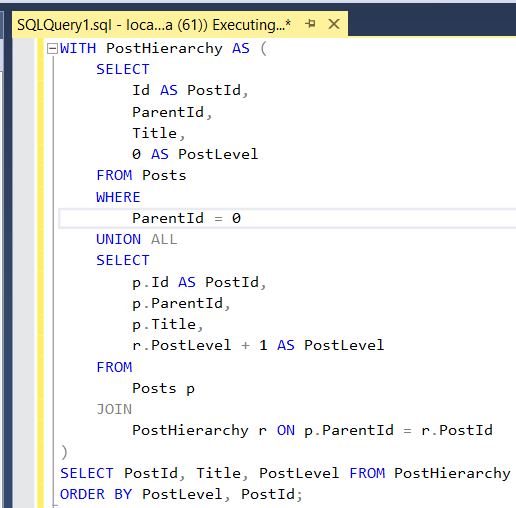


Slika 4.13. Primer definisanja i izvršenja CTE-a.

Postoje dva tipa CTE-ova, a to su rekruzivni i nerekurzivni.

Nerekurzivni CTE-ovi su jednostavniji od rekurzivnih, i oni se koriste prvenstveno za poboljšanje čitljivosti i organizacije SQL upita. Oni se ne referenciraju na sebe, što ih čini suštinski jednakim izvedenim tabelama ili podupitima. Osim što omogućavaju razbijanje kompleksnih upita u manje celine koje se mogu pozivati više puta, CTE omogućava nezavisnu optimizaciju delova upita od strane Query optimizera, što dovodi do poboljšanja performansi.

Rekurzivni CTE-ovi su kompleksniji od nerekurzivnih, i oni omogućavaju rekuretno izvršenje upita. Rekurzivni CTE-ovi mogu se referencirati na sebe, omogućavajući ponavljanja istog izraza radi dobijanja podataka na hierarhijski način.



Slika 4.14. Primer rekurzivnog CTE-a.

Ono što je bitno napomenuti da oba tipa važe samo u toku izvršenja upita.

### 4.5.2. Korišćenje privremenih tabela (temporary tables)

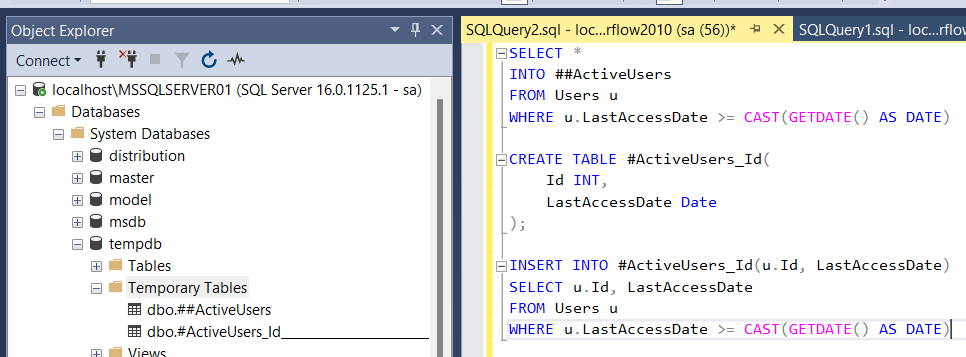
Privremene tabele u Microsoft SQL Serveru se koriste za smeštanje privremenih rezultata i pojednostavljanje kompleksnih SQL upita. Ove tabele postoje privremeno u bazi podataka i dizajnirane su da traju u toku sesije. One se kreiraju u tempdb bazi podataka i automatski se brišu na kraju sesije ili eksplictinim brisanjem.

Na osnovu dužine postojanja privremenih tabela možemo ih podeliti u lokalne i globalne privremene tabele.

Lokalne privremene tabele su definisane jednim simbolom # nakon čega sledi ime privremene tabele. Važenje lokalnih privremenih tabela je jednako trajanju sesije, i nakon zatvaranja sesije automaski se brišu. Takođe, vidljive su samo sesiji koja ih je kreirala.

Globalne privremene tabele su definisane dvostrukim simbolom # nakon čega sledi ime privremene tabele i one su vidljive svim sesijama. Brišu se tek nakon zatvaranja poslednje sesije koja se na njih referencira.

Postoje dva načina kreiranja privremenih tabela u Microsoft SQL Serveru korišćenjem „SELECT INTO” i „CREATE TABLE” naredba. „SELECT INTO” naredba automatski kreira strukturu nove privremene tabele na osnovu rezultata upita, dok „CREATE TABLE” zahteva eksplicitno definisanje strukture tabele preko „CREATE TABLE” naredbe. [6]

Slika 4.15. Primer kreiranja privremenih tabela.

Privremene tabele i CTE izrazi deluju slično, ali postoje veoma bitne razlike u:

* Izvršenju: CTE se izvršava prilikom svakog referenciranja u nekom delu upita radi pribavljanja podataka, dok kod privremenih tabela pribavljanje podataka se radi samo prilikom kreiranja.
* Čuvanju podataka: Privremene tabele skladište podatke fizički u memoriji tokom celog trajanja sesije, a CTE ne čuva podatke fizički već ih obrađuje dinamički prilikom svakog poziva.
* Trajanju: CTE prestaje da postoji nakon završetka izvršenja upita u kojem je definisan, dok privremene tabele ostaju dostupne dok se ne obrišu eksplicitnim brisanjem ili se brišu automatski nakon zatvaranja sesije.

Privremene tabele i CTE imaju različite svrhe i načine upotrebe u SQL-u. Tako da na osnovu samog zahteva problema treba da se izabere korišćenje ili CTE ili prevremenih tabela.

# 5. Napredne tehnike optimizacije upita

## 5.1. Heuristička optimizacija

U SQL Serveru, optimizacija upita je složen proces koji dodatno uključuje kombinaciju heurističkih pravila i algoritama za pronalaženje najefikasnijeg plana izvršavanja. Cilj je minimizirati vreme izvršavanja upita i potrošnju resursa, poput memorije i procesorske snage, birajući optimalne operacije pretrage, spajanja i filtriranja podataka.

Heuristička optimizacija u SQL Serveru uključuje nekoliko ključnih komponenti koje doprinose njenoj efikasnosti.

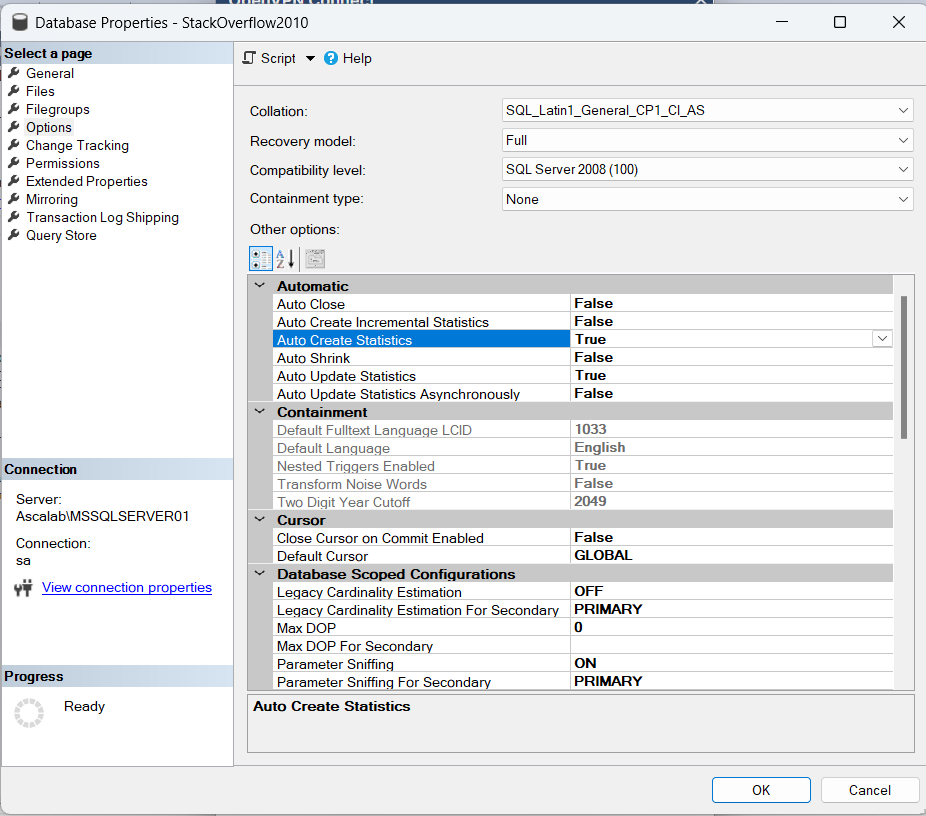
1. Procena troškova: Jedna od ključnih faza heurističke optimizacije je procena troškova različitih planova izvršenja. SQL Server koristi heurističke metode za procenu troškova na osnovu statističkih podataka o distribuciji podataka, sistemskim parametrima i karakteristikama hardvera. Na primer, trošak selekcije i kardinalnosti, tj. procena broja rezultujućih redova, igra ključnu ulogu u određivanju optimalnog plana izvršenja. Heuristički optimizatori teže da odaberu planove sa nižim procenjenim troškovima, čime se obezbeđuje efikasnija upotreba resursa i bolje performanse upita.
2. Redosled spajanja: Kod upita koji uključuju više tabela, optimalan redosled spajanja je ključan za smanjenje vremena izvršavanja. SQL Server koristi heuristike kao što su algoritmi pohlepe (*greedy algorithm*) i dinamičko programiranje kako bi odabrao najefikasniji redosled spajanja. Ove metode omogućavaju da se brzo ispitaju različite opcije za spajanje tabela, dok se kardinalnost, odnosno veličina međurezultata, koristi kao kriterijum za donošenje odluka o redosledu spajanja. SQL Server pokušava da prvo spoji tabele sa manjim brojem redova kako bi smanjio količinu podataka koja se procesira u kasnijim fazama.
3. Preuređenje upita: Heuristička optimizacija uključuje i transformaciju upita u drugačijem obliku koji je pogodniji za izvršavanje. SQL Server koristi heuristike kako bi identifikovao mogućnosti za promenu strukture upita, na primer, pomeranjem operacija ili razbijanjem upita na manje podupite.
4. Selektovanje indeksa: Indeksi igraju ključnu ulogu u ubrzavanju upita, a heuristička optimizacija pomaže u izboru najboljih indeksa za konkretne upite. SQL Server koristi heurističke pristupe kako bi odabrao indekse koji imaju najveći uticaj na brzinu izvršavanja upita, uzimajući u obzir selektivnost upita i trošak aktiviranja indeksa. Ova strategija omogućava da se smanji vreme pristupa podacima i ubrza izvršavanje upita. [7]

Heuristička optimizacija u SQL Serveru predstavlja efikasan pristup za ubrzanje procesa optimizacije upita, posebno kod složenih sistema sa velikim količinama podataka. Korišćenjem heurističkih metoda za procenu troškova, redosled spajanja, selekciju indeksa i prepisivanje upita, SQL Server može generisati planove izvršenja koji značajno poboljšavaju performanse upita.

## 5.2. Statistika

SQL Server koristi statistiku kako bi dobio precizne procene o distribuciji podataka u tabelama i indeksima. Ove informacije omogućavaju optimizatoru upita da odabere najefikasniji plan izvršavanja, čime se optimizuje upotreba resursa i smanjuje vreme obrade. Statistički podaci sadrže informacije o broju redova, rasponu vrednosti, učestalosti i raznolikosti podataka, što pomaže optimizatoru da tačno predvidi troškove različitih operacija, poput skeniranja indeksa (*index scan*) i pretrage indeksa (*index seek*). Time se smanjuje verovatnoća prekomerne potrošnje resursa tokom izvršavanja upita. [8]

SQL Server automatski kreira statistiku prilikom kreiranja indeksa na tabelama ili indeksiranim pogledima. Pored toga, optimizator upita može generisati statistiku za neindeksirane kolone tokom izvršavanja upita, ukoliko je omogućena opcija *Auto Create Statistics*. Ova funkcionalnost omogućava dinamičko kreiranje statističkih objekata na osnovu potreba optimizatora, što dodatno doprinosi poboljšanju performansi upita.



Slika 5.1. *Auto Create Statistics* opcija.

Statistika uključuje kreiranje histograma i *density* vektora.

### 5.2.1. Histogram

Histogram je ključna komponenta statističkih objekata u SQL Serveru, koja pruža detaljne informacije o raspodeli vrednosti u jednoj koloni. SQL Server kreira histogram kada se statistika generiše za kolonu, i on se sastoji do 200 korpi (*buckets*), koje predstavljaju određene raspone vrednosti u toj koloni.

*Bucket* u histogramu SQL Servera predstavlja korak unutar histograma, gde svaki korak sadrži:

1. *range\_high\_key*: Gornja granica raspona vrednosti u tom koraku.
2. *equal\_rows:* Broj redova koji imaju vrednost tačno jednaku *range\_high\_key*.
3. *range\_rows:* Broj redova čije vrednosti padaju unutar raspona između dve susedne gornje granice, ali isključujući same granične vrednosti.
4. *distinct\_range\_rows*: Broj jedinstvenih vrednosti unutar tog raspona.

SQL Server kreira histogram tako što sortira kolonu i grupiše vrednosti do 200 kontinuiranih koraka , gde svaki korak opisuje raspon vrednosti, bez uključivanja graničnih vrednosti (gornja granica jednog koraka postaje donja granica sledećeg). Na ovaj način, optimizator upita može efikasno proceniti raspodelu podataka i doneti odluke o optimizaciji, kao što je izbor između skeniranja i pretrage indeksa.

Ovaj proces je podeljen u tri faze:

1. Inicijalizacija histograma: Sortirane vrednosti kolone se procesiraju, i prikupljaju se gornje granice raspona (*range\_high\_key*) i broj redova za svaku vrednost (*equal\_rows*).
2. Spajanje koraka (*Scan with bucket merge*): Ako je potrebno kreirati novi raspon, neki od postojećih koraka se spajaju kako bi se minimizirao gubitak informacija. Na ovaj način se osigurava da broj koraka ostane maksimalno 200, dok se zadržava preciznost.
3. Konsolidacija histograma: Dodatni koraci mogu biti spajani ako se proceni da neće doći do značajnog gubitka informacija, što rezultira histogramom sa manje od 200 koraka, čak i kada postoji više od 200 različitih vrednosti u koloni. [8]

Histogram je najkorisniji kada je potrebno proceniti selektivnost upita, tj. koliko redova odgovara specifičnoj vrednosti ili rasponu vrednosti u koloni. Na primer, ukoliko upit sadrži uslov koji filtrira podatke po određenoj vrednosti, optimizator koristi histogram kako bi procenio koliko redova ispunjava taj uslov. Ova procena direktno utiče na odabir strategije izvršavanja upita, kao što je izbor između *index seek* i *index scan*.

### 5.2.2. Density vector

*Density vector* predstavlja meru koja opisuje količinu duplikata u određenoj koloni ili kombinaciji kolona u tabeli. Izračunava se kao 1/(broj jedinstvenih vrednosti). Ova vrednost pomaže SQL Server optimizatoru upita u proceni kardinalnosti za upite koji vraćaju više kolona iz iste tabele ili indeksiranog pogleda. [8]

Veća količina jedinstvenih vrednosti ukazuje da je selektivnost vrednosti veća, što znači da će manje redova odgovarati uslovu upita. Nasuprot tome, veća gustina ukazuje na to da postoji više duplikata, pa će veći broj redova ispuniti dati kriterijum.

Optimizator koristi gustinu kako bi preciznije procenio kardinalnost pri izvršavanju upita koji uključuju više kolona. Na taj način optimizator može odabrati efikasniji plan izvršavanja, kao što je odabir odgovarajućeg indeksa ili strategije za pristup podacima, čime se poboljšavaju performanse.

### 5.2.3. Kreiranje statistike

Statistike se automatski kreiraju u sledećim situacijama:

* Kreiranje indeksa: Kada se kreira indeks nad tabelom ili pogledom, optimizator automatski kreira statistiku za kolone koje su deo indeksa. U slučaju filtriranih indeksa, kreiraju se filtrirane statistike na istom podskupu redova.
* *Auto Create Statistics* opcija: Ako je omogućena opcija, SQL Server automatski kreira statistiku za kolone koje se pojavljuju u predikatu upita, čak i ako za njih ne postoji indeks.

Pored automatskog kreiranja, SQL Server omogućava da se ručno kreiraju statistike koristeći T-SQL naredbu *CREATE STATISTICS*. Ovo je korisno kada želimo da kreiramo statistike za specifične kolone ili skupove kolona koje nisu pokrivene indeksima ili za složene upite koji koriste više kolona u predikatu.

Pošto se u bazi podataka dešavaju česte promene, tako i same statistike trebaju da se ažuriraju periodično kako bi njihov uticaj na izvršenje upita bio tačan. Ažuriranje statistika moguće je ručno ili automatski:

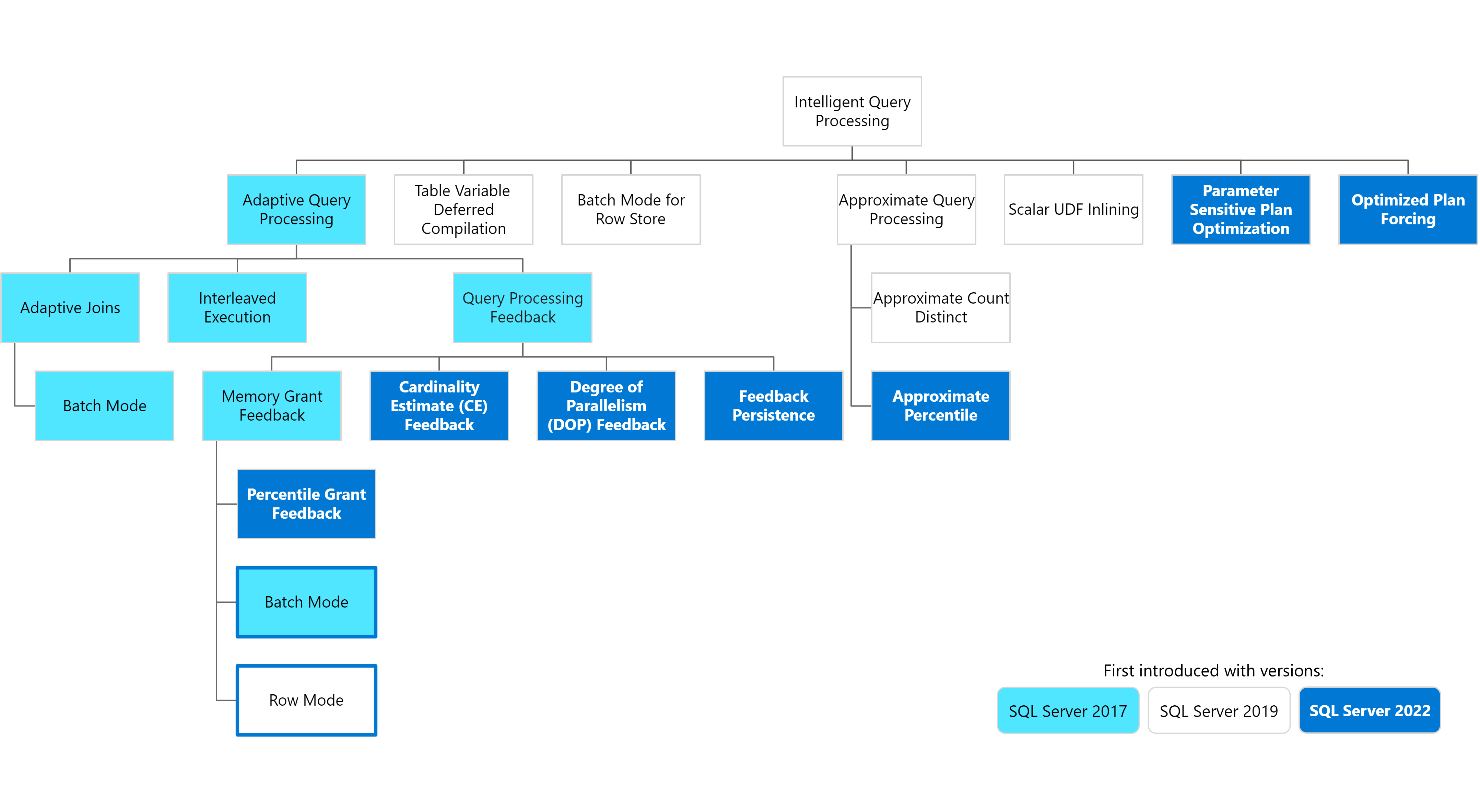
* *Auto Update Statistics* opcija: Ako je omogućena opcija, SQL Server automatski ažurira statistike kada detektuje značajne promene u distribuciji podataka. Ovaj mehanizam funkcioniše tako što ažurira statistike samo kada su potrebne za generisanje plana upita.
* Ručna ažuriranja: Postoje situacije kada ručno ažuriranje statistika može značajno poboljšati performanse. Na primer, ako se izvrši velika promena u podacima (poput masovnog umetanja novih redova ili brisanja). Takođe, umetanje u kolone sa rastućim ili opadajućim vrednostima, poput *identity* kolona ili kolona sa datumima, može zahtevati češće ažuriranje statistika, jer novi podaci mogu značajno promeniti distribuciju vrednosti. Ili ako primetimo da je izvršavanje upita postalo sporije, potrebno je proveriti kada su statistike poslednji put ažurirane i izvršiti ručno ažuriranje ako je potrebno. Ažuriranje statistika moguće je izvršiti pomoću UPDATE STATISTICS ili procedure sp\_updatestats.

Iako SQL Server automatski kreira i ažurira statistike, postoje specifične situacije kada je potrebno ručno intervenisati kako bi se postigle bolje performanse. Ručno kreiranje i ažuriranje statistika u određenim situacijama može značajno poboljšati performanse upita i celokupnu efikasnost.

## 5.3. Inteligentna obrada upita (Intelligent Query Processing - IQP)

Inteligetna obrada upita predstavlja skup tehnika koje SQL Server koristi kako bi unapredio performanse izvršenja upita automatskim prilagođavanjem planova izvršenja, bez potrebe za ručnim ažuriranjem upita. Ovaj mehanizam predstavlja deo napredne optimizacije upita i oslanja se na učenje iz prethodnih izvršenja kako bi se postigli bolji rezultati prilikom ponovnog izvršenja upita. [9]

Osnovna svrha IQP je smanjenje nepredviđenih troškova izvršenja, koji mogu nastati zbog nekonzistentnih procena selektivnosti ili promenljivih uslova u bazi podataka. IQP omogućava SQL Serveru da reaguje na promene u opterećenju i veličini podataka putem dinamičkog prilagođavanja planova izvršenja, čime se postiže veća fleksibilnost.

Slika 5.2. IQP komponente

Ovo su neke od komponenti koje IQP koristi kako bi kreirao što efikasniji plan izvršenja:

**1. Adaptivni spojevi** (*Adaptive joins*) - Ova tehnika omogućava SQL Serveru da odloži odluku o vrsti spajanja dok se o trenutka kada se dovoljno podataka ne prikupi tokom izvršenja upita. Tačnije, ova tehnika dinamički bira najoptimalniji algoritam za spajanje na osnovu broja redova koji se obrađuje, čime se smanjuje mogućnost pogrešne procene.

**2. Preplitano izvršenje** (*Interleaved Execution*) - SQL Server optimizator pravi plan izvršavanja na osnovu procena broja redova pre nego što upit bude izvršen. Kod objekata poput skalarnih funkcija ili TVF-ova (*Table-Valued Functions*), SQL Server ne zna tačan broj redova koji će biti vraćen, pa koristi unapred definisane procene, što može rezultirati neoptimalnim planovima. Kod ovog pristupa, SQL Server delimično izvršava deo upita koji se odnosi na TVF ili funkciju, prikuplja tačne informacije o broju redova koje će funkcija vratiti, a zatim te informacije koristi da bi kreirao tačniji i efikasniji plan izvršavanja.

**3. Povratna informacija o dodeli memorije** (*Memory Grant Feedback*) - Kada SQL Server izvršava upit, dodeljuje memoriju na osnovu procene broja redova i složenosti operacija kao što su sortiranje i agregacije. Ako se dodeli premalo memorije, SQL Server koristi *tempdb* za operacije koje nisu stale u memoriju, što može značajno usporiti izvršenje. S druge strane, previše dodeljene memorije može negativno uticati na druge upite jer se resursi ne koriste optimalno. Ova tehnika omogućava SQL Serveru da prilagodi memorijski grant (dodelu memorije) u sledećim izvršenjima istog upita na osnovu stvarne memorijske potrošnje iz prethodnog izvršavanja.

**4.** **Odložena kompilacija promenljivih tabela** (*Table Variable Deferred Compilation*) je funkcionalnost uvedena u SQL Server kako bi se poboljšalo izvršavanje upita koji koriste promenljive tabele. Kada SQL Server kompajlira upit koji sadrži promenljivu tabelu, optimizator pravi plan izvršenja bez stvarnog uvida u to koliko će redova ta promenljiva tabela sadržati. SQL Server pretpostavlja da će promenljive tabele imati vrlo malo redova. Ova pretpostavka može dovesti do loših planova izvršenja, naročito ako promenljiva tabela zapravo sadrži mnogo redova, što zahteva drugačiju strategiju izvršenja upita. Ovaj proces odlaže fazu kompilacije sve dok se stvarno ne popuni promenljiva tabela. Tek kada promenljiva tabela dobije stvaran broj redova, SQL Server kompilira upit, koristeći tačne informacije o broju redova u promenljivoj tabeli. To omogućava SQL Serveru da generiše precizniji i efikasniji plan izvršenja.  
Na primer, ako promenljiva tabela sadrži veliki broj redova, SQL Server može odabrati efikasniji plan, kao što je *Hash Join* ili *Merge Join* umesto *Nested Loops Join*, koji bi bio odabran pod uslovom da tabela sadrži mali broj redova.

**5.** ***Batch mode on rowstore* -** je funkcionalnost u SQL Serveru koja omogućava izvršavanje operacija nad klasičnim skladištima podataka u redovima (*rowstore*) koristeći prednosti *batch* režima izvršavanja. SQL Server korisiti *row mode* za izvršavanje upita nad podacima organizovanim u redovima, pri čemu se obrađuje jedan po jedan red. *Batch mode* omogućava SQL Serveru da obradi grupe redova (*batch-ove*) odjednom, što je efikasnije u poređenju sa *row mode.* Ako upit izvršava grupisanje velikog broj redova prema određenom kriterijumu, SQL Server može koristiti *batch mode on rowstore* kako bi izvršio grupisanje nad više redova istovremeno. Umesto da radi red po red, podaci će biti paralelno obrađeni u grupama, što će značajno ubrzati proces.

**6.** ***Scalar UDF Inlining* (*UDF - User Defined Function*) -** Scalar UDF Inlining je mehanizam koji automatski optimizuje skalarne funkcije tako što ih inline-uje u telo upita, čime se značajno poboljšavaju performanse i smanjuje *overhead* tradicionalnog izvršavanja UDF-ova u SQL Serveru. [9]

Glavna prednost IQP-a je mogućnost automatskog prilagođavanja u scenarijima gde se struktura podataka ili opterećenje značajno menja. Time se smanjuje potreba za ručnim podešavanjem parametara i omogućava optimizatoru da samostalno donosi bolje odluke. Osim toga, IPQ koristi znanja na osnovu prethodnih izvršenja upita i ta znanja koristi kako bi unapredio naredna izvršenja. Takođe, IQP smanjuje rizik od loših procena koje mogu nastati zbog zastarelih statistika ili nekonzistentnih podataka.

5.4. Automatsko podešavanje (Automatic Tuning)

Automatsko podešavanje u SQL Serveru je funkcionalnost koja omogućava prepoznavanje potencijalnih problema u performansama upita, preporučivanje rešenja, i automatsko ispravljanje identifikovanih problema. Ova tehnlogija omogućava SQL Serveru da proaktivno optimizuje izvršavanje upita, bez potrebe za manuelnom intervencijom administratora.

Automatsko podešavanje nudi nekoliko mogućnosti poboljšanja izvršavanja samih upita:

* Automatsko ispravljanje planova (*Automatic Plan Correction*) opcija omogućava da SQL Server prepozna lošije performanse zbog promene plana izvršenja upita. Ako se primeti da novi plan uzrokuje slabije performanse, SQL Server može automatski primeniti prethodni stabilan plan koji je ranije funkcionisao bolje. Na ovaj način, smanjuje se rizik od negativnih posledica loših planova izvršenja.  
  Ovaj mehanizam koristi *Query Store* za praćenje istorije planova i omogućava automatsku korekciju. SQL Server kontinuirano prati učinak primenjenog plana i ako primeti da stari plan nije bolji od novog, vraća se na novi plan izvršavanja. Time se obezbeđuje da SQL Server uvek koristi najefikasniji plan za izvršavanje upita.
* Automatsko upravljanje indeksima (*Automatic Index Management*), kod ove opcije SQL Server može identifikovati indekse koje treba dodati ili ukloniti kako bi se poboljšale performanse upita. Indeksi koji se ne koriste mogu biti izbrisani, dok SQL Server može automatski kreirati nove indekse koji pomažu u ubrzanju najčešće korišćenih upita. [10]

Automatsko podešavanje koristi verifikaciju kako bi vršio praćenje samih performansi nakon primenjenih promena. Ako promena ne poboljšava performanse, automatski će biti vraćeno na prethodno stanje, odnosno biće vraćen stari plan izvršenja kao bolja opcija.

Ove dve funkcionalnosti značajno doprinose automatizaciji sistema baze, čime nas oslobađaju potrebe da dodatno vodimo računa o optimizaciji.

# 6. Alati za poboljšanje performansi izvršenja upita

Kako bi se poboljšale performanse izvršenja upita, SQL Server nudi alate i tehnike koji omogućavaju detaljnu analizu i optimizaciju procesa izvršenja. Ovi alati pružaju uvid u ponašanje upita, identifikuju potencijalne probleme i preporučuju rešenja koja mogu značajno unaprediti performanse.

U nastavku će biti predstavljeni alati koji se koriste za analizu i optimizaciju upita, zajedno sa njihovim ključnim karakteristikama i primenama u praksi.

## 6.1. SQL Server Profiler

SQL Server Profiler je alat za praćenje koji Microsoft pruža od verzije SQL Server 2000. Koristi se za praćenje aktivnosti i operacija izvršenih na određenoj SQL Server bazi podataka.

SQL Server Profiler se koristi za aktivnosti kao što su:

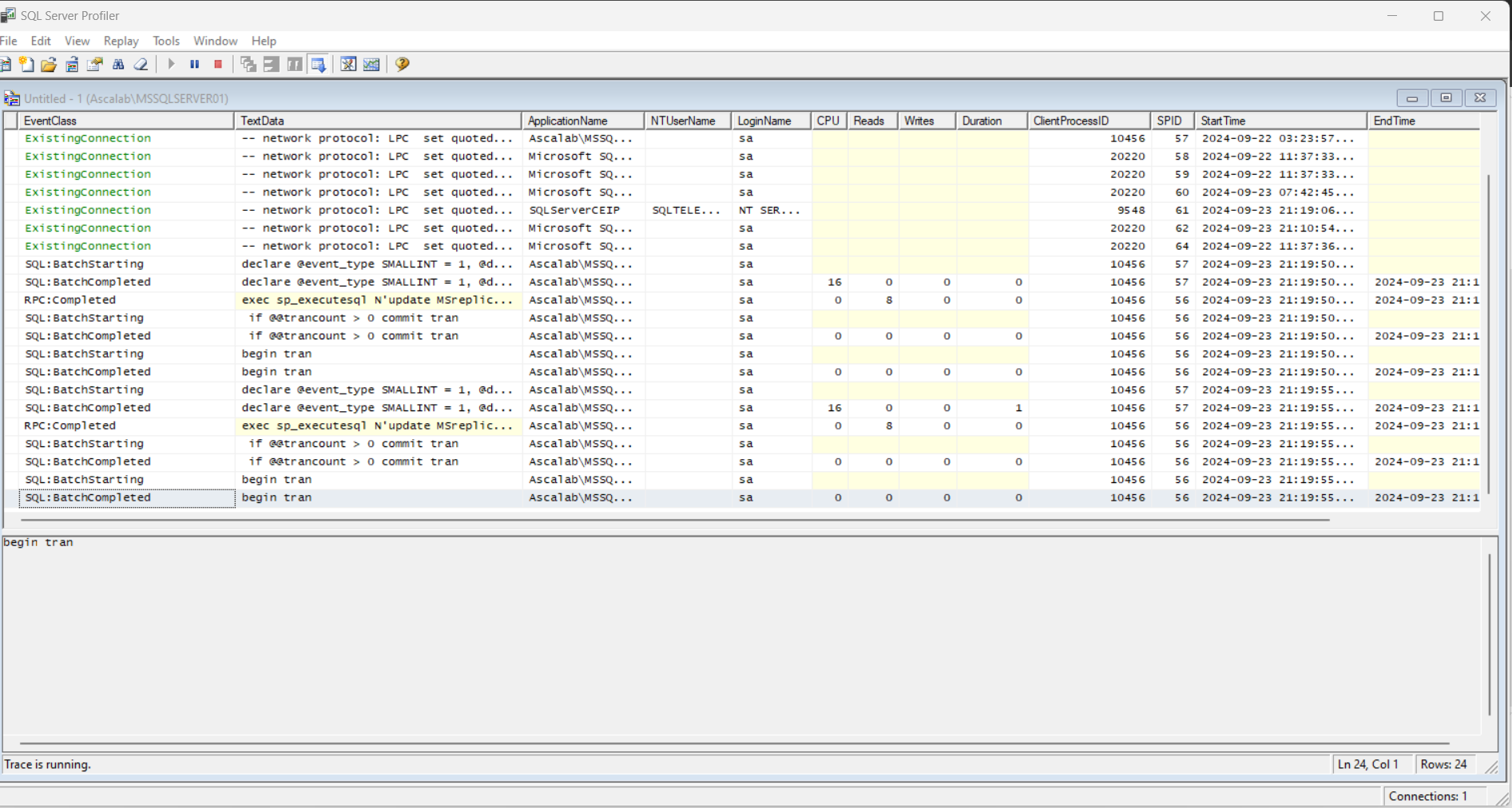
* Pronalaženje i dijagnostikovanje upita koji se izvršavaju sporo.
* Praćenje performansi SQL Server-a radi optimizacije opterećenja.
* Prolazak kroz problematične upite kako bi se otkrio uzrok problema.
* Hvatanje serije Transact-SQL naredbi koje dovode do problema. Sačuvani trag može potom replicirati problem na test serveru gde se problem može dijagnostikovati. [11]

SQL Profiler prikazuje sve podatke generisane određenim događajem u jednom redu. Svaki red sadrži više kolona koje detaljno opisuju događaj, omogućavajući dubinsku analizu i praćenje rada SQL Servera. Primeri događaja koji se mogu pratiti:

1. Povezivanje i neuspešno povezivanje prilikom login pokušaja.
2. Izvršenje Transact-SQL naredbi kao što su SELECT, INSERT, UPDATE, i DELETE.
3. Status izvršenja *batch* udaljene procedure (RPC).
4. Početak ili završetak *stored* procedure.
5. Početak ili završetak izjava unutar *stored* procedura.
6. Početak ili završetak SQL *batch-a*.
7. Greške zapisane u SQL Server log.
8. Zaključavanje ili oslobađanje zaključavanja na objektu baze podataka.
9. Otvoreni kursor.
10. Provere sigurnosnih dozvola.

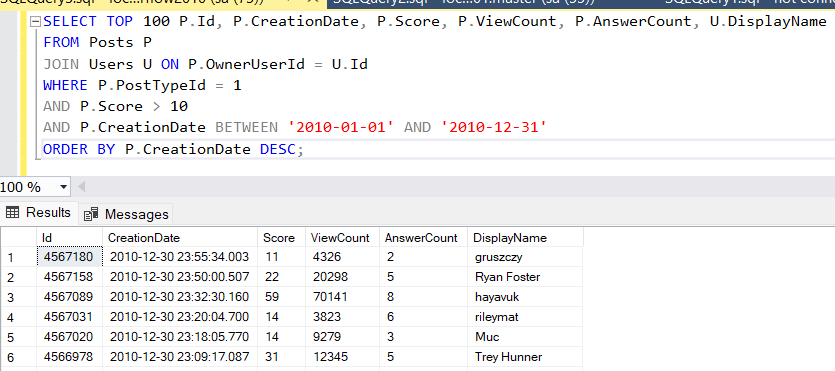
### 6.1.1. Primer korišćenja

Profiler je samostalni alat instaliran sa SQL Server-om. Možete mu pristupiti iz Windows Explorer-a ili SQL Server Management Studio-a.

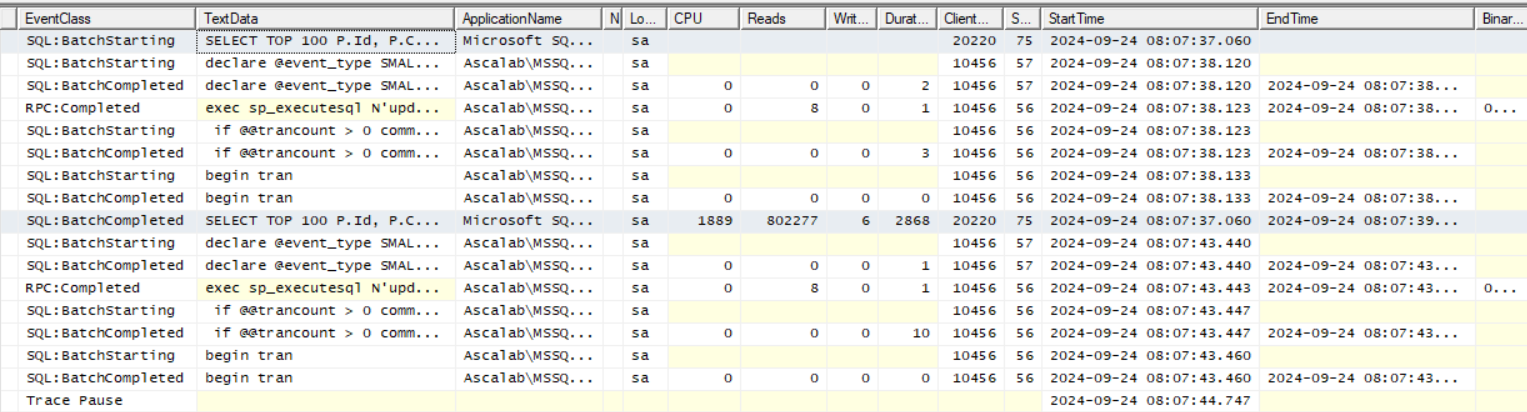
Slika 6.1. Izgled SQL Server Profiler alata.

Na samom početku potrebno je otvoriti konekciju ka željenom SQL Serveru. Nakon toga potrebno je definisati gde će se čuvati praćenje. Moguće je odabrati da se praćenje čuva u fajlu ili tabeli. Osim toga moguće je definisati filter kako bi se pratila određena vrsta događaja. Potvrđivanjem forme, Profiler započinje praćenje sistema baze.

Za samu demonstraciju rada Profilera koristićemo upit koji filtrira postove (pitanja) sa ocenom većom od 10 u toku jedne godine, i spaja ih sa informacijama o korisnicima.

Slika 6.2. Upit nad kojim vršimo testiranje.

Nakon pokretanja upita detalje izvršenja upita možemo videti u Profileru.

Slika 6.3. Izgled praćenja u Profileru nakon izvršenja upita.

Rezultati iz SQL Profiler-a:

* CPU vreme: 1889 ms
* Broj čitanja (Reads): 802277
* Broj upisa (Writes): 6
* Trajanje (Duration): 2868 ms

Utrošeno vreme CPU-a od 1889 ms ukazuje na značajno opterećenje procesora tokom izvršenja upita. Ovaj upit obrađuje veliki broj podataka, jer je upit kreiran da filtrira postove sa ocenom većom od 10 u jednoj godini, i da zatim sortira te rezultate. Opterećenje procesora ukazuje da SQL Server mora intenzivno procesirati podatke kako bi izveo ovakvu pretragu i sortiranje.

Broj čitanja (802277) je veliki broj čitanja i ukazuje na to da SQL Server mora pregledati veliki deo baze podataka da bi pronašao tražene rezultate. To može biti znak da je potrebno kreirati ili optimizovati postojeće indekse na koloni *Score* ili *CreationDate*. Veliki broj čitanja utiče na ukupnu brzinu upita, jer SQL Server mora da pristupi velikom broju stranica iz memorije ili sa diska.

Broj upisa (6), iako upit ne upisuje direktno podatke u bazu (jer je u pitanju SELECT upit), broj upisa može biti povezan sa *tempdb* bazom gde SQL Server smešta privremene podatke potrebne za izvršenje upita (npr. za sortiranje rezultata). Ovo ukazuje na mogućnost da je upit zahtevao privremenu memoriju kako bi izvršio sortiranje.

Ukupno vreme izvršenja upita je nešto manje od 3 sekunde (2868ms). Iako to nije preterano dugo za ovaj tip upita, može se dalje optimizovati smanjenjem broja čitanja kroz poboljšanje indeksa ili restrukturiranje upita.

Ovaj primer jasno demonstrira kako SQL Profiler pomaže u analizi i optimizaciji upita, omogućavajući detaljan pregled resursa koje upit koristi.

## 6.2. Database Engine Tuning Advisor

Database Engine Tuning Advisor (DETA) automatski analizira SQL upite i radno opterećenje baze (*workload*), a zatim pruža preporuke za optimizaciju, poput kreiranja ili brisanja indeksa, dodavanja particionisanja, ažuriranje ili kreiranja statistike ili promena konfiguracije baze. Glavni cilj DETA je smanjenje opterećenja na sistemske resurse i poboljšanje ukupne efikasnosti izvršavanja upita prilagođavanjem načina na koji baza podataka obrađuje dobavljanje i manipulaciju podacima.

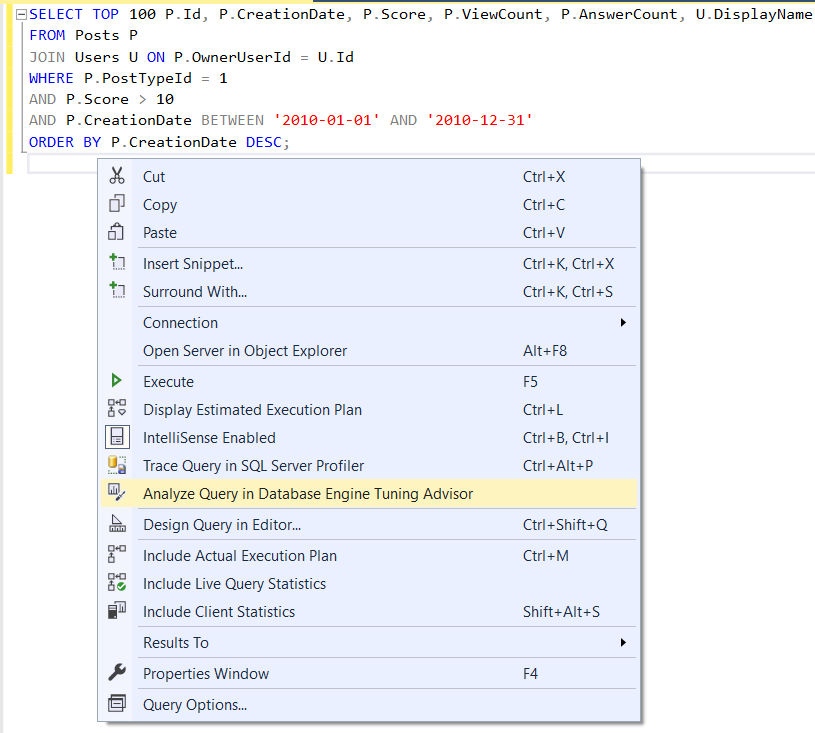
DETA nudi dva tipa korisničkog interfejsa, jedan je grafički a drugi je putem konzole.

Ovaj alat nudi mogućnost:

1. Rešavanja problema sa performansama specifičnih upita analizom njihovog izvršenja i dobijanjem preporuka za optimizaciju.
2. Optimizovanja većeg broja upita u jednoj ili više baza podataka kako bismo poboljšali ukupne performanse sistema.
3. Istraživanja potencijalnih promena u fizičkom dizajnu baze, poput dodavanja indeksa ili particionisanja, kako bismo unapred procenili njihov uticaj na performanse bez stvarne implementacije.
4. Efikasnog upravljanja skladišnim prostorom kroz uklanjanje nepotrebnih indeksa i bolje raspoređivanje podataka particionisanjem, čime se štedi prostor i povećava efikasnost.

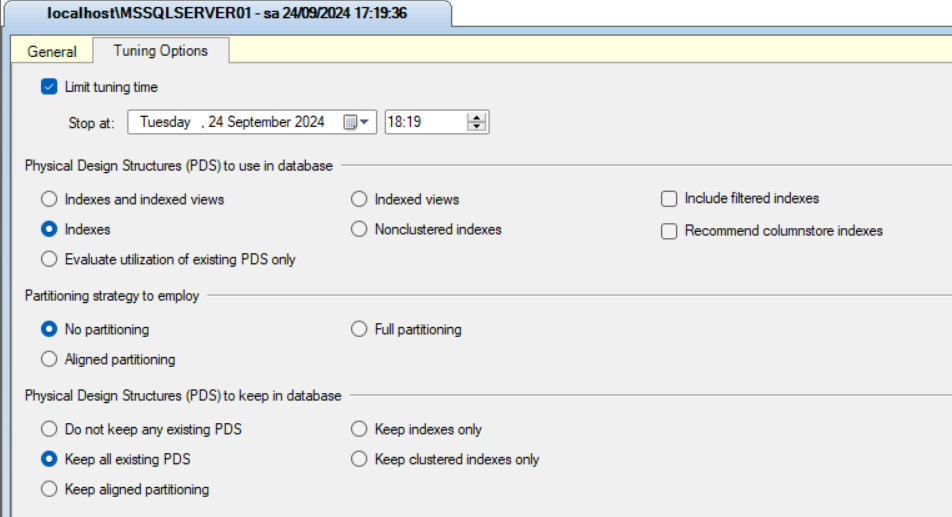
### 6.2.1. Primer korišćenja

Database Engine Tuning Advisor se može lako pokrenuti iz SSMS-a, desnim klikom na radnoj površini pa zatim *Analyze Query in Database Engine Tuning Advisor.* Na taj način možemo da pokrenemo DETA za konkretni upit i izvršimo njegovu analizu.



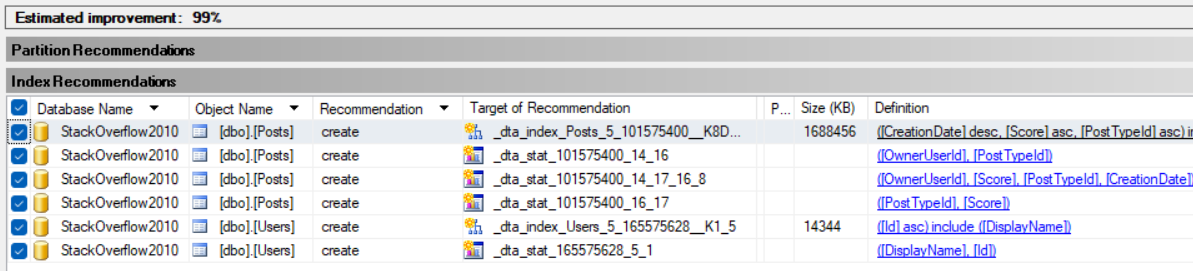
Slika 6.4. Primer pokretanja DETA alata.

Nakon pokretanja DETA alata, dobijamo prozor na kome vršimo podešavanje *tuning-a,* odnosno postavljamo opcije za koje želimo da vršimo analizu.

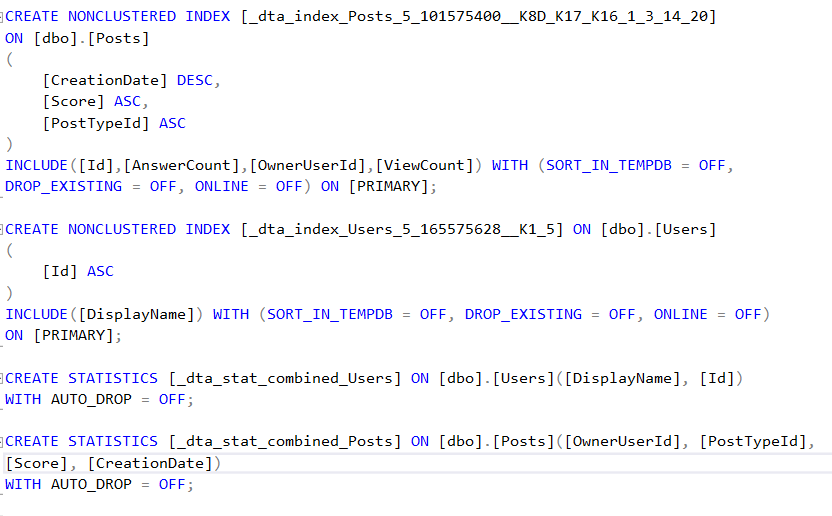
Slika 6.5. *Tuning* opcije DETA alata.

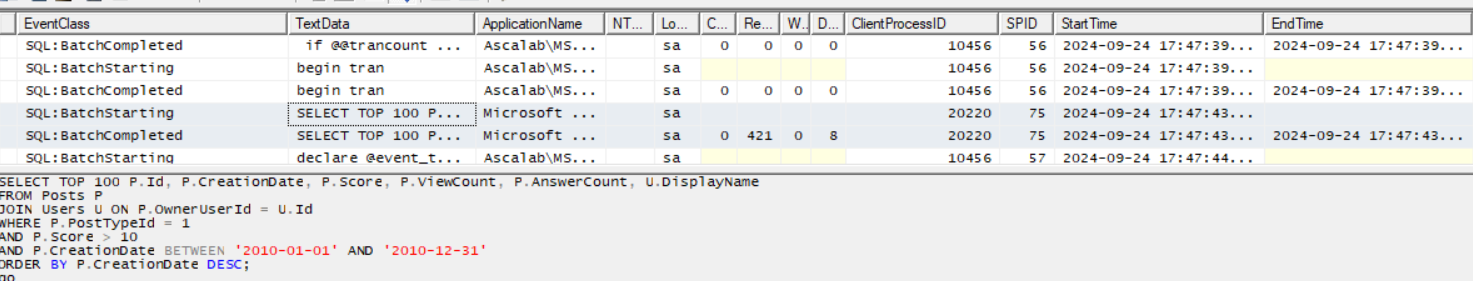
Konrektno u ovom slučaju vršićemo analizu upita koji smo koristili i u prethodnom poglavlju kada smo analizirali Profiler. Za ovaj slučaj pokrenućemo analizu sa podešavanjima datim na slici 6.5. Klikom na *Start Analysis* u gornjem levom uglu pokreće se analiza.

Nakon završetka analize dobili smo sledeće rezultate:

Slika 6.6. Rezultati analize.

DETA je generisao određene optimizacije koje se mogu primeniti u cilju poboljšanja izvršenja upita za koji smo vršili analizu. Kao rezultat smo dobili kreiranje dva neklasterisana indeksa nad tabelama *Posts* i *Users* i predlog za kreiranje statistika.

Slika 6.7. Predložene optimizacije.

Slika 6.8. Rezultati nakon izvršene optimizacije.

Nakon primene predloga DETA, kao što su kreiranje dodatnih indeksa i statistika, izvršenje upita pokazuje značajno poboljšanje performansi. U prvobitnom upitu, vreme izvršenja je bilo 2868 ms, opterećenje CPU-a od 1889 ms, sa 802277 pročitanih reda i 6 upisa. Nakon primene optimizacija, vreme izvršenja je smanjeno na samo 8 ms, uz minimalno opterećenje CPU-a (0 ms), dok je broj pročitanih redova smanjen na 421, a broj upisa na 0. Ovaj drastični pad u resursima potrebnim za izvršenje upita ukazuje na efikasnost optimizacija predloženih od strane DETA, posebno kroz kreiranje neklasterisanih indeksa i poboljšanje statistika.

# 7. Zaključak

U radu su analizirane različite tehnike i pristupi optimizacije upita u SQL Serveru sa fokusom na strukturu i čitanje plana izvršenja upita, identifikaciju uskih grla, kao i na praktične metode za unapređenje performansi upita. Indeksi, kako klasterisani tako i neklasterisani, predstavljaju osnovne alate za poboljšanje efikasnosti upita, a kroz primere su ilustrovane njihove prednosti.

Particionisanje tabela dodatno omogućava paralelizaciju pristupa podacima, smanjujući latenciju i povećavajući skalabilnost upita. Korišćenjem *Common Table Expressions* i privremenih tabela omogućava se efikasnije organizovanje podataka u složenijim upitima, ali je važno razumeti njihovu implikaciju na performanse u kontekstu plana izvršenja.

Napredne tehnike, poput heurističke optimizacije i inteligentne obrate upita, predstavljaju nove pristupe koji automatizuju deo procesa optimizacije na osnovu statističkih podataka i dinamike opterećenja baze. Kreiranje i održavanje statistika, ključni su za tačnije procene optimizatora prilikom izbora plana izvršenja.

Na kraju, automatsko podešavanje parametara (*Automatic Tuning*) nudi visok stepen samoodrživosti u optimizaciji performansi, posebno u velikim i složenim sistemima, gde manuelno podešavanje može biti previše kompleksno.

Radom je pokazano da razumevanje detalja plana izvršenja i korišćenje različitih optimizacionih tehnika može značajno unaprediti performanse SQL Server sistema, čime se postižu efikasniji upiti, smanjenje troškova resursa i brža obrada podataka.

# 8. Literatura

[1] <https://www.sqlshack.com/sql-server-execution-plans-overview/> [Poslednji put posećeno 09.04.2024.]

[2] <https://www.sqlshack.com/sql-server-execution-plans-types/> [Poslednji put posećeno 09.04.2024.]

[3] <https://www.sqlshack.com/how-to-analyze-sql-execution-plan-graphical-components/> [Poslednji put posećeno 11.04.2024.]

[4] <https://learn.microsoft.com/en-us/sql/relational-databases/indexes/indexes?view=sql-server-ver16> [Poslednji put posećeno 15.04.2024.]

[5] <https://learn.microsoft.com/en-us/sql/relational-databases/partitions/partitioned-tables-and-indexes?view=sql-server-ver16> [Poslednji put posećeno 16.04.2024.]

[6] <https://www.sqlservertutorial.net/sql-server-basics/sql-server-temporary-tables/> [Poslednji put posećeno 21.04.2024.]

[7] <https://www.geeksforgeeks.org/what-is-heuristic-optimization-in-dbms/> [Poslednji put posećeno 11.09.2024.]

[8] [https://learn.microsoft.com/en-us/sql/relational-databases/statistics/statistics](https://learn.microsoft.com/en-us/sql/relational-databases/statistics/statistics?view=sql-server-ver16) [Poslednji put posećeno 12.09.2024.]

[9] <https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing-details?view=sql-server-ver16> [Poslednji put posećeno 14.09.2024.]

[10] <https://learn.microsoft.com/en-us/sql/relational-databases/automatic-tuning/automatic-tuning> [Poslednji put posećeno 15.09.2024.]

[11] <https://learn.microsoft.com/en-us/sql/tools/sql-server-profiler/sql-server-profiler?view=sql-server-ver16> [Poslednji put posećeno 16.09.2024.]

[12] Dataset: <https://www.brentozar.com/archive/2015/10/how-to-download-the-stack-overflow-database-via-bittorrent/>