OPTIMIZACIJA SQL UPITA
SQL QUERY TUNING

Dejan Stjepanović, *Elektrotehnički fakultet Banja Luka*

Sadržaj – *U praksi se pokazalo da se problemi sa performansama informacionih sistema vrlo često ne mogu riješiti intervencijom na hardverskom nivou, kao što je dodavanje memorije, dodavanje procesora i sl. Optimizacija rada sa bazom podataka često zahtjeva niz različitih akcija, a pojedini postupci, kao npr. optimizacija upita u bazu podataka, često daju željene rezultate. U ovom radu su opisani neki postupci optimizacije upita i njihov uticaj na vrijeme izvršenja upita. Izložen je primjer optimizacije SQL upita i uticaj optimizacije na performanse konkretnog informacionog sistema „Pisarnica“ (protokol) Administrativne službe Grada Banja Luka, gdje se evidentiraju svi predmeti (podnesci) koji ulaze u Administrativnu službu Grada.*

Abstract - *The practice proved that the problems with the performance of information systems often cannot be solved by tuning a hardware level, such as adding memory, adding a processor, etc. Optimization of databases, especially query optimization in the database often gives the desired results. This paper presents some techniques of the query optimization, and its influence on query execution time. The results of SQL Query optimization on information system performance is shown for the real environment, i.e. for the information system "Pisarnica" (Protocol) of City Administration of Banja Luka, where all the cases (submissions) entered the City Administration of Banja Luka are recorded.*

.

1. UVOD

Postoji više segmenata nekog informacionog sistema u kojima se može vršiti optimizacija, pri čemu efekti optimizacije pojedinih dijelova mogu na različite načine da utiču na rad čitavog sistema.

Prilikom optimizacije, posebnu pažnju treba obratiti na optimizaciju upita za dobijanje podataka iz baze koji se pozivaju iz izvornog koda same aplikacije, jer se u praksi pokazalo da se optimizacijom upita često mogu postići daleko veći pozitivni efekti na performanse aplikacije nego optimizacijom hardvera, podešavanjem parametara operativnog sistema ili konfiguracije SQL servera [1]. Neoptimizovani upit u bazu podataka može da zauzme sve raspoložive resurse i na taj način da smanji uticaj uloženih sredstava na performanse čitave aplikacije.

2. UZROCI LOŠIH PERFORMANSI UPITA

U nastavku su navedene situacije koje loše utiču na performanse SQL upita, a time i čitave aplikacije.

Loše indeksiranje (eng. *Poor indexing*) može drastično da utiče na vrijeme izvršenja upita, jer se prilikom izvršenja upita pristupa većoj količini podataka što direktno dovodi do povećanja pristupa disku, korištenja memorije i procesora, a što direktno utiče na povećanje vremena izvršenja upita.

Netačna statistika (eng. *Inaccurate Statistics*) može značajno da naruši performanse upita. Naime, optimizator upita u okviru sistema za upravljanje bazama podataka, na osnovu netačne statistike o distribuciji podataka, može da donosi pogrešnu procjenu o npr. broju zapisa koje upit treba da vrati, a koji je važan parametar optimizatoru za definisanje optimalnog i efikasnog plana izvršenja upita.

Previše međusobnog blokiranja transakcija i zastoja (eng. *Excessive blocking and deadlocks*) dovode do značajnog narušavanja performansi upita. Ukoliko dvije transakcije pristupaju istom podatku u tabeli sa konfliktom (npr. jedna da čita, a druga transakcija ažurira podatak) dolazi do blokiranja jedne od transakcija i na taj način do čekanja ili vraćanja blokirane transakcije na ponovno izvršenje (eng. *rollback*). Blokiranje je normalna pojava pri izvršavanju upita, ali previše blokiranja može da značajno naruši performanse [2]. Zastoj se dešava kada dvije transakcije istovremeno blokiraju jedna drugu (jedna transakcija čeka završetak druge, a druga čeka završetak prve). Sistemi za upravljanje bazama podataka periodično tragaju za zastojima i jednu od transakcija proglašavaju žrtvom i vraćaju je na početak izvršavanja.

Operacije koje nisu zasnovane na radu sa skupovima podataka (eng. *Non-Set-Based Operations*), nego se zasnivaju na obradi podataka upotrebom kursora i petlji (umjesto *join* i *subquery* operacija), značajno mogu da naruše performanse upita, a samim tim i kompletne aplikacije [1].

Loša specifikacija upita (eng. *Poor Query Design*) je najčešće razlog loših performansi upita, jer se željeni rezultat može dobiti definisanjem upita na više različitih načina, a neki od tih načina onemogućuju optimizator da upotrijebi adekvatan indeks i smanji vrijeme izvršavanja upita.

Loša organizacija baze podataka (eng. *Poor Database Design*), odnosno nedovoljna ili prekomjerna normalizacija

baze podataka, može negativno da utiče na vrijeme izvršenja upita zbog nepotrebnog povećanja obima podataka koje upit obrađuje i povećanja broja blokiranja upita ili povećanja broja *join* operacija. Naime, tabela koja nije normalizovana sadrži redundanciju podataka, pa se zbog toga povećava broj pristupa disku i broj međusobnog blokiranja transakcija, a prekomjerna normalizacija baze podataka po pravilu rezultuje povećanjem broja *join* operacija u upitima (*join* operacija u upitu negativno utiče na performanse upita).

Plan izvršenja upita koji se ne kešira (eng. *Nonreusable Execution Plan*) narušava performanse upita, jer optimizator upita sistema za upravljanje bazama podataka troši određeno vrijeme za kreiranje plana izvršenja upita i produžava ukupno vrijeme izvršenja upita.

Česta rekompilacija plana izvršenja (eng. *Frequent Recompilation Execution Plan*) povećava vrijeme izvršenja upita, jer se troši određeno vrijeme za rekompilaciju plana izvršenja. Za izbjegavanje rekompilacije, plan izvršenja upita mora biti nezavisan od parametara koji se prosljeđuju upitu prilikom izvršenja, a to se najčešće postiže upotrebom uskladištenih procedura (eng. *stored procedures*).

Nepravilna upotreba kursora (eng. *Improper Use of Cursors*) povećava vrijeme izvršenja upita. Upotrebom kursora u okviru upita podaci iz tabele se dobavljaju red po red, a sistemi za upravljanje bazama podataka su optimizovani za čitanje i obradu čitavih skupova redova. Ukoliko se kursori ipak moraju koristiti, onda treba voditi računa o tome da postoje različite vrste kursora koje različito utiču na povećanje vremena izvršenja upita.

3. PREPORUKE ZA OPTIMIZACIJU UPITA

Upit se često može napisati na dva ili više načina, a da se pri tome dobija isti rezultat nakon njegovog izvršenja. Upiti napisani na različite načine obično daju različite planove izvršenja upita, koje na osnovu napisanog upita kreira optimizator upita u okviru sistema za upravljanje bazama podataka. Različiti planovi izvršenja upita daju različita vremena izvršenja upita, pa stoga postoje planovi koji su bolji od drugih planova. Prilikom pisanja upita, ukoliko je moguće, uvijek treba definisati upit na način da se dobije dobar (ili optimalan) plan izvršenja, a time i manje vrijeme izvršenja upita [2].

Sa ciljem postizanja najboljih performansi upita poželjno je pridržavati se sljedećih preporuka [1]:

- rad sa malim skupovima podataka
- efektivna upotreba indeksa
- izbjegavati davanje uputa optimizatoru
- koristiti domenski i referencijalni integritet
- izbjegavati upite koji intenzivno koriste resurse
- smanjiti komunikaciju preko mreže
- smanjiti trajanje transakcija

Međutim, navedene preporuke različito utiču na poboljšanje performansi upita u različitim realnim okruženjima, pa je najbolje izvršiti testiranja različitih formi istog upita u realnom okruženju i na osnovu dobijenih rezultata izvršiti optimizaciju upita.

Rad sa malim skupovima podataka. Smanjenjem i ograničenjem skupa podataka (kolona i redova) nad kojima se izvršava upit smanjuje se upotreba resursa i povećava efikasnost indeksa, što pozitivno utiče na performanse upita. Jedan od načina za smanjenje skupa podataka u upitu je da se lista kolona u okviru *select* liste upita ograniči samo na kolone koje su neophodne, odnosno da se izbjegava upit tipa *select* *. Drugi način za smanjenje skupa podataka u upitu je da se upotrebom *where* klauzule smanjuje broj redova ukoliko je to moguće, jer se na taj način optimizacija upita realizuje efikasnijom upotrebom indeksa. Ukoliko se kao rezultat upita korisniku treba prezentovati velika količina podataka, onda se može, ukoliko je prihvatljivo, izvršiti straničenje podataka (eng. *paging*), odnosno na zahtjev korisnika slati jedan po jedan skup zapisa.

Efektivna upotreba indeksa. Poznato je da indeksi (*clustered index* – podaci su smješteni na disku redoslijedom kako su sortirani i *non-clustered indexes* – sortirani su samo indeksi na podatke koji se nalaze razbacani na disku) definisani nad odgovarajućim kolonama (najčešće korištenim u upitima) ubrzavaju izvršenje upita [4]. Međutim, podjednako je važno da ovi indeksi budu efikasno iskorišteni prilikom definisanja upita. Efikasna upotreba indeksa nad kolonom koja se nalazi u okviru *where* klauzule upita zavisi od izraza, odnosno operacija koje se vrše nad tom kolonom u okviru *where* klauzule (brza *Index Seek* operacija izvršava se ukoliko se radi o visoko selektivnom upitu, odnosno ukoliko se za dobijanje rezultata upita mora obraditi mali broj redova, 10-15% redova od ukupnog broja redova u tabeli [5]). Na primjer, operator „!=“ zahtijeva skeniranje čitave tabele iako je nad kolonom definisan indeks, što predstavlja neefikasnu upotrebu indeksa (sporija *Index Scan* operacija). Nepoželjni operatori, koje treba izbjegavati nad kolonama sa indeksom u okviru *where* klauzule, su operatori isključenja (<>, !=, !>, !<, NOT EXISTS, NOT IN i NOT LIKE IN, OR), a nekada i operator LIKE (LIKE '%tekst') [1]. Naravno, ne mogu se uvijek izbjeći ovi operatori, ali kada je moguće predefinisati upit tako da vraća isti rezultat korištenjem drugih operatora, onda to treba razmotriti u cilju poboljšanja performansi upita.

Neoptimizovani LIKE upit:

SELECT zaposleni.ime FROM zaposleni WHERE zaposleni.ime LIKE 'Dej%'

Optimizovan LIKE upit:

SELECT zaposleni.ime FROM zaposleni WHERE zaposleni.ime >= 'Dej' AND zaposleni.ime < 'DeK'

Vrlo često optimizator sistema za upravljanje bazama podataka, ukoliko je to moguće, izvrši konverziju upita u optimizovani oblik tako što konvertuje nepoželjni izraz u odgovarajući poželjni izraz (npr. !< zamjeni sa >=). Da bi se postigla efikasnija upotreba indeksa, potrebno je **izbjegavati aritmetičke operacije** nad kolonama u okviru *where* klauzule:

Neoptimizovani upit:

*SELECT zaposleni.ime, zaposleni.pozicija FROM zaposleni WHERE zaposleni.godine * 2 > 60*

Optimizovan upit:

SELECT zaposleni.ime, zaposleni.pozicija FROM zaposleni WHERE zaposleni.godine > 60/2

Izbjegavati funkcije nad kolonama u okviru *where* klauzule, npr:

Neoptimizovani upit:

SELECT zaposleni.ime FROM zaposleni WHERE SUBSTRING(zaposleni.ime,1,1) = 'D'

Optimizovan upit:

SELECT zaposleni.ime FROM zaposleni WHERE zaposleni.ime>='F' AND zaposleni.ime<'G'

Izbjegavati davanje uputa optimizatoru. Davanjem uputa optimizatoru se onemogućava da optimizator dinamički definiše plan izvršenja upita u vrijeme njegovog planiranja i realizacije. Na ovaj način se obično narušavaju performanse upita, pa treba izbjegavati *JOIN* upute (eng. *join hints*), *INDEX* upute (eng. *index hints*) i *FORCEPLAN* upute (eng. *forceplan hints*) prilikom definisanja upita.

Koristiti domenski i referencijalni integritet. Domenski integritet predstavlja ograničenje skupa vrijednosti koje neki atribut skupa entiteta (neka kolona u tabeli) može da sadrži, a referencijalni integritet je ograničenje da se neka vrijednost koja se pojavljuje u jednoj relaciji mora nalaziti i u drugoj relaciji [6]. Domenski i referencijalni integritet omogućavaju optimizatoru da provjeri validnost podataka bez fizičkog pristupa podacima. Smanjivanje broja fizičkih pristupa podacima, pozitivno utiče na performanse upita, smanjujući vrijeme izvršenja upita. Primjer uticaja domenskog integriteta na performanse upita je *NOT NULL* ograničenje (definisanje da kolona mora sadržavati neku vrijednost). Upit definisan na sljedeći način

SELECT zaposleni.ime, zaposleni.godine FROM zaposleni WHERE zaposleni.godine != 30

je optimizovan, ali ukoliko kolona *zaposleni.godine* može da sadrži *NULL* vrijednost, ovaj upit ne vraća sve zapise iz tabele koji u potpunosti zadovoljavaju uslov. Ukoliko je potrebno da upit vrati i zapise sa vrijednošću *null* u koloni *zaposleni.godine*, neophodno je proširiti uslov na sledeći način:

SELECT zaposleni.godine FROM zaposleni WHERE zaposleni.godine != 30 OR zaposleni.godine IS NULL

Međutim, proširenjem upita mijenja se plan izvršenja upita tako da dolazi do povećanja vremena izvršenja upita. Preporučljivo je, kada je to moguće, onemogućiti da kolona može da sadrži *NULL* vrijednost. Definisanjem referencijalnog integriteta između dvije tabele poboljšavaju se performanse *JOIN* upita između te dvije tabele.

SELECT zaposleni.zaposleniid, zaposleni.ime, kontakti.adresa FROM zaposleni JOIN kontakti ON zaposleni.kontaktid = kontakti.kontaktid WHERE zaposleni.zaposleniid = 2345

Naime, ukoliko postoji referencijalni integritet definisan između kolona *kontaktid* u tabelama i ukoliko kolona *kontaktid* u tabeli *zaposleni* ne može sadržati *NULL* vrijednost, optimizator će preskočiti skeniranje tabele *kontakti*, jer na osnovu referencijalnog i domenskog integriteta pouzdano zna da za svaki zapis u tabeli *zaposleni* postoji jedan zapis u tabeli *kontakti*. Na ovaj način se smanjuje broj fizičkih pristupa podacima i poboljšavaju performanse upita.

Izbjegavati upite koji intenzivno koriste resurse servera. Da bi upiti manje koristili resurse servera moraju se definisati tako da se u okviru njih:

- izbjegava konverzija iz jednog tipa podataka u drugi tip prilikom poređenja kompatibilnih kolona tabele ili vrijednosti (npr. *varchar* sa *nvarchar*);
- za provjeru postojanja nekih podataka koristi *EXISTS* umjesto *COUNT(*)*;
- kad god je to moguće umjesto *UNION* koristi *UNION ALL* pri čemu treba imati u vidu da se tada, ukoliko postoje, ne eliminišu duplikati u okviru rezultata upita;
- koriste indeksi nad kolonama nad kojima se koriste agregatne funkcija ili sortiranje;
- da se izbjegava *sp* prefiks kod definisanja imena ugnježdenih procedura jer ukoliko ime procedure počinje sa *sp* onda server prvo traži proceduru u sistemskim procedurama i na taj način gubi dragocjeno vrijeme.

Smanjiti mrežnu komunikaciju. Da bi se smanjila mrežna komunikacija, koja značajno utiče na vrijeme izvršavanja upita posmatrano sa strane aplikacije i korisnika, neophodno je smanjiti broj mrežnih ciklusa (eng. *network round-trips*) i trajanje mrežnog ciklusa. Trajanje mrežnog ciklusa je vrijeme potrebno da paket stigne od izvora (računara koji inicira komunikaciju) do destinacije i opet nazad do izvora komunikacije [7]. Smanjivanje broja mrežnih ciklusa se može postići izvršavanjem više upita zajedno kao *batch* upit (jedan ili više SQL iskaza koji se izvršavaju kao jedan iskaz [7]) ili kao uskladištena procedura (eng. *stored procedure*). Smanjivanje vremena mrežnog ciklusa se može postići definisanjem da se kao rezultat upita ne vraća broj zapisa koji su obrađeni u okviru upita (*batch* upit ili *stored* procedura). Broj obrađenih zapisa (redova) vrlo često je nebitan podatak i može se naredbom *SET NOCOUNT ON* izbjeći njegovo vraćanje nakon izvršenja upita (*SET NOCOUNT OFF* je suprotna operacija u okviru SQL servera).

Smanjiti trajanje transakcija. Prilikom izvršavanja jednog upita, koji se posmatra kao *atomic* akcija (akcija koja ne može biti prekinuta drugom akcijom i pri kojoj baza podataka prelazi iz jednog konzistentnog stanja u drugo), vrši se upis dva konzistentna stanja u transakcijski log baze podataka (eng. *transaction log*) na disku. Ovo je vremenski zahtjevna operacija, koja značajno utiče na performanse upita. Broj ovih operacija se može smanjiti grupisanjem više upita u jednu transakciju (više upita se izvršava kao jedna *atomic* akcija).

Neoptimizovan upit:

BEGIN

INSERT INTO zaposleniVALUE (Milan,Ivić,programer)
INSERT INTO zaposleniVALUE (Petar,Jović,programer)
INSERT INTO zaposleniVALUE (Luka,Lukić,programer)
END

Optimizovani upit:

BEGIN TRANSACTION
BEGIN
INSERT INTO zaposleni VALUE (Milan,Ivić,programer)
INSERT INTO zaposleni VALUE (Petar,Jović,programer)
INSERT INTO zaposleniVALUE (Luka,Lukić,programer)
END
COMMIT

Međutim, na ovaj način se povećava vrijeme kada su pojedini zapisi u tabeli zaključani što može negativno uticati na performanse drugih upita. Naime, SQL iskazi (*SELECT*, *INSERT*, *UPDATE*, *DELETE*) vrše zaključavanje zapisa u tabelama kako bi se podaci koje obrađuju zaštitili od drugih SQL iskaza. Zaključavanje zapisa povećava čekanje drugih upita i na taj način se produžava vrijeme njihovog izvršavanja. Početna postavka servera baza podataka je da se prilikom izvršavanja jednog upita zaključavanje vrši na nivou zapisa, tako da se ne dešava često da neki drugi upit zahtjeva pristup zaključanom zapisu i da mora da čeka na otključavanje tog zapisa. Međutim, ukoliko jedan upit u toku izvršavanja obrađuje veliku količinu zapisa, a zaključavanje se vrši na nivou zapisa, onda će se naizmjenično izvršavati veliki broj operacija zaključavanja i otključavanja zapisa, čime se usložnjava upravljanje ovim operacijama i povećava vrijeme neophodno za njihovo izvršavanja, a samim tim i vrijeme izvršavanja upita. Ovo vrijeme se može smanjiti povećanjem nivoa na kojem se vrši zaključavanje tako da se na primjer zaključavanje vrši na nivou stranice (eng. *pagelock*) ili čak na nivou cijele tabele (eng. *tablock*):

SELECT zaposleni.ime, zaposleni.prezime FROM zaposleni WITH(PAGLOCK ili *TABLOCK)*

Ukoliko se iz neke tabele podaci najčešće samo čitaju, a vrlo rijetko upisuju, mijenjaju ili brišu, onda je vrlo korisno upite (*SELECT*) izvršavati bez zaključavanja zapisa:

SELECT zaposleni. FROM zaposleni WITH(NOLOCK)*

4. OPTIMIZACIJA UPITA BAZE „Pisarnica“

Performanse izvršenja upita, kao i performanse baze podataka uopšte, u direktnoj su vezi sa brojem istovremenih korisnika aplikacije koja radi sa bazom podataka, kao i sa količinom podataka koja je pohranjena. Da bi prikaz uticaja raznih načina optimizacije upita na performanse upita bio što očigledniji, za testiranje je izabrana baza podataka „Pisarnica“ Administrativne službe Grada Banja Luka, koja sadrži veliku količinu podataka i koju konstantno koristi veliki broj korisnika zaposlenih u Administrativnoj službi. U okviru ove baze podataka se vodi evidencija o svim dokumentima (podnescima) koji svakodnevno ulaze u Administrativnu službu Grada.

Testno okruženje se sastoji od servera (Intel Xeon, 8GB DDR2, 4x72 GB HDD) za upravljanje bazama podataka (MS SQL Server 2005), u okviru kojeg se nalazi baza podataka „Pisarnica“, i alata za definisanje, izvršavanje, praćenje i optimizaciju upita (MS SQL Server Management Studio). Osnovna tabela za testiranje je „PIS_PISARNICA“ (23 kolone i 500 000 zapisa ili redova). Prilikom upotrebe aplikacije koja koristi bazu podataka „Pisarnica“ uočeno je da se funkcija za pretragu predmeta mjesnih zajednica po godinama u kojim je predmet protokolisan izvršava veoma sporo i da je tu funkciju potrebno optimizovati.

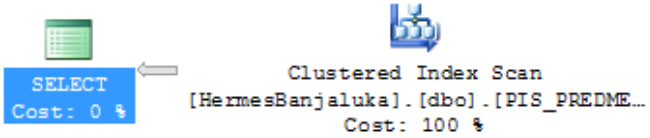
Optimizacija performansi je iterativan proces, gdje je najprije potrebno identifikovati usko grlo kod izvršenja upita, zatim pokušati eliminisati to usko grlo, izmjeriti poboljšanje nakon optimizacije i vratiti se na prvi korak i ponavljati sve dok se ne dobiju zadovoljavajuće performanse, odnosno dok se ne postigne cilj optimizacije. Pri ovome, treba voditi računa da optimizacija jednog dijela aplikacije može prouzrokovati pad performansi u drugom dijelu aplikacije. Usko grlo funkcije koju je potrebno optimizovati je SQL upit, jer najveći dio vremena (oko 95%) izvršenja ove funkcije otpada na vrijeme izvršenja SQL upita, koji funkcija poziva kroz izvršni kod aplikacije.

Kako bi testiranje i optimizacija performansi bila efikasna, neophodno je prvo izvršiti definisanje cilja optimizacije, odnosno koliko je minimalno očekivano poboljšanje performansi (npr. za 50% smanjiti vrijeme izvršenja upita), i onda optimizaciju vršiti u skladu sa tim ciljem. Optimizacja će se vršiti tako što se prilikom izvršavanja upita prati plan izvršenja upita i vrijeme izvršenja upita. Prilikom testiranja je dobro vršiti jednu po jednu izmjenu u upitu prilikom svakog iterativnog koraka, odnosno prilikom svakog novog mjerenja performansi. Izvršenje upita i mjerenje ključnih parametara, prije i posle izmjene upita u cilju optimizacije, je neophodno vršiti više (5 do 10) puta i uzeti prosječnu vrijednost, jer vrijeme izvršenja upita obično varira prilikom svakog novog izvršenja. SQL upit, koji je predstavljao usko grlo aplikativne funkcije koju je bilo neophodno optimizovati, je prikazan u niže navedenom tekstu:

*SELECT * FROM* PIS_PREDMETI
*WHERE SUBSTRING(*PODNOSILAC*, 1, 3) =* 'MZ' *AND YEAR(*DATUM*) IN* (2004,2005,2006,2007,2008,2009,2010).

Izvršavanjem i mjerenjem parametara neoptimizovanog SQL upita su dobijeni sledeći rezultati i prikazan je plan izvršenja upita:

Table 'PIS_PREDMETI'. Scan count 1, logical reads 50495, CPU time = 1500 ms, elapsed time = 1970 ms.



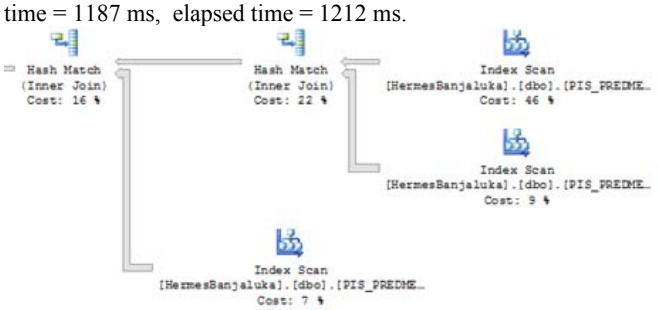
Slika 1.: Plan izvršenja neoptimizovanog upita

Na osnovu plana izvršenja upita može se uočiti da se prilikom izvršenja upita vrši kompletno skeniranje tabele iako su definisani indeksi za kolone u *where* klauzuli, što znači da se indeksi ne koriste na adekvatan način. U definiciji SQL upita je odmah uočeno nekoliko elemenata koje bi u skladu sa preporukama navedenim u dijelu 2 trebalo izbjegavati

kako bi se dobile bolje performanse upita. Prvi korak u postupku optimizacije ovog upita je da se upit predefiniše tako da se smanji skup podataka koje upit vraća navođenjem samo kolona koje su zaista neophodne:

SELECT ID, NAZIV, DATUM, PODNOSILAC
FROM PIS_PREDMETI
*WHERE SUBSTRING(*PODNOSILAC*, 1, 3) =* 'MZ ' *AND YEAR(*DATUM*) IN* (2004,2005,2006,2007,2008,2009,2010)

Prvim korakom optimizacije dobijeni su sledeći rezultati:
Table 'PIS_PREDMETI'. Scan count 3, logical reads 24828, CPU time = 1187 ms, elapsed time = 1212 ms.

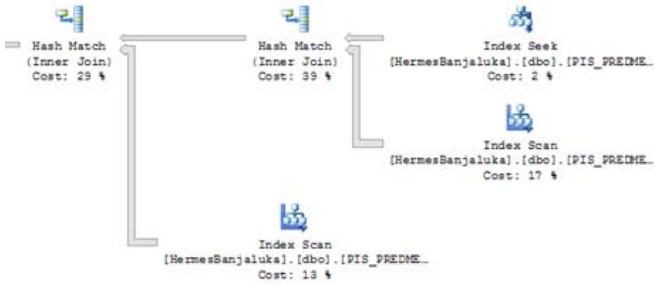


Slika 2.: Plan izvršenja optimizovanog upita I

Nakon prvog koraka optimizacije se može uočiti da je broj operacija logičkog čitanja znatno smanjen, kao i vrijeme izvršenja upita. Drugi korak pri optimizaciji ovog upita je da se upit predefiniše tako što se zamjeni funkcija *substring* nad kolonom tabele u *where* kaluzuli odgovarajućim iskazom:

SELECT ID, NAZIV, DATUM, PODNOSILAC
FROM PIS_PREDMETI *WHERE*
PODNOSILAC >= 'MZ ' *AND* PODNOSILAC < 'MZ!' *AND YEAR(*DATUM*) IN* (2004,2005,2006,2007,2008,2009,2010)

Drugim korakom optimizacije dobijeni su sledeći rezultati:
Table 'PIS_PREDMETI'. Scan count 3, logical reads 5791, CPU time = 969 ms, elapsed time = 1022 ms.

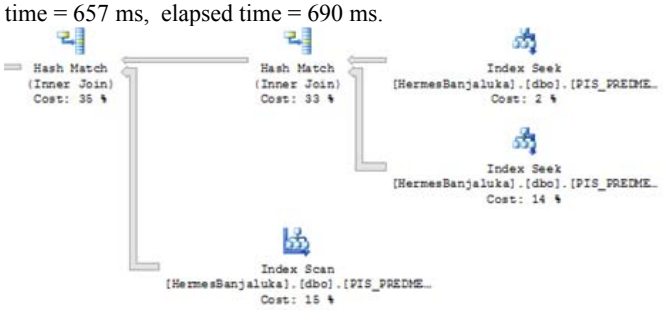


Slika 3.: Plan izvršenja optimizovanog upita II

Nakon drugog koraka optimizacije, optimizator upita je kreirao plan izvršenja upita koji efikasnije iskorištava indeks za kolonu „PODNOSILAC“, jer se sada umjesto skeniranja čitave tabele, izvršava pretraga tabele indeksa (eng. *Index Seek*) za kolonu „PODNOSILAC“, što je znatno brža operacija. Treći korak pri optimizaciji ovog upita je da se upit predefiniše tako što se zamjeni funkcija *year* nad kolonom DATUM u *where* kaluzuli odgovarajućim iskazom:

SELECT ID, NAZIV, DATUM, PODNOSILAC
FROM PIS_PREDMETI *WHERE*
PODNOSILAC >= 'MZ ' *AND* PODNOSILAC < 'MZ!' *AND* DATUM >= '1/1/2004' *AND* DATUM < '1/1/2011'

Trećim korakom optimizacije dobijeni su sledeći rezultati:
Table 'PIS_PREDMETI'. Scan count 3, logical reads 5043, CPU time = 657 ms, elapsed time = 690 ms.



Slika 4.: Plan izvršenja optimizovanog upita III

Nakon trećeg koraka optimizacije, optimizator upita je kreirao plan izvršenja upita koji efikasnije koristi i indeks za kolonu „DATUM“ (pretraga tabele indeksa za kolonu „DATUM“), što daje kraće vrijeme izvršenja upita. Vrijeme izvršenja upita nakon trećeg koraka optimizacije je manje za 50% ili više od vremena izvršenja neoptimizovanog upita i na taj način je ispunjen unaprijed postavljeni cilj optimizacije.

5. ZAKLJUČAK

Sistemi za upravljanje bazama podataka sadrže optimizatore upita, koji prije izvršenja upita definišu plan izvršenja ca ciljem da se obezbijede optimalne performanse izvršavanja upita. Međutim, vrlo često optimizator upita ne može unaprijed da specifikuje optimalan način izvršenja upita. U ovom radu je pokazano da se definisanjem upita na adekvatan način može pomoći optimizatoru da kreira bolji plan izvršenja upita, čime se mogu značajno poboljšati performanse upita. Važno je napomenuti da je mjerenjem vremena izvršenja upita uočeno da je vrijeme izvršenja upita prvi put značajno duže, nego prilikom svakog sljedećeg uzastopnog izvršenja upita sa istim parametrima. Ova pojava je posledica učenja optimizatora upita zasnovanog na statistici koja se prati prilikom svakog izvršenja upita i privremenog skladištenja podataka u brzu (*RAM*) memoriju. Ovo je važna karakteristika izvršenja upita, jer je rijetka situacija kada se upit sa istim parametrima izvršava više puta uzastopno. Prilikom optimizacije upita treba voditi računa da se i vrijeme prvobitnog izvršenja upita smanji u skladu sa postavljenim ciljom optimizacije.

LITERATURA

[1] G. Fritchey and S. Dam, *SQL Server 2008 Query Performance Tuning Distilled*, New York: Apress 2009.
[2] K. Delaney, Inside Microsoft SQL Server 2005: Query Tuning and Optimization, Microsoft Press 2007.
[3] Sitansu S. Mittra, Database Performance Tuning and Optimization, New York: Springer-Verlag 2003.
[4] http://mysoftskill.blogspot.com/2009/11/sql-server-index-tuning-clustered-vs.html
[5] http://blog.sqlauthority.com/2007/03/30/sql-server-index-seek-vs-index-scan-table-scan/
[6] A. Silberschatz, Henry F. Korth, S. Sudarshan, Database System Concepts, McGrow-Hill 1999.
[7] http://searchnetworking.techtarget.com