SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Martina Šestak

GRAF BAZE PODATAKA

ZAVRŠNI RAD

Varaždin, 2014.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Martina Šestak

Matični broj: 40112/11-R

Studij: Informacijski sustavi

GRAF BAZE PODATAKA

ZAVRŠNI RAD

Mentor:

Izv. prof. dr. sc. Kornelije Rabuzin

Varaždin, rujan 2014.

Sadržaj

1. Uvod

Baza podataka se najčešće definira kao organizirana kolekcija podataka koja opisuje stanje realnog sustava (Rabuzin K., 2011). Da bi se neka baza podataka mogla smatrati optimiziranom za upotrebu, potrebno je iz nje izbaciti sve suvišne (redundantne) podatke, odnosno normalizirati tu bazu podataka.

Svaka baza podataka se temelji na odgovarajućem modelu podataka, a danas se u praksi najčešće koristi relacijski model, a samim time i relacijske baze podataka. Međutim, zbog uvećanih zahtjeva poslovnih i inih situacija (primjerice, pohrana velike količine kompleksnih podataka, brže pretraživanje zapisa, fleksibilnija baza podataka i sl.), ali i rezultata korištenja relacijskih baza podataka u određenim područjima, nedostaci relacijskih baza podataka postaju vidljivi u performansama, ali i troškovima implementacije sustava. Kao alternativa relacijskim bazama u novije su se vrijeme nametnule tzv. NoSQL baze podataka, a pod tim se nazivom podrazumijeva nekoliko tipova baza podataka.

Svrha ovog rada je detaljnije opisati i prikazati jednu od tih alternativa, graf baze podataka (eng. *graph databases*) koje se sve više koriste u svakodnevnom radu. Osim općenitog opisa NoSQL baza podataka i njihove usporedbe s relacijskim bazama podataka, u radu će naglasak biti na temeljnim pojmovima vezanim uz graf baze podataka, na njihovim performansama tijekom pretraživanja, ali i načinu pohranjivanja podataka. Navedena će svojstva biti prikazana na primjeru Neo4j baze podataka.

Teorijske osnove rada s takvim bazama bit će primijenjene tijekom izrade web aplikacije za evidenciju filmova, glumaca i redatelja, gdje će se za pohranu podataka koristiti Neo4j. Osnovni dijelovi i funkcionalnosti aplikacije bit će također prikazani i detaljno opisani u radu.

2. NoSQL baze podataka

NoSQL (eng. *Not only SQL*, na početku *No to SQL*) je naziv koji obuhvaća nekoliko različitih tehnologija u radu s bazama podataka, a takve su se baze podataka pojavile kao alternativa relacijskim bazama podataka, u kojima se upiti izvode prvenstveno pomoću SQL naredbi.

Slika 2.1. Logotip NoSQL baza podataka
(Izvor: http://www.appdynamics.com/blog/wp-content/uploads/2011/05/nosql3.png)

Razlog porasta njihove popularnosti leži u nedostacima otkrivenim tijekom rada s relacijskim bazama podataka. Relacijske baze podataka su, dakako, još uvijek dominantne na tržištu, no sve se više korisnika okreće upotrebi upravo nekih od NoSQL baza podataka.

Način i kapacitet pohranjenih podataka pokazao se vrlo efikasnim i korisnim, a osim efikasne pohrane podataka (podaci se ne pohranjuju isključivo u relacije, nego u neke druge objekte), kao prednost NoSQL baza podataka ističu se dostupnost (*open source*) i distribuiranost, visoke performanse u rješavanju pojedinih zadataka, tolerancija na greške, a kao posebice važna ističe se skalabilnost (s.n., w3resource, 2014).

2.1. Karakteristike NOSQL baza podataka

Svoju prednost nad relacijskim bazama podataka NOSQL baze podataka ostvaruju na temelju sljedećih nekoliko karakteristika koje su detaljnije objašnjene u nastavku.

2.1.1. Model podataka

Korisnički zahtjevi su često promjenjivi pa je definiranje sheme u relacijskim bazama podataka vrlo zahtjevno i nepredvidivo. Jedanput definirana shema u relacijskim bazama podataka može se promijeniti na zahtjev korisnika, ali to utječe na trajanje i kompleksnost postupka izrade baze podataka. Podaci se u relacijskim bazama podataka pohranjuju u tablice, a za pretraživanje istih potrebno je provesti nekoliko spajanja nad tablicama što povećava vrijeme izvršavanja upita.

Kod NoSQL baza podataka model podataka je fleksibilniji, a podaci se ne spremaju u tablice, nego u različite objekte ovisno o njihovoj vrsti (dokumente, grafove itd.).

2.1.2. Skalabilnost

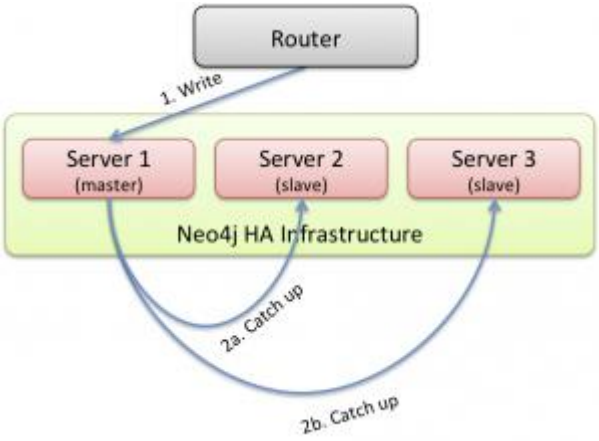
Zbog povećanog broja korisnika i povećane količine podataka koje treba pohraniti, potrebno je planirati način pohrane podataka na serverima. U tu svrhu razvijena su dva pristupa: vertikalno ili horizontalno skaliranje. Vertikalno skaliranje podrazumijeva dodavanje resursa serveru na kojem su pohranjeni podaci, dok se pod horizontalnim skaliranjem podrazumijeva dodavanje novih servera i pohranu podataka na više različitih fizičkih ili virtualnih servera. Tehnologija relacijskih baza podataka podržava tzv. vertikalnu skalabilnost, što povećava troškove neke aplikacije. Ovakav centralizirani pristup zahtijeva server s više resursa, a osim toga je nesigurniji ukoliko dođe do neočekivane greške.

Umjesto takvog pristupa NoSQL baze podataka omogućavaju tzv. horizontalnu skalabilnost. Ukoliko se broj korisnika ili količina podataka poveća, potrebno je samo dodati novi server, a ako se jedan od servera sruši, podacima se još uvijek može pristupiti preko drugih servera u klasteru.

2.1.3. Replikacija podataka

Svaki server u klasteru sadrži potpunu kopiju svih podataka u toj bazi podataka, pa se u slučaju pada nekog servera sprječava potpuni gubitak podataka. Ukoliko se neki server sruši, podaci za izvršavanje određenog upita će se dobaviti s druge instance servera u tom klasteru.

Kao i kod relacijskih baza podataka replikacija podataka se odvija po principu *master-slave* (Slika 2.3.). Svi podaci, koji se unose u bazu podataka, se pohranjuju na glavnom serveru (eng. *master node*) te se kasnije repliciraju na ostale, podređene, servere u klasteru (eng. *slave nodes*). Pritom ta replikacija može biti sinkrona (*slave* serveri su stalno povezani s *master* serverom pa se promjene odvijaju istodobno) ili asinkrona (*slave* serveri nisu stalno povezani s *master* serverom pa se promjene koje su se dogodile na *master* serveru pohranjuju u red i kasnije odvijaju na *slave* serveru) (Sprava, 2013).



Slika 2.3. "Master-slave" replikacija s „write master“ opcijom
(Izvor: http://jimwebber.org/wp-content/uploads/2011/02/2-300x221.png)

2.1.4. Mogućnost oporavka baze podataka

Jedan od iznimno važnih parametara za suvremene i kompleksne aplikacije jest oporavak baze podataka sa što manje posljedica. U stvarnosti su rijetki primjeri servera koji se nikad nisu srušili i neprekidno funkcioniraju pa je potrebno pravovremeno istražiti i izraditi strategiju oporavka ukoliko dođe do nekih problema sa serverom na kojem se nalazi baza podataka.

Kod NoSQL baza podataka pojedini dijelovi baze podataka su pohranjeni na više servera koji su međusobno hijerarhijski povezani odnosom „gospodar-sluga“ (eng. *master-slave*) opisanom u poglavlju 2.1.3.

Nakon pada servera sustav za upravljanje bazama podataka pregledava zapisnik aktivnih transakcija (eng. *active transaction log*) i ponovno aktivira one transakcije koje su se izvodile netom prije pada servera. Isti postupak odvija se na nekoliko servera pa će podaci upotrijebljeni u transakcijama biti međusobno konzistentni.

2.2. Usporedba NoSQL i relacijskih baza podataka

Usporedbu između NoSQL i relacijskih baza podataka moguće je napraviti s obzirom na nekoliko čimbenika, a neki od njih su (s.n., mongoDB Resources, 2014):

a) Model pohrane podataka
b) Postojanje/nepostojanje shema
c) Skalabilnost
d) Razvojni model
e) Upravljanje podacima
f) Konzistentnost podataka
g) Model izvođenja transakcija

Navedeni čimbenici i usporedba relacijskih i NoSQL baza podataka su detaljnije prikazani u tablici 1.

Tablica 1. Usporedba relacijskih i NOSQL baza podataka
(Prema: http://www.mongodb.com/learn/nosql)

	Relacijske baze podataka	NoSQL baze podataka
Model pohrane podataka	Pojedinačni zapisi se pohranjuju u redove u tablicama, gdje svaki stupac sadrži dio podatka o zapisu i naziva se atributom. Tijekom izvođenja upita spajaju se različite tablice, a time i različiti tipovi podataka, na temelju zajedničkih stupaca (atributa).	Model podataka se razlikuje ovisno o tipu baze podataka, pa se tako podaci mogu pohraniti kao skup parova ključ-vrijednost, u JSON ili XML formatu u dokument ili u čvorove i veze koji čine graf.
Sheme	Strukturu i tipove podataka potrebno je odrediti na početku, a tako definirana struktura se u potpunosti mijenja svaki put kad je potrebno dodati novi tip podataka ili novu vezu. Za vrijeme promjene strukture baza podatka je nedostupna korisnicima.	Struktura baze podataka je većinom dinamična budući da se podaci ne pohranjuju u tablice. Kod promjene strukture baza podataka je dostupna korisnicima i ne mijenja se u potpunosti. Struktura baze se može jednostavno mijenjati po potrebi.

Skalabilnost	Vertikalna, što zahtijeva server s puno resursa za pohranu velike količine podataka. Zbog centraliziranog pristupa sustav ima nisku toleranciju na greške.	Horizontalna, podaci su pohranjeni na nekoliko skupina fizičkih ili virtualnih servera, što omogućuje višu toleranciju sustava na greške.
Razvojni model	Neke baze podataka su *open source* (primjerice PostgreSQL, MySQL), dok su neke komercijalne (npr. Oracle, DB2).	Sve su baze podataka ove skupine *open source*.
Upravljanje podacima	Operacije nad podacima izvode se pomoću SQL naredbi.	Operacije se izvode preko objektno-orijentiranih programskih sučelja koja su međusobno različita i pristupa im se pomoću različitih programskih jezika.
Konzistentnost podataka	Ovakve baze podataka zahtijevaju, ali i pružaju strogu konzistentnost podataka u svakom trenutku.	Neki sustavi pružaju strogu konzistentnost podataka, dok neki tek eventualnu konzistentnost.
Model izvođenja transakcija	Relacijske baze podataka temelje se na ACID principu koji zahtijeva strogu konzistentnost podataka. Kod ovih sustava bitnija je konzistentnost podataka u svakom trenutku u odnosu na njihovu dostupnost.	BASE arhitektura sustava je pristup karakterističan za transakcije u NoSQL bazama podataka i predstavlja „ublaženi“, ali i tzv. optimističan oblik ACID arhitekture. Zbog toga su sustavi stalno dostupni korisnicima i općenito bolji u performansama. Mehanizmi kojima se to postiže su prilično jednostavni u usporedbi s kompleksnim mehanizmima ACID modela.

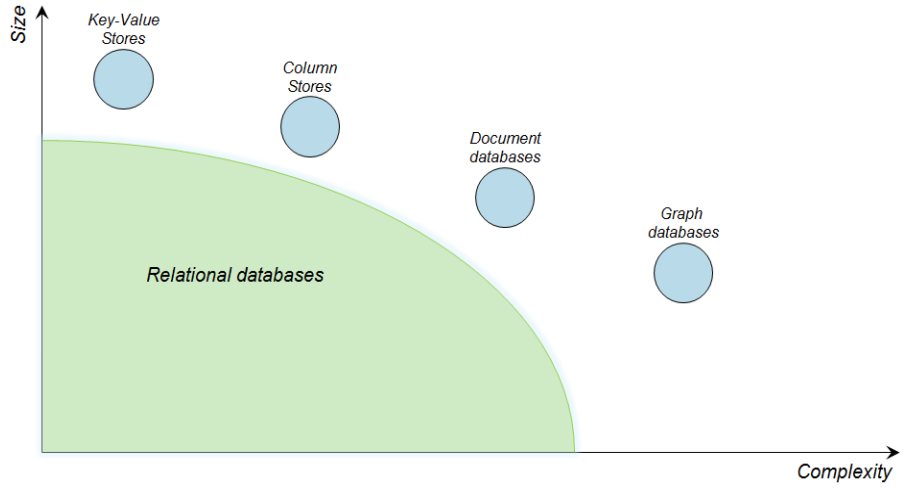
NoSQL baze podataka su, u usporedbi s relacijskim bazama podataka, puno bolje u postizanju kompromisa između fleksibilnosti modela podataka, dostupnosti i složenosti modeliranja i procesiranja složenijih struktura poput stabla i sl.

2.3. Vrste NoSQL baza podataka

Pod pojmom NoSQL podrazumijevaju se četiri različite kategorije baza podataka ovisno o korištenom modelu podataka:

1) **Ključ-vrijednost (eng. *key-value stores*)**
   - najjednostavnija i najčešće korištena skupina NoSQL baza podataka zbog svoje upotrebe u raznim programskim jezicima
   - podaci se spremaju u obliku parova ključ-vrijednost
   - pretraživanje podataka se izvršava na temelju ključa
   - BASE arhitektura
   - neučinkovitost u pretraživanju i ažuriranju određenog podskupa skupa vrijednosti (Rafique, 2013)
   - najpoznatiji predstavnici su Cassandra razvijena od strane Facebooka, Voldemort korišten na LinkedInu, DynamoDB korišten kod Amazona, itd.
2) **Temeljene na dokumentima (eng. *document databases*)**
   - Podaci pohranjeni u XML ili JSON formatu
   - Model podataka je kolekcija parova ključ-vrijednost, gdje je vrijednost složenija struktura – čitav dokument, što predstavlja višu razinu u odnosu na kategoriju ključ-vrijednost (s.n., mongoDB Resources, 2014)
   - Efikasnije dinamičko pretraživanje podataka
   - Najpoznatiji predstavnici ove kategorije su CouchDB i dr., a najpopularniji je MongoDB
3) **Graf baze podataka (eng. *graph databases*)**
   - Temelje se na teoriji grafova
   - Model podataka čine čvorovi, veze i svojstva
   - Najpoznatiji predstavnici su Neo4j i Infogrid
4) **Eng. *column oriented databases***
   - Model podataka čine tablice kod kojih su zajedno pohranjene vrijednosti stupaca
   - Koriste se za pohranu velike količine podataka

- Najpoznatiji predstavnici ove kategorije su Apache Hadoop/HBase, najpopularnija Apache Cassandra, BigTable u upotrebi Google-a, itd.



Slika 2.2. Ekosustav NoSQL baza podataka
(Izvor: http://blog.octo.com/wp-content/uploads/2012/07/QuadrantNoSQL.png)

Na slici 2.2. prikazan je tzv. ekosustav baza podataka u obliku grafa, gdje os apscisa predstavlja složenost, a os ordinata količinu podataka. Kao što je već spomenuto, NoSQL baze podataka priznate su kao moguća alternativa relacijskim bazama podataka zbog dokazanih prednosti u smislu fleksibilnosti sheme i modela podataka, dostupnosti sustava (zbog BASE arhitekture) itd. Kod odabira baze podataka potrebno je uzeti u obzir parametre koji su na slici 2.2. prikazani na koordinatnim osima, dakle, veličinu i složenost podataka koje u bazu podataka treba pohraniti i kasnije nad njima izvršavati različite operacije.

Kao što je to prikazano na slici 2.2., relacijske baze podataka mogu pohraniti ograničenu količinu podataka određene (ali i ograničene!) složenosti da bi njihove performanse tijekom izvođenja operacija bile zadovoljavajuće. Poveća li se vrijednost bilo kojeg od navedenih parametara, relacijske baze podataka će se pokazati kao nedovoljno efikasne i efektivne.

Baze podataka „ključ-vrijednost“ pružaju mogućnost pohrane velike količine podataka, ali manje složenosti, budući da se podaci pohranjuju u obliku parova ključ-vrijednost pa pohrana složenijih struktura poput stabla ili grafova nije moguća.

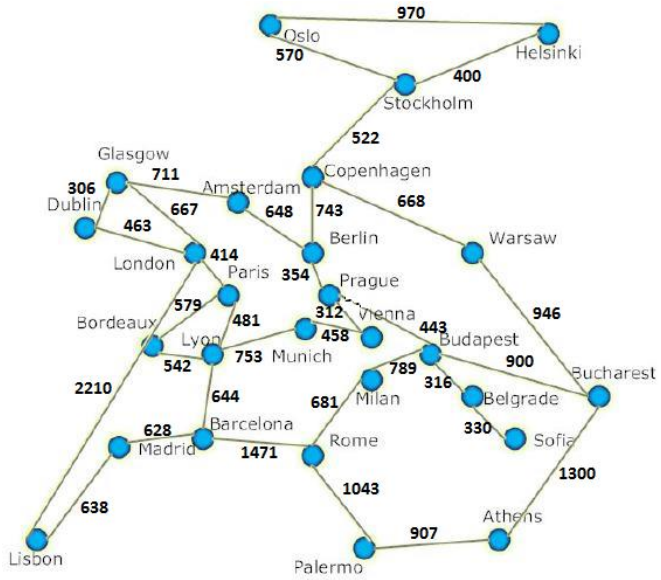
U bazama podataka sa stupčastim modelom pohrane podataka mogu se pohraniti malo manje količine podataka, ali podaci mogu biti složenije strukture.

Baze podataka temeljene na dokumentima mogu pohraniti manje količine podataka veće razine složenosti, budući da u parovima „ključ-vrijednost“ vrijednost predstavlja dokument unutar kojeg se mogu smjestiti složenije strukture.

Graf baze podataka pokazale su se kao kategorija baza podataka u kojima se mogu pohraniti najsloženiji podaci zbog njihove jake međusobne povezanosti, o čemu će više biti riječi u nastavku

3. Graf baze podataka

Graf baze podataka su kategorija NoSQL baza podataka koje svakodnevno povećavaju svoju zastupljenost na tržištu i čija popularnost među korisnicima redovito raste. Zahvaljujući sposobnosti pohrane i upravljanja velikom količinom jednostavnih, ali i složenih podataka, ova kategorija NoSQL baza podataka se upotrebljava na poznatim društvenim mrežama poput Facebooka, Twittera ili pak tražilicama poput Google-a. Najpoznatiji sustavi za upravljanje bazama podataka ove kategorije su Neo4j, FlockDB, InfiniteGraph, OrientDB itd. Danas se često upotrebljavaju zbog prirode suvremenih podataka za čiji je prikaz najčešće najbolji oblik grafa. Graf baze podataka mogu se upotrijebiti u prikazu povezanosti više osoba, određivanju najbliže udaljenosti između dvije lokacije, sustavima za preporuke, ali i kod vizualizacije pohranjenih podataka i sl.



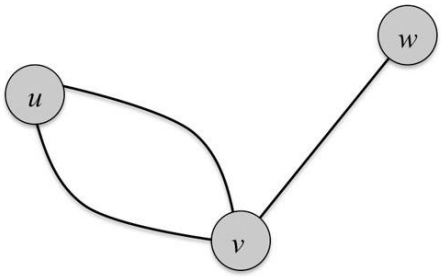
Slika 3.1. Prikaz udaljenosti različitih lokacija pomoću grafa
(Izvor: http://i1121.photobucket.com/albums/l503/rhose87/hartaoraselor.png)

Princip funkcioniranja graf baza podataka temelji se na teoriji grafova pa je, prije upoznavanja s ovom kategorijom NoSQL baza podataka, potrebno proučiti temeljne koncepte te teorije.

3.1. Teorija grafova

Prema definiciji graf je „uređena trojka G=(V,E,φ), gdje je V=V(G) neprazan skup čije elemente nazivamo **vrhovima**, E=E(G) je skup disjunktan s V čije elemente nazivamo **bridovima** i φ je funkcija koja svakom bridu e iz E pridružuje par {u,v}, ne nužno različitih vrhova iz V“ (Golemac i sur., 2012).

Iako se prema navedenoj definiciji graf predstavlja kao skup vrhova, bridova i funkcija koje povezuju te bridove i vrhove, u praksi se često graf jednostavno prikazuje kao skup vrhova i bridova. Na slici 3.2. tako je prikazan graf s tri vrha (*u*, *v*, w) i tri brida (dva brida povezuju vrhove *u* i *v* te jedan povezuje vrhove *v* i *w*). Graf prikazan na toj slici se naziva i višestrukim jer postoje dva brida koji povezuju iste vrhove (dva brida između vrhova *u* i *v*). Grafom se može smatrati i skup koji sadrži samo jedan vrh, gdje brid počinje i završava u tom vrhu te tako čini petlju.



ida

(Izvor: http://e.math.hr/sites/default/files/br14/figure3.png)

Šetnja grafom je „alternirajući niz vrhova i bridova, koji počinje i završava vrhom, u kojem je svaki vrh incidentan bridu koji mu prethodi i bridu koji mu slijedi u tom nizu, a vrh koji prethodi bridu i vrh koji slijedi taj brid krajnji su vrhovi tog brida“ (Fošner i Kramberger, 2009). Graf se smatra povezanim ako postoji put između bilo koja dva vrha tog grafa.

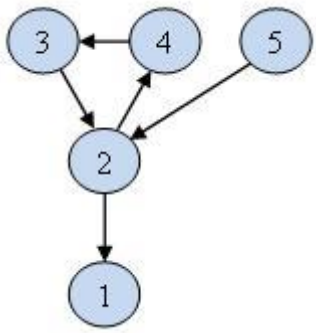
S obzirom na usmjerenost bridova u grafu, on se može smatrati:

a) **Usmjerenim**
b) **Neusmjerenim**

3.1.1. Usmjereni graf

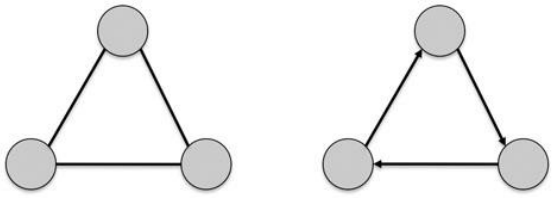
Prema definiciji usmjereni graf ili digraf je „uređena trojka D=(V,A,ψ), gdje je V=V(D) neprazan skup čije elemente nazivamo **vrhovima**, A=A(D) skup disjunktan s V čije elemente nazivamo **lukovima** i ψ funkcija koja svakom luku a iz A pridružuje uređeni par (u,v), pri čemu u,v∈V nisu nužno različiti“ (Golemac i sur., 2012). Pritom se vrh *u* smatra izvorištem, a *v* odredištem luka, a smjer tog luka se prikazuje kao *a=(u,v)*. U praksi se često usmjereni graf prikazuje kao skup vrhova i lukova.

Na slici 3.3. prikazan je jedan usmjereni graf s ukupno 5 vrhova povezanim pomoću 5 lukova. Kod ovakvih grafova potrebno je uzeti u obzir smjerove tih lukova jer nije isti slučaj kad se šetnja grafom odvija, primjerice, od vrha 1 do vrha 2 ili obratno.



Slika 3.3. Prikaz usmjerenog grafa s 5 vrhova i 5 lukova
(Izvor: http://www.zemris.fer.hr/predmeti/rg/diplomski/08Popovski/graphics/slika17b.jpg)

Pojmovi i definicije vezane uz općenite grafove vrijede i za usmjereni graf. Neusmjereni graf, za razliku od usmjerenog, nema usmjerene bridove koji povezuju vrhove (Slika 3.4.).



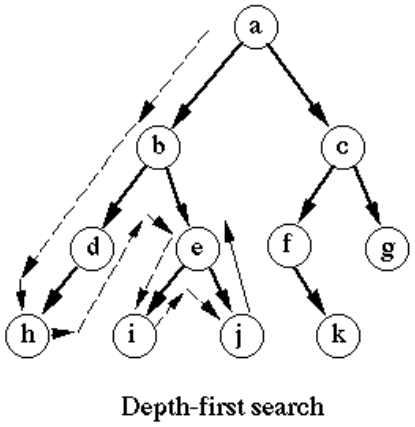
Slika 3.4. Prikaz neusmjerenog i usmjerenog grafa
(Izvor: http://e.math.hr/sites/default/files/br14/figure2.png)

3.1.2. Algoritmi pretraživanja grafa

Jedna od osnovnih operacija koje se izvode nad grafom je obilazak grafa, a od velike je koristi prilikom traženja određenog čvora u grafu. Svaki obilazak grafa započinje određivanjem početnog čvora, s kojeg se pomoću veza kreće prema ostalim čvorovima.
Za obilazak grafa se najčešće koriste dva algoritma:

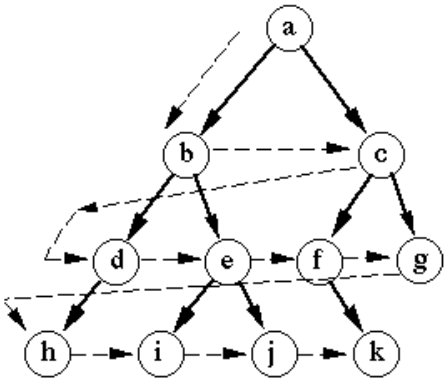
1. **Algoritam pretraživanja u dubinu** (eng. *depth-first search algorithm),* gdje se graf obilazi tako da se od početnog čvora čim brže dođe do rubnih čvorova (eng. *leaf nodes*). Pritom se graf može obilaziti najprije od čvora prema njegovoj djeci (*preorder* obilazak), najprije od djece prema čvoru (*postorder* obilazak) ili najprije nekoliko djece, pa čvor, pa ostatak djece (*inorder* obilazak).
2. **Algoritam pretraživanja u širinu** (eng. *breadth-first search algorithm*), gdje se graf obilazi razinu po razinu

Na prikazanom grafu od 11 čvorova bi se algoritmom pretraživanja u dubinu graf obilazio algoritmom *preorder* obilaska na sljedeći način : *a b d h e i j c f k g* (Slika 3.5.).



Slika 3.5. Algoritam pretraživanja grafa u dubinu
(Izvor: http://www.cse.unsw.edu.au/~billw/Justsearch.html)

Isti bi se graf algoritmom pretraživanja u širinu obilazio redoslijedom: *a b c d e f g h i j k* (Slika 3.6.).



Slika 3.6. Algoritam pretraživanja u širinu
(Izvor: http://www.cse.unsw.edu.au/~billw/Justsearch.html)

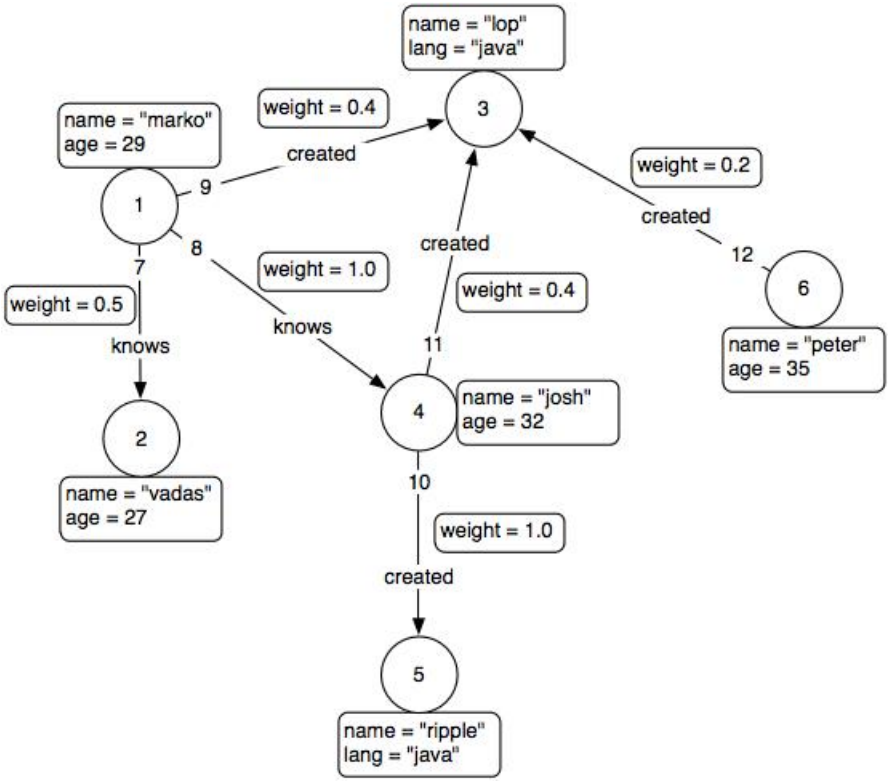
Navedeni algoritmi pretraživanja grafa se koriste kao polazište algoritmima pretraživanja u upitnim jezicima koji su objašnjeni u poglavlju 4.

3.2. Model grafa sa svojstvima

Model grafa sa svojstvima (eng. *property graph model*) je najčešće korišten model za implementaciju graf baza podataka.

Svaki model grafa sa svojstvima se, budući da on ustvari predstavlja graf, sastoji od skupa vrhova (eng. *vertices*) i bridova (eng. *edges*). Kao što postoje pravila koje treba poštivati tijekom izrade relacijskog modela baze podataka, tako su i uz izradu modela grafa sa svojstvima vezana neka pravila (s.n., TinkerPop, 2012):

1. Svaki vrh i brid imaju svoje jedinstvene identifikatore
2. Svaki vrh ima bridove koji u njega ulaze i iz njega izlaze
3. Svaki vrh i brid imaju skup svojstava pohranjenih po principu ključ-vrijednost
4. Svaki brid ima početni vrh (izvorište) i završni čvor (odredište)
5. Svaki brid ima oznaku (eng. *label*) koja označava tip veze koja povezuje dva vrha



Slika 3.7. Prikaz modela grafa sa svojstvima
(Izvor: http://www.infoq.com/resource/news/2010/01/Gremlin/en/resources/graph1.jpg)

Na slici 3.7. prikazan je primjer model grafa sa svojstvima u kojem su prikazane veze između pojedinih osoba i naziva programa koje su oni izradili. Model se sastoji od 6 vrhova i 6 bridova. Svaki vrh ima svoj identifikator pa ako, primjerice, pogledamo vrh s identifikatorom 1, možemo vidjeti kako on ima svojstvo *name* i *age*, a vrijednosti tih svojstava su *marko* i *29*. Ovaj vrh ima samo izlazne bridove s identifikatorima 7, 8 i 9. Brid s identifikatorom 7 ima izvorište u vrhu s identifikatorom 1, a odredište u vrhu s identifikatorom 2. Oznaka ovog brida je *knows* a njegovo svojstvo je *weight* s vrijednošću 0.5. Na temelju oznake možemo vezu prikazati u obliku rečenice: *Marko poznaje Vadasa* (eng. *Marko knows Vadas*). Na isti se način mogu analizirati i ostali vrhovi i bridovi.

3.3. Modeliranje graf baza podataka

Kao i kod svakog modeliranja, prvi korak je upoznavanje domene modeliranja i razumijevanje predmeta modeliranja te veza i pravila po kojima su oni međusobno povezani.

Kod relacijskih baza podataka se takav konceptualni model domene prikazuje u obliku E-R ili ERA dijagrama, logičkog modela koji se može smatrati grafom. Entiteti tog dijagrama predstavljaju čvorove grafa, a koncept vanjskog ključa ili asocijativnog entiteta za povezivanje dva ili više entiteta predstavlja vezu između dva čvora. Međutim, ERA dijagramom mogu se prikazati samo jednostavne i, opcionalno, imenovane veze, što dovodi do zaključka da se modelom grafa u graf bazama podataka može detaljnije prikazati problemska domena.

Kod modeliranja graf baza podataka nakon početne analize domene i identificiranja čvorova i veza koje ih povezuju slijedi dodatno usavršavanje modela grafa. To znači da se ponovno provjeravaju čvorovi (jesu li svi entiteti domene prikazani u obliku čvora, jesu li atributi čvora potpuni itd.) i veze među njima (jesu li sve veze koje postoje među entitetima u stvarnosti prikazane na modelu grafa, sadrže li one ispravna svojstva i sl. Upotpunjavanje modela domene vodi k potpunijem i točnijem modelu grafa.

Za razliku od relacijskih baza podataka, gdje se identificirani entiteti i veze među njima prikazani na ERA dijagramu u bazi podataka transformiraju u tablice u bazi podataka, taj korak kod graf baza podataka nije potreban. Prema tome, početni konceptualni model u kojem se identificiraju čvorovi i veze među njima je upravo ono što se pohranjuje u bazi podataka, a time se povećava povezanost između stvarne situacije i situacije prikazane modelom grafa.

Postoji nekoliko načina pomoću kojih se može provjeriti točnost i ispravnost izrađenog modela grafa, odnosno nekoliko pravila koja bi točan i ispravan model grafa trebao zadovoljavati (Robinson i sur., 2013):

1. Model grafa mora biti čitljiv i logički ispravan, tj. smislen
   - Cijeli model grafa se može interpretirati kao niz smislenih, logički povezanih rečenica, bez praznog hoda.
2. Na modelu grafa se treba primijeniti koncept **dizajna za pretraživanje** (eng. *design for queryability*)

- Ovim pravilom osigurava se da se nad grafom mogu izvoditi upiti koji su od značaja za krajnje korisnike (prikazano u obliku slučajeva korištenja korisnika), a da bi se to moglo, prethodno je potrebno opisati navedene upite.
- Model grafa se može smatrati točnijim ukoliko se za svaki slučaj korištenja krajnjih korisnika može kreirati upit nad grafom koji će biti ispravan i vraćati željene rezultate.

3.4. Karakteristike graf baza podataka

Model grafa sa svojstvima je implementiran u graf bazama podataka. U takvim bazama podataka su implementirani principi i definicije teorije grafova uz male izmjene u terminologiji – pojedini vrh iz teorije grafova u bazama podataka predstavlja **čvor**, dok se brid koji povezuje vrhove naziva **vezom** između dva čvora. Pritom ne postoje neka pravila za imenovanje tih veza, ali se u praksi one zbog lakše čitljivosti najčešće pišu velikim početnim slovima (time se nazivi veza mogu jasno razlikovati od naziva oznaka čvorova ili samih čvorova).

Graf baza podataka, odnosno sustav za upravljanje takvom bazom podataka, može se općenito definirati kao „online sustav za upravljanje bazom podataka s metodama za kreiranje, čitanje, ažuriranje i brisanje (eng. *Create, Read, Update, Delete*) za prikaz modela podataka“ (Robinson i sur., 2013).

Zahvaljujući svojoj arhitekturi, a time i performansama u izvršavanju zadataka, graf baze podataka postaju sve češći izbor za pohranu velike količine kompleksnih podataka. Ta sposobnost postoji zahvaljujući sljedećim svojstvima:

1) **Načinu pohrane podataka**

   Podaci se u graf bazama podataka pohranjuju u obliku grafa (eng. *native graph storage*) ili u neke druge vrste baza podataka (relacijske, objektno-orijentirane i sl.). Kod prvog načina pohrane podataka svaki čvor grafa je „fizički“ povezan sa svojim susjednim čvorovima, tj. sadrži pokazivače na te čvorove, što ubrzava izvođenje upita nad graf bazom podataka.

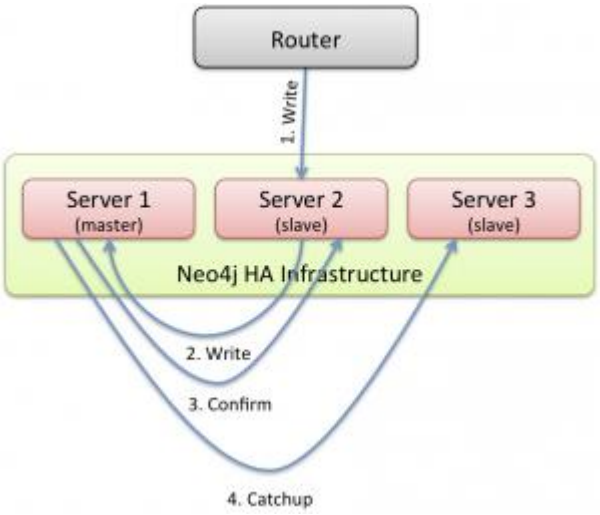
2) **Mehanizmu obrade podataka**

   Mehanizam obrade grafa se odnosi na „tehnologiju koja omogućuje algoritmima obradu velikih količina podataka u grafu“ (Robinson i sur., 2013). Naglasak kod ovakvih mehanizama je na globalnim upitima kojima se utvrđuje odnos

između podataka na višoj, generalnoj razini (primjerice, za graf koji prikazuje popis filmova i glumaca koji u njima glume ovaj mehanizam se koristi da bi se dobio odgovor na upit u koliko je filmova neki glumac glumio i sl.).

3) **Dostupnosti i replikaciji baze podataka**

Neke graf baze podataka (npr. Neo4j) pružaju mogućnost unosa podataka na *slave* server, što ih čini fleksibilnijima, budući da korisnička aplikacija može unositi podatke na bilo koji server (Slika 3.8.). Nakon zapisivanja podataka u *slave* server podaci se moraju uskladiti i na *master* serveru, a nakon toga ona šalje potvrdu *slave* serveru nakon koje se ponovno zapisuju podaci.



Slika 3.8. "Master-slave" replikacija s "write slave" opcijom
(Izvor: http://jimwebber.org/wp-content/uploads/2011/02/3-300x255.png)

4) **Skalabilnosti**

Kao što je već spomenuto u poglavlju 2.2, tehnologija NoSQL baza podataka pruža mogućnost horizontalne skalabilnosti. Kod graf baza podataka performanse sustava nisu toliko ovisne o kapacitetu (ukupan broj čvorova i veza koje se mogu pohraniti) koji zauzimaju podaci u bazi podataka, a ni o vremenu odgovora, pa se upiti izvršavaju jednakom brzinom, bilo da je u bazi podataka pohranjeno 10 ili 100 čvorova, jer izvršavanje upita ovisi o vremenu potrebnom za pronalaženje početnog čvora, a kasnije se upit izvodi prema određenom uzorku. Novi podaci, odnosno čvorovi grafa, se mogu dodati bez prevelikog utjecaja na ostatak postojećeg grafa.

3.5. Usporedba relacijskih i graf baza podataka

U tablici 2. prikazana je usporedba relacijskih i graf baza podataka s obzirom na nekoliko parametara:

- Shemu podataka
- Upitne jezike
- Performanse (brzina izvođenja upita nad bazom podataka)
- Način pohrane podataka
- Mehanizam povezivanja, tj. ostvarivanja veza između podataka

Tablica 2. Usporedba relacijskih i graf baza podataka
(Prema: Robinson i sur., 2013)

	Relacijske baze podataka	Graf baze podataka
Shema podataka	Fiksna. Svaka tablica u bazi podataka je kolekcija fiksnih atributa i njihovih vrijednosti. Bilo kakve strukturne promjene nad bazom podataka, tzv. migracije (dodavanje, ažuriranje ili brisanje tablica i njihovih atributa) zahtijevaju puno vremena i novaca.	Fleksibilna. Ažuriranje grafa se provodi na jednostavan način uz malo utrošenog vremena i novca (potrebno je samo locirati dio grafa nad kojim se žele provesti promjene).
Upitni jezici	SQL	Cypher, Gremlin
Performanse	Kod manjih baza podataka performanse su zadovoljavajuće i izvođenje upita nad bazom podataka je dovoljno brzo. Međutim, kod većih baza podataka brzina izvođenja upita se smanjuje, budući da se pretražuju svi podaci u bazi podataka da bi se pronašli oni koji zadovoljavaju kriterije navedene u upitu (velik broj JOIN operacija tijekom izvođenja upita). Zbog toga je potrebno više vremena da bi se takvi podaci pronašli. (Adell, 2011)	Kod manjih baza podataka performanse su približno jednake performansama relacijskih baza podataka. S porastom količine podataka u bazi podataka, performanse ove kategorije NOSQL baza podataka su bolje u odnosu na performanse koje pružaju relacijske baze podataka u tom slučaju. Upiti se izvršavaju brže jer se traže samo oni podaci koji su direktno povezani s podacima navedenim u upitu. Upiti se izvršavaju samo nad određenim dijelom grafa koji zadovoljava kriterije opisane u upitu. Pronalaženje susjednih čvorova u grafu je jednostavno, budući da svaki čvor sadrži

		direktni pokazivač na njemu susjedan čvor, bez ikakvih indeksa.
Način pohrane podataka	Podaci se pohranjuju u tablice.	Podaci se pohranjuju u čvorove.
Mehanizam povezivanja podataka	Veze među podacima pohranjenim u različitim tablicama ostvaruju se pomoću vanjskih ključeva ili dodatnih tablica.	Veze među podacima se ostvaruju izravno povezivanjem čvorova, gdje veze pritom mogu biti imenovane i sadržavati vlastite atribute.

Na temelju prikazane usporedbe može se zaključiti kako su relacijske baze podataka pogodne za upotrebu u situacijama kad u bazu podataka nije potrebno pohraniti veću količinu podataka (primjerice tradicionalne poslovne aplikacije) te u kojima je u početku poznata shema podataka koji će pohraniti u bazi podataka (tabelarni i dobro strukturirani podaci), a koja se u budućnosti neće često mijenjati.

Suprotno tome, u situacijama u kojima je u bazu podataka potrebno pohraniti veliku količinu podataka s puno korisnika (primjerice suvremene društvene mreže), a u kojima shema podataka nije u potpunosti poznata u početku i često se mijenja tijekom vremena, poželjno je koristiti graf baze podataka.

Prema tome, relacijske baze podataka su **orijentirane shemi** (eng. *schema-oriented*), dok su graf baze podataka **orijentirane pojavama** (eng. *occurence-oriented*) (Zicari, 2013).

3.6. Prednosti i nedostaci graf baza podataka

Na slici 3.9. prikazane su najčešće prednosti graf baza podataka zbog koji se sve više korisnika odlučuje za njih, ali i nedostaci čiji uzrok uglavnom leži u kratkoj povijesti ove kategorije NOSQL baza podataka.

Prednosti

- Brže izvođenje upita neovisno o količini podataka zahvaljujući svojstvu susjedstva slobodnog indeksa, čime se u kratkom roku mogu utvrditi veze među čvorovima grafa
- Mogućnost reprezentacije povezanih podataka na prirodan način kao skup objekata (čvorova) povezanih skupom objekata (veza).
- Programer može odmah početi s izradom aplikacije, budući da se model domene može izraditi u vrlo kratkom roku i nisu potrebni postupci normalizacije i denormalizacije.
- Fleksibilnost i elastičnost na moguće promjene tijekom razvoja ili nakon isporuke aplikacije.
- Jednostavno modeliranje domene i korisničkih slučajeva.

Nedostaci

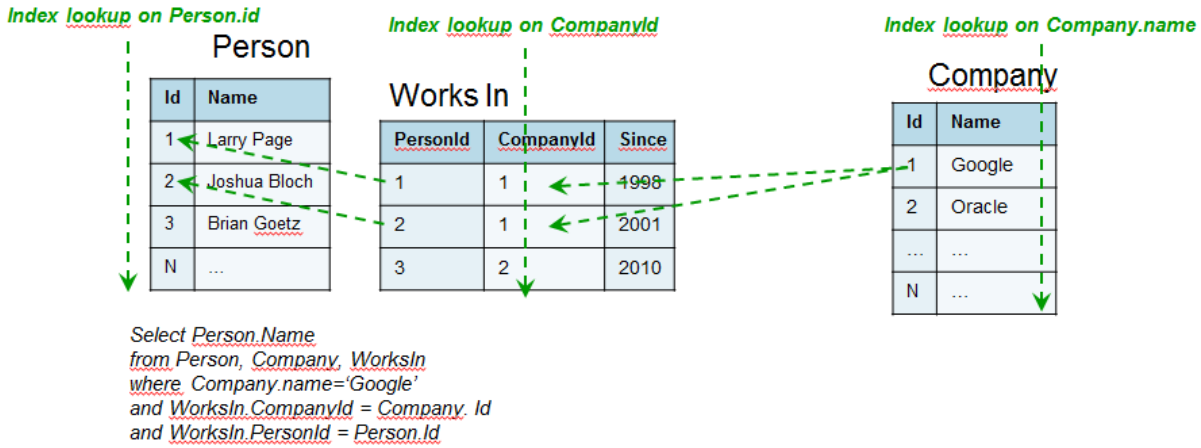
- Niska razina zrelosti (razina detaljnosti testiranja i starost sustava) i podrške, budući da je ova kategorija relativno nova, a relacijske baze podataka su još uvijek najkorištenija vrsta baza podataka.
- Nedostatak podrške za upravljanje većim brojem korisnika zbog čega upravljanje korisnicima mora biti implementirano izravno u aplikaciji.
- Veći troškovi implementacije u praksi jer, da bi se spriječila smanjenja u performansama u slučaju više korisnika, graf baze podataka moraju biti implementirane i pohranjene na jednom serveru.

Slika 3.9. Prednosti i nedostaci graf baza podataka
(Prema: Eifrem, 2012)

U nastavku će na primjeru (Slike 3.10. i 3.11.) biti detaljnije objašnjen način izvođenja upita u relacijskim i graf bazama podataka. Slike prikazuju baze podataka u kojima se nalaze podaci o zaposlenicima koji su zaposleni u poduzećima. Uzmimo sljedeći slučaj: želimo saznati koje osobe rade u kojem poduzeću i koliko dugo.

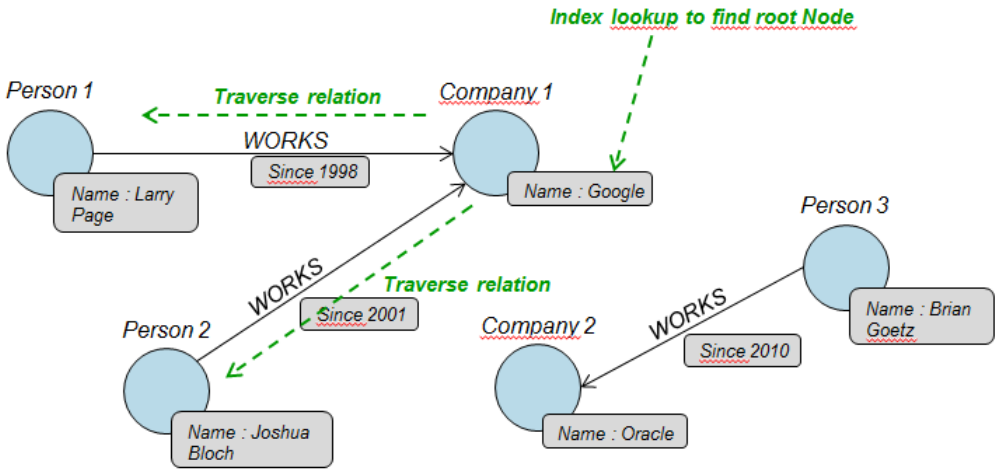
Model domene se u relacijskim bazama podataka implementira kao 3 tablice (*Person, WorksIn, Company*), gdje tablica *WorksIn* predstavlja tablicu s dvokomponentnim primarnim ključem koja je dodana zbog veze više-više između preostale 2 tablice. Da bismo saznali koje osobe rade u poduzeću Google, potrebno je ukupno pretražiti 3 indeksa. Najprije se pretražuje indeks tablice *Company* i traži se poduzeće čiji je naziv jednak Google. Nakon toga se izvršava JOIN operacija nad tablicama *Company* i *WorksIn* prema jednakim ID

poduzeća (*CompanyId*). Potom se pretražuje indeks u potrazi za ID osobe (*PersonId*) koja ima odgovarajući ID poduzeća (*CompanyId*). Nakon što saznamo ID te osobe izvršava se još jedna JOIN operacija nad tablicama *Person* i *WorksIn* kojom dobivamo rezultat koji sadrži ID osoba koji je jednak u obje tablice. Potom se indeks pretražuje prema ID osobe na temelju kojeg možemo saznati imena osoba koja rade u poduzeću Google.



Slika 3.10. Primjer izvođenja upita u relacijskoj bazi podataka
(Izvor: http://blog.octo.com/wp-content/uploads/2012/07/RequestInSQL.png)

U graf bazi podataka model domene se transformira u skup čvorova osoba i poduzeća koji su međusobno povezani imenovanim usmjerenim vezama *WORKS* koje sadrže atribut sa značenjem otkad rade u tom poduzeću. Prilikom izvršavanja istog upita nad graf bazom podataka pretražuje se indeks koji sadrži čvorove poduzeća. Pronalazi se čvor poduzeća gdje je vrijednost atributa *Name* jednaka Google i taj se čvor uzima kao korijenski.



Slika 3.11. Primjer izvođenja upita u graf bazi podataka
(Izvor: http://blog.octo.com/wp-content/uploads/2012/07/RequestInGraph.png)

Budući da korijenski čvor sadrži direktne pokazivače na svoje susjedne čvorove, pomoću veza obilaska dobivamo njegove susjedne čvorove i to predstavlja rezultat upita. Vrijeme izvođenja će uglavnom ovisiti o vremenu potrebnom za pronalaženje korijenskog čvora, a nakon toga će se upit izvoditi konstantno.

4. Upitni jezici u graf bazama podataka

Prilikom izvršavanja svakog upita u graf bazi podataka u pozadini se izvršava neki algoritam pretraživanja grafa koji se, pak, temelji na algoritmima pretraživanja u širinu ili dubinu objašnjenim u poglavlju 3.1.2.

Najčešći algoritmi pretraživanja kojima se koriste upitni jezici u graf bazama podataka su (Robinson i sur., 2013):

1. **Dijkstrin algoritam**

   Ovaj se algoritam temelji na algoritmu pretraživanja u širinu, a koristi se za izračunavanje najkraćeg puta između dva čvora u grafu, što se, primjerice, može upotrijebiti u sustavima za navigaciju. Općenito se smatra vrlo učinkovitim zahvaljujući činjenici da izračunava najkraći put između čvorova unutar manjeg dijela grafa.

2. **A* algoritam**

   Ovaj algoritam predstavlja spoj karakteristika Dijkstrinog algoritma (pretražuje čvorove bliske početnom čvoru) i algoritma pretraživanja najboljeg čvora (eng. *best-first search algorithm*) u kojem prednost imaju čvorovi bliski odredišnom čvoru.

Ovakvi se algoritmi koriste prilikom pretraživanja graf baza podataka pomoću upitnih jezika, a najčešći su Cypher i Gremlin. U nastavku će biti prikazane osnove Cypher upitnog jezika, budući da se taj upitni jezik zbog jednostavnosti i drugih svojstava najčešće koristi u praksi.

4.1. Cypher – struktura i najčešće klauzule

Prema definiciji Cypher je „deklarativni upitni jezik grafova koji omogućuje ekspresivno i efikasno izvršavanje upita i ažuriranje grafa“ (s.n., The Neo4j Manual v2.1.2, 2014). Vrlo je jednostavan za učenje i omogućuje korisnicima da svoju pažnju usmjere na problemsku domenu i upite koji im trebaju umjesto na način povezivanja s bazom podataka. Za razliku od proceduralnog jezika Gremlina, Cypher kao deklarativni upitni jezik definira samo koje podatke iz grafa će prikazati, ali ne i kako će to učiniti. Upiti se izvršavaju na temelju uzoraka zadanih u klauzuli MATCH koji predstavljaju putanje između čvorova. Svaki čvor ima soje atribute koji su pohranjeni kao parovi ključ-vrijednost.

Cypher je dio svoje strukture posudio iz upitnog jezika relacijskih baza podataka SQL-a pa se upiti pišu u obliku klauzuli (eng. *clauses*) kojih može biti više unutar jednog upita.

U nastavku su navedene i na primjerima ukratko objašnjene najčešće korištene klazule, a svi primjeri Cypher upita bit će prikazani nad bazom podataka koja se koristi u aplikaciji Neo4jBook opisanoj u poglavlju 6.

4.1.1. Klauzula MATCH

Ova klauzula predstavlja temelj za razumijevanje Cypher upitnog jezika i načina pretraživanja graf baza podataka. U njoj je naveden uzorak prema kojem se pretražuju čvorovi i veze unutar grafa, a nakon pronalaženja početnog čvora (izvorišta) identificira se podgraf u kojem se nastavlja pretraživanje. Korisnik precizno definira koji će mu se podaci prikazati kao rezultati izvođenja upita pomoću klauzule RETURN, broj rezultata klauzulom LIMIT, a rezultate sortira (s obzirom na svojstva, a ne same čvorove i veze) klauzulom ORDER BY.

Primjerice, upit koji vraća nazive prvih 5 knjiga u bazi podataka glasi ovako:

```
MATCH (k:Knjiga)
RETURN k.Naziv
LIMIT 5
```

Upit koji vraća nazive knjiga i književnih žanrova kojima one pripadaju, koji su u bazi podataka povezani vezom tipa :PRIPADA, glasi:

```
MATCH (a:Knjiga)-[r:PRIPADA]->(b:Žanr)
RETURN a.Naziv, b.Naziv
```

Međutim, osim ove klauzule, za definiranje uzorka pretraživanja može se koristiti i klauzula OPTIONAL MATCH. Ova klauzula ima sličnu ulogu kao klauzula MATCH, međutim, ukoliko se njome ne pronađu rezultati za dijelove zadanog uzorka u upitu, ti će dijelovi poprimiti vrijednost NULL. Tako bi se, primjerice, prethodni upit izvršio i za knjige za koje nije definirano kojem žanru pripadaju.

4.1.2. WHERE

Naredba WHERE nije klauzula u punom smislu te riječi, već se samo koristi u kombinaciji s klauzulama MATCH i OPTIONAL MATCH za filtriranje rezultata dobivenih tim klauzulama, pa je time njezina uloga jednaka istoimenoj klauzuli u SQL jeziku.

Primjerice, upit koji vraća nazive knjiga koje je posudio Korisnik Pero Perić glasi:

```
MATCH (a:Korisnik)-[r:POSUDIO]->(b:Knjiga)
WHERE a.Ime='Pero' AND b.Prezime='Perić'
RETURN b.Naziv
```

U ovom upitu se najprije klauzulom MATCH pronalaze korisnici koji su posudili neku knjigu, a zatim se taj rezultat filtrira na samo oni korisnici koje se zovu Pero Perić.

4.1.3. Klauzule CREATE i DELETE

Klauzula CREATE se koristi za kreiranje čvorova i veza. Prilikom kreiranja čvorova i veza mogu im se dodijeliti atributi. Primjerice, dodavanje novog autora u bazu podataka i dodjeljivanje knjige tom autoru odvija se izvođenjem upita:

```
CREATE a = (b:Autor { Ime: 'Amelie', Prezime: 'Nothomb' })-[:NAPISAO]->(c:Knjiga { Naziv: 'Protiv Katiline' })
RETURN a
```

Ovim upitom se u klauzuli CREATE definira čitav uzorak te se u jednom koraku, ukoliko oni u trenutku izvođenja upita ne postoje, kreiraju novi čvorovi i veze između tih čvorova.

Klauzulom DELETE moguće je obrisati čvor, ali i sve njegove veze prema drugim čvorovima. Za brisanje Korisnika Pere Perića i svih njegovih veza (u ovom slučaju posudbi) upit glasi:

```
MATCH (a:Korisnik { Ime: 'Pero', Prezime: 'Perić' })-[r]-()
DELETE a, r
```

Veza definirana u MATCH klauzuli nije usmjerena, budući da se traži uzorak u kojem je Korisnik Pero Perić početni, ali i završni čvor veze.

4.1.4. Klauzule SET i REMOVE

Klauzula SET se koristi za dodavanje oznaka i atributa čvorovima te dodavanje atributa vezama, odnosno ima ulogu sličnu klauzuli UPDATE u SQL jeziku.

Primjerice, za dodavanje atributa Godina objave Knjigu „Protiv Katiline“ potrebno je izvesti sljedeći upit:

```
MATCH (a:Knjiga { Naziv: 'Protiv Katiline' })
SET a.Godina_objave='2004'
RETURN a
```

Klauzula REMOVE ima ulogu suprotnu klauzuli SET; koristi se za brisanje svojstava kod čvorova i veza.

4.1.5. Klauzula MERGE

Ova klauzula predstavlja spoj klauzuli MATCH i CREATE. Naime, njom se osigurava da definirani uzorak postoji u grafu. Ukoliko definirani uzorak ne postoji unutar grafa, taj se uzorak kreira (kreiraju se novi čvorovi i veze). Pritom se može specificirati koji se upit izvršava ukoliko uzorak postoji (ON MATCH) ili ukoliko ne postoji pa ga je potrebno kreirati (ON CREATE).

Upit kojim se provjerava postoji li u bazi podataka Korisnik „Ana Anić“ i, ukoliko ne postoji, kreira tog korisnika glasi:

```
MERGE (a:Korisnik { Ime: 'Ana', Prezime: 'Anić' })
RETURN a
```

4.1.6. Klauzula START

Klauzula START se koristi za definiranje početnog čvorova ili veze u indeksima (moraju postoji, inače upit vraća grešku). Pritom se čvor ili veza mogu pronaći pretraživanjem indeksa ili na temelju identifikatora.

Primjerice, da bismo u indeksu naziva „autori“ pronašli čvor naziva „Amelie Nothomb“ potrebno je izvršiti sljedeći upit:

```
START n=node:autori(Ime = ''Amelie'', Prezime = ''Nothomb'')
RETURN n
```

Isti čvor, ukoliko je npr. njegov ID jednak 1, možemo pronaći na sljedeći način:

```
START n=node(1)
RETURN n
```

Kao i u SQL-u, pomoću Cypher upitnog jezika je moguće kreirati funkcije te indekse i implementirati različite operacije nad podacima (agregacije, unije i sl.).

5. Neo4j

Neo4j je prema definiciji „robusna, skalabilna baza podataka visokih performansi“ (s.n., The Neo4j Manual v2.1.2, 2014) pa se u praksi može koristiti za pohranjivanje svih ili samo dijela podataka s glavnih servera velikih poduzeća. Tipična je *open-source* graf baza podataka razvijena od strane Neo Technology kompanije.

Slika 5.1. Logotip Neo4j baze podataka
(Izvor: s http://www.neotechnology.com/wp-content/uploads/2013/04/neo4j_notag_whitebg.png)

Glavne karakteristike ove graf baze podataka su (s.n., The Neo4j Manual v2.1.2, 2014):

a) **ACID model transakcija** koji se razlikuje od tipičnog BASE principa korištenog u NoSQL bazama podataka, a kojim Neo4j garantira konzistentnost podataka u svakom trenutku
b) **Visoka razina dostupnosti** podataka krajnjim korisnicima
c) **Jednostavno skaliranje** u slučaju potrebe za dodavanjem čvorova i veza s manjim, gotovo zanemarivim utjecajem na performanse sustava
d) **Brzo izvođenje upita** zahvaljujući učinkovitim algoritmima obilaska povezanih čvorova u grafu i korištenjem deklarativnog upitnog jezika Cypher
e) **Jednostavno pristupanje** putem REST[1] API-a [2] ili objektno-orijentiranog Java API-a

[1] REST (eng. *Representational State Transfer*) je stil dizajniranja arhitekture softvera koji koristi HTTP protokol za slanje, čitanje i brisanje podataka (Elkstein, 2008)
[2] API (eng. *Application Programming Interface*) je naziv za skup funkcija, protokola i alata za izradu aplikacija kojim se definira način komunikacije između programskih komponenti (Beal., s.a.)

Osim toga, Neo4j baza podataka može se koristiti s različitim programskim jezicima tijekom izrade aplikacija (PHP, Python, C#, Ruby on Rails, Javascript itd.), a za lakši razvoj takvih aplikacija Neo4j je integriran u nekoliko razvojnih okolina (Neoclipse, Neo4jClient, itd.).

5.1. Svojstva Neo4j baza podataka

Neo4j baza podataka omogućuje da čvorovi i veze sadrže svojstva, odnosno atribute. Atributi predstavljaju parove ključ-vrijednost, gdje je ključ skup znakova, odnosno string, a vrijednost može biti jednostavna (string, broj) ili složena (polje jednostavnih tipova). Za razliku od relacijskih baza podataka, NULL vrijednost atributa se ne smatra valjanom i u tom se slučaju radije izostavlja ključ s tom vrijednošću. Pritom svaki čvor može imati jednu ili više oznaka.

Neo4j je baza podataka neovisna o shemi, premda prisutnost sheme pridonosi poboljšanju performansi i modeliranju. Osim sheme performanse Neo4j baze podataka mogu se poboljšati kreiranjem indeksa kojima se ubrzava pretraživanje čvorova u bazi podataka. Pritom se kreirani indeksi s vremenom automatski ažuriraju u skladu s promjenama nad određenim čvorovima i njihovim svojstvima. Međutim, indeksi nisu stalno dostupni. Prilikom prvog kreiranja indeksa, on je odmah dostupan, ali nakon toga se ažurira i upotpunjava u pozadini pa tek nakon toga postaje dostupan za pretraživanje.

Da bi se tijekom brojnih ažuriranja sačuvala vjerodostojnost podataka i spriječila njihova redundancija, u verziji Neo4j 2.0 uvedena su ograničenja kojima se definiraju pravila validnosti podataka.

5.2. Pristup Neo4j bazi podataka

Neo4j bazi podataka se pristupa kao serveru na standardno definiranom portu 7474 putem REST API-a ili nekog API-a za programski jezik (najčešće Java). Trenutno je najnovija verzija Neo4j baze podataka 2.1.2. Neo4j ne jamči nikakvu razinu sigurnosti na razini podataka, ali se ona može osigurati drugim metodama (npr. osiguravanjem porta i pristupne veze, podrške za SSL[3] enkripciju podataka putem protokola HTTPS[4], pravila za serversku autorizaciju itd.).

[3] SSL (eng. *Secure Sockets Layer*) je sigurnosni protokol za slanje sadržaja putem Weba u kriptiranom obliku (Kyrnin, s.a.)

6. Primjer aplikacije Neo4jBook

Sučelje aplikacije Neo4jBook je implementirano u ASP.NET programskom jeziku (MVC uzorak dizajna) i Visual Studio razvojnoj okolini. Za izradu aplikacije potrebno je instalirati Neo4j server koji je nakon pokretanja prema uobičajenim postavkama dostupan na adresi: www.localhost/7474 . Neo4jClient je .NET biblioteka (eng.*library*) namijenjena pristupu Neo4j bazi podataka koja olakšava pisanje Cypher upita u C#, a dostupna je kao paket za Visual Studio. Nakon uspješne instalacije potrebno ju je dodati kao referencu u cjelokupni projekt.

Bazu podataka čini nekoliko čvorova označenih oznakama *Knjiga, Korisnik, Autor* i *Žanr* povezanih vezama tipa *PRIPADA, NAPISAO, POSUDIO* (vidi sliku 6.1.). Prilikom unosa u bazu podataka svaki čvor dobiva svoj jedinstveni broj (primjerice, 1052), a klikom na taj čvor mogu se detaljnije vidjeti njegovi atributi (primjerice, za čvor s oznakom *Knjiga* to su *Naziv, GodinaObjave, Status*).

Za povezivanje sučelja aplikacije na bazi podataka napravljena je posebna klasa Neo4jDb koja sadrži konstruktor čijim se povezivanjem otvara konekcija na bazu sadržana u objektu *_client*:

```
_client =new GraphClient(new Uri("http://localhost:7474/db/data"));
_client.Connect();
```

[4] HTTPS (eng. *HyperText Transport Protocole Secure*) je protokol sličan HTTP protokolu koji koristi dodatan SSL protokol za enkripciju podataka, a koristi se kod web servera za siguran prikaz i slanje sadržaja putem Weba (Kyrnin., s.a.)

Za dodavanje novog čvora tipa *Knjiga, Korisnik, Autor* ili *Žanr* poziva se odgovarajuća metoda *Dodaj()* kojoj se prosljeđuje objekt odgovarajućeg tipa. Primjerice, za dodavanje čvora s oznakom *Knjiga* u bazu podataka izvršava se sljedeći upit gdje se koristi naredba MERGE (status nove knjige je automatski postavljen na vrijednost „Slobodna“), a pohranjuju se vrijednosti sadržane u objektu klase *Knjiga*, dok se kao rezultat upita vraća kreirani čvor (naredba RETURN) u obliku elementa liste tipa *Knjiga*:

```
public Knjiga Dodaj (Knjiga item)
        {
          return
Neo4jDb.Instance.Client.Cypher.Merge("(k:Knjiga{Naziv:'"+item.Naziv+
"',GodinaObjave:'"+item.GodinaObjave+"',Status:'Slobodna',Id:'"+item.Id+"'})")
       .Return(k => k.As<Knjiga>()).Results.SingleOrDefault();
        }
```

Za dohvaćanje svih čvorova određenog tipa iz baze podataka koristi se metoda *DohvatiSve()* koja vraća čvorove u obliku liste tipa, primjerice, *Knjiga* pomoću naredbe MATCH.

```
public IEnumerable<Knjiga> DohvatiSve()
        {
            return Neo4jDb.Instance.Client.Cypher
                .Match("(k:Knjiga)")
                .Return(k => k.As<Knjiga>())
                .Results.ToList();
        }
```

Ažuriranje određenog čvora se izvršava pomoću 2 metode:

1. Najprije je potrebno dohvatiti upravo taj čvor iz baze podataka. Tome služi metoda *Pronadji()* koja koristi ID čvora kao identifikator pomoću kojeg dohvaća podatke o tom čvoru iz baze podataka korištenjem Cypher naredbe WHERE i vraća ih u obliku liste:

```
public Knjiga Pronadji(Guid id)
        {
            return Neo4jDb.Instance.Client.Cypher
                .Match("(k:Knjiga)")
                .Where((Knjiga k) => k.Id == id)
                .Return(k => k.As<Knjiga>()).Results.Single();
        }
```

2. Ažurirane podatke se pohranjuje u bazu podataka pomoću metode *Azuriraj()* kojoj se kao argument prosljeđuje ažurirani čvor kao objekt, a ID tog čvora služi za pronalaženje tog čvora unutar grafa pomoću naredbe MATCH. Naredbom SET se u bazu podataka upisuju podaci o ažuriranoj knjizi.

```
public Knjiga Azuriraj(Knjiga item)
        {
         return Neo4jDb.Instance.Client.Cypher.Match("(k:Knjiga)")
                .Where((Knjiga k) => k.Id == item.Id)
                .Set("k.Naziv='" + item.Naziv +
"',k.GodinaObjave='"+item.GodinaObjave+"',k.Status='"+item.Status+"'")
                .Return(k=>k.As<Knjiga>()).Results.SingleOrDefault();
        }
```

Osim toga, prilikom vraćanja knjige vrijednost atributa Status postavlja se na „Slobodna“ pozivanjem metode *AzurirajStatus()*.

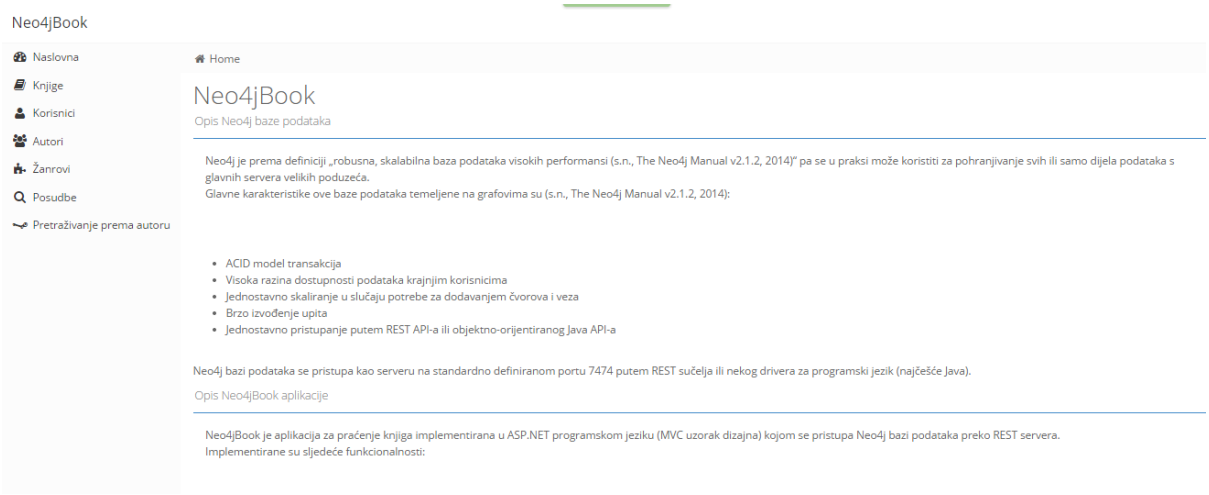
```
public Knjiga AzurirajStatus(Knjiga item)
        {
            return Neo4jDb.Instance.Client.Cypher.Match("(k:Knjiga)")
                .Where((Knjiga k) => k.Id == item.Id)
                .Set("k.Status='Slobodna'").Return(k =>
k.As<Knjiga>()).Results.SingleOrDefault();
        }
```

Tijekom brisanja odabranog čvora (primjerice čvora tipa *Knjiga*) potrebno je najprije obrisati sve njegove veze prema drugim čvorovima, kao i taj čvor, čemu služi metoda *Brisi()* kojoj je prosljeđuje objekt zadanog tipa koji sadrži ID čvora koji treba pronaći u grafu pomoću naredbe WHERE. Kad je taj čvor pronađen, naredbom OPTIONAL MATCH pronalaze se njegove veze prema drugim čvorovima.

```
public void Brisi(Knjiga item)
        {
            Neo4jDb.Instance.Client.Cypher.OptionalMatch("(k:Knjiga)-[r]-
()").Where((Knjiga k) => k.Id == item.Id).Delete("k,r").ExecuteWithoutResults();
            Neo4jDb.Instance.Client.Cypher.Match("(k:Knjiga)").Where((Knjiga k) =>
k.Id == item.Id).Delete("k").ExecuteWithoutResults();
        }
```

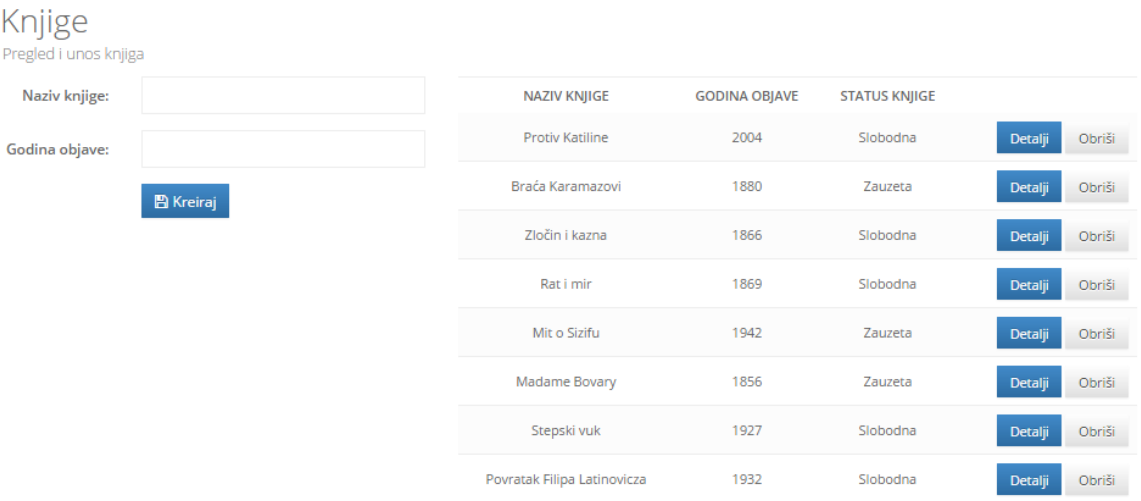
Pronađeni čvorovi i veze se brišu pomoću naredbe DELETE, a potom se briše čitav čvor pomoću iste naredbe.

Na slici 6.1. prikazana je početna stranica Neo4jBook aplikacije na kojoj su ukratko opisane karakteristike Neo4j baza podataka i funkcionalnosti Neo4jBook aplikacije.



Slika 6.1. Početna stranica Neo4jBook aplikacije

Odabirom opcije Knjige u glavnom izborniku otvara se forma za unos nove knjige, a klikom na gumb Pohrani pomoću AJAX-a se ažurira tablica koja prikazuje postojeće knjige u bazi podataka, gdje je svaku knjigu moguće ažurirati ili obrisati.



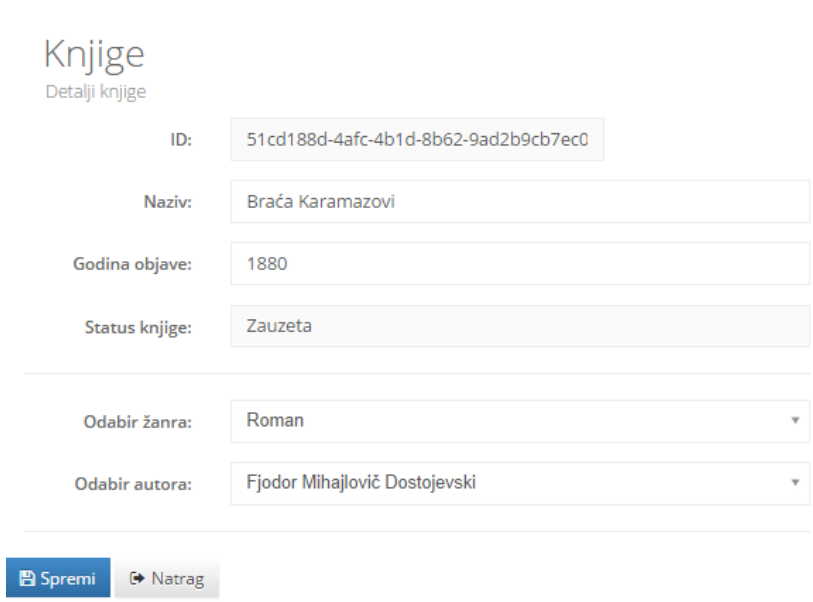
Slika 6.2. Forma za unos knjige i pregled postojećih knjiga u bazi podataka

Pritiskom na gumb Kreiraj izvršava se poziva se funkcija *Kreiraj* u *Knjiga* kontroleru:

```
public ActionResult Kreiraj(string naziv, string godinaObjave)
        {
            _knjigaRepository.Dodaj(new Knjiga() {Id = Guid.NewGuid(), Naziv =
naziv, GodinaObjave=godinaObjave });

            var knjige = _knjigaRepository.DohvatiSve();

            var model = knjige.Select(knjiga => new KnjigaViewModel
            {
                Id = knjiga.Id,
                Naziv = knjiga.Naziv,
                GodinaObjave=knjiga.GodinaObjave
            }).ToList();

            return Json(model);
        }
```

Za popunjavanje tablice s postojećim knjigama u bazi podataka poziva se metoda *DohvatiSve()*.

Ukoliko korisnik želi detaljno vidjeti neku knjigu i pritom je ažurirati (dodijeliti joj autora i žanr čime se kreiraju veze prema čvorovima drugih oznaka), odabirom opcije Detalji otvara se forma koja sadrži podatke o toj knjizi.

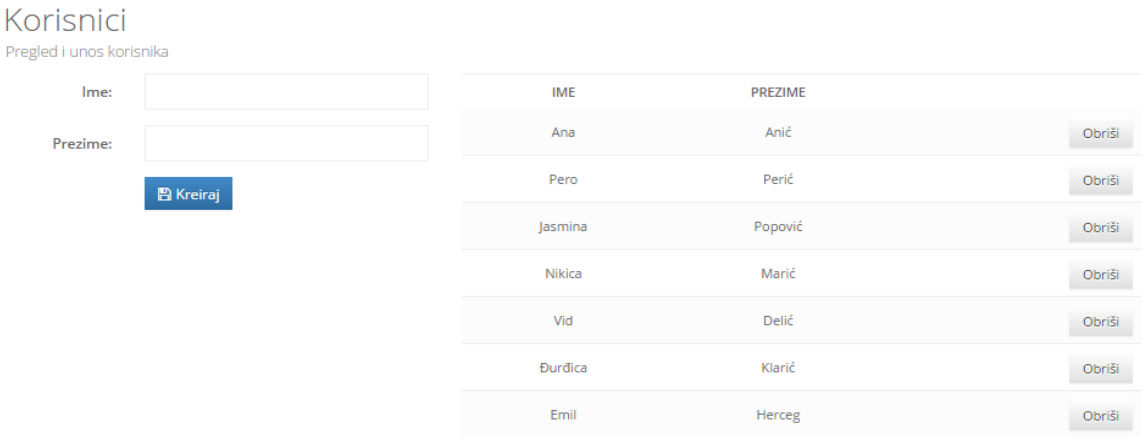


Slika 6.3. Forma za ažuriranje knjiga

Klikom na gumb Spremi izvršava se Cypher upit kojim se kreira veza između, primjerice, knjige i odabranog žanra. Pritom se na temelju proslijeđenih ID-ova objekata klasa *Knjiga* i *Žanr* pronalaze potrebni čvorovi u bazu podataka (naredba WHERE) i pomoću naredbe CREATE se kreira veza tipa PRIPADA između pronađenih čvorova.

```
public void PoveziKnjigaZanr(Knjiga knjiga, Zanr zanr)
        {
            Neo4jDb.Instance.Client.Cypher.Match("(k:Knjiga)","(z:Žanr)")
                .Where((Knjiga k)=>k.Id==knjiga.Id)
                .AndWhere((Zanr z)=>z.Id==zanr.Id)
                .Create("k-[:PRIPADA]->z")
                .ExecuteWithoutResults();
        }
```

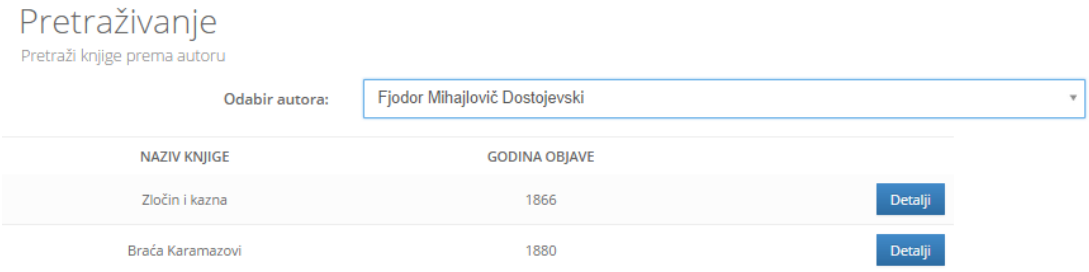
Odabirom opcija Korisnici, Autori ili Žanrovi iz glavnog izbornika, korisniku se otvaraju odgovarajuće forme i tablica s postojećim zapisima u bazi podataka.



Slika 6.4. Forma za unos i pregled postojećih korisnika

Način implementacije navedenih funkcionalnosti je identičan, jedino što ih razlikuje su oznake čvorova nad kojima se izvršava Cypher upit.

Osim osnovnih CRUD operacija, implementiran je Cypher upit pomoću kojeg je za odabranog autora moguće pregledati sve njegove knjige (slika 6.5.).



Slika 6.5. Pretraživanje autora i njihovih knjiga

Navedeno pretraživanje je implementirano na sljedeći način:

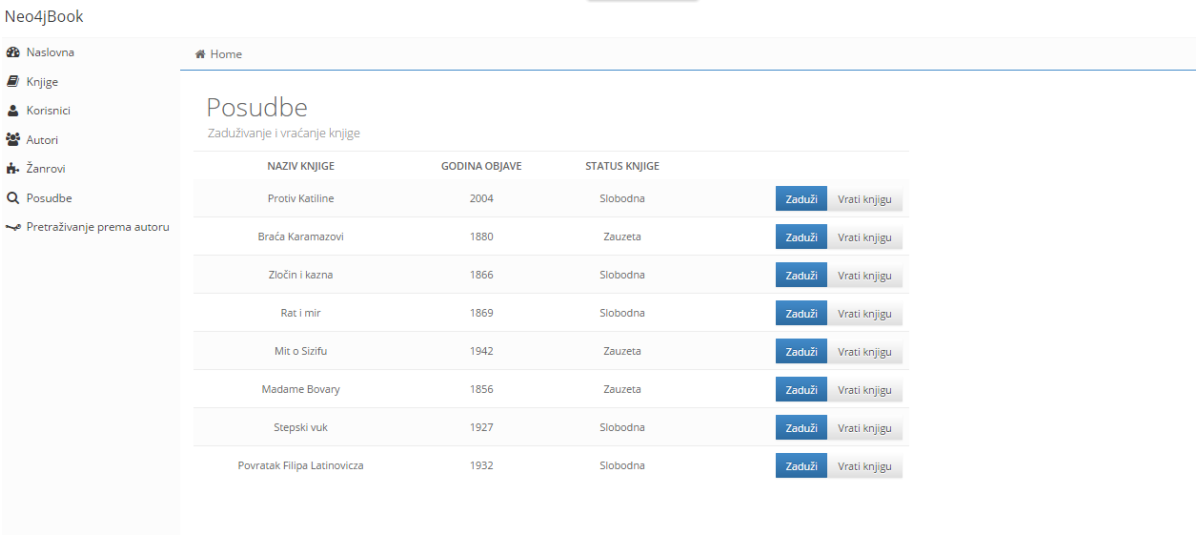
Odabirom autora se metodi *TraziKnjige()* u *Autor* kontroleru prosljeđuje ID odabranog autora.

```
public ActionResult TraziKnjige(Guid autor)
     {
       var knjige = _service.DohvatiKnjigePremaAutoru(new Autor() { Id = autor });
       var result = knjige.Select(knjiga => new KnjigaSearchViewModel()
{Id = knjiga.Id, Naziv = knjiga.Naziv , GodinaObjave =
knjiga.GodinaObjave}).ToList();
            return Json(result);
        }
```

Na temelju tog ID-a metoda *DohvatiKnjigePremaAutoru()*izvršava zadani Cypher upit nad bazom podataka kojim se najprije pomoću naredbe MATCH traže čvorovi koji zadovoljavaju traženi uzorak, potom se oni selektiraju na temelju ID-a autora pomoću naredbe WHERE, a dobiveni rezultat se vraća u obliku liste, koji se u JSON obliku šalje kao odgovor na AJAX upit čime se popunjava tablica s knjigama.

```
public List<Knjiga> DohvatiKnjigePremaAutoru(Autor autor)
        {
            return Neo4jDb.Instance.Client.Cypher.Match("(a:Autor)-[:NAPISAO]-
>(k:Knjiga)")
                .Where((Autor a) => a.Id == autor.Id)
                .Return(k => k.As<Knjiga>())
                .Results.ToList();
        }
```

Odabirom opcije Posudbe u glavnom izborniku korisnik može pregledati sve knjige i njihove statuse (slobodna ili zauzeta) te izvršiti zaduživanje/vraćanje knjige (slika 6.6.).



Slika 6.6. Pregled knjiga za zaduživanje/vraćanje

Zaduživanje knjiga se izvršava metodom *Zaduzi()* kojoj se kao argumenti šalju elementi pripadajuće forme.

```
public ActionResult Zaduzi(Guid id, string naziv, string godinaObjave, string
status, Guid korisnik)
        {
            var knjiga = new Knjiga() { Id = id, Naziv = naziv, GodinaObjave =
godinaObjave, Status = status };

            _knjigaRepository.Azuriraj(knjiga);

            _knjigaService.BrisiVezaKorisnikKnjiga(knjiga);

            if (korisnik != null)
            {
                _knjigaService.PoveziKorisnikKnjiga(new Korisnik() { Id = korisnik
}, knjiga);
            }
```

Pritom se ažuriraju vrijednosti atributa čvora s oznakom *Knjiga*, briše prethodno definirana veza između knjige i korisnika ukoliko ona postoji, a potom se ona kreira pozivanjem metode *PoveziKorisnikKnjiga()*.

```
public void PoveziKorisnikKnjiga(Korisnik korisnik, Knjiga knjiga)
        {
            Neo4jDb.Instance.Client.Cypher.Match("(u:Korisnik)", "(k:Knjiga)")
                .Where((Knjiga k) => k.Id == knjiga.Id)
                .AndWhere((Korisnik u) => u.Id == korisnik.Id)
                .Create("u-[:POSUDIO]->k")
                .ExecuteWithoutResults();
        }
            return Json(new { result = "OK" });
        }
```

Budući da se sličan postupak odvija prilikom vraćanja knjiga (promjena statusa knjige na vrijednost „Slobodna“ i brisanje veze između knjige i korisnika), nema potrebe za detaljnim prikazivanjem.

7. Zaključak

Na temelju analiziranih karakteristika graf baza podataka i njihove usporedbe s karakteristikama tradicionalnih relacijskih baza podataka, može se zaključiti kako ova kategorija NoSQL baza podataka može zadovoljiti potrebe suvremenih korisničkih zahtjeva. Već se u današnje vrijeme koristi u velikim korporacijama i na društvenim mrežama pa broj korisnika i stručnjaka na tom području svakodnevno raste, što će povećati fond znanja o graf bazama podataka, a time i olakšati početnicima tijekom njihova učenja ili programerima tijekom rješavanja određenih problema.

Budući da model podataka predstavlja usmjereni graf gdje svaki čvor sadrži pokazivač na njegov susjedni čvor, u graf bazama podataka nije potrebno kreirati indekse ni povezivati podatke na temelju zajedničkog atributa prilikom izvođenja upita. Potrebno je samo pronaći početni čvor, a zatim se od njega može nastaviti pretraživanje traženih podataka zahvaljujući usmjerenim vezama. Pritom će postupak biti brži, budući da se upit izvodi nad određenim dijelom grafa, a ne nad cijelom bazom podataka.

Premda je broj korisnika relacijskih baza podataka još uvijek puno veći, prednosti graf baza podataka kao što su jednostavno modeliranje baze podataka, brže izvođenje upita, jednostavnost i fleksibilnost baze podataka će zasigurno s vremenom privući njihovu pozornost, budući da navedene prednosti olakšavaju razvoj složenih aplikacija i održavanje baza podataka i smanjuju troškove ukoliko su performanse manje važan čimbenik.

8. Literatura

[1] Rabuzin K. (2011) *Uvod u SQL*. Varaždin: Fakultet organizacije i informatike

[2] s.n. w3resource (2014) *NoSQL*. Preuzeto 13. veljače 2014. s http://www.w3resource.com/mongodb/nosql.php

[3] Sprava M. (2013) *Database Master-Slave Replication in the Cloud*. Preuzeto 3.rujna 2014. s http://blog.jelastic.com/2013/01/15/database-master-slave-replication-in-the-cloud/

[4] Beal V (s.a.) *API – Application Programming Interface*. Preuzeto 3. rujna 2014. s http://www.webopedia.com/TERM/A/API.html

[5] Elkstein M (2008) *What is REST?*. Preuzeto 3.rujna 2014. s http://rest.elkstein.org/2008/02/what-is-rest.html

[6] Kyrnin J. (s.a.) *SSL*. Preuzeto 3.rujna 2014. s http://webdesign.about.com/od/ssl/g/bldefssl.htm

[7] Kyrnin J. (s.a.) *HTTPS*. Preuzeto 3.rujna 2014. s http://webdesign.about.com/od/http/g/bldefhttps.htm

[8] s.n., mongoDB Resources (2014) *NoSQL Databases Explained*. Preuzeto 13. veljače 2014. s http://www.mongodb.com/learn/nosql

[9] Rafique A (2013) *Evaluating NOSQL Technologies for Historical Financial Data* . Uppsala:Teknisk- naturvetenskaplig fakultet, pp.18-24

[10] Golemac A, Mimica A, Vučičić T (2012) *Teorija grafova: Od koenigsberških mostova do kineskog poštara* . Preuzeto 18. veljače 2014. s http://e.math.hr/category/klju-ne-rije-i/teorija-grafova. Split: Prirodoslovno-matematički fakultet Sveučilišta u Splitu

[11] Fošner M, Kramberger T (2009) *Teorija grafova i logistika* . Preuzeto 18. veljače 2014. s http://e.math.hr/math_e_article/br14/fosner_kramberger#content_marker2 . Maribor: Faculty of Logistics

[12] s.n. (s.a.) *Methods of Search*. Preuzeto 21. srpnja 2014. s http://www.cse.unsw.edu.au/~billw/Justsearch.html . Australia:The School of Computer Science and Engineering, Sydney

[13] s.n., TinkerPop (2012) *Property Graph Model*. Preuzeto 19. veljače 2014. s https://github.com/tinkerpop/blueprints/wiki/property-graph-model

[14] Robinson I, Webber J, Eifrem E (2013) *Graph Databases* (1.izd), Sebastopol: O'Reilly Media

[15] Adell J (2011) *Performance of Graph vs. Relational Databases*. Preuzeto 21.7.2014. s http://blog.everymansoftware.com/2011/07/performance-of-graph-vs-relational.html

[16] Zicari R (2013) *Graphs vs. SQL. Interview with Michael Blaha* . Preuzeto 17.7.2014. s http://www.odbms.org/blog/2013/04/graphs-vs-sql-interview-with-michael-blaha/

[17] Eifrem E (2012) *Graph Databases: The New Way to Access Super Fast Social Data* . Preuzeto 17.7.2014. s http://mashable.com/2012/09/26/graph-databases/

[18] s.n., The Neo4j Manual v2.1.2 (2014) *What is Cypher?*. Preuzeto 22.7.2014. s http://docs.neo4j.org/chunked/milestone/cypher-introduction.html

[19] s.n., The Neo4j Manual v2.1.2 (2014) *Neo4j Highlights*. Preuzeto 22.7.2014. s http://docs.neo4j.org/chunked/stable/introduction-highlights.html

9. Popis korištenih slika i tablica