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 grafovske baze podataka. Uvodni deo odnosi se na uopšteno fizičko projektovanje baza podataka, sa poređenjem relacionih i NoSQL baza podataka, a onda se osvrćemo na konkretnu implementaciju, planiranje, modeliranje i osobine Neo4j grafovske baze podataka.

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. Relacione baze podataka su jako fleksibilnije (ali teže za kreiranje i održavanje) od baza poznatih kao Flat-file koje podrazumevaju skladištenje svih podataka u jednu tabelu. Važnost relacione baze podataka ogleda se u stvaranju univerzalnog modela čuvanja podataka na računarima.

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

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.
- **Konceptualna (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.
- **Spoljašnja šema**
  Najviši nivo apstrakcije koji predstavu o podacima iz baze prilagođava potrebama korisnika (grupe korisnika). Ova shema opisuje kako korisnici vide podatke.

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 fizičkom nivou. Reprezentacija koja se nalazi na konceptualnom 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 se često naziva i „modeliranje podataka“ (engl. *Data modeling*). Od značaja je interna (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.

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

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 oređene implementacije.

Relacione baze podataka imaju mnoge pozitivne strane, medjutim, one imaju svoju cenu:

- relativno visoka cena čitanja
  – zbog neredundantnosti i stroge strukture podataka, odnosno, zbog čestog spajanja podataka
- otežano distribuiranje
  – zbog konzistentnosti podataka
- skupa promena strukture
  – zbog povezanosti strukture sa upotrebom i optimizacijama

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. Kod grafovske baze podataka, modeliranje se vrši na prvom koraku pri razmišljanju o entitetima i povezivanju nacrtanih primera na tabli 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.

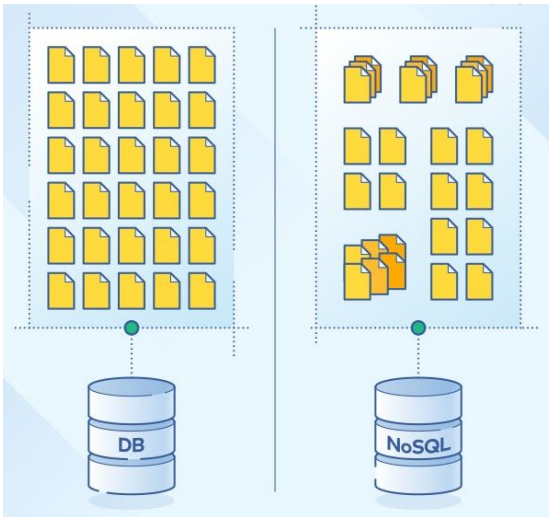
Graf baze podataka i Neo4j

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.

Prednosti graf baza u odnosu na relacione

Osim prethodno pomenutih prednosti i mana relacionih baza podataka u nastavku će biti opisane jače strane kod grafovskih 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. 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.



Slika1. *Prikaz sistema relacione 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 (dodavanje, ažuriranje, brisanje) zahtevaju puno vremena i novca. Kao što je prethodno pomenuto, kod graf baza šema je fleksibilna i ažuriranje grafa sprovodi se jednostavno, bez velikog utroška resursa. 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.

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 rađen 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.

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.

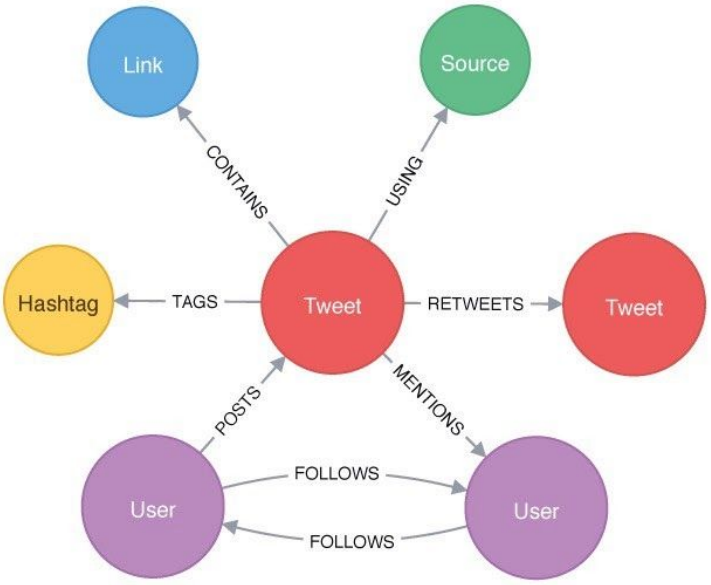
Svojstva (engl. Properties):

- 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.

Neo4j koristi indekse za pretraživanje u cilju pronalaženja čvorova i relacija na koje će se primeniti grafovske operacije. Indeksi se koriste kako bi obezbedili jedinstvenost vrednosti pojedinih svojstava. Podrazumevani indeks je *Lucene*, *open-source* biblioteka koja obezbeđuje mehanizme za indeksiranje i pretraživanje podataka. Na Slika2 prikazani su **čvorovi** i **veze** u pokaznom primeru Neo4j baze za društvenu mrežu *twitter*.



***Slika2**. Prikaz primera čvorova i veza u Neo4j bazi društvene mreže twitter*

Skladište grafa i GPE

Postoje i dva značajna pozadinska svojstva kod grafovskih baza podataka. **Skladište grafa** (engl. *Graph storage*) koje može biti posebno dizajnirano za smeštanje i organizaciju grafova ili bazirano na relacionom modelu ili objektno-orijentisanim bazama. **Mašina za procesiranje grafa** (engl. *Graph processing engine*) koja može biti *native,* za obrađivanje čvorova koji su međusobno povezani i stvaraju puteve ili *non-native* koja 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 operaciju čitanja ili upisa. Povezani podaci zahtevaju neobično strogu potrebu za integritetom podataka koji prevazilazi ostale NoSQL modele. Da bismo spremili vezu između dva entiteta, moramo napisati zapis 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. ***Native* sistemi** osiguravaju interne strukture 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.

Cypher Query Language

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 jednostavno izraženi, 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[2].

[2] *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.

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

Graf označenih svojstava

Jedan 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 prioritetne e-mail adrese. Na ovaj način predstavljena je denormalizacija kod relacionih baza.
Kod grafovskih baza postoje dve tehnike koje pomažu u ovom slučaju. Jednostavnija tehnika 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.

Upitni jezik Cypher dozvoljava kreiranje indeksa po kombinaciji oznaka i atributa. Za jedinstvene vrednosti atributa mogu se definisati ograničenja (engl. *constraint*). Pretrage ne zahtevaju indekse, ali njihovim dodavanjem mogu biti poboljšane. U nastavku su prikazane komande koje ilustruju prethodno navedene karatkeristike.

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

Kreiranje indeksa za Alumni članove, koji se zasniva na njihovom atributu “ime”.

```
CREATE CONSTRAINT 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;
```

Indeksirane pretrage mogu da funkcionišu u manjim mrežama, ali su skupe za upite kod većih grafova. U prethodnom poglavlju je pomenuto da *native* graf baze koriste pridruživanje bez indeksa (engl. *index-free adjacency*) umesto indeksiranja.

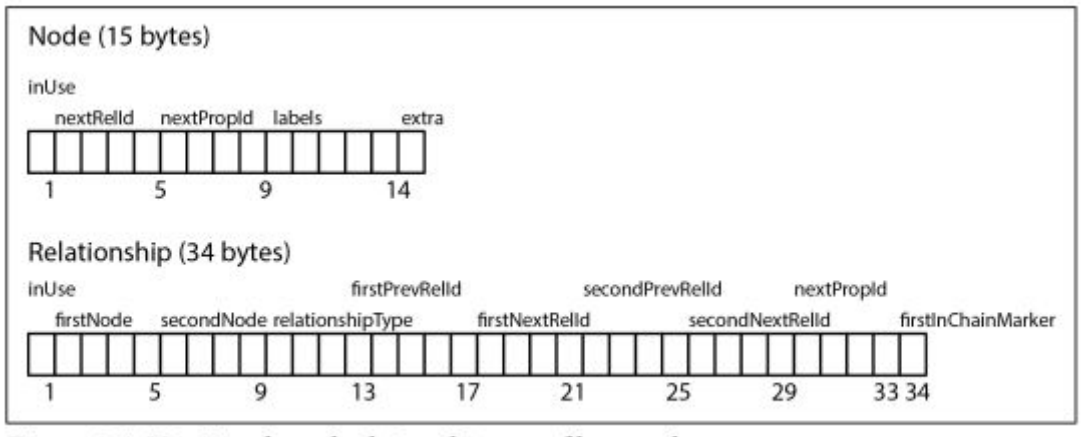
Glavna primena indeksa kod graf baze je u nalaženju početne tačke za obilazak grafa, jer se na dalje obilazak oslanja na strukturu grafa za visoke performanse. Indeksi mogu biti dodati u bilo kom trenutku i oni se automatski koriste prilikom slanja upita. Postoje i složeni indeksi koji obuhvataju više 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.
Za proveru definisanih indeksa u bazi možemo koristiti ugrađenu proceduru

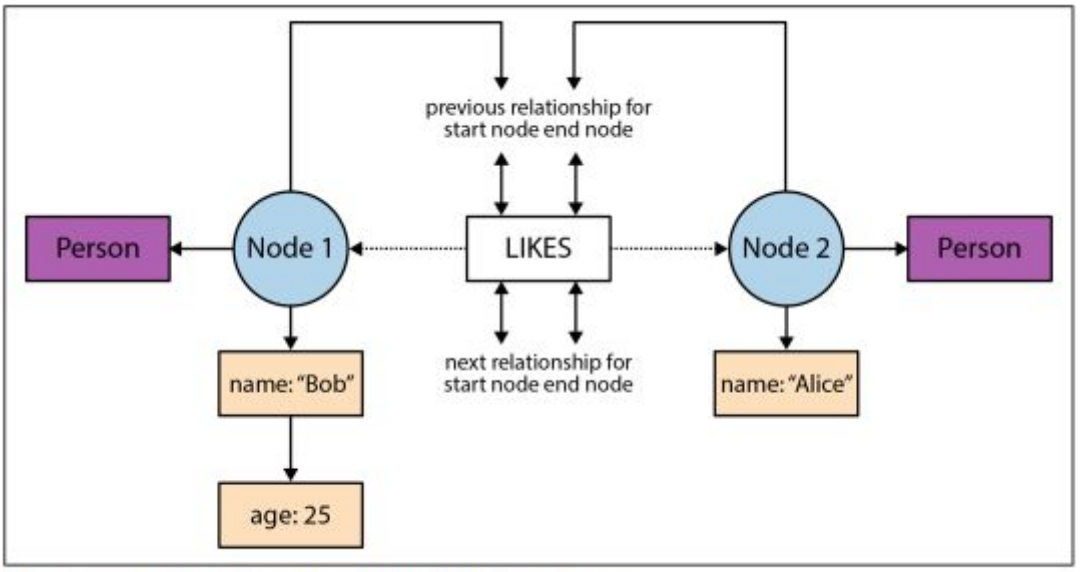
```
CALL db.indexes
YIELD description, tokenNames, properties, type;
```

U prethodnom poglavlju bilo je reči i o skladištenju podataka *native* i *non-native* graf baza. 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 Slika3.



Slika3. *Fajlovi za skladištenje Neo4j čvora i veze*

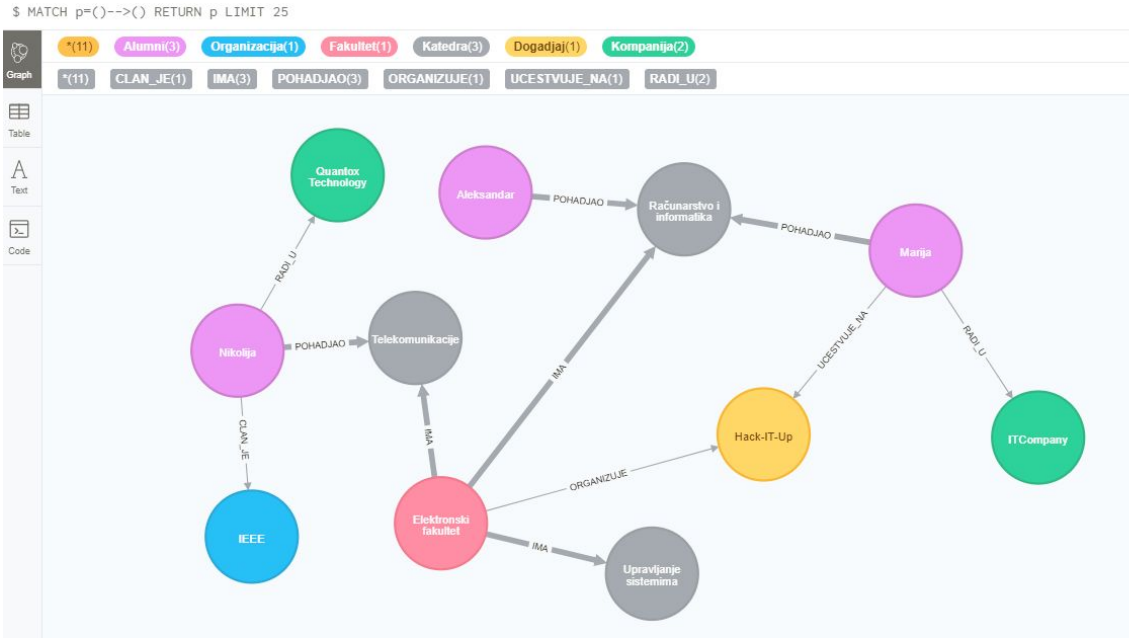
Svaki čvor koji se kreira na korisničkom nivou grafa smešta se u fajl za čuvanje čvora (engl. *node store file*). Fiksna veličina fajla omogućava bržu pretragu čvovora u fajlu *neostore.nodestore.db*, jer sistem može da preračuna memorijsku lokaciju na osnovu ID-ja i to složenošću O(1). Slično tome, 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.



Slika4. *Fizičko smeštanje Neo4j grafa*

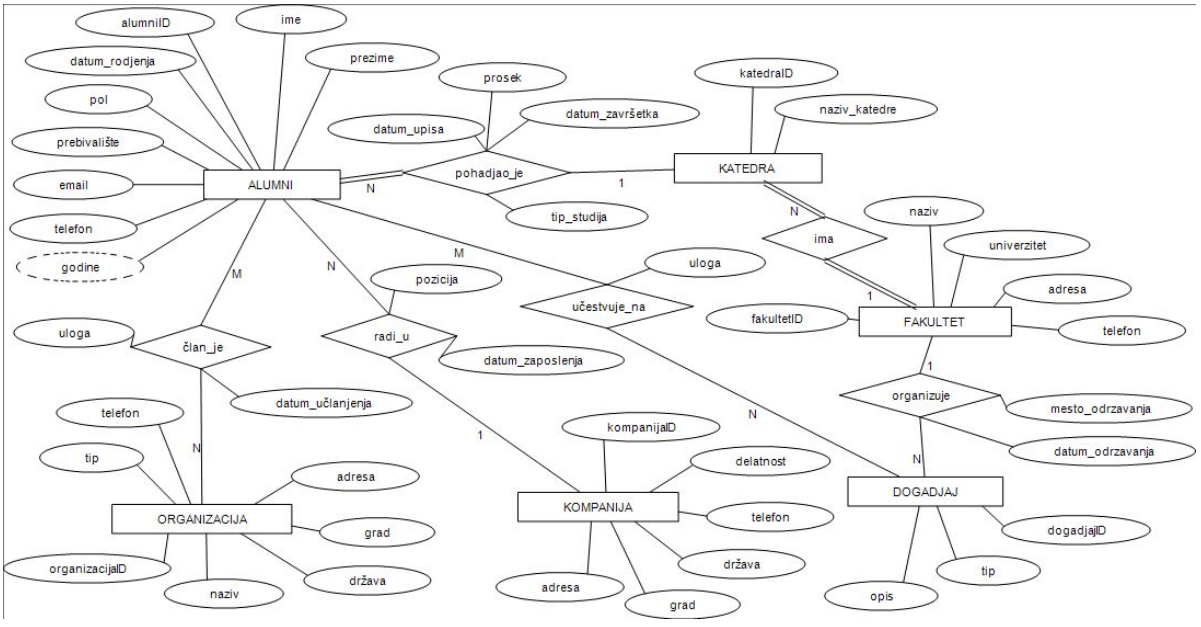
Na sličan način čuvaju se i atributi u fajlu *neostore.propertystore.db*, koji su takođe fiksne veličine. Svi atributi nalaze se u jednostruko lančanoj listi tako da svaki atribut ukazuje na sledeći preko ID-ja.

Praktični primeri pri modeliranju graf baze



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, 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 modelovanja.



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)

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@ITC.rs', 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.

Ukoliko želimo da postojeću relacionu bazu modeliramo kao graf, treba obratiti pažnju na sledeća preslikavanja i korake:

- Svaki tabela entiteta predstavljena je kroz oznake na čvorovima;
- Svaka torka u tabeli entiteta predstavljena je kao čvor;
- Svaka kolona u tabeli entiteta predstavlja atribut čvora;
- Primarni ključevi sa tehničke strane ne postoje;
- Primarni ključevi sa biznis strane dodaju se kroz ograničenja jedinstvenosti;
- Podaci sa podrazumevanim (engl. *default*) vrednostima se ne čuvaju;
- Denormalizovani podaci i duplikati se dodaju kao posebni čvorovi;
- JOIN tabele modeliraju se kroz veze, a kolone kroz atribute odgovarajuće veze.

Modeliranje je apstrakcija koju motiviše određena potreba ili cilj. Ono što je karakteristično kod graf baza podataka je **bliskost logičkog i fizičkog modela**. Najbolje rečeno modeliranje grafa predstavlja pokušaj stvaranja strukture koja izražava pitanja koja želimo da postavimo domenu. Odnosno ***query-first* pristup**:

1. Opisati potrebe klijenta ili krajnjeg korisnika, kao i njihove ciljeve;
2. Konstruisati pitanja na osnovu prethodno sakupljenih potreba i ciljeva;
3. Identifikovati entitete i veze koje se pominju u konstruisanim pitanjima;
4. Prevesti entitete i veze u odgovarajuće izraze putanja (obilaska);
5. Izraziti ove putanje kroz konkretne Cypher naredbe.

Obično kroz pitanja i tekstualne opise možemo na osnovu vrsta reči utvrditi različita svojstva i ta su pravila skoro standardizovana na sledeći način:

- Imenice postaju oznake (Alumni, Fakultet, Kompanija, Dogadjaj);
- Glagoli postaju veze (RADI_U, POHADJAO_JE, UCESTVUJE_NA).

Zaključak

U radu je predstavljen i bliže okarakterisan način modeliranja graf baza podataka sa osvrtom na internu organizaciju podataka u bazi. Kroz poglavlja su opisani načini modeliranja i dobre prakse kod relacionih i graf baza podataka i slučajevi korišćenja za karakteristične primere. U ovom radu za praktični primer korišćena je projektovana baza za alumni članove fakulteta, čiji je ER dijagram prikazan na Slika6, a naredbe za kreiranje čvorova i veza u graf bazi objašnjene.

Neo4j je native graf baza podataka i ona efikasno implementira strukturu grafa čak 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 baze u Neo4j nije striktan, odnosno ona je schema-free, što znači da lako može evoluirati i implementirati promene kroz vreme.

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/