Grafovski model za baze podataka

Kapov, Karmen

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO-MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Karmen Kapov

GRAFOVSKI MODEL ZA BAZE PODATAKA

Diplomski rad

Voditelj rada:

prof. dr. sc. Robert Manger

Zagreb, srpanj, 2019

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____ .

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Od početka upotrebe računalnog softvera, podaci koji obrađuju aplikacije postali su sve složeniji. Složenost podataka ne uključuje samo njihovu veličinu, već i njihovu međusobnu povezanost, njihovu stalno promjenjivu strukturu i istodobno pristup podacima.
Uz sve ove aspekte promjene podataka, poznato je da relacijske baze podataka, koje su dugo vremena de facto standard za pohranu podataka, nisu najprikladnije za rješavanje problema koji sadrže sve složenije podatke. Kao rezultat toga, stvoren je veliki broj novih tehnologija za pohranu podataka, a čiji je cilj rješavanje složenijih problema, a prepoznajemo ih pod imenom NoSQL.
Ono što može razlikovati modeliranje grafovske baze podataka od drugih tehnologija za pohranu podataka jest bliska povezanost između logičkih i fizičkih modela. Tehnike upravljanja relacijskim podacima zahtijevaju odstupanje od reprezentacije prirodnog jezika domene: prvo se transformira u logički model, a zatim u fizički model. Navedenim transformacijama uvodi se semantički nesklad između konceptualizacije svijeta i uvođenja baze podataka u taj model. Uz grafovske baze, nesklad se znatno smanjuje.

U prvom poglavlju opisat će se grafovi i grafovski model baze podataka te njegovi prednosti i nedostaci, a zatim opisat će se i relacijski model i njihova usporedba.
U drugom poglavlju detaljno se opisuje grafovski model baze podataka kroz sam opis programa najpoznatijeg javno dostupnog softverskog paketa za rad s grafovskim bazama podataka, Neo4j-a. Istaknuti će prednosti i nedostaci, opisati način rada i specifičan jezik za postavljanje upita, Cypher.
Zadnje poglavlje donosi primjer grafovske baze podataka u softveru Neo4j. Na konkretnoj bazo primjenjuju se jednostavni i složeni upiti, ali i specifični upiti koji su u relacijskom modelu teško izvedivi.

1. Grafovski i relacijski model baze podataka

1.1. Općenito o grafu:

Graf je uređen par bridova i čvorova, tj. sastoji se od čvorova i veza koje povezuju čvorove. Grafovi prikazuju entitete kao čvorove i na koji način se entiteti odnose u svijetu kao veze**.**
Usmjereni graf ili digraf je uređena trojka čvorova, veza i funkcije koja svakoj vezi pridužuje uređeni par *(u,v).* Kažemo da je *u* početak, a *v* kraj veze, a smjer ili orijentacija veze je od *u* prema *v,* i koristi se oznaka *a=(u,v)* ili *a=uv.*

Ova ekspresivna, najprisutnija prirodna struktura omogućuje modeliranje svih vrsta scenarija, od izrade svemirske rakete do sustava cesta, lanca opskrbe, povijesti bolesti i slično.

Primjena grafa

Prvi znanstveni rad o teoriji grafova smatra se rješenje povijesnog problema "sedam mostova Königsberga", koji je 1774. godine objavio švicarski matematičar Leohnard Euler.

Problem "sedam mostova" začetnik je današnje teorije grafova, a njegov model i algoritmi i danas se primjenjuju u raznim znanstvenim i inženjerskim disciplinama.

Postoji mnogo primjera u modernoj znanosti gdje se koristi graf: relativni položaji atoma i molekula u kemijskim spojevima, stabla evolucije u biologiji, zakoni atomskih veza u fizici, u analizi mreže (modeliranju prometa, telekomunikacijama i topologiji), računalna znanost, društvene mreže, popisi kontrola i mnogi drugi primjeri.

1.2. Općenito o grafovskom i relacijskom modelu baze podataka

Grafovska baza podataka je sustav upravljanja bazama podataka s metodama stvaranja, čitanja, ažuriranja i brisanja (CRUD)[1] koji koristi grafovski model za prikaz podataka. Grafovske baze podataka obično se izrađuju za korištenje u transakcijskim (OLTP)[2] sustavima. U skladu s tim, one su obično optimizirane za transakcijske izvedbe i osmišljene su s obzirom na transakcijski integritet i operativnu dostupnost.

Dva su svojstva grafovskih baza podataka koje treba uzeti u obzir pri samom istraživanju*:*

Spremanje podataka: Neke grafovske baze podataka koriste izvorno pohranjivanje grafova, koje su optimizirane i dizajnirane za spremanje i upravljanje. Međutim, baze pohranjuju podatke o grafu u relacijsku bazu podataka, objektnu orijentiranu bazu ili neku drugu bazu podataka opće namjene.

Mehanizam za obradu: Neke baze podataka koriste "susjedstvo čvorova" bez indeksa, što znači da su povezani čvorovi fizički usmjereni jedan na drugog u bazi podataka. Općenito, svaka baza podataka koja se iz perspektive korisnika ponaša kao grafovska baza podataka (tj. prikazuje grafovski model podataka s operacijama CRUD-a) kvalificira se kao grafovska baza podataka.

Veze su najvažniji dio grafovskog modela podataka. To nije slučaj u drugim sustavima za upravljanje bazama podataka, gdje se veze između entiteta uspostavljaju korištenjem stranih ključeva ili „out of the box"[3] obradom kao što je sažimanje podataka. Spajanjem

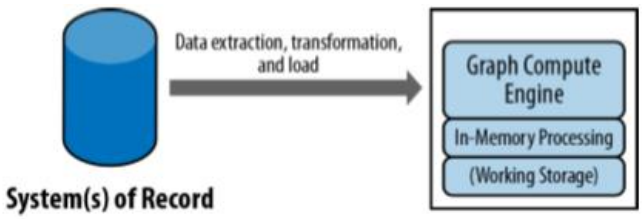
[1] U računalnom programiranju, kreiranje, čitanje, ažuriranje i brisanje (CRUD) su četiri osnovne funkcije trajnog pohranjivanja.
[2] OLTP sustavi (On-line Transaction Processing) daju izvorne podatke skladištima podataka, karakterizira veliki broj kratkih on-line transakcija (INSERT, UPDATE, DELETE).
[3] -"out-of-the-box" - neposredno upotrebljiv, nepotrebno je raditi ikakve prilagodbe.

čvorova i veza u povezane strukture, grafovske baze podataka omogućuju izgradnju proizvoljno sofisticiranih modela koji su bliski problemskoj domeni. Dobiveni modeli su jednostavniji i istodobno izražajniji od modela relacijskih baza podataka i drugih NoSQL baza podataka.

1.3. Alat za generiranje grafa

Graph compute engine je tehnologija koja omogućuje rad grafovskih algoritama s velikim skupovima podataka. *Graph compute engin*e alat osmišljen je za obavljanje procesa poput klasteriranja ili analiziranja prosječnog broja veza korisnika u društvenoj mreži.

Zbog naglaska na globalne upite, alati su optimizirani za skeniranje i obradu velikih količina informacija u skupinama, te su time slični tehnologijama kao što su rudarenje podacima i OLAP[4], koji su korišteni u relacijskom svijetu. Dok neki programi generiranja grafova sadrže prostor za pohranu grafova, drugi se strogo bave obradom podataka koji se unose iz vanjskog izvora, a zatim vraćaju rezultate za pohranu negdje drugdje.



Slika 1.1. Tipični raspored alata za generiranje grafova

Slika 1.1. prikazuje arhitekturu alata za generiranja grafova. Arhitektura uključuje sustav zapisa (SOR-*system of record*) baze podataka s OLTP svojstvima (kao što su MySQL, Oracle ili Neo4j), servisira zahtjeve i odgovara na upite aplikacije (korisnika) tijekom

[4] OLAP (Online Analytical processing) je vrsta obrade podataka koja daje brzi odgovor na višedimenzijske upite. OLAP je dio šire kategorije poslovnog izvještaja, koje također obuhvaća relacijsko izvješćivanje i rudarenje podataka.

izvođenja. Instrukcije kao Extract, Transform i Load (ETL) premještaju podatke iz sustava baze podataka u “Graph Compute Engine” za offline upite i analizu.

1.4. Prednosti i nedostaci grafovske baze podataka

1.4.1. Prednosti

Da grafovska baza podataka pruža snažnu, ali novu tehniku modeliranja podataka, samo po sebi ne pruža dovoljno opravdanje za zamjenu već dobro utemeljenje, dobro shvaćene podatkovne platforme. Osim toga, mora postojati neposredna i vrlo značajna praktična svrha. Motivira činjenica da se izvođenje upita za podatke implementirane u grafovskom modelu ubrzava za jedan ili više redova veličine. Uz to, grafovske baze podataka nude iznimno fleksibilan model podataka i način isporuke usklađen s današnjim agilnim načinima isporuke softvera.

Izvođenje: Jedan od uvjerljivijih razloga zašto koristiti grafovsku bazu podataka je povećanje performansi kada se radi o povezanim podacima u odnosu na relacijske i NoSQL baze podataka. Vremenski period za izvršenje upita kod grafovske baze podataka ostaje relativno konstantan bez obzira na broj upita intezivnog pridruživanja, čak i kad skup podataka raste.

Fleksibilnost: Programerima i arhitektima podataka bitno je povezati podatke, dopuštajući strukturi i shemi fleksibilnost za bolje razumijevanje problema, umjesto da se unaprijed nametne rješenje kad se u tom trenutku najmanje zna o stvarnom obliku i zamršenosti podataka.
Grafovi su prirodno aditivni, što znači da se u postojeću strukturu mogu dodati nove vrste veza, novi čvorovi, nove oznake i novi podgrafovi, bez ometanja postojećih upita i funkcionalnosti aplikacija. Takve značajke imaju općenito pozitivnu konotaciju za produktivnost i rizik projekta. Zbog fleksibilnosti modela grafova nije potrebno unaprijed iscrpno modelirati svoju domenu - model je prilagodljiv naknadnim promjenama poslovnih zahtjeva. Aditivna priroda grafova također znači da je moguće obavljanje manjih promjena, čime se smanjuje opterećenje održavanja i rizik.

Agilnost: Cilj je razvijati model podataka u korak s ostatkom aplikacije, koristeći tehnologiju usklađenu s isporukom softvera. Moderne grafovske baze podataka omogućuju razvoj i održavanje sustava. Grafovski model podataka bez sheme zajedno s aplikacijskim programskim sučeljem (API-*application programming interface*) grafovske baze podataka i upitnog jezika kojeg je moguće testirati, omogućuje razvoj aplikacije na kontrolirani način.
Upravo zato što su bez sheme, grafovske baze podataka nemaju onu vrstu upravljačkih mehanizama podataka orijentiranih na sheme koje su poznate u svijetu relacija, što omogućuje mnogo vidljivije i djelotvornije načine upravljanja.
To više nije kontroverzna praksa: za razliku od relacijske baze, razvoj grafovske baze podataka podržava današnje agilne metode razvoja softvera i metode razvoja koje je moguće testirati, omogućujući aplikacijama baziranim na grafovskoj bazi razvijati se u korak s promjenjivim poslovnim okruženjima.

1.4.2. Nedostaci grafovskog modela

Grafovske baze nisu učinkovite u obradi velikih količina transakcija i nisu dobre u obradi upita koji obuhvaćaju cijelu bazu podataka. Budući da nisu optimizirane za pohranjivanje i dohvaćanje poslovnih subjekata kao što su klijenti ili dobavljači, potrebno je kombinirati grafovsku bazu podataka s relacijskom ili NoSQL bazom podataka.

Samo korištenje grafovske baze podataka nije MDM[5] rješenje, odnosno ne pruža MDM funkcionalnost. Grafovska baza podataka samo je pohrana podataka i ne prikazuje korisničko sučelje koje se postavlja u poslovnom okruženju za postavljanje upita ili upravljanje vezama. Također, neće pružiti naprednu funkcionalnost upita.

Grafovske baze podataka ne stvaraju bolje veze. One jednostavno pružaju brzi dohvat podataka za povezane podatke. Poboljšano pretraživanje je sjajno, ali ne i ako se veza

[5] MDM (*Master Data Management*) je metoda kojom se definiraju i upravljaju kritični podaci organizacije kako bi se, uz integraciju podataka, osigurala jedinstvena referentna točka.

nije uspješno “dohvatila”.
Sve podatke grafovske baze podataka uglavnom je potrebno pohraniti na jednom poslužitelju. Neke grafovske baze podataka ograničene su na jedan čvor i ne mogu se skalirati iznad određene točke.

Grafovske baze podataka nisu optimizirane za analitičke upite velikog volumena tipične za skladištenje podataka. Na primjer, grafovska baza ne bi odgovorila na jednostavna, ali višestruka pitanja kao što su: "Tko su svi bili kupci s prihodom preko 100.000 dolara u dobi od 35 do 50 godina?".

Jednostavno rečeno, grafovska baze podataka omogućuje brzo pretraživanje podataka vezanih uz pojedinačni zapis (osoba, proizvod, mjesto). Međutim, nije u mogućnosti izvršiti upite za masovnu analitiku svih veza i zapisa.

Prema vodećim analitičarima u industriji, tvrtke izražavaju zabrinutost zbog sigurnosti tehnologija grafovskih baza otvorenog koda. Očekuje se da će ova rasprava postati prioritet u bliskoj budućnosti.

1.5. Usporedba relacijskih i grafovskih modeliranja

Modeliranje je aktivnost apstrahiranja motivirana određenom potrebom ili ciljem. Modeliranje je potrebno kako bi se specifični aspekti nesređene domene doveli u oblik u kojem se mogu strukturirati i manipulirati. Nemoguće je prezentirati situacije ili poslovno okruženje u potpunosti, zato se apstrahira i pojednostavljuje.
Da bi sagledali modeliranje grafova, opisat će se modeliranje domene pomoću relacijskih i grafovskih tehnika. Većini programera i stručnjaka za podatke poznati su RDBMS (sustavi za upravljanje relacijskim bazama podataka) i povezane tehnike modeliranja podataka. Kao rezultat toga, usporedba modela istaknut će neke sličnosti i mnoge razlike. Konkretno, u ovom poglavlju pokazat će se kako je lako prijeći s konceptualnog grafovskog modela na fizički grafovski model i kako grafovski model u maloj mjeri deformira ono što se pokušava prezentirati u odnosu na relacijski model.

1.5.1 Općenito o relacijskom modelu:

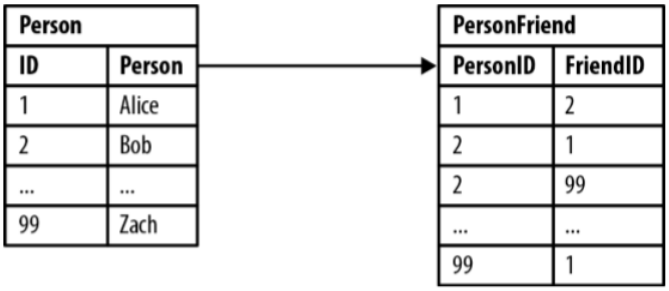
Relacijski model je bio teorijski zasnovan još krajem 60-ih godina 20.og stoljeća, u radovima Edgara Codda. Model se dugo pojavljivao samo u akademskim raspravama i knjigama. Prve realizacije na računalu su bile suviše spore i neefikasne. Zahvaljujući intezivnom istraživanju te napretku samih računala, efikasnost relacijskih baza postupno se poboljšavala. Sredinom 80-ih godina 20.stoljeća relacijski model postao je prevladavajući. Većina DBMS-ova (*Data Base Management System* – Sustav za upravljanje bazom podataka) koristi se baš tim modelom.

Relacijski model zahtjeva da se baza podataka sastoji od skupa pravokutnih tablica- tzv. relacija. Svaka relacija ima svoje ime po kojem se razlikuje od ostalih u istoj tablici. Jedan stupac relacije obično sadržava vrijednost jednog atributa (za entitet ili vezu) – zato se stupac poistovjećuje s atributom i obratno. Atribut ima svoje ime po kojem se razlikuje od ostalih u istoj relaciji. Dopušta se da dvije relacije imaju atribute s istim imenom, no tada se podrazumijeva da su to ustvari atributi s istim značenjem.

Desetljećima programeri pokušavaju prilagoditi povezane skupove podataka unutar relacijskih baza podataka. Kako su relacijske baze u početku bile dizajnirane kako bi kodificirale papirnate oblike i tablične strukture, muče se u pokušaju modeliranja ad hoc, izvanrednih odnosa koji se pojavljuju u stvarnom svijetu. Ironično, ali relacijske baze podataka loše prikazuju odnose.

U govoru relacijskih baza podataka postoje veze, ali samo kod modeliranja, kao sredstvo spajanja tablica. Često je potrebno razjasniti semantiku povezanih podataka koji povezuju entitete, i ocijeniti njihovu težinu. Relacijski odnosi ne čine ništa slično. Štoviše, kako se vanjski podaci množe, cijela struktura postaje opterećena složenija i neujednačena, relacijski model postane opterećen veliki tablicama, rijetko popunjenim redovima. Porast povezanosti u relacijskom svijetu povećava i broj spajanja tablica, što otežava razvoj postojeće baze podataka kao odgovor na promjenjive poslovne potrebe.

Radi boljeg razumijevanja “cijene” izvođenja povezujućih upita u relacijskoj bazi podataka, slijede neki jednostavni i neki složeniji upiti u domeni društvene mreže.



Person	
ID	Person
1	Alice
2	Bob
...	...
99	Zach

PersonFriend	
PersonID	FriendID
1	2
2	1
2	99
...	...
99	1

Slika 1.2. Prikaz modela prijatelja i prijatelji prijatelja u relacijskoj bazi podataka

Primjer 1.1 Slijedi jednostavan SQL upit:

Tko su Bobijevi prijatelji?“

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
  ON PersonFriend.FriendID = p1.ID
JOIN Person p2
  ON PersonFriend.PersonID = p2.ID
WHERE p2.Person = 'Bob'
```

Odgovor je: Alice i Zach. Upit nije “skup”, jer je ograničen broj redova koji se razmatraju pomoću filtera WHERE Person.person='Bob'.

Prijateljstvo nije uvijek simetrična relacija, pa upit glasi: Tko je prijatelj s Bobom?

Primjer 1.2.

```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
  ON PersonFriend.PersonID = p1.ID
JOIN Person p2
  ON PersonFriend.FriendID = p2.ID
WHERE p2.Person = 'Bob'
```

Odgovor na upit je: Alice. Nažalost, Zach ne smatra Boba prijateljem. Upit je još uvijek jednostavno implementirati, ali s gledišta baze podataka je "skuplje" jer baza podatka sada mora uzeti u obzir sve retke u tablici PersonFrend.
Moguće je dodati indeks, ali upit je i dalje posredan, što je skupo**.** Problem se dodatno komplicira na pitanje: "Tko su prijatelji prijatelja?"

Hijerarhije u SQL-u koriste rekurzivne spojeve, što upit čini sintaksno i računski složenijim, kao što je prikazano u idućem primjeru:

Primjer 1.3

```
SELECT p1.Person AS PERSON, p2.Person AS FRIEND_OF_FRIEND
FROM PersonFriend pf1 JOIN Person p1
  ON pf1.PersonID = p1.ID
JOIN PersonFriend pf2
  ON pf2.PersonID = pf1.FriendID
JOIN Person p2
  ON pf2.FriendID = p2.ID
WHERE p1.Person = 'Alice' AND pf2.FriendID <> p1.ID
```

Ovaj upit je složen iako se bavi samo Alicinim prijateljima i ne ulazi dublje u Alicinu društvenu mrežu. Problem postaje složeniji i skuplji što dublje se ulazi u mrežu. Iako je moguće dobiti odgovor na pitanje: “Tko su prijatelji prijatelja prijatelja?”, u razumnom vremenskom periodu, upiti koji se protežu na razini prijateljstva četiri, pet ili šest značajno se pogoršavaju zbog računalne i prostorne složenosti rekurzivnog spajanja tablica.

1.5.2. Relacijsko modeliranje

Glavni rezultat modeliranja trebala bi biti cijela shema baze, izgrađena u skladu s pravilima korištenog modela podataka te zapisana na način da je korišteni DBMS može razumjeti. Budući da je modeliranje prilično složena aktivnost, dijeli se u obično tri faze.

Početna faza, *konceptualno modeliranje*, slična je prvoj fazi mnogih drugih tehnika modeliranja podataka: analiziranje entiteta u domeni, njihovu povezanost i pravila koja upravljanju njihovim stanjima prijelaza. Zorno opisuje sadržaj baze i načine povezivanja podataka u njoj. Prikaz je jezgrovit, neformalan i pogodan ljudima za razumijevanje, no još je nedovoljno razrađen da bi izravna implementacija bila moguća.

Logičko modeliranje. Kao glavni rezultat druge faze nastaje logička shema koja je u slučaju relacijskog modela sastavljena od relacija (tablica), odnosno kreiranje dijagrama (entity-relationship dijagram). Ovom transformacijom konceptualnog modela u logički model unaprijeđuje se domena pomoću strožeg zapisa.

ER dijagrami odmah demonstriraju nedostatke relacijskog modela. ER dijagrami dopuštaju samo pojedinačne, neusmjerene, imenovane veze između entiteta. U tom smislu, relacijski model nije prikladan za domene u stvarnom svijetu gdje su veze između entiteta brojne i semantički bogate i raznolike.
Dakle, pri definiranju logičkog modela, mapira ga u tablice, koje se normaliziraju. *Normalizacija* je primjena posebnih pravila koja nastoji popraviti logičku strukturu samih relacija, tako da se bolje prilahodi inherentnim osobinama samih podataka i da se

eliminira zalihost podataka. Dakle, normalizacijom dobiven je model koji je relativno vjeran domeni. Međutim, normalizirana shema koja je teoretski prikladna za odgovaranje na bilo koju vrstu ad hoc pitanja, u praksi se mora dodatno prilagoditi i specijalizirati za specifične obrasce pristupa. Dakle, da bi relacijske baze podataka bile dovoljno dobre za potrebe aplikacija, potrebno je promijeniti model podataka kako bi odgovarao korisniku. Dakle, tehnika kontroliranog narušavanja normalnih formi u cilju poboljšanja performansi baze podataka naziva se *denormalizacija*.

Denormalizacija uključuje dupliciranje podataka (u nekim slučajevima) kako bi se dobila brža izvedba i bolje performanse upita. Denormalizacija obično nije trivijalan zadatak, a normalizirani model potrebno je pretvoriti u denormalizirani usklađen s karakteristikama temeljnog RDBMS-a. Pri tome svjesno se prihvaća znatna zalihost podataka, ali normalizati-denormalizirati dizajn nije jednokratan zadatak. Kako se poslovni zahtjevi mijenjaju ili se redovni zahtjevi razvijaju, sustavi se mijenjaju ne samo tijekom razvoja, već i tijekom radnog vijeka, pa se tako mijenjaju i sustavi i strukture podataka na kojima su izrađeni.
Tehnički mehanizam kojim se uvodi strukturna promjena u bazu podataka zove se *migracija*. Migracije pružaju strukturiran, pristup "korak po korak" u refaktoriranju baze podataka tako da se baze mogu razviti u skladu s promjenjivim potrebama aplikacija koje ih koriste. Refaktoriranje baze podataka može potrajati tjednima ili mjesecima. Osim toga, to je skup i rizičan proces.

Bez stručne pomoći i striktnog planiranja, migracija denormalizirane baze podataka predstavlja nekoliko rizika. Ako se migracije ne uspijevaju pohraniti, performanse mogu imati teškoće. Osim toga, ako namjerno duplicirani podaci ostanu "bez roditelja" nakon migracije, riskira se ugrožavanje integriteta podataka u cjelini.

Fizičko oblikovanje: Glavni rezultat treće faze oblikovanja je fizička shema cijele baze, dakle opis njene fizičke građe. U slučaju uporabe DBMS-a zasnovanog na jeziku SQL, pojam fizičkog oblikovanja treba shvatiti uvjetno. Fizička shema zapravo je niz SQL naredbi kojima se relacije iz logičke sheme realiziraju kao tablice. Pritom se dodaju pomoćne strukture i mehanizmi za postizanje traženih performansi te čuvanje integriteta i

sigurnosti podataka. Također se mogu uključiti i SQL naredbe kojima se pogledi za pojedine aplikacije realiziraju kao virtualne tablice izvedene iz stvarnih tablica.

Relacijske baze podataka - sa svojim krutim shemama i složenim karakteristikama modeliranja - nisu osobito dobar alat za brze promjene.

Ono što je potrebno je model koji je usko usklađen s domenom, ali da ne utječe na efikasnost performansi. Model koji podržava evoluciju, a istovremeno održava integritet podataka jer se brzo mijenja i raste. Taj model je grafovski model.

1.5.3. Grafovsko modeliranje

U ranim fazama analize, postupak je sličan relacijskom pristupu: pomoću "lo-fi"[6] metoda, kao što su crteži u fazi konceptualnog oblikovanja, opisuje se i dorađuje domena. Nakon toga, metodologije se razlikuju. Umjesto pretvaranja konceptualnog modela domene u tablice, obogatit će se model s ciljem stvaranja točnog prikaza dijelova domene relevantnih za ciljeve primjene. To jest, za svaki entitet u domeni, utvrđuje se njegova relevantna uloga kao oznaka, njegov atribute kao svojstvo, i njegova veza sa susjednim entitetima.

Modeliranje domene je potpuno izomorfno modeliranju grafova, što je jako korisno. Osiguravanjem ispravnosti modela domene implicitno se poboljšava model grafova, jer u grafovskoj bazi podataka ono što se skicira u fazi konceptualnog oblikovanja, obično je ono što se pohrani u bazu podataka. U izrazima na grafu, osigurava se da svaki čvor ima odgovarajuće oznake specifične za određeni profil i svojstva, tako da se mogu ispuniti svoje uloge za domenu usmjerenu na podatke. Ali također potrebno je čvor smjestiti u ispravan semantički kontekst; to se postiže stvaranjem imenovanih i usmjerenih (i često pripisanih) veza između čvorova kako bi se obuhvatili strukturni aspekti domene.

I logično, to je sve što je potrebno učiniti. Nema tablica, nema normalizacije, nema denormalizacije. Jednom kada se točno prikaže model domene, premještanje u bazu podataka je trivijalno, kao što će se vidjeti uskoro.

6 A low-fidelity prototype je brz i jednostavan opipljiv prikaz koncepta, protoka upotrebe ili informacijske strukture stvorene za dobivanje brzih povratnih informacija i poboljšanje proizvoda.

2. Softverska podrška za rad s grafovskim bazama podataka

2.1. Općenito o Neo4j

Neo4j je "open-source", NoSQL, izvorna grafovska baza podataka koja osigurava ACID-kompatibilan transakcijski backend[7] za aplikacije. Neo4j je sustav za upravljanje grafovskom bazom podataka koji je razvila korporacija Neo4j. Neovi osnivači, Emil Eifrem, Johan Svensson and Peter Neubauer, naišli su na probleme s performansama RDBMS-a i počeli su graditi prvi prototip Neo4j-a 2000.godine. Zatim, 2002.godine razvili prvu verziju Neo4j-a, a 2007. godine osnovana je švedska tvrtka Neo4j te je prvo izdanje pušteno pod GPL-licencom.
Verzija 1.0 izdana je u veljači 2010,a verzija 2.0 u prosincu 2013. godine. Neo4j verzija 3.0 koristi se od travnja 2016. godine.
Zadnje stabilno izdanje 3.5.4 izdano je 5. travnja 2019. Sa sjedištem u San Mateu, SAD, Neo4j ima urede u Švedskoj, Velikoj Britaniji i Njemačkoj. Neo4j je najpopularnija grafovska baza podataka prema rangiranju DB-Enginesa i 22. najpopularnija baza podataka.

Izvorni kod napisan je u programskom jeziku Java i Scala, te je dostupan besplatno na GitHubu ili kao "user-friendly" desktop aplikacija. Neo4j ima i izdanja *Community Edition* i *Enterprise Edition*. The Enterprise Edition sadrži sve što Community Edition, ali i dodatne zahtjeve tvrtki, kao što su sigurnosne kopije, klasteriranje i sposobnost oporavka od greške.

[7] Dio računalnog sustava ili aplikacije koji nije izravno dostupan korisniku, obično je odgovoran za spremanje i manipuliranje podacima

Na Slici 2.1. prikazan je logo Neo4j baze podataka. Mali bijeli krugovi predstavljaju detalje ili svojstva odnosa između tri glavna čvora, većih zelenih krugova. Pozadina plavih krugova grafa u globusu podsjeća na (graphs)-[:ARE]-(everywhere).

Slika 2.1. Logotip Neo4j baze podataka

Neo4j naziva se izvorna grafovska baza podataka jer učinkovito implementira grafovski model sve do razine pohrane. Dakle, podaci su pohranjeni točno onako kako su zaprimljeni izvorno, a baza podataka koristi strelice i kretanje kroz graf. Za razliku od neposredne obrade grafova, Neo4j također pruža potpune karakteristike baze podataka, uključujući usklađenost s ACID transakcijama, podršku klastera i nadogradnju na "runtime failover"[8] - što ga čini pogodnim za grafovski prikaz podataka.

Neke od sljedećih značajki čine Neo4j vrlo popularnim među programerima, projektantima administratorima baze podataka:

- Cypher, deklarativni upitni jezik sličan SQL-u, ali optimiziran za grafove. Danas se koriste druge baze podataka kao što su SAP HANA Graph i Redis graf putem openCypher projekta.
- Manji utrošak vremena u pretragama velikih grafovskih baza po dubini i širini. Omogućuje povećanje do milijardi čvorova na prosječnom hardveru.
- Fleksibilna shema grafovskih svojstava koja se može prilagoditi tijekom vremena, omogućujući da se dodaju nove veze kasnije kako bi se stvorili prečaci i ubrzali podaci o domeni kako se mijenjaju poslovne potrebe.

8 Runtime failover - osigurava da Security Servis Module i dalje pruža sigurnosne usluge čak i ako vanjske komponente na koje se oslanja (kao što je baza podataka za provjeru autentičnosti) postaju nedostupne tijekom izvođenja.

- Sučelja za popularne programske jezike, uključujući Java, JavaScript, .NET, Python i mnoge druge.

2.2. Neo4j u NoSQL

Iako se naziv NoSQL zadržao, daje pogrešan dojam da je protiv SQL-a kao koncepta. Bolji naziv bi vjerojatno bio ne-relacijske baze podataka, jer je relacijska/nerelacijska paradigma predmet rasprave, dok je SQL samo jezik koji se koristi u relacijskim tehnologijama. Pokret NoSQL je stvoren kao potvrda kako su nove tehnologije potrebne koje bi se nosile s promjenama podataka. Neo4j i grafovske baze podataka općenito su dio NoSQL pokreta, zajedno s mnogo drugih, više ili manje povezanih tehnologija pohrane.

Jedna od najuočljivijih karakteristika većine NoSQL baza podataka je što ne zahtijevaju definiciju sheme podataka, kao što je to slučaj s relacijskim bazama podataka. NoSQL baze podataka su zbog toga dinamičke, tip i broj atributa nekog entiteta je fleksibilan, odnosno moguće je po potrebi mijenjati bez da to utječe na druge podatke. Prednost ovog pristupa je u tome što je jednostavnije preslikati strukturu podataka u nekom programskom jeziku na sličnu strukturu podataka neke takve baze podataka, čime se pojednostavljuje interakcija između aplikacije i baze podataka.
Još jedna važna karakteristika NoSQL baza podataka je to da su one napravljene za distribuiran način upotrebe – dok su relacijske baze podataka napravljene za rad na jednom računalu, NoSQL sustavi napravljeni su tako da se više njih raspoređuje na veći broj računala gdje svaki radi sa svojom skupinom podataka. Time se značajno ubrzava prihvat upita i obrada podataka.

Postoji nekoliko vrsta NoSQL baza podataka, a izdvojene su 3 takve tehnologije: sustavi ključ-vrijednost, dokument-sustavi i stupčane baze podataka.
Sustavi ključ-vrijednost spadaju u one jednostavnije. Takvi sustavi podatke pohranjuju u obliku parova, gdje je prvi član ključ, a drugi vrijednost koja je pridružena tom ključu. Ovakve baze podataka imaju odlične performanse, ali nisu praktične za kompleksne

modele podataka.

Dokument-sustavi rade s podacima koji se sastoje od jedinstvenog identifikatora i vrijednosti, što se skupa naziva dokumentom. Vrijednost dokumenta može sadržavati i drugi dokument, tako da ovakvi sustavi podržavaju ugnježđene vrijednosti, što omogućava veliku fleksibilnost strukturiranja podataka.

Stupčane baze podataka podatke koji se nalaze u istom stupcu (ako ih se promatra kroz dvodimenzionalnu tablicu) fizički pohranjuju zajedno, što omogućava dobre performanse za određene vrste pretraživanja.

ACID-compilant database:

Cilj Neo4j-a je biti grafovska baza podataka s naglaskom na bazi podataka. To znači puna ACID podrška za Neo4j baze podataka:

- *Atomiziranost* (*Atomicity*) -Moguće je pokriti više operacija baze podataka u jednoj transakciji i provjeriti jesu li sve izvršene ispravno; ako jedna od operacija ne uspije, cijela će se transakcija neutralizirati
- *Dosljednost* (*Consistency*) - Kada se upisuju podaci u Neo4j bazu podataka, garantirana je sigurnost da svaki klijent koji pristupi bazi podataka nakon toga čita najnovije ažurirane podatke.
- *Izolacija* (*Isolation*) - Sigurno je da će operacije unutar jedne transakcije biti izolirane jedna od druge, tako da pisanje u jednoj transakciji neće utjecati na čitanje u drugoj transakciji.
- *Trajnost* (*Durability*) -Podaci upisani u Neo4j bit će zapisani na disk i dostupni nakon ponovnog pokretanja baze podataka ili pada poslužitelja.

No, postoji jedan ključni aspekt Neo4j koji nijedan od ostalih NoSQLbaza nema - jedan koji je vrlo važan kada je u pitanju usvajanje novih tehnologija pohrane u poslovnom svijetu: **transakcijsko ponašanje.**

Vrlo je važno naglasiti da je Neo4j transakcijska baza podataka. Sve što je već poznato za transakcije iz svijeta relacijskih baza odnosi se i na Neo4j. Podrška za transakcije jedan je od ključnih razlika između Neo4j i ostalih NoSQL baza podataka koje su ranije

spomenute, a koje ne podržavaju sva svojstva ACID-a. ACID transakcijska podrška osigurava sigurnost i praktičnost u radu s podacima u grafu.

Važno je imati garanciju ACID-a u Neo4j jer je uobičajeno mijenjati brojne entitete grafa (čvorove, veze i unose indeksa) unutar iste transakcije. Osim toga, lakše je razmišljati o ponašanju sustava koji su kompatibilni s ACID-om u usporedbi s konzistentnim sustavima, gdje ne postoji jamstvo o tome kad će promjene biti vidljive u klasteru, te u usporedbi s tehnologijama u kojima nisu moguće transakcije te djelomični rezultati većeg poslovanja mogu postati privremeno vidljivi istovremenim transakcijama. Aspekt trajnosti ACID transakcija također pruža visoki stupanj povjerenja da nakon izvršenja transakcije podaci neće biti izgubljeni.

Još jedna ključna razlika između relacijskih i Neo4j baza podataka: **upiti**. U Neo4j nema tablica i stupaca, niti postoje naredbe odabira i pridruživanja koje se temelje na SQL-u. Neo4j, kao i sve grafovske baze podataka, uzima matematički koncept iz teorije grafova i koristi ga kao učinkovit alat za upite po bazi podataka. Ovo je koncept pretraživanja grafa, a to je jedan od glavnih karakteristika koji Neo4j čini tako učinkovit za rad s velikim podacima u grafu.

Zašto kao bazu podataka odabrati grafovsku bazu podataka, točnije Neo4j? Kao što je ranije spomenuto, često je sasvim prirodno da ljudi pokušaju modelirati ili opisati određenu domenu problema koristeći grafovske strukture i koncepte, iako možda ne koriste grafovsku bazu podataka kao krajnju pohranu podataka. Odabir prave pohrane podataka može učiniti aplikaciju stabilnom, ali može ju odabirom pogrešne pohrane jednako lako i “srušiti”.

Korisnik zaključuje koja je semantička povezanost između entiteta i između modela podataka, ali same baze podataka nemaju mogućnost zaključivanja. Cilj je stvoriti mrežu iz nepovezanih podataka, a zatim procesuirati bilo koji upit koji se pojavljuje.
Dakle, traženi rezultat je kohezivna slika cjeline, uključujući veze između elemenata. U

svijetu grafova povezani podaci pohranjeni su kao povezani podaci. Tamo gdje postoje veze u domeni, postoje veze u podacima.

Često je potrebno kategorizirati čvorove u mrežama u skladu s ulogama koje imaju. Neki čvorovi mogu predstavljati korisnike, dok drugi predstavljaju narudžbe ili proizvode. U Neo4j koriste se oznake za predstavljanje uloga koje čvor ima na grafu. Budući da čvor može ispuniti nekoliko različitih uloga u grafu, Neo4j omogućuje dodavanje više oznaka čvoru. Koristeći oznake na ovaj način, moguće je grupirati čvorove. Možemo zatražiti bazu podataka, na primjer, da pronađe sve čvorove označene kao korisnik.
Gdje čvor predstavlja korisnika, dodana je oznaka korisnika; gdje predstavlja narudžbu koju je dodana oznaku narudžbe, i tako dalje.

Veze u grafu prirodno oblikuju putanju. Upit - ili tranzicija grafa određuje putanju. Zbog fundamentalno orijentirane prirode modela podataka, većina operacija temeljenih na putanji vrlo su usklađene s načinom prikaza podataka, što ih čini iznimno učinkovitim.

Uveden je grafovski model baze podataka. Za stvaranje sofisticiranih i semantički bogatih modela potrebno je sljedeće:

• Označeni graf svojstava sastoji se od čvorova, veza, svojstava i oznaka.

• Čvorovi sadrže svojstva. Čvorovi su kao dokumenti koji pohranjuju svojstva u obliku proizvoljnih parova ključ-vrijednost. U Neo4j, ključevi su nizovi, a vrijednosti su Java stringovi i primitivni tipovi podataka, te nizovi tih tipova.

• Čvorovi mogu biti označeni s jednom ili više oznaka. Oznake grupiraju čvorove i označavaju uloge koje imaju unutar skupa podataka.

• Veze povezuju čvorove i strukturiraju graf. Veza uvijek ima smjer, ime i početni čvor i krajnji čvor - nema nestabilnih veza. Zajedno, smjer i ime dodaju semantičku jasnoću u strukturiranju čvorova.

• Kao i čvorovi, veze također mogu imati svojstva. Mogućnost dodavanja svojstava u veze posebno je korisna za pružanje dodatnih metapodataka za algoritme grafa, dodavanje dodatne semantike vezama (uključujući kvalitetu i težinu) te za ograničavanje upita tijekom izvođenja.

2.3. Grafovska platforma Neo4j

Tvrtka Neo4j želi pružiti mogućnosti rješavanja mnogih različitih vrsta poslovnih i tehničkih potreba. Cilj je jednostavni proizvod koji odgovara različitim slučajevima uporabe. Bilo da se koriste grafovi za transakcije, analizu tržišta, optimizaciju poslovanja ili bilo što drugo, Neo4j nastoji pružiti besprijekoran proces integriranja alata s ostatkom postojećeg sustava.
Neke od mogućnosti u Neo4j grafovskoj platformi uključuju pomoć razvojnim programerima pri unosu podataka u graf, poslovnim analitičarima istraživanje podataka, i stručnjacima za podatke donošenje odluke na temelju rezultata analize. Bez obzira na ulogu unutar korisnikove organizacije, cilj je pojačati snagu grafa kako bi korisnik poboljšao poslovnu vrijednost i ispunio tehničke potrebe.

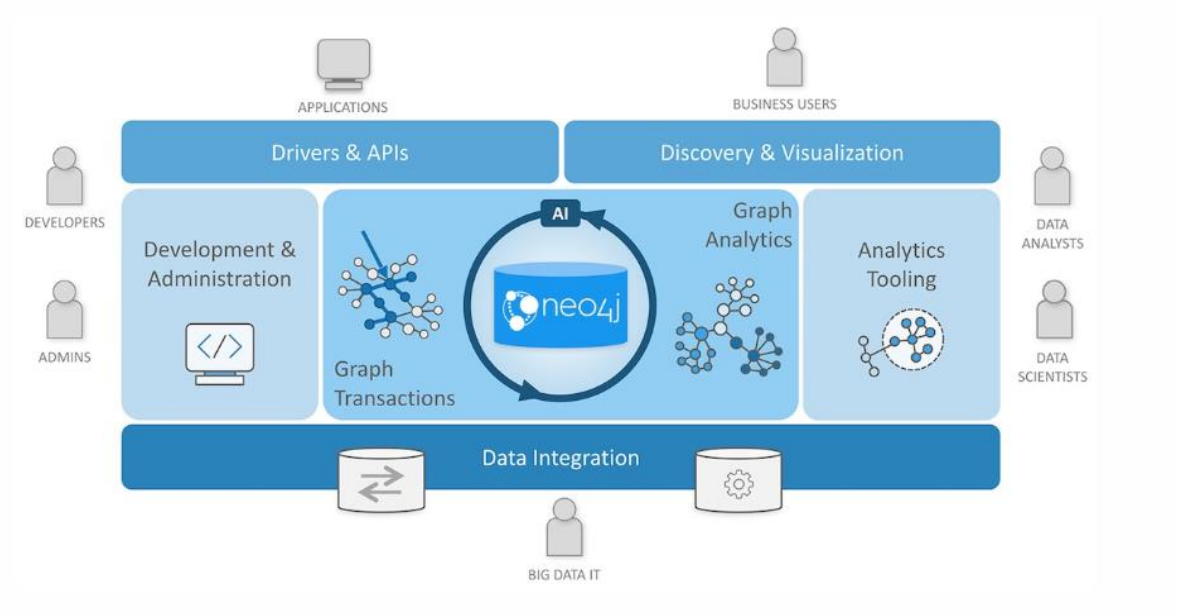
Grafovska platforma Neo4j izgrađena je oko izvorne baze podataka Neo4:

- **Neo4j izvorna grafovska baza podataka *(The Neo4j native graph database)*** podržava transakcijske aplikacije i analizu grafova
- **Analitika grafova (*Graph analytics*)** pomaže stručnjacima za podatke dobiti nove poglede na podatke
- **Integracija podataka (*Data integration*)** ubrzava razdvajanje tabličnih podataka i velikih podataka u grafove
- **The Cypher** je upitni jezik koji je most do velikih analitičkih alata za analizu podataka

- **Vizualizacija grafa (*Graph visualization and discovery*)** pomaže pojednostavniti prikaz veza
- **Poslovna arhitektura (*Enterprise architecture*)** podupire i podržava masivne podatke u grafu

U Neo4j postoji nekoliko načina interakcije podataka u grafu i njihovo korištenje. Od osnovnog proizvoda (grafovska baza podataka) do vizualizacije za poslovne korisnike, Neo4j pruža različite mogućnosti za poboljšanje poslovanja i pojednostavljenje podataka.

Komponente grafovske platforme Neo4j



Slika 2.2. Komponente grafovske platforme Neo4j

Svaki od elemenata koji će biti navedeni, osmišljen je kako bi ispunio poslovnu ili tehničku potrebu kao što je prikazano na Slici 2.2. I dalje se poboljšavaju različite perspektive iz kojih možemo vidjeti podatke, kao i mogućnosti samih proizvoda.

Neo4j Graph Database - osnovna grafovska baza podataka koja je izgrađena za pohranu i dohvaćanje povezanih podataka. Postoji Community Edition i Enterprise Edition.

Neo4j Desktop - aplikacija za upravljanje lokalnim instancama Neo4j. Besplatno preuzimanje uključuje licencu Neo4j Enterprise Edition.

Neo4j Browser - online sučelje za upit i pregled podataka u bazi podataka. Osnovne mogućnosti vizualizacije pomoću upitnog jezika Cypher.

Neo4j Bloom - alat za vizualizaciju poslovnim korisnika koji ne zahtijeva nikakve kodne ili programske vještine za pregled i analizu podataka. Neo4j Bloom pomaže tradicionalnim korisnicima Neo4j da komuniciraju sa svojim rukovoditeljima na jednostavan način. Preduvjeti: Neo4j Bloom zahtijeva pristup pokrenutoj instanci Neo4j Enterprise Edition, kao i Bloom-ovu licencu priloženu toj instanci poslužitelja.

Također koriste se različiti prošireni „library[9]“ i alati za razvojne programere koji se mogu dodati postojećim proizvodima kako bi se poboljšala funkcionalnost.

Grafovska platforma Neo4j je posebno optimizirana za mapiranje, analizu, pohranu i prolazak kroz mrežu povezanih podataka kako bi se otkrili nevidljivi konteksti i skriveni odnosi. Pomoću intuitivnog mapiranja podatkovnih točaka i veza između njih, Neo4j upravlja inteligentnim aplikacijama u stvarnom vremenu koje se bave današnjim najtežim izazovima poduzeća, uključujući:

- Artificial Intelligence (AI) and machine learning
- The Internet of Things (IoT)
- Real-time recommendations and personalization
- Master Data Management (MDM)
- Fraud detection
- Network and IT operations
- Identity and access management

[9] Programska biblioteka

Slijede neke novosti u izdanju Neo4j Database 3.5:

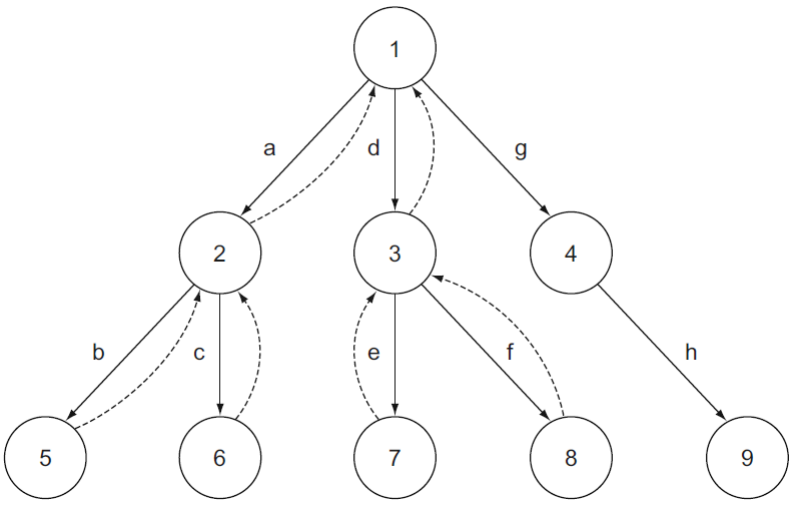
- Indeksiranje i pretraživanje cijelog teksta omogućuje brzo kretanje grafovskom bazom podatka pomoću pretraživanja teksta u oznakama, vrstama veza i vrijednostima svojstava
- Prošireno prirodno indeksiranje ubrzava unos podataka do 5x za sve vrste podataka (uključujući prostorne, vremenske i logičke vrijednosti)
- Smanjena su opterećenja kod pisanja zahvaljujući novom podsustavu za upravljanje transakcijama
- Poboljšanja preformansi prilikom sortiranja rezultata s indeksom ORDER BY upita u Cypheru

2.4. Prelazak po grafu

Pisanje učinkovitih prolaza po grafu je ključ za uspješno ispitivanje podataka grafova. Svaki put kada posjetitelj dođe do nekog čvora, mora donijeti odluku o tome koju će vezu pratiti i koji će čvor posjetiti sljedeći, ponavljajući postupak. Odabirom optimalne putanje kroz graf, pretraga po grafu se može brže završiti i koristiti manje memorije. U teoriji grafova postoje dva glavna algoritma pretrage, a oni se koriste za većinu prijelaza u Neo4j: pretraga po dubini (depth-first) i pretraga po širini (breadth-first).

2.4.1 Pretraga po dubini

Svaki put kada se posjeti čvor, potrebno je odlučiti koju vezu slijediti, ili koji sljedeći čvor prijeći. Algoritam za pretragu po dubini posjećuje prvo dijete čvora na kojem se trenutno nalazi, a nije posjećen. Ako su svi potomci već posjećeni, vraća se unatrag do prvog čvora čije dijete još nije posjećeno.



Slika 2.3. Pretraga po dubini

Koristeći prethodno navedena pravila, na kraju će hodati grafom sljedećim redoslijedom čvorova:

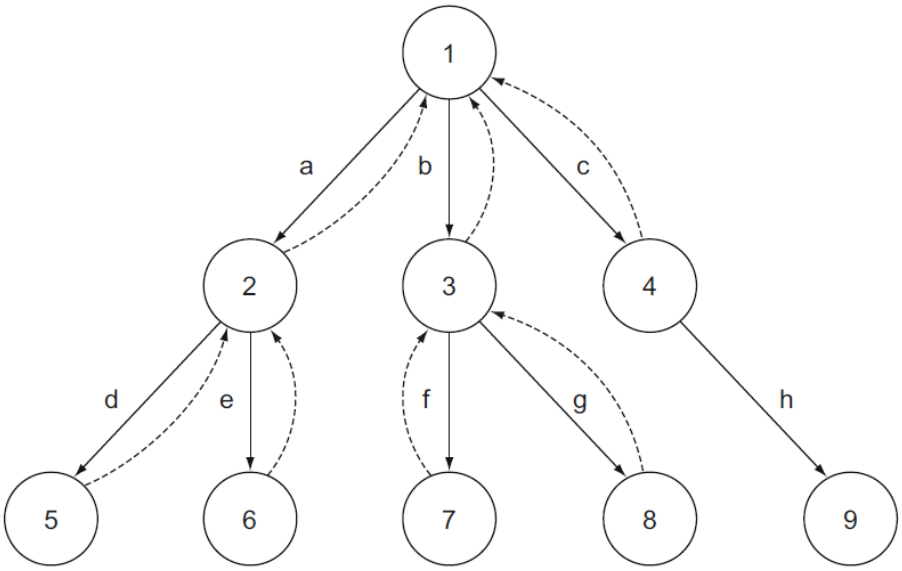
1 → 2 → 5 → 6 → 3 → 7 → 8 → 4 → 9

Na Slici 2.3, prikazan je redoslijed a, b, c, d, e, f, g, h kojim su posjećeni čvorovi tijekom prolaska.

Pretraga po dubini je podrazumijevana je strategija prelaska grana u Neo4j.Ako nije naveden način obilaska pri izradi opisa za prijelaz, koristit će se pretraga po dubini.

2.4.2 Pretraga po širini

Kao što ime sugerira, algoritam širine prvog pokuša ići što je moguće šire kroz graf, prije nego što posjeti čvorove udaljenije od početnog čvora. Kao dio algoritma pretrage po širini, obilazak će najprije posjetiti sve susjede trenutnog čvora prije nego što se preseli na njihovu djecu. Susjedni čvorovi su čvorovi na istoj udaljenosti od korijenskog čvora od čvora koji je trenutno posjećen.



Slika 2.4. Pretraga po širini

Redoslijed čvorova koji su posjećeni koristeći algoritam širine je sljedeći:

1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9

Prilikom pretrage po širini posjećeni su čvorovi bliže korijenskom čvoru, ostavljajući čvorove dalje od korijenskog čvora za kasnije.

Dakle, pretraga po dubini će preferirati pretraživanja s rješenjem u lijevom dijelu grafa. Pretraga po širini će biti brži kada je rezultat bliži početnom čvoru nego ako je to veća udaljenost od početnog čvora. Ove dvije karakteristike su vrlo važne pri dizajniranju grafa i pretrage po grafu.

Broj prolaza i posjećenih čvorova izravno je povezan s vremenom potrebnim za prelazak po grafu. Korištenje stvarnog vremena za mjerenje performansi ovisilo bi o dostupnim hardverskim resursima. Neo4j može posjećivati između nekoliko desetaka tisuća i milijuna čvorova u sekundi, ovisno o složenosti hardvera, memorije i prolaza.

Dakle, potrošnja memorije je još jedan aspekt koji treba uzeti u obzir prilikom odlučivanja o načinu prelaska po grafu. Može se usporediti prelazak po grafu s malim robotom koji skokovima kroz veze prelazi iz čvora u čvor. Robot će morati pamtiti koje je čvorove posjetio i koje čvorove tek treba posjetiti. Što je graf veći, potrebno je više memorije za pohranjivanje tog stanja. No, odabrani redoslijed upita utječe na potrebnu memoriju.

Kada se koristi pretragu po dubini, pokušava se ući duboko u graf što je dublje moguće. Čim se dođe do dna grafa (posjeti se čvor koji nema potomke a ranije nije posjećen), može se potpuno zaboraviti tu granu grafa - iz perspektive pretrage, ona se smatra potpunom. U svijetu Neo4j Java, to znači da se taj put može dereferencirati iz memorije i ostaviti ga za *garbage collection*. Budući da će neki od tih putova biti rano posjećeni tijekom prijelaza, memorija se dealocira čim krene pretraga.

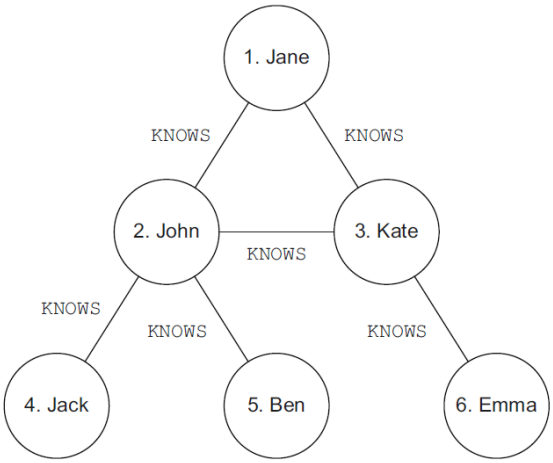
Pretraga po širini prvog stupnja pokušava ići što šire prije nego što krene pretraga po čvorovima na sljedećoj razini. Kao rezultat, za vrijeme obilaska moraju se pamtiti svi čvorovi koji su posjećeni i koje tek treba posjetiti. Veći i složeniji graf, potrebno je zapamtiti više čvorova, što rezultira većim potrebama za memorijom.

Da bi odabrali način pretrage, kao arhitekt i razvojni programer velika prednost je znati i razumjeti model domene na kojoj se radi. Na temelju uvida u domenu može se odrediti koja će se pretraga koristiti za svaki od prelaza, što može varirati od slučaja do slučaja na istom grafu.
Ako je rješenje blizu početnog čvora, pretraživanje po širini bi vjerojatno bilo bolje. Ali ako je graf vrlo gust (to jest, ima puno veza po čvoru), raspoređivanje po širini može koristiti previše memorije da bi bilo praktično.
Razumijevanje organiziranja i uvida u domenu problema pružit će dobru polaznu točku, ali da bi odredili najbolju opciju, treba eksperimentirati s različitim opcijama.

Ekspander je komponenta Neo4j Traversal API-ja koja je odgovorna za odlučivanje koje će se veze slijediti iz bilo kojeg čvora posjećenog tijekom prolaska. Osim odabira tipova veza i smjerova, ekspander je odgovoran za redoslijed kojim se prate veze. Neo4j nudi brojne implementacije "out of the box"[5]. Zadana implementacija, koja se najčešće koristi, je *StandardExpander*.

2.4.3. Dvosmjerni prolazi

Svi prolazi kroz graf kreću od jednog odabranog početnog čvora sve dok se ne pronađe rješenje ili dok se ne obiđe cijeli graf. Nema početnih točaka iz kojih je moguće početi prolaziti kroz graf. To je bilo ograničenje Neo4j Traversal API-a sve do Neo4j verzije 1.8, koja je uvela koncept dvosmjernih prijelaza.

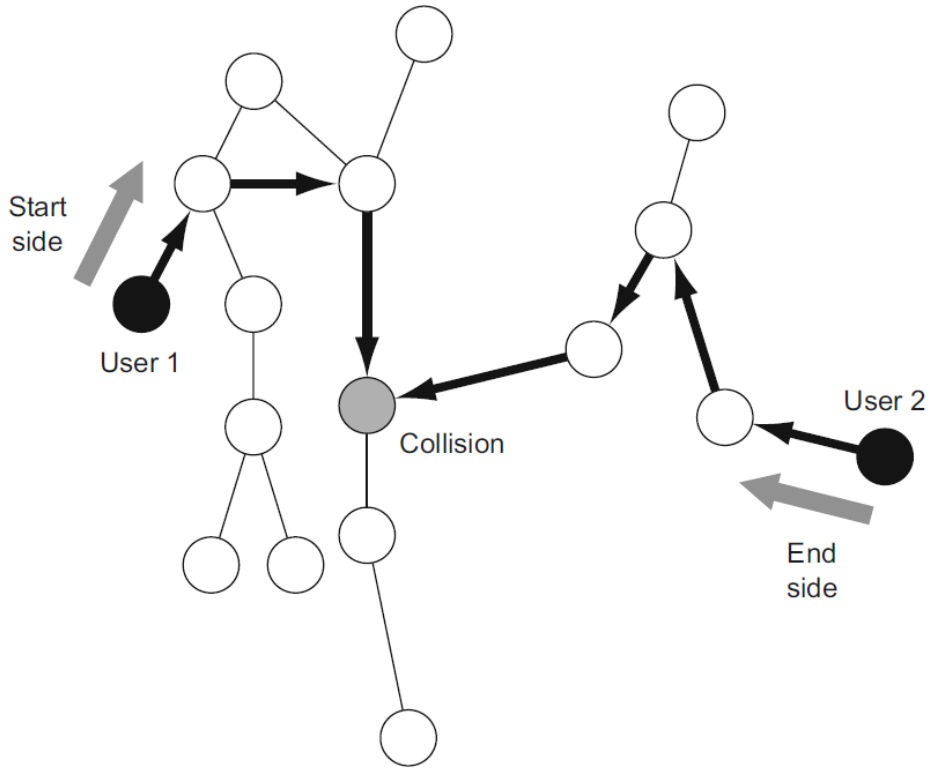


Slika 2.5. Jednostavan primjer društvene mreže

Na slici 2.5. prikazan je tipičan grafovski prikaz društvene mreže s tisućama korisnika. Korisnici koji se međusobno poznaju povezani su putem KNOWS veze. Jedan od tipičnih upita za prolaz u takvom grafu bio bi jesu li odabrana dva korisnika (korisnik 1 i korisnik 2) u istoj mreži, odnosno jesu li odabrani korisnici povezani i, ako da, kako su povezani.

Taj problem može se riješiti korištenjem standardnog API-a za Traversal: počevši od korisnika 1, slijedi se KNOWS veza; za svaki posjećeni čvor, provjeri se je li taj čvor korisnik 2. Ako jest, pronašla se veza; ako nije, nastavlja se dalje, dok se ne iscrpi cijeli graf. To je sasvim ispravno rješenje, ali ako je graf vrlo velik i gusto povezan, pronalaženje rješenja može biti prilično dugotrajno. Postavlja se pitanje je li moguće početi od korisnika 1 u smjeru korisnika 2, a istodobno pokrenuti još jednu skretanje od korisnika 2 u suprotnom smjeru. Kada se dva prijelaza sretnu (točka sudara), pronašla se

veza i može se odrediti cijeli put između korisnika. To je upravo ono što omogućuje dvosmjerno vođenje. Slika 2.6 ilustrira dvosmjerni prolaz u akciji.



Slika 2.6. Dvosmjerni prijelazi

Dvosmjerni prolazi uključuju dodatne komponente API-a za prolaz:

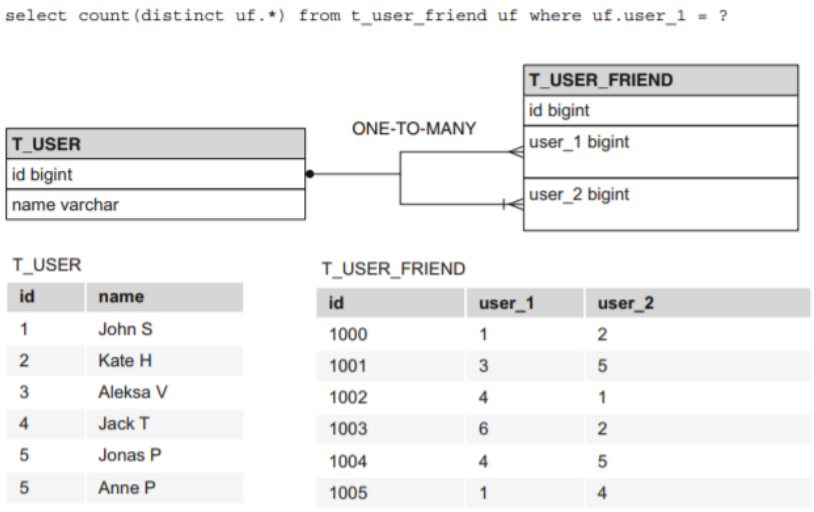
- Prolaz s početne strane - definicija za početnu (izlaznu) stranu dvosmjernih prelazaka. Opis obilaska kreira se pomoću standardnog API-a za prolaz u jednom smjeru.
- Prolaz s krajnje strane - definicija za prolaz krajnje (dolazne) strane dvosmjernog prijelaza.

Procjenitelj sudara - Svaki put kad se sudaraju putanje za početnu i krajnju stranu, postoji potencijalno rješenje prolaza. Procjenitelj sudara određuje treba li rješenje uključiti u krajnji rezultat prolaza.

- Bočni selektor - Ova komponenta određuje koliko brzo treba pomicati prolaze na svakoj strani. Naizmjenično se koristi izmjenični bočni selektor, odnosno naizmjenično se kreću između pomicanja početne i krajnje bočne putanje.

2.5. Usporedba relacijske baze i Neo4j-a u pogledu vremena potrebnog za izvršavanje upita

Radi usporedbe koristit se tradicionalna relacijska bazu podataka jer je to obično zajednički nazivnik većini ljudi kada je u pitanju razumijevanje mogućnosti pohrane podataka.



Slika 2.7. SQl dijagram tablica predstavlja t_user i t_user_friend

Da bi se prikazale performanse upita, više puta je pokrenut upit pretrage prijatelja na skupu podataka od 1.000 korisnika, ali je povećana razina pretraživanja sa svakim ponovnim upitom te su zabilježeni rezultati pretrage. Na skupu podataka od 1000 korisnika, gdje svaki korisnik ima prosječno 50 prijatelja, tablica t_user sadrži 1000 zapisa, dok tablica t_user_friend sadrži 1000 x 50 = 50000 zapisa. Na svakoj razini upit se proveo 10 puta, kako bi se uključila sva spremišta podataka koja bi mogla pomoći u izvedbi. Zabilježeno je najbrže vrijeme izvršenja za svaku razinu. Nije izvršeno dodatno podešavanje izvedbe baze podataka. Tablica 2.1 prikazuje rezultate istraživanja.

Depth	Execution time (seconds) for 1,000 users	Count result
2	0.028	~900
3	0.213	~999
4	10.273	~999
5	92.613	~999

Tablica 2.1. Vrijeme izvršavanja za višestruke upite spajanja pomoću MySQL baze podataka na skupu podataka od 1000 korisnika

Svi eksperimenti izvršeni su na Intelovom i7 računalu s 8 GB RAM-a.
S pretragom na razini 3, 4 i 5, vraća kao rezultat pretraživanja broja prijatelja broj približan 999. Zbog malog skupa podataka, bilo koji korisnik u bazi podataka povezan je sa svima ostalima.

MySQL daje prilično prihvatljive rezultate s upitima na razini 2 i 3. To nije neočekivano - operacije spajanja su uobičajene u svijetu relacija, tako da je većina poslužitelja baze podataka osmišljena i usklađena s tim. Korištenje indeksa baze podataka na relevantnim stupcima također je pomoglo relacijskoj bazi podataka da maksimizira učinkovitost tih pridruženih upita. Na razini 4 i 5, međutim, vidi se značajna degradacija performansi: upit na razini 4 traje više od 10 sekundi za izvršenje, dok na razini 5 izvršavanje traje predugo - više od minute i pol, iako se rezultat broja ne mijenja. Ovo pokazuje ograničenje MySQL-a pri modeliranju grafa podataka: duboki grafovi zahtijevaju višestruka pridruživanja, što relacijske baze podataka obično ne obrađuju najbolje.

U Tablici 2.2 prikazana su mjerenja performansi prijelaza po grafu koja sadrži iste podatke kao u prethodnoj MySQL bazi podataka (gdje je prolaz funkcionalno isti kao i kod upita koji su prethodno izvedeni u bazi podataka, pronalazeći prijatelje prijatelja do definirane razine). Ovo je također za skup podataka od 1.000 korisnika s prosječno 50 prijatelja po korisniku.

Depth	Execution time (seconds) for 1,000 users	Count result
2	0.04	~900
3	0.06	~999
4	0.07	~999
5	0.07	~999

Tablica 2.2. Vrijeme izvršenja za prelazak po grafu pomoću Neo4j na skupu podataka od 1.000 korisnika

Dakle, performanse Neo4j znatno su bolje za sve upite, osim za najjednostavnije. Samo u potrazi za prijateljima prijatelja (na razini 2) je MySQL izvedba usporediva s izvedbom Neo4j. Prelazak prijatelja na razini 3 je četiri puta brži od MySQL-a. Za obavljanje prijelaza na razini 4, rezultati su pet puta bolji. Rezultat Neo4j-a na razini 5 je 10 milijuna puta brži u usporedbi s MySQL upitom.

Drugi zaključak koji se može izvesti iz rezultata u Tablici 2.2 je da se performanse upita samo neznatno degradiraju s razinom prijelaza kada broj vraćenih čvorova ostaje isti. Performanse upita MySQL degradiraju se s razinom upita zbog Kartezijevih produkata koji se izvršavaju prije nego što se većina rezultata odbaci. Neo4j prati putanju posjećenih čvorova, tako da može preskočiti čvorove koji su prije posjećeni i stoga značajno poboljšati performanse.

Kako bi pronašli sve prijatelje na razini 5, MySQL će pet puta izvesti Kartezijev produkt na tablici t_user_friend, što će rezultirati 50.000^5 zapisa, od kojih su svi osim 1000 odbačeni. Neo4j će jednostavno posjetiti čvorove u bazi podataka, a kada više nema čvorova za posjet, zaustavit će se. To je razlog zašto Neo4j može održavati konstantne performanse sve dok je broj vraćenih čvorova isti, dok je pogoršanje performansi pri korištenju MySQL upita značajno. Prolasci po grafu značajno su bolji od ekvivalentnih MySQL upita (tisuće puta bolji s na razini 4 i 5). U isto vrijeme, učinkovitost prijelaza se

ne smanjuje dramatično s povećanjem razine - prelazak na razinu 5 je samo 0,03 sekunde sporije od obilaska na razini 2. Izvedba najsloženijih MySQL upita je više od 10.000 puta sporija od jednostavne.

Depth	Execution time (seconds) for 1 million users	Count result
2	0.016	~2,500
3	30.267	~125,000
4	1,543.505	~600,000
5	Not finished	—

Tablica 2.3. Vrijeme izvršavanja za višestruke upite pridruživanja pomoću MySQL baze podataka na skupu podataka od 1 milijuna korisnika

Uspoređujući ove rezultate s rezultatima MySQL-a za skup podataka od 1.000 korisnika, učinkovitost je upita na razini 2 ostala ista, što se može objasniti dizajnom MySQL-a za obradu tablica koji učinkovito spaja indekse. Upiti na razinama 3 i 4 pokazuju mnogo lošije rezultate, za najmanje dva reda veličine. SQL upit za sve prijatelje na razini 5 nije završio.

Ovi rezultati jasno pokazuju da je MySQL relacijska baza podataka optimizirana za pojedinačne upite spajanja, čak i na velikim skupovima podataka. Performansa višestrukih upita spajanja na velikim skupovima podataka značajno se degradira, do točke da neki upiti nisu ni izvršivi (na primjer, prijatelji na razini 5 za skup podataka od 1 milijuna korisnika).

Ponovljen je isti eksperiment s Neo4j bazom podataka. Milijun čvorova predstavljaju korisnike, a oko 50 milijuna veza pohranjeno je u Neo4j. Provedena su četiri prolaza po razinama ista kao u prethodnom primjeru i dobiveni rezultati su u Tablici 2.4.

Depth	Execution time (seconds) for 1 million users	Count result
2	0.01	~2,500
3	0.168	~110,000
4	1.359	~600,000
5	2.132	~800,000

Tablica 2.4. Vrijeme izvršenja za prelazak grafa pomoću Neo4j na skupu podataka od 1 milijuna korisnika

Dakle, povećanje podataka za tisuću puta nije značajno utjecalo na izvedbu Neo4j-a. Prelazak postaje sporiji što se povećava razina. Izvedba se linearno usporava s povećanjem veličine rezultata, a predvidiva je čak i s većim skupom podataka i stupnjom razine. Osim toga, barem je sto puta bolje od pripadne MySQL performanse.

Glavni razlog za predvidljivost Neo4j je lokalizirana priroda prijelaza grafova. Bez obzira na to koliko se čvorova i veza nalazi u grafu, prolaz će posjetiti samo one koji su spojeni na početni čvor, u skladu s pravilima kretanja. Relacijske operacije spajanja izračunavaju Kartezijev produkt prije odbacivanja nevažećih rezultata, što utječe na performanse eksponencijalno s rastom skupa podataka. Neo4 posjećuje samo čvorove koji su relevantni za prolaz, pa je u stanju održati predvidljive performanse bez obzira na ukupnu veličinu skupa podataka. Kako se povećava razina, više čvorova treba proći i time je sporija pretraga. No, ovo povećanje je linearno i još uvijek je neovisno o ukupnoj veličini grafa.

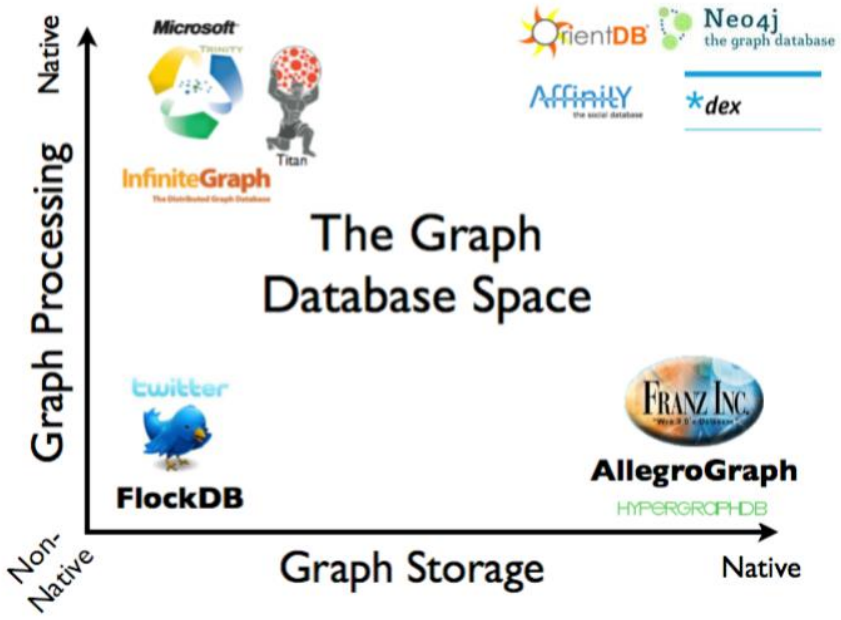
Slikoviti primjer bi bio mali lokalni nogometni stadion. Na pitanje koliko ljudi sjedi 15 koraka oko određenog gledatelja, brojit će se ljudi oko gledatelja što je brže moguće. Sada ako isto se isto pitanje postavi na nacionalnom stadionu, s mnogo više gledatelja, prebrojat će se otprilike isti broj ljudi i za to će trebati podjednako vrijeme, s obzirom da je gustoća ljudi na oba stadiona ista. Dakle, bez obzira na to koliko ljudi može stati na

stadion, prebrojat će se ljudi na predvidivoj brzini; cilj je znati samo broj ljudi koji se nalaze oko 15 koraka oko gledatelja, bez obzira na broj sjedala na drugom kraju stadiona. To je točno kako Neo4j radi u primjeru - posjećuje čvorove spojene na početni čvor, uz predvidljivu brzinu.

Čak i kada se broj čvorova u grafu povećava (s obzirom na sličnu gustoću čvora), performanse mogu ostati predvidljivo brze.

Primjenom nogometne analogije na upite relacijske baze podataka, izbrojali bi se svi ljudi na stadionu i uklonili oni koji nisu oko određenog gledatelja, što nije najučinkovitija strategija s obzirom na međusobnu povezanost podataka.

2.6. Primjena grafova



Slika 2.8. Prikaz nekih grafovskih baza podataka na tržištu danas, na temelju njihovih modela za pohranu i obradu

Primjer društvene mreže ilustrira kako se različite tehnologije bave povezanim podacima. U stvarnosti, nije potrebno pronaći takve udaljene "prijatelje". Ali ako se zamijeni bilo koja druga domena umjesto društvenih mreža, a vidjet će se prednosti izvedbe, modeliranja i održavanja. Bilo da se radi o upravljanju glazbenim ili podatkovnim centrima, bioinformatičkim ili nogometnim statistikama, mrežnim senzorima, grafovi pružaju snažan uvid u podatke. Budući da su grafovi prirodno višedimenzionalne strukture, onda je prilično jednostavno postaviti složenija pitanja o podacima.

Iz podataka i perspektive korisnika, jasno je da je grafovska baza podataka tehnologija uspješna u rješavanju složenih, varijabilno strukturiranih, gusto povezanih podataka, odnosno sofisticirani skupovi podataka su nezgrapni u bilo kojem obliku osim u grafu.

Još slučajeva kada je najpogodnije koristiti Neo4j bazu podataka:

- Otkrivanje i prevencija prevara – Velike tvrtke gube milijarde dolara godišnje zbog prevaranata koji se služe raznim sofisticiranim trikovima i prevarama kao što su krađa identiteta, lažno predstavljanje, prevare sa kreditnim karticama i pranje novca. Neo4j pomaže u njihovom otkrivanju.
- Praćenje mrežne infrastrukture
- Sistem preporuke u realnom vremenu – za online prodavaonicu,uz pomoć Neo4j baze podataka moguće je korisnicima preporučiti dodatne artikle iz ponude na osnovu onoga što pretražuju.
- Prava pristupa i kontrola identiteta

Danas, tisuće organizacija od startupova do tvrtki iz Fortune 500 koriste Neo4j za izgradnju novih i inovativnih aplikacija koje koriste veze u podacima kao što su preporuke, analiza utjecaja za mrežne i IT operacije, usmjeravanje u realnom vremenu za logistiku i poslovne aplikacije kao što je kao upravljanje matičnim podacima, upravljanje pristupom, upravljanje sadržajem, otkrivanje prijevara i rizik.

Neo4j danas koriste tisuće tvrtki i organizacija u gotovo svim industrijama, uključujući financijske usluge, vladu, energetiku, tehnologiju, maloprodaju i proizvodnju. Uspješna, aktivna zajednica koja okružuje tehnologiju i dalje rade na poboljšanju proizvoda i usluge za programere i tvrtke.

Slika 2.9. Dio tvrtki koje koriste Neo4j

2.7.Cypher

Do sada su svi modeli bili u obliku dijagrama. Dijagrami su izvrsni za opisivanje grafa u bilo kojem tehnološkom kontekstu, ali kada je riječ o korištenju baze podataka, potreban je neki drugi mehanizam za stvaranje, manipuliranje i pretraživanje podataka, odnosno potreban je upitni jezik.

Upitni jezik grafovskih baza podataka koji se koristi za većinu primjera je Cypher. Iako postoji nekoliko jezika za postavljanje upita na grafovima, Cypher je najčešće korišten, što ga čini de facto standardom. Također lako ga je naučiti i razumjeti, posebno za one koji poznaju SQL.

Tvrtka Neo4j odlučila je 2017.godine otvoriti izvorni kod jezika Cypher i učiniti najpopularniji upitnim jezikom grafova s ciljem da Cypher postane "SQL za grafove". Tako je stvoren openCypher projekt.

Osim što je najbolji način za interakciju s podacima u Neo4j, Cypher je i open source[10]. Podržan je od nekoliko tvrtki u industriji baza podataka i omogućuje implementatorima baza podataka i klijentima da slobodno koriste i pridonose razvoju openCypher jezika. Cypher je deklarativni upitni jezik koji koristi podudaranje uzoraka grafa kao glavni mehanizam za odabir podataka (za čitanje i mijenjanje).
Iako je trenutno specifičan za Neo4j, njegov način predstavljanja grafova kao dijagrama čini ga idealnim za programsko opisivanje grafova. Cypherom bolje se “ilustriraju” upiti po grafu i konstrukciji grafa. Cypher je vjerojatno najlakši upitni jezik, i odlična osnova za učenje o grafovima. Jednom kada se savlada Cypher, postaje vrlo lako učiti i druge upitne jezike za grafove.

Njegova jednostavnost korištenja proizlazi iz činjenice da je u skladu s načinom na koji su intuitivno opisani grafovi pomoću dijagrama. Cypher omogućuje korisniku (ili

[10] Open source softver je softver s izvornim kodom koji svatko može pregledati, modificirati i poboljšati.

aplikaciji koja djeluje u ime korisnika) da od baze podataka traži podatke koji odgovaraju određenom uzorku.

Cypher-ova sintaksa u stilu ASCII-arta[11] nudi čitljivi način usklađivanja uzoraka čvorova i veza unutar skupova podataka grafova.

Cypher upit fiksira na jedan ili više dijelova uzorka određene lokacije u grafu, a zatim prilagođava 'nefiksirane' dijelove kako bi pronašao lokalne podudarnosti. Točke fiksiranja u grafu na koje su vezani neki dijelovi uzorka koji Cypher određuje na temelju oznaka i predikata svojstava u upitu. U većini slučajeva Cypher koristi metainformaciju o postojećim indeksima, ograničenjima i predikatima kako bi automatski izračunao traženo.

2.7.1. Cypher za Apache Spark

Cypher za Apache Spark ™ (CAPS**)** je preliminarna verzija softvera poklonjena openCypher projektu i Apache Spark zajednici. CAPS nadopunjuje postojeće SQL upite, odnosno omogućuje izvršavanje Cypher upita na svojstvenim grafovima pohranjenim u Apache Spark klasteru na isti način na koji SparkSQL izvršava upite na tabličnim podacima.

Cypher za Apache Spark prvo ekstrahira skupove podataka iz baze podataka. Zatim dodaje te podatke grafovima konstruiranim u Neo4j, omogućujući spremanje ili snimanje rezultata kao (novih) grafova u stvarnom vremenu u Neo4j ili kao datoteke natrag u HDFS-u[12], ili oboje.

Osim toga, Cypher za library Apache Spark uvodi nove značajke u Cypher, uključujući:

Kompozicijski grafovski upiti(*Compositional Graph Queries*) omogućuje spremanje rezultata upita grafa u obliku grafa da kao rezultat daje tablični prikaz. Ta funkcionalnost omogućuje kompozicijske upite koji se mogu povezati u funkcijski lanan grafovskih algoritama, a pruža temelje za višestruke grafovske upite. Višestruka grafovska podrška

[11] ASCII art je tehnika grafičkog dizajna koju koriste računala za prezentaciju i sastoji se od slika sastavljenih od 95 znakova za ispis (od ukupno 128) koje definira ASCII. ASCII art odnosi se na slike koje su stvorene pomoću ASCII znakova. Slike se stvaraju ili pretvaranjem postojeće slike u ASCII znakove

[12] Hadoop Distributed File System -Distribuirani datotečni sustav

omogućuje upitima vraćanje prepoznatljivih grafova koji sadrže podatke za određenu domenu, odijeljene aplikacijom ili logičkom "bazom podataka".

Višestruko imenovani grafovi (*Multiple Named Graphs*) omogućuju Cypher-u da identificira određene skupove podataka grafova na kojima se želi raditi.

Sljedeće značajke su u tijeku s openCyper zajednicom, planiranom za buduće verzije Cypher implementacija, uključujući Neo4j poslužitelj i Spark izdanja.

Kontrola putanje *(Path Control*) pruža korisniku kontrolu nad odabirom putanje.

Regularni izrazi putanje (*Regular Path Expressions***)** primjenjuju sintaksu regularnog izraza u sažetom obliku složenih obrazaca putanje za izvod podgrafova. Primjenjuje učinkovit jezik za izražavanje staze koji se temelji na vodećem GXPath istraživačkom jeziku.

2.7.2. Sintaksa Cyphera

Sintaksa Cypher sastoji se od četiri različita dijela, od kojih svaki ima specifičnu ulogu:

- START - Određuje jednu ili više eksplicitnih početnih točaka (čvorova ili veza) na grafu.
- MATCH - podudara se s grafovskim uzorcima, omogućujući pronalazak podgrafova traženih podataka
- WHERE - filtrira podatke na temelju nekih kriterija
- RETURN - određuje koji čvorovi, veze i svojstva u podudarnim podacima trebaju biti vraćeni klijentu

Najjednostavniji upiti sastoje se od klauzule MATCH i RETURN.

MATCH klauzula

MATCH klauzula dio je većine Cypher upita. Koristeći ASCII znakove za predstavljanje čvorova i veza, “crtaju se”" podaci koji su u području interesa. Čvorovi se pišu u vitičastim zagradama, a veze pomoću parova crtica s znakovima većih ili manjih od (--> i

<--). Znakovi „<“ i „>“ označavaju smjer veze. Između crtica, postavljenih uglatim zagradama i prefiksom dvotočke, unosi se ime veze. Oznake čvorova su u prefiksu dvotočke. Parovi ključ-vrijednosti čvora (i veze) se zatim određuju u vitičastim zagradama (slično kao Javascript objekt).

Ostale naredbe:

CREATE i CREATE UNIQUE -stvaraju čvorove i veze.

MERGE - osigurava da dobiveni uzorak postoji u grafu, bilo ponovnom upotrebom postojećih čvorova i veze koji odgovaraju isporučenim predikatima, ili stvaranjem novih čvorova i veza.

DELETE - uklanja čvorove, veze i svojstva.

SET - postavlja vrijednosti svojstava.

FOREACH - ažurira za svaki element na popisu.

UNION - spaja rezultate iz dva ili više upita.

WITH - povezuje sljedeće dijelove upita i prosljeđuje rezultate od jednog do drugog.

Dakle, izrazi su slični SQL upitima. Namjera je da Cypher bude dovoljno “poznat” kako bi se brže naučio. Istovremeno, dovoljno je drugačije da se može uočiti da se koristi grafovima, a ne relacijama.

Postupak postavljanja upita:
Najprije je potrebno pronaći početni čvor pomoću ID-a čvora, koji se može postići u Cypher-u koristeći ključne riječi START. Bitno je zabilježiti ime početnog čvora (npr. korisnik) tako da ga kasnije moguće koristit u istom Cypher upitu. START se odbacuje u korist navođenja fiksiranih točaka u MATCH klauzuli.

Zatim, potrebno je postaviti uzorak potreban za postizanje rezultata. Uzorak je opis svih željenih podgrafova, a sastoji se od više čvorova povezanih vezama. Tipični grafovski obrasci sastoje se od dva čvora povezana vezom, koja je predstavljena u obliku () - [] - (). Čvorovi se navode pomoću zagrada (), a veze se navode u uglatim zagradama [], pomoću parova crtica s znakovima većih ili manjih od (--> i <--). Znakovi < i > označavaju smjer veze.
Čvor u klauzuli START (npr. korisnik) koristi se za fiksiranje lijeve strane uzorka. Cijeli uzorak ima dva poznata elementa (lijevi čvor i veze), koji se koriste za pronalazak svih čvorova koji odgovaraju desnoj strani (koji je povezan točno određenom vezom). Konačno, koristi se ključna riječ RETURN za vraćanje rezultata (opet, upućivanjem na čvor s imenom koje je ranije unešeno). Točka-zarez označava kraja upita Cypher.

Kada se pokrene Cypher upit, vidjet će se potpuno isti rezultati kao s odgovarajućim Java API rješenjem, ali s mnogo jednostavnijom sintaksom upita. Cypher je mnogo jednostavniji za čitanje i razumijevanje programerima, operativnom osoblju i svakom tko ima dovoljno znanja o domeni.

Cypher za SQL Developer

Cypher se temeljio na snazi i mogućnostima SQL-a,standarda za pretraživanje relacijskih baza podataka. Međutim, Cypher je dizajniran i optimiziran posebno za podatke o grafu i prijelaze. Dakle, svaki od jezika optimiziran je za svoj način pretrage i svaki od njih ima svoje prednosti i ciljeve.

Postoji nekoliko načina na kako izvršiti Cypher upite. Neo4j se isporučuje s nekoliko alata koji podržavaju Cypher izvršenja, a Cypher se također može izvršiti Java kodom, poput SQL-a. Tablica 2.5 prikazuje opcije izvršavanja Cypher-a.

Tool	Description
Neo4j Shell	Command-line tool
Neo4j Web Admin Console	Web-based interface
Java	Programmatically
REST	Over HTTP using REST interface

Tablica 2.5. Alati i tehnike za izvršavanje Cypher upita

3. Primjer grafovske baze podataka

3.1. Grafovska baza podataka - LinkedIn

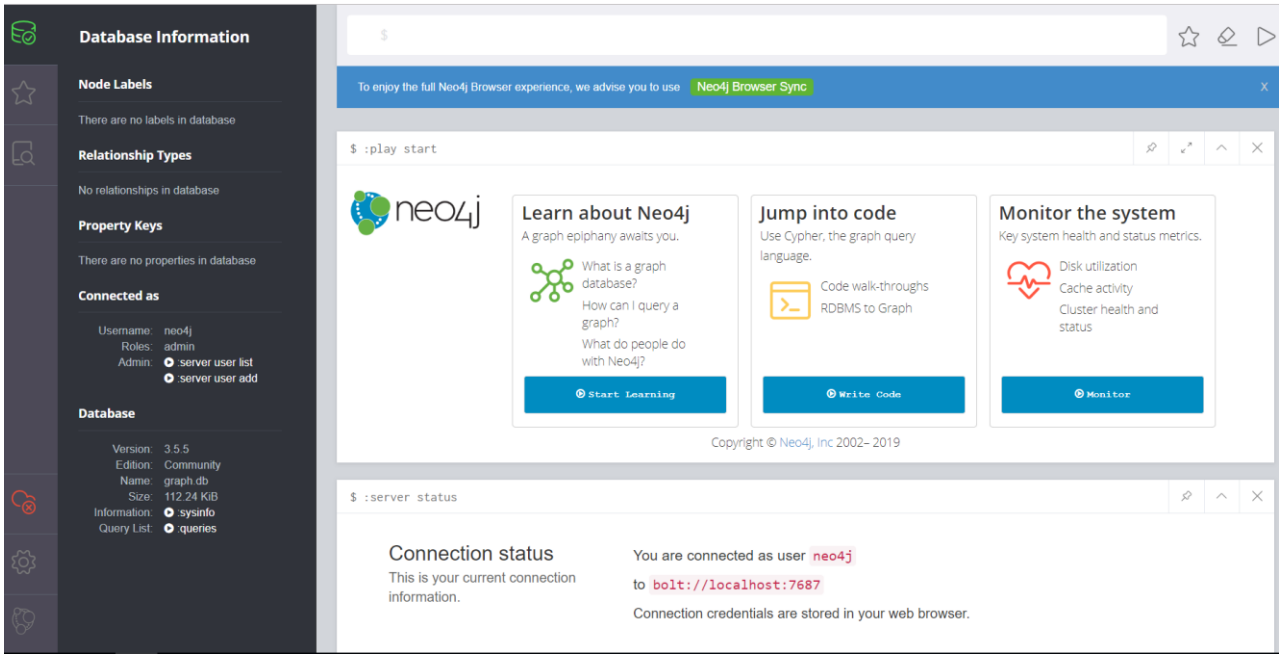
U društvenoj mreži kao što je LinkedIn, kao i u mnogim slučajevima povezanih podataka u stvarnom svijetu, veze između entiteta ne pokazuju unformnost u cijeloj domeni, domena je varijabilno strukturirana. Društvena mreža je popularan primjer gusto povezane, varijabilno strukturirane mreže, koja se opire "ukalupiranju" shema jednakih veličina. Fleksibilnost grafovskog modela omogućila je dodavanje novih čvorova i novih veza bez ugrožavanja postojeće mreže ili migracije podataka, odnosno izvorni podaci i njihova namjena ostaju netaknuti. Dakle, grafovske baze podataka pogodne su kao baza podataka u društvenim mrežama.

Upute za instalaciju:
Instalacija se razlikuje ovisno da li se instaliraju na Windowsu, Linuxu, Mac OS-u.
Slijedi primjer u Windowsu.
Ako već nije instaliran, potrebno je preuzeti OpenJDK 8 or Oracle Java 8, preporučen za Neo4j 3.0.x Version 7 za verzije prije 2.3.0.

Upute:

- Preuzeti zip dokument "neo4j-community-3.5.5-windows"
- Smjestiti "raspakiranu" datoteku u trajnu datoteku home na serveru, npr. **D:\neo4j**. Direktorij najviše razine naziva se NEO4J_HOME.
    - Da bi se pokrenuo Neo4j kao konzolna aplikaciju, upotrijebiti: **<NEO4J_HOME>\bin\neo4j console**
    - Da bi se instalirao Neo4j kao usluga: **<NEO4J_HOME>\bin\neo4j install-service**.

Nakon instalacije i pokretanja servisa, na portu 7474 dostupno je pregledno korisničko sučelje koje omogućuje rukovanje bazom i postavkama sustava. Nakon otvaranja adrese www.localhost:7474/ u pregledniku, korisnik tada mora definirati novu lozinku koju će koristiti ubuduće. Na Slici 3.1. prikazan je Neo4j Graph Database Platform**.**



Slika 3.1. Sučelje Neo4j-a

U ovom primjeru grafovskog modela podataka definirana su četiri tipa čvora: **korisnik** (User), **tvrtka** (Company), **fakultet** (Faculty) i **interes** (Interest). Korisnik ima svojstva: korisničko ime (*username*) i poziciju radnog mjesta (*position*), tvrtka ime (*name*) i grad (*city*), fakultet ime (*faculty_name*) i grad (city) i interes temu (*topic*).
Veze s čvorom korisnik: RADI (WORKS_IN) koja ga povezuje s tvrtkom, STUDIRAO (EDUCATED_IN) povezuje s čvorom fakultet, ZAINTERESIRAN_ZA (INTERESTED_IN) povezuje s interesom, a veza PRATI (FOLLOWS) povezuje korisnike. Dakle, tip čvora korisnik je povezan s više tipova čvorova, a ostali tipovi čvorova međusobno nisu povezani.

3.2. Unos, ažuriranje i brisanje podataka

U primjeru koji slijedi prikazan je unos podataka u bazu, točnije, unos tvrtki i njihovih svojstava. Kao što je navedeno, nazivi (avl, bank i ericsson) ispred dvotočke nazivi su veza koje se koriste, što uvelike olakšava povezivanje s prethodnim čvorovima u daljnjim upitima.

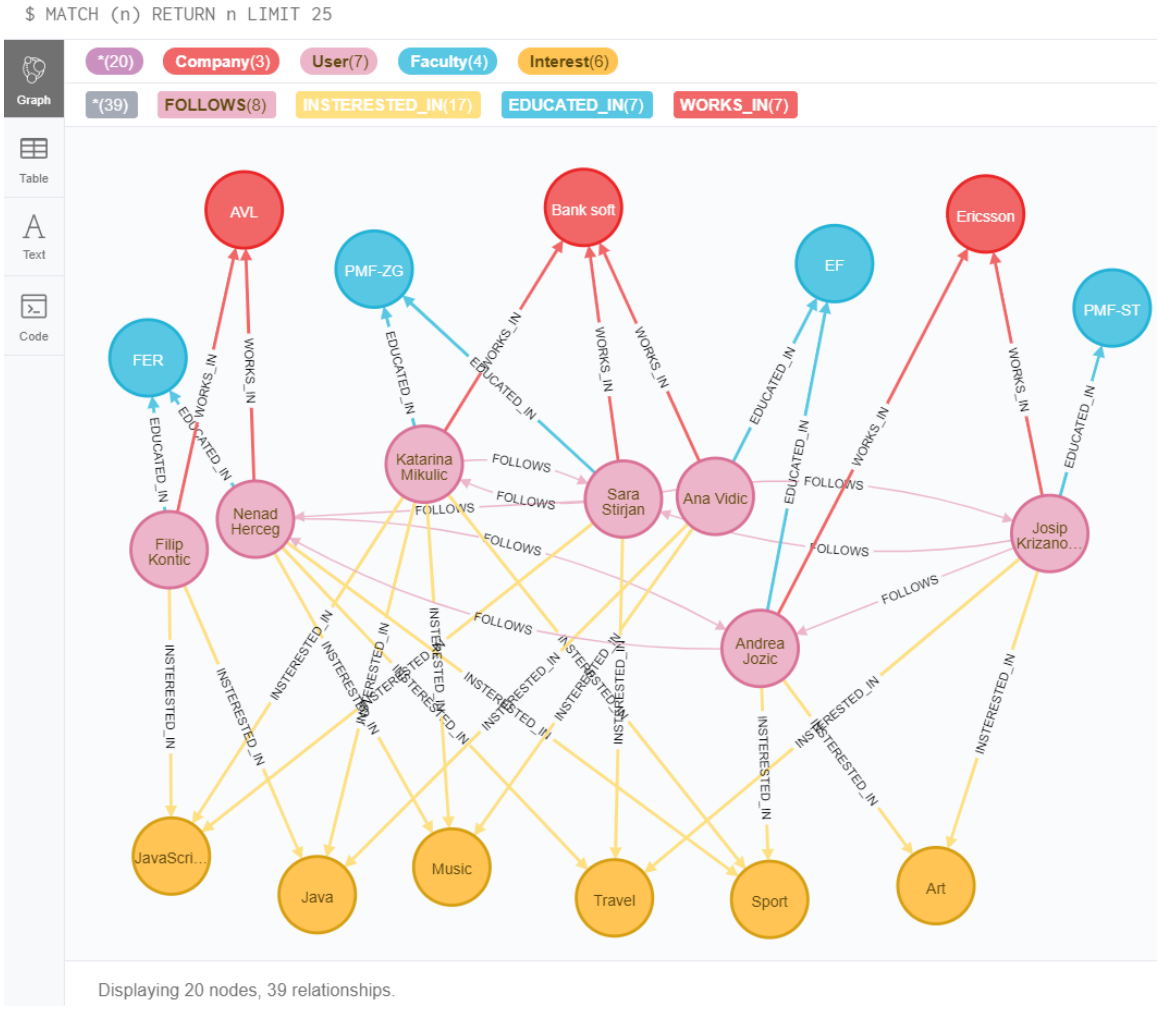
Unos podataka i upiti unose se preko sučelja prikazanom na Slici 3.1. (bijeli prozor na vrhu s desne strane).

Unos podataka moguć je i učitavanjem csv datoteke, posebice velikih količina podataka, što je puno preglednije. Naredbom “LOAD CSV” u Cypheru omogućuje nam unos putanje do datoteke, različita ograničenja i Cypher izraze za način na koji se žele modelirati tablični podaci u grafu.

Prilikom unosa u bazu podataka svaki čvor dobiva svoj jedinstveni broj (primjerice, 11), a klikom na taj čvor mogu se detaljnije vidjeti njegovi atributi (npr. čvor s oznakom tvrtka, a atributi su ime i grad).

```
CREATE (avl:Company {name:"AVL", city: "Zagreb"}),
       (bank:Company {name:"Bank soft", city: "Zagreb"}),
       (ericsson: Company {name:"Ericsson",city:"Split"})
```

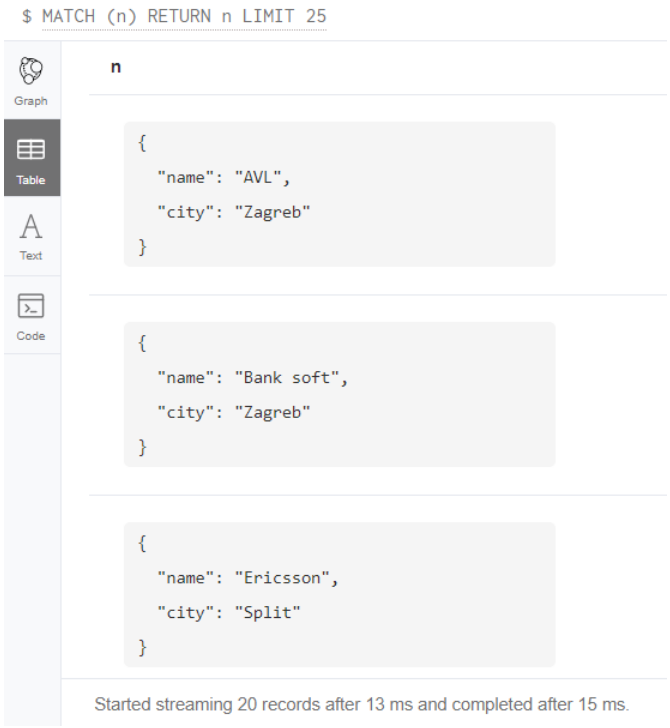
Na Slici 3.1. baza podataka prikazana je u obliku grafa nakon unesenih podataka u Neo4j.



Slika 3.2. Grafovski model cijele baze podataka

Dakle, u bazi LinkedIn postoji 4 grupe čvorova (*Node Labels*): User, Company, Faculty, Interest, i 4 tipa veze(*Relationship Types*): WORKS_IN, EDUCATED_IN, INTERESTED_IN, koje su jednosmjerne, i FOLLOWS koja je dvosmjerna. Sve su obojane različitim bojama radi lakšeg raspoznavanja međusobnih veza i grupa čvorova, što uvelike olakšava preglednost grafa. Boje je moguće mijenjati proizvoljno.

Slika 3.3. prikazuje drugačiji izgled baze čvorova, koji se dobije klikom na “Table”. Čvorovi su zapisani u vitičastim zagradama kao objekti s pripadnim svojstvima.



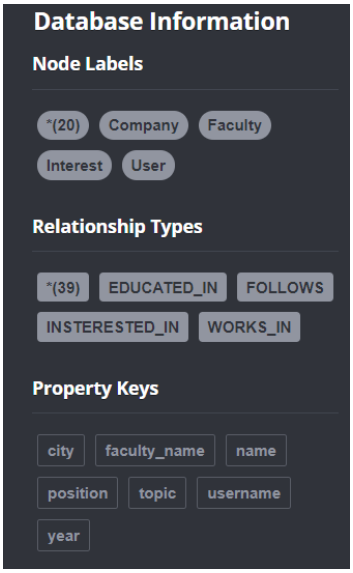
Slika 3.3. Prikaz dijela baze podataka u obliku tablice, gdje se vide i svojstva objekata.

U odjeljku Text u samoj aplikaciji postoji i prikaz čvorova koji poprilično nalikuje tipičnoj relacijskoj bazi podataka podaci su prikazani u obliku tablice, što se vidi na Slici 3.4.

Na vrhu slike vide se i upiti koji se automatski ispisuju prilikom klika.

Slika 3.4. Tekstualni prikaz baze podataka

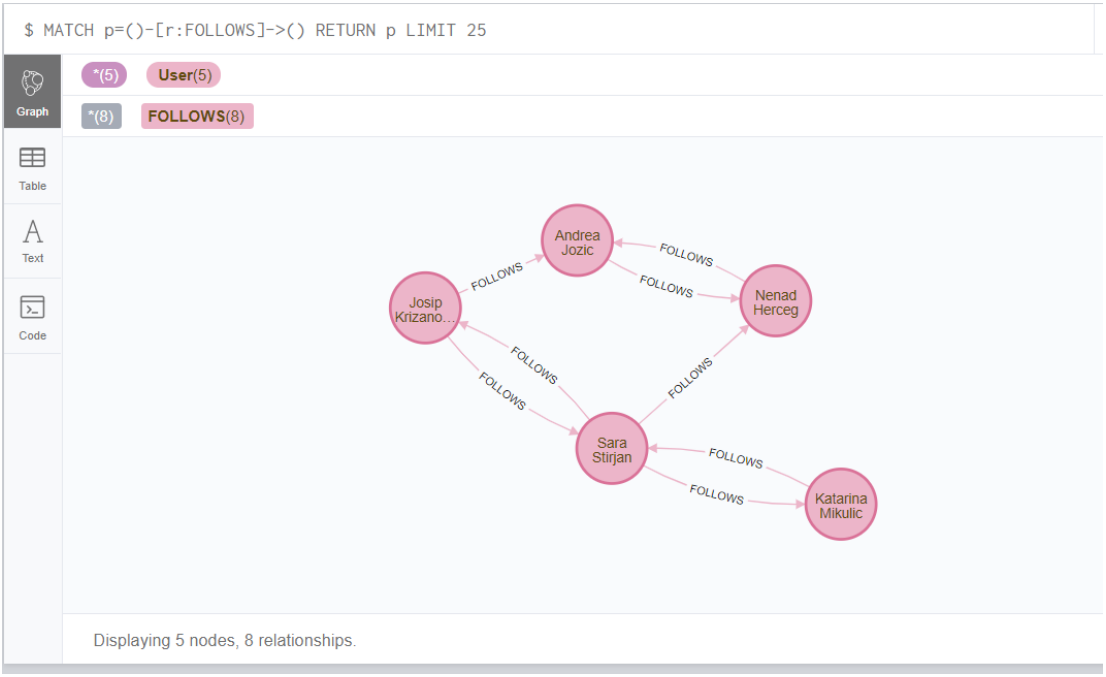
Također s lijeve strane sučelja nalazi se Database Information, vrlo pregledan prikaz grupa čvorova i veza. (Slika 3.5)



Slika 3.5. Database Information

Broj (*20) koji je prikazan u odjeljku *Node Labels* označava ukupan broja čvorova, a (*39) broj veza u bazi podataka. Također se može primjetit pri dnu Slike 3.2 broj čvorova i veza. Klikom na čvor, filtrira se veza ili svojstvo određene grupe čvorova ili veze, bez potrebe upita. Klikom na brojke, prikaže se grafovski prikaz čitave baze. Odabirom određene grupe čvorova, veze ili svojstva, prikazat će se samo članovi te grupe, ili veze zajedno s čvorovima koji su povezani tom vezom. Primjeri se mogu vidjeti na Slici 3.6 i Slici 3.7.

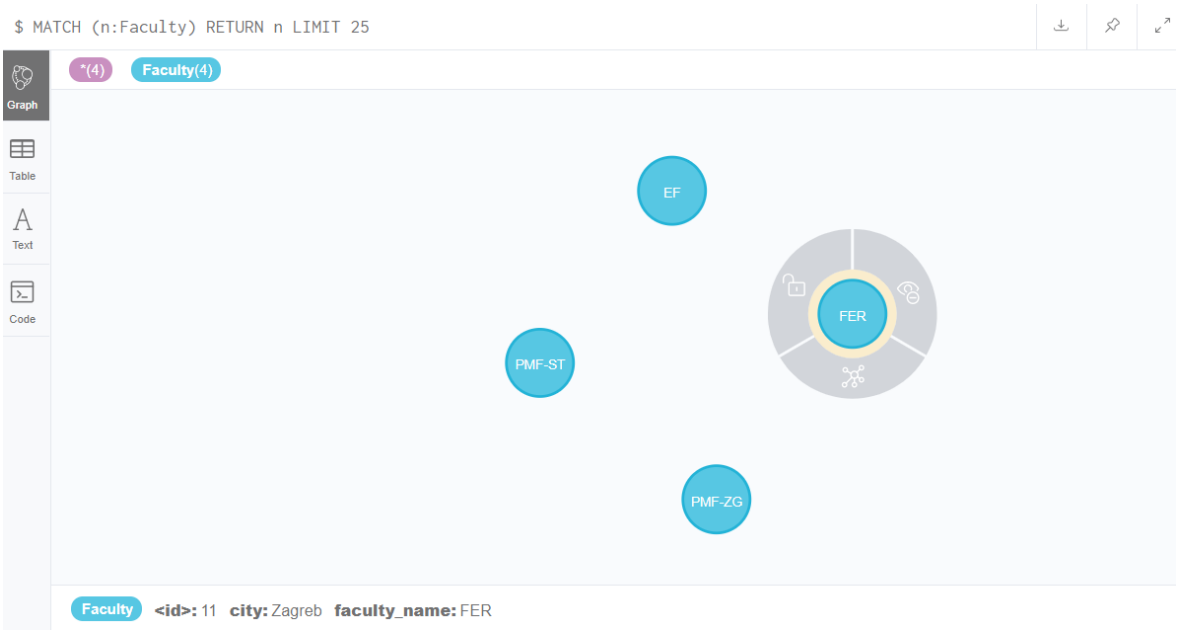
Slika 3.6.Prikaz čvorova koji su u grupi pod „Interest“



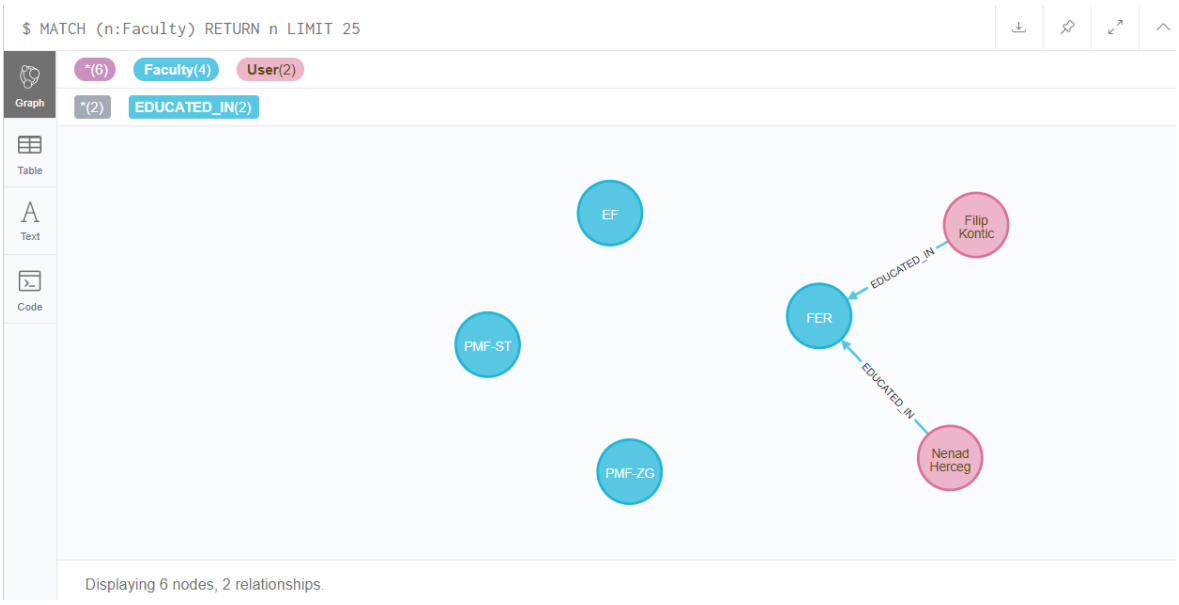
Slika 3.7. Prikaz veze„Follows“

Čvorovi u grafu prikazani su u obliku krugova ispunjenih bojom, a veze su usmjerene bridovi na kojima su prikazana imena veza. Može se primijetiti da se grupe čvorova unose početnim velikim slovom, veze velikim tiskanim slovima, a svojstva malim tiskanim slovima.

Klikom na jedan od čvorova na grafu, npr. na čvor „PMF-ST“ pod grupom „Faculty, taj čvor je označen i postoje tri opcije: zaključati, vidjeti koji čvorovi su povezani s njim i mogućnost uklanjanja iz prikazanog „lanca povezanosti“. Na Slici 3.8 i 3.9 prikazan je označeni čvor “FER”, i s kojim je ostalim čvorovima i kojim vezama čvor “FER” povezan.



Slika 3.8 Označeni čvor



Slika 3.9. Prikazane vezes kojima je određeni čvor još povezan u grafu

U slučaju potrebe ažuriranja podataka zapisanih u bazi, dohvaća se željeni čvor, a onda i svojstvo koje treba promijeniti, a zatim željenu promjenu.S naredbom RETURN, prikaže "novi" čvor.

```
MATCH (u:User{username:"Andrea Jozic"})
SET u.username="Andrea Previsic"
RETURN u;
```

Dakle, rezultat je grafovski.

Slika 3.10. Rezultat ažuriranja podataka u bazi

Kao što je prethodno navedeno, klauzula DELETE koristi se za brisanje čvorova, veza ili putova.

Slijedi primjer brisanja čvora. Jednostavno, pronađe se određeni čvor i izbriše:

```
MATCH (andrej:User{username:"Andrej Sarcevic"})
DELETE andrej
```

Rezultat uspješnog upita je: Deleted 1 node, completed after 1 ms.

U slučaju neuspješnog upita, na sučelju pojavi se obavijest o greški, kao i objašnjenje o kojoj se pogrešci radi, što uvelike olakšava ispravak upita.

3.3. Jednostavni upiti

U ovom poglavlju dani su primjeri jednostavnih upita u bazu podataka.
U prvom primjeru traže se imena “Usera” koji imaju poziciju radnog mjesta “Financial analyst”, i rade u gradu Zagrebu. Upiti uneseni u Editor prikazani su poviše svakog rezultata.

```
$ MATCH (p:User), (c:Company), (p)-[:WORKS_IN]->(c) WHERE (p.position="Financial analyst" and c.city="Zagreb") RETURN p.username;
```

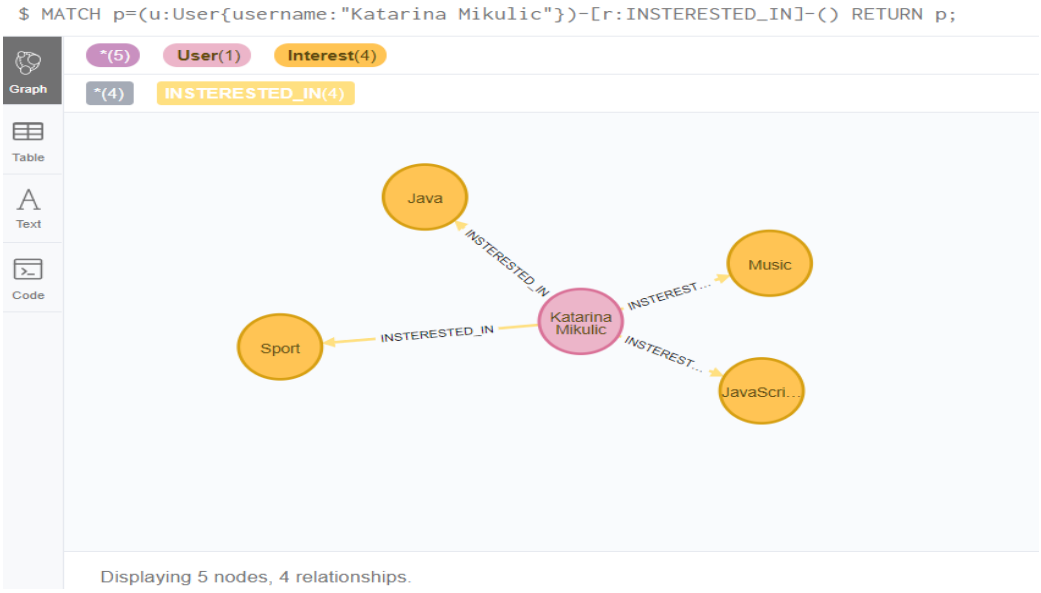
Table | Text | Code

p.username
"Ana Vidic"
"Katarina Mikulic"

Rezultat je prikazan u obliku tablice. Razlika je u tom što je u upitu traženo ime User-a, tj. njeno svojstvo (RETURN p.username), a ne cijeli čvor (RETURN p).

Idući primjer otkriva interese “Usera” Katarina Mikulic. Traženi upit sintaksno je drugačiji od prethodnog primjera, cijela relacija koja nas zanima je imenovana “p”, i rezultat je prikazan grafovski.
Rezultat na dnu sučelja prikazuje broj čvorova i veza koji su rezultat upita.

```
$ MATCH p=(u:User{username:"Katarina Mikulic"})-[r:INSTERESTED_IN]-() RETURN p;
```



Slijedi primjer složenijeg upita. Traže se korisnici s pozicijom “Web developer” koji imaju više od 3 godine iskustva i da je rezultat ispisa poredan silazno s obzirom na broj godina iskustva. Dakle, postavljena su dva uvjeta. Prvi uvjet je određen na način da sadašnju godinu oduzmemo od godine rada u tvrtki, a drugi je struka “Web developer”. Rezultat upita je ime “Usera” i “experience” kao podnaslov u tablici, što predstavlja upravo broj godina iskustva po kojem će Useri biti poredani. Prikazan je unos u editoru netom prije pokretanja, zbog veličine upita i preglednosti.

```
MATCH (u:User),
(u)-[w:WORKS_IN]->()
WHERE (((2019-w.year) > 3) AND u.position="Web developer")
RETURN u.username, (2019-w.year) AS experience
ORDER BY experience DESC
```

Dakle, rezultat su korisnička imena Josip i Filip, sa silaznim poretkom s obzirom na iskustvo.

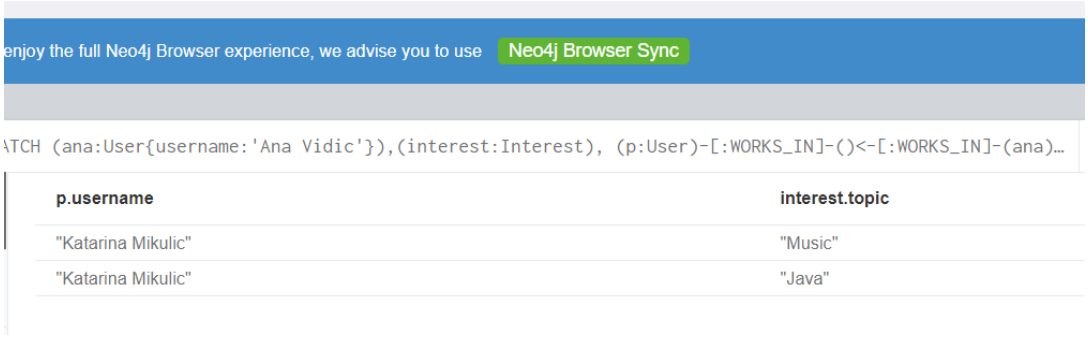
u.username	experience
"Filip Kontic"	14
"Josip Krizanovic"	5

3.4. Složeni upiti

Bitno je prikazati upite koji su u relacijskom modelu jako teški i spori. Kako baza LinkedIn nije velika, u svakom slučaju ne bi presudna bila brzina. Međutim, cilj je prikazati lakoću pisanja složenog upita na prirodan i logičan način.

U prvom primjeru složenog” upita traže se imena Usera koji rade u tvrtki u kojoj User Ana radi, a imaju zajedničke interese, i imena tih interesa.

```
1 MATCH (ana:User{username:'Ana Vidic'}),(interest:Interest),
2  (p:User)-[:WORKS_IN]-()<-[:WORKS_IN]-(ana)
3 where ((p)-[:INSTERESTED_IN]-(interest)<-[:INSTERESTED_IN]-(ana))
4 RETURN DISTINCT  p.username, interest.topic;
```



Drugi primjer upita traži imena prijatelja prijatelja i u kojem gradu rade. Fiksiran je čvor od kojeg se traže prijatelji prijatelja i grad u kojem rade.

```
1 MATCH (josip:User{username:"Josip Krizanovic"}), (c:Company),
2 (josip)-[:FOLLOWS]->(:User)-[:FOLLOWS]->(friend:User)
3 WHERE ((friend)-[:WORKS_IN]->(c) AND (not friend.username=josip.username))
4 RETURN DISTINCT friend.username, c.city;
```

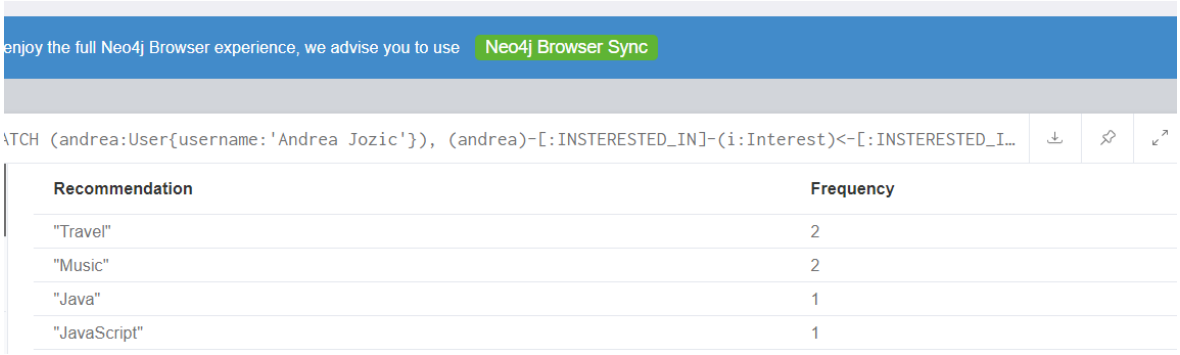
Dakle, dio upita u kojem se traže prijatelji prijatelja je logičan. Kreće se od fiksnog čvora "josip", i onda vezom "FOLLOWS" dolazi se do "prijatelja" koji nisu fiksirani, jer u ovom upitu nisu bitni, već njihovi prijatelji. Oni su spona do traženih čvorova. Zatim se na isti način nastavlja putanja i dolazi do traženih "User-a". Nakon postavljaju se uvjeti, gdje je prvi relacija i zanimaju nas tvrtke pod nazivom "Company" u kojima rade prijatelji prijatelja, te da među prijateljima prijatelja ne bude sam čvor "josip" koji je fiksiran na početku upita. Za rezultat očekuju se različita (DISTINCT) imena prijatelja te imena gradova:

friend.username	c.city
"Nenad Herceg"	"Zagreb"
"Katarina Mikulic"	"Zagreb"

Posljednji primjer složenijeg upita traži interese koje bi preporučili Andrei. Upit je

kreiran tako da se traže svi interesi korisnika koji imaju interese kao i Andrea, ali i dodatne potencijalne interese koje Andrea još uvijek nema. Interesi se ispisuju pod imenom “Recommendation”, a poredani su po učestalosti (“Frequency”) kojom se pojavljuju, te ispisani u padajućem obliku ovisno o učestalosti.

```
1 MATCH (andrea:User{username:'Andrea Jozic'}),
2  (andrea)-[:INSTERESTED_IN]-(i:Interest)<-[:INSTERESTED_IN]-(u:User)<-[:INSTERESTED_IN]->(reco)
3 WHERE not ((andrea)-[:INSTERESTED_IN]->(reco))
4 RETURN DISTINCT reco.topic as Recommendation,count (*) as Frequency
5 ORDER BY Frequency DESC LIMIT 5;
```



Recommendation	Frequency
"Travel"	2
"Music"	2
"Java"	1
"JavaScript"	1

U upitu se navodi uvjet „WHERE not ((andrea) –[: INTERESTED_IN]->(reco)“, jer nije potreban popis interesa koje korisnik Andrea već ima. Naizgled malo teži primjer, ali svaka strelica u MATCH klauzuli upita Cypher predstavlja vezu koja bi se modelirala kao tablica JOIN „mnogo-s-mnogo“ u relacijskom modelu s dva JOIN-a. Dakle,u SQL-u ovaj jednostavan upit obuhvaća potencijalno šest spajanja tablica.

3.5. Upiti u aplikaciji

Grafovske baze podataka korisne su u aplikacijama, pogotovo u kojima su važni veze među podacima. Za razliku od relacijskog modela podataka, u takvim sustavima veze među podacima specificirani su eksplicitno. Slijedi primjer već spomenutog upita: “ Tko su prijatelji prijatelja Johna?”, napisan u JavaScript-u, ali samo s oglednim primjerom baze podataka u “data”-u.

Konkretno, u ovoj bazi odgovor bi bio Anna (Joe zna Sally, a Sally Anu).

Prethodno, potrebno je preuzimanje i instaliranje Neo4j driver zs JavaScript:

Install neo4j-driver

```
$ npm install --save neo4j-driver
```

Ovo je jedan primjer kako izgleda kod u JavaScriptu

```
var neo4j = require('neo4j')

var db = new neo4j.GraphDatabase('http://neo4j:<password>@localhost:7474');

var insertQuery =

  "UNWIND {pairs} as pair \

   MERGE (p1:User {username}) \

   MERGE (p2:Person {username}) \

   MERGE (p1)-[:FOLLOWS]-(p2)";

var foafQuery =

 "MATCH (person:User)-[:FOLLOWS]-(friend)-[:FOLLOWS]-(foaf) \

  WHERE person.username = {username} \

   AND NOT (person)-[:FOLLOWS]-(foaf) \

  RETURN foaf.name AS name";

var commonFriendsQuery =

"MATCH (user:User)-[:FOLLOWS]-(friend)-[:FOLLOWS]-(foaf:Person) \

 WHERE user.username = {name1} AND foaf.username = {name2} \

 RETURN friend.username AS friend";


var connectingPathsQuery =

"MATCH path = shortestPath((p1:User)-[:FOLLOWS*..6]-(p2:User)) \

 WHERE p1.name = {name1} AND p2.name = {name2} \
```

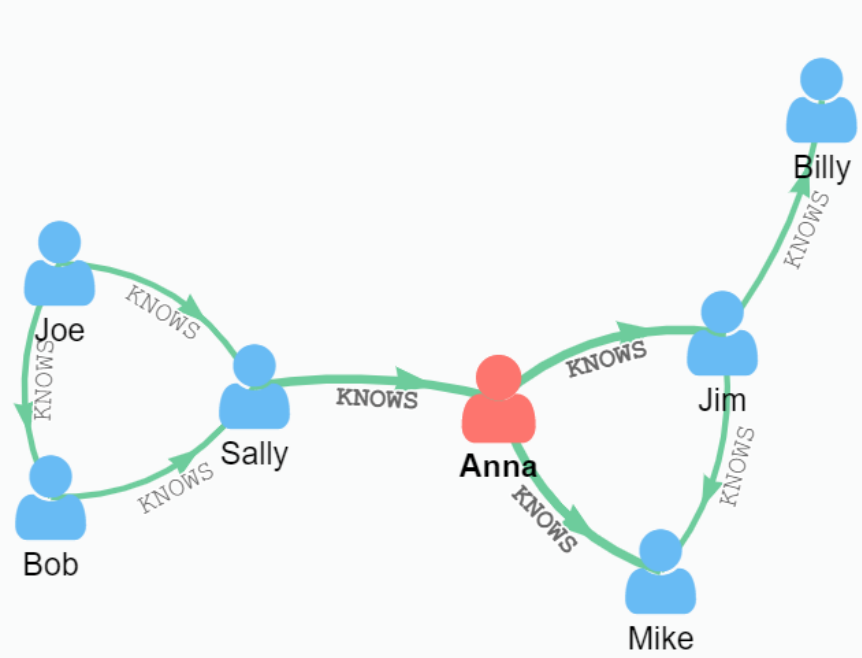
```
 RETURN [n IN nodes(path) | n.name] as names";

var data = [["Jim","Mike"],["Jim","Billy"],["Anna","Jim"],
["Anna","Mike"],["Sally","Anna"],["Joe","Sally"],
["Joe","Bob"],["Bob","Sally"]];


function query(query, params, column, cb){
    function callback(err, results) {
        if (err || !results) throw err;
        if (!column) cb(results)
        else results.forEach(function(row){ cb(row[column]) });
    };
    db.cypher({ query: query, params: params}, callback);
}

query(insertQuery,{pairs:data}, null,function(){
    query(foafQuery, {name:"Joe"},"name", console.log);
    query(commonFriendsQuery, {name1:"Joe", name2:"Sally"},"friend",
console.log);
    query(connectingPathsQuery, {name1: "Joe", name2:"Billy"},"names",
function(res){ console.log(res)});
});
```

Kako bi se upit bolje shvatio, slijedi grafovski prikaz na Slici 3.11.



–Slika 3.11. Prikaz baze podataka u aplikaciji

Literatura:

[1] Ian Robinson, Jim Webber,Emil Eifrem: *Graph Databases - New opportunities for connected data*, 2nd Edition, USA, 2015.

[2] Aleksa Vukotic, Nicki Watt,Tareq Abedrabbo, Dominic Fox, Jonas Partner: *Neo4j in Action*, NY, 2015.

[3] Robert Manger: *Baze podataka*, PMF, Element, Zagreb, 2012.

[4] Službena web stranica Neo4j-a https://neo4j.com

[5] Anka Golemac: *Osnove teorije grafova*, PMF, Sveučilište u Splitu

[6] Grafovske baze podatka- Pregled istraživanja i budućih trendova https://bib.irb.hr/datoteka/940636.Grafovske_baze_podataka__pregled_istrazivanja_i_buducih_trendova.pdf

[7] Nedostaci Neo4j-a https://tdwi.org/articles/2017/03/14/good-bad-and-hype-about-graph-databases-for-mdm.aspx

[8] Prednosti Neo4j-a https://imi.pmf.kg.ac.rs/imi-blog/grafovske-baze-podataka-neo4j/

Sažetak

Tema ovog rada je grafovska baza podataka. Na temelju analiziranih karakteristika grafovskih baza podataka i njihove usporedbe s karakteristikama tradicionalnih relacijskih baza podataka, zaključuje se kako ova kategorija NoSQL baza podataka zadovoljava potrebe suvremenih korisničkih zahtjeva. Proučavan je softverski paket za rad s grafovskim bazama podataka Neo4j, te njegov način rada, kao i prolazak po grafu. Iako mlada i nezrela tehnologija naspram relacijske baze podataka, danas se koristi u velikim korporacijama i u društvenim mrežama. Postupak pretraživanja je puno brži, budući da se upit izvodi nad određenim dijelom grafa, a ne nad cijelom bazom podataka, što predstavlja problem kod upita koji obuhvaćaju cijelu bazu podataka. Opisan je specifičan jezik za postavljanje upita, Cypher.

Na kraju rada dan je primjer grafovske baze podataka u Neo4j. Na njoj su prikazani jednostavni, složeni upiti, ali i specifični upiti koji su teško izvedivi u relacijskoj bazi podataka. Prikazan je primjer upita grafovske baze podataka u aplikaciji.

Summary

In this work, graph databases have been studied. Based on the analyzed characteristics of graph databases and their comparison with the traditional relational databases, this category of NoSQL responds user requirements. A software package Neo4j was studied, its description of work and search. Although young and immature technology versus relational database, it is now used in large corporations and social networks. The search is much faster, because the query is focused in specific part of the graph rather than the entire database, which is a problem with queries that inludes the entire database. Cpyher, specific query language is described.

At the end of the work, there is an example of graph database in Neo4j. It shows simple, complex queries. There are examples of specific queries, which are difficult to implement in the relational database. An example of a graph database query is shown in the application.

Životopis

Rođena sam 24. kolovoza 1993. u Splitu, gdje sam pohađala Opću gimnaziju „Marko Marulić“. Godine 2012. upisujem preddiplomski sveučilišni studij Matematika i informatika na Prirodoslovno - matematičkom fakultetu u Splitu, a 2016. godine upisujem diplomski studij Računarstvo i matematika na Prirodoslovno - matematičkom fakultetu u Zagrebu. Tijekom ljeta 2018. počela sam raditi u IT odjelu tvrtke AVL-AST u Zagrebu, gdje i danas radim na poziciji *Software developera*.