Elektronski fakultet, Univerzitet u Nišu
Katedra za Računarstvo i informatiku

SEMINARSKI RAD

Fizičko projektovanje i optimizacija podataka za Neo4j NoSQL bazu podataka

Predmet: Sistemi za upravljanje bazama podataka

Student: Nikola Stanisavljević
Broj indeksa: 940

Sadržaj

Uvod

Težnja ovog rada je da prikaže način projektovanja baze podataka za Neo4j rešenje, odnosno modeliranje i optimizaciju grafovske baze podataka. Uvodni deo odnosi se na uopšteno fizičko projektovanje baza podataka, u radu se porede relaciona i NoSQL rešenja, a zatim se osvrćemo na konkretnu implementaciju, planiranje, modeliranje i osobine Neo4j grafovske baze podataka.

Relacione baze podataka su i dalje veoma pristupačne, ali proces modeliranja kod njih je znatno komplikovaniji, a sa ubrzanim razvojem web aplikacija javila se i potreba za drugačijim tehnikama za skladištenje podataka. Na web-u postoji ogromna količina polustruktuiranih podataka sa fleksibilnim šemama. **NoSQL** je pokret koji je predstavljen 1998. godine, a obuhvata sve nerelacione baze podataka. Neki od primera *Not-Only-SQL* baza podataka su *Google BigTable* iz 2006. godine, Amazonov *Dynamo* iz 2007. godine, *Cassandra* u upotrebi Facebook-a iz 2008. godine i *Voldemort* koji koristi LinkedIn iz 2009. godine. Relacione baze podataka su korisne za efikasno skladištenje podataka, jer imaju podršku za ACID[1] transakcije i kompleksne SQL upite. Kada su u pitanju podaci na web-u i obrada tih podataka pravila se menjaju i upotreba relacionih baza podataka gubi na značaju posebno zbog nedostatka fleksibilnosti. Osnovna podela NoSQL baza podataka data je u nastavku sa primerima implementacija u zagradama:

- Key/Value Stores (*Dynamo, Redis, Voldemort*)
- Column stores (*BigTable, Cassandra*)
- Document stores (*MongoDB*)
- Graph databases (*Neo4j*)

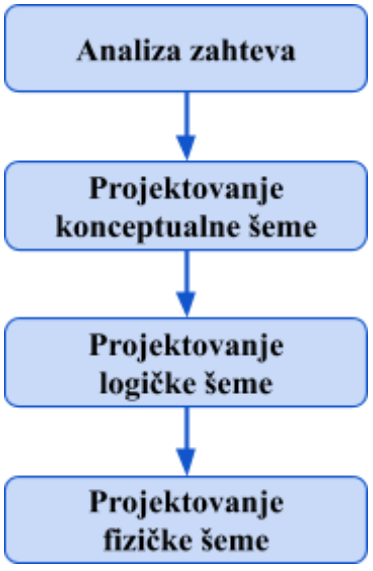
[1] *ACID - Atomicity Consistency Isolation Durability* - predstavlja skup svojstava transakcije u bazama podataka namenjenih da garantuju validnost čak i u slučaju grešaka, nestanka struje i slično.

Fizičko projektovanje i optimizacija podataka

Projektovanje baze podataka

U opštem slučaju postoji 3 nivoa apstrakcije podataka u okviru jednog Sistema za upravljanje bazom podataka (skr. DBMS):

- **Fizička šema**
  Opisuje datoteke i indekse koji su korišćeni pri implementaciji na nekom fizičkom uređaju.
- **Logička šema**
  Srednji nivo apstrakcije na kome se definiše logička struktura baze, odnosno način na koji se podaci iz fizičke baze podataka predstavljaju korisniku u opštem slučaju.
- **Konceptualna ili Eksterna šema**
  Najviši nivo apstrakcije koji predstavu o podacima iz baze prilagođava potrebama korisnika (grupe korisnika). Ova šema opisuje kako korisnici vide podatke.



***Slika1**. Osnovni koraci u projektovanju baze podataka*

Struktuiranje i organizacija imaju poseban značaj kada se radi sa bazama podataka, jer je podatke potrebno struktuirati na nivou korisnika, a organizovati za što optimalnije održavanje na nivou mašine. Reprezentacija koja se nalazi na logičkom nivou apstrakcije naziva se model podataka. Modelom podataka se predstavlja logička struktura svih podataka u bazi, kao i skup svih operacija koje korisnik može izvršiti nad tim podacima.
Fizičko projektovanje baze podataka podrazumeva internu (fizička) organizacija podataka (prostori za tabele, kontejneri, stranice, baferi) kao i pomoćne komponente (indeksi).

Zadatak određenog sistema za upravljanje bazama podataka (engl. *Database Management System*) je da zahteve koje korisnik postavlja na konceptualnom nivou nad logičkim tipovima podataka (npr. relacijama i torkama) preslika, na unutrašnjem nivou, u zahteve nad internom organizacijom podataka. Terminologija, kao i rešenja pojedinih problema u ovoj oblasti bitno se razlikuju od sistema do sistema, ali principi fizičke reprezentacije i pristupa podacima uglavnom su standardni.

Optimizacija podataka

Brzim rastom količine podataka javlja se stalna potreba za proširivanjem baze, što dovodi do otežanog pristupa podacima. Neophodne su efikasnije tehnike za čuvanje i obradu podataka. Administrator baze podataka mora da ima slobodu da promeni fizičku reprezentaciju ili pristupne tehnike radi boljih performansi bez promena postojećih aplikacija. Da bi se obezbedile visoke performanse neophodno je vršiti manuelnu i automatsku optimizaciju. Takođe, baza treba da bude sposobna da se širi i da bude skalabilna. Skalabilnost je sposobnost sistema da podnese povećanje zahteva i broja korisnika, tako da sistem ne postane previše kompleksan i skup.

Postizanje boljih performansi može se opisati kroz tri široke kategorije:

- Optimizacija na nivou interne organizacije podataka;
  Ne menja se logički model, odnosno skup tabela i kolona. Koriste se pomoćne komponente (indeksi) i resursi (upravljanje memorijom).
- Optimizacija na nivou upita;
- Optimizacija na nivou strukture podataka.
  Promena fizičke strukture podataka u odnosu na logički model.

Modeliranje podataka predstavlja proces apstrakcije. Ovaj proces podrazumeva da se korisničke potrebe i biznis model mapiraju se na određenu strukturu koja služi za organizaciju i smeštanje podataka, ali ovaj proces uopšte nije jednostavan kod tradicionalnih sistema za upravljanje bazama podataka.

Relaciono i graf modeliranje

Relacioni model je implementacioni model podataka, jer nudi koncepte koji su pogodni za direktnu implementaciju na računaru. Tvorac ovog modela je E.F. Ted Codd, a za predstavljanje informacije u ovom modelu koristi se kolekcija tabela. U bazi, relacija predstavlja tabelu, a atribut kolonu, domen predstavlja skup dozvoljenih vrednosti atributa, a torka jedan red u tabeli. Relacioni model podataka dobija se direktnim prevođenjem ER ili EER modela. EER model predstavlja prošireni (enhanced) ER model. Ovaj model osnovnom ER modelu dodaje koncept klase, podklase, nadklase, nasleđivanja, specijalizacije, generalizacije i kategorije.

Dok logička organizacija vidi relaciju odnosno tabelu kod RDBMS-a, fizička organizacija podataka ide daje od toga. Tu se primećuje osnovni **skladišni prostor za tabele** (engl. *Table space*) sa veličinom fizičke stranice, načinom i uslovima baferovanja stranica i kontejnerima (diskovi, particije, fajlovi, direktorijumi) koji čine prostor za tabele. Svaka tabela ili indeks sastoje se od stranica, koje sadrže redove koji se pretražuju. Postoji i **bafer stranica** (engl. *Buffer pool*) predviđen je za čuvanje kopije dela stranica radi omogućavanja bržeg pristupa podacima, najčešće za čuvanje indeksa (ili prvih nekoliko nivoa indeksa). Indeksi su pomoćne strukture podataka koje omogućavaju brži pristup podacima, odnosno bržu pretragu podataka po unapred izabranom ključu.

Dobra konfiguracija prostora za tabele i bafera stranica mogu biti od presudnog značaja za performanse. Osim navedenog, fizička organizacija podataka bavi se i particionisanjem tabela, kompresijom podataka i mnogim drugim konceptima koji su često specifični za određene implementacije.

Relacione baze podataka imaju mnoge pozitivne strane, međutim, one imaju svoje nedostatke u vidu:

- relativno visoke cene čitanja
  – zbog neredundantnosti i stroge strukture podataka, odnosno, zbog čestog spajanja podataka;
- otežanog distribuiranja
  – zbog konzistentnosti podataka;
- skupe promene strukture
  – zbog povezanosti strukture sa upotrebom i optimizacijama;

Kod grafovske baze podataka, modeliranje se vrši na prvom koraku pri razmišljanju o entitetima i povezivanju nacrtanih primera na beloj tabli (engl. *whiteboard*) ili papiru. Posebno je značajno to što je ovaj model fleksibilan i omogućava brze i lake promene u budućnosti. Naravno, iako je proces modeliranja podataka znatno olakšan kod grafovskog modela, neophodno je obratiti pažnju na konkretni slučaj korišćenja, kako bi dizajn baze bio što efektivniji. U nastavku biće više reči o graf bazama podataka i konkretnim primerima projektovanja i optimizacije.

Graf baze podataka i Neo4j

Graf baze podataka predstavljaju tip NoSQL baza podataka koje su inspirisane matematičkom teorijom grafova. Po definiciji, graf baza podataka je bilo koji sistem za skladištenje podataka koji omogućava definisanje relacija nezavisno od indeksnih i *lookup* struktura. To znači da svaki element može da sadrži direktan pokazivač ka drugim elementima bez potrebe za postojanjem indeksnih struktura.

Graf predstavlja kolekciju **čvorova** i **potega**, odnosno veza između njih. Model grafa je prirodan način modeliranja podataka u ljudskom umu. Svaki entitet predstavlja jedan čvor, a taj čvor je povezan usmerenim potegom sa drugim čvorom. Svaki poteg ima svoj izvorni i odredišni čvor. Uz to, svaki čvor i poteg mogu imati **svojstva**, što su u stvari atributi entiteta ili veza. Još jedna od karakteristika graf baza je ta što nemaju unapred definisanu šemu baze, već se šema kreira unošenjem podata u bazu.

Pogodne operacije za graf baze podataka su: obilazak grafa, traženje najkraćeg puta između dva čvora, provera da li postoji put između dva čvora, određivanje suseda i slično. Graf baze podataka se koriste u sistemima sa velikom količinom podataka, gde su veze između podataka važan deo sistema.

Neo4j

Jedan od najpoznatijih i najčešće korišćenih sistema za upravljanje graf bazom podataka je Neo4j, razvijen od strane kompanije *Neo Technology*. U pitanju je *open-source* i komercijalno rešenje dostupno za različite operativne sisteme. Verzija 1.0 objavljena je februara 2010. godine, verzija 2.0 decembra 2013. godine, verzija 3.0 aprila 2016. godine, a najnovija 4.0.2 verzija 17. marta 2020. godine.

Neo4j je od početka razvijan kako bi bio graf baza podataka, što znači da je njegova arhitektura dizajnirana tako da zadovolji optimizaciju brzog upravljanja, skladištenja i pronalaska čvorova i veza između njih.

Kod relacionih baza podataka, operacija spajanja (engl. JOIN) će se degradirati eksponencijalno sa povećanjem broja veza koje učestvuju u tom spajanju. Odgovarajuća akcija u Neo4j je linearna, s' obzirom da se izvodi kao navigacija od jednog do drugog čvora. Ovakav pristup kod skladištenja podataka i izvođenja upita nad vezama između entiteta se pokazala jako uspešnom. Sa obzirom da je većina pretraživanja graf baze podataka lokalizovana u širem susedstvu čvora, **količina podataka skladištenih u bazi neće uticati na vreme izvršenja upita**. Ovakvo posvećeno upravljane memorijom i visoko skalabilne i memorijski efikasne operacije uveliko doprinose performansama.

Neo4j omogućava upotrebu istog modela kroz koncepciju, dizajn, implementaciju, skladištenje i vizuelizaciju tih podataka. Glavna prednost ovoga je što omogućava **učešće svim partnerima iz domena problema u razvojnom ciklusu**. Takođe, **moguće je razvijati**

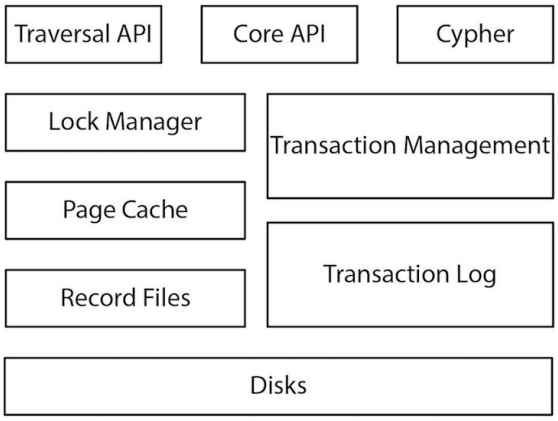
domenski model neprestano dok se menjaju zahtevi, bez skupih promena šeme i migracija. Za reprezentaciju podataka koji se šalju Neo4j serveru i dobijaju od Neo4j servera koristi se JSON, koji se šalje u delovima.

Native i *Non-native* graf baze

Dva glavna elementa kod kojih se uočava razlika *native* i *non-native* graf baza podataka su **skladište grafa** (engl. *Graph storage*) i **mašina za procesiranje grafa** (engl. *Graph processing engine*). Skladište grafa predstavlja strukturu baze koja sadrži graf i može biti posebno dizajnirano za smeštanje i organizaciju grafova ili bazirano na relacionom modelu ili objektno-orijentisanim bazama. **Mašina za procesiranje grafa** može biti *native,* za obrađivanje čvorova koji su međusobno povezani i stvaraju obrasce ili *non-native* kada na neki drugi način obrađuje CRUD operacije i nije optimizovana za grafove.

U trenutku upisivanja, **pridruživanje bez indeksa** (engl. *index-free adjacency*) ubrzava procesiranje time što osigurava da svaki čvor bude čuvan direktno u svojim susednim čvorovima i vezama. U trenutku čitanja, pridruživanje bez indeksa omogućava veoma brzo pretraživanje bez oslanjanja na indekse.

***Non-native* obrada grafova** uglavnom koristi veliki broj indeksa, što značajno usporava operacije čitanja ili upisa. Povezani podaci zahtevaju neobično strogu potrebu za integritetom podataka koji prevazilazi ostale NoSQL modele. Da bismo ažurirali vezu između dva entiteta, moramo napisati upit veze i ažurirati čvor na svakom kraju te veze. Ako bilo koja od ovih operacija upisa bude neuspešna rezultat je oštećen graf. Upisi zato moraju zadovoljavati *ACID* svojstva.



***Slika2**. Prikaz arhitekture Neo4j native graf baze*

***Native* sistemi** osiguravaju upotrebu internih struktura i obezbeđuju kvalitet podataka. Kod Neo4j baze svaki arhitekturni sloj od Cypher upitnog jezika do datoteka na disku, je optimizovan za čuvanje i procesiranje grafovskih podataka. Ovi podaci čuvaju se u skladišnim datotekama, od kojih svaka sadrži podatke za određeni deo grafa, kao što su čvorovi, veze, oznake i svojstva. Ovakva podela skladišta poboljšava performanse efikasnih

grafovskih prolazaka (engl. *traversals*). Matična baza grafa za svrhu ima da ukazuje na spiskove veza, oznaka i svojstava, što je čini veoma rasterećenom.

Non-native graf skladišta koriste relacionu bazu ili neki slični generalni sistem za skladištenje koji nije specifično projektovan za jedinstvene karakteristike grafova, što utiče na performanse i skalabilnost. Zbog toga što nisu projektovane i optimizovane za skladištenje grafova, ovakve baze nemaju ni memorijsku strukturu koja je prethodno opisana, tako da čvorove, veze, oznake i svojstva mogu da čuvaju na različitim mestima u memoriji. Tako se brzo stvaraju problemi pri čitanju, jer za svaki upit biće neophodno da se ceo graf ponovo sastavi iz memorije.

Graf baza podataka ima *native* sposobnosti procesiranja ukoliko koristi pridruživanje bez indeksa. To znači da svaki čvor direktno upućuje na svoje susede, delujući kao mikro-indeks za sve obližnje čvorove. Pridruživanje bez indeksa je jeftinije i efikasnije kod obavljanja istog zadatka sa indeksima, jer su vremena izvšenja upita proporcionalna veličini pretraženog grafa, a ne povećavaju se sa ukupnom veličinom podataka. Budući da se veze čuvaju kao prvoklasne celine, one se lakše prelaze u bilo kom smeru pomoću native obrade grafa. S druge strane, *non-native* graf baze koriste mnogo vrsta indeksa za povezivanje čvorova, čineći proces skupljim i sporijim.

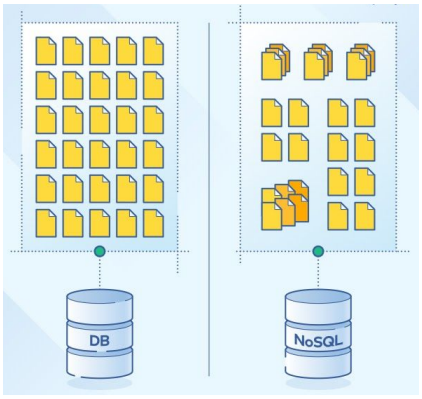
Prednosti graf baza u odnosu na relacione

Pored već pomenutih osobina relacionih i graf baza vezanih za modeliranje ovih sistema, u nastavku su dodatno obrazložene karakteristike ova dva tipa baza podataka.

Relaciona baza podataka je tip baze podataka kod koje se organizacija podataka zasniva na **relacionom modelu**. Podaci su organizovani u skup relacija između kojih se definišu određene veze. Međutim, nedostatak relacionih baza je taj što postaju jako komplikovane kada je u pitanju pristup vezama među podacima. Kada se podaci normalizuju da bi se eliminisali duplikati i inkonzistencije, mnoga polja počnu da referenciraju auto-generisane brojeve i tada čitanje i održavanje podataka postaje komplikovano. Normalizacija predstavlja sistematski metod za osiguravanje da je struktura baze pogodna za upite opšteg tipa (unos, ažuriranje i brisanje), odnosno da ne ispoljava anomalije pri ovim upitima i dovodi do gubitka integriteta podataka. Relaciona baza je normalizovana ukoliko se nalazi u trećoj normalnoj formi (3NF).

Relacione baze podataka imaju striktnu šemu koja je definisana na samom početku implementacije baze, dok je kod graf baza podataka šema fleksibilna i može se menjati u hodu. Još neke od prednosti graf baza podataka su:

- Brzina izvođenja upita ne zavisi od količine podataka;
- Mogućnost reprezentacije podataka na prirodan način, kao skup objekata (čvorova) povezanih skupom objekata (veza);
- Domenski model se može izraditi u kratkom roku obzirom da procesi normalizacije i denormalizacije nisu potrebni.



***Slika3**. Prikaz sistema relacionih naspram NoSQL baza podataka*

Kod relacionih baza šema podataka je fiksna. Svaka tabela u bazi podataka je kolekcija fiksnih atributa i njihovih vrednosti. Bilo kakve strukturne promene predefinisane baze zahtevaju puno vremena i novca. Kao što je prethodno pomenuto, **kod graf baza šema je fleksibilna** i ažuriranje grafa novim funkcionalnostima sprovodi se jednostavno, bez velikog utroška resursa i bez promene postojećih podataka u bazi.

Kod relacionih baza **veze između podataka** koji su sačuvani u različitim tabelama ostvaruju se pomoću stranih ključeva ili dodatnih tabela. Veze između podataka u graf bazi ostvaruju se direktno povezivanjem čvorova i mogu biti imenovane i sadržati svoje atribute.

Performanse kod manjih sistema relacionih baza podataka su zadovoljavajuće i izvođenje upita je dovoljno brzo. Međutim, kod većih relacionih baza, 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). **Sa porastom količine podataka u bazi, performanse graf baza su bolje u odnosu na performanse koje pružaju relacione baze podataka.** Upiti se izvršavaju brže, jer se traže samo oni podaci koji su direktno povezani sa podacima navedenim u upitu. Upiti se izvršavaju samo nad određenim delom grafa. Relacione baze su optimizovane za agregaciju podataka, dok su graf baze optimizovane za veze između podataka.

Dubina	Vreme izvršenja u sekundama [s]		Broj vraćenih redova
	RDBMS	Neo4j	
2	0.016	0.01	~ 2 500
3	30.267	0.168	~ 110 000
4	1543.505	1.359	~ 600 000
5	/	2.132	~ 800 000

***Tabela1**. Poređenje performansi relacione i Neo4j baze*

Tabela1 prikazuje poređenje relacionih baza podataka sa konkretnim primerom graf baze Neo4j, koja će dodatno biti opisana u narednim poglavljima. U pitanju je pretraga po dubini za slučaj društvene mreže, gde se traže friends-of-friends od dubine 2. Na ovom nivou se sa

korisničke strane ne primećuje tolika razlika, jer su u pitanju milisekunde. Kako dubina pretrage raste tako se vidi i zaostajanje relacione baze u odnosu na Neo4j. Za dubinu 5, u slučaju relacione baze upit nije mogao da bude završen, dok je kod baze Neo4j bilo potrebno 2.132 sekundi za ukupno vraćenih preko 800 000 redova.

Tabela2 u nastavku prikazuje neke od osnovnih karakteristika baza podataka opisanih kod Neo4j kao predstavnika graf baza podataka, relacionih baza i uopšteno NoSQL rešenja.

	Neo4j	**RDBMS**	**NoSQL DB**
Smeštanje podataka	Struktura grafa	Fiksne, predefinisane tabele	Povezanost podataka nije podržana na nivou db
Modeliranje podataka	Fleksibilan model	Model baze mora biti razvijen iz logičkog modela	Nije primenljiv na enterprajz arhitekture
Performanse upita	Odlične performanse bez obzira na dubinu veza	Porastom podataka se znatno usporava procesiranje podataka	Veze moraju biti kreirane na nivou aplikacije
Upitni jezik	**Cypher**: native upitni jezik za graf baze	**SQL**: kompleksnost raste sa brojem JOIN operacija	Različiti jezici se koriste, ali slabost je što nisu krojeni za povezane podatke (veze u grafu)
Karakteristike kod transakcija	Održana ACID svojstva	ACID	BASE[2]
Skalabilnost	Upiti na osnovu obrasca (engl. *pattern*) ilustruju skalabilnost	Skaliranje kroz replikaciju je skup proces	Skaliranje je moguće, ali integritet podataka nije osiguran

***Tabela2**. Prikaz osobina Neo4j, relacionih i uopšteno NoSQL baza podataka*

[2] *BASE - Basically Available, Soft State, Eventual consistency* - Skup svojstava transakcija koje ispunjavaju NoSQL baze podataka, nasuprot ACID svojstvima kod relacionih baza.

Osnovni koncepti Neo4j graf baze

Graf čine dva osnovna elementa: čvor i veza. Svaki čvor predstavlja određeni entitet (osobu, lokaciju, stvar, kategoriju i sl.), a svaka veza predstavlja način na koji su povezana dva čvora. Pored čvorova i veza, u nastavku su opisani i ostali koncepti Neo4j graf baze kroz koje su predstavljeni podaci.

Čvorovi (engl. *Nodes*):

- Predstavljaju entitete;
- Sa drugim čvorovima povezani su vezama;
- Mogu imati neograničen broj svojstava;
- Imaju jednu ili više oznaka koje grupišu čvorove i opisuju njihovu ulogu u grafu.

Veze (engl. *Relationships*):

- Povezuju čvorove i struktuiraju graf;
- Imaju usmerenje, naziv i izvorni i odredišni čvor;
- Mogu imati jedno ili više svojstava.

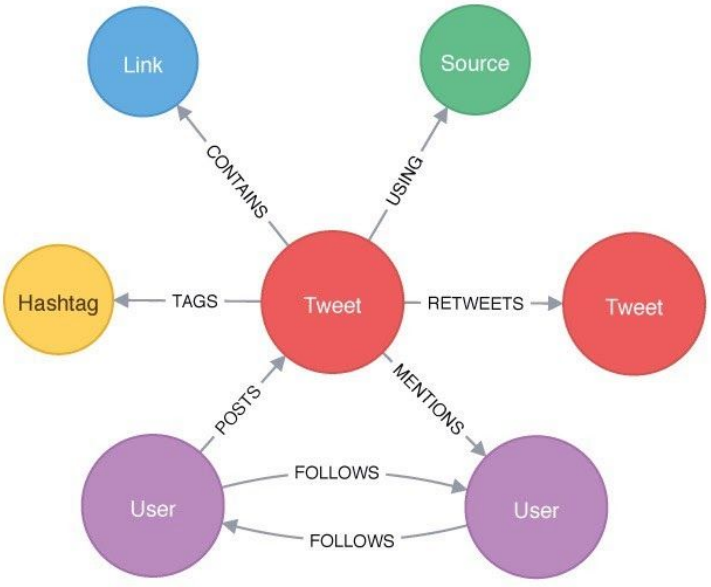
Svojstva (engl. *Properties*):

- Predstavljaju atribute entiteta ili veza;
- Predstavljaju ključ-vrednost parove, gde je ključ string.

Oznake (engl . *Labels*):

- Grupišu čvorove u setove;
- Indeksiraju se da bi ubrzale pretraživanje čvorova u grafu.

Na Slika4 prikazani su **čvorovi** i **veze** u pokaznom primeru Neo4j baze za društvenu mrežu *twitter*.



***Slika4**. Prikaz primera čvorova i veza u Neo4j bazi društvene mreže twitter*

Cypher Query Language sa primerima

Cypher predstavlja deklarativni upitni jezik grafova koji omogućava ekspresno i efikasno izvršavanje upita i ažuriranje grafa. Deklarativan znači da se fokusira na aspekte rezultata, a ne na metode ili načine za dobijanje rezultata, tako da je lako čitljiv. Dizajniran je da bude jednostavan, ali moćan. Komplikovani upiti mogu biti izraženi na prost način, što programeru daje mogućnost fokusiranja na domenski model, umesto na komplikovani pristup i komplikovano ažuriranje baze. Budući da je vrlo čitljiv, upiti napisani u Cypher jeziku su veoma laki za održavanje, čime je pojednostavljeno održavanje aplikacija. Upitni jezik **Cypher je inspirisan SQL jezikom**, što se može videti i po nazivima određenih naredbi i klauzula koje su identične onima u SQL jeziku, ali ima i elemente SPARQL-a[3].

Glavna prednost Cypher jezika je njegova orijentacija ka graf bazama i jednostavnije projektovanje upita za graf strukture, za razliku od SQL upita koji mora koristiti više JOIN operacija kako bi došao do informacija koje su projektovane kroz relacije.

U nastavku je objašnjeno kako se osnovni koncepti graf baza prikazuju Cypher jezikom.
Cypher koristi par zagrada () za reprezentaciju **čvora**:

```
() - nekategorizovan anonimni čvor
(clan) - imenovan nekategorizovan čvor
(:Alumni) - anonimni kategorizovan čvor
(clan:Alumni) - imenovan i kategorizovan čvor
(clan:Alumni {ime: 'Aleksandra'})
```

Prazne zagrade predstavljaju nekategorizovan, anonimni čvor. Ukoliko nam treba neka referenca na čvor, njemu se daje naziv (clan), a ukoliko želimo da napravimo oznaku to se radi dodavanjem dvotačke ispred naziva (:Alumni). Referenca predstavlja konkretan čvor u sistemu, dok se oznake koriste za grupisanje čvorova. Dodatni atributi čvora definišu se i deklarišu u vitičastim zagradama u *key-value* parovima.
Za **veze** koristi se par crta (--), ukoliko dodamo i simbol veće ili manje veza postaje usmerena (<--, -->). U uglastim zagradama [] se dodaju detalji, a unutar njih mogu biti definisani atributi veze.

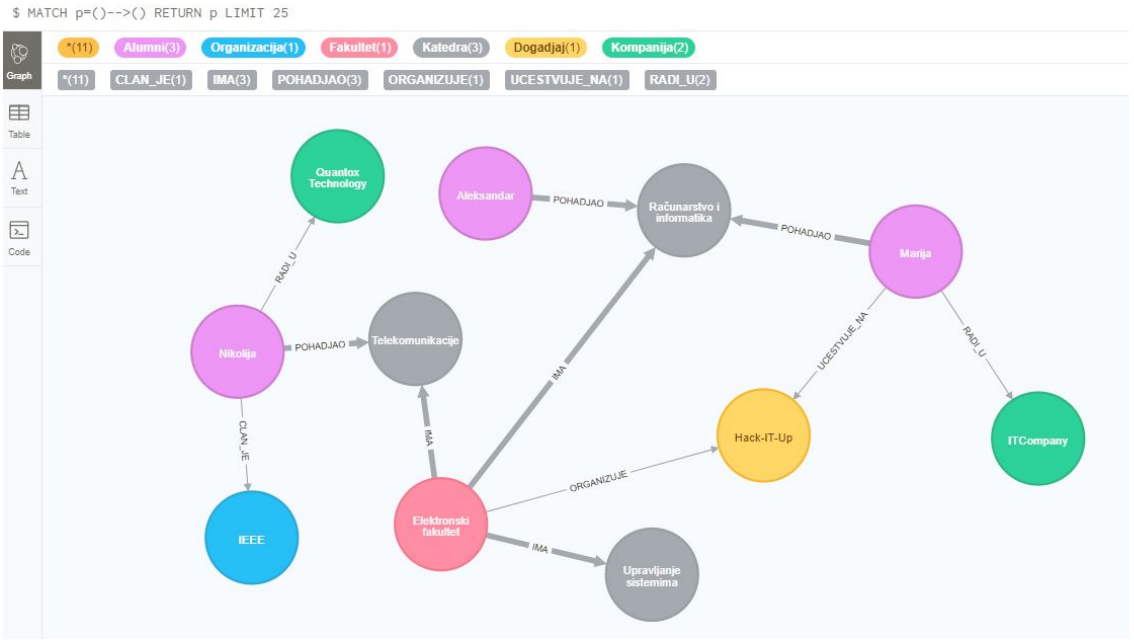
```
-[:POHADJAO_JE {datum_upisa: '1979-10-01'}]->
```

Sintaksa definisanja veza je slična kao kod čvora.
Kombinacijom definicija čvora i veza izražavju se **obrasci** (engl. *pattern*). Na primer:

```
(a1:Alumni {ime: 'Aleksandra', prezime: 'Stanojević'})
-[:POHADJAO_JE {datum_upisa: '1979-10-01'}]->
 (k1:Katedra {naziv_katedre: 'Računarstvo i informatika'})
```

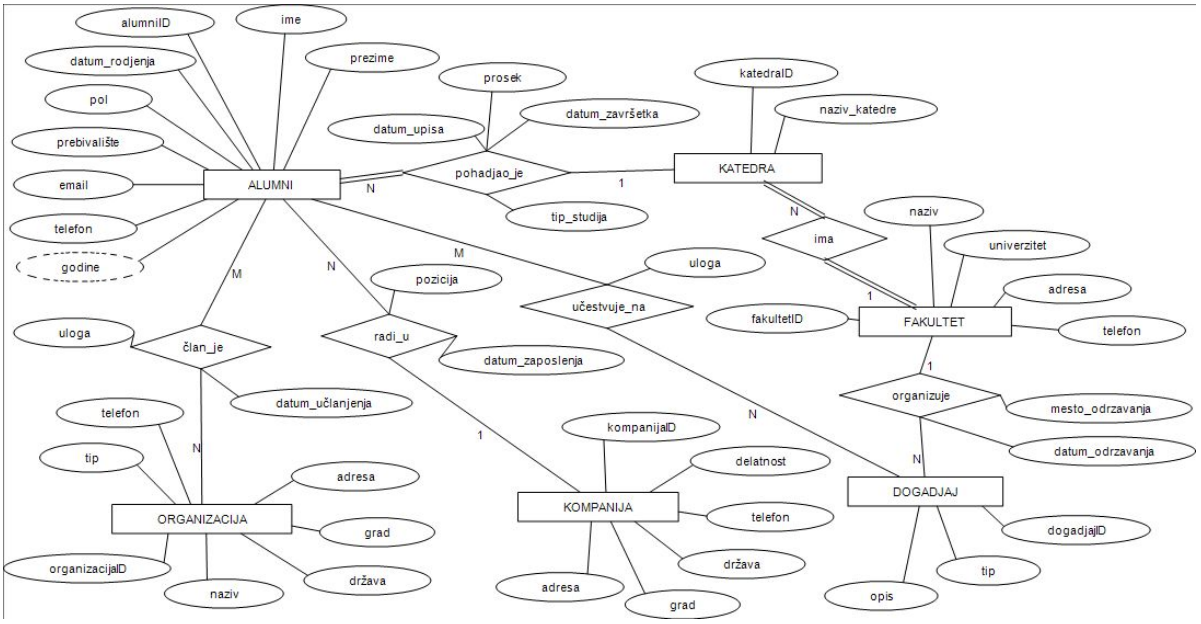
Zarad povećanja modularnosti i smanjivanja ponavljanja, Cypher omogućava da obrasci budu dodeljeni promenljivama.

[3] *SPARQL - Protocol and RDF Query Language* - upitni jezik koji omogućava rad sa triplestore, upitima koji vrše konjukciju, disjunkciju i koriste opcione paterne. Posebno je značajan u radu sa semantic web-om.



Slika5. *Prikaz primera graf baze u Neo4j Browser-u*

Na Slika5 prikazan je primer grafovske baze sa **čvorovima**: *Alumni, Organizacija, Fakultet, Katedra, Događaj* i *Kompanija* i **vezama**: *CLAN_JE, IMA, POHADJAO_JE, ORGANIZUJE, UCESTVUJE_NA* i *RADI_U*. Klikom na svaki od čvorova ili veza, biće prikazani atributi odnosno svojstva iste. Radi lakšeg pregleda primerne baze Slika6 prikazuje ER dijagram za slučaj relacionog modeliranja.



Slika6*. ER model primerne baze*

Primer naredbe **CREATE** za kreiranje odgovarajućih čvorova je dat u nastavku za entitete *Alumni, Katedra, Fakultet* i za vezu *POHADJAO_JE* između alumni člana i katedre, i vezu *IMA* između fakulteta i katedre.

```
CREATE (Alumni1:Alumni {ime:'Marija', datum_rodjenja:'1960-01-20',
pol:'Z', prebivaliste:'Niš', email:'marija@itc.rs',
telefon:'+38160123456', prezime:'Marković'})

CREATE (Katedra1:Katedra {naziv_katedre:'Računarstvo i
informatika'})

CREATE (Alumni1)-[:POHADJAO_JE {datum_upisa:'1979-10-01',
prosek:'9,4', datum_zavrsetka:'1984-06-01', tip_studija:'Osnovne
akademske'}]->(Katedra1)

CREATE (Fakultet1:Fakultet {naziv:'Elektronski fakultet',
univerzitet:'Univerzitet u Nišu', adresa:'Aleksandra Medvedeva
14', telefon:'+381 18 529105'})

CREATE (Fakultet1)-[:IMA]->(Katedra1)
```

U nastavku vidimo prednost Cypher jezika u odnosu na SQL na primeru u kome želimo da prikažemo imena svih alumni članova koji su učestvovali na događajima tipa hakaton. Konkretan primer SQL upita sa M-N vezom i dva LEFT JOIN-a:

```
SELECT ime FROM Alumni
LEFT JOIN Ucestvuje_na
ON Alumni.alumniID = Ucestvuje_na.alumniID
LEFT JOIN Dogadjaj
ON Dogadjaj.dogadjajID = Ucestvuje_na.dogadjajID
WHERE tip = "hakaton"
```

Analogni primer u jeziku Cypher:

```
MATCH(a:Alumni)-[:UCESTVUJE_NA]->(d:Dogadjaj)
WHERE d.tip = "hakaton"
RETURN a.ime
```

Karakteristike modeliranja graf baza

Kao i kod projektovanja bilo kojeg tipa baze, **model podataka igra značajnu ulogu u osmišljavanju logike i načinu predstavljanja podataka**. Glavna razlika u odnosu na relacione baze je to što Neo4j nema striktnu šemu (engl. *schema-free*) i može se lako prilagoditi i izmeniti.

Dobro struktuirana baza podrazumeva sledeće:

- Čuva memorijski prostor eliminisanjem suvišnih podataka;
- Održava tačnost i integritet podataka;
- Pruža pristup podacima na korisne načine.

Fizički model graf baza nalaže da se sledeći aspekti uzmu u razmatranje:

1. Jedinstvenost i identitet;
2. Tipovi podataka;
3. Atributi veza;
4. Povezanost podataka;
   Jedan tip veze može biti usmeren ka različitim tipovima objekata (čvorova). Sa jedne strane to znači da je intuitivna i oslikava ljudsku logiku, ali sa druge strane može dovesti do komplikacija pri obilasku grafa ukoliko semantika nije dovoljno jasna.

Graf označenih svojstava

Jedna od najpopularnijih formi za modeliranje grafovskih baza podataka zove se **Graf označenih svojstava** (engl. *Labeled property graph*) i on ima sledeće osobine:

- Sadrži čvorove i veze;
- Čvorovi sadrže svojstva i čine *Key-Value* parove;
- Čvorovi mogu imati jednu ili više oznaka;
- Veze su imenovane i usmerene, tako da uvek imaju početni i krajnji čvor;
- Veze mogu imati svojstva.

Čvorove često kategorizujemo prema korisničkim ulogama (engl. *user roles*), pa se oznake koriste da bi ukazale na grupe korisnika. Ove karakteristike čine način modeliranja razumljivim i intuitivnim, a ipak dovoljno dobrim da opiše različite slučajeve korišćenja na adekvatan način.

Graf baze su prirodno aditivne, tako da nove veze, čvorove, oznake i podgrafove možemo dodavati bez remećenja postojeće strukture i funkcionalnosti aplikacije. Zbog toga u početnoj fazi razvoja baze ili neke aplikacije ne moramo imati unapred definisanu, finaliziranu i striktnu šemu baze.

Jedan od izazova relacione paradigme je taj što normalizovani modeli generalno nisu dovoljno brzi za potrebe aplikacija u realnom vremenu. Često je potrebno odraditi

denormalizaciju, odnosno uskladiti model za mašinu za procesiranje baze podataka (engl. *database engine*) umesto za korisničku interpretaciju. U ovom procesu često dolazi do velikog dupliranja podataka, kako bi se poboljšale performanse upita. Na primer, korisnik može imati više e-mail adresa i tipično, u normalizovanom modelu postojala bi posebna tabela koja čuva sve adrese sa referencama na korisnike. Kako bi se smanjila količina JOIN operacija u upitima, tabela koja sadrži korisnika bi morala da sadrži dodatne kolone za sve e-mail adrese koje su nam često potrebne. Na ovaj način predstavljena je denormalizacija kod relacionih baza.

Kod grafovskih baza postoje tehnike koje pomažu u ovom slučaju. Jedna od njih je provera čitanja grafa izborom početnog čvora i obilaskom grafa. **Obilaskom i čitanjem oznaka i veza** možemo doći do zaključka da li predviđena struktura ima smisla. U ovom slučaju moramo razumeti i potrebe korisnika, kako bi se struktuirali i proverili upiti koji se šalju bazi. Razumevanjem modela koji definišemo i definisanjem upita dobijamo uvid u način na koji će oni biti izvršavani i nivo njihove kompleksnosti.

Programski okvir za opisivanje resursa

Postoji i drugačiji način za modeliranje grafovskih baza podataka i on se zove **Programski okvir za opisivanje resursa** (engl. *Resource Description Framework*). RDF predstavlja standard za modeliranje Web podataka. On je kreiran za čitanje i razumevanje od strane kompjutera, za razliku od Grafa označenih svojstava koji je predviđen za ljude. Ovaj model identifikuje stvari korišćenjem Web identifikatora odnosno URI-ja[4] i opisuje ih pomoću svojstva i vrednosti atributa.

Kao graf, RDF sadrži čvorove i označene usmerene potege (veze) koji povezuju parove čvorova, a predstavljeni su kao trojka (engl. *triple*): **čvor subjekat, predikat i čvor objekat**. Čvorovi mogu biti RDF URI reference, RDF literali ili prazni čvorovi. Predikati su RDF URI reference i mogu se interpretirati ili kao veza između dva čvora ili kao definisane vrednosti atributa za neke subjekte čvora.

Ovim modelom omogućeno je da se struktuirani i polustruktuirani podaci mešaju, prikazuju i dele kroz različite aplikacije.

RDF model koristi se za razmenu podataka, dok je Graf označenih svojstava fokusiran na smeštanje podataka i brzo izvršenje upita. **RDF model ima značajnu ulogu u semantičkom web-u.**

[4] *URI - Uniform Resource Identifier* - String karaktera koji jedinstveno identifikuje određeni resurs

Neo4j indeksi

Neo4j može koristiti indekse za pretraživanje u cilju pronalaženja čvorova i relacija na kojima će primeniti određene operacije. Indeksi se koriste kako bi obezbedili jedinstvenost vrednosti pojedinih svojstava. Iako postoji mogućnost kreiranja indeksa, njihova upotreba zavisi od projektovane baze, njene veličine i načina procesiranja. Neo4j je projektovana kao *native graph-first* baza i može funkcionisati bez indeksiranja, što je opisano u jednom od prethodnih poglavlja. [*Native i Non-native graf baze*]

Cypher omogućava **kreiranje indeksa za jedan ili više atributa za svaki označeni čvor** i to jednovrednosni indeks (engl. *single-property index*) i složeni indeks (engl. *composite index*). Dobra praksa je imenovanje indeksa, koje mora biti jedinstveno u skupu svih indeksa i ograničenja. Ukoliko se ne dodeli ime indeksu pri kreiranju ono će biti automatski generisano.

Pretrage ne zahtevaju indekse, ali njihovim dodavanjem mogu biti poboljšane u određenim slučajevima. U nastavku su prikazane komande koje ilustruju prethodno navedene karatkeristike.

```
CREATE INDEX a_ime FOR (a:Alumni) ON (a.ime)
ili
CREATE INDEX a_ime ON :Alumni(ime)
```

Kreiranje indeksa za Alumni članove, koji se zasniva na njihovom atributu “ime”.
Indeks se može ukloniti navođenjem imena ili kombinacije oznake i atributa:

```
DROP INDEX a_ime
ili
DROP INDEX ON :Alumni(ime)
```

Glavna primena indeksa kod graf baze je u nalaženju početne tačke za obilazak grafa, jer se na dalje obilazak oslanja na strukturu grafa za visoke performanse. Indeksi mogu biti dodati u bilo kom trenutku i oni se automatski koriste prilikom slanja upita.

Složeni indeksi obuhvataju više atributa svih čvorova sa određenom oznakom.
Na primer

```
CREATE INDEX ON :Alumni(ime, email)
```

kreiraće složeni indeks za svaki Alumni čvor koji ima atribute “ime” i “email”. Ukoliko neki čvor ne bude imao jedan od ova dva atributa on neće biti indeksiran u ovom slučaju.
Za proveru definisanih indeksa u bazi možemo koristiti ugrađenu proceduru

```
CALL db.indexes;
```

Prethodno opisani standardni indeksi su zapravo **B-tree[5] indeksi** koji se mogu vući iz dva izvora *native-btree-1.0* sa ograničenjem od 8kB po ključu i *lucene+native-3.0* koji pruža 32kB kao memorijski limit za ključeve. Podrazumevano podešavanje koristi *native-btree-1.0* za B-tree indekse.

Posebna vrsta indeksiranja je moguća za **potpuno tekstualne indekse** (engl. *full-text indexes*). Podrazumevano se koristi *Apache Lucene*, *open-source* biblioteka koja obezbeđuje mehanizme za indeksiranje i pretraživanje podataka. Potpuno tekstualni indeksi omogućavaju da se pretraga vrši po sadržaju vrednosti stringa indeksiranog atributa. B-tree indeksi koji su opisani na početku vrše tačno podudaranje (engl. *exact matching*) ili podudaranje prefiksa stringova pri pretrazi, dok tekstualni indeksi tokenizuju termine u okviru indeksiranog atributa.

Potpuno tekstualni indeksi:

- podržavaju indeksiranje čvorova i veza;
- podržavaju konfiguraciju dodatnih analizatora, van *Lucene* biblioteke;
- mogu da budu pretraživani Lucene upitnim jezikom;
- mogu prikazati score za svaki rezultat upita;
- automatski se ažuriraju, sa svakom promenom čvorova i veza;
- može im se pristupiti iz Cypher procedura;
- mogu se konfigurisati tako da budu eventualno konzistentni upotrebom pozadinske niti za ažuriranje indeksa.

Nasuprot B-tree indeksima, potpuno tekstualni indeksi mogu biti primenjeni na više od jedne oznake ili tipova veza, ali i više atributa. B-tree složeni indeksi odnose se samo na objekte koji se podudaraju sa indeksiranom oznakom i svim indeksiranim atributima, a potpuno tekstualni indeksi primenjuju se na objekte koji imaju bar jednu od indeksiranih oznaka ili tipa veze, i bar jedan od indeksiranih atributa.

Funkcije koje se koriste za kreiranje potpuno tekstualnog indeksa za čvor ili vezu su date u nastavku:

```
db.index.fulltext.createNodeIndex

db.index.fulltext.createRelationshipIndex
```

Obe komande imaju opcioni *config* parametar koji može omogućiti dodatna podešavanja i to:

- **analizator** (engl. *analyzer*)
  Kojim se bliže određuje analizator za indeksiranje i pretragu podataka. Komanda db.index.fulltext.listAvailableAnalyzers prikazuje listu dostupnih analizatora.
- **eventualna konzistentnost** (engl. *eventually consistent*)
  Koja omogućava da se ažuriranja u transakcijama vrše u pozadinskim nitima ukoliko je podešena na ‘true’

[5] *B-tree* - struktura podataka koja predstavlja binarno stablo sa sortiranim čvorovima.

Kreiranje potpuno tekstualnog indeksa za Alumni članove koji se zasnivaju na njihovom atributimu "email" i Fakultete koji imaju "email" i "naziv":

```
CREATE (Alumni1:Alumni {ime:'Marija', datum_rodjenja:'1960-01-20',
pol:'Z', prebivaliste:'Niš', email:'marija@itc.rs',
telefon:'+38160123456', prezime:'Marković'})
CREATE (Fakultet1:Fakultet {naziv:'Elektronski fakultet',
univerzitet:'Univerzitet u Nišu', adresa:'Aleksandra Medvedeva
14', telefon:'+381 18 529105', email:'efinfo@elfak.ni.ac.rs' })

CALL db.index.fulltext.createNodeIndex("EmailiNaziv",
["Alumni", "Fakultet"], ["email", "naziv"])
```

U ovom slučaju potpuno tekstualni indeks je kreiran i za Alumni člana i za Fakultet, iako nemaju oba atributa nad kojima je kreiran indeks.
Lucene dozvoljava i upotrebu logičkih operatora AND i OR:

```
CALL db.index.fulltext.queryNodes("EmailiNaziv", 'elektronski
AND fakultet') YIELD node, score
RETURN node.naziv, score
```

Ovaj upit vraća Elektronski fakultet, jer on u nazivu jedini ima kombinaciju 'Elektronski' i 'fakultet' kako smo naveli u upitu. Takođe, moguće je eksplicitno navesti prema kom atributu želimo da izvršimo pretragu, navođenjem ispred njegove vrednosti:

```
CALL db.index.fulltext.queryNodes("EmailiNaziv",  'naziv:
"Elektronski fakultet"') YIELD node, score
RETURN node.naziv, score
```

Neo4j ograničenja

Za jedinstvene vrednosti atributa mogu se definisati **ograničenja** (engl. *constraint*) koja mogu biti imenovana ili ne, slično kao kod indeksa. Sledeće komande ilustruju pravljenje ograničenja kome će naziv biti automatski generisan ili eksplicitno dodeljen:

```
CREATE CONSTRAINT ON (k:Katedra) ASSERT k.naziv IS UNIQUE
ili
CREATE CONSTRAINT nazivKatedre ON (k:Katedra)
ASSERT k.naziv IS UNIQUE
```

Kreiranje ograničenja za atribut "naziv" čvorova "Katedra" kako bi bili jedinstveni. Dodavanje UNIQUE organičenja implicitno će indeksirati atribut za koji je dodato, ali ukoliko se ograničenje ukloni biće uklonjen i implicitno stvoreni indeks.

Kroz ograničenja postavljamo različita pravila u našem domenu, a kako bismo pogledali definisana ograničenja u bazi koristi se ugrađena metoda

```
CALL db.constraints;
```

Ograničenja mogu biti definisana i za atribute čvorova i za atribute veza. Sledeći primer ilustruje EXISTS ograničenje koje obavezuje da navedeni atribut uvek postoji za dat tip veze.

```
CREATE CONSTRAINT tipStudija
ON ()-[P:POHADJAO_JE]-()
(k:Katedra)
ASSERT EXISTS (P.tip_studija)
```

Postoji i ograničenje NODE KEY koje osigurava da svi čvorovi sa datom oznakom imaju skup definisanih atributa čija je kombinacija vrednosti jedinstvena i svi atributi su prisutni u skupu.

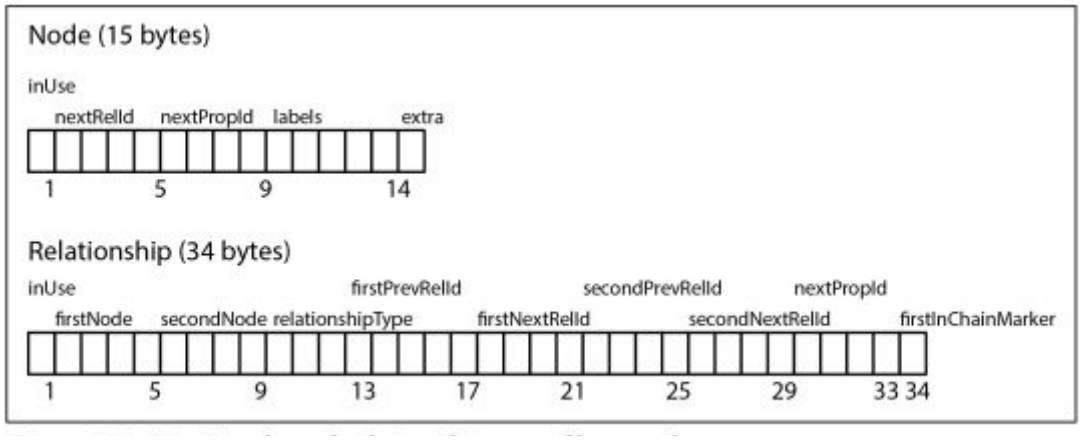
```
CREATE CONSTRAINT fakultet
ON (f:Fakultet)
ASSERT (f.naziv, f.adresa, f.univerzitet) IS NODE KEY
```

Definisana ograničenja mogu biti uklonjena komandom:

```
DROP CONSTRAINT naziv_ograničenja
```

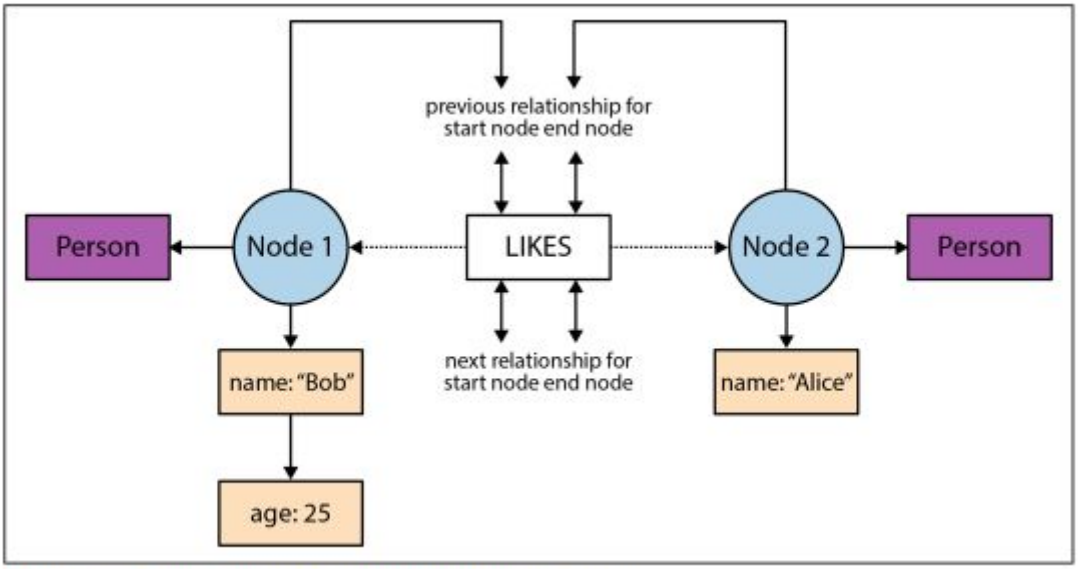
Skladištenje podataka

Kod Neo4j graf baze svaki fajl sadrži podatke o nekom delu grafa, odnosno graf se smešta u delovima u više različitih fajlova. Memorijska struktura je prikazana na Slika7.



***Slika7**. Fajlovi za skladištenje Neo4j čvora i veze*

Svaki čvor koji se kreira na korisničkom nivou grafa smešta se u **fajl za čuvanje čvora** (engl. *node store file*). Fiksna veličina fajla omogućava bržu pretragu čvovora u fajlu *neostore.nodestore.db*, jer sistem može da preračuna memorijsku lokaciju na osnovu ID-ja i to složenošću O(1). Slično tome, veze se smeštaju u **fajl za čuvanje veza** (engl. *relationship store file*) i fiksne su veličine. Svaki fajl sadrži ID početnog i krajnjeg čvora za datu vezu. Kod veza postoje pokazivači na dvostruko lančane liste koje predstavljaju veze početnog i krajnjeg čvora, što omogućava brz dvosmerni obilazak i efikasno dodavanje ili brisanje veza.



***Slika8**. Fizičko smeštanje Neo4j grafa u delovima*

Na sličan način čuvaju se i atributi u fajlu *neostore.propertystore.db*, koji su takođe fiksne veličine. Svi atributi nalaze se u jednostruko lančanoj listi tako da svaki atribut ukazuje na sledeći preko ID-ja.

Praktični primeri pri modeliranju graf baze

Jedno od pitanja sa kojim se najranije susrećemo pri modeliranju je da li je nešto atribut čvora ili veza sa drugim čvorom. Za korišćen primer to može biti pitanje za kompanije. Da li je potrebno čuvati kompanije kao zasebne čvorove ili kao jedan atribut u čvoru Alumni? Jednostavnim upitom možemo dobiti kompaniju u kojoj konkretan alumni radi, ali izgubićemo druge podatke koje nam veza "RADI_U" omogućava, kao što je na primer koji to sve alumni rade u određenoj kompaniji i koliko ih ima.
U nastavku su ilustrovani upiti u jeziku Cypher za slučaj da je alumni član modelovan sa atributom kompanija, odnosno:

```
CREATE (Alumni1:Alumni {ime:'Marija', datum_rodjenja:'1960-01-20',
pol:'Z', prebivaliste:'Niš', email:'marija@google.com',
telefon:'+38160123456', prezime:'Marković', kompanija:'Google'})
```

Pronalaženje kompanije za konkretnog alumni člana:

```
MATCH (a:Alumni {ime: "Marija", prezime: "Marković"})
RETURN a.kompanija;
```

Pronalaženje svih alumni članova koji rade u istoj kompaniji:

```
MATCH (a1:Alumni), (a2:Alumni)
WHERE any(x IN a1.kompanija WHERE x IN a2.kompanija)
AND a1 <> a2
RETURN a1, a2;
```

Umesto ovakvog načina modelovanja, kompanije i alumne možemo staviti u zasebne čvorove kao što je inicijalno predloženo i povezati usmerenim potegom.

```
CREATE (Alumni1)-[:RADI_U]->(Kompanija1)
```

U ovom slučaju upiti koji su predstavljeni na određen način u prethodnom delu, sada su prikazani na drugačiji način u nastavku.
Pronalaženje kompanije za konkretnog alumni člana:

```
MATCH   (a:Alumni {ime: "Marija", prezime: "Marković"}),
           (a)-[:RADI_U]->(k:Kompanija)
RETURN k.naziv;
```

Pronalaženje svih alumni članova koji rade u istoj kompaniji:

```
MATCH   (a1:Alumni)-[:RADI_U]->(k:Kompanija),
        (a2:Alumni)-[:RADI_U]->(k)
RETURN a1, a2, k;
```

Oba prethodno opisana načina modeliranja su prihvatljiva i zavise isključivo od tipova upita kakvi će biti izvršavani nad našom bazom. Zato je glavni savet pri modelovanju da se krene od upita. Ukoliko znamo na koja ćemo pitanja davati odgovore, to je odlična tačka za početak odlučivanja o strukturi modela podataka. Prioritizacijom upita odnosno pitanja koja smatramo da će biti postavljana, bliže možemo odrediti kakav model podataka želimo da implementiramo.

Optimizacija upita

Cilj manualne optimizacije upita je **poboljšanje performansi baze** i osiguravanje da se samo neophodni podaci vrate iz grafa. Podaci koji su nam potrebni bi trebalo da se filtriraju što ranije moguće kako bi se izbeglo gomilanje posla pri izvršenju upita.
Svaki Cypher upit se optimizuje i transformiše u **plan izvršenja** uz pomoć Cypher planer upita. Postoje dve opcije Cypher optimizacije koje se mogu koristiti i to:

- **EXPLAIN**
  Opcija koja prikazuje plan izvršenja bez pokretanja upita. Nakon pokretanja upita sa ovom naredbom baza ostaje nepromenjena.
- **PROFILE**
  Ova opcija određuje koji operatori upita imaju najveće opterećenje, vodeći računa o broju čvorova/veza kroz koje prolazi operator i o interakciji sa memorijom.

```
PROFILE
MATCH (a {ime: "Marija"})
RETURN a;
```

```
+----------------+----------------+------+---------+----------------+------------------+---------
------------+-----------+--------------------------+
| Operator       | Estimated Rows | Rows | DB Hits | Page Cache Hits | Page Cache Misses | Page
Cache Hit Ratio | Variables | Other                    |
+----------------+----------------+------+---------+----------------+------------------+---------
------------+-----------+--------------------------+
| +ProduceResults |             16 |    1 |       0 |              0 |                 0 |
0.0000 | a         |                          |
| |
+----------------+------+---------+----------------+------------------+----------------------+----
-------+--------------------------+
| +Filter         |             16 |    1 |     23 |              0 |                 0 |
0.0000 | a         | a.ime = $`  AUTOSTRING0` |
| |
+----------------+------+---------+----------------+------------------+----------------------+----
-------+--------------------------+
| +AllNodesScan   |           23 |  23 |     24 |              0 |                 0 |
0.0000 | a         |                          |
```

U slučaju navedenom iznad primećuje se upotreba *AllNodesScan* operatora, što znači da su pretraženi svi čvorovi iz baze da bi pronašli Alumni člana sa imenom Marija. Što čini ovaj proces neefikasnim, jer već znamo da su nam potrebne Marije koje predstavljaju Alumni

članove. Iz tog razloga možemo jednostavnim eksplicitnim definisanjem oznake “Alumni” poboljšati ovu pretragu. U ovom slučaju umesto *AllNodesScan* pokrenuće se operator *NodeByLabelScan*. Ukoliko postoji i indeks za promenljivu prema kojoj vršimo pretragu operator *NodeIndexSeek* će znatno uštedeti resurse i ubrzati proces traženja.

Predikati koji se mogu koristiti za omogućavanje napredne optimizacije su:

- Postojanje (`WHERE exists(alumni.ime)`)
- Jednakost (`WHERE alumni.ime = 'Marija`)
- Opseg (`WHERE alumni.uid > 1000 AND alumni.uid < 2000`)
- Prefiks (`WHERE fakultet.naziv STARTS WITH 'Elektronski'`)
- Sufiks (`WHERE katedra.naziv ENDS WITH 'informatika'`)
- Podstring (`WHERE katedra.naziv CONTAINS 'informatika'`)
- Kombinacija predikata operatorom **OR** nad istim atributom (`WHERE fakultet.naziv STARTS WITH 'Elektronski' OR fakultet.naziv CONTAINS 'Elektronski'`)

Zaključak

U radu je predstavljen i bliže okarakterisan način modeliranja graf baza podataka sa osvrtom na internu organizaciju podataka u Neo4j bazi. Kroz poglavlja su opisani načini modeliranja i dobre prakse kod graf baza podataka i slučajevi korišćenja za karakteristične primere, kao i poređenje relacionih i Neo4j graf baza. U ovom radu za praktični primer korišćena je projektovana baza za alumni članove fakulteta, koja je prikazana kroz ER dijagram i Neo4j graf rešenje.

Neo4j je *native* graf baza podataka i ona efikasno implementira strukturu grafa do nivoa skladištenja podataka. Ovo znači da se podaci čuvaju baš onako kako su logički modelovani, a za navigaciju kroz graf i obilaske na osnovu određenih upita koriste se pokazivači unutar svakog čvora i veze. Model podataka u Neo4j bazi nije striktan, odnosno ona je *schema-free*, što znači da lako može evoluirati i implementirati promene kroz vreme što je čini superiornijom u odnosu na relacione baze pogotovu ako govorimo o velikim kolicinama podataka.

Graf baze podataka mogu biti projektovane kroz graf označenih svojstava koji je predviđen za ljudsko procesiranje ili RDF koji omogućava procesiranje web podataka i doprinosi obradi podataka u semantičkom web-u. U ovom radu fokus je bio na ljudskom projektovanju kroz čvorove, veze, atribute i oznake za konkretno rešenje Neo4j.

Pored indeksiranja i uvođenja ograničenja radi optimizacije baze podataka, pomenuta je i karakteristika *native* graf baza *index-free adjacency*, a obrađene su i metode analize opterećenja pri pisanju upita i njihove optimizacije.

Literatura

[1] I.Robinson, J. Webber, E. Eifrem, Graph Databases, 2nd edition, O'Reilly Media
[2] Dr. J. Graovac, Projektovanje baza podataka, Matematički fakultet, Univerzitet u Beogradu 2016.
[3] B. M. Sasaki, Data modeling basics, Neo4j Blog
https://neo4j.com/blog/data-modeling-basics/
[4] B. M. Sasaki, Data modeling pitfalls, Neo4j Blog
https://neo4j.com/blog/data-modeling-pitfalls/?ref=blog
[5] Modeling designs, Neo4j Developer
https://neo4j.com/developer/modeling-designs/
[6] Modeling tips, Neo4j Developer
https://neo4j.com/developer/modeling-tips/
[7] Neo4j overview, Tutorials point
https://www.tutorialspoint.com/neo4j/neo4j_overview.htm
[8] J. Chao, Native vs Non-native graph technology, Neo4j Blog
https://neo4j.com/blog/native-vs-non-native-graph-technology/
[9] Concept: Relational to graph, Neo4j Developer
https://neo4j.com/developer/graph-db-vs-rdbms/
[10] J. Barrasa, RDF Triple store vs Labeled Property Graphs
https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/