

**Univerzitet u Nišu**

**Elektronski fakultet**

**Katedra za računarstvo**

**Seminarski rad**

**Optimizacija upita kod Microsoft SQL Server baze podataka**

**Sistemi za upravljanje bazama podataka**

|  |  |  |
| --- | --- | --- |
| **Mentor**: |  | **Student**: |
| Dr Aleksandar Stanimirović |  | Miloš Veljanovski 1559 |

**Niš, april 2023.**

**Sadržaj**

[1. Uvod 3](#_Toc132766702)

[1.1. Uvod u SQL 3](#_Toc132766703)

[1.2. Uvod u SQL Server 3](#_Toc132766704)

[1.3. Komponente SQL Servera 4](#_Toc132766705)

[2. Obrada upita 4](#_Toc132766706)

[2.1. Uvod 4](#_Toc132766707)

[2.2. Kako SQL Server radi 5](#_Toc132766708)

[2.3. Svrha optimizacije SQL upita 6](#_Toc132766709)

[2.4. Metrike za analizu performansi upita 6](#_Toc132766710)

[2.4.1. Vreme izvršavanja 6](#_Toc132766711)

[2.4.2. IO Statistika 7](#_Toc132766712)

[2.4.3. Plan izvršenja 8](#_Toc132766713)

[3. Tehnike optimizacije SQL upita 9](#_Toc132766714)

[3.1. Indeksiranje 10](#_Toc132766715)

[3.1.1. Smernice za izbor indeksa 11](#_Toc132766716)

[3.2. Selekcija 11](#_Toc132766717)

[3.3. Izbegavajte upotrebu SELECT DISTINCT 12](#_Toc132766718)

[3.4. INNER JOIN naspram WHERE klauzule 13](#_Toc132766719)

[3.5. TOP komanda 13](#_Toc132766720)

[3.6. IN naspram EXISTS 14](#_Toc132766721)

[3.7. Petlje u odnosu na grupno umetanje/ažuriranje 15](#_Toc132766722)

[3.8. Sažetak tehnika optimizacije 17](#_Toc132766723)

[4. Uskladištene procedure 18](#_Toc132766724)

[4.1. Uloga uskladištenih procedura u optimizaciji upita 18](#_Toc132766725)

[5. Zaključak 19](#_Toc132766726)

[6. Literatura 20](#_Toc132766727)

[7. Listing 21](#_Toc132766728)

# 1. Uvod

## **1.1. Uvod u SQL**

SQL (Structured Query Language) je standardni jezik za pristupanje, komunikaciju i manipulaciju relacionim bazama podataka. SQL se koristi za različite operacije na bazi podataka kao što su kreiranje tabela, ubacivanje podataka u tabele, modifikacija podataka u tabelama i brisanje podataka iz tabela. SQL takođe omogućava način za dobijanje podataka iz jedne ili više tabela u bazi podataka korišćenjem upita. Postoje nekoliko komponenti (jezika) kod SQL-a:

* DDL (Data Definition Language): Koristi se za kreiranje i modifikaciju objekata baze podataka kao što su tabele, indeksi i pregledi. Uključuje naredbe za stvaranje i promenu šeme baze podataka. Primeri: CREATE, DROP, ALTER, TRUNCATE.
* DML (Data Manipulation Language): Koristi se za ubacivanje, ažuriranje i brisanje podataka iz tabela. Uključuje naredbe za menjanje podataka u bazi podataka. Na primer: SELECT, UPDATE, INSERT, DELETE.
* DCL (Data Control Language): Koristi se za kontrolu pristupa podacima u bazi podataka. Uključuje naredbe za pristup bazi podataka. Na primer: GRANT, REVOKE.
* TCL (Transaction Control Language): Koristi se za upravljanje transakcijama u bazi podataka. Uključuje naredbe poput COMMIT, ROLLBACK.

## **1.2. Uvod u SQL Server**

SQL Server je moćni sistem za upravljanje relacionim bazama podataka (RDBMS) koji je razvio Microsoft. Nudi robustnu i skalabilnu platformu za upravljanje, skladištenje i manipulaciju podacima na organizovan i efikasan način.

SQL Server je, dakle, sveobuhvatni RDBMS koji podržava širok spektar zadataka upravljanja podacima, kao što su skladištenje, preuzimanje, analiza i manipulacija podacima. Izgrađen je na vrhu Structured Query Language (SQL), standardizovanog programskog jezika koji je dizajniran za upravljanje i upitovanje („querying“) relacionih baza podataka. SQL Server pruža obimni set alata za upravljanje i interakciju sa podacima, uključujući modeliranje podataka, upitovanje, izveštavanje i integraciju podataka.

Danas se SQL Server široko koristi u organizacijama svih veličina za razne aplikacije, od jednostavnog spremanja i dohvatanja podataka do složenih analitičkih, izveštajnih i integracijskih zadataka. Moćne karakteristike, skalabilnost i čvrsta sigurnost SQL Servera čine ga idealnim izborom za upravljanje i manipulaciju velikim količinama podataka na pouzdan i efikasan način.

## **1.3. Komponente SQL Servera**

SQL Server se sastoji od nekoliko komponenti koje zajedno pružaju sveobuhvatno rešenje za upravljanje podacima. Neke od ključnih komponenti uključuju:

* Database Engine: Glavna komponenta koja je odgovorna za skladištenje, procesiranje i osiguravanje podataka. Database Engine pruža kontrolisan pristup podacima i održava integritet i doslednost podataka.
* SQL Server Integration Services (SSIS): Moćan alat za integraciju podataka koji omogućava ekstrakciju, transformaciju i učitavanje (ETL) podataka iz različitih izvora u SQL Server bazu podataka.
* SQL Server Analysis Services (SSAS): Multidimenzionalna komponenta za analizu podataka koja omogućava kreiranje i upravljanje modelima podataka za složenu poslovnu analizu i izveštavanje.
* SQL Server Reporting Services (SSRS): Alat za izveštavanje i vizualizaciju koji omogućava kreiranje, upravljanje i dostavu interaktivnih izveštaja.
* SQL Server Management Studio (SSMS): Centralizovana upravljačka konzola koja pruža korisnički prijateljsko okruženje za upravljanje, konfigurisanje i administriranje SQL Server instanci i baza podataka.

# 2. Obrada upita

## **2.1. Uvod**

Pre nego što pređemo na glavnu temu optimizacije SQL upita, prvo je neophodno da razumemo postupak obrade upita.

Obrada upita se definiše kao skup faza koje su povezane sa izvlačenjem podataka iz baze podataka. To uključuje pretvaranje upita napisanih u jeziku visokog nivoa kao što je SQL u oblik koji može biti razumljiv za fizičku implementaciju baze podataka, tehnike optimizacije SQL upita i originalnu evaluaciju upita. Postoje tri glavna koraka uključena u obradu upita:

1. Parser i prevodilac: Prvi korak u obradi upita je parsiranje i prevođenje. Parser, kao parser u kompajlerima, proverava sintaksu upita, da li su navedene relacije prisutne u bazi podataka ili ne. Upitni jezik visokog nivoa, kao što je SQL, pogodan je za upotrebu od strane ljudi, ali je potpuno neprikladan za sistemsku internu reprezentaciju. Zato je potrebno izvršiti prevođenje. Interna reprezentacija može biti proširena forma relacione algebre.
2. Optimizacija: SQL upit može biti napisan na mnogo različitih načina. Optimizovani upit takođe zavisi od načina na koji se podaci skladište u organizaciji datoteka. Upit može takođe imati različite odgovarajuće izraze relacione algebre. Na primer:

SELECT Marks FROM Student WHERE Marks>80

Gore navedeni upit može biti napisan u dva oblika relacione algebre:

ili .

Zato će često zavisiti od implementacije fajl sistema koji je bolji.

1. Plan izvršenja: Sistematsko korak po korak izvršenje primitivnih operacija za izvlačenje podataka iz baze podataka naziva se planom evaluacije upita. Različiti planovi evaluacije za određeni upit imaju različite troškove upita. Troškovi mogu uključivati broj pristupa disku, vreme CPU-a za izvršenje upita, vreme komunikacije u slučaju distribuiranih baza podataka.

## **2.2. Kako SQL Server radi**

U SQL Server-u, obrada upita prati sličan obrazac s nekim dodatnim koracima specifičnim za sistem. Kada korisnik pošalje upit SQL Serveru, upit se obrađuje pomoću Database Engine-a, koji izvršava upit u nekoliko koraka:

1. Parsiranje i normalizacija: Ulazni upit se parsira, a njegova sintaksa i semantika se validiraju. Tokom ove faze, SQL Server takođe vrši normalizaciju, što pojednostavljuje i ponovo piše upit u kanonskom obliku.
2. Algebrizacija: Normalizovani upit se prevodi u internu reprezentaciju nazvanu upitno stablo. Ovo stablo predstavlja logičku strukturu upita i koristi se za dalju obradu i optimizaciju.
3. Optimizacija: Optimizator upita generiše efikasan plan izvršenja upita koristeći različite tehnike optimizacije zasnovane na troškovima. Optimizator razmatra faktore kao što su dostupni indeksi, metode spajanja i raspodela podataka da bi stvorio optimalni plan.
4. Kompilacija: Optimizovani plan upita se kompajlira u izvršni oblik. Ovaj korak uključuje generisanje mašinskog koda i alokaciju resursa za izvršenje upita.
5. Izvršavanje: Kompajlirani plan upita se izvršava od strane baze podataka, koja dohvata ili modifikuje podatke specificirane upitom. Baza podataka čita podatke sa diska, obrađuje ih u memoriji i vraća rezultate klijentu.

## **2.3. Svrha optimizacije SQL upita**

Glavne svrhe optimizacije SQL upita su:

* Smanjenje vremena odziva: Glavni cilj je poboljšanje performansi smanjenjem vremena odziva. Razlika između zahteva korisnika za podacima i dobijanja odgovora treba biti minimalna radi boljeg korisničkog iskustva.
* Smanjenje vremena trajanja izvođenja na procesoru: Vreme izvođenja upita na procesoru treba biti smanjeno kako bi se dobili brži rezultati.
* Povećanje propusnosti: Broj resursa kojima treba pristupiti kako bi se dobili svi potrebni podaci treba biti minimiziran. Broj redova koji se trebaju dohvatiti u određenom upitu treba biti efikasan na način da se koristi što manje resursa.

## **2.4. Metrike za analizu performansi upita**

Postoji nekoliko metrika za izračunavanje troška upita u smislu prostora, vremena, iskorišćenja CPU-a i drugih resursa.

### 2.4.1. Vreme izvršavanja

Najvažnija metrika za analizu performansi upita je vreme izvršavanja upita. Vreme izvršavanja tj. trajanje upita definiše se kao vreme koje je potrebno upitu da vrati redove iz baze podataka. Vreme izvršavanja upita možemo pronaći upotrebom sledećih komandi:

SET STATISTICS TIME ON

SELECT \* FROM [dbo].[TradingSessions];

Rezultat:

SQL Server parse and compile time:

CPU time = 0 ms, elapsed time = 2 ms.

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 0 ms.

(1167 rows affected)

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 43 ms.

Completion time: 2023-04-13T19:55:10.0371915+02:00

Primer 1. Korišćenje SET STATISTICS TIME ON komande

Korišćenjem "SET STATISTICS TIME ON", možemo videti vreme analize, vreme kompilacije, vreme izvršavanja i vreme završetka upita.

* Vreme analize i kompilacije: Vreme potrebno za analizu i kompilaciju upita kako bi se proverila sintaksa upita naziva se vreme analize i kompilacije.
* Vreme izvršavanja: Vreme procesora koje upit koristi za dobijanje podataka naziva se vreme izvršavanja.
* Vreme završetka: Tačno vreme kada je upit vratio rezultat naziva se vreme završetka.

Analizom ovih vremena, možemo dobiti jasnu sliku da li se upit izvršava u skladu sa očekivanjima ili ne.

### 2.4.2. IO Statistika

IO („input/output“) je glavni vremenski trošak prilikom pristupa memorijskim baferima za operacije čitanja u slučaju upita. Pruža uvid u latenciju i druga uska grla za izvršavanje upita. Postavljanjem STATISTICS IO ON, dobijamo broj obavljenih fizičkih i logičkih čitanja za izvršavanje upita.

SET STATISTICS IO ON

SELECT \* FROM [dbo].[TradingSessions];

Rezultat:

SQL Server parse and compile time:

CPU time = 0 ms, elapsed time = 1 ms.

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 0 ms.

(1167 rows affected)

Table 'TradingSessions'. Scan count 1, logical reads 10, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0.

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 49 ms.

Completion time: 2023-04-13T20:05:34.8235472+02:00

Primer 2. Korišćenje SET STATISTICS IO ON komande

* Logička čitanja: Broj čitanja koja su izvršena iz bafera.
* Fizička čitanja: Broj čitanja koja su izvršena sa uređaja za skladištenje jer nisu bila dostupna u baferu.

### 2.4.3. Plan izvršenja

Plan izvršenja je detaljan, korak po korak, plan obrade koji se koristi za dohvatanje redova podataka. Pomaže nam u analizi glavne faze izvršenja upita. Takođe, možemo otkriti koji deo izvršenja traje najduže i optimizovati ga.

SELECT [Caption], [SymbolCode], [MarketCap], [PlatformFeeRate]

FROM [Products] p INNER JOIN [ProductStatistics] ps

ON p.ProductID = ps.ProductID

Primer 3. Korišćeni upit za analizu plana izvršenja

Text

Description automatically generated

Slika 1. Procenjeni plan izvršenja

Kao što možemo videti gore, plan izvršenja prikazuje kojim tabelama je pristupano i koje vrste skeniranja indeksa su izvršene za dohvatanje podataka. Ako su prisutna spajanja („joins“), prikazuje kako su tabele spojene. Takođe, možemo videti detaljniju analizu svake podoperacije koja se izvršava tokom izvršavanja upita. Pogledajmo analizu skeniranja indeksa:

Table

Description automatically generated

Slika 2. Analiza skeniranja indeksa

Kao što vidimo gore, možemo dobiti vrednost broja pročitanih redova, stvarnog broja paketa, procenjenog troška operatora, procenjenog troška CPU-a, procenjenog troška podstabala, kao i procenjeni broj izvršavanja. To nam daje detaljan pregled brojnih troškova uključenih u izvršavanje upita.

# 3. Tehnike optimizacije SQL upita

Do sada smo videli kako se upit izvršava i različite mere za analizu performansi upita. Sada ćemo naučiti tehnike za optimizaciju performansi upita u SQL-u. Postoje neke korisne prakse za smanjenje troškova. Međutim, proces optimizacije je iterativan. Potrebno je napisati upit, proveriti performanse upita koristeći statistiku IO-a ili plan izvršenja, a zatim ga optimizovati. Ovaj ciklus treba slediti iterativno radi optimizacije upita. Sam SQL Server takođe pronalazi optimalan i minimalni plan za izvršenje upita.

## **3.1. Indeksiranje**

Indeks je struktura podataka koja se koristi za brzi pristup tabeli na osnovu ključa pretrage. Pomaže u minimiziranju pristupa disku za dobavljanje redova iz baze podataka. Operacija indeksiranja može biti skeniranje ili traženje. Skeniranje indeksa je prolazak kroz ceo indeks u potrazi za podudarajućim kriterijumima, dok traženje indeksa filtrira redove prema odgovarajućem filteru.

Na primer:

SELECT [Caption], [SymbolCode], [MarketCap], [PricePerUnit], [PlatformFeeRate]

FROM [Products] p INNER JOIN [ProductStatistics] ps

ON p.ProductID = ps.ProductID

INNER JOIN [ProductValuations] pv

ON p.ProductID = pv.ProductID

WHERE p.ProductID > 1

Primer 4. Korišćeni upit za analizu indeksiranja

Diagram, schematic

Description automatically generated

Slika 3. Procenjeni plan izvršenja primera za indeksiranje

U gornjem upitu, možemo videti da ukupno 63% vremena izvršenja upita odlazi na operaciju traženja indeksa. Stoga to predstavlja važan deo procesa optimizacije.

### 3.1.1. Smernice za izbor indeksa

* Indeksi bi trebalo da budu napravljeni na ključevima koji se često pojavljuju u WHERE klauzulama i naredbama za spajanje.
* Indeksi ne bi trebalo da budu napravljeni na kolonama koje se često menjaju, odnosno na kojima se često primenjuje UPDATE naredba.
* Indeksi bi trebalo da budu napravljeni na stranim ključevima gde se istovremeno vrše INSERT, UPDATE i DELETE operacije. Ovo omogućava ažuriranje master tabele bez zaključavanja weak entity-a.
* Indeksi bi trebalo da budu napravljeni na atributima koji se često zajedno pojavljuju u WHERE klauzulama koristeći AND operator.
* Indeksi bi trebalo da budu napravljeni na vrednostima ključa za sortiranje.

## **3.2. Selekcija**

Potrebno je vršiti selekciju redova koji su potrebni umesto da se biraju svi redovi. SELECT \* je veoma neefikasan jer skenira celu tabelu.

SET STATISTICS TIME ON

SELECT \* FROM [dbo].[TradingSessions];

Rezultat:

(1167 rows affected)

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 84 ms.

Completion time: 2023-04-16T20:39:26.7127927+02:00

Primer 5. Korišćenje SELECT \* upita

SET STATISTICS TIME ON

SELECT [TradingSessionID], [OpenOnUtc], [ClosedOnUtc]

FROM [dbo].[TradingSessions];

Rezultat:

(1167 rows affected)

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 44 ms.

Completion time: 2023-04-16T20:42:54.4369545+02:

Primer 6. Selekcija kolona od interesa

Kao što možemo videti iz gornja dva izlaza, vreme se smanjuje skoro upola kada koristimo SELECT izraz da biramo samo one kolone koje su potrebne.

## **3.3. Izbegavajte upotrebu SELECT DISTINCT**

Komanda SELECT DISTINCT u SQL-u se koristi za dobijanje jedinstvenih rezultata i uklanjanje dupliranih redova iz relacije. Da bi se izvršio ovaj zadatak, ona praktično grupiše povezane redove i zatim ih uklanja. GROUP BY operacija je skupa operacija. Zato bi za dobijanje jedinstvenih redova i uklanjanje dupliranih redova trebalo koristiti više atributa u SELECT operaciji.

Uzmimo za primer:

SET STATISTICS TIME ON

SELECT DISTINCT [TradingSessionID], [OpenOnUtc], [ClosedOnUtc] FROM [dbo].[TradingSessions];

Rezultat:

(1167 rows affected)

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 66 ms.

Completion time: 2023-04-16T20:58:14.5774166+02:00

Primer 7. Korišćenje SELECT DISTINCT naredbe

SET STATISTICS TIME ON

SELECT [TradingSessionID], [OpenOnUtc], [PreClosedUtc], [ClosedOnUtc], [SettlementDateEst]

FROM [dbo].[TradingSessions];

Rezultat:

(1167 rows affected)

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 41 ms.

Completion time: 2023-04-16T21:00:22.0600148+02:00

Primer 8. Pribavljanje više atributa u SELECT naredbi

Kao što vidimo iz izvršenja prethodna dva upita, DISTINCT operacija zahteva više vremena da izvuče jedinstvene redove. Stoga je bolje dodati više atributa u SELECT upit kako bi se poboljšale performanse i dobili jedinstveni redovi.

## **3.4. INNER JOIN naspram WHERE klauzule**

Treba koristiti INNER JOIN za spajanje dve ili više tabela umesto korištenja WHERE klauzule. WHERE klauzula stvara CROSS JOIN odnosno Dekartov (*„Cartesian“*) proizvod za spajanje tabela. Dekartov proizvod dve tabele zahteva puno vremena.

SET STATISTICS TIME ON

SELECT [Caption], [SymbolCode], [MarketCap]

FROM [Products] p, [ProductStatistics] ps

WHERE p.ProductID = ps.ProductID

Rezultat:

(12974 rows affected)

SQL Server Execution Times:

CPU time = 16 ms, elapsed time = 154 ms.

Completion time: 2023-04-16T23:09:53.1191831+02:00

Primer 9. Korišćenje WHERE klauzule

SET STATISTICS TIME ON

SELECT [Caption], [SymbolCode], [MarketCap]

FROM [Products] p INNER JOIN [ProductStatistics] ps

ON p.ProductID = ps.ProductID

Rezultat:

(12974 rows affected)

SQL Server Execution Times:

CPU time = 16 ms, elapsed time = 101 ms.

Completion time: 2023-04-16T23:19:13.5637885+02:00

Primer 10. Korišćenje INNER JOIN-a

Tako možemo videti iz gore navedenih izlaza da INNER JOIN uzima znatno manje vremena u poređenju sa spojem korištenjem WHERE klauzule.

## **3.5. TOP komanda**

TOP komanda se koristi za kontrolisanje broja redova koji će biti prikazani iz rezultujućeg skupa podataka. Rezultujući skup podataka treba da prikaže samo one redove koji su potrebni. Stoga, treba koristiti ovaj limit sa produkcionim skupom podataka i obezbediti potreban proračun redova za produkcijske svrhe po potrebi.

SET STATISTICS TIME ON

SELECT TOP 10 [Caption], [SymbolCode], [Sequence]

FROM [Products]

Rezultat:

(10 rows affected)

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 0 ms.

Completion time: 2023-04-16T23:45:16.6843027+02:00

Primer 11. Korišćenje TOP komande

Gornji upit ispisuje prvih 10 redova iz rezultujućeg skupa. Ovo drastično poboljšava performanse upita.

## **3.6. IN naspram EXISTS**

Operator IN je skuplji od operatora EXISTS u pogledu pretrage, posebno kada je rezultat podupita veliki skup podataka. Zato bismo trebali koristiti EXISTS umesto IN za preuzimanje rezultata podupita.

Ovo ćemo ilustrovati primerom:

SET STATISTICS TIME ON

SELECT [Caption], [SymbolCode], [Sequence] FROM [Products]

WHERE [ProductID] IN

(SELECT [ProductID] FROM [ProductQuotePrices])

Rezultat:

(22 rows affected)

SQL Server Execution Times:

CPU time = 16 ms, elapsed time = 8 ms.

Completion time: 2023-04-17T00:03:02.3462898+02:00

Primer 12. Korišćenje IN operatora

SET STATISTICS TIME ON

SELECT [Caption], [SymbolCode], [Sequence] FROM [Products]

WHERE EXISTS (SELECT [ProductID] FROM [ProductQuotePrices])

Rezultat:

(24 rows affected)

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 0 ms.

Completion time: 2023-04-17T00:04:08.3233506+02:00

Primer 13. Korišćenje EXISTS operatora

Izvršili smo slični upit sa podupitom koji koristi IN i EXISTS operatore, i primećujemo da EXISTS operator traje dosta manje vremena u poređenju sa IN operatorom, a broj fizičkih i logičkih skeniranja je vrlo nizak.

U ovom primeru je bitno primetiti da dati upiti nisu ekvivalentni. Prvi upit vraća sve redove iz tabele „Products” čija je vrednost kolone „ProductID" prisutna u tabeli „ProductQuotePrices". Drugi upit, sa druge strane, vraća sve redove iz tabele „Products" gde postoji bar jedan red prisutan u tabeli „ProductQuotePrices".

Drugim rečima, prvi upit vraća samo one redove iz „Products" koji imaju odnos sa „ProductQuotePrices", dok drugi upit vraća sve redove iz „Products" ako postoji bar jedan red u „ProductQuotePrices".

## **3.7. Petlje u odnosu na grupno umetanje/ažuriranje**

Ukoliko je moguće, petlje treba izbegavati jer zahtevaju pokretanje istog upita mnogo puta. Umesto toga, trebalo bi da se odlučimo za masovna (*„bulk”*) umetanja.

SET STATISTICS TIME ON

DECLARE @Counter INT

SET @Counter=1

WHILE ( @Counter <= 10)

BEGIN

PRINT 'Counter value is = ' + CONVERT(VARCHAR,@Counter)

INSERT INTO [ReportTypes]

([Uid]

,[Enumeration]

,[Description]

,[Sequence])

VALUES

(NEWID()

,'Type '+ CONVERT(VARCHAR,@Counter)

,'Type '+ CONVERT(VARCHAR,@Counter) + ' Report'

,CONVERT(INT,@Counter))

SET @Counter = @Counter + 1

END

Rezultat:

(10 rows affected)

SQL Server Execution Times:

CPU time = 16 ms, elapsed time = 18 ms.

Completion time: 2023-04-17T01:39:38.2926452+02:00

Primer 14. Upis podataka u tabelu pomoću petlje

SET STATISTICS TIME ON

INSERT INTO [ReportTypes]

([Uid]

,[Enumeration]

,[Description]

,[Sequence])

VALUES

(NEWID()

,'Type 1'

,'Type 1 Report'

,1),

(NEWID()

,'Type 2'

,'Type 2 Report'

,2),

--...

(NEWID()

,'Type 10'

,'Type 10 Report'

,10)

GO

Rezultat:

(10 rows affected)

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 0 ms.

Completion time: 2023-04-17T02:01:41.2594251+02:00

Primer 15. Bulk upis podataka u tabelu

## **3.8. Sažetak tehnika optimizacije**

Ostale generalne stvari koje treba imati na umu, a koje treba uzeti sa rezervom jesu:

* Izbegavajte korišćenje ugnježđenih upita.
* Izbegavajte INNER JOIN-e sa uslovima jednakosti ili OR operatorom.
* Proverite da li zapisi postoje pre nego što probate da ih preuzmete.
* Pravilno koristite indeksiranje, pokušajte da kreirate više indeksa za preuzimanje složenih kolona.
* Mudro koristite džokere („*Wildcards*“).
* Pokušajte da koristite WHERE umesto HAVING. HAVING koristite samo za objedinjene (agregirane) vrednosti.

# 4. Uskladištene procedure

Uskladištene procedure su sastavni deo programiranja baza podataka u SQL Serveru, i nude nekoliko pogodnosti koje pomažu u optimizaciji SQL upita. Uskladištena procedura je grupa SQL instrukcija uskladištenih u bazi podataka i izvršavaju se kao jedinstvena celina. Primarna prednost korišćenja uskladištenih procedura je sposobnost efikasnog izvršavanja složenih upita.

## **4.1. Uloga uskladištenih procedura u optimizaciji upita**

Uskladištene procedure mogu optimizovati SQL upite na nekoliko načina. Jedna od glavnih prednosti korišćenja uskladištenih procedura je smanjenje mrežnog saobraćaja. Kada se izvrši uskladištena procedura, prenose se samo naziv procedure i parametri, umesto celog upita. Ovo značajno smanjuje mrežni saobraćaj i može poboljšati performanse upita.

Drugi način na koji uskladištene procedure optimizuju upite je kroz sposobnost prekompilacije (kompajliranje unapred) upita. Kada se uskladištena procedura kreira, serverska baza podataka kompajlira SQL instrukcije u okviru procedure i skladišti plan izvršenja. Ovo znači da kada se uskladištena procedura izvrši, serverska baza podataka može brzo da preuzme plan izvršenja, što smanjuje vreme izvršavanja upita.

Uskladištene procedure takođe omogućavaju upotrebu parametrizovanih upita, što može dodatno poboljšati performanse upita. Parametrizovani upiti su SQL izrazi koji uključuju rezervisana mesta za vrednosti parametara. Kada se izvrši uskladištena procedura, vrednosti parametara se prosleđuju na rezervisana mesta i upit se izvršava. Ovo omogućava serveru baze podataka da ponovo upotrebi plan izvršavanja za slične upite, što može poboljšati performanse upita.

Osim toga, uskladištene procedure takođe omogućavaju upotrebu transakcija, koje mogu pomoći u održavanju integriteta podataka. Transakcije osiguravaju da grupa SQL naredbi bude izvršena kao jedinstvena radna jedinica. Ako bilo koja naredba u transakciji ne uspe, cela transakcija se poništava i baza podataka se vraća u prethodno stanje. Ovo pomaže u održavanju integriteta podataka i sprečava neusaglašenosti podataka.

U sažetku, uskladištene procedure predstavljaju jedan od ključnih alata za optimizaciju SQL upita u SQL Serveru. Pružaju nekoliko prednosti, uključujući smanjenje mrežnog saobraćaja, prekompilaciju upita, omogućavanje parametrizovanih upita i podršku za transakcije. Iskorišćavanjem ovih prednosti, programeri mogu kreirati efikasnije i pouzdanije aplikacije za baze podataka.

# 5. Zaključak

Zaključno, optimizacija upita u SQL Serveru je ključan aspekt upravljanja bazama podataka koji može značajno uticati na ukupne performanse baze podataka. Kao što smo diskutovali, SQL Server engine koristi niz koraka za procesiranje i optimizaciju upita, pri čemu svaki korak ima za cilj identifikaciju i eliminaciju neefikasnosti u upitu.

Proces optimizacije upita počinje sa parsiranjem i povezivanjem SQL naredbe, nakon čega se kreira plan izvršenja koji opisuje korake potrebne za dobijanje podataka traženih upitom. Ovaj plan se zatim optimizuje na osnovu raznih faktora, uključujući veličinu skupa podataka, dostupne indekse i složenost upita.

Tokom ovog procesa, ključno je razumeti razne alate i tehnike dostupne za optimizaciju upita u SQL Serveru. To uključuje upotrebu indeksa, statistike, saveta za upit i druge tehnike za optimizaciju performansi. Iskorišćavanjem ovih alata i tehnika na efikasan način, administratori baza podataka mogu poboljšati performanse upita, smanjiti upotrebu resursa i poboljšati ukupnu efikasnost baze podataka.

Međutim, važno je napomenuti da je optimizacija upita neprekidan proces koji zahteva stalno praćenje i prilagođavanje. Kako se količina podataka povećava i upiti postaju složeniji, administratori baza podataka moraju kontinuirano procenjivati performanse svojih upita i vršiti prilagođavanja po potrebi.

Osim toga, važno je imati na umu da je optimizacija upita samo jedan aspekt upravljanja bazom podataka. Ostali važni faktori uključuju sigurnost, sigurnosnu kopiju i oporavak podataka te integritet podataka. Efikasno upravljanje bazom podataka zahteva holistički pristup koji uzima u obzir sve ove faktore.

Zaključno, optimizacija SQL Server upita ključna je komponenta upravljanja bazom podataka koja zahteva pažnju, opreznost i stalno praćenje. Korištenjem različitih alata i tehnika za optimizaciju upita, administratori baza podataka mogu poboljšati performanse i efikasnost svojih baza podataka, što dovodi do boljih ukupnih performansi sistema i korisničkog iskustva u celini.

# 6. Literatura

1. *TutorialsPoint. "MS SQL Server - Overview". [*[*Online*](https://www.tutorialspoint.com/ms_sql_server/ms_sql_server_quick_guide.htm)*], [Accessed: 12-Apr-2023]*
2. *Microsoft Learn. "Editions and supported features of SQL Server 2022". [*[*Online*](https://learn.microsoft.com/en-us/sql/sql-server/editions-and-components-of-sql-server-2022?view=sql-server-ver16)*], [Accessed: 12-Apr-2023]*
3. *Microsoft Learn. "Introduction to SQL Server 2022". [*[*Online*](https://learn.microsoft.com/en-us/training/modules/introduction-to-sql-server-2022/)*], [Accessed: 12-Apr-2023]*
4. *Sachdeva, Siddharth (2021). "A Detailed Guide on SQL Query Optimization". Analytics Vidhya, [*[*Online*](https://www.analyticsvidhya.com/blog/2021/10/a-detailed-guide-on-sql-query-optimization/)*], [Accessed: 15-Apr-2023]*
5. *Microsoft Learn (2023). "Creating and Modifying Stored Procedures in SQL Server". [*[*Online*](https://learn.microsoft.com/en-us/sql/relational-databases/stored-procedures/create-a-stored-procedure?view=sql-server-ver16)*], [Accessed: 16-Apr-2023]*
6. *Singh, Harpreet (2022). "Stored Procedures In SQL Server". [*[*Online*](https://www.c-sharpcorner.com/UploadFile/219d4d/stored-procedures-in-sql-server/)*], [Accessed: 16-Apr-2023]*

# 7. Listing

Slika 1. Procenjeni plan izvršenja 8

Slika 2. Analiza skeniranja indeksa 9

Slika 3. Procenjeni plan izvršenja primera za indeksiranje 10

Primer 1. Korišćenje SET STATISTICS TIME ON komande 6

Primer 2. Korišćenje SET STATISTICS IO ON komande 7

Primer 3. Korišćeni upit za analizu plana izvršenja 8

Primer 4. Korišćeni upit za analizu indeksiranja 10

Primer 5. Korišćenje SELECT \* upita 11

Primer 6. Selekcija kolona od interesa 11

Primer 7. Korišćenje SELECT DISTINCT naredbe 12

Primer 8. Pribavljanje više atributa u SELECT naredbi 12

Primer 9. Korišćenje WHERE klauzule 13

Primer 10. Korišćenje INNER JOIN-a 13

Primer 11. Korišćenje TOP komande 14

Primer 12. Korišćenje IN operatora 14

Primer 13. Korišćenje EXISTS operatora 15

Primer 14. Upis podataka u tabelu pomoću petlje 16

Primer 15. Bulk upis podataka u tabelu 16