UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET
KATEDRA ZA RAČUNARSTVO

Interna struktura i organizacija skladišta Neo4j baze podataka

Predmet: Sistemi za upravljanje bazama podataka

Mentor:

doc. dr Aleksandar Stanimirović

Student:

Krstić Katarina, broj indeksa: 1400

SADRŽAJ

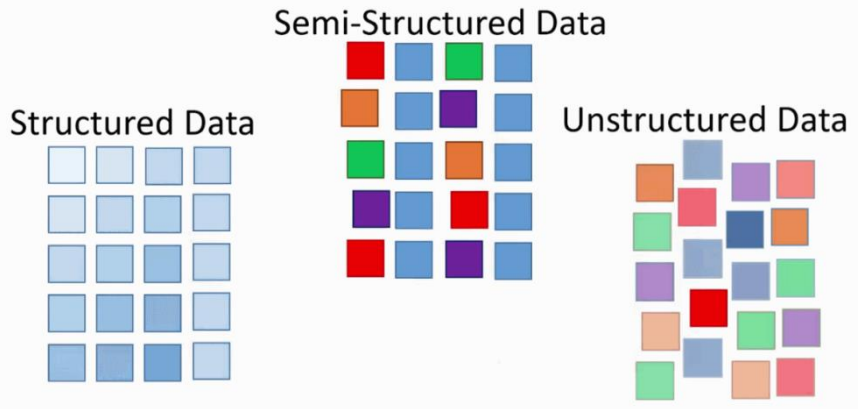
1. UVOD

Do relativno nedavno se tehnologija suočavala u radu isključivo sa struktuiranim podacima, koji su vrlo jednostavni za analizu, čitljivi i upravo formatirani tako da se mogu lako pretraživati, upoređivati i proširivati. Međutim, pojava polustruktuiranih, a zatim i nestruktuiranih podataka dovela je i do problema koji se tiču načina rada sa ovakvim podacima, ali i načinima skladištenja ovakvih podataka.

Za početak, polustruktuirani podaci su uglavnom tekstualni podaci organizovani u određene kategorije. Vrlo jednostavno se ovakvi podaci mogu organizovati spram tih kategorija, ali su podaci unutar tih kategorija nestruktuirani. Primer ovakve grupe podataka su mejlovi, koje vrlo lako možemo svrstati u kategorije: *Sent, Inbox, Draft* i tako dalje, ali sam e-mail tekst nema neku striktnu definisanu strukturu.

Nestruktuirani podaci, kao što i sam naziv nalaže, nisu struktuirani prema nekim parametrima. To su najčešće tekstualni podaci, poput konverzacije na socijalnim mrežama, na primer, ali mogu biti i kompleksnije vrste podataka poput fotografija, video ili audio snimaka.

Upravo zbog svoje kompleksnosti po pitanju strukture, ovi podaci doveli su do niza pitanja, od kojih su se kao glavna nametala pitanja načina skladištenja ovakvih podataka. Do tada su poznati bili samo relacioni modeli koji su strogo šematski orijentisani. U relacionim modelima koriste se tabele sa redovima i kolonama i pritom svaki red u tabeli ima istu strukturu. Za razliku od ovakvih modela, polustruktuirani i nestruktuirani podaci ne moraju imati istu strukturu kao neki drugi podaci vrste slične njihovoj.



Slika1: Struktuirani, polustruktuirani i nestruktuirani podaci

Kao odgovor na pojavu ovakvih podataka javile su se *NoSQL* baze podataka. Naime, svaka od baza podataka koja pripada ovoj grupi razvijena je s težnjom da reguliše određeni problem koji je uviđen kod relacionih, a pritom da reši i problem skladištenja i nestruktuiranih podataka.

2. NOSQL(**N**OT **O**NLY **SQL**)

Akronim *NoSQL* prvi put upotrebio je *Carlo Strozzi* 1998. godine prilikom definisanja *open source* relacione baze podataka koja nije koristila *SQL*. Ovaj termin javio se zatim ponovo 2009. godine, kada su *EricEvans* i *JohanOskarsson* težili da definišu nerelacione sisteme. Ne bi li se naglasila sposobnost ovakvih sistema da podrže i neke *SQL-likequery* jezike, opšte prihvaćeno značenje termina *NoSQL* je ***NotOnlySQL***.

Web podaci, potreba za obradom nestruktuiranih podataka i, svakako neizostavno, potreba za bržom obradom istih bili su ključni elementi koji su i bili glavni razlog za razvoj *NoSQL* sistema. Nerelacioni sistemi koriste *ad-hoc* pristup kada je organizovanje podataka u pitanju i imaju mogućnost procesiranja velike količine podataka različitih struktura velikom brzinom. Samim tim, *NoSQL* baze podataka bile bi očigledniji izbor u odnosu na relacione baze podataka kada je potrebno obraditi veliku količinu nestruktuiranih podataka.

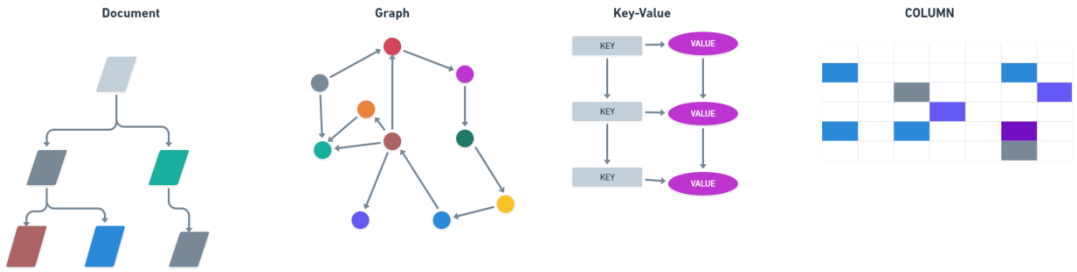
Svakako presudnu ulogu u razvoju i upotrebi *NoSQL* baza podataka imala je i njihova sposobnost ne samo da obrade velike količine struktuiranih ili nestruktuiranih podataka, već i podataka kakve nazivamo *BigData*. Upravo ova sposobnost bila je ključ koji je otvorio vrata masovne upotrebe *NoSQL* baza u velikim kompanijama kakve su *Facebook*, *Twitter*, *Linkedin* i *Google*.

NoSQL baze podataka osiguravaju dostupnost, brzinu i particionisanje podataka, međutim većina njih vrši kompromis po pitanju konzistentnosti podataka. Koncept kojeg se drže ovakve grupe baza podataka nazivamo *eventual consistency*, a njegova suština leži u ideji da se svim čvorovima promene nastale u bazi dostave u nekom trenutku. To može dovesti do toga da čitanje podataka može vratiti neosvežene podatke.

Postoji četiri različitih karakterističnih struktura podataka koje *NoSQL* baze podataka koriste i sve one se smatraju izuzetno fleksibilnim. Pritom, u potpunosti se koncept organizacije podataka u svakoj od ovih grupa razlikuje od koncepta organizacije podataka koji koriste relacione baze podataka.

Strukture podataka koje *NoSQL* baze podataka koriste su:

- *Key-value store*
- *Wide column store*
- *Document store*
- *Graph store*



Slika2: NoSQL baze podataka

Osim činjenice da se koncept organizacije podataka u *NoSQL* bazama podataka razlikuje od koncepta organizacije podataka u relacionim bazama podataka, takođe se i organizacija svake od ovih pojedinačnih grupa u svojoj srži i suštini potpuno razlikuje od preostalih grupa. Samim tim, pogodnost upotrebe neke od ovih vrsti baza podataka usko se vezuje i za problem za koji bi se takva baza koristila kao rešenje.

3. GRAF BAZE PODATAKA

Podaci kao takvi imaju neprocenjivu vrednost i važnost, međutim, neizostavna je činjenica da su podaci među sobom itekako povezani i da su veze između podataka gotovo i jednako važne kao i sami podaci. Relacione baze podataka imaju sposobnost da skladište veze između podataka, međutim operacija poput *JOIN* su jako skupe. Osim toga, sam koncept spram koga se u relacionim modelima veze među podacima realizuju je dosta kompleksan. Svaka tabela bi sadržala posebno polje koje bi se referenciralo na primarni ključ tabele sa kojom se ona vezuje. I to je slučaj kada su veze među podacima jednostavne. Kada je reč o *N*:*M* ili n-arnim relacijama, u relacionim modelima se zahteva kreiranje nove tabele koja bi takvu relaciju opisala. Vrlo slično, viševrednosni atributi se takođe u relacionim bazama podataka predstavljaju posebnim tabelama. Graf baze podataka razvijene su s ciljem da se većina ovih problema prevaziđe.

Naime, graf baze podataka skladište informacije u vidu čvorova i potega koji te čvorove povezuju i time ostvaruju veze među njima. Skladištenje podataka bez ograničavanja na striktan, strogo definisani model ovaj postupak čini jako fleksibilnim, što se može uvrstiti kao jedna od tri najvažnije prednosti ove grupe baza podataka. Za razliku od relacionih modela, u graf bazama podataka se veze među podacima čuvaju u mnogo fleksibilnijem formatu. Naime, poteg između dva čvora grafa zapravo oslikava vezu između tih entiteta.

Slika3: Grafovi u svetu podataka

Može se reći da su graf baze podataka nastale prvenstveno s ciljem da se pojednostave relacije među podacima u svakom mogućem smislu. Upravo iz tog razloga je rad sa relacijama u ovoj grupi baza podataka maksimalno pojednostavljen u odnosu na bilo koji drugi tip baza podataka. Kao takve, graf baze podataka nude rešenje za jednostavnu i, što je možda još važnije, jeftiniju i bržu navigaciju kroz veze među podacima, otkrivanje skrivenih veza među udaljenim podacima i slično.

Na vrlo jednostavan način možemo izvršiti prelazak sa relacionog modela na graf bazu podataka. Svaka tabela postala bi poseban čvor, dok bi relacije(ili relacijske tabele) postale potezi među čvorovima. To bi nas dovelo do formiranja graf modela podataka. Međutim, ovo svakako nije obavezan način na koji bismo morali da kreiramo jednu ovakvu bazu podataka.

Graf baze podataka možemo definisati kao sisteme u kojima se *CRUD* operacije izvršavaju nad graf modelom podataka. Naime, one ne zahtevaju postojanje striktnog modela po kojem će se kreirati kompletna baza podataka. Naprotiv, svaki od čvorova može imati potpuno drugačiju strukturu, a isto važi i za relacije. Srodne čvorove možemo grupisati u entitete, ali čak ni to nas ne ograničava da broj i vrednosti njihovih atributa moraju biti isti.

Osim pogodnosti koje nude kada je reč o relacijama, graf baze podataka obezbeđuju i promenu poslovne logike i proširivanje modela na jako jednostavan način.

Velike kompanije, kakve su *PayPal, Google, Facebook* i *Linkedin,* koriste tehnologije graf baze podataka jer su upravo i namenjene da se na jednostavan način nose sa vezama između velike količine podataka i da, pritom, na jednostavan način omogućavaju i širenje ovakve mreže. Svakako da se logično nameće i pomisao na to da bi ovakva grupa baza podataka bila najpogodnije rešenje kada se radi o socijalnim mrežama, ali ovo nije jedino mesto gde bi one pronašle svoj smisao i pružile doprinos i optimizaciju. Tako ih *enterprise* organizacije današnjice neretko koriste za detekciju prevara, autentifikaciju i autorizaciju, *recommendation engine*, graf znanja, i slično.

4. NEO4J

Neo4j se izdvaja kao vodeća svetska graf baza podataka i smatra se jednim od najvažnijih predstavnika ove grupe *NoSQL* baza podataka. Samim tim, *Neo4j* podatke skladišti u vidu grafa, odnosno podatke predstavlja kao čvorove, dok veze među njima predstavlja usmerenim potezima između čvorova. Razvijen je 2007. godine i aktivno se unapređuju njegove mogućnosti.

Ono što možda najviše izdvaja *Neo4j* u odnosu na ostale *NoSQL* baze podataka jeste to što obezbeđuje *ACID* svojstva. Naime, *ACID* transakcije podrazumevaju sledeće:

- Atomičnost(***A**tomicity*) – sve operacije tokom jedne transakcije izvršavaju se kao jedna operacija. To znači da će se jednom transakcijom izvršiti i sačuvati sve promene ili, ukoliko bilo koja od njih ne uspe sa izvršavanjem, sve promene nastale u toj transakciji do tada biće poništene
- Konzistentnost(***C**onsistency*) – podaci nakon svake transakcije moraju da budu u konzistentnom stanju
- Izolacija(***I**solation*) – interno stanje jedne transakcije nije vidljivo ostalim transakcijama. To znači da su konkurentne transakcije efektivno serijalizovane
- Trajnost(***D**urability*) – promene nastale uspešnom transakcijom se trajno čuvaju

Kao što je već rečeno većina *NoSQL* baza podataka ne garantuje sva ova svojstva, što se izdvaja kao njihov glavni nedostatak u poređenju sa relacionim bazama podataka. Sa druge strane, *Neo4j* može garantovati i *ACID* svojstva kao i relacione baze podataka, a, pritom, rešava mnoga otvorena pitanja i probleme koje se u relacionim bazama podataka mogu naći.

Relacione baze podataka koriste *SQL query* jezik. Sa druge strane, *Neo4j* nudi jako moćan deklarativni *query* jezik koji je razvijen tako da oponaša engleski jezik i dosta je srodan *SQL*-u. Naime, ovaj jezik naziva se *Cypher* i dizajniran je tako da maksimalno pojednostavi kreiranje i pribavljanje, pre svega, relacija među podacima, a da pritom nema potrebe za upotrebom složenih operacija poput *join*.

4.1. KREIRANJE NEO4J GRAF MODELA I BAZE PODATAKA

Proces kreiranja graf modela podataka je proces u kojem korisnik opisuje konkretan proizvoljni domen pomoću čvorova, veza između njih, atributa i labela. Kompletan postupak dovodi do formiranja konkretne graf baze podataka na kraju, kojom će se definisati i sama struktura skladišta.

4.1.1. OSNOVNI ELEMENTI GRAF MODELA PODATAKA

Da bismo mogli da kreiramo model moramo znati koji su osnovni gradivni elementi sa kojima raspolažemo. Možemo koristiti čvorove, relacije, atribute i labele. Pored njih, koncept šeme u ovom slučaju predstavlja jedan opcioni koncept koji može biti kreiran pre formiranja modela, ali to nije obavezan postupak. Indeksi i ograničenja se u graf modelu podrazumevaju pod konceptom šeme i, kao takvi, oni mogu biti predstavljeni u bilo kojem budućem trenutku egzistiranja same graf baze podataka ili već na samom početku. Indeksi su

tu da poboljšaju performanse samog sistema, dok se *contraint*-i mahom koriste ne bi li se uvela određena pravila u sistem koja se tiču mogućih vrednosti podataka.

Čvorovi

Prvi element koji predstavljamo obično bude čvor. Naime, čvorovi i relacije predstavljaju fundamentalne segmente jednog graf modela, a najjednostavniji vid grafa sadrži samo jedan čvor. U zavisnosti od domena koji opisujemo, čvorovi mogu reprezentovati različite stvari, međutim, najčešće ih koristimo da njima predstavimo osnovne entitete domena. Čvorovi se u domenu vrlo jednostavno i identifikuju. Zapravo je dovoljno pronaći elemente koji u domenu predstavljaju imenice i imaćemo već skup čvorova našeg modela. Tako, entiteti poput osobe, automobila, kompanije, radnika, i slično, predstavljaju čvorove domena od interesa.

Svaki čvor bliže određuju atributi, odnosno *key-value* parovi, koji pružaju više detalja o samom entitetu koji se opisuje. Tako, osoba može imati atribute poput imena, datuma rođenja, adrese, broja telefona, pola i tako dalje. Takođe, nije neophodno ni pobrojavati sve atribute koji neki entitet opisuju, već isključivo one koji su relevantni za neki domen od interesa.

Na primer, ukoliko bi trebalo da izdvojimo čvorove na osnovu informacija: *Studenti Petar i Milan su kolege i obojica studiraju na Elektronskom Fakultetu*, najjednostavnije bi bilo da sagledamo koje su to imenice ili entiteti koji sačinjavaju ovakav domen. Tako, čvorovi bi u ovom slučaju bili *Petar*, *Milan* i *Elektronski Fakultet*.

Slika4: Izdvajanje čvorova

Labele

Labele se u graf modelu koriste da bi srodni čvorovi bili grupisani u setove ili kategorije i definisanje labela najčešće sledi nakon definisanja čvorova. Naime, kada znamo sa kojim čvorovima raspolažemo onda možemo na jednostavan način da ih grupišemo prema karakteristikama i, pre svega, pojmovima koje opisuju. Čvorovi ne moraju biti označeni labelom, ali, ukoliko jesu, svi oni koji su označeni istom labelom, pripadaju istoj kategoriji. Pored toga, jednom čvoru može biti dodeljen i veći broj labela, pri čemu bi to značilo da on pripada većem broju kategorija.

Osim toga što se na ovaj način vrši grupisanje čvorova, pre svega, izvršavanje upita u graf modelu mnogo je jednostavnije kada se koriste labele, jer neretko određene stvari želimo da primenimo isključivo na jednu srodnu grupu podataka.

Labele možemo posmatrati kao imenovane klase objekata u objektno orijentisanom programiranju, s tim da one u ovom slučaju nisu obavezne da bi čvor mogao da egzistira u graf modelu. One su tu kao olakšica, dakle, a ne kao obavezan član, a koriste se jer njima možemo da bliže odredimo pojam koji čvor predstavlja.

Takođe, labele mogu tokom života graf baze podataka da se dodaju ili uklanjaju iz čvora ili relacije i na taj način možemo upotrebiti labelu da njome, na primer, označimo stanje u kojem se čvor nalazi.

U primeru koji je pomenut u postupku izdvajanja čvorova mogli bismo da studente Petra i Milana grupišemo koristeći labelu *Student*, dok bismo Elektronski Fakultet mogli da označimo labelom *Fakultet*. Apropo činjenice da se čvor može označiti većim brojem labela, i čvor Petar i čvor Milan mogu da se opišu labelom *Student*, ali takođe i labelom *Osoba*.

Još je bitno pomenuti i da se u graf modelu svaki poseban entitet predstavlja posebnim čvorom, bez obzira na to što deli labelu sa drugim entitetima. Tako vidimo da i Petar i Milan, iako su oba čvora označeni labelom *Student*, predstavljaju nezavisne, zasebne čvorove.

Slika5: Uvođenje labela

Relacije

Relacije su drugi neizostavni deo graf modela podataka, pored čvorova. One povezuju čvorove među sobom i omogućavaju nam da pronađemo podatke koji su udaljeno povezani putujući od jednog čvora kroz njegove veze do drugog čvora.

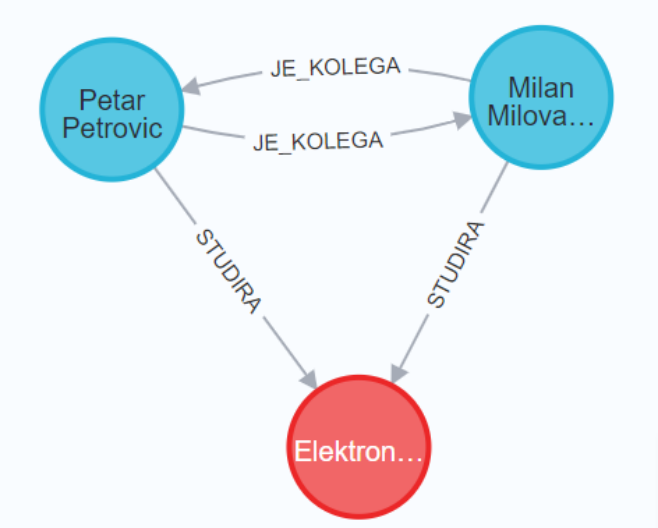
Poteg u grafu je uvek usmeren, što znači da mora imati početni i odredišni čvor. Pored toga, postoje i bidirekcione veze koje je u graf modelu moguće ostvariti. Relacija može imati jedan smer kojim će definisati način na koji su čvorovi povezani, ali to ne ograničava mogućnosti da se u grafu krećemo i u smeru koji je suprotan smeru te relacije.

Jedno od najvažnijih pravila kada je *Neo4j* u pitanju jeste *No broken links*, što znači da nam ni u jednom trenutku nije dozvoljeno da ostvarimo vezu koja nema i početni i krajnji čvor, jer bi to dovelo do situacije u kojoj je veza ostvarena ili ni iz čega ili ni ka čemu ili i jedno i drugo. Samim tim, u *Neo4j*-u nam nije dozvoljeno da obrišemo čvor koji ima ostvarene relacije sa drugim čvorovima. Od nas se tada zahteva da prvo obrišemo sve zavisnosti koje on ima sa preostalim čvorovima kada su veze u pitanju i tek onda će biti omogućeno da se obriše i sam čvor.

Kada je reč o načinu na koji bismo mogli da prepoznamo koje veze među čvorovima postoje, najjednostavnije bi bilo da prepoznamo glagole koji povezuju entitete u domenu.

Ukoliko se sada vratimo na naš primer: *Studenti Petar i Milan su kolege i obojica studiraju na Elektronskom Fakultetu* i izdvojene čvorove: *Petar, Milan* i *Elektronski Fakultet* možemo i jednostavno da zaključimo šta je to što ove čvorove povezuje. Dakle, ukoliko idemo deo po deo doći ćemo i do krajnjeg zaključka. *Studenti Petar i Milan su kolege*. Na osnovu ove tvrdnje zaključujemo:*Petru je kolega Milan* i*Milanu je kolega Petar*. Veza između ovih čvorova je bidirekciona(u oba smera). *Obojica studiraju na Elektronskom Fakultetu*. Ova tvrdnja nam govori sledeće: *Petar studira na Elektronskom Fakultetu*; *Milan studira na Elektronskom Fakultetu*. Ovo su dve nezavisne veze, prva povezuje čvor *Petar* sa čvorom *Elektronski Fakultet* i usmerena je upravo tako, dok druga povezuje čvor *Milan* sa čvorom *Elektronski Fakultet* i usmerena je takođe upravo tako.

Ovo bismo mogli da oslikamo na sledeći način:



Slika6: Definisanje relacija

Jedno veliko pitanje koje se tiče baza podataka gotovo uvek su načini na koji se realizuju relacije i različiti tipovi tih relacija. Naime, poznata je činjenica da postoje *1:1, 1:N* i *N:M* relacije koje možemo ostvariti između različitih entiteta. Kada je reč o *Neo4j*-u, on je i projektovan s ciljem da maksimalno pojednostavi postupak po kojem u njemu funkcionišu, realizuju se i pribavljaju relacije među podacima. Samim tim, on ne pravi razliku između ovih tipova veza. Apsolutno se svaki od ovih tipova veza realizuje na identičan način: *čvor je u vezi sa drugim čvorom*. Takođe, ne postoji potreba za kreiranjem stranih ključeva koji bi to

omogućili, već se jednostavno definiše koji čvor se nalazi sa jedne, a koji sa druge strane relacije.

Kada je reč o *n*-arnim relacijama, *Neo4j* uvodi pojam poznat pod nazivom *hyperedge*, koji se još naziva i *intermmediate node*. Naime, kako *n*-arne relacije povezuju veći broj entiteta, u ovom slučaju veći broj čvorova(više od dva čvora), ova relacija može biti nepotpuna ukoliko se ne reprezentuje na ovakav način.

I, naravno, moguće je da čvor ostvari relaciju sa samim sobom, gde bi, u tom slučaju, isti čvor predstavljao i početni i odredišni čvor relacije.

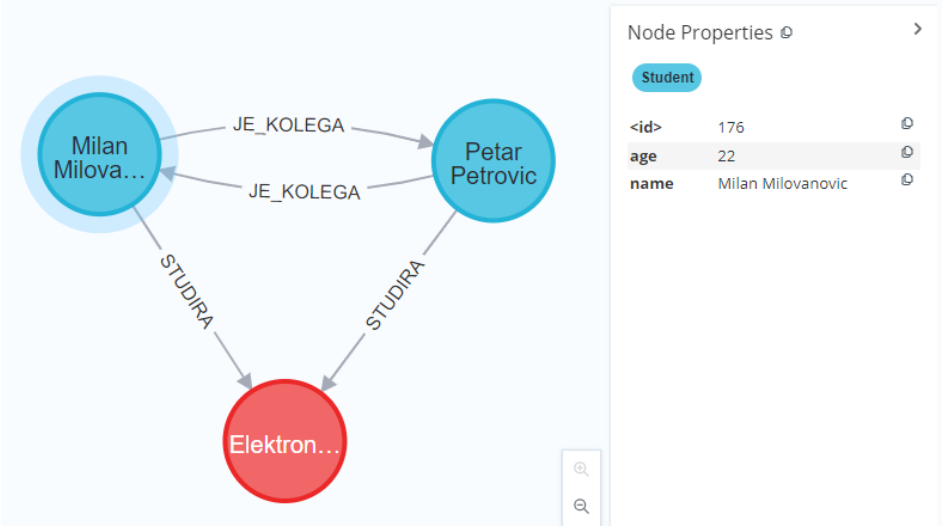
Upravo su relacije ti elementi koji formiraju strukturu u kojoj će se podaci nalaziti.

Za razliku od čvorova koji mogu i ne moraju imati dodeljenu labelu(ili više njih), relacije moraju imati dodeljenu labelu, odnosno tip relacije, da bi se znalo koju to relaciju dva čvora ostvaruju.

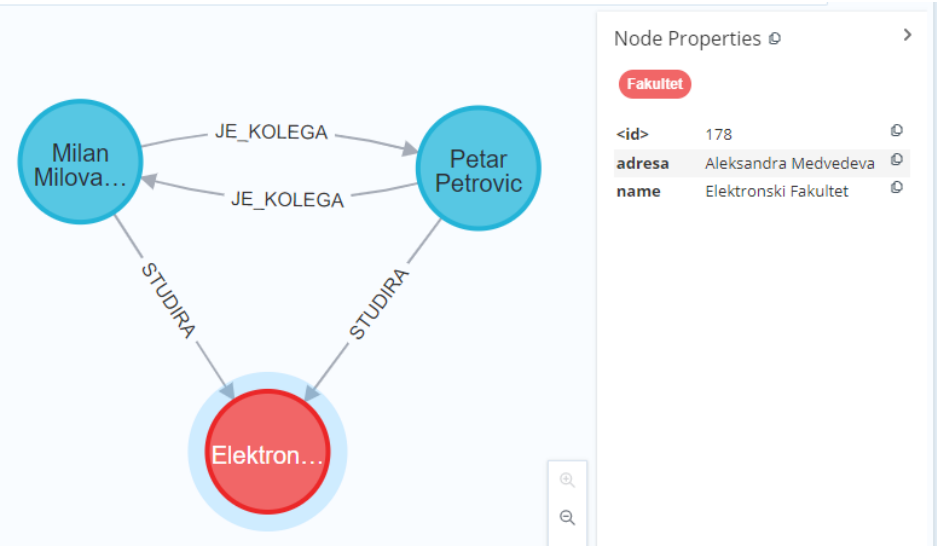
Atributi

Poslednji element koji uključujemo u model su atributi. Atribute može imati čvor ili relacija, a oni se koriste da nam pruže nešto više detalja o samom elementu koji opisujemo. To su parovi naziv-vrednost koji mogu poslužiti za filtriranje podataka i za pružanje nekih ključnih informacija o njima.

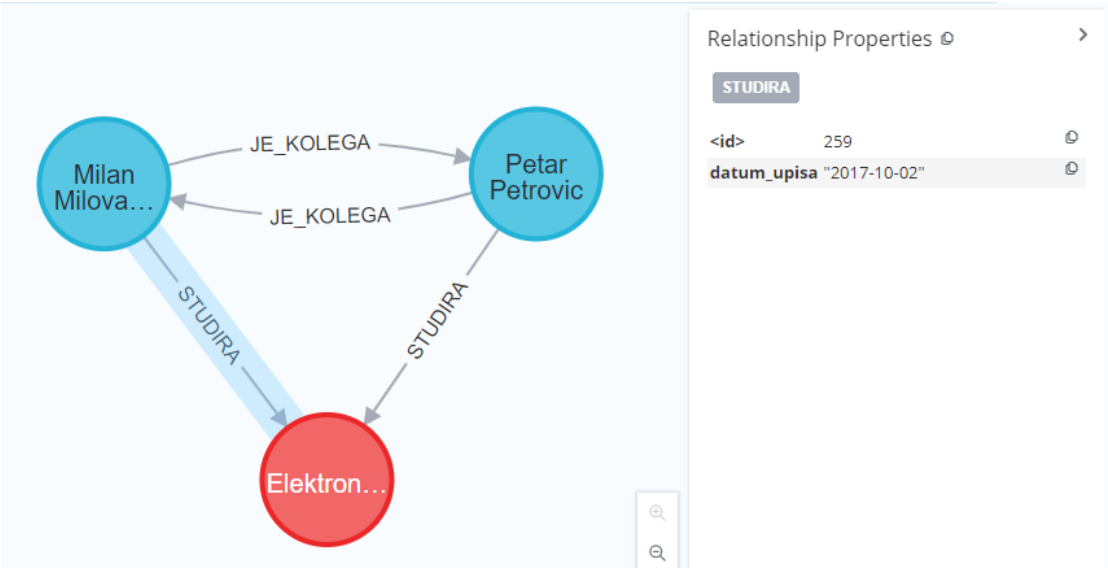
Kako za svaki entitet i relaciju postoji mnoštvo karakteristika koje ih opisuju, neophodno je da na neki način taj broj ograničimo na one koji bi bili relevantni za domen koji opisujemo. To znači da je najjednostavniji način na koji bismo mogli da dođemo do konačnog broja atributa zapravo da postavimo ključna pitanja koja mogu biti od značaja kada je reč o onome što opisujemo. To u ovom slučaju mogu biti, na primer, broj godina koje student ima, godina upisa za relaciju *STUDIRA* i adresa za čvor *Elektronski Fakultet*.



Slika7a: Dodavanje atributa čvoru



Slika7b: Dodavanje atributa čvoru



Slika7c: Dodavanje atributa relaciji

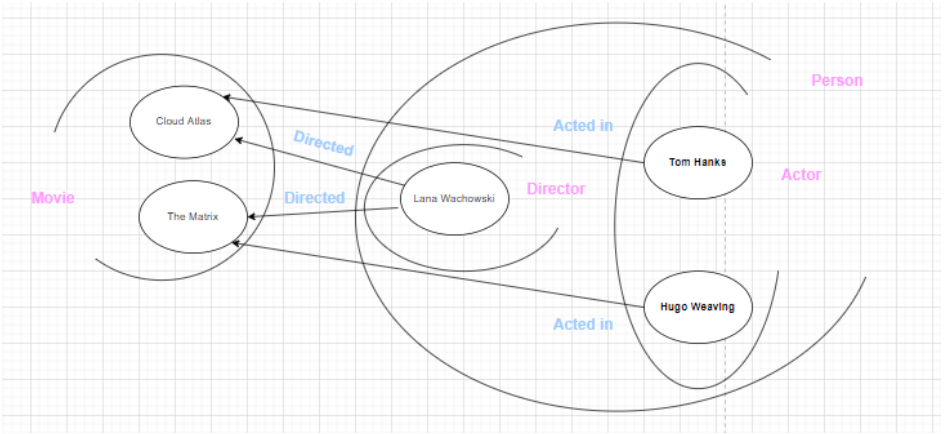
U mnogim slučajevima se dešava da nije tako jednostavno razgraničiti da li bi određeni pojam trebalo da bude novi atribut nekog čvora ili bi trebalo da predstavlja potpuno novi entiet. Ukoliko znamo da *Elektronski Fakultet pripada Univerzitetu u Nišu* i želimo da imamo takav podatak u svom domenu vrlo je tanka nit koja odlučuje da li bi trebalo uvesti novi čvor *Univerzitet u Nišu* i zatim relaciju *PRIPADA* iz čvora *Elektronski Fakultet* ka ovom čvoru ili bi bilo dovoljno uvesti novi atribut u čvoru *Elektronski Fakultet*, čiji bi naziv bio *pripada*, a vrednost *Univerzitet u Nišu*. Zapravo, konačni način koji ćemo odabrati umnogome zavisi prvenstveno od toga da li je ovaj pojam relevantan za domen koji opisujemo. U slučaju da opisujemo niške studente, ovaj pojam bi izgubio smisao kao poseban entitet, dok bi u slučaju da opisujemo srpske studente imalo smisla uvesti ga kao poseban čvor. Dakle, sve zavisi od širine domena i problema koji rešavamo.

4.1.2. POSTUPAK KREIRANJA GRAF BAZE PODATAKA

Čak i prilikom projektovanja relacionog modela, najčešće je prvi korak u procesu iscrtavanje osnovnih entiteta na tabli ili papiru i zatim njihovo povezivanje, ne bi li smo uočili zavisnosti među podacima koje bismo koristili. Potom, tako razvijen model bi se transformisao u normalizovane tabele, strane ključeve i *constraint*-e.

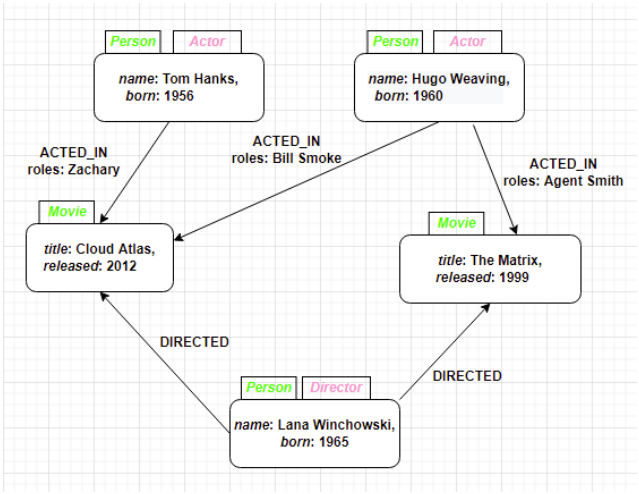
Vrlo slično funkcioniše postupak i za formiranje graf modela. Međutim, za razliku od postupka razvoja relacionog modela, u ovom slučaju razvijeni model ostaje gotovo potpuno isti kao na tabli.

Dakle, redosledom koji je opisan u prethodnom poglavlju izdvajamo čvorove, njihove labele, relacije među njima i eventualno atribute ukoliko su neophodni. Sve ove elemente povezujemo i smeštamo na tablu na kojoj započinjemo sa razvojem svog graf modela. Ovaj način implementacije modela olakšava mogućnost da i samim klijentima, koji možda i nisu upoznati sa radom same baze podataka i načina na koji ona funkcioniše, objasnimo o čemu se tu zapravo radi. Osim toga, ovakav model mnogo je jednostavnije proširiti po potrebi.



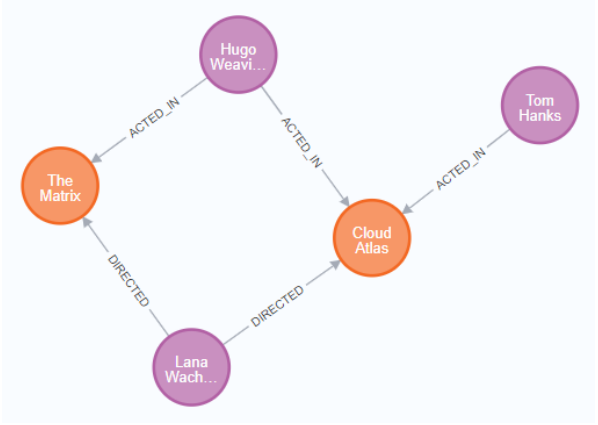
Slika8a: Prvi korak u kreiranju graf baze podataka

Onda kada imamo sve neophodne elemente smeštene na papir ili tablu, možemo ih prilagoditi sintaksi koja bi zadovoljila potrebe *query*-a same baze podataka. Tako, ovaj model nešto bolje organizujemo i formatiramo.



Slika8b: Drugi korak u kreiranju graf baze podataka

Poslednji korak u modelovanju jeste da ovakav graf kreiramo u *Neo4j*-u koristeći odgovarajući *query* jezik. Rezultat koji bismo dobili bio bi poput:

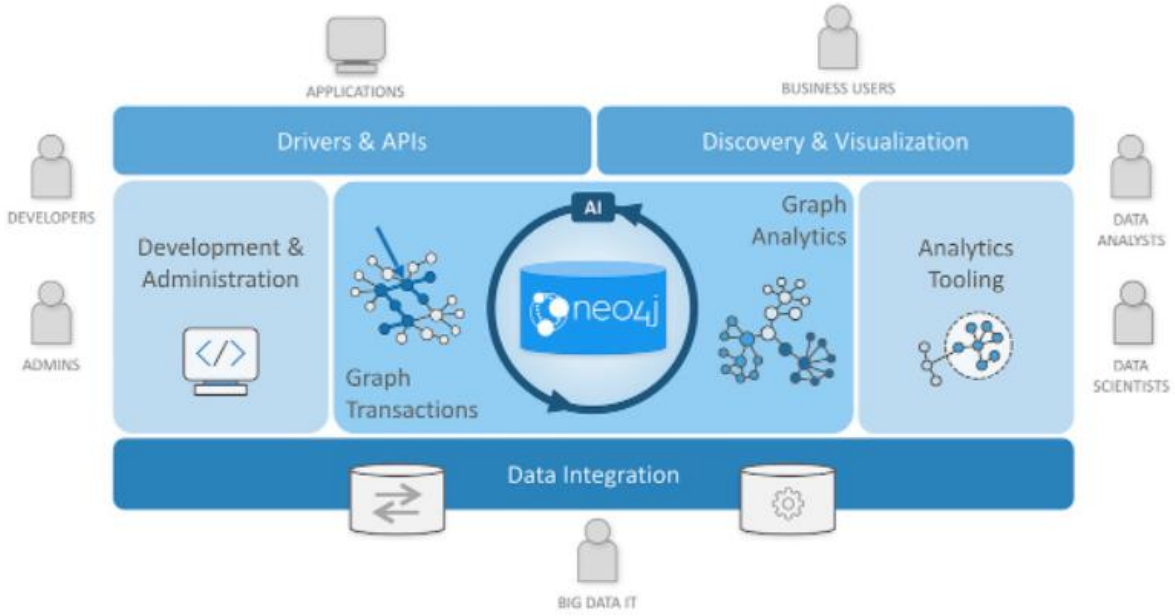


Slika8c: Treći korak u kreiranju graf baze podataka

Upravo i zbog svoje jednostavnosti kada je modelovanje u pitanju, *Neo4j* i generalno graf baze podataka zauzimaju veoma važno mesto u svetu baza podataka, jer ne postoji potreba niti za formiranjem *ER* dijagrama koje bismo potom transformisali u relacioni model, a potom i u tabele, već je gotov model potpuno identičan onom koji predstavimo u početnom koraku. Samim tim, i kompletna biznis logika i planovi koji se mogu predstaviti i osobama koje nisu upoznate sa radom baza podataka mnogo su lakše u ovakvim grupama baza podataka.

4.2. NEO4J GRAF PLATFORMA

Platforma koju *Neo4j* nudi je raznolika i svaki segment ove platforme dizajniran je tako da može da zadovolji poslovne potrebe koje se mogu javiti kada je upotreba graf baze podataka u pitanju.



Slika9: Neo4j graf platforma

Postoji puno načina, dakle, na koje je moguće upotrebiti podatke koji su organizovani u graf.

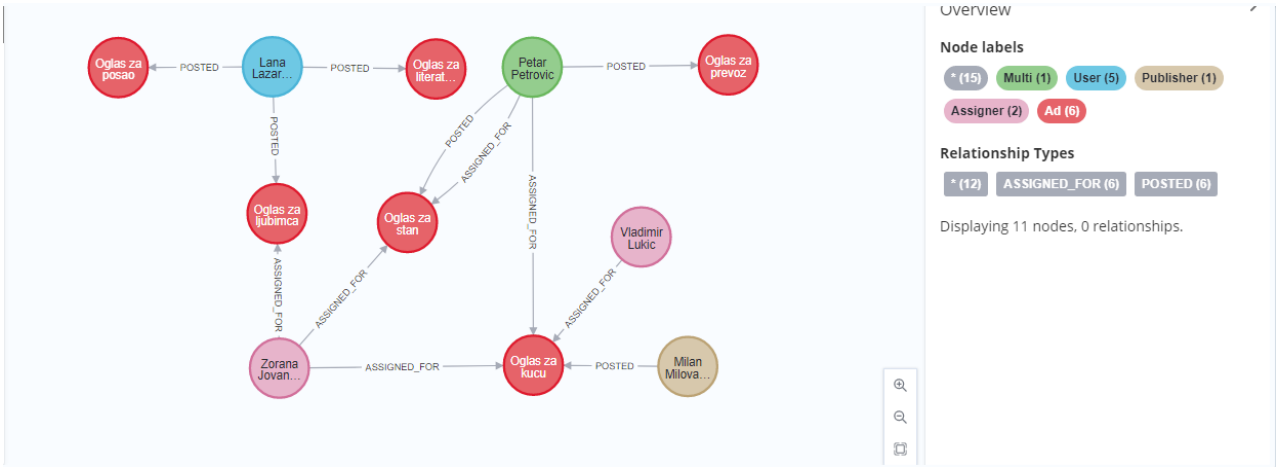
- *Neo4j Graph Database* – sama baza podataka iz koje je moguće pribaviti ili u koju je moguće skladištiti podatke
- *Neo4j Desktop* – desktop aplikacija koja se može koristiti za kreiranje lokalne graf baze podataka
- *Neo4j Browser* – *onlinebrowser* interfejs koji se može koristiti za izvršavanje upita nad *Neo4j* bazom podataka
- *Neo4j Bloom* – alat za vizuelizaciju namenjen prvenstveno za analizu podataka
- *Neo4j AuraDB* – *database-as-a-service* rešenje
- *Graph Data Science* – biblioteka namenjena za izvršavanje graf algoritama

5. PRIMER KREIRANJA GRAF BAZE PODATAKA U NEO4J-U

Prvi korak u razvoju bilo kakve baze podataka uvek je zapravo definisanje domena. Pretpostavimo da imamo domen u kojem imamo određene korisnike i oglase. Svaki oglas izdao je jedan korisnik, a jedan korisnik može izdati veći broj oglasa. U isto vreme, svaki korisnik može se prijaviti na veći broj oglasa, a samim tim i na jedan oglas se može prijaviti više od jednog broja korisnika. O svakom korisniku možemo znati ime, godinu rođenja, i on potencijalno može imati jedan ili veći broj nadimaka.

U domenu postoje 5 korisnika: *Lana Lazarević*, *Petar Petrović*, *Milan Milovanović*, *Zorana Jovanović* i *Vladimir Lukić* i imamo 6 oglasa: *Oglas za posao*, *Oglas za ljubimca*, *Oglas za prevoz*, *Oglas za literaturu*, *Oglas za stan* i *Oglas za kuću*. *Lana Lazarević izdala je Oglas za posao*, *Oglas za literauru* i *Oglas za ljubimca*; *Petar Petrović izdao je Oglas za prevoz* i *Oglas za stan*, a *Milan Milovanović izdao je Oglas za kuću*. *Zorana Jovanović prijavila se za Oglas za ljubimca*, *Oglas za stan* i *Oglas za kuću*; *Petar Petrović prijavio se za Oglas za stan* i *Oglas za kuću*, a *Vladimir Lukić se prijavio za Oglas za kuću*.

Petar Petrović ima 3 *nadimka*: *Pera*, *Petrović* i *Perica*, dok *Lana* ima *nadimak Lazarevićka*. Ostali korisnici nemaju nadimke.



Slika10: Graf baza podataka u Neo4j-u

Pretpostavimo da konačan ishod treba da ima izgled kao na slici.

U opisu samog domena možemo da zaključimo da tip veze koji je opisan kao izdavanje oglasa jeste *1:N*, dok je tip veze opisan kao prijavljivanje na oglas *N:M*. Već sa slike možemo da uočimo da nema razlike u kojoj će se ove veze iako umnogome drugačije, predstaviti u graf modelu.

5.1. WHITEBOARD

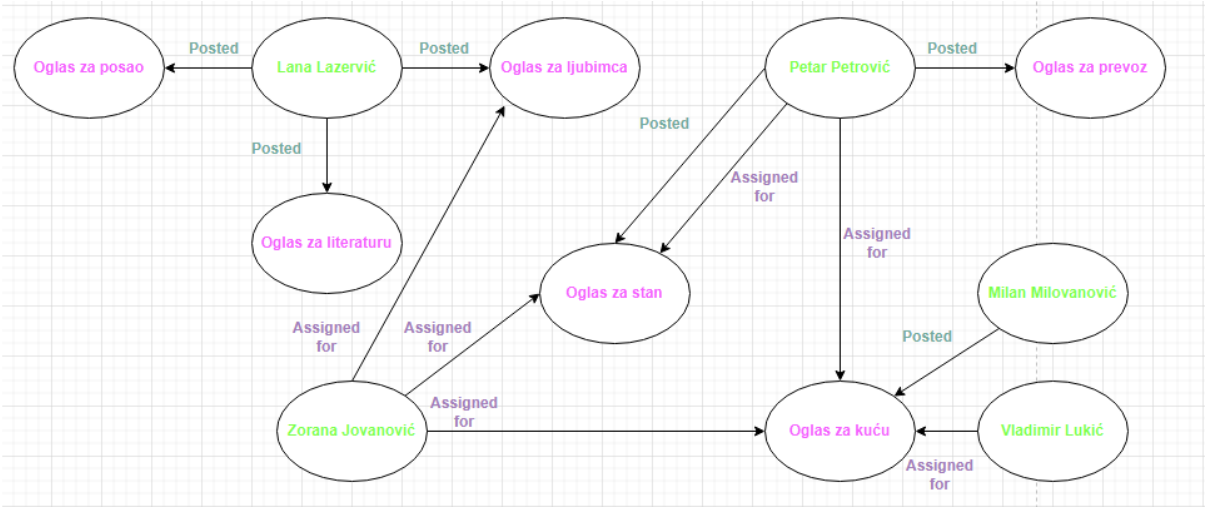
Kao što je i opisano u postupku iz prethodnog poglavlja, prvi korak u razvoju jeste izdvajanje čvorova iz domena.

Slika11: Whiteboard - Izdvajanje čvorova iz domena

Naredni korak u postupku bio bi definisanje labela.

Slika12: Whiteboard - Uvođenje labela čvorova u domen

Potom, povezujemo čvorove domena usmerenim potezima.

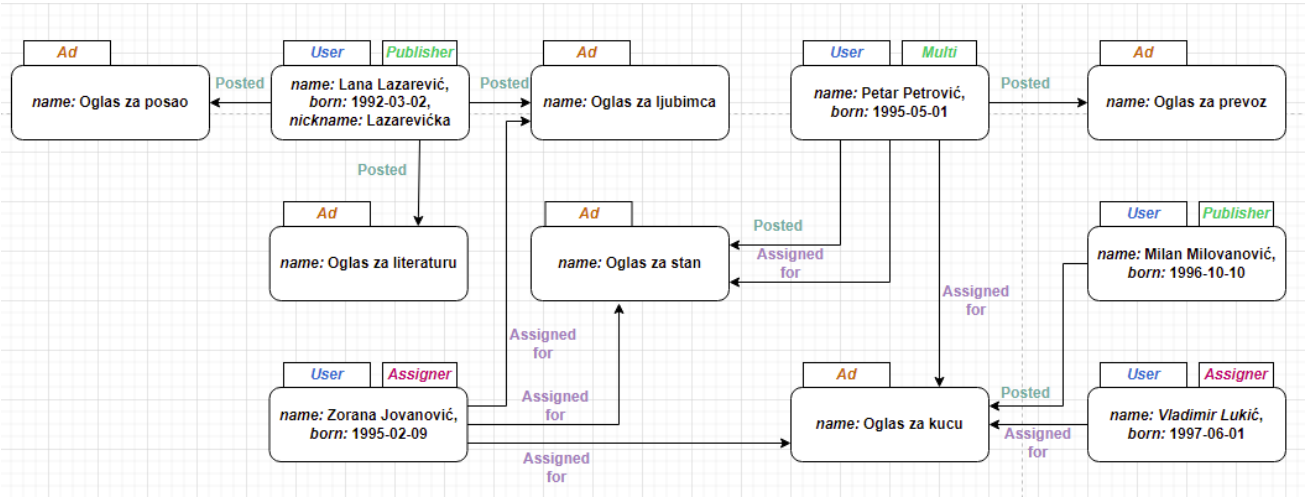


Slika13: Whiteboard - Povezivanje čvorova u domenu

U ovom trenutku možemo da vidimo da redosled po kojem ćemo određivati parametre graf modela, poput labela, atributa ili relacija, nije obavezno takav da labele idu pre relacija ili pre atributa. Evo i iz kog razloga. Naime, vidimo već sa whiteboard grafa da određeni korisnici imaju samo *posted* veze, dok neki imaju samo *assigned for* veze, a neki imaju i jedne i druge. Samim tim, možemo dodatno podeliti korisnike u podkategorije, dakle, da imamo *izdavače oglasa*, *korisnike oglasa* i *korisnike koji zadovoljavaju oba uslova*. Ovo, naravno, zavisi od potreba domena. Kako se labele mogu vremenom dodavati i eliminsati iz čvorova, na osnovu trenutnog stanja korisnikovih relacija sa ostalim čvorovima možemo da zaključimo i koju bi on labelu trebalo da ima kao podkategoriju. Tako, *Lana* i *Milan* mogu predstavljati *izdavače*, *Zorana* i *Vladimir korisnike oglasa*, dok je *Petar i jedno i drugo*.

Poslednja stavka su atributi, koji su definisani već u samom domenu problema.

5.2. PRILAGOĐAVANJE WHITEBOARD GRAFA SINTAKSI KOJA SE KORISTI ZA KREIRANJE GRAF BAZE PODATAKA



Slika14: Formatiranje

5.3. KREIRANJE GRAFA U NEO4J-U

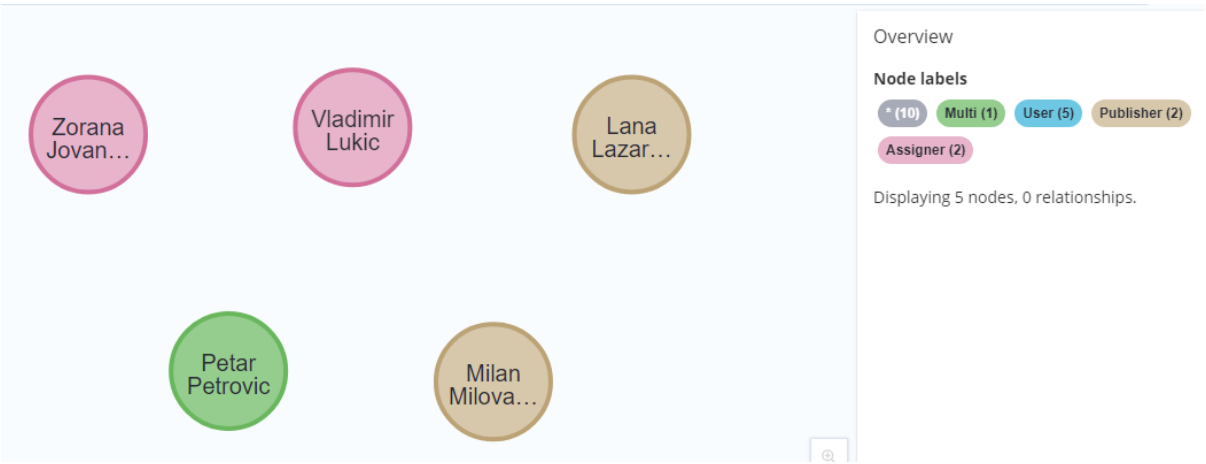
Vidimo da se ni graf dobijen u *whiteboard* delu niti graf dobijen nakon toga ne razlikuju puno od krajnjeg grafa.

5.3.1. KREIRANJE ČVOROVA

Prvi korak u formiranju graf modela u *Neo4j*-u jeste kreiranje čvorova i pritom je moguće definisati i njihove labele i njihove atribute. Bez obzira na to da li to učinimo prilikom kreiranja samog čvora ili ne, *Cypher* nam nudi mogućnost da i naknadno te informacije dodamo čvoru.

```
CREATE (petar: User: Multi {name: "Petar Petrovic", born: date("1995-05-01"), nickname: ["Pera",
"Petrovic", "Perica"]})
CREATE (milan: User: Publisher {name: "Milan Milovanovic", born: date("1996-10-10")})
CREATE (lana: User: Publisher {name: "Lana Lazarevic", born: date("1992-03-02"), nickname:
["Lazarevicka"]})
CREATE (vladimir: User: Assigner {name: "Vladimir Lukic", born: date("1997-06-01")})
CREATE (zorana: User: Assigner {name: "Zorana Jovanovic", born: date("1995-02-09")})
RETURN petar, milan, lana, vladimir, zorana
```

Slika15a: Naredbe za kreiranje čvorova korisnika



Slika15b: Rezultat izvršenja naredbi za kreiranje čvorova korisnika

Naime, *CREATE* naredba koristi se za kreiranje čvora ili relacije. Prilikom kreiranja čvorova navode se *()* i sve što navedemo unutar toga, odnosi se na čvor koji kreiramo, jer se zagrade u *Cypher*-u koriste da reprezentuju čvor. U samom čvoru možemo definisati varijablu *petar, milan, lana, vladimir, zorana,* koje se koriste da bismo naknadno mogli ponovo da upotrebimo te čvorove za dalje upite, ili, u ovom slučaju, da bismo kreirane čvorove vratili i prikazali *RETURN* naredbom.

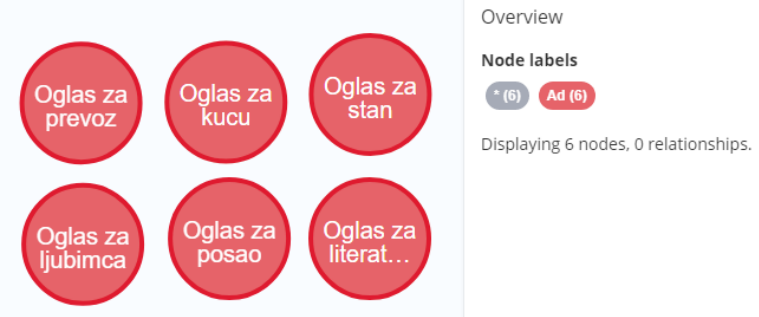
Takođe, u okviru čvora definišemo i labele kojih može biti, ali ne mora, a može ih biti i više. Labele se definišu tako što se navode *:* iza koje sledi naziv labele. Ukoliko ih je više isti postupak važi i za svaku narednu labelu.

Sve atribute čvorova definišemo u okviru velikih zagrada i svaki pojedinačni atribut odvajamo zapetom od narednog atributa ukoliko ih je više. Jedan atribut definiše se tako što se navodi njegov naziv potom *:* iza koje sledi vrednost tog atributa. Kada je reč o jednovrednosnom atributu to će biti sama vrednost, dok, ukoliko se radi o viševrednosnom atributu, navodimo *[]* kao oznaku za listu i u okviru nje navodimo vrednosti. Tako, vidimo da se u okviru jednog čvora čuvaju svi njegovi atributi, bili oni jednovrednosni ili viševrednosni. Isto važi i za relacije i njihove atribute ukoliko postoje.

Na sličan način vrši se i kreiranje čvorova koji predstavljaju oglase.

```
CREATE (stan: Ad {name: "Oglas za stan"})
CREATE (kuca: Ad {name: "Oglas za kucu"})
CREATE (prevoz: Ad {name: "Oglas za prevoz"})
CREATE (literatura: Ad {name: "Oglas za literaturu"})
CREATE (ljubimac: Ad {name: "Oglas za ljubimca"})
CREATE (posao: Ad {name: "Oglas za posao"})
RETURN stan, kuca, prevoz, literatura, ljubimac, posao
```

Slika16a: Naredbe za kreiranje čvorova oglasa



Slika16b: Rezultat izvršenja naredbi za kreiranje čvorova oglasa

5.3.2. KREIRANJE RELACIJA

Imamo dva tipa relacija u primeru koji je predstavljen: *izdavanje oglasa* i *prijavljivanje na oglas*. Jedna je *1:N*, a druga *N:M*. Ipak, to u *Neo4j*-u ne utiče niti na način na koji će ona biti kreirana niti na način na koji će biti predstavljena. Uvek kada se u *Neo4j*-u kreiraju relacije, postupak je isti:

- Pronaći izvorni čvor
- Pronaći odredišni čvor
- Kreirati relaciju iz izvornog ka odredišnom čvoru

Samim tim, neće se puno razlikovati ni pristup ni način na koji ćemo realizovati ova dva tipa relacija.

```
MATCH (petar: User {name: "Petar Petrovic"})
MATCH (milan: User {name: "Milan Milovanovic"})
MATCH (lana: User {name: "Lana Lazarevic"})
MATCH (stan: Ad {name: "Oglas za stan"})
MATCH (kuca: Ad {name: "Oglas za kucu"})
MATCH (prevoz: Ad {name: "Oglas za prevoz"})
MATCH (ljubimac: Ad {name: "Oglas za ljubimca"})
MATCH (literatura: Ad {name: "Oglas za literaturu"})
MATCH (posao: Ad {name: "Oglas za posao"})
```

Slika17a: Naredbe za pribavljanje čvorova koji će učestvovati u relaciji POSTED

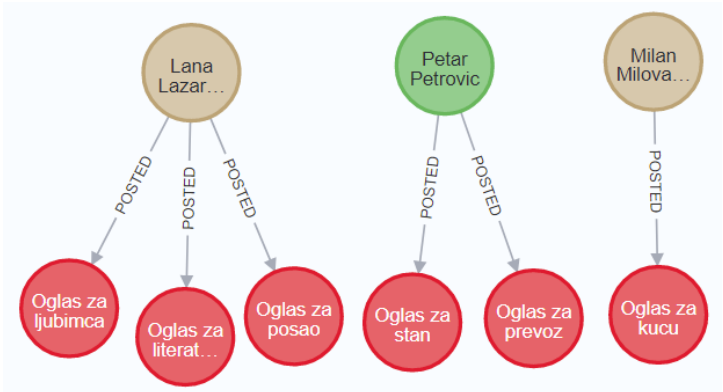
```
CREATE (petar)-[izdao1: POSTED]→(stan)
CREATE (petar)-[izdao2: POSTED]→(prevoz)
CREATE (milan)-[izdao3: POSTED]→(kuca)
CREATE (lana)-[izdao4: POSTED]→(ljubimac)
CREATE (lana)-[izdao5: POSTED]→(literatura)
CREATE (lana)-[izdao6: POSTED]→(posao)
```

Slika17b: Naredbe za kreiranje relacija POSTED između odgovarajućih čvorova

Ipak, postupak kojim kreiramo relacije u odnosu na postupak kojim kreiramo čvorove se nešto razlikuje. Dakle, prvo je potrebno da pronađemo čvorove koje ćemo koristiti. *MATCH* naredba je nešto slično onome što *SELECT* predstavlja u *SQL*-u. Koristi se da bi se izvršilo filtriranje i pribavljanje isključivo onih podataka koji su neophodni. Sve pribavljene čvorove smeštamo u odgovarajuće varijable, jer ih kasnije koristimo da formiramo relacije između njih.

U samoj naredbi kreiranja relacije se na jednom kraju nalazi jedan čvor, na drugom kraju se nalazi drugi čvor, a između njih se definiše i sama relacija. Relacije se u *Cypher*-u predstavljaju pomoću *–[]->* ili *<-[]-* pri čemu se unutar srednjih zagrada može navesti varijabla koja će se koristiti da čuva relaciju i nalazi se, u ovom slučaju obavezna labela kojom se definiše tip relacije koji se između čvorova kreira. Smer strelice u relaciji određuje i smer u kojem se ona kreira.

Postavlja se pitanje zbog čega nismo jednostavno u *CREATE* naredbi naveli i paterne za pronalazak čvorova koje smo navodili u *MATCH* naredbi. Kada je *CREATE* naredba u pitanju, ona je dizajnirana tako da posmatra kompletan patern koji je naveden i da njega kao takvog, kompletnog, kreira. Iz tog razloga, neophodno je prvo pronaći čvorove koje želimo da povežemo, a zatim njihove varijable iskoristiti i za realizaciju relacije među njima. Takođe, moguće je i upotrebiti pristup u kojem i čvorove koji su u relaciji i same relacije kreiramo istovremeno. To bi značilo da, ukoliko nismo prethodno kreirali čvor čija je varijabla *petar* i čvor čija je varijabla *stan*, mogli bismo jednom *CREATE* naredbom kreirati dva čvora i jednu relaciju među njima, navođenjem samo jednog paterna.

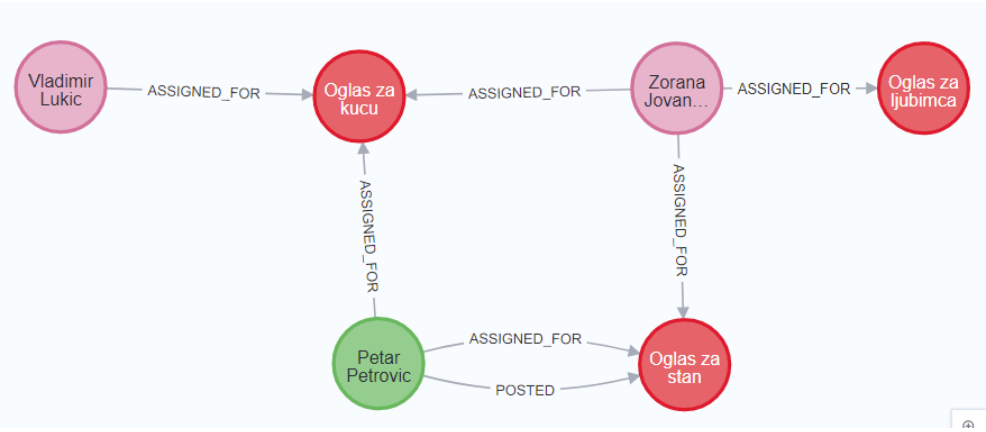


Slika17c: Rezultat kreiranja relacija POSTED između odgovarajućih čvorova

Vrlo sličnim postupkom obavljamo i kreiranje N:M relacije.

```
MATCH (petar: User {name: "Petar Petrovic"})
MATCH (vladimir: User {name: "Vladimir Lukic"})
MATCH (zorana: User {name: "Zorana Jovanovic"})
MATCH (stan: Ad {name: "Oglas za stan"})
MATCH (kuca: Ad {name: "Oglas za kucu"})
MATCH (ljubimac: Ad {name: "Oglas za ljubimca"})
CREATE (petar)-[prijavio1: ASSIGNED_FOR]→(stan)
CREATE (zorana)-[prijavio2: ASSIGNED_FOR]→(stan)
CREATE (petar)-[prijavio3: ASSIGNED_FOR]→(kuca)
CREATE (zorana)-[prijavio4: ASSIGNED_FOR]→(kuca)
CREATE (vladimir)-[prijavio5: ASSIGNED_FOR]→(kuca)
CREATE (zorana)-[prijavio6: ASSIGNED_FOR]→(ljubimac)
```

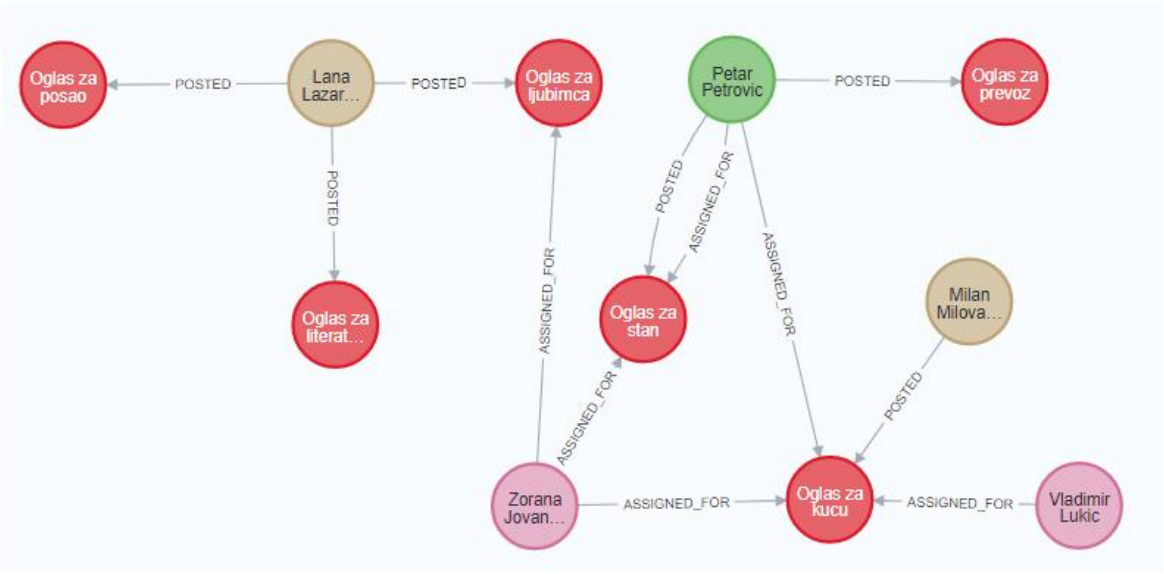
Slika18a: Naredbe za kreiranje relacija ASSIGNED_FOR između odgovarajućih čvorova



Slika18b: Rezultat izvršenja naredbi za kreiranje relacija ASSIGNED_FOR između odgovarajućih čvorova

Zaključak već sa ove dve slike jasno stavlja do znanja činjenicu da *Neo4j* uopšte ne pravi razliku između kardinalnosti i participacije različitih tipova relacija.

Pošto smo kreirali i ovaj tip veza, kao konačni skup kompletnog modela koji smo kreirali dobijamo model kao sa početka.



Slika19: Konačni graf model

6. RAZLIKE NEO4J BAZE PODATAKA U ODNOSU NA RELACIONE I PREOSTALE NOSQL BAZE PODATAKA

Kao što je i isticano u samom radu više puta, glavna snaga *Neo4j* baze podataka leži u načinu na koji se ona odnosi prema relacijama koje postoje među podacima. Bez obzira na to, prednosti i nedostaci bilo koje tehnologije uvek postoje i samim tim, na osnovu onoga kakve su potrebe i zahtevi, odlučuje se po kojim bi se pitanjima mogao napraviti kompromis kada su karakteristike u pitanju.

6.1. NEO4J U POREĐENJU SA RELACIONIM BAZAMA PODATAKA

Relacione baze podataka prisutne su već jako dugo, i sama činjenica da su i dalje jedan od najpopularnijih rešenja koja se koriste, govori jako puno u prilog samoj tehnologiji. Naime, *query* jezik koji koriste je jako jednostavan i srodan engleskom govornom jeziku. Pored toga, postoji jako puno okruženja u kojima se mogu upotrebiti relacione baze podataka da se čak i poznavanje *SQL*-a ne zahteva da bi se sama baza podataka koristila.

Konzistentnost, sigurnost, garantovanje *ACID* svojstava i stroga struktuiranost relacionog modela je takođe skup elemenata koji mogu da se uvrste u njegove dobre osobine. Upravo to strogo ograničavanje na šemu modela i tabelarna reprezentacija jasno govore šta možemo da očekujemo.

Međutim, relacioni model upravo ta stroga struktuiranost vrlo često može koštati kompleksnog načina proširivanja modela. Osim toga, pojava nestruktuiranih podataka dovodi do sve veće nekompatibilnosti upotrebe jednog ovako striktnog modela. Očekuje se konzistentnost u strukturi podataka od podataka koji to ne mogu da garantuju.

Sa druge strane imamo graf bazu podataka, *Neo4j*, koja koristi gotovo jednako jednostavan *query* jezik za izvršavanje upita, koji se takođe bazira na engleskom govornom jeziku. Takođe, i *Neo4j* nudi mnoštvo platformi na kojima se na vrlo jednostavan i intuitivan način može koristiti i manipulisati bazom podataka.

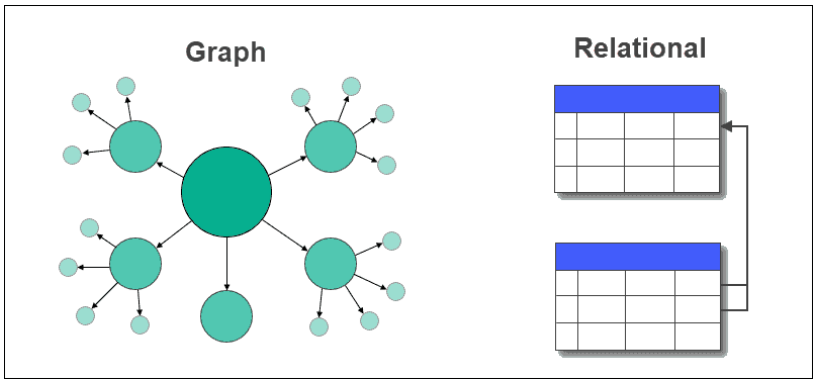
Kao i relacioni model i *Neo4j* garantuje *ACID* svojstva, samim tim i konzistentnost podataka i sigurne transakcije, međutim, za razliku od njega, u *Neo4j* ne postoji ograničavanje na jednu strukturu modela. Ovo je *schema free* tip baze podataka, koji je i došao sa pojavom nestruktuiranih podataka. Naime, ne postoji tačan skup atributa ili njihovih vrednosti koje određeni entitet mora imati kada se koristi *Neo4j*. Sama ova činjenica olakšava mogućnost proširivanja modela.

Kada je reč o relacijama, *Neo4j* je i nastao kao motiv da se prevaziđe problem koji je relacioni model imao prilikom rada sa relacijama. Naime, da bi se u relacionom modelu ostvarila relacija, kreiralo bi se ili posebno polje u tabeli ili posebna tabela kojom bi se zatim povezivali primarni ključevi tabela koje su u relaciji. Na ovaj način, povezivani su jedinstveni identifikatori ovih elemenata, ali ne konkretni entiteti. Osim toga, prilikom bilo kakve obrade podataka koje zahtevaju rad sa relacijama, proces bi bio jako skup, kompleksan i sam upit

takođe. Upravo su sve ove stvari i bile skup problema koje je trebalo rešiti uvođenjem *Neo4j*-a.

Neo4j sa druge strane vrlo jednostavno manipuliše relacijama. U ovom se slučaju povezuju konkretni čvorovi, a ne njihovi jedinstveni identifikatori. Sa jedne strane relacije je jedan čvor, a sa druge drugi čvor i ne postoje kompleksne i skupe operacije koje bi trebalo u svrhu rada sa njima koristiti. U ovom se slučaju ne vrši *JOIN* dve tabele, već se pronalaskom relacije jednostavno nalaze čvorovi koji su sa obe strane te relacije. Takođe, nije neophodno poznavanje ni strukture ni organizacije samih čvorova da bismo mogli da *''putujemo''* kroz čvorove relacijama, već čak, iako bi relacija bila ostvarena u jednom smeru, mogli bismo da se kroz čvorove krećemo u oba smera njome.

Uzevši u obzir i viševrednosne atribute i činjenicu da za realizaciju ovakvih propertija relacioni model zahteva kreiranje nove tabele i relacije, a *Neo4j* to sve skladišti u vidu jednoj atributa čvora koji može imati listu vrednosti, možemo zaključiti da je širina i kompleksnost skladišta koju dobijamo korišćenjem relacionog modela umnogome veća od one koja se dobija korišćenjem *Neo4j*-a.



Slika20: Poređenje relacionog modela sa graf modelom

Ipak, potpuno bi nezahvalno bilo posmatrati ova dva rešenja bez uzimanja u obzir i konkretnog slučaja u kojem bi se upotrebili. Tako, relacioni model i dalje ostaje neprikosnoveno rešenje za sve transakcijski vođene slučajeve korišćenja, poput novčanih transakcija pre svega, zbog garantovane sigurnosti, bezbednosti i konzistentnosti podataka, činjenice da se jasno zna šta može da se očekuje. Sa druge strane, za bilo koju situaciju u kojoj su podaci izuzetno umreženi, kao što su, pre svega, socijalne mreže, *Neo4j* se nameće kao pravi izbor.

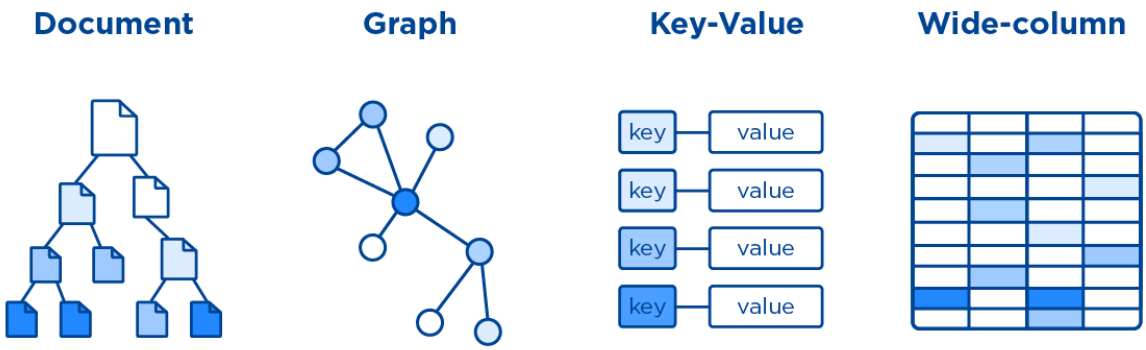
6.2. NEO4J U ODNOSU NA PREOSTALE NOSQL BAZE PODATAKA

Za razliku od relacionog modela, u slučaju razgovora o *NoSQL* bazama podataka, svaka zapravo od njih podržava koncept nestruktuiranih podataka. Svi *NoSQL* sistemi su agregacijski orijentisani i grupišu podatke na osnovu određenog kriterijuma i same strukure koju ta baza podataka podržava(*document, key-value,* i tako dalje).

Kako su *NoSQL* sistemi generalno nastali sa ciljem da reše određene probleme i izazove sa kojima bi se do tada suočavala relaciona baza podataka, svaka od ovih grupa

orijentisana je na rešavanje jednog određenog problema. Samim tim, svaka je upravo po pitanju tog kriterijuma neprikosnovena u poređenju sa preostalim, ali vrlo često vrši kompromis po pitanju preostalih kriterijuma.

Osim graf baza podataka, kakva je *Neo4j*, u *NoSQL* baze podataka spadaju još i *key-value store*, *wide column store* i *document store*.



Slika21: NoSQL baze podataka

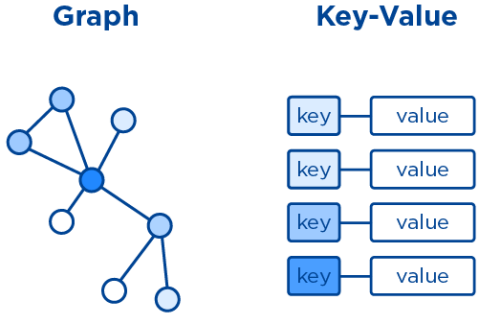
6.2.1. NEO4J U POREĐENJU SA KEY-VALUE STORE BAZAMA PODATAKA

Kada je reč o *key-value store NoSQL* bazama podataka, glavni motiv prilikom razvoja bio je povećavanje brzine. Tako, ova grupa baza podataka radi umnogome brže od *Neo4j* baze podataka, ali i od svih preostalih *NoSQL* baza podataka i upravo se u tome i ističe glavna prednost ove grupe.

Isto tako, *Neo4j* znatno bolje reguliše pitanje relacija od *key-value store* baza podataka. Samim tim, možemo reći da je *Neo4j*, odnosno graf baze podataka, fleksibilniji tip od *key-value store*-a.

Key-value store funkcioniše i organizovan je kao *hash* tablica. Osim brzine, jedna od glavnih prednosti je još i skalabilnost.

Kao i u slučaju poređenja relacionih baza podataka sa *Neo4j*-em, i u ovom slučaju je potpuno nezahvalno posmatrati rešenje bez definicije problema za koji se koristi. Tako, za neke svakodnevne aktivnosti, poput *online shopping*-a na primer, *key-value store* je idealno rešenje. Ipak, sa druge strane, kada imamo neke kompleksnije zadatke poput modelovanja grafa znanja i ontologije, *Neo4j* bi bio očigledniji izbor.



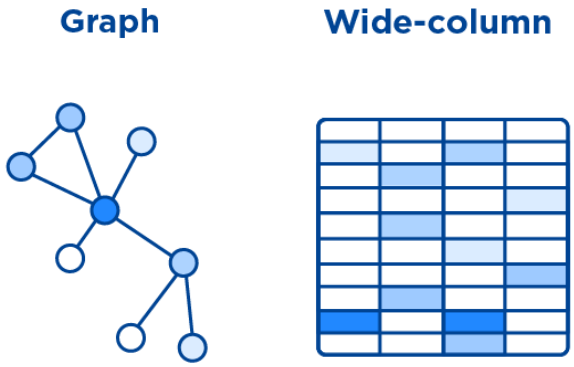
Slika22: Poređenje graf modela sa key-value store-om

6.2.2. NEO4J U POREĐENJU SA WIDE COLUMN STORE BAZAMA PODATAKA

Wide column store se može posmatrati kao dvodimenzionalni *key-value store*. Naime, ova grupa baza podataka organizovana je u tabele, redove i kolone, međutim, ne kao relacione baze podataka. Za razliku od njih, u ovom slučaju nema skladištenja kolona sa *null* vrednostima za određene redove, pa je samim tim struktura podataka dinamička. Kao takve, *wide column* baze podataka su jako fleksibilne i pružaju mogućnost skladištenja velike količine različite strukture podataka unutar jedne iste tabele.

Izuzev osnovnih elemenata koji izdvajaju obe pojedinačne grupe baza podataka koje poredimo, glavna razlika predstavnika *Wide column store*-a, *Cassandra* baze podataka u odnosu na *Neo4j* jeste što *Neo4j* podržava *ACID*, a *Cassandra* ne. Naime, *Cassandra* nije tip baze podataka koji garantuje konzistentnost u podacima, što dovodi do toga da prilikom čitanja podataka možemo dobiti podatke koji nisu *update*-ovani.

Opet, iako je tako, u nekim situacijama kada nam je potrebno veće skladište pogodnije je koristiti *wide column store* i *Cassandra*-u nego *Neo4j* i obrnuto. Kada nam nije presudni zahtev da podaci u svakom trenutku budu u konzistentnom stanju, možemo vršiti kompromis po tom pitanju i samim tim se odlučiti za *wide column store*. Ali, kada nam je to neizbežno potrebno, onda *Neo4j* nosi prednost.

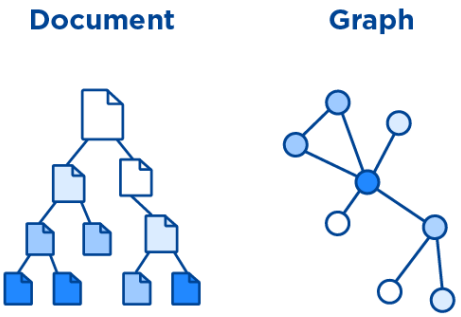


Slika23: Poređenje graf modela sa wide-column store-om

6.2.3. NEO4J U POREĐENJU SA DOCUMENT STORE BAZE PODATAKA

U *document store*-u se podaci organizuju u stablo, te pružaju potpuno novu taksonomiju organizacije podataka. Hierarhija pritom može biti promenjena tokom života baze podataka i svaki dokument može imati različite skupove atributa. Pritom su vrednosti podataka ili listovi stabala zapravo u *XML*, *JSON* ili *BSON* formatu. *Document* baze podataka kao takve vrlo se lako mogu proširiti i vrlo brzo mogu pribavljati željene podatke na osnovu metapodataka dokumenata. Sve ovo čini ih jednom od vodećih budućih baza podataka, s obzirom na činjenicu da omogućavaju korišćenje različitih formata podataka kao vrednosti i samim tim, omogućavaju podršku i za neke strukture podataka koje će tek biti kreirane u budućnosti.

Graf baze podataka i *document* baze podataka imaju zapravo samo različit pristup za rešavanje slične vrste problema.



Slika24: Poređenje graf modela sa document store-om

7. ZAKLJUČAK

Sa pojavom nestruktuiranih podataka javila se i potreba da se njima nekako upravlja. *NoSQL* baze podataka nametnule su se upravo kao rešenje za to.

Poznavajući prednosti i nedostatke, samim tim i mogućnosti tehnologije možemo i odlučiti da li je ona pogodno rešenje za problem pred nama ili ne. Sagledavajući stroge zahteve pred nama, pitanja po kojima možemo praviti kompromis i pogodnosti koje tehnologija nudi, znaćemo i da li je možemo upotrebiti.

Stoga, poznavajući jednostavnost u radu koju *Neo4j* nudi kada su u pitanju relacije i činjenicu da on obezbeđuje sva *ACID* svojstva, a zatim i ne tako široko i kompleksno skladište koje se dobija prilikom korišćenja ove baze podataka, možemo zaključiti da bi u mnogim realnim problemima upravo ova grupa baza podataka trebalo da se nametne kao logično rešenje. Tako, u poljima kao što su socijalne mreže ili veštačka inteligencija, *Neo4j* se pokazao kao najbolji sklad.

8. SPISAK SLIKA

9. LITERATURA

[1] – Neo4j dokumentacija sa oficijalnog sajta - https://neo4j.com/docs/

[2] – Materijal sa predavanja i vežbi za predmet Napredne baze podataka - https://cs.elfak.ni.ac.rs/nastava/