SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Martina Šestak

USPOREDBA JEZIKA ZA GRAF BAZE PODATAKA

DIPLOMSKI RAD

Varaždin, 2016.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Martina Šestak

Matični broj: 43591/14-R

Studij: Informacijsko i programsko inženjerstvo

USPOREDBA JEZIKA ZA GRAF BAZE PODATAKA

DIPLOMSKI RAD

Mentor:

Izv. prof. dr. sc. Kornelije Rabuzin

Varaždin, 2016.

Sadržaj

1. Uvod

Baze podataka oduvijek su se smatrale važnim dijelom brojnih aplikacija različite namjene, prvenstveno zbog mogućnosti pohrane i ponovnog dohvaćanja važnih podataka kada je to potrebno. Danas se još uvijek najčešće koriste relacijske baze podataka. Svoju popularnost zadržale su sve do danas, što najbolje pokazuje njihovu korisnost i prednosti korištenja baza podataka općenito. U relacijskim bazama podataka podaci su spremljeni u tablice, gdje svaka obično ima i primarni ključ kojim se jednoznačno identificira svaki zapis u toj tablici. Korištenjem vanjskih ključeva moguće je referencirati zapis iz jedne tablice u drugoj tablici, što je posebice važno kod izvršavanja upita. Međutim, vanjski ključevi utječu na performanse njihova izvršavanja, što je u okviru suvremenih zahtjeva u kojem podaci moraju biti dostupni u stvarnom vremenu uz što niže troškove bitan i neizbježan nedostatak. O ovom, kao i ostalim problemima u izvršavanju upita nad relacijskim bazama podataka, bit će više riječi u sljedećem poglavlju ovog rada.

Usporedno, zadnjih su godina svoj zamah u korištenju doživjele i tzv. NoSQL baze podataka koje svojim osobinama i načinom upravljanja podacima nadoknađuju pojedine nedostatke relacijskih baza podataka. Jedna od kategorija NoSQL baza podataka su i graf baze podataka. U ovom će radu naglasak biti na proučavanju svojstava i načinu izvršavanja upita nad spomenutom kategorijom baza podataka te na prednostima i nedostacima prilikom njihova izvršavanja. Detaljnije će se analizirati načini izvršavanja upita, ali i upitni jezici Gremlin i, u novije vrijeme sve popularniji, Cypher Query Language (u nastavku Cypher). Spomenuti upitni jezici će se usporediti s SQL upitnim jezikom za relacijske baze podataka s obzirom na nekoliko karakteristika.

Za prikaz načina izvršavanja upita pomoću Cypher i Gremlin upitnih jezika kreirana je aplikacija za prikaz knjiga, njihovih autora i žanra te korisnika (točnije dvije). U tu svrhu korišteni su Neo4j (upiti se izvršavaju korištenjem Cyphera) te Rexster (upiti se izvršavaju putem Gremlina) kao predstavnici sustava za upravljanje graf bazama podataka. U obje verzije aplikacije je jednaka funkcionalnost implementirana korištenjem različitih tehnologija. Aplikacije su izvorno razvijane za potrebe projekta „Novi jezik za graf baze podataka“ koji je vodio izv. prof. dr. sc. Kornelije Rabuzin.

2. Relacijske baze podataka

Relacijska baza podataka se može definirati kao *organizirana kolekcija podataka u kojoj su podaci organizirani u skup relacija* (Darwen, 2012). Za uvođenje koncepta relacijskih baza podataka najvećim je dijelom zaslužan Edward Codd koji ga je prvi spomenuo u svom radu 1970. godine. U navedenom radu E. Codd pojašnjava značajke i mogućnosti relacijskog modela koji čini osnovu ove najčešće korištene vrste baza podataka. Kao jednu od važnijih značajki tog modela dr. Codd izdvaja činjenicu da su sve informacije u relacijskim bazama podataka dostupne i reprezentirane kao vrijednosti u tablicama. Ova značajka omogućuje programerima, ali i krajnjim korisnicima, da pristupe podacima preko vrijednosti podataka, a ne preko njihove pozicije u datotečnom sustavu (kao što je bila ranija praksa u domeni baza podataka) (Codd, 1970).

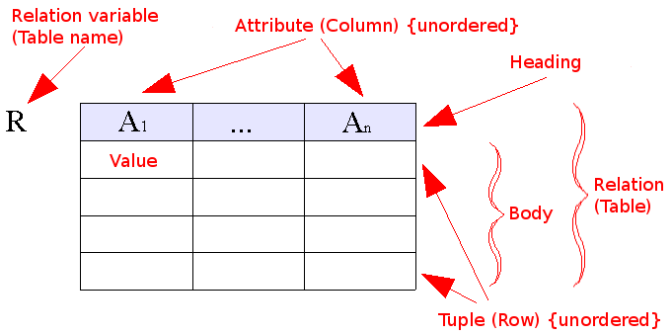
Osnovna je karakteristika relacijskih baza podataka reprezentacija podataka u obliku dvodimenzionalnih tablica. Zbog načina njihove vizualizacije u praksi mnogi koriste pojam *tablica* kao zamjenu za pojam *relacija*, *stupac tablice* kao zamjenu za *atribut* ili, pak, *redak tablice* kao zamjenu za *n-torka*. Dakle, relacijske baze podataka svoju dugogodišnju popularnost duguju svojoj jednostavnosti i razumljivosti za osobe s različitim razinama znanja i vještina.

2.1. Relacijski model podataka

Relacijski model je posebna vrsta podatkovnog modela pa se on sastoji od tri međusobno ovisna dijela (komponente):

1) **Struktura;** podaci se pohranjuju u relacije reprezentirane u obliku tablica (slika 1)
2) **Operativna komponenta;** algebarski operatori (selekcija, unija, presjek...) pomoću kojih se relacije transformiraju u druge relacije
3) **Uvjeti integriteta;** pravila integriteta poput entitetskog integriteta (primarni ključ za svaki red u tablici je jedinstven i različit od drugih primarnih ključeva u tablici te ne smije sadržavati NULL vrijednost), referencijalnog integriteta (vanjski ključ referencirajuće tablice mora biti jednak primarnom ključu referencirane tablice), NOT NULL (vrijednost atributa u relaciji ne smije biti NULL) i UNIQUE (zapisi u relaciji se ne smiju ponavljati) ograničenja (Watt, 2013)

Opisani relacijski model predstavlja matematičku osnovu za relacijske baze podataka.



Slika 1. Relacijski model podataka (Izvor: https://en.wikipedia.org/wiki/Relational_model, 2006)

E. Codd je u svom radu naveo kako relacijski model podataka pruža nekoliko važnih karakteristika koje čine upravljanje i korištenje baza podataka jednostavnijim, otpornim na pogreške te predvidljivim (Melton & Simon, 1997).

Relacijski model podataka opisuje podatke na prirodan način, pomoću njihove prirodne strukture, bez dodavanja struktura radi lakše implementacije. Osim toga, važno je spomenuti da ovaj model podataka podržava neovisnost podataka od njihove fizičke reprezentacije, od veza između podataka te od načina implementacije.

2.2. Relacijsko procesiranje

Relacijski model podataka ne uključuje samo relacijske strukture u obliku dvodimenzionalnih tablica, nego i poseban način procesiranja podataka koji se naziva **relacijsko procesiranje** (Codd, 1970). Spomenuti način procesiranja je specifičan po tome što sa svakom relacijom u njenoj cjelini postupa kao s operandom. U procesiranju podataka sudjeluju i operatori relacijske algebre poput, primjerice, operatora selekcije koji uzima jednu relaciju i na temelju nje kreira novu relaciju koja sadrži izabrane n-torke iz prve relacije.

Temeljni operatori relacijske algebre su:

1) **Operator selekcije** – unarni operator koji odabire n-torke iz relacije koje odgovaraju zadanom predikatu (uvjetu) selekcije i vraća relaciju s tim n-torkama
2) **Operatori unije, presjeka i razlike** – binarni operatori koji djeluju nad relacijama kao skupovima pa nad njima primjenjuju skupovne operatore unije, presjeka i razlike
3) **Operator projekcije** – operator koji vraća relaciju koja sadrži podskup atributa koji zadovoljavaju zadani predikat (uvjet)

4) **Operator kompozicije** – operator koji omogućuje kombiniranje n-torki iz različitih relacija i njihovo povezivanje u novu relaciju
5) **Operator preimenovanja** – unarni operator koji mijenja nazive atributa u relaciji bez promjene njihove vrijednosti

Operatori relacijske algebre poput selekcije, kompozicije i projekcije se smatraju snažnim „alatom“ za procesiranje podataka, a time i relacijska algebra čini osnovu za sve izvedene i specijalizirane jezike baza podataka (u literaturi se često koristi naziv eng. *data sublanguage*), što i jest cilj procesiranja relacijskog modela.

Sustav za upravljanje bazama podataka (eng. *Database Management System*, u daljnjem tekstu DBMS) prima naredbe od korisnika koje moraju biti napisane u jeziku koji taj sustav podržava. Relacijski DBMS omogućuje kreiranje i povezivanje tablica te definira odgovarajući jezik za pristup i upravljanje podacima (npr. SQL ili QUEL) (Codd, 1970). DBMS izvršava primljene naredbe, odnosno upite nad bazom podataka. **Upit** se može definirati kao *izraz koji svojim izvršavanjem kreira neki rezultat izveden iz baze podataka* (Darwen, 2012). Uloga je DBMS-a da osigura potporu za što više naredbi i da je rezultat izvršavanja upita dostupan korisniku.

2.3. Upitni jezici za relacijske baze podataka

Upitni jezik se formalno definira kao *skup izraza L i funkcije značenja μ takve da je za svaki izraz e u skupu L μ(e) upit* (Vardi, 1982). Isti je autor u svom članku podijelio upitne jezike za relacijske baze podataka u dvije kategorije:

1) **Logički** upitni jezici su u neproceduralni, a sastoje od formula koje prilikom primjene nad nekom bazom podataka kao odgovor vraćaju sve n-torke koje ih zadovoljavaju (npr. relacijski račun)
2) **Algebarski** upitni jezici su proceduralni i primjenom algebarskih operatora nad ulaznim relacijama vraćaju relacije kao izlaz (npr. relacijska algebra)

U praksi se upitni jezici za relacijske baze podataka razlikuju od osnovnog relacijskog modela; neki sadrže dodatne mogućnosti koje su nadogradnja modela ili pak iste nisu uopće definirane u modelu. Osim toga, iako je u teoriji redoslijed atributa i n-torki unutar relacije proizvoljan i nebitan, upitni jezici pružaju način kontrole redoslijeda za atribute i redoslijed sortiranja za n-torke (Maier, 1983).

Upitni jezici za relacijske baze podataka po svojim karakteristikama nisu nimalo slabiji od (teorijske) relacijske algebre. Međutim, u nekim slučajevima se željeni rezultat može ostvariti putem jednog izraza, dok je u drugim slučajevima za to potrebno izvršiti niz naredbi, odnosno izraza. Upiti izraženi putem relacijske algebre uvijek rezultiraju konačnim odgovorima jer su operandi takvih upita relacije koje su po svojoj definiciji konačni objekti. Vrijednosti koje se nalaze u rezultatu upita nimalo ne ovise o ostalim vrijednostima u domeni iz koje su napravljene relacije, već samo o vrijednostima ulaznih relacija i konstantnim vrijednostima zadanim u upitu (Levene i Loizou, 2012).

Neki od najpoznatijih upitnih jezika za relacijske baze podataka su ISBL, QUEL, SQL, QBE, PIQUE te će o njima biti više riječi u nadolazećim poglavljima.

2.3.1. ISBL upitni jezik

ISBL (eng. *Information System Base Language*) je upitni jezik razvijen od strane IBM-a koji se temelji na relacijskoj algebri. U tom upitnom jeziku postoji ukupno šest operatora koji su semantički ekvivalentni algebarskim operatorima, a pomoću kojih se kreiraju ISBL izrazi (Maier, 2015). To su operatori unije, presjeka, kompozicije, razlike, selekcije te operatori projekcije i preimenovanja koji su spojeni u jedan operator. Međutim, ISBL nema vlastitih operatora za izvršavanje računskih operacija nad vrijednostima pa se u tu svrhu koriste već izračunate relacije koje predstavljaju funkcije određenog programskog jezika opće namjene (Maier, 2015). Za ispis rezultata ISBL izraza koristi se tzv. lista ključnih riječi (eng. *keyword list*).

Osim toga, ISBL pruža podršku za virtualne relacije putem mehanizma odgođenih evaluacija izraza, što znači da se svaki ISBL izraz evaluira tek kad je to potrebno.

2.3.2. QUEL jezik

QUEL (eng. *QUEry Language*) je jezik za manipulaciju podacima koji je prvenstveno bio korišten za INGRES (eng. *INteractive Graphics and REtrieval* System) relacijski sustav za upravljanje bazama podataka.

Osim funkcija i naredbi za dohvaćanje podataka iz baze podataka, QUEL sadrži i naredbe za njihovo ažuriranje, autorizaciju i očuvanje integriteta baze podataka te za definiranje pogleda (eng. *view*). QUEL jezik se temelji na relacijskom računu orijentiranom na n-torke (eng. *tuples relational calculus*) koji je E. Codd predložio kao alternativu relacijskoj algebri, a u kojem se kod definiranja upita zadaje predikat koji n-torke u relacijama

trebaju zadovoljavati. Osim toga, potrebno je kvantificirati varijable n-torke (eng. *tuple variables*) i povezati ih s odgovarajućom relacijom.

QUEL pruža podršku za korištenje operatora za usporedbu *(=, <, >, !=* itd.) i logičkih operatora (*and, or, not*), ali i operatora za agregaciju podataka (*sum, avg, min, max* itd.).

2.3.3. QBE upitni jezik

QBE (eng. *Query-By-Example*) je relacijski jezik za manipulaciju podacima koji je razvio M. M. Zloof (Maier, 2015). Od ostalih upitnih jezika za relacijske baze podataka razlikuje se po načinu pisanja upita. Naime, korisnici upite pišu u grafičkom sučelju, a prilikom njihova pisanja na ekranu se pomoću tzv. varijabli domene kreiraju tablice koje predstavljaju preslike tablica u bazi podataka. Sadrži relativno malen broj koncepata pa je korisnicima lagan za učenje i razumijevanje (Ramakrishnan i Gehrke, 2002).

Osim funkcija za dohvaćanje podataka, QBE sadrži i funkcije za ažuriranje, mehanizme autorizacije i očuvanja integriteta, deklaracije domena te funkcije za definiranje pogleda.

Upiti definirani u QBE jeziku su dvodimenzionalni, a kreiraju se popunjavanjem tzv. kostura (eng. *skeleton*) koji čine naziv relacije i njeni atributi. Tako definirani kostur popunjava se retcima konstanti i varijabli. Popunjeni kostur semantički odgovara pripadajućim tablicama u bazama podataka. Kao i QUEL, i QBE jezik pruža podršku za korištenje operatora za usporedbu *(=, <, >, !=* itd.) i logičkih operatora (*and, or, not*), ali i operatora za agregaciju podataka (*sum, avg, min, max* itd.).

2.3.4. PIQUE upitni jezik

PIQUE (eng. *PIts QUEry language*) je eksperimentalni jezik za dohvaćanje podataka za PITS (eng. *Pie-In-The-Sky*) sustav za upravljanje bazama podataka (Maier, 2015). Budući da se temelji na relacijskom računu orijentiranom na n-torke, u svojoj sintaksi sličan je već spomenutom QUEL jeziku. Ipak, razlikuju se u tome što se u PIQUE jeziku varijable n-torke, umjesto s relacijama u QUEL jeziku, implicitno povezuju s prozorima (eng. *windows*), odnosno relacijama dobivenim vrednovanjem prozora nad trenutnim stanjem baze podataka. Posljedica toga je da korisnici ne moraju eksplicitno specificirati povezivanja (eng. *bindings*) tijekom pisanja upita, već to umjesto njih radi DBMS (Maier i sur., 1987).

Poput prethodno opisanih jezika, i PIQUE jezik pruža podršku za upotrebu operatora usporedbe i logičkih operatora tijekom pisanja upita.

2.3.5. Structured Query Language

Structured Query Language (u daljnjem tekstu SQL) je specijalizirani jezik za pristup relacijskim bazama podataka kojima upravlja relacijski DBMS (Melton i Simon, 1997). Budući da pripada skupini specijaliziranih jezika, on se koristi zajedno s aplikacijskim (programskim) jezicima (PHP, C, C#...) za pristup potrebnim podacima jer sam po sebi nije dovoljan za izgradnju čitavih aplikacija. Iako postoji još nekoliko specijaliziranih jezika za pristup relacijskim bazama podataka koji su identično ekspresivni i prate relacijski model, SQL je jedini standardizirani jezik za pružanje sučelja za pristup relacijskim bazama podataka (prema nekoliko međunarodnih standarda) pa se najčešće koristi u svakodnevnoj praksi. Kao i drugi upitni jezici za relacijske baze podataka, i SQL se temelji na relacijskoj algebri i predikatnom računu (Baranović i Zakošek, 2007).

SQL pripada skupini **neproceduralnih** jezika čija je glavna karakteristika stavljanje fokusa na rezultate operacija, što znači da se prilikom pisanja upita definira samo **što** se želi dobiti kao rezultat izvršavanja upita, a ne i **kako** to dobiti. Ovakvim su jezicima važniji rezultati izvršenih operacija od načina na koji su oni dobiveni. Način dobivanja rezultata za njih ostaje tzv. crna kutija i rezultati operacija se promatraju na globalnoj razini.

Prednosti SQL-a mogu koristiti različite skupine korisnika, od krajnih korisnika do programera. Neke od mogućnosti SQL-a su (Lorentz i Gregoire, 2003):

- Obrada skupova podataka kao grupa (umjesto kao individualnih jedinica)
- Automatsko usmjeravanje prema podacima koje omogućuje korisnicima rad s podacima na logičkoj razini, bez da moraju voditi računa o detaljima implementacije (deklarativni jezik)
- Korištenje kompleksnih i neovisnih naredbi

Korištenjem SQL-a podaci se obrađuju na logičkoj razini pa korisnik ne mora voditi računa o detaljima implementacije (osim ako mora upravljati podacima). Prilikom pristupa podacima koristi se tzv. optimizator (eng. *optimiser*) pomoću kojeg se određuje najefikasniji način pristupa podacima.

Različite akcije nad bazom podataka izvršavaju se putem SQL naredbi (eng. *statements*). One su općenito podijeljene u dvije kategorije:

1) **DML naredbe** mijenjaju objekte baza podataka i u ovu skupinu pripadaju naredbe poput INSERT, UPDATE i DELETE
2) **DDL naredbe** mijenjaju strukturu baza podataka i u ovu skupinu pripadaju naredbe poput CREATE TABLE ili DROP TABLE

Osim ovih osnovnih dviju kategorija, SQL naredbe je moguće svrstati i u još nekoliko dodatnih kategorija (Lorentz i Gregoire, 2003):

- ***Transaction Control*** naredbe upravljaju transakcijama (npr. COMMIT, ROLLBACK)
- ***Session Control*** naredbe dinamički upravljaju svojstvima korisničkih sesija (npr. ALTER SESSION, SET ROLE)
- ***System Control*** naredba ALTER SYSTEM dinamički upravlja svojstvima instance, primjerice, Oracle baze podataka
- ***Embedded*** SQL naredbe ugrađuju DDL, DML i naredbe za kontrolu transakcija u program koji koristi proceduralni jezik

Iako je u standardu definiran velik broj različitih naredbi, za upravljanje podacima je dovoljno koristiti četiri osnovne SQL naredbe (Baranović i Zakošek, 2007):

1) SELECT za zadavanje upita nad bazom podataka
2) INSERT za unos n-torke (ili više njih) u relaciju
3) UPDATE za ažuriranje n-torke u relaciji
4) DELETE za brisanje n-torke iz relacije

Budući da je SQL standardizirani upitni jezik za relacijske baze podataka, to značajno olakšava njegov primjenu u izgradnji različitih aplikacija različitih namjena. Koriste ga svi proizvođači (eng. *vendors*) koji razvijaju sustave za upravljanje bazama podataka. Dodatna prednost SQL-a je njegova jednostavnost i lakoća razumijevanja za korisnike s različitim razinama znanja o bazama podataka. Ostale prednosti SQL-a koje je važno spomenuti su (More Process - s.n., 2013):

- Prenosivost – SQL upiti se mogu izvršavati u različitim okolinama (lokalni sustavi, internet i intranet) na različitim uređajima (osobna računala, mobilni uređaji). Omogućuje jednostavne migracije baza podataka s jednog uređaja na drugi.
- Interaktivnost – SQL jezik je moguće koristiti za interakciju s različitim bazama podataka
- Može se koristiti i kao programski i kao interaktivni (upitni) jezik i ne zahtijeva puno programiranja od korisnika
- Potpun jezik za baze podataka u kojem je moguće kreirati bazu podataka, upravljati njenom sigurnošću te dohvaćati podatke potrebne korisniku
- Dinamičnost – korištenjem SQL-a moguće je dinamički mijenjati strukturu baza podataka

- Pruža podršku za objektno-orijentirano programiranje i tzv. *enterprise* aplikacije pa ga mogu koristiti i velike korporacije za pohranu velike količine podataka

2.4. Nedostaci i izazovi relacijskih baza podataka

Tijekom posljednjih nekoliko godina broj kritičara primjenjivosti i praktičnosti relacijskog pristupa upravljanju bazama podataka raste. Najviše rasprava nastaje oko pitanja može li relacijski DBMS pružati onoliko usluga i mogućnosti kao i neki drugi DBMS sustavi te, ako može, može li to pružati jednako učinkovito kao sve popularniji nerelacijski (tzv. NoSQL) DBMS sustavi.

Uz standardne mogućnosti koje pružaju DBMS sustavi poput pohrane, dohvaćanja i ažuriranja podataka, podrške za transakcije, mehanizma oporavka u slučaju pada sustava te mehanizma autorizacije kod pristupa bazama podataka, relacijski DBMS sustavi pružaju potpunije podatkovne mogućnosti od nerelacijskih sustava (npr. ekstrakcija smislenih relacija iz baze podataka, pogledi kao virtualne relacije, moguće dinamičke promjene u fizičkoj i logičkoj organizaciji podataka tijekom izvođenja transakcija). Zahvaljujući zadovoljavajućoj kombinaciji navedenih mogućnosti relacijske su baze osigurale svoju dominaciju na tržištu kroz dugi niz godina.

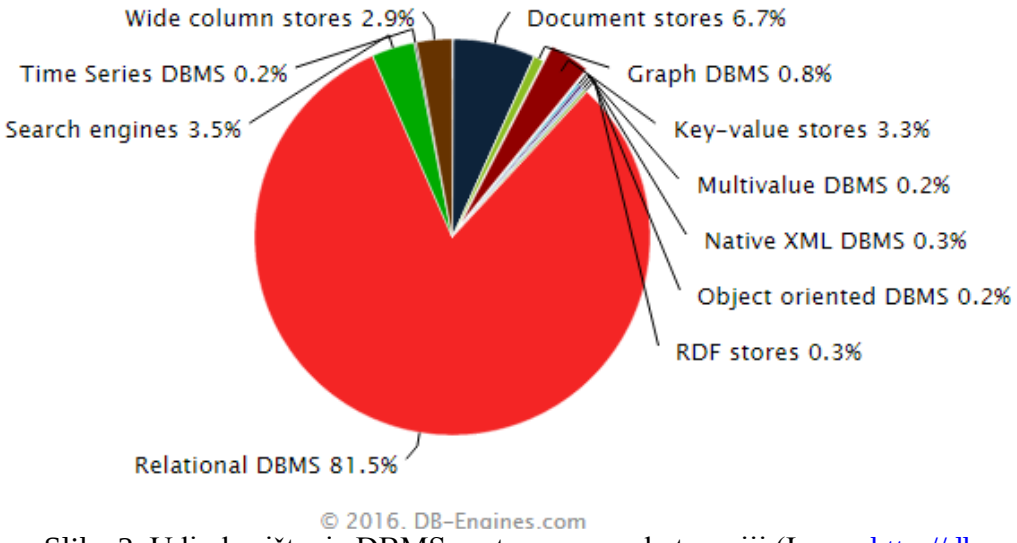
S druge strane, iako pružaju široki spektar mogućnosti, performanse relacijskih baza podataka u svakom aspektu su često lošije od drugih rješenja fokusiranih na specifičnu funkcionalnost. Promjena zahtjeva u okolini dovela je do povećanja važnosti skalabilnosti baza podataka općenito. U tom području relacijske baze podataka ostvaruju zadovoljavajuće rezultate kad je riječ o skaliranju na jednom serverskom čvoru. Skaliranje između više čvorova dovodi do različitih komplikacija zbog kompleksnosti relacijskih baza podataka (Bain, 2009). Povećanjem broja čvorova za skaliranje povećava se i kompleksnost te operacije, što smanjuje održivost relacijskih DBMS sustava kao platformi za velike distribuirane sustave kojih danas na tržištu ima sve više. Osim skalabilnosti, jedan od najčešćih problema s kojima se susreću korisnici relacijskih baza podataka su poteškoće u pretraživanju nestrukturiranih podataka ili obradi nepoznatih formata podataka (Reeve, 2012). Alternativno rješenje za navedene nedostatke relacijskih baza podataka u novije vrijeme predstavljaju graf baze podataka kao specifična skupina NoSQL baza podataka, čiji se broj korisnika zbog njihovih svojstava sve više povećava.

3. Graf baze podataka

S povećanjem složenosti i povezanosti podataka modeliranje podataka u obliku međusobno povezanih relacija više nije zahvalan zadatak. Zbog toga se danas kao prirodniji način modeliranja podataka sve više nameće modeliranje podataka u obliku grafova koji se sastoje od međusobno povezanih čvorova. U tim grafovima čvorovi (eng. *nodes*) predstavljaju objekte u stvarnom svijetu i okolini, dok veze (eng. *edges*) predstavljaju veze između tih objekata. Tehnologija graf baza podataka omogućuje potpuno iskorištavanje potencijala suvremenih podataka u jednostavnijim, ali i složenijim situacijama.

Posljednjih godina skup mogućnosti, korisnost i performanse vodećih graf baza podataka su došle na zavidnu razinu zrelosti pa graf baze podataka već i danas zauzimaju značajan udio na tržištu, a koji konstantno raste. Osim toga, graf baze podataka na važnosti dobivaju i zbog razvoja koncepta Interneta stvari[1] (eng. *Internet of Things*, u daljnjem tekstu IoT) jer on u svojoj u definiciji uključuje povezivanje uređaja, a time i podataka, što čini osnovu graf baza podataka.

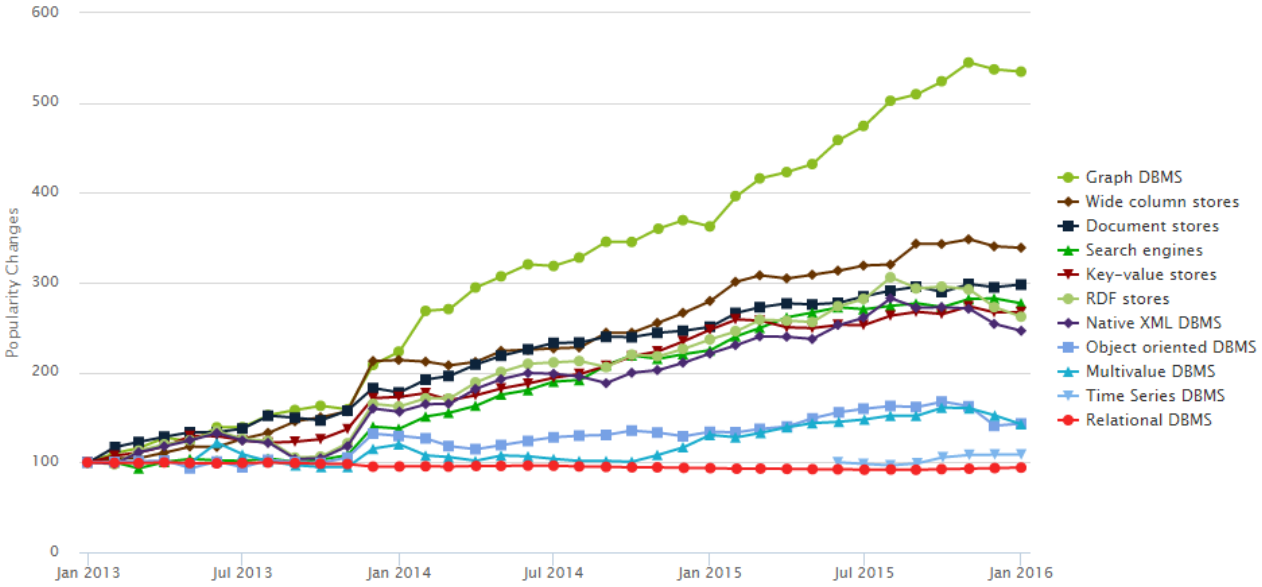
Sustav za upravljanje graf bazama podataka je sustav s *Create, Read, Update i Delete* operacijama koje razotkrivaju i djeluju nad graf modelom podataka (Robinson i sur., 2015). Prema posljednjim podacima u siječnju 2016.godine sustavi za upravljanje graf bazama podataka zauzimaju 0.8% (slika 2). Gledajući još uvijek velik udio relacijskih DBMS sustava (81.5%), taj se postotak možda čini neznatnim.



Slika 2. Udio korištenja DBMS sustava prema kategoriji (Izvor: http://db-engines.com/en/ranking_categories, 2016)

[1] Internet stvari je koncept spajanja bilo koje uređaja na Internet ili s nekim drugim uređajem (Morgan, 2014)

Međutim, istraživanja pokazuju kako graf DBMS sustavi ostvaruju najveći porast u korištenju, odnosno popularnosti među korisnicima (slika 3).



Slika 3. Trendovi popularnosti DBMS sustava prema kategoriji (Izvor: http://db-engines.com/en/ranking_categories, 2016)

Jedan od razloga velikog porasta popularnosti graf baza podataka na tržištu zasigurno je činjenica da vrlo uspješno utječu na složene i dinamičke veze između čvrsto povezanih podataka (Robinson i sur., 2015).

Dakle, graf baze podataka su sve popularnija kategorija NoSQL baza podataka koju je najbolje koristiti za reprezentaciju i pretraživanje povezanih podataka značajnih veličina.

3.1. Definicija grafa

Matematički gledano, graf predstavlja „uređenu trojku 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). Svaki vrh grafa ima jedinstveni identifikator (ekvivalent primarnom ključu kod relacijskih baza podataka) te oznaku (eng. *label*) koja može biti svojstvena većem broju vrhova u grafu.

Graf možemo formalno promatrati i kao kolekciju vrhova i bridova, odnosno čvorova i veza (Robinson i sur., 2015).

U praksi ovako definirana jednostavna struktura grafa omogućuje modeliranje podataka iz bilo koje problemske domene. Grafovi se mogu iskoristiti za lakše razumijevanje različitih životnih područja poput znanosti, poslovnih aktivnosti, vlade, društvenih mreža, medicine, itd.

3.2. Algebra grafova

Algebra grafova na kojoj se temelji graf model podataka predstavlja proširenje već spomenute relacijske algebre na kojoj se temelji istoimeni model podataka. Međutim, postoje značajne razlike između navedenih algebri (He, 2007).

Nekim operatorima relacijske algebre spomenutih u poglavlju 2.2. pridružen je određeni operator u algebri grafova (s razlikama ili bez), ali su u algebri grafova definirani i novi operatori.

Dr. He u svojoj knjizi (He, 2007) spominje sljedeće operatore algebre grafova:

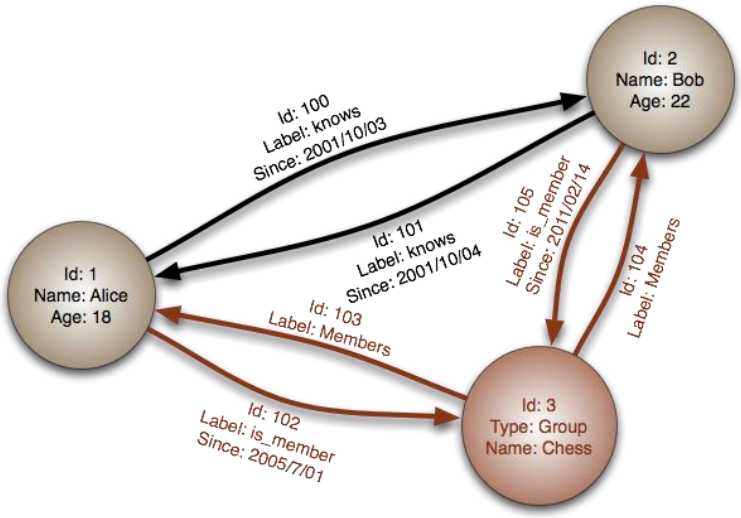
1) **Operator selekcije –** operator koji na temelju ulazne kolekcije grafova vraća kolekciju grafova koji odgovaraju zadanom uzorku grafa, čime se primjena tog operatora generalizira na pojam pronalaženja uzoraka u grafu (eng. *graph pattern matching*)
2) **Operatori unije, presjeka i razlike** – operatori koji nad grafovima primjenjuju operacije unije, presjeka i razlike, no za razliku od istoimenih operatora relacijske algebre djeluju nad grafovima kao kolekcijama (a ne skupovima)
3) **Operator kartezijskog produkta** – operator koji na temelju dvije kolekcije grafova kao rezultat vraća kolekciju grafova u kojoj se svaki graf sastoji od po jednog grafa iz svake ulazne kolekcije
4) **Operator spajanja** *(*eng. *join*) – operator koji nad kartezijskim produktom dvije kolekcije primjenjuje operator selekcije
5) **Operator kompozicije** – operator koji kreira nove grafove korištenjem informacija u pronađenim grafovima

3.3. Graf model podataka sa svojstvima

Graf model podataka sa svojstvima (eng. *property graph data model*) čini temelj graf baza podataka, a sadrži povezane entitete s atributima navedene u samoj definiciji grafa spomenutoj u poglavlju 3.1:

- **Čvorovi** predstavljaju entitete u stvarnom svijetu, svaki ima svoj identifikator i svojstva u obliku parova ključ-vrijednost, dok više čvorova može imati istu oznaku (jednu ili više)
- **Veze** su usmjerene, imenovane i semantički relevantne veze između dva čvora (entiteta u stvarnosti), svaka ima identifikator, smjer, tip te početni i završni čvor (izvorište i odredište), a može imati i svojstva (najčešće su kvantitativna poput težine, cijene i sl.)

Na modelu prikazanom na slici 4 nalaze se tri čvora koja su međusobno povezana s ukupno 6 veza. Svaki čvor ima svoj identifikator, a među njima se prema svojstvu *Type* mogu uočiti dva tipa čvorova: *Person* i *Group*. Čvorovi tipa *Person* imaju svojstva *Name* i *Age*, dok čvor tipa *Group* ima svojstvo *Name*.



Slika 4. Primjer graf modela podataka sa svojstvima (Izvor: http://blog.orgvue.com/graph-vs-relationship-databases prema Wikipedia, 2013)

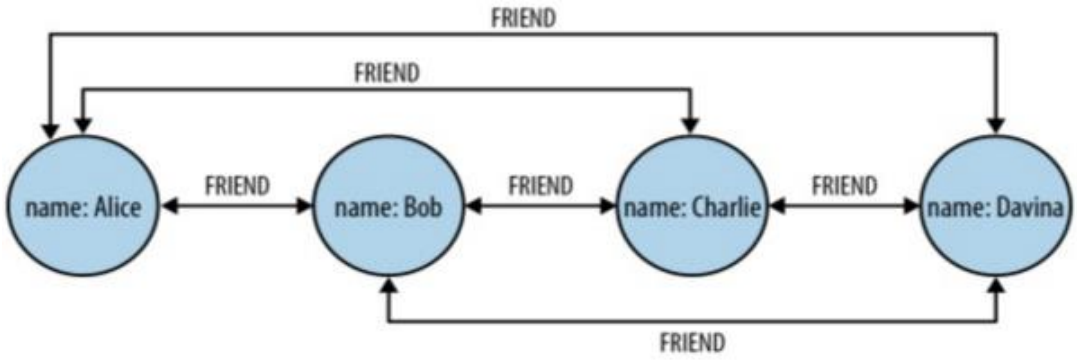
Slično tome, na promatranom modelu mogu se s obzirom na njihovu oznaku uočiti tri tipa veza: *knows*, *is_member* te *Members*. Veza tipa *knows* povezuje čvorove tipa *Person* i ima

svoj identifikator i svojstvo *Since*, dok veza tipa *is_member* povezuje čvor tipa *Person* s čvorom tipa *Group* te također ima svoj identifikator i svojstvo *Since*. Nasuprot tome, veza tipa *Members* ima samo identifikator i povezuje čvor tipa *Group* sa čvorom tipa *Person*.

3.4. Obrada podataka u grafu

Procesiranje podataka u graf bazama podataka jedna je od važnijih svojstava te tehnologije na koju treba obratiti pažnju. Naime, graf baze podataka koriste tzv. *susjedstvo bez indeksa* (eng. *index-free adjacency*) za povezivanje čvorova u grafu. To znači da je svaki čvor u grafu „fizički" povezan sa svojim susjednim čvorovima, odnosno u sebi sadrži vezu na druge čvorove.

Ovaj je mehanizam obrade podataka znatno brži od uobičajenih indeksa te se naziva eng. *native graph storage* (slika 5).



Slika 5. Nativna pohrana podataka u grafu (Izvor: http://www.slideshare.net/nasscom/graph-db-nasscom-29175277, 2013)

Za razliku od relacijskog modela u kojem se za pretraživanje podataka kreira posebna tablica koja predstavlja indeks povezanih tablica, odnosno podataka, u graf bazama podataka to nije potrebno jer svaki čvor održava referencu na svoj susjedni čvor, kao što svaka veza održava reference na svoje susjedne veze i čvorove koje ona povezuje. Na taj način je obrada podataka puno brža i efikasnija.

3.5. Pronalaženje uzoraka u grafu

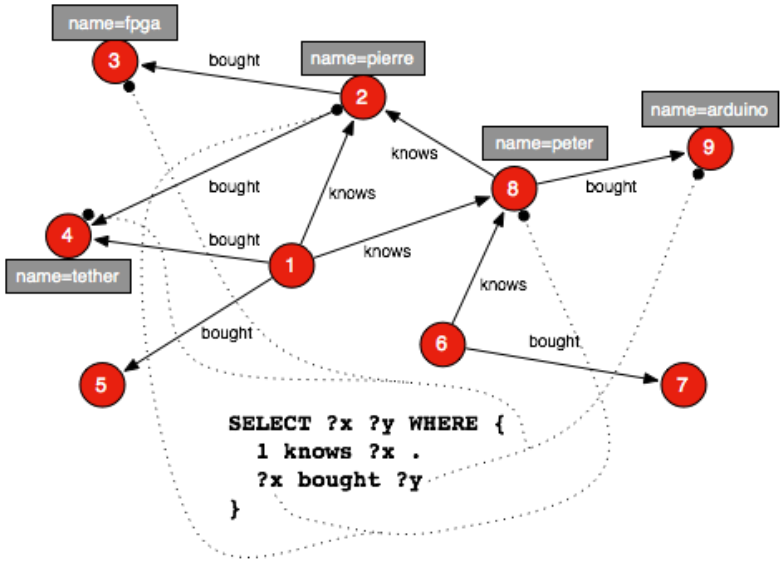
Kao što je već spomenuto, popularnost graf baza podataka i algoritama temeljenih na grafu u novije vrijeme sve više raste, posebice u znanstvenom okruženju.

Zahvaljujući tome i visokoj razini ekspresivnosti u modeliranju složenih podatkovnih struktura, broj mogućih primjena za modeliranje podataka u obliku grafa raste. Relativno brzo

pretraživanje grafa moguće je zahvaljujući algoritmu pronalaženja uzoraka u grafu koji se koristi u izvođenju upita nad graf bazama podataka. Uzorkom u grafu smatra se izomorfna slika tog grafa[2] (Valiente i Martinez, 1997), što predstavlja podgraf ciljnog grafa, pa se pronalaženje uzoraka u grafu (eng. *graph pattern matching*) često naziva i tzv. problem izomorfizma podgrafa.

Pronalaženje uzoraka u grafu je problem pronalaženja izomorfne slike tog grafa u drugom, promatranom grafu koji se naziva cilj (eng. *target*), odnosno pronalaženja svih podgrafova ciljnog grafa koji odgovaraju zadanom uzorku. Algoritam pronalaženja uzoraka u grafu se može smatrati ekvivalentnim algoritmu pretraživanja stabala. U tom su slučaju listovi stabla kandidati za izomorfizam grafova, a traženi je izomorfizam koji predstavlja najlakše pronaći pretraživanjem u dubinu (eng. *depth-first search*).

Primjer pronalaženja uzorka prikazan je na slici 6. Zadan je graf s čvorovima koji predstavljaju osobe i artikle koje te osobe mogu kupiti te veze koje označavaju da je osoba kupila artikl ili da zna neku drugu osobu.



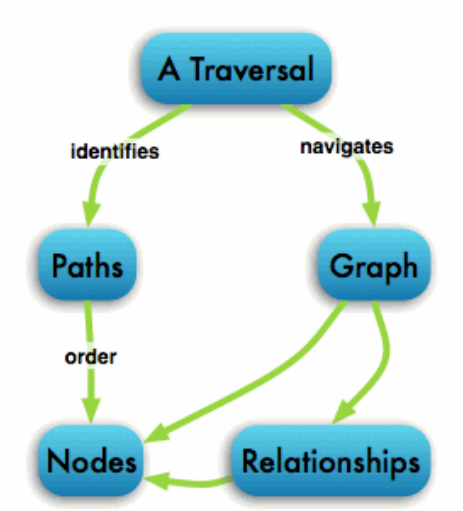
Slika 6. Pronalaženje uzoraka u grafu (Izvor: http://markorodriguez.com/2011/06/15/graph-pattern-matching-with-gremlin-1-1/, 2011)

Nad tako definiranim grafom zadan je upit u kojem se traže osobe koje poznaje neka osoba (čvor 1) te artikli koje je tražena osoba kupila. Algoritam pronalaženja uzorka u grafu pokušat će pronaći uzorak, odnosno podgraf, u kojem se nalazi čvor 1, sve njegove izlazne veze tipa *knows* prema drugim čvorovima te izlazne veze tipa *bought* tih čvorova. U ovom

[2] Izomorfizam grafova G i H je uređen par $f = (\theta, \varphi)$, gdje je θ bijekcija koja preslikava skup vrhova grafa G u skup vrhova grafa H, a φ bijekcija koja preslikava skup bridova grafa G u skup bridova grafa H (Klobučar, 2012)

slučaju, u traženom uzorku moguće je jedino da u varijabli „x“ budu pohranjeni čvorovi 2 i 8, a u varijabli „y“ čvorovi 3, 4 i 9.

Pronalaženje uzoraka u grafu predstavlja generalizaciju pronalaženja skupine znakova (eng. *string*) i dvodimenzionalnog pronalaženja uzoraka, zahvaljujući čemu grafovi mogu riješiti probleme u modeliranju višedimenzionalnih struktura. Osim traženja uzoraka u ciljnom grafu na razini čitavog grafa, u novije vrijeme se radi boljih performansi koriste i algoritmi koji uzorke pronalaze i na razini svake povezane komponente uzorka u ciljnom grafu. Istraživanja pokazuju da se u tim slučajevima izmorfizam bolje ponaša pa se uzorak pronalazi brže i s manje posjećenih čvorova i veza (Valiente i Martinez, 1997).



Slika 7. Obilazak grafa (Izvor: http://neo4j-org-dev.herokuapp.com/learn/graphdatabase, s.a.)

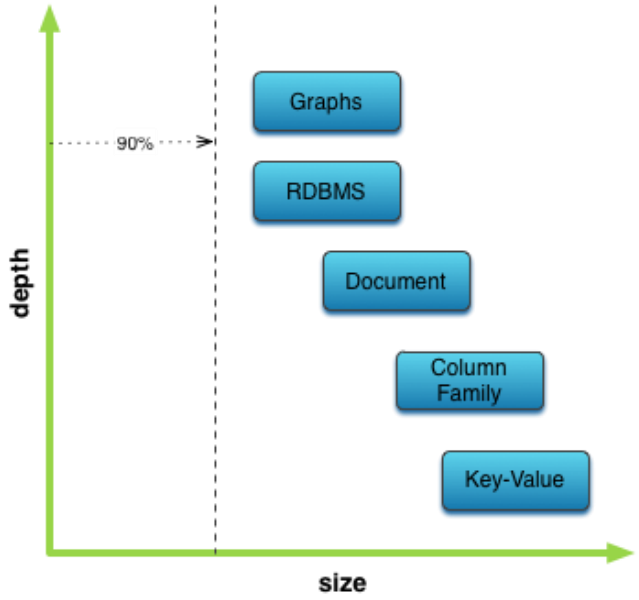
Važno je napomenuti razliku između pojmova obilazak grafa i pronalaženje uzoraka u grafu. Obilazak grafa odnosi se na način izvođenja upita nad grafom, gdje se kreće od početnog čvora prema povezanim čvorovima prema određenom algoritmu (Neo4j, s.a.). Cilj obilaska je pronaći putanje koje kreiraju određeni raspored čvorova kroz koje algoritam „prolazi“ tijekom izvođenja upita (slika 7).

3.6. Usporedba graf baza podataka s drugim bazama podataka

Graf baze podataka moguće je usporediti s relacijskim, ali i drugim kategorijama NoSQL baza podataka s obzirom na različite karakteristike.

Usporedbom relacijskih i svih kategorija NoSQL baza podataka s obzirom na dubinu pretraživanja i veličinu podataka (slika 8) može se uočiti da su graf baze podataka jedini predstavnik NoSQL baza podataka koji može postići veću dubinu pretraživanja podataka (više

razina) u odnosu na relacijske baze podataka. Međutim, kad se radi o sposobnosti pohrane, spremišta tipa ključ-vrijednost ostvaruju bolje performanse i mogu pohraniti više podataka u odnosu na relacijske i graf baze podataka.



Slika 8. Položaj baza podataka s obzirom na dubinu pretraživanja (Izvor: http://neo4j.com/developer/graph-db-vs-nosql/, s.a.)

3.6.1. Usporedba s relacijskim bazama podataka

Kao što im sam naziv kaže, graf baze podataka je jako dobro primijeniti za pretraživanje podataka strukturiranih u obliku grafa ili izvedenih stabala i kad je važno naglasiti značaj veza između povezanih podataka. Njihove prednosti najviše dolaze do izražaja kod zadavanja upita za čiju je obradu i pronalaženje rezultata potrebno napraviti obilazak (eng. *traversal*) kroz graf i pronaći traženi uzorak zadan u upitu. Nasuprot tome, relacijske baze podataka je najbolje primijeniti kod upita za pronalaženje svih zapisa u tablicama koji zadovoljavaju zadane uvjete ili pak kod korištenja funkcija za agregiranje podataka.

Zahvaljujući fizičkim pokazivačima na susjedne čvorove i veze, obilazak graf baza podataka je puno brži i efikasniji nego u relacijskim bazama podataka. Tamo se u tu svrhu koriste indeksi kreirani spajanjem tablica po vanjskim ključevima, a spajanje tablica, posebice ako ih ima više, troši vrijeme pa je zbog toga pretraživanje tih baza podataka često nedovoljno brzo i efikasno.

Razlike u performansama u izvođenju upita posebice dolaze do izražaja s porastom veličina baza podataka, odnosno povećanjem volumena pohranjenih podataka, ili kad je važna dubina obilaska, odnosno broj razina pretraživanja veza koji je potrebno proći da bi se došlo do traženih podataka. Primjerice, ako je zadana baza podataka s 1000 korisnika, gdje svaki prosječno ima 50 prijatelja, odnosno veza prema drugoj osobi, za utvrđivanje je li neka osoba povezana s drugom u manje od 4 „skoka" (obilaska veza) kod graf baza podataka potrebno je izdvojiti 2 milisekunde vremena, a kod relacijskih baza podataka to traje 2000 milisekundi. Ukoliko se broj korisnika u bazi podataka poveća na milijun, tada će prethodno opisani upit u graf bazama podataka trajati još uvijek 2 milisekunde, dok se nad relacijskim bazama podataka taj upit neće ni izvršiti u konačnom vremenu (Adell, 2011).

Relacijske baze podataka pružaju podršku za ACID[3] princip konzistentnosti podataka, zbog čega je prema CAP teoremu (eng. *Consistency, Availability, Partition Tolerance*) veća pažnja usmjerena prema očuvanju konzistentnosti i dostupnosti podataka. Posljedica toga je da relacijski DBMS sustavi ne mogu jamčiti toleranciju na particioniranje podataka, što ograničava mogućnost skaliranja baze podataka. Za razliku od toga, graf baze podataka su zahvaljujući primjeni drukčijeg principa konzistentnosti (eng. *Basic Availability, Soft-state, Eventual Consistency*, u daljnjem tekstu BASE[4]) skalabilnije.

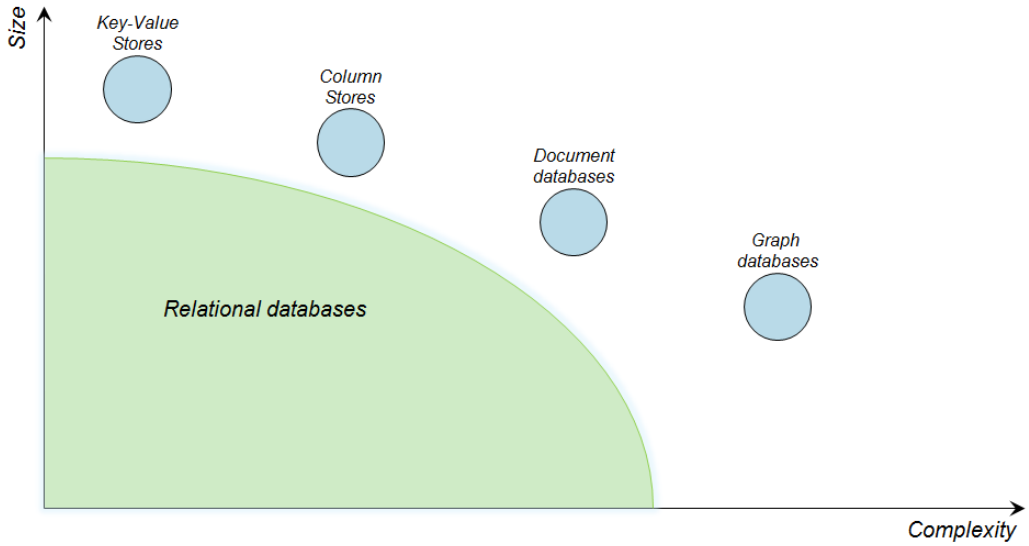
Graf baze podataka se mogu smatrati relacijskim bazama podataka sljedeće generacije koje naglašavaju važnost veza između čvorova (implicitnih veza putem vanjskih ključeva u relacijskim bazama podataka). Upravo je snaga tih veza čimbenik koji omogućuje graf bazama podataka pronalaženje rezultata u obliku asocijativnih skupova podataka u kojima su pohranjene informacije agregacije veza između čvorova, pa je izvođenje upita brže u odnosu na relacijske baze podataka (Merenyi, 2013).

[3] ACID princip je koncept koji se odnosi na 4 svojstva transakcija u bazama podataka: atomarnost (svaka transakcija se izvršava u potpunosti ili se ne izvršava), konzistentnost (samo valjani i ispravni podaci se zapisuju u bazu podataka), izolacija (transakcije se istovremeno obrađuju sigurno i individualno) te trajnost (po završetku transakcije sve promjene su spremljene na trajni medij pohrane) (Sasaki, 2015)

[4] BASE princip je koncept koji se odnosi na svojstva transakcija u bazama podataka: osnovna dostupnost (privid da baza podataka radi cijelo vrijeme), labavo stanje (spremišta nisu uvijek konzistentna za pisanje), eventualna konzistentnost (spremišta postižu konzistentnost s vremenom) (Sasaki, 2015)

3.6.2. Usporedba s ostalim NoSQL bazama podataka

NoSQL (eng. *Not only SQL*) je široka kategorija grupe perzistentnih rješenja koja se ne temelje na relacijskom modelu i ne koriste SQL kao upitni jezik (Neubauer, 2010). Graf baze podataka samo su jedna kategorija NoSQL baza podataka. Osim njih, u tu skupinu spadaju baze podataka temeljene na dokumentima (eng. *document databases*), spremišta tipa ključ-vrijednost (eng. *key-value stores*) te spremišta temeljena na stupcima (eng. *column stores*).

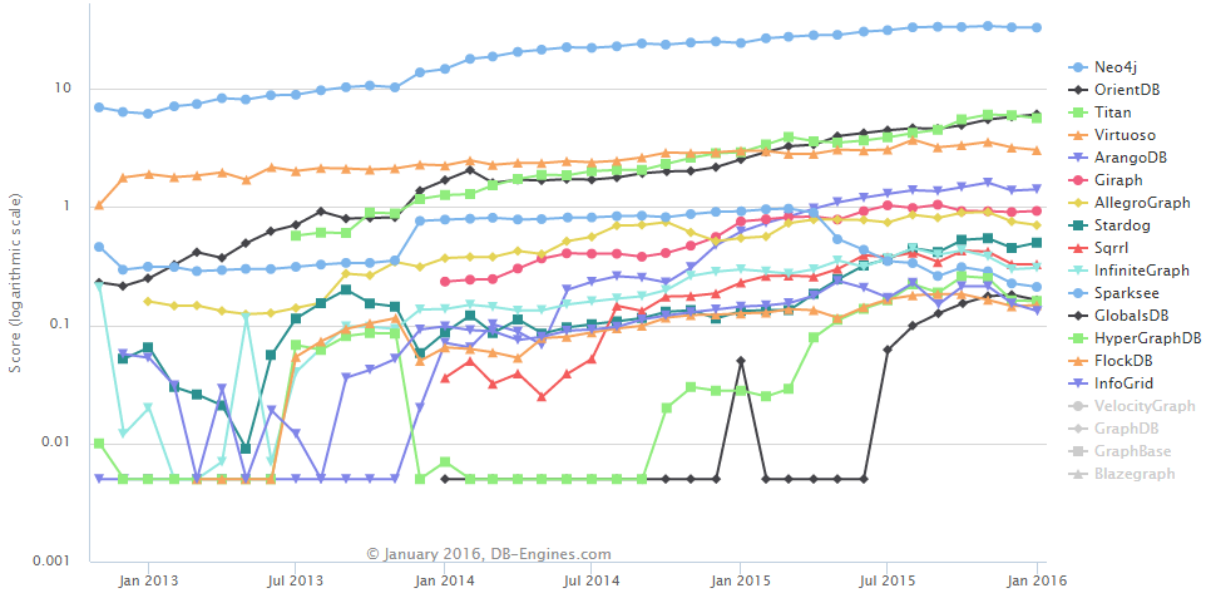


Slika 9. Ekosustav NoSQL baza podataka (Izvor: http://blog.octo.com/en/graph-databases-an-overview/, 2012)

U ekosustavu NoSQL baza podataka (slika 9) gledajući dubinu, odnosno razinu do koje različite kategorije baza podataka mogu pretraživati podatke graf baze podataka pružaju mogućnost pohrane i jednostavnog upravljanja kompleksnim i fleksibilnim modelima visoko povezanih podataka, dok, primjerice, spremišta tipa ključ-vrijednost mogu pohraniti samo jednostavne modele podataka koji su slabo međusobno povezani, ali veći u odnosu na graf baze podataka. Spremišta temeljena na stupcima i baze podataka temeljene na dokumentima pozicionirane su između analiziranih kategorija NoSQL baza podataka.

4. Sustavi za upravljanje graf bazama podataka

Sustavi za upravljanje graf bazama podataka imaju sličnu ulogu kao i relacijski DBMS sustavi opisani u poglavlju 2.2. Zastupljenost takvih sustava na tržištu u novije vrijeme sve više raste (slika 10).



Slika 10. Trendovi korištenja graf DBMS sustava na tržištu (Izvor: http://db-engines.com/en/ranking_trend/graph+dbms, 2016)

Najčešće korišteni graf DBMS je Neo4j čija popularnost konstantno raste. Slijede ga OrientDB i Titan koji, zajedno s Neo4j, ostvaruju najveći porast u korištenju i zastupljenosti na tržištu. O Neo4j graf DBMS sustavu će biti više riječi u nastavku. Osim njega, detaljnije će biti pojašnjen i Rexster server korišten u implementaciji aplikacije.

4.1. Neo4j

Neo4j je sustav otvorenog koda (najnovija verzija je 2.3.) koji pohranjuje podatke u graf čime ih je kasnije lakše dohvatiti (Kozielski i sur., 2015). Temelji se na mrežno-orijentiranom modelu podataka sa svojstvima u kojem su veze najvažniji objekti (Angles, 2012). Implementiran je u Java programskom jeziku i pripada skupini starijih NoSQL baza podataka.

Slika 11. Logotip Neo4j baze podataka (Izvor: http://neo4j.com/wp-content/themes/neo4jweb/assets/images/neo4j-logo-2015.png, 2015)

Neo4j pruža podršku za nativni pristup podacima, ali je podacima moguće pristupiti i putem Cypher i Gremlin upitnog jezika. Standardni jezik za upravljanje podacima u Neo4j bazi podataka je Cypher pa je Cypher upite moguće definirati i izvršavati putem Neo4j web sučelja. Zbog svoje robustnosti, skalabilnosti, opcionalne sheme i visokih performansi, Neo4j baze podataka moguće je koristiti i u velikim korporacijama (Neo4j, s.a.).

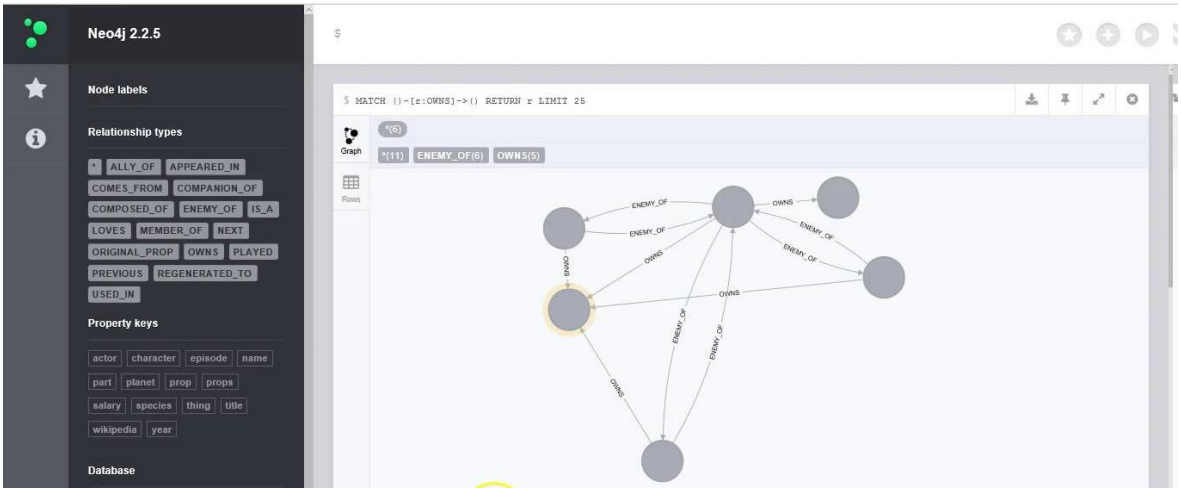
Neo4j bazu podataka karakterizira sljedeće (Angles, 2012):

- Pruža podršku za pohranu podataka u glavnu i vanjsku memoriju, ali i kreiranje indeksa nad bazom podataka
- Za upravljanje podacima nudi programska sučelja (eng. *Application Programming Interface*), ali ne i DML ni DDL jezik, dok ima djelomičnu sposobnost korištenja upitnog jezika (Cypher ili Gremlin)
- Moguće je pohraniti samo grafove sa svojstvima u kojima čvorovi i veze imaju svoje oznake i svojstva, a veze su usmjerene
- Pruža podršku za prikaz entiteta i relacija u obliku objektnih čvorova i čvorova s vrijednostima te veza

Upiti nad Neo4j bazom podataka se izvode pomoću obilazaka, a neki od čimbenika njene visoke zastupljenosti na tržištu su ACID princip konzistentnosti i visoka razina dostupnosti.

Za grafički pristup Neo4j bazi podataka u obliku korisničkog sučelja koristi se Neo4j web sučelje koje je dostupno korisnicima na lokalnoj IP adresi http://127.0.0.1:7474/ nakon što instaliraju Neo4j Server (slika 12).

Osim ovog, Neo4j nudi komandnu liniju za izvršavanje Cypher upita (*Neo4j Shell*) koja je od velike koristi za razvoj Neo4j baze podataka (Neo4j, s.a.).



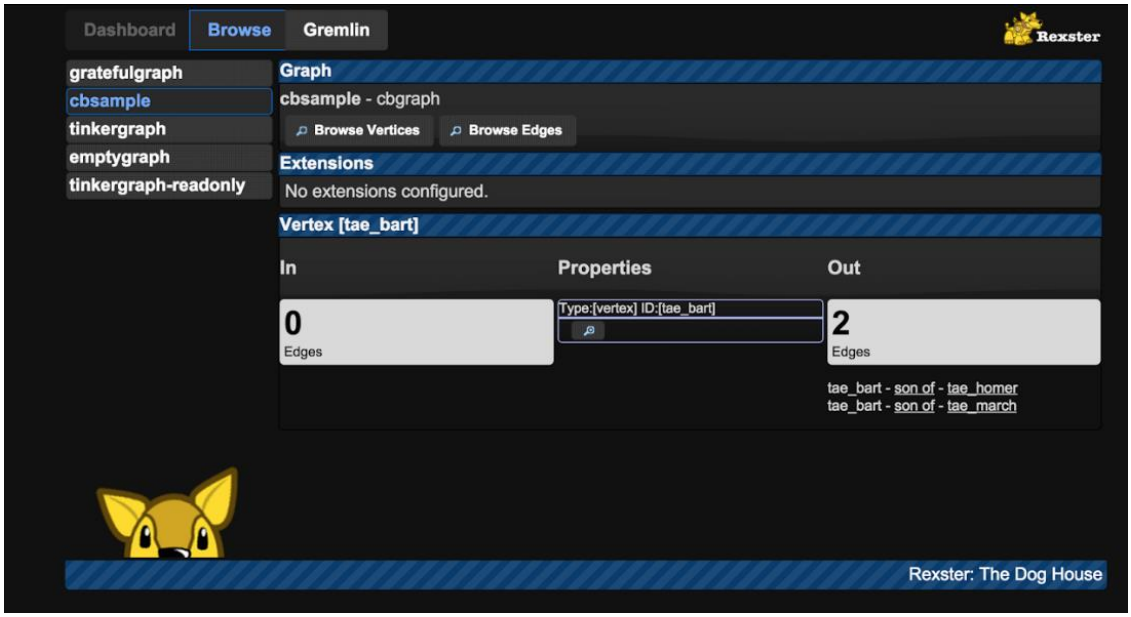
Slika 12. Neo4j web sučelje (Izvor: https://i.ytimg.com/vi/4zD9uwwmbUY/maxresdefault.jpg, 2015)

4.2. Rexster

Rexster je server za graf baze podataka koji omogućuje pristup graf bazi podataka putem HTTP/REST[5] protokola i binarnog protokola RexPro koji omogućuje slanje Gremlin skripti na udaljenu instancu Rexster servera (Rodriguez, 2015). Korištenje HTTP protokola pruža podršku za svoje metode GET, POST, PUT i DELETE, fleksibilan model podataka koji je moguće proširiti brojnim ekstenzijama te procedure pohranjene na serveru za brže izvršavanje Gremlin upita.

[5] HTTP/REST (eng. *Hypertext Transfer Protocol/Representational State Transfer*) je stil arhitekture za dizajniranje mrežnih aplikacija koji omogućuje dohvaćanje mrežnih resursa putem HTTP protokola (Elkstein, s.a.)

Grafički način pristupa Rexster serveru putem korisničkog sučelja moguć je putem *DogHouse* sučelja koje korisniku omogućuje pregled čvorova, veza i njihovih svojstava na jednom mjestu (slika 13), ali i izravno pisanje Gremlin upita u Gremlin konzoli.



Slika 13. Doghouse sučelje Rexster servera (Izvor: http://nosqlgeek.blogspot.hr/2015/06/using-rexter-27-graph-server-with.html, 2015)

5. Upitni jezici za graf baze podataka

Povećanje mogućih područja primjene graf baza podataka tijekom nekoliko posljednjih desetljeća dovelo je do razvoja nekoliko različitih upitnih jezika za graf baze podataka. Ti su upitni jezici nastajali pod različitim utjecajima: hipertekstualnih sustava u 80-im godinama prošlog stoljeća, polustrukturiranih podataka i objektnih baza podataka u 90-im godinama prošlog stoljeća pa sve do najnovijeg utjecaja semantičkog weba i društvenih mreža (Wood, 2012). Kao što je već spomenuto u prethodnim poglavljima, prilikom izvođenja upita koriste se različiti algoritmi obilaska grafa koji traže zadani uzorak.

Općenita shema obilaska u svakom od tih algoritama je da obilazak počinje u jednom vrhu iz kojeg se on nastavlja preko bridova tog vrha prema ostalim vrhovima grafa. Putem posjećenih bridova obilazak se nastavlja prema posjećenim vrhovima koji imaju neposjećene bridove tako dugo dok se ne obiđu svi bridovi grafa.

5.1. Algoritmi za obilazak grafa

Iako postoji puno algoritama obilazaka grafa, u ovom radu bit će spomenuti i ukratko pojašnjeni sljedeći algoritmi:

- Algoritam traženja u dubinu (eng. *depth-first search*, u daljnjem tekstu DFS)
- Algoritam traženja u širinu (eng. *breadth-first search*, u daljnjem tesktu BFS)
- Dijkstrin algoritam
- Algoritam tri boje (eng. *tricolor algorithm*)
- Ullmanov algoritam pronalaženja uzoraka

Osim Ullmonovog algoritma, za pronalaženje uzoraka u grafu mogu se koristiti i VF2, GraphQL, QuickSI i drugi algoritmi.

5.1.1. Algoritam traženja u dubinu

Algoritam traženja u dubinu primjenjuje se u obilaženju i pretraživanju stabala i struktura podataka u obliku grafa. Nakon što se graf prikaže u obliku stabla, ovaj je algoritam ekvivalentan tzv. *preorder* algoritmu obilaska stabla u kojem se najprije obilazi vrh, a potom njegova djeca, odnosno u terminologiji grafova, njegovi susjedni vrhovi. Kao zaštita od beskonačnih petlji u obilasku istih vrhova, ovaj algoritam koristi zastavice koje označavaju je li neki vrh već bio posjećen ili nije. Ideja ovog algoritma je čim dublje obilaziti, tj. posjećivati vrhove samo jedanput, sve dok se nije potrebno vratiti natrag na višu razinu.

Pseudokod algoritma traženja u dubinu je sljedeći (Heap, 2002):

```
DFS(G,v)
{
   stog S := {};
   za svaki vrh u
      postavi posjecen[u] u false;
   stavi vrh v na stog S;
   dok S nije prazan
     {
     uzmi vrh u sa stoga S;
     ako je posjecen[u] = false tada
        postavi posjecen[u] u true;
        za svaki neposjeceni vrh w od u
           dodaj w na stog S;
     }
  }
```

Pretpostavka ovog algoritma je da je graf povezan. Ako to nije slučaj u zadanom grafu, tada algoritam neće posjetiti one vrhove koji nemaju zajedničkih veza s posjećenim vrhovima. Konačni rezultat ovog algoritma je šuma (skup stabala) koja prikazuje povezanost komponenti grafa (Heap, 2002). Danas se mnogi algoritmi za graf baze podataka temelje upravo na algoritmu traženja u dubinu (npr. algoritam tri boje).

5.1.2. Algoritam traženja u širinu

Algoritam traženja u širinu po svojim je karakteristikama sličan prethodno prikazanom algoritmu. Ovaj algoritam obilazi graf slojevito po razinama, odnosno najprije obilazi sve susjedne vrhove odabranog vrha razine 0 i pridružuje im razinu 1, nakon toga njihove susjedne vrhove (razina 2) i tako redom dok ne obiđe sve vrhove grafa. Vrlo je učinkovit u već spomenutom problemu pronalaženja uzoraka u grafu pa se i najčešće koristi u tu svrhu.

Pseudokod algoritma traženja u širinu je sljedeći (Heap, 2002):

```
BFS(G,v)
{
   red Q := {};
   za svaki vrh u
      postavi posjecen[u] u false;
   dodaj vrh v u red Q;
   za svaki susjedni vrh w od v
     dodaj w u Q
   postavi posjecen[v] u true
   dok Q nije prazan
     {
          makni vrh u iz reda Q;
          za svaki susjedni vrh u od w
          {
            ako je posjecen[u] false
                postavi posjecen[u] u true;
                dodaj vrh u u red Q
          }
          postavi posjecen[w] u true
     }

 }
```

5.1.3. Dijkstrin algoritam

Dijkstrin algoritam koristi se za pronalaženje najkraćeg puta između dva vrha grafa gdje su svim bridovima grafa pridružene nenegativne težine. Kao i prethodni algoritmi, i ovaj algoritam započinje od odabranog početnog vrha te postupno izgrađuje stablo obilaženjem svih vrhova grafa pritom odabirući vrhove najbliže početnom vrhu (najmanja težina bridova koji ih povezuju). Pomoću ovog algoritma moguće je istovremeno pronaći najkraći put od izvorišnog vrha prema svim ostalim vrhovima grafa. Pretpostavka ovog algoritma je da je graf koji reprezentira najkraće puteve od početnog do ostalih vrhova razapinjuće stablo.

Na početku algoritma inicijalizira se tzv. jedinstveni izvor (eng. *single-source*) gdje se za svaki vrh grafa definira da nema prethodnika (susjednih vrhova), a procjene troška svakog puta između vrhova se postavljaju na beskonačnost.

Pojednostavljeni pseudokod ovog algoritma je sljedeći (Morris, 1998):

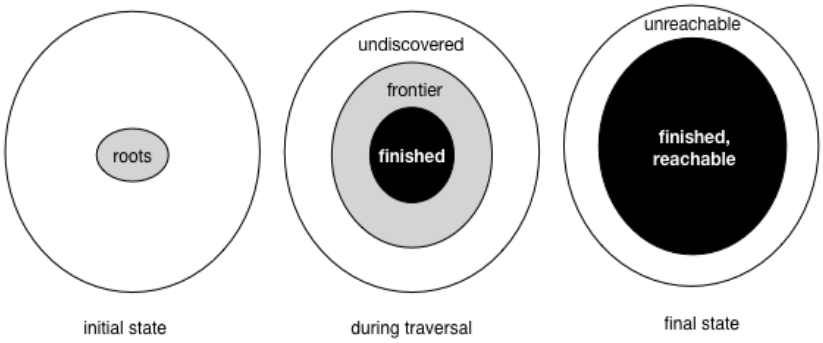
```
Dijkstra(G, v)
    {
        inicijaliziraj jedinstveni izvor (G, v);
        skup S := {0};
        dodaj vrhove grafa G u red Q
        dok Q nije prazan
        {
            pronađi vrh u u redu Q s najkraćim putem
            dodaj vrh u u skup S
            za svaki susjedni vrh w od u
                ažuriraj trošak puta od u prema w
        }
    }
```

5.1.4. Algoritam tri boje

Ovaj algoritam omogućuje prikaz obilaska grafa na apstraktnoj razini, što se može iskoristiti u već opisanim algoritmima (BFS, DFS i Dijkstrin). Naime, ovaj algoritam svakom vrhu grafa pridružuje jednu od tri moguće boje koje se mijenjaju tijekom vremena (Myers, 2012):

- **Bijela** boja označava da vrh dosad nije bio posjećen u trenutnom obilasku, ali i da možda ne može biti posjećen.
- **Siva** boja označava da je vrh grafa posjećen, ali algoritam još nije završio s njihovim obilaskom.
- **Crna** boja označava da je vrh moguće posjetiti i da je algoritam posjetio taj vrh, tj. da je taj vrh već posjećen.

Napredak algoritma kroz vrijeme izvršavanja prikazan je na slici 14. Na početku su korijenski vrhovi označeni sivom bojom, dok su svi ostali vrhovi grafa bijele boje.



Slika 14. Napredak algoritma tri boje kroz vrijeme (Izvor: http://www.cs.cornell.edu/Courses/cs2112/2015fa/lectures/lec_traversals/tricolor.png, 2012)

S vremenom se udio vrhova s bijelom bojom smanjuje, dok se udio vrhova s crnom bojom koji su posjećeni i s kojima je algoritam završio povećava. U konačnom stanju, kad algoritam završi sa svojim izvođenjem, svi su vrhovi, osim onih koje nije moguće posjetiti (imaju bijelu boju), označeni crnom bojom.

5.1.5. Ullmanov algoritam pronalaženja uzoraka

Ovaj je algoritam prvi praktični algoritam namijenjen pronalaženju uzoraka u grafu, odnosno pronalaženju izomorfnih podgrafova u grafu (Lee i sur., 2013). Spada u kategoriju algoritama s približnim pronalaženjem uzoraka u grafu, a omogućuje pronalaženje rješenja uvećavanjem djelomičnih rješenja ili njihovim napuštanjem kad utvrdi da ih nije moguće izvršiti, odnosno da ne odgovaraju traženom uzorku.

Cilj ovog algoritma je za zadani graf utvrditi i pronaći postoji li podgraf koji odgovara traženom uzorku koji se također može smatrati grafom. U obilaženju grafa koristi se algoritmom traženja u dubinu i temelji se na matricama susjedstva između vrhova grafa (Vogl, 2011). Algoritam je rekurzivan i za njegovo izvršavanje potrebno je najprije napraviti matrično množenje za svaki čvor list u rekurzivnom stablu pa je ponekad skup za izvođenje (Neumann, s.a.). Danas je još uvijek najpoznatiji i najkorišteniji algoritam za pronalaženje uzoraka u grafu ili podgrafu.

Kao poboljšanja ili zamjena Ullmanovog algoritma u novije su vrijeme razvijeni dodatni algoritmi poput VF2, GraphQL, QuickSI, SPath, GADDI itd. Performanse tih algoritama su ispitane u jednom istraživanju autora J. Lee-a i suradnika u kojem su zaključili kako se prosječni broj rekurzivnih poziva kod svakog algoritma smanjuje s povećanjem veličine upita (Lee i sur., 2013). Istraživanje je pokazalo kako je GraphQL algoritam s najmanje rekurzivnih poziva, dok je s obzirom na performanse QuickSI najbrži algoritam.

5.2. Izvođenje upita na graf bazama podataka

Zbog širenja područja primjene graf baza podataka tijekom vremena su razvijeni brojni upitni jezici za pretraživanje i upravljanje grafovima s različitim atributima i strukturama. Povećana heterogenost i veličina podataka pohranjenih u graf bazama podataka stvorila je potrebu za nativnim pristupom bazi podataka. Suvremeni koncept upita koji upravljaju grafom na općenitoj, globalnoj razini označava da se ograničenja i pravila nad čvorovima i vezama u grafu mogu definirati istovremeno nad skupom čvorova i veza koji čine

traženi objekt, odnosno podgraf promatranog grafa, za razliku od dosadašnjeg načina definiranja nad svakim pojedinačnim čvorom ili vezom u više iteracija (He i Singh, 2008).

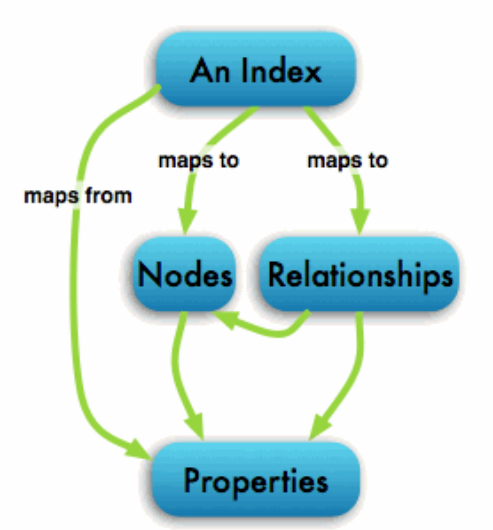
Svaki upit nad graf bazom podataka prima ulazne podatke u obliku uzorka koji je potrebno pronaći u grafu, pomoću algoritama za pronalaženje uzoraka objašnjenih u prethodnom poglavlju pronalazi podgraf s traženim uzorkom u grafu (može ih biti više) te vraća taj podgraf ili više njih kao izlazni rezultat izvođenja. Takav se upit u literaturi naziva eng. *graphs-at-a-time* upit (He i Singh, 2008).

Današnji upitni jezici za graf baze podataka trebaju pružati podršku za sljedeće karakteristike (He i Singh, 2008):

- Grafovi su osnovna jedinica za informacije – svrha upita je pružiti traženu informaciju korisniku u rezultatu izvođenja, a kod upitnih jezika za graf baze podataka rezultat izvođenja je jedan ili više grafova
- Upiti trebaju biti tipa *graphs-at-a-time*, što znači da za traženi uzorak kao ulaz upit vraća jedan ili više grafova kao izlaz

5.2.1. Indeksiranje graf baza podataka

Indeksi u graf bazama podataka su najkorisniji za ubrzanje postupka pronalaženja uzoraka u grafu, odnosno za ubrzavanje izvršavanja odgovarajućih algoritama za pronalaženje uzoraka. Indeks predstavlja poseban oblik obilaska grafa u kojem je, primjerice, na temelju zadanog svojstva moguće relativno brzo putem preslikavanja otkriti koji čvorovi ili veze ga sadrže (Neo4j, s.a.) (slika 15).



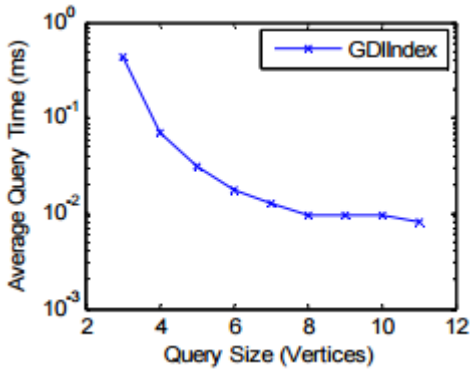
Slika 15. Indeks nad graf bazom podataka (Izvor: http://neo4j-org-dev.herokuapp.com/learn/graphdatabase, s.a.)

Indeks nad graf bazom podataka sastoji se od dva dijela, odnosno dvije podatkovne strukture (Williams i sur., 2007):

1) **Usmjereni aciklički graf** koji sadrži čvor za svaki jedinstveni podgraf originalnog grafa
2) **Tablica sažetaka** (eng. *hash table*) pomoću koje se pretražuju podgrafovi radi bržeg pronalaženja izomorfizama

U tablicu sažetaka spremaju se sažeti ključevi podgrafova dobiveni matematički pomoću matrica susjedstva neovisno o njihovom izomorfizmu.

William i suradnici u svom članku kao metodu indeksiranja graf baza podataka predlažu postupak dekompoziciju grafa. Dekompoziciju grafa definiraju kao „proces izvođenja dijelova komponenti iz grafa korištenjem raspoloživog skupa operacija“ (Williams, 2007). Rezultat dekompozicije grafa je kolekcija svih povezanih i uočenih podgrafova u promatranom grafu.



Slika 16. Odnos veličine upita i vremena izvršavanja upita nad indeksom
(Izvor: Williams i sur., 2007)

U promatranom članku autori su proveli istraživanje utjecaja kreiranja indeksa na vrijeme trajanja upita za pronalaženje uzoraka, tj. izomorfizama u grafu. Rezultati provednih istraživanja prikazani su na slici 16 i pokazuju značajno poboljšanje u vremenu izvođenja upita nakon kreiranih indeksa nad graf bazom podataka. S povećanjem upita (brojem čvorova uključenih u upitu) eksponencijalno se smanjuje prosječno vrijeme izvođenja upita.

5.2.2. G upitni jezik

Jedan od najranijih upitnih jezika za graf baze podataka razvili su davne 1987. godine Cruz, Mendelzon i Wood i nazvali ga G. Ovaj je jezik izvršio utjecaj na brojne upitne jezike sljedećih generacija (Barcelo, 2013).

Budući da je nastao u vrijeme kad je utjecaj i primjena graf baza podataka bila neznatna u odnosu na relacijske baze podataka pa su u njih uglavnom bili pohranjeni jednostavni podaci što je strukturu grafova činilo jednostavnom, upitni jezik G koristi jednostavnu sintaksu i semantiku za pronalaženje traženih podataka u graf bazama podataka. Ovaj upitni jezik koristi podatkovni model u obliku označenih, usmjerenih grafova.

Sintaksu upitnog jezika G čine upiti koje čine skupovi parova grafova, gdje se svaki par sastoji od traženog uzorka u grafu (eng. *graph pattern*) i agregiranog grafa (eng. *summarized graph*) (Wood, 2012). Dakle, upit je unija tzv. uzoraka u grafu, gdje svaki uzorak predstavlja graf bazu podataka u kojoj je svaki čvor identifikator čvora ili varijabla čvora, a svaka veza je označena regularnim izrazom nad abecedom proširenom svim varijablama oznaka (Barcelo, 2013). Formalno, to izgleda ovako:

Upit Π u jeziku G nad konačnom abecedom Σ je izraz oblika $\bigcup_{1 \le i < k} \pi_i \ (k \geq 1)$, *gdje je svaki* $\pi_i (1 \leq i \ < k)$ *uzorak u grafu nad Σ.*

Semantika upitnog jezika G ovisi o semantici uzoraka u grafu koja se temelji na homeomorfizmu, odnosno funkcijama koje preslikavaju varijable čvorova u njihove identifikatore, varijable oznaka u elemente abecede Σ te svaku vezu označenu regularnim izrazom u jednostavnu putanju u graf bazi podataka (Barcelo, 2013).

5.2.3. GraphLog upitni jezik

GraphLog upitni jezik su kreirali Consens i Mendelzon da bi nadomjestili nedostatak ekspresivnosti ranije kreiranog upitnog jezika G. Ovaj je jezik karakterističan po tome što prikazuje i podatke i upite kao grafove (Barcelo, 2013). Osnovna jedinica za definiranje upita su čvorovi i veze te se oni obrađuju pomoću pravila logike prvog reda. Kao i kod prethodnog upitnog jezika, podatkovni model predstavlja označeni i usmjereni graf.

U GraphLog jeziku kao poboljšanoj verziji jezika G agregirani graf u uzorku zamijenjen sa specifičnom vezom te su radi povećanja ekspresivnosti dodane operacije nad vezama (inverzija, negacija) te funkcije agregacije. Ono po čemu se ovaj upitni jezik najviše razlikuje od jezika G je semantika jezika. Naime, budući da je po svojim karakteristikama GraphLog ekvivalentan Datalog logičkom jeziku, na njemu se i temelji značenje upita definiranih u jeziku GraphLog (njihova semantika).

GraphLog omogućava definiranje tzv. vizualnih upita koji označavaju uzorke u grafu u kojima su čvorovi označeni nizom varijabli i konstanti, a veze su označene regularnim izrazima putanja nad relacijama (Consens i Hasan, 1993). Tijekom izvođenja upita pomoću algoritama se pronalaze sve pojavnosti traženog uzorka u grafu te se nad pronađenim instancama izvršavaju određene operacije.

5.2.4. GraphQL upitni jezik

GraphQL je upitni jezik koji, za razliku od prethodnih upitnih jezika, kao osnovnu jedinicu za definiranje i izvođenje upita koristi upravo graf. Upiti definirani u ovom upitnom jeziku su orijentirani na skupove i prikazuje polustrukturirane podatke zbog njihove heterogenosti. S obzirom na skup operacija koje je moguće izvršavati u ovom upitnom jeziku, GraphQL ima algebarski sustav sličan SQL upitnom jeziku (algebra grafova) pa je zadržao njegovu ekspresivnost.

Osnovna jedinica za upite kod ovog upitnog jezika je uzorak u grafu koji se sastoji od strukture grafa i predikata (He, 2007). Kao osnovna jedinica informacije koristi se graf, pritom GraphQL pruža podršku za grafove s varijabilnim atributima i veličinom podataka pa svaki čvor, veza ili graf mogu imati proizvoljan broj atributa (1 ili više). Osim navedenih karakteristika, GraphQL je efikasan jezik koji je moguće implementirati tako da može pružiti zadovoljavajuće performanse u kompleksnijim područjima primjene.

5.2.5. Cypher upitni jezik

Cypher upitni jezik je danas najkorišteniji i najperspektivniji upitni jezik za graf baze podataka te se u njegov razvoj i najviše ulaže. Pripada skupini deklarativnih upitnih jezika (prilikom definiranja upita potrebno je samo definirati traženi rezultat, a ne i način njegova dobivanja) te je po sintaksi sličan SQL-u, što utječe na jednostavnost učenja i razumijevanja tog upitnog jezika. Službeni je jezik za izvođenje upita nad Neo4j bazom podataka o kojoj će više riječi biti u sljedećim poglavljima.

Cypher omogućuje efikasno i ekspresivno pretraživanje, ali, za razliku od prethodnih upitnih jezika, pruža podršku za operacije kreiranja, ažuriranja te brisanja. Struktua Cypher upitnog jezika posuđena je od SQL-a pa se upiti definiraju korištenjem različitih naredbi (eng. *clause*). U jednom upitu moguće je koristiti više naredbi.

Upiti definirani u Cypher upitnom jeziku u svojoj sintaksi slijede pristup u obliku dijagrama, što omogućuje primjenu graf baza podataka u različitim interesnim područjima (Raj, 2015). Sintaksa zadanog upita je ekvivalentna stvarnoj strukturi grafa.

Temelj ovog upitnog jezika čine uzorci i pronalaženje uzoraka zadanih u upitu u promatranom grafu, što čini Cypher efikasnijim u odnosu na druge upitne jezike za graf baze podataka. Korištenjem uzoraka moguće je prilikom definiranja upita odrediti oblik traženog rezultata.

Naredbe za upravljanje podacima u Cypher upitnom jeziku se mogu podijeliti u sljedeće skupine/kategorije:

- Opće naredbe
- Naredbe za čitanje, tj. pretraživanje
- Naredbe za pisanje

Osim navedenih naredbi, Cypher omogućuje izvršavanje funkcija (predikata, skalarnih, matematičkih, znakovnih, skupovnih) nad podacima, a od verzije Neo4j 2.0 graf baze podataka podržano je i kreiranje opcionalne sheme podataka temeljene na konceptu oznaka (eng. *labels*) koju čine indeksi i ograničenja (Neo4j Manual, s.a.).

5.2.5.1. Opće naredbe

U ovu skupinu naredbi pripadaju naredbe koje se mogu koristiti u svrhu čitanja (dohvaćanja) podataka, ali i zapisivanja podataka u bazu podataka. Korištenjem ovih naredbi ne mijenja se rezultat izvršenog upita (sadržaj), nego njegova struktura. Po svojoj su ulozi, tj. namjeni, jednaki istoimenim naredbama u SQL upitnom jeziku za relacijske baze podataka.
Najpoznatije naredbe ove skupine su:

- **RETURN**
  - definira što treba sadržavati rezultat izvršenog upita, odnosno koji su dijelovi zadanog uzorka od interesa korisniku (čvorovi, veze ili njihovi atributi)
- **ORDER BY**
  - Klauzula koja označava da rezultat upita treba sortirati te definira kako (uzlazno ili silazno) te u upitu definira nakon RETURN naredbe
  - Korištenje ove naredbe i agregiranih izraza je međusobno isključivo zbog sprječavanja promjene rezultata

- **LIMIT**
    - Ograničava broj redova u rezultatu izvršenog upita te kao argument uzima bilo koju pozitivnu cjelobrojnu vrijednost koja nije čvor ni veza
- **SKIP**
    - Definira od kojeg reda treba započeti prikazivati redove rezultata upita
    - Korištenjem ove naredbe konačni skup koji predstavlja rezultat upita se obrađuje od početka
- **UNION**
    - Koristi se za kombiniranje rezultata više različitih upita u jedan konačni skup rezultata koji uključuje sve redove dobivene u svim upitima

5.2.5.2. Naredbe za pretraživanje podataka

Naredbe ove skupine koriste se za dohvaćanje podataka iz baze podataka te su semantički ekvivalentne SELECT naredbi u SQL-u. Svaki je podatak zapisan u obliku ključ-vrijednost pa se traženi podaci dohvaćaju na temelju njihovih ključeva.

Najpoznatije naredbe ove skupine su:

- **MATCH**
    - Koristi se za pronalaženje uzorka opisanog u toj naredbi te omogućuje definiranje uzorka koji će Cypher tražiti u graf bazi podataka
- **OPTIONAL MATCH**
    - Slično kao i MATCH naredba, koristi se za pronalaženje traženog uzorka u grafu, ali koristi NULL vrijednosti za dijelove uzorka koji nisu pronađeni u grafu
    - Uzorak može biti pronađen u cijelosti ili on nije pronađen ako ima dijelova koji nedostaju
    - Ekvivalent vanjskom spajanju (eng. *outer join*) u SQL-u
- **START**
    - Koristi se za pronalaženje ishodišne točke uzorka (čvor ili veza) od koje kreću algoritmi za pronalaženje uzorka
- **WHERE**
    - Dodaje ograničenja općenitim upitima s MATCH ili OPTIONAL MATCH naredbama te ih tako čini specifičnima i filtrira njihove rezultate

- **AGGREGATION**
    - Omogućuje agregiranje podataka (ekvivalentno naredbi GROUP BY u SQL-u)
    - Uzima ulazne vrijednosti i izračunava agregirane vrijednosti na temelju njih
    - **DISTINCT** modifikator uklanja duplikate iz ulaznih vrijednosti

5.2.5.3. Naredbe za pisanje podataka

Ova skupina naredbi se koristi za zapisivanje, ažuriranje te brisanje podataka iz baze podataka. Najpoznatije naredbe ove skupine su:

- **CREATE**
    - Koristi se za kreiranje elemenata baze podataka (čvorova i veza)
    - Moguće je kreirati individualne elemente ili sve elemente zadanog uzorka
- **MERGE**
    - Osigurava da traženi uzorak postoji u grafu tako da kreira njegove elemente ukoliko oni ne postoje u graf bazi podataka
    - Kombinacija MATCH i CREATE naredbi s mogućnošću definiranja ponašanja u slučaju da uzorak postoji (ON MATCH) ili ne postoji pa je kreiran (ON CREATE)
- **SET**
    - Koristi se za ažuriranje oznaka čvorova i svojstava čvorova i veza u bazi podataka, što uključuje njihovo brisanje i dodavanje
- **DELETE**
    - Koristi se za brisanje elemenata graf baze podataka (čvorova, veza ili cijelih putanja/uzorka)
    - Brisanjem čvorova brišu se i njihove veze prema susjednim čvorovima
- **REMOVE**
    - Koristi se za brisanje svojstava i oznaka elemenata graf baze podataka

5.2.5.4. Optimizacija Cypher upita

Za ostvarivanje boljih performansi Cypher upita S. Raj u svojoj knjizi predlaže nekoliko tehnika (Raj, 2015), od kojih su najznačajnije:

- Izbjegavanje skeniranja podataka na globalnoj razini tako da se kod definiranja upita u rezultatu traži samo ono što je stvarno potrebno korisniku jer Cypher kao pohlepni

upiti jezik (eng. *greedy*) neće zanemariti neke podatke osim ako to nije eksplicitno zadano

- Indeksiranje i postavljanje ograničenja za brže pretraživanje, čime se izbjegavaju nepotrebne i suvišne provjere traženog uzorka u bazi podataka
- Korištenje više uzoraka u MATCH nego u WHERE naredbama jer WHERE naredba primarno nije namijenjena pronalaženju uzoraka, već filtriranju pronađenih rezultata pa je pronalaženje uzoraka definiranih u MATCH naredbi brže
- Parametriziranje upita omogućava da Cypher ponovno koristi postojeće upite, bez da svaki put obrađuje upite zadane pomoću literala koji zahtijevaju dodatne resurse

5.2.6. Gremlin upitni jezik

Gremlin je upitni jezik specifičan za domenu koji se koristi za obilazak grafova sa svojstvima (GremlinDocs, s.a.). Nastao je kao projekt otvorenog koda koji održava Tinkerpop pod sponzorstvom Apache Software Foundation korporacije.

Gremlin je ekspreksivni jezik koji ima konstrukte temeljene na povezanim operacijama za efikasan obilazak grafa. Orijentiran je prema putanjama za kompleksno obilaženje grafa, a pomoću njega se mogu dohvaćati, analizirati i modificirati podaci u grafu (Titan Documentation, s.a.). Omogućuje izvršavanje kompleksnih funkcionalnosti koje se nazivaju koraci (eng. *steps*), a koji su pridruženi obilasku grafa.

Slika 17. Logotip Gremlin upitnog jezika (Izvor: https://upload.wikimedia.org/wikipedia/en/5/54/Gremlin_(programming_language).png, 2015)

Podržan je u većini graf baza podataka, a, budući da se koristi za grafove sa svojstvima (eng. *property graph*), može se koristiti i s Neo4j bazom podataka. Danas je uz Cypher najkorišteniji upitni jezik za graf baze podataka. Najčešće se koristi u komandnoj liniji u kojoj je moguće definirati interaktivne upite. Razvijen je u programskom jeziku Java, točnije Groovy pa je poput Jave i Gremlin i virtualni stroj i upitni jezik (Rodriguez, 2015). Gremlin virtualni stroj održava biblioteku koraka definiranih u upitu, gdje je redoslijed koraka proizvoljan.

Upit napisan u upitnom jeziku Gremlin je lanac operacija koje se evaluiraju slijeva nadesno (Titan Documentation, s.a.), čime se stvara putanja za obilazak grafa, na čemu se i temelji ovaj upitni jezik. Putanja se kreira spajanjem operacija operatorom „.“, tako da je svaka operacija korak obilaska u kojem se obrađuje objekt dobiven u operaciji koja je prethodila trenutnom koraku. Moguće je definirati neograničen broj ovakvih koraka, dok se promjenom redoslijeda definiranih operacija mijenja semantika upita. Za pisanje Gremlin upita potrebno je najprije pozvati funkciju koja će kreirati graf određenog tipa. Primjerice, za kreiranje predefiniranog Tinkerpop grafa potrebno je u Gremlin konzoli izvršiti sljedeću funkciju:

```
gremlin> g = TinkerGraphFactory.createTinkerGraph()
```

Operacije/funkcije u Gremlin upitnom jeziku su kategorizirane u sljedeće skupine:

- Funkcije za transformiranje
- Funkcije za filtriranje
- Funkcije za grananje
- Funkcije za sporedni učinak (eng. *sideffect*)
- Metode
- Recepti

5.2.6.1. Funkcije za transformiranje

Funkcije za transformiranje uzimaju zadani objekt u koraku obilaska, izvršavaju neki oblik transformacije nad njim te sljedećim koracima u obilasku vraćaju transformirani oblik objekta. Neke od najčešće korištenih funkcija ove skupine su:

- **_**
    - Funkcija identiteta koja transformira proizvoljni objekt u „cjevovod“ (eng. *pipeline*) koji predstavlja putanju obilaska grafa
- **both**
    - Funkcija koja dohvaća susjedne vrhove promatranog vrha do kojih se dolazi putem ulaznih i izlaznih veza
- **E**
    - Iterator veza u grafu koji se koristi za prolaženje kroz sve postojeće veze u grafu

- Vraća listu objekata veza koji sadrže informacije u identifikatoru početnog i završnog čvora i oznake veze

- **V**
  - Iterator čvorova u grafu koji se koristi za prolaženje kroz sve čvorove u grafu
  - Vraća listu objekata čvorova
- **id**
  - Dohvaća identifikator promatranog elementa grafa
- **in**
  - Dohvaća susjedne čvorove promatranog čvora
  - Specifičnija funkcija inE dohvaća ulazne veze promatranog čvora
  - Specifičnija funkcija inV dohvaća završni čvor promatrane veze prema kojem je ona usmjerena
- **out**
  - Dohvaća susjedne čvorove promatranog čvorova prema kojima on ima izlazne veze
  - Specifičnija funkcija outE dohvaća izlazne veze promatranog čvora
  - Specifičnija funkcija outV dohvaća početni (ishodišni) čvor promatrane veze
- **order**
  - Sortira elemente niza (objekte) prema zadanom ili predefiniranom kriteriju sortiranja

5.2.6.2. Funkcije za filtriranje

Funkcije za filtriranje služe za donošenje odluke o tome smije li se promatrani objekt proslijediti sljedećem koraku u putanji ili ne, ovisno o tome ispunjava li zadani kriterij. Ove funkcije omogućuju pronalaženje traženih podataka u graf bazi podataka na temelju zadanih kriterija. Pritom je podatke moguće pretraživati putem njihovih ključeva (eng. *key index lookup*) ili korištenjem funkcije *has*.

Najčešće korištene funkcije ove skupine su:

- **[i]**
  - Funkcija filtriranja koja dohvaća objekt sa zadanim indeksom (na zadanoj poziciji)
  - Moguće je pomoću funkcije [i..j] definirati interval indeksa objekata koje se želi dohvatiti

- **and**
    - Na temelju skupa cjevovoda vraća samo one objekte koji s obzirom na zadane kriterije vrijede u svim cjevovodima

- **filter**
    - Donosi odluku o tome treba li objekt proslijediti sljedećoj funkciji u putanji, ovisno o tome zadovoljava li zadani kriterij
- **has**
    - Prosljeđuje objekt sljedećoj funkciji ukoliko on zadovoljava kriterij funkcije
- **or**
    - Na temelju skupa cjevovoda vraća sve objekte koji zadovoljavaju zadane kriterije u barem jednom od svih cjevovoda

5.2.6.3. Funkcije za grananje

Ove se funkcije koriste za donošenje odluke o sljedećem koraku obilaska grafa. Predstavnici ove skupine funkcija su:

- **ifThenElse**
    - Funkcija koja sadrži logiku selekcije (odabira)
- **loop**
    - Omogućuje izvršavanje petlje nad odabranim koracima u cjevovodu

5.2.6.4. Funkcije za sporedni učinak

Funkcije sporednog učinka prosljeđuju zaprimljeni ulazni objekt sljedećem koraku u cjevovodu, ali pritom stvaraju određeni oblik sporednog učinka.

Najčešće korištene funkcije ove skupine su:

- **as**
    - Vraća ulazni objekt, ali pritom imenuje prethodni korak u cjevovodu
- **groupBy**
    - Vraća ulazni objekt, ali ga prethodno grupira s obzirom na zadani ključ i vrijednost
    - Ekvivalent SQL naredbi GROUP BY

- **groupCount**
    - Vraća ulazni objekt, ali prethodno ažurira mapu koja sadrži informacije o broju pojavnosti objekata u grafu
    - Ekvivalent SQL naredbi COUNT

5.2.6.5. Metode

Metode označavaju funkcije koje omogućuju brže izvođenje Gremlin upita kod pozivanja vanjskih programskih sučelja.

Najčešće korištene metode u Gremlin upitima su:

- **Element.keys / Element.values**
    - Metoda dohvaća sve ključeve, odnosno vrijednosti elementa u grafu, tj. objekta u cjevododu
- **Element.remove**
    - Briše element iz grafa
- **Graph.addEdge/ Graph.addVertex**
    - Dodaje vezu (potrebno definirati identifikator veze, početni i završni čvor te oznaku veze), odnosno čvor u graf (potrebno definirati identifikator čvora)
- **Graph.e / Graph.v**
    - Dohvaća vezu ili skup veza, odnosno čvor ili skup čvorova definiranjem željenog identifikatora
- **Graph.removeEdge / Graph.removeVertex**
    - Briše vezu, odnosno čvor, iz grafa

5.2.6.6. Recepti

Recepti označavaju često korištene uzorke u definiraju Gremlin upita. Omogućuju ponovno korištenje dijelova Gremlin upita te time olakšavaju korištenje Gremlin upitnog jezika.

Neki od češće korištenih recepata su:

- **Duplicate Edges**
    - Iako nije moguće da u bazi podataka postoje veze s istim identifikatorom, ovaj recept osigurava da ne postoje veze s istim početnim i završnim čvorovima te oznakama

- **Finding Edges Between Vertices**
    - Recept koji se koristi za utvrđivanje postoje li veze između promatranih čvorova grafa
- **Shortest Path**
    - Recept koji se koristi za određivanje najkraćeg puta između dva čvora u grafu, pritom koristeći odgovarajuće algoritme
- **Subgraphing**
    - Omogućuje izdvajanje dijela grafa u zasebni graf tako da dijelovi tog grafa budu sporedni učinci originalnog grafa

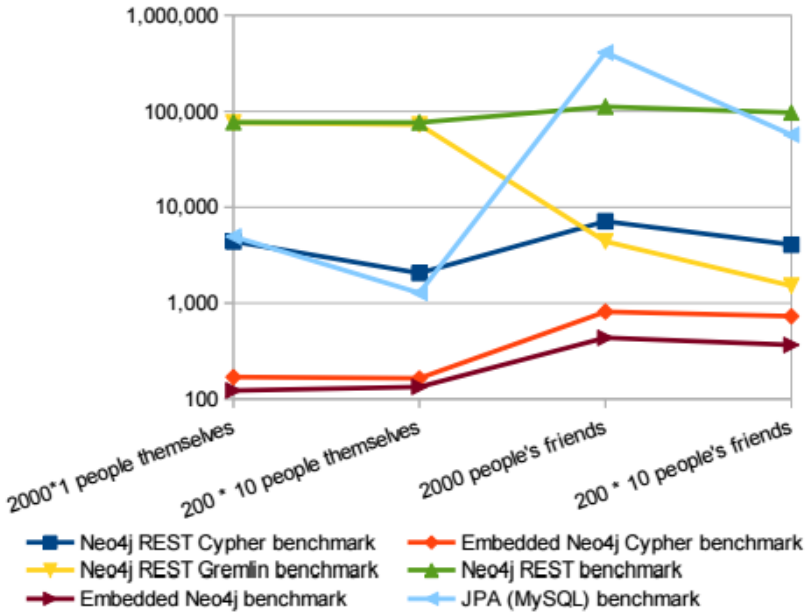
5.2.7. Nativni pristup

Nativni pristup dohvaćanju potrebnih podataka iz graf baze podataka omogućuje dohvaćanje traženih objekata, ali i svih zavisnih (podređenih) objekata (Holzschuher i Peinl, 2013). Ovakav pristup podrazumijeva dohvaćanje dodatnih, često nepotrebnih objekata u rezultatu upita, čime se nepotrebno dohvaćaju redundantni podaci. Međutim, nativni pristup ima određenih prednosti koje će biti spomenute i pojašnjene u sljedećem poglavlju.

Graf baze podataka mogu, a i ne moraju, definirati pristupne točke, tj. programska sučelja koja omogućuju izravan, nativni pristup podacima i njihovo dohvaćanje bez njihove obrade u međuvremenu.

5.2.8. Usporedba performansi upitnih jezika za relacijske i graf baze podataka

Holzschuher i Peinl su u svom članku proveli istraživanje u kojem su uspoređivali performanse različitih pristupa dohvaćanju potrebnih podataka iz relacijske i graf baze podataka. Kao predstavnik relacijskih baza podaka izabran je MySQL, a graf baze podataka je predstavljao Neo4j. Pritom je za pristup Neo4j bazi podataka korišteno nekoliko pristupa: putem Cypher i Gremlin upitnog jezika te nativni (eng. *embedded*) pristup. U bazi podataka nad kojom su se izvodili upiti pohranjene su informacije o osobama i njihovim prijateljstvima s drugim osobama.



Slika 18. Vrijeme izvođenja upita s različitim pristupima bazama podataka (ms) (Izvor: Holzschuher i Peinl, 2013)

Istraživanje je pokazalo da su, primjerice, nativni i Cypher pristup dohvaćanju osoba iz Neo4j baze podataka vrlo slični u svojoj brzini dohvaćanja iako je nativni pristup najbrži jer je neovisan o mreži, dok je Gremlin dosta sporiji u izvršavanju istog zadatka (slika 18). Međutim, brzina dohvaćanja podataka putem Gremlin upitnog jezika znatno se smanjuje u slučaju kompleksnijih upita, odnosno u dohvaćanju 10 prijatelja za 200 osoba te se malo razlikuje od brzine Cypher i nativnog pristupa. Budući da nativni pristup često dohvaća i suvišne podatke, za sprječavanje toga potrebno je, u usporedbi s veličinom Cypher i Gremlin upita, napisati veći upit koji će korisniku vratiti upravo ono što mu treba.

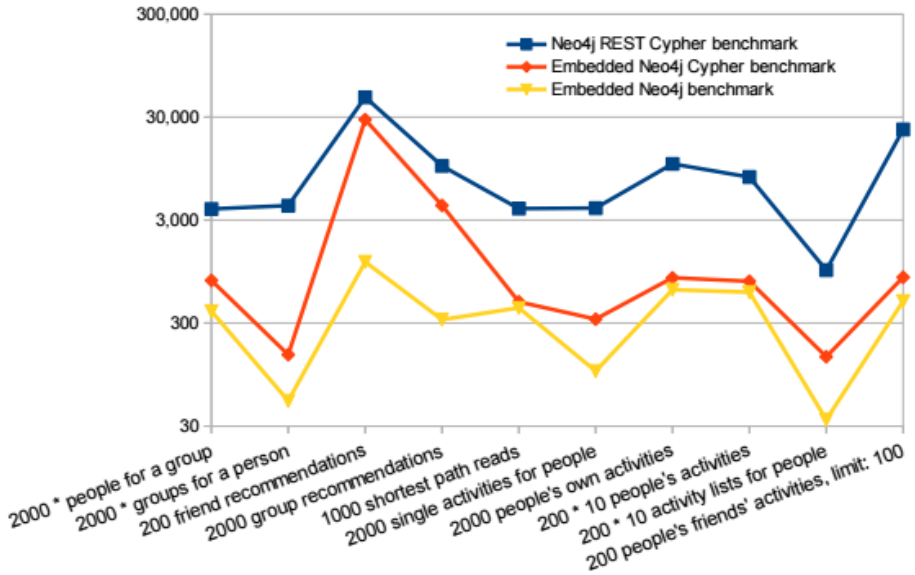
Međutim, iako su Cypher upiti manji i korisniku razumljiviji, njihov rezultat je često potrebno konvertirati u traženi oblik podataka.

Usporedbom rezultata upita za dohvaćanje prijatelja osoba za relacijske i graf baze podataka može se zaključiti kako su zbog već objašnjenih karakteristika graf baza podataka performanse takvog upita puno bolje u odnosu na relacijske baze podataka, bez obzira o kojem se pristupu radi. To posebice dolazi do izražaja kod povećanja veličine baze podataka.

S druge strane, Gremlin i Cypher upiti ostvaruju slične performanse kad se radi o običnom dohvaćanju podataka (dohvaćanje osoba iz baze podataka). Međutim, Gremlin upiti su se pokazali učinkovitijim u slučajevima kad je potreban obilazak grafa na više razina (dohvaćanje prijatelja osoba). Razlog tome je već spomenuta potreba za konverzijom

podataka dobivenih u rezultatu Cypher upita. Također, istraživanje je pokazalo da je za izvođenje kompleksnih upita s puno svojstava, čvorova i veza i pronalaženja uzoraka u grafu najbolje koristiti Cypher upite.

Kao što je očekivano, istraživanje je pokazalo da je nativni pristup puno brži u dohvaćanju podataka iz graf baze podataka u odnosu na Cypher i Gremlin pristup (slika 19), i to za 10 do čak u nekim slučajevima 200%. U nekim slučajevima, u kojima je čitljivost rezultata važnija od brzine njihova dohvaćanja, to može biti zanemariva činjenica pa se tada za dohvaćanje podataka koristi Cypher pristup.



Slika 19. Vrijeme dohvaćanja podataka za Cypher i nativni pristup (ms)
(Izvor: Honzschuher i Peinl, 2013)

6. Aplikacija za rad s graf bazama podataka

Radi prikaza definiranja i izvršavanja upita pomoću najčešće korištenih upitnih jezika Cypher i Gremlin napravljene su dvije aplikacije korištenjem različitih tehnologija.

Za izvršavanje Cypher upita napravljena je aplikacija pomoću sljedećih tehnologija:

- ASP.NET C# za kreiranje korisničkog sučelja i implementaciju aplikacijske logike, kao i definiranje WCF (eng. *Windows Communication Foundation*) servisa koji vraća informacije o oznakama čvorova i veza
- Neo4jClient biblioteka u Visual Studio razvojnoj okolini omogućuje jednostavan pristup Neo4j bazi podataka
- Neo4j baza podataka (verzija 2.0.4 *Community Edition*)

S druge strane, za izvršavanje Gremlin upita napravljena je aplikacija pomoću sljedećih tehnologija:

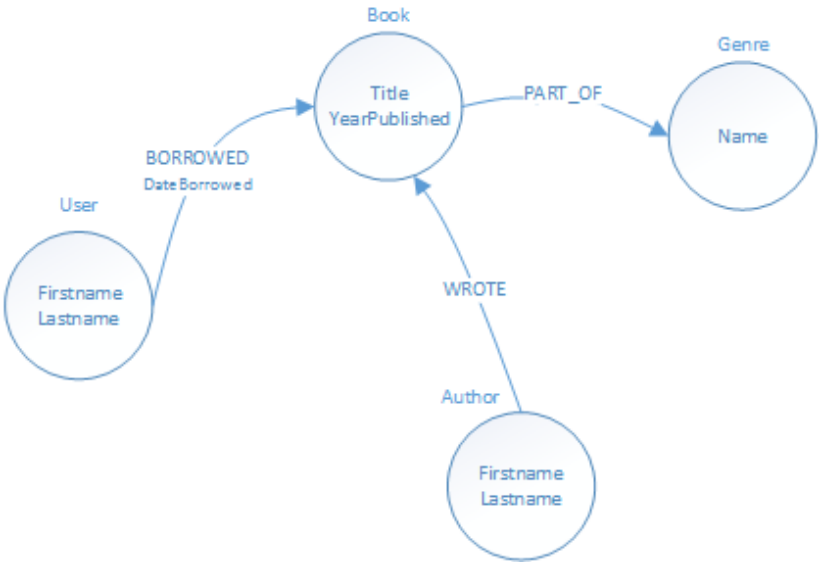
- Django web razvojni okvir za Python programski jezik za kreiranje korisničkog sučelja i implementaciju aplikacijske logike i definiranje programskog sučelja koje predstavlja servis s informacijama o oznakama čvorova i veza u bazi podataka
- Rexster baza podataka dostupna putem Rexster servera i Doghouse web sučelja
- Bulbs API biblioteka za Python za rad s graf bazama podataka

Za izvršavanje druge aplikacije potrebno je najprije pokrenuti Django i Rexster servere.

Aplikacije su originalno razvijane za potrebe projekta „Novi jezik za graf baze podataka“ koji je vodio izv. prof. dr. sc. Kornelije Rabuzin. Na projektu sam imala prilike sudjelovati u fazi implementacije predloženih rješenja, te sam temeljem toga i napisala ovaj diplomski rad.

6.1. Model podataka aplikacije

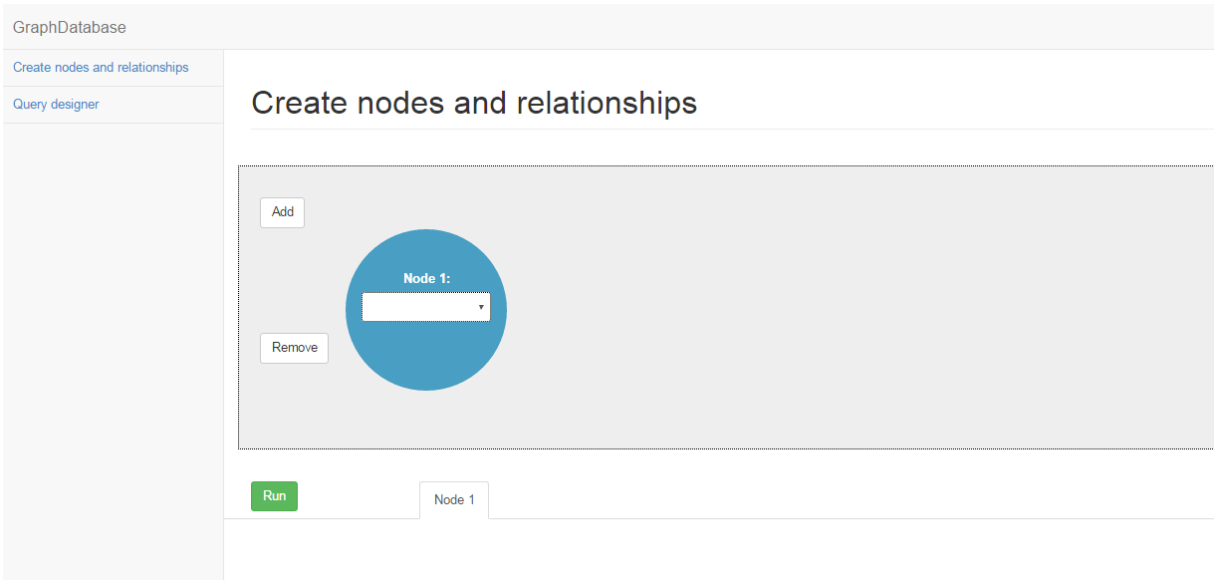
Bazu podataka čine čvorovi s 4 moguće oznake (Author, Book, Genre, User) i svojstvima te 3 tipa veza s oznakama BORROWED, PART_OF i WROTE (slika 20). Na konceptualnoj razini autor je napisao knjigu koja je dio određenog žanra, a koju može na određeni datum posuditi određeni korisnik.



Slika 20. Model podataka aplikacije (Izvor: vlastita izrada)

Svaka od aplikacija omogućuje dodavanje autora, knjiga, žanrova i korisnika te njihovo povezivanje. U kontekstu grafova, funkcionalnosti aplikacija su sljedeće:

- Kreiranje jednog čvora
- Kreiranje veze između dva čvora
- Kreiranje dvije veze između tri čvora
- Pretraživanje jednog čvora
- Pretraživanje veze između dva čvora
- Pretraživanje dviju veza između tri čvora



Slika 21. Početno sučelje za dodavanje novih čvorova i veza (Izvor: vlastita izrada)

6.2. Kreiranje jednog čvora

Za kreiranje jednog čvora, primjerice korisnika, izvršava se metoda *CreateUser* u prvoj aplikaciji, odnosno *create_user* u drugoj aplikaciji, kojoj se kao argument prosljeđuje objekt korisnika s njegovim svojstvima.

6.2.1. Implementacija za izvršavanje Cypher upita

Metoda *CreateUser()* izgleda ovako:

```
public User CreateUser(User user)
{
    var result = Neo4jDb.Instance.Client.Cypher.Merge("(n:User {Firstname:'" +
user.Firstname + "', Lastname:'" + user.Lastname + "'})")
             .Return(n => n.As<User>()).Results.FirstOrDefault();
    return result;
}
```

Primjerice, za dodavanje korisnika Ivana Horvata, ova će metoda izvršiti sljedeći Cypher upit:

```
MERGE (b: Book {Firstname: 'Ivan', Lastname: 'Horvat'}) RETURN b
```

Nakon izvršenja upita u bazu podataka bit će dodan novi čvor s oznakom *User* (slika 22).

Slika 23. Kreiran korisnik u bazi podataka (Izvor: vlastita izrada)

6.2.2. Implementacija za izvođenje Gremlin upita

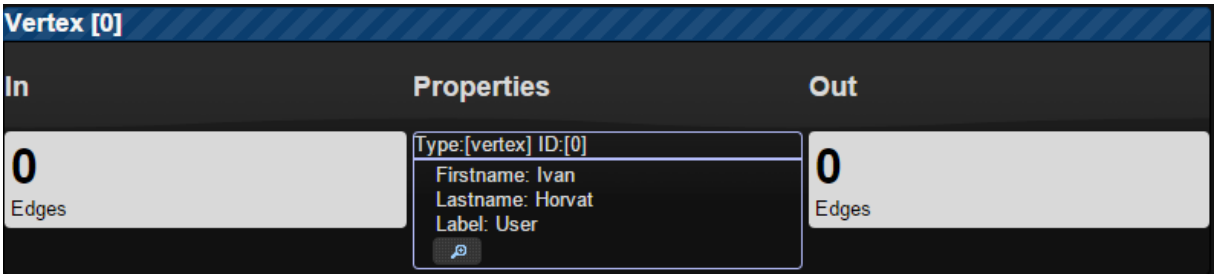
Metoda *create_user* izgleda ovako:

```
@staticmethod

def create_user(self, user):
    client = RexsterClient()
    script = client.scripts.get("get_vertices")
    self.gremlin = client.gremlin(script, params=None)
    result = self.gremlin.results
    if result:
        for v in result:
            if v:
                if v.get("Label") == "User":
                    if v.get("Firstname") == user.Firstname and v.get("Lastname") == user.Lastname:
                        return v
                else:
                    d = dict(Label=user.element_type, Firstname=user.Firstname, Lastname=user.Lastname)
                    u = client.create_vertex(data=d).results
                    result_node = u
                    return result_node
        d = dict(Label=user.element_type, Firstname=user.Firstname, Lastname=user.Lastname)
        u = client.create_vertex(data=d).results
        result_node = u
    else:
        d = dict(Label=user.element_type, Firstname=user.Firstname, Lastname=user.Lastname)
        u = client.create_vertex(data=d).results
        result_node = u
        return result_node
    return result_node
```

Ova će metoda pozivom *create_vertex* metode nad instancom *RexsterClient* klase izvršiti sljedeći Gremlin upit:

```
g.addVertex(null,[Firstname:  'Ivan', Lastname: 'Horvat', Label: 'User'])
```

Rezultat izvršenog Gremlin upita je novi korisnik u Rexster bazi podataka:



Slika 23. Kreirani čvor korisnika u Rexster bazi podataka (Izvor: vlastita izrada)

6.3. Kreiranje veze između dva čvora

Za kreiranje dva čvora i veze između njih, primjerice veze tipa WROTE između čvorova *Author* i *Book*, izvršava se metoda *ConnectBookToAuthor* s argumentima objekta autora i knjige u prvoj aplikaciji, odnosno *ConnectAuthorToBook* u drugoj aplikaciji, kojoj se kao argument prosljeđuju objekti autora, knjige te veze tipa *wrote* s njegovim svojstvima.

U upitu se izvršava provjera postojanja traženih čvorova u bazi podataka te se oni, ukoliko ne postoje, kreiraju zajedno s vezom između njih.

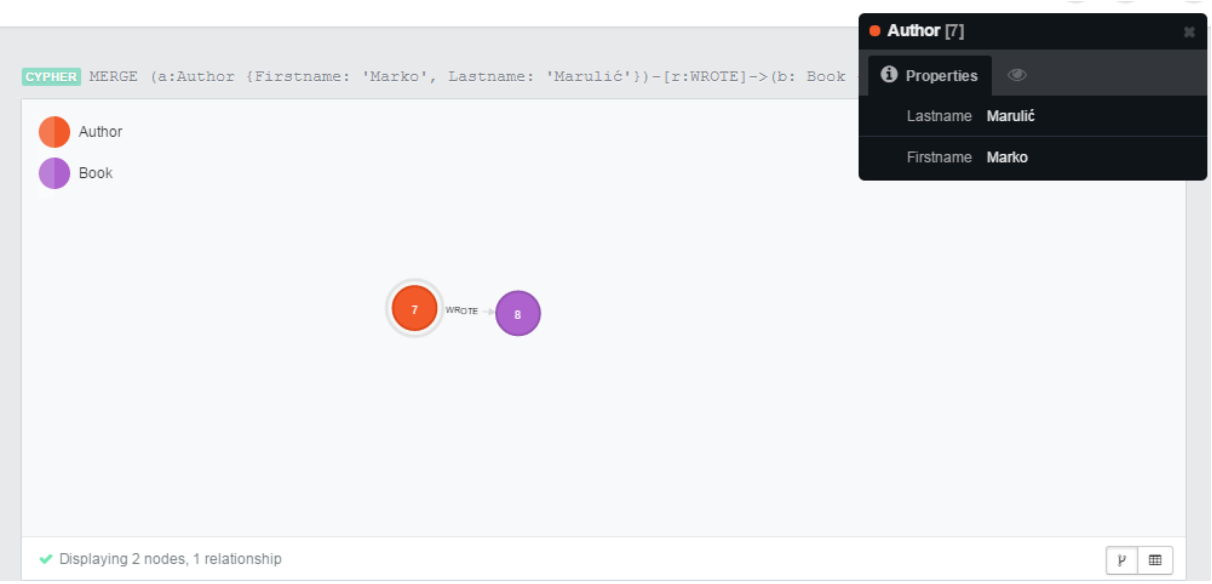
6.3.1. Implementacija za izvođenje Cypher upita

Metoda *ConnectBookToAuthor()* izgleda ovako:

```
public void ConnectBookToAuthor(Book book, Author author)
{
    Neo4jDb.Instance.Client.Cypher.Match("(b:Book)", "(a:Author)")
         .Where((Book b) => b.Title == book.Title)
         .AndWhere((Book b) => b.YearPublished == book.YearPublished)
         .AndWhere((Author a) => a.Firstname == author.Firstname)
         .AndWhere((Author a) => a.Lastname == author.Lastname)
         .CreateUnique("a-[:WROTE]->b")
         .ExecuteWithoutResults();
}
```

Primjerice, za kreiranje veze tipa *WROTE* između autora Marka Marulića i knjige Judita, ova će metoda izvršiti sljedeći Cypher upit:Nakon izvršenja upita u bazi podataka bit će kreirana nova veza tipa WROTE između Marka Marulića i Judite (slika 24).

```
MERGE (a:Author {Firstname: 'Marko', Lastname: 'Marulić'})-[r:WROTE]->(b: Book {Title:
'Judita', YearPublished: '1521'}) RETURN r
```



Slika 25. Kreirana veza između dva čvora (Izvor: vlastita izrada)

6.3.2. Implementacija za izvođenje Gremlin upita

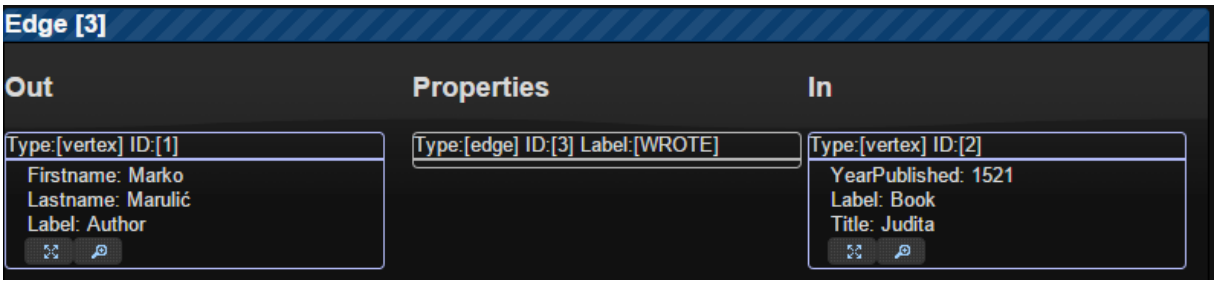
Metoda *ConnectAuthorToBook* najprije poziva metode za kreiranje čvorova, a zatim kreira veze između njih:

```
def ConnectAuthorToBook(self, author, book, wrote):
    g = GraphCon.g
    a = MainRepository.MainRepository.create_author(g, author)
    b = MainRepository.MainRepository.create_book(g, book)

    client = RexsterClient()
    script = client.scripts.get("get_vertices")
    response = client.gremlin(script, params=None)
    result = response.results

    if result:
        for v in result:
            if v.get('Label') == 'Author':
                if v.get('Firstname') == a.get('Firstname') and v.get('Lastname') == a.get('Lastname'):
                    outVert = v.get('_id')
            elif v.get('Label') == 'Book':
                if v.get('Title') == b.get('Title') and v.get('YearPublished') == b.get('YearPublished'):
                    inVert = v.get('_id')
    if outVert and inVert:
        response = client.create_edge(outV=outVert,label=wrote,inV=inVert,data={}).results
        return response

    else:
        return "NOT FOUND"
```

Ova će metoda pozivom *create_edge* metode nad instancom *RexsterClient* klase izvršiti sljedeće Gremlin upite:

```
v1 = g.V('Label', 'Author').next()
v2 = g.V('Label', Book).next()
g.addEdge(v1, v2, 'WROTE')
```

Rezultat izvršenog Gremlin upita je nova veza između autora i knjige u Rexster bazi podataka (slika 25).



Slika 5. Kreirana veza između dva čvora (Izvor: vlastita izrada)

6.4. Kreiranje dviju veza između tri čvora

Za kreiranje dviju veza i čvorova koje one spajaju, primjerice veze tipa BORROWED između čvorova *User* i *Book* te veze tipa *PART_OF* između čvorova *Book* i *Genre*, izvršava se metoda *ConnectUserBookGenre* s argumentima objekta korisnika, knjige i posudbe u prvoj aplikaciji, odnosno *ConnectUserToBookToGenre* u drugoj aplikaciji, kojoj se kao argument prosljeđuju objekti korisnika, knjige te veze tipa *borrowed* s njihovim svojstvima.

U upitu se izvršava provjera postojanja traženih čvorova u bazi podataka te se oni, ukoliko ne postoje, kreiraju zajedno s vezom između njih.

6.4.1. Implementacija za izvođenje Cypher upita

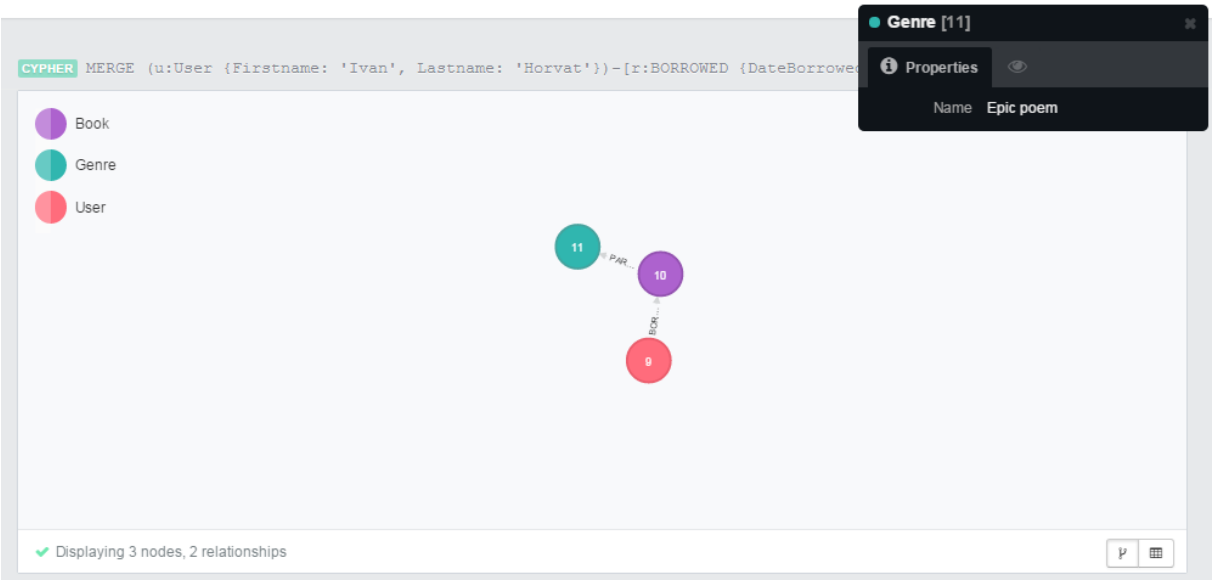
Metoda *ConnectUserBookGenre()* izgleda ovako:

```
public void ConnectUserBookGenre(User user, Book book, Genre genre, string dateBorrowed)
{
       Neo4jDb.Instance.Client.Cypher.Match("(u:User), (b:Book), (g:Genre)")
                 .Where((Book b) => b.Title == book.Title)
                 .AndWhere((Book b) => b.YearPublished == book.YearPublished)
                 .AndWhere((User u) => u.Firstname == user.Firstname)
                 .AndWhere((User u) => u.Lastname == user.Lastname)
                 .AndWhere((Genre g) => g.Name == genre.Name)
                 .CreateUnique("(u)-[:BORROWED {DateBorrowed:'" + dateBorrowed + "'}]-
>(b)-[:PART_OF]->(g)")
                 .ExecuteWithoutResults();
}
```

Primjerice, za kreiranje veze tipa *BORROWED* između korisnika Ivana Horvata i knjige Judita te veze tipa *PART_OF* između knjige Judita i žanra epska poema, ova će metoda izvršiti sljedeći Cypher upit:

```
MERGE (u:User {Firstname: 'Ivan', Lastname: 'Horvat'})-[r:BORROWED {DateBorrowed:
'12.4.2015.'}]->(b: Book {Title: 'Judita', YearPublished: '1521'})-[p:PART_OF]->(g:Genre
{Name: 'Epic poem'}) RETURN r, p
```

Nakon izvršenja upita u bazi podataka bit će kreirane nove veze tipa BORROWED između korisnika Ivana Horvata i knjige Judita te veza tipa PART_OF između te knjige i žanra Epska poema (slika 26).



Slika 26. Kreirane dvije veze između tri čvora (Izvor: vlastita izrada)

6.4.2. Implementacija za izvođenje Gremlin upita

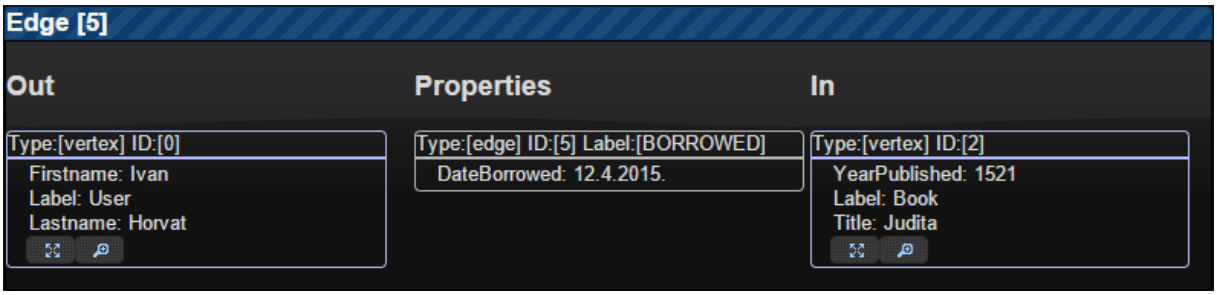
Metoda *ConnectUserToBookToGenre* najprije poziva metode za kreiranje čvorova, a zatim kreira veze:

```
def ConnectUserToBookToGenre(self, user, book, borrowed, genre, part_of, args):
    g = GraphCon.g
    u = MainRepository.MainRepository.create_user(g, user)
    b = MainRepository.MainRepository.create_book(g, book)
    genre = MainRepository.MainRepository.create_genre(g, genre)

    client = RexsterClient()
    script = client.scripts.get("get_vertices")
    response = client.gremlin(script, params=None)
    result = response.results

    if result:
        for v in result:
            if v.get('Label') == 'User':
                if v.get('Firstname') == u.get('Firstname') and v.get('Lastname') == u.get('Lastname'):
                    outVert = v.get('_id')
            elif v.get('Label') == 'Book':
                if v.get('Title') == b.get('Title') and v.get('YearPublished') == b.get('YearPublished'):
                    midVert = v.get('_id')
            elif v.get('Label') == 'Genre':
                if v.get('Name') == genre.get('Name'):
                    inVert = v.get('_id')
    if outVert and midVert and inVert:
        response1 = client.create_edge(outV=outVert,label=borrowed,inV=midVert,data=args).results
        response2 = client.create_edge(outV=midVert,label=part_of,inV=inVert,data={}).results
        dictResponse = {'response1':response1, 'response2':response2}
        return dictResponse
    else:
        return "NOT FOUND"
```

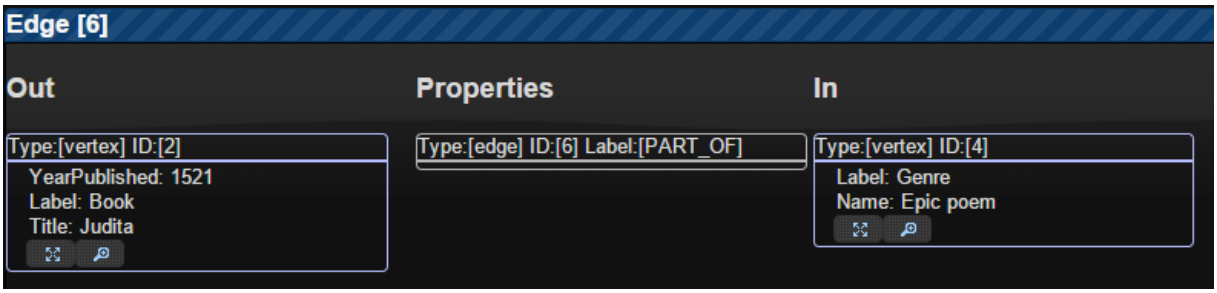
Ova će metoda dvostrukim pozivom *create_edge* metode nad instancom *RexsterClient* klase izvršiti sljedeće Gremlin upite:

```
v1 = g.V('Label', User).next()
v2 = g.V('Label', Book).next()
v3 = g.V('Label', 'Genre').next()
g.addEdge(v1, v2, 'BORROWED', [DateBorrowed : '12.4.2015.'])
g.addEdge(v2, v3, 'PART_OF')
```

Rezultat izvršenih Gremlin upita su dvije nove veze između korisnika i knjige, odnosno između knjige i žanra, u Rexster bazi podataka (slike 27 i 28).



Slika 27. Kreirana veza između čvorova korisnika i knjige (Izvor: vlastita izrada)



Slika 27. Kreirana veza između čvorova knjige i žanra (Izvor: vlastita izrada)

6.5. Pretraživanje jednog čvora

Pretraživanje podataka u korištenim graf bazama podataka moguće je uz definiranje 0, 1 ili 2 kriterija, sortiranje dobivenih rezultata, opcionalni prikaz pojedinih svojstava čvorova ili veza te ograničavanje broja redova u rezultatu. Primjerice, za dohvat 4 korisnika uzlazno sortiranih prema imenu iz baze podataka bez postavljenih kriterija potrebno je izvršiti metodu *SearchUser*() u prvoj aplikaciji, odnosno *search_one_node(čvor, parametri upita)* u drugoj aplikaciji.

6.5.1. Implementacija za izvođenje Cypher upita

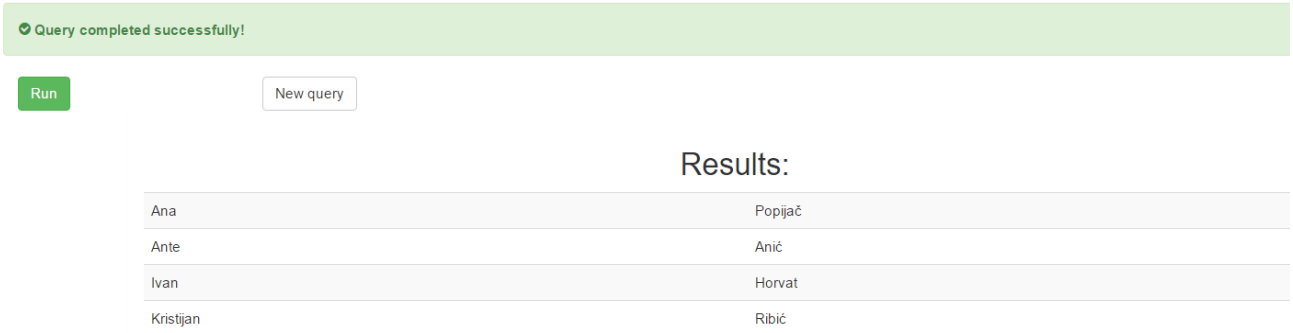
Metoda *SearchUser(<parametri upita>)* izgleda ovako:

```
public void SearchUser(<parametri>)
{
	IEnumerable<User>result = Neo4jDb.Instance.Client.Cypher.Match("(u:User)")
				.Return(u => u.As<User>())
				.OrderBy("u.Firstname")
				.Limit(Convert.ToInt32(limit))
				.Results.ToList<User>();
}
```

Cypher upit koji se pritom izvršava glasi:

```
MATCH (u:User) RETURN u ORDER BY u.Firstname LIMIT 4
```

Kao rezultat izvršenog upita dobiva se uzlazno sortirana lista korisnika (slika 29).



Slika 28. Rezultati upita za dohvaćanje 4 korisnika sortirana uzlazno prema imenu (Izvor: vlastita izrada)

6.5.2. Implementacija za izvođenje Gremlin upita

Izvršavanje upita u metodi *search_one_node*() implementirano je na sljedeći način:

```
result = client.gremlin("g.V('Label' , '" + node1 + "').order{it.a." +
prop1 + " <=> it.b." + prop1 + "}._()[0.." + str(limit-1) + "]").content
return result
```

Upit koji se u pozadini izvršava u upitnom jeziku Gremlin glasi:

```
g.V('Label', 'User').order{it.a.Firstname <=> it.b.Firstname}._()[0..3]
```

Rezultat upita je sortirana lista korisnika prikazana na slici 29.

6.6. Pretraživanje veze između dva čvora

Pretraživanje veze između dva čvora također se može napraviti definiranjem kriterija pretraživanja, sortiranjem rezultata prema svojstvima, opcionalnim prikazom svojstava te ograničavanjem redova u rezultatu. Primjerice, za dohvat 5 korisnika koji se zovu Ivan ili Ana uzlazno sortiranih prema imenu koji su posudili knjigu iz baze potrebno je izvršiti metodu *SearchUserBook*() u prvoj aplikaciji, odnosno *search_two_nodes(čvor1, čvor2, veza, parametri upita)* u drugoj aplikaciji.

6.6.1. Implementacija za izvođenje Cypher upita

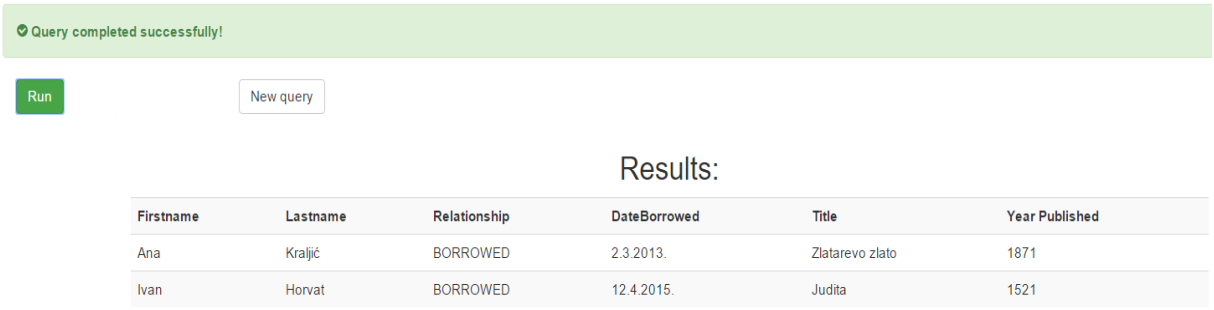
Metoda *SearchUserBook(<parametri upita>)* izgleda ovako:

```
public void SearchUserBook(<parametri>)
{
      IEnumerable<User> result = Neo4jDb.Instance.Client
                              .Cypher.Match("(user:User)-[r:BORROWED]->(book:Book)")
                              .Where((User user) => user.Firstname == criteria1Prop1Node1)
                              .OrWhere((User user) => user.Firstname == criteria2Prop1Node1)
                              .Return((user, book, r) => new { User = user.As<User>(), Book
= book.As<Book>(), R = r.As<BORROWED>() })
                              .OrderBy("user.Firstname")
                              .Limit(Convert.ToInt32(limit))
                                    .Results.ToList<Object>();
}
```

Cypher upit koji se pritom izvršava glasi:

```
MATCH (u:User)-[r:BORROWED]->(b:Book) WHERE u.Firstname='Ivan' OR u.Firstname='Ana'
RETURN u ORDER BY u.Firstname LIMIT 5
```

Kao rezultat izvršenog upita dobiva se uzlazno sortirana lista korisnika koji se zovu Ivan ili Ana (slika 30).



Slika 29. Rezultat upita za dohvaćanje 5 korisnika imena Ana ili Ivan (Izvor: vlastita izrada)

6.6.2. Implementacija za izvođenje Gremlin upita

Izvršavanje upita u metodi *search_two_nodes*() implementirano je na sljedeći način:

```
result = client.gremlin("g.V('Label', '" + node1 + "').has('" +
prop1Node1 + "', '" + criteria1Prop1Node1 + "').order{it.a." + prop1Node1
+ " <=> it.b." + prop1Node1 + "}copySplit(_().outE.inV.has('Label', '" +
node2 + "')).fairMerge.path._()[0.." + str(limit-1) + "]").content

return result
```

Funkcija *copySplit* se koristi radi za dohvaćanje liste objekata različitih tipova, a *T.in* se koristi za postavljanje višestrukih kriterija.

Upit koji se u pozadini izvršava u upitnom jeziku Gremlin glasi:

```
g.V('Label', 'User').has('Firstname', T.in, ['Ana', 'Ivan']).order{it.a.Firstname <=>
it.b.Firstname}copySplit(_().outE.inV.has('Label', 'Book')).fairMerge.path._()[0..4]
```

Rezultat upita je sortirana lista korisnika koji se zovu Ana ili Ivan prikazana na slici 31.

6.7. Pretraživanje dvije veze i tri čvora

Pretraživanje dvije veze između tri čvora može se napraviti definiranjem kriterija pretraživanja, sortiranjem rezultata prema svojstvima, opcionalnim prikazom svojstava te ograničavanjem redova u rezultatu. Primjerice, za dohvat 3 korisnika uzlazno sortiranih prema imenu koji su posudili knjigu bez zadanih kriterija, a koju su napisali određeni autori iz baze podataka potrebno je izvršiti metodu *SearchUserBookAuthor*() u prvoj aplikaciji, odnosno *search_three_nodes(čvor1, veza1, čvor2, veza2, čvor3, parametri upita)* u drugoj aplikaciji.

6.7.1. Implementacija za izvođenje Cypher upita

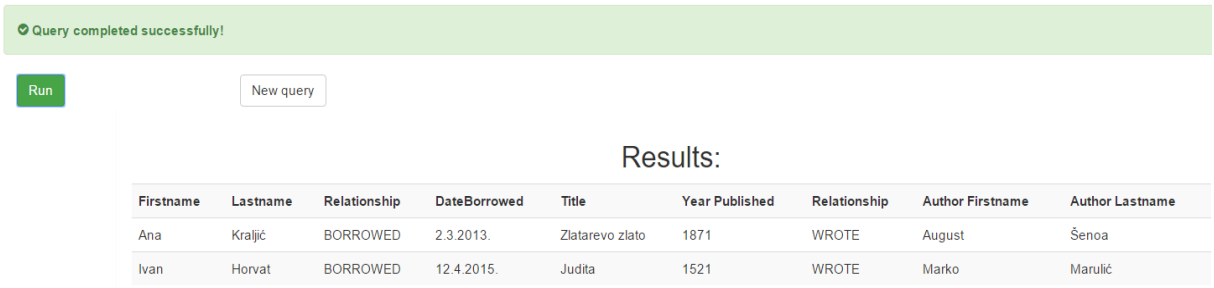
Metoda *SearchUserBookAuthor(<parametri upita>)* izgleda ovako:

```
public void SearchUserBook(<parametri>)
{
      IEnumerable<User> result = Neo4jDb.Instance.Client
                              .Cypher.Match("(user:User)-[r:BORROWED]->(book:Book)<-
[:WROTE]-(author:Author)")
                              .Return((user, book, r, author) => new { User =
user.As<User>(), Book = book.As<Book>(), R = r.As<BORROWED>(), Author =
author.As<Author>() })
                              .OrderBy("user.Firstname")
                              .Limit(Convert.ToInt32(limit))
                                    .Results.ToList<Object>();
}
```

Cypher upit koji se pritom izvršava glasi:

```
MATCH (u:User)-[r:BORROWED]->(b:Book)<-[t:WROTE]-(a:Author) RETURN r, t ORDER BY u.Firstname LIMIT 3
```

Kao rezultat izvršenog upita dobiva se uzlazno sortirana lista korisnika koji su posudili knjigu te autora tih knjiga (slika 31).



Slika 30. Rezultat upita za dohvaćanje korisnika koji su posudili knjigu i autora tih knjiga (Izvor: vlastita izrada)

6.7.2. Implementacija za izvođenje Gremlin upita

Izvršavanje upita u metodi *search_three_nodes*() implementirano je na sljedeći način:

```
result = client.gremlin("g.V('Label', '" + node1 + "').order{it.b." +
prop1Node1 + " <=> it.a." + prop1Node1 + "}copySplit(_().outE.inV.has('Label',
'" + node2 + "').inE.outV.has('Label', '" + node3
+"')).fairMerge.path._()[0.." + str(limit-1) + "]").content

return result
```

Upit koji se u pozadini izvršava u upitnom jeziku Gremlin glasi:

```
g.V('Label', 'User'). order{it.a.Firstname <=>it.b.Firstname}copySplit(().outE.inV.has
('Label', 'Book').inE.outV.has('Label', 'Author')).fairMerge.path._([0..2]
```

Rezultat upita je sortirana lista korisnika koji su posudili knjigu i autora knjiga prikazana na slici 31.

7. Zaključak

Graf baze podataka dobivaju sve veću pozornost korisnika ponajprije zahvaljujući svojim karakteristikama pojašnjenim u prethodnim poglavljima ovog rada. Njihovim korištenjem u različitim područjima moguće je pohraniti kompleksne podatke nad kojima se kasnije mogu izvršavati upiti na više razina koji će vraćati rezultate u konačnom vremenu, što za relacijske baze podataka nije uvijek slučaj. Njihovom usporedbom može se zaključiti kako je u slučaju povećanja složenosti i količine podataka u bazi podataka standardna devijacija vremena izvođenja upita nad tim podacima kod graf baza podataka puno manja u odnosu na relacijske baze podataka. Osim toga, često je pohrana podataka u obliku grafa puno prirodnija nego njihova pohrana u redove tablica jer je graf model podataka fleksibilniji.

Neki autori smatraju graf baze podataka relacijskim bazama podataka sljedeće generacije, budući da su uspjele preuzeti i proširiti neke prednosti relacijskih baza poput matematičke algebre i razumljivosti modela podataka, ali i popraviti i zamijeniti njihove uočene nedostatke.

U radu su prikazani osnovni koncepti grafova koji čine model podataka graf baza podataka, ali i pojmovi poput pronalaženja uzoraka u grafu i obilaska grafa koje je potrebno razlikovati. Detaljnije su pojašnjeni najčešće korišteni algoritmi pronalaženja uzoraka i obilaska grafa. Ti su algoritmi samo jedan od čimbenika koji utječu na efikasno izvođenje upita na graf bazama podataka.

Upiti nad graf bazama podataka mogu se izvoditi putem nekoliko upitnih jezika, od kojih su danas najzastupljeniji Cypher i Gremlin. Usporedbom njihovih performansi u radu je pokazano kako Cypher ostvaruje bolje performanse u dohvaćanju podataka, ali se ta razlika smanjuje s povećanjem složenosti podataka koje je potrebno dohvatiti jer ih Cypher nakon dohvaćanja mora konvertirati u traženi oblik.

Osim teorijskog, ovaj radi sadrži i praktični dio u kojem su napravljene dvije aplikacije za rad s Neo4j i Rexster bazom podataka te izvođenje Cypher, odnosno Gremlin, upitima nad njima. Svrha tih aplikacija bila je pokazati da je moguće napraviti jednostavno korisničko sučelje u kojem će korisnik dodavati i pretraživati podatke iz baze podataka, dok će se u pozadini izvršavati upiti u navedenim upitnim jezicima.

Na temelj analiziranih svojstava graf baza podataka može se zaključiti kako će njihova zastupljenost na tržištu i u budućnosti nastaviti svoj trend porasta zahvaljujući stalnom ulaganju u njhov razvoj, ali i povećanoj potrebi upravljanja složenim podacima.

8. Popis korištenih slika

9. Literatura

[1] Adell J. (2011) *Performance of Graph vs. Relational Databases* . Preuzeto 1.veljače 2016. s http://blog.everymansoftware.com/2011/07/performance-of-graph-vs-relational.html

[2] Angles R. (2012) *A Comparison of Current Graph Database Models* . Preuzeto 1.veljače 2016. s http://campuscurico.utalca.cl/~rangles/files/gdm2012.pdf

[3] Bain T. (2009) *Is the Relational Database Doomed?* . Preuzeto 1.veljače 2016. s http://readwrite.com/2009/02/12/is-the-relational-database-doomed

[4] Baranović M., Zakošek S. (2007) *Baze podataka* . Materijali za predavanja . Zagreb: Fakultet elektrotehnike i računarstva . Preuzeto 1. veljače 2016. s https://www.fer.unizg.hr/_download/repository/BazePodataka_SQLPredavanja.pdf

[5] Barcelo P. (2013) *Querying Graph Databases* . New York : ACM . Preuzeto 1.veljače 2016. s http://users.dcc.uchile.cl/~pbarcelo/pods001i-barcelo.pdf

[6] Codd E.F. (1970) *Relational Database: A Practical Foundation for Productivity*. Preuzeto 1. veljače 2016. s http://cgi.di.uoa.gr/~std06057/history/articles/turing_award_lecture.pdf

[7] Consens M.P., Hasan M.Z. (1993) *Supporting Network Management through Declaratively Specified Data Visualizations* . Zbornik radova međunarodne konferencije *IFIP/IEEE Third International Symposium on Integrated Network Management* . Toronto . Preuzeto 1.veljače 2016. s https://www.researchgate.net/profile/Masum_Hasan/publication/2444629_Supporting_Network_Management_through_Declaratively_Specified_Data_Visualizations/links/53e14b870cf2d79877a7591a.pdf

[8] Conte D., Foggia P., Sansone C., Vento M. (2004) *Thirty years of graph matching in pattern recognition* . International Journal of Pattern Recognition and Artificial Intelligence (3.izd): 265-298, World Scientific Publishing Company . Preuzeto 1.veljače 2016. s https://www.researchgate.net/profile/Mario_Vento/publication/220359794_Thirty_Years_Of_Graph_Matching_In_Pattern_Recognition/links/0c960516871e01e0b1000000.pdf

[9] Darwen H. (2012) *An Introduction to Relational Database Theory*. Peterlee, UK. Ventus Publishing ApS

[10] Elkstein M. (s.a.) *Learn REST: A Tutorial* . Preuzeto 1.veljače 2016. s http://rest.elkstein.org/

[11] He H. (2007) *Querying and Mining Graph Databases* . ProQuest Information and Learning Company, pp. 1-70

[12] He H., Singh A. (2008) *Graphs-at-a-time: Query Language and Access Methods for Graph Databases* . Vancouver: ACM . Preuzeto 1.veljače 2016. s http://sites.fas.harvard.edu/~cs265/papers/he-2008.pdf

[13] Heap D. (2002) *Breadth-first Search* . Preuzeto 1.veljače 2016. s http://www.cs.toronto.edu/~heap/270F02/node37.html

[14] Heap D. (2002) *Depth-first Search* . Preuzeto 1.veljače 2016. s http://www.cs.toronto.edu/~heap/270F02/node36.html

[15] Hellerstein J., Gonzalez, J. (2015) *Intro to Database Systems* . Preuzeto 1.veljače 2016. s http://inst.eecs.berkeley.edu/~cs186/fa06/lecs/08SQL1.pdf

[16] Holzschuher F., Peinl R. (2014) *Performance optimization for querying social network data* . Zbornik radova na 17.međunarodnoj konferenciji *EDBT/ICDT Joint Conference* Atena. Preuzeto 1.veljače 2016. s http://ceur-ws.org/Vol-1133/paper-38.pdf

[17] Klobučar A. (2012) *Osnove teorije grafova* . Materijali s predavanja . Osijek : Sveučilište J.J.Strossmayera . Preuzeto 1.veljače 2016. s http://www.mathos.unios.hr/kidm/pred10.pdf

[18] Kozielski S., Mrozek D., Kasprowski P., Malysiak-Mozek B., Kostrzewa D. (2015) *Beyond Databases, Architectures and Structures* . Zbornik radova 11. međunarodne konferencije *BDAS* Poljska

[19] Lee J., Han W., Kasperovics R., Lee J. (2013) *An In-depth Comparison of Subgraph Isomorphism Algorithms in Graph Databases* (2.izd) . Zbornik radova 39. međunarodne znanstvene konferencije *Very Large Databases* Trento : Italija . Preuzeto 1.veljače 2016. s http://www.vldb.org/pvldb/vol6/p133-han.pdf

[20] Levene M., Loizou G. (2012) *A Guided Tour of Relational Databases and Beyond* . Springer Science & Business Media

[21] Lorentz D., Gregoire J. (2003) *Oracle Database : SQL Reference* (10.izd) . Oracle Corporation . Preuzeto 1.veljače 2016. s https://docs.oracle.com/cd/B12037_01/server.101/b10759.pdf

[22] Maier D. (1983) *The Theory of Relational Databases* . Rockwell : Computer Science Press Inc. . Preuzeto 1.veljače 2016. s http://web.cecs.pdx.edu/~maier/TheoryBook/MAIER/

[23] Maier D., Rorenshtein D., Salveter S., Stein J., Warren D. S. (1987) *PIQUE : a relational query language without relations* . CSETech. Paper 152 . Preuzeto 4.veljače 2016. s http://digitalcommons.ohsu.edu/cgi/viewcontent.cgi?article=1151&context=csetech

[24] Melton J., Simon A.R. (1997) *Understanding the new SQL: A Complete Guide* . San Francisco: Morgan Kaufmann Publishers Inc., pp. 3-11

[25] Merenyi R. (2013) *What Are the Differences Between Relational and Graph Databases?* . Preuzeto 1.veljače 2016. s http://www.seguetech.com/blog/2013/02/04/what-are-the-differences-between-relational-and-graph-databases

[26] Morgan J. (2014) *A Simple Explanation of 'The Internet of Things'* . Preuzeto 1.veljače 2016. s http://www.forbes.com/sites/jacobmorgan/2014/05/13/simple-explanation-internet-things-that-anyone-can-understand/#48a7f20a6828

[27] Morris J. (1998) *Dijkstra's Algorithm* . University of Auckland . Preuzeto 1.veljače 2016. s https://www.cs.auckland.ac.nz/software/AlgAnim/dijkstra.html

[28] Myers A. (2012) : *Graph traversals* . Cornell University . Preuzeto 1.veljače 2016. s http://www.cs.cornell.edu/courses/cs2112/2012sp/lectures/lec24/lec24-12sp.html

[29] Neubauer P. (2010) *Graph Databases, NOSQL and Neo4j* . Preuzeto 1.veljače 2016. s http://www.infoq.com/articles/graph-nosql-neo4j

[30] Neumann A. (s.a.) *Ullman's Subgraph Isomorphism Algorithm* . Preuzeto 1.veljače 2016. s https://adriann.github.io/Ullman%20subgraph%20isomorphism.html

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

[32] Rabuzin K. (2014) *SQL – napredne teme* . Varaždin: Fakultet organizacije i informatike

[33] Raj S. (2015) *Neo4j High Performance* . Mumbai: Packt Publishing

[34] Ramakrishnan R. i Gehrke J. (2002) *Database Management Systems* (3.izd) . Preuzeto 4. veljače 2016. s http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/qbe.pdf

[35] Reeve A. (2012) *Big Data and NoSQL: The Problem with Relational Databases* . Preuzeto 1.veljače 2016. s https://infocus.emc.com/april_reeve1/big-data-and-nosql-the-problem-with-relational-databases/

[36] Robinson I., Webber J., Eifrem E. (2015) *Graph Databases* (2.izd) . Neo Technology Inc. Sebastopol: O'Reilly Media Inc.

[37] Rodriguez M.A. (2015) *The Benefits of the Gremlin Graph Traversal Machine*. Preuzeto 1.veljače 2016. s http://www.datastax.com/dev/blog/the-benefits-of-the-gremlin-graph-traversal-machine

[38] Sasaki B.M. (2015) *ACID vs. BASE Explained* . Preuzeto 1.veljače 2016. s http://neo4j.com/blog/acid-vs-base-consistency-models-explained/

[39] s.n. – Gremlin Docs (s.a.) . Preuzeto 1.veljače 2016. s http://gremlindocs.spmallette.documentup.com/

[40] s.n. – More Process (2013) *Pros, advantages of SQL* . Preuzeto 1.veljače 2016. s http://www.moreprocess.com/database/sql/pros-advantages-of-sql-structured-query-language

[41] s.n. – Neo4j (s.a.) *Relational Databases vs Graph Databases: A Comparison* . Preuzeto 1.veljače 2016. s http://neo4j.com/developer/graph-db-vs-rdbms/

[42] s.n. – Neo4j Manual (s.a.) *The Neo4j Graph Database* . Preuzeto 1.veljače 2016. s http://neo4j.com/docs/stable/graphdb-neo4j.html

[43] s.n. – Titan Documentation (s.a.) *Gremlin Query Language* . Preuzeto 1.veljače 2016. s http://s3.thinkaurelius.com/docs/titan/0.5.0/gremlin.html

[44] Valiente G., Martinez C. (1997) *An algorithm for grapht pattern-matching* . Preuzeto 1.veljače 2016. s http://www.cs.upc.edu/~valiente/abs-wsp-1997.pdf

[45] Vardi M.Y. (1982) *The Complexity of Relational Query Languages* . Department of Computer Science, Stanford Unversity . Preuzeto 1.veljače 2016. s http://oem.stanford.edu/publications/vardi/p137-vardi.pdf

[46] Vogl W. (2011) *Graph Matching Algorithm* . Materijal s predavanja, Beč: University Of Technology . Preuzeto 1.veljače 2016. s http://oldwww.prip.tuwien.ac.at/teaching/ss/strupr/vogl.pdf

[47] Watt A. (2013) *Database Design* . Open Textbook Project . Preuzeto 1.veljače 2016. s http://opentextbc.ca/dbdesign/

[48] Williams D.W., Huan J., Waing W. (2007) *Graph Database Indexing Using Structured Graph Decomposition* . Preuzeto 1.veljače 2016. s http://web.cs.ucla.edu/~weiwang/paper/ICDE07_1.pdf

[49] Wood P.T. (2012) *Query Languages for Graph Databases* (1.izd) . London: Department of Computer Science and Information Systems . Preuzeto 1.veljače 2016. s http://users.dcc.uchile.cl/~pbarcelo/wood.pdf