UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

MAGISTRSKO DELO

GRAFNE PODATKOVNE BAZE S PRIMERI UPORABE V ZAVAROVALNIŠTVU

Ljubljana, februar 2020 ŠPELA ŠKVARČ

IZJAVA O AVTORSTVU

Podpisana Špela Škvarč, študentka Ekonomske fakultete Univerze v Ljubljani, avtorica predloženega dela z naslovom Grafne podatkovne baze s primeri uporabe v zavarovalništvu, pripravljenega v sodelovanju s svetovalcem prof. dr. Mihaelom Permanom

I Z J A V L J A M

1. da sem predloženo delo pripravila samostojno;

2. da je tiskana oblika predloženega dela istovetna njegovi elektronski obliki;

3. da je besedilo predloženega dela jezikovno korektno in tehnično pripravljeno v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani, kar pomeni, da sem poskrbela, da so dela in mnenja drugih avtorjev oziroma avtoric, ki jih uporabljam oziroma navajam v besedilu, citirana oziroma povzeta v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani;

4. da se zavedam, da je plagiatorstvo – predstavljanje tujih del (v pisni ali grafični obliki) kot mojih lastnih – kaznivo po Kazenskem zakoniku Republike Slovenije;

5. da se zavedam posledic, ki bi jih na osnovi predloženega dela dokazano plagiatorstvo lahko predstavljalo za moj status na Ekonomski fakulteti Univerze v Ljubljani v skladu z relevantnim pravilnikom;

6. da sem pridobila vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v predloženem delu in jih v njem jasno označila;

7. da sem pri pripravi predloženega dela ravnala v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobila soglasje etične komisije;

8. da soglašam, da se elektronska oblika predloženega dela uporabi za preverjanje podobnosti vsebine z drugimi deli s programsko opremo za preverjanje podobnosti vsebine, ki je povezana s študijskim informacijskim sistemom članice;

9. da na Univerzo v Ljubljani neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve predloženega dela v elektronski obliki, pravico reproduciranja ter pravico dajanja predloženega dela na voljo javnosti na svetovnem spletu preko Repozitorija Univerze v Ljubljani;

10. da hkrati z objavo predloženega dela dovoljujem objavo svojih osebnih podatkov, ki so navedeni v njem in v tej izjavi.

V Ljubljani, dne _____ Podpis študentke: _____

KAZALO

UVOD **4**
1 PODATKOVNE BAZE **6**
1.1 Sistemi za upravljanje podatkovnih baz **6**
1.2 Zastopanost SUPB po kategorijah **8**
1.3 Relacijske podatkovne baze **10**
1.4 NoSQL podatkovne baze **11**
1.4.1 Skalabilnost 11
1.4.2 Cena 13
1.4.3 Prilagodljivost 13
1.4.4 Razpoložljivost 13
1.5 Tehnologija distribucijskih modelov **13**
1.6 Načelo BASE in CAP teorem **14**
1.7 Kategorije NoSQL podatkovnih baz **16**
1.7.1 Dokumentne shrambe 16
1.7.2 Ključ-vrednost shrambe 17
1.7.3 Shrambe družin stolpcev 18
1.7.4 Grafne podatkovne baze 20
1.8 Primerjava SUPB **20**
1.9 Agregirane shrambe **22**
2 GRAFNE PODATKOVNE BAZE **23**
2.1 Možnosti shranjevanja zelo povezanih podatkov **24**
2.2 Delitev področja grafov **28**
2.3 Prednosti grafnih podatkovnih baz **30**
2.3.1 Zmogljivost 30
2.3.2 Prilagodljivost 31
2.3.3 Agilnost 31
2.4 Poizvedovalni jeziki **31**
2.4.1 Cypher 32
2.4.1.1 Indeksi in omejitve *34*
2.4.2 Druge možnosti poizvedovanja 35

2.5 Način delovanja podatkovne baze Neo4j ... 36
2.6 Avtohtono obdelovanje ... 36
2.7 Avtohtono shranjevanje ... 37
2.8 Primerjava grafnih podatkovnih baz ... 39
2.9 Uporaba grafnih podatkovnih baz ... 40
3 UPORABA V ZAVAROVALNIŠTVU ... 41
3.1 Zloraba identitete ... 44
3.2 Prometne nesreče ... 49
3.3 Elektronsko poslovanje ... 53
SKLEP ... 56
LITERATURA IN VIRI ... 57
PRILOGE ... 61

KAZALO SLIK

Slika 1: Trend priljubljenosti podatkovnih baz ... 5
Slika 2: Trg sistemov za upravljanje podatkovnih baz ... 8
Slika 3: Število SUBP po kategorijah, november 2019. ... 9
Slika 4: SUPB po kategorijah v odstotkih, november 2019 ... 9
Slika 5: Vertikalna in horizontalna razširljivost ... 12
Slika 6: Ključ-vrednost shramba kot porazdeljena razpršena struktura ... 18
Slika 7: Štiri primeri grajenja blokov v shrambah družin stolpcev ... 19
Slika 8: Primernost kategorij glede na velikost in kompleksnost podatkov ... 21
Slika 9: Problem sedmih mostov Königsberga ... 24
Slika 10: Primer grafne podatkovne baze Twitterja ... 24
Slika 11: Primer povezovanja v relacijski podatkovni bazi ... 25
Slika 12: Primer povezovanja v agregirani shrambi ... 26
Slika 13: Priljubljenost grafnih podatkovnih baz ... 28
Slika 14: Grafni vzorec ... 32
Slika 15: Neavtohtona obdelava ... 36
Slika 16: Struktura zapisov v shrambi vozlišč in razmerij ... 38
Slika 17: Shranjevanje grafa ... 39
Slika 18: Večplastni pristop odkrivanja goljufij ... 42
Slika 19: Lažne identitete ... 44
Slika 20: Iskanje vzorcev ... 45
Slika 21: Shema zavarovanca ... 46
Slika 22: Primer grafa za odkrivanje sintetičnih identitet ... 46
Slika 23: Iskanje skupnih podatkov – prva poizvedba ... 47

Slika 24: Iskanje skupnih podatkov – druga poizvedba 48
Slika 25: Izpis krogov goljufov, sortiranih po številu skupnih podatkov..... 49
Slika 26: Shema avtomobilskih goljufij 50
Slika 27: Shema prometnih nesreč 50
Slika 28: Rezultat poizvedbe o vpletenih osebah prometne nesreče 51
Slika 29: Pogostost vpletenosti v vlogi priče prometne nesreče 52
Slika 30: Goljufije pri elektronskem poslovanju 54

KAZALO PRILOG

Priloga 1: Cypher – kreiranje grafa s sintetičnimi identitetami 1
Priloga 2: Cypher – kreiranje grafa s prometnimi nesrečami 2

SEZNAM KRATIC

ang. – angleško

CRM – (ang. Customer Relationship Management); upravljanje odnosov s strankami

CRUD – (ang. Create, Read, Update, Delete); ustvari, preberi, posodobi, izbriši

DSL – (ang. Domain Specific Language); domensko specifični jezik

ER – (ang. Entity-Relationship); entitetno-relacijski

ETL – (ang. Extract, Transfrom, Load); pridobivanje, preoblikovanje, nalaganje

GQL – (ang. Graph Query Language); grafni poizvedovalni jezik

OLAP – (ang. Online Analitical Processing); sprotna analitična obdelava

OLTP – (ang. Online Transactional Processing); sprotna obdelava transakcij

RDF – (ang. Resource Description Framework); ogrodje za opisovanje virov

REST – (ang. Representational State Transfer); arhitektura za izmenjavo podatkov med spletnimi storitvami

SQL – (ang. Structured Query Language); strukturirani povpraševalni jezik

SUPB – (ang. Database Management System); sistem za upravljanje podatkovnih baz

UVOD

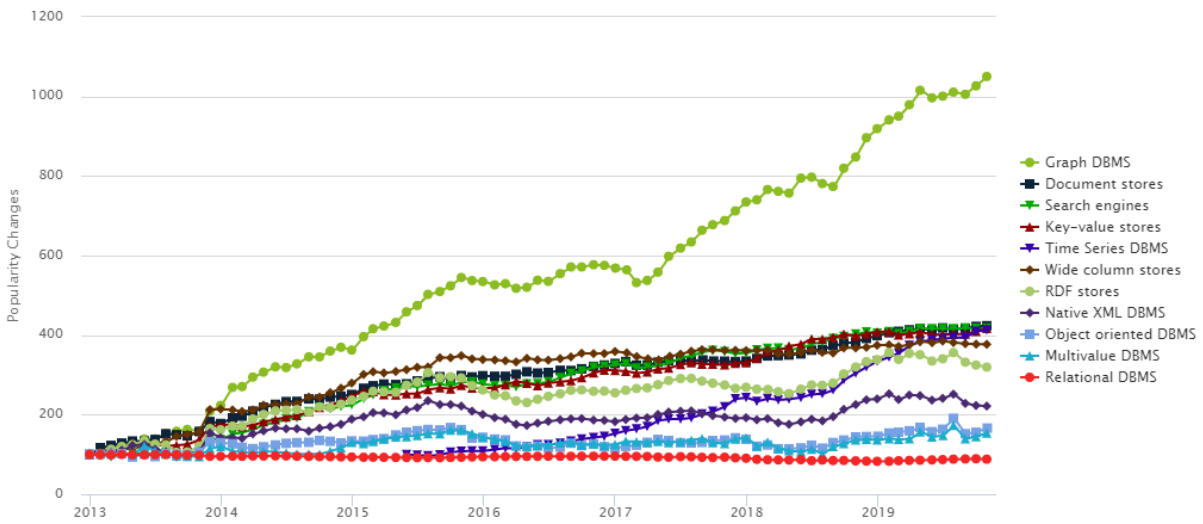
Povezanost je najbolj razširjena značilnost današnjih omrežij in sistemov. Od interakcij beljakovin do socialnih omrežij, od komunikacijskih sistemov do električnih omrežij in od maloprodajnega poslovanja do oskrbovalnih verig – niti najbolj preprosta omrežja niso naključna, kar pomeni, da povezave niso enakomerno porazdeljene niti statične. Zato preprosta statistična analiza ne more opisovati, kaj šele napovedovati vedenja v povezanih sistemih. Ker svet postaja vse bolj povezan in sistemi postajajo vse bolj zapleteni, je nujno, da uporabljamo tehnologije, ki so zgrajene na odnosih oziroma razmerjih in njihovih dinamičnih značilnostih. Podatki v omejenem obsegu nimajo pomembne vrednosti, ko pa se jih združi z milijoni drugih podatkov, dobimo vzorce, ki imajo veliko moč. In bolj kot razumemo povezave med njimi in več povezav kot najdemo med podatki, bolj nam lahko pri poslovanju pomagajo.

Podatke shranjujemo v podatkovne baze, ki niso nič drugega kot kolekcije podatkov, s katerimi upravljajo sistemi za upravljanje podatkovnih baz. Najbolj razširjene in ene najpogostejših oblik za shranjevanje podatkov so relacijske podatkovne baze. Podatke shranjujejo v obliki tabel, torej v vrstice in stolpce, kjer so podatki lahko razumljivi in enostavni za pregledovanje. Prvi komercialno dostopni sistem upravljanja relacijskih podatkovnih baz, Oracle, je bil izdan že leta 1979 in je še danes med prevladujočimi za namene shranjevanja in pridobivanja podatkov. Ampak ker se obseg shranjevanja podatkov iz dneva v dan povečuje, sta njihovo razumevanje in način predstavitve vedno bolj kompleksna. Povezani podatki so podatki, katerih interpretacija in vrednost zahtevata, da najprej razumemo načine, kako so povezani njihovi elementi. Da ustvarimo to razumevanje, moramo poimenovati in kvalificirati povezave med stvarmi. Posledično so se začele razvijati grafne podatkovne baze in čeprav so velike korporacije že pred časom spoznale in začele uvajati lastne tehnologije za obdelavo grafov, smo zdaj v obdobju, ko se je ta tehnologija demokratizirala. Danes grafne podatkovne baze uporabnikom omogočajo, da izkusijo prednosti povezanih podatkov, ne da bi vlagali v izgradnjo lastne grafne infrastrukture.

Zanimivo je, da sama teorija grafov ni nova, ampak se je začela razvijati že v 18. stoletju. Šele v zadnjem desetletju pa se je bila razvita tudi na področju upravljanja informacij in podatkov. V začetku so večji poudarek grafnim podatkovnim bazah dala predvsem komercialno zelo uspešna podjetja, kot so Facebook, Google in Twitter, danes pa veliko zanimanje zanje izkazujejo podjetja iz vseh področij (Robinson, Webber & Eifrem, 2015). Po besedah direktorja vodilne in trenutno najbolj uveljavljene grafne podatkovne baze Neo4j, je njenih uporabnikov večina vodilnih trgovcev na drobno, kot sta Walmart in Ebay, polovica največjih družb finančnih storitev ter večina največjih prodajalcev programske opreme (Barker, 2017). Ocenjeno je, da so grafne podatkovne baze, glede na stopnjo razvoja, trenutno na ravni relacijskih podatkovnih baz v drugi polovici osemdesetih let prejšnjega stoletja. V sedemdesetih letih se je razvoj relacijskih podatkovnih baz začel, v devetdesetih

se je začel razvoj sedaj standardiziranega povpraševalnega jezika (ang. structured query language) (v nadaljevanju SQL) za relacijske podatkovne baze. To pomeni, da so grafne podatkovne baze šele v začetni fazi razvoja in lahko na tem področju pričakujemo še veliko izboljšanj in vedno več zanimanja zanje. Tudi podatki s spletne strani DB-engines, ki prikazujejo trenutne trende vseh podatkovnih baz in shramb, nam kažejo daleč največjo priljubljenost grafnih podatkovnih baz v preteklih šestih letih, kot nam prikazuje slika 1. Vse kaže, da tu ne bo šlo samo za modno muho, saj se trend zanimanja in razvoja še vedno povečuje.

Slika 1: Trend priljubljenosti podatkovnih baz



Vir: DB-engines (2019).

Magistrska naloga je v grobem razdeljena na tri dele. V prvem delu naloge bomo obrazložili celoten potek razvoja podatkovnih baz. Natančneje si bomo kasneje ogledali predvsem grafne podatkovne baze, ki pa so le ene izmed nerelacijskih baz. Te so se začele razvijati v začetku 21. stoletja prav zaradi vse večjega obsega podatkov, ki se hitro spreminjajo in tradicionalni, relacijski sistemi za upravljanje podatkovnih baz zanje niso bili več primerni (Sullivan, 2015). Ta skupina nerelacijskih sistemov je poznana pod imenom NoSQL, ki se ga lahko razlaga kot »Non SQL« oziroma »Not Only SQL«. Med njih uvrščamo dokumentne shrambe, ključ-vrednost shrambe, shrambe družin stolpcev ter grafne podatkovne baze. Relacijske podatkovne baze in vsako izmed NoSQL podatkovnih baz bomo v prvem delu predstavili, opisali možnosti uporabe, njihove prednost in slabosti, jih primerjali med seboj in z grafnimi podatkovnimi bazami.

V drugem delu magistrske naloge se bomo osredotočili na odprtokodno grafno podatkovno bazo Neo4j, ki je bila razvita v Neo4j, Inc. in je prvi in najvidnejši predstavnik grafnih podatkovnih baz, ki je popolnoma prilagojen za delo z grafi, tako pri obdelovanju podatkov kot samem shranjevanju podatkov (Robinson, Webber & Eifrem, 2015). Na kakšen način to doseže, bomo opisali v razdelkih o avtohtonem obdelovanju ter avtohtonem shranjevanju.

Začeli pa bomo v tem drugem poglavju na splošno, z raziskovanjem o možnostih, kako in kam naj shranjujemo zelo povezane podatke ter obrazložili glavne prednosti grafnih podatkovnih baz. V tem delu bomo navedli in opisali tudi predstavnike poizvedovalnih jezikov, ki jih lahko uporabljamo na grafnih podatkovnih bazah in izpostavili problem, da še nimamo standardiziranega grafnega jezika, ki bi bil kot SQL za relacijske podatkovne baze. Navedli bomo še nekaj drugih ponudnikov grafnih podatkovnih baz in lastnosti, na katere moramo biti pozorni, ko se odločamo o tem, katero grafno podatkovno bazo bi uporabili za svoj projekt. V zadnjem razdelku drugega dela predstavimo nekaj možnosti uporabe grafnih podatkovnih baz.

Tretji del magistrske naloge je namenjen predstavitvi uporabe grafnih podatkovnih baz v zavarovalništvu, predvsem na področju razpoznave goljufij. Te namreč zavarovalnicam predstavljajo velik problem, saj tu ne gre samo za manjše, priložnostne goljufije, ampak za organizirane obroče goljufov, ki jih zavarovalnice težko odkrijejo, četudi ti dejanja lahko ponavljajo. Za Slovenijo velja letna ocena za zavarovalniške goljufije v višini dobrih sto milijonov evrov (Košir, 2017), za Združene države Amerike pa več kot osemdeset milijard dolarjev (Coalition Against Insurance Fraud, 2018). Še v praksi se bomo spoznali z jezikom Cypher, s pomočjo katerega bomo kreirali primere grafov, po katerih bomo poizvedovali in predstavili načine, kako po bazi raziskujemo in iščemo sumljive vzorce, ki lahko razkrijejo goljufije.

1 PODATKOVNE BAZE

Podatkovne baze in tehnologija podatkovnih baz so ključnega pomena za sodobne organizacije, ki morajo omogočati dnevno poslovanje in učinkovito odločanje. Tehnologija podatkovnih baz je doživela izjemen razvoj v zadnjih 50 letih. Kljub temu, da Oracle še vedno prevladuje med sistemi za upravljanje podatkovnih baz, industrija ostaja zelo konkurenčna z nadaljnjo visoko stopnjo inovativnosti.

1.1 Sistemi za upravljanje podatkovnih baz

Definirajmo najprej, kaj točno sistem za upravljanje podatkovnih baz pomeni in predstavlja. Sistem za upravljanje podatkovnih baz (ang. Database Management System) (v nadaljevanju SUPB (ang. DBMS)), je programska oprema, ki uporabnikom omogoča definiranje, kreiranje in vzdrževanje podatkovne baze ter hkrati zagotavlja nadzorovan dostop do podatkovne baze (Lonarčić in drugi, 2002). Je programska oprema, ki ravna z vsemi dostopi do podatkov v podatkovni bazi in je odgovorna za izvajanje postopkov in preverjanje avtorizacije uporabnikov in veljavnosti podatkov. SUPB je torej zbirka programov, ki:

- omogoča kreiranje nove podatkovne baze in definiranje njene strukture,
- omogoča učinkovito poizvedovanje in spreminjanje podatkov,
- varuje podatke pred nesrečami in neavtoriziranimi dostopi,

– nadzoruje sočasen dostop večjega števila uporabnikov do podatkov.

V splošnem so glavne funkcije SUPB torej naslednje: omogočanje definiranja in spreminjanja opisa podatkovne baze ter njeno kreiranje, manipuliranje s podatki, izvajanje transakcij, zaščita podatkov, sledenje dogodkom, zagotavljanje integritete podatkov, vzdrževanje podatkovnega slovarja, zagotavljanje podatkovne neodvisnosti in izvajanje nadzora hitrosti.

Glavna obdobja v razvoju tehnologije podatkovnih baz so naslednja:

– Obdobje pred pojavitvijo podatkovnih baz (do 60 let prejšnjega stoletja). To je bil čas datotek, ki omogočajo direktni dostop do podatkov in trajno hranjenje razmeroma velikih količih podatkov. Edini pogled na podatke je fizični in vse operacije nad podatki so prilagojene fizični predstavitvi. To obdobje zaznamuje rast števila aplikacij in s tem narašča število deljenih datotek in različnih formatov zapisa, kar vodi do redundance, nekonsistentnosti in težav s sledenjem dogodkom.
– 1. obdobje (konec 60 let). Predstavlja začetek komercialne uporabe podatkovnih baz, uvajajo se mehanizmi za opis podatkov – hierarhični in mrežni podatkovni model. Prvič se pojavlja delitev pogledov na podatke – fizični in logični model. Beleženje dogodkov v sistemu postane avtomatizirano.
– 2. obdobje (80 leta). Postavi se koncept relacijskega podatkovnega modela (pionir na tem področju je bilo podjetje IBM), vpelje se pojem relacija. Posledično se predstavi entitetno-relacijski (ER) diagram in razvije deklarativni jezik, namenjen delu z relacijsko podatkovno bazo, imenovan Structured Query Language – SQL, ki postane standard v poizvedovanju nad relacijskimi podatkovnimi bazami. Največje uspehe doseže DB2, razvit pri IBM-u.
– 3. obdobje (začetek 90 let). Uporaba novih, objektno relacijskih in objektnih podatkovnih modelov sega na specializirana področja, ki zahtevajo procesiranje zelo kompleksnih podatkov. Aplikacije za delo s podatkovnimi bazami se širijo tudi na področje svetovnega spleta in e-poslovanja.
– 4. obdobje (konec 90 let). Začne se obdobje objektnih in objektno-relacijskih podatkovnih baz, kar prinese razvoj novih aplikacij za sprotno analitično obravnavo (ang. online analitical processing) (v nadaljevanju OLAP). Močno pride do izraza podatkovna baza MySQL. Po letu 2000 se začnejo pojavljati baze NoSQL, kjer ne gre več samo za hranjenje podatkov, ampak se uveljavijo sistemi kot že zgoraj omenjeni OLAP, podatkovno rudarjenje, podatkovno skladiščenje, XML podatkovne zbirke in t. i. aktivne podatkovne baze, ki so se sposobne same odzivati na zunanje dogodke.

Kljub prevladujočemu položaju, ki ga ima Oracle na trgu SUPB, industrija ostaja zelo konkurenčna. Na primer, na področju relacijskih SUBP so njegovi konkurenti Microsoft SQL Server, IBM DB2, Teradata, SAP Sybase. V zadnjem času komercialne SUPB vedno bolj izpodbijajo odprtokodni produkti SUBP. Na tem področju je vodilni MySQL, ki mu sledijo MongoDB, PostgreSQL in MariaDB.

Slika 2: Trg sistemov za upravljanje podatkovnih baz

Vir: Visual-Paradigm (brez datuma).

Napredek SUPB v zadnjih letih podpira poslovno inteligenco za integracijo podatkov in uporabo zbirnih podatkov. NoSQL tehnologija je bila razvita za potrebe Big Data in se od začetkov v letu 2009 hitro razvija. V veliki meri se relacijske sisteme v zadnjem času želi menjati za NoSQL podatkovne baze z naslednjimi lastnostmi:

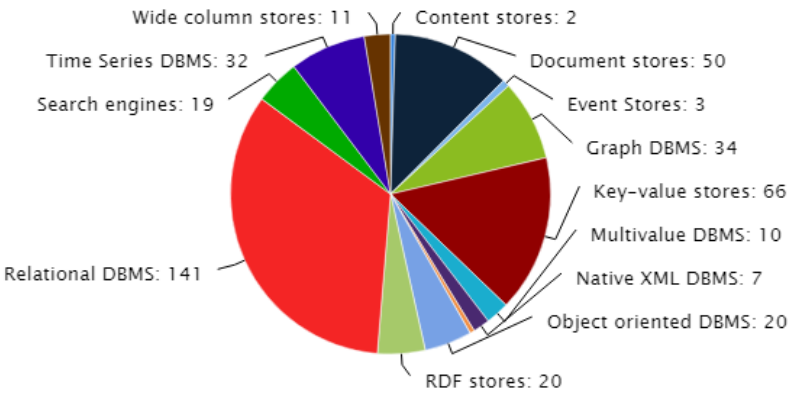
- zmožnost horizontalnega skaliranja čez več strežnikov,
- zmožnost replikacije in particioniranja podatkov čez več strežnikov,
- šibkejši model nadzora nad sočasnim izvajanjem transakcij kot pri ACID transakcijah,
- zmožnost dostopa s preprostim API vmesnikom,
- zmožnost dinamičnega dodajanja novih atributov trenutnim zapisom.

Več o teh lastnostih si bomo ogledali v poglavjih, ki sledijo.

1.2 Zastopanost SUPB po kategorijah

V tem poglavju si bomo ogledali lestvice, pridobljene iz DB-engines (2019), ki zagotavljajo informacije o aktualni priljubljenosti več kot 350 sistemov za upravljanje s podatkovnimi bazami. Merila vključujejo število rezultatov iskanja po imenu nekega sistema, Google trende, razprave s foruma Stack Overflow, število ponudb delovnih mest z navedbami določenih sistemov, število profilov iz profesionalnih omrežjih (kot npr. LinkedIn) z navedbo določenih sistemov in omembe v socialnih omrežjih (kot npr. Twitter). Lestvice z razvrstitvijo se posodabljajo mesečno.

Slika 3: Število SUBP po kategorijah, november 2019.

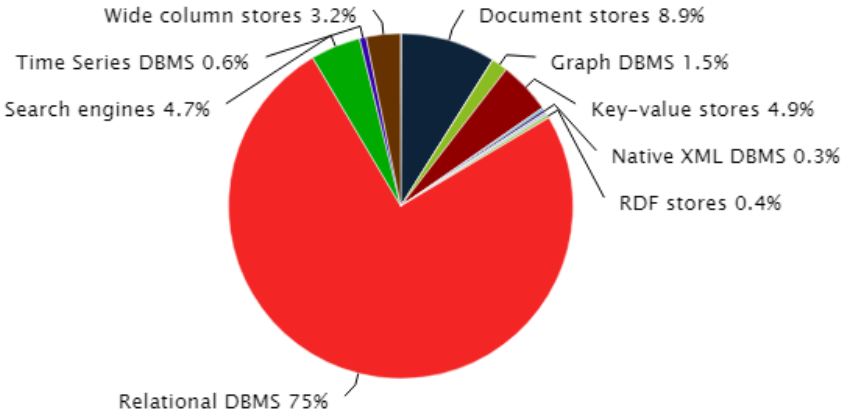


Vir: DB-engines (2019).

DB-engines prikazuje 357 različnih shramb in SUPB za mesec november 2019, ki so razvrščeni po kategorijah (relacijski SUPB, ključ-vrednost shrambe, dokumentne shrambe, itd.). Na zgornjem krožnem grafikonu na sliki 3 je prikazano število SUPB glede na vsako izmed kategorij. Nekateri sistemi spadajo pod več kategorij. Največje število sistemov spada pod relacijske SUBP (ang. relational DBMS), ki jih je kar 141, sledijo jim ključ-vrednost shrambe (ang. key-value stores) s 66 in dokumentne shrambe (ang. document stores) s 50 predstavniki ter grafni SUPB (ang. graph DBMS) s 34 predstavniki.

Če pa bi namesto števila želeli opazovati priljubljenost vsake kategorije, se seznam sistemov skrči. Spodnji krožni grafikon na sliki 4 prikazuje torej priljubljenost posameznih sistemov na kategorijo. Vsota rezultatov razvrstitve je enaka 100 %. Večina uporabnikov uporablja relacijske SUPB in to s 75 %, sledijo jim dokumentne shrambe z 8.9 %, ključ-vrednost shrambe s 4.9 %, iskalniki (ang. search-engines) s 4.7 %, shrambe družin stolpcev oziroma shrambe s širokimi stolpci (ang. wide-column stores) s 3.2 % in grafni SUPB z 1.5 %.

Slika 4: SUPB po kategorijah v odstotkih, november 2019



Vir: DB-engines (2019).

Jasno je razvidno, da so relacijski SUPB tisti, katerih uporaba je privzeta. Vendar se stanje spreminja z novostmi v tehnologiji podatkovnih baz in čeprav so odstotki priljubljenosti NoSQL podatkovnih baz majhni, so to nove podatkovne baze, ki se še razvijajo in vseeno zanimanje zanje hitro narašča, na kar kažejo tudi historični podatki iz DB-engines (razvidno na sliki 1).

1.3 Relacijske podatkovne baze

Relacijske podatkovne baze se množično uporabljajo za shranjevanje podatkov. Njihov podatkovni model je zelo enostaven za razumevanje in sestavljen iz tabel, ki predstavljajo logični pogled na podatke. Sestavljene so iz stolpcev ali atributov in vrstic ali zapisov ter imajo definirana medsebojna razmerja. Vsak atribut ima pripadajočo domeno, ki določa podatkovni tip vrednosti, ki jih atribut lahko zavzame in integritetne omejitve za zagotavljanje celovitosti podatkov (npr. vrednost atributa ne sme biti prazna, vrednost atributa so lahko samo cela števila). Vsaka vrstica iz tabel je enolično določena s primarnim ključem, s čimer se preprečuje podvajanje podatkov. Ta primarni ključ je hkrati tudi indeks, tj. struktura, ki omogoča hiter dostop do vrstic tabele med poizvedovanjem.

Tabele se med seboj povezujejo z uporabo tujih ključev. Primarni ključ nadrejene tabele nastopa v podrejeni tabeli kot tuji ključ. Za zagotavljanje celovitosti podatkov v povezanih tabelah se uporablja referenčna integriteta, ki zahteva, da tuji ključ v podrejeni tabeli lahko zavzame le vrednosti, ki jih zavzema primarni ključ v nadrejeni tabeli (Knez, 2012, str. 8).

Podvojitve podatkov v podatkovnem modelu preprečujemo z normalizacijo, tako da delimo stolpce v več tabel in uporabljamo tuje ključe za povezave. S tem pa se podatkovni model širi in postaja kompleksnejši in zahtevnejši za razumevanje.

K priljubljenosti relacijskih podatkovnih baz je gotovo pripomogel poizvedovalni jezik SQL (ang. structured query language), ki je najbolj podprt standardni jezik za podatkovne baze. Je funkcijsko zelo bogat in uporablja preprosto, deklarativno sintakso. Nastopa lahko kot:

- jezik za definiranje podatkovni struktur in tipov (ang. data definition language DDL), ki zagotavlja ukaze za definiranje relacijske sheme, oblikovanje indeksov in definiranje pogledov (ang. create), brisanje relacij (ang. drop), spreminjanje relacijske sheme (ang. alter) in dodeljevanje pravic dostopa (ang. grant),
- jezik za manipulacijo s podatki (ang. data manipulation language DML), ki vključuje ukaze za vnos (ang. insert), brisanje (ang. delete) in posodabljanje (ang. update) ali
- poizvedovalni jezik (ang. query language), ki omogoča poizvedovanje nad eno ali več tabelami (Knez, 2012, str. 10).

1.4 NoSQL podatkovne baze

NoSQL podatkovne baze so se začele razvijati zaradi vse večjega in hitro spreminjajočega obsega podatkov, za katere tradicionalni relacijski SUPB niso bili več primerni (Sadalage & Fowler, 2013). Relacijske podatkovne baze imajo omejeno razširljivost in problem, da je razpoložljivost pomembnejša od doslednosti. Težko obvladujejo veliko istočasnih uporabnikov in veliko zahtevnih obdelav. Težave se pojavljajo tudi zaradi preveč kompleksnih in nerazumljivih podatkovnih modelov. Sicer se lahko tudi relacijske baze nadgradi z dodatnimi procesorji, dodatnim pomnilnikom ali hitrejšimi napravami za shranjevanje (t. i. vertikalna razširitev sistema), ampak so to drage opcije, s katerimi lahko dosežemo samo določeno stopnjo, preden pridemo spet do istih težav. Obstajajo namreč omejitve glede števila procesorjev in pomnilnika na enem strežniku. Druga možnost je ta, da bi oblikovalci relacijskih podatkovnih modelov preoblikovali shemo in tako izboljšali učinkovitost. To se do neke mere lahko doseže z denormalizacijo, ki pa povečuje tveganje podatkovnih anomalij. Še ena možnost bi bila uporaba več strežnikov, kar je mogoče, vendar je delovanje enega relacijskega SUPB na več strežnikih zahtevno in otežuje dolgoročno upravljanje in vzdrževanje. Tu nastopijo tudi možni problemi pri izvajanju transakcij, ki se izvajajo na različnih strežnikih in morajo vsi uspeti ali noben od teh (Sullivan, 2015).

Pritisk vseh omenjenih težav je bil motivacija za razvoj NoSQL podatkovnih baz. Poimenovanje se je prvotno nanašalo na izraz »Non SQL«, kjer je SQL (ang. structured query languange) poizvedovalni jezik, ki ga uporabljamo na relacijskih bazah. Druga možnost obrazložitve imena NoSQL je kot »Not Only SQL«, ki poudarja, da lahko ti sistemi podpirajo tudi jezik poizvedb SQL ali pa se skupaj z relacijsko bazo dopolnjujejo. NoSQL podatkovne baze za razliko od relacijskih ne potrebujejo tabelarične sheme, ne uporabljajo »join« operacij za povezovanje, najpomembnejše pa so jim naslednje značilnosti:

- skalabilnost (ang. scalability),
- cena (ang. cost),
- prilagodljivost (ang. flexibility),
- razpoložljivost (ang. availability).

Odvisno od potreb posamezne aplikacije so lahko nekatere od teh značilnosti pomembnejše od drugih.

1.4.1 Skalabilnost

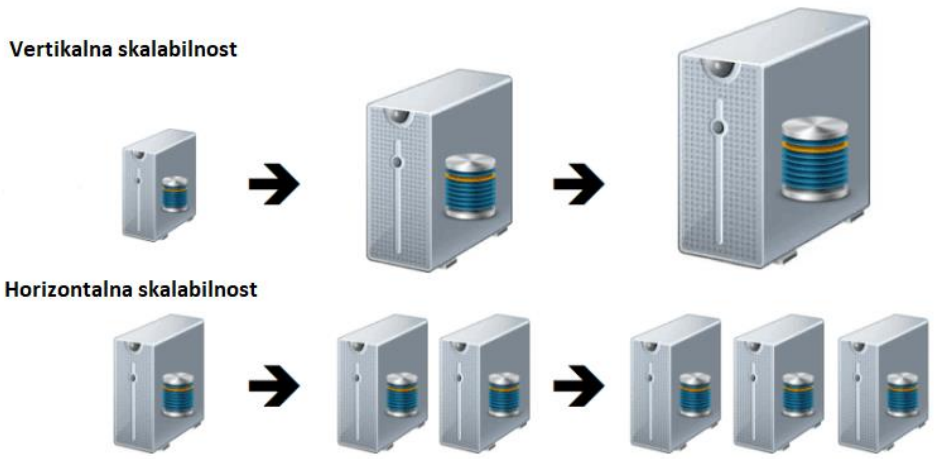
Skalabilnost sistema je sposobnost učinkovitega razporejanja različnih delovnih obremenitev. Govorimo predvsem o možnosti dodeljevanja virov, ki so potrebni za tekoče delovanje aplikacij. Poznamo dva pristopa za zagotavljanje skalabilnosti:

- vertikalna skalabilnost sistema (ang. scale up/scale vertically),
- horizontalna skalabilnost sistema (ang. scale out/scale horizontally).

Vertikalna skalabilnost sodi med preproste načine, s katerimi se je večina sistemskih administratorjev že srečevala v preteklosti. Gre predvsem za dodajanje novih sistemskih virov (npr. procesorska moč, delovni pomnilnik, diskovno polje) obstoječim strežnikom. Izbira tega načina skalabilnosti ne prinaša večje kompleksnosti ali tveganj, saj se koncept delovanja obstoječe aplikacije ne spremeni. Kljub nadgradnji, ki je lahko precej draga, pa lahko kmalu dosežemo nove omejitve (Rojko, 2014, str. 52).

Horizontalna skalabilnost poveča odzivnost aplikacij z možnostjo dodajanja novih vozlišč – strežnikov in tako porazdeli obremenitve oz. zahteve. Horizontalna skalabilnost je bolj fleksibilna od vertikalne, saj se strežnike lahko doda ter odstrani po potrebi in izkaže se, da je ta možnost v veliko primerih tudi cenejša, čeprav se uporablja več strežnikov, saj so ti manjši in manj zmogljivi. Ta možnost je tudi zanesljivejša in odpornejša, saj celoten grozd (ang. cluster) strežnikov lahko nadaljuje delo, kljub temu, da kakšen izmed strežnikov odpove. NoSQL podatkovne baze so namenjene uporabi strežnikov v grozdu. Ko so odstranjeni oz. dodani novi strežniki, NoSQL SUPB sam prilagaja uporabo novega nabora razpoložljivih strežnikov (Sadalage & Fowler, 2013).

Slika 5: Vertikalna in horizontalna razširljivost



Vir: Prirejeno po R Interview Bubble (2018).

Kot omenjeno, relacijske podatkovne baze niso namenjene za uporabo na grozdih. Ob dodatni programski opremi so to nekateri ponudniki sicer želeli doseči, npr. Oracle z Oracle Real Application Cluster (RAC) in Microsoft SQL Server, ki pa delujeta na konceptu skupnega diska v podsistemu. Uporabljata grozdenju prilagojen sistem, ki zapisuje na razpoložljiv sistem, a je zaradi skupnega podsistema še vedno ena sama točka, kjer se lahko zgodijo napake.

1.4.2 Cena

Stroški licenc sistemov za upravljanje podatkovnih baz so očitna skrb za vsako podjetje ali organizacijo. Ponudniki izdelkov uporabljajo različne modele licenciranja, ki vključujejo zaračunavanje glede na velikost strežnika SUPB, število sočasnih uporabnikov podatkovne baze ali število uporabnikov, ki jim je dovoljena uporaba programske opreme. Ti pa se lahko spreminjajo s časom in pride lahko do povpraševanja po večjem številu uporabnikov. Težko se je odločiti tudi o tem, za kakšno število uporabnikov naj podjetje kupi licence, ali naj bo to za največje možno število uporabnikov, povprečno število uporabnikov, naj vnaprej razmišlja o tem, koliko uporabnikov bo uporabljajo sistem čez pol leta ali eno leto. Uporabniki odprtokodne programske opreme se izognejo tem vprašanjem, saj je ta programska oprema brezplačna za uporabo na poljubnem številu strežnikov poljubnih velikosti. Mednje sodi večina NoSQL podatkovnih baz.

1.4.3 Prilagodljivost

Relacijski SUPB so prilagodljivi v vrsti težav, ki jih je mogoče rešiti z uporabo relacijskih podatkovnih modelov. Se pa na začetku projekta pričakuje, da se točno določi vso strukturo tabel in stolpcev, ki so potrebni v aplikaciji. Večina NoSQL SUPB za razliko od relacijskih ne zahteva fiksne strukture podatkovnih modelov, kar posledično omogoča dinamično dodajanje novih atributov.

1.4.4 Razpoložljivost

NoSQL SUPB so sestavljeni iz več manj zmogljivih strežnikov in tudi ko se na enem izmed njih pojavi napaka in je izvzet iz sistema, lahko ostali strežniki v grozdu prevzamejo celotno delovno obremenitev. Izvajanje je lahko nekoliko počasnejše, vendar ostane aplikacija še vedno na voljo. Če se podatkovna baza izvaja na enem samem strežniku in če se tam pojavi napaka, postane aplikacija nedostopna, če nimamo na voljo strežnika za varnostno kopiranje. Če se to zgodi, rezervni strežnik prevzame izvajanje, ki jo je primarni strežnik obdeloval. Ta strežnik sicer ne pomaga pri izvajanju, torej je to precej neučinkovita konfiguracija, saj večino časa ostaja v neuporabi.

1.5 Tehnologija distribucijskih modelov

Ena glavnih lastnosti NoSQL podatkovnih baz je torej zmožnost horizontalnega skaliranja čez več strežnikov. Postavitev takega grozda strežnikov je lahko zelo kompleksna, zato morajo biti koristi tega prepričljive. Glede na distribucijski model lahko dobimo podatkovno shrambo, ki omogoča obvladovanje večjih količin podatkov, možnost obdelave večjih zapisov oz. branj ali večje razpoložljivosti v primeru upočasnitev ali prekinitev omrežja. V splošnem obstajata dva načina porazdelitve: replikacija (ang. replication) in delitev (ang. sharding). Pri replikaciji se isti podatki kopirajo na različna vozlišča – strežnike, pri delitvi

pa gre za razbitje podatkov na posamezna vozlišča (Sadalage & Fowler, 2013). Replikacija v primeru porazdeljenih podatkovnih baz pomeni, da je podatek shranjen v več kot enem vozlišču, kar je zelo koristno pri povečanju zmogljivosti podatkovne baze, saj omogoča porazdeljenost bralnih operacij čez več vozlišč (Knez, 2012, str. 22). Slabost replikacije podatkov so pisalne operacije. Pri replikaciji imamo dve možnosti:

- nadrejeni-podrejeni (ang. master-slave), kjer eno vozlišče prevzame glavno vlogo in upravlja s pisanjem, medtem ko ostali sinhrono z glavnim vozliščem upravljajo z branjem (v primeru okvare na glavnem strežniku baza zavrača vse operacije zapisovanja, dokler ni napaka odpravljena),
- omrežje enakovrednih (ang. peer-to-peer), kjer je pisanje omogočeno na katerokoli vozlišče in vozlišča sinhronizirajo svoje kopije podatkov.

Replikacija nadrejeni-podrejeni zmanjša možnost konfliktov pri posodabljanju, omrežje enakovrednih pa se izogne zapisovanju na samo en strežnik, ki sicer predstavlja eno samo točko okvare.

Pri delitvi, kjer so podatki porazdeljeni po vozliščih, je prednost ta, da se vozlišča lahko doda grozdu z namenom povečanja zmogljivosti bralnih ali pisalnih operacij, brez da je potrebno spreminjati aplikacijo. Lahko se tudi zmanjša velikost grozda, če se količina zahtev zmanjša. Slabost delitve je v tem, da sicer enostavne operacije podatkovne baze (npr. »join« operacija za povezovanje) postanejo kompleksne in neučinkovite (Knez, 2012, str. 22).

Sistem lahko uporablja eno ali kombinacijo obeh tehnik distribucije. Na primer ključ-vrednost shramba Riak uporablja tako replikacijo kot delitev podatkov, ki zagotavljata tolerantnost na napake, visoko razpoložljivost in učinkovitost.

1.6 Načelo BASE in CAP teorem

Pred pojavom NoSQL je bil v relacijskih podatkovnih bazah princip ACID tisti, ki je zagotavljal, da transakcije v podatkovni bazi vodijo iz enega veljavnega stanja v drugo veljavno stanje. Zagotavljati ga je moral SUPB. ACID je kratica za:

- **Atomarnost** (ang. **A**tomicity): Ena transakcija mora biti v celoti izvedena ali v celoti zavrnjena, torej se morajo izvesti vse zahtevane spremembe ali pa nobena.
- **Konsistentnost** (ang. **C**onsistency): Transakcija je veljavna le, če upošteva vse integritetne omejitve podatkovne baze. Če stanje po izvedbi transakcije ni več veljavno, se transakcija zavrne v celoti (vrnemo se v prejšnje veljavno stanje podatkovne baze).
- **Izolacija** (ang. **I**solation): Transakcije se izvajajo neodvisno ena od druge. Delni rezultati transakcije niso vidni drugim transakcijam, dokler ni ta izvedena do konca.
- **Trajnost** (ang. **D**urability): Rezultati uspešno izvedene transakcije so trajno shranjeni. Če jih želimo razveljaviti, moramo to narediti z novo transakcijo, ki privede bazo v prvotno stanje.

Te lastnosti pomenijo, da so ob zaključku transakcije rezultati v bazi konsistentni (ang. write consistency) in stabilni na disku (ali več diskih), kar se za razvijalce lahko sliši odlično. So pa to stroge zahteve in v veliko primerih uporabe se izkaže, da ni potrebno, da bi bile vse izpolnjene. Za primer poglejmo podatkovno bazo spletne trgovine. Če bi se ta držala ACID principa in bi kupec opravljal nakup, bi transakcija nakupa, še preden bi bila zaključena, blokirala del podatkovne baze (npr. zalogo) in vsi drugi potencialni kupci bi morali čakati, dokler se ta transakcija ne zaključi. Zato spletna trgovina uporablja predpomnjenje ali celo odklenjene zapise, kar pa sicer lahko vodi v nekonsistentnost in bi se v skrajnem primeru lahko odražalo v tem, da bi dva kupca istočasno opravljala nakup zadnjega izdelka v trgovini, dobil ga bi pa seveda samo en izmed njiju (Vaish, 2013). Model ACID v nekaterih situacijah torej ovira delovanje podatkovne baze, zato se pri NoSQL bazah (kjer imamo omogočeno horizontalno skaliranje čez več strežnikov in je možnost nekonsistence precej verjetna) uporablja model BASE. Ta je sestavljen iz treh principov:

- **Večinoma razpoložljiv** (ang. **B**asically **A**vailable): NoSQL podatkovne baze se osredotočajo na razpoložljivost podatkov, tudi v pristnosti napak. Sistem se obnaša, kot da vedno deluje. Namesto da bi vzdrževali eno veliko shrambo podatkov in skrbeli, da ne pride do napak, raje uporabimo razdeljene podatke čez več shramb (z replikacijo ali delitvijo) in če pride do napake, to ne vodi v popoln izpad delovanja podatkovne baze.
- **Mehko stanje** (ang. **S**oft state): Stanje sistema se zaradi asinhronih posodobitev čez čas spremeni tudi brez sprememb s strani uporabnika. Posledica tega je, da so kopije podatkov v nekem času lahko nekonsistentne.
- **Sčasoma dosleden** (ang. **E**ventually consistent): Edina zahteva NoSQL podatkovnih baz o doslednosti je ta, da bodo podatki v poljubnem času v prihodnosti dosledni, vendar ni jamstva za to, kdaj bo tako stanje zagotovljeno. To je posledica dejstva, da imajo nekatera vozlišča točne podatke, druga vozlišča še stare podatke in po tem ko se med vozlišči izvedejo vse posodobitve, postanejo kopije podatkov dosledne.

Lastnosti BASE bi lahko povzeli na način, da aplikacija deluje v bistvu ves čas (večinoma razpoložljiva), ni nujno, da je ves čas konsistentna (mehko stanje), vendar sčasoma bo (sčasoma dosledna). Te lastnosti so torej ohlapnejše od ACID lastnosti in največja razlika je ta, da ne zagotavljajo konsistentnosti vozlišč ob pisanju. Kdaj in kako bo konsistentnost zagotovljena, je odvisno tudi od ponudnika sistema, mogoče v prihodnosti ob času branja (npr. Riak), mogoče samo za pretekle preseke podatkov (npr. Datomic) (Robinson, Webber & Eifrem, 2015, str. 196).

V skladu z BASE načelom je Eric Brewer leta 2000 predstavil svoje domneve glede distribuiranih sistemov, ki sta jih par let kasneje Seth Gilbert in Nancy Lynch tudi formalno dokazala in so danes poznane kot CAP teorem. **CAP teorem** oz. tudi **Brewerjev izrek** pravi, da lahko distribuirani sistem ob treh lastnostih konsistence, razpoložljivosti in particijske tolernce zadosti le dvema (Sadalage & Fowler, 2013):

- **Konsistentnost** (ang. **C**onsistency): Konsistentnost pri CAP teoremu nima enakega pomena kot pri modelu ACID, saj se tam podatki, ki ne ustrezajo vnaprej določenim omejitvam, ne ohranijo. Tu je zahteva po konsistentnih kopijah podatkov na različnih strežnikih, (Sullivan, 2015).
- **Razpoložljivost** (ang. **A**vailability): Sistem mora biti na voljo za pisanje oz. branje, ko je to potrebno (pa čeprav morda podatki še ne bodo osveženi ali konsistentni).
- **Particijska toleranca** (ang. **P**artition tolerance): Grozd vozlišč lahko preživi komunikacijske prelome, tudi če ti ločijo grozd na več particij, ki ne morejo več komunicirati med seboj. Tudi ko deli baze niso dostopni, je še vedno mogoče pisati in brati v nekatere dele baze, in ko postanejo prej nedostopna vozlišča spet dostopna, se ustrezno posodobijo (Sadalage & Fowler, 2013). Sistem torej vedno deluje kot celota, četudi je medtem mogoče kakšen strežnik nedostopen ali se na njem pojavi napaka.

Teorem CAP navaja, da je za vsako distribucijo podatkov nemogoče zagotoviti vse tri od zgoraj naštetih lastnosti. Izbiramo lahko samo med dvema lastnostma, pri tretji je potrebno sprejeti kompromis, možne so kombinacije CP, AP ali CA:

- konsistentni in razpoložljivi sistemi (CA) imajo težavo z deljenjem podatkov, ki jo običajno obvladujejo z replikacijo,
- konsistentni in porazdeljeni sistemi (CP) imajo težavo z razpoložljivostjo podatkov ob naporu za zagotavljanje njihove konsistentnosti,
- razpoložljivi in porazdeljeni sistemi (AP) dosegajo doslednost sčasoma, z replikacijo podatkov in občasnim preverjanjem konsistentnosti v vozliščih.

Od vrst podatkovnih baz in ponudnikov je odvisno, kateri dve lastnosti bodo zagotavljali.

Na tem mestu je primerno omeniti še dejstvo, da za NoSQL baze ne velja nujno, da ne morejo zagotavljati ACID lastnosti. Zagotavljata jih npr. bazi CouchDB in Neo4j (Vaish, 2013).

1.7 Kategorije NoSQL podatkovnih baz

Razvitih je bilo že kar nekaj NoSQL sistemov, v splošnem pa se delijo v štiri skupine:

- ključ-vrednost shrambe,
- dokumentne shrambe,
- shrambe družin stolpcev,
- grafne podatkovne baze.

1.7.1 Dokumentne shrambe

Dokumentne podatkovne baze so v osnovi shrambe tipa ključ-vrednost, ki za shranjevanje podatkov uporabljajo bolj kompleksne podatkovne strukture – dokumente. Te predstavljajo

nabor tipov ključ-vrednost, strukturiranih v obliko dokumenta. Pogosto so zastopani v obliki XML in JSON datotek, saj so te visoko prenosljive, kompaktne in standardizirane. Od ključ-vrednost shramb se razlikujejo predvsem v načinu obdelave podatkov. Podatki v ključ-vrednost shrambah se namreč s strani baze štejejo za zelo nepregledne, v dokumentnih shrambah pa je to rešeno z njihovo vgrajeno strukturo dokumentov. Na najosnovnejšem nivoju so lahko dokumenti shranjeni oz. vrnjeni po ID-ju (identiteta dokumenta), v splošnem se za pridobivanje nekega nabora dokumentov iz shrambe uporabljajo indeksi (Klančar, 2013, str. 7). Podobno kot v relacijskih podatkovnih bazah nam indeksi omogočajo večjo učinkovitost branja v zameno za slabšo zmogljivost pisanja. Pri branju pa moramo za razliko od relacijskih podatkovnih baz preiskati veliko manjšo količino podatkov, da pridemo do želenih podatkov. Pisanje je zahtevnejše, ker mora podatkovna baza vzdrževati tudi indekse. Kjer se indeksov ne uporablja, pa so poizvedbe običajno počasnejše, saj se mora preiskati celoten nabor podatkov. Temu pa se običajno želimo izogniti.

Večina dokumentnih shramb zahteva načrtovanje delitve podatkov (ang. sharding), da dosežejo horizontalno skaliranje. Za razliko, ključ-vrednost shrambe in shrambe družin stolpcev tega ne zahtevajo, saj alocirajo podatke na replike kot običajen del njihovega delovanja.

Dokumentne shrambe se močno razlikujejo od tradicionalnih relacijskih podatkovnih baz. Relacijske podatkovne baze običajno shranjujejo podatke v ločene tabele (definirane s strani podatkovnega arhitekta ali programerja) in posamezen objekt se lahko razporedi čez več tabel. Dokumentne shrambe pa shranjujejo vse informacije za dani objekt na enem samem mestu v bazi, kar odpravlja potrebo po objektno-relacijskem preslikovanju (ang. object-relational mapping) med nalaganjem podatkov v shrambo. Tako ima lahko vsak dokument svojo strukturo, odvisno od uporabniških potreb.

Pri pisanju dokumentne shrambe zagotavljajo, da je transakcija omejena na raven posameznega zapisa. To pomeni, da je za zapise v posamezen dokument dosežena atomarnost (ang. atomicity).

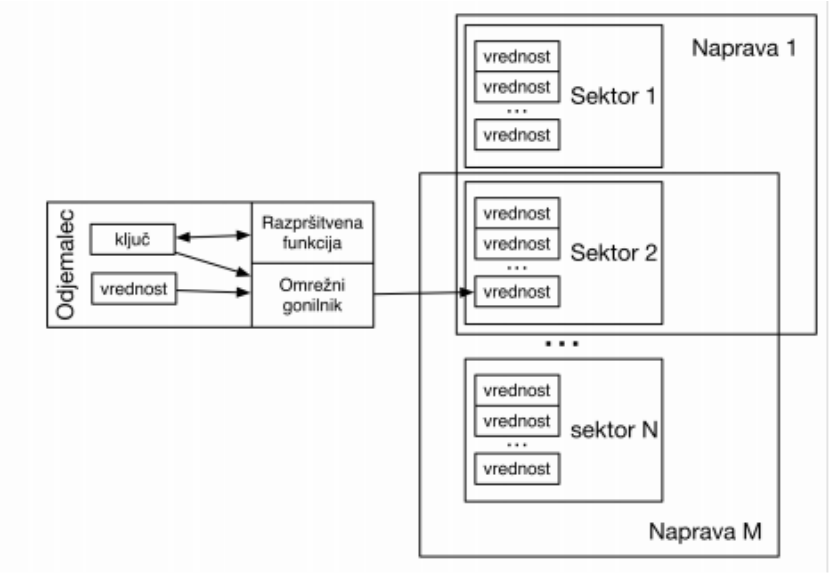
Med bolj razpoznavne dokumentne shrambe sodijo OrientDB, MongoDB, Cloudant, Apache CouchDB, BaseX, MarkLogic, Couchbase, Amazon DynamoDB.

1.7.2 Ključ-vrednost shrambe

Ključ-vrednost shrambe delujejo kot velike, porazdeljene razpršene strukture, ki pridobivajo in shranjujejo podatke na osnovi ključa. Podatki so s pomočjo razprševalne funkcije (ang. hashmap) razporejeni čez različne sektorje (ang. bucket) na spletu. Sektorji so replicirani na različne naprave oz. strežnike, kar zagotavlja toleranco na napake. Kolikokrat mora biti sektor repliciran, določa formula $\mathrm{R} = 2F + 1$, kjer je F število napak, ki jih sistem lahko prenese. Algoritem za repliciranje pa zagotavlja, da naprave niso točne kopije druga druge (Robinson, Webber & Eifrem, 2015, str. 199).

Ti preprosti sistemi običajno niso primerni za kompleksne aplikacije. Po drugi strani pa je prav ta preprostost v določenih okoliščinah privlačna.

Slika 6: Ključ-vrednost shramba kot porazdeljena razpršena struktura



Vir: Klančar (2013).

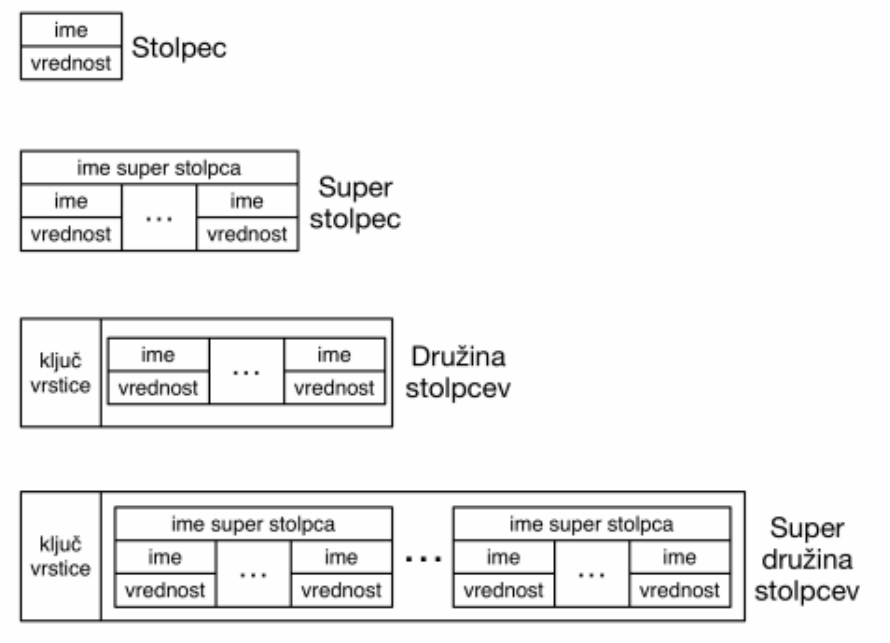
Ključ-vrednost shrambe so podobne dokumentnim shrambam, razlikuje jih le stopnja vpogleda v njune podatke. Ključ-vrednost shrambe ne vedo, kakšne podatke imajo shranjene, vrednost je zapisana kot celota, nič se ne ve o njeni notranji strukturi. Pogosto pa se zgodi, da uporabnik ne potrebuje celotne shranjene vrednosti, potrebuje le delček informacije, ki je shranjena v vrednosti. Potrebne podatke se v takih primerih navadno izlušči s pomočjo zunanjega programa, velikokrat se uporablja MapReduce (Robinson, Webber & Eifrem, 2015, str. 201).

Glavni predstavniki ključ-vrednost shramb so Redis, Riak KV, Oracle NoSQL, Aerospike, Oracle Berkeley DB, Amazon DynamoDB.

1.7.3 Shrambe družin stolpcev

Shrambe družin stolpcev (ang. column family), shrambe s širokimi stolpci (ang. wide column datastores) ali shrambe razširljivega zapisa (ang. extensible record stores) so izdelane po Googlovem BigTable porazdeljenem sistemu za shranjevanje podatkov. Podatkovni model temelji na redko poseljenih tabelah, katerih vrstice vsebujejo stolpce (Robinson, Webber & Eifrem, 2015, str. 202).

Slika 7: Štiri primeri grajenja blokov v shrambah družin stolpcev



Vir: Klančar (2013).

Na sliki 7 vidimo štiri primere grajenja blokov v shrambah družin stolpcev. Najenostavnejša enota shrambe je stolpec (ang. column), ki je sestavljen iz para ime-vrednost. Poljubno število stolpcev lahko združimo v super stolpec (ang. super column), ki da ime naboru stolpcev. Stolpci so shranjeni v vrstice in ko je vrstica sestavljena iz samih stolpcev, se imenuje družina stolpcev (ang. column family). Ko pa vrstica vsebuje super stolpce, se imenuje super družina stolpcev (ang. super column family).

Osnovni koncept delovanja shramb družin stolpcev je delitev tako vrstic kot stolpcev preko vozlišč (vertikalno in horizontalno particioniranje):

- Vrstice so razdeljene po vozliščih na podlagi delitve primarnega ključa (ali t. i. ključa vrstice). Delitev je izvedena glede na določeno zalogo vrednosti ključev, ne z neko razprševalno funkcijo. Posledično v primeru poizvedovanja po obsegu vrednosti ni potrebno obiskati vsakega vozlišča.
- Stolpci tabele so porazdeljeni čez več vozlišč na podlagi koncepta družin stolpcev. Družine stolpcev so način definiranja podatkov, ki sodijo skupaj.

Super stolpci, družine oz. super družine stolpcev niso nujno identično organizirani po vrsticah in posledično je baza učinkovita, tudi ko ima zelo redke podatke (nobena od teh lastnosti ne velja za npr. relacijske podatkovne baze).

Glavni predstavniki shramb družin stolpcev so Apache Cassandra, Apache HBase, Microsoft Azure Table Storage, Hypertable, Amazon DynamoDB.

1.7.4 Grafne podatkovne baze

Grafne podatkovne baze so namenjene shranjevanju podatkov, predstavljenih v obliki grafov (elementi so med seboj povezani z nedoločenim številom relacij med njimi). Vozlišča in povezave imajo lahko atribute in lastnosti. Podrobno bomo grafne podatkovne baze predelali čez celotno delo.

1.8 Primerjava SUPB

Množična uporaba relacijskih podatkovnih baz je velikokrat pomemben dejavnik pri izbiri glavnega sistema v mnogih podjetjih. Vendar trenutni trendi kažejo, da moramo pri odločanju upoštevati tudi vse ostale kategorije NoSQL podatkovnih baz. Poglejmo torej nekaj prednosti in slabosti v primerih uporabe vsake kategorije.

Relacijske podatkovne baze:

- Prednosti: so transakcijsko usmerjene podatkovne baze, primerne za aplikacije, kjer je potrebno podpreti sočasnost številnih transakcij in zaščititi celovitost podatkov (bančne aplikacije, spletne rezervacije).
- Slabosti: neprimerne za analitično usmerjeno shranjevanje z veliko količino podatkov in redkimi posodobitvami (relacijske podatkovne baze slabo podpirajo skalabilnost), neprimerne, če jih želimo ponovno uporabljati za nove aplikacije (Sullivan, 2015).

Ključ-vrednost shrambe:

- Prednosti: za predpomnjenje iz relacijskih podatkovnih baz za izboljšanje hitrosti, shranjevanje podatkov o konfiguraciji in uporabniških podatkih za mobilne aplikacije, sledenje prehodnim atributom v spletni aplikaciji kot je npr. spletni nakupovalni voziček (tj. za aplikacije z manjšimi količinami podatkov za pisanje in branje na preprostem podatkovnem modelu) (Sullivan, 2015, str. 429). Velika prednost je horizontalno skaliranje, ki je lahko zelo učinkovito in ga je preprosto izvesti. Ključem vzamemo samo del njihove vrednosti in jih razvrstimo po ostalih strežnikih (Malej, 2016, str. 6).
- Slabosti: iskanje točno določenih shranjenih vrednosti, težavno definiranje povezav, neprimerne so za shranjevanje podatkov, ki se pogosto posodabljajo.

Dokumentne shrambe:

- Prednosti: zasnovane za fleksibilnost, za aplikacije, ki zahtevajo zmožnost shranjevanja različnih atributov skupaj z velikimi količinami podatkov (elektronsko poslovanje, podpora spletnim stranem z veliko količino branja in pisanja), s hitrejšimi poizvedbami kot v ključ-vrednost shrambah. V primerjavi z relacijskimi podatkovnimi bazami imajo boljše zmogljivosti pisanja in branja, zaradi uporabe fleksibilne sheme lahko hranijo različne tipe dokumentov, dobro podpirajo skalabilnost.

– Slabosti: neprimerne so za izvajanje kompleksnih poizvedovanj z veliko operacijami.

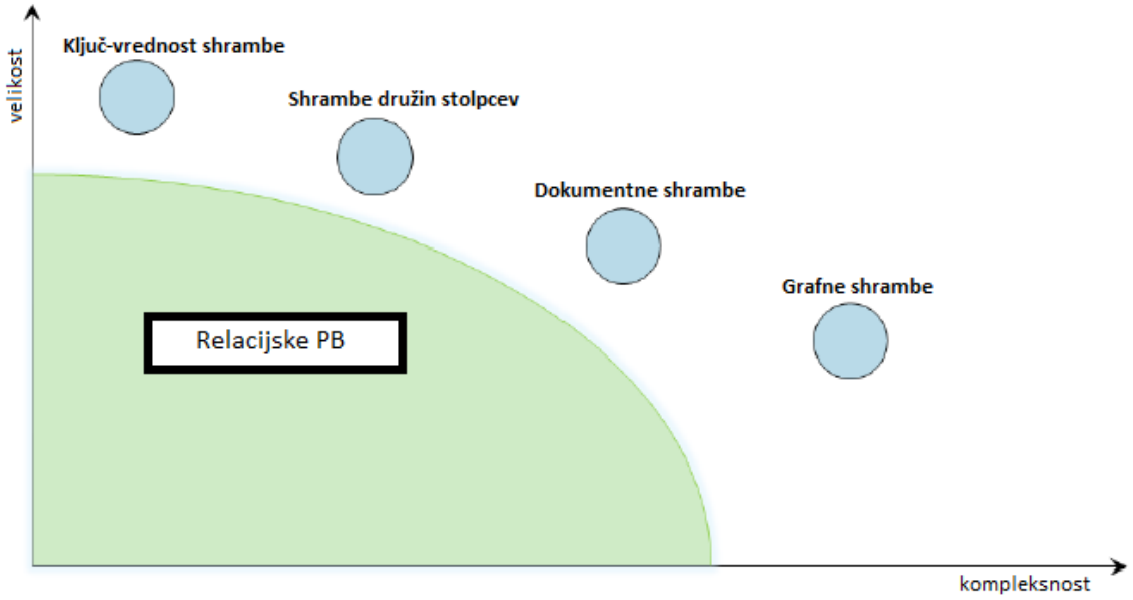
Shrambe družin stolpcev:

– Prednosti: zasnovane so za zelo velike količine podatkov, imajo dobro zmogljivost pisanja in branja ter visoko razpoložljivost. Primerne za aplikacije, ki pišejo v baze, razporejene na več podatkovnih centrih, ki vsebujejo dinamična polja, ob dopuščanju krajših obdobjih nekonsistentnosti (varnostni sistemi z uporabo omrežnega prometa, analize borznega trga, spletne aplikacije za iskanje) (Sullivan, 2015, str. 432).
– Slabosti: čas poizvedovanja je odvisen od poizvedbe, v nekaterih primerih se lahko občutno izboljša, v drugih primerih je lahko še počasnejši kot ob uporabi relacijske podatkovne baze. Te shrambe so neprimerne za kompleksno poizvedovanje in če se vzorci poizvedb skozi čas velikokrat spremenijo.

Grafne podatkovne:

– Prednosti: za modele z eksplicitnimi razmerji, ki so lahko predstavljeni kot graf in kjer je potreba po hitrem pregledovanju (razpoznava goljufij, poizvedovanje, omrežja in informacijska tehnologija, socialna omrežja).
– Slabosti: neprimerne so za velika podatkovna skladišča, ki potrebujejo model BASE.

Slika 8: Primernost kategorij glede na velikost in kompleksnost podatkov



Vir: Prirejeno po Domenjoud & Vial (2012).

Na sliki 8 je prikazanih pet kategorij podatkovnih baz, ki so razvrščene po primernosti za uporabo. Razvidno je, da ni vsaka primerna za katerikoli namen. To je v veliki meri odvisno od količine in kompleksnosti podatkov, ki jih bomo v podatkovno bazo shranjevali. Njihovi primeri uporabe se ne prekrivajo, kar utemeljuje, da mora podjetje pred izbiro SUPB dobro

preučiti potrebe in zahteve. Tako so ključ-vrednost shrambe najprimernejše za shranjevanje res velikih količin podatkov, ki pa ne smejo biti preveč kompleksni, grafne podatkovne baze pa so najprimernejše za zelo kompleksne podatke, a v ne tako veliki količini. Ne smemo pa se popolnoma zanašati na zgornji prikaz primernosti, saj lahko tudi kombiniramo uporabo različnih podatkovnih baz. Na primer za obdelovanje nekega velikega socialnega omrežja bi hoteli uporabiti grafno podatkovno bazo, ampak za shranjevanje in nalaganje podatkov bi uporabili shrambo družin stolpcev, ki je bolj primerna za res velike količine podatkov. Na tem sistemu bi nato delali z operacijami grafnih podatkovnih baz (tak način omogoča grafna podatkovna baza Titan) (Sullivan, 2015, str. 432).

Za razliko od relacijskih podatkovnih baz, ki so v bistvu nadomestile vse svoje predhodnike, bodo NoSQL podatkovne baze obstajale in delovale sočasno z relacijskimi, saj se potreba po različnih vrstah aplikacij z različnimi zahtevami veča. Uporaba NoSQL in relacijskih podatkovnih baz skupaj se dopolnjuje, saj relacijske podatkovne baze nudijo številne funkcije, ki zagotavljajo celovitost podatkov in zmanjšujejo tveganje za nepravilnosti. V drugih primerih, kjer je učinkovitost delovanja pomembnejša od konsistentnosti ali podpore ACID transakcij, bi bile NoSQL primernejša rešitev. Relacijske podatkovne baze bodo še naprej podpirale sisteme za obdelavo transakcij in aplikacij poslovne inteligence. Desetletja dela s sistemi za obdelavo transakcij in podatkovnimi skladišči so namreč privedle do zelo dobrih rešitev, ki še vedno ustrezajo potrebam podjetij, vlad in drugih organizacij, a hkrati se te organizacije prilagajajo tehnologijam, ki prej niso obstajale. Spletne aplikacije, mobilne storitve in Big Data bi sicer v nekaterih primerih lahko delovale na relacijskih podatkovnih bazah, v drugih primerih pa ne. Trenutno okolje zahteva različne tehnologije baz podatkov in izbiranje podatkovne baze je proces izbire pravega orodja za opravljanje dela. Bolj kot je raznolik nabor vseh zastavljenih del oziroma projektov, širši bo nabor orodij. In tako kot ni najprimernejšega programskega jezika, ni najprimernejšega sistema za upravljanje podatkovnih baz.

1.9 Agregirane shrambe

Ogledali smo si nekaj osnovnih lastnosti kategorij NoSQL baz in na tem mestu je primerno poudariti dejstvo, da je med ključ-vrednost shrambo, dokumentno shrambo in shrambo družin stolpcev več podobnosti kot razlik. Podobnosti med temi tremi shrambami so tako velike, da jih pogosto skupaj uvrščamo v skupino agregiranih shramb.

V relacijskih podatkovnih bazah so podatki shranjeni v vrsticah, ki zajemajo nek nabor vrednosti, ne omogočajo pa gnezdenja. Rezultati operacij so vedno vrstice. Pri agregiranih shrambah pa je pristop drugačen, saj imajo kompleksnejšo strukturo, ki običajno omogočajo gnezdenje struktur. Izraz agregacija izhaja iz domensko vodenega oblikovanja (ang. domain-driven design), kjer je agregat zbirka sorodnih podatkov, ki jih želimo obravnavati kot enoto (Klančar, 2013, str. 7). Čeprav ima vsaka agregirana shramba svojo strategijo shranjevanja (za ključ-vrednost shrambe so to vrednosti, za dokumentne shrambe so dokumenti in za

shrambe družin stolpcev so to družine stolpcev), imajo veliko skupnega pri poizvedbah. Shranjevanje agregatov kot temeljne enote ima veliko smisla pri shranjevanju podatkov v grozdih strežnikov. Agregati tvorijo naravne enote za distribucijske strategije, kot je na primer delitev (ang. sharding).

Celoten pristop agregiranih shramb deluje zelo dobro, ko je dostop do podatkov usklajen z agregati, a obstaja pomembna slabost. Problem je, če želimo podatke pogledati na različne načine za različne ciljne skupine. Za te kompleksnejše poizvedbe gredo običajno agregati skozi zunanje orodje (npr. MapReduce), ki poskrbi, da pridobimo podatke, ki jih želimo.

MapReduce je tako kot BigTable še ena tehnologija, ki prihaja s strani Googla. Prevladujoča odprtokodna implementacija MapReduce je Apache Hadoop. MapReduce je programski model, ki se izvaja vzporedno in deli podatke, na katerih nato izvaja operacije paralelno, preden jih spet zbere in združuje (Robinson, Webber & Eifrem, 2015, str. 205).

Agregirane shrambe so namenjene velikim, ne pretirano povezanim podatkom, sploh pri izvajanju v grozdih. Za zelo povezane podatke so bolj primerne grafne podatkovne baze, sploh če želimo manipulirati s podatki na različne načine in ne samo na način, usklajen s strukturo agregatov.

2 GRAFNE PODATKOVNE BAZE

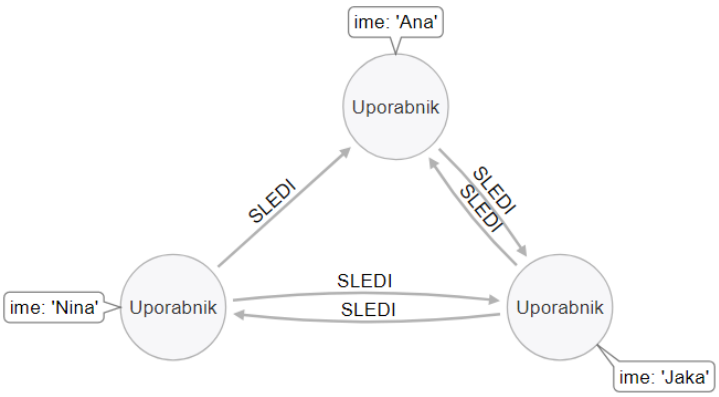
Čeprav bomo skozi delo opazovali modele grafov, tu ne obravnavamo teorije grafov. Grafi v grafnih podatkovnih bazah sicer temeljijo na teoriji grafov z vsemi podpornimi ugotovitvami, ampak za razumevanje podatkovnih baz te vrste ne potrebujemo tega predznanja. Teorija grafov je matematično in računalniško področje, ki raziskuje značilnosti grafov in ima začetke v letu 1736, ko je Leonard Euler rešil problem sedmih mostov mesta Königsberg, ki se je nanašal na vprašanje, ali se je mogoče sprehoditi po mestu tako, da se preči vsakega od mostov natanko enkrat in se vrne na začetno točko. Euler je problem predstavil z grafom, kjer so deli mesta označeni z vozlišči, mostovi pa s povezavami. S tem, ko je prikazal, da nobeno vozlišče nima sodega števila povezav, je dokazal, da takšna pot ne obstaja. Pot, kjer prehodimo vsako povezavo natanko enkrat in se vrnemo na izhodišče, danes imenujemo Eulerjev obhod. Možen je le v grafu, v katerem so vsa vozlišča sode stopnje.

Slika 9: Problem sedmih mostov Königsberga

Vir: Kemper (2015).

Za razumevanje grafnih podatkovnih baz je dovolj definicija grafa kot množice točk oz. vozlišč in povezav med njimi. Za občutek si oglejmo, kako bi izgledal fragment Twitter podatkov, upodobljen v grafni podatkovni bazi. Vsako vozlišče ima oznako »Uporabnik«, ki označuje njegovo vlogo v omrežju, med njimi pa so povezave, ki razložijo njihova razmerja. Na primer Ana sledi Jaku in Jaka sledi Ani, Nina in Jaka si tudi sledita, medtem ko Nina sledi Ani, Ana pa Nini ne.

Slika 10: Primer grafne podatkovne baze Twitterja



Vir: lastno delo.

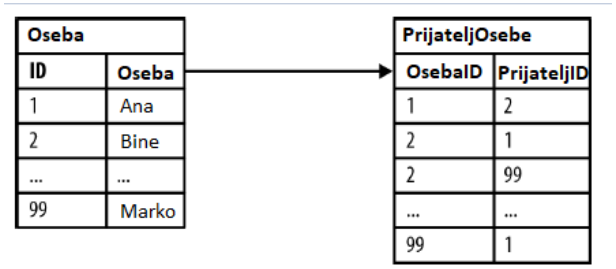
Prava grafna podatkovna baza Twitterja je približno stomilijonkrat večja, a je zgrajena na prav ta način (Robinson, Webber & Eifrem, 2015, str. 2). Tako kot smo ustvarili vozlišča z oznako »Uporabnik«, bi lahko dodali še vozlišča z oznako »Objava« in jih ustrezno povezali z uporabniki po kronološki razvrstitvi.

2.1 Možnosti shranjevanja zelo povezanih podatkov

Živimo v zelo povezanem svetu in tehnologije se morajo soočati s tem dejstvom. Kot že omenjeno, razvijalci že več desetletij poskušajo prilagoditi povezane, pol strukturirane podatkovne zbirke v relacijske podatkovne baze. Ker pa so v njih podatki shranjeni v

tabelarnih strukturah, niso najprimernejši za hranjenje povezav in odnosov med podatki, saj lahko te definiramo samo z združevanjem tabel. Pogosto bi radi tudi ločevali povezave po vsebini, pa tudi po njihovi teži ali moči, kar v relacijskih podatkovnih bazah ni lahko izvedljivo oziroma sploh ni. Slabost relacijskih podatkovnih baz za hranjenje velikega števila povezav je tudi nepreglednost, zapletenost in manjša enotnost podatkovnega modela, ki sledi dodajanju povezav. To se izraža v velikem številu združenih tabel, kar ovira delovanje in otežuje razvoj obstoječe podatkovne baze v odgovor na spreminjajoče podatkovne potrebe. Da bo težava povezovanja podatkov v relacijskih podatkovni bazi lažje razumljiva, si poglejmo primer iz poljubnega socialnega omrežja.

Slika 11: Primer povezovanja v relacijski podatkovni bazi



Vir: Prirejeno po Robinson, Webber & Eifrem (2015).

Če se vprašamo, kdo so Binetovi prijatelji, je poizvedba preprosta in nezahtevna:

SELECT o1.Oseba FROM Oseba o1
JOIN PrijateljOsebe ON PrijateljOsebe.PrijateljID = o1.ID
JOIN Oseba o2 ON PrijateljOsebe.OsebaID = o2.ID
WHERE o2.Oseba = »Bine«

Pri našem primeru je vrnjen rezultat Ana in Marko. Ta poizvedba ni zahtevna, saj omejuje število obravnavanih vrstic s filtrom WHERE o2.Oseba = »Bine«. Prijateljstvo pa ni vedno obojestransko, zato si oglejmo še poizvedbo, ki vrne osebe, ki imajo Bineta za prijatelja.

SELECT o1.Oseba FROM Oseba o1
JOIN PrijateljOsebe ON PrijateljOsebe.OsebaID = o1.ID
JOIN Oseba o2 ON PrijateljOsebe.PrijateljID = o2.ID
WHERE o2.Oseba = »Bine«

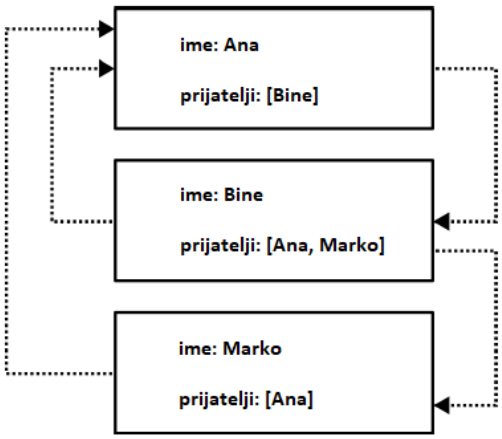
Odgovor je Ana, Marko pa Bineta nima za prijatelja. Ta poizvedba je še vedno enostavna, je pa na strani podatkovne baze zahtevnejša, saj mora upoštevati vse vrstice iz tabele PrijateljOsebe. Še zahtevnejšo poizvedbo bi ustvarili, če bi se vprašali, kdo so prijatelji mojih prijateljev. Hierarhije v SQL uporabljajo rekurzivna združevanja, zaradi katerih je poizvedba računsko bolj zahtevna (obstajajo sicer funkcije, ki poenostavijo poizvedbo, a ne vplivajo na računsko zahtevnost, na primer Oracle ima v ta namen funkcijo CONNECT BY (Robinson, Webber & Eifrem, 2015, str. 14)).

SELECT o1. Oseba AS OSEBA, o2.Oseba AS PRIJATELJ_PRIJATELJA FROM PrijateljOsebe po1
JOIN Oseba o1 ON po1.OsebaID = o1.ID
JOIN PrijateljOsebe po2 ON po2.OsebaID = po1.PrijateljID
JOIN Oseba o2 ON po2.PrijateljID = o2.ID
WHERE o1.Oseba = »Ana« AND po2.PrijateljID != o1.ID

Ta poizvedba je računsko zahtevna, čeprav se ukvarja samo z Aninimi prijatelji prijateljev in ne gre globje v njeno socialno omrežje. Stvari postajajo še bolj zapletene in računsko dražje, ko gremo globje v omrežje. Odgovor na to poizvedbo bi zelo verjetno še dobili v razumnem času, se pa potem na četrtem, petem, šestem nivoju omrežja znatno poslabša odzivni čas poizvedb zaradi rekurzivnega združevanja tabel.

NoSQL shrambe, dokumentne, ključ-vrednost ali shrambe družine stolpcev hranijo množice nepovezanih dokumentov/vrednosti/stolpcev, kar pomeni, da tudi vanje težko shranjujemo zelo povezane podatke. Strategija za dodajanje razmerji med podatki v agregirane shrambe je vgraditev identifikatorja agregata znotraj področja, ki pripada drugemu agregatu, tako kot delujejo tuji ključi v relacijskih podatkovnih baz. Toda to zahteva združevanje identifikatorjev na ravni aplikacije, ki lahko hitro postane računsko prezahtevno, saj mora aplikacija sama graditi povezave med dokumenti/vrednostmi/stolpci, med drugim pa zagotavljati, da posodablja in briše te identifikatorje agregatov skupaj s preostalimi podatki. Drugi problem je ta, da ni nobenih identifikatorjev, ki bi kazali nazaj na manj podrobne nivoje. To pomeni, da je operacija zahtevna, če želimo na primer za zapis iz najpodrobnejšega nivoja izvedeti, v katero skupino na prvem nivoju spada. Za take primere bi morali podatke izvoziti in obdelati prek zunanje računske infrastrukture, kot je npr. Hadoop (Robinson, Webber & Eifrem, 2015, str. 16), ali pa vgraditi reference, ki bi kazale nazaj, v drugo smer. Poglejmo si na primeru, kako bi lahko povezovali dokumente v agregiranih shrambah.

Slika 12: Primer povezovanja v agregirani shrambi



Vir: lastno delo.

S tako strukturo je lahko pridobiti neposredne prijatelje določene osebe, problem bi pa bilo najti vse pare prijateljev. Lahko bi pregledovali celotno podatkovno shrambo, kar je časovno zahtevno, ali pa denormalizirali model z dodajanjem povezav v drugo smer, s čimer bi vsaki osebi dodali podatek, kdo ga ima za prijatelja. Ampak tudi to poveča zahtevnost, večja je zakasnitev pisanja, povečani so stroški uporabe diska za shranjevanje metapodatkov in pregledovanje povezav je še vedno drago, ker vsak korak zahteva iskanje indeksa. Tudi ta struktura omejuje možnosti za razširitev socialnega omrežja.

Agregirane shrambe ne ohranjajo konsistentnosti povezanih podatkov niti ne podpirajo t. i. brezindeksne sosednosti (ang. index-free adjacency), ki pomeni, da elementi kažejo neposredno na svoje sosede. Mnogo sistemov poskuša ohraniti videz grafne obdelave podatkov, a ne zagotavljajo interakcij v realnem času.

Kar si želimo, je povezana celota, vključno s povezavami in elementi. V nasprotju z relacijskimi podatkovnimi bazami in agregiranimi shrambami so v grafnih podatkovnih bazah povezani podatki shranjeni kot povezani podatki. Kjer so povezave med podatki, tam so tudi povezave v podatkovnem modelu. Te v grafu tvorijo poti, po katerih se s poizvedbami samo sprehodimo. V (Batra & Tyagi, 2012), (Vukotic, Watt, Abedrabbo, Fox & Partner, 2015) in (Malej, 2016) je bilo na primerih prikazano, da grafna podatkovna baza lahko v poizvedovanju deluje neprimerno hitreje kot relacijska podatkovna baza.

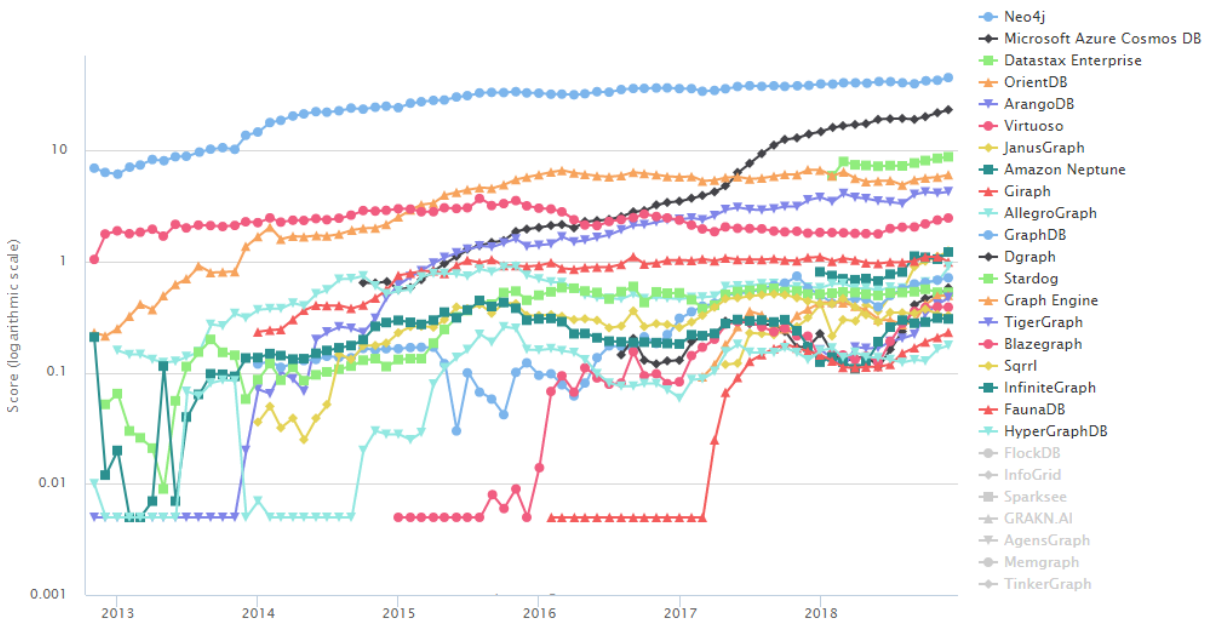
V nadaljevanju dela se bomo omejili na obravnavo grafne podatkovne baze Neo4j , ki je prva in v skladu z ocenami iz DB-engines (2019) še vedno vodilna ter najbolj razširjena med grafnimi podatkovnimi bazami. Podjetje Neo4j, Inc. (prej Neo Techology) ponuja dve različici podatkovne baze, odprtokodno različico *Community* ter zaprtokodno verzijo, imenovano *Enterprise*, ki je namenjena večjim komercialnim podjetjem. Omejili se bomo na odprtokodno različico. Neo4j je napisana v programih Java in Scala, s preprostim uporabniškim vmesnikom, ki se ga je enostavno naučiti. Zanjo je na voljo veliko dokumentacije, primerov uporabe različnih podjetij in velika ter aktivna skupnost uporabnikov. Poleg tega gre za visoko zmogljivo podatkovno bazo, za katero veljajo naslednje lastnosti:

- visoka zmogljivost branja in pisanja,
- skalabilnost,
- ACID transakcije,
- visoka razpoložljivost,
- preprosto učenje in uporaba,
- hitre poizvedbe,
- deklarativen grafni poizvedovalni jezik Cypher.

Na spodnjem grafikonu na sliki 13 je razvidno, da je Neo4j res prevladujoča grafna podatkovna baza in upoštevati moramo dejstvo, da je rezultat prikazan v logaritemski lestvici, zato je razlika v priljubljenosti z ostalimi grafnimi podatkovnimi bazami res velika.

Opazimo tudi precejšnjo rast v priljubljenosti Microsoft Azure Cosmos DB, ki se je pojavil leta 2014. Argument za to je dobra integriranost Microsoft Azure na trgu programske opreme in dejstvo, da to ni samo grafna podatkovna baza, ampak podpira še dokumentne, ključ-vrednost shrambe in shrambe družin stolpcev. Nekaj ponudnikov grafnih podatkovnih baz je na spodnji sliki prikritih, ampak le zaradi boljše preglednosti. Vsi prikriti predstavniki so bili uvrščeni nižje od Neo4j.

Slika 13: Priljubljenost grafnih podatkovnih baz



Vir: DB-engines (2019).

2.2 Delitev področja grafov

Preden se poglobimo v delovanje in arhitekturo baze Neo4j, si še oglejmo, kako v splošnem delimo področje obdelovanja podatkov o obliki grafov:

1. Tehnologije, namenjene za uporabo v transakcijskih sistemih, kjer iz aplikacije dostopamo do podatkov v realnem času. Te tehnologije so imenovane **grafne podatkovne baze**. Ekvivalentne so običajnemu transakcijskemu procesiranju (ang. online transactional processing) (v nadaljevanju OLTP) v relacijskih podatkovnih bazah. Na te je usmerjena pozornost v magistrskem delu.
2. Tehnologije, ki se jih uporablja predvsem v analitiki in so imenovane **orodja za obdelavo grafov** (ang. graph compute engine). Optimizirane so za procesiranje velikih količin podatkov in so v tem podobne podatkovnem rudarjenju in OLAP aplikacijam v relacijskih podatkovnih bazah. Teh v delu ne bomo obravnavali.

Grafne podatkovne baze so spletni sistemi za upravljanje podatkovnih baz z operacijami za ustvarjanje (ang. **C**reate), branje (ang. **R**ead), posodabljanje (ang. **U**pdate) in brisanje (ang. **D**elete) (v nadaljevanju CRUD). Običajno so grafne podatkovne baze narejene za uporabo s transakcijskimi (ang. online transactional processing) sistemi. V skladu s tem so običajno optimizirane za uspešnost transakcij in zasnovane za celovitost transakcij ter razložljivost operacij. Obstajata dve lastnosti, po katerih delimo grafne podatkovne baze. To sta **osnovno shranjevanje** (ang. underlying storage) in **obdelovanje** (ang. processing engine). Nekatere grafne podatkovne baze uporabljajo avtohtono grafno shranjevanje (ang. native graph storage), ki je optimizirano in prilagojeno za shranjevanje in upravljanje grafov, nekatere pa se tega ne poslužujejo. Nekatere grafne podatkovne baze uporabljajo brezindeksno sosednost (ang. index-free adjacency), kjer povezana vozlišča v bazi kažejo neposredno drug na drugega. Kakorkoli, med grafne podatkovne baze uvrščamo vse podatkovne baze, ki se navzven obnašajo kot grafi. Brezindeksno sosedstvo omogoča hitrejše in boljše obdelovanje, zato se pojem avtohtonega obdelovanja (ang. native graph processing) uporablja za grafne podatkovne baze, kjer imamo možnost brezindeksnega sosedstva (Robinson, Webber & Eifrem, 2015). Avtohtono shranjevanje in obdelovanje si bomo kasneje podrobneje ogledali na primeru grafne podatkovne baze Neo4j v razdelkih 2.6 ter 2.7. Pomembno je poudariti, da avtohtono shranjevanje in avtohtono obdelovanje nista niti dobri niti slabi lastnosti, ampak le še en izmed kompromisov, ki ga moramo sprejeti. Korist avtohtonega shranjevanja je zmogljivost in skalabilnost. Nasprotno, koristi neavtohtonega shranjevanja so v dobro poznanem okolju (npr. MySQL, Oracle), katerega karakteristike so dobro poznane vsem. Avtohtono obdelovanje ima prednost v zmogljivem in hitrem prečkanju grafa, ki ni odvisen od indeksov, saj graf sam po sebi zagotavlja naraven indeks sosednosti. A za poizvedbe, ki prečenja vozlišč ne bi uporabljale ali bi potrebovale obdelavo celotnega grafa, bi bila obdelava časovno zahtevna in kompleksna.

Poleg odločanja o načinu shranjevanja ter obdelovanja podatkov se je potrebno odločiti tudi o podatkovnem modelu. Grafne podatkovne baze glede na model delimo na tri področja (Robinson, Webber & Eifrem, 2015, str. 206):

1. Graf lastnosti (ang. property graph). Je najbolj razširjena oblika podatkovnega modela, o kateri smo že pisali in samo ta model bomo skozi nadaljnjo delo tudi obravnavali. Graf lastnosti je zelo intuitiven, enostaven za razumevanje ter ima naslednje lastnosti:

   – vsebuje vozlišča, razmerja (povezave) in lastnosti,
   – vozlišča imajo lastnosti (vozlišča z lastnostmi si lahko predstavljamo kot dokument v obliki ključ-vrednost),
   – vozlišča so lahko označena z eno ali več oznakami (te opredeljujejo vozlišču vlogo, ki jo ima v grafu),
   – razmerja povezujejo vozlišča in oblikujejo model grafa (so vedno usmerjena, imajo ime ter začetno in končno vozlišče),
   – tudi razmerja imajo lahko lastnosti.

2. Hipergraf (ang. hypergraph). Je posplošen model grafa, pri katerem je lahko eno razmerje (t. i. hiperpovezava) povezano z več vozlišči. Za razliko od grafa lastnosti, pri katerem imamo lahko le eno začetno in končno vozlišče, imamo lahko v hipergrafu več vozlišč na vsaki strani razmerja. Uporabljamo jih predvsem v primerih, kjer je veliko mnogo-proti-mnogo (ang. many-to-many) razmerij. Hipergraf in graf lastnosti sta izomorfna, tj. podatke iz hipergrafa lahko vedno predstavimo z grafom lastnosti (z dodatnimi povezavami in vmesnimi vozlišči). Od načina razmišljanja in vrste aplikacije je odvisno, za katero vrsto modela se odločimo. Večinoma velja, da graf lastnosti predstavlja najboljše ravnovesje med učinkovitostjo in prilagodljivostjo.
3. Trojice (ang. triples). So podatkovna struktura v obliki osebek-povedek-predmet. S tako strukturo zajamemo dejstva, kot so »Marko ima rad sladoled« ali »Bine pleše z Ano«. Posamezno izgledajo trojice precej osiromašeno, ampak skupaj predstavljajo bogato množico podatkov. Shrambe trojic navadno zagotavljajo SPARQL (poizvedovalni jezik za RDF) in so shranjene v RDF formatu (Liu, 2014). Niso avtohtone grafne baze, saj ne podpirajo brezindeksne sosednosti in tudi njihova shramba ni prilagojena za shranjevanje grafov lastnosti. Trojice so shranjene v shrambi kot neodvisni podatki, kar jim dovoljuje horizontalno skaliranje, ne omogoča pa hitrega prečkanja razmerij. Za poizvedbe morajo shrambe trojic narediti povezane strukture iz neodvisnih podatkov, kar pa privede do zakasnitev. Posledično se trojice uporabljajo predvsem v analitiki in ne za obdelovanje transakcij.

Tako kot vsaka kategorija NoSQL podatkovnih baz, je tudi vsak izmed modelov grafnih podatkovnih baz primeren za svoj namen. Hipergrafi so namenjeni za zajemanje meta-podatkov in trojice za semantični splet. Za spletno obdelavo transakcij s hitrimi prečkanji povezav med podatki so primerni grafi lastnosti.

2.3 Prednosti grafnih podatkovnih baz

Podatki so vedno bolj povezani med seboj in večino problemov bi lahko predstavili z grafi. Grafne podatkovne baze predstavljajo močno orodje za modeliranje takih problemov, a so precej nova tehnologija, zato same po sebi še niso zadosten razlog za zamenjavo utečenih podatkovnih baz. Dokazano je bilo že na veliko primerih, da grafne podatkovne baze občutno izboljšajo delovanje in zmanjšajo zakasnitve. In zmogljivost ni edina prednost. Grafne podatkovne baze imajo tudi zelo prilagodljiv podatkovni model (Agrawal & Patel, 2016).

2.3.1 Zmogljivost

Ena glavnih prednosti grafnih podatkovnih baz je hitrost in zmogljivost delovanja na povezanih podatkih v primerjavi z ostalimi NoSQL shrambami in relacijskimi podatkovnimi bazami. V relacijskih podatkovnih bazah pride med združevanjem tabel do pogostih zakasnitev, saj je potrebno računati kartezične produkte indeksov tabel, ki bi jih radi združili.

V grafnih podatkovnih bazah so povezovalne operacije že vnaprej izračunane in eksplicitno prisotne v bazi. Združevanje podatkov je preprosto in primerljivo skoku od enega vozlišča do drugega. Te operacije pogosto imenujemo poizvedbe grafnih vzorcev (ang. pattern matching queries), kar pomeni, da ob izvedbi poizvedbe poiščemo le začetno vozlišče oz. več začetnih vozlišč. Ni potrebno uporabiti celotnih podatkov, ampak le del grafa. Delovanje je torej relativno konstantno, tudi ob povečanju količine podatkov.

2.3.2 Prilagodljivost

Grafi so aditivni, torej lahko kadarkoli dodamo nove povezave, vozlišča, oznake in lastnosti, ne da bi to vplivalo na obstoječo podatkovno strukturo, obstoječe poizvedbe ali delovanje aplikacije. To je za razvijalce in podatkovne arhitekte privlačna lastnost, saj lahko gradijo podatkovno strukturo sproti oz. jo po potrebi spreminjajo, ko so vedno bolj seznanjeni s problemom. Pri relacijskih podatkovnih bazah je najbolje že na samem začetku definirati kar se da popolno strukturo modela, saj kasnejše spremembe zahtevajo kar nekaj dela in časa.

2.3.3 Agilnost

Grafne podatkovne baze so prilagojene za iterativno razvijanje programske opreme in ker je njihov model brez sheme, se enostavno testirajo. Posledično lahko svojo aplikacijo razvijamo v kontroliranem okolju.

2.4 Poizvedovalni jeziki

Do sedaj smo si ogledali, kakšne so prednosti grafnih podatkovnih baz v primerjavi z relacijskimi in ostalimi NoSQL podatkovnimi bazami, nismo pa se še dotaknili vprašanja, kako grafe modeliramo in kako po grafnih podatkovnih bazah poizvedujemo. V ta namen se bomo spoznali z jezikom Cypher.

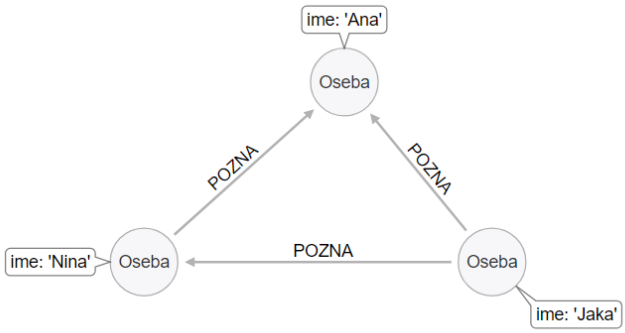
Čeprav obstaja več jezikov za poizvedbo in kreiranje grafov lastnosti, je Cypher tisti, ki prevladuje v kombinaciji z grafno podatkovno bazo Neo4j. Za razliko od relacijskega sveta, kjer je SQL standardni poizvedovalni jezik, ga med grafnimi podatkovnimi bazami še ni. Razlog je v tem, da so grafne podatkovne baze še precej novo področje, kjer z vsako verzijo še vedno prihaja do izboljšanj, poizvedovalni jeziki postajajo hitrejši in bolj odzivni, sintakse se nadgrajujejo in posledično se celotno stanje še ni ustalilo. V zadnjem času prihaja do vse večjih pobud o enotnem poizvedovalnem jeziku za grafe lastnosti, GQL (ang. graph query language). Trenutno so si trije jeziki dovolj podobni, da bi lahko prišlo do združitve: Cypher (Neo4j), PGQL (Oracle) in raziskovalni jezik G-CORE (Linked Data Benchmark Council) (Green, 2019), (Rathle, 2018), (Plantikow, 2018). Različni jeziki za različne produkte namreč nikomur ne koristijo, skupni standardni jezik pa bi omogočil izboljšanje na celotnem trgu grafnih podatkovnih baz, koristi bi imeli tako uporabniki kot prodajalci. Iniciative skupnega poizvedovalnega jezika se osredotočajo na koristi pri podatkovnem modeliranju,

ETL (ang. extract transform load) orodjih za pridobivanje, preoblikovanje in nalaganje podatkov ter orodjih za vizualizacijo podatkov.

2.4.1 Cypher

Cypher se je začel razvijati v letu 2011 v Neo4j, Inc. in je bil prvotno namenjen samo za bazo Neo4j, vendar se je v letu 2015 odprl prek projekta openCypher, kar pomeni, da ga lahko implementira katerakoli grafna podatkovna baza. Cypher je zasnovan po principu ujemanja vzorcev, zato je enostaven za uporabo in razumevanje. Podobno kot SQL je tudi Cypher deklarativen poizvedovalni jezik, kar pomeni, da se ne ukvarja s tem, kako, ampak kaj pridobiti iz grafa (Gupta, 2015), (Merkl Sasaki, Chao & Howard, 2018). V nasprotju z deklarativnimi so imperativni poizvedovalni jeziki, za katere je potrebno točno določiti, kako se do podatkov pride in kaj naj se z njimi stori.

Slika 14: Grafni vzorec



Vir: lastno delo.

Če bi torej želeli v Cyphru z ASCII znaki prikazati poznanstva med tremi osebami, kot prikazuje slika 14, bi zapisali:

(ana)<−[:POZNA]−(jaka)–[:POZNA]−>(nina)–[:POZNA]−>(ana)

Ta vzorec opisuje pot, ki povezuje vozlišča »ana«, »nina« in »jaka«. Te oznake so identifikatorji, ki omogočajo, da se lahko sklicujemo na isto vozlišče več kot samo enkrat, ko opisujmo vzorec. Vzorci v Cyphru zelo naravno sledijo načinu, ki je ekvivalenten risanju grafa na list papirja.

Prejšnji vzorec opisuje zelo enostavno strukturo grafa, vendar se še ne nanaša na nobene posebne podatke v bazi podatkov. Za povezavo vzorca z določenimi vozlišči in razmerji v obstoječih podatkih moramo določiti vrednosti lastnosti in oznake vozlišč, ki nam pomagajo najti ustrezne elemente:

(ana:Oseba {ime: 'Ana'})<−[:POZNA]−(jaka:Oseba {ime: 'Jaka'})−[:POZNA]−>(nina:Oseba {ime: 'Nina'})−[:POZNA]−>(ana)

Povezali smo vsako vozlišče s svojim identifikatorjem z lastnostjo imena in oznako 'Oseba'. Kot vsak poizvedovalni jezik, je tudi Cypher sestavljen iz stavkov. Najpreprostejše poizvedbe so sestavljene iz MATCH in RETURN stavkov.

MATCH (a:Oseba {ime: 'Ana'})−[:POZNA]−>(b)−[:POZNA]−>(c), (a)−[:POZNA]−>(c)
RETURN b, c

Zgoraj je primer Cypher poizvedbe, ki uporabi ta dva stavka, da poišče skupne znance osebe z imenom Ana in vseh njenih znancev.

Kot že omenjeno, morajo podatkovne baze omogočati CRUD (ang. create, read, update, delete) operacije. Cypher jih zagotavlja na sledeče načine:

- Ustvari (ang. create). Za kreiranje podatkov je ključna beseda CREATE. V spodnjem primeru sta ustvarjeni dve vozlišči, povezani z razmerjem 'pozna':
  CREATE (nina:Oseba {ime: 'Nina'})
  CREATE (jaka:Oseba {ime: 'Jaka'})
  CREATE (nina)−[:POZNA]−>(jaka)
  Ekvivalentno bi to lahko zapisali tudi krajše:
  CREATE (nina:Oseba {ime: 'Nina'})−[:POZNA]−>(jaka:Oseba {ime: 'Jaka'})
- Preberi (ang. read). Podatke beremo s pomočjo stavkov MATCH in RETURN, kot že prikazano.
- Posodobi (ang. update). Ko želimo neke podatke posodobiti, jih moramo najprej dobiti. To dosežemo z načinom branja, potem s stavkom SET apliciramo želene spremembe. Če ima na primer vozlišče z oznako 'Oseba' podatek o starosti, ji moramo to lastnost ob rojstnem dnevu posodobiti:
  MATCH (j:Oseba {ime: 'Jaka'})
  SET j.starost = j.starost + 1
  RETURN j
  Istočasno lahko več lastnostim posodabljamo vrednosti.
- Izbriši (ang. delete). Tako kot pri ostalih ukazih, moramo tudi pri brisanju želena vozlišča najprej poiskati. Če želimo npr. izbrisati vozlišče z oznako 'Oseba' in imenom 'Ana', to storimo na sledeč način:
  MATCH (a:Oseba {ime: 'Ana'})
  DELETE a
  Brisanje bo uspelo samo v primeru, da iz in v vozlišče ne vodi nobena povezava. V nasprotnem moramo najprej izbrisati vse take povezave ali pa uporabiti stavek DETACH DELETE, ki briše kaskadno, najprej izbriše vse povezave iz in v vozlišče, nato še vozlišče samo.

Poleg zgoraj navedenih ukazov je poznanih še več Cypher stavkov:

- WHERE zagotavlja kriterij za filtriranje rezultatov.
- MERGE zagotavlja obstoj podanega vzorca, bodisi z uporabo že obstoječih vozlišč in povezav, ki se ujemajo z novo podanimi, ali s kreiranjem novih vozlišč in povezav. V praksi uporabljamo stavek MERGE, ko podvajanje ni dovoljeno. Ko dodajamo v graf in nas podvajanje podatkov ne ovira, uporabljamo stavek CREATE.
- FOREACH izvede dejanje posodabljanja za vsak element iz seznama.
- UNION združi rezultate dveh ali več poizvedb.
- WITH veže naknadne dele poizvedbe in posreduje rezultate od enega do drugega.

2.4.1.1 Indeksi in omejitve

Identifikatorji ostanejo na voljo za čas trajanja trenutnega obsega poizvedbe, ne ostanejo pa shranjeni za sledeče poizvedbe. Če bi to želeli, imamo na voljo kreiranje indeksov za določeno kombinacijo oznake in lastnosti, podobno kot smo tega vajeni iz relacijskih podatkovnih baz. Omogočimo lahko tudi zagotavljanje unikatnosti z omejitvami. Za razliko od relacijskih podatkovnih baz ni v podatkovni bazi Neo4j nobenih dodatnih tujih ključev in kardinalnih omejitev, ki jih moramo vidno vključevati med prave podatke.

Glavni razlog za uporabo indeksov v grafnih podatkovnih bazah je v iskanju začetnega vozlišča oz. vozlišč za prečkanje grafa. Indeks je redundantna kopija nekaterih podatkov iz baze za namen učinkovitejšega iskanja. To pomeni več porabljenega prostora za shranjevanje na disku in počasnejše zapisovanje. Posledično je odločitev o uporabi indeksov pogosto zahtevna. Dodamo jih lahko kadarkoli, zato lahko tudi opazujemo spremembe v hitrosti izvajanja poizvedb na bazi ob uporabi indeksov ali brez. Cypher omogoča ustvarjanje indeksov za eno ali več lastnosti za vozlišča s podano oznako.

- Indeks, ki je ustvarjen na posamezni lastnosti in oznaki, se imenuje indeks posamezne lastnosti (ang. single-property index).

  CREATE INDEX ON :Oseba(ime)
- Indeks, ki je ustvarjen na več lastnostih za dano oznako, se imenuje sestavljeni indeks (ang. composite index).

  CREATE INDEX ON :Oseba(starost, država)

Predpostavimo, da izvedemo poizvedbo CREATE (a:Oseba {ime: 'Ana', starost: '15', država: 'Slovenija'}), (n:Oseba {ime: 'Nina', starost: '30'}). Vozlišče 'a' ima lastnost starosti in države, zato ji bo dodeljen sestavljen indeks iz zgornjega primera. Ker vozlišče 'n' nima lastnosti države, ne bo dodano sestavljenemu indeksu. Ko je indeks ustvarjen, ga bo baza podatkov avtomatsko upravljala in posodobljala, ko bo graf spremenjen. Neo4j bo samodejno vzela indeks in ga začela uporabljati med poizvedbami.

Omejitve se uporabljajo za zagotovitev skladnosti podatkov s pravili domene. Če bi želeli ustvariti omejitev, ki zagotavlja, da baza podatkov ne bo nikoli vsebovala več kot eno vozlišče z oznako 'Oseba' in lastnostjo ime, bi določili naslednje:

CREATE CONSTRAINT ON (oseba: Oseba) ASSERT oseba.ime IS UNIQUE

Kreiranje te omejitve unikatnosti (ang. unique property constraint) bo implicitno dodalo indeks na to lastnost. Če bo omejitev opuščena in bo indeks še potreben, bo moral biti ponovno ustvarjen. Omejitve lahko tako kot indekse dodamo v podatkovno bazo kadarkoli. Zahteva pa se, da obstoječi podatki ustrezajo omejitvam, ki se jih kreira (Jordan, 2014).

V *Enterprise* različici podatkovne baze Neo4j je na voljo še možnost kreiranja omejitev obstoja (ang. property existance constraint) in ključev vozlišč (ang. node keys). Omejitve obstoja zagotavljajo, da neka lastnost obstaja za vsa vozlišča z določeno oznako ali za vsa razmerja z določenim tipom. Vse poizvedbe, ki bi poskušale ustvariti nova vozlišča ali razmerja brez te lastnosti ter poizvedbe, ki bi poskušale odstraniti obvezne lastnosti, bi bile neuspešne. Ključi vozlišč pa zagotavljajo, da za dano oznako in množico lastnosti vse lastnosti obstajajo na vseh vozliščih s to oznako in da je kombinacija vrednosti lastnosti unikatna.

2.4.2 Druge možnosti poizvedovanja

Obstajajo še drugi načini poizvedovanja grafnih podatkovnih baz. Mnoge, vključno z Neo4j, podpirajo poizvedovalni jezik SPARQL, Gremlin in GraphQL. Gremlin je poleg PGQL (Property Graph Query Language, Oracle) najbolj primerljiv s Cypher poizvedovalnim jezikom. Od leta 2009 se razvija med projektom Apache Tinkerpop. Podpira ga več grafnih podatkovnih baz, med njimi tudi Neo4j. Je domensko specifičen jezik, DSL (ang. domain specific language), ki je lahko deklarativen ali imperativen in se ga lahko uporablja za OLTP ali OLAP grafne podatkovne baze. Je zasnovan na Groovyju, vendar ima veliko jezikovnih različic, ki razvijalcem omogočajo pisanje Gremlinovih poizvedb v programskih jezikih, kot so Java, JavaScript, Python, Scala, Clojure in Groovy (Mahmoud, 2017). Gremlin je po naravi vseeno bolj imperativen kot Cypher ali SPARQL in je bolj usmerjen v prečkanja grafov kot k principu ujemanja vzorcev. SPARQL je jezik, zasnovan za poizvedovanje RDF grafov, ki so osnovni format za zapis podatkov na semantičnem spletu. Ni pa SPARQL med tistimi, ki so v boju za GQL oz. »SQL za grafe«. Kot jezik bo vsekakor ostal, vendar se zdi, da se bo največ uporabljal za poizvedovanje iz večjih naborov podatkov s semantičnega spleta (Jorgé, 2015). Omeniti je potrebno še GraphQL, ki ga je ustvaril Facebook in je prilagodljiv protokol API, ki ni specifičen za grafne podatkovne baze. Uporabniki definirajo strukturo podatkov, ki jih želijo in dobijo natanko to, kar so zahtevali in ničesar drugega. Neo4j-GraphQL integracija je na voljo šele od leta 2018 in sicer kot razširitev za Neo4j ali kot JavaScript knjižnica. Zaenkrat še najmanj znana poizvedovalna jezika, ki se potegujeta za standard poizvedovanja grafnih podatkovnih baz, sta GSQL in AQL. GSQL se uporablja

na grafni bazi TigerGraph, AQL (ArangoDB Query Language) pa na bazi ArangoDB. Pri obeh sintaksa temelji na SQL (TigerGraph, 2018).

2.5 Način delovanja podatkovne baze Neo4j

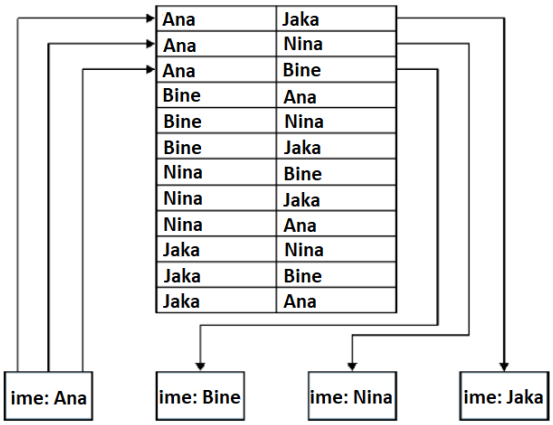
Podatkovna baza Neo4j ima na razpolago dva načina delovanja:

- REST način. Ime kratice REST (ang. Representational State Transfer) predstavlja način komunikacije med klientom in strežnikom. Tu gre za bolj običajen oddaljen dostop do baze, kjer poizvedbe in podatke pošiljamo preko mreže, kot smo tega navajeni iz relacijskega sveta podatkovnih baz. Podatki se prenašajo po načinu REST, relacijski sistemi sicer za upravljanje podatkovnih baz večinoma uporabljajo gonilnik ODBC/JDBC.
- Vgrajeni način (ang. embedded mode). Pri tem načinu baza (strežnik) deluje znotraj aplikacije. Posledično je delovanje veliko hitrejše. Pomanjkljivost je ta, da smo pri tem načinu omejeni samo na uporabe Jave in da lahko do baze dostopamo samo iz ene aplikacije.

2.6 Avtohtono obdelovanje

Omenili smo že pojem avtohtonega obdelovanja (ang. native graph processing), ki velja za grafne podatkovne baze, ki imajo lastnost brezindeksnega sosedstva (ang. index-free adjacency). Pri teh ima vsako vozlišče neposredne reference na svoja sosednja vozlišča. Vsako vozlišče torej deluje kot mikro indeks vseh svojih sosedov, kar je veliko ceneje kot uporaba globalnih indeksov. To pomeni, da časi poizvedb niso odvisni od skupne velikosti grafa, ampak so sorazmerni z velikostjo preiskanega grafa. Nasprotno, neavtohtone grafne podatkovne baze uporabljajo za povezovanje vozlišč (globalne) indekse in posledično je potrebnega več obdelovanja in izvajanje je počasnejše (Lal, 2015).

Slika 15: Neavtohtona obdelava



Vir: lastno delo.

Avtohtona obdelava je učinkovitejša, ker je računska zahtevnost iskanja globalnega indeksa enaka *O(log n)*, zahtevnost brezindeksne sosednosti pa *O(1)*. Če se želimo sprehoditi po *m* vozliščih, zahtevnost iskanja globalnih indeksov znaša *O(m log n)*, pri avtohtoni obdelavi pa *O(m)*. Na sliki 15 je prikazano neavtohtono delovanje. Za iskanje Aninih prijateljev je potrebno izvesti iskanje indeksov z zahtevnostjo *O(log n)*, kar je še sprejemljivo. Hitro pa se računska zahtevnost poveča, ko bi npr. želeli poiskati osebe, ki imajo Ano za prijateljico. Indekse bi morali preiskati za vseh *m* oseb, ki bi lahko imele Ano za prijateljico, torej bi bila zahtevnost *O(m log n)*. Pri brezindeksni sosednosti so obojestranska razmerja shranjena kot povezave med vozlišči. Po grafu se tako lahko sprehajamo naprej in nazaj zelo učinkovito in hitro.

Iskanje indeksov postane računsko prezahtevno predvsem pri velikih količinah podatkov.

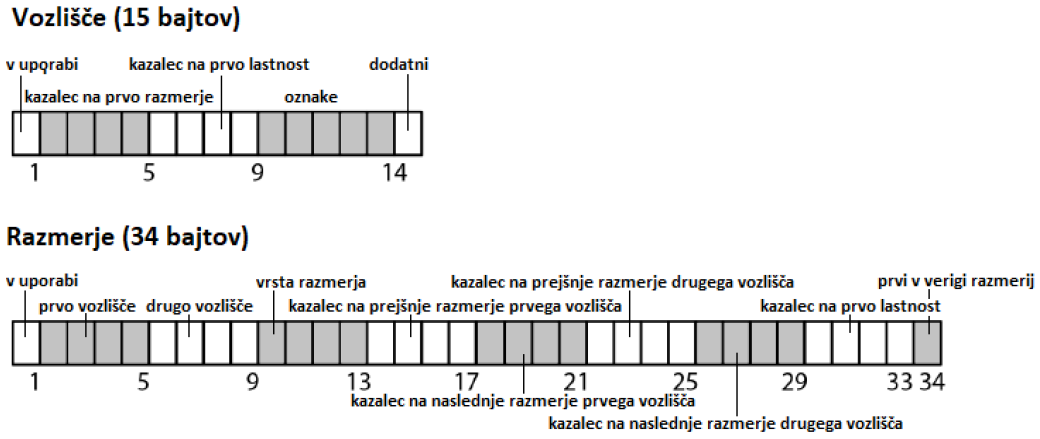
2.7 Avtohtono shranjevanje

Neo4j je grafna podatkovna baza, ki je popolnoma prilagojena tako avtohtonemu obdelovanju kot avtohtonemu shranjevanju. Podatke shranjuje v več različnih shramb, vsaka vsebuje podatke, specifične za določen del grafa (npr. obstajajo ločene shrambe za vozlišča, povezave, oznake in lastnosti). Taka razdelitev shranjevanja omogoča lažje pregledovanje grafa, čeprav je uporabnikov pogled na graf drugačen od dejanskih zapisov na disku. Najprej si oglejmo, kako shramba vozlišč shranjuje vsako novo ustvarjeno vozlišče. Na disku je ta shramba fizično shranjena v datoteki *neostore.nodestore.db*. Tako kot večina shramb pri Neo4j, je tudi shramba vozlišč fiksne velikosti. Vsak zapis je dolžine 15 bajtov. Prav zaradi fiksne dolžine se lahko vozlišča izjemno hitro najde. Če na primer iščemo vozlišče s številko 100, potem vemo, da je shranjeno 1500 bajtov od začetka shrambe. Baza tako z zahtevnostjo *O(1)* najde lokacijo zapisa. Z iskanjem ustreznega indeksa v neavtohtoni shrambi, bi bila ta zahtevnost *O(log n)*.

Strukturo vozlišča in povezave vidimo na sliki 16. Prvi bajt zapisa vozlišča predstavlja oznako tega, ali je vozlišče že v uporabi za shranjevanje določenega vozlišča, ali lahko v zapis shranimo novo vozlišče. Naslednji štirje bajti predstavljajo kazalec na prvo razmerje, povezano z vozliščem, še naslednji štirje bajti pa kazalec na prvo lastnost vozlišča. Sledečih pet bajtov, namenjenih za oznake, kaže na shrambo oznak vozlišč. Zadnji bajt je rezerviran za zastavico, ena izmed možnosti bi bila zastavica za identifikacijo gosto povezanih vozlišč (tj. vozlišč z veliko povezavami). Zapis vozlišča je torej samo peščica kazalcev do seznamov razmerij, oznak in lastnosti. Na podoben način so povezave shranjene v shrambo razmerij, *neostore.relationshipstore.db*. Tudi ta hrani zapise fiksnih dolžin, in sicer po 34 bajtov. Vsak zapis razmerja vsebuje podatek o vozlišču na začetku in koncu povezave, kazalec na vrsto razmerja (ki je shranjen v shrambi vrst razmerij) ter kazalce na prejšnjo in naslednjo povezavo za tako začetno kot končno vozlišče. Tem štirim kazalcem pravimo veriga razmerij

(ang. relationship chain). Zadnji bajt predstavlja zastavico o tem, ali je določen zapis prvi v verigi razmerij.

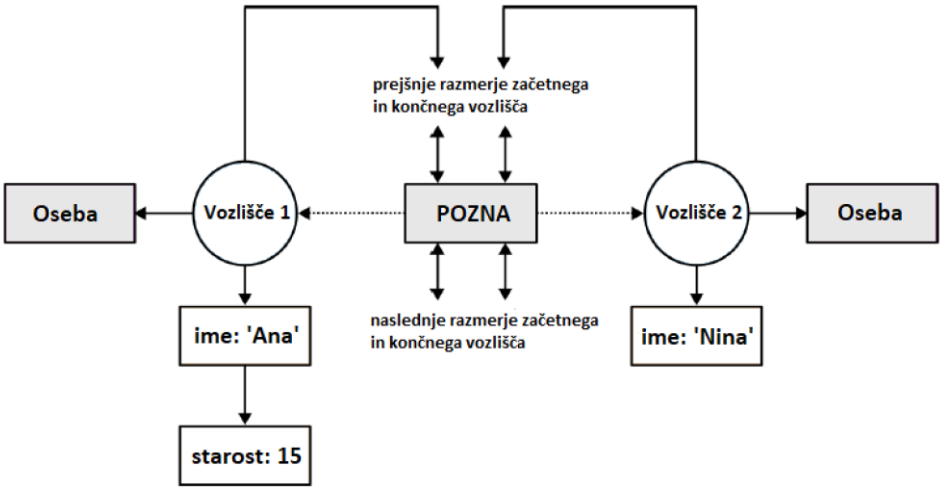
Slika 16: Struktura zapisov v shrambi vozlišč in razmerij



Vir: lastno delo.

Shrambi vozlišč in razmerij zagotavljata strukturo grafa, ne pa tudi lastnosti. Obe shrambi uporabljata zapise fiksnih dolžin, zato je lokacija nekega zapisa takoj določena. To zagotavljajo izjemno hitra prečkanja grafov. Na sliki 17 je prikazano, kako različne shrambe med seboj komunicirajo. Vsako izmed dveh vozlišč vsebuje kazalec na prvo lastnost in prvo razmerje v verigi razmerij. Da pridemo do drugih lastnosti vozlišča, sledimo enojno povezanem seznamu, ki se začne s kazalcem na prvo lastnost. Da dobimo razmerja, sledimo kazalcu do prvega razmerja iz dvojno povezanega seznama (do razmerja 'POZNA' v našem primeru) in od tu naprej po seznamu razmerij za izbrano vozlišče (dvojno povezan seznam začetnega ali končnega vozlišča), dokler ne najdemo razmerja, ki nas zanima. Ko ta zapis dobimo, lahko preberemo lastnosti razmerja (če obstajajo) na enak način kot pri lastnostih vozlišč, tj. z enojno povezanim seznamom, ali pa raziskujemo zapis vozlišč, ki ju najdeno razmerje povezuje. S kazalci na dva dvojno povezana seznama pridemo do vseh razmerij začetnega oz. končnega vozlišča. Dvojno povezan seznam nam preprosto omogoča, da hitro preberemo ta seznam v obe smeri in učinkovito dodajamo ali brišemo povezave.

Slika 17: Shranjevanje grafa



Vir: lastno delo.

Lastnosti grafa so shranjene na disku v datoteki *neostore.propertystore.db*, kjer so podatki shranjeni v obliki ključ-vrednost. Ker Neo4j dovoljuje lastnosti tako za vozlišča kot povezave, se na shrambe lastnosti sklicujejo tako zapisi vozlišč kot razmerij. Tudi te shrambe so sestavljene iz zapisov fiksnih dolžin. Vsak zapis lastnosti je sestavljen iz štirih delov in kazalca na naslednjo lastnost v verigi lastnosti (iz enojno povezanega seznama lastnosti). Vsaka lastnost lahko zasede od enega do vseh štirih delov, torej ima lahko zapis lastnosti največ štiri lastnosti (Robinson, Webber & Eifrem, 2015, str. 152).

2.8 Primerjava grafnih podatkovnih baz

Prva med grafnimi podatkovnimi bazami je bila res Neo4j, ki ji je kmalu sledilo še nekaj manj priljubljenih grafnih podatkovnih baz. V zadnjem času pa so tudi glavni akterji v industriji podatkovnih baz pripravili ponudbo le teh. Oracle z *Oracle Spatial and Graph*, Microsoft z *Graph Engine*, SAP s *SAP Hana* in IBM s *Compose for JanusGraph*. Nazadnje se je leta 2018 pridružil še Amazon z *Amazon Neptune* (Varangaonkar, 2018).

Med najpopularnejše grafne podatkovne baze pa poleg Neo4j spadajo še *AllegroGraph*, *OrientDB*, *ArangoDB*, *InfiniteGraph* ter *TigerGraph*. Ob vedno večji izbiri ponudnikov je pred nami težka odločitev, ko izbiramo primerno bazo za določen projekt (Wu, 2018). V nadaljevanju bomo pregledali nekaj kriterijev, ki nam pri tem pomagajo (Fernandes & Bernardino, 2018):

1. Avtohtono ali neavtohtono shranjevanje in procesiranje. Veliko grafnih podatkovnih baz je neavtohtonih, kar pomeni, da shranjujejo grafni model v relacijskem modelu, RDF trojicah ali ključ-vrednost parih. V teh primerih je zmanjšana hitrost prečkanja in preiskovanja grafa.

2. Možnost poizvedovanja v grafu. Nekatere baze imajo svoj poizvedovalni jezik, nekatere uporabljajo imperativne jezike, druge deklarativne. Pri nekaterih je možnost uporabe več različnih jezikov, pri drugih ne.
3. Možnost skaliranja. Npr. TigerGraph in InfiniteGraph omogočata vertikalno in horizontalno skaliranje za shranjevanje in procesiranje, Amazon Neptune ne omogoča horizontalnega skaliranja, Neo4j ga omogoča samo v *Enterprise* različici. Pri Janus Graph in ArangoDB je možno horizontalno skaliranje samo za shranjevanje, ne pa tudi procesiranje.
4. Podpora OLTP ali OLAP. Primerjali smo ju že v razdelku 2.2.
5. Možnosti delovanja. ArangoDB na primer ne pozna vgrajenega načina, temveč samo dostop preko vmesnika REST. OrientDB, Neo4j, JanusGraph omogočajo oba načina, OrientDB pozna celo tretji način, pomnilniški, ki je najhitrejši in vse podatke shranjuje samo na pomnilniku, kar pa pomeni, da je zelo omejen pri količini podatkov.
6. Večmodelni (ang. multi-model) SUPB. Veliko podatkovnih baz se odloča za podporo več tipov shranjevanja. Tako je bil npr. OrientDB sprva zasnovan kot dokumentna shramba, sedaj pa podpira dokumentne, grafne in ključ-vrednost shrambe. ArangoDB prav tako.
7. Shema. Res, da je ena izmed prednosti NoSQL podatkovnih baz ta, da ne potrebujejo sheme, vseeno pa bi si jo v nekaterih primerih želeli imeti ob načrtovanju in arhitekturi modelov. Posledično jo nekatere grafne podatkovne baze imajo (AllegroGraph, JanusGraph, TigerGraph), nekatere imajo to možnost na izbiro (Neo4j), nekatere pa je sploh ne nudijo (OrientDB, SAP HANA).

2.9 Uporaba grafnih podatkovnih baz

Količina podatkov se neprestano povečuje in v današnjem času podjetja ne le upravljajo s podatki, ampak si morajo ustvariti tudi kritični vpogled v obstoječe podatke. To pomeni, da razmerja med podatki postanejo še bolj pomembna kot podatki sami. Obstaja veliko primerov uporabe grafnih podatkovnih baz, spodaj jih je navedenih le nekaj:

- priporočilni sistemi: zaznavanje novih interesov strank in priporočanje ustreznih izdelkov, vsebin ali storitev,
- odkrivanje goljufij: odkrivanje zlorab in kaznivih dejanj v realnem času s hitro analizo podatkovnih razmerij,
- upravljanje infrastrukture omrežja in baz podatkov: analiziranje odvisnosti v omrežnih sistemih (npr. kakšne so posledice težave na določenem strežniku),
- socialna omrežja: modeliranje interakcij med ljudmi,
- problem pretoka: iskanje najboljše poti čez omrežje (npr. telekomunikacijska, plinska, letalska omrežja),
- preglednost dobavne verige: modeliranje kompleksnih verig z vsemi možnostmi prilagoditve, končna rešitev je veliko večja preglednost, ki pomaga pri iskanju neučinkovitosti in racionalizaciji poslovanja,

- upravljanje z matičnimi podatki: organiziranje in obdelava podatkov za namen sprejemanja ustreznih poslovnih odločitev,
- spletni iskalniki: za doseganje ustreznejših rezultatov iskanja,
- biologija: veliko modeliranja z grafnimi strukturami (npr. geni, molekule, beljakovine).

3 UPORABA V ZAVAROVALNIŠTVU

V zadnjem delu magistrske naloge se bomo posvetili uporabi grafnih podatkovnih baz v zavarovalništvu, sploh na področju razpoznave goljufij. Neo4j in Cypher bomo spoznali še s praktičnega vidika, ustvarili primer grafa in po njem poizvedovali.

V Sloveniji je ocena zavarovalniških goljufij za več kot 100 milijonov evrov letno, od bolj organiziranih, pa do kopice manjših goljufij (Košir, 2017). V Združenih državah Amerike je ta številka precej večja in znaša kar 80 milijard dolarjev (Coalition Against Insurance Fraud, 2018). Kako točno je število zavarovalniških goljufij, je nemogoče vedeti, a podatki kažejo, da se število odkritih goljufij veča tako v svetu kot pri nas. Mogoče je razlog za to trud zavarovalnic pri odkrivanju le teh ali pa se njihovo število res povečuje. Glede na dejstvo, da je vedno več spletnega poslovanja tudi pri sklepanju zavarovalniških pogodb in prijavi škod ter vse manj osebnega stika z zavarovanci, je možno, da je goljufijo še lažje izvesti. Čeprav so kazni za odkrite goljufe precej visoke (v Sloveniji od enega do treh let zapora: v primeru, da je bila goljufija samo načrtovana in še ne izvedena, je kazen do enega leta zapora, v primeru izvršitve in vložitve lažne prijave škodnega dogodka pa je kazen do treh let zapora (Zavarovalnica Triglav, 2015, str. 6)), se goljufi prilagajajo spreminjajoči tehnologiji in uporabljajo vse bolj izpopolnjene metode. Za Slovenijo velja ocena, da v povprečju vsak posameznik na leto plača od 50 do kar 80 evrov več premij zaradi nepoštenosti manjšine (Košir, 2017). V zavarovalnicah, kjer je večji delež goljufij, morajo zviševati cene premij, kar je slabo za konkurenčnost, saj zavarovanci ne bodo več hoteli sklepati zavarovanj pri njih. Zaradi goljufij pa zavarovanci niso na slabšem le zaradi višjih premij, ampak tudi zaradi storitev, saj zavarovalnice zaradi prevelikih zlorab prenehajo ponujati kritje za določene dogodke ali pa navzgor omejijo izplačila. To je se v Sloveniji zgodilo po izrazitem povečanju zahtevkov zaradi poškodb mehkih tkiv pri prometnih nesrečah (nekatere med njimi tudi zaigrane). V Združenih državah Amerike je to poznano pod izrazom *whiplash for cars*. Ker je bolečine v vratu (sicer pogosta posledica prometnih nesreč) nemogoče dokazati, so mnogi hlinili to poškodbo, a posledično so zavarovalnice s časom izključile zavarovanje mehkih tkiv iz svojih polic oz. omejile višino izplačila.

Do zavarovalniške goljufije pride v primeru, da:

- pride do uničenja ali poškodbe s ciljem prejema odškodnine s strani zavarovalnice,
- se sklene zavarovanje po nastanku poškodbe,
- se sklene zavarovanje z napačnimi ali zavajajočimi dokumenti ali se priloži lažne dokumente.

Najpogostejše oblike goljufij so tako imenovane mehke goljufije, pri katerih se škodni dogodek res zgodi, a zavarovanci skušajo z lažnimi podatki dobiti višjo odškodnino. V tem primeru gre za priložnostne goljufe in odkrivanje tovrstnih primerov je razmeroma preprosto (Zavarovalnica Triglav, 2015). V drugih primerih pa so odškodninski zahtevki povsem lažni oziroma goljufi načrtno priredijo dogodek ali dokaze. Tu gre za organizirane goljufe, ki v večini primerov delujejo usklajeno. Ti za zavarovalnice predstavljajo večji problem in višjo denarno škodo, saj so zelo težko odkriti. Odkrivanju takih primerov goljufij se bomo posvetili s pomočjo grafne podatkovne baze Neo4j in predstavili učinkovite načine, ki ne samo odkrivajo, ampak tudi preprečujejo potencialne goljufije. V splošnem je ocenjeno, da kar 80 odstotkov vseh zavarovalniških goljufij predstavlja organiziran kriminal (Malby in drugi, 2013).

Tradicionalne metode odkrivanja goljufij, kot je odstopanje od običajnih in pričakovanih vzorcev, se osredotočajo na diskretne podatke in ne na povezave med njimi. Čeprav so diskretne metode koristne za razpoznavo goljufov, ki delujejo individualno, pa niso dovolj uspešne pri zaznavanju organiziranih kriminalnih združb. Poleg tega so diskretne metode nagnjene k napačnim pozitivnim rezultatom (ang. false positive), ki ustvarjajo nezaželene stranske učinke pri zadovoljstvu zavarovancev in izgubljene priložnosti za prihodke. Iskanje vzorcev v obsežnih in kompleksnih podatkih pa postaja izvedljivo z uporabo grafnih podatkovnih baz. Da bomo lažje razumeli, kako uporabna je lahko grafna podatkovna baza pri odkrivanju goljufij, si najprej oglejmo večplastni pristop pri razpoznavanju goljufij, predlagan pri Gartner, Inc. Ta je bil privzet s strani vseh, ki se z goljufijami soočajo (Litan, 2011).

Slika 18: Večplastni pristop odkrivanja goljufij

Vir: Prirejeno po Tiao (2018).

Analitiki pri Gartnerju so mnenja, da nobena posamezna plast na sliki 18 ne zadostuje za preprečitev goljufij v poslovnih sistemih. Za to je potrebna kombinacija spremljanja in nadzora, od najosnovnejših varnostnih ukrepov in vrednotenja posameznikov ter dejanj do

najbolj zapletenih povezav med njimi (analiza odnosov). Večina današnjih rešitev za preprečevanje goljufij obravnava samo prve tri oz. štiri plasti priporočenega pristopa podjetja Gartner. Primer so banke, ki večinoma uporabljajo sisteme za spremljanje transakcij, za katere so podlaga relacijske podatkovne baze. Ti pa trende oz. zlorabe razpoznajo šele po tem, ko so se ti že zgodili in ne omogočajo razpoznave organiziranih ter povezanih goljufov, ki goljufije načrtujejo. Poleg tega so organiziranim goljufom že dobro poznani načini, ki bi razpoznali njihova dejanja in vedo, kako se odkrivanju izogniti. Rešitev je v analizi povezav, kjer z odnosi in vzorci prepoznamo mogoče goljufije in to v realnem času.

Analiza goljufij v realnem času lahko zaščiti podjetja pred velikimi denarnimi izgubami, vendar se še vedno veliko podjetji ne odloča za integriranje te strategije ter ostaja pri velikem vlaganju časa v razvoj kompleksnih modelov. Težava pri odkrivanju goljufij v realnem času izhaja iz treh glavnih dejavnikov: vedno večje kompleksnosti modelov, velike rasti obsega podatkov, ki jih morajo modeli obdelati ter vedno večjih zahtev današnjih strank. Posledično sistemi, ki izvajajo algoritme za odkrivanje goljufij na relacijskih podatkovnih bazah, pogosto ne uspejo dovolj hitro obdelati podatkov in zahtevajo preveč časa, preden vrnejo rezultate. Stranke ne bi tolerirale takih sistemov, ki bi od njih na primer zahtevali, da čakajo na telefonu ali na odziv spletne strani po več minut, da bi samo predložili zavarovalni zahtevek. Ključni dejavnik učinkovitosti sistemov za razpoznavo goljufij je zmožnost prečkanja in analize razmerij med strankami, produkti, lokacijami in drugimi entitetami. Prav to zahteva tudi zadnja, peta plast Gartnerjevega pristopa, ki je z grafnimi podatkovnimi bazami lahko dosegljiva. Razumevanje povezav med podatki in pridobivanje pomena iz teh povezav ne pomeni nujno zbiranja novih podatkov. Iz obstoječih podatkov je mogoče pridobiti nov vpogled samo s preoblikovanjem problema in njegovo obravnavo na nov način, kot graf.

Začnimo torej z obravnavo problema goljufij in kako se vse sploh začne. Kot rečeno, zanimale nas bodo organizirane goljufije, torej skupine dveh ali več goljufov, ki načrtujejo prevare in velikokrat to naredijo večkrat, z namenom pridobivanja denarja s strani zavarovalnic. Prav taki primeri oškodujejo zavarovalnice za velike vsote denarja in jim predstavljajo velik problem, ki ga je zelo težko odkriti. V take verige goljufov se namreč povežejo celo zdravniki, odvetniki, priče in zavarovalniški agenti. Do takega povezovanja ne pride samo med znanci, ki bi se jim slučajno porodila ta ideja, ampak so vloge v celotnem procesu goljufije celo naprodaj ali pa se zanje nekomu plača. Na primer prodaja mesta v avtomobilu, s katerim se bo povzročilo prometno nesrečo, z namenom pridobivanja odškodnin vsem vpletenim. Ali pa ponujanje plačila osebi za mesto sopotnika v avtomobilu, ki bo po dogodku lahko potrdila »pristnost« dejanja.

Predpostavljamo, da imajo zavarovalnice vse relevantne podatke o zavarovancih, zavarovalcih in upravičencih, ki so jim te podatke zavestno dali oziroma zaupali. Le s tem in ostalimi podrobnostmi o škodnih dogodkih je mogoča nadaljnja obravnava in analiza dogodkov.

3.1 Zloraba identitete

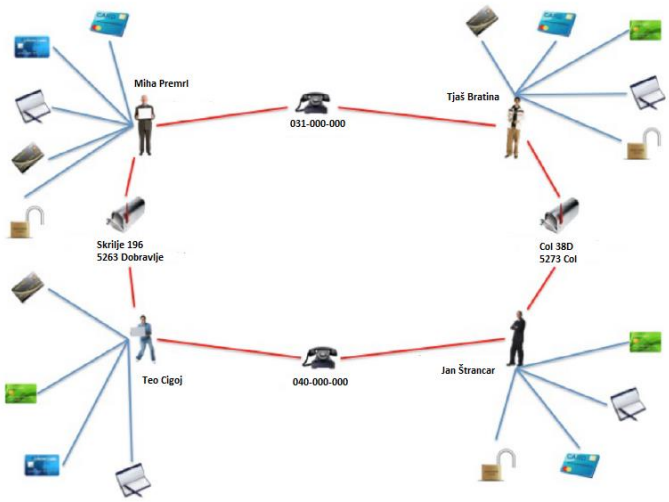
Začne se lahko že zelo preprosto in sicer z ustvarjanjem lažnih identitet. Zloraba identitete je zapleten problem s pomembnimi finančnimi in pravnimi posledicami. Kraje identitet in sintetične prevare identitete (ang. synthetic identity fraud) ustvarjajo dvoumne, napačno predstavljene in nejasne identitete, zaradi česar imajo zavarovalnice težko nalogo pri ugotavljanju, ali je stranka dejansko tista, za katero trdi, da je. Goljufije z identiteto se pojavljajo v različnih oblikah, npr. z manipulacijo imena (uporaba različic istega imena), ukradeno identiteto, izmišljeno identiteto (ob pomoči tehnologije do popolnoma lažnih identitet, z vsemi zahtevanimi podatki in zgodovino). V velikem deležu pa se goljufije pojavljajo z uporabo sintetičnih identitet, pri katerih goljufi uporabljajo kombinacije legitimnih in izmišljenih informacij več različnih ljudi, za ustvarjanje nove fiktivne identitete, ki se nato uporabi za izvajanje goljufij. Te identitete se nato uporabi pri zavarovalnih policah, vpisih v zdravstvene storitve, vlogah za kredite (IBM, 2018), itn. Zloraba identitete je kompleksna in goljufije te vrste so veliko bolj razširjene, kot si ljudje mislimo. Ker pa tudi za goljufe te vrste goljufij predstavljajo visoke stroške pri pridobivanju novih podatkov za ponarejene identitete, te podatke »reciklirajo« z namenom ustvarjanja še več identitet.

Za uprizoritev takega primera, opazujmo delovanje dveh goljufov:

- Jaka Jamnik, naslov: Skrilje 196, 5263 Dobravlje, telefonska številka: 031-000-000
- Marko Peljhan, naslov: Col 38D, 5273 Col, telefonska številka: 040-000-000

Namesto kraje identitete, sta goljufa s kombiniranjem le dveh lastnih osebnih podatkov (naslov in telefonska številka), dobila kar štiri lažne identitete z izmišljenimi imeni, kot prikazuje slika 19.

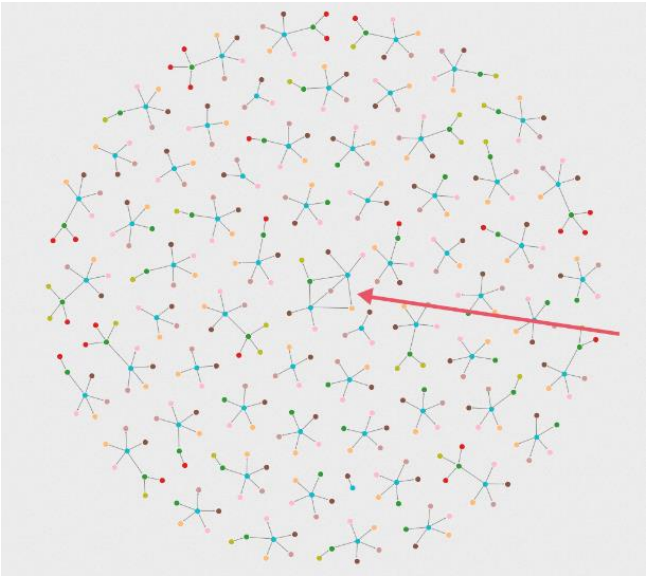
Slika 19: Lažne identitete



Vir: Prirejeno po Sadowski & Rathle (2017).

Goljufi želijo povečati število vseh kaznivih dejanj, da bi prejeli več denarja. S povezovanjem še več goljufov v to združbo pa ne narašča le finančno breme zavarovalnic, ampak tudi zahtevnost preračunavanja, ki je potrebna za razkritje družbe. In s številom goljufov problem eksponentno narašča. Če vključimo tretjo osebo, se število sintetičnih identitet poveča na devet. Obroč n goljufov ($n \geq 2$), ki si delijo m osebnih podatkov (npr. ime, datum rojstva, telefonska številka, naslov, EMŠO, itn.) lahko ustvari do n^m sintetičnih identitet, kjer je vsaka identiteta (če si jo predstavljamo kot vozlišče v grafu), povezana z $m \times (n-1)$ drugimi vozlišči, torej je povezav skupaj kar $(n^m \times m \times (n-1))/2$. S tradicionalnimi tehnologijami relacijskih podatkovnih baz je modeliranje tolikšnih povezovanj in združevanj lahko zelo zahtevno in ne omogoča preračunavanj z realnem času. Z grafi pa lahko problem predstavimo z vozlišči in povezavami, da ugotovimo, če se kateri izmed podatkov pojavljajo pri več različnih osebah, kot kaže slika 20.

Slika 20: Iskanje vzorcev

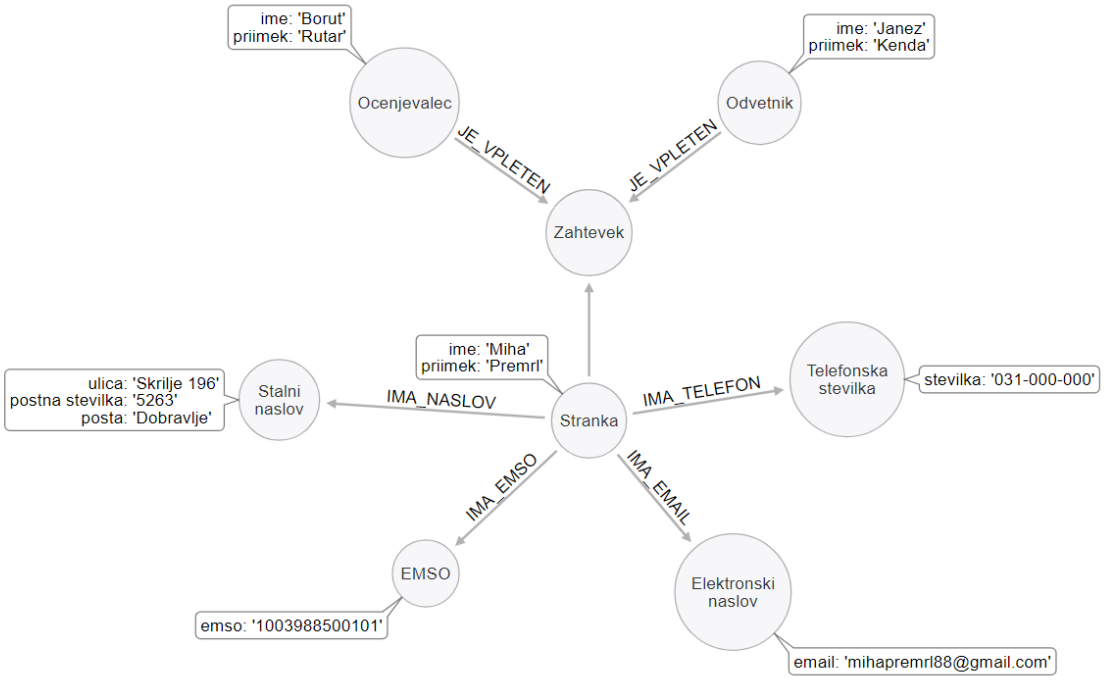


Vir: Villedieu (2018).

V našem primeru je vsaka oseba v grafu predstavljena kot vozlišče z lastnostjo imena in priimka. V praksi bi poleg teh lastnosti spadala še vrsta sklenjenega zavarovanja ter številka zavarovalne police. Iz tega vozlišča potem vodijo povezave do osebnih podatkov, ki so predstavljeni z novimi vozlišči, npr. naslov prebivališča, telefonska številka in elektronska pošta. Shema je prikazana na sliki 21.

Sheme grafov smo ustvarili z uporabo odprtokodnega orodja Arrow Tool, ki je namenjen predstavitvam in prikazom podatkovnih modelov. Ko zaključimo z oblikovanjem sheme, jo lahko neposredno izvozimo v Cypher sintaksi, ki jo potem poženemo v Neo4j Browser in tako definiramo graf tudi v podatkovni bazi.

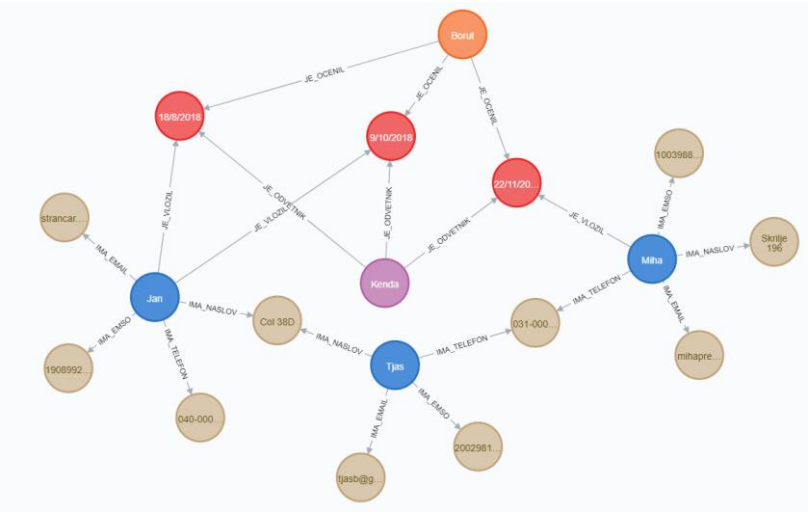
Slika 21: Shema zavarovanca



Vir: lastno delo.

Iz vozlišča zavarovanca pa vodijo povezave še do odškodninskih zahtevkov, ki so lahko naprej povezani tudi z ocenjevalci in odvetniki ali zdravniki. V aplikaciji Neo4j Desktop smo ustvarili nov projekt in v njem nov graf. V vmesniku, Neo4j Browser, kjer lahko izvajamo poljubne Cypher poizvedbe, smo definirali vozlišča in povezave, med katerimi bomo iskali podatke, ki so mogoče skupni več strankam zavarovalnice. Cypher poizvedbe za kreiranje grafa so navedene v prvi prilogi. Celoten preprost graf je prikazan na sliki 22.

Slika 22: Primer grafa za odkrivanje sintetičnih identitet



Vir: lastno delo.

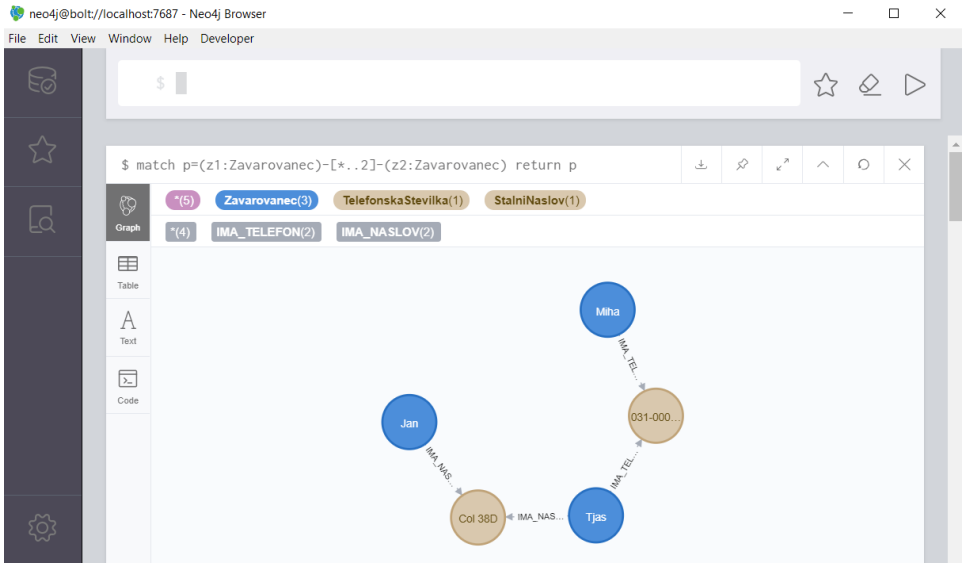
Da smo dobili zavarovance s skupnimi osebnimi podatki, smo zagnali spodnjo poizvedbo:

MATCH p = (z1:Zavarovanec)−[*..2]−(z2:Zavarovanec) RETURN p

Ta nam je v vmesniku vrnila del grafa, ki ga vidimo na sliki 23. V poizvedbi lahko vidimo, da nismo definirali usmerjene povezave, tj. nismo točno določili, katero je začetno in katero končno vozlišče. To smo dosegli tako, da nismo postavili puščice −>, ampak samo − simbol. Z [*..2] pa smo določili, da sta med zavarovancema dve povezavi, povezavi do potencialnega skupnega podatka.

Zavarovance smo v vmesniku označili z modro barvo, njihove osebne podatke pa z rjavo.

Slika 23: Iskanje skupnih podatkov – prva poizvedba



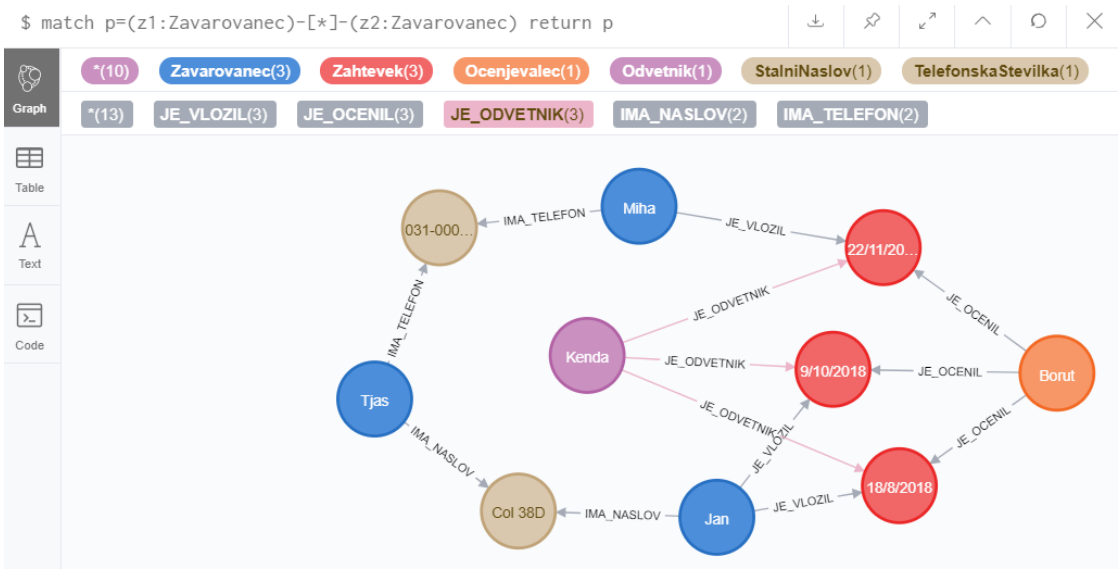
Vir: lastno delo.

V nadaljevanju bi nas seveda zanimalo, če so te osebe vložile kakšne odškodninske zahtevke. Da smo poizvedbo nadgradili, smo pognali še eno poizvedbo, ki nam je vrnila rezultat na sliki 24.

MATCH p = (z1:Zavarovanec)−[*]−(z2:Zavarovanec) RETURN p

S to poizvedbo smo prejšnji dodali še vsa vozlišča, ki so lahko poljubno oddaljena od že prej odkritih vozlišč. Dobimo podatke o odškodninskih zahtevkih, pa tudi odvetnikih in ocenjevalcih. Na ta način smo hitro odkrili sumljive stranke in njihove prijavljene zahtevke, ki nas lahko mogoče vodijo tudi do goljufij s strani ocenjevalcev ali odvetnikov.

Slika 24: Iskanje skupnih podatkov – druga poizvedba



Vir: lastno delo.

Ko bi imeli ogromno podatkov z večjim številom zgoraj opisane situacije, bi hoteli vedeti, pri komu naj s preiskovanjem začnemo. Kot prve bi seveda želeli odkriti najhujše primere goljufij, da jih čim hitreje tudi ustavimo in tako preprečimo nove denarne izgube zavarovalnice. Tokrat nam je v pomoč poizvedba, ki združbe povezanih goljufov (t. i. kroge goljufov) razvršča padajoče, glede na število vpletenih oseb v določen krog.

```
MATCH      (z:Zavarovanec)−[]−>(osebnipodatki)
WITH       osebnipodatki,
           count(z) AS VelikostKroga
MATCH      (osebnipodatki)<−[]−(z)
WITH       collect(z.ime) AS zavarovanci,
           osebnipodatki, VelikostKroga
WHERE      VelikostKroga > 1
RETURN     zavarovanci AS KrogGoljufov,
           labels(osebnipodatki) AS TipPodatka,
           VelikostKroga
ORDER BY   VelikostKroga DESC
```

Na primeru našega grafa dobimo rezultate na sliki 25. Pojavita se samo dva kroga, z dvema vpletenima osebama, ki si delita en podatek, telefonsko številko oziroma naslov prebivališča.

Slika 25: Izpis krogov goljufov, sortiranih po številu skupnih podatkov

```
$ MATCH (z1:Zavarovanec)-[]->(osebnipodatki) WITH osebnipodatki, count(z1) AS VelikostKroga MATCH (osebnipodatki)<-[]-(z1) WITH collect(z1.ime)…
```

Table | Text

KrogGoljufov	TipPodatka	VelikostKroga
["Tjas", "Miha"]	["TelefonskaStevilka"]	2
["Jan", "Tjas"]	["StalniNaslov"]	2

Vir: lastno delo.

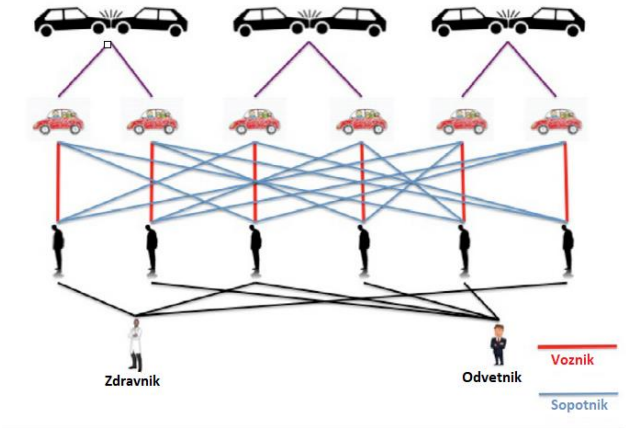
Ob takem preiskovanju bomo seveda imeli tudi primere, ki ne bodo sporni. Vsaka družina si na primer deli naslov prebivališča ali stacionarno telefonsko številko. Ampak če smo na sledi družini, potem bi se tem osebam zelo verjetno morali ujemati tudi priimki, kar pa v našem primeru ni držalo. Grafne podatkovne baze nam torej ne bodo razkrile sumljivih primerov same, ampak jih moramo izluščiti sami z lastno presojo.

3.2 Prometne nesreče

Avtomobilska zavarovanja so naslednja v vrsti, ki se v veliki meri soočajo z goljufijami. Te vključujejo lažne prometne nesreče, a pravi denar. Zločinci za temi dejanji so dobro organizirani, od voznikov, sopotnikov, prič in zdravnikov, od ene nesreče do druge. Iste osebe nastopajo v več različnih prometnih nesrečah, le vloge si med seboj menjujejo, kar privede do kompleksnih shem, kot kaže slika 26. Slabost zavarovalnic je, da se soočajo s tako velikim številom zahtevkov, da zaposleni težko opazijo sumljivo shemo v podatkih. Običajno je v krogu goljufov nekdo vodja, ki naredi načrt za celotno akcijo, postopek je po navadi sledeč:

- Vodje določijo primerne člane, ki bodo nastopili v vlogi voznika, sopotnika, priče, itd. Ti so tisti, ki bodo kasneje od zavarovalnic zahtevali odškodnino. Te osebe lahko obstajajo ali ne, saj se tudi v tem primeru lahko uporabi sintetične identitete.
- Vodje poiščejo avtomobile, s katerimi bo povzročena nesreča.
- Izvedba nesreče. Lahko se res zgodi ali pa ne, vendar mora biti vse o njej dokumentirano. Vsi udeleženi se morajo točno dogovoriti o času, kraju in scenariju.
- Po nesreči vsi vpleteni izpolnijo odškodninske zahtevke, poročila o nesreči in povzročitelj podpiše izjavo o odgovornosti. Na tem koraku lahko goljufom pomagajo tudi zdravniki in ocenjevalci za popis materialne škode, ki potrdijo in utemeljijo zahtevke.
- Zavarovalnica obravnava zahtevek, ki ga lahko dodatno razišče ali ne. Velikokrat pa dobra priprava goljufije v vsakem primeru goljufe ščiti pred razkritjem.

Slika 26: Shema avtomobilskih goljufij

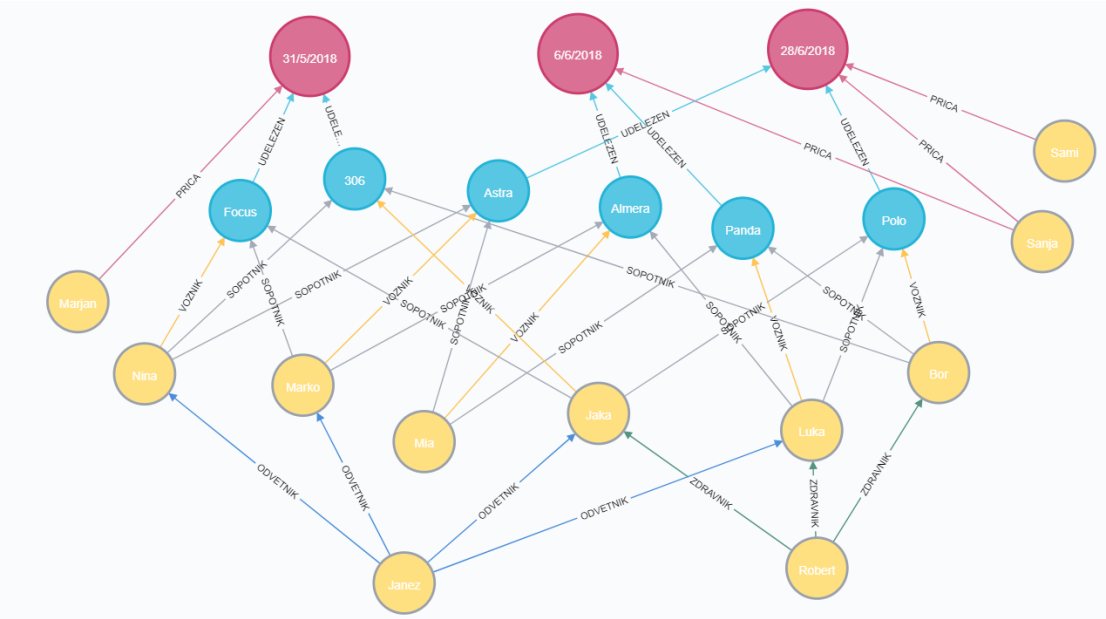


Vir: Prirejeno po Sadowski & Rathle (2017).

Združene države Amerike (Coalition Against Insurance Fraud, 2018) podajajo ocene, da je možno od ene prometne nesreče z dvema avtomobiloma zahtevati 20.000 dolarjev na vpleteno osebo in 5.000 dolarjev za avtomobil. Če je v vsakem avtomobilu pet oseb, je skupni znesek 210.000 dolarjev (Sadowski & Rathle, 2017).

Ponovno bomo kreirali preprost primer grafa, na katerem bomo prikazali, kako uporabne so grafne podatkovne baze pri reševanju zgoraj opisanega problema. S podatki, ki jih imajo zavarovalnice na voljo, bi lahko ustvarile sheme, kot jo imamo na sliki 27. Vozlišča v rumeni barvi predstavljajo osebe, modra volišča avtomobile ter rdeča vozlišča povzročene prometne nesreče.

Slika 27: Shema prometnih nesreč



Vir: lastno delo.

Recimo, da zaposlenega v službi za odkrivanje zavarovalniških goljufij in nepravilnosti zanima vse, kar je povezano s točno določeno prometno nesrečo. V našem primeru nas recimo zanima nesreča, ki se je zgodila v Vipavi, 28.6.2018. Če poženemo spodnjo poizvedbo

```
MATCH                (nesreca)<−[]−(avtomobil)<−[r]−(oseba)
WHERE                nesreca.lokacija = 'Vipava' AND nesreca.datum = '28/6/2018'
RETURN DISTINCT oseba.ime AS Ime, type(r) as Vloga, nesreca.lokacija AS Lokacija,
avtomobil.model AS Model,
```

nam rezultat vrne seznam vseh vpletenih oseb v prometni nesreči in kakšne so bile njihove vloge. Če nas zanima še podatek o tem ali so bile pri prometni nesreči mogoče navzoče priče, poženemo še poizvedbo

```
MATCH                (nesreca)<−[]−(avtomobil)<−[r]−(oseba)
WHERE                nesreca.lokacija = 'Vipava' AND nesreca.datum = '28/6/2018'
RETURN DISTINCT oseba.ime AS Ime, type(r) as Vloga,
                     nesreca.lokacija AS Lokacija, avtomobil.model AS Model
UNION
MATCH                (nesreca)<−[r:PRICA]−(oseba)
WHERE                nesreca.lokacija = 'Vipava' AND nesreca.datum = '28/6/2018'
RETURN DISTINCT oseba.ime AS Ime, type(r) as Vloga,
                     nesreca.lokacija AS Lokacija, null as Model,
```

ki vrne rezultate podobne prejšnjim, le da sta tu dodana še podatka o dveh pričah, ki sta bili prisotni pri nesreči. Rezultati so prikazani na sliki 28.

Slika 28: Rezultat poizvedbe o vpletenih osebah prometne nesreče

$ MATCH (nesreca)<-[]-(avtomobil)<-[r]-(oseba) WHERE nesreca.lokacija = 'Vipava' RETURN DIS…

Table | Text | Code

Ime	Vloga	Lokacija	Model
"Nina"	"SOPOTNIK"	"Vipava"	"Astra"
"Mia"	"SOPOTNIK"	"Vipava"	"Astra"
"Marko"	"VOZNIK"	"Vipava"	"Astra"
"Luka"	"SOPOTNIK"	"Vipava"	"Polo"
"Jaka"	"SOPOTNIK"	"Vipava"	"Polo"
"Bor"	"VOZNIK"	"Vipava"	"Polo"
"Sami"	"PRICA"	"Vipava"	null
"Sanja"	"PRICA"	"Vipava"	null

Vir: lastno delo.

Ko vidimo, da sta bili pri prometni nesreči navzoči dve priči, lahko za celoten graf takoj preverimo, če se mogoče katera izmed teh oseb pojavlja kot priča pri več prometnih nesrečah.

```
MATCH      (o:Oseba)−[p:PRICA]−>(n:Nesreca)
WITH       count(p) AS KolikokratPrica,
           o.ime AS Prica, collect(n.lokacija) as Lokacija
RETURN     Prica, KolikokratPrica, Lokacija
ORDER BY KolikokratPrica DESC
```

V našem primeru imamo v celotnem grafu skupno tri priče, a ena se je pojavila pri kar dveh prometnih nesrečah, kot vidimo na sliki 29. Obstaja možnost, da ta oseba deluje organizirano z ostalimi vpletenimi v prometni nesreči, z namenom potrjevanja pristnosti nesreče možnim preiskovalcem.

Slika 29: Pogostost vpletenosti v vlogi priče prometne nesreče

Ime_Price	Kolikokrat_Prica	Lokacija_Nesrece
"Sanja"	2	["Razdrto", "Vipava"]
"Marjan"	1	["Ajdovscina"]
"Sami"	1	["Vipava"]

Vir: lastno delo.

Preiskovanje se nadaljuje tako, da raziščemo, če so se udeleženci iz obravnavane nesreče pojavili še v kateri prometni nesreči. To bi namreč nakazovalo na povečano verjetnost goljufije. V vmesnik zapišemo naslednjo poizvedbo, ki bo vrnila prometne nesreče, v katere je bila vpletena katera izmed oseb ali kateri izmed avtomobilov iz nesreče v Vipavi.

```
MATCH
(nesreca1)<−[]−(avtomobil1)<−[]−(oseba)−[]−>(avtomobil2)−[]−>(nesreca2:Nesreca)
WHERE                nesreca1.lokacija = 'Vipava' AND nesreca1.datum = '28/6/2018'
RETURN DISTINCT nesreca2.lokacija AS Lokacija, nesreca2.datum as Datum
```

Ta poizvedba nam vrne podatka še o dveh prometnih nesrečah. Prva, ki se je zgodila v Ajdovščini na datum 31.5.2018 in druga, ki se je zgodila na Razdrtem, 6.6.2018. Kar je sumljivo. Zdaj želimo razkriti celoten krog, vse osebe in avtomobile, vpletene v goljufijo. To pomeni, da začnemo pri eni nesreči, po povezavah sledimo do druge, naprej do tretje nesreče, itn. Za relacijske podatkovne baze bi to pomenilo izvajanje kompleksnih združevanj tabel, ki bi bile zahtevne po definiciji in času izvedbe. Z grafno podatkovno bazo je poizvedba enostavna in nam vrne seznam vseh udeležencev prometne nesreče, osumljenih sodelovanja pri goljufiji.

```
MATCH (nesreca)<−[*]−(potencialnigoljuf:Oseba) WHERE nesreca.lokacija = 'Vipava'
RETURN DISTINCT potencialnigoljuf.ime AS Ime, potencialnigoljuf.priimek AS Priimek
```

Tehnologija grafnih podatkovnih baz nam omogoča, da postavimo zahtevna vprašanja o povezavah v naših podatkih. Poleg tega nam ne pomagajo samo pri odkrivanju goljufij, ki so že bile storjene, ampak tudi pri iskanju sumljivih vzorcev v realnem času. Ti so nato

podrobneje raziskani in tako preprečimo goljufije, ki bi se verjetno zgodile v prihodnosti. Poiščemo lahko verige ljudi, ki so vpleteni v različne nesreče. Če je nekdo vpleten v nesrečo z osebo, ki je vpletena v nesrečo z nekom drugim, in je ta spet udeležena v nesreči z nekom, itd. smo morda na sledi obroču goljufov. Cypher poizvedba, ki nam je pri tem v pomoč, je naslednja:

```
MATCH (oseba1:Oseba)-[*..2]->(nesreca1:Nesreca)<-[*..2]-(oseba2:Oseba)-[*..2]->(nesreca2:Nesreca)<-[*..2]-(oseba3:Oseba)-[*..2]->(nesreca3:Nesreca)
RETURN DISTINCT oseba1, oseba2, oseba3
```

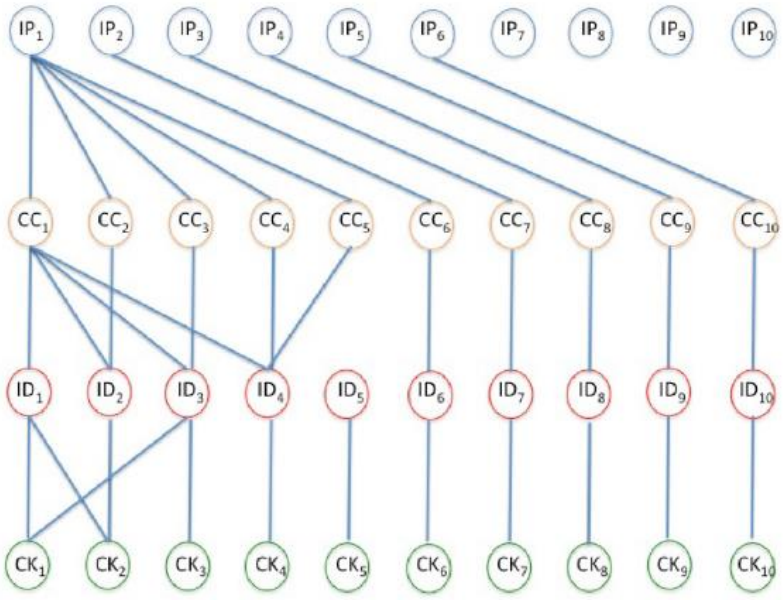
Ne samo, da bi nas poizvedba seznanila z vsemi sumljivimi primeri, ampak to poizvedbo bi morali sprožiti tudi v vseh ključnih trenutkih stranke zavarovalnice. Na primer, ko se ta odloča za neko polico, ko ureja novo zavarovanje avtomobila, ob vloženem zahtevku po prometni nesreči, itd.

3.3 Elektronsko poslovanje

K uspešnosti goljufij v zavarovalništvu pripomore tudi možnost upravljanja z zavarovanji prek spleta. Sploh avtomobilska zavarovanja so eden izmed najhitreje rastočih produktov, kar zadeva elektronsko poslovanje. Na primer v letu 2015 je prodaja obveznega avtomobilskega zavarovanja preko spleta in telefona narasla za 39 odstotkov, prodaja kasko avtomobilskega zavarovanja preko spleta in telefona pa za 29 odstotkov (Varčuj24, 2016). V Sloveniji vse zavarovalnice sicer še ne nudijo teh možnosti, a bodo prav gotovo kmalu sledile temu trendu. Prek spleta lahko sklenemo zavarovanje, oddamo odškodninske zahtevke ter plačujemo s kreditnimi, plačilnimi karticami oz. spletno banko. To pa pomeni vedno manj osebnega stika med zavarovanci in zastopniki. Prav tako ni nobenega stika niti med osebo, ki kupi zavarovalno polico, in osebo, ki vloži zahtevek po določeni nesreči.

Pomembno je, da so tudi vse informacije iz spletnih sej vključene v odkrivanje goljufij, njihovo analizo in preiskavo. Je pa to področje za zavarovalnice še vedno precej novo, zato se ga morajo naučiti od tistih, ki jim je elektronsko poslovanje primarno delo. Med analiziranjem goljufij v elektronskem poslovanju in analiziranjem zavarovalniških goljufij obstaja veliko podobnosti. Razlika pa je, da damo v analizi e-poslovanja manj teže fizičnim in vnesenim atributom, a več pozornosti podatkom, zajetim na internetu, ki potem določijo razmerja med subjekti in nakupi. Pri analizi goljufij v elektronskem poslovanju je namreč privzeto, da so vsi vneseni podatki nepravilni. Še vedno so to pomembni podatki, saj ustvarjajo pomembne vzorce o tem, kako ljudje ustvarjajo napačne podatke. A še pomembnejše je to, kar se skriva za informacijami, torej IP naslovi, številke kreditnih kartic, ID-ji naprav, ID-ji uporabnikov, piškotki, itd. Pri zavarovalniških goljufijah pa je vsa teža večinoma na uporabnikih in podatkih, ki jih vnesejo. To bi se moralo spremeniti.

Slika 30: Goljufije pri elektronskem poslovanju



Vir: Sadowski & Rathle (2017).

Primer goljufije ja lahko ta, prikazan na sliki 30 Prikazuje transakcije iz različnih IP naslovov, kjer je na naslovu IP_1 najverjetneje prišlo do goljufije. CC_i predstavljajo različne številke kreditnih kartic, ID_i unikatne uporabniške ID-je, CK_i pa shranjene piškotke iz sistema. Goljufija je bila zelo verjetno storjena, saj je bilo iz enega IP naslova izvedenih več transakcij in to z uporabo petih različnih kreditnih kartic. Ena od teh (CC_1) je bila uporabljena za več različnih uporabnikov in vsak izmed piškotkov CK_1 in CK_2 se sklicuje na dva različna uporabniška ID-ja, do česar v splošnem ne bi smelo priti (Sadowski & Rathle, 2017, str. 9). Goljufije bi lahko odkrili tudi tako, da bi raziskovali primere, kjer je več zavarovanj plačanih z isto številko kreditne kartice, a različnih IP naslovov, kar bi bilo tudi potrebno podrobneje raziskati. Ker bi grafne podatkovne baze te povezave zelo hitro preračunavale, bi lahko bile sumljive transakcije tudi zavrnjene. Na primer dve osebi iz različnih IP naslovov in z različnimi imeni ne bi mogli uporabiti iste številke kreditne kartice. Prav tako bi bila lahko preprečena izvršitev številnih transakcij iz istega IP naslova, z različnimi kreditnimi karticami.

Vse to nas pripelje do dejstva, da je za analizo in razpoznavo goljufij v zavarovalnicah nujno vključevanje podatkov, pridobljenih na spletu. Predvsem v trenutnem času, ob prenosu poslovanja na splet, morajo zavarovalnice za modeliranje goljufij shranjevati podatke tudi o ID-jih sej, ID-jih naprav, IP naslovih, ID-jih uporabnikov, metodah plačevanja ter piškotkih ob izvedenih transakcijah.

Ogledali smo si glavne probleme goljufij, s katerimi se zavarovalnice soočajo in nakazali, kako bi grafne podatkovne baze implementirale rešitve zanje. Niso pa to edine možnosti grafnih podatkovnih baz, ki bi služile zavarovalnicam. Upravljanje odnosov s strankami

(ang. customer relationship management) (v nadaljevanju CRM) in analiza družbenih omrežji se razvijata tako, da igrata vedno večjo vlogo v poslovni strategiji zavarovalnic in ne gre več samo za preprosto sledenje obnašanju strank. Trženje je sedaj osredotočeno posebej na odnose ter razmerja. Zaradi velike raznolikosti ter široke dostopnosti vseh podatkov, bi s shranjevanjem v grafne podatkovne baze zavarovalnice lažje napovedovale dejanja strank in jih po svojih željah usmerjale k drugačnim odločitvam. Na primer, če je pred kratkim najbližji prijatelj določene stranke zamenjal ponudnika zavarovanja, se je povečala možnost, da zavarovalnico zapusti tudi ta stranka. Na tem koraku je vredno premisliti, če bi ji ob obnovitvi zavarovalne pogodbe mogoče ponudili popust in jo tako odvrnili od teh pomislekov. Če povežemo še obilico drugih faktorjev, nam lahko hitro razkrijejo, katere stranke so nagnjene k temu, da bi zavarovalnico menjale in katere ne. Grafne podatkovne baze bi bile koristne tudi na področju geoprostorske analize. Tudi ta lahko doprinese k razpoznavi obročev goljufov, saj lahko npr. razkrijemo zdravnika, ki je odšel na teren iz svojega okrožja do kraja zaigrane prometne nesreče in potrdil zdravstveno stanje, bi pa bila to naloga katerega izmed zdravstvenih delavcev, ki so bili v neposredni bližini tega mesta (Marane, 2004). Geoprostorska analiza je ključna še pri identifikaciji zahtevkov po naravnih katastrofah, saj s točno lokacijo in geocentričnimi vremenskimi podatki in podatkih uničenja preverimo ujemanje podatkov iz odškodninskega zahtevka in dejanskega stanja (Marane, 2004). Poleg tega pa tudi lažje določimo cene zavarovanj z vključitvijo faktorjev, kot so poplavna ogroženost, indeks kriminala, požarna ogroženost in življenjski standard. Z grafnimi podatkovnimi bazami upravljamo tudi omrežje kompleksnih povezav aplikacij (npr. CRM), spletnih strani, različnih internih aplikacij, spletnih strežnikov, strežnikov podatkovnih baz ter strežnikov strojne opreme, kar je v zadovoljstvo sploh sistemskim administratorjem podjetja, ki lažje odstranjujejo in nadgrajujejo opremo, sledijo poslovnim zahtevam ter lažje nadzorujejo tveganje izpadov omrežja. Kot zadnje pa bi omenili izgradnjo priporočilnih sistemov na grafnih podatkovnih bazah, ki bi jih lahko zavarovalnice ponujale na njihovih spletnih straneh in bi strankam pomagali pri izbiri zavarovanja.

Zavarovalnice v tujini že preverjajo tudi družbena omrežja pri morebitnih zavarovalniških goljufijah. Družbena omrežja so postala sestavni del večine ljudi, ki na njih objavljajo razne fotografije, komentarje, ki lahko dajejo koristne podatke zavarovalnicam. Na primer, zdravstvena nezgoda, ki onemogoči delo, vendar zavarovanec objavlja na družbenih omrežjih slike raznih športnih aktivnosti. Ali pa zavarovanec, ki objavlja slike z dopusta po tem ko se mu pripeti zdravstvena nezgoda, pa sklene zavarovanje za tujino, pa čeprav imajo vse zavarovalnice v splošnih pogojih zapisano, da je zavarovanje potrebno skleniti pred odhodom v tujino. In s primeri v magistrskem delu smo že prikazali, da so družbena omrežja idealen primer za implementacijo v grafne podatkovne baze.

SKLEP

Z obravnavo teme v magistrskem delu sem želela pritegniti pozornost k grafnim podatkovnim bazam, področja, ki je celoten razvoj in napredek doživelo v zadnjem desetletju. Glede na vedno večji obseg podatkov, ki jih želimo shranjevati, je njihov pomen težko razumeti in mogoče je ravno v grafnih podatkovnih bazah priložnost, da obilico podatkov lažje obvladujemo, hitreje poizvedujemo in bolje razumemo. Dobljeni vzorci so zanimivi za katerokoli panogo, industrijo ali podjetje in jim lahko zagotovo prinesejo pomembne koristi.

V delu smo se osredotočili na grafno podatkovno bazo Neo4j, ki je bila prva na tem področju in z njo se je pojem grafnih podatkovnih baz sploh uveljavil. Zanimivo je dejstvo, da je Neo4j dejansko začela kot baza podatkov, ki je podpirala sistem za upravljanje vsebin (ang. content management system), ki je prvotno deloval na relacijski podatkovni bazi. Problem, ki je ustvaril danes poznano podatkovno bazo Neo4j, je ta, da je bilo potrebno za izvršitev nekaterih poizvedb v temu sistemu napisali kopico zelo zapletenih, nerazumljivih SQL poizvedb, ki so bile težke za vzdrževanje. Glavna težava je bila ta, da je bila uporabljena relacijska podatkovna baza za shranjevanje in poizvedbo podatkov, model razmišljanja o teh poizvedbah pa je bil o povezanih podatkih, o poteh med povezanimi vsebinami, metapodatki in oznakami. Posledično je prišlo do ogromne kognitivne vrzeli med načinom, kako so o podatkih razmišljali in kako so bili ti podatki dejansko shranjeni in vprašani.

Na primeru zavarovalnic smo se v magistrskem delu seznanili z možnostmi grafnih podatkovnih baz, ki bi služile zavarovalniškem poslovanju. Podrobneje smo si ogledali predvsem načine, na katere bi zavarovalnice lahko povečale delež razkritih goljufij oz. te tudi preprečevale.

LITERATURA IN VIRI

1. Agrawal, S. & Patel, A. (februar 2016). A Study on Graph Storage Database of NoSQL. *International Journal on Soft Computing, Artificial Intelligence and Applications (IJSCAI), 5*(1), 33–39.
2. Barker, C. (20. november 2017). Neo4j Looks beyond the graph database. *ZDNet*. Pridobljeno 20. januar 2019 iz https://www.zdnet.com/article/from-a-quarter-of-the-database-market-in-three-years-graph-is-taking-off/.
3. Batra, S. & Tyagi, C. (maj 2012). Comparative Analysis of Relational And Graph Databases. *International Journal of Soft Computing and Engineering (IJSCE), 2*(2), 509–512.
4. Coalition Against Insurance Fraud. (2018). *By the numbers: fraud statistics.* Pridobljeno 8. januar 2019 iz http://www.insurancefraud.org/statistics.htm
5. DB-engines. (november 2019). *DBMS popularity broken down by database model*. Pridobljeno 29. november 2019 iz https://db-engines.com/en/ranking_categories.
6. Domenjoud, M. & Vial, T. (12. julij 2012). Graph databases: an overview. *OCTO*. Pridobljeno 6. december 2018 iz https://blog.octo.com/en/graph-databases-an-overview/
7. Fernandes, D. & Bernardino, J. (2018). *Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4j and OrientDB.* Coimbra: Polytechnic of Coimbra - ISEC.
8. Green, A. (16. september 2019). GQL Is Now a Global Standards Project alongside SQL. *Neo4j*. Pridobljeno 16. januar 2020 iz https://neo4j.com/blog/gql-standard-query-language-property-graphs/
9. Gupta, S. (2015). *Building Web Applications with Python and Neo4j*. Birmingham: Packt Publishing.
10. IBM. (20. april 2018). Strike Back Against Identity Fraud. *IBM*. Pridobljeno 11. januar 2019 iz https://www.ibm.com/industries/insurance/resources/identify-fraud-paper/
11. Jordan, G. (2014). *Practical Neo4j.* New York: Apress.
12. Jorgé. (26. oktober 2015). Graph Query Languages: GraphQL, OpenCypher, Gremlin and SPARQL. *React, etc. Tech Stack*. Pridobljeno 6. januar 2019 iz https://react-etc.net/entry/graph-query-languages-graphql-opencypher-gremlin-and-sparql
13. Kemper, C. (2015). *Beginning Neo4j*. New York: Apress.
14. Klančar, J. (2013). *Grafne podatkovne baze in vizualizacija grafov.* Ljubljana: Fakulteta za računalništvo in informatiko.
15. Knez, N. (2012). *Primerjava relacijske in NoSQL podatkovne baze in opredelitev kriterijev za pomoč pri izbiri najprimernejše podatkovne baze.* Ljubljana: Fakulteta za računalništvo in informatiko.
16. Košir, K. (10. september 2017). Zavarovalniške goljufije v Sloveniji. *Svet kapitala*. Pridobljeno 20. november 2018 iz https://svetkapitala.delo.si/finance/zavarovalniske-goljufije-v-sloveniji-3291
17. Lal, M. (2015). *Neo4j Graph Data Modeling.* Birmingham: Packt Publishing.

18. Litan, A. (2011). *The Five Layers of Fraud Prevention and Using Them to Beat Malware.* Stamford: Gartner, Inc.
19. Liu, Y. (2014). *A survey of Persistent Graph Databases.* Kent: Kent State University.
20. Lonarčić, T., Vehovec, A., Kastelic, M., Drofenik, D., Divjak, S., Kavčič, A., Marolt, M., Privošnik, M., Podbrešček, M., Harej, J. & Florjančič, A. (2002). Načrtovanje in postavitev podatkovnih baz. *E-gradiva*. Pridobljeno 27. november 2018 iz http://www.egradiva.net/moduli/podatkovne_baze/01_datoteka.html
21. Mahmoud, A. (12. januar 2017). No more joins: An overview of Graph database query languages. *IBM developer*. Pridobljeno 6. januar 2019 iz https://developer.ibm.com/dwblog/2017/overview-graph-database-query-languages/
22. Malby, S., Mace, R., Holterhof, A., Brown, C., Kascherus, S. & Ignatuschtschenko, E. (2013). *Comprehensive Study on Cybercrime.* Dunaj: United Nations Office on Drugs and Crime.
23. Malej, B. (2016). *Delo z grafi v relacijskih in sodobnih nerelacijskih bazah.* Ljubljana: Fakulteta za računalništvo in informatiko.
24. Marane, A. (26. februar 2004). Using Geospatial Analysis To Investigate Insurance Fraud. *Understanding Link Analysis From The Analytic Workshop*. Pridobljeno 14. januar 2019 iz http://linkanalysisnow.com/2010/01/using-geospatial-analysis-to.html
25. Merkl Sasaki, B., Chao, J. & Howard, R. (2018). *Graph Databases for Beginners.* San Mateo: Neo4j, Inc.
26. Oracle. (brez datuma). *Introduction to Oracle Database*. Pridobljeno 17. november 2018 iz https://docs.oracle.com/database/121/CNCPT/intro.htm#CNCPT958
27. Plantikow, S. (14. maj 2018). Summary Chart of Cypher, PGQL, and G-Core. *THE GQL MANIFESTO*. Pridobljeno 9. december 2018 iz https://gql.today/comparing-cypher-pgql-and-g-core/
28. R Interview Bubble. (25. april 2018). *What is the difference between scaling horizontally vs scaling vertically? | SCALE UP VS SCALE OUT.* Pridobljeno 28. november 2018 iz https://interviewbubble.com/what-is-the-difference-between-scaling-horizontally-vs-scaling-vertically-scale-up-vs-scale-out/
29. Rathle, P. (18. maj 2018). It's Time for a Single Property Graph Query Language. *Database Zone*. Pridobljeno 8. december 2018 iz https://dzone.com/articles/its-time-for-a-single-property-graph-query-languag
30. Robinson, I., Webber, J. & Eifrem, E. (2015). *Graph Databases* (2. izd.). Sebastopol: O'Reilly Medai.
31. Rojko, M. (2014). *Integracija povpraševanj nerelacijskih podatkovnih baz in doseganje visoke razpoložljivosti v računalniškem oblaku.* Ljubljana: Fakulteta za računalništvo in informatiko.
32. Sadalage, P. J. & Fowler, M. (2013). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistance.* Saddle River: Addison-Wesley.
33. Sadowski, G. & Rathle, P. (2017). *Fraud Detection: Discovering Connections with Graph Databases.* San Mateo: Neo4j, Inc.
34. Sullivan, D. (2015). *NoSQL for Mere Mortals.* Hoboken: Addison-Wesley.

35. Tiao, S. (7. maj 2018). 5 Innovative ways to use Graph Analytics. *Oracle Big Data Blog*. Pridobljeno 10. januar 2018 iz https://blogs.oracle.com/bigdata/fraud-detection-use-cases-graph-technology
36. TigerGraph. (2018). *A modern graph query language.* Pridobljeno 7. januar 2019 iz https://www.tigergraph.com/gsql/.
37. Vaish, G. (2013). *Getting Started with NoSQL.* Birmingham: Packt Publishing.
38. Varangaonkar, A. (4. maj 2018). 2018 is the year of graph databases. Here's why. *Packt*. Pridobljeno 27. november 2018 iz https://hub.packtpub.com/2018-year-of-graph-databases/
39. Varčuj24. (29. maj 2016). *Sklenitev avtomobilskega zavarovanja preko interneta je vse bolj popularno.* Pridobljeno iz http://varcuj24.si/svetovalec/sklenitev-avtomobilskega-zavarovanja-preko-interneta-je-vse-/306
40. Villedieu, J. (22. marec 2018). Insurance Fraud Investigation Using Linkurious. *Linkurious*. Pridobljeno 10. januar 2019 iz https://linkurio.us/blog/using-linkurious-to-investigate-insurance-fraud/
41. Visual-Paradigm. (brez datuma). *What is Entity Relationship Diagram (ERD)?* Pridobljeno 27. november 2018 iz https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/.
42. Vukotic, A., Watt, N., Abedrabbo, T., Fox, D. & Partner, J. (2015). *Neo4j in Action.* Shelter Island: Manning Publications.
43. Wu, M. (18. avgust 2018). What Are the Criteria fo Differentiate Between Graph Databases? *Database Zone*. Pridobljeno 8. januar 2019 iz https://dzone.com/articles/what-are-the-criteria-to-differentiate-graph-datab
44. Zavarovalnica Triglav. (2015). *Ko videz vara.* Ljubljana: Zavarovalnica Triglav.

PRILOGE

Priloga 1: Cypher – kreiranje grafa s sintetičnimi identitetami

```
CREATE
  (zahtevek1:Zahtevek {datum:'22/11/2018',tip:'materialna skoda'}) ,
  (odvetnik:Odvetnik {ime:'Janez',priimek:'Kenda'}) ,
  (ocenjevalec:Ocenjevalec {ime:'Borut',priimek:'Rutar'}) ,
  (miha:Zavarovanec {ime:'Miha',priimek:'Premrl'}) ,
  (telefon_jaka:TelefonskaStevilka {stevilka:'031-000-000'}) ,
  (naslov_jaka:StalniNaslov {ulica:'Skrilje 196',postna_stevilka:'5263',posta:'Dobravlje'}) ,
  (emso1:EMSO {emso:'1003988500101'}) ,
  (email1:ElektronskiNaslov {email:'mihapremrl88@gmail.com'}) ,
  (zahtevek2:Zahtevek {datum:'9/10/2018',tip:'materialna skoda'}) ,
  (zahtevek3:Zahtevek {datum:'18/8/2018',tip:'materialna skoda'}) ,
  (tjas:Zavarovanec {ime:'Tjas',priimek:'Bratina'}) ,
  (email2:ElektronskiNaslov {email:'tjasb@gmail.com'}) ,
  (emso2:EMSO {emso:'2002981500555'}) ,
  (naslov_marko:StalniNaslov {ulica:'Col 38D',postna_stevilka:'5273',posta:'Col'}) ,
  (jan:Zavarovanec {ime:'Jan',priimek:'Strancar'}) ,
  (emso3:EMSO {emso:'1908992500232'}) ,
  (telefon_marko:TelefonskaStevilka {stevilka:'040-000-000'}) ,
  (email3:ElektronskiNaslov {email:'strancar.jan@gmail.com'}) ,
  (odvetnik)−[:JE_ODVETNIK]−>(zahtevek1),
  (ocenjevalec)–[:JE_OCENIL]−>(zahtevek1),
  (miha)−[:JE_VLOZIL]−>(zahtevek1),
  (miha)−[:IMA_TELEFON]−>(telefon_jaka),
  (miha)−[:IMA_NASLOV]−>(naslov_jaka),
  (miha)−[:IMA_EMSO]−>(emso1),
  (miha)−[:IMA_EMAIL]−>(email1),
  (odvetnik)−[:JE_ODVETNIK]−>(zahtevek2),
  (odvetnik)−[:JE_ODVETNIK]−>(zahtevek3),
  (ocenjevalec)−[:JE_OCENIL]−>(zahtevek2),
  (ocenjevalec)−[:JE_OCENIL]−>(zahtevek3),
  (tjas)−[:IMA_TELEFON]−>(telefon_jaka),
  (tjas)−[:IMA_EMAIL]−>(email2),
  (tjas)−[:IMA_EMSO]−>(emso2),
  (tjas)−[:IMA_NASLOV]−>(naslov_marko),
  (jan)−[:IMA_NASLOV]−>(naslov_marko),
  (jan)−[:IMA_EMSO]−>(emso3),
  (jan)−[:IMA_TELEFON]−>(telefon_marko),
  (jan)−[:IMA_EMAIL]−>(email3),
  (jan)−[:JE_VLOZIL]−>(zahtevek2),
  (jan)−[:JE_VLOZIL]−>(zahtevek3)
```

Priloga 2: Cypher – kreiranje grafa s prometnimi nesrečami

```
CREATE
 (nina:Oseba {ime:'Nina',priimek:'Skvarc',starost:'30', kraj_bivanja:'Nova Gorica'}),
 (nesreca1:Nesreca {datum:'31/5/2018',lokacija:'Ajdovscina'}),
 (nesreca2:Nesreca {datum:'6/6/2018',lokacija:'Razdrto'}),
 (nesreca3:Nesreca {datum:'28/6/2018',lokacija:'Vipava'}),
 (jaka:Oseba {ime:'Jaka',priimek:'Jamnik',starost:'21',kraj_bivanja:'Ajdovscina'}),
 (fordfocus:Avtomobil {znamka:'Ford',model:'Focus',letnik:'2008'}),
 (odvetnikKenda:Odvetnik {ime:'Janez',priimek:'Kenda'}),
 (marko:Oseba {ime:'Marko',priimek:'Peljhan',starost:'25'}),
 (opelastra:Avtomobil {znamka:'Opel',model:'Astra',letnik:'2002'}),
 (peugeot306:Avtomobil {znamka:'Peugeot',model:'306',letnik:'2005'}),
 (nissanalmera:Avtomobil {znamka:'Nissan',model:'Almera',letnik:'2005'}),
 (luka:Oseba {ime:'Luka',priimek:'Skvarc',starost:'35',kraj_bivanja:'Ajdovscina'}),
 (mia:Oseba {ime:'Mia',priimek:'Premrn',starost:'19'}),
 (volkswagenpolo:Avtomobil {znamka:'Volkswagen',model:'Polo',letnik:'2003'}),
 (zdravnikNovak:Zdravnik {ime:'Robert',priimek:'Novak'}),
 (bor:Oseba {ime:'Bor',priimek:'Lemut',starost:'29',kraj_bivanja:'Vipava'}),
 (fiatpanda:Avtomobil {znamka:'Fiat',model:'Panda'}),
 (fordfocus)−[:UDELEZEN]−>(nesreca1),
 (odvetnikKenda)−[:ODVETNIK]−>(nina),
 (odvetnikKenda)−[:ODVETNIK]−>(marko),
 (nina)−[:VOZNIK]−>(fordfocus),
 (marko)−[:SOPOTNIK]−>(fordfocus),
 (nina)−[:SOPOTNIK]−>(opelastra),
 (nina)−[:SOPOTNIK]−>(peugeot306),
 (marko)−[:VOZNIK]−>(opelastra),
 (marko)−[:SOPOTNIK]−>(nissanalmera),
 (odvetnikKenda)−[:ODVETNIK]−>(luka),
 (odvetnikKenda)−[:ODVETNIK]−>(jaka),
 (mia)−[:SOPOTNIK]−>(opelastra),
 (mia)−[:VOZNIK]−>(nissanalmera),
 (luka)−[:SOPOTNIK]−>(nissanalmera),
 (jaka)−[:SOPOTNIK]−>(fordfocus),
 (jaka)−[:VOZNIK]−>(peugeot306),
 (jaka)−[:SOPOTNIK]−>(volkswagenpolo),
 (luka)−[:SOPOTNIK]−>(volkswagenpolo),
 (zdravnikNovak)−[:ZDRAVNIK]−>(luka),
 (zdravnikNovak)−[:ZDRAVNIK]−>(jaka),
 (zdravnikNovak)−[:ZDRAVNIK]−>(bor),
 (mia)−[:SOPOTNIK]−>(fiatpanda),
```

(luka)−[:VOZNIK]−>(fiatpanda),
(bor)−[:SOPOTNIK]−>(fiatpanda),
(bor)−[:SOPOTNIK]−>(peugeot306),
(bor)−[:VOZNIK]−>(volkswagenpolo),
(opelastra)−[:UDELEZEN]−>(nesreca3),
(peugeot306)−[:UDELEZEN]−>(nesreca1),
(nissanalmera)−[:UDELEZEN]−>(nesreca2),
(peugeot306)−[:UDELEZEN]−>(nesreca2),
(volkswagenpolo)−[:UDELEZEN]−>(nesreca3)