UNIVERZITET U NIŠU

ELEKTRONSKI FAKULTET

KATEDRA ZA RAČUNARSTVO

**SEMINARSKI RAD:**

**Sistemi za upravljanje bazama podataka**

**Tema: Indeksiranje kod Neo4j DBMS i njihova implementacija**

Student:

Aleksandar Cenić 1062

S A D R Ž A J

**Strana**

[1.](#_Toc191445171) Uvod [3](#_Toc191445171)

[2.](#_Toc191445172) GDBMS [4](#_Toc191445172)

[2.1. Poređenje relacionih i graf baza podataka 5](#_Toc191445173)

[2.2.](#_Toc191445174) Neo4j [6](#_Toc191445174)

[3. Indeksiranje kod Neo4j baze podataka 8](#_Toc191445175)

[3.1. B-tree indeks](#_Toc191445176) 9

[3.1.1.](#_Toc191445176) Ograničenja9

[3.1.2.](#_Toc191445176) Procedure za kreiranje indeksa i indeksom podržana ograničenja10

[3.2. Full-text indeks 1](#_Toc191445177)1

[4.](#_Toc191445178) Praktičan primer indeksiranja13

[4.1.](#_Toc191445179) B-tree indeksiranje13

[4.2.](#_Toc191445179) Full-text indeksiranje18

[5.](#_Toc191445180) Zaključak24

Literatura25

# 

# 1. uvod

Veoma je teško da među velikim brojem podataka pronaći podatak koji nam je potreban. Kada, na primer, u knjizi tražimo određeno poglavlje pogledamo sadržaj i broj stranice kako ne bi smo listali jedan po jedan list i trošili vreme. Međutim, kada uz pomoć upita želimo da pronađemo određeni podatak, DBMS mora da prođe kroz sve podatatke sa zadatim parametrima koje je pohranio kako bi pronašao traženi podatak. Ovakav način pretraživanja podataka ne zahteva puno vremena, ako je broj podataka kroz koji treba da se prođe mali.

Sa uvećanjem broja podataka koji su prisutni u bazi, vreme potrebno za pronalaženje određenog podatka se uvecava. Postavlja se pitanje na koji način možemo povećati brzinu pronalazenja podataka? Jedano od mogucih resenja ovog problema je sortiranje podataka po određenom kritrijumu i grupisanje sličnih u određene strukture. Jedna od osnovnih struktura koje se mogu koristiti za grupisanje sličnih podataka jeste niz ili lista.

Kada imamo n elemenata u nizu, koji je nesortiran, vremenska kompleksnost pronalaženja podatka je O(n). Ovo nam govori da sa povećanjem broja n raste i potrebno vreme. Medjutim, mozemo smanjiti vremensku složenost tako što ćemo za grupisanje koristiti sortirani niz. Vremenska kompleksnost za pronalaženje potrebnog podatka u sortiranom nizu je O(log n). Pored sortiranog niza struktura koju mozemo koristiti jeste binarno stablo trazenja, cija je vremenska slozenost za pronalazenje podatka ista kao i kod sortiranog niza O(log n).

Kako svaki element u osnovnoj stukturi za grupisanje sličnih podataka ima svoju jedinstvenu poziciju koja se naziva indeks, tako je ceo postupak za poboljsanje performansi pretrage podataka u računarstvu nazvan upravo indeksiranje. U ovom radu biće obrađeno indeksiranje kod NoSQL Neo4j GDBMS-a i način njihove implementacije.

# 2. GDBMS

GDBMS predstavlja *graph database menagement system.* Graf sistem za upravljanje bazama podataka je online sistem za osnovnim CRUD metodama koje otkrivaju model podatala.

Šta predstavlja graf? Graf predstavlja skup cvorova relacija (veza) izmedju njih. Ako malo bolje pogledamo svet oko sebe, videćemo da je mnogo primera koje mozemo opisati strukturom grafa.

**Primer:**

Drustvena mreza *Instagram* lako bi se mogla predstaviti putem grafa. Profile ljudi na mrezi mozemo predstaviti ako cvorove grafa, a odnose izmedju njih putem grana grafa.

## 

Slika: 1

Fleksibilnost grafa daje graf NoSQL bazama podataka prednost u odnosu na SQL relacione baze podataka gde su entiteti predstavljeni tabelama.

Prednost i fleksibilnost se ogleda u tome sto kod graf baza podataka mozemo dodavati nove relacije (veze), cvorove, labele, podgrafove na postojecu strukturu baze bez narusavanja same te strukture baze i aplikacije. Dok kod tradicionalnih SQL baza podataka veze izmedju entiteta mozemo ostvariti tako sto cemo „udruzivati“ tabele. U nastavku ćemo izvšiti poređenje graf i relacionih baza podataka.

## 2.1 Poređenje relacionih i graf baza podataka

U tabeli 1. prikazano je poređenje relacionih i graf baza podataka s obzirom na nekoliko parametara:

* Šemu podataka.
* Upitne jezike.
* Performanse (brzina izvođenja upita nad bazom podataka).
* Način pohrane podataka.
* Mehanizam povezivanja, tj. ostvarivanja veza između podataka.

|  |  |  |
| --- | --- | --- |
|  | **Relacione baze podataka** | **Graf baze podataka** |
| **Šema podataka** | Fiksna. Svaka tablica u bazi podataka je kolekcija fiksnih atributa i njihovih vrednosti. Bilo kakve strukturne promene nad bazom podataka, tzv. migracije (dodavanje, ažuriranje ili brisanje tablica i njihovih atributa) zahtevaju puno vremena i novaca. | Fleksibilna. Ažuriranje grafa se provodi na jednostavan način uz malo utrošenog vremena I novca (potrebno je samo locirati deo grafa nad kojim se žele izvršiti promene). |
| **Upitni jezici** | SQL | Cypher, Gremlin |
| **Performanse** | Kod manjih baza podataka performanse su zadovoljavajuće i izvođenje upita nad bazom podataka je dovoljno brzo. Međutim, kod većih baza podataka brzina izvođenja upita se smanjuje, budući da se pretražuju svi podaci u bazi podataka da bi se pronašli oni koji zadovoljavaju kriterijume navedene u upitu (veliki broj JOIN operacija tokom izvođenja upita). Zbog toga je potrebno više vremena da bi se takvi podaci pronašli. (Adell, 2011) | Kod manjih baza podataka performanse su približno jednake performansama relacijskih baza podataka. S porastom količine podataka u bazi podataka, performanse ove kategorije NOSQL baza podataka su bolje u odnosu na performanse koje pružaju relacijske baze podataka u tom slučaju.Upiti se izvršavaju brže jer se traže samo oni podaci koji su direktno povezani s podacima navedenim u upitu. Upiti se izvršavaju samo nad određenim dijelom grafa koji zadovoljava kriterije opisane u upitu.  Pronalaženje susednih čvorova u grafu je jednostavno, budući da svaki čvor sadrži direktni pokazivač na njemu susjedan čvor, bez ikakvih indeksa. |
| **Način pohrane podataka** | Podaci se pohranjuju u tablice. | Podaci se pohranjuju u čvorove. |
| **Mehanizam povezivanja podataka** | Veze među podacima pohranjenim u različitim tablicama ostvaruju se pomoću spoljnih ključeva ili dodatnih tablica. | Veze među podacima se ostvaruju direktno povezivanjem čvorova, gde veze pritom mogu biti imenovane i sadržavati vlastite atribute. |

Tablela: 1

Na temelju prikazanog poređenja može se zaključiti kako su relacione baze podataka pogodne za upotrebu u situacijama kad u bazu podataka nije potrebno pohraniti veću količinu podataka (primera radi tradicionalne poslovne aplikacije) te u kojima je u početku poznata šema podataka koji će pohraniti u bazi podataka (tabelarni i dobro strukturirani podaci), a koja se u budućnosti neće često menjati.

Suprotno tome, u situacijama u kojima je u bazu podataka potrebno pohraniti veliku količinu podataka sa puno korisnika (primera radi savremene društvene mreže), a u kojima shema podataka nije u potpunosti poznata u početku i često se mijenja tokom vremena, poželjno je koristiti graf baze podataka. Prema tome, relacione baze podataka su **orijentirane šemi** (eng. Schema-oriented), dok su graf baze podataka **orijentirane pojavama** (eng. Occurence-oriented) ( Zicari, 2013).

**2.2 Neo4j**

Neo4j je prema definiciji „robusna, skalabilna baza podataka visokih performansi“ (s.n., The Neo4j Manual v2.1.2, 2014) pa se u praksi može koristiti za pohranjivanje svih ili samo dela podataka s glavnih servera velikih poreduzeća. Tipična je open-source graf baza podataka razvijena Neo Technology kompanije.

Slika: 2

Neo4j baza podataka omogućuje da čvorovi i veze sadrže svojstva, odnosno atribute. Atributi predstavljaju parove ključ-vrednost, gde je ključ skup znakova, odnosno string, a vrednost može biti jednostavna (string, broj) ili složena (polje jednostavnih tipova). Za razliku od relacionih baza podataka, NULL vrednost atributa se ne smatra valjanom i u tom se slučaju radije izostavlja ključ s tom vrednošću. Pritom svaki čvor može imati jednu ili više oznaka.

Neo4j je baza podataka nezvisna o šemi, premda prisutnost šeme pridonosi poboljšanju performansi i modeliranju. Osim šeme performanse Neo4j baze podataka mogu se poboljšati kreiranjem indeksa kojima se ubrzava pretraživanje čvorova u bazi podataka. U narednom poglavlju ćemo detaljnije obraditi indeksiranje kod Neo4j baze podataka.

# 3. Indeksiranje kod neo4j baze podataka

Indeks kod baza podataka, predstavlja redudantnu kopiju određenih podataka iz same baze u svrhu poboljšanja performansi pretraživanja povezanih podataka. Cena poboljšanja performansi pretraživanja putem indeksa je memorija za smeštanje indeksa i sporiji upis, jer odluka šta pripada indeksu, a šta ne pripada, je važan i čest ne-trivijalni zadatak. Pritom se kreirani indeksi s vremenom automatski ažuriraju u skladu s promenama nad određenim čvorovima i njihovim svojstvima. Međutim, indeksi nisu stalno dostupni. Prilikom prvog kreiranja indeksa, on je odmah dostupan, ali nakon toga se ažurira i upotpunjava u pozadini pa tek nakon toga postaje dostupan za pretraživanje.

Kod Neo4j baze podataka indeksi mogu biti kreirani i odbačeni (obrisani) uz pomoć upitnog jezika Cypher-a. Korisnici ne moraju znati ništa o indeksu u nameri da ga koriste, Cypher-ov planer upita će odlučiti koji indeks da upotrebi u zavisnosti od situacije.

Cypher nam omogućava da kreiramo indeks na jedan ili više property-ja (svojstava) svim čvorovima koji imaju dodeljenu labelu (oznaku):

* Indeks kreiran na samo jednom property-ju, za bilo koju datu labelu naziva se single-property index.
* Indeks kreiran na više property-ja, za bilo koju datu labelu naziva se composite index.

**Primer: Kreiranje single-property index-a:**

Index na samo jednom property-ju za sve čvorove sa datom labelom može se kreirati sa uputom: CREATE INDEX index\_name FOR (n:Label) ON (n.property)

CREATE INDEX index\_name FOR (n:Person)

ON (n.surname)

**Primer: Kreiranje *composite index-a*:**

Indeks na više property-ja za sve čvorove sa datom labelom može se kreirati sa upitom:

CREATE INDEX index\_name FOR (n:Label) ON (n.prop1, …​, n.propN)

Samo čvorovi označeni specificiranom labelom i čvorovi koji sadrže sve property-je specificirane u definiciji indeksa.

CREATE INDEX index\_name FOR (n:Person)

ON (n.age, n.country)

**Primer: Odbacivanje (brisanje) single-property index-a:**

Index na samo jednom property-ju za sve čvorove sa datom labelom može se odbaciti (obrisati) upitom: DROP INDEX ON :Label(property)

DROP INDEX ON :Person(firstname)

**Primer: Odbacivanje (brisanje) *composite index-a*:**

Indeks na više property-ja za sve čvorove sa datom labelom može se odbaciti (obrisati) upitom: DROP INDEX ON :Label(prop1, …., propN)

DROP INDEX ON :Person(age, country)

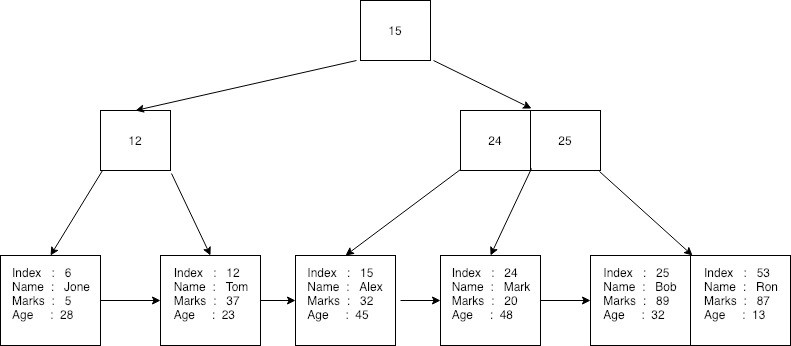
Upit će odbaciti indeks na svim čvorovima sa property-ma age i country.

Kod Neo4j-a postoje dva različita tipa indeksa:

* B-tree ideksi
* Full text indeksi

**3.1 B-tree indeks**

U račinarskoj nauci, B-tree je samobalansirajuća struktura podataka tipa stabla koja čuva sortirane podatke i omogućava sekvencijalni pristup, dodavanje i brisanje podataka. U B-tree strukturi podaci se čuvaju tako da svaki od čvora sadrži ključeve poređane u rastućem redosledu. Saki ključ poseduje dve reference ka dva čvora (deteta). Kljucevi levog deteta su manji od ključeva roditelja, dok su ključevi desnog deteta veći od roditeljskih ključeva. Ako cvor u stablu sadrži n broj ključeva onda može maksimalno imati n+1 deteta.



Slika: 3

B-tree indeks je pogodan za precizan pregled podataka za sve tipove vrednosti i skeniranje opsega, kompletno skeniranje i prefiksno traženje. B-tree index kod Neo4j može biti podržan od dva različita provajdera, **native-btree-1.0** i **lucene+native-3.0**. Ako nije eksplicitno postavljeno koristiće se **native-btree-1.0**.

**3.1.1 Ograničenja**

**Ograničenja za upite koji koriste CONTAINS i ENDS WITH**

Indeks provajder **native-btree-1.0** ima ograničenu podršku za **CONTAINS** i **ENDS WITH** upite. Ovi upiti neće biti sposobni da urade optimizovanu pretragu. U budućnosti ovakvi upiti biće podržani samo od full text indeksa.

**Ograničenja veličine ključa**

Indeks provajder **native-btree-1.0** ima ograničenu veličinu ključa oko 8kB.

Ako transakcija prekorači ograničenu veličinu ključa, ta transakcija će propasti pre podtvrde bilo kakvih promena. Ako se prekorači ograničenje tokom indeksiranja, to će rezultirati neuspešnim stanjem indeksa i indeks će biti neupotrebljiv. Ukoliko nam je potrebna veća veličina ključa možemo koristiti provajder **lucene+native-3.0** čije je ograničenje ključa 32kB.

**3.1.2 Procedure za kreiranje indeksa i indeksom podržana ograničenja**

Indekse i ograničenja najbolje je kreirati uz pomoć Cypher-a, ali kada je potrebno detaljnije konfigurisati indekse i ograničenja, tada možemo koristiti sledeće procedure:

**db.createIndex**: Procedura za kreiranje indeksa sa ograničenjem:

CALL db.createIndex("MyIndex", ["Person"], ["name"], "native-btree-1.0", {+`spatial.cartesian.max+`: [100.0,100.0], +`spatial.cartesian.min+`: [-100.0,- 100.0]})

**db.createUniquePropertyConstraint**: Procedura za kreiranje ograničenja jedinstvenog property-ja čvora: CALL db.createUniquePropertyConstraint("MyIndex", ["Person"], ["name"], "native- btree-1.0", {+`spatial.cartesian.max+`: [100.0,100.0], +`spatial.cartesian.min+`: [-100.0,-100.0]})

**db.createNodeKey**: Procedura za kreiranje ograničenja ključa čvora: CALL db.createNodeKey("MyIndex", ["Person"], ["name"], "native-btree-1.0", {+`spatial.cartesian.max+`: [100.0,100.0], +`spatial.cartesian.min+`: [-100.0,- 100.0]})

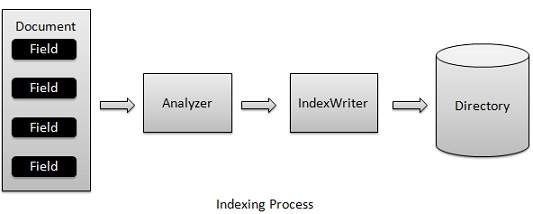
**3.2 Full-text indeks**

Full-text indekse pokreće Apache Lucene biblioteka za indeksiranje i pretragu. Može se koristiti za indeksiranje čvorova i njihovih veza po string property-ma. Full-text nam dozvoljava da pišemo upite koji poklapaju sadržaj indeksa i string property-ja. Radi poređenja, predhodnoni B-tree indeksi mogu vršiti precizno ili prefiksno poklapanje na stringove. Full-text indeks će tokenizirati string vrednosti i moći će da uradi poklapanje sa bilo kojim terminom u stringu. Kako su indeksirani stringovi tokenizirani i podeljeni na termiine, odlučuje to sa kojim analizatorom je full-text indeks konfigurisan.

Dostupne analizatore možemo izlistati pozivom procedure:

**CALL db.index.fulltext.listAvailableAnalyzers;**

U uprošćavanju termina i tokeniziranju, Lucene tok indeksiranja prolazi kroz dva koraka. Prvi korak je **Analizator**, koji vrši procesiranje teksta. Drugi korak je **IndexWriter**, koji vrši upis rezultata analizatora u indeks.



Slika: 4

Neke osobine full-text indeksa:

* Podržava indeksiranje čvorova i veza.
* Podržava konfigurisanje posebnog analizatora čak i onog koji nije uključen u Lucene.
* Mogu se postavljati upiti i sa Lucene upitnim jezikom.
* Mogu vraćati poene za svaki rezultat upita.
* Održava indekse ažuriranim automacki, kako se novi čvorovi i veze dodaju ili brišu.

**3.2.1 Procedure za korišćenje full-text indeksa**

Full-text indeksi se mogu kreirati uu pomoć procedura:

* **db.index.fulltext.createNodeIndex**
* **db.index.fulltext.createRelationshipIndex**

Indeksu moramo dodeliti jedinstveni naziv kada ga kreiramo, jer koristi za referenciranje određenog indeksa kada pravimo upit ili ga brišemo. Indeks se veže za listu čvorova ili listu veza, zavisno kako specificiramo.

CALL db.index.fulltext.createNodeIndex("titlesAndDescriptions",["Movie", "Book"], ["title", "description"])

Sada je čvor ubačen u indeks čak i ako je jedini sa tom oznakom i samo sa jednim property-jem.

CALL db.index.fulltext.createRelationshipIndex("taggedByRelationshipIndex", ["TAGGED\_AS"],["taggedByUser"],{analyzer:"url\_or\_email", eventually\_consistent: "true" })

Sada je veza ubacena u indeks. U narednom poglavlju cemo praktično obraditi indekse.

# 4. Praktičan primer indeksiranja

U ovom poglavlju ćemo pokazati praktičnu primenu indeksa. Da bi smo uspeli pokazati praktično sve prednosti koje indeksiranje pruža, moramo imati najmanje 100 miliona podataka. Kako je vrlo teško sakupiti 100 miliona podataka, kreiraćemo uz pomoć skripte test podatke koji su nam potrebni.

WITH ["Aleksandar","Nemanja","Nenad","Marko","Petar","Dijana","Milica","Sneza","Maksa","Ceda","Ana","Matea","Jelena","Maja","Vesna"] AS names,

["Cenic","Markovic","Ilic","Janic","Tomic","Ristic","Stojkovic","Jovanovic","Makulovic","Petrovic","Lazic","Maric","Nikolic","Aksic"] AS lastnames,

[1994,1995,1993,1996,1991,1992,1990,1997,1998,1989,1988,1987,1977,1979] AS birthyears

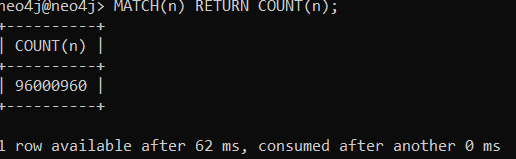
FOREACH (r IN range(0,100000) | CREATE (:User {name:names[timestamp()\*r % size(names)], lastname:lastnames[timestamp()\*r % size(lastnames)], birthyear:birthyears[timestamp()\*r % size(birthyears)]}));

Pre izvršenja skripte, moramo u Neo4j otkomentarisati komandu za inicijalnu i max heap velicinu i povećati max heap. Skripta prolazi kroz definisane nizove 100 hiljada i jedanput. Nizovi sadrže podatke za imena, prezimena i godinu rođenja. Kako ne bi došlo do greške, jer nam je potrebno više miliona podataka, računaru je lakše da ovu skriptu izvrši više puta. U prilogu ovog rada biće dat txt fajil sa, od prilike, 1000 puta kopiranom skriptom koja se iskopira u Cypher-shell i sačeka da se kreiraju podaci.

**NAPOMENA**: Izvršavanje više puta iskopirane skripte zahtevaće malo više vremena i memorije za skladištenje podataka od više gigabajta.

**4.1 B-tree indeksiranje**

Na početku ćemo izvršiti skriptu nekoliko puta i potom videti koliko ima čvorova u bazi.

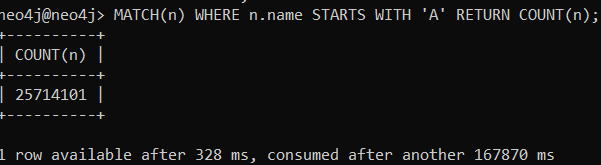


Slika: 5

U nastavku ćemo izvršiti nekoliko upita nad podacima koji nisu indeksirani i zabeležiti vreme za koje su se upiti izvršili. Potom napraviti indeks nad oređenim property-jem i zatim ponovo izvršiti upite, zabeležiti vreme uporediti sa predhodnim zabeleženim vremenima i izračunati postotak popoljšanja.

Sada ćemo izvršiti 4 upita i pogledati vreme za koje su se izvršili:

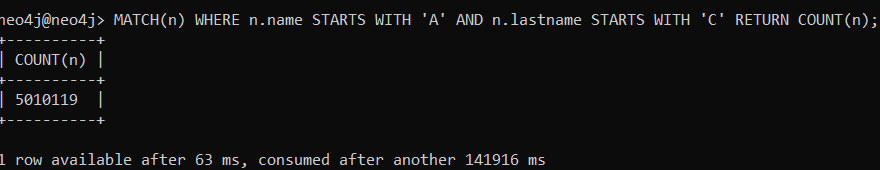
1. Broj svih korisnika kojima ime počinje na A: MATCH(n) WHERE n.name STARTS WITH 'A' RETURN COUNT(n);



Slika: 6

Kao što vidimo upit se izvršio za: 328ms+167870ms=168198ms=168s=2m i 48s.

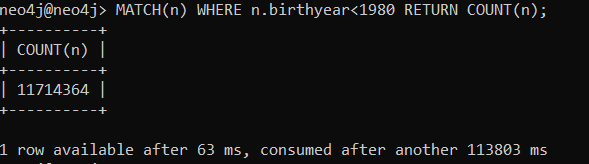
1. Broj svih korisnika kojima ime počinje na A i prezime počinje na C: MATCH(n) WHERE n.name STARTS WITH 'A' AND n.lastname STARTS WITH 'C' RETURN COUNT(n);



Slika: 7

Upit se izvršio za: 63ms+141916ms=141979ms=141s=2m i 21s.

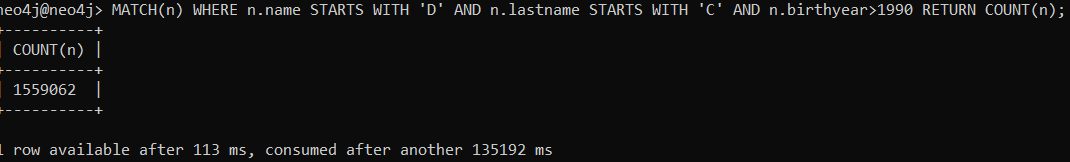
1. Broj svih korisnika čija je godina rođenja manja od 1980: MATCH(n) WHERE n.birthyear<1980 RETURN COUNT(n);



Slika: 8

Upit se izvršio za: 63ms+113803ms=113866ms=113s=1m i 53s.

1. Broj svih korisnika čije ime počinje na D, prezime počinje na C i godina rođenja veća od 1990: MATCH(n) WHERE n.name STARTS WITH 'D' AND n.lastname STARTS WITH 'C' AND n.birthyear>1990 RETURN COUNT(n);



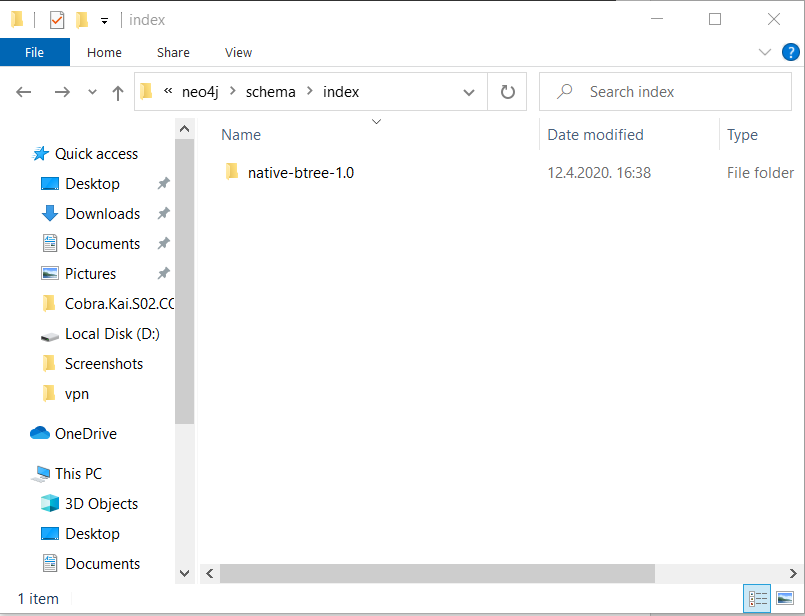
Slika: 9

Upit se izvršio za: 113ms+135192ms=135305ms=135s=1m i 15s.

Sada ćemo kreirati B-tree indeks na property-ju name, nakon toga ponovo izvršiti upite i uporediti vreme izvršenja. Indeks kreiramo komandom: CREATE INDEX ON :User(name);



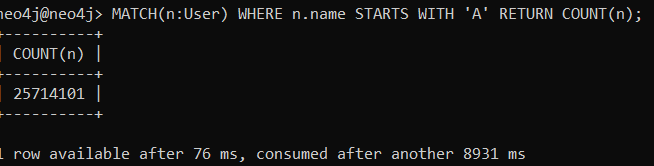
Slika: 10



Slika: 11

Kako što vidimo na slici 11, u folderu data\databases\neo4j\schema\index kreirao se folder **native-btree-1.0** koji sadrži indeks. Sada krećemo sa ponovnim izvršenjem predhodnih upita.

1. MATCH(n:User) WHERE n.name STARTS WITH 'A' RETURN COUNT(n) ;

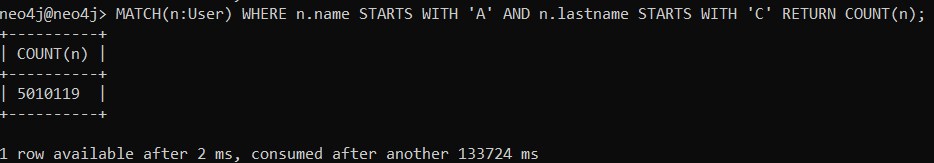


Slika: 12

Upit se izvršio za: 76ms+8931ms=9007ms=9s.

**NAPOMENA**: Nakon kreiranja indeksa potrebno je proći izvesno vreme kako bi indeks postao dostupan.

1. MATCH(n:User) WHERE n.name STARTS WITH 'A' AND n.lastname STARTS WITH 'C' RETURN COUNT(n);

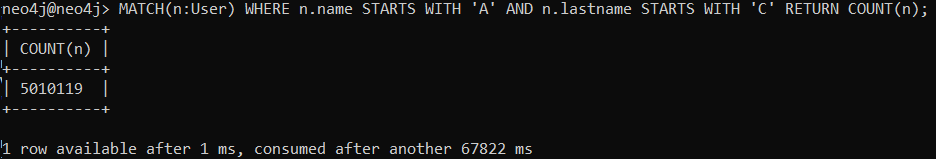


Slika: 13

Kao što vidimo poboljšanja na ovom upitu nema zbog parametara lastname koji nije indeksiran. Zato ćemo napraviti kompozitni indeks na property-ma name i lastname: CREATE INDEX ime\_i\_prezime FOR (n:User) ON (n.name,n.lastname);



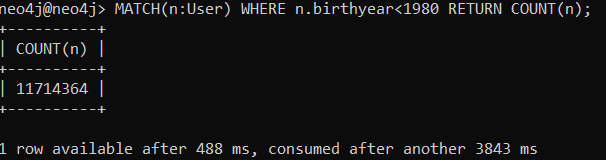
Slika: 14



Slika: 15

Sada na izvršenju ovog upita imamo poboljšanje, jer se upit izvršio za: 1ms+67822ms=67823ms=67s=1m i 17s. Za sledeći upit pravimo indeks na birthyear property-ju. CREATE INDEX ON :User(birthyear);

1. MATCH(n:User) WHERE n.birthyear<1980 RETURN COUNT(n);



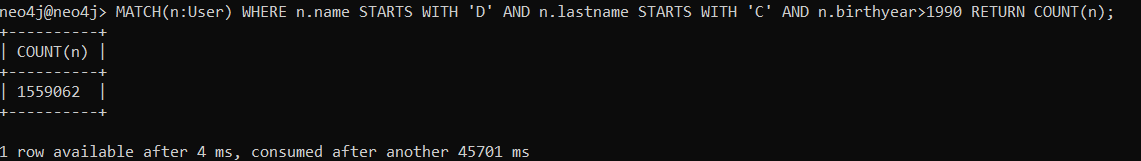
Slika: 15

Upit se izvršio za: 488ms+3843ms=4331ms=4s. Za četvrti upit napravićemo kompozitni indeks na sva tri property-ja name, lastname i birthyear: CREATE INDEX sve FOR(n:User) ON(n.name,n.lastname,n.birthyear);



Slika: 16

1. MATCH(n:User) WHERE n.name STARTS WITH 'D' AND n.lastname STARTS WITH 'C' AND n.birthyear>1990 RETURN COUNT(n);



Slika: 17

Upit se izvršio za: 4ms+45701ms=45705ms=45s.

Kako bi smo izračunali postotak poboljšanja sabraćemo vremena upita bez i sa indeksima i upotrebiti formulu: 100-Vreme\_izvršenja\_bez\_indeksa/Vreme\_izvršenja\_sa\_indeksom\*100.

|  |  |
| --- | --- |
| Vreme\_izvršenja\_bez\_indeksa | Vreme\_izvršenja\_sa\_indeksom |
| 1. 168198ms | 1) 9007ms |
| 1. 141979ms | 2) 67823ms |
| 1. 113866ms | 3) 4331ms |
| 1. 135305ms | 4) 45705ms |
| Ukupno = 559348ms | Ukupno = 126866ms |

Tablela: 2

Poboljšanje=100-559348/126866\*100=56%

**4.2 Full-text indeksiranje**

Na primeru B-tree indeksiranja smo pokazali klasičnu moć indeksiranja. Redukovali smo vreme koje je potreno za pronalaženje podataka. U ovom delu pokazaćemo razliku između B-tree i Full-text indeksiranja. Kao što smo naveli u predhodnom poglavlju, Full-text indeksiranje, koristi se za indeksiranje string property-ja i za razliku od B-tree indeksiranja možemo indeksirati i relacije. Koristićemo modifikovanu skriptu iz prošlog primera.

WITH ["Aleksandar","Nemanja","Nenad","Marko","Petar","Dijana","Milica","Sneza","Maksa","Ceda","Ana","Matea","Jelena","Maja","Vesna"] AS names,

["Cenic","Markovic","Ilic","Janic","Tomic","Ristic","Stojkovic","Jovanovic","Makulovic","Petrovic","Lazic","Maric","Nikolic","Aksic"] AS lastnames,

[1994,1995,1993,1996,1991,1992,1990,1997,1998,1989,1988,1987,1977,1979] AS birthyears,

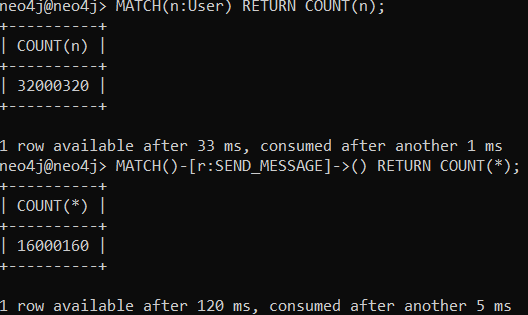
["Kako si?","Gde si","Šta radiš?","Šta ima?","Šta je bilo","Šta radiš?"] AS messages,

["09:44","07:10","10:35","20:05","12:00","15:00","16:50","21:21"] AS times,

["10.11","12.9","04.1","05.5","01.20","02.1"] AS dates

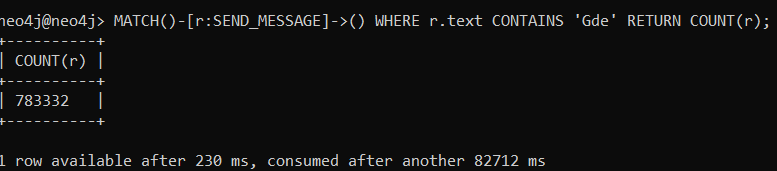
FOREACH (r IN range(0,100000) | CREATE (u:User {name:names[timestamp()\*r % size(names)], lastname:lastnames[timestamp()\*r % size(lastnames)], birthyear:birthyears[timestamp()\*r % size(birthyears)]})-[m:SEND\_MESSAGE {text:messages[timestamp()\*r % size(messages)], time:times[timestamp()\*r % size(times)], date:dates[timestamp()\*r % size(dates)]}]->(u2:User {name:names[timestamp()\*r % size(names)], lastname:lastnames[timestamp()\*r % size(lastnames)], birthyear:birthyears[timestamp()\*r % size(birthyears)]}));

U ovoj skripti se pored čvorova se automacki kreiraju i relacije SEND\_MESSAGE sa atributima text, time i date. Sada cemo izvršiti 4 upita, nakon toga napraviti Full-text indekse ponovo izvršiti upite uporditi vremena i videti score. Broj čvorova i relacija sa kojima radimo je:



Slika: 18

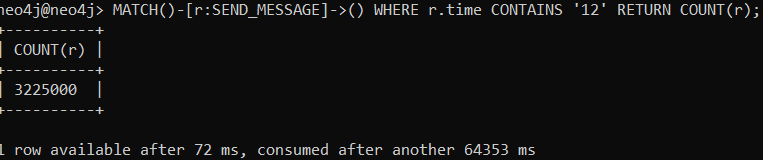
1. Prvi upit koji ćemo izvršiti je vraćanje broja relacija SEND\_MESSAGE koji u text-u sadrže reč „Gde“. MATCH()-[r:SEND\_MESSAGE]->() WHERE r.text CONTAINS 'Gde' RETURN COUNT(r);



Slika: 19

Upit se izvršio za: 230ms+82712ms=82942ms=82s=1m i 22s.

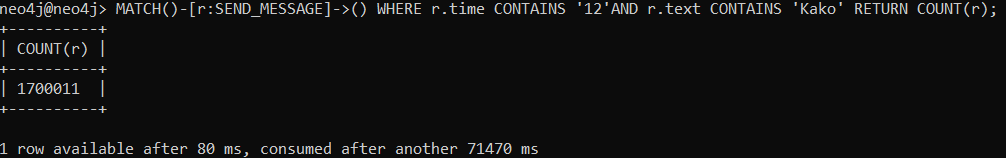
1. Upit koji vraća broj relacija (poruka) poslatih u 12h. MATCH()-[r:SEND\_MESSAGE]->() WHERE r.time CONTAINS '12' RETURN COUNT(r);



Slika: 20

Upit se izvršio za: 72ms+64353ms=64425ms=64s=1m i 4s.

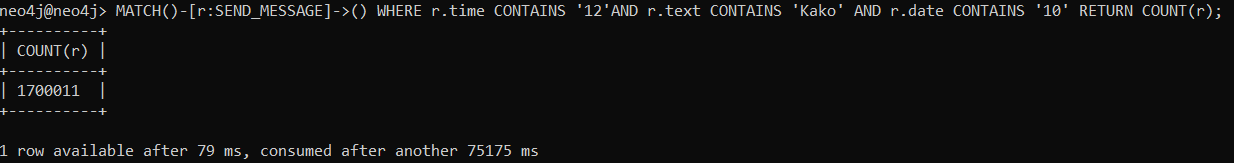
1. Upit koji vraća broj relacija (poruka) poslatih u 12 I koje u tekstu sadrže reč “Kako”. MATCH()-[r:SEND\_MESSAGE]->() WHERE r.time CONTAINS '12'AND r.text CONTAINS 'Kako' RETURN COUNT(r);



Slika: 21

Upit se izvršio za: 80ms+71470ms=71550ms=7.5s

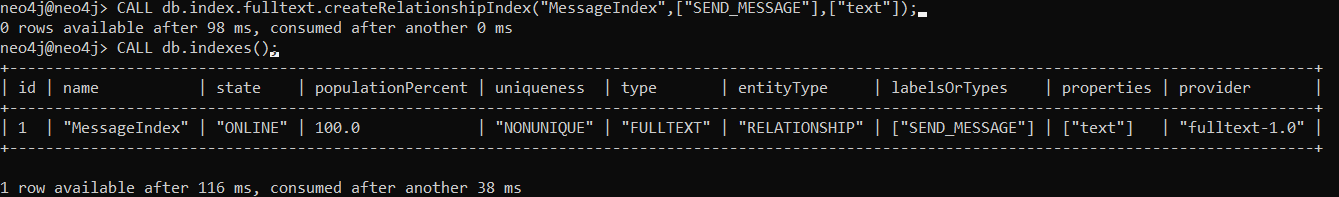
1. Upit koji vraća broj relacija (poruka) koje u tekstu sadrže reč “Kako”, poslate u 12h i 10-og meseca ili dana. MATCH()-[r:SEND\_MESSAGE]->() WHERE r.time CONTAINS '12'AND r.text CONTAINS 'Kako' AND r.date CONTAINS '10' RETURN COUNT(r);



Slika: 22

Upit se izvršio za: 79ms+75175ms=75254ms=75s=1m i 15s.

Sada ćemo da napravimo Full-text indeks. CALL db.index.fulltext.createRelationshipIndex("MessageIndex",["SEND\_MESSAGE"],["text"]);

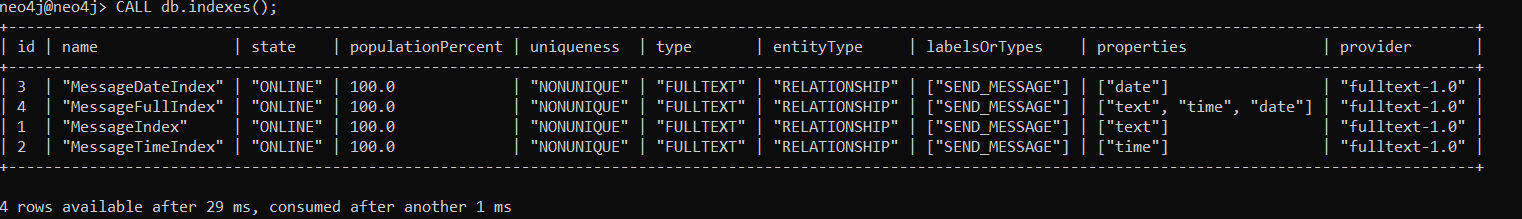


Slika: 23

Kreiramo sada indekse na ostalim property-ma. CALL db.index.fulltext.createRelationshipIndex("MessageTimeIndex",["SEND\_MESSAGE"],["time"]);

CALL db.index.fulltext.createRelationshipIndex("MessageDateIndex",["SEND\_MESSAGE"],["date"]);

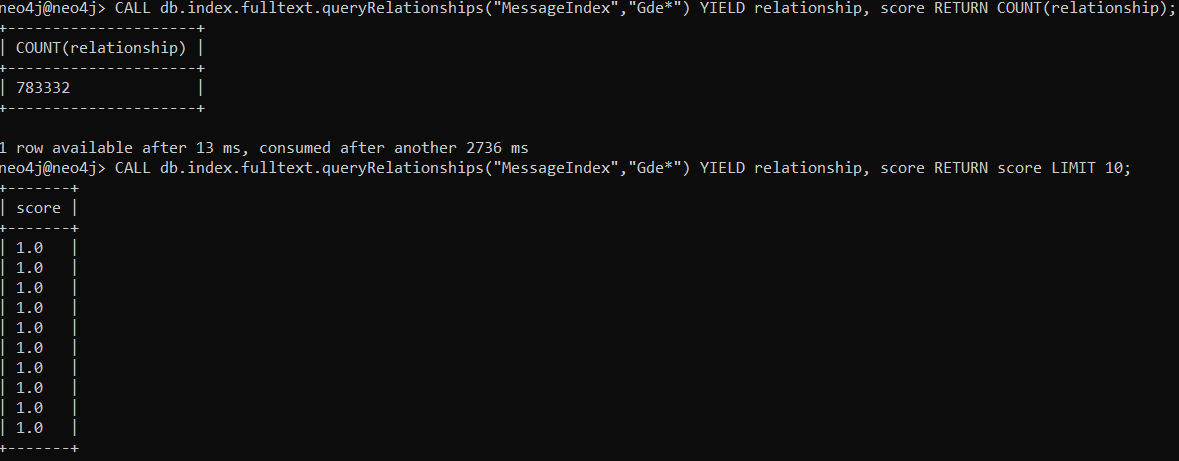
CALL db.index.fulltext.createRelationshipIndex("MessageFullIndex",["SEND\_MESSAGE"],["text","time","date"]);



Slika: 24

U nastavku izvršićemo upite nad indeksiranim podacima.

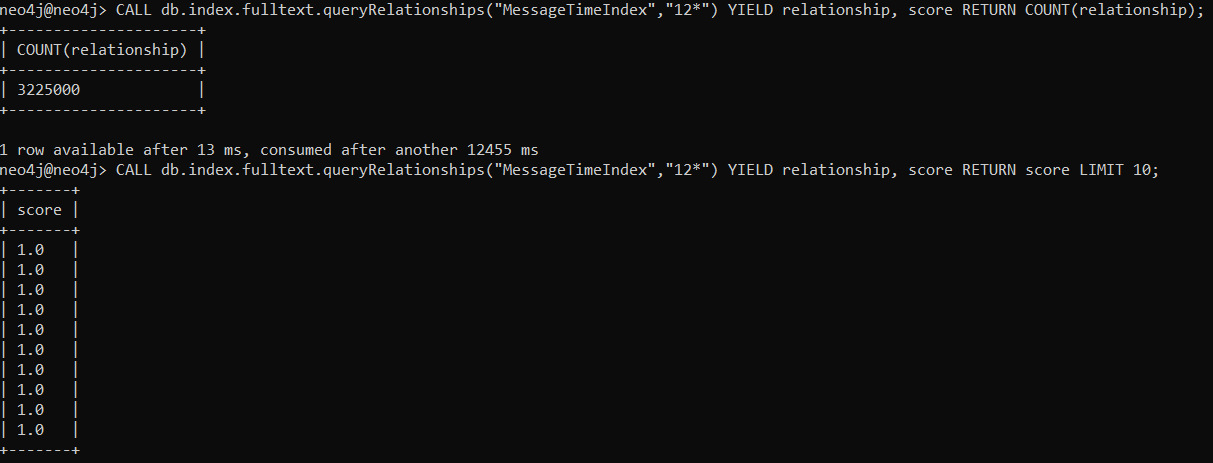
1. CALL db.index.fulltext.queryRelationships("MessageIndex","Gde\*") YIELD relationship, score RETURN COUNT(relationship);



Slika: 25

Upit se izvršio za: 13ms+2736ms=2749ms=2s. Vidomo da je score 1.0.

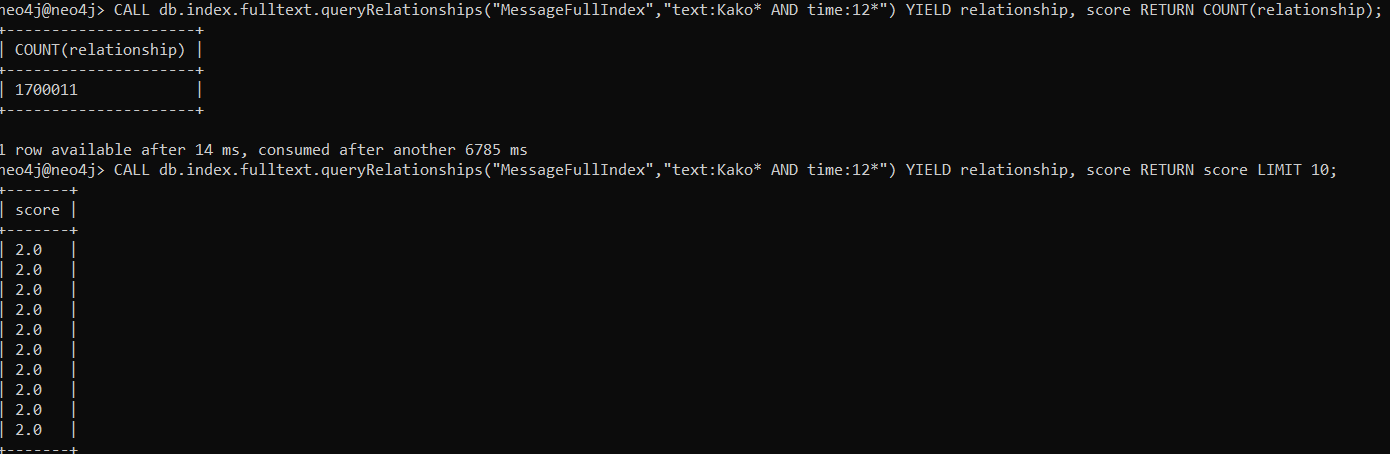
1. CALL db.index.fulltext.queryRelationships("MessageTimeIndex","12\*") YIELD relationship, score RETURN COUNT(relationship);



Slika: 26

Upit se izvršio za: 13ms+12455ms=12468ms=12s i score je 1.0.

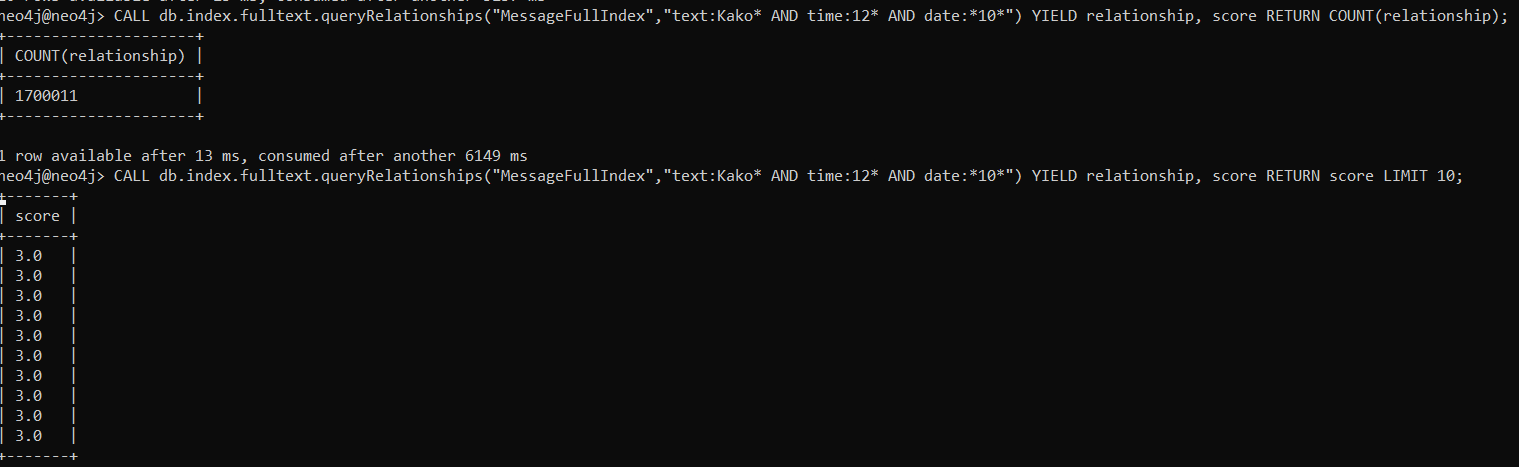
1. CALL db.index.fulltext.queryRelationships("MessageFullIndex","text:Kako\* AND time:12\*") YIELD relationship, score RETURN COUNT(relationship);



Slika: 27

Upit se izvršio za: 14ms+6785ms=6799ms=6s i score je 2.0.

1. CALL db.index.fulltext.queryRelationships("MessageFullIndex","text:Kako\* AND time:12\* AND date:\*10\*") YIELD relationship, score RETURN COUNT(relationship);



Slika: 28

Upit se izvršio za: 13ms+6149ms=6162ms=6s i score je 3.0.

Kao I u predhodnom primeru upoređujemo vremena I izračunavamo poboljšanje.

|  |  |
| --- | --- |
| Vreme\_izvršenja\_bez\_indeksa | Vreme\_izvršenja\_sa\_indeksom |
| 1. 82942ms | 1) 2749ms |
| 1. 64425ms | 2) 12455ms |
| 1. 71550ms | 3) 6799ms |
| 1. 75254ms | 4) 6162ms |
| Ukupno = 294171ms | Ukupno = 28165ms |

Tablela: 3

Kako bi smo izračunali postotak poboljšanja sabraćemo vremena upita bez i sa indeksima i upotrebiti formulu: 100-Vreme\_izvršenja\_bez\_indeksa/Vreme\_izvršenja\_sa\_indeksom\*100.

Pobljšanje iznosi: 100-294171/28165\*100= približno 91%.

# 5. zaključak

Indeksiranje se koristi kako bi smo poboljšali performance pretraživanja podataka u bazi.

Videli smo kroz praktične primere da je indeksiranje dalo značajan postotak poboljšanja pretraživanja podataka. U prvom primeru postotak poboljšanja je bio 56%, dok u drugom primeru poboljšanje je približno 91%. B-tree tip indeksa, pogodno je koristiti kada se indeksiraju samo čvorovi koji imaju različite tipove property-ja (Većim brojem numeričke tipove) za pretraživanje podataka o korisnicima. Drugi tip indeksa Full-text, kao što mu samo ime kaže, pogodan je kada treba pretraživati text, a i veze (relacije) i čvorove koji sadrže dosta string property-ja. Na primer kada se pretražuje baza komentara ili blogova.

# literatura

[1] Knjiga: Graph Databases. Ian Robinson, Jim Webber & Emil Eifrem. (Prvo poglavlje).

[2] Završni rad: Graf baze podataka. Martina Šestak.

(3.5. Usporedba relacijskih i graf baza podataka, 5. Neo4j).

[3] Članak: Neo4j 4.0 Cypher Manuel. (5.2. Indexes for search performance, 5.3. Indexes for full-text search ). Link1:<https://neo4j.com/docs/cypher-manual/current/administration/indexes-for-search-performance/>

Link2:<https://neo4j.com/docs/cypher-manual/current/administration/indexes-for-full-text-search/>

[4] Članak: How Database B-Tree Indexing Works. Link:<https://dzone.com/articles/database-btree-indexing-in-sqlite>

[5] Članak: Wikipedia B-tree. Link: <https://en.wikipedia.org/wiki/B-tree>

[6] Članak:Exploring the full-text search index in Neo4j on a movies dataset. Link: <https://towardsdatascience.com/exploring-the-full-text-search-index-in-neo4j-on-a-movies-dataset-3cddca69db7a?gi=b3d818328573>

Датум предаје: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Комисија:

Председник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Испитивач \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члан \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Коментар:

Датум одбране: \_\_\_\_\_\_\_\_\_\_\_\_\_ Оцена\_\_\_\_\_\_\_\_\_\_ (\_\_\_)