Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jure Klančar

Grafne podatkovne baze in vizualizacija grafov

diplomsko delo

univerzitetni študij računalništva in informatike

Mentor: prof. dr. Borut Robič

Ljubljana 2013

Rezultati diplomskega dela so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

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: 01939 / 2013
Datum: 3.9.2013

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

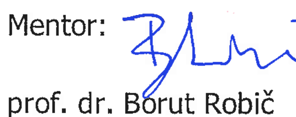
Kandidat: **JURE KLANČAR**

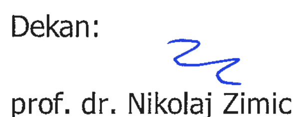
Naslov: **GRAFNE PODATKOVNE BAZE IN VIZUALIZACIJA GRAFOV**
GRAPH DATABASES AND GRAPH VIZUALIZATION

Vrsta naloge: DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Tematika naloge:

Predstavite osnovno o grafnih podatkovnih bazah, njihovo uporabo in prednosti pred relacijskimi podatkovnimi bazami. Razvijte preprosto aplikacijo, ki za shranjevanje podatkov uporablja grafno podatkovno bazo Neo4j.

Mentor:

prof. dr. Borut Robič

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Jure Klančar, z vpisno številko **63070050**, sem avtor diplomskega dela z naslovom:

Grafne podatkovne baze in vizualizacija grafov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Boruta Robiča
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 13. december 2013 Podpis avtorja:

Zahvaljujem se mentorju prof. dr. Borutu Robiču, da me je vzel pod svoje mentorstvo. Posebej bi se rad zahvalil asistentu dr. Urošu Čibeju, ki mi je s svoji hitro odzivnostjo in dobrimi zamislimi zelo pomagal pri pisanju diplomske naloge. Zahvalil bi se tudi staršema, ki sta me tekom študija vseskozi podpirala. Nazadnje bi se rad zahvalil še Evi, za strpnost v času pisanja diplomske naloge, ter pomoč pri lektoriranju.

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Podatkovne baze **3**

2.1 Kaj so podatkovne baze? 3
2.2 Kratka zgodovina 4
2.3 NoSQL 5
2.4 ACID in BASE principa 5
2.5 Agregirane shrambe 7
2.6 Dokumentne shrambe 7
2.7 Ključ-vrednost shrambe (KV) 8
2.8 Shrambe družin stolpcev 9

3 Grafne podatkovne baze **11**

3.1 Kaj je graf? 11
3.2 Grafne podatkovne baze 12
3.3 Prednosti grafnih podatkovnih baz 15
3.4 Razmerja v grafnih podatkovnih bazah 16
3.5 Neo4j 17

4 Aplikacija **25**

4.1 Postavitev podatkovne baze 25

4.2 Splošno o aplikaciji 28
4.3 Opis vidnega dela aplikacije 29
4.4 Pomembnejši problemi 35
4.5 Kratek pregled orodij za vizualizacijo grafov 36

5 Zaključek **39**
5.1 Nadaljne delo 40

Povzetek

V diplomski nalogi so predstavljene grafne podatkovne baze. Grafne podatkovne baze spadajo v skupino NoSQL podatkovnih baz, zato smo predstavili tudi osnove NoSQL podatkovnih baz. Osredotočili smo se na prednosti grafnih podatkovnih baz v primerjavi z relacijskimi. Podrobno delovanje grafnih podatkovnih baz smo si ogledali na eni izmed avtohtonih grafnih podatkovnih baz (Neo4j). Da bi delovanje grafnih podatkovnih baz bolje spoznali, smo razvili še preprosto aplikacijo, ki za shranjevanje podatkov uporablja Neo4j podatkovno bazo. Naši podatki so transakcije med javno upravo Slovenije za obdobje med letoma 2003 in 2013. Transakcije je mogoče lepo predstaviti z grafom, zato naša aplikacija omogoča vizualizacijo podatkov v obliki grafa. Aplikacija omogoča tudi filtriranje podatkov in ima sposobnost risanja vozlišč različnih velikosti v odvisnosti od izbranega atributa.

Ključne besede: grafne podatkovne baze, vizualizacija grafov, Neo4j grafna podatkovna baza

Abstract

The thesis presents graph databases. Graph databases are a part of NoSQL databases, which is why this thesis presents basics of NoSQL databases as well. We have focused on advantages of graph databases compared to relational databases. We have used one of native graph databases (Neo4j), to present more detailed processing of graph databases. To get more acquainted with graph databases and its principles, we developed a simple application that uses a Neo4j graph database to store its data. Our data are transactions between Slovenian public administration for a period from year 2003 to 2013. Transactions that we used can be nicely presented with a graph, so application visualizes our dataset to a graph. Application has several filters to filter the data and it has the ability to draw nodes of diffenent sizes, which is dependent on specified atribute.

Key words: graph databases, graph visualization, Neo4j graph database

Poglavje 1

Uvod

Komisija za preprečevanje korupcije Slovenije ima na svoji spletni strani spletno aplikacijo, s pomočjo katere si lahko ogledamo vse javne transakcije med javnimi organi in poslovnimi subjekti. Transakcije so prikazane v obliki tabel. Transakcije je mogoče izjemno dobro predstaviti tudi z grafom. Ker je namen te diplomske naloge spoznati grafne podatkovne baze, ki so prilagojene shranjevanju grafov, in ker nas zanima delo z grafiko, smo se odločili, da te podatke prikažemo v obliki grafa.

Grafne podatkovne baze so v zadnjem obdobju doživele svoj razcvet. Razlog za to je dobro obvladovanje podatkov, ki se hitro spreminjajo in so med seboj zelo povezani. Trenutno najbolj razvita oblika podatkovnih baz, relacijska, se s temi podatki težko sooča. Shema relacijskih podatkovnih baz mora biti predhodno podana, kar predstavlja določene težave, če se množica podatkov spremeni. Prav tako je težja sama postavitev podatkovne baze, saj redko kdo že na začetku dobro pozna množico podatkov. Grafne podatkovne baze ne potrebujejo predhodno pripravljene sheme, kar omogoča izjemno lahek razvoj same podatkovne baze. Prav tako nam kakršno koli spreminjanje podatkovne baze ne predstavlja nobene težave. Za razliko od relacijskih so grafne podatkovne baze prilagojene povezanim podatkom. V primeru, ko je množica podatkov izjemno povezana, se hitrost relacijskih podatkovnih baz nikakor ne more primerjati z grafnimi.

Diplomska naloga je razdeljena na dva večja dela. V prvem delu se bomo ukvarjali s teorijo podatkovnih baz, predvsem z grafnimi. Ta del je razdeljen na dve poglavji. V prvem poglavju so predstavljene različne vrste podatkovnih baz, med katerimi bomo največ pozornosti namenili NoSQL podatkovnim bazam, saj mednje sodijo tudi grafne podatkovne baze. V drugem poglavju si bomo pogledali grafne podatkovne baze. Ogledali si bomo tudi avtohtono grafno podatkovno bazo, s katero bomo bolje spoznali osnovno shranjevanje in obdelovanje grafne podatkovne baze. Drugi del je namenjen aplikaciji. V njem si bomo ogledali, kako smo do podatkov prišli, kako smo jih umestili v podatkovno bazo in kako smo jih grafično prikazali.

Poglavje 2

Podatkovne baze

2.1 Kaj so podatkovne baze?

Podatkovne baze so pomemben del vsakega podjetja. V sedanjem času baze uporabljajo vsi, večja in manjša podjetja. Baza se skriva praktično za vsako informacijo, ki jo želimo pridobiti. Kaj pa sploh je podatkovna baza? V bistvu ni podatkovna baza nič drugega kot skupek informacij, ki obstajajo skozi daljša časovna obdobja, ponavadi več let. Na podatkovno bazo lahko gledamo tudi kot na kolekcijo podatkov, s katerimi upravlja sistem za upravljanje podatkovnih baz (angl. Database Management System), ki mu na kratko pravimo DBMS. DBMS naj bi:

- dovoljeval uporabniku ustvariti novo podatkovno bazo in določiti shemo oziroma lastnosti;
- omogočal uporabniku poizvedbe o podatkih (poizvedba je vprašanje o podatkih) in spreminjanje podatkov s primernim povpraševalnim jezikom;
- podpiral shranjevanje ogromnih količin podatkov skozi daljša časovna obdobja in zagotavljal učinkovit dostop do podatkov;
- omogočal vzdržljivost, obnovitev podatkov v primeru, da se podatkovna baza sesuje, bodisi zaradi napak na sami podatkovni bazi bodisi

zaradi namerne zlorabe same podatkovne baze;

- nadziral dostop za več različnih uporabnikov do podatkovne baze istočasno, brez nepričakovanih interakcij med uporabniki (izolacija angl. isolation) in brez delnih operacij nad podatki (atomarnost angl. atomicity).

Primarni cilj diplomske naloge je pobližje spoznati grafne podatkovne baze, zato se v relacijske podatkovne baze ne bomo poglabljali. Lahko pa si več preberete v knjigi [1].

2.2 Kratka zgodovina

Začetki podatkovnih baz segajo v sredino 60-tih let prejšnjega stoletja. V tem času se je pojavilo kar nekaj splošno namenskih podatkovnih baz. Povečevalo se je zanimanje po standardizaciji, zato so znotraj CODASYL (Conference on Data Systems Languages) leta 1971 pripravili standard, ki mu pravimo pristop Codasyl (Codasyl approach). Ta pristop je temeljil na vodeni navigaciji. Zgodnji modeli DBMS-jev niso podpirali visokonivojskega povpraševalnega jezika, tako da je bilo pisanje poizvedb zelo zahtevno.

Leta 1970 je Edgar Codd napisal serijo papirjev, v katerih je predstavil nov pristop za grajenje podatkovnih baz. Iz tega pristopa so se kmalu razvile relacijske podatkovne baze. Za razliko od prejšnjih modelov, ki so imeli podatke shranjene v nekakšnem povezanem seznamu, so bili tu podatki shranjeni v tabeli. Vsaka tabela predstavlja različno entiteto. Poizvedbe v relacijskih podatkovnih bazah so lažje za uporabo. Poizvedbe so lahko izražene v visokonivojskem povpraševalnem jeziku. Eden prvih takih jezikov je bil strukturiran povpraševalni jezik SQL (structured query language). Tako relacijske podatkovne baze kot tudi SQL se uporabljajo še danes.

Po letu 1980 so se začeli razširjati namizni računalniki, z njimi pa tudi prve namizne podatkovne baze. Ena izmed bolj znanih iz tistega časa je dBase, ki je bila lahko razumljiva tudi za navadne uporabnike namiznih računalnikov. V istem času so se z vzponom objektno usmerjenih program-

skih jezikov začele razvijati tudi objektno usmerjene podatkovne baze. Podatki v podatkovnih bazah so se sedaj šteli kot objekti.

Po letu 1998 se je začela razvijati nova generacija podatkovnih baz, imenovana NoSQL (not only SQL), ki vključuje hitro ključ-vrednost (key-value) shranjevanje in dokumentno orientirane podatkovne baze. NoSQL podatkovne baze so navadno zelo hitre. Sem spadajo tudi grafne podatkovne baze. V zadnjem času se pojavljajo tudi moderni relacijski DBMS (Relational DBMS), ki še vedno uporabljajo SQL, a ciljajo na zmogljivost oziroma hitrost NoSQL-a. Tem podatkovnim bazam pravimo NewSQL [2].

2.3 NoSQL

V preteklosti je večina aplikacij delovala na relacijskih podatkovnih bazah. V zadnjem desetletju se je velikost podatkov zelo povečala. Podatki se hitro spreminjajo in tradicionalni RDBMS-ji niso več primerni za shrambo teh podatkov. Začeli so razvijati NoSQL.

Relacijske podatkovne baze z velikim naborom podatkov se izjemno slabo odzivajo pri poizvedbah. Tu ni problem v samih podatkovnih bazah, ampak je težava v podatkovnem modelu, ki zgradi množico vseh možnih odgovorov in šele nato filtrira ven odgovor, ki ga iščemo. NoSQL je prilagojen za rokovanje z izjemno velikimi množicami podatkov, zato se tu veliko bolje obnese kot pa relacijski model. A obseg podatkov ni edini problem, s katerim se soočamo danes. Podatki se spreminjajo izjemno hitro. Hitrost (angl. velocity) je stopnja, po kateri se podatki spreminjajo čez čas. Hitrost ima še en vidik, in sicer spreminjanje strukture podatkov. Obe obliki hitrosti pa sta zelo problematični v relacijskem svetu.

2.4 ACID in BASE principa

V svetu relacijskih podatkovnih baz poznamo ACID transakcije. ACID nam priskrbi varno okolje, v katerem lahko izvajamo operacije na podatkih:

- Atomarnost – Vse operacije v transakciji so uspešne, ali pa se vse operacije razveljavijo.
- Doslednost – Ko se transakcija konča, ostane baza strukturno takšna kot je bila
- Izolacija – Transakcije ne sovpadajo ena z drugo. Za to poskrbi baza in transakcije izgledajo kot da tečejo zaporedno.
- Vzdržljivost – Rezultat transakcij je končen, tudi če pride do napake.

Te lastnosti pomenijo, da so podatki po zaključeni transakciji dosledni. To je dobra novica za razvijalca, saj ni potrebno, da skrbi za doslednost podatkov. Načeloma so ACID transakcije prestroge in so v NoSQL že izginile. Uveljavila se je nova strategija, ki ima manj stroge zahteve za shranjevanje, imenovana BASE:

- Osnovna razpoložljivost – Shramba je videti, kot da deluje večino časa.
- Mehko stanje – Ni potrebno, da je pisanje pisalno dosledno in tudi za različne odzive ni nujno, da so medsebojno dosledni ves čas.
- Morebitna doslednost – Shrambe postanejo dosledne kasneje.

Že na prvi pogled se vidi, da sta ACID in BASE zelo različna. BASE temelji predvsem na razpoložljivosti, a ne zagotavlja doslednosti. Za doslednost mora poskrbeti razvijalec. Ta mora sam presoditi, kaj potrebuje in to zagotoviti na aplikacijskem nivoju.

NoSQL ima več različnih podatkovnih modelov. Tri si bomo na hitro pogledali v tem poglavju. Vsi trije so pripadniki agregiranih shramb. To so dokumentne shrambe, ključ-vrednost (od sedaj naprej bomo te shrambe označevali kot KV) shrambe in shrambe družin stolpcev. Podrobneje pa si bomo ogledali grafne podatkovne baze, in sicer v naslednjem poglavju.

2.5 Agregirane shrambe

V relacijskem svetu shranjujemo podatke v vrstice (angl. tuple). Vrstica je limitirana struktura, saj zajema nabor vrednosti in ne omogoča gnezdenja vrstic. Rezultat operacij je vedno vrstica.

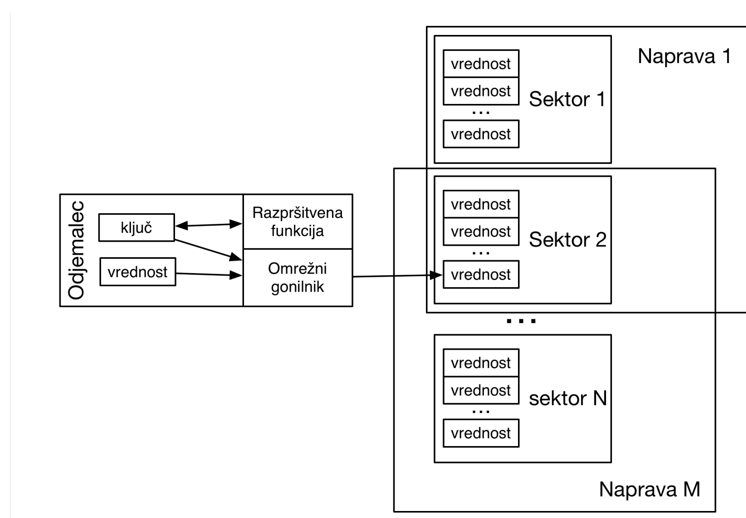
Agregacija pa ima drugačen pristop. Omogoča operacije na podatkih, ki imajo kompleksnejšo strukturo kot preprosta množica vrstic. Te kompleksnejše strukture navadno omogočajo gnezdenje struktur. V sami strukturi so lahko tudi seznami, ki delujejo kot kazalci na druge strukture. Izraz agregacija izhaja iz domensko vodenega oblikovanja (angl. Domain-driven design) [3]. V njem je agregacija zbirka sorodnih podatkov, ki jih želimo obravnavati kot enoto.

Različne agregirane shrambe imajo različne tehnike shranjevanja podatkov, vendar pa imajo veliko skupnega pri poizvedbah. Vsaka shramba ima svoje funkcije, ki omogočajo enostavne poizvedbe. Tu predvsem mislimo na indeksiranje, povezovanje dokumentov in povpraševalni jezik. Pri zapletenih poizvedbah pa gredo navadno agregacije skozi zunanje orodje, ki poskrbi, da pridobimo podatke, ki jih želimo.

Agregirane shrambe so namenjene velikim podatkom, ki med seboj niso pretirano povezani. Za zelo povezane podatke pa so bolj primerne grafne podatkovne baze.

2.6 Dokumentne shrambe

Dokumentne podatkovne baze delujejo kot hierarhične strukture. Dokumenti so sestavljeni hierarhično, podobno kot formata JSON in XML. Na najosnovnejšem nivoju so lahko dokumenti shranjeni oziroma vrnjeni po ID-ju (identita dokumenta). Načeloma pa se dokumentne shrambe opirajo na indekse, ki olajšajo dostop do dokumentov s pomočjo atributov. Indekse se uporablja za pridobitev množice sorodnih dokumentov, katere lahko aplikacija uporabi. Pri dokumentnih shrambah je pisanje dražja operacija, ker mora pisanje vseskozi vzdrževati tudi indekse, podobno kot pri relacijskih podat-



Slika 2.1: KV shrambe se obnašajo kot porazdeljene razpršene podatkovne strukture

kovnih bazah. Pri branju pa moramo za razliko od relacijskih podatkovnih baz preiskati veliko manjšo količino podatkov, da pridemo do želenega odgovora. Pri aplikacijah, v katerih se ogromno piše, lahko indeksiranje poslabša skupno hitrost baze. Na drugi strani pa je branje v neindeksiranih bazah izjemno počasno, saj moramo preiskati celotno množico.

2.7 Ključ-vrednost shrambe (KV)

KV shrambe so podobne dokumentnim shrambam, le da so se izrodile iz Amazon's Dynamo database [4]. Delujejo kot velike, porazdeljene razvrščevalne podatkovne strukture, ki shranjujejo in pridobivajo podatke na osnovi ključa. Podatki so s pomočjo razprševalne funkcije razpršeni skozi različne sektorje (angl. bucket) na spletu. Sektorji so podvojeni na različnih napravah, kar lahko vidimo na sliki 2.1.

To nam zagotavlja toleranco na napake. Kolikokrat mora biti sektor podvojen določa formula R = 2F + 1, kjer je F število napak, ki jih lahko toleriramo. Podvojevalni algoritem poskrbi, da sistemi niso točne kopije drug

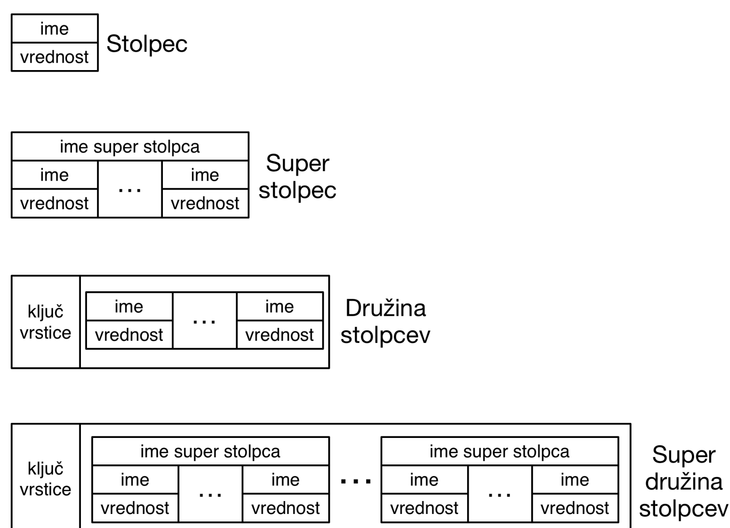
drugega.

Noben sistem ni popolnoma zanesljiv. V primeru napak na napravi se mora sistem obnašati kot da se ni nič zgodilo. Veliko sistemov si pri tem pomaga z odvečnimi podatki. Če prva naprava preneha delovati, preklopi sistem na drugo napravo, ki je kopija prve. Pri KV shrambah v primeru neodzivnosti računalnika nanj ne moramo shranjevati novih vrednosti, zato se v času, ko se računalnik popravlja, izvede tako imenovano dosledno razprševanje (angl. consistent hashing). S to tehniko se območje razvrščevalne funkcije, ki je bila na pokvarjeni napravi, preslika na naslednjo napravo, ki je na voljo. Ko se nedosegljiva naprava popravi, se preslikajo dodane vrednosti iz pomožne naprave nazaj na prvotno napravo.

KV shrambe ne vedo, kakšne podatke imajo shranjene. Njihova naloga je, da poskrbijo za učinkovito shranjevanje in pridobivanje podatkov. Pogosto se zgodi, da uporabnik ne potrebuje celotne shranjene vrednosti, potrebuje le delček informacije, ki je shranjena v vrednosti. Iz pridobljene vrednosti je potrebno podatek izluščiti z odstranjevanjem nepotrebnih informacij. Za to navadno uporablja zunanji program, eden izmed teh programov je MapReduce [5]

2.8 Shrambe družin stolpcev

Shrambe družin stolpcev (column family) so izdelane na podlagi Google's BigTable [6]. Podatkovni model bazira na redko poseljenih tabelah, katerih vrstice vsebujejo stolpce. Ključi vrstic zagotavljajo naravno indeksiranje. Na sliki 2.2 se lepo vidijo štirje pogosti primeri grajenja blokov v shrambah družin stolpcev. Najenostavnejša enota shrambe je stolpec (column), ki je sestavljen iz para ime-vrednost. Poljubno število stolpcev lahko združimo v super stolpec (ang. super column). Super stolpci so slovarji (angl. dictionary). Stolpci so shranjeni v vrstice. Vrstica, sestavljena iz samih stolpcev, se imenuje družina stolpcev (angl. column family). Družina stolpcev je shranjena na disk. Vsi podatki znotraj ene družine stolpcev so shranjeni v



Slika 2.2: Bloki za grajenje pri družini stolpcev

skupnem setu datotek. Vrstici, ki je sestavljena iz super stolpcev, pravimo super družina stolpcev (angl. super column family). Stolpci in super stolpci ne zavzamejo nič prostora, če ne vsebujejo nobene vrednosti.

Pri družinah stolpcev ni sheme, definiramo lahko le ime in kako bodo ključi razvrščeni. Izjemno pomembno je samo načrtovanje podatkovne baze, saj se lahko zgodi, da ob napačnem načrtovanju ne bo več moč priti do shranjenih podatkov. Samo poizvedovanje je navadno na voljo v eni izmed dveh oblik, in sicer poizvedba s ključem oziroma poizvedba z območjem ključa (angl. key range). Ključ določa, kje so fizični podatki dejansko locirani. Podatki so shranjeni glede na razvrščanje, ki ga definiramo na začetku. Samega razvrščanja pa se načeloma ne da spreminjati (razen izbire med naraščajočim in padajočim razvrščanjem) [7].

Poglavje 3

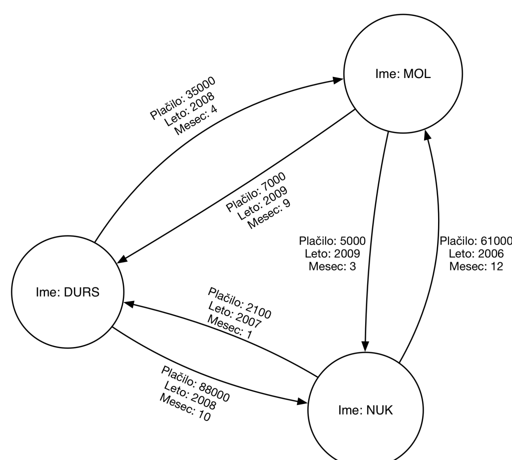
Grafne podatkovne baze

3.1 Kaj je graf?

Graf je matematična struktura, ki jo uporabljamo za modeliranje razmerij med objekti [8]. Formalno je graf zbirka točk in povezav, ki te točke povezujejo. V svetu grafnih podatkovnih baz pravimo, da je graf množica vozlišč in razmerij med temi vozlišči. Graf je lahko neusmerjen ali usmerjen. Neusmerjen graf ima razmerje med vozlišči simetrično. Povezave niso usmerjene, tako da ne vemo, katero vozlišče je začetno in katero končno. Pri usmerjenem grafu so razmerja med vozlišči nesimetrična. Zaradi usmerjenosti povezav vemo, katero vozlišče je začetno in katero končno.

Na sliki 3.1 je primer usmerjenega grafa. Krogi predstavljajo vozlišča, medtem ko puščice kažejo v kakšnem razmerju so vozlišča. Vidimo tudi, da ima vsako vozlišče in vsako razmerje določene lastnosti. To je zato, ker spada graf na sliki v model grafa značilk, ki si ga bomo pogledali v nadaljevanju. Več o sami teoriji grafov si lahko ogledate v [9].

Graf je izjemo uporaben za razumevanje velike množice podatkov na področjih znanosti, vlade in poslovanja. Realni svet je zelo raznolik, bogat in medsebojno povezan. Take podatke je izjemno težko predstaviti v relacijskem svetu, zato so se začele razvijati grafne podatkovne baze.



Slika 3.1: Ta graf, spada v vrsto grafov značilk

3.2 Grafne podatkovne baze

Grafne podatkovne baze so spletni sistem za upravljanje podatkovnih baz s CRUD metodami. CRUD je kratica za angleške besede Create (slov. ustvari), Read (slov. preberi), Update (slov. posodobi), Delete (slov. izbirši). Podatki so shranjeni kot graf. Grafne podatkovne baze so navadno narejene za uporabo s transakcijskim sistemom (OTLP). Poznamo pa tudi orodje za obdelovanje grafov (angl. graph compute engine), ki se uporablja v analitiki.

Grafne podatkovne baze se v grobem delijo po dveh lastnostih. To sta osnovno shranjevanje (angl. underlying storage) in obdelovanje (angl. processing engine).

Nekatere grafne podatkovne baze uporabljajo avtohtono grafno shranjevanje, ki je optimizirano in prilagojeno za shranjevanje in upravljanje grafov. Ne uporabljajo pa vse tehnologije avtohtonega shranjevanja grafov. Nekatere tehnologije serializirajo podatke grafa v relacijske ali objektno usmerjene podatkovne baze. Nekatere grafne podatkovne baze uporabljajo brezindeksno sosednost, kar pomeni, da vozlišča v podatkovni bazi fizično kažejo drug na drugega. V grafne podatkovne baze vključujemo vse podatkovne baze, ki se

navzven obnašajo kot grafne podatkovne baze. Brezindeksno sosedstvo deluje bolje in hitreje ter je že po sami definiciji podobno grafom. Brezindeksno sosedstvo zaradi prednosti pred ostalimi metodami smatramo kot avtohtono obdelovanje.

Osnovno shranjevanje in obdelovanje si bomo podrobneje pogledali s pomočjo Neo4j grafne podatkovne baze.

3.2.1 Graf značilk

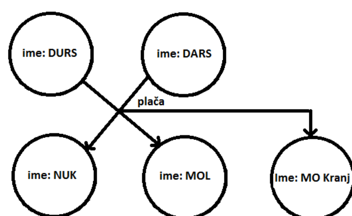
Graf značilk je najbolj razširjena oblika grafnega modela. Primer grafa značilk smo videli že na sliki 3.1. Graf značilk ima naslednje lastnosti:

- Vsebuje vozlišča, razmerja in značilke.
- Vozlišča vsebujejo značilke. Vozlišča si lahko predstavljamo kot dokument, v katerem so shranjene značilke v obliki KV parov.
- Razmerja imajo ime in so usmerjena, vedno imajo začetno in končno vozlišče.
- Tudi razmerja imajo lahko značilke.

Graf značilk je zelo intuitiven in lahek za razumevanje. Kljub njegovi preprostosti pa lahko z njim opišemo večino primerov uporabe na način, ki omogoča uporaben vpogled v naše podatke.

3.2.2 Hipergraf

Hipergraf je posplošen model grafa, pri katerem je lahko eno razmerje (hiperpovezava) povezano z večimi vozlišči. Za razliko od grafa značilk, pri katerem imamo lahko samo eno vozlišče tako na začetku kot na koncu, imamo lahko tu več vozlišč na vsaki strani razmerja. Hipergrafi so uporabni predvsem v domenah, ki so sestavljene iz veliko-za-veliko (angl. many-to-many) razmerij. Na sliki 3.2 vidimo hipergraf z eno hiperpovezavo. Če bi hoteli isti graf predstaviti z grafom značilk, bi morali uporabiti šest razmerij.



Slika 3.2: Primer hipergrafa

Hipergraf in graf značilk sta izomorfna, kar pomeni, da lahko podatke v hipergrafu vedno predstavimo tudi z grafom značilk. Kateri graf uporabiti je odvisno od samega modeliranja in izdelave aplikacije.

3.2.3 Trojice

Trojice (angl. triples) so osebek-povedek-predmet (angl.subject-predicate-object) podatkovna struktura. S tako strukturo lahko zajamemo dejstva, kot so "Janez pleše z Marijo", ali pa "Marija ima rada sladoled". Vsaka trojica zase izgleda dokaj osiromašeno, ob združitvi pa predstavljajo bogato množico podatkov. Shrambe trojic navadno zagotavljajo SPARQL (povpraševalni jezik za RDF) in so shranjene v RDF formatu:

```
   <rdf:RDF xmlns:rdf="www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns="www.example.org/terms/">
    <rdf:Description rdf:about="www.example.org/janez">
        <ime>Janez Novak</ime>
            <poklic>plesalec</poklic>
            <partner rdf:resource="www.example.org/marija"/>
    </rdf:Description>
    <rdf:Description rdf:about="www.example.org/marija">
        <ime>Marija Novak</ime>
        <poklic>plesalec</poklic>
        <obozuje rdf:resource="www.example.org/sladoled"/>
```

```
    </rdf:Description>
</rdf:RDF>
```

Trojice spadajo v sekcijo grafnih podatkovnih baz, ker so podatki logično povezani, ko so obdelani. Niso pa to avtohtone grafne baze, saj ne podpirajo brezindeksne sosednosti. Tudi sama njihova shramba ni prilagojena za shranjevanje grafov značilk. Trojice so shranjene v shrambi kot neodvisni podatki, kar jim dovoljuje horizontalno skaliranje, ne omogoča pa hitrega sprehajanja po razmerjih. Za izvajanje grafnih poizvedb na trojicah morajo shrambe trojic narediti povezane strukture iz neodvisnih podatkov, kar pa prinese zakasnitev k vsaki poizvedbi. To je razlog, da se trojice uporabljajo predvsem v analitiki, kjer je zakasnitev postranskega pomena.

3.3 Prednosti grafnih podatkovnih baz

V današnjem svetu lahko večino problemov predstavimo z grafi. Grafne podatkovne baze predstavljajo zelo močno orodje za modeliranje takih problemov, vendar pa to ni dovolj močan razlog za zamenjavo že utečenih podatkovnih baz. Obstaja mnogo množic podatkov, pri katerih je delovanje izboljšano, zakasnitev pa je manjša v primerjavi z agregiranimi procesi. Zmogljivost pa ni edina prednost. Grafne podatkovne baze so tudi izjemno prilagodljiv podatkovni model.

3.3.1 Zmogljivost

Ena glavnih prednosti grafnih podatkovnih baz je odlično delovanje pri povezanih podatkih v primerjavi z NoSQL shrambami in relacijskimi podatkovnimi bazami. Relacijske podatkovne baze z veliko količino podatkov imajo v primeru združevalnih (angl. join) operacij izjemno pogoste zakasnitve. Več združevalnih operacij uporabimo skupaj, slabše bo delovanje. Pri grafnih podatkovnih bazah pa ostaja delovanje relativno konstantno, tudi pri povečanju množice podatkov. Razlog zato je v tem, da so poizvedbe lokalizirane, zato

je čas vsake poizvedbe odvisen od podgrafa, po katerem smo se sprehodili, da smo zadostili poizvedbi.

3.3.2 Prilagodljivost

Razvijalci in podatkovni arhitekti ponavadi ne vedo strukture podatkovnega modela na začetku. Ponavadi se struktura spremeni, ko se bolj spoznamo s problemom. Prednost grafnih podatkovnih baz je, da na začetku ni potrebno izdelati strukture podatkovnega modela, kot moramo to storiti pri relacijskih podatkovnih bazah, ampak se le-ta lahko gradi sproti. Grafom lahko dodajamo nova razmerja in nova vozlišča v obstoječo strukturo, ne da bi pri tem motili obstoječe poizvedbe ali delovanje aplikacije.

3.3.3 Agilnost

Moderne grafne podatkovne baze so prilagojene za inkrementalno in iterativno razvijanje programske opreme. Grafne podatkovne baze so po naravi brez sheme in se enostavno testirajo. Prav zato lahko svojo aplikacijo razvijemo v kontroliranem okolju.

3.4 Razmerja v grafnih podatkovnih bazah

Uporabnik podatkovnih baz ve oziroma sklepa, kako so podatki oziroma entitete povezane med seboj. Sami podatkovni modeli pa se teh povezav ne zavedajo. To se pozna predvsem pri zelo počasnih poizvedbah.

Grafni podatkovni model pa je zasnovan drugače. Podatki, med katerimi je povezava v sami domeni, so povezani tudi v podatkovni bazi. Tem povezavam pravimo razmerja.

Razmerja v grafu tvorijo poti. Poizvedba oziroma obhod grafa vključuje sprehajanje po poti. Grafne podatkovne baze, ki imajo podatke shranjene kot graf in so prilagojene za delo z grafi, so izjemno hitre. Za lažjo predstavo hitrosti pa si oglejmo spodnji primer.

V knjigi [10] so naredili preizkus hitrosti na socialni mreži. Iskali so prijatelje prijateljev do globine pet. Baza je vsebovala približno milijon ljudi, vsak pa je imel okoli petdeset prijateljev. Poizvedbe so opravili na relacijskem modelu in na grafni podatkovni bazi Neo4j. Če pogledamo rezultate, vidimo, da so grafne podatkovne baze zelo hitre na zelo povezanih podatkih.

Globina	čas pri RDBMS (s)	čas pri Neo4j (s)	število rezultatov
2	0.016	0.01	2500
3	30.267	0.168	110000
4	1543.505	1.359	600000
5	Nedokončano	2.132	800000

3.5 Neo4j

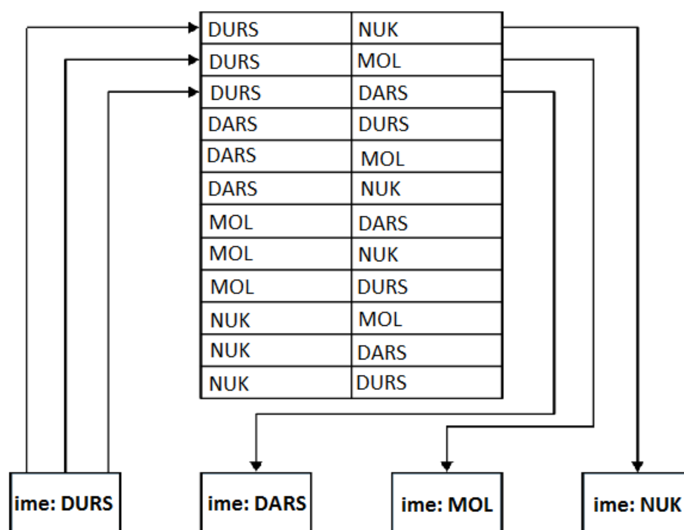
Neo4j je ena najbolj razširjenih grafnih podatkovih baz. Gre za zelo robustno, skalabilno in visoko zmogljivostno podatkovno bazo. Zajema:

- ACID transakcije;
- visoko razpoložljivost;
- skalira do milijardo vozlišč in razmerij;
- izjemno hitre poizvedbe, ki delujejo kot obhodi grafa;
- deklarativen grafni povpraševalni jezik.

Neo4j je avtohtona grafna podatkovna baza. To je tudi razlog, zakaj si bomo na njej podrobneje pogledali, kako grafne podatkovne baze delujejo in kako shranjujejo podatke.

3.5.1 Avtohtona grafna obdelava

Vsaka grafna podatkovna baza, ki ima lastnost, ki ji pravimo brezindeksna sosednost, ima avtohtone sposobnosti obdelovanja grafa. Brezindeksna sosednost pomeni, da ima vsako vozlišče direktno referenco na sosednje vozlišče,



Slika 3.3: Globalno indeksiranje na neavtohtonih grafnih podatkovnih bazah.

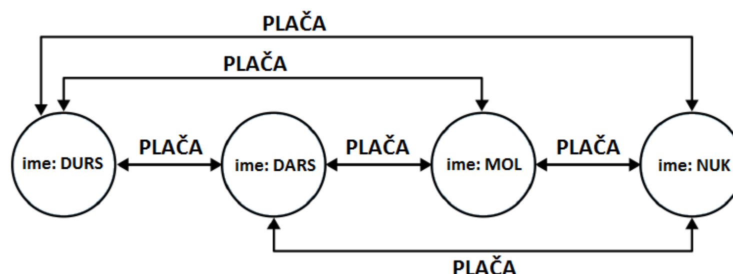
tako da se vozlišče obnaša kot nekakšen mikro indeks vseh sosednih vozlišč. To je veliko ceneje, kot če bi uporabljali globalne indekse. To pomeni, da je čas poizvedbe neodvisen od celotne velikosti grafa, ampak je sorazmeren količini preiskanega grafa.

Neavtohtone podatkovne baze uporabljajo globalne indekse, s katerimi povezujejo vozlišča med seboj, kot je prikazano na sliki 3.2. Ti indeksi dodajo dodatno plast nedirektnosti, kar zahteva več obdelovanja.

Da bi razumeli, zakaj je grafno obdelovanje dosti učinkovitejše, si oglejmo naslednji primer. Algoritmična zahtevnost iskanja indeksa pri globalnih indeksih je $O(log\ n)$, pri direktnih indeksih je ta zahtevnost $O(1)$. Če se želimo sprehoditi po vozliščih, je zahtevnost globalnih indeksov $O(m\ log\ n)$, medtem ko je pri brezindeksni sosednosti zahtevnost $O(m)$.

Globalni indeksi so primerni za manjšo množico podatkov. Za večje množice podatkov se uporabljajo grafne podatkovne baze, ki omogočajo brezindeksno sosednost. Takšna podatkovna baza je tudi Neo4j.

Neo4j uporablja za sprehod po grafu razmerja. Primer razmerja lahko vidimo na sliki 3.4. Po grafu se lahko sprehajamo naprej ali nazaj zelo



Slika 3.4: Avtohtone grafne podatkovne baze imajo podatke povezane z razmerji.

poceni.

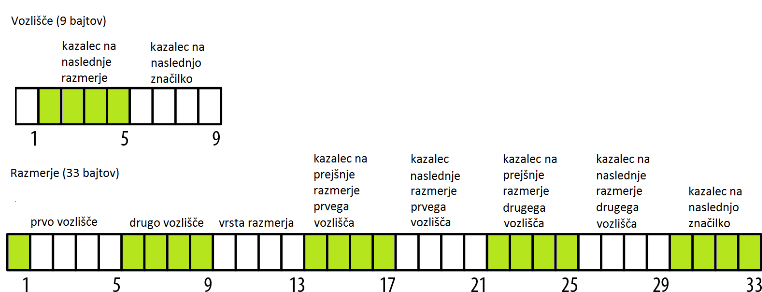
3.5.2 Avtohtono shranjevanje

Že prej smo omenili, da bomo avtohtono shranjevanje podrobneje predstavili s pomočjo Neo4j podatkovne baze. Neo4j je baza, ki je popolnoma prilagojena za delo z grafi, tako pri obdelovanju podatkov, kot tudi pri samem shranjevanju podatkov.

Neo4j shranjuje podatke v številne različne shrambne datoteke. Vsaka taka datoteka vsebuje podatke o specifičnem delu grafa (npr. vozlišča, razmerja, značilke).

Shramba vozlišč shranjuje vozlišča. Vsako novo vozlišče, ki ga naredi uporabnik, se shrani v to shrambo. Shramba je na disku fizično shranjena v datoteki neostore.nodestore.db. Zapisi v shrambi so fiksne dolžine, vsak zapis pa je dolg 9 bajtov. Zaradi fiksne dolžine lahko izjemno hitro najdemo vozlišča. V primeru, da iščemo vozlišče s številko 10, vemo, da je shranjeno 90 bajtov od začetka shrambe vozlišč. Algoritemska zahtevnost je *O(1)*, saj točno vemo, kje je vozlišče shranjeno. Strukturo vozlišča in razmerja lahko vidimo na sliki 3.5.

Prvi bajt zapisa vozlišča je zastavica, ki nam pove, ali je na tem mestu že shranjeno vozlišče, drugače se lahko tu shrani novo vozlišče. Naslednji štirje bajti predstavljajo kazalec na prvo razmerje, ki je povezano na vozlišče.



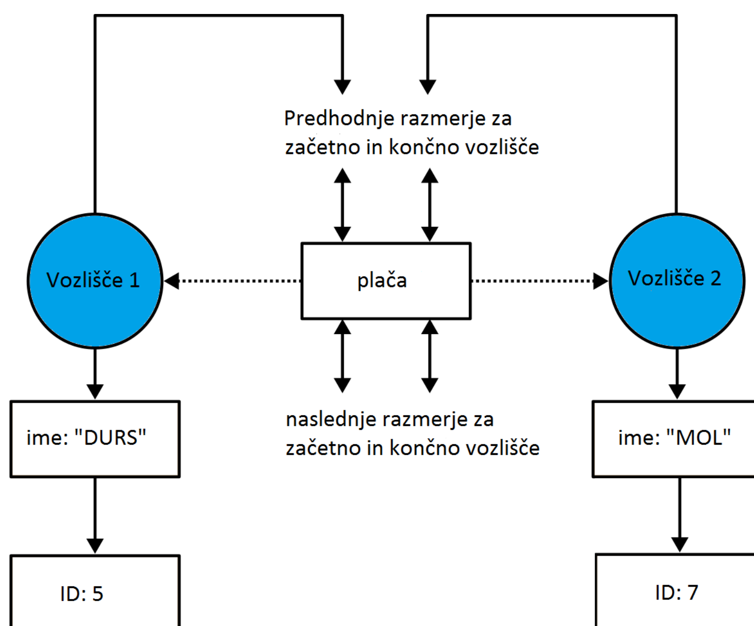
Slika 3.5: Struktura zapisa vozlišča in razmerja.

Zadnji štirje bajti pa predstavljajo kazalec prve značilke, ki pripada temu vozlišču.

Podobno kot vozlišča so tudi razmerja shranjena v shrambo razmerij. Datoteki pravimo neostore.relationshipstore.db. Tudi zapisi razmerij so fiksne dolžine, le da je tu vsak zapis velik 33 bajtov. Vsak zapis ima kazalec na začetno in končno vozlišče razmerja, kazalec na vrsto razmerja (ta je shranjen na shrambi vrst razmerij), kazalce za naslednji in prejšnji zapis razmerij, za vsako začetno in končno vozlišče. Tem štirim kazalcem pravimo veriga razmerij. Zadnji kazalec pa kaže na naslednjo značilko.

Značilke so na disku shranjene v datoteki neostore.propertystore.db. Tako kot zapisi razmerij in vozlišč imajo tudi značilke fiksno dolžino zapisov. Vsak zapis je sestavljen iz štirih blokov in kazalca na naslednjo značilko v verigi značilk. Vsaka značilka zasede od enega do štiri bloke. Zapis značilke lahko vsebuje največ štiri značilke. Zapis značilke vsebuje vrsto značilke in kazalec na indeksno datoteko značilk, v kateri je shranjeno ime značilke. Vsaka vrednost značilke vsebuje zapis kazalca na dinamično shrambo zapisov ali pa ima vrednost vključeno. Dinamične shrambe so namenjene shranjevanju velikih vrednosti. Poznamo dve dinamični shrambi, in sicer dinamična shramba nizov in dinamična shramba polij. Dinamični zapisi so v bistvu povezani seznami zapisov s fiksno dolžino. Veliki podatki lahko zasedejo več zapisov.

Na sliki 3.6 lahko vidimo, kako so različne shrambe povezane med seboj. Obe vozlišči vsebujeta kazalec na prvo značilko in prvo razmerje v verigi



Slika 3.6: Predstavitev grafa na disku

razmerij. Če želimo prebrati značilko vozlišča, moramo slediti enojno povezanemu seznamu značilk, ki se začne s kazalcem na prvo značilko. Pri razmerjih je pristop podoben. Glavna razlika je ta, da se pri razmerjih sprehajamo po dvojno povezanem seznamu razmerij. Ko najdemo razmerje, ki ga iščemo, lahko preberemo njegove značilke tako, kot smo prebrali značilke vozlišča.

3.5.3 Predpomnilnik

Pogledali smo si, kako ima Neo4j shranjene podatke na disku. Shranjevanje podatkov je prilagojeno za hitre obhode po grafu, vendar avtohtono shranjevanje samo po sebi še ne obljublja hitrosti. Baza podatkov je običajno prevelika, da bi jo lahko imeli v celoti v pomnilniku. Podatek, ki ni v pomnilniku, je potrebno prebrati z diska. Ker so diski zelo počasni, je branje običajno zamudno. Večina grafnih podatkovnih baz uporablja predpomnilnik, da bi na ta način zmanjšale zakasnitev.

Neo4j uporablja dvonivojski predpomnilnik. Nižji nivo je predpomnilnik datotečnega sistema. Ta razdeli vsako shrambo v diskretne regije in potem drži fiksno število regij shrambe. Strani se izmenjujejo s strategijo *najmanj pogosto uporabljeni* (angl. least frequently used). Neo4j prepusti menjavanje strani operacijskemu sistemu, tako da sam operacijski sistem menja strani in odloča, katere segmente bo obdržal v pomnilniku in katere bo shranil na disk. Predpomnilnik datotečnega sistema je posebej koristen, ko spreminjamo povezane komponente grafa, ki so shranjene na eni strani. To se velikokrat pojavlja pri pisanju.

Na višjem nivoju je objektni predpomnilnik. Ta je optimiziran predvsem za branje. Predpomnilnik shrani vozlišča, razmerja in značilke v objektno predstavitev. Vozlišča v objektnem predpomnilniku vsebujejo tako značilke kot tudi kazalce na razmerja, kar predstavlja bogatejšo predstavitev kot predstavitev v shrambni datoteki na disku. Za razliko od vozlišč so razmerja v objektnem predpomnilniku veliko bolj enostavna, saj vsebujejo le pripadajoče značilke. Vozlišče ima razmerja grupirana po vrsti in usmeritvi, kar omogoča hitro iskanje razmerij.

3.5.4 Cypher - grafni povpraševalni jezik

Cypher je grafni povpraševalni jezik. Zasnovan je na način, ki je lahek za uporabo in ga lahko razumejo vsi, ki bi radi komunicirali s podatkovno bazo. Podobno kot SQL je tudi Cypher deklarativen povpraševalni jezik, kar pomeni, da se Cypher ukvarja s tem, kaj pridobiti z grafa in ne kako.

Kot večina povpraševalnih jezikov je tudi Cypher sestavljen iz stavkov. Najenostavnejše poizvedbe so sestavljene iz START stavka, ki mu sledi MATCH in RETURN stavek. Da si lahko stavke lažje razložimo, poglejmo naslednji primer, v katerem poiščemo vse Janezove skupne prijatelje:

```
START a=node:uporabnik(ime='Janez')
MATCH (a)-[:POZNA]->(b)-[:POZNA]->(c), (a)-[:POZNA]->(c)
RETURN b, c
```

Podrobneje si poglejmo vse tri stavke, ki nastopajo v poizvedbi.

START: Začetne točke v grafu, ki so lahko vozlišča ali razmerja. Začetne točke lahko pridobimo z indeksnim iskanjem, ali pa direktno z identiteto vozlišča oziroma razmerja.

V našem primeru iščemo začetno vozlišče v indeksu, ki mu pravimo uporabnik. Ta indeks povprašamo, katero vozlišče ima značilko ime, katere vrednost je Janez. Vrednost, ki jo iskanje vrne, je potem povezana z identifikatorjem, v našem primeru je to 'a'. Ta identifikator nam omogoča, da se lahko sklicujemo na začetno vozlišče skozi celotno poizvedbo.

MATCH: To je specifikacija glede na primer. Z ASCII znaki predstavimo vozlišča in razmerja ter z njimi narišemo podatke, ki jih iščemo. Vozlišča narišemo z oklepaji, medtem ko za razmerja potrebujemo pomišljaj in znak za večje ali manjše. Puščica razmerja nam pove tudi, kakšna je usmeritev razmerja. Med pomišljaje napišemo vrsto razmerja. Ta mora biti napisana znotraj oglatih oklepajev in se začeti s podpičjem.

V našem primeru imamo vzorec (a)-[:POZNA]->(b)-[:POZNA]->(c), (a)-[:POZNA]->(c). Ta vzorec predstavlja pot med tremi vozlišči. Eno vozlišče je vezano na identifikator 'a', ostali dve vozlišči pa sta vezani na 'b' in 'c'. Vozlišča so povezana s "POZNA"razmerji. V teoriji bi se ta vzorec lahko znotraj grafa pojavil velikokrat. Da bi lokalizirali poizvedbo, je potrebno del poizvedbe zasidrati nekam v graf.

To smo naredili že s stavkom START, v katerem smo si izbrali začetno vozlišče. Vezali smo vozlišče "Janez"na identifikatior 'a'. S tem smo se postavili oziroma zasidrali nekam v graf. Cypher potem preveri še preostali del vzorca okoli začetne točke. Med preverjanjem odkrije še druga vozlišča, ki ustrezajo vzorcu in ta vozlišča veže na preostala identifikatorja. V našem primeru bomo vedno zasidrani v vozlišču "Janez", medtem ko se bosta ostala identifikatorja vezala na različna vozlišča med samo poizvedbo.

RETURN: Ta stavek določa, katera vozlišča, razmerja in značilke iz podatkov, ki ustrezajo vzorcu, bomo vrnili uporabniku. V primeru naše poi-

zvedbe smo vrnili vozlišča vezana na 'b' in 'c' identifikator.

Poleg osnovnih treh stavkov poznamo pri Cypheru tudi druge stavke:

- WHERE: Filter.
- CREATE: Ustvari vozlišča in razmerje.
- DELETE: Izbriše vozlišča, razmerja in značilke.
- SET: Nastavi vrednost značilki.
- FOREACH: Izvede posodobitev enkrat na element v seznamu.
- WITH: Razdeli poizvedbo na več različnih delov.

Poglavje 4

Aplikacija

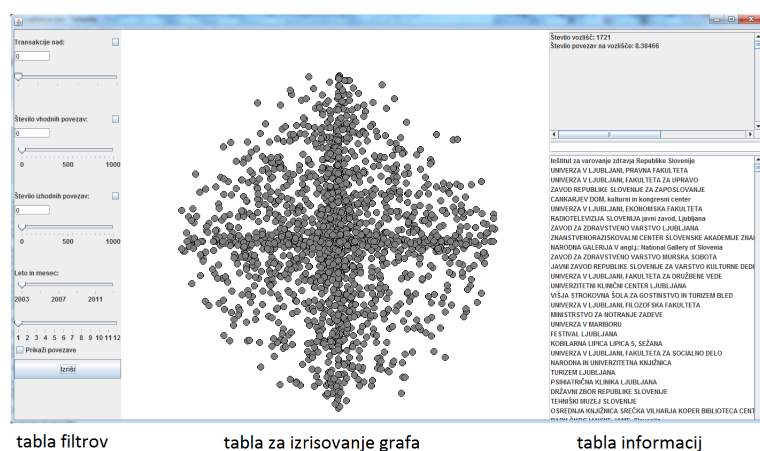
Cilj aplikacije je bila enostavna vizualizacija realnih podatkov. Aplikacijo lahko rahlo pomanjšano vidimo na sliki 4.2. Podatki v našem primeru so transakcije med slovenskimi javnimi organi. Podatke smo pridobili na spletni strani komisije za preprečevanje korupcije, kjer so ti podatki tudi javno dostopni [11]. V aplikaciji so podatki shranjeni v grafni podatkovni bazi Neo4j, ki smo si jo natančneje ogledali v prejšnjem poglavju.

4.1 Postavitev podatkovne baze

Preden smo lahko postavili podatkovno bazo, je bilo potrebno podatke urediti, saj so med njimi manjkale določene informacije, ki so pomembne za našo aplikacijo.

4.1.1 Pridobitev in priprava podatkov

Podatki, ki smo jih pridobili, so v tekstovnem formatu. Podatki obsegajo vse javno dostopne transakcije od leta 2003 pa do marca leta 2013. V datoteki, ki smo jo pridobili, je vsaka vrstica predstavljala eno transakcijo. Da si lažje predstavljamo, kako so podatki shranjeni, poglejmo format vrstice. V vrstici imamo 6 atributov, in sicer: šifra proračunskega upravičenca (dajalec), davčna številka (prejemnik), leto in mesec transakcije, vsota prejemkov,



Slika 4.1: Glavno okno aplikacije, z vsemi tremi tablami

nepovratna sredstva in fiduciarni posli. Več informacij o posameznem atributu je dostopnih na [12]. V tem dokumentu so tudi neposredne povezave do spletnih strani, s katerih smo pridobili sezname za preslikave, ki so omenjene v nadaljevanju. Za potrebe naše aplikacije smo zadnja dva atributa spustili. Dva atributa, šifra proračunskih upravičencev in davčna številka, sta prestavljena kot številka. Vsem šifram in davčnim številkam je bilo potrebno dodeliti dejanska imena.

Preslikavo za šifre proračunskih uporabnikov smo pridobili na spletni strani Uprave Republike Slovenije za javna plačila. Tudi ti podatki so bili v tekstovni datoteki. Vsaka vrstica je predstavljala preslikavo za eno šifro. Vrstica ima okoli 25 atributov, kar je mnogo več kot smo potrebovali, zato smo ven izluščili le podatke, ki jih potrebujemo. Za to je poskrbel razred `Sifra_PU_parser`. Razred je zelo enostaven, saj se sprehodi po vseh vrsticah in iz vsake vzame le podatke, ki jih potrebujemo. Te podatke nato zapiše v drugo datoteko, v kateri imamo le podatke, ki jih želimo oziroma potrebujemo.

Podobno je potrebno storiti tudi za davčne številke. Preslikavo davčnih številk v nazive smo pridobili na spletni strani Davčne uprave Republike Slovenije. Razred, ki nam izlušči podatke, je `Davcne_Stevilke_Parser`.

Tudi seznam davčnih številk je v tekstovni datoteki, tako da je razred, ki nam preslika davčne številke, zelo podoben razredu `Sifra_PU_parser`. Glavna razlika je v tem, da imamo nekaj dodatne kode za razbitje podatkov, saj so podatki v datoteki zapisani zelo neurejeno. Z navadno delitvijo vrstice v tem primeru ne gre, zato si pomagamo z regularnimi izrazi.

Vse skupaj sedaj združimo v veliko tekstovno datoteko, ki je človeku lahko berljiva. Za to poskrbi razred `Transaction_parser`. Baza podatkov je izjemno velika, saj vsebuje okoli 15 milijonov transakcij, zato smo se odločili, da se omejimo zgolj na transakcije javne uprave. S to omejitvijo smo zmanjšali število transakcij na okoli 1.1 milijona.

4.1.2 Izdelava baze

Vse potrebne podatke imamo v eni skupni tekstovni datoteki, sedaj pa je potrebno te podatke dodati v podatkovno bazo. Podatki so med seboj zelo povezani in so že v osnovi graf. Za tako vrsto podatkov so najbolj primerne grafne podatkovne baze, zato si izberemo Neo4j grafno podatkovno bazo.

Javni organi so v bazi predstavljeni kot vozlišča, medtem ko so transakcije med njimi predstavljene kot razmerja. Vsako vozlišče ima značilko *name*, ki predstavlja ime javnega organa. Vsaka transakcija je tipa *PAID* in ima tri značilke. Te značilke so *amount*, ki predstavlja znesek nakazila ter *year* in *month*, ki predstavljata leto oziroma mesec nakazila. Vsako transakcijo med dvema javnima organoma dodamo kot novo razmerje. Razlog za to je, da lahko kasneje pri vizualizaciji vidimo, kdaj je določen javni organ zapravljal največ oziroma najmanj. Transakcije združimo le, če je več transakcij med dvema javnima organoma v istem letu in istem mesecu.

V Neo4j dodajamo povezave in vozlišča znotraj transakcij, kar predstavlja manjši problem pri dodajanju velikih količin podatkov naenkrat. Če dodamo vse podatke v isti transakciji, nam hitro zmanjka pomnilnika, tako da ta možnost odpade. Naslednja logična izbira je, da dodamo vsak podatek v bazo kot posamezno transakcijo. V tem primeru s pomnilnikom nimamo težav, vendar pa je vse skupaj deluje nekoliko počasneje. Ker nobena opcija

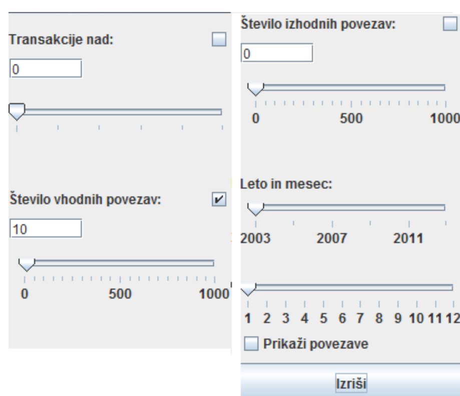
ni pretirano dobra, naredimo neke vrste hibrid obeh opcij. Podatke razbijemo na več manjših delov, ki jih potem znotraj ene transakcije dodamo v bazo. V našem primeru dodajamo po 200 tisoč podatkov naenkrat. To se izvede relativno hitro, tako da je vseh 1.1 milijona transakcij dodanih v bazo v tridesetih sekundah.

4.2 Splošno o aplikaciji

Naša aplikacija je napisana v programskem jeziku Java. Javo smo izbrali, ker je v njej napisan tudi Neo4j. Za operacije nad podatkovno bazo smo uporabili javanske knjižnice za Neo4j, ki omogočajo komunikacijo s podatkovno bazo znotraj same aplikacije. Knjižnica je zelo bogata in ne omogoča le osnovnih operacij, ampak še kopico drugih. Tu predvsem mislimo na grafne algoritme. Mi smo za potrebe naše aplikacije uporabljali le osnovne operacije, tako da se v to, kaj vse knjižnica omogoča, ne bomo poglabljali.

Sama vizualizacija je narejena v javanskem oknu. Zanimalo nas je, kako narediti vizualizacijo grafa brez pomoči knjižnic, ki so temu namenjene, zato nismo uporabljali nobenih knjižnic, razen osnovnih, kot so `swing` in `awt` (osnovni javanski knjižnici za delo z grafiko). Iz tega razloga aplikacija navzven mogoče tudi ne izgleda najbolj estetsko, vendar pa ima kar nekaj funkcionalnosti, ki pripomorejo k analizi grafa.

Kot smo že omenili, imamo ogromno bazo podatkov. Ker se pri takšni količini podatkov zelo težko kaj razbere iz samih povezav med njimi, smo se pri izdelavi aplikacije osredotočili predvsem na vozlišča. Z različnimi filtri, kot so število vhodnih in izhodnih povezav, spreminjamo velikost vozlišč. Če v določeni kategoriji kakšno vozlišče izstopa, nam bo takoj padlo v oči, saj je velikost tega vozlišča bistveno večja od velikosti ostalih vozlišč.



Slika 4.2: Tabla filtrov. Razdeljena je na dva dela, desni del table, je v aplikaciji pod levim.

4.3 Opis vidnega dela aplikacije

Vidni del naše aplikacije je javansko okno (JFrame), na katerem imamo tri glavne table (JPanel). Levo je tabla filtrov, v sredini je tabla za izris grafa, desno pa je tabla informacij o grafu. Vse table si bomo podrobneje pogledali.

4.3.1 Tabla filtrov

Tabla filtrov je svoj razred z imenom `FilterPanel` in razširja razred `JPanel`. Na tabli filtrov, slika 4.2, imamo filter, v obliki tekstovnih polj (JTextField) in drsnikov (JSlider), nekaj potrditvenih polj (JCheckBox), oznak (JLabel) in gumb (JButton).

Prvi drsnik je namenjen filtriranju transakcij po znesku transakcije. Nahaja se pod oznako *transakcije nad*. Njegovo območje je od 0 do 10 milijonov. Vsakemu drsniku pripada tudi tekstovno polje, na katerem je vrednost, ki smo jo izbrali na drsniku. Ob vsakem premiku drsnika se vrednost tekstovnega polja spremeni. Vrednost filtra lahko nastavimo tudi z vpisom vrednosti v polje. V tem primeru pa se drsnik nastavi na vrednost v polju. Ob vsakem izrisu grafa se vse povezave, za katere velja, da imajo vrednost zneska manjšo

kot na drsniku, ne bodo upoštevale.

Drugi in tretji drsnik sta izjemno podobna, zato ju bomo obravnavali skupaj. To sta drsnika za filtriranje vozlišč glede na število vhodnih in izhodnih povezav. Nahajata se pod oznako *število vhodnih povezav* in *število izhodnih povezav*. Oba drsnika imata svoje območje od 0 do 1000. Podobno kot pri prvemu drsniku tudi tema dvema drsnikoma pripada tekstovno polje. Razmerje med drsnikom in tekstovnim poljem je enako kot pri prejšnjem drsniku. Ta filter poskrbi, da se vsa vozlišča, katerih število vhodnih oziroma izhodnih povezav je manjše kot na drsniku, ne izrišejo.

Tudi zadnja dva drsnika lahko obravnavamo skupaj, to sta leto in mesec. Ta dva drsnika nimata pripadajočega tekstovnega polja, saj je izbor vrednosti hitro viden že iz drsnika. Drsnik za leto ima območje od 2003 do 2013, medtem ko ima drsnik za mesec območje od 1 do 12. S tema filtroma izberemo, katere povezave transakcije bomo uporabili pri vizualizaciji grafa. Vedno je izbrano posamezno leto in mesec, tako da je graf vedno viden le za določen mesec. S tem pridobimo na dinamičnosti grafa.

Pri zgornjih treh filtrih imamo potrditvena polja, ki so namenjena prikazu vozlišč v različnih velikostih. Z drsniki smo zmanjšali število povezav in vozlišč, s potrditvenimi polji pa spreminjamo velikost vozlišč. Če recimo izberemo potrditveno polje pri številu vhodnih povezav, se bo graf izrisal z vozlišči, katerih velikost bo odvisna od števila njihovih prejemkov. Večje število povezav ima vozlišče, večje bo izrisano. Velikost vozlišča je določena z enostavno linearno funkcijo. Podobno velja za število izhodnih povezav. Da bi izrisali vozlišča v odvisnosti od vsote transakcij, pa je potrebno označiti potrditveno polje pri transakcijah in še enega od ostalih dveh. Če bi želeli izris po vsoti prejemkov, izberemo zraven še potrditveno polje pri vhodnih povezavah, če pa bi želeli izris po vsoti izdatkov, pa izberemo potrditveno polje pri izhodnih povezavah.

Na tabli imamo tudi potrditveno polje, s katerim izberemo, ali želimo izris povezav ali ne. Izris povezav na najvišjem nivoju ni uporaben, saj se iz povezav ne da razbrati ničesar. Povezav je enostavno preveč. Povezave

postanejo bolj uporabne, ko so vozlišča že filtrirana in je njihovo število zmanjšano.

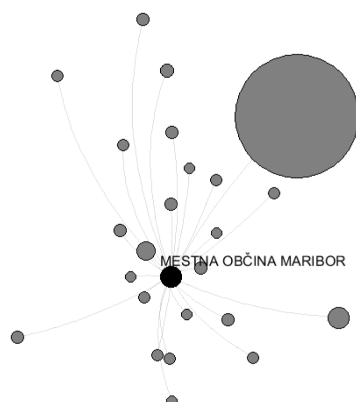
Zadnja stvar na tabli filtrov pa je gumb *Izriši*. Vsaka sprememba filtra se bo na grafu poznala šele po pritisku tega gumba. Sprememba na drsniku za izbor leta ali meseca, pa bo sama sprožila novo izrisovanje grafa. Razlog za to je, da se lažje sprehajamo po posameznih mesecih.

4.3.2 Tabla za izris grafa

Tudi ta tabla je svoj razred in razširja razred `JPanel`. Ime razreda je `GraphPanel`. Ta razred združuje vse tri table in vsebuje vso logiko za izris grafa. Graf je na tabli predstavljen s krogi, kar vidimo na sliki 4.3, ki predstavljajo vozlišča in krivulje, ki predstavljajo povezave. Ker smo se odločili, da ne bomo uporabljali knjižnic, se je prvi problem pojavil že pri razporeditvi vozlišč na tablo. Vse knjižnice ali programi za vizualizacijo imajo nek algoritem, s katerim razporedimo vozlišča. Z grafa z lepo razporejenimi vozlišči, se da hitro razbrati določene lastnosti samega grafa. Sama implementacija takih algoritmov pa je zelo zahtevna in se je v naši aplikaciji nismo lotili. Vozlišča imamo v aplikaciji razporejena naključno in so ob vsakem zagonu programa na drugem mestu. To ni moteče, saj je mogoče iz grafa še vedno razbrati stvari, za katere je bila aplikacija implementirana.

Povezav je v naši aplikaciji veliko, če ne uporabimo filtrov. Če imamo na grafu veliko vozlišč, se izrisovanje povezav odsvetuje, saj se iz njih nič ne vidi. Na najvišjem nivoju so vse povezave sive in med njimi ne ločimo, katera je vhodna in katera je izhodna. Ob izboru določenega vozlišča se vse povezave, ki izhajajo iz tega vozlišča, obarvajo rdeče, vse povezave, ki prihajajo v izbrano vozlišče, pa se obarvajo zeleno. Debelina povezav je en piksel, izjema so povezave, ki so obarvane rdeče in zeleno. Te so debeline dveh pikslov.

Vsako vozlišče lahko premaknemo kamor koli na tablo. Potrebno je le povleči vozlišče na novo lokacijo z levo miškino tipko. To je predvsem uporabno, če veliko vozlišče pokriva več manjših. Ker so manjša vozlišča prekrita, ne



Slika 4.3: Tabla za izris grafa. Velikost vozlišča je določena s številom vhodnih povezav

moremo do njih drugače, kot da veliko vozlišče premaknemo.

Na podoben način lahko premaknemo tudi celoten graf. Premik opravimo podobno kot pri vozliščih, in sicer tako, da z levo miškino tipko povlečemo na novo lokacijo.

Premikanje grafa in premikanje vozlišča izvedemo popolnoma enako. Aplikacija sama prepozna, ali smo kliknili na vozlišče ali kam drugam. Če smo povlekli vozlišče, se le-ta premakne, če pa smo povlekli praznino, se premakne graf.

Vozlišča so zelo skupaj, saj jih je zelo veliko. V veliko primerih se celo prekrivajo, zato smo implementirali tudi skaliranje. Skaliramo lahko s pomočjo miškinega kolesa. Sprva smo poskusili skaliranje implementirati s pomočjo razreda AffineTransform, ki je namenjen transformaciji. V tem razredu je metoda `scale (x, y)`, ki skalira po x in po y koordinati. Pri uporabi te metode nam je na zgornji sredini grafa naključno utripal manjši pravokotnik in vozlišča v tem pravokotniku so bila zamaknjena. Vozlišča, ki so se prekrivala, so ostala prekrita tudi po skaliranju. Da bi uspešno rešili nastalo težavo, smo implementirali svoje skaliranje.

Če pogledamo na graf, vidimo le kroge, ki predstavljajo vozlišča, ne vemo

pa za katero vozlišče gre. Izpis oznak na vseh vozliščih ni možen, saj postane to v celoti nepregledno. V naši aplikaciji lahko dobimo ime vozlišča tako, da nanj postavimo miškin kazalec. Če postavimo miškin kazalec na vozlišče, se bo to vozlišče obarvalo črno, nad njim pa se bo izpisalo njegovo ime.

4.3.3 Tabla informacij

Tabla informacij je še zadnja tabla v naši aplikaciji. Tudi ta razširja `JPanel`, razred pa ima ime `ListPanel`. Na tej tabli izpisujemo podatke o grafu in vozlišču. Tablo si lahko ogledamo na sliki 4.4. Tabla se deli na tri manjše sekcije, in sicer imamo na vrhu podatke o številu vozlišč in povezav oziroma podatke o posamezni transakciji. Na dnu imamo našteta vsa vozlišča, ki so trenutno prikazana na grafu, v sredini pa imamo majhno tekstovno polje, s pomočjo katerega iščemo po seznamu vozlišč.

Za izpisovanje informacij uporabimo oznako (JLabel). Na najvišjem nivoju se tu zapišejo le podatki o številu vozlišč in povezav, če pa imamo izbrano vozlišče, pa se dodajo še seznami vseh vhodnih in izhodnih povezav tega vozlišča. Teh povezav je lahko zelo veliko, zato oznako ovijemo z drsnikom (JScrollPane). To nam omogoča, da lahko vidimo vse podatke. Če je podatkov več, kot je velika oznaka, enostavno zdrsimo nižje in pridobimo ostale podatke.

Na dnu imamo seznam (JList) vseh vozlišč, ki so trenutno prikazana na grafu. Če kliknemo na izbrano vozlišče na seznamu, je efekt popolnoma enak, kot če bi kliknili na vozlišče na grafu. S klikom se bo izbralo novo vozlišče, in izrisale se bodo njegove povezave.

V sredini imamo tekstovno polje, ki služi kot iskalnik po seznamu vozlišč. Deluje tako, da vsaka sprememba v polju (napišemo ali zbrišemo črko), sproži filtriranje. Ta postopek poteka na način, da se vzorec, ki smo ga vpisali v polje, poišče znotraj vseh podnizov imen vozlišč. Torej, če je nekje v imenu vozlišča vzorec znakov, ki smo ga napisali v polje, bo to vozlišče na seznamu.

MESTNA OBČINA MARIBOR

Število vozlišč: 25
Število vhodnih povezav: 34

Število izhodnih povezav: 118

Izhodne povezave:
Vhodne povezave:
ZAVOD REPUBLIKE SLOVENIJE ZA ZAPOSLOVANJE: 24799.6481 EU
DRŽAVNI ZBOR REPUBLIKE SLOVENIJE: 130.8212 EUR

ZAVOD REPUBLIKE SLOVENIJE ZA ZAPOSLOVANJE
DRŽAVNI ZBOR REPUBLIKE SLOVENIJE
MESTNA OBČINA MARIBOR
ZAVOD REPUBLIKE SLOVENIJE ZA ŠOLSTVO
MINISTRSTVO ZA OBRAMBO REPUBLIKE SLOVENIJE
OBČINA MIKLAVŽ NA DRAVSKEM POLJU
OBČINA BENEDIKT
GEODETSKI INŠTITUT SLOVENIJE
OBČINA CERKVENJAK
OBČINA LOVRENC NA POHORJU
OBČINA STARŠE
OBČINA PESNICA
OBČINA KUNGOTA
OBČINA MOZIRJE
OBČINA ŽETALE
OBČINA RADLJE OB DRAVI
OKROŽNO SODIŠČE V MARIBORU
OBČINA IDRIJA
OBČINA RIBNICA NA POHORJU
OBČINA RAČE-FRAM
OBČINA SLOVENSKE KONJICE
OKROŽNO DRŽAVNO TOŽILSTVO V MARIBORU

Slika 4.4: Tabla informacij, na kateri so trenutno prikazani podatki za Mestno občino Maribor.

4.4 Pomembnejši problemi

Pri razvoju aplikacije smo naleteli na veliko težav. Nekatere so bile lažje rešljive, medtem ko je bilo potrebno za reševanje drugih porabiti nekaj več časa. Tu bomo pogledali, katere težave so se pojavljale in kako smo jih rešili.

4.4.1 Skaliranje

Glavni cilj skaliranja je bil razmakniti vozlišča, ki se prekrivajo. To smo storili tako, da se vozlišča, ki so bolj oddaljena od središča, bolj premaknejo ob vsakem nivoju skaliranja. Vozlišča, ki so blizu centra, pa se praktično ne premaknejo. Središče skaliranja je vedno miškin kazalec. Velikost vozlišča se pri skaliranju povečuje zelo počasi, medtem ko se vozlišča premikajo od centra zelo hitro. S tem dosežemo, da se vozlišča po nekaj nivojih skaliranja ne prekrivajo več. Za implementacijo skaliranja smo uporabili seznam, v katerem imamo shranjene koordinate vozlišč za vse nivoje skaliranja do trenutnega nivoja. Prvi nivo je enak začetnim koordinatam vozlišč, vsak naslednji pa se izračuna na podlagi prejšnjega. Nova koordinata vozlišča, tako x kot y, je: stara koordinata + tretjina razdalje od središča.

Za obratno smer skaliranja je vse skupaj bolj enostavno, saj ne potrebujemo ponovnega računanja, potrebno je le vzeti že izračunane kooordinate za tisti nivo.

Velikost vozlišča pa spreminjamo tako, da za vsak nov nivo prejšnjo velikost povečamo za 20 %.

4.4.2 Dostop do podatkov s pomočjo značilk

Vsa vozlišča grafa so na začetku iste velikosti. Ta velikost je 12 pikslov. Velikost vozlišč lahko spreminjamo le na dva načina. Prvi je skaliranje, ki je opisan zgoraj. Spreminjanje velikosti pri skaliranju je enostavno, saj se vozlišče vedno poveča za isto vrednost. Drugi način pa je izris vozlišč, ki imajo velikost določeno z enim od štirih atributov, ki smo jih omenili pri tabli filtrov. Tu je določitev velikosti nekoliko težji problem.

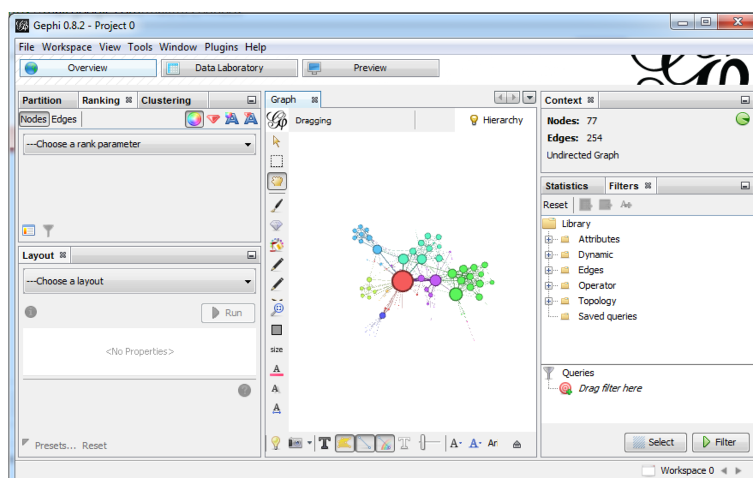
Povezavo lahko pridobimo iz baze glede na njen tip (v našem primeru PAID), ali glede na smer, ki je bodisi prihajajoča (*Direction.INCOMING*) ali odhajajoča (*Direction.OUTGOING*). Brez predhodnih nastavitev na podatkovni bazi ne moramo dostopati do podatkov s pomočjo značilk. Naša aplikacija potrebuje nove podatke vsakokrat ko zamenjamo leto ali mesec transakcij. Ker sta mesec in leto značilki, je bilo potrebno podatkovno bazo prehodno nastaviti, da je omogočala indeksiranje. Dodali smo novo značilko *yearmonth*, kjer imamo shranjeno leto in mesec pod skupno značilko. Podatkovno bazo smo nastavili tako, da so vse nove transakcije sedaj indeksirane po značilki *yearmonth*. Nato smo podatkovno bazo postavili ponovno. Ko imamo podatke tako pripravljene, lahko do njih dostopamo tudi s pomočjo značilk. Vsakič ko v aplikaciji spremenimo leto ali mesec transakcije, se sedaj prenesejo v aplikacijo le transakcije, ki ustrezajo temu letu in mesecu.

4.5 Kratek pregled orodij za vizualizacijo grafov

Ob gradnji aplikacije smo črpali ideje tudi iz že obstoječih knjižnic in programov za vizualizacijo. Za konec si na kratko poglejmo, katera orodja so primerna za vizualizacijo grafov.

4.5.1 Gephi

Gephi je odprtokodni program, ki je namenjen vizualizaciji in analizi grafov. Program je napisan v Javi. Gephi je uporabniku prijazen program, tako da se uporabnik hitro znajde v njem. Kako Gephi vidi uporabnik, lahko vidimo na sliki 4.5. Program sprejema večino znanih formatov za shranjevanje grafov (.gexf, .gml in druge). Vanj lahko uvozimo tudi Neo4j bazo, vendar pa mora biti ta baza v starejši različici Neo4j-ja. Program nam omogoča uporabo raznih filtrov, če se želimo znebiti odvečnih povezav oziroma vozlišč. Spreminjamo lahko barvo in velikost vozlišč na podlagi atributa vozlišča oziroma



Slika 4.5: Gephijev grafični vmesnik.

povezave. Program vsebuje tudi algoritem za razvrščanje v gruče. Izbiramo lahko med celo vrsto različnih razporeditev vozlišč. Za vsako razporeditev imamo na voljo še dodatne lastnosti, s katerimi lahko razporeditev še dodatno spreminjamo. Po grafu lahko rišemo in s tem pobarvamo vozlišča kakor želimo. Program ima na voljo še kopico algoritmov, kot je recimo iskanje najkrajše poti med dvema vozliščema. Na voljo imamo ogromno statističnih podatkov o samem grafu.

Obstaja tudi nabor Gephi knjižnic za Javo, s pomočjo katerih lahko uporabnik zgradi graf v svojem programu in uporablja že pripravljene algoritme. Naše podatke smo poskusili vizualizirati s pomočjo teh knjižnic. Uspelo nam je izrisati le osnovni nivo grafa, nato pa se je postopek ustavil, saj je dokumentacija izjemno nerazumljivo napisana.

Potrebno je omeniti, da ima program še vedno razmeroma veliko programskih napak, kljub temu pa delo z Gephijem večji del poteka nemoteno.

Lahko rečemo, da je Gephi, kljub svoji preprostosti, zelo močno orodje za vizualizacijo in analizo grafov. Vizualizacija mogoče estetsko res ni najlepša, vendar pa je izjemno lahko berljiva.

4.5.2 JUNG

JUNG je ogrodje za izdelovanje grafov v Javi. Je odprtokodna programska knjižnica, ki omogoča modeliranje, analizo in vizualizacijo podatkov, ki jih lahko predstavimo kot graf.

JUNG je zasnovan tako, da podpira različne predstavitve podatkov in povezav med njimi. Podatke lahko predstavimo kot usmerjen ali neusmerjen graf, hipergraf, večmodalen graf ali graf s paralelnimi povezavami. JUNG nam omogoča, da grafom, entitetam in razmerjem dodajamo metapodatke. Z uporabo metapodatkov lahko JUNG uporabimo za razvoj analitičnih orodij za zelo kompleksne množice podatkov.

JUNG ima na voljo tudi mnogo algoritmov iz teorije grafov, podatkovnega rudarjenja in socialnih omrežij.

Z ogrodjem JUNG lahko podatke tudi vizualiziramo. Vizualizacija podatkov je dokaj enostavna, samo orodje pa nam omogoča več različnih razporeditev (layout) podatkov. Ogrodje omogoča uporabniku, da napiše razporeditev vozlišč po meri. Uporabnik ima na voljo različne filtre, ki mu omogočajo, da se osredotoči le na podatke, ki ji želi analizirati.

4.5.3 D3.js

D3.js je JavaScript knjižnica za manipulacijo dokumentov na osnovi podatkov. D3 podatke vizualizira z uporabo HTML-ja, predlog, ki določajo izgled spletnih strani (CSS) in stopnjevanih vektorskih slik (SVG). Knjižnica je namenjena vizualizaciji različnih vrst podatkov in se ne omejuje le na vizualizacijo grafov. V samo knjižnico se nismo poglabljali. Omenili smo jo zato, ker je vizualizacija zelo lepa na pogled in interaktivna. Več o sami knjižnici si lahko ogledate na [14].

Poglavje 5

Zaključek

V tem diplomskem delu smo si pogledali grafovne podatkovne baze. Videli smo njene prednosti pred ostalimi podatkovnimi bazami. Ogledali smo si tudi, kako deluje avtohtona grafna podatkovna baza (Neo4j). Osredotočili smo se predvsem na to, kako tovrstna grafna podatkovna baza shranjuje in obdeluje podatke. Poskušali smo pokazati, zakaj je tako delovanje podatkovne baze boljše in hitrejše. Grafne podatkovne baze se zaradi svojega načina delovanja zelo dobro soočajo z močno povezanimi podatki, kar za druge podatkovne baze predstavlja težave.

Prednosti grafnih podatkovnih baz smo izkoristili tudi sami, saj smo izdelali aplikacijo, ki ima svoje podatke shranjene v Neo4j grafni podatkovni bazi. Aplikacija omogoča preprosto vizualizacijo grafa. V aplikaciji imamo na voljo tudi nekaj filtrov in drugih orodjih, ki poskrbijo, da so podatki preglednejši.

Pri izdelavi diplomskega dela smo se naučili veliko novega, predvsem s področja grafnih podatkovnih baz, ki nam je bilo prej skoraj nepoznano. Dobro smo osvojili tudi osnovne grafične knjižnice v Javi. Izdelava aplikacije za vizualizacijo grafov, brez dodatnih knjižnic, je bila kar velik zalogaj. Aplikacija ima še mnogo prostora za izboljšanje. Imeli smo veliko zamisli, vendar časa je bilo premalo. Kaj smo imeli v mislih, pa si poglejmo v naslednji sekciji.

5.1 Nadaljne delo

Čeprav je izdelava aplikacije z naše strani zaključena, ima še mnogo prostora za izboljšanje. V aplikacijo bi lahko vgradili algoritem za razporeditev vozlišč, kar bi prineslo lepši izgled samega grafa in tudi večjo preglednost. Dodali bi lahko tudi algoritme za gručenje, ki bi graf združili po gručah. Iz tovrstnega grafa bi bilo možno hitro ugotoviti, okoli katerih vozlišč se pretaka največ denarja. To sta dve večji izboljšavi same vizualizacije, ki pa zahtevata kar nekaj dela.

Literatura

[1] Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Wildom. *Database systems - The complete book, 2nd edition*, Department of Computer Science, Standford University, 2009

[2] Database. `http://en.wikipedia.org/wiki/Database` (Dostop 6.11.2013)

[3] Eric Evans. *Domain-Driven Design*, Addison-Wesley, 2004

[4] *Amazon DynamoDB* [online]. Dostopno na: `http://aws.amazon.com/dynamodb`

[5] Jeffrey Dean, Sanjay Ghemawat. *MapReduce: Simplied Data Processing on Large Clusters* [online]. Dostopno na : `http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/sl//archive/mapreduce-osdi04.pdf`

[6] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber. *Bigtable: A Distributed Storage System for Structured Data* [online]. Dostopno na: `http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/sl//archive/bigtable-osdi06.pdf`

[7] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases*, O'Reilly Media, 2013

[8] Graph theory. http://en.wikipedia.org/wiki/Database (Dostop 12.11.2013)

[9] Richard J. Trudeau. *Intoduction To Graph Theory*, Dover, 1993

[10] Jonas Partner, Aleksa Vukotic, and Nicki Watt. *Neo4j in Action*, Manning publication co., 2013

[11] Projekt Transparentnost https://www.kpk-rs.si/sl/projekt-transparentnost (Dostop 17.4.2013)

[12] Opis kopije podatkovne baze mesečnih transakcij iz aplikacije supervizor https://www.kpk-rs.si/upload/datoteke/Opis_kopije_podatkov_Supervizor_do_okt-2013.pdf (Dostop 6.11.2013)

[13] Pramod J. Sadalage, Martin Fowler. *NoSQL Distilled*, Addison-Wesley, 2013

[14] Data-Driven Documents. http://d3js.org (Dostop 2.12.2013)