

|  |  |
| --- | --- |
|  | UNIVERZITET U NIŠU ELEKTRONSKI FAKULTET |

**Optimizacija upita Neo4j**

Seminarski rad

Studijski program: Računarstvo i informatika

Modul: Softversko inženjerstvo

|  |  |
| --- | --- |
| Student: | Profesor: |
| Nikola Mitić, br. ind. 1433 | Doc. dr Aleksandar Stanimirović |
|  |  |

Niš, april 2022. godina

Sadržaj:

[1. Uvod 3](#_Toc101983409)

[2. Optimizacija upita 3](#_Toc101983410)

[3. Optimizacija upita Neo4j 7](#_Toc101983411)

[4. Zaključak 11](#_Toc101983412)

[5. Literatura 12](#_Toc101983413)

# Uvod

Cilj ovog rada biće upoznati se sa pojmom optimizacije upita u DBMS-u, kao i pokazati na konkretnim primerima, koristeći Neo4j, kako doći do podizanja performansi samog sistema prilikom postavljanja upita bazi. Jasno je da je sama optimizacija upita nastala sa pojavom prvih upitnih jezika, a kasnije, njihovim razvojem i konstantnim korišćenjem došlo se do adekvatnih načina za optimizaciju kako bi se što brže dobili podaci iz baze, a pritom iskoristilo što manje resursa. U nastavku će biti malo više reči o tome na koji način i kojim metodama postići odgovarajuću optimizaciju upita.

# Optimizacija upita

Da bi sistem uopšte mogao da zadovolji traženu efikasnost mora da poseduje neki optimizator upita. Optimizator upita određuje najefikasniji način da se izvrši dati upit razmatranjem mogućih planova upita. Plan upita (ili plan izvršenja upita) je niz koraka koji se koriste za pristup podacima u sistemu za upravljanje nekom bazom podataka.

Upiti se mogu napisati na mnogo različitih načina a da se pritom dobije isti rezultat prilikom njihovog izvršenja. Ovo nam govori da različita konstrukcija upita daje različiti plan izvršenja upita koji kreira optimizator upita u okviru samog sistema. Različiti planovi izvršenja upita daju različita vremena izvršenja pa samim tim i različitu efikasnost, te su zato neki planovi bolji od drugih. Prilikom pisanja upita potrebno je voditi računa o dobijanju optimalnog plana za izvršenje upita.

Da bi se postigle najbolje performanse upita treba se pridržavati sledećih preporuka:

* *rad sa malim skupovima podataka* – Smanjenjem i ograničenjem skupa podataka nad kojima se izvršava upit smanjuje se upotreba resursa i povećava efikasnost indeksa, što pozitivno utiče na performanse upita. Jedan način za smanjenje skupa podataka u upitu je da se lista kolona u okviru SELECT naredbe ograniči samo na kolone koje su neophodne, odnosno da se izbegava upit tipa SELECT \*. Drugi način je da se upotrebom WHERE klauzule smanji broj redova ukoliko je to moguće, jer se na taj način optimizacija upita realizuje efikasnijom upotrebom indeksa;
* *efektivna upotreba indeksa* – Indeksi definisani nad odgovarajućim kolonama ubrzavaju izvršenje upita. Podjednako je važno da ovi indeksi budu efikasno iskorišćeni 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. Na primer, operator „!=“ zahteva skeniranje čitave tabele iako je nad kolonom definisan indeks, što predstavlja neefikasnu upotrebu indeksa. Nepoželjni operatori, koje treba izbegavati nad kolonama sa indeksom u okviru WHERE klauzule, su operatori isključenja (<>, !=, !>, !<, NOT EXISTS, NOT IN i NOT LIKE IN, OR);
* *izbegavati davanje nagoveštaja(hints) optimizatoru upita* – Davanjem nagoveštaja optimizatoru se onemogućava da optimizator dinamički definiše plan izvršenja upita u vreme njegovog planiranja i realizacije. Na ovaj način se obično narušavaju performanse upita, pa treba izbegavati JOIN nagoveštaje (join hints), INDEX nagoveštaje (index hints) i FORCEPLAN nagoveštaje (forceplan hints) prilikom definisanja upita;
* *koristiti domenski[[1]](#footnote-1) i referencijalni[[2]](#footnote-2) integritet* – Domenski i referencijalni integritet omogućavaju optimizatoru da proveri validnost podataka bez fizičkog pristupa podacima. Smanjivanje broja fizičkih pristupa podacima, pozitivno utiče na performanse upita, smanjujući vreme izvršenja upita. Primer uticaja domenskog integriteta na performanse upita je NOT NULL ograničenje (definisanje da kolona mora sadržati neku vrednost). Definisanjem referencijalnog integriteta između dve tabele poboljšavaju se performanse JOIN upita između te dve tabele;
* *izbegavati upite koji intenzivno koriste resurse* – Upite definisati tako da se izbegava konverzija iz jednog tipa podataka u drugi tip prilikom poređenja kompatibilnih kolona tabele ili vrednosti (npr. varchar sa nvarchar), za proveru postojanja nekih podataka koristiti EXISTS umesto COUNT(\*), kad god je to moguće umesto UNION koristiti UNION ALL pri čemu treba imati u vidu da se tada, ne eliminišu duplikati u okviru rezultata upita, koristiti indekse nad kolonama nad kojima se koriste funkcije agregacije ili sortiranja i izbegavati sp prefiks kod definisanja imena ugnježdenih procedura;
* *smanjiti komunikaciju preko mreže* – Da bi se smanjila mrežna komunikacija, koja značajno utiče na vreme izvršavanja upita posmatrano sa strane aplikacije i korisnika, neophodno je smanjiti broj mrežnih ciklusa i trajanje mrežnog ciklusa. Trajanje mrežnog ciklusa je vreme potrebno da paket stigne od izvora do destinacije i opet nazad do izvora komunikacije. 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) ili kao uskladištena procedura. Smanjivanje vremena mrežnog ciklusa se može postići definisanjem da se kao rezultat upita ne vraća broj zapisa koji su obrađeni u okviru upita. Broj obrađenih redova vrlo često je nebitan podatak i može se naredbom SET NOCOUNT ON izbeći njegovo vraćanje nakon izvršenja upita;
* *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 na disku. Ovo je vremenski zahtevna operacija, koja značajno utiče na performanse upita. Broj ovih operacija se može smanjiti grupisanjem više upita u jednu transakciju;

Ono što bi takođe trebalo naglasiti to je da se pri samoj konstrukciji upita može pojaviti potreba za ugnježdenim upitom, odnosno da do nekog podatka možemo doći samo ako u WHERE klauzuli jednog upita dodamo drugi upit. U ovim situacijama je vrlo važno raščlaniti upit na manje blokove, tj. izvršiti dekompoziciju upita na manje delove. Blok je upit koji sadrži samo jednu SELECT naredbu, jednu FROM klauzulu i jednu WHERE, GROUP BY i HAVING kaluzulu. Ugnježdeni blok se optimizuje nezavisno od spoljašnjeg, ali se zato nakon optimizacije unutrašnjeg, spoljašnji optimizuje na osnovu cene poziva ugnježdenog bloka. Ono što je sigrno, to je da se verzije upita bez ugnježdenih blokova bolje optimizuju, pa je zato bolje, ukoliko je moguće, izbegavati ovakve upite.

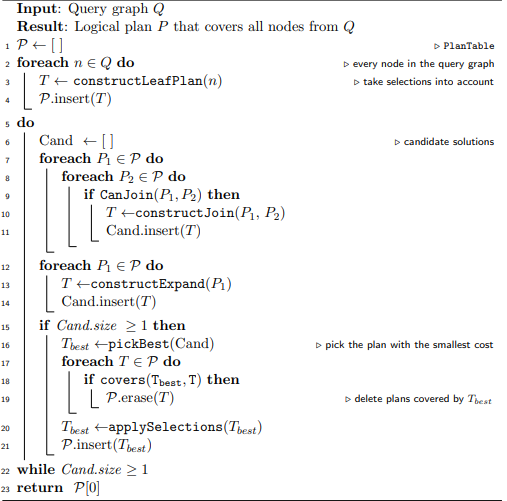
Kada smo već kod stvari koje treba izbegavati prilikom pisanja upita osvrnimo se na to koji su najčešći uzroci loših performansi upita:

* *loše indeksiranje* – prilikom izvršenja upita ako se pristupa većoj količini podataka dolazi do povećanja pristupa disku, korišćenja memorije i procesora što automastki povećava samo vreme izvršavanja upita;
* *netačna statistika* – optimizator upita na osnovu netačne statistike o distribuciji podataka može da donese pogrešnu procenu o broju zapisa koje treba da vrati, a to je važan parametar prilikom definisanja optimalnog plana izvršenja upita;
* *međusobno blokiranje transakcija i zastoj* – ukoliko dve transakcije pristupaju istom podatku u tabeli sa konfliktom dolazi do uzajamnog blokiranje jedne od transakcija i na taj način do čekanja na ponovno izvršenje, odnosno, zastoja;
* *operacije koje nisu zasnovane na radu sa skupovima podataka* – operacije koje se zasnivaju na obradi podataka upotrebom kursora i petlji;
* *loša specifikacija upita* – najčešći urok loših performansi; Upiti se definišu tako da npr. ne koriste ceo indeks;
* *loša organizacije baze podataka* – nedovoljna ili prekomerna normalizacija baze podataka može negativno uticati na vreme izvršavanja upita zbog nepotrebnog povećanja obima podataka koje upit obrađuje i povećanja broja blokiranja;
* *plan izvršenja upita koji se ne kešira* – optimizator upita troši doddatno vreme za ponovno kreiranje plana izvršenja;
* *česta rekompilacija plana izvršenja* – za izbegavanje rekompilacije plan izvršenja upita mora biti nezavisan od parametara koji se proselđuju upitu prilikom izvršenja a to se postiže korišćenjem uskladištenih procedura;
* *nepravilna upotreba kursora* – upotrebom kursora podaci se iz tabele u bazi dobavljaju red po red, retko se koriste, a kada se koriste mora se voditi računa o tome da postoje različite vrste kursora koje različito utiču na povećanje vremena izvršavanja upita;

# Optimizacija upita Neo4j

Nakon definisanja same optimizacije upita, šta ona podrazumeva i na koje sve stvari treba obratiti pažnju prilikom pisanja upita hajde da konkretizujemo celu priču na grafovskoj bazi podataka, odnosno Neo4j bazi podataka. Neo4j je NoSQL baza podataka koja umesto relacionog modela koristi koncept grafa kao model podataka. U čvorovima se čuva podatak organizovan kao ključ/vrednost dok grane predstavljaju veze između podataka. Izvršavanje upita kod ovih baza podataka svodi se na praćenje putanja između čvorova, odnosno veza među čvorovima. Upitni jezik koji koristi Neo4j naziva se Cypher.

Opšti cilj ručne optimizacije performansi upita kod grafovske baze podataka je da obezbedi da se iz grafa preuzimaju samo potrebni podaci. U najmanju ruku, podatke treba filtrirati što je ranije moguće kako bi se smanjio obim posla koji se obavlja u kasnijim fazama izvršenja upita. Ono što treba izbegavati to je vraćanje celih čvorova i veza, već samo vraćati podatke koji su zaista potrebni. Svaki Cypher upit se optimizuje i transformiše u plan izvršenja pomoću Cypher planera upita. Da bi se smanjili resursi koji se koriste za ovo najbolje bi bilo koristiti parametre umesto literala kad god je to moguće. Ovo omogućava Cypher-u da ponovo koristi iste upite umesto da analizira i pravi nove planove izvršenja. Ono što je potrebno naglasiti pre prelaska na konkretne primere jeste da se čitanje plana izvršenja vrši odozdo naviše. A sam plan izvršenja nastaje korišćenjem Greedy algoritma za kreiranje logičkih planova, koji je prikazan na slici ispod:

c

Da bismo opisali algoritam, moramo definisati specifičan odnos između para logičkih planova. Kažemo da logički plan P (algebarski izraz) pokriva drugi logički plan K, ako je F(K) ⊂ F(P), drugim rečima ako plan P pokriva sve čvorove iz plana K. U smislu graf upita to znači da K sadrži rešenje za podgraf grafa upita za P. Imajte na umu da sam K ne mora da bude podizraz za P, već da sadrži rešenje podproblema za P. Algoritam počinje postavljanjem PlanTable P planovima listova koji pokrivaju pojedinačne čvorove u grafu upita (redovi 1-4). Na svakom koraku prikupljamo rešenje kandidata koje se formira spajanjem para planova iz PlanTable (redovi 7-11) ili proširenjem jednog plana preko jedne od ivica grafa upita (redovi 12-14). Moramo imati na umu da stvarna implementacija takođe uzima u obzir najkraće putanje i agregate koji su izostavljeni u pseudokodu. Algoritam zatim bira plan kandidata sa najboljom cenom i briše sve planove iz P koji su njime obuhvaćeni (redovi 16-19). Zatim primenjujemo sve sintaksički moguće selekcije i stavljamo plan u P. Procedura se prekida čim nema kandidata za razmatranje. U ovom trenutku, pošto je graf upita povezan, PlanTable mora da sadrži jedan plan koji pokriva sve čvorove, što mi vraćamo kao rezultat. Glavna petlja (redovi 5-22) se ponavlja najviše n puta (pri čemu je n broj čvorova u grafu upita), pošto počinjemo sa n planova u P i uklanjamo najmanje jedan plan u svakom koraku. Ovim smo dobili najefikacnije logički plan od kojeg kreiramo plan izvršenja.

Recimo sada da imamo grafovsku bazu podataka koja se sastoji od čvorova koji označavaju filmove, kao i čvorova koji označavaju glumce. Ako napišemo upit u sledećem obliku:

MATCH (p {name: 'Tom Hanks'})

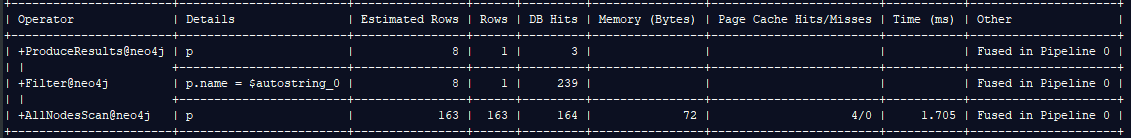
RETURN p

Dobićemo traženi rezultat, ali šta je problem ovde? Problem je što će sa povećanjem čvorova u grafu ovaj upit biti sve spori i sporiji. Ukoliko iskoristimo PROFILE[[3]](#footnote-3) komandu koju nudi Cypher možemo da dobijemo plan izvršenja i uverimo se u prethodni zaključak:

PROFILE

MATCH (p {name: 'Tom Hanks'})

RETURN p



*Rezultat PROFILE komande pre optimizacije upita*

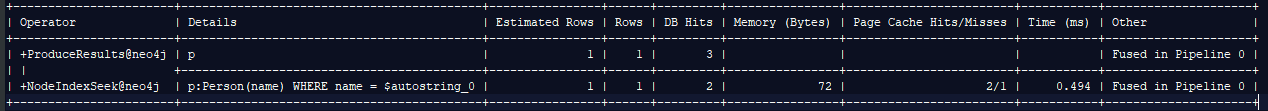
Prva stvar koju primećujemo je kolona „Rows“ koja ima preveliki broj s obzirom da u bazi postoji samo jedan čvor sa vrednošću „Tom Hanks“. Ako pogledamo kolonu „Operator“ shvatićemo da je korišćen operator „AllNodesScan“ što znači da je plan izvršenja skenirao sve čvorove u bazi. Nakon toga imamo korišćenje operatora „Filter“ koji proverava atribut „name“ kod svakog od čvorova koje je vratio prethodni operator. Jasno je sada da ovo nije dobar način za dobijanje čvora sa imenom „Tom Hanks“ jer moramo proći kroz sve čvorove u bazi da bismo našli odgovarajući.

Jedan od načina da optimizujemo izvršenje ovog upita je da nevedemo koji je tip čvora koji tražimo:

MATCH (p:Person {name: 'Tom Hanks'})

RETURN p

Ovo će dodatno suziti potragu ali i dalje neće biti najoptimalnije moguće izvršenje. Sledeći nivo optimizacije je da postavimo indeks na atribut „name“ za tip čvora „Person“. Ovako dobijemo skoro pa trenutni rezultat izvršenja što može pokazati i sam plan izvršenja ako opet upotrebimo komandu PROFILE:



*Rezultat PROFILE komande posle optimizacije upita*

Plan izvršenja se je sveo na jedan red koristeći operator „NodeIndexSeek“ dobili smo odmah odgovarajući čvor.

Uzmimo sada za primer upit koji u WHERE klauzuli neće koristiti atribut „name“ na kome smo postavili indeks:

MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

RETURN

p.name AS name,

count(m) AS count

Kao rezultat izvršenja ovog upita dobićemo listu imena nad kojom će biti potrebno uraditi dodatanu selekciju odnosno biće potrebno novo obraćanje bazi podataka. Zato je potrebno uvek u WHERE klauzuli staviti atribut na kome imamo definisani indeks. Ovaj upit bi „ispravno“ trebalo da izgleda npr. ovako:

MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

WHERE p.name STARTS WITH 'Tom'

RETURN

p.name AS name,

count(m) AS count

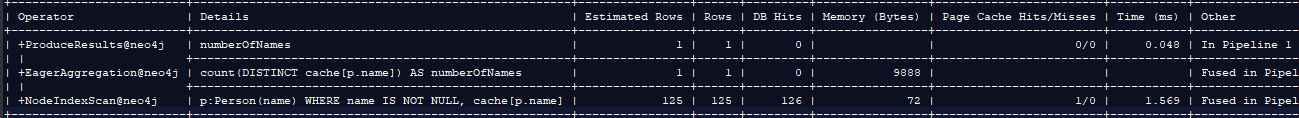
Ono što je zanimljivo za Cypher i što nam ide u korist kada koristimo ovaj upitni jezik i Neo4j bazu podataka jeste to da za sve ugrađene funkcije agregacije, optimizacija pretraživanja svojstava zasnovana na indeksu može se koristiti čak i bez predikata. Pogledajmo upit koji vraća broj različitih imena ljudi na skupu podataka koji imamo o filmovima:

PROFILE

MATCH (p:Person)

RETURN count(DISTINCT p.name) AS numberOfNames

Kako izgleda plan izvršenja:



Primetimo da „NodeIndexScan“ operator u „Details“ koloni sadrži cache[p.name] i da EagerAggregation operator nema DB pogodaka. U ovom slučaju, semantika funkcija agregacije funkcioniše kao implicitni predikat postojanja jer čvorovi „Person“ bez imena svojstva neće uticati na rezultat agregacije.

Takođe, treba naglasiti da od Neo4j 3.5 i kasnije ukoliko koristimo komandu ORDER BY ona neće biti izvršena ako Cypher planer prepozna da indeks već vraća podatke u ispravnom redosledu i preskočiće operaciju sortiranja.

Agregacije obično koriste operator „EagerAggregation“ čime je urađeno skeniranje svih čvorova u indeksu da se pronađe vrednost koja je prva po abecednom redu. Ovim štedimo vreme koje bi nam dodatno oduzeo sam operator sortiranja, pa je samim tim i efikasnost celog sistema povećana.

# Zaključak

U ovom radu pokrili smo više problema vezanih za optimizaciju upita u bazama podataka. Počevši od standardnih preporuka kojih se treba držati prilikom same konstrukcije upita pa sve do najčešćih problema koji se javljaju u kreiranim upitim i na koji način ih otkloniti. U poglavlju 3 smo, takođe, na konkretnim primerima u grafovskoj bazi podataka Neo4j pokazali kako uraditi analizu postavljenog upita a zatim i dali predloge na koji način doći do najoptimalnijeg rešenja. Sam rad predstavlja osnovu za dalju i detaljniju analizu koju je moguće sprovesti u pogledu grafovskih baza podataka i njihove optimizacije. Suptilnije, ali ponekad i ključne korelacije se mogu javiti između same strukture grafa i atributa njegovih čvorova.

Ono što zasigurno možemo reći to je da su indeksi jedna od krucijalnih struktura baze podataka i da ih je potrebno pažljivo, ali zasigurno koristiti pri samom projektovanju baze podataka, jer oni najčešće doprinose najvećoj brzini i efikasnosti samih upita.

# Literatura

[1] Query optimization

<https://en.wikipedia.org/wiki/Query_optimization>

[2] Optimizacija upita

<https://cs.elfak.ni.ac.rs/nastava/pluginfile.php/17937/mod_resource/content/2/Optimizacija%20upita%202016.pdf>

[3] Basic query tuning example

<https://neo4j.com/docs/cypher-manual/current/query-tuning/basic-example/>

[4] Advanced query tuning example

<https://neo4j.com/docs/cypher-manual/current/query-tuning/advanced-example/>

[5] Grafovske baze podataka – Neo4j

<https://blog.imi.pmf.kg.ac.rs/grafovske-baze-podataka-neo4j/>

[6] Andrey Gubichev, 2015, Query Processing and Optimization in Graph Databases, Technical University of Munich

1. Domenski integritet predstavlja ograničenje skupa vrednosti koje neki atribut skupa entiteta može da sadrži. [↑](#footnote-ref-1)
2. Referencijalni integritet je ograničenje da se neka vrednost koja se pojavljuje u jednoj relaciji mora pojavljivati i u drugoj relaciji. [↑](#footnote-ref-2)
3. PROFILE komanda se koristi kako bi se moglo videti koji operatori obavljaju većinu posla. Napomena da prilikom korišćenja ove komnde sistem troši više resursa. [↑](#footnote-ref-3)