UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

MATEJ HLAD

PODATKOVNA BAZA GRAFOV NEO4J IN PRIMERJAVA UČINKOVITOSTI Z RELACIJSKO PODATKOVNO BAZO

DIPLOMSKO DELO

VISOKOŠOLSKI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Mentor: viš. pred. dr. Aljaž Zrnec

Ljubljana, 2013

Besedilo je oblikovano z urejevalnikom besedil Microsoft Word 2010.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00462/2013
Datum: 15.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEJ HLAD**

Naslov: **PODATKOVNA BAZA GRAFOV NEO4J IN PRIMERJAVA UČINKOVITOSTI Z RELACIJSKO PODATKOVNO BAZO**

GRAPH DATABASE NEO4J AND PERFORMANCE COMPARISON WITH RELATIONAL DATABASE

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Podatkovne baze ki temeljijo na grafih, predstavljajo eno izmed NoSQL tehnologij za obvladovanje podatkov. V diplomski nalogi predstavite podatkovno bazo, ki temelji na grafih - Neo4j. V okviru teoretičnega dela diplome predstavite podatkovni model, ki ga ta baza uporablja, podprte vmesnike do baze, poizvedovalni jezik za delo s podatki in primere uporabe take podatkovne baze. Izpostavite tudi dobre in slabe lastnosti baz, ki temeljijo na grafih. V okviru praktičnega dela izvedite performančno analizo, v kateri primerjate učinkovitost omenjene in klasične relacijske podatkovne baze.

Mentor:

viš. pred. dr. Aljaž Zrnec

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:

prof. dr. Nikolaj Zimic

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00462/2013
Datum: 15.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEJ HLAD**

Naslov: **PODATKOVNA BAZA GRAFOV NEO4J IN PRIMERJAVA UČINKOVITOSTI Z RELACIJSKO PODATKOVNO BAZO**

GRAPH DATABASE NEO4J AND PERFORMANCE COMPARISON WITH RELATIONAL DATABASE

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Podatkovne baze ki temeljijo na grafih, predstavljajo eno izmed NoSQL tehnologij za obvladovanje podatkov. V diplomski nalogi predstavite podatkovno bazo, ki temelji na grafih - Neo4j. V okviru teoretičnega dela diplome predstavite podatkovni model, ki ga ta baza uporablja, podprte vmesnike do baze, poizvedovalni jezik za delo s podatki in primere uporabe take podatkovne baze. Izpostavite tudi dobre in slabe lastnosti baz, ki temeljijo na grafih. V okviru praktičnega dela izvedite performančno analizo, v kateri primerjate učinkovitost omenjene in klasične relacijske podatkovne baze.

Mentor:

viš. pred. dr. Aljaž Zrnec

Dekan:

prof. dr. Nikolaj Zimic

Zahvala

Najlepše se zahvaljujem za pomoč, usmerjanje ter nasvete pri izdelavi diplomske naloge svojemu mentorju viš. pred. dr. Aljažu Zrnecu.

Velika zahvala gre tudi Tini Bajc za spodbujanje in motivacijo ter Ani in Tomotu za pomoč pri urejanju in lektoriranju dela.

Hvala tudi vsem, ki so mi stali ob strani med časom študija s svojimi vzpodbudami.

Izjava

Izjavljam, da sem diplomsko nalogo izdelal samostojno pod vodenjem mentorja viš. pred. dr. Aljaža Zrneca. Ostale, ki so kakorkoli pomagali pri nastajanju dela, sem navedel v zahvali.

Vrhpolje, Avgust 2013 Matej Hlad

Kazalo vsebine

Seznam slik

Seznam tabel

Seznam uporabljenih kratic

ACID	Atomicity, Consistency, Isolation, Durability (atomarnost, konsistentnost, izolacija, trajnost), lastnosti transakcij v podatkovni bazi
API	Application Programming Interface, vmesnik, ki specificira, kako naj bi programske komponente komunicirale med sabo
CSV	Comma-Separated Values, format podatkov, kjer so podatki ločeni z vejico (ali s kakim drugim znakom), ki se uporablja za izmenjavo ali pretvarjanje podatkov
DBMS	Database Management System, sistem za upravljanje s podatkovno bazo (sl. SUPB)
HTTP	HyperText Transfer Protocol, protokol za prenos informacij preko spleta
IDE	Integrated Development Environment, integrirano razvojno okolje, razvojno okolje (aplikacija), namenjen razvijanju programske opreme
JSON	JavaScript Object Notation, enostaven in preprost format za izmenjavo podatkov, ki temelji na podmnožici programskega jezika JavaScript
NoSQL	s to oznako označujemo podatkovne baze, ki uporabljajo manj stroge modele podatkov kot relacijske podatkovne baze
OLTP	Online Transaction procesing, spada v razred informacijskih sistemov, ki olajšajo in upravljajo transakcijsko orientirane aplikacije
SPARQL	SPARQL Protocol and RDF Query Language, povpraševalni jezik za delo z neralacijskimi podatkovnimi bazami
SNMP	Simple Network Management Protocol, protokol za upravljanje naprav v omrežjih
SQL	Structured Query Language, strukturirani povpraševalni jezik za delo z relacijskimi podatkovnimi bazami
URI	Uniform Resource Identifier, standardna sestavljena oznaka za lociranje internetnega vira z naslovnimi sistemi (URL, URN)
URL	Uniform Resource Locator, enotni lokator virov, je enoten način za poimenovanje dokumentov ali drugih virov na svetovnem spletu
URN	URN Uniform Resource name, enotno ime vira, označuje vir, ki je neodvisen od lokacije, protokola, gostitelja in omogoča trajen dostop do kontinuiranih virov
XML	eXtensible Markup Language, razširljiv označevalni jezik, za izmenjavo ali prenos strukturiranih podatkov

Povzetek

Zaradi vedno večje rasti kompleksno povezanih in dinamičnih podatkov, se vedno bolj uveljavljajo podatkovne baze, ki shranjujejo podatke v obliki grafov. Sprva so jih začeli uporabljati za namene socialnih omrežjih, sedaj pa se njihova uporaba vedno bolj širi. V diplomskem delu obravnavamo podatkovno bazo Neo4j, ki temelji na omenjenem načinu shranjevanja podatkov. Podatkovno bazo Neo4j predstavimo tako s teoretičnega, kot tudi s praktičnega vidika. V teoretičnem delu predstavimo splošne lastnosti o podatkovnih bazah, nakar se podrobneje spustimo v podatkovni model podatkovne baze grafov in opišemo njene značilnosti. Obravnavamo tudi poizvedovalni jezik Cypher, ki ga uporabljamo za pisanje poizvedb nad podatki v grafu Neo4j. V praktičnem delu podatkovno bazo Neo4j z uporabo testnih primerov primerjamo z relacijsko podatkovno bazo MySQL. V zaključku povzamemo ugotovitve o podatkovni bazi Neo4j.

Ključne besede:

podatkovna baza, graf, podatkovna baza grafov, podatki, poizvedba, vozlišče, relacija, lastnost

Abstract

Due to increasing growth of complex and dynamic linked data, there is becoming more and more databases that store data in graph form. First who used them were social networks. Their usage increasly spreads nowdays. In the work we treat database Neo4j, based on the above-mentioned method of storing data in graph. We introduce Neo4j database on both point of view–theoretical and the practical. In the theoretical part we introduce general properties about databases. Then we describe data model of graph databases and its features in detail. We treat also Cypher query language, which database use for querying over stored data. In practical part by using test cases we compare graph database Neo4j with relational database MySQL. In conclusion we summarize of observation over graph database Neo4j.

Keywords:

database, graph, graph database, data, query, node, relationship, property

1. UVOD

Namen diplomskega dela je predstaviti podatkovno bazo, ki temelji na grafih – Neo4j in jo z vidika učinkovitosti primerjati z relacijsko podatkovno bazo MySQL. V uvodnem delu diplomske naloge bomo v splošnem predstavili podatkovne baze, ki temeljijo na grafih. Predstavili bomo osnovne pojme o tem, kaj je graf in kaj podatkovna baza. Opisali bomo strukturo grafa in podatkovne baze, ki temelji na grafih ter se seznanili s prednostmi le te.

V osrednjem delu bomo spoznali eno izmed najbolj znanih odprtokodnih implementacij podatkovne baze grafov[1] Neo4j. Spoznali bomo vmesnike, ki omogočajo vpogled v podatke in izvajanje raznih operacij nad njimi ter samo strukturo podatkov v grafu. Podrobneje bomo predstavili orodje Neoclipse, ki se uporablja za delo s podatkovno bazo Neo4j in omogoča urejanje podatkov v grafu podatkovne baze ter grafični prikaz strukture grafa.

V nadaljevanju bomo predstavili tudi deklarativni poizvedovalni jezik Cypher, ki se uporablja v okviru podatkovne baze grafov. Preučili bomo način njegove uporabe v poizvedbah nad podatki v grafu. Spoznali bomo strukturo, povzeli ključne primere in naredili nekaj testnih poizvedb. Jezik Cypher bomo primerjali s poizvedovalnim jezikom relacijskih podatkovnih baz – SQL.

V zadnjem delu diplomskega dela bomo primerjali podatkovno bazo Neo4j z eno izmed bolj znanih relacijskih podatkovnih baz - MySQL. S pomočjo programskega jezika Java in orodja Eclipse bomo izdelali program, ki bo generiral podatke za uvoz v obe podatkovni bazi. S primerjanjem časov izvajanja poizvedb nad eno in drugo podatkovno bazo bomo spoznali razlike med njima.

V zaključku bomo predstavili sklepne ugotovitve do katerih smo prišli tekom diplomskega dela.

[1] Namesto termina podatkovna baza, ki temelji na grafih, bomo v diplomskem delu raje uporabljali izraz *podatkovna baza grafov*.

2. PODATKOVNE BAZE GRAFOV

2.1. UVOD

Relacijske podatkovne baze so zaradi svojih zmogljivosti in ACID lastnosti transkacij desetletja predstavljale tehnologijo za obvladovanje strukturiranih podatkov. S pojavom algoritmov za porazdeljeno procesiranje podatkov in z uporabo poceni strojne opreme se je odprla možnost za obdelavo velikih količin podatkov. Klasične relacijske baze se pri tem izkažejo slabše, zaradi česar so se pojavile nove tehnologije za obvladovanje velikih količin podatkov - različne vrste podatkovnih baz NoSQL. Ena takih vrst so tudi podatkovne baze grafov [17].

S pomočjo grafov si lažje predstavljamo povezanost podatkov in jih lažje analiziramo. To prednost grafov s pridom izkoriščajo podatkovne baze, ki temeljijo na shranjevanju podatkov v obliki grafa. Teorija grafov sama po sebi ni nič novega. Prvi jo je oblikoval Euler v 18. stoletju, od takrat pa jo aktivno raziskujejo in izboljšujejo matematiki, sociologi in antropologi. Ideja, da bi teorijo grafov uporabili kot temelj za shranjevanje podatkov, se je razvila že pred leti. Posebno pozornost grafom so posvečala komercialno uspešna podjetja kot so Facebook, Google in Twitter[6].

2.2. RELACIJSKA PODATKOVNA BAZA

V okviru tega razdelka predstavimo glavne lastnosti relacijskih podatkovnih baz, tako da bomo lahko kasneje lažje izvajali primerjavo.

Teorijo relacijskih podatkovnih baz je zasnoval Edgar F. Codd leta 1969, naslednje leto pa je izdal članek z naslovom 'A Relational Model of Data for Large Shared Data Banks' [3]. Relacijska podatkovna baza (ang. relational database) je organizirana zbirka podatkov. Ti so v njej organizirani v obliki relacij, ki jih upodobljamo v tabelarični obliki [2]. Relacijske podatkovne baze so namenjene obdelavi strukturiranih podatkov, ki so shranjeni centralno. V relacijski podatkovni bazi se lahko nad podatki izvajajo različne operacije: poizvedovanje, vstavljanje, brisanje in posodabljanje. Ena izmed prednost sistema za upravljanje s podatkovnimi bazami je, da lahko nudi dostop do podatkov več odjemalcem[2] hkrati [9].

V relacijski podatkovni bazi lahko podatke shranjujemo v eno ali več tabel. Vsaka tabela v njej je sestavljena iz dveh delov. To sta čelna vrstica, ki se imenuje tudi relacijska shema in iz podatkovnih vrstic, ki se imenujejo relacije. V čelni vrstici je pojasnjen pomen posameznih stolpcev v tabeli. Na podlagi ujemajoče vsebine posameznih stolpcev se lahko tabele medsebojno povezuje [5]. Če bi želeli v relacijsko podatkovno bazo shraniti študente in profesorje, bi morali vsako skupino posebej shranili v posamezno tabelo. Povezavo med tema tabelama (skupinama oseb, ki so shranjeni v dveh tabelah) bi predstavljajla relacija, da profesor uči vsaj enega študenta in da ima vsak študent enega ali več profesorjev. Ta relacija bi bila realizirana z uporabo posebne - povezovalne tabele.

Do vsebine tabel uporabnik dostopa s pomočjo poizvedovalnih jezikov, ki so osnovani na postopkovnih ali nepostopkovnih formalnih jezikih. Najbolj uveljavjen nepostopkovni jezik,

[2] S tem izrazom označujemo programsko opremo ali uporabnika, ki preko odjemalca dostopa do podatkovne baze, ki se nahaja na oddaljenem strežniku ali lokalno [1].

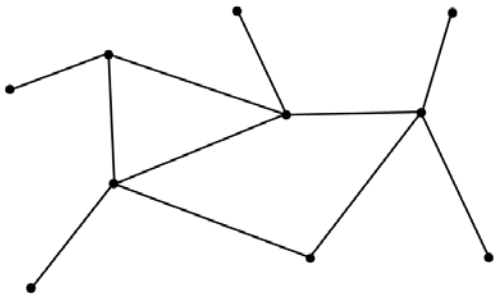
ki je tudi standardiziran je SQL, ki ni le poizvedovalni jezik, pač pa obsega tudi konstrukte za definicijo podatkovnih struktur [5].

Leta 1974 je bil definiran SEQUEL (Structured English Query Language), predhodnik današnjega jezika SQL. SQL se je najprej uporabljal predvsem kot poizvedovalni jezik, kasneje pa je postal tudi jezik za specifikacijo notranje in zunanje sheme podatkovne baze. S koncem 70-ih let je postal standarden jezik za delo z relacijskimi podatkovnimi bazami. ANSI pa ga je formalno standardiziral leta 1986 [7].

Sistem za upravljanje s podatkovno bazo (SUPB – ang. DBMS) je programska oprema, ki služi kot vmesnik (ang. interface) med odjemalci in samo podatkovno bazo. Ker je tako zelo tesno povezan s podatkovno bazo, se termin 'podatkovna baza' skoraj vedno uporablja za poimenovanje samega sistema SUPB, ki upravlja s podatki in podatkovne baze. Danes dobro znane podatkovne baze so: MySQL, SQLite, Microsoft SQL Server, Microsoft Access, Oracle, SAP in dBase [9]. Glavne naloge sistema SUPB so upravljanje s pomnilnikom, obnavljanje podatkovne baze po nesrečah, optimizacija poizvedb in sočasni nadzor nad izvajanjem transkacij [5].

2.3. TEORIJA GRAFOV

V matematiki predstavlja graf množico objektov oziroma strukturo, kjer so nekateri pari objektov povezani s povezavami. Objekti, ki so medsebojno povezani, so predstavljeni z matematičnimi abstrakcijami, ki jih imenujemo vozlišča (ang. nodes, vertices). Črte, ki povezujejo ta vozlišča med seboj, so povezave (ang. relations, edges). Običajno je graf upodobljen kot množica točk, ki označujejo vozlišča in so med seboj povezana s povezavami v obliki daljic ali krivulj, kar nam ponazarja primer na Sliki 1. Grafi so eno izmed področij, ki jih preučuje diskretna matematika s pomočjo teorije grafov [15].



Slika 1: Primer grafa z desetimi točkami in enajstimi povezavami.

Povezave med točkami grafa so lahko usmerjene ali neusmerjene. Usmerjeno povezavo med točkami prikažemo s puščico, neusmerjeno pa brez. Če podamo primer, da vozlišča predstavljajo osebe na zabavi, bi z neusmerjeno povezavo lahko prikazali povezavo rokovanja med dvema osebama. Če se oseba A rokuje z osebo B, potem velja tudi, da se oseba B rokuje z osebo A - povezava velja v obe smeri. Za drug primer predpostavimo, da vozlišča predstavljajo ljudi na zabavi in obstaja povezava od osebe A do osebe B. Če oseba A pozna osebo B, je to usmerjena povezava, ker ni nujno da oseba B pozna osebo A [15]. Razliko med usmerjenim in neusmerjenim grafom prikazuje slika 2.

Slika 2: Razlika med usmerjenim in neusmerjenim grafom.

2.4. TRANSAKCIJE V PODATKOVNIH BAZAH

V več poglavjih omenjamo transakcije v podatkovnih bazah, zato smo to poglavje namenili kratki predstavitvi, kaj je to transakcija in kaj so ACID lastnosti transakcij.

Transakcija predstavlja ukaz ali serijo ukazov podanih s strani odjemalca, katerih namen je branje ali spreminjanje vsebine podatkovne baze. Sama izvršitev transakcije povzroči spremembo vsebine podatkovne baze. Poizvedbe nad podatkovno strukturo v podatkovni bazi, spadajo tudi med transakcije [1]. Primer transakcije predstavlja vnos novega podatka v podatkovno bazo. Pred vnosom podatka ima podatkovna baza začetno stanje (stanje pred transakcijo), po vnosu se doda podatek, ki spremeni njeno stanje – vsebino ali strukturo.

Zagotavljanje ACID lastnosti transakcij je naloga SUPB. V slovenskem jeziku kratica predstavlja pojme: atomarnost, konsistentnost, neodvisnost in trajnost. Vsak izmed pojmov se nanaša na lastnosti transakcij. Atomarnost pomeni, da se mora transakcija, ki se izvaja, izvesti v celoti drugače se v celoti zavrne. Konsistentnost transakcij, poskrbi za zavrnitev celotne transakcije (podatkovna baza se vrne v prejšnje stanje) v primeru neveljavnega stanja (prišlo je do napake) podatkovne baze med izvajanjem transakcije. Ker se v podatkovni bazi lahko hkrati izvaja več transakcij, se mora vsaka izvajati neodvisno, drugače obstaja možnost vpliva rezultatov ene transakcije na rezultate drugih – neodvisnost. Transakcija je trajna, če se je uspešno izvedla, njen rezultat pa trajno shranjen tudi kasneje [1].

2.5. PODATKOVNA BAZA GRAFOV

Za razliko od relacijskih podatkovnih baz, kjer se podatki shranjujejo v obliki tabel, uporablja podatkovna baza grafov za shranjevanje podatkov strukturo grafa. Sistem, ki upravlja s podatkovnimi bazami grafov je skoraj identičen sistemu za upravljanje z relacijskimi podatkovnimi bazami. Prav tako omogoča osnovne operacije: ustvari, beri, posodobi, briši (ang. Create, Read, Update, Delete), nudi dostop več odjemalcem hkrati, ter izvaja transakcije nad podatkovno bazo. Bistvena razlika je, da ta sistem deluje nad podatkovnim modelom v obliki grafa [6].

Veliko podatkovnih baz tipa NoSQL ne podpira transakcij. Podatkovne baze grafov omogočajo izvajanje transakcij, vendar vse ne podpirajo ACID lastnosti transakcij [6].

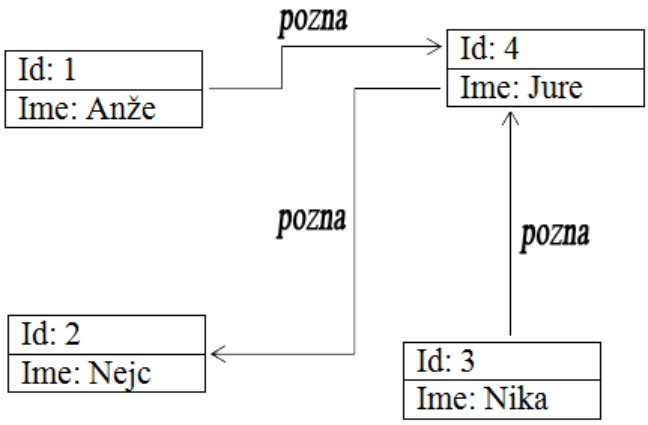
2.5.1. PODATKOVNI MODEL – STRUKTURA

Podatkovne baze grafov temeljijo na teoriji grafov. Podatkovni model, ki opredeljuje strukturo podatkov, vsebuje naslednje gradnike: množico vozlišč, povezav in lastnosti. Vsi ti gradniki skupaj tvorijo strukturo – graf. Vozlišča lahko primerjamo z vrsticami tabel iz relacijskih podatkovnih baz. Predstavljajo entitete kot so na primer osebe, posli, računi ali drugi predmeti iz realnega sveta. Vsako vozlišče vsebuje eno ali več lastnosti. Za razliko od relacijskih podatkovnih baz, kjer imamo shemo tabele v čelni vrstici, posamezno vozlišče nima predpisane sheme, zato mu lahko dodajamo ali odstranjujemo lastnosti, brez vpliva na ostala vozlišča [16].

Relacije v grafu so predstavljene s povezavami, ki povezujejo vozlišča med seboj. Vozlišča so lahko povezana z usmerjenimi ali neusmerjenimi povezavami – glej poglavje Teorija grafov. Vsaka od povezav ima svojo oznako, ki označuje pomen povezave med dvema vozliščema. Podatki v podatkovnih bazah grafov se nahajajo tudi na povezavah, ki povezujejo vozlišča. Ti povedo, v kakšnem razmerju sta dve povezani vozlišči – kakšen pomen ju povezuje med seboj. Vzorci podatkov v grafu se pojavijo, ko nekdo preučuje medsebojne povezave vozlišč in lastnosti vozlišč. Enostaven primer vzorca, je iskanje vseh vozlišč, povezanih z začetnim vozliščem iz grafa podatkovne baze, ki ga sami določimo [6]. Če vzamemo za primer skupino oseb, bi povezave med njimi lahko predstavljale relacijo 'pozna'. Usmerjenost povezav pa bi tako povedala, kdo pozna koga.

Lastnosti so podatki, ki se nanašajo na posamezno vozlišče. Lastnost je predstavljena z oznako lastnosti in vrednostjo, ki jo ima ta lastnost. Na primer, če bi bila oseba z imenom 'Matej' eno izmed vozlišč, bi lahko to vozlišče vsebovalo lastnosti kot so: 'starost', 'višina' itd., ter njihove vrednosti.

Slika 3 prikazuje štiri vozlišča, ki so zaporedno označena z identifikacijskimi šteilkami od 1 do 4 ter ponazarjajo osebe o katerih hranimo podatek 'ime'. Med seboj so povezana z usmerjenimi povezavami, iz katerih lahko razberemo, kako se osebe med seboj poznajo. Primer: Osebo z imenom 'Jure' poznata osebi z imenoma 'Anže' in 'Nika'.



Slika 3: Primer podatkovne strukture v podatkovni bazi Neo4j.

2.6. ZMOŽNOSTI PODATKOVNIH BAZ GRAFOV

Ker podatkovna struktura podatkovnih baz grafov ni tako toga kot pri relacijskih podatkovnih bazah, so te bolj primerne za delo s pol strukturiranimi in nestrukturiranimi podatki, kjer shema ni natančno opredeljena. Tukaj se kaže njena prednost za uporabo pred relacijskimi podatkovnimi bazami [16].

Zmožnosti podatkovnih baz grafov se pokažejo pri izvajanju poizvedb iskanja nad povezanimi podatki. Za razliko od relacijskih podatkovnih baz, ki povezujejo podatke s pomočjo stičnih operacij, so tukaj podatki povezani že sami v podatkovni strukturi. To se odraža pri učinkovitosti izvajanja poizvedb. Poizvedbe se hitreje izvedejo, ker se pri iskanju enostavno pomikamo po strukturi grafa. Tako točno vemo, kateri podatki so povezani med seboj in kateri ne. Izkažejo se tudi pri iskanju povezanih podatkov, kateri so povezani med seboj preko ene ali več povezav. Primer lahko predstavlja iskanje oseb, ki so prijatelji prijateljev neke osebe. Tukaj bi relacijske podatkovne baze morale izvajati dvojne stične operacije, katere bi zelo negativno vplivale na učinkovitost izvajanja poizvedb [6].

2.7. PRIMERI UPORABE PODATKOVNE BAZE GRAFOV

Primere uporabe predstavljajo predvsem področja, kjer se srečamo s povezanimi podatki in izvajamo poizvedbe, ki so namenjene iskanju povezanih podatkov.

Področje na katerem imamo največ primerov uporabe podatkovnih baz grafov predstavlja področje socialnih omrežij. Njihovo strukturo lahko zelo lepo prikažemo v obliki grafa, kjer osebe predstavljajo vozlišča, relacije med osebami pa so predstavljene s povezavami med vozlišči. Ker se ključni podatki v socialnih omrežjih nahajajo na povezavah, se podatkovne baze grafov dobro izkažejona tem področju [6].

Smiseln primer uporabe bi lahko bilo tudi shranjevanje zaposlenih nekega večjega podjetja v podatkovno bazo grafov. Zaposlene iz različnih oddelkov bi lahko združili v podgrafe, te pa v celotno strukturo – graf. S pomočjo nastalega grafa (potem, ko bi združili vse podgrafe), bi dobili zelo jasno strukturo, kako je povezan celoten kolektiv[3] podjetja. Temu lahko drugače rečemo iskanje neformalne organizacije znotraj poslovnega sistema.

2.8. PREDNOSTI PODATKOVNE BAZE GRAFOV

2.8.1. UČINKOVITOST

Učinkovitost podatkovnih baz grafov, se kaže pri delu s povezanimi podatki. Zaradi podatkovne strukture, ki temelji na grafu, se pri poizvedbah nad podatki znebimo stičnih operacij. Stične operacije z večanjem količine podatkov močno poslabšujejo zmogljivost v primeru relacijskih baz, dočim se v primeru podatkovnih bazah grafov zmogljivosti stičnih operacij s količino podatkov bistveno ne spreminjajo [6].

[3] Skupnost ljudi, ki jih povezuje skupno delo ali interesi.

2.8.2. FLEKSIBILNOST

Fleksibilnost grafov omogoča, da lahko obstoječi podatkovni strukturi dodajamo nove vrste povezav, nova vozlišča in nove podgrafe, brez motenja že sestavljenih poizvedb. Ta lastnost ima pozitivne posledice, ko se srečamo s področji, kjer obstaja možnost, da se bo podatkovna shema spreminjala [6].

2.8.3. AGILNOST

Agilnost podatkovne baze grafov se nanaša na njihovo fleksibilno strukturo. Ta omogoča enostavno prilagajanje strukture shranjenih podatkov v primeru spreminjanja podatkovnega modela.

2.9. SLABOSTI PODATKOVNE BAZE GRAFOV

Čeprav je fleksibilnost grafa pozitivna lastnost, se lahko zgodi, da postane struktura grafa zelo kompleksna. Enako se zgodi s podatki shranjenimi v podatkovni bazi grafov, če poljubno vstavljamo in povezujemo podatke v njej.

Zaradi kompleksnejše podatkovne strukture (ogromno število povezav med vozlišči), potrebujemo več podatkov za njem opis (meta podatki), kar se odraža v večji porabi prostora na pomnilnem mediju [6].

Ena izmed slabosti podatkovne baze grafov se kaže v podatkovni strukturi. Povezave med vozlišči v podatkovni strukturi so enosmerne, zaradi česar se lahko pomikamo po njej le v eni smeri. Zaradi tega se slabše izvajajo poizvedbe, pri katerih bi potrebovali možnost premikanja po strukturi nazaj. Primer take poizvedbe je iskanje oseb, ki imajo pod prijatelji osebo Matej. Taka poizvedba mora za vsako osebo v podatkovnem grafu preveriti ali pozna osebo Matej. Če bi povezave bile dvosmerne, se bi preprosto premaknili v vozlišče, ki vsebuje lastnost Matej in pogledali kam se lahko pomaknemo nazaj ter vrnili ta vozlišča [6].

Podatki v podatkovni bazi se hranijo na trdem disku ali podobnem mediju. Pri izvajanju poizvedb se prenesejo podatki iz trdega diska v glavni pomnilnik računalnika, ker je njihovo izvajanje potem hitrejše. Slabost podatkovnih baz grafov se pokaže, pri delovanju nad večjimi količinami podatkov, ki jih ne moremo v celoti prenesti v glavni pomnilnik. Tukaj nastane problem, kako razdeliti graf na več delov [6].

3. NEO4J

Neo4j je odprtokodna podatkovna baza, ki temelji na shranjevanju podatkov v obliki strukture grafa. Kakor relacijske podatkovne baze, tudi Neo4j podpira ACID lastnosti transakcij, ki se izvajajo nad podatki. Omogoča shranjevanje več milijard vozlišč, povezav in lastnosti. Ponuja možnost njene porazdelitve čez več računalniških sistemov. Za poizvedbe nad njenimi podatki in podatkovno strukturo se uporablja poizvedovalni jezik Cypher [25].

3.1. NAMESTITEV

3.1.1. IZBIRA RAZLIČICE

Neo4j ponuja dve različici strežnika. Prva različica je tako imenovana stabilna izdaja (Stable Release) [11]. Stabilna izdaja programske opreme imenujemo različico, ki se v prihodnosti ne bo spreminjala (izgled, funkcionalnosti, specifikacije, uporabniški vmesnik). Edine spremembe, ki so mogoče na taki različici programske opreme, so popravki varnostnih lukenj in hroščev. Večina programske opreme z oznako stabilna izdaja, je namenjena za uporabo širšemu krogu ljudi [24]. Druga različica izdaje se imenuje mejna izdaja (ang. milestone – mejni kamen, mejnik). Ponekod v računalništvu se tako različico programske opreme imenuje tudi beta različica. Taka vrsta izdaje je namenjena širši skupnosti razvijalcev in programerjev, ki želijo testirati in preizkušati najnovejše zmožnosti različice, ki se še razvija. Ti lahko podajo razvijalcem, ki razvijajo novo različico, svoje mnenje, ponudijo lastne popravke, izboljšave in razširitve, zato da bi bila naslednja različica stabilne izdaje boljša od prejšnje [19].

3.1.2. LICENCA

Neo4j ponuja tri vrste licence programske opreme. Neo4j Community je odprtokodna rešitev pod GPLv3 licenco. To je osnovna različica programske opreme, namenjena širšemu krogu ljudi. Ne vsebuje nekaterih dodatkov, ki so del drugih različic in nima zagotovljene tehnične in servisne pomoči. Različici Neo4j Advanced in Neo4j Enterprise sta na voljo pod komercialno licenco in licenco AGPL. Ti dve različici vsebujeta več dodatkov, kot je na primer razširjen spletni vmesnik ter omogoča spremljanje prometa s pomočjo protokola SNMP. Ponujata tudi tehnično in servisno pomoč. Zaradi več dodatkov sta bolj primerni za uporabo v poslovnih spletnih aplikacijah. Vse tri rešitve lahko brezplačno testiramo, pod odprtokodno licenco [22].

3.1.3. PRENOS IN NAMESTITEV

Na voljo so različice, ki so prilagojene za vse najbolj razširjene platforme (Linux, Mac OS in Windows) [11].

3.1.4. MOŽNI PROBLEMI

Omenili bomo še nekaj problemov, ki se lahko pojavijo pri zagonu strežnika Neo4j. Pogosto se zgodi, da v sistemu nimamo nameščene ustrezne različice programske opreme Java, ki je potrebna za delovanje strežnika Neo4j. Sistem na to napako opozori pri zagonu strežnika in nato lahko namestimo ali posodobimo Javo v sistemu.Redkeje se zgodi, da so zasedena vrata 7474, preko katerih strežnik izvaja komunikacijo, kar lahko rešimo na več načinov:

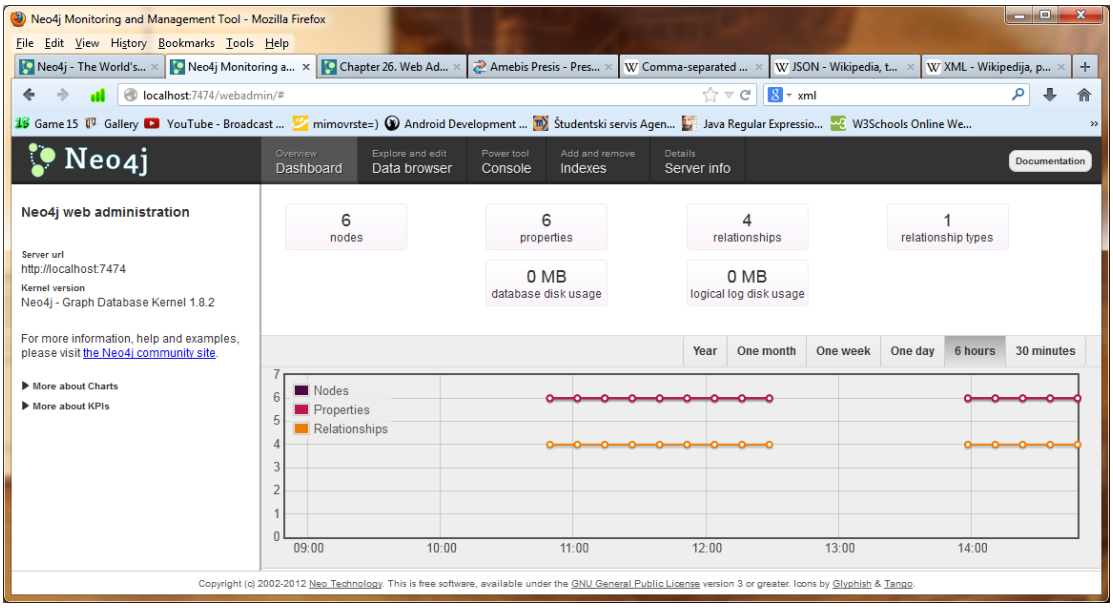
- lahko zapremo program, ki komunicira preko teh vrat in nato zaženemo strežnik,
- lahko spremenimo vrata programu, ki zaseda ta vrata,
- lahko spremenimo vrata Strežniku Neo4j datoteki *conf/neo4j-server.properties*. Namesto vrat 7474 lahko izberemo katera druga, ki niso zasedena. Na spletu poiščemo spisek vrat in programov, ki poslušajo na teh vratih, ter izberemo nam najbolj ustrezna [23]. Ob spremembi vrat, ne smemo pozabiti spremeniti lokacije v spletnem uporabniškem vmesniku, pri dostopanju do strežnika. Poleg tega moramo spremeniti tudi vrata preko katerih aplikacije komunicirajo s strežnikom Neo4j.

3.2. DOSTOP DO STREŽNIKA

Za dostop do strežnika Neo4j in njegove podatkovne baze so na voljo različna orodja. Osnovni orodji, ki sta nam na voljo pri namestitvi, sta spletni uporabniški vmesnik in ukazna lupina (ang. shell). Ukazno lupino Neo4j najdemo v mapi bin. Z njo preko tipkovnice komuniciramo s strežnikom. Obstaja še programsko orodje Neoclipse, ki je namenjeno za uporabo s podatkovnimi bazami grafov. Z njim lahko pregledujemo, spreminjamo, raziskujemo in obdelujemo podatke shranjen v podatkovni bazi Neo4j. Neoclipse je podrobneje opisan v poglavju 3.4.

3.3. SPLETNI UPORABNIŠKI VMESNIK

Spletni uporabniški vmesnik je orodje, ki omogoča komuniciranje s podatkovno bazo Neo4j. Namenjeno je spremljanju stanja strežnika Neo4j in upravljanju s podatki, ki se nahajajo v podatkovni bazi.



Slika 4: Spletni uporabniški vmesnik.

3.3.1. PREDOGLED

Ob vpisu naslova *http://localhost:7474/* v brskalnik, se odpre vmesnik (Slika 4), ki omogoča upravljanje z Neo4j. Odpre se pod pogojem, da imamo strežnik Neo4j nameščen in zagnan [26]. V nasprotnem primeru brskalnik vrne informacijo, da ne more vzpostaviti povezave z z naslovom, ki smo ga vnesli.

V zavihku za predogled (ang. Dashboard) spremljamo lastnosti strežnika Neo4j. Na levi strani se izpiše URL strežnika ter različica jedra nameščene podatkovne baze Neo4j. Večji del vmesnika zavzema izpis stanja grafa podatkovne baze. Izpis informacij vsebuje število vozlišč, lastnosti, povezav in tipov povezav v podatkovnem grafu. Poleg se nahaja pregled nad porabo prostora na trdem disku, ki ga zaseda podatkovna baza in dnevnik podatkovne baze. Spodaj se izrisuje graf števila vozlišč, lastnosti in relacij, ki se samodejno osvežuje in označuje količino posameznih elementov v točno določenem časovnem obdobju. Izbiramo lahko med časovnimi obdobji od trideset minut pa vse tja do enega leta.

3.3.2. PREGLEDOVALNIK PODATKOV

V pregledovalniku podatkov lahko pregledujemo in urejamo podatke v grafu. Vsebuje vnosno polje, kamor vnašamo različne poizvedbe v jeziku Cypher. Z njim lahko urejamo strukturo shranjenih podatkov (dodajamo,odstranjujemo, posodabljamo elemente in povezave med njimi). Pod njo se nahaja statusna vrstica, ki nam izpiše kaj se je zgodilo ob izvajanju poizvedbe. Lahko se izpiše število vrstic, ki jih je vrnila poizvedba, koliko časa je ta potrebovala za izvajanje ali pa vrne poročilo o napaki ob primeru napačnega vnosa poizvedbe. Pod njo se izpisuje rezultat izvršene poizvedbe. Namesto izpisa v vrsticah vmesnik ponuja tudi izpis rezultata poizvedbe v obliki grafa. Zraven spadata še gumba za ustvarjanje vozlišč in relacij, s katerima lahko hitro dodajamo nove elemente v graf podatkov.

3.3.3. KONZOLA

Konzola, ki je vgrajena v spletni uporabniški vmesnik, omogoča tri različne načine upravljanja podatkovne baze Neo4j:

- upravljanje s skriptnim jezikom Gremlin, ki se uporablja pri delu z grafi. Z njim spreminjamo strukturo grafa (dodajamo, spreminjamo odstranjujemo vozlišča ter povezave) [18].
- upravljanje preko konzole Cypher, kjer izvajamo poizvedbe v jeziku Cypher.
- upravljanje preko protokola HTTP.

3.3.4. INFORMACIJE STREŽNIKA

Pod zavihkom informacije o strežniku (*Server info*) se nahajajo podrobnejše informacije o nastavitvah strežnika.

3.4. NEOCLIPSE

3.4.1. PREDSTAVITEV

Program Neoclipse je nastal kot del projekta Neo4j. Neoclipse je samostojna aplikacija, s katero lažje dostopamo do Neo4j (strežnika in podatkovne baze). Prednost Neoclipsa je lepša vizualizacija podatkovnega grafa, kot jo ponuja spletni uporabniški vmesnik. Omogoča tudi možnost označevanja s pomočjo ikon, ki jih dodajamo vozliščem, da si lažje predstavljamo, v katero skupino podatkov spada vozlišče. Omogoča preprosto povečanje ali zmanjšanje prečne globine grafa[4], ki ga prikazuje, filtriranje glede na tipe relacij, ter poudarjanje vozlišč in relacij s pomočjo barv [21].

3.4.2. NAMESTITEV

Neoclipse se nahaja na spletni strani Neo4j. Obstajajo različice za sisteme Linux, Mac OS in Windows. Pogoj za delovanje programa je nameščena Java različice 1.6 ali več [21].

3.4.3. STRUKTURA ZASLONSKE MASKE PROGRAMA

Poleg vrstice z gumbi na vrhu zaslonske maske je v privzetih nastavitvah še pet odprtih oken, ki si jih lahko prilagajamo po lastnih željah in potrebah. Na dnu zaslonske maske najdemo statusno vrstico, kjer so izpisani podatki o grafu. Izpisani so podatki o presečni globini grafa, ter številu vozlišč in relacij v njem. Videz zaslonske maske in primer izpisa podatkovne baze nam prikazuje Slika 5.

Vrstica z gumbi je sestavljena iz sedmih gumbov. S klikom na prvega dostopamo do nastavitev programa. Ostalih šest gumbov se uporablja za odpiranje in zapiranje oken (povezave, prikaz grafa, pomoč, lastnosti ...) pri delu s podatkovno bazo.

Na levi strani zaslonske maske je okno, kjer se nahajajo povezave s podatkovnimi bazami. Aktivna povezava je označena z zeleno barvo. V njem lahko ustvarjamo nove povezave, spreminjamo obstoječe, prekinemo aktivno povezavo in vzpostavimo novo.

Okno na sredini je najpomembnejše in je sestavljeno iz dveh zavihkov. Prvi zavihek je *Database graph*, kjer Neoclipse izrisuje graf podatkovne baze, na katero smo trenutno povezani. Vozlišča so prikazana kot lističi, kjer so izpisane lastnosti v obliki atributov in vrednosti. Vozlišče, ki je trenutno aktivno, je obarvano drugače kot ostala vozlišča. Med povezanimi vozlišči program izriše povezavo in na vsaki označi tip povezave. Puščice, ki so na koncih povezav označujejo njihovo smer. Nad poljem za izris grafa se nahaja še nekaj gumbov, od katerih so najbolj uporabni gumbi za ustvarjanje in prekinjanje povezave, brisanje elementov, ter gumbi za različne ureditve in razporeditve pogleda nad elementi grafa. V drugem zavihku se nahaja urejevalnik Cypher. V tem zavihku izvajamo poizvedbe nad grafom, ki jih vnesemo v jeziku Cypher. Po izvršitvi poizvedbe, program izpiše rezultat, ki je odvisen od tipa poizvedbe, ki se je izvršila. Obstaja tudi možnost izvoza rezultatov poizvedb v treh različnih označevalnih jezikih (CSV, JSON, XML).

Spodaj levo je postavljeno okno *Properties*. Tukaj se izpisujejo lastnosti trenutno izbranega elementa. V levem stolpcu se izpiše tip elementa (vozlišče ali relacija) in lastnosti, v drugem

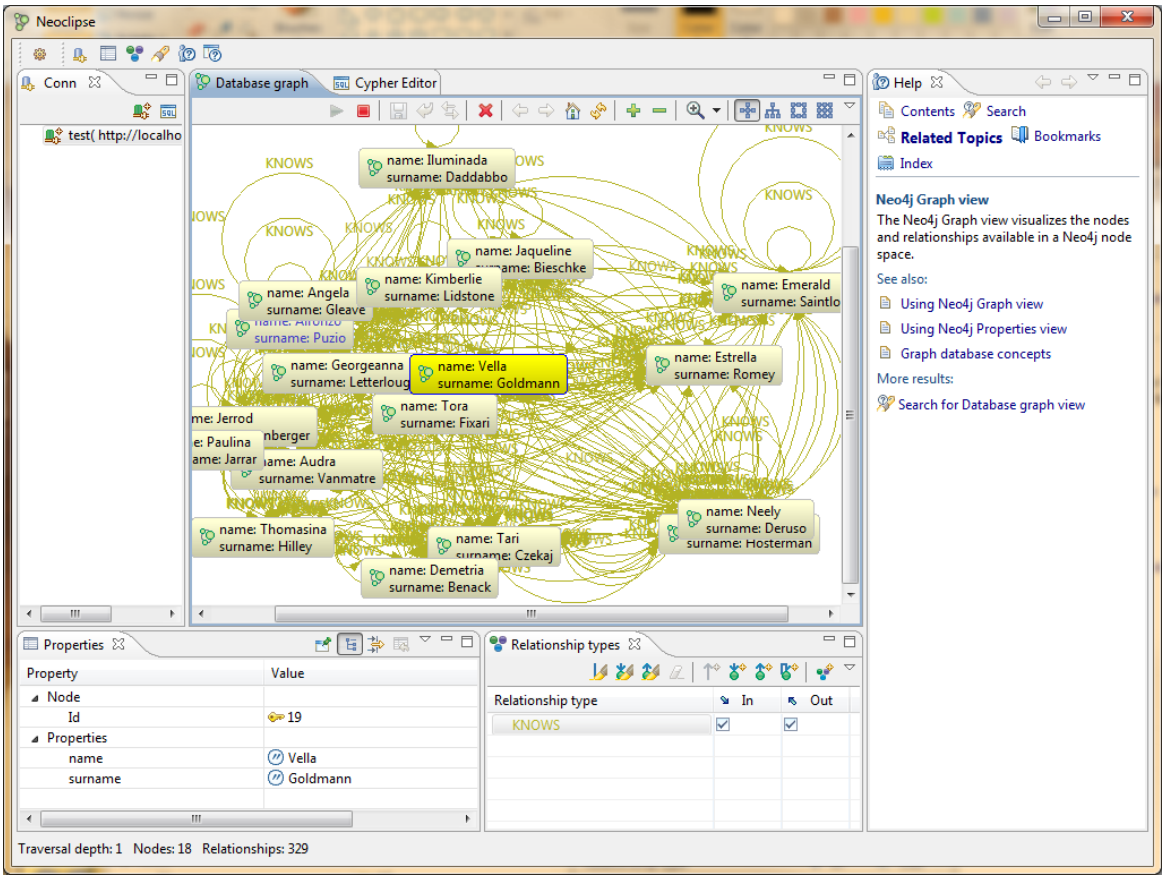
[4] Prečna globina grafa pove koliko je dolga najdaljša pot med dvema vozliščema, ki se nahaja v grafu.

stolpcu pa so vrednosti izbranega elementa. Ob identifikacijskih vrednostih je narisan ključ, ob vrednostih lastnosti pa tip vrednosti kot na primer niz, število ...

Poleg se nahaja okno, kjer urejamo relacije med vozlišči. Tukaj se izpisujejo vsi tipi relacij, ki obstajajo v grafu. Na vrhu okna se nahajajo gumbi, s katerimioznačujemo relacije:

- Označimo lahko vse relacije izbranega tipa ter začetna ali končna vozlišča relacij izbranega tipa.
- Lahko ustvarjamo nove tipe relacij, nove relacije med obstoječimi vozlišči ter dodajamo nova vozlišča z relacijami.

Ostane še okno za pomoč, ki se nahaja na desni strani. Tu se izpisujejo informacije, ki so v pomoč pri uporabi orodja. Okno omogoča brskanje po pomoči, vsebuje pa tudi priročnik za uporabo. Ob menjavi aktivnega okna se izpišejo obstoječe teme pomoči, ki se nanašajo na aktivno okno.



Slika 5: Program Neoclipse in primer izpisa vsebine grafa podatkovne baze.

3.4.4. KONFIGURACIJA POVEZAVE

Ko namestimo program, je potrebno vzpostaviti povezavo s podatkovno bazo Neo4j. Novo povezavo z bazo ustvarimo s klikom na gumb *New Connection*, ki se nahaja v levem podoknu z imenom Connections. S klikom odpremo okno, ki je prikazano na sliki 6. Najprej vnesemo ime povezave. Priporočen je vnos imena, ki ponazarja bistvo shranjenih podatkov. Če imamo

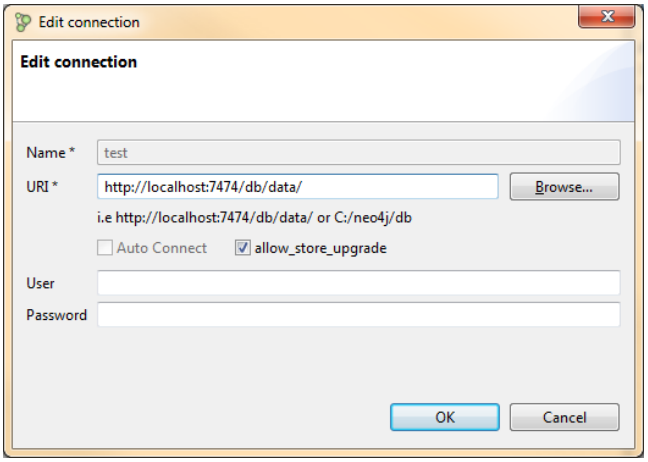
na primer v bazi shranje podatke o filmih lahko damo ime povezavi *filmi*. Naslednji vnos je URI, ki ga lahko vnesemo na dva načina:

- enotno ime vira (URN) ali
- enotni lokator vira (URL).

Pod vnosnim poljem že sam program ponuja primera obeh vnosov. Prva možnost je URL. Z njim se povežemo z bazo, ki se nahaja na nekem spletnem viru ali na lokalnem računalniku. Če se baza nahaja na lokalnem računalniku, moramo vnesti pravilna vrata, v nasprotnem primeru nam Neoclipse vrne napako pri povezovanju. Druga možnost je URN. To možnost se večinoma uporablja za dostopanje do podatkovnih baz, ki se nahajajo na lokalnem računalniku, ker vnašamo neposredno pot do nje. Pod URI vnosom imamo možnost *Auto Connect*, ki poskrbi za avtomatsko povezavo na bazo.

Obstajata še dve opcijski polji za vnos uporabniškega imena in gesla. Če Neo4j namestimo na lokalnem računalniku, teh podatkov ni potrebno vnašati, ker po privzetih nastavitvah podatkovne baze niso označeni kot obvezni. Za dostop do baze, ki se nahaja nekje na svetovnem spletu, pa se vedno uporablja uporabniško ime in geslo, da ne pride do zlorabe. Ko so vnešeni vsi potrebni podatki, potrdimo vnos s klikom na gumb. V okencu Connections se pojavi povezava, ki smo jo ustvarili. Ko s klikom aktiviramo povezavo, se v osrednjem oknu izriše graf podatkov, ki se nahajajo v bazi.

Različica programa, ki smo ga preizkušali, ni omogočala hkratne aktivnosti več povezav. Če bi torej želeli urejati še kakšno drugo podatkovno bazo, bi morali najprej prekiniti prvo povezavo in vzpostaviti novo.



Slika 6: Konfiguracija povezave na lokalno bazo Neo4j.

4. POIZVEDOVALNI JEZIK – CYPHER

4.1. POIZVEDOVALNI JEZIK

Pri sporazumevanjem med ljudmi, lahko prihaja do nepravilnega razumevanja sporočil, kar privede do napak pri prenosu informacij ali njihovem razumevanju. Podobno velja za komunikacijo med podatkovno bazo in odjemalcem. Sporočila v človeškem sporazumevalnem jeziku so lahko dvoumna in nenatančna in jih ne moremo v taki obliki neposredno posredovati sistemu SUPB. Zato je bilo potrebno uvesti sistem pravil, ki so bolj natančna in strukturirana, hkrati pa imajo manj možnosti za napačno razumevanje in predstavo. Te sisteme pravil poznamo pod imenom poizvedovalni jeziki, ki so standardizirani in poenoteni za uporabnike, ter razvijalce podatkovnih baz [4].

4.2. CYPHER

Cypher je deklarativen poizvedovalni jezik, ki se uporablja za pisanje poizvedb v podatkovni bazi grafov – Neo4j. Omogoča ekspresivno ter efektivno povpraševanje (z enostavnim izrazom izrazimo učinkovito iskanje kompleksnega dela podatkov, ki se nahaja v grafu) in posodabljanje shranjene oblike grafa. Primer je stavek MATCH (poglavje 4.4.2.), kjer z enostavnim izrazom povemo kaj iščemo. Jezik še vedno raste in se razvija, kar pomeni, da se bo njegova sintaksa še spreminjala in dopolnjevala [8].

Načrtovan je tako, da bi bil primeren tako za razvijalce kot za uporabnike, ki izvajajo poizvedbe nad bazo podatkov [8].

Cypher spodbuja veliko število pristopov in načinov na ustaljenih praksah za ekspresivno povpraševanje. Večina ključnih besed kot na primer WHERE in ORDER BY je povzetih po jeziku SQL [8].

Kot deklarativni jezik, se osredotoča na jasnost izražanja, kako lahko manipuliramo s podatki shranjenimiv grafu podatkovne baze Z njim torej povem kaj želimo, ne pa kako se to izvede [8].

4.3. SINTAKSA JEZIKA CYPHER

4.3.1. OPERATORJI

Cypher uporablja povzeto po [8]:

- Matematičnie operatorje: +, -, *, / in %.
- Operatorje za primerjavo: =, <>, <, >, <=, >=.
- Boolean operatorje: and, or, not.
- Operatorje za delo z nizi: nize se lahko združuje z + operatorjem.
- Operatorje za delo z zbirkami: zbirke se lahko združuje z + operatorjem.
- Operatorje lastnosti: dva posebna operatorja ? in !.

Posebne operatorje se uporablja pri poizvedbah, kjer primerjamo lastnosti elementov. Primerjava z lastnostjo, ki ne obstaja, povzroči napako. V izogib preverjanju obstoja lastnosti nekega elementa, lahko uporabimo operatorja povzeto po [8]:

- ?, ki vrača vrednost true, če lastnost elementa v izrazu ne obstaja v podatkih.
- !, ki vrača vrednost false, če lastnost elementa v izrazu ne obstaja v podatkih.

Ta predikat bo ovrednotil vrednost true, če lastnost elementa n.name ne obstaja.

WHERE n.name? = ''matej''

Ta predikat bo ovrednotil vrednost false, če lastnost elementa n.name ne obstaja.

WHERE n.name! = ''matej''

4.3.2. IZRAZI

V jeziku Cypher so podatki predstavljeni na več načinov povzeto po [8]:

- S celoštevilskimi ali decimalnimi vrednostmi: 81, 10045, 0.12, 78.1.
- Z nizi: "Matej", 'pozdravljen', 'Neo4J' (enojni ali pa dvojni narekovaji).
- Z boolean operatorji: true, false, TRUE, FALSE.
- S parametri: {param}, {13}, {1,4,3,45}.
- Z zbirko izrazov: ["a","z"], [], [11,12,13].
- S klici funkcij: length(n), nodes(n).
- Z agregatnimi funkcijami: avg(x.prop), count(*).

4.3.3. IDENTIFIKATORJI

Kadar se sklicujemo na del vzorca (poglavje 4.3.5), mu damo imena oziroma identifikatorje. V spodnjem primeru sta identifikatorja x in y [8].

START x=node(1) MATCH x-->y RETURN y

Imena identifikatorjev so občutljiva na velike in male črke. Vsebujejo lahko alfanumerične znake (A-Z,a-z, 0-9), vendar je obvezna uporaba črke za prvi znak [8].

4.3.4. KOMENTARJI

Komentarje v jeziku Cypher dodajamo z uporabo dvojnih poševnic (//). Dodajamo jih lahko na konec posamezne vrstice, ali pa za komentar uporabimo celotno vrstico. Tako omogočimo lažje razumevanje nekomu, ki bere kodo za nami, hkrati pa si s komentarji lahko čez čas tudi sami osvežimo spomin [8].

Primer uporabe komentarja:

START n=node(1) RETURN n //to je komentar na koncu vrstice

START n=node(1)

//to je celovrstični komentar

RETURN n

4.3.5. VZORCI

Vzorci so bistven del jezika Cypher. Z vzorci opisujemo obliko podatkov, ki jih iščemo. Uporabljajo se v stavku MATCH. Razumevanje vzorcev je za učinkovito rabo jezika Cypher zelo pomembno [8].

Vzorec je ena ali več povezav, ki so ločene z vejicami. Povezave lahko definiramo kot skupino zaporednih vozlišč, povezanih med seboj brez premorov, z usmerjenimi ali neusmerjenimi povezavami. Primer povezave lahko prikažemo z dvema točkama *a* in *b*, ki sta med seboj povezani: *(a)-->(b)*. Ta primer vzorca pokaže usmerjeno povezavo od točke *a* do točke *b* [8].

Vzorci imajo tudi robne točke oziroma začetne točke. Te so del vzorca, ki je že povezan z množico vozlišč ali relacij. Vsi deli vzorca morajo biti neposredno ali posredno povezani z začetno točko grafa – vsa vozlišča morajo biti povezana v celotno struktoro [8].

4.4. BRALNI STAVKI

To so tiste besede oziroma stavki, ki jih uporablja jezik Cypher za vračanje vrednosti kot rezultat poizvedbe [8].

4.4.1. START

S stavkom START označujemo začetek poizvedbe. Vsaka poizvedba ima lahko eno ali več začetnih točk. Začetno točko lahko predstavlja vozlišče ali povezava. Te točke označujemo z indeksi. Če bi v poizvedbi želeli uporabiti indeks, ki ne obstaja, bi poizvedba vrnila napako [8].

Primer poizvedbe:

START n=node(1) RETURN n

Poizvedba nam vrne vozlišče z indeksom 1. Če bi na primer, namesto node(1) vnesli node(4) in vozlišče s takim indeksom ne bi obstajalo, bi nam poizvedba vrnila napako[8].

4.4.2. MATCH

Stavek MATCH dodamo, ko želimo izvedeti katera vozlišča so v relaciji z vozliščem, ki smo ga izbrali za začetno. Poleg besede MATCH se uporablja tudi simbol --, ki ponazarja relacijo. Vrstica *MATCH (n)--(x)* na primer poišče vsa vozlišča (ne glede na smer relacije), ki so v

relaciji z začetnim vozliščem (n). Če simbolu dodamo še smer <-- ali --> potem iščemo usmerjena vozlišča, ki so v relaciji z začetnim [8].

4.4.3. WHERE

S stavkom WHERE filtriramo vsebino iz vzorca, ki jo iščemo. Uporablja se v večini primerov z boolean in primerjalnimi operatorji. Primer *WHERE n.age < 30* filtrira vsa vozlišča, ki imajo vrednost lastnosti vozlišča (lastnost age) manjšo od 30 [8].

4.4.4. RETURN

S tem stavkom definiramo, kateri del vzorca poizvedbe nas zanima in ga želimo izpisati. Izpišemo lahko vozlišča, relacije ter njihove lastnosti. Za vrnitev vozlišča postavimo vozlišče (ali seznam vozlišč) v stavek RETURN. Za vračanje lastnosti vozlišč postavimo (.) ob ime vozlišča, ter za njo ime lastnosti, ki jo želimo izpisati. Stavek *RETURN n.age* nam vrne starost, ki se nahaja v vozlišču n. Poseben znak je zvezdica (*), ki jo uporabimo, ko želimo izpisati vsa vozlišča, relacije najdene v poizvedbi (*RETURN **) [8].

4.4.5. AGREGATNE FUNKCIJE

Agregatne funkcije uporabljamo, ko obdelujemo večje število vhodnih vrednosti. Z njimi izračunamo njihovo agregatno vrednost. Primeri agregatnih funkcij so:

- funkcija AVG, ki izračuna povprečno vrednost večjega števila vhodnih numeričnih vrednosti,
- funkcija MIN, ki najde najmanjšo numerično število iz množice,
- funkcija COUNT, prešteje število vseh elementov podane množice.

Stavek *MATCH (a)-->(x) RETURN a, count(*)* na primer vrne vozlišče a, ter prešteje vsa vozlišča, ki so povezana z njim [8].

4.4.6. ORDER BY

S stavkom ORDER BY sortiramo rezultate poizvedbe. Posebnost tega stavka je, da ga ne moremo uporabiti za sortiranje vozlišč ali relacij, ampak le njihovih lastnosti. Stavek ORDER BY se vedno postavi za stavkom RETURN. Kot pri stavkih SQL, lahko tudi tukaj nizamo lastnosti po katerih sortiramo izpis. Če želimo izpis elementov najprej sortirati po lastnosti *age*, nato pa po lastnosti *name*, uporabimo vrstico *ORDER BY n.aget, n.name* [8].

4.4.7. LIMIT

Stavek LIMIT omogoča vračanje le dela rezultata poizvedbe. Kot stavek ORDER BY, se tudi ta uporablja za stavkom RETURN. Poleg stavka dodamo še poljubno številko, ki omeji število izpisov poizvedbe. Če je ta številka večja od števila izpisov bo poizvedba vrnila toliko zapisov kot jih je v rezultatu poizvedbe [8].

Primer poizvedbe, ki vrne prvih 5 elementov rezultata poizvedbe:

MATCH a RETURN a LIMIT 5

4.5. STAVKI ZA VSTAVLJANJE IN POSODABLJANJE PODATKOV

4.5.1. CREATE

Stavek CREATE ustvarja elemente: vozlišča, njihove lastnosti, ter relacije. S preprostim stavkom *CREATE (b {name : 'Matej'})* ustvarimo novo vozlišče z atributom *name* in vrednostjo 'Matej'. Če želimo dodati več atributov in vrednosti v pravkar ustvarjeno vozlišče v zavite oklepaje dodamo atribute in vrednosti ločene z vejicami (*{name : 'Matej', profession: 'student'}).* Za ustvarjanje povezave potrebujemo vsaj dve shranjeni vozlišči. Usmerjeno povezavo z oznako *knows* ustvarimo s *CREATE a-[r:knows]->b* [8].

4.5.2. SET

Lastnosti vozlišč in relacij spreminjamo s stavkom SET. Če želimo spremeniti vrednost atributa 'ime' vozlišča n v 'Jure', potem naredimo to z uporabo stavka *SET n.name = 'Jure'*. Če tak atribut v vozlišču ne obstaja, isti stavek samo doda nov atribut in vrednost v vozlišče. S stavkom SET lahko tudi kopiramo lastnosti elementov med seboj. Lahko pa tudi brišemo lastnosti tako, da nastavimo vrednosti atributa na 'null' (*SET n.name = null*) [8].

4.5.3. DELETE

S stavkom DELETE odstranjujemo vozlišča, relacije in lastnosti elementov. Pri odstranjevanju elementov moramo biti zelo pazljivi. Če hočemo odstraniti element ali lastnost, ki ne obstaja, baza vrne napako. Prav tako ne moremo izbrisati vozlišč, ki so še povezana. Če želimo izbrisati vozlišče, ki je povezano z drugimi vozlišči, moramo najprej odstraniti vse povezave s tem vozliščem. Vse povezave z vozliščem z indeksom 72 odstranimo s stavkom *START a = node(72) MATCH a-[r]-() DELETE a,r* [8].

4.6. FUNKCIJE

4.6.1. PREDIKATNE FUNKCIJE

Predikatne funkcije spadajo med logične funkcije, ker vračajo vrednost true ali false. Uporablja se jih kot del stavka WHERE, za filtriranje poizvedb. Predikatne funkcije so povzeto po [8]:

- funkcija ALL, katera preizkuša, če predikat drži za vse elemente neke zbirke,
- funkcija ANY,ki preveri če predikat drži vsaj za en element zbirke,
- funkcija NONE, ki vrača vrednost true če predikat ne drži za vse elemente zbirke,
- funkcija SINGLE, ki vrača vrednost true če predikat drži za natančno en element zbirke.

Stavek WHERE all (x in nodes(p) WHERE x.age < 27) na primer vrača vsa vozlišča, ki imajo lastnost age manjšo od 27.

4.6.2. SKALARNE FUNKCIJE

Te funkcije vedno vračajo le eno vrednost. Vsako izmed njih lahko postavimo v stavek RETURN. Ena izmed bolj uporabljenih je funkcija LENGTH. Obstaja v mnogo programskih jezikih in vrača dolžino niza, ali število elementov, ki jih vsebuje množica ... V primeru jezika Cypher, vrača dolžino poti rezultata poizvedbe (primer: *RETURN length(p)*) [8].

4.6.3. FUNKCIJE ZA DELO Z NIZI

Te funkcije uporabljamo samo za delo z nizi. Ob morebitnem vhodu katere druge vrednosti vrnejo napako. Izjema je funkcija STR, ki pretvarja vrednosti izrazov v nize. Kot vse preostale funkcije tudi te postavlja v stavek RETURN. Funkcijo REPLACE lahko uporabimo za vračanje niza, v katerem funkcija zamenjala vse pojavitve niza (ali znaka), ki ga iščemo z nekim novim nizom ali znakom. Primer: *RETURN replace "Sonce sije","s","l'*. Izraz bo vrnil vrednost "Sonce lije".

Med funkcije za delo z nizi, spadajo povzeto po [8]:

- funkcija TRIM, ki odstranjuje presledke na obeh straneh nizov,
- funkcija LOWER, ki pretvarja črke niza v velike tiskanke,
- funkcija UPPER, ki pretvarjata črke niza v male tiskanke.

4.7. PRIMERJAVA CYPHER – SQL

Primerjava jezikov Cypher in SQL je namenjena predstavitvi razlik v strukturi poizvedb med njima. Te opišemo v naslednjih nekaj poglavjih. Poleg prikažemo še nekaj primerjav poizvedb obeh jezikov, ki vračajo enak rezultat.

4.7.1. ZAČETNI STAVEK

Vsaka poizvedba ima začetni stavek. jeziku Cypher je to START, ki označuje eno ali več začetnih točk poizvedbe, kjer se ta začne izvajati. Te točke predstavljajo vozlišča ali relacije. Njegov pomen je drugačen od stavka SELECT v SQL. Z stavkom SELECT najprej izberemo podatke nad katerimi se bo izvajala poizvedba. Iz tega sledi, da sta stavka pomensko različna in ju ne moremo neposredno primerjati [8].

Tabela 1: Primerjava poizvedbe Cypher – SQL.

Cypher poizvedba:	SQL poizvedba:
START oseba=node:Oseba(ime = 'Matej') *RETURN oseba*	*SELECT ** *FROM oseba* *WHERE ime = 'Matej';*

Tabela 1 prikazuje primer poizvedbe v obeh jezikih, ki vrne enak rezultat. V obeh primerih vrneta podatke o osebi z imenom Matej.

4.7.2. MATCH – JOIN

Stavek Match v jeziku Cypher specificira obliko vzorca podatkov, ki jih iščemo. Primerjamo ga lahko z operacijo stika v SQL. Povezava (a)-->(b) v stavku MATCH predstavlja stično operacijo (iz SQL) med vozliščema a in b. Razlikujeta se v tem, da MATCH deluje nad par elementi, medtem ko join združuje podatke tabel. [8].

Stik predstavlja operacijo nad tabelami v relacijski podatkovni bazi, kjer združimo dve ali več obstoječih tabeli v novo tabelo. Z vidika učinkovitosti predstavlja stik problematično operacijo, saj se s količino podatkov učinkovitost te operacije znižuje. , zaradi tega ker zmanjšuše učinkovitosti poizvedb v relacijskih podatkovnih bazah [1].

Tabela 2: Primerjava poizvedbe Cypher – SQL.

CYPHER poizvedba	SQL poizvedba
START oseba=node:Oseba(ime = 'Matej') *MATCH oseba-[:eposta]->eposta* *RETURN eposta*	*SELECT eposta.** *FROM oseba* *JOIN eposta ON oseba.id = eposta.oseba_id;* *WHERE oseba.ime = 'Matej';*

Poizvedbi, ki sta navedeni v tabeli 2, vrneta vse elektronske naslove, ki jih uporablja oseba z imenom Matej. Poizvedba Cypher poišče vozlišče z imenom Matej ter pogleda, kateri elektronski naslovi so z njim povezani, medtem ko SQL naredi stik dveh tabel, iz katerih potem izlušči ustrezne podatke.

4.7.3. WHERE

Koncept stavka WHERE je podoben v obeh poizvedovalnih jezikih. V obeh jezik se izvede filtriranje nad podatki. V primeru jezika Cypher se to izvede nad podgrafih, v SQL pa nad množico podatkov v tabeli. [8]. Razliko med stavkoma WHERE v obeh jezikih prikazuje Tabela 3.

Tabela 3: Primerjava poizvedbe Cypher – SQL.

CYPHER poizvedba:	SQL poizvedba:
*START oseba=node:Oseba('ime: *')* *WHERE oseba.starost > 18 AND oseba.oci = 'modre'* *RETURN oseba*	*SELECT ** *FROM oseba* *WHERE oseba.starost > 18 AND oseba.oci = 'modre';*

4.7.4. VRAČANJE VREDNOSTI

S temi stavki poizvedba vrača rezultat. Stavek RETURN, jezika Cypher, je po funkciji primerljiv s stavkom SELECT jezika SQL. Za razliko od SQL, je pri jeziku Cypher RETURN postavljen na konec poizvedbe. Spodnja stavka prikazujeta razliko stavkov v obeh jezikih, ki vračata isti rezultat (imena oseb, ki so shranjene v podatkovni bazi).

- Cypher: *RETURN osebe.ime*
- SQL: *SELECT 'osebe'.'ime'*

5. PRIMERJAVA UČINKOVITOSTI RELACIJSKE PODATKOVNE BAZE IN PODATKOVNE BAZE GRAFOV

V tem poglavju bomo primerjali učinkovitost podatkovne baze grafov Neo4j in relacijske podatkovne baze MySQL. Predstavili bomo orodja, ki smo jih uporabljali skozi testiranje, ter pripravo testnih podatkov. Opisali bomo tudi potek in povzetek ugotovitev testiranja.

5.1. DODATEK GEOFF

Če želimo preveriti delovanje določene podatkovne baze moramo imeti večje količine podatkov, s katerimi izvajamo teste. Problem nastane, če podatkov v bazo ne zmoremo vnesti, kajti ročno vpisovanje velikega števila podatkov je zamudno. Temu se lahko izognemo z določenimi dodatki, kot je na primer Geoff, ki je bil ustvarjen za podatkovno bazo Neo4j.

Geoff je deklarativna notacija s katero lahko predstavimo podatke v boljberljivi obliki. Format je bil zgrajen za oblikovanje neodvisnih podgrafov, ki so predstavljeni izven okolja podatkovne baze Neo4j. V taki obliki omogoča shranjevanje podatkov ter njihov prenos ali uvoz. Notacija Geoff temelji na formatu notacije JSON [13].

Za namestitev dodatka Geoff, moramo s spleta prenesti datoteki *geoff-core.jar* in *geoff-plugin.jar* [14].

5.2. XAMPP

Poglavje je namenjeno opisu orodja XAMPP, ker vsebuje relacijsko podatkovno bazo MySQL, katero bomo v nadaljevanju primerjali z podatkovno bazo grafov Neo4j.

Programsko orodje XAMPP temelji na distribuciji Apache. Poleg manjših orodij, ki se jih uporablja pri razvijanju spletnih aplikacij, sta njegovi glavni komponenti strežnik Apache in podatkovna baza MySQL.Vsebuje spletni uporabniški vmesnik preko katerega se vrši nadzor nad komponentami orodja, na primer nad podatkovno bazo MySQL. Poleg spletnega vmesnika ima še ukazno lupino, katera izvaja ukaze nad orodjem XAMPP [27].

Na voljo so štiri distribucije orodja XAMPP in sicer za sisteme Linux, Mac OS, Windows ter Solaris. Za vsako distribucijo lahko na spletni strani orodja izvemo, na katerih predstavnikih posameznega operacijskega sistema je bila testirana in katere module vsebuje [27].

Ko se program zažene, se odpre nadzorno okno nad komponentami orodja. V njem zaženemo modula Apache in MySQL. Strežnik Apache uporablja vrata 80, ki se jih poslužuje še veliko drugih programov. Če so vrata zasedena, strežnika ne bomo uspeli zagnati. Težavo rešimo s spremembo vrat, ki jih uporablja Apache, kar storimo z urejanjem konfiguracijskih datotek strežnika Apache [27].

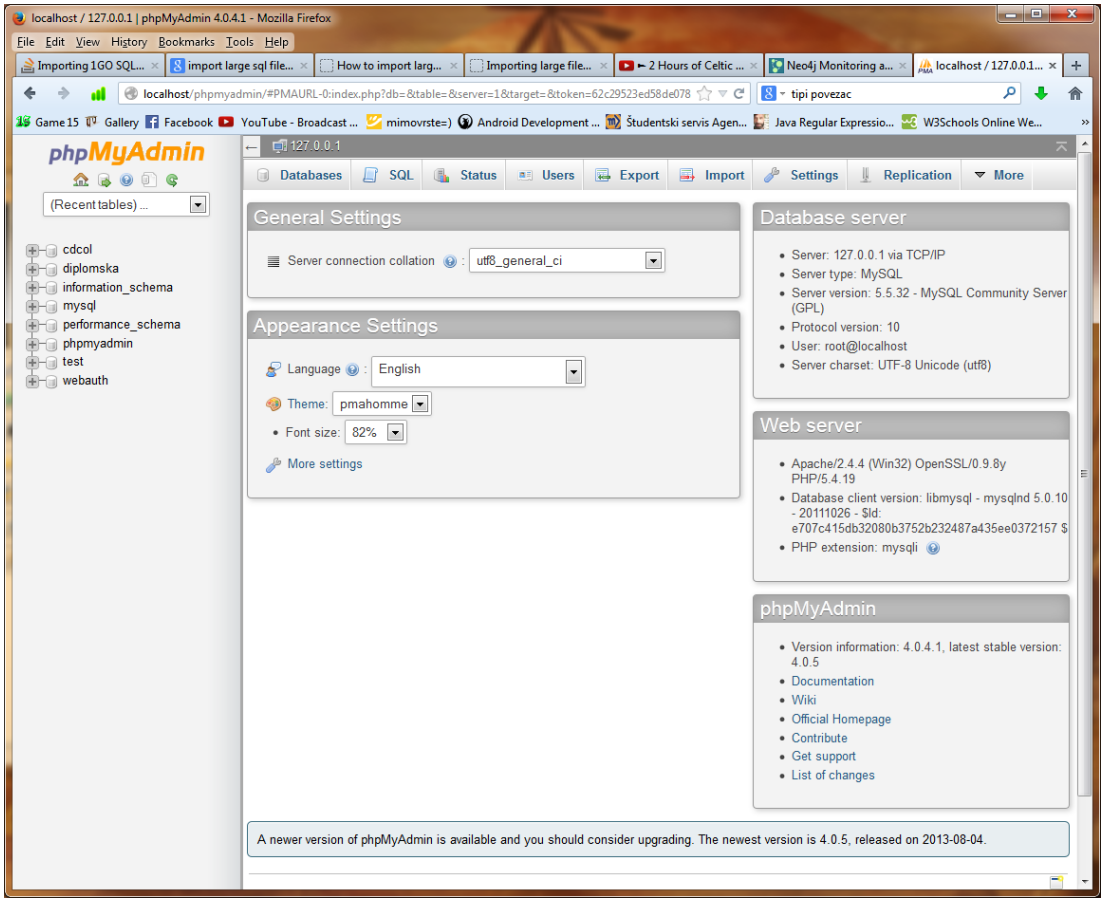
Ko je strežnik zagnan, v brskalnik vnesemo naslov *http://localhost:80/phpmyadmin/* in odpre se spletni uporabniški vmesnik orodja phpMyAdmin s katerim lahko urejamo podatkovno bazo MySQL.

5.2.1. SPLETNI UPORABNIŠKI VMESNIK

Preko spletnega vmesnika dostopamo do orodja phpMyAdmin, ki omogoča delo s podatkovno bazo MySQL. Na levi strani vmesnika se izpišejo imena vseh podatkovnih baz, ki se nahajajo na MySQL strežniku. Na vrhu se nahaja meni, preko katerega izberemo ukaze, ki jih bomo izvajali nad izbrano podatkovno bazo:

- ukazi za izvajanje poizvedb,
- ukazi za uvoz podatkov,
- ukazi za izvoz podatkov
- ukaze za urejanje podatkovne strukture

Poleg se izpišejo informacije o podatkovni bazi MySQL in strežniku Apache. Slika 7 prikazuje spletni uporabniški vmesnik orodja phpMyAdmin.

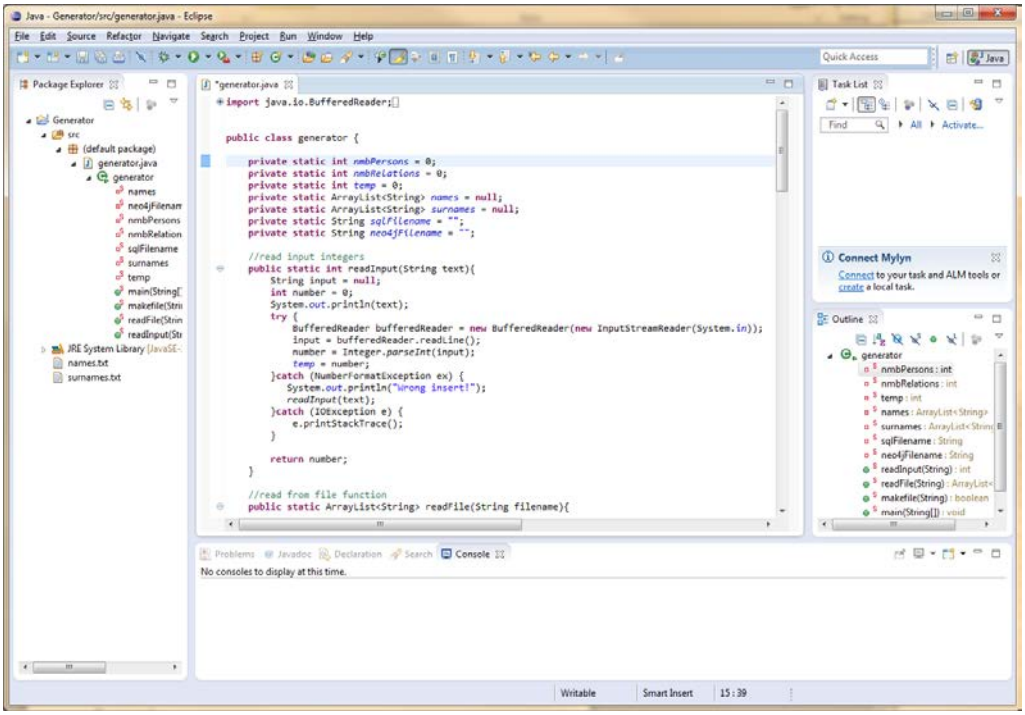


Slika 7: Spletni uporabniški vmesnik orodja phpMyAdmin.

5.3. PRIPRAVA TESTNIH PODATKOV

3.4.5. GENERATOR TESTNIH DATOTEK

S programskim jezikom Java in orodjem Eclipse, ki nam ga prikazuje Slika 8 [12], smo sprogramirali generator naključnih podatkov. Z njim smo ustvarili datoteke z različnim številom podatkov, ki smo jih vnašali v posamezno podatkovno bazo.



Slika 8: IDE Eclipse.

Opis delovanja generatorja in njegova izvorna koda se nahajata v prilogi Generator testnih datotek. Za testiranje obeh podatkovnih baz je bilo potrebno pripraviti različne množice podatkov, nad katerimi so se izvajale poizvedbe. Najlažji način za uvažanje velike količine podatkov v podatkovno bazo je vnos vnaprej pripravljene datoteke s podatki.

5.3.1. UVOZ PODATKOV PRI NEO4J

Pri podatkovni bazi Neo4j smo za uvoz podatkov uporabili programski dodatek Geoff.

Pred uvozom podatkov v graf podatkovne baze Neo4j, je bilo treba dodati v lupino Neo4j ukaze, ki dovoljujejo vnašanje podatkov v delujočo bazo. S spleta smo prenesli datoteko z ukazi za lupino in jo razširili v knjižnico ukazov strežnika [20]. Nato smo lahko žačeli uvažati podatke. Datoteke smo uvažali s preprostim ukazom *import-geoff -g input.geoff*.

Primer vnosa:

neo4j-sh (0)$ import-geoff -g 1000p5000r.geoff
==> Geoff import of 1000p0r.geoff created 1000 entities

V tem primeru smo vnesli datoteko, ki je vsebovala 1000 vozlišč z lastnostmi in 5000 povezav med njimi. Po potrditvi ukaza lupina izpiše, da je bilo ob uvozu datoteke ustvarjenih 1000 novih entitet, kar pomeni 1000 novih vozlišč v grafu.

5.3.2. UVOZ PODATKOV PRI MYSQL

Pri MySQL lahko na enostaven način uvažamo podatke preko spletnega uporabniškega vmesnika. Njegova slabost je, da ne dopušča uvoza datotek večjih od 2 MB. Zato smo datoteke s podatki v MySQL uvažali preko ukazne lupine XAMPP, kjer nismo omejeni z velikostjo datotek. Za vnos smo uporabili ukaz *mysql -u root -p databaseName < path/file.sql*.

5.4. TESTIRANJE

Najprej je bilo potrebno opredeliti strukturo podatkov, nad katerimi se bodo izvajale testne poizvedbe. Osnovno strukturo smo povzeli kar iz socialnih omrežij, kjer neka oseba z imenom ter priimkom pozna več drugih oseb. Večje število podatkov smo generirali z lastnim generatorjem (poglavje 5.3). Ker se strukturi obeh podatkovnih baz razlikujejo, so tudi podatki v njiju različno shranjeni. Podatke o tem, katere osebe pozna naključna oseba, se v grafu Neo4j shranjuje v povezave med vozlišči; pri MySQL pa smo morali ustvariti novo tabelo, ki je vsebovala identifikator osebe v enem stolpcu, v drugem pa identifikator osebe, ki jo ta oseba pozna.

Še preden smo se lotili testiranja, smo morali napisati poizvedbe, ki jih bomo potem izvajali nad podatki. Poizvedbe nad podatkovno bazo Neo4j so bile napisane v jeziku Cypher, za podatkovno bazo MySQL pa v jeziku SQL. Ko so bile poizvedbe napisane smo jih testirali na manjši reprezativni množici podatkov, da smo preverili pravilnost izpisa rezultata.

Preden smo začeli izvajati poizvedbe nad podatkovno bazo MySQL smo morali vnesti še indekse nad tabele, da so se lahko poizvedbe hitreje in bolj učinkovito izvajale. Indekse smo vnesli nad stolpca idFriend in idPerson v tabeli povezav z ukazom [10]:

```
CREATE INDEX PIndex
ON Persons (idFriend, idPerson);
```

Poizvedbe smo vedno izmenično poganjali na Neo4j in MySQL, nikoli pa na obeh bazah hkrati, da ne bi vplivali na čas izvajanja poizvedb. Šele ko je ena poizvedba vrnila rezultat, smo pognali poizvedbo na drugi podatkovni bazi. Vsako poizvedbo smopetkrat ponovili in v tabelo časov izvajanja vnesli njihovo povprečje. Vse poizvedbe nad podatki smo izvajali preko spletnega uporabniškega vmesnika.

Pri testiranju smo primerjali trajanje izvajanja poizvedbe v odvisnosti od količine podatkov, ki so bili vnešeni za posamezno podatkovno bazo. Privzeli smo dva različna primera uporabe in za njiju ustvarjali poizvedbe. V prvem primeru, nam je poizvedba morala vrniti vse tiste osebe, katerih ime in priimek se začneta na določeno črko (glej zgornji poizvedbi v tabeli Tabela 4). Za drugi primer pa je bilo potrebno izpisati tiste osebe, ki jih neka oseba pozna (glej poizvedbi v tabeli Tabela 4).

Tabela 4: Primera izvajanih poizvedb nad podatki v Neo4j in MySQL.

Cypher poizvedbi:	SQL poizvedbi:
START n=node(*) WHERE n.name =~ 'J.*' AND n.surname =~ 'R.*' RETURN n.name, n.surname	SELECT persons.name, persons.surname FROM persons WHERE persons.name LIKE 'J%' AND persons.surname LIKE 'R%'
START n=node(*) MATCH (n)-->(x) WHERE n.name="Damon" and n.surname='Twiet' RETURN x	SELECT persons.name, persons.surname FROM persons WHERE persons.id IN (SELECT relations.idFriend FROM relations WHERE relations.idPerson = (SELECT id FROM persons WHERE name = 'Damon' AND surname = 'Twiet'))

Ko sta bili poizvedbi izvršeni in časi izvajanja vnešeni v tabelo, je bilo potrebno izprazniti obe podatkovni bazi in vnesti druge podatke ter nad njimi spet izvesti zgornji poizvedbi. Pri brisanju iz podatkovne baze Neo4j smo morali paziti, da smo najprej izbrisali povezave med vozlišči. Za brisanje povezav smo uporabili naslednjo poizvedbo *start r=relation(*) delete r*. Nato smo s *start r=node(*) delete r i*zbrisali še vozlišča in njihove lastnosti. Za izbris podatkov iz podatkovne baze MySQL, smo morali izprazniti vsako tabelo posebej. Tabele smo izpraznili z ukazom *DELETE FROM imeTabele*.

5.5. POVZETEK TESTOV

Pri prvem paru poizvedb, ki smo ju izvajali nad podatkovnima bazama, smo primerjali čase trajanja posamezne poizvedbe. Poganjali smo ju nad različnim številom oseb v posamezni podatkovni bazi. Začeli smo pri številu 1000 oseb, nato pa v šestih korakih to vrednost povečevali (Tabela 5).

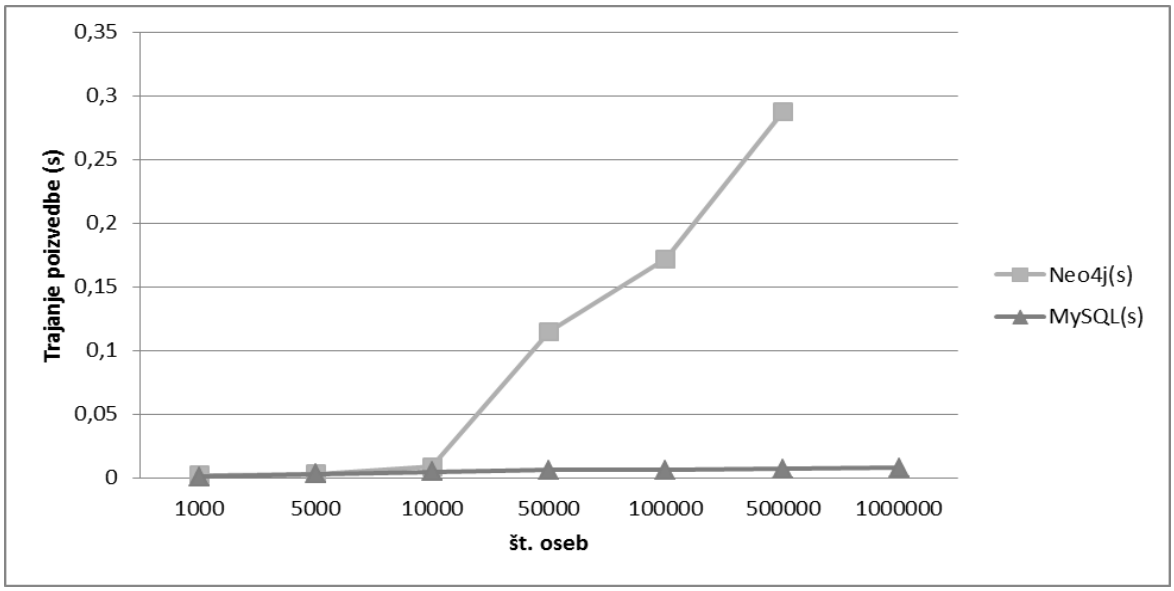
Želeli smo opraviti teste za še večje število oseb in povezav v obeh bazah, vendar smo naleteli na omejitve pri uvozu datoteke s podatki v podatkovno bazo Neo4j. Preko lupine se vnos datoteke z 1 000 000 osebami in 5 000 000 povezavami ni izvedel do konca. Poizvedba se je izvajala, dokler ni prišlo do prekinitve povezave s strežnikom (povezava s strežnikom se je morda prekinila zaradi predolgega izvajanja). Pri pregledu sistemskih sredstev operacijskega sistema smo opazili, da je Java med izvajanjem uporabljala več kot 2GB pomnilnika ter zasedla večino zmogljivosti procesorja.

Graf na sliki Slika 8 prikazuje primerjavo med časi izvajanja poizvedb nad različnim številom oseb. Razlika med časi, ki ga potrebujeta ena in druga poizvedba do 10000 oseb je minimalna. Nato začne čas trajanja poizvedbe pri Neo4j naraščati, medtem ko je ta pri poizvedbah pri MySQL skoraj konstanten tekom vseh količin podatkov. Z naraščajočo količino podatkov bi se verjetno razlika v času trajanja poizvedb med obema podatkovnima bazama povečevala.

Zadnja poizvedba pri Neo4j je označena z zvezdico, ker nam datoteke s podatki nismo uspeli uvoziti.

Tabela 5: Časi izvajanja poizvedb nad osebami.

Število oseb:	Neo4j (s)	MySQL (s)
1000	0.002	0.001
5000	0.003	0.003
10000	0.009	0.005
50000	0.115	0.006
100000	0.172	0.006
500000	0.287	0.007
1000000	*	0.008



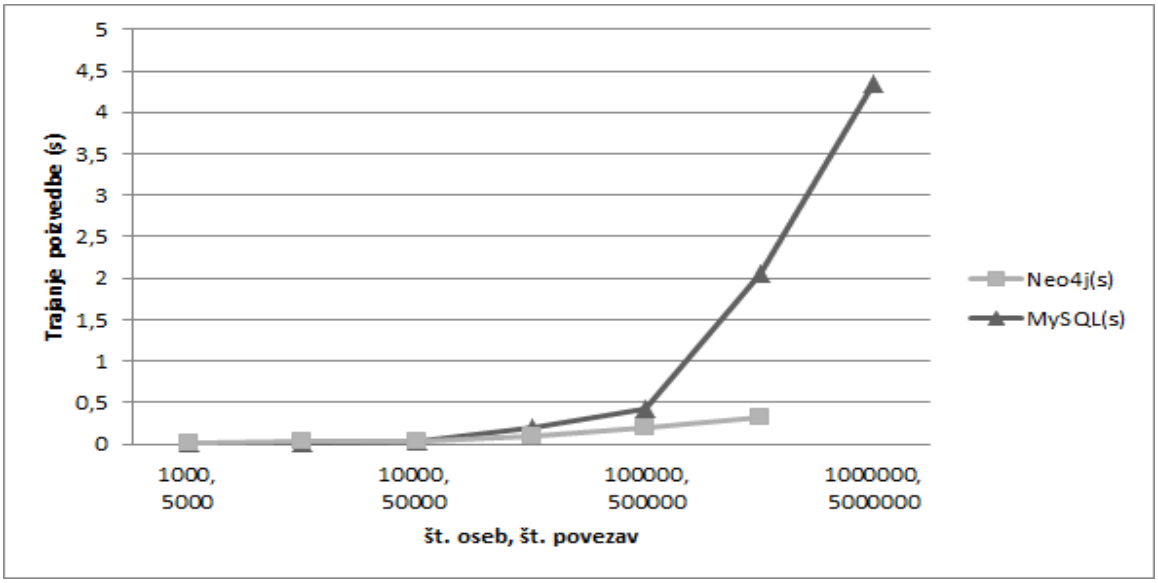
Slika 8: Graf primerjave časov izvajanja poizvedb nad osebami.

Pri naslednjih poizvedbah smo uporabili še povezave (relacije) med osebami. V podatkovni bazi MySQL so bile te povezave shranjene v dodatni tabeli, v podatkovni bazi Neo4j pa v povezavah med vozlišči. Cilj poizvedb je bil poiskati v bazi vse 'prijatelje' oziroma vse tiste osebe, ki jih neka oseba pozna. Poizvedbe smo začeli izvajati nad 1000 osebami, med katerimi je bilo 5000 povezav. Za vsako naslednjo izvajanje poizvedbe, smo obe količini povečevali, kar prikazuje Tabela 6.

Zadnja poizvedba pri Neo4j je označena z zvezdico, ker nam datoteke s podatki ni uspelo uvoziti.

Tabela 6: Časi izvajanja poizvedb nad osebami in relacijami.

Število oseb:	Število povezav:	Neo4j (s)	MySQL (s)
1000	5000	0.003	0.005
5000	25000	0.031	0.018
10000	50000	0.036	0.042
50000	250000	0.103	0.203
100000	500000	0.187	0.447
500000	2500000	0.324	2.048
1000000	5000000	*	4.345



Slika 9: Graf primerjave časov izvajanja poizvedb nad osebami in relacijami.

5.6. UGOTOVITVE

Po končanih meritvah in primerjavi časov izvajanja poizvedb, smo lahko na podlagi le teh prišli do ugotovitev, da sta podatkovni bazi različni in da ni bistvenega pomena čas, ki ga je potrebovala posamezna poizvedba za izvajanje, temveč kaj lahko iz teh meritev sklepamo.

Graf na sliki Slika 8, kjer primerjamo čase izvajanja poizvedb nad nepovezanimi podatki, kaže, da sta si tu podatkovni bazi blizu. Obe sta vračali rezultat v nekaj milisekundah ali desetinkah sekunde. Menimo, da je v prednosti podatkovna baza MySQL, ker je struktura tabele v katero shranjujemo podatke preprostejša od strukture grafa, kar je verjetno razlog za krajši čas izvajanja poizvedb nad bazo MySQL.

Graf, ki ga prikazuje Slika 9, prikazuje zaostanek podatkovne baze MySQL za Neo4j, ki pa se z večanjem števila podatkov povečuje skozi celoten graf. Vzrok za zaostanek, je uporaba stične operacije v poizvedbi SQL, s katero smo povezali tabeli med seboj. Potek grafa, ki prikazuje čas izvajanja poizvedb nad podatkovno bazo Neo4j, bi bil bolj jasen, če bi uspelo uvoziti datoteko za zadnjo meritev izvajanja poizvedbe, ki je vsebovala največ podatkov. Iz naraščajoče oblike grafa lahko sklepamo, da bi bil čas trajanja zadnje poizvedbe pri Neo4j krajši, kot čas, ki ga je dosegla podatkovna baza MySQL.

Pregled grafov pokaže, da poizvedbe nad povezanimi podatki hitreje izvaja Neo4j. Podatkovna baza MySQL pa je bolj učinkovita, ko imamo v obdelavi nepovezane podatke.

Poleg strukture in načina izvajanja poizvedb nad podatki, ima vpliv na izvajanje tudi zmogljivost sistema, kjer teče podatkovna baza. Odločilno vlogo pri tem pa igra tudi hitrost trdega diska ali drugega medija, kjer so ti podatki shranjeni.

6. ZAKLJUČEK

V današnjem času je shranjevanje podatkov in pridobivanje informacij iz njih zelo pomembno področje računalništva, kar je povzročilo razvoj različnih načinov shranjevanja podatkov. Izbor podatkovne baze, ki bi bila za nas najbolj primerna, mora biti osnovana na podlagi namena, za katerega jo bomo uporabili. Morda jo potrebujemo samo za shranjevanje podatkov, morda bomo nad njimi izvajali veliko število transakcij. Lahko se bo spreminjala shema podatkov, podatki, ki jih bomo shranjevali bodo povezani ali nepovezani... Vsaka izbira mora biti osnovana na podlagi prednosti in pomanjkljivosti posameznega sistema za določen namen uporabe.

Nekatere izmed podatkovnih baz so bile razvite z določenim namenom, ki ustreza področju uporabe. Tako so bile na primer podatkovne baze, ki temeljijo na grafih, razvite za delo na področjih, kjer se srečamo z povezanimi podatki (primer: socialna omrežja).

V tej diplomski nalogi smo preučili podatkovno bazo Neo4j, ki predstavlja primer sistema, ki shranjuje podatke v grafih. Spoznali smo podatkovno strukturo, ki jo uporablja ta podatkovna baza. Seznanili smo se tudi z uporabniškimi vmesniki za delo z njo.

Spoznali smo programsko orodje Neoclipse, ki so ga razvili za uporabo s podatkovno bazo Neo4j. Z njim smo se povezali na podatkovno bazo Neo4 in preizkusili njegove fukcionalnosti ter zmožnosti.

Analizirali smo jezik Cypher, ki ga uporabljamo za pisanje poizvedb nad podatki v podatkovni bazi Neo4j. Njegovo strukturo smo razčlenili in opisali posamezne elemente. Ker smo primerjali podatkovni bazi Neo4j in MySQL, smo naredili tudi krajšo primerjavo med jezikoma Cypher in SQL.

V praktičnem delu smo med seboj primerjali dve različni podatkovni bazi. Primerjali smo Neo4j, ki je primer nerelacijske podatkovne baze in MySQL, ki spada med relacijske podatkovne baze. Odločili smo se za primerjavo glede na hitrost izvajanja poizvedb nad podatki v posamezni podatkovni bazi. S pomočjo programskega jezika Java in z uporabo orodja Eclipse, smo napisali program za generiranje različnih količin naključnih podatkov, ki smo jih nato vnašali v podatkovni bazi. Nad podatki smo izvajali različne poizvedbe, čase izvajanja pa smo vpisovali v tabele in jih nato grafično primerjali med seboj.

Za primerjavo smo uporabili dve poizvedbi, ki predstavljata najbolj pogosto uporabljene primere (poizvedba nad nepovezanimi, ter nad povezanimi podatki). V prvem primeru poizvedbe smo izpisovali podatke o osebi, v drugem pa osebe, ki jih poljubno izbrana oseba pozna.

Pri iskanju osebe in podatkov o njej, se je podatkovna baza Neo4j izkazala za malenkost počasnejšo kot MySQL. Sklepamo, da je za to kriv način shranjevanja podatkov v bazi Neo4j. Podatkovna baza Neo4j namreč podatke shranjuje v graf, ki je kompleksnejša struktura kot tabela. Pri drugi poizvedbi, kjer smo iskali osebe, ki jih določena oseba pozna, je bila hitrejša podatkovna baza Neo4j. Sklepamo, da so rezultati za MySQL slabši, ker je morala narediti stik dveh tabel (podatki so bili shranjeni v dveh tabelah) in je baza zato potrebovala več časa za procesiranje poizvedbe.

Izpostavili bi, da spletni uporabniški vmesnik Neo4j vsebuje zelo malo orodij ter možnosti za delo s podatkovno bazo Neo4j in je uporabniku neprijazen. Razvoj uporabniku bolj prijaznega

spletnega vmesnika predstavlja možnost za nadaljni razvoj. Namen uporabniškega vmesnika je namreč, da skrije izvajanje v ozadju in s tem razbremeni uporabnika. Tako lažje pritegnemo večje število uporabnikov, ki bi uporabljali nek sistem (recimo podatkovno bazo Neo4j).

V prihodnosti se bomo zagotovo še srečali s spremembami pri obeh predstavnicah podatkovnih baz. Čeprav trenutno prevladujejo relacijske podatkovne baze, lahko pričakujemo, da bodo tudi predstavnice podatkovnih baz tipa NoSQL (prav tako Neo4j) pridobile na pogostosti uporabe na področju informacijske tehnologije.

Literatura

[1] T. Connolly, C. Begg, »Database Systems: A practical Approach to Design, Implementation, and Management (Fourth Edition)«, Pearson Education Limited, Harlow, 2005.

[2] H. Darwen, »An Introduction to Relational Database Theory«. Dostopno na: http://bookboon.com/en/an-introduction-to-relational-database-theory-ebook. Hugh Darwen && bookboon.com (Ventus Publishnih ApS), 2012.

[3] F. Holzschuher, R. Peinl, »Performance of Graph Query Languages Institute for Information Systems«. Dostopno na: http://www.edbt.org/Proceedings/2013-Genova/papers/workshops/a29-holzschuher.pdf, Institute for Information Systems (iisys), 2013Institute for Information Systems (iisys), Hof, 2013.

[4] I. McGraw-Hill, »Database Managemet Systems Designing and Building Business Aplications«, The McGraw-Hill Companies Inc, New York, 2005.

[5] T. Mohorič, »Podatkovne baze«, © BI-TIM d.o.o., Ljubljana 2002.

[6] I. Robinson, J. Webber, E. Eifrem, »Graph Databases«. Dostopno na: http://graphdatabases.com/. Neo Technology, Sebastopol, 2013, pogl.: 1.

[7] S. Sales Harkins, M. W. P. Reid, »SQL: Access to SQL Server«,Copyright by Susan Sales Harkins and Martin W.P.Reid, New York, 2002.

Spletni viri

[8] Cypher Query Language (2013). Dostopno 29.8.2013 na: http://docs.neo4j.org/chunked/stable/cypher-query-lang.html.

[9] Database (2013). Dostopno 29.8.2013 na: http://en.wikipedia.org/wiki/Database.

[10] Database index (2013). Dostopno 29.8.2013 na: http://en.wikipedia.org/wiki/Database_index.

[11] Download and Install Neo4j (2013). Dostopno 29.8.2013 na: http://www.neo4j.org/download.

[12] Eclipse (2013). Dostopno 29.8.2013 na: http://www.eclipse.org/.

[13] Geoff (2013). Dostopno 29.8.2013 na: http://nigelsmall.com/geoff.

[14] Geoff installation (2013). Dostopno 29.8.2013 na: http://nigelsmall.com/geoff/getting-started.

[15] Graph (mathematics) (2013). Dostopno 29.8.2013 na: http://en.wikipedia.org/wiki/Graph_%28mathematics%29.

[16] Graph database (2013). Dostopno 29.8.2013 na: http://en.wikipedia.org/wiki/Graph_database.

[17] Graph Databases, NOSQL and Neo4j (2010). Dostopno 29.8.2013 na: http://www.infoq.com/articles/graph-nosql-neo4j.

[18] Gremlin (2013). Dostopno 29.8.2013 na: http://en.wikipedia.org/wiki/Gremlin_%28programming_language%29.

[19] OpenRemote (2009). Dostopno 29.8.2013 na: http://www.openremote.org/pages/viewpage.action?pageId=2424902.

[20] Neo4j shell tools (2013). Dostopno 29.8.2013 na: https://github.com/jexp/neo4j-shell-tools#import-data-into-your-neo4j-database-from-the-neo4j-shell-command

[21] Neoclipse (2013). Dostopno 29.8.2013 na: https://github.com/neo4j/neoclipse/wiki.

[22] Price list (2013). Dostopno 29.8.2013 na: http://neotechnology.com/products/price-list/.

[23] Securing access to Neo4j Server (2013). Dostopno 29.8.2013 na: http://docs.neo4j.org/chunked/stable/security-server.html.

[24] The Bit Depth Blog (2013). Dostopno 29.8.2013 na: http://bitdepth.thomasrutter.com/2010/04/02/stable-vs-stable-what-stable-means-in-software/.

[25] What is Neo4j? (2013). Dostopno 29.8.2013 na: http://www.neo4j.org/learn/neo4j.

[26] Web Administration (2013). Dostopno 29.8.2013 na: http://docs.neo4j.org/chunked/1.8.2/tools-webadmin.html.

[27] XAMPP (2013). Dostopno 29.8.2013 na: http://www.apachefriends.org/en/xampp.html.

Uporabljeni programi

brskalnik Firefox, verzija: 23

Neo4j, verzija: 1.8.2 Enterprise

XAMPP, verzija: 1.8.2

Neoclipse, verzija: 1.8

Eclipse IDE for Java Developers, verzija: Kepler

Specifikacija sistema

OS Windows 7 64

AMD Phenom x6 1090T

OS disk OCZ Vertex3 60GB

8Gb DDR3 system memory

zunanji Toshiba USB 2 2.5" 300GB HDD (*obe podatkovni bazi sta bili nameščeni na tem disku*)

Priloge

Primeri Cypher in SQL poizvedb

Izpis vseh oseb, ki jih pozna oseba s poljubnim imenom in priimkom (Cypher).

```
START n=node(*)
MATCH (n)-->(x)
WHERE n.name="James" and n.surname='Hummer'
RETURN DISTINCT x
```

Izpis vseh oseb, ki jih pozna oseba s poljubnim imenom in priimkom (SQL).

```
SELECT persons.name, persons.surname
FROM persons
WHERE persons.id
IN (
        SELECT DISTINCT relations.idFriend
        FROM relations
        WHERE relations.idPerson = (
                SELECT id
                FROM persons
                WHERE name LIKE 'James' AND surname LIKE 'Hummer' )
)
```

Izpis oseb, katerih ime in priimek se pričneta s izbrano črko (Cypher).

```
START n=node(*)
WHERE n.name =~ 'J.*' AND n.surname =~ 'R.*'
RETURN n.name, n.surname
```

Izpis oseb, katerih ime in priimek se pričneta s izbrano črko (SQL).

```
SELECT persons.name, persons.surname
FROM persons
WHERE persons.name LIKE 'J%'
AND persons.surname LIKE 'R%'
```

Poizvedba, s katero izbrišemo vse povezave in vozlišča v Neo4j grafu.

```
START n=node(*)
MATCH n-[r?]-()
WHERE ID(n) <> 0
DELETE n,r
```

Poizvedbi, s katerima lahko izbrišemo vse podatke, ki se nahajajo v tabeli (SQL).

```
DELETE FROM persons;
TRUNCATE persons;
```

Generator testnih datotek

Opis delovanja:

Program prebere števili iz konzole, ki predstavljata število oseb in povezav med njimi. Te bo ustvaril tekom delovanja programa. Nato prebere tekstovni datoteki A in B. Datoteka A vsebuje seznam imen, datoteka B pa seznam priimkov. Program vsebino obeh datotek shrani v posamezna seznama. V enega shrani imena, v drugega pa priimke. Ko so shranjeni podatki imen in priimkov, program ustvari prazni datoteki C in D. V datoteko C shranjuje generirane podatke za podatkovno bazo Neo4j, v datoteko D za podatkovno bazo MySQL. V prvi zanki programa, tolikokrat kot je vnešeno število oseb, izbere ime in priimek iz seznamov, ter ju zapiše ustrezni obliki za jezik Cypher in SQL v datoteki C in D. Naslednja zanka se ponavlja tolikokrat, kot je vnešeno število povezav. Program naključno ustvari povezavo med dvema osebama v intervalu 0, do števila oseb, ki smo jih vnesli. Te povezave po kreiranju shrani v datoteki C in D. Po končanem generiranju podatkov zapre obe datoteki in konča z delovanjem.

Datoteki, ki smo ju označili z črkama A in B smo prenesli s spleta in sicer z naslova *http://www.quietaffiliate.com/free-first-name-and-last-name-databases-csv-and-sql*. S pomočjo programa Microsoft Excel smo njuno obliko pretvorili iz oblike CSV v tekstovno obliko. V obliki CSV so bili podatki ločeni s posebnimi znaki, v pretvorjeni tekstovni obliki pa je bil vsak podatek v posamezni vrstici datoteke.

Izvorna koda generatorja:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Random;

public class generator {

    private static int nmbPersons = 0;
    private static int nmbRelations = 0;
    private static int temp = 0;
    private static ArrayList<String> names = null;
    private static ArrayList<String> surnames = null;
    private static String sqlFilename = "";
    private static String neo4jFilename = "";

    //read input integers
    public static int readInput(String text){
        String input = null;
        int number = 0;
        System.out.println(text);
```

```
                try {
               BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(System.in));
               input = bufferedReader.readLine();
               number = Integer.parseInt(input);
               temp = number;
             }catch (NumberFormatException ex) {
              System.out.println("Wrong insert!");
              readInput(text);
             }catch (IOException e) {
               e.printStackTrace();
             }
                   return number;
          }

          //read from file function
          public static ArrayList<String> readFile(String filename){
                   ArrayList<String> list = new ArrayList<String>();
        try {
           FileInputStream fstream = new FileInputStream(filename);
           DataInputStream in = new DataInputStream(fstream);
           BufferedReader br = new BufferedReader(new InputStreamReader(in));
           String line;

           while ((line = br.readLine()) != null)   {
                    list.add(line);
           }
           in.close();
             }catch (Exception e){//Catch exception if any
                   System.err.println("Error: " + e.getMessage());
             }
                   return list;
          }

          //function create file
          public static boolean makefile(String filename){
                   File f = new File(filename);
                   if ( !f.exists()){
                            try {
                                     f.createNewFile();
                                     return true;
                            } catch (IOException e) {
                                      System.err.println("Error: " + e.getMessage());
                            }
                   }
                   return false;
          }

          //main function
          public static void main(String[] args) {
```

```
                int counter = 0;
                Random randomNmb = new Random();
                String name = "";
                String surname = "";
                int firstP = 0;
                int secondP = 0;

                //read number of persons
                readInput("Please insert number of people: ");
                nmbPersons = temp;

            //read number of relations
                readInput("Please insert number of relations: ");
                nmbRelations = temp;

                //check if input is correct
                //System.out.println("Number of persons: " + nmbPersons);
                //System.out.println("Number of relations: " + nmbRelations);

                //read names
                names = readFile("names.txt");
                //System.out.println(names.size());

                //read surnames
                surnames = readFile("surnames.txt");
                //System.out.println(surnames.size());

                //create filenames
                sqlFilename = nmbPersons + "p" + nmbRelations + "r.sql";
                neo4jFilename = nmbPersons + "p" + nmbRelations + "r.geoff";

                //create empty files
                //makefile(sqlFilename);
                //makefile(neo4jFilename);

                 try {
                        BufferedWriter outSql = new BufferedWriter(new
FileWriter(sqlFilename));
                        BufferedWriter outNeo = new BufferedWriter(new
FileWriter(neo4jFilename));

                        //sql need some extra data
                        outSql.write("insert into persons(id,name,surname)");
                        outSql.newLine();
                        outSql.write("values ");

                        //generate random names and surnames and write them in both
files
                        for (counter = 0; counter<nmbPersons; counter++){
```

```
                                name = names.get(randomNmb.nextInt(names.size()));
                                surname =
surnames.get(randomNmb.nextInt(surnames.size()));
                                outNeo.write("(" + (counter+1) + ")" + " {" + "\"" +
"name" + "\"" + ": " + "\"" + name + "\"" + "," + "\"" + "surname" + "\"" + ": " +
"\"" + surname + "\"" + "}");
                                outNeo.newLine();
                                outSql.write("("+ (counter+1) + "," + "'" + name  + "'"
+ "," + "'" + surname + "'" + ")");

                                // write ";" on the end of insert
                                if ((counter) == (nmbPersons-1))
                                        outSql.write(";");
                                else
                                        outSql.write(", ");
                        }

                        if (nmbRelations > 0){
                                //blank line between persons and relations
                                outNeo.newLine();
                                outSql.newLine();

                                //sql need some extra data
                                outSql.write("insert into relations(idPerson,idFriend)");
                                outSql.newLine();
                                outSql.write("values ");

                                //generate random relations and write them in both files
                                for (counter = 0; counter<nmbRelations; counter++){
                                        firstP = (randomNmb.nextInt(nmbPersons))+1;
                                        secondP =
(randomNmb.nextInt(nmbPersons))+1;
                                        outNeo.write("(" + (firstP) + ")" + "-
[:KNOWS]->" + "(" + (secondP) + ") {}");
                                        outNeo.newLine();
                                        outSql.write("(" +  firstP + "," + secondP +
")");

                                       // write ";" on the end of insert
                                        if ((counter) == (nmbRelations-1))
                                                outSql.write(";");
                                        else
                                                outSql.write(", ");
                                }
                        }
                        //close both files
                        outSql.close();
                        outNeo.close();
                        System.out.println("Files had been created.");
                } catch (IOException e){
```

```
                    System.out.println("Exception ");
                }
            }
        }
```