

Univerzitet u Nišu, Elektronski fakultet

Katedra za računarstvo

Niš, 2019

Danilo Vulović

Seminarski rad

Interna struktura Neo4j baze podataka

**Mentor**: **Student**:

Prof. dr Aleksandar Stanimirović Danilo Vulović 1065

#### Sadržaj

[1. Uvod 4](#_Toc69322341)

[2. SQL i NoSQL baze podataka 6](#_Toc69322342)

[2.1. Osnove SQL-a 6](#_Toc69322343)

[2.2. Osnove NoSQL-a 6](#_Toc69322344)

[2.3. Glavne razlike, prednosti i mane 6](#_Toc69322345)

[2.3.1. Struktura 7](#_Toc69322346)

[2.3.2. Jezik za upite 8](#_Toc69322347)

[2.3.3. Skalabilnost 8](#_Toc69322348)

[2.3.4. Podrška 9](#_Toc69322349)

[3. Osnove graf i Neo4j baze podataka 11](#_Toc69322350)

[3.1. Baze podataka bazirane na grafovima 11](#_Toc69322351)

[3.2. Neo4j 12](#_Toc69322352)

[3.2.1. Glavne odlike Neo4j baze 13](#_Toc69322353)

[3.2.2. Neo4j upotreba 13](#_Toc69322354)

[4. Interna struktura Neo4j-a 15](#_Toc69322355)

[4.1. Native graf procesiranje 15](#_Toc69322356)

[4.2. Native graf skladište 16](#_Toc69322357)

[4.2.1. Lančane liste 16](#_Toc69322358)

[4.2.2. Struktura čvorova i relacija 17](#_Toc69322359)

[4.2.3. Neo4j API 19](#_Toc69322360)

[5. Zaključak 22](#_Toc69322361)

[6. Literatura 23](#_Toc69322362)

# Uvod

Baza podataka predstavlja skup, na neki logičan način, povezanih podataka. Omogućava korisniku da efikasno pribavi, uneti nove ili obriše podatke. Podaci su organizovani u tabele, poglede, šeme itd. Na primer, baza podataka Elektronskog fakulteta u Nišu void računa o studentima, profesorima i asistentima, predmetima, nastavnim programima, rokovima itd.

Možda se “na prvu loptu” ne vidi prednost ovakvih sistema ali pogledajmo sledeći primer. Zamislite da koristite standardan fajl sistem za gore pomenutu fakultetsku bazu. Neki od problema na koje nailazimo su:

* Redundantnost podataka - za reduntantne podatke kažemo da su takvi ukoliko su kopirani na više mesta
  + podaci o istom studentu će se nalaziti u više sekcija
  + ažuriranje broja telefona studenta zahteva da broj bude ažuriran u svim sekcijama
  + brisanje studenta iz sekcije sa studentima će ostaviti tragove tog studenta u ostalim sekcijama koji takođe moraju biti obrisani
* Nekonzistentnost podataka - za podatke kažemo da su nekonzistentni ukoliko nekoliko kopija istog podatka nisu iste
  + ako broj telefona studenta u studentskoj službi i u administraciji nije isti, dolazi do nekonzistentnosti podataka
  + do nekonzistentnosti se može doći raznim programskim greškama kao i ako sve kopije podatka nisu ažurirane
* Otežan pristup podacima – pristup podacima o određenom studentu može biti izuzetno komplikovan.
  + zamislite da imate 10.000 studenata i trebate naći podatke o nekolicini koji uspunjavaju određene uslove.
* Neovlašćen pristup – korisnik pristupa podacima koji nisu njemu namenjeni
  + Student bi mogao da pristupi sekciji sa ocenama i promeniti ih
* Nemogućnost oporavka od greške ili kreiranja rezervnih kopija
* ...

Ovo su samo neki od problema koji su naterali da se obavi skok sa fajl sistema na nešto ozbiljnije.

Sistemi za upravljanje bazama podataka, kao i sama baza, predstavljaju srce svakog kompjuterskog programa. Oni predstavljaju skup programa koji zajedno rukovode bazom podataka i omogućavaju krajnjim korisnicima da izvršavaju operaciji nad bazom. Korisniku je omogućeno da na jednostavan način definiše kako izgledaju podaci koje baza podataka skladišti, kako se ti podaci memorišu, pružaju korisniku jednostavan način za manipulisanje podacima u bazi, štite podatke od bilo kakve vrste sistemske greške ili krađe i omogućavaju različite privilegije različitim korisnicima.

Skoro svaki sistem za upravljanje bazama podataka je (u jednom ili drugom obliku) sastavljen iz sledeće tri komponente:

* Procesor upita
* Sistem za upravljanje skladištem podataka
* Skladište na disku

Oni rade u koheziji da obezbede visoku funkcionalnost čitavog sistema, da omoguće izvesnu vrstu fleksibilnosti ukoliko je neki koncept baze podataka potrebno izmeniti kao i visoke performanse uz visok procenat bezotkaznog rada a lakoća korišćenja podrazumeva da je korisniku omogućena automatizacija raznih zadataka.

Sve ovo su samo neki od zahteva koji se očekuju od jednog sistem za upravljanje bazama podataka. U ostatku rada, bavićemo se razlikom SQL i NoSQL baza podataka (drugo poglavlje), analiziraćemo kako izgleda Neo4j baza podataka i koje su njene glavne prednosti i mane (treće poglavlje), kako izgleda Neo4j interna struktura i zašto se određene stvari implementirane na takav način (četvrto poglavlje) kao i da odgovorimo na pitanje zašto je bitno znati internu strukturu određene baze podataka (peto poglavlje). Poslednje poglavnje sadrži literaturu koju sam upotrebio za pisanje ovog rada (šesto poglavlje).

# SQL i NoSQL baze podataka

## Osnove SQL-a

Kada kažemo SQL baza podataka, obično mislim na relacionu bazu podataka. Podaci su logički organizovani u tabele. SQL (*Structured query language*) služi da omogući korisniku jednostavnu manipulaciju podacima u bazi podataka. To je veoma moćan alat i koncepti relacionih baza podataka su detaljno istraženi i njihova efektivnost je dokazana. Pisanje kompleksnih upita je izuzetno jednostavno i lako možete naći eksperta za rad sa bazama podataka. Ono što SQL zahteva je da postoji predefinisana (statička) šema podataka i svi podaci u bazi moraju da prate tu šemu. Ovo može značajno da poveća vreme planiranja strukture baze jer bilo kakve naknadne promene mogu biti teške ili ometati ostatak sistema. Neke od najpopularnijih baza su:

* MySQL
* Microsoft SQL server
* SQLite
* Oracle
* ...

## Osnove NoSQL-a

NoSQL baze podataka, kao što im ime kaže, ne koriste SQL[[1]](#footnote-1), odnosno, šema im je dinamička i moguće je pamtiti nestrukturne podatke. Podaci koji se pamte mogu da budu dokument orijentisani, orijentisani kolonama, bazirani na grafovima ili da budu predstavljeni KeyValue skladištem. Svaki dokument može biti struktuiran na sebi svojstven način i tu strukturu je moguće promeniti u bilo kom trenutku. Neke od najpoznatijih NoSQL baza podataka su:

* MongoDB, Elasticsearch, ... (dokument orijentisane)
* MariaDB, Cassandra, ... (kolonama orijentisane)
* Neo4j, Amazon Neptune, ... (graf baze)
* Dynamo, Apache Ignite, ... (KeyValue skladišta)
* ...

## Glavne razlike, prednosti i mane

Glavne razlike SQL i NoSQL baza podataka su date u sledećoj tabeli:

|  |  |  |
| --- | --- | --- |
| Ključne razlike | SQL baze podataka | NoSQL baze podataka |
| Tip baze podataka | Relaciona | Nerelaciona |
| Šema | Predefinisana (statička) | Dinamička |
| Kategorije baze podataka | Bazirane na tabelama | Dokument baze, KeyValue skladišta, graf baze... |
| Kompleksni upiti | Pogodna | Nezgodne |
| Skalabilnost | Vertikalno skalabilne | Horizontalno skalabilne |
| Svojstva | ACID[[2]](#footnote-2) svojstva | CAP[[3]](#footnote-3) teorema |
| Podrška | Odlična, od strane svih provajdera | Uglavnom se oslanja na podršku zajednice |

*Tabela 1 – Glavne razlike SQL i NoSQL baza podataka*

U nastavku ćemo pomenuti neke razlike da bi napravili jasnu granicu između SQL i NoSQL baza podataka.

### Struktura

Statička šema zahteva da struktura podataka koji se čuvaju bude unapred definisana i da svi podaci mogu da se „ukalupe“ u okvire definicije šeme. Dinamička šema, kao što joj ime kaže, je podložna promenama u bilo kom trenutku rada. Razmotrimo primer da je za evidentiranje vakcinisanja potreban način identifikacije svakog korisnika. Prvo što nam pada na pamet je da možemo za to iskoristiti jedinstveni matični broj građana. U SQL bazi bi ovo mogla biti kolona koja sadrži niz cifara JMBG-a.

Dobili smo zahtev da nije potreban JMBG građana da bi se vakcinisao već je moguće podneti zahtev vozačkom dozvolom ili zdravstvenom karticom. Ovo zahteva da se promeni SQL šema i da se sama identifikacija izmesti u zasebnu tabelu gde će biti nekoliko kolona za svaki od načina identifikacije gde će biti dovoljno da jedna od kolona bude popunjena za svaki red. NoSQL baza ne mora da primi nikakve promene, prihvatiće drugačiju identifikaciju kao i opcione dodatne podatke o tome.

Dobili smo zahtev da je moguće vakcinisati i strane državljane ali svaka država ima svoj način za jedinstvenu identifikaciju građana. Nije izgledno dodati zasebnu kolonu za svaku državu na svetu a ne bi bilo pametno oslanjati se na brojeve pasoša jer oni ne garantuju da ne postoji još jedan pasoš na svetu sa istim brojem. Morali bi osmisliti neku vrstu kompozitnog ključa broja pasoša i države iz koje strani državljanin. NoSQL neće imati problem da primi identifikaciju drugačijeg oblika.

Primer je možda suviše banalan ali dovoljno opisuje problem sa kojim se statička baza suočava i zašto su u prethodnoj deceniji NoSQL baze naglo dobile na popularnosti i zašto NoSQL tehnologije imaju predviđen rast od 56% po godini za razliku od SQL baza koje takođe očekuju rast ali od svega 12%.

### Jezik za upite

SQL baze koriste SQL za definisanje i manipulisanje podacima i jezik je sam po sebi, izuzetno moćan ali i restriktivan. Zahteva definisane strukture podataka pre početka rada i svi podaci moraju da slede definiciju što zahteva preciznost i pedantnost.

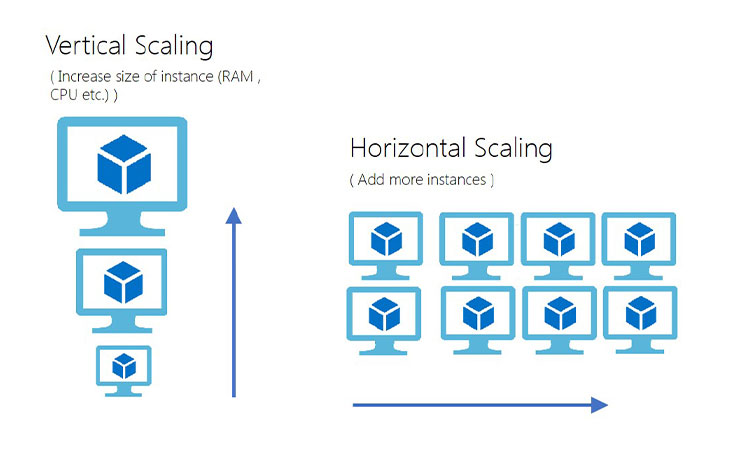
NoSQL baze su fleksibilnije i dozvoljavaju dodavanje polja u bilo kom trenutku i jedinstvenu strukturu za svaki podatak. Eliminisanje planiranja i definisanja strukture omogućava spremnost za rad u rekordnom roku ali baratanje i validacija podataka mora biti odrađena ručno. Mnoge NoSQL baze koriste jezike koji su nalik na SQL-u (npr. Neo4j koristi Cypher) kako bi programerima tranzicija bila što prirodnija i korišćenje jednostavno i prepoznatljivo. Takođe, neke NoSQL baze podržavaju SQL upite uz određenu cenu pretprocesiranja takvog upita.

### Skalabilnost

Kako količina podataka koje skladištimo raste, sasvim je prirodno razmišljati o skaliranju kako bi sistem podneo veći teret. U zavisnosti od toga da li koristimo SQL ili NoSQL bazu, tehnika skaliranja se fundamentalno razlikuje.

Vertikalno skaliranje podrazumeva poboljšanje performansi servera koje su postignute poboljšavanjem hardvera. Ovo uključuje veću RAM memoriju, brži CPU, korišćenje bržeg SSD diska itd. Iako se ovo čini dosta logično i razumno, postoji jedan „ogroman“ problem. Postoji limit za ovakva poboljšanja jer možete kupiti trenutno najbrži procesor na tržištu i tu se CPU poboljšanja završavaju. Takođe, kupovina najnovijeg hardvera je dosta skupa. Ovakvo skaliranje je karakteristično za SQL baze podataka.

Horizontalno skaliranje se postiže povećanjem broja servera. U teoriji, možemo dodati beskonačan broj servera koji paralelno rade. S obzirom da su instance servera distribuirane, možemo čuvati više podataka ali uvodimo probleme distribuiranog sistema. NoSQL baze podataka se prirodno skaliraju na ovaj način obično nude mehanizme za rešavanje ovih problema.



Slika 1 – Vertikalno i horizontalno skaliranje

Koncept *sharding-a* (podela na manje delove) je ključan za skaliranje i odnosi se i na SQL i na NoSQL baze. Kao što ime kaže, sharding deli bazu na više delova. Svaki deo ima jedinstven indeks kojim se pretražuju podaci i na osnovu koga možemo podeliti podatke na više delova (npr. možemo podeliti bazu na dva dela gde se u prvom delu nalaze ljudi čije prezime počinje sa nekim od slova od A do M, dok se u drugom delu nalaze ljudi čija prezimena počinju od N do Š). SQL baze ne podržavaju automatsko deljenje na delove i inženjeri moraju ručno da brinu o sharding-u. Vrlo često je izuzetno teško uraditi sharding jer promene strukture, koje su svakako vremenski skupe, zahtevaju i izmene shardova. Sa druge strane, NoSQL baze su inicijalno dizajnirane sa skaliranjem na umu, tako da je sam proces skaliranja, odnosno shardinga, skoro neprimetan za krajnjeg korisnika.

### Podrška

SQL ima veliku istoriju i mnogo razvijeniju zajednicu. Zasniva se na matematičkom modelu, postoje brojne najbolje prakse za razne slučajeve korišćenja i postoji veliki broj ljudi koji su eksperti. SQL jezik se sam po sebi ne razlikuje puno kod različitih baza podataka izuzev nekih finesa a provajderi baza podataka garantuju za pouzdanost same baze.

NoSQL baze su relativno mlade u poređenju sa SQL ekvivalentima pa stoga ne postoji jaka podrška provajdera već se programeri uglavnom oslanjaju na podršku zajednice. Kao i broj NoSQL korisnika, podrška rapidno raste i možemo očekivati da će u nekom trenutku biti standardizovane na zavidnijem nivou.

# Osnove graf i Neo4j baze podataka

## Baze podataka bazirane na grafovima

Najkraće moguće rečeno, graf baza je baza koja je dizajnirana tako da relacijama između čvorova pridaje jednaku važnost kao i samim čvorovima, odnosno podacima. Podaci mogu da se čuvaju bez pre-definisanog modela. U realnom svetu, pa i u onom sistemu koji modeliramo na osnovu realnog sveta, ne postoji informacija koja je sama po sebi izolovana i nema nikakvu relaciju ni sa čim. Čak suprotno, oko nas se nalaze domeni koji su vrlo bogato povezani. Samo baza koja ove veze prirodno prihvata, može jednostavno da ih obrađuje, procesira i pretražuje. SQL baze koriste vremenski skupocene JOIN operacije dok se u graf bazama, relacije između entiteta čuvaju odmah uz njih.

Pristup čvorovima i njihovim relacijama u graf bazi se može radi efikasno jer se relacijama pristupa za konstantno vreme i moguće je obići milione relacija u sekundi na svakom jezgru. Nezavisno od ukupne veličine skupa podataka sa kojim se radi, graf nadmašuje standardne SQL baze kada su podaci u skupu podataka visoko povezani.

A picture containing accessory

Description automatically generated

Slika 2 – Vizuelizacija Neo4j baze koja prikazuje nekoliko filmova u kojima je glumio Tom Henks kao i reditelje tih filmova

Postoje nekoliko različitih pristupa pri modeliranju ovakvih graf baza. U narednom delu teksta, skoncentrisaćemo se na model grafa koji nam govori da su podaci organizovani u čvorove, relacije između čvorova i iz svojstava čvorova i relacija.

**Čvorovi** su entiteti grafa. Oni mogu da čuvaju beskonačno mnogo atributa (obično Key-Value parovi) koje kratko zovemo, svojstva. Čvorovi mogu da budu tagovani labelama, koje mogu da posluže kao neka vrsta metapodatka.

**Relacije** su usmerene i semantički bitno imenovane putanje između čvorova (npr. Radnik *RADI\_U* Fabrika; *RADI\_U* je relacija). Relacija uvek ima smer, tip, početni čvor i krajnji čvor. Kao i čvorovi, i relacije mogu imati pridodata svojstva (npr. relacija iz prethodnog primera može da ima svojstvo { *RadiOd:* 2017 *},* što znači da radnik radi od 2017. godine u fabrici). U većini slučajeva, relacijama se pridodaju kvantitativna svojstva kao što su težine, cene, distance, vremenski intervali i sl. Zbog efikasnog načina na koji se relacije čuvaju, dva ista čvora mogu imati neograničen broj relacija između sebe bez ikakvog gubitka u performansama. Važno je pomenuti da su relacije usmerene ali da to ne znači da je nemoguće navigirati se kroz graf u bilo kom smeru (npr. graf će jednako brzo naći relaciju A -> B kao i B <-A.

Chart, bubble chart

Description automatically generated

Slika 3 – Primer čvorova i relacija u grafu

## Neo4j

Neo4j je NoSQL graf baza, otvorenog koda koja obezbeđuje aplikacijama „bekend“ sa transakcijama koje su u skladu sa ACID svojstvima. Baza je pisana u Javi i Scali i razvoj je počeo 2003. godine ali je od 2007. kod javno dostupan. Dolazi u dve edicije, *Community* verzija koja je besplatna, dostupna svima i *Enterprise* verziju koja nudi sve što nudi i *Community* uz razne dodatne pogodnosti poput čuvanja rezervnih kopija baze, oporavka od grešaka kao i podršku za klasterizaciju.

Za Neo4j se kaže da je „native“ graf baza jer efikasno implementira prethodno pomenuti graf model sa svojstvima sve do nivoa skladišta. Podaci su čuvani tačno onako kako se i koriste, a baza koristi pokazivače za navigaciju kroz graf. Za razliku od graf procesiranja ili biblioteka koje simuliraju graf baze kroz *in-memory* model, Neo4j obezbeđuje sve karakteristike normalne baze podataka, transakcije koje su saglasne sa ACID svojstvima, podršku za klasterizaciju i oporavak od grešaka što olakšava korišćenje graf modela čak i u proizvodnji.

### Glavne odlike Neo4j baze

Diagram

Description automatically generated

Slika 4 – Glavne odlike Neo4j baze

Neke od glavnih odlika Neo4j baze zbog koji je ova baza vrlo popularna među programerima, arhitektama i administratora baza podataka su:

* **Cypher**, deklarativni jezik za upite sličan SQL-u ali optimizovan za graf baze. Ovaj jezik je usvojen i od strane nekih drugih graf baza.
* **Konstantno vreme obilaska**, vreme obilaska grafa se ne povećava drastično sa povećanjem broja čvorova. Na modernism hardverima, broj čvoroma može biti i nekoliko milijardi (upit koji vraća ukupan broj čvorova u grafu koji iznosi preko 31 milion čvorova traje manje od 1ms).
* **Fleksibilnost**, šemu je moguće menjati i adaptirati vremenom, ili dodati nove relacije između određenih entiteta za koje se ranije nije znalo i koje mogu ubrzati pretragu
* **Drajveri**, za sve popularne jezike poput Jave, Pythona, Javascripta, .NET-a...

### Neo4j upotreba

Prva stvar koja deluje kao logičan scenario za korišćenje Neo4j baze su društvene mreže. Iako to jeste jedna od situacija gde broj relacija nadmašuje broj čvorova, to je samo jedan slučaj upotrebe koji ne opisuje dovoljno i ne pruža uvid u pun potencijal ove graf baze.

* Detekcija prevara

Jednostavno detektuje kada je, recimo, određen čvor koristio veliki broj kreditnih kartica u kratkom perioudu što može da

* Sistemi za preporuke u realnom vremenu
* Rutiranje u mreži
* Menadžment „master“ podacima
* Pretrage bazirane na grafu
* ...

Ovi „use-case“-ovi ne opisuju dovoljno šta sve Neo4j može da radi bolje od drugih NoSQL i SQL rešenja ali daje dobru osnovu za razmišljanje koji sve problemi za koje je potrebna napredna biznis logika i složeni SQL upiti, su skoro rešeni samom činjenicom da se koristi graf baza poput Neo4j.

# Interna struktura Neo4j-a

Arhitektura Neo4j baze je data na sledećoj slici:

Diagram, schematic

Description automatically generated

Slika 5 – Neo4j arhitektura

U narednih nekoliko podglavlja objasnićemo, počevši odozdo prema gore, neke od glavnih komponenata Neo4j baze.

## Native graf procesiranje

Pomenuli smo već da su čvorovi povezani imenovanim i usmerenim relacijama i da i čvorovi i relacije mogu da posluže kao neka vrsta kontejnera za svojstva. Iako se model sam po sebi, ne razlikuje puno u različitim implementacijama graf baze, postoji mnogo načina da kodirate i pamtite graf u glavnoj memoriji servera baze podataka. Za graf kažemo da ima prirodne sposobnosti obilaska grafa ako poseduje osobinu „index-free adjacency“[[4]](#footnote-4). Svaki čvor čuva direktnu referencu na susedne čvorove i na taj način „glumi“ neku vrstu mikro indeksa za svoje susede. Ovakav indeks je mnogo jeftiniji od globalnog indeksa i iz ovoga se vidi zašto vremena upita se ne povećavaju sa povećanjem grafa drastično. Vremenski interval izvršenja upita zavidi od dela grafa koji je pretražen a ne od ukupne veličine grafa.

Velika količina indeksa ubrzava pretragu u odnosu na „brute-force“ pristup ali je i dalje potrebna određena vrsta izračunavanja. Pogledajmo kako to Neo4J rešava bez indeksa.

Diagram, schematic

Description automatically generated

Slika 6 – Neo4J koristi relacije a ne indekse, za brzo pretraživanje.

Na slici 5, recimo da je potrebno pronaći sve Čarlijeve (Charlie) prijatelje. Sve što je potrebno uraditi je pregledati Čarlijeve odlazne *FRIEND* relacije kako bi videli ko mu je sve prijatelj. Ukoliko trebamo naći ljude kojima je Čarli prijatelj, sve što treba uraditi je naći sve ulazne *FRIEND* relacije koje pripadaju Čarliju. Za oba ova pristupa, potrebno nam je O(1) za svakog prijatelja.

Očigledno je da obilasci Neo4j grafa u teoriji, mogu biti vrlo efikasni ali da bi to i u praksi bila istina, neophodno je obezbediti adekvatnu arhitekturu u te svrhe.

## Native graf skladište

### Lančane liste

Lančane liste su srce graf baze podataka i uopšte bilo kakve implementacije grafa u programiranju. To su linearne kolekcije elemenata čiji redosled nije podrazumevan njihovim redosledom u memoriji. Umesto toga, svaki element pokazuje na sledećeg u nizu. U najosnovnijem slučaju, svaki čvor lančane liste poseduje *data* deo i *link* deo, odnosno pokazivač na sledeći element u nizu. Glavna prednost lančanih listi je ta što ne alociraju višak memorije pre vremena i dopušta jednostavno upisivanje i brisanje elemenata iz niza. Vreme pristupa je linearno jer je neophodno obići listu dok se ne nađe traženi element i to je glavni minus lančanih lista u poređenju sa nizovima[[5]](#footnote-5).

### Struktura čvorova i relacija

Ako je „index-free adjacency“ ključ visoko efikasnih upita i obilaska grafova, ključni aspekt projektovanja graf baze podataka je način na koji se sam graf pamti. Dobar format za memorisanje grafova mora da podrži brze obilaske proizvoljnim algoritmom za obilazak grafa. Neo4j skladište se sastoji iz nekoliko store-ova:

* Nodestore (Skladište čvorova) – čuva podatke vezane za čvorove
* Relationshipstore (Skladište relacija) – čuva podatke za kreirane i definisane relacije
* Propertystore (Skladište svojstava) – čuva KeyValue svojstva za čvorove i relacije
* Labelstore (Skladište labela) – čuva labele vezane za graf

Pošto Neo4j ne koristi predefinisanu šemu, postoje fiksne veličine svih podataka u store-ovima i koristi se offset kako bi se našao potreban čvor u fajlu. Pretpostavimo da je veličina čvora u bajtovima 15. Traženje čvora sa *Id* 10 će biti jednostavno jer se on nalazi na offsetu 150 (15B x 10 =150). Pristup je O(1) umesto nepotrebnih pretraga koje mogu biti minimum O(log n).

A picture containing chart

Description automatically generated

Slika 7 – Struktura rekorda čvora i relacije u store-u[[6]](#footnote-6)

Kao što se vidi sa slike 7, struktura relacije je takođe fiksne veličine da bi se jednostavno izračunavala njena lokacija. Objasnićemo ove dve strukture kako bi bilo očigledno kako rade.

Prvi bajt node-a, *inUse* bajt govori bazi podataka da li se node koristi ili je memorija slobodna za korišćenje. Naredna četiri bajta predstavljaju pokazivač na prvu relaciju koja je vezana za ovaj čvor, a naredna četiri pokazivač na prvo svojstvo čvora. Bitno je pomenuti da su ovi pokazivači zapravo lančane liste i da na taj način možemo pregledati sve relacije/svojstva čvora. Narednih 5 bajta se koriste kao pokazivač na store za labele (ukoliko ima relativno malo labela, biće zapamćene inline, bez pokazivača) dok se poslednji bajt koristi za razne *flagove* od kojih su neki u upotrebi dok su ostali ostavljeni za neku buduću funkcionalnost. Kao što se primećuje, čvor sam po sebi je prilično „lightweight“ jer sadrži nekoliko pokazivača tako da se pretraga zapravo svodi na skokove kroz memoriju pomoću ovih pokazivača.

Struktura relacija je drastično složenija, bilo da je to po broju pokazivača ili po načinu na koji su implementirane relacije. Osim *inUse* bajta, koristi pokazivače na startni i krajnji čvor, pokazivač na tip relacije (koji se nalazi u zasebnom store-u za tipove relacija) i pokazivači za prethodnu i narednu relaciju i startnog i krajnjeg čvora. Ovi pokazivači formiraju dve duplo lančane liste. Ovo možda zvuči komplikovano ali bitno je da se uoči da oba čvora poseduju relaciju i da bi bilo reduntantno čuvati dve instance ove relacije. Zbog toga, jedna duplo lančana lista predstavlja relacije vidljive sa startnog čvora koja se može obilaziti u oba smera a druga lančana lista predstavlja relacije vidljive sa krajnjem čvora.

Diagram

Description automatically generated

Slika 8 – Fizička reprezentacija grafa u Neo4j

Takođe, primetimo da relacija poseduje i tip relacije u zasebnom store-u. Ako pretpostavimo da čvor A poseduje 5 relacija tipa R1 i 1000 relacija tipa R2, pretraživanje relacija tipa R1 je i dalje O(n=5).

Pošto znamo da podaci u storeovima imaju fiksnu veličinu, možemo napraviti pretpostavku koliko će sama graf baza zauzimati memorije za određen skup podataka. Pretpostavimo scenario gde je potrebno sačuvati 4 miliona čvorova, gde svaki čvor ima 3 svojstva. Broj relacija je 2 miliona gde svaka relacija ima jedno svojstvo.

* Čvorovi: 4.000.000x15B = 60.000.000B ≈ 60MB
* Relacije: 2.000.000x34B = 68.000.000B ≈ 68MB
* Svojstva: 14.000.000x41B = 574.000.000B ≈ 574MB
* **Ukupno**: 703MB

Povećanje broja relacija u grafu na 3 miliona će prouzrokovati lako predvidiv rast memorije:

* Čvorovi: 4.000.000x15B = 60.000.000B ≈ 60MB
* Relacije: 3.000.000x34B = 102.000.000B ≈ 102MB
* Svojstva: 15.000.000x41B = 615.000.000B ≈ 615MB
* **Ukupno**: 777MB

Jednostavnom kalkulacijom možemo brzo doći do približne količine memorije koja će biti potrebna za čuvanje naših podataka u Neo4j bazi.

### Neo4j API

Iako su skladište Neo4j-a i strukture za keševe fascinantni sami po sebi, programmer retko dolazi u priliku da radi sa njima. Umesto toga, programer koristi upite kako bi manipulisao grafom. Ti upiti mogu da budu deklarativni i imperativni[[7]](#footnote-7). Cypher je deklarativni jezik koji Neo4j podržava, jednostavan je i lako se koristi. To je samo jedan od načina kako programer može da manipuliše Neo4j grafom. Na slici 9 nalazi se logički pregled API-ja koje Neo4j nudi.

Table

Description automatically generated

Slika 9 – Pregled API-ja koje Neo4j nudi

**Kernel** API nudi mehanizam za obradu događaja tokom transakcija. Programer može jednostavno da „prisluškuje“ transakcije dok se one izvšavaju i reaguje na događaje po želji. Npr. možete napraviti handler za događaj brisanja čvora i modifikovati da se brisanje ne izvršava zaista već samo markira čvorove kao „logički obrisani“ (safe delete).

**Core** API je zapravo imperativan Java API koji daje programeru pristup strukturama koje graf koristi (čvor, relacije, labele, ...). Podaci koji se čitanju se evaluiraju tek kada budu zatraženi tkz. „lazy loading“, a kada je potreban upis, ovaj API se brine da transakcija radi nesmetano i da ACID principi budu ispunjeni.

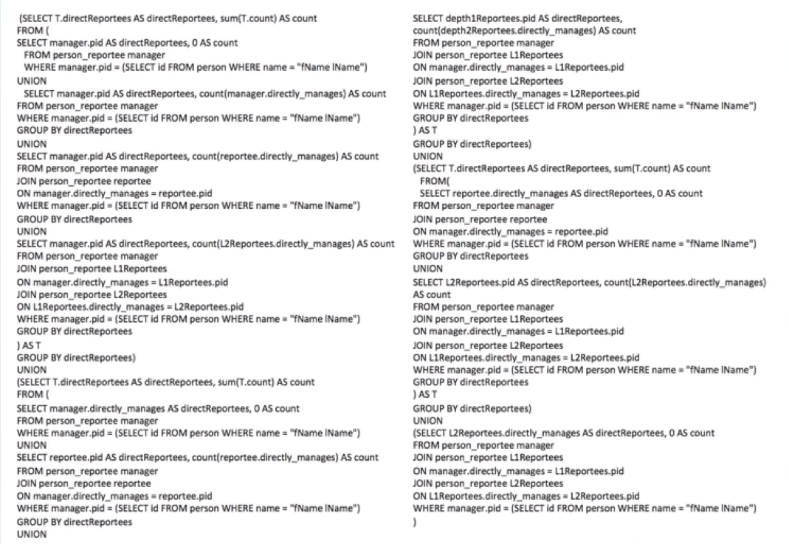
Graphical user interface, text, application, email

Description automatically generated

Slika 10 – Primer koda koji detektuje da li je Node n1 neprijatelj (ENEMY\_OF) Node-u n2

Kod na slici 10 je krajnje imperativan, da bi smo videli da li čvor n1 neprijatelj čvora n2, moramo proći kroz sve izlazne relacije čvora n1 i ukoliko naiđemo na relaciju čiji je krajnji čvor n2, vraćamo *true*.

**Traversal API** je deklarativan je služi programeru da specificira niz ograničenja koja sprečavaju da određeni delovi grafa budu obiđeni, koje relacije dolaze u obzir za pretragu, da li se pretraga vrši po dubini ili po širini kao i da definiše evaluatorsku funkciju koja se pokreće kod posete svakom čvoru. Evaluatorske funkcije mogu biti imperativnog tipa ukoliko je potrebno neko izračunavanje.   
 **Cypher**, koji smo već pomenuli, je takođe deklarativan jezik nalik klasičnom SQL-u ali sa moćnim dodatkom sintakse za pretragu relacija. Uzmimo za primer query kojim treba da vratimo sve menadžere i broj podređenih do najviše 3 nivoa ispod njega. Sa leve strane je SQL upit a sa desne je ekvivalentni Cypher upit.

Text

Description automatically generated

Očigledno je da ne treba samo razmišljati o izboru baze već i od alata koje ta baza nudi za specifične probleme. Možda je moguće napisati i kraći SQL upit iz prethodnog primera ali rad i čitljivost upita su daleko manji nego kod Cypher-a.

# Zaključak

Znati internu strukturu tehnologije koju koristimo je bitno iz mnoštva razloga. Iako se nismo isuviše detaljno čitavom Neo4j internom strukturom, date su neke osnove koje su dovoljne da imate jasnu sliku od toga šta možete očekivati od ovakve NoSQL baze. Kao što se vidi iz primera u prethodnom poglavlju, koristiti pravi alat za rešavanje određenog problema može da olakša vreme projektovanja i programiranja kao i da smanji cenu održavanja. Kao što možete da primetite, Neo4j voli da domenski entiteti budu u relaciji i očekuje da entiteti budu visoko povezani kako bi se ispunio pun potencijal ovakve baze podataka. Znati kako Neo4j izgleda iznutra može da da prilično dobre pretpostavke o tome, kakvi se podaci očekuju u bazi kao i koliko će memorije biti potrebno za čuvanje ovakve baze. NoSQL iako ima brojne prednosti, ali u realnom svetu, najčešće se koriste i SQL i NoSQL rešenja kako bi određeni delovi sistema koristili prednosti koje tehnologija nudi. Svrha ovog rada je bila da pokaže kakva je Neo4j interna struktura i na koji način pruža izuzetne performanse kada se rešavaju problemi vezani za obilaske i pretraživanje grafova.

# Literatura

1. Robinson, I, Webber, J, Eifrem E, 2015, *Graph Databases: New opportunities for connected data*, Second edition, O’Reilly Media, Inc, Sebastopol CA 95472 USA
2. Baton, J, Bruggen, RV, 2017, *Learning Neo4j 3.x: Effective data modeling, performance tuning and data visualization techniques in Neo4j,* Second edition, Packt Publishing, Birmingham.
3. Partner, J, Vukotic, A, Watt, N, 2014, *Neo4j in Action*, Manning Publications Co., 3 Lewis Street Greenwich, CT, United States
4. McCreary, D, Kelly, A, 2013, *Making Sense of NoSQL: A guide for managers and the rest of us,* Manning Publications Company, 20 Baldwin Road, Shelter Island, NY 11964
5. Neo4j: Graph academy:

<https://neo4j.com/graphacademy/>

1. Neo4j: Understanding data on disk: <https://neo4j.com/developer/kb/understanding-data-on-disk/>
2. NoSQL: The future of data economy: <https://www.livemint.com/Opinion/ub2sqRtvuGDgDPDDTP7gtL/NoSQL-The-future-of-data-economy.html>

1. NoSQL je zapravo abrevijacija za *Not Only SQL*, mnogi NoSQL sistemi zapravo podržavaju i SQL upite iako su podaci nestruktuirani ali uglavnom imaju neke druge mehanizme (jezike) ekvivalentne SQL-u koji su adaptirani za specifičan slučaj korišćenja. [↑](#footnote-ref-1)
2. ACID svojstva predstavljaju skraćenicu za atomičnost (A – atomicity), konzistentnost (C – consistency), izolaciju (I – isolation) i izdržljivost (D – durability). Ovo su svojstva koje svaka SQL baza garantuje. [↑](#footnote-ref-2)
3. CAP teorema kaže da svaka distribuirana baza podataka ne može da obezbedi više od dva navedena svojstva - konzistencije, dostupnosti i tolerancijе razdvojenosti. [↑](#footnote-ref-3)
4. Globalni indeksi ne postoje već svaki čvor predstavlja mikroindeks za svoje susede. Ovo smanjuje vreme traženja sa O(log n) u grafovima sa mnogo indeksa, na O(1) na grafovi koji imaju index-free adjacency svojstvo. [↑](#footnote-ref-4)
5. Na modernim mašinama gde CPU imaju optimizovane keš memorije u više nivoa, normalni nizovi mogu vrlo često da daju približne ili bolje rezultate od lančanih listi čak i ako je u pitanju operacija za koju lančana lista na papiru, ima prednost. Lančane liste se koriste za implementaciju različitih grafova kao i za specijalizovane kompleksne strukture kao što je Fibonačijev Heap. [↑](#footnote-ref-5)
6. Ovo važi za Neo4j verzije 3.0 – 3.5 [↑](#footnote-ref-6)
7. Deklarativni jezik se koristi kada hoćemo da kažemo šta želimo ali nas ne zanima kako ćemo doći do rezultata dok imperativnim jezikom opisujemo kako doći do podataka. U mnogo slučajeva, koristi se miks ove dve paradigme tako da nije sve toliko „crno-belo“. [↑](#footnote-ref-7)