Neo4j – Interna struktura i organizacija skladišta podataka

Student: Anđela Sokolović, 944

Sadržaj

1. Uvod

U današnje vreme, kada je svaki podatak koji se generiše gotovo istog trenutka povezan sa milionima drugih podataka, graf baze podataka stiču sve veću popularnost. Naime, njihova glavna karakteristika je upravo obrada podataka koji imaju visok nivo povezanosti. Više nije od značaja posmatrati podatak kao zasebnu celinu, već je fokus na analizi povezanosti tog podatka sa mnogim drugim, zbog čega ove baze nalaze primenu u društvenim mrežama, raznim pretraživačima, kao i u marketingu i serviranju pravog tipa reklame odgovarajućem korisniku, i veoma su interesantne za proučavanje i primenu.

U ovom radu fokus će biti na bazi podataka Neo4j i njenoj internoj strukturi, kao i organizaciji skladišta podataka. Pri tome će biti osvrta i na nerelacione baze podataka kojima Neo4j i pripada, a uz to će pažnja biti posvećena i graf bazama podataka uopšte.

2. Nerelacione baze podataka

Pre nego što se posvetimo implementaciji konkretne nerelacione baze kao što je Neo4j, valja se podsetiti šta su nerelacione baze uopšteno. Nerelaciona baza podataka, takođe poznata kao NoSQL baza, predstavlja svaku bazu koja ne koristi tradicionalni način smeštanja podataka u tabele, već kao skladište koristi neku drugu strukturu. Osnovna prednost ovakve baze je veća skalabilnost i fleksibilnost u odnosu na relacione prethodnike. Postoji puno vrsta nerelacionih baza, od kojih su najpoznatije *Key-value, Document store* i *Graph* baze podataka. Graf bazi će biti posvećena posebna pažnja u nastavku, sa fokusom na njenom glavnom predstavniku: Bazi Neo4j.

3. Šta je graf baza podataka?

Kao što joj ime kaže, graf baza podataka kao internu strukturu koristi graf, u čije čvorove i potege se smeštaju informacije. Jedna od karakteristika koja ovu bazu ističe je to što se relacije među entitetima smatraju jednako bitnim kao i sami entiteti, zbog čega poteg između dva čvora može nositi istu količinu informacija kao i sam čvor. Zbog prirode grafa i njegovog obilaska, ovakva baza je od izuzetne koristi pri analizi jako povezanih podataka i kompleksnih upita. Graf baze podataka se mogu koristiti za smeštanje bilo kog grafa, ali postoje i specijalizovane graf baze: *Triplestore i Network.*

Triplestore (RDF)

Triplestore baza podataka spada u graf baze podataka, ali je u svojoj implementaciji konkretizovana i predviđena za pronalaženje takozvanih *trojki* – semantičkih celina u vidu subjekta, predikta i objekta. RDF naziv potiče od načina importovanja i eksprotovanja trojki, kroz **R**esource **D**escription **F**ramework.

Network

Network ili mrežne baze podataka imaju generilizovaniju implementaciju od RDF baza podataka. Njihova struktura podseća na mrežu ili stablo, gde svaki čvor u mreži može imati više roditelja i više potomaka. Network model je nešto kompleksnija varijacija hijerarhijskog modela, čija je struktura stablo u kome je svaki čvor imao samo jednog roditelja.

4. Neo4j baza podataka

Neo4j predstavlja nativnu graf bazu podataka, što znači da su podaci skladišteni i prikazani u vidu usmerenog grafa. Ovakav način smeštanja podataka (koji će detaljno biti objašnjen u nastavku rada), ima veliku prednost u odnosu na druge vrste skladištenja kada su u pitanju visoko povezani podaci. Naime, relacione baze podataka, ali i nerelacione koje ne koriste strukturu grafa, imaju veoma loše performance prilikom pretraživanja veza između dva entiteta. Za ovo pretraživanje najčešće koriste posebnu strukturu koja se naziva indeks, i za svaku vezu između dva entiteta indeks se mora obići iznova. Ova operacija je veoma skupa i spora. Za razliku od njih, veze su sastavni deo entiteta u bazi Neo4j, zbog čega su operacije pronalaska veza veoma brze i jeftine.

Neke od najčešćih primera ove baze su detekcija prevare *(fraud detection),* upravljanje lancem nabavke, pronalaženje najkraćeg puta i tako dalje.

Upravo zbog svojih karakteristika, ova baza je doživela porast popularnosti u poslednjoj deceniji, i zadobila poverenje velikih klijenata poput eBay-a, airBnb-a, Walmarta, Linkedin-a i drugih.

4.1. Osnovna svojstva

- ACID *(**A**tomicity, **C**onsistency, **I**solation, **D**urability)* – Ova baza garantuje konzistentnost podataka za sve operacije izvršene unutar transakcija.

- Podržava indeksiranje korišćenjem Apache Lucene indeksa
- Podržava UNIQUE ograničenje – čvor može biti jedinstven u sistemu na osnovu zadatog atributa (recimo ID).
- Podržava REST API koji se može koristiti iz mnogih programskih jezika
- Cypher Query jezik – Deklarativni jezik koji služi za pronalaženje veza između podataka na jednostavan način.

4.2. Osnovne komponente strukture baze Neo4j

Čvor

Čvorovi predstavljaju entitete u grafu, pri čemu svaki čvor može sadržati neograničen broj atributa, koji su memorisani u JSON formatu, kao parovi ključ-vrednost. Svaki čvor može imati svoju labelu, čime se omogućava da svaki čvor ima različitu ulogu u sistemu.

Labela

Labela je opcioni parametar koji služi da podeli graf u manje skupove. Ovo čini pretragu efikasnijom, jer je moguće pretraživati čvorove koji pripadaju samo određenom podskupu grafa, umesto pretraživanja čitavog grafa. Jedan čvor može ili ne mora imati više različitih labela.

Veza (poteg)

Poteg ili veza predstavlja direktnu i (opciono) usmerenu vezu između dva čvora (ili jednog čvora, ako je taj čvor istovremeno i početni i krajnji). Svaku vezu karakteriše *usmerenje, tip, početni i krajnji čvor*. Veza može imati atribute kao i čvor, pri čemu je atribut najčešće težina, cena, vremenski interval itd. Dva čvora mogu biti povezana neograničenim brojem veza bez uticaja na performanse, jer će se prilikom obilaska veze neodgovarajućih tipova jednostavno ignorisati. Svaka veza može imati najviše jedan tip. Važno je napomenuti da se bez obzira na usmerenje potega čvorovi mogu obilaziti i u suprotnom smeru ako za to postoji potreba. Na usmerenje potega treba obraćati pažnju samo ako je od interesa za analizu podataka, što znači da nema potrebe praviti poteg u suprotnom smeru samo da bi povezanost bila u oba smera, sem ako on ne predstavlja drugačiju vezu od one koja već postoji.

Bitno je navesti da je nemoguće imati poteg koji ne vodi nigde. Ovo znači da se prilikom brisanja čvora iz baze, brišu i svi potezi koji vode ka njemu.

Atributi (svojstva)

Kao što je već rečeno, atributi su svojstvo i čvorova i potega, i može ih biti neograničen broj. Upisuju se u Json formatu kao parovi ključ-vrednost, pri čemu vrednost može biti atomična, ili niz atomičnih vrednosti. Podržani tipovi svojstava u bazi Neo4j su:

- Boolean
- Byte
- Short
- Integer
- Long
- Float
- Double
- Char
- String

4.3. Identifikovanje komponenata prilikom modelovanja sistema

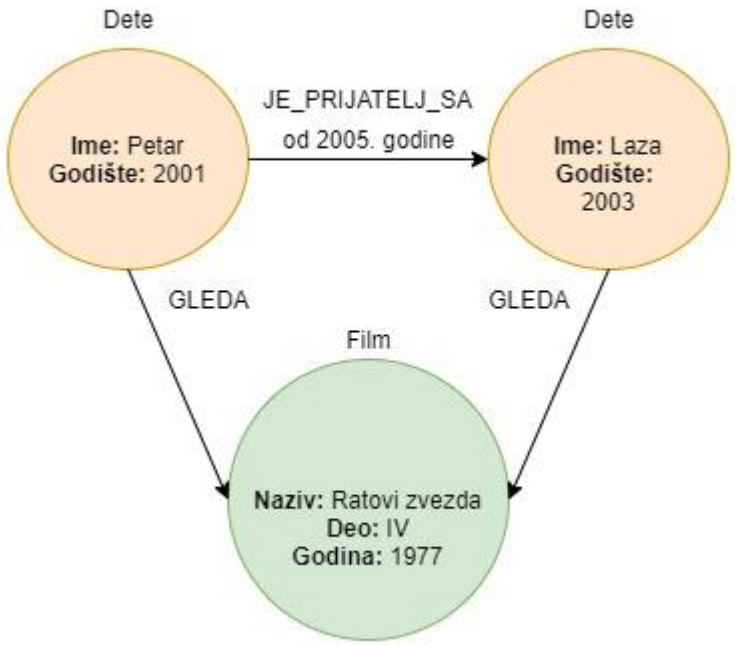
Kako Neo4j baza podataka ima prilično intuitivan pristup skladištenju podataka, koji se često naziva i *whiteboard friendly*[1], nije teško prebaciti potrebe korisnika iz govornog jezika u šemu baze. Tako se osnovne komponente mogu izdvojiti prostom analizom rečenice, pa recimo, iz rečenice *Petar i Laza gledaju Ratove zvezda*, možemo jednostavnim pravilima modelirati osnovnu strukturu baze podataka:

- *Petar, Laza,* i *Ratovi zvezda* su imenice, koje predstavljaju čvorove u sistemu.
- Za nalaženje odgovarajućih labela, proširujemo rečenice odgovarajućim atributima: *Deca* Petar i Laza gledaju *film* Ratovi zvezda. Ovim bližim opisom smo dobili upravo kategorije kojima čvorovi pripadaju, to jest labele.
- Petar i Laza *gledaju* film. Sada je fokus na akciji koju rade, koja predstavlja poteg između čvorova. Ovde se može izdvojiti i implicitna akcija, Petar *je prijatelj sa* Lazom.

[1] Whiteboard friendly – Termin koji ukazuje na to da struktura podataka izgleda upravo onako kako bi je korisnik nacrtao na beloj tabli prilikom razrade same ideje. Nije potrebno mnogo izmena kako bi struktura sa table prešla u model sistema.

- Na kraju, ostaje pronaći atribute. Oni se mogu izvući proučavanjem slučajeva korišćenja, to jest analize pitanja na koje bi baza trebalo da odgovori. Neka od pitanja bi mogla biti *Ko su glumci u filmu koji gledaju Petar i Laza, Kada je film snimljen, Koliko godina ima Petar,* i tako dalje. Sva pitanja od značaja dobijaju svoj odgovor u vidu parova ključ-vrednost koji predstavljaju atribute čvorova.

Nakon analize ovog primera, dobija se sledeća struktura (slika 1):



Slika 1 - Primer pretvaranja govornog jezika u graf

Na ovaj način veoma jednostavno može se doći do modela baze podataka. Kako se povećavaju zahtevi, tako se i čvorovi dodaju u sistem zajedno sa svojim potezima, što čini ovu bazu veoma fleksibilnom i skalabilnom.

4.4. Skladištenje podataka

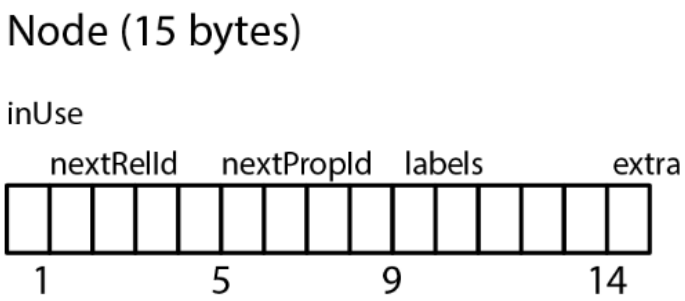
U Neo4j bazi, različiti delovi grafa se skladište u različitim fajlovima, što znači da postoje posebni fajlovi za skladištenje čvorova, potega, labela i atributa. Svi fajlovi predstavljaju *record store,* i radi lakšeg snalaženja i obilaska, svaki record (ulaz, jedinica) u fajlu je iste veličine. Pregled fajlova i velčina njihovih jedinica data je na slici 2. U nastavku će biti objašnjena svaka struktura ponaosob.

Store File	Record size	Contents
neostore.nodestore.db	15 B	Nodes
neostore.relationshipstore.db	34 B	Relationships
neostore.propertystore.db	41 B	Properties for nodes and relationships
neostore.propertystore.db.strings	128 B	Values of string properties
neostore.propertystore.db.arrays	128 B	Values of array properties

Slika 2 - Veličina fajlova u kojima su smeštene komponente grafa

Čvorovi

Svaki čvor koji se napravi u bazi skladišti se u istom fajlu: `neostore.nodestore.db`. Čvor se pamti kao jedinica u fajlu, i dužina svake jedinice za pamćenje čvora iznosi 15 bajtova. Struktura jedinice za pamćenje čvora data je na slici 3.



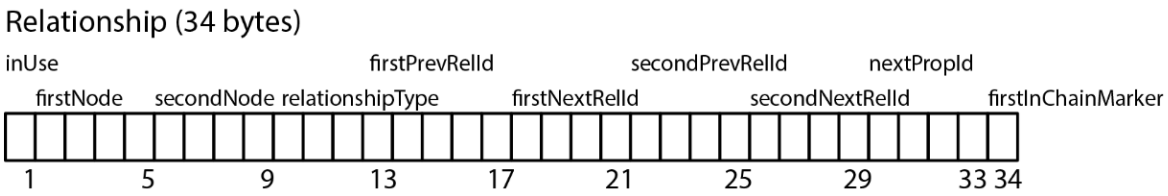
Slika 3 - Struktura jedinice čvora

Pozicije bajtova imaju sledeće značenje:

- *In use* – Prvi bajt se koristi kao fleg koji označava da li je jedinica u upotrebi. Ako nije, može se iskoristiti za smeštanje novog čvora.
- *Next Relationship Id* – Naredna četiri bajta predstavljaju Id prvog potega iz datog čvora. Id je mesto čvora u record storu.
- *Next Property Id* – Sledeća četiri bajta su rezervisana za Id prvog atributa (svojstva) čvora.
- *Labels* – Pet bajtova označavaju labele kojima čvor pripada.
- *Extras* – Preostali bajt se koristi za flegove.

Potezi

Potezi se nalaze u nešto komplikovanijoj (i samim tim i većoj) strukturi, čija jedinica zauzima 34 bajta. Slično čvorovima, ove jedinice se nalaze u fajlu `neostore.relationshipstore.db` i njihova veličina je takođe fiksna (slika 4).



Slika 4 - Struktura jedinice veze

Na slici se mogu uočiti sledeća polja:

- *In Use* – Kao i kod čvorova, prvi bajt je rezervisan za fleg koji ukazuje da li je jedinica u upotrebi.
- *First Node, Second Node* – Narednih osam bajtova čuvaju pokazivače na prvi i drugi čvor veze (po četiri bajta za svaki čvor)
- *Relationship Type* – Tip veze se skladišti u naredna četiri bajta. Tip veze se čuva u posebnom fajlu.
- *First Previous Relationship Id, First Next Relationship Id, Second Previous Relationship Id, Second Next Relationship Id* – Po četiri bajta za svaku od ovih stavki predstavljaju pokazivače na prvu prethodnu i sledeću vezu prvog i drugog čvora, respektivno.
- *First In Chain Marker* – Poslednji bajt je fleg koji ukazuje na to da li je posmatrana jedinica prva u lancu.

Atributi

Jedinice predviđene za čuvanje atributa nalaze se u fajlu `neostore.propertystore.db`, i svaka jedinica zauzima 41 bajt, od kojih je 32 bajta odvojeno za blokove (4 bloka po 8 bajtova). Svaka jedinica može čuvati informaciju o ključu, vrednosti, ili i o ključu i o vrednosti. Svako svojstvo, zavisno od tipa, može zauzeti između jednog i četiri bloka, što znači da svaka jedinica može skladištiti maksimalno četiri svojstva. U jedinici je takođe sačuvana informacija o tipu svojstva i ključu (3.5 bajta – 4 bita za ključ i 24 bita za tip), i pokazivač na indeks fajl svojstva. U indeks fajlu (`neostore.propertystore.db.index`) nalaze se imena svojstava. Za svaku vrednost koju svojstvo može da ima, jedinica skladišti ili samu vrednost, ili pokazivač na vrednost koja se nalazi u nekoj drugoj strukturi. Ovo zavisi od dužine vrednosti. Vrednosti primitivnih tipova upisuju se direktno, dok se vrednosti u vidu dužih stringova i nizova

upisuju kao pokazivači na strukture (`neostore.propertystore.db.strings`) i (`neostore.propertystore.db.arrays`). Kraći stringovi i nizovi mogu biti upisani u blokove direktno, ako njihova veličina to dozvoljava. Kako string i niz nemaju ograničenu veličinu, oni mogu zauzeti više jedinica kako bi ceo podatak mogao biti smešten, a da se pritom ne ugrozi pravilo o fiksnoj veličini jedinice.

Na osnovu prikazanih podataka, uočava se da je korišćena struktura zapravo lančana lista: jednostruka u slučaju čvorova i dvostruka u slučaju potega. Uzevši u obzir da su sve jedinice u fajlu iste veličine, obilazak grafa se svodi na praćenje pokazivača u fajlovima.

Da bi se obišla jedna veza između čvorova, dovoljno je ispratiti sledeće korake:

- U jedinici čvora naći pokazivač na vezu i pomnožiti ga sa fiksnom veličinom jedinice veze (34 bajta) da bismo znali gde se tačno tražena jedinica nalazi.
- U jedinici veze, naći drugi čvor, i pomnožizi ga fiksnom veličinom jedinice čvora (15 bajtova), što nas dovodi upravo do tražene jedinice drugog čvora.
- Ako želimo da u pretragu uključimo labelu ili tip veze, to ćemo uraditi na identičan način, množenjem pokazivača tipa ili labele sa veličinom jedinice koja se koristi za smeštanje ovih podataka.
- Sada se drugi čvor može pretražiti po nekom svojstvu u fajlu gde su smeštena svojstva, na već opisani način, množeći pokazivač veličinom jedinice u kojoj je memorisano svojstvo.

4.5. Memorija

Kao što smo već pokazali, nakon što su komponente grafa podeljene u jedinice i smeštene u fajlove, pretraživanje postaje veoma jednostavno i brzo. Impelementacija Neo4j nalaže da se svi generisani fajlovi smeštaju u glavnu memoriju. Međutim, problem koji može nastati je veličina grafa: Ako graf postane preveliki, nemoguće je držati čitav njegov sadržaj u glavnoj memoriji. Iz ovog razloga, Neo4j uvodi keširanje podataka.

Data cache (LRU-K stranični keš)

Uloga *data* keša je da podeli svaki fajl u regione i da čuva fiksni broj regiona po fajlu. Svaki fajl koristi sopstveni fajl bafer, koji deli fajl u regione iste veličine. Keš čuva najaktivnije regione u memoriji, i prati odnos keš pogodataka i promašaja. Kada broj pristupa fajlu koji

nije u kešu (broj promašaja) prestigne broj pristupa fajlu koji jeste u kešu, oni menjaju mesta. Logika za čuvanje stranica u kešu je LRU *(least frequently used),* što znači da će se iz keša izbaciti ona stranica koja je korišćena najdavnije. Parametar K označava K-ti pristup memoriji, na osnovu koga se izbacuju stranice. Na primer, ako je K=1, u obzir će se uzimati samo poslednji pristup stranici, dok ako je K=2, opstanak stranice u glavnoj memoriji će zavisiti od pretposlednjeg pristupa. Dakle, stranice se izbacuju iz glavne memorije na osnovu vremena proteklog od K-tog pristupa.

Na keširanje se može uticati prilikom konfiguracije Neo4j baze. Neki od parametara koji se mogu menjati su:

- *Use memory mapped buffers* – Ovo podešavanje ukazuje na to da li će se koristiti mapiranje memorije svojstveno operativnom sistemu, ili će se upotrebiti implementacija same Neo4j baze. Ako se koristi Neo4j implementacija bafera (vrednost *false)*, oni će biti smešteni na heapu, pa treba voditi računa o veličini heapa i povećati je shodno kapacitetu bafera. Ako je vrednost *true,* veličina heapa ne sme prevazilaziti ukupnu memoriju računara, od koje se oduzima memorija predviđena za bafere. Podrazumevana vrednost ovog parametra je *true.*
- *Mapped memory* – maksimum memorije koji je moguće iskoristiti za keš. Ovo podešavanje se može primeniti na sve fajlove, pa tako imamo podešavanja `neostore.nodestore.db.mapped_memory`, `neostore.relationshipstore.db.mapped_memory` itd.

Ako parametri nisu konfigurisani prilikom pokretanja, Neo4j će automatski konfigurisati sve parametre koji nisu specificirani.

Object cache

Objektni keš se često naziva i keš visokog nivoa, i on je predviđen za skladištenje podataka u formi koja je optimizovana za brz obilazak grafa. Keširaju se pojedinačni čvorovi, veze i atributi, u formi koja je bliska objektno orjentisanom API-ju i obilasku grafa. Čitanje iz ovog keša je 5 do 10 puta brže od čitanja iz *data* keša. Za razliku od data keša, ovaj keš je smešten direktno na heap-u, i njegova veličina direktno zavisi od trenutno dostupne memorije na heap-u.

Čim se pristupi čvoru ili vezi, oni se dodaju u objektni keš, ali se dodaju na *lazy* način, što znači da će se dodati samo čvor ili veza, bez svojih atributa, dok god ti atributi ne treba da budu pročitani. Zatim će se dodati samo atribut koji se traži. Isto važi i za veze: u kešu će biti samo ona veza kojoj se trenutno pristupa.

Izbacivanje funkcioniše isto kao u *data* kešu, primenom LRU algoritma.

Objektni keš se može konfigurisati, i podržava četiri načina keširanja:

- *None* – Keširanje visokog nivoa je onemogućeno.
- *Soft* – Podrazumevani način rada. Poželjan za manje grafove i za obilaske sa visokim performansama. Nije dobro rešenje ako se često pristupa delovima grafa koji se ne nalaze u kešu.
- *Weak* – Objekti u kešu ostaju veoma kratko. Pogodan za grafove u kojima je potrebno obići više čvorova nego što može stati u keš.
- *Strong* – Ova vrsta keša omogućava da se keširaju svi podaci iz celog grafa. Deo memorije heap-a koji pripada ovoj vrsti keša nikada neće biti oslobođen. Pogodna je za grafove koji su dovoljno mali i mogu stati u memoriju u celosti.

5. Dodatak

Nakon opisane strukture i skladišta podataka same baze, valjalo bi pomenuti još neke koncepte koji su karakteristični za nju. Iz tog razloga, i bez prevelikog ulaženja u detalje, u nastavku će biti reči o jeziku koji se koristi za formiranje upita u bazi Neo4j, kao i o indeksiranju koje se može iskoristiti za poboljšanje performansi prilikom pretrage.

5.1. Cypher

Glavna ideja prilikom kreiranja Cyphera je bila napraviti jednostavan jezik koji će biti lak za učenje jer sintaksno podseca na Sql sa kojim se većina korisnika već susrela, uz dodatak onog što je karakteristično za samu bazu: Praćenje veza i obrazaca. Sintaksa je vrlo slična vizuelnoj reprezentaciji grafa:

```
//data stored with this direction
CREATE  (p:Person)-[:LIKES]->(t:Technology)

//query relationship backwards will not return results
MATCH  (p:Person)<-[:LIKES]-(t:Technology)

//better to query with undirected relationship unless sure of direction
MATCH  (p:Person)-[:LIKES]-(t:Technology)
```

Na slici se može uočiti sledeće: Čvor P pripada labeli Person. U ovom slučaju, ime varijable (P) je ono koje je dodeljeno čvoru, i po tom imenu će se kasnije i vršiti pretraga. Ime varijable je opcioni parametar, ali ako se ne navede, čvor se nikada neće naći među rezultatima pretrage. Kao što je već rečeno, sa desne strane imena varijable se nalazi labela kojoj čvor pripada. Dva čvora su povezana vezom koja je prikazana strelicom ako je usmerena, ili samo crtama ukoliko nije usmerena. Usmerenost ukazuje na to da se ta veza može pretražiti samo u navedenom smeru. Ako se pretraži u suprotnom smeru, neće se dobiti rezultati. Naziv (tip) veze upisan je unutar uglastih zagrada.

Čvorovima se mogu dodati i atributi. Oni se upisuju kao parovi ključ vrednost, unutar vitičastih zagrada:

```
(p:Person {name: "Jennifer"})-[rel:LIKES]->(g:Technology {type: "Graphs"})
```

Ključne reči

- CRUD operacije

Za CRUD operacije, koriste se ključne reči CREATE, RETURN, SET i DELETE, u kombinaciji sa ključnom reči MATCH koja će biti detaljnije objašnjena u nastavku.

- MATCH

Ova reč se koristi prilikom pretrage, i može se primeniti na čvor, vezu, labelu, atribut ili obrazac. Rezulati pretrage mogu se kombinovati sa CRUD operacijama, to jest, rezultat se može iskoristiti za kreiranje novih entiteta, ili za izmenu ili brisanje postojećih:

```
MATCH (jennifer:Person {name: 'Jennifer'})
MATCH (mark:Person {name: 'Mark'})
CREATE (jennifer)-[rel:IS_FRIENDS_WITH]->(mark)
```

```
MATCH (p:Person {name: 'Jennifer'})
SET p.birthdate = date('1980-01-01')
RETURN p
```

```
MATCH (j:Person {name: 'Jennifer'})-[r:IS_FRIENDS_WITH]->(m:Person {name:
'Mark'})
DELETE r
```

- RETURN

Reč return se koristi za specificiranje onoga što želimo da dobijemo kao rezultat prilikom pretrage. Kao rezultat se može dobiti čvor, veza, atribut ili obrazac.

```
MATCH (:Person {name: 'Jennifer'})-[:WORKS_FOR]->(company:Company)
RETURN company.name
```

Prilikom definisanja RETURN ključne reči, mogu se promeniti imena atributa koje želimo da vratimo kao rezultat, u cilju dobijanja strukture koja je lakša za čitanje:

```
//poorly-named property
MATCH (kristen:Customer {name:'Kristen'})-[rel:PURCHASED]-(order:Order)
RETURN order.orderId, order.orderDate, kristen.customerIdNo,
order.orderTotalNoOfItems

//cleaner printed results with aliasing
MATCH (kristen:Customer {name:'Kristen'})-[rel:PURCHASED]-(order:Order)
RETURN order.orderId AS OrderID, order.orderDate AS `Purchase Date`,
       kristen.customerIdNo AS CustomerID, order.orderNumOfLineItems AS
`Number Of Items`
```

5.2. Indeksiranje

Iako zbog same implementacije nije neophodno, Neo4j podržava i korišćenje indeksa u cilju poboljšanja performansi. Postoje dve vrste indeksa: nativni indeksi za poboljšanje performansi pretrage, i *full text search* indeksi.

Indeksi za poboljšanje performansi pretrage (B-tree)

B-tree indeksi su dobili ime po strukturi koja se koristi za njihovo čuvanje – balansirano stablo (balanced tree). U ovakav indeks smešta se jedan ili više atributa iz čvorova koji pripadaju istoj labeli. Na osnovu broja atributa koji učestvuju u indeksu, možemo razlikovati sledeće indekse:

- *Single property* – ovakva vrsta indeksa obuhvata samo jedan atribut pod određenom labelom, i kreira se na sledeći način:

```
CREATE INDEX ON :Book(title)
```

- *Composite index* - obuhvata više atributa pod nekom labelom. Kreira se slično kao single property index:

```
CREATE INDEX ON :Book(price, publisher)
```

Kada se indeks jednom kreira, ne mora se eksplicitno tražiti njegovo obilaženje. Cypher je svestan postojanja indeksa, i ako se ukaže upit čiji je fokus na atributima koji u indeksu postoje, rezultati će automatski biti formirani pretraživanjem indeksa. Takođe, kreiranje novog čvora će automatski rezultirati i upisom istog u indeksnu strukturu, ako njegovi atributi odgovaraju strukturi indeksa:

```
CREATE (a:Book {title: 'Harry Potter', price: 299, publisher: 'Readers'}), (b:Book {title: 'Narnia', price: 499}).
```

U gorenavedenom primeru, možemo videti kreiranje dva čvora: knjige koja sadrži naziv, cenu i izdavača, i knjige koju karakterišu samo naziv i cena. Kako je kompozitni indeks naveden u prethodnom primeru baziran na atributima *cena* i *izdavač,* samo prva knjiga će biti dodata u indeksnu strukturu.

Full text search indeksi

Ova vrsta indeksa se bazira na korišćenju Apache Lucene biblioteke za indeksiranje, i njena glavna karakteristika je indeksiranje čvorova i veza po atributima čiji je tip string, što znači da se čvorovi mogu pretraživati po sadržaju samog stringa. Prethodno navedeni tip indeksa će kao rezultate izdvojiti samo one čvorove koji imaju savršeno poklapanje sa datim kriterijumom, dok će full text indeks će posmatrati string razbijen na tokene i tako ga pretraživati, što omogućava nalaženje čvorova i veza koji delimično odgovaraju pretrazi. Stringovi se tokenizuju u reči različito, zavisno od analizatora koji se koristi i prirode samog jezika. Glavne prednosti full text indeksa u odnosu na b-tree indekse su:

- Primenjivanje na više labela odjednom
- Primenjivanje na više tipova veza
- Primenjivanje na više atributa odjednom, sa jednom bitnom razlikom u odnosu na kompozitni indeks: sada se atributi čvora ne moraju poklapati sa svim traženim atributima da bi se on našao među rezultatima.

Full text index se kreira na sledeći način:

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

Prilikom kreiranja indeksa, mogu se navesti i opcioni parametri koji se tiču konfiguracije same strukture. Jedan od parametara se odnosi na analizator koji će se koristiti prilikom pretrage indeksa, a drugi na pravilo konzistencije koje će se primenjivati:

```
CALL
db.index.fulltext.createRelationshipIndex("taggedByRelationshipIndex",["TAGGED_AS"],["taggedByUs
er"], { analyzer: "url_or_email", eventually_consistent: "true" })
```

Parametar eventually-consistent nam govori o tome kada će se odvijati ažuriranje indeksa: ako je postavljen na *false*, ažuriranje će se raditi odmah, u toku transakcije, dok vrednost *true* označava da će indeks biti ažuriran što je pre moguće, na pozadinskoj niti. Eventualna konzistencija može poboljšati performanse jer nema čekanja da se transakcija izvrši, ali takođe može dovesti do toga da korisnik dobije neažurnu vrednost.

Kada je ovakav indeks kreiran, u njegovu strukturu će biti dodati svi čvorovi koji bar delimično odgovaraju traženim atributima, pa će tako u gorenavedenom indeksu biti

zapamćeni i čvorovi koji imaju samo naslov ili samo opis. Podsetnika radi, u b-tree indeksu se pamti samo čvor koji zadovoljava sve atribute.

Za razliku od b-tree indeksa koji se pretražuju automatski, full text search indeksi se moraju pretražiti eksplicitno:

```
CALL db.index.fulltext.queryNodes("titlesAndDescriptions", "Full Metal Jacket") YIELD node,
score

RETURN node.title, score
```

Kao rezultat pretrage ovog indeksa može se dobiti atribut *score*, koji nam govori o stepenu poklapanja dobijenog rezultata sa traženim parametrima. Rezultati su sortirani u opadajućem redosledu, pa će prvi dobijeni rezultat biti onaj koji ima najveći *score,* to jest najveći stepen poklapanja sa onim što smo tražili (slika 5):

node.title	score
"Full Metal Jacket"	1.4111186265945435
"Full Moon High"	0.44524085521698
"The Jacket"	0.3509606122970581
"Yellow Jacket"	0.3509606122970581
4 rows	

Slika 5 - Rezultati pretrage indeksa

Ako pak želimo da izdvojimo jedinstven rezultat, samo onaj koji se tačno poklapa sa traženim parametrima, potrebno je parametre staviti pod dodatne navodnike:

```
CALL db.index.fulltext.queryNodes("titlesAndDescriptions", "\"Full Metal Jacket\"") YIELD node,
score

RETURN node.title, score
```

U tom slučaju, dobijeni rezultat će biti jedinstven, i predstavljaće čvor koji ima tačno poklapanje sa navedenim parametrima, ako takav čvor postoji (slika 6):

node.title	score
"Full Metal Jacket"	1.4111186265945435
1 row	

Slika 6 - Jedinstven rezultat pretrage

6. Zaključak

Iako je struktura baze Neo4j na prvi pogled delovala veoma intuitivno i neskriveno, dubljom analizom studentkinja je zaključila da se iza jednostavne zamisli krije kompleksna i dobro osmišljena struktura. Počev od dubljeg proučavanja same strukture baze o kojoj je mnogo toga studentkinji već bilo poznato, došli smo do skladištenja podataka čija je implementacija zaista nije vidljiva na prvi pogled. Novo saznanje o ovoj oblasti o kojoj studentkinja pre pisanja rada nije imala nikakav uvid, čini je, po njenom mišljenju, najzanimljivijim delom ovog rada. Kao i na početku, gde se rad osvrće na nerelacione i graf baze podataka uopšteno, pre nego što uvede čitaoca u konkretnu implementaciju, i kraju rada se može uočiti šira slika: komponente baze koje je studentkinja smatrala bitnim, a nisu se ticale konkretne teme. Cilj ovakvog pristupa je bilo povezivanje svih komponenata baze u jednu opštu i široku celinu, koja bi čitaocu dala dovoljan uvid u osnovne koncepte, na osnovu kojih bi mogao da se opredeli za korišćenje baze Neo4j.

Literatura

- Zvanična dokumentacija baze Neo4J
- Ian Robinson, Jim Webber, Emil Eifrem: *Graph Databases,* drugo izdanje