Izrada sustava za preporuke korištenjem grafovske baze podataka

Bojan, Kavur

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Bojan Kavur

Izrada sustava za preporuke korištenjem grafovske baze podataka

DIPLOMSKI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Bojan Kavur

Matični broj: 0016111158

Studij: Baze podataka i baze znanja

Izrada sustava za preporuke korištenjem grafovske baze podataka

DIPLOMSKI RAD

Mentor:

Izv. prof. dr. sc. Schatten Markus

Varaždin, rujan 2021.

Bojan Kavur

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Tema ovog diplomskog rada je izrada sustava za preporuke korištenjem grafovske baze podataka. Sustavi za preporuke su, u današnje vrijeme, vrlo popularni i gotovo ih svi koriste na neki način. Netflix je jedna on najpopularnijih tvrtki koja nudi streaming usluge i dostupna je korisnicima diljem svijeta, za što je zaslužan upravo sustav za preporuke koji tvrtka posjeduje. 2006. godine Netflix je organizirao i natjecanje u kojem su nudili velike novčane nagrade timovima koji će napraviti bolji sustav za preporuke od onog koji je Netflix tada posjedovao. Samim time možemo uočiti kolika je važnost sustava za preporuke i kako oni mogu privući korisnike da konzumiraju više sadržaja ili kupe proizvode za koje nisu niti znali da im trebaju prije nego im je sustav dao preporuku. S druge strane grafovske baze podataka spadaju u NoSQL baze podataka i koriste grafovski model podataka za modeliranje. Takav model je vrlo intuitivan, jednostavan i ekspresivan jer je vrlo sličan načinu na koji mi percipiramo određenu domenu.

U samom radu bit će detaljnije opisani sustavi za preporuke i algoritmi ili metode koje oni koriste za davanje preporuka. Nadalje, bit će opisane NoSQL baze podataka, koje vrste postoje, koje su njihove prednosti i nedostaci nad popularnim relacijskim bazama podataka uz detaljniji opis grafovskih baza podataka i algoritama temeljenih na grafovima korištenih za izradu sustava za preporuke. Na posljetku izrađen je sustav za preporuke knjiga uz korištenje Neo4j sustava za upravljanje grafovskim bazama podataka koji je povezan s programskim jezikom Java te sustav posjeduje i pripadno web sučelje za interakciju korisnika s aplikacijom.

Ključne riječi: sustavi za preporuke; NoSQL; grafovske baze podataka; model označenog grafa sa svojstvima; Neo4j; Cypher; teorija grafova; algoritmi temeljeni na grafovima

Sadržaj

1. Uvod

Cilj sustava za preporuke je korisniku pružiti relevantan sadržaj ili proizvod koji bi ga trenutno mogao zanimati. Ako se vratimo nekoliko desetaka godina, kada još takvi sustavi nisu postojali i nisu bili popularni, i nalazimo se u poziciji da smo pročitali odličnu knjigu koja nam se jako svidjela i željeli bismo pročitati sličnu knjigu. U tom slučaju mogli smo pitati prijatelje za preporuku, otići u knjižaru ili knjižnicu te sami pregledavati slične knjige prema autoru i žanru ili pitati zaposlenika što bi nam preporučio temeljem knjige koju smo pročitali. Time se oslanjamo na sadržaj koji je poznat prijatelju, zaposleniku ili se trenutno nalazi u knjižnici i knjižari čime postoji velika mogućnost da nam ne preporuče najbolju knjigu ili da ju ne nađemo. Isto bi bilo da se nalazimo u situaciji odabira filma za gledanje, glazbe za slušanje pa čak i hrane ili restorana te mjesta za posjet. Razvojem sustava za preporuke, takve stvari su nam puno lakše i kako su sami sustavi s vremenom napredovali daju relevantnije preporuke pojedinom korisniku temeljem njegovih interesa i interakcija sa sustavom. Naravno niti jedan sustav nije savršen pa tako niti sustavi za preporuke, kod njih je vrlo bitno koji algoritmi su korišteni i kakva je interakcija korisnika sa sustavom. Postoje i neki problemi kao što je hladan start (engl. „cold start“) i dugi rep (engl. „long tail“) koji će biti detaljnije objašnjen u samom radu.

Sustavi za preporuke postali su popularni razvojem weba, raznih društvenih mreža, platformi za slušanje glazbe i gledanje filmova te online dućana. Pomoću sustava za preporuke korisniku se pruža sadržaj koji je sličan njegovim interesima te tako sam korisnik konzumira više sadržaja ili se odluči na kupnju proizvoda koje nije inicijalno imao u planu kupiti. Postoje i određeni algoritmi ili metode koje se koriste u sustavima za preporuke i oni će biti detaljnije objašnjeni u radu. Kako bi shvatiti važnost ovih sustava bit će navedeni i primjeri uspješnih sustava za preporuke te Netflix-ova nagrada, što je natjecanje koje je Netflix organizirao sa svrhom razvoja boljeg sustava za preporuke od onog kojeg je tada koristio. Glavna nagrada bio je milijun dolara što nam govori da je njihov tadašnji sustav bio zaslužan za većinu njihovog profita, tj. korisnici su uglavnom konzumirali onaj sadržaj koji im je sustav za preporuke predložio.

Za izradu sustava za preporuke bit će korištena grafovska baza podataka koja spada u NoSQL baze podataka. Svima su dobro poznate relacijske baze podataka koje još uvijek dominiraju u svijetu baza podataka, ali se u posljednje vrijeme sve više spominju i NoSQL baze podataka koje su postale popularne kasnih 2000-ih godina. Popularne su zato što podržavaju polustrukturirane podatke, tj. imaju fleksibilnu shemu čime se lako mogu prilagoditi promjenama u domeni, skalabilne su i nude visoke performanse. To ne znači da relacijske baze podataka nisu dobre, nego da su se za određene domene NoSQL baze podataka

pokazale boljima i zato nije dobro sve prilagođavati relacijskom modelu podataka. U radu će biti detaljnije objašnjene NoSQL baze podataka i koje vrste postoje s fokusom na grafovske baze podataka koje se temelje na teoriji grafova. Neki od poznatijih modela podataka za grafovske baze podataka su RDF i model označenog grafa sa svojstvima. Postoje brojne grafovske baze podataka i sami sustavi za upravljanje grafovskim bazama podataka, od kojih je, u današnje vrijeme, najpoznatiji i najpopularniji Neo4j.

Praktični dio ovog rada opisivat će izgradnju sustava za preporuke knjiga. To je samo jedan primjer takvog sustava, a mogu se preporučivati filmovi, serije, proizvodi u online dućanu i drugo. Takve preporuke se baziraju na proizvodima koje je korisnik kupio i koje trenutno gleda ili sadržaju koji pretražuje. Svaki proizvod može se svrstati u razne kategorije te temeljem toga možemo korisniku preporučiti relevantne proizvode koji spadaju u istu kategoriju, isto tako korisnici mogu ocijeniti pojedini proizvod i sustav će preporučiti slične proizvode onima kojima je korisnik dao bolju ocjenu u odnosu na one koji se korisnicima iz nekog razloga ne sviđaju. Za implementaciju sustava bit će korištena Neo4j grafovska baza podataka, koja će biti povezana programskim jezikom Java te će aplikacija sadržavati i pripadno web sučelje za interakciju korisnika sa sustavom. Budući da se grafovske baze podataka temelje na teoriji grafova, postoje i razni algoritmi temeljeni na grafovima. Grafovske baze podataka koriste algoritme obilaska grafa za dohvat određenih podataka, ali uz njih postoje i drugi algoritmi koji se mogu koristiti u razne svrhe, a ovom će radu biti fokus na algoritmima sličnosti jer će se tražiti slični korisnici i slične knjige za preporuku.

U daljnjem sadržaju ovog rada prvo slijedi opis sustava za preporuke i NoSQL baza podataka uz naglasak na grafovske baze podataka. Nakon toga slijedu praktični dio rada u kojem će biti detaljniji opis Neo4j grafovske baze podataka, algoritama temeljenih na grafovima koji će se koristi za implementaciju sustava za preporuke knjiga te sama implementacija sustava uz primjer korištenja aplikacije.

2. Sustavi za preporuke

Razvojem interneta kao medija i velikih količina informacija koje su putem njega dostupne, razni proizvodi i usluge kao i socijalna povezanost, ovakvi sustavi su nam sve više i više potrebni. U današnje vrijeme, sustavi za preporuke su gotovo neizbježni i neku vrstu takvog sustava možemo naći gotovo na svakoj web stranici. Jedan primjer, s kojim su se gotovo svi susreli su Google oglasi ili reklame. Svaki korisnik Google-ovih servisa se u nekom trenutku susreo s raznim reklamama i oglasima, ako obratimo pažnju na reklame koje se pojavljuju možemo uočiti da su uglavnom povezane sa sadržajem kojeg trenutno gledamo, kojeg smo nedavno pretraživali ili sa sadržajem kojeg često pretražujemo.

Pitamo se zašto nam uopće trebaju sustavi za preporuke i odgovor leži u tome da smo pretrpani informacijama (engl. „Information Overload“). Kao ljudi ograničeni smo na to koliko informacija možemo procesirati u određenom vremenskom periodu i zbog toga su nam potrebni sustavi za preporuke da nam smanje količinu informacija i samim time olakšaju proces odabira. Svaka osoba ima svoje preferencije i na temelju njih sustav može pružiti relevantne preporuke, recimo da kupujemo sportsku obuću i plava nam je omiljena boja. U tom slučaju sustav će filtrirati podatke i kao preporuku nam nuditi samo sportsku obuću plave boje jer posjeduje takve podatke o nama i vrlo je vjerojatno da tražimo upravo takav proizvod. [1]

Prema Falku [2, str. 13] sustav za preporuke je sustav koji izračunava te korisniku pruža relevantan sadržaj na temelju podataka dostupnih o korisniku, sadržaju i interakciji između korisnika i sadržaja. Općenito, možemo reći da su sustavi za preporuke zapravo algoritmi koji na temelju dostupnih podataka korisnicima daju relevantne preporuke.

Potrebno je razlikovati reklame i preporuke. Kao što je vidljivo iz prethodne definicije korisnik dobije preporuku temeljem podataka koji su sustavu dostupnim o njemu. Prije postojanja samih sustava za preporuke i prikupljanja podataka o korisnicima temeljem njihove interakcije sustavom ili aplikacijom, preporuku smo mogli dobiti od prijatelja ili osobe koja dobro poznaje domenu te ima neke podatke o korisniku. Ključno je da taj netko tko daje preporuku zna što korisnik voli i kakav sadržaj ga interesira, naravno kvaliteta same preporuke ovisi o dostupnim informacijama i koliko dobro osoba koja daje preporuku poznaje domenu ili ako se radi o sustavu za preporuke koji algoritmi su korišteni. S druge strane reklame ili oglasi su nešto što dobijemo od nekoga tko može, ali i ne mora znati ništa o korisniku kojem su namijenjene. S obzirom na to, reklamom nećemo dobiti ono što bi bilo najbolje za nas ili bi nas potencijalno moglo zanimati već reklamiraju ono što imaju i to su uglavnom novi proizvodi ili usluge. Reklame nam često daju razni supermarketi u obliku nekog letka u kojem se nalazi lista proizvoda koja je na akciji. [3]

Ranije navedeni Google oglasi ili reklame spadaju u preporuke jer se baziraju na sadržaju kojeg pretražujemo ili ako imamo kreiran korisnički račun tada sustav prikuplja podatke o sadržaju kojeg smo pretraživali i koji nam se sviđa te pruža oglas koji je povezan s tim sadržajem. Dakle, korisniku pruža oglas ili reklamu temeljem podataka koji su sustavu dostupni o tom korisniku s ciljem davanja personalizirane preporuke.

Sustavi za preporuke igraju veliku ulogu u e-poslovanju, u današnje vrijeme većina ljudi odlučuje se za online kupovinu. Tako mogu obaviti kupnju u bilo koje vrijeme i na bilo kojem mjestu pa čak i iz udobnosti svog doma bez trošenja vremena na odlazak do trgovine. Da bi kompanije povećale svoj profit potrebno je privući pozornost korisnika i što duže ga zadrži na svojim stranicama, a to mogu napraviti tako da mu ponude one proizvode koji su mu potrebni u tom trenutku ili nešto što korisnik voli te bi potencijalno mogao kupiti. Osim online trgovina, sustavi za preporuke koriste se i za preporuku sadržaja koji je korisniku interesantan. Postoje razni mediji za gledanje videa i filmova, čitanje online knjiga i slušanje glazbe, kako bi privikli korisnika da što duže koristi medij potrebno mu je dati relevantne preporuke jer će tako biti više zainteresiran za sadržaj nego da sam filtrira gomilu dostupnog sadržaja. Takvim sustavima ne profitiraju samo kompanije, već i korisnici jer mogu lakše i brže pronaći sadržaj koji ih interesira. [4], [5]

Možemo zaključiti da sustavi za preporuke pretvaraju dostupne podatke o korisniku u personalizirane preporuke za tog korisnika. Takvim preporukama možemo postići bolje zadovoljstvo korisnika, zadržati ih dulje na stranicama i povećati količinu prodaje ili količinu sadržaja kojeg će pregledati. [6]

2.1. Problemi u sustavima za preporuke

Sustavi za preporuke, kao i većina sustava, susreću se s određenim problemima koji se javljaju prilikom razvoja i korištenja sustava. Autori Lakshmi [7] u svom članku navode probleme koji se javljaju u sustavima za preporuke i to su:

- Problem dugog repa
- Problem hladnog starta
- Skalabilnost sustava
- Pouzdanost preporuka
- Raznolikost preporuka i davanje novih preporuka
- Pogrešni i nepravilni podaci s kojima sustav radi
- Rješavanje konflikata nastalih korištenjem hibridnih metoda preporuke
- Rangiranje preporuka

- Utjecaj mobilnosti i sveprisutnosti
- Veliki podaci
- Privatnost

Skalabilnost je problem s kojim se susreću i mnogi drugi sustavi, to je zapravo sposobnost sustava da održi željene performanse kako raste i posjeduje sve više podataka. Sustavi za preporuke bi trebali zadržati svoje performanse s rastom podataka jer je korisniku potrebno dati preporuke u realnom vremenu i s rastom broja korisnika i proizvoda to može biti izazovno i zahtijeva više resursa. [8]

Pouzdanost preporuka je vrlo važna jer sustavi za preporuke korisniku trebaju pružiti one proizvode koji bi ga u tom trenutku mogli zanimati jer u suprotnom neće biti zadovoljni sa sustavom i neće ga htjeti koristiti što nije dobro. Isto tako je bitna i raznolikost preporuka što znači da je potrebno uzeti u obzir što više proizvoda, tj. da se ne preporučuju uvijek isti i najpopularniji proizvodi. [8]

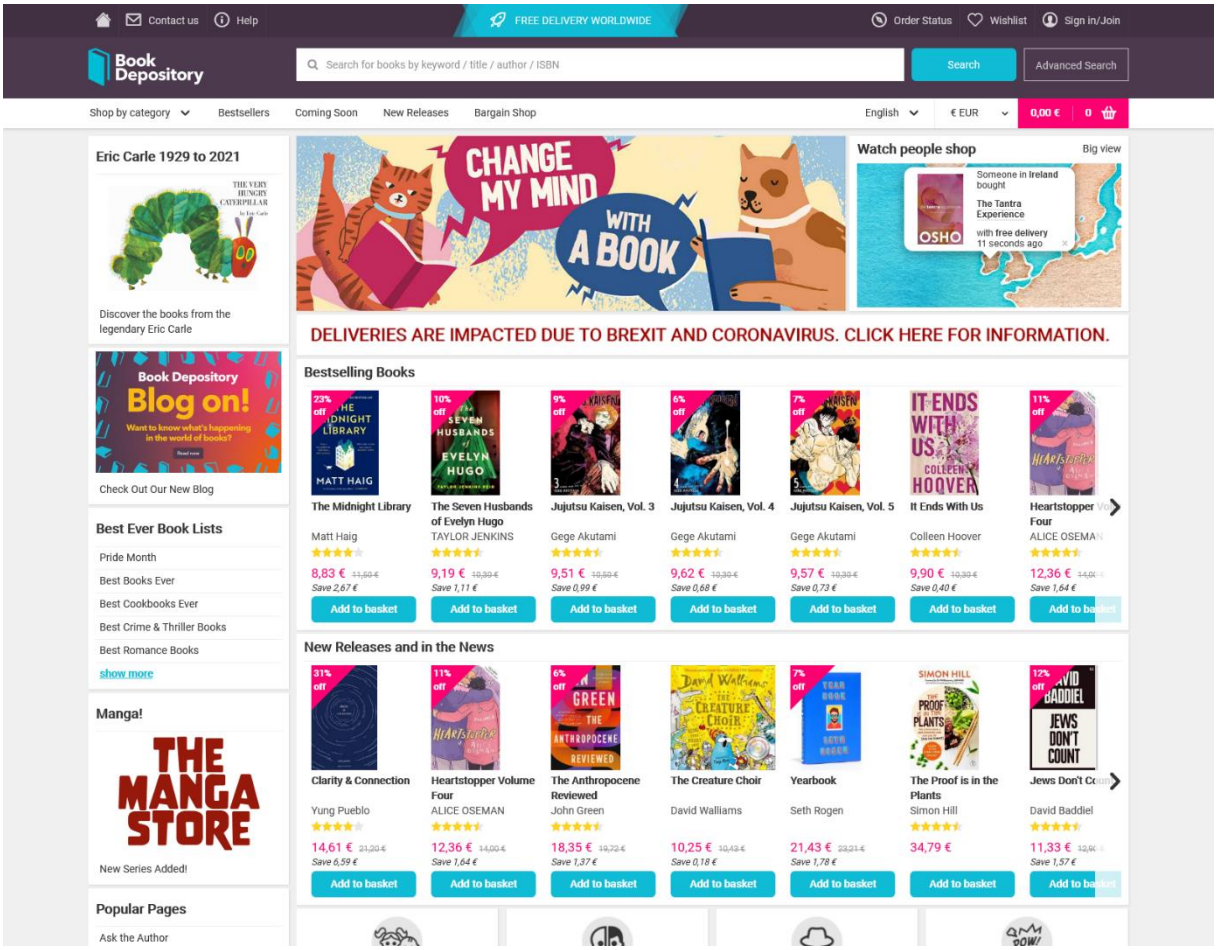
Rangiranje preporuka također je bitno jer je potrebno na neki način izračunati koji proizvodi bi u kojoj mjeri mogli zanimati korisnika i samim time preporučiti one proizvode koji više odgovaraju profilu tog korisnika. Nadalje, privatnost je u današnje vrijeme vrlo važna i izazovna te je potrebna gotovo u svakom sustavu pa tako i u sustavu za preporuke. Sustav treba određene podatke o korisniku kako bi mogao pružati kvalitetnije preporuke i samim time je potrebno čuvati takve podatke tajnima. [8]

Navedeni problemi mogu u većoj ili manjoj mjeri utjecati na kvalitetu samog sustava za preporuke i stoga ih je potrebno riješiti u određenoj mjeri. U idućim potpoglavljima bit će detaljnije opisani veći problemi s kojima se susreću sustavi za preporuke.

2.1.1. Problem dugog repa

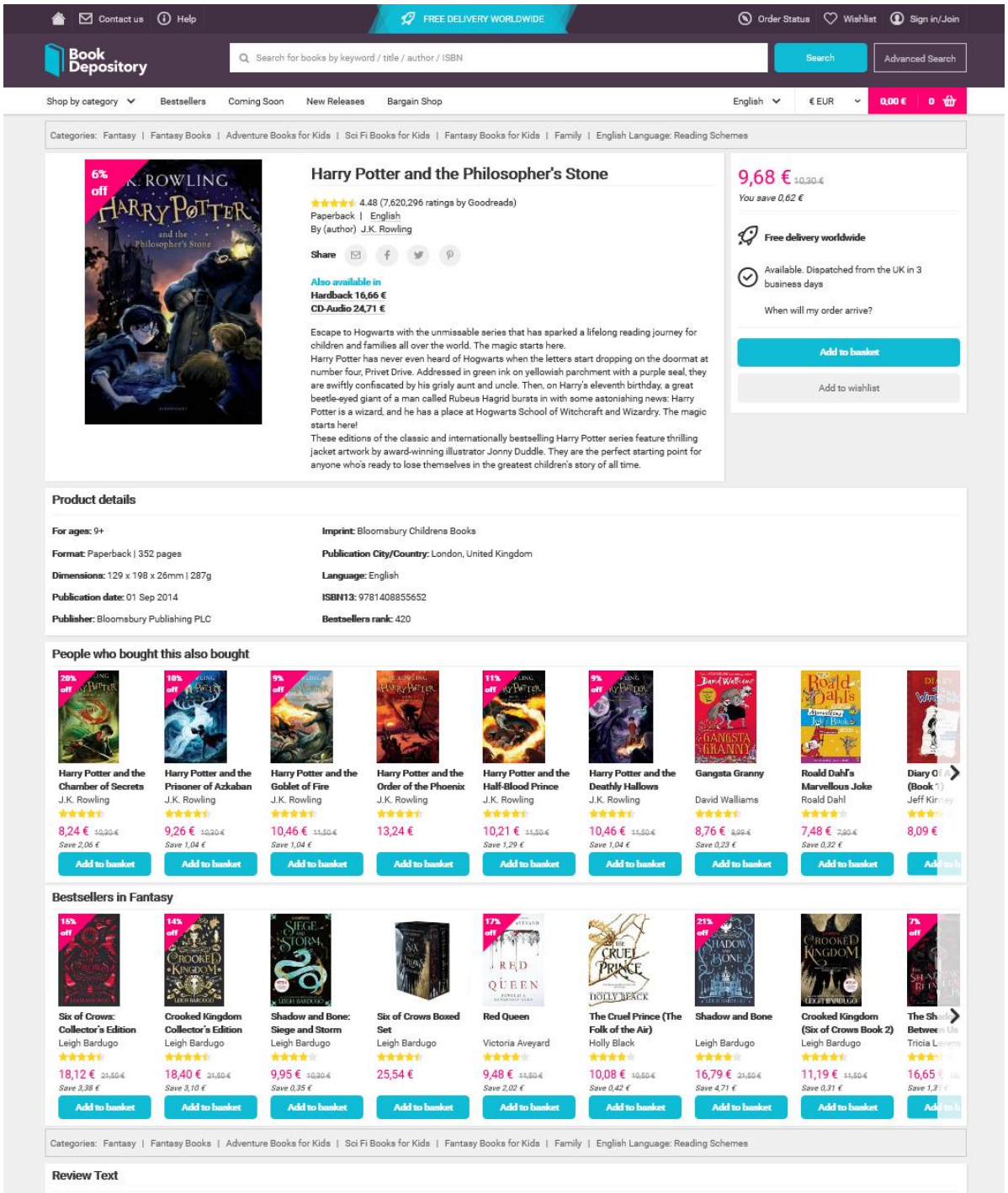
Problem dugog repa (engl. „The Long Tail Problem“) javlja se većinom u online kupovini i može se riješiti korištenjem sustava za preporuke. Radi se o problemu koji je vezan uz ograničenu količinu proizvoda koji se mogu izložiti. U fizičkoj trgovini možemo vidjeti samo dio asortimana jer trgovina ima ograničen prostor u kojem može izložiti proizvode. Isto tako imamo i ograničen broj kupaca jer ljudi moraju fizički doći u trgovinu da bi nešto kupili. Iz tog razloga u trgovini poput knjižare možemo vidjeti sekcije kao što su najprodavanije, najpopularnije i nove knjige. Naravno postoji i dio asortimana koji je izložen uglavnom prema žanru, ali je nemoguće izložiti sav asortiman zbog ograničenog prostora. Samim time trgovci moraju odlučiti što će biti izloženo, a što će se nalaziti u nekom skladištu i kupcima biti dostupno jedino ako prodavaču daju upit o tom artiklu. S obzirom na to, u knjižari ćemo uglavnom naći one

knjige koje su vrlo popularne i nova izdanja, dok one manje popularne nema smisla držati u izlogu jer je mogućnost prodaje puno manja. Ukoliko ista knjižara posjeduje svoj online dućan tada više nema ograničenje prostora već može korisnicima prezentirati čitav asortiman i time dolazimo do problema dugog repa. Recimo da korisnik ne zna što želi, tada će imati problem s nalaženjem sadržaja, točnije knjiga koje bi ga mogle zanimati. Upravo ovdje dolaze sustavi za preporuke kao rješenje navedenog problema tako da korisnicima pomažu pronaći one knjige za koje možda niti ne znaju da postoje i koje odgovaraju profilu korisnika. Ovakve preporuke se odnose na korisnike koji imaju korisnički račun i koji su imali neku interakciju sa sustavom kako bi sustav mogao prikupiti podatke o tom korisniku i dati relevantnije preporuke. Naravno potrebno je na neki način filtrirati sadržaj što se uglavnom radi na isti način kao i u fizičkom dućanu, ako odemo na web stranicu neke knjižare prvo ćemo vidjeti najprodavanije knjige i nova izdanja. Isto tako imat ćemo mogućnost pretrage prema žanru ili autoru tako da korisnik može filtrirati onaj sadržaj koji ga interesira. [2, str. 5 i 6], [4], [6]



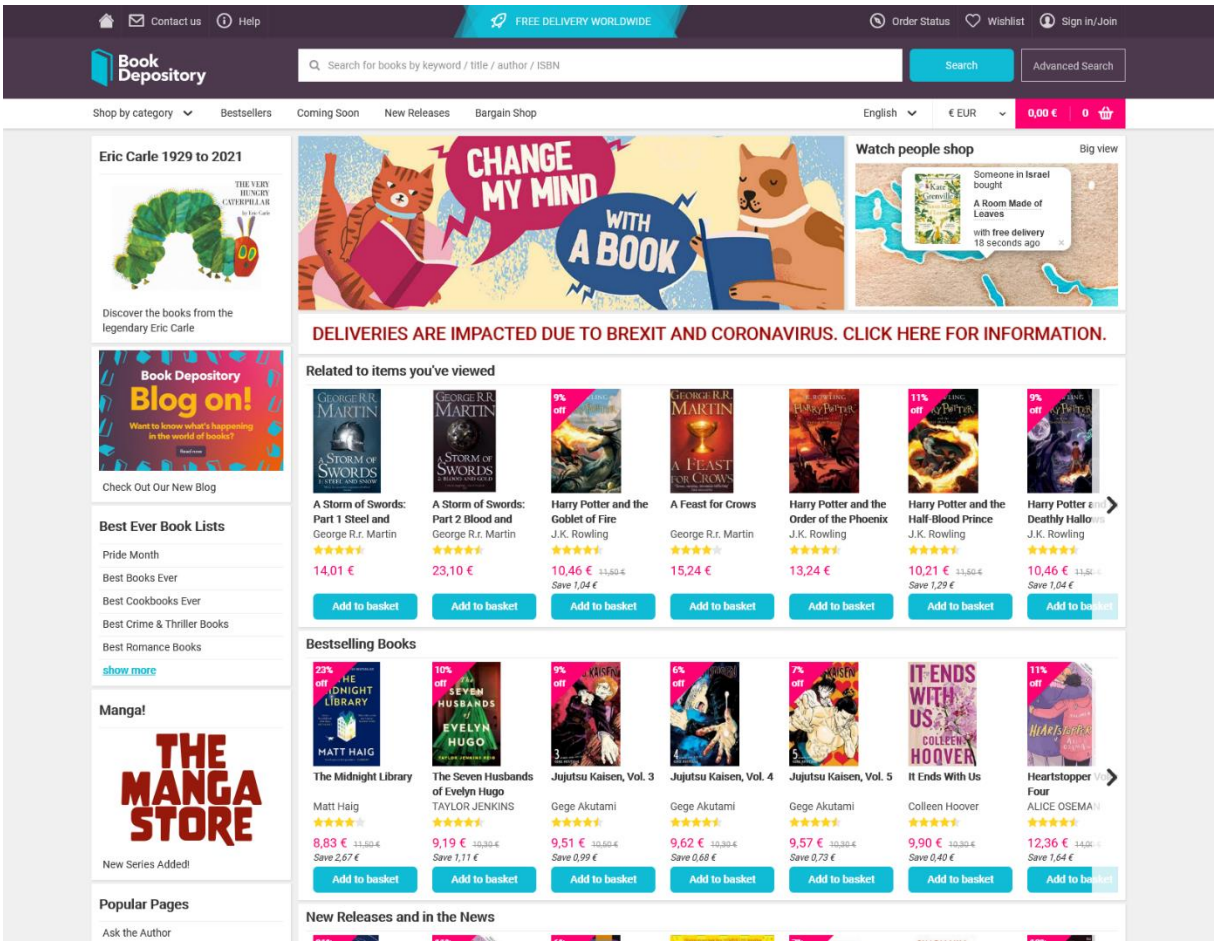
Slika 1: Book Depository - početna stranica [9]

Prethodna slika prikazuje početnu stranicu web shopa Book Depository gdje možemo vidjeti da nam kao preporuku nude najprodavanije knjige i nova izdanja. Ukoliko otvorimo određenu knjigu i pogledamo detalje tada možemo vidjeti da nam sustav za preporuke predlaže knjige koje su drugi korisnici kupili uz knjigu koju trenutno gledamo te popularne knjige koje pripadaju istom žanru. Na idućoj slici vidljive su takve preporuke za knjigu Harry Potter i Kamen mudraca (engl. „Harry Potter and the Philospoher's Stone“).



Slika 2: Book Depository – Harry Potter i kamen mudraca [9]

Nadalje, ako pogledamo još jednu Harry Potter knjigu i recimo neke knjige od autora George R. R. Martina i vratimo se na početnu stranicu, možemo uočiti da je sustav pratio našu interakciju i knjige koje smo pregledavali. Samim time vidimo nove preporuke knjiga koje su slične onima koje smo pretraživali (engl. „Related to items you've viewed“).



Slika 3: Book Depository - početna stranica preporuke [9]

Takvim preporukama sustav pomaže korisnicima pronaći sadržaj koji ih interesira temeljem podataka koje je prikupio o korisniku i samim time privući korisnika na kupnju preporučenih knjiga. Bez takvog sustava uvijek bi vidjeli najprodavanije i nove knjige koje nam možda nisu interesantne i potrošili bi više vremena na traženje knjiga koje nas zanimaju. Tada bi korisnici uglavnom pretraživali knjige koje ih potencijalno ne zanimaju i na kraju bi kupili manji broj knjiga ili ih uopće ne bi kupili. S druge strane ovakav sustav korisniku može predložiti knjige za koje nije niti znao da postoje, a spadaju u kategoriju knjiga koja interesira korisnika i time povećava šansu da taj korisnik kupi preporučenu knjigu. Dakle, sustavi za preporuke

pomažu korisnicima da lakše pronađu sadržaj koji ih interesira i ujedno ga potiču da kupi, u ovom primjeru, više knjiga te tako i poduzeće ostvaruje veći profit.

2.1.2. Problem hladnog starta

Problem hladnog starta (engl. „Cold start problem“) javlja se u situacijama kada se u sustav dodaje novi proizvod ili novi korisnik. Ovo je zapravo jedan on najvećih problema sustava za preporuke jer se radi o manjku podataka potrebnih za davanje personaliziranih preporuka. Ako u sustav dođe novi korisnik koji još nije ocijenio niti jedan proizvod i nemamo informaciju koji su njegovi interesi tada im ne možemo dati relevantnu preporuku proizvoda koji bi ga mogao interesirati. S druge strane ako u sustav dođe novi proizvod kojeg još nitko nije ocijenio, tada ne možemo znati da li je on dobar ili ne i kome ga preporučiti. [2, str. 128 i 129], [10]

Što se tiče novih proizvoda, vrlo je važno da ih na neki način prezentiramo korisnicima kako bi ih oni mogli vidjeti i ocijeniti te samim time sustav prikuplja potrebne informacije pomoću kojih kasnije može drugim korisnicima preporučivati te proizvode. Ako se vratimo na sliku 1 koja prikazuje početnu stranicu web shopa Book Depository, možemo uočiti sekciju novih proizvoda, tj. nova izdanja knjiga (engl. „New Releases and in the News“). Ovo je ujedno i rješenje kako promovirati nove proizvode, radi se o reklami ili oglasu koji korisnicima daje do znanja da su izašli novi proizvodi. Kako većina ljudi voli pogledati što je novo izašlo, vrlo je velika vjerojatnost da će vidjeti te proizvode i potencijalno ih kupiti čime sustav za preporuke dobiva potrebne informacije za generiranje preporuka drugim korisnicima. Naravno s vremenom kako će sustav imati više informacija, moći će kreirati i kvalitetniju preporuku. [2, str. 130], [10]

Nadalje, novim korisnicima, o kojima sustav još nema dovoljno informacija, on ne može dati relevantnu preporuku. Svi korisnici bi htjeli da im se što prije nudi sadržaj koji ih interesira, ali su brojna istraživanja dokazala da je vrlo teško napraviti relevantnu preporuku prije nego što korisnik ocijeni minimalno 20 do 50 proizvoda. Naravno, kao i s proizvodom, preporuka će biti kvalitetnija ako sustav ima više informacija o korisniku. Isto tako, ako imamo malo podatak o korisniku postoji mogućnost da će sustav stvoriti djelomičnu ili pogrešnu sliku o korisniku te će mu dati loše preporuke. Na primjer možemo uzeti korisnika koji je ocijenio samo jednu knjigu koja spada u žanr kriminalistike i sustav tom korisniku počinje preporučivati samo one knjige koje spadaju u taj žanr. U tom trenutku sustav ne zna da taj isti korisnik voli fikciju i fantastični roman možda i više nego kriminalistiku. Takva implementacija, kao što vidimo iz ovog malog primjera, nije najbolja i stoga je potrebno prikupiti više podataka kako bi preporuke bile kvalitetnije. [2, str. 130 i 131], [10]

Problem hladnog starta javlja se i kod korisnika koji nisu prijavljeni u sustav, tj. nemaju kreiran korisnički račun ili se iz nekog razloga nisu prijavili u sustav. Postoji puno korisnika koji koriste više uređaja i nisu prijavljeni na svakom uređaju već samo žele pogledati detalje o nekom proizvodu. U prethodnom problemu dugog repa opisan je kratak primjer web shopa Book Depository i kao što je vidljivo niti jedan korisnik nije prijavljen u sustav, ali nakon pregleda knjige ili više njih vidljive su preporuke. Radi se o tome da web preglednik sprema kolačiće na računalo korisnika i tako jedinstvenim identifikatorom, uglavnom ID sesije, može pamtiti to računalo i web preglednik kao anonimnog korisnika. Takav pristup je dobar u slučaju da korisnik koristi samo jedan uređaj za pristup web shopu i da taj uređaj ne dijeli s drugim ljudima. Ako, za primjer, jedna obitelj ima računalo koje je dostupno svim ukućanima i nitko se ne želi prijaviti u sustav preko svog korisničkog računa tada će ovakve preporuke biti vrlo loše. Mogu se eventualno s vremenom popraviti ukoliko jedan član obitelji pretraži puno sadržaja koji ga zanima, ali čim drugi član pogleda jednu knjigu izvan interesa prethodnog člana tada će preporuke opet postati loše. Drugi problem ovog pristupa je brisanje kolačića, ako se obrišu kolačići s računala koji su povezani sa stranicom web shopa tada će se prilikom novog pokretanja stranice korisniku i računalu dodijeliti novi ID sesije koji će web preglednik spremiti kao kolačić na računalo korisnika i kako sustav nema podatke za taj ID sesije, neće niti moći dati preporuke. Ovakav pristup je ograničen na to da se mora koristiti isti uređaj i web preglednik za pristup stranici, ako se i dalje držimo primjera web shopa Book Depository i koristimo web preglednik Chrome te se nakon nekog vremena odlučimo za korištenje preglednika Mozilla Firefox, tada novi preglednik dobiva svoj kolačić za tu stranicu koji sprema na računalo korisnika za sebe. Dakle web preglednici međusobno ne dijele kolačiće i time se gube podaci koje je sustav uspio prikupiti o takvom anonimnom korisniku. Isto tako ograničenje se odnosi na korištenje više uređaja, ako korisnik koji je za interakciju sa sustavom koristio računalo prijeđe na mobilni uređaj tad će imati drugi kolačić na tom uređaju i samim time sustav ne može znati da je to isti korisnik. Što se tiče sustava, ovakav pristup zahtjeva spremanje ID-a sesije za svaki kolačić koji se kreira. Ako sustavu pristupa velik broj anonimnih korisnika može doći do problema sa spremanjem i dohvatom tih podataka, uz samog korisnika potrebno je spremati i njihovu interakciju sa sustavom kako bi se mogle dati preporuke. Sve u svemu, najbolje je da korisnik koji želi dobiti relevantne preporuke kreira svoj korisnički račun te se na svakom uređaju prijavi u sustav s tim računom kako bi mu sustav mogao dati kvalitetnije preporuke temeljem sadržaja koji pretražuje i ocjenjuje. [2, str. 134], [10]

Kao rješenje problema hladnog starta korisnika može se koristiti hibridna metoda ili algoritam sustava za preporuke, a kao rješenje hladnog starta proizvoda osim prethodno navedenih reklama ili oglasa koristi se i algoritam sustava za preporuke na temelju sadržaja.

Postoje tri metode ili algoritma koji se koriste u sustavima za preporuke i oni će biti objašnjeni u idućim poglavljima. [2, str. 133]

U svom članku autori Bahadorpour i Nadimi–Shahraki navode metode i strategije pomoću kojih se rješava problem hladnog starta korisnika. Kao rješenje navode metodu pitaj za ocjenu (engl. „ask–to–rate“) i to je zapravo izravan način prikupljanja podataka o novom korisniku. Na taj način, preko kratke pitalice ili intervjua, od novoregistriranog korisnika možemo prikupiti određene podatke pomoću kojih tada sustav za preporuke tom korisniku može dati kvalitetnije preporuke nego bi to mogao bez da ima neke informacije o njemu. Ukoliko se koristi ova metoda treba voditi računa o tome da se korisnika ne preoptereti pitanjima i da iz tog kratkog intervjua sustav prikupi što kvalitetnije informacije o korisniku. Postoje adaptivne i neadaptivne metode pitanja korisnika za ocjenu. [11]

Neadaptivne metode daju nešto manje informacija o korisniku jer se ne uzima u obzir prethodno ocijenjen proizvod. Recimo da korisniku nudimo 5 proizvoda na ocjenu tada se za drugi proizvod neće uzeti u obzir ocjena prvog proizvoda u smislu da se ponudi sličan proizvod ukoliko je korisnik dao dobru ocjenu ili da se promijeni tip proizvoda ako je dao lošu ocjenu. U neadaptivne metode spada random strategija kojom se korisniku nude nasumični proizvodi na procjenu i ta strategija je jedna od najlošijih jer se uzimaju u obzir svi proizvodi i velika je mogućnost da se korisniku većina neće svidjeti što znači da kasnije sustav za preporuke neće imati podatke koji su mu potrebni za davanje relevantne preporuke. Bolja neadaptivna strategija je strategija popularnosti u kojoj se u obzir uzima popularnost proizvoda što znači da se korisniku na procjenu daju neki od najpopularnijih proizvoda. Većini korisnika će se svidjeti popularni proizvodi, ali tako zanemarujemo one manje popularne i nove proizvode. [11]

Postoje i adaptivne metode koje u obzir uzimaju prethodne ocjene proizvoda i temeljem toga nude idući proizvod na procjenu. Time se kvalitetnije kontrolira proces intervjuiranja novog korisnika i tijekom procesa se već traže oni proizvodi koji bi ga mogli više interesirati te mu se nude za procjenu. Tako sustav za preporuke odmah prikupi kvalitetnije podatke i može dati relevantnije preporuke tom korisniku. Jedna od strategija je personalizirana na temelju proizvod–proizvod i radi na principu da se u obzir uzimaju oni proizvodi koje je korisnik prethodno ocijenio s boljom ocjenom te se kao idući proizvod za procjenu nudi neki sličan tom proizvodu. Naravno za prvi proizvod ili nekoliko njih moraju se uzeti popularniji proizvod pa neki koji spada u drugu kategoriju sve dok se ne nađe onaj koji se korisniku sviđa te se daljnji proizvodi nude na temelju njega. Tu se već zapravo radi o nekoj vrsti sustava za preporuke jer se pokušavaju dati na procjenu oni proizvodi koji bi korisnika mogli interesirati. [11]

Metoda pitaj za procjenu nije previše popularna jer zahtjeva od korisnika da utroši vrijeme i ocjeni neke proizvode koji ga možda niti ne interesiraju. Uglavnom se koristi hibridni

algoritam sustava za preporuke, koji će biti objašnjen u jednom od idućih poglavlja ovog rada, jer on ne zahtijeva da korisnik procjenjuje neke proizvode već na početku daje preporuke na temelju sadržaja dok ne prikupi dovoljno informacija za davanje personaliziranijih preporuka na temelju kolaborativnog filtriranja.

2.2. Netflix – ova nagrada

Netflix je 2006. godine pokrenuo otvoreno natjecanje vezano uz rudarenje podacima i strojno učenje za predviđanje ocjene filma. U tom natjecanju Netflix je nudio nagradu od milijun dolara timu koji će uspjeti izraditi bolji sustav za preporuke od njihovog trenutnog sustava Cinematch, točnije sustav treba biti najmanje 10% bolji. Cinematch je sustav za preporuke koji je na tjednoj bazi automatski analizirao ocjene koje su korisnici dali te koristio neke varijante Pearsonove korelacije za izračun sličnih filmova. Uz to, postoji i dio sustava koji je u stvarnom vremenu izračunavao personalizirane preporuke temeljem ocjene koju je korisnik dao za film. Kako bi timovi mogli razvijati sustav Netflix je objavio i set podataka koji je sadržavao više od 100 milijuna anonimnih ocjena filmova. [12], [13]

U to vrijeme Netflix je filmove nudio na DVD-u i online, ali online gledanje filmova još nije bilo toliko popularno. Godinu nakon što su započeli natjecanje pokrenuli su i uslugu streaminga u trenutnom vremenu (engl. „instant streaming service“) što je uvelike promijenilo način interakcije korisnika sa sustavom i količinu podataka koje su prikupljali. Promijenila se i vrsta podataka koje je sustav mogao prikupljati o korisniku, kako su filmove gledali online sustav može vidjeti da li je korisnik pogledao cijeli film ili samo dio filma, da li je pogledao samo detalje filma (pretražio film) ili je taj film i pogledao. Što se tiče količine podataka, kako su korisnici mogli gledati film u bilo koje vrijeme i nisu trebali čekati da im dođe DVD, mogli su pogledati više filmova i samim time dati više ocjena. Još jednu veliku promjenu, nakon pokretanja streaminga u trenutnom vremenu, napravili su 2008. godine tako da su streaming s web stranice prebacili i na druge uređaje poput Androida i Apple-a te su napravili integraciju s Xbox-om. Samim time trebali su i kvalitetnije algoritme za preporuke te su ih morali prilagoditi novim promjenama. Uspjeli su postići da gotovo 75% sadržaja kojeg ljudi gledaju dolazi iz preporuka koje im nude. [12]

Što se tiče natjecanja, u 2009. godini dva tima uspjela su nadmašiti Cinematch za nešto više od traženih 10%. Timovi su se natjecali u rujnu 2009. godine i imali su identičan rezultat, ali je pobjedu odnio tim BellKor's Pragmatic Chaos jer su svoje rezultate predali 20 minuta ranije. Na idućoj slici vidljivo je najboljih 10 timova i njihovi rezultati u natjecanju. [14]

Leaderboard

Showing Test Score. Click here to show quiz score

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59

Slika 4: Netflix – ova nagrada, top 10 [15]

Tijekom natjecanja dvojica američkih znanstvenika uspjela su identificirati korisnike iz anonimnog seta podataka koji je Netflix objavio za natjecanje. Identificirali su ih tako da su povezivali podatke s drugim javno dostupnim podacima, točnije s podacima iz IMDb-ove baze podataka. S obzirom na to, nekolicina Netflix–ovih korisnika pokrenula je tužbu protiv njih zbog curenja privatnih podataka. Smatralo se da je zbog tužbe natjecanje i završilo ranije jer je, prema pravilniku, bilo predviđeno da traje 5 godina što znači da je trebalo završiti 2011. godine. Na posljetku, Netflix nije koristio pobjednički algoritma zbog promjena koje su krenule 2008. godine sa streamingom u stvarnom vremenu. Uz to naveli su da algoritam koji je pobijedio u natjecanju nije bio u potpunosti usavršen za produkciju. [14]

Možemo uočiti kolika je važnost sustava za preporuke iz samog iznosa nagrade koju je Netflix nudio za bolji algoritam. Ako uzmemo u obzir postotak sadržaja, u iznosu od iznosi 75%, koji ljudi gledaju temeljem preporuka zaključujemo da su takvi sustavi ključni za uspjeh poslovanja. Naime, Netflix svoj profit ostvaruje na temelju pretplata i cilj im je korisnike zadržati što duže. Kako bi ih zadržali moraju im pružiti sadržaj koji će ih zainteresirati da ostanu za što je zadužen sustav za preporuke. Ukoliko si pokušamo zamisliti Netflix bez sustava za preporuke gdje bi korisnik sam morao pretraživati sadržaj koji je prikazan na identičan način, na primjer abecedno sortiran, tada je vrlo velika vjerojatnost da će korisnik pogledati ono što zna da ga interesira i prestati koristiti servis ili aplikaciju. Problem je u tome da bi korisnik tada imao dostupan sav sadržaj ili proizvode koje bi bilo vrlo teško pretraživati. S druge strane sustavi za preporuke prate što korisnik voli i kakav sadržaj gleda te mu na temelju toga ponude i druge proizvode koji bi ga mogli zanimati i za koje korisnik niti ne zna da postoje te ga na taj

način uspiju i zadržati. Netflix je samo jedan primjer sustava u kojem sustavi za preporuke mogu drastično unaprijediti poslovanje, pa čak i ako nisu korišteni najbolji algoritmi.

2.3. Algoritmi sustava za preporuke

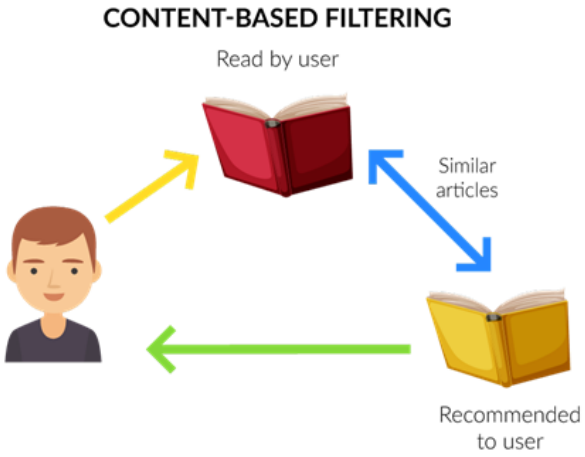
Sustavi za preporuke rade na temelju dva tipa informacija. Prvi tip su informacije o proizvodima i korisnicima kao što su karakteristike proizvoda koje korisnik pretražuje, ključne riječi i kategorije u koje spadaju proizvodi te preferencije korisnika. Drugi tip informacija predstavlja interakcije korisnika i proizvoda u vidu ocjenjivanja proizvoda, stavljanja na listu želja i kupnje proizvoda. Temeljem tih informacija postoje tri algoritma ili metode koje se koriste u sustavima za preporuke i to su sustavi za preporuke na temelju sadržaja (engl. „Contetn – based systems“), sustavi za preporuke na temelju kolaborativnog filtriranja (engl. „Collaborative filtering systems“) i hibridni sustavi za preporuke (engl. „Hybrid systems“) koji su zapravo kombinacija prethodna dva sustava te rade s oba tipa informacija s ciljem da se izbjegnu problemi koje pojedini algoritam posjeduje. [16]

U nekoliko idućih potpoglavlja bit će objašnjen pojedini algoritam sustava za preporuke uz pripadne primjere vezane uz preporuku knjiga te prednosti i nedostatke pojedinog algoritma ili metode. Što se tiče sličnosti (engl. „similarity“) i konkretnih algoritama, kao što su Pearsonov koeficijent sličnost i k najbližih susjeda (engl. „k Nearest Neighbors“), oni će biti detaljnije objašnjeni u praktičnom dijelu rada uz implementaciju sustava za preporuke.

2.3.1. Sustavi za preporuke na temelju sadržaja

Ovaj algoritam koristi informacije o proizvodima i korisnicima za kreiranje preporuka. Ako za primjer uzmemo knjigu tada ona može imati kategorije kao što je žanr i autor, dok neki drugi proizvodi, kao što je auto, mogu imati tip motora, marku, model, oblik karoserije i drugo. Ovakvi sustavi temelje se na pretpostavci da će korisnik u budućnosti biti zainteresiran za proizvode koji ga trenutno interesiraju, kao i za proizvode koji su ga ranije interesirali. Vidimo da se ovaj algoritam u potpunosti oslanja na podatke dostupne o proizvodu i korisniku, tj. nisu potrebni podaci o drugim korisnicima sustava. [16]

Sljedeća slika prikazuje jedan primjer preporuke na temelju sadržaja, korisnik je pročitao knjigu koja spada u neku kategoriju te na temelju toga sustav pronalazi drugu knjigu koja spada u istu kategoriju i korisniku preporučuje tu knjigu.



Slika 5: Filtriranje na temelju sadržaja [17]

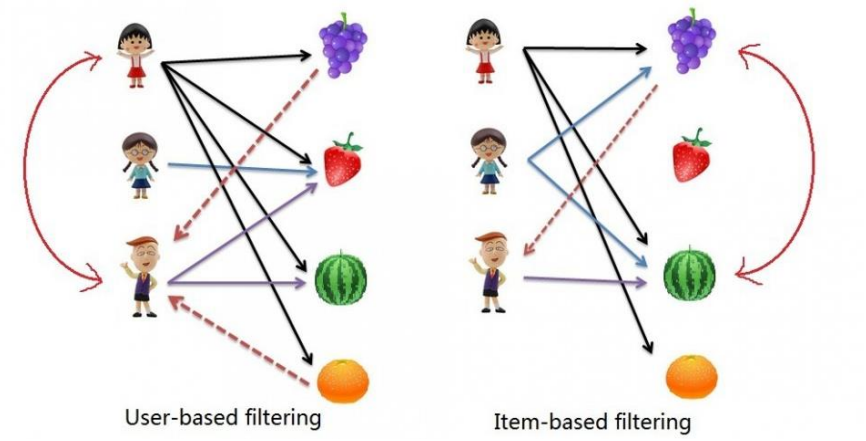
Jedna od glavnih prednosti sustava za preporuke na temelju sadržaja je da imaju mogućnost preporuke svih proizvoda pa čak i onih koji još nisu ocijenjeni niti od jednog korisnika. Ostale prednosti su da se novi proizvodi mogu vrlo lako naći u preporukama, preporuke mogu početi davati gotovo odmah (nakon što korisnik ostavi prvu ocjenu ili kod pregleda proizvoda) i veća je sigurnost jer algoritam ne treba podatke o drugim korisnicima za davanje preporuka. S druge strane najveći nedostatak im je to da ovise o podacima o proizvodu, što znači da svaki proizvod mora biti detaljno opisan i sustavu moraju biti dostupni svi podaci o proizvodu. Postoji još jedan nedostatak i to je da tako implementiran algoritam ne može znati sve što korisnik voli. Na primjer korisnik trenutno gleda ili je pročitao knjigu koja spada u žanr kriminalistike i tada sustav korisniku preporučuje samo knjige koje spadaju u taj žanr, ali ne zna da korisnik voli i fikciju. Sustav će korisniku moći preporučivati i knjige koje spadaju u žanr fikcije tek nakon što korisnik pročita neku takvu knjigu. Kao rješenje navedenog problema može se korisnika prilikom registracije pitati da odabere svoje omiljene žanrove i autore kako bi sustav imao potrebne informacije za relevantnije i kvalitetnije preporuke, a kasnije će sustav prikupiti nove podatke na temelju interakcije korisnika sa sustavom te će ih koristiti za davanje kvalitetnijih preporuka. Može se desiti da korisnik s vremenom promijeni svoje interese i stoga je vrlo važno da se prikupljaju novi podaci. [2, str. 282], [6], [18], [19]

Ako se prisjetimo problema hladnog starta možemo vidjeti da ovaj algoritam u nekoj mjeri rješava taj problem. Naime, za davanje preporuke nisu potrebni podaci o korisniku već samo o proizvodu, iako bi s podacima o korisniku preporuka bila personaliziranija. Za primjer, ovim algoritmom može se implementirati rješenje da se na stranici detalja pojedine knjige preporučuju druge knjige koje je napisao isti autor i koje spadaju u isti žanr. Naravno bilo bi

dobro da se doda i mjera sličnosti kako bi se na vrhu preporuka nalazile najsličnije knjige, a one malo manje slične na dnu.

2.3.2. Sustavi za preporuke na temelju kolaborativnog filtriranja

Drugi tip informacija su interakcije korisnika i proizvoda i ovaj algoritam koristi upravo takve informacije, što znači da koristi implicitnu ili eksplicitnu ocjenu korisnika za kreiranje personaliziranih preporuka. Preporuke se kreiraju na način da se pronađu drugi korisnici koji imaju identične interese našima, ali su kupili i ocijenili proizvode koje mi još nismo i sustav nam tada preporučuje upravo te proizvode. Sama ideja ovog algoritma leži na tome da ljudi, tj. korisnici pomažu jedni drugima tako da se pronađu slični korisnici koji tada čine malu zajednicu koja ima identične interese. Takvo filtriranje na temelju susjedstva može se podijeliti na dva tipa, temeljeno na korisniku i temeljeno na proizvodu. [2, stt. 181-184], [6], [18]



Slika 6: Kolaborativno filtriranje [20]

Prethodna slika prikazuje oba tipa filtriranja na temelju susjedstva. Možemo uočiti da su svi korisnici ocijenili ili kupili neki proizvod što je ključno za sustave za preporuke na temelju kolaborativnog filtriranja jer se preporuke daju na temelju ocjena.

Vratimo se na primjer s knjigama, filtriranje temeljeno na korisniku radi na principu da se pronađu korisnici koji imaju slične interese našima (pročitali su i ocijenili neke knjige koje smo i mi) i tada nam sustav predlaže one knjige koje su ti korisnici pročitali, a mi još nismo. U tom slučaju, kao preporuku, ne dobivamo najbolje ocijenjene proizvode već nam sustav preporučuje one proizvode koje su korisnici slični nama pročitali. Cilj ove metode je da se pronađe k–najbližih susjeda i preporuče oni proizvodi koji su najpopularniji među susjedima i

koje mi još nismo kupili ili ocijenili. Korištenjem ovog pristupa moramo voditi računa o tome kako pronalazimo slične korisnike. Kao primjer možemo uzeti korisnika koji je pročitao prva tri dijela serijala knjiga Harry Potter i dao im ocjenu 5. Uz to uzmimo još dva korisnika koja će imati slične interese, recimo da je jedan od njih pročitao prvu knjigu serijala Harry Potter i dao joj ocjenu 5 te je uz to pročitao i jednu knjigu od autora George R. R. Martina, a drugi korisnik je pročitao prva 4 dijela serijala knjiga Harry Potter i prva dva dijela ocijenio s ocjenom 5, a treći dio s ocjenom 4. Ako za izračun sličnosti ne uzmemo u obzir sve parametre kao što su broj knjiga koje su korisnici pročitali i ocjene koje su dali tada se može desiti da će se prvi korisnik u potpunosti podudarati s korisnikom kojem dajemo preporuku jer imaju samo jednu zajedničku knjigu koji su identično ocijenili, iako s drugim korisnikom ima 3 zajedničke knjige od kojih se u jednoj ne slažu, u potpunosti, s ocjenom. Takav izračun sličnosti rezultirat će lošijim preporukama jer nismo u obzir uzeli količinu zajedničkog sadržaja koji se sviđa obojici korisnika i sustav će prednosti dati knjizi od autora George R. R. Martina u odnosu na četvrti dio serijala knjiga Harry Potter. Isto se može desiti ako u obzir uzmemo samo količinu ili broj knjiga koje je pojedini korisnik procijenio, a zanemarimo ocjenu koju je dao. Recimo da je drugi korisnik pročitao prva 4 dijela serijala knjiga Harry Potter, ali im je dao ocjenu 1. U tom slučaju se može desiti da su ti korisnici sličniji jer su pročitali i ocijenili 3 zajedničke knjige, ali ako uzmemo u obzir i ocjene tada to nije slučaj. S obzirom na to, potrebno je uzeti u obzir što više faktora prilikom izračuna i pronalaženja sličnih korisnika kako bi preporuke bile kvalitetnije. [2, str. 184], [4], [6]

Algoritam filtriranja temeljen na proizvodu uzima u obzir pozitivne ocjene korisnika te daje preporuke na temelju sličnosti proizvoda. Dva proizvoda možemo smatrati sličnima ukoliko je većina korisnika imala interakciju s oba proizvoda na identičan način, ako se gleda sustav ocjenjivanja da su oba korisnika dala identičnu ocjenu za oba proizvoda, uz to gledaju se i neki od atributa proizvoda kao što je kategorija u koju proizvod spada. Ovaj tip preporuka u obzir uzima one proizvode koje je korisnik najbolje ocijenio te preporučuje proizvode koji su najbliži ili najsličniji tim proizvodima. [2, str. 184], [4], [6]

Najveći nedostatak sustava za preporuke na temelju kolaborativnog filtriranja je navedeni problem hladnog starta. Ovom algoritmu potrebni su podaci kako bi mogao pružati relevantne preporuke i ako korisnik još nije imao interakciju sa sustavom algoritam ne može pružati dobre preporuke. Isto tako ako uvedemo novi proizvod kojeg još nitko nije kupio niti ocijenio ne može se znati da li je taj proizvod dobar i ovaj algoritam ga neće smatrati dobrim za preporuku. Drugi problem koji se javlja je količina proizvoda koje korisnici ocjenjuju što znači da će algoritam gotovo uvijek pronaći najpopularnije proizvode kao dobre preporuke jer takvi proizvodi uglavnom imaju najviše ocjena. Većina korisnika ocijeni svega par proizvoda i tada je teško naći susjede zbog manjka podataka koji su ovom algoritmu potrebni. Još jedan

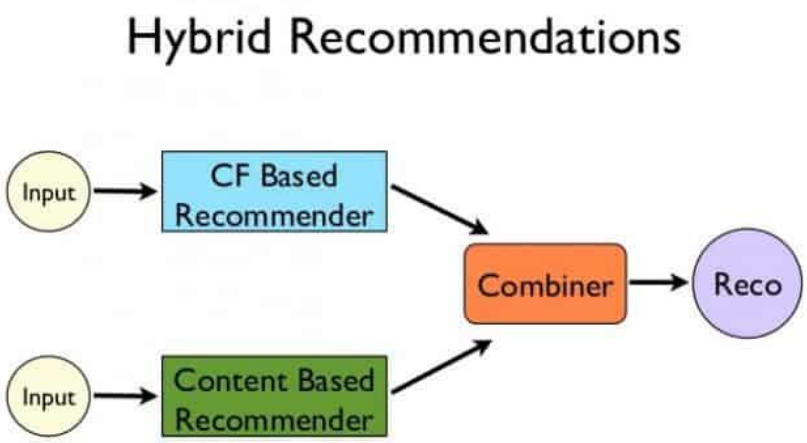
problem koji se može javiti je povezan sa skalabilnošću sustava. Budući da je potrebno pronaći najbliže susjede koji imaju slične interese, ovaj algoritam može postati spor kako se poveća količina podataka potrebna za obradu. Što se tiče prednosti, ovaj algoritam daje preporuke na temelju interakcije svih korisnika sa sustavom što znači da nisu potrebni detaljni podaci o proizvodu i korisniku kao što je to slučaj sa sustavima za preporuke na temelju sadržaja. S obzirom na to, sustav pronalazi susjede koji imaju identične interese našima te na temelju toga kreira relevantne preporuke tako da pronađe one proizvode koji se sviđaju našim susjedima, a da ih mi još nismo ocijenili, kupili ili na neki način konzumirali. [2, str. 209 i 210], [6], [18], [19]

2.3.3. Hibridni sustavi za preporuke

Hibridni sustavi za preporuke, kao što govori i sam naziv, su kombinacija prethodna dva sustava za preporuke. Svrha hibridnih sustava je da se uzmu prednosti pojedinog sustava te se kombiniraju kako bi se dobile bolje preporuke i kako bi se smanjile pogreške, tj. zaobišli nedostaci pojedinog sustava. Znamo da sustavi za preporuke na temelju kolaborativnog filtriranja pate od problema hladnog starta te je u tom slučaju bolje koristiti sustave za preporuke na temelju sadržaja sve dok sustav ne prikupi više informacija o korisnicima preko interakcija sa sustavom. Nakon što sustav ima dovoljno informacija bolje je koristiti sustave za preporuke na temelju kolaborativnog filtriranja jer daju personaliziranije preporuke. Naravno nije ideja da se koristi samo jedan od algoritama nego ih je potrebno na neki način kombinirati, recimo da se algoritmi implementiraju zasebno te se na kraju kombiniraju rezultati pojedinog algoritma ili da se kombiniraju algoritmi jedan unutar drugog. [8], [18], [19]

Provedena su istraživanja usporedbe hibridnog pristupa s filtriranjem na temelju sadržaja i kolaborativnim filtriranjem zasebno i dokazano je da hibridni sustavi za preporuke korisnicima daju relevantnije preporuke i uspješno rješavaju neke od problema koje pojedini sustav posjeduje. [21]

Iduća slika prikazuje kako se kreiraju preporuke korištenjem hibridnog sustava koji funkcionira tako da se kombiniraju rezultati algoritama sustava za preporuke na temelju sadržaja i sustava za preporuke na temelju kolaborativnog filtriranja.



Slika 7: Hibridni sustavi za preporuke [21]

Netflix je naveden kao jedan vrlo dobar primjer hibridnih sustava za preporuke. Naime, oni za kreiranje preporuka kombiniraju podatke prikupljene od korisnika temeljem gledanja i pretraživanja filmova te slične korisnike grupiraju u zajednicu (sustavi za preporuke na temelju kolaborativnog filtriranja) i uz to koriste podatke o filmovima koje je korisnik dobro ocijenio te preporučuju one filmove koji imaju slične karakteristike tim filmovima (sustavi za preporuke na temelju sadržaja). [21]

2.4. Primjeri uspješnih sustava za preporuke

U današnje vrijeme se većina proizvoda pretražuje i kupuje online i stoga je vrlo važno da se na neki način privuku kupci te da se isti i zadrže. Sustavi za preporuke su osmišljeni upravo zbog toga i davanje relevantnih preporuka korisniku je najbolji način za unaprjeđenje prodaje i poboljšanje korisničkog iskustva. Vrlo je važno da sustav za preporuke bude kvalitetan i da korisniku ne nudi proizvode koje je već kupio ili ocijenio ili one proizvode koji su slični proizvodima koji se korisniku nisu svidjeli (ocijenio ih je lošom ocjenom). [22]

Jedan od najvećih primjera je Netflix koji je spomenut i ranije u radu, 75% sadržaja koji ljudi gledaju na Netflix-u dolazi iz preporuka koje sustav kreira za korisnike. Isto tako, Netflix je naveo da im sustavi za preporuke uštede oko milijardu dolara godišnje jer uspijevaju zadržati postojeće pretplatnike te ne moraju pronalaziti nove da bi ih zamijenili. [16], [23]

Drugu primjer je YouTube, to je mreža na kojoj korisnici mogu besplatno pregledavati i postavljati razne videozapise. YouTube svojim sustavom za preporuke uspijeva privući korisnike tako da više od 70% vremena koje provedu gledajući video materijale dolazi iz

preporuka. Sustavi za preporuke su ključni za opstanak YouTube servisa jer korisnici tako odmah nalaze sadržaj koji im je potencijalno interesantan i koji će gledati. [23], [24]

Amazon je jedan od primjera online dućana koji 35% svojih proizvoda proda kupcima preko sustava za preporuke. 35% je vrlo velik postotak s obzirom na veličinu dućana i količinu proizvoda koje prodaje i time možemo vidjeti koliki profit im donosi sustav za preporuke. Sustav se uglavnom bazira na kolaborativnom filtriranju te daje preporuke prema tome što su drugi korisnici kupili uz proizvode koji nam se trenutno nalaze u košarici. Osim Amazona, sustavi za preporuke unaprijedili su poslovanje i drugih dućana kao što su eBay i Walmart. [16], [21], [25]

Još jedan primjer u kojem su sustavi za preporuke unaprijedili poslovanje je Spotify, servis za slušanje glazbe. Slično kao i Netflix, oni svoj profit ostvaruju pretplatama korisnika. Sustavi za preporuke pomažu korisnicima da pronađu glazbu koju još nisu slušali, a odgovara njihovim interesima. Samim time uspijevaju zadržati pretplatnike čime ostvaruju profit. Uglavnom se koriste sustavi za preporuke na temelju kolaborativnog filtriranja, ali i oni temeljeni na sadržaju. Spotify je uveo i analizu audio zapisa. Analizom karakteristika kao što su tempo glazbe, glasnoća i vrijeme trajanja pronalazi sličnu glazbu koju tada preporučuje korisnicima. [25]

3. Grafovske baze podataka

Grafovska baza podataka jedna je vrsta NoSQL (engl. „Not only SQL“ ili „No to SQL“) ili polustrukturiranih baza podataka. Polustrukturirani model podataka razvijen je kako bi se mogli prikazati heterogeni podaci koju su često kompleksni i nepotpuni. Potreba za ovakvim modelom podataka nastala je razvojem World Wide Weba putem kojeg se prenose velike količine raznih podataka. Prednost polustrukturiranog modela podataka je to što ima dinamičnu shemu te se često karakterizira i kao samo–opisujući model podataka, takav model podataka je vrlo fleksibilan te omogućuje lakšu prenosivost informacija. [26, str. 245]

3.1. NoSQL baze podataka

Edgar Frank Codd tvorac je relacijskog modela podataka i nakon objave rada “A relational model of data for large shared data banks“ 1970. godine, sustavi za upravljanje relacijskom bazom podataka (SURBP), koji se zasnivaju na relacijskom modelu podataka, postali su glavna tehnologija za pohranu podataka. S vremenom su SURBP postali dominanta tehnologija za pohranu strukturiranih podataka te se smatralo da ne postoji alternativa za relacijske baze podataka. Samim time težilo se da se SURBP koriste za sve, „jedna veličina odgovara svima“ (engl. „one size fits all“), ta se težnja s vremenom počela propitkivati te su nastale razne alternative baze podataka poznate pod nazivom NoSQL. [27, str. 1]

NoSQL baze podataka postale su popularne kasnih 2000-ih godina kako je tehnologija napredovala i pali su troškovi pohrane podataka. Samim time porasla je i količina podataka koju aplikacije prikupljaju te oni mogu biti strukturirani i polustrukturirani za što je gotovo nemoguće kreirati relacijsku shemu niti relacijske baze podataka mogu zadovoljiti sve potrebe modernih aplikacija. Kao odgovor na razne vrste podataka i dinamičnost koja je potrebna u modernim aplikacijama nastale su NoSQL baze podataka. Često se smatra da NoSQL baze podataka ne mogu spremati strukturirane podatke što naravno nije točno. U NoSQL baze podataka mogu se spremiti strukturirani i nestrukturirani podaci, samo što se strukturirani podaci spremanju na malo drugačiji način nego u relacijskim bazama podataka koje je većina navikla koristiti. [28]

Uz pojam NoSQL baza podataka često se spominje i skraćenica 3V koja dolazi od eng. Volume, Velocity i Variety. To su zapravo tri izazova s kojima se susreću relacijske baze podataka i koji se rješavaju korištenjem NoSQL baza podataka. Obujam (engl. „Volume“) predstavlja velike količine podataka koje je potrebno pohraniti. Takvi veliki skupovi podataka postaju nezgrapni za spremanje u relacijske baze podataka jer tada one više ne daju

zadovoljavajuće performanse. JOIN je vrlo skupa operacija u SURBP i vrijeme izvršavanja raste kako raste i količina podataka u bazi. NoSQL baze podataka imaju bolje performanse u radu s velikim skupovima podataka, ali većina ih nije toliko izražajna kao relacijski model, osim modela grafa u grafovskim bazama podataka koji je zapravo izražajniji od relacijskom modela. Drugi V označava brzinu rasta podataka kroz vrijeme (engl. „Velocity“). U današnje vrijeme podaci se brzo mijenjaju i brzo nastaju novi, što znači da je ih je potrebno vrlo brzo obraditi i vršiti puno ažuriranja u bazi podataka za što relacijske baze podataka nisu dizajnirane. Uz brzu promjenu i rast podataka česte su i promjene u strukturi samih podataka čime se mijenja model podataka. Budući da relacijske baze podataka imaju fiksnu shemu i nisu dizajnirane da podrže neprekidna ažuriranja, imat ćemo velike operativne troškove i troškove procesiranja pokušavajući riješiti navedene izazove. NoSQL baze podataka su optimizirane za učestala ažuriranja i posjeduju fleksibilnu shemu ili model podataka čime uspješno rješavaju problem velikog rasta podataka. Posljednje V označava raznolikost podataka (engl. „Variety“), podaci ne moraju uvijek biti potpuni, mogu biti strukturirani i nestrukturirani te povezani i nepovezani. Sve je to posljedica raznih sustava i tehnologija iz kojih dolaze podaci i upravo zbog toga današnji SURBP imaju puno NULL vrijednosti u tablicama. S druge strane, NoSQL baze podataka zbog svoje fleksibilne sheme rješavaju navedeni problem i novi podaci se mogu vrlo lako dodati. [29. str 29 i 30], [30, str. 229]

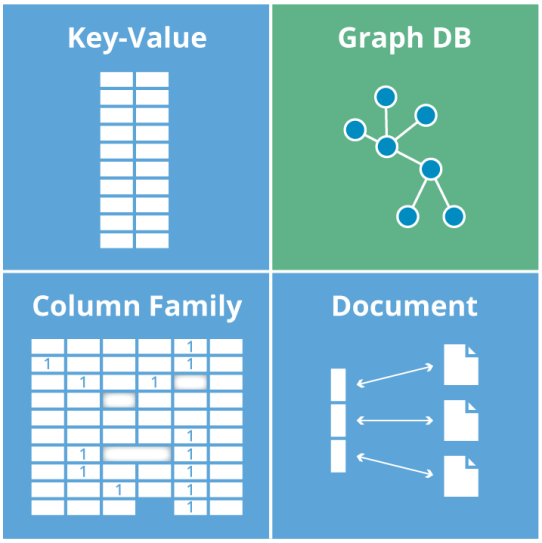
Iz navedenih izazova s kojima se susreću relacijske baze podataka možemo vidjeti da NoSQL baze podataka zbog svoje fleksibilnosti i mogućnosti rada s velikim skupovima podataka imaju prednost nad relacijskim bazama podataka. To ne znači da više ne trebamo koristiti relacijske baze podataka, već nam NoSQL baze podataka daju nove mogućnosti za pohranu podataka. U nekoliko idućih poglavlja bit će navedene vrste NoSQL baza podatka te koje su njihove prednosti i nedostaci kako bi mogli lakše odlučiti koji tip baze podataka najviše odgovara podacima s kojima radimo i aplikacijskoj domeni. Uz to bit će i posebno poglavlje posvećeno usporedbi relacijskih i NoSQL baza podataka.

3.1.1. Vrste NoSQL baza podataka

Pojavom NoSQL baza podataka više nismo ograničeni na korištenje relacijskih baza podataka, već imamo i druge opcije koje se pokazuju boljima za određene vrste podataka, točnije za polustrukturirane i nestrukturirane podatke. Nakon dugog niza godina prevladavanja relacijskih baza podataka javili su se i brojni izazovi s kojima se one susreću te su uglavnom povezani sa skraćenicom 3V. Radi se o problemima vezanim uz količinu podataka, skalabilnosti sustava, performansama, raznolikost podataka koje je potrebno pohraniti i brze promjene podataka i njihove strukture. Svi problemi proizlaze razvojem tehnologije i potrebama modernih aplikacija u čemu veliku ulogu igra društveni segment modernog Weba.

SURBP jednostavno nisu dovoljno efikasni da bi se ispunile takve specifične potrebe te se kao rješenje koriste razne vrste NoSQL baza podataka.

Razvila su se četiri glavna tipa NoSQL baza podataka: ključ-vrijednost baze podataka, stupčane baze podataka, dokument baze podataka i grafovske baze podataka. Iduća slika prikazuje navedene tipove NoSQL baza podatka.

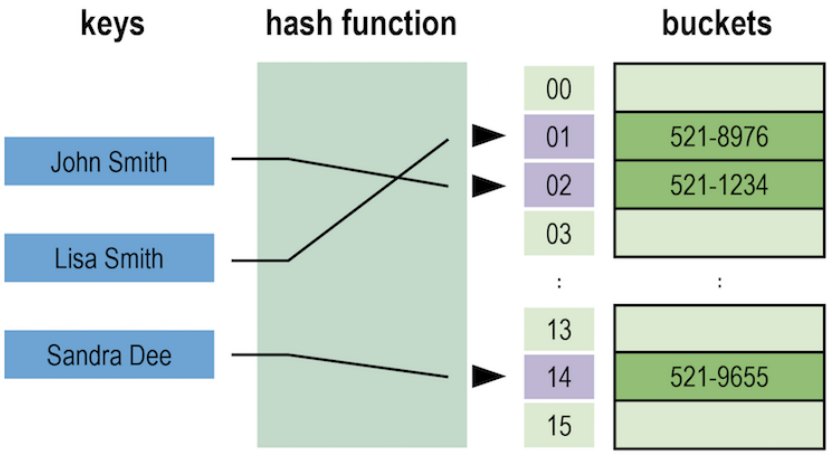


Slika 8: NoSQL baze podataka [29, str. 33]

U idućim potpoglavljima ukratko su objašnjena tri tipa NoSQL baza podataka koja su označena plavom bojom na slici, dok će grafovske baze podataka biti detaljnije objašnjene u zasebnom poglavlju rada.

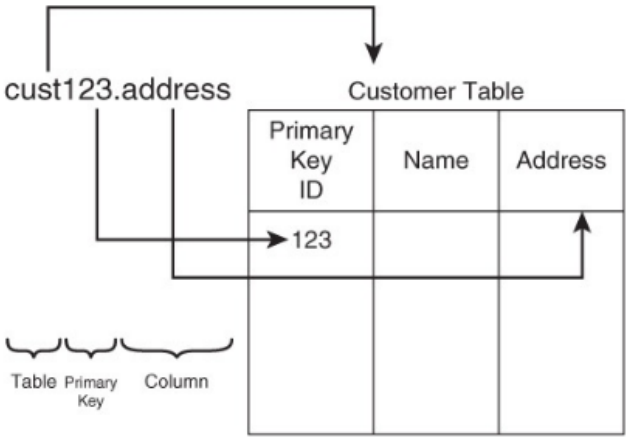
3.1.1.1. Ključ – vrijednost baze podataka

Ključ-vrijednost (engl. „Key-Value“) baze podataka smatraju se najjednostavnijim tipom NoSQL baza podataka. Podaci se pohranjuju kao parovi oblika ključ–vrijednost te se sastoje od naziva atributa i njegove vrijednosti. Ova vrsta baze podataka uspoređuje se sa strukturom podataka hash mape unutar koje su podaci organizirani prema vrijednosti ključa. Vrijednosti se iz baze dohvaćaju preko ključa tako da je za dohvat potrebno poznavati ključ ili način na koji se on izračunava. Ključ zapravo predstavlja identifikator za vrijednost koja je pohranjena. Vrijednost mogu biti neki jednostavni tipovi podataka kao što su string (niz znakova) i broj, ali se može pohraniti i složeni tip ili struktura podataka kao što je lista, skup i slika kao BLOB (engl. „Binary Large Object“) tip podataka. [27, str. 52], [28], [29, str. 33 i 34], [30, str. 230], [31, str. 68-71]



Slika 9: Ključ-vrijdnsot baza podataka [29, str. 33]

Prethodna slika prikazuje osnovni princip spremanja podataka u ključ-vrijednost bazu. Vidimo da postoji više kolekcija ili blokova (engl. „buckets“) u koje se spremaju vrijednosti i oni se mogu replicirati na više računala ili servera kako bi se postigla veća sigurnost podataka. Na taj način se može i uravnotežiti opterećenje, tako da se dostupni serveri međusobno optimiziraju. Naravno podaci se ne repliciraju na servere tako da se rade identične kopije već se na svakom nalazi dio podataka. Preko blokova ili kolekcija zapravo se kreira imenski prostor (engl. „Namespace“) unutar kojeg se spremaju podaci i time možemo stvoriti privid relacijske sheme. Ukoliko se držimo neke konvencije imenovanja ključa tada ga možemo oblikovati tako da odgovara nazivu tablice, primarnog ključa i stupca i relacijskoj shemi. Recimo da imamo entitet ili tablicu naziva korisnik koja ima atribut ID kao primarni ključ te uz to i atribute ime i adresa. Uzevši to u obzir ključ možemo oblikovati kao „korisnik1.ime“ ili „korisnik1.adresa“, u tom slučaju referenciramo se na korisnika koji ima ID 1 te mu dodjeljujemo ime i adresu kao vrijednost pojedinog ključa. Primjer takve konvencije naziva ključa vidljiv je na idućoj slici. [29, str. 34], [31, str. 72]



Slika 10: Konvencija imenovanja ključa prema relacijskoj shemi [31, str. 72]

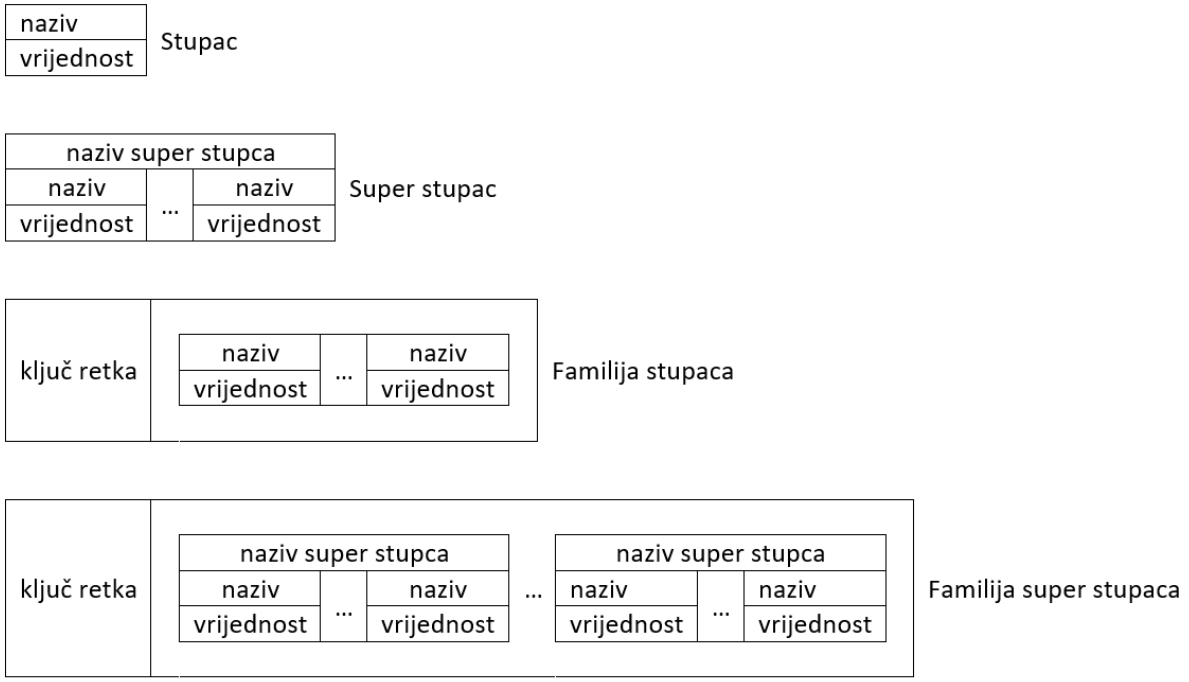
Glavne prednosti ključ-vrijednost baza podataka su to što su optimizirane za rad s velikom količinom podataka te su podaci lako dostupni. Iako su podaci dostupni poznavanjem ključa, nedostatak je to što se uglavnom dohvaća cijela vrijednost te se mora naknadno filtrirati kako bi se maknuo dio podataka koji nam nije potreban. Isto tako ove baze rade s nepovezanim podacima i teško je uočiti relacije između podataka. S obzirom na to, ako želimo dohvatiti skup informacija koji je pohranjen u nekoliko zapisa (ključ-vrijednost) tada moramo provesti obradu tih informacija za što se koriste algoritmi kao što je MapReduce. Za takve obrade potrebni su određeni resursi i ukoliko se radi o nekom većem setu podataka trebat će nešto više vremena da se dohvate željeni podaci. [29, str. 34]

Ključ–vrijednost baze podataka su skalabilne, što znači da zadržavaju svoje performanse kako raste količina podataka te su optimizirane za rad s velikom količinom podataka. Uzevši to u obzir, ove baze su dobre ako je potrebno spremiti velike količine podataka koji su lako i brzo dostupni uz uvjet da nam nisu potrebni složeni upiti nad podacima koji se nalaze u više zapisa. Primjeri popularnih ključ–vrijednost baza podataka su: Redis, Riak, Memcached, Amazon DynamoDB i druge. [28]

3.1.1.2. Stupčane baze podataka

Stupčane (engl. „Column Family“) baze podataka, što se tiče njihove strukture, smatraju se najkompleksnijim tipom NoSQL baza podataka. Podaci se pohranjuju u tablice, redove i stupce koji su dinamički, što znači da svaki red ne mora imati isti broj stupaca (struktura je varijabilna) te mnogi smatraju da su one zapravo dvodimenzionalne ključ–vrijednost baze podataka. Glavna jedinica pohrane je stupac koji sadrži ime (ključ) i vrijednost i takav stupac se tada pohranjuje u red koji sadrži svoj jedinstveni ključ. Taj red, kao vrijednost sadrži N prethodno navedenih stupaca koji tada čine familiju stupaca (engl.“Column Family“). Nadalje, više stupaca se može grupirati u tzv. super stupac i ako neki redak sadrži takav super stupac tada ga nazivamo familija super stupaca (engl. „Super Column Family“). [27, str. 104], [28], [29, str. 34 i 35], [30, str. 230], [31, str. 74 i 75]

Iduća slika prikazuje blokove od kojih se sastoje stupčane baze podataka i na familiji stupaca te familiji super stupaca je vidljivo zašto se one smatraju dvodimenzionalnim ključ–vrijednost bazama podataka.

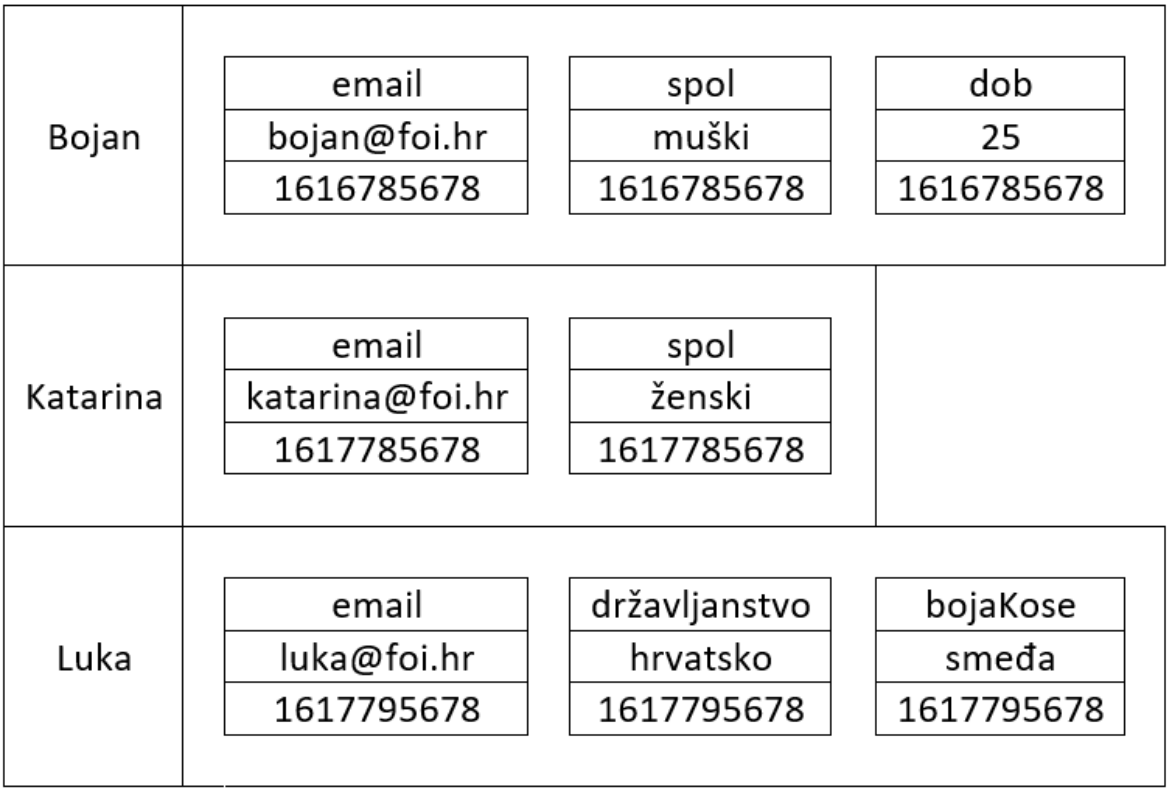


Slika 11: Blokovi od kojih se sastoje stupčane baze podataka (prema [29, str. 34])

Možemo uočiti da se podaci spremaju u stupce unutar nekog retka i taj redak kao vrijednosti u familiji stupaca sadrži sve podatke o pojedinoj instanci entiteta ili objektu. S druge strane u relacijskim bazama podataka, podaci o objektu se nalaze u retku unutar tablice i mogu biti pohranjeni u više tablica te se koristi JOIN kako bi se ti podaci objedinili. Uz to, važno je napomenuti da ove baze podataka imaju, za svaki stupac, pohranjenu i vremensku komponentu (engl. „Timestamp“) kako bi se moglo pohraniti više verzija podataka. Sustav sam spremi datum i vrijeme kada je podatak pohranjen ili ažuriran kako bi se mogla odrediti posljednja verzija podatka te dohvatiti starije verzije ako su potrebne. Uglavnom sustav posjeduje parametar koji određuje broj verzija koje se čuvaju i ne preporučuje se da taj broj bude velik jer će trebati puno resursa kako bi se održali svi podaci. Ovo je jedna od prednosti koja čini stupčane baze podataka vrlo popularnima. Slično kao i ključ–vrijednost baze podataka, namijenjene su za rad s velikom količinom podataka i imaju nešto ekspresivniji model podataka jer se svi podaci o objektu nalaze u jednom redu koji ima više stupaca. Isto tako možemo lakše dohvatiti određeni podatak tako da se referenciramo na stupac koji trebamo bez potrebe za filtriranjem nepotrebnih podataka. Ostale prednosti ovih baza su vrlo lako dodavanje novih stupaca zbog fleksibilnosti sheme, vrlo su efikasne kod kompresije podataka što uvelike ubrzava dohvat, tj. čitanje, zbog svoje strukture omogućuju brzo izvršavanje agregirajućih operacija poput izračuna sume, prosjeka, pobrojenja i drugih te su

lako skalabilne što znači da su namijenjene za paralelno procesiranje (spremanje podataka na više računala ili servera). [28], [29, str. 35 i 36], [30, str. 230 i 236], [31, str. 76], [32]

Na slici 12 prikazan jer primjer stupčane baze podataka koja se sastoji od familije stupaca. Sa slike je vidljivo da su podaci spremljeni u stupce od kojih svaki ima svoj naziv, vrijednost i vremensku komponentu koju sustav zapisuje u trenutku unosa ili promjene podatka u stupcu. Isto tako možemo uočiti da je struktura, tj. shema varijabilna jer pojedini red ima svoju familiju stupaca koja se djelomično razlikuje od ostalih redaka.



Slika 12: Primjer familije stupaca (prema [32])

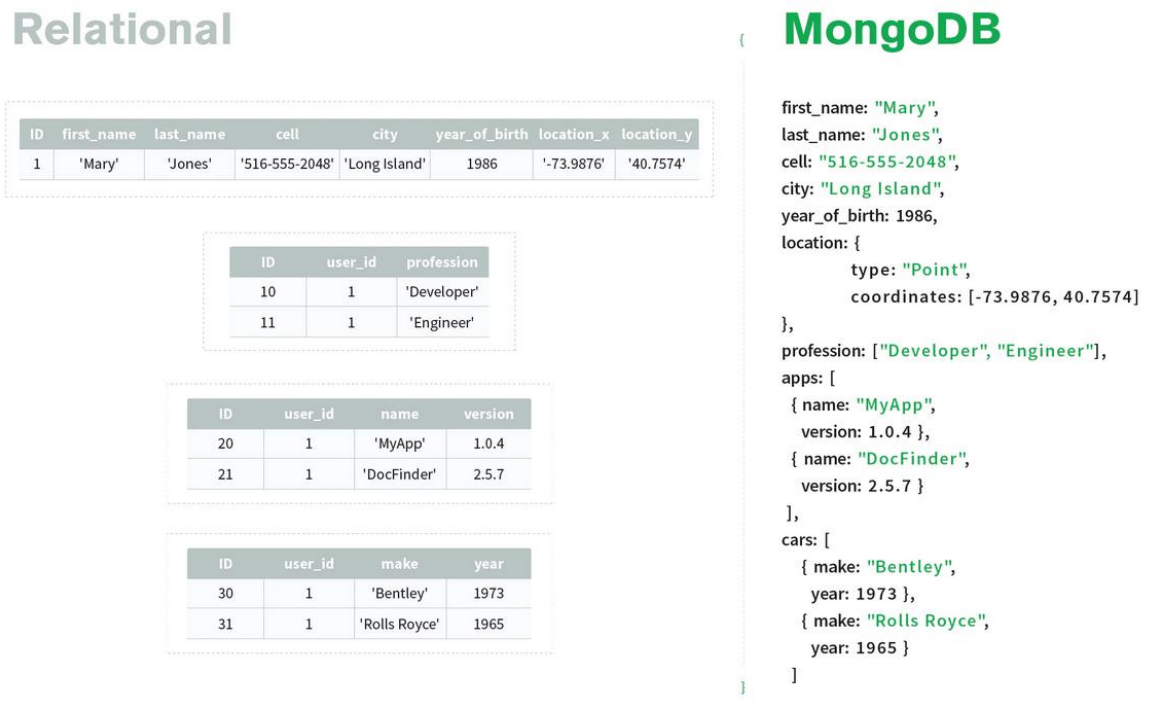
Ako uzmemo u obzir navedene prednosti stupčanih baza podataka, one se preporučuju za korištenje ako je potrebno spremiti velike količine podatka i ako znamo otprilike koje podatke ćemo dohvaćati kako bi organizirali stupce na pravilan način. Isto tako, zbog svoje strukture i mogućnosti masivnog paralelnog procesiranja mogu se koristiti za izgradnju aplikacija kod kojih su potrebne visoke performanse. Najpopularnije stupčane baze podataka su Cassandra i HBase, a drugi primjeri su Bigtable, Accumulo, Elassandra i druge. [27 str. 104-114], [28]

3.1.1.3. Dokument baze podataka

Dokument (engl. „Document“) baze koriste, već dobro poznat, ključ–vrijednost pristup za pohranu podataka te podatke pohranjuju kao dokumente. Ovdje se ne radi o Word, Google ili PDF dokumentima, već je riječ o JSON, BSON, YAML i XML dokumentima. Dokument je u ovom slučaju homonimni koncept koji ima različita značenja. [30, str. 230], [31, str. 72], [33], [34, str. 40]

Koncept se sastoji od tri komponente: simbola, intenzije i ekstenzije te ga definiramo kao K = (simbol, intenzija, ekstenzija), gdje je simbol oznaka ili naziv koncepta, intenzija definicija koncepta i ekstenzija skup primjera za koncept. Prema tome homonimni koncept definiramo kao Kh = (s, (i1, i2), (e1, e2)), gdje je i1 ≠ i2 i e1 ≠ e2. Temeljem toga zaključujemo da prethodno naveden koncept ima isti naziv „dokument“, ali različitu definiciju i skup primjera. [30, str. 205 - 207]

Unutar dokumenta, kao vrijednost, mogu se pohraniti razne vrste podataka od jednostavnih stringova, brojeva i boolean vrijednosti do složenih tipova podataka poput nizova i objekata. Model podataka u obliku dokumenta je najpopularnija alternativa za relacijske baze podataka jer se podaci spremaju kao dokument čija struktura obično odgovara objektima s kojima radimo u aplikaciji, što znači da su nam svi podaci dostupni na jednom mjestu te ne moramo raditi skupe JOIN-ove kao u relacijskim bazama podataka. Kao i kod ostalih NoSQL baza podataka, shema nije fiksna što znači da svaki dokument može imati svoj set atributa. [28], [29, str. 36], [33], [35]



Slika 13: Usporedba relacijske baze podataka s dokument bazom podataka (MongoDB) [35]

Prethodna slika prikazuje na koji se način jedan objekt iz relacijske baze podataka može zapisati u dokument bazi podataka, u ovom slučaju MongoDB. Vidimo da se radi o intuitivnom zapisu gdje se svi podaci o jednom objektu nalaze unutar jednog dokumenta, za razliku od relacijske baze gdje se ti podaci nalaze u nekoliko tablica. Prilikom spremanja novog dokumenta sustav za upravljanje bazom podataka, u daljnjem tekstu SUBP, automatski dodijeli ID koji jednoznačno identificira taj dokument kako bi ga kasnije mogli dohvatiti. Kako tablica u relacijskoj bazi podataka predstavlja entitet i reci predstavljaju konkretnu instancu ili objekt, tako se u dokument bazama kreiraju kolekcije koje sadrže dokumente ili instance entiteta kojeg kolekcija predstavlja. Recimo da imamo entitet zaposlenik, tada prema slici 13 možemo dodati više zaposlenika u tu kolekciju s tim da nismo ograničeni shemom, tj. da svaki objekt ima fleksibilnu strukturu ili proizvoljan set atributa. Takav dokument identičan je retku u tablici relacijske baze podataka, dok je kolekcija dokumenata slična tablici. Što se tiče dohvata podataka svaki SUBP dokument baza podataka ima svoje aplikacijsko sučelje ili upitni jezik preko kojeg se dohvaćaju podaci. Podatke možemo dohvatiti i prema vrijednosti, što nije moguće u ključ–vrijednost bazama podatak, ali isto tako možemo ih dohvatiti prema ključu i ID-u. Na primjer, možemo napisati upit koji će dohvaćati sve zaposlenike iz kolekcije zaposlenici (pretpostavimo da je to naziv kolekcije unutar koje su pohranjeni zaposlenici) koji žive u određenom gradu ili imaju određena zanimanja (engl. „profession“). Na slici možemo uočiti i da imamo ugniježđene dokumente unutar dokumenta zaposlenik, lokacija (engl. „location“) je jedan takav primjer. To se može implementirati i na način da se kreira posebna kolekcija lokacije te se stavi referenca preko ID-a na određenu lokaciju gdje zaposlenik stanuje. [29, str. 36], [31, str. 73], [36]

Dokument baze podataka, u odnosu na ključ–vrijednost baze podataka, mogu kao vrijednost pohraniti i druge dokumente koje tada nazivamo ključ–dokument par, što ih čini složenijima i nešto naprednijima. Prednost, kao i kod ostalih NoSQL baza je skalabilnosti, ali i zbog svoje sheme omogućuju kreiranje prirodnih hijerarhija što ih čini vrlo intuitivnima za rad. Ove baze podataka mogu se koristiti u aplikacijama u kojima je shema varijabilna i potrebna je intuitivna hijerarhija podataka. Isto tako možemo se referencirati na druge dokumente što znači da se dokumenti mogu razdvojiti i particionirati na više dokumenata, naravno uz uvjet da imamo takav model podataka. S druge strane dokument baze podataka bi trebali izbjegavati ako baza ima velik broj veza ili relacija i potrebna je normalizacija. Neke od čestih primjena su u mobilnim aplikacijama, e-poslovanju, sustavi za upravljanje sadržajem i brojni drugi te se zapravo mogu koristiti kao baze podataka opće namjene. Ranije navedena MongoDB dokument baza podataka ujedno je i najpoznatija, a uz nju postoje i Firebase Realtime Database, CouchDB, Realm i druge. [28], [33], [36]

3.2. Usporedba NoSQL baza podataka s relacijskim bazama podataka

Sada kada smo upoznati s većinom tipova NoSQL baza podataka napravit ćemo kratku usporedbu s relacijskim bazama podataka, koje su prednosti i nedostaci pojedinih te usporedbu NoSQL baza međusobno.

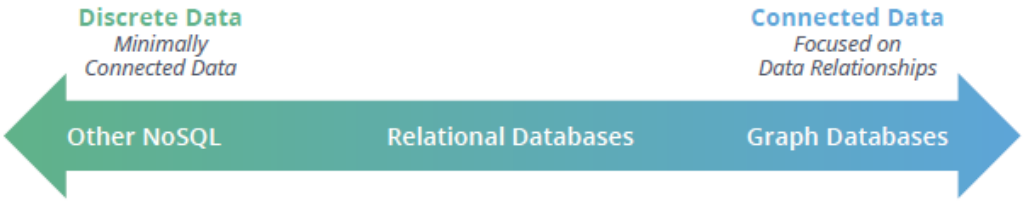
Strauch je u svojoj knjizi „NoSQL Databases“ predstavio klasifikaciju baza podataka prema nekim osnovnim karakteristikama. U idućoj tablici prikazana je usporedba četiri osnovna tipa NoSQL baza podataka s relacijskim bazama podataka prema Benu Scofieldu i Alexu Popescu. [27, str. 25 i 26]

Tablica 1: Usporedba NoSQL s relacijskim bazama podataka

	Performanse	Skalabilnost	Fleksibilnost	Složenost	Funkcionalnost
Ključ – vrijednost baze podataka	visoke	visoka	visoka	vrlo niska / nema je	varijabilna
Stupčane baze podataka	visoke	visoka	umjerena	niska	minimalna
Dokument baze podataka	visoke	varijabilna / visoka	visoka	niska	varijabilna
Grafovske baze podataka	varijabilne	varijabilna	visoka	visoka	teorija grafova
Relacijske baze podataka	varijabilne	varijabilna	niska	umjerena	relacijska algebra

(Izvor: prema [27, str. 26])

Iz prethodne usporedbe i klasifikacije baza podataka vidljivo je da NoSQL baze podataka pružaju visoke performanse, vrlo su skalabilne i fleksibilne te im je složenost vrlo niska. Jedina iznimka su grafovske baze podataka koje u nekoj mjeri podudaraju s relacijskim bazama podataka, ali su nešto složenije i pružaju fleksibilnu shemu.

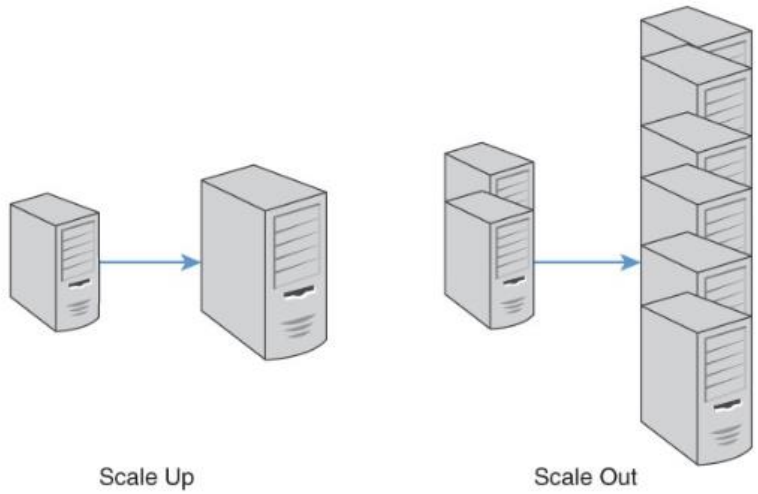


Slika 14: Korištenje baze podataka ovisno o vrsti podataka [29, str. 37]

Prethodna slika prikazuje zašto se grafovske baze nešto više podudaraju s relacijskim bazama podataka, radi se o tome da su fokusirane na povezane podatke i relacije između podataka. S druge strane, ostale NoSQL baze podataka fokusirane su na diskretne nepovezane podatke, tj. imaju minimalističke veze između podataka. One su vrlo efikasne za spremanje velikih količina diskretnih i heterogenih podataka, ali time žrtvuju model podataka i mogućnost povezivanja podataka. Naime, kako je već ranije navedeno NoSQL baze podataka mogu spremati strukturirane i povezane podatke, samo to rade na nešto drugačiji način gdje su podaci uglavnom razdvojeni što uvelike otežava kasnije povezivanje podataka prilikom dohvata te sam dohvat može biti vrlo spor ako zahtjeva kompleksna filtriranja. Grafovkse baze podataka nemaju problem povezanosti jer su fokusirane na relacije između podataka i podaci su međusobno povezani. [29, str. 37]

Klasifikacija je u tablici 1 napravljena prema osnovnim karakteristikama od kojih neke spadaju u četiri karakteristika SUBP-a koje su vrlo važne za rad s velikom količinom podataka. Te karakteristike su prema Sullivanu [31, str. 47]: skalabilnost, cijena, fleksibilnost i dostupnost. Možemo vidjeti da su ovo zapravo i najvažnije karakteristike o kojima treba voditi računa prilikom odabira SUBP-a.

Skalabilnost je sposobnost sustava da zadrži željene performanse prilikom velikog opterećenja. Postoji horizontalna i vertikalna skalabilnost, vertikalna je povezana s nadogradnjom postojećeg servera (engl. „scale up“) te se takvi sustavi smatraju centraliziranim, a horizontalna se odnosi na dodavanje novih servera prema potrebi (engl. „scale out“) što se pak smatra distribuiranim sustavima. [31, str. 47]



Slika 15: Vertikalna (lijevo) i horizontalna (desno) skalabilnost [31, str. 48]

Relacijske baze podataka su uglavnom vertikalno skalabilne, dok horizontalnu skalabilnost mogu postići korištenjem specifičnog softvera što zahtjeva nešto više resursa i

same operacije postaju kompleksnije. Kao što je vidljivo na slici 15, vertikalna skalabilnost postiže se nadogradnjom postojećeg servera tako da se nadogradi hardver (dodavanje memorije, procesora i sl.) ili da se server zamijeni s novim, boljim serverom ili poslužiteljem. Takva nadogradnja rezultirat će određenim vremenom u kojem server neće biti dostupan, tj. aplikacija se neće moći koristiti jer neće biti povezana s bazom podataka. Zamjena servera s novim serverom naziva se još i migracija gdje je potrebno prenijeti sve podatke sa starog poslužitelja na novi što zahtjeva i provjeru podataka nakon migracije. [31, str. 47 i 48]

Većina NoSQL baza podataka je horizontalno skalabilna što znači da se serveri mogu lako dodati i maknuti. S obzirom na arhitekturu NoSQL baza podataka koja podržava horizontalnu skalabilnost, njihov SUBP će se vrlo lako prilagoditi na korištenje novih servera koji su pokrenuti prilikom velikog opterećenja te na korištenje manjeg broja servera kada se smanji opterećenje i nepotrebni serveri se ugase. Ovim pristupom omogućujemo i lakšu nadogradnju pojedinih servera jer uvijek možemo neke ostaviti dostupnima dok druge gasimo za nadogradnju i tako omogućujemo nesmetan rad aplikacije. [31, str. 47 i 48]

Cijena se odnosi na licencu SUBP-a i ova karakteristika je samo opisujuća. Postoje brojni modeli naplate licence kao što je veličina servera, broj korisnika i sl. Ovo je vrlo važan faktor kod izrade aplikacije jer je teško predvidjeti broj korisnika koji će koristiti aplikaciju. Može se desiti da tvrtka plaća licencu za veći broj korisnika ili za veći server te se kasnije pokaže da aplikaciju koristi puno manje korisnika ili, u suprotnom, da se odluči za manji broj korisnika da bi se kasnije pokazalo da je aplikacija popularna te da je potreban puno veći broj korisnika. S ovim problemima se najčešće susreću web aplikacije te se kao rješenje koristi softver otvorenog koda (engl. „open source software“) koji je besplatan za korištenje. Većina NoSQL SUBP-a je upravo takva, dok je za većinu poznatijih SURBP potrebno kupiti licencu. [31, str. 48]

Fleksibilnost je jedna od glavnih prednosti NoSQL baza podataka te se odnosi na shemu. Relacijske baze podataka imaju fiksnu shemu i stoga je na početku razvoja aplikacije potrebno znati koje ćemo sve podatke spremati kako bi mogli pravilno oblikovati bazu podataka da sadrži sve potrebne tablice i atribute. Pretpostavka je da će se većina atributa koristiti u svim zapisima (redovima) kako bi se izbjegle NULL vrijednosti. Nadalje, sve daljnje promjene podrazumijevanju promjenu modela baze podataka ili sheme tablice što nije tako jednostavno i može zahtijevati puno izmjena. Recimo da imamo aplikaciju e-poslovanja koja sadrži razne proizvode poput bijele tehnike, kućanskih aparata, namještaja, računala, mobitela i drugih proizvoda. U relacijskim bazama podataka bi podrška za sve proizvode bila nešto kompleksnija za modeliranje, možemo kreirati jednu tablicu koja će imati sve moguće atribute koji opisuju sve proizvoda, ali ćemo onda imati puno NULL vrijednosti. Druga opcija je kreirati zasebnu tablicu za svaki tip proizvoda čime bi imali model s puno tablica koje je potrebno

kasnije i održavati. S druge strane, NoSQL baze podataka imaju fleksibilnu shemu što znači da možemo dinamički dodavati nove atribute po potrebi i ako se u budućnosti doda i neki novi proizvod s novim atributima može se jednostavno dodati u bazu bez potrebe za promjenom modela baze podataka ili sheme tablice kao u relacijskoj bazi podataka. [31, str. 49]

Dostupnost je posljednja karakteristika te je usko povezana sa skalabilnošću. Svi korisnici očekuju da su im aplikacije dostupne u bilo koje vrijeme što znači da je idealan slučaj svaki dan od 0 do 24 sata. Kako postoje i konkurentne tvrtke, ukoliko ona koju korisnik traži nije dostupna ili je vrlo spora, velika je vjerojatnost da će otići na stranice druge tvrtke, a nitko ne želi izgubiti kupce. Budući da su NoSQL baze podatak dizajnirane za rad s distribuiranim sustavima, tj. horizontalno su skalabilne, puno je lakše održati njihovu dostupnost jer jednostavno može dodati novi server kada je to potrebno. Kod relacijskih baza podataka koje su uglavnom vertikalno skalabilne to je nešto teže jer rade na jednom serveru i u slučaju ispada tog servera aplikacija nije dostupna. Kao rješenje može se uvesti pomoćni server koji će se uključiti u slučaju ispada glavnog servera. Naime, pomoćni server mora uvijek držati ažurne kopije podataka kako bi aplikacija mogla nastaviti s radom što znači da mora stalno raditi kako bi se održali ažurni podaci te da može odmah prilikom ispada zamijeniti glavni server. Takvim rješenjem moramo održavati i plaćati pomoćni server koji nam možda nikada neće ni trebati što je vrlo nezgrapno i stvara dodatan trošak. [31, str. 49]

Dostupnost je povezana i s CAP teoremom koji objašnjava karakteristike distribuiranih sustava te će biti objašnjen u idućem potpoglavlju. Sljedeća tablica prikazuje neke od glavnih razlika između relacijskih i NoSQL baza podataka, nekoliko ih je već ranije spomenuto, ali se ovdje nalaze sve na jednom mjestu.

Tablica 2: Usporedba relacijskih i NoSQL baza podataka

	Relacijske baze	NoSQL baze
Model pohrane podataka	Tablice s fiksnim atributima	Dokumenti, ključ–vrijednost parovi, tablice s redovima i dinamičkim stupcima, grafovi
Povijest razvoja	Razvijene 1970. s fokusom na smanjenje redundancije podataka	Razvijene kasnih 2000-ih godina s fokusom na skalabilnost i brzim promjenama u aplikacijama
Namjena	Opća namjena	Dokument baze: opća namjena; Ključ – vrijednost baze: velike količine podataka bez potrebe za složenim upitima; Stupčane baze: velike količine podataka s predvidljivim upitima; Grafovske baze: analiza međusobno povezanih podataka

Shema	Fiksna / stroga	Fleksibilna
Skalabilnost	Vertikalna	Horizontalna
ACID transakcije	Podržane	Uglavnom nisu podržane, ali ih neki SUBP poput MongoDB i Neo4j podržavaju
JOIN-ovi	Uglavnom su potrebni	Uglavnom nisu potrebni
Mapiranje relacija u objekte	Potrebno je objektno – relacijsko mapiranje	Većinom nije potrebno mapiranje, već se relacije direktno mapiraju u objekte

(Izvor: prema [37])

Prema izdvojenim kategorijama možemo uočiti da postoje brojne razlike između navedenih baza podataka. Jedna od bitnih kategorija je podrška za ACID transakcije, ACID i BASE su modeli konzistencije podataka od kojih ACID pruža strogu konzistentnost, a BASE pruža veću dostupnost. Svaki od navedenih modela ima svoje prednosti i mane koje će, uz modele konzistencije, biti objašnjene u potpoglavlju nakon CAP teorema.

3.3. CAP teorem

CAP (engl. „Consistency, Availability and Partition Tolerance“) je akronim za osnovne karakteristike distribuiranih sustava, a CAP teorem nalaže da distribuirani sustavi ne mogu istovremeno pružati konzistentnost (C), dostupnost (A) i toleranciju particioniranja (P). [27, str. 30], [31, str. 62]

Konzistentnost označava je li sustav, u ovom slučaju baza podataka, u konzistentnom stanju nakon izvršavanja operacije nad podacima. Drugim riječima, vide li svi korisnici iste podatke nakon što se izvrši neka operacija (npr. ažuriranje podataka) što znači da, bez obzira s kojeg servera dolaze, podaci moraju biti konzistentni. [27, str. 30], [31, str. 62], [38, str. 3]

Dostupnost je objašnjena i u prethodnom poglavlju i predstavlja sposobnost sustava da nastavi s radom u slučaju ispada servera (poslužitelja) ili njegove nadogradnje. To znači da sustav, u slučaju ispada nekog od poslužitelja, mora moći odgovoriti na zahtjev korisnika u vidu dohvata traženih podataka te mora dozvoliti i pisanje novih podataka u bazu. [27, str. 30], [31, str. 62], [38, str. 3]

Toleriranje particioniranja predstavlja sposobnost sustava normalno radi prilikom nadogradnje ili premještanja na novi server, da podrži dinamičko dodavanje i micanje servera te rješava situacije u kojima serveri ne mogu međusobno komunicirati. Serveri zapravo sadrže particije ili kopije baze podataka koje bi trebale sadržavati konzistentne podatke. Situacije u kojima serveri ne mogu međusobno komunicirati mogu nastati u slučaju ispada mreže koja ih

povezuje. U tom slučaju možemo ostaviti servere da rade, ali će doći će do nekonzistentnosti u podacima nakon ažuriranja ili unosa podataka jer serveri nisu usklađeni. Druga opcija je da isključimo server ili više njih koji nisu povezani s ostatkom kako bi sačuvali konzistentnost podataka, ali tada žrtvujemo dostupnost sustava. [27, str. 30], [31, str. 62 i 64], [38, str. 3]

Sustav, prema teoremu, istovremeno može postići najviše dvije karakteristike te izbor ovisi o domeni u kojoj će se koristiti baza podataka. Samim time postoje tri moguće kombinacije. Ako je potrebno osigurati konzistentnost podataka i toleriranje particioniranja tada se obično radi o ACID modelu, ako je ipak veća važnost stavljena na dostupnost i toleriranje particioniranja onda je sustav okarakteriziran BASE modelom, posljednja kombinacija je konzistentnost i dostupnost za koji ne postoji model konzistencije te takvi sustavi ne mogu u potpunosti funkcionirati s više poslužitelja. [27, str. 30]

3.4. ACID i BASE

Postoje dva modela konzistencije koji su poznati pod akronimima ACID i BASE. ACID model je povezan s relacijskim bazama podataka te je danas većina developera upoznata s ovim modelom i ACID svojstvima transakcija. S druge strane BASE je nešto noviji model koji označava svojstva zajednička većini NoSQL baza podataka. Prilikom odabira modela konzistencije potrebno je poznavati domenu i prema tome odabrati odgovarajući model, odabir modela s kojim smo bolje upoznati ili koji je trenutno popularniji nije dobar jer postoji velika mogućnost nastanka problema tijekom razvoja aplikacije. S obzirom na to, potrebno je poznavati oba modela te odabrati onaj koji više odgovara aplikacijskoj domeni.

3.4.1. ACID

ACID osigurava sigurno okruženje za rad s podacima te je akronim za atomarnost (engl. „atomicity“), konzistentnost (engl. „consistency“), izolaciju (engl. „isolation“) i trajnost (engl „durability“). [29, str. 31]

Atomarnost (A) označava da se moraju izvršiti sve operacije unutar transakcije, ili niti jedna, tj. da transakciju treba poništiti. **Konzistentnost (C)** osigurava da se baza, nakon izvršenja transakcije, nalazi u konzistentnom stanju što znači da se ne smije narušiti integritet podatak i da moraju zadovoljena sva ograničenja. **Izolacija (I)** nam govori da se transakcije izvršavaju „izolirano“ jedna od druge kako ne bi došlo do njihovog međusobnog sukobljavanja. Time se stvara privid slijednog izvršavanja transakcija, ali se radi o konkurentnim sustavima koji dozvoljavaju paralelno izvršavanje više operacija pa tako i transakcija čime dolazimo do problema konkurentnosti. Ti problemi rješavaju se mehanizmom zaključavanja koji sinkronizira izvršavanje konfliktnih operacija te onemogućuje nekontrolirano sukobljavanje transakcija.

Trajnost (D) je posljednje svojstvo i označava da su svi rezultati uspješno izvršenih transakcija trajni, čak i u slučaju greške ili pada sustava. Drugim riječima, po uspješnom završetku transakcije sve promjene spremaju se na trajni medij pohrane podataka. [29, str. 31], [30, str. 93-95], [31, str. 65]

Relacijske baze podataka su dizajnirane tako da podržavaju ACID model, ali ga podržava i većina grafovskih baza podataka. U posljednje vrijeme i druge NoSQL baze podataka rade na podršci ACID transakcija od kojih je jedna dokument baza podataka MongoDB. Ovaj model nalaže strogu konzistentnost i sigurnu pohranu podataka i neke aplikacije, poput bankarskih aplikacija, ga jednostavno moraju podržavati. Isto tako može biti izazovan za implementaciju na distribuiranim sustavima te zahtijeva napredan mehanizam zaključavanja koji može usporiti rad sustava i jednostavno nije potreban za određene aplikacije. Sve u svemu, ACID model je savršen za aplikacije u kojima je ključna pouzdanost i konzistentnost podataka. [29, str. 32]

3.4.2. BASE

BASE model konzistentnosti u fokus stavlja dostupnost i horizontalnu skalabilnost za što je potrebno ublažiti zahtjeve poput stroge konzistentnosti i točnosti. Akronim predstavlja: osnovnu dostupnost (engl. „Basic Availability“), labavo stanje (engl. „Soft State) i eventualnu konzistentnost (engl. „Eventual Consistency“). [29, str. 31]

Osnovna dostupnost (BA) znači da imamo privid da baza podataka radi cijelo vrijeme ili pak radi većinu vremena. Što znači da u slijedu greške dijela distribuiranog sustava samo taj dio ne radi, ali ostatak sustava radi normalno. Ukoliko se podaci u bazi podataka ne repliciraju kroz servere tada samo dio korisnika, čiji se podaci nalaze na serverima koji imaju grešku, neće moći koristiti aplikaciju dok će svi ostali moći. S druge strane, ako se drže kopije (replike) podataka na više servera tako da će svi moći nesmetano koristiti aplikaciju uz moguć pad performansi. **Labavo stanje (S)** označava da spremišta ne moraju uvijek biti konzistentna za pisanje, niti da su sve replike podataka međusobno konzistentne. Ovo svojstvo povezano je s posljednjim svojstvom **eventualne konzistencije (E)** koje nam govori da spremišta postižu konzistentnost s vremenom, tj. da baza može u nekim kraćim vremenskim intervalima biti nekonzistentna. Na primjer, ako držimo replike podataka na više servera, tada postoji mogućnost da će one biti nekonzistentne u nekom kraćem vremenu. Recimo da korisnik ažurira adresu na svom profilu i podaci se pohrane na najbliži server, ostale replike neće odmah imati tu novu adresu i može se desiti da različiti korisnici vide različite adrese, ovisno s koje replike dohvaćaju podatke. NoSQL baze će s vremenom ažurirati sve replike i vratiti se u konzistentno stanje. Period u kojem su podaci nekonzistentni ovisi o faktorima kao što je brzina mreže kojom su serveri povezani i udaljenost između servera. [29, str. 32], [31, str. 65 i 66]

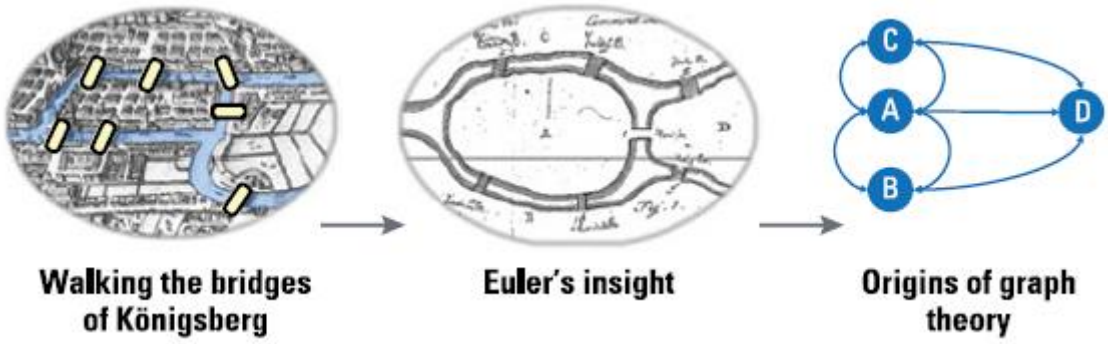
BASE model konzistencije nastao je kako bi se osigurala dostupnost i omogućilo lako horizontalno proširenje sustava. Ova svojstva transakcija vezana su za NoSQL baze podatak te se primarno koristi u ključ–vrijednost, stupčanim i dokument bazama podataka, ali ga podržavaju i grafovske baze podataka. [29, str. 32]

Može nam se činiti da ovaj model, zbog labavog stanja i eventualne konzistencije, nije optimalan, ali ako u fokus stavimo dostupnosti i horizontalnu skalabilnost on je savršen za određene domene. Kao primjer možemo uzeti društvene mreže, ne možemo si zamisliti situaciju da zbog ispada servera aplikacije poput Facebooka, Instagrama i Twittera nisu dostupne. S druge strane, zapitajmo se koliko nam je bitno da odmah vidimo novu objavu ili promjenu na nečijem profilu? U većini slučajeva nije nam važno hoćemo li je vidjeti kroz nekoliko sekundi ili odmah ako to znači da je aplikacija uvijek dostupna i da će pružiti optimalne performanse. Uglavnom se konzistentnost postiže u intervalima manjima od jedne sekunde tako da je ovaj pristup idealan za društvene mreže i slične domene.

3.5. Teorija grafova

Iako su grafovske baze podataka vrlo intuitivne i razumljivije od relacijskih baza podataka, ukratko ćemo se upoznati s teorijom grafova i osnovnim definicijama radi lakšeg razumijevanja.

Porijeklo teorije grafova bilježi se davne 1736. godine i potječe iz grada Konigsberga. Temeljni koncept postavio je matematičar Leonhard Euler rješavajući problem sedam mostova Konigsberga. Grad je obuhvaćao dva velika otoka i dva kopnena dijela koji su međusobno povezani sa sedam mostova. Cilj zagonetke bio je kreirati šetnju kroz cijeli grad tako da se svaki most prijeđe samo jednom. Euler je postavio pitanje da li je to uopće moguće te došao do zaključka da nije. Zaključio je da su jedina važna svojstva kopnene površine i mostovi čime je apstrahirao problem na način da je svaku kopnenu površinu prikazao kao čvor, a svaki most kao brid ili vezu čime je stvorio graf. [39, str. 4]



Slika 16: Porijeklo teorije grafova [39, str. 4]

Graf G definiramo kao uređeni par (V, E), gdje je V skup vrhova ili čvorova (engl. „vertices“) i $E \subseteq V\ x\ V$ skup bridova (engl. „edges“). [40, str. 4]

Usmjereni graf G je graf kod kojeg za svaki brid vrijedi $e = (v_i, v_j)$, gdje je v_i izvorište, a v_j odredište brida e. [26, str. 245], [40, str. 4]

Težinski graf (engl. „Weighted graph“) je graf u kojem svaki brid ima svoju vrijednost te se definira kao trojka (V, E, w) gdje $w: E \rightarrow \mathbb{R}$ dodjeljuje vrijednost bridu. [40, str. 4]

Put u usmjerenom grafu G je svaki niz bridova $b_1 / b_2 / \dots / b_n$ za koje vrijedi da je $odredište\ (b_i) = izvorište\ (b_{i+1}),\ i = 1, 2, \dots,\ n - 1$. Duljina puta je broj bridova n u putanji. [26, str. 245]

Korijen usmjerenog grafa G = (V, E) je čvor v_n, ako i samo ako postoji put od v_n do svakog vrha v_i, gdje je $v_i \in V, i \neq n$. [26, str. 245]

Ciklus u usmjerenom grafu G definiramo kao svaki put koji počinje u nekom čvoru v_n i završava u istom tom čvoru v_n, što znači da je to svaki put koji počinje i završava u istom čvoru. Graf u kojem nema ciklusa nazivamo **aciklički graf**. [26, str. 246]

Usmjereni graf G = (V, E) smatra se **stablom** ako i samo ako postoji jedinstveni put od čvora v_n do svakog čvora v_i, gdje je $v_i \in V, i \neq n$. [26, str. 246]

List usmjerenog grafa G = (V, E) je svaki čvor $v \in V$ za koji vrijedi da ne postoji niti jedan brid $e \in E$ takav da je $izvorište\ (e) = v$. Drugim riječima, to su čvorovi koji nemaju niti jedno dijete. [26, str. 246]

Podatkovni graf je usmjereni graf $G_p = (V, E)$ koji ima označene bridove i vrhovi su podatkovni objekti. Podatkovni objekti mogu biti atomarni (list) ili složeni (sadrže bridove prema drugim čvorovima). Ovi grafovi su najčešće aciklički. [26, str. 247]

3.6. O grafovskim bazama podataka

Nakon što smo upoznati s osnovama teorije grafova vrijeme je da detaljnije definiramo posljednji tip NoSQL baza podataka, grafovske baze podataka.

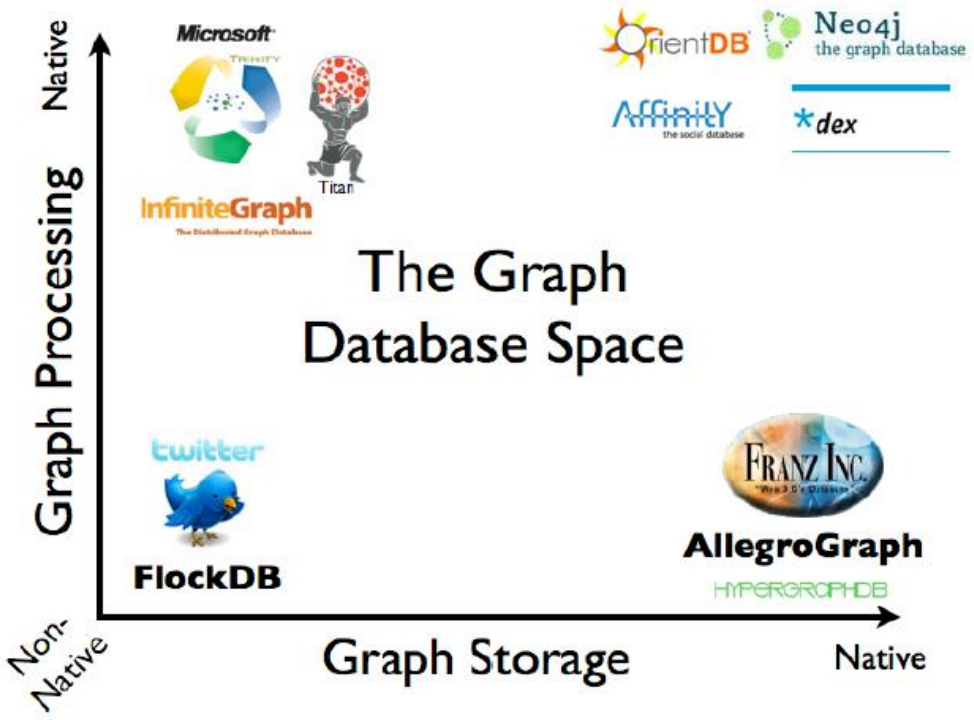
Grafovske (engl. „Graph“) baze podataka temelje se na teoriji grafova i njihov je fokus na vezama između podataka, što znači da su dobre za rad s podacima koji su međusobno povezani. Jednostavno rečeno, sastoje se od čvorova koji su međusobno povezani preko usmjerenih veza (usmjereni graf), a podaci se pohranjuju u čvorove i veze od kojih svaki posjeduje atribute kako bi se pohranili dodatni detalji. [30, str. 230], [31, str. 76], [33]

Sustav za upravljanje grafovskom bazom podataka, nadalje SUGBP, ili grafovska baza podataka je sustav koji posjeduje CRUD (engl. „Create, Read, Update, Delete“) metode za rad s grafovskim modelom podataka. Kao što je i ranije spomenuto, grafovske baze podataka uglavnom podržavaju ACID transakcije jer su izgrađene za rad s transakcijskim (OLTP) sustavima. Što se tiče fizičke pohrane podataka iz grafovskog modela podataka, ona može biti u nativnom (graf) i nenativnom (tablice, ključ–vrijednost parovi, dokumenti i drugi.) obliku. Uz to važno je napomenuti da u grafovskom modelu podataka nativnih baza čvorovi sadrže fizičke „pokazivače“ na susjedne čvorove s kojima su povezani (engl. „index-free adjacency“), takva implementacija modela višestruko unaprjeđuje brzinu izvođenja upita i samo pretraživanje baze podataka. [41], [42, str. 5]

Ono što razlikuje grafovske baze podataka od drugih baza podataka je grafovski model podataka koji je istovremeno jednostavniji i ekspresivniji od drugih modela. Razlog tome je to što su u fokusu grafovskog modela podataka veze između podataka, dok kod drugih modela veze izvodimo iz samog modela koristeći vanjske ključeve ili algoritme kao što je map–reduce. [42, str. 6]

Upotreba grafovskih baza podataka je vrlo široka, a njihova je primjena u slučajevima kada radimo s povezanim podacima i želimo pronaći određene uzorke u tim podacima. Neke od takvih domena su društvene mreže, sustavi za preporuke, detekcija računalnih prijevara, sustavi za upravljanje sadržajem i druge. Najpoznatije grafovske baza podataka su Neo4j, OrientDB, JanusGraph, AllegroGraph i druge [33], [36], [41]

Sljedeća slika prikazuje neke grafovske baze podataka temeljem modela pohrane i procesiranja. U idućim poglavljima rada bit će detaljnije objašnjen grafovski model podataka i kako se isti modelira, razlike između nativnih i nenativnih grafovskih baza podataka te koje su glavne prednosti, a i nedostaci grafovskih baza podataka.



Slika 17: Pregled grafovskih baza podataka [42, str. 6]

3.7. Grafovski model podataka i modeliranje

Poznato nam je iz svijeta relacijskih baza podataka da postoji konceptualno, logičko i fizičko oblikovanje baze podataka. Temelj konceptualnog oblikovanja ili modeliranja baze podataka su koncepti i veze između koncepata te je cilj modeliranja opisati aplikacijsku domenu. S obzirom na to, modeliranjem želimo apstrahirati i pojednostaviti domenu kako bi dobili shemu ili model baze podataka. Nadalje, podaci se u bazu podataka pohranjuju u obliku odgovarajuće strukture modela podataka nad kojima možemo izvršavati određene operacije podržane tim modelom. Grafovski model podatka koristi se za implementaciju podataka u grafovskim bazama podataka, slično kao i relacijski model podataka u relacijskim bazama podataka. Ono što grafovski model podataka razlikuje od ostalih je uska povezanost logičkog i fizičkog modela. U relacijskom modelu moramo konceptualni model pretvoriti u logički model tako da zadovoljimo potrebne zavisnosti i normaliziramo model, nakon čega možemo kreirati fizički model u odabranom SURBP-u. Samim time dobivamo nešto drugačiju reprezentaciju između naše percepcije domene i modela implementiranog u SURBP-u što nije slučaj s grafovskim modelom podataka. Možemo reći da je grafovski model podataka vrlo sličan ljudskoj percepciji domene te ga nazivaju i „whiteboard friendly“ modelom. Taj naziv je dobio iz razloga jer obično radimo skicu modela (konceptualni model) prilikom opisa domene koja je u konačnici vrlo slična grafovskom modelu kojeg implementiramo u SUGBP-u. S druge strane,

tako skicirani model, u relacijskim bazama podataka bi morali normalizirati da bi dobili relacijski model što se u konačnici može uvelike razlikovati od početne skice. Što se tiče ekspresivnosti grafovskog modela, smatra se da je najtočniji s aspekta modeliranja stvarnosti i minimizira razliku između ljudske percepcije stvarnosti i implementacije modela u bazi podataka. [30, str. 167 i 204], [41], [42, str. 25 i 26]

Robinson, Webber i Eifrem [42, str. 26] navode da su grafovski model podataka i upiti koje postavljamo nad istim zapravo dvije strane istog novčića. Razlog tome je ekspresivnost grafovskog modela kojom ne samo da je sličan našoj percepciji stvarnosti, već jasno izražava pitanja unutar domene na koja i mi sami želimo dobiti odgovor.

Kada govorimo o grafovskom modelu podataka važno je napomenuti da ne postoji standardni model, već je razvijeno nekoliko modela od kojih je svaki prilagođen specifičnim potrebama. Iako ih ima više, svaki model posjeduje tri ključne komponente: [41]

1) Strukturalna – predstavlja strukturu pohrane podataka. U grafovskom modelu podataka to su elementi grafa: čvorovi i veze.
2) Operativna – razni operatori iz teorije grafova koji omogućuju transformacije grafa (traženje uzoraka u grafu, podgrafovi, putanje i drugi).
3) Integritetna – odnosi se na integritetna ograničenja nad shemom i samim podacima. Takva ograničenja mogu biti jedinstveni nazivi za oznake čvorova i veza, jedinstvena vrijednost atributa i slično.

Neki od razvijenih modela spomenuti u [40, str. 4 - 6] su jednostavan model grafa, model hipergrafa, model označenog grafa sa svojstvima (engl. „Labeled Property Graph Model“), RDF (engl. „Resource Description Framework“) i varijante između označenog grafa sa svojstvima i RDF-a.

Temeljem ključnih komponenti grafa možemo uočiti da jednostavan model grafa koji se sastoji od čvorova i veza nije dovoljno izražajan da se ispravno modelira neka domena te se koristi kao temelj za složenije modele. Točnije, kao osnovni model podataka koristi se usmjereni graf što je jedna od varijanti jednostavnog modela grafa. U današnje vrijeme najpopularniji su model označenog grafa sa svojstvima i RDF model grafa. [41]

RDF je standardizirani format i podatkovni model u obliku označenog usmjerenog grafa koji se koristi za prikaz i razmjenu podataka na Webu. RDF grafovi se prikazuju kao kolekcije trojki subjekt–predikat–objekt. Subjekt je entitet koji se opisuje i može biti jedinstveni identifikator, URI (engl. „Uniform Resource Identifier“) ili anonimni resurs (engl. „blank node“), a objekt može biti URI, anonimni resurs ili literal (jednostavna vrijednost kao znak, broj i slično). RDF izraze kreiramo povezivanjem RDF trojki, a niz takvih povezanih RDF izraza tvori RDF graf. Formalno RDF trojku opisujemo kao: [40, str. 5 i 6], [41]

$$(s, p, o) \in (URI \cup anonimni\_resurs) \times (URI) \times (URI \cup anonimni\_resurs \cup literal)$$

Vidimo da je ovo samo formalan zapis prethodno navedenih elemenata (URI, anonimni resurs i literal) koji čine RDF trojku, ti elementi nazivaju se i RDF termi.

Budući da će se za izradu aplikacije koristiti Neo4j SUGBP koji se temelji na modelu označenog grafa sa svojstvima, navedeni model bit će i detaljnije objašnjen u idućem poglavlju. Prethodno je samo ukratko objašnjen RDF model grafa tako da se upoznamo s oba modela koja su vrlo popularna u današnje vrijeme.

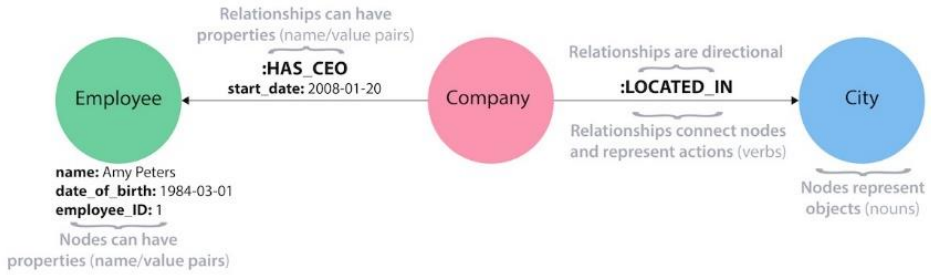
3.7.1. Model označenog grafa sa svojstvima

Kao što nam govori i samo ime modela, na jednostavan usmjereni graf dodajemo oznake (engl. „labels“) i svojstva (engl. „properties“). Preko oznaka kreiramo klase čvorova i bridova (veza), a svaki čvor i veza može imati svojstva ili atribute koji ih detaljnije opisuju. Formalno, model označenog grafa sa svojstvima definiramo kao: $(V, E, L, l_V, l_E, K, W, p_V, p_E)$, gdje je V skup čvorova, E skup veza i L skup oznaka. $l_V: V \mapsto P(L)$ i $l_E: E \mapsto P(L)$ su funkcije koje čvorovima i vezama pridružuju oznake. Nadalje, svakom čvoru i vezi mogu se pridružiti svojstva. Svojstva ili atributi su zapravo parovi ključ–vrijednost: $p = (ključ,\ vrijednost)$, gdje je $ključ \in K$ i $vrijednost \in W$ te K i W predstavljaju skup svih mogućih ključeva i vrijednosti. Na kraju, $p_V(u)$ označava skup svih svojstava čvora u, a $p_E(e)$ skup svih svojstava veze e. [40, str. 5]

Model označenog grafa sa svojstvima smatra se najpopularnijim grafovskim modelom podataka i možemo reći da posjeduje sljedeće karakteristike: [42, str. 4, 26 i 27]

- Sastoji se od čvorova i veza
- Čvorovi sadrže svojstva/atribute u obliku ključ–vrijednost parova
- Čvor može biti označen s jednom ili više oznaka. Oznake tada grupiraju čvorove, tj. kreiraju klase istovrsnih čvorova
- Veze su imenovane (sadrže samo jednu oznaku), usmjerene i uvijek imaju izvorišni i odredišni čvor te tako strukturiraju graf. Ne može postojati neka „viseća“ veza koja nema izvorišni i odredišni čvor.
- Veze, kao i čvorovi, mogu imati svojstva/atribute

Nakon što smo upoznati s elementima modela, slijedi njegov slikovni prikaz na kojem su označeni svi navedeni elementi.



Slika 18: Emlementi modela označenog grafa sa svojstvima (Izvor: https://dist.neo4j.com/wp-content/uploads/property_graph_elements.jpg)

Označeni graf sa svojstvima na prethodnoj slici sastoji se od tri čvora koji predstavljaju entitete Zaposlenik (engl. „Employee“), Kompanija (engl. „Company“) i Grad (engl. „City“). Nazivi entiteta ujedno predstavljaju i oznake čvorova. Isto tako vidljive su i dvije veze, jedna s oznakom „:HAS_CEO“, koja ujedno predstavlja i tip veze, čiji je izvorišni čvor Kompanija, a odredišni je Zaposlenik (smjer je prikazan strelicom na grafu). Druga veza ima oznaku „:LOCATED_IN“ i povezuje čvor Kompanija s čvorom Grad, što čitamo kao „Kompanija se nalazi u Gradu“. Znamo da čvorovi, kao i veze, mogu sadržavati svojstva ili atribute koji ih opisuju pa tako čvor Zaposlenik ima svojstva „name“, „date_of_birth“ i „employee_ID“ od kojih svako svojstvo ima pridruženu odgovarajuću vrijednost. Što se tiče atributa veza, oni su uglavnom količinske prirode, npr. ocjena, težinska vrijednost, cijena, udaljenost, vremenski interval i slično. Većinom se koriste takve vrijednost kako bi kasnije raznim algoritmima mogli izračunavati vrijednosti kao što je udaljenost, prosječna ocjena i slično. Važno je napomenuti da, zbog načina pohrane, čvorovi mogu sadržavati neograničen broj veza bilo kojeg tipa bez da se naruše performanse. [43], [44]

Prethodno je spomenuto da je grafovski model podataka „whiteboard friendly“ što možemo i vidjeti iz prethodne slike. Kao ljudskim bićima, prirodno nam je razmišljati na način da imamo kompaniju koja se nalazi u određenom gradu ili više njih (ako ima više ureda ili poslovnica) te da ima svoje zaposlenike. Na taj način skicirali bi model u kojem bi imali čvor s nazivom kompanije i čvor s nazivom grada te bi kompaniju povezali s tim gradom. Slično bi kreirali i čvorove zaposlenika te ih povezali s odgovarajućim kompanijom, pa čak i gradom ukoliko želimo imati podatak u kojem gradu zaposlenik živi. Naravno, imali bi i razne veze (tipove veza) kako bi mogli pravilno povezati entitete. Naposljetku takav model potrebno je obogatiti s oznakama i svojstvima. Oznake nam pomažu da grupiramo istovrsne čvorove, a svojstva da detaljnije opišemo čvor i vezu. Takav model tada možemo prenijeti u grafovsku bazu podataka, bez potrebe za promjenom modela.

3.7.2. Reprezentacija grafa

Graf se može reprezentirati na više načina, od kojih se najčešće koriste tri podatkovne strukture:

1) Matrica susjedstva (engl. „Adjacency matrix“) – graf koji ima $|V|$ čvorova reprezentiramo kao matricu dimenzija $|V| \times |V|$ koja sadrži boolean vrijednosti (0 i 1), ovisno o tome jesu li čvorovi povezani ili ne. Drugim riječima, vrijednost u retku i i stupcu j je 1 ako i samo ako u grafu postoji veza (i, j). Ako se radi o težinskom grafu tada možemo pohraniti vrijednost (težinu) u matricu ukoliko postoji veza, a u suprotnome staviti null vrijednost ili prazno polje. Ova struktura je vrlo efikasna ako želimo provjeriti jesu li dva čvora povezana, što čini u vremenu $O(1)$ jer se u matrici nalazi taj zapis. Najveći nedostatak je to što uzima $O(n^2)$ prostora za pohranu, čak i ako se radi o rijetkom (engl. „sparse“) grafu koji ima vrlo malo veza što znači da će matrica sadržavati puno vrijednosti koje su 0 ili null. Još jedan nedostatak je pronalazak svih susjednih čvorova određenog čvora jer moramo proći kroz cijeli red matrice, iako možda taj čvor ima samo jednog susjeda. Važno je napomenuti da je za neusmjereni graf matrica susjedstava simetrična, što nije slučaj s usmjerenim grafom. [40, str. 7], [45]
2) Lista veza (engl. „Edge list“) – predstavlja graf kao jednostavnu listu veza $|E|$. Veza se reprezentira tako da se spremi izvorišni i odredišni čvor, a ako se radi o težinskom grafu jednostavno dodamo treću vrijednost u listu koja predstavlja „težinu“ veze. Prednost ove strukture je to što uzima puno manje prostora za pohranu jer se spremaju samo veze koje sadrže dvije ili tri vrijednosti ovisno o tipu grafa. S druge strane, glavni nedostatak je provjera povezanosti čvorova jer se u najgorem slučaju taj zapis može nalaziti na kraju liste što znači da moramo proći kroz cijelu listu. [40, str. 7], [45]
3) Lista susjedstva (engl. „Adjacency list“) – na neki način kombinira prethodne dvije podatkovne strukture i smatra se hibridnom strukturom. Svaki čvor u ima pridruženu listu koja sadrži identifikatore svih njemu susjednih čvorova A_u. Formalno: $v \in A_u \Leftrightarrow (u, v) \in E$, čvor v spada u listu susjednih čvorova A_u ako postoji veza E između čvorova u i v. Efikasnost ove strukture leži u pronalasku svih susjeda od traženog čvora, što znači da se mogu lako provesti standardni upiti kao što je prolazak grafom. Nešto je sporija provjera povezanosti čvorova jer moramo prvo pronaći izvorišni čvor te u najgorem slučaju proći kroz cijelu listu susjeda tog čvora (ako je odredišni čvor posljednji u listi). Što se tiče pohrane, zahtjeva nešto više prostora nego lista veza, ali i dalje puno manje nego matrica susjedstva. [40, str. 7], [45]

Ovo su samo tri osnovne strukture koju najpopularnije, većina SUGBP-ova koristi neke modifikacije navedenih struktura. Na primjer, sustavi Neo4j, OrientDB, JanusGraph koriste neke svoje varijante liste susjedstva jer efikasna za provedbu standardnih upita nad strukturom grafa kao što je obilazak susjednih čvorova. Isto tako na navedenim modelima nema svojstava tako da je i njih potrebno, na neki način, pohraniti. [40, str. 7]

3.8. Nativne i nenativne grafovske baze podataka

Razvojem NoSQL baza podatak uočili smo da jedna tehnologija (do tada relacijske baze podataka) ne može biti dobra za sve namjene, ovo se ne odnosi samo na SUBP-ove već na sve tehnologije općenito. S obzirom na to, pojedini SUBP je bolji za izvršavanje određenih zadataka od drugog i suprotno. Što se tiče SUGBP-ova, možemo ih podijeliti u dvije skupine: nativne i nenativne. Razlike između pojedinih skupina očituju se u vidu organizacije pohrane podataka i implementaciji procesora upita. Nativne grafovske baze podataka su, svojom arhitekturom, rađene sa svrhom efikasnog rada s grafovima. [29, str. 41], [42, str. 5]

3.8.1. Organizacija pohrane podataka

U prethodnom poglavlju spomenute su tri osnovne strukture reprezentacije grafa i nativne grafovske baze podataka koriste upravo te strukture ili neke njihove varijante za organizaciju pohrane podataka. Time osiguravaju efikasnu pohranu podataka na način da su čvorovi i veze pohranjeni blizu jedni drugima, tj. tako da se vrlo lako i brzo mogu pronaći povezani podaci. Takva pohrana omogućuje vrlo brz prolazak strukturom grafa i brzo izvršavanje algoritama temeljnih na grafovima. Što se tiče samog Neo4j sustava, on koristi datoteke za pohranu podataka (postoji zasebna datoteka za čvorove, veze, svojstva i oznake) te su zapisi u datotekama fiksne veličine. Zapisi fiksne veličine omogućuju vrlo brz pronalazak traženog podatka, npr. čvorovi imaju veličinu od 9 bajtova i ako tražimo čvor koji ima ID 10 tada znamo da se taj čvor nalazi nakon 90. bajta u datoteci čvorova. Na taj način se vrlo brzo može izračunati gdje se nalazi pojedini čvor što znatno poboljšava performanse sustava. Svaki čvor ima pokazivač na oznaku, prvu vezu i prvo svojstvo. Ima pokazivač samo na prvo svojstvo jer su svojstva pohranjena kao vezana lista, što znači da se može lako doći do ostalih svojstava čvora ili veze. Što se tiče veza, one su povezane kao dvostruko vezana lista što znači da imaju pokazivač na izvorišni i odredišni čvor. Tako se pripadnost veze ne stavlja samo na jedan čvor već je moguća iteracija u oba smjera. Druga prednost ove implementacije je lakoća dodavanja novih i brisanja postojećih veza. [29, str. 41 i 42], [42, str. 152 - 156]

S druge strane, nenativne grafovkse baze podataka koriste druge modele za pohranu podataka o grafu koji zapravo nisu razvijeni, niti prvotno zamišljeni, za njihovu pohranu. To su

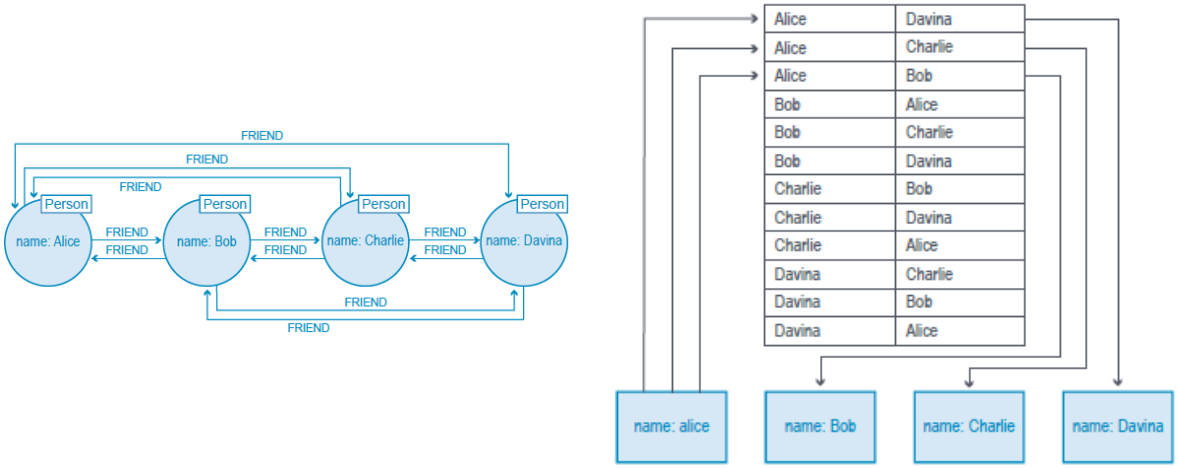
zapravo modeli ili baze podataka koji su serijalizacijom podataka prilagođene za modeliranje i pohranu grafova. U njih ubrajamo relacijske baze podataka, objektno–orijentirane baze podataka, RDF baze podataka, ključ–vrijednost baze podataka i ostale NoSQL baze podataka. Tako bi model grafa u relacijskoj bazi podataka pohranili kao kolekciju tablica na način da bi jedna sadržavala podatke o čvorovima, a druga o vezama. Svaki čvor posjeduje jedinstveni identifikator (ID) ili primarni ključ te bi veze postigli referenciranjem na primarne ključeve izvorišnog i odredišnog čvora (kreiranjem vanjskih ključeva). Možemo uočiti da ovakav pristup pohrani grafa nije nimalo optimalan jer podaci mogu biti dosta razdvojeni jedni od drugih te bi uglavnom morali koristiti veze više naprema više što zahtjeva određeno procesiranje prilikom prolaska po grafu i traženja susjednih čvorova. Isto tako znamo da su relacijske baze podataka vertikalno skalabilne što nas dovodi do problema skalabilnosti, a navedeni način pohrane nam ukazuje i problem performansi. [29, str. 42], [40, str. 7 - 10], [42, str. 5]

Zaključujemo da su nativne grafovske baze podataka dizajnirane za optimalnu pohranu grafova što znači da će nam pružiti optimalne performanse, brže izvršavati upite i zahtijevati manje resursa. Ovisno o potrebama aplikacije, postoji mogućnost pohrane grafova u nenativnom obliku. Ako radimo s manjim setom podataka i upoznati smo s konceptom relacijskih baza podataka možemo odabrati takvu nenativnu grafovsku bazu podataka jer neće puno utjecati na performanse. Tendencija je da podaci rastu s vremenom tako da treba voditi računa o tome prilikom izbora SUGBP-a. Nativne grafovske baze podataka su skalabilne i rast podataka neće utjecati na njihovu brzinu. [29, str. 44]

3.8.2. Procesor upita

Već je ranije spomenuto da u nativnim grafovskim bazama podataka čvorovi sadrže fizičke „pokazivače“ na svoje susjedne čvorove što se zove svojstvo susjedstva bez indeksa (engl. „indeks-free adjacency“). To svojstvo ne samo da ubrzava dohvat podataka i pretraživanje grafa, već ubrzava i samu pohranu podataka te osigurava da je svaki čvor pohranjen tako da je povezan sa susjednim čvorovima i vezama koje ih povezuju. Samim time vrijeme izvršavanja upita je identično neovisno o količini podataka koja je pohranjena, već se vrijeme povećava samo s veličinom grafa kojeg je potrebno pretražiti. [29, str. 41 i 43]

Nenativne grafovske baze podataka nemaju prethodno navedeno svojstvo već koriste globalne indekse za povezivanje susjednih čvorova. Ovakav pristup je puno skuplji jer se veze moraju izvoditi prilikom izvršavanja pojedinog upita i brzina izvođenja upita će znatno porasti dodavanjem sloja veza koji je potrebno proći da bi se upit izvršio. Isto tako dolazimo do problema ako želimo prolaziti grafom u suprotnom smjeru, za to bi trebali kreirati indeks za obrnuto pretraživanje za svaki scenarij prolaska grafom ili moramo izvršiti „brute-force“ pretragu globalnog indeksa. [29], [42]



Slika 19: Nativno (lijevo) i nenativno (desno) procesiranje (Izvor: [29, str. 43])

Prethodna slika prikazuje razliku između nativnog (svojstvo susjedstva bez indeksa) i nenativnog (globalni indeks) procesiranja. Vidimo da povezivanje putem globalnog indeksa nije efikasno i vrlo je teško pretraživati graf preko više slojeva veza. Sad kad smo upoznati s razlikom nativnih i nenativnih grafovskih baza podataka možemo se vratiti na sliku 17 na kojoj je prikazan pregled nekih grafovskih baza podataka te njihova pripadnost nativnim ili nenativnim bazama ovisno o pohrani podataka i procesoru upita kojeg koriste.

3.9. Prednosti i mane grafovskih baza podataka

Budući da su grafovske baze podataka relativno nova tehnologija, one ne mogu jednostavno zamijeniti tradicionalne, dobro usvojene i svima dobro poznate relacijske baze podataka. Činjenica je da nam je model grafa prirodniji te gotovo sve možemo prikazati preko grafa, ali svaka tehnologija ima svoje prednosti i mane te je potrebno odabrati najbolju ovisno o domeni koju modeliramo. Iduće karakteristike, prema [42], predstavljaju glavne prednosti grafovskih baza podataka:

- **Performanse** u radu s povezanim podacima. Svojim procesorom upita i organizacijom pohrane mogu zadržati identične performanse izvođenja upita čak i nakon rasta podataka. Razlog tome je što su upiti lokalizirani samo na dio grafa te je vrijeme izvođenja upita proporcionalno veličini tog dijela grafa koji se pretražuje. S druge strane u relacijskim bazama podataka JOIN je vrlo skupa operacija i performanse padaju rastom količine podataka. [42, str. 8]
- **Fleksibilnost** se odnosi na shemu grafovskog modela podataka. Ova karakteristika objašnjena je u ranijim poglavljima rada.

- **Agilnost** grafovske baze podataka i njena fleksibilna shema omogućavaju njenu prilagodbu učestalim promjenama korisničkih zahtjeva. Ova karakteristika podudara se s današnjim agilnim i iterativnim razvojem softvera te razvojem softvera vođenim testiranjem. [29, str. 2], [42, str. 9]

Uz navedene prednosti navodi se i prethodno spomenut intuitivan i jednostavan model podataka. Najveći nedostatak grafovskih baza podataka je nepostojanje standardnog upitnog jezika. Relacijske baze podataka imaju standardiziran upitni jezik SQL (engl. „Structured Query Language“), s druge strane većina grafovskih baza podataka ima svoj vlastiti upitni jezik ili svoju jedinstvenu sintaksu što ih čini kompleksnijim za učenje i otežava migraciju baze podataka. Neki od najpopularnijih upitnih jezika su Cypher, Gremlin, SPARQL i GraphQL. Već je navedeno da su grafovske baze podataka relativno nova tehnologija, što nas dovodi do drugog nedostatka. Brz razvoj otežava odabir najbolje tehnologije jer svaki SUGBP uvodi nove funkcionalnosti koje ga čine boljim od drugih, ali kako su te funkcionalnosti nove postoji i mogućnost grešaka u radu što je vrlo teško testirati. Posljednji nedostatak odnosi se na neke grafovske baze podataka koje nisu horizontalno skalabilne, što znači da se podaci ne mogu spremati na više servera. [46]

4. Neo4j

Neo4j je, NoSQL, nativna grafovska baza podataka otvorenog koda koja podržava ACID transakcije te je javno dostupna od 2007. godine (najnovija dostupna verzija je 4.3). Ovaj SUGBP je većinski implementiran u programskom jeziku Java i nešto manjim dijelom u jeziku Scala. Budući da se radi o softveru otvorenog koda, izvorni kod dostupan je na git[1] repozitoriju. Postoje dvije verzije Community Edition i Enterprise Edition, a razlika je u tome što je Enterprise verzija naprednija i ima dodatne funkcionalnosti kao što je sigurnosna kopija podataka, detaljniji monitoring sustava i druge, ali uključuje i dio komponenti koje nisu otvorenog koda te je, za Enterprise verziju, potrebna licenca. Što se tiče modela podataka, Neo4j implementira model označenog grafa sa svojstvima te je nativna grafovska baza podataka u punom smislu te riječi. U poglavlju 3.8.1 je objašnjeno na koji način Neo4j fizički pohranjuje podatke, a uz to osigurava i nativno procesiranje (posjeduje svojstvo susjedstva bez indeksa). Trenutno se smatra vodećom grafovskom bazom podataka. [43], [47]

Slika 20: Neo4j logotip (Izvor: https://neo4j.com/brand/)

Pojedinosti koje Neo4j SUGBP čine najpopularnijim su: [43], [48]

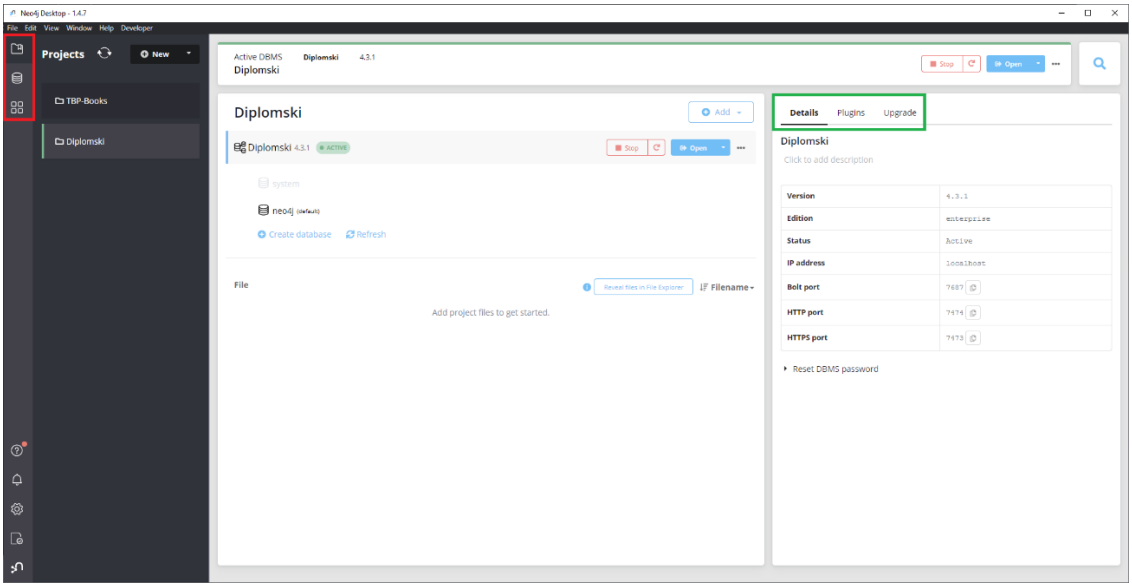
- Intuitivan model podataka
- Nativno procesiranje i pohrana podataka
- Podrška za ACID transakcije
- Konstantne performanse izvođenja upita čak i uslijed rasta podataka
- Fleksibilna shema označenog grafa sa svojstvima
- Ekspresivan i intuitivan upitni jezik – Cypher
- Integracija s većinom popularnih programskih jezika poput Jave, Pythona, .NET-a, JavaScripta i drugih
- Lako se ući i koristi
- Podrška za učitavanje podataka iz raznih izvora
- Velika i najaktivnija zajednica za grafovske baze podataka

[1] Git repozitorij Neo4j grafovske baze podataka: https://github.com/neo4j/neo4j

Kao i svaka tehnologija, Neo4j SUGBP, posjeduje i određene slabosti te su to prema [47] vertikalna skalabilnost (podaci se nalaze na jednom serveru), tj. nema podršku za horizontalnu skalabilnost, postoji gornja granica veličine baze podataka (deseci milijardi čvorova, svojstava i veza u jednom grafu) i ne postoji sigurnost na razini podataka niti podrška za eksplicitnom enkripcijom podataka što znači da aplikacija koja koristi Neo4j grafovsku bazu podataka mora odraditi taj dio ako je potrebna enkripcija podataka.

4.1. Neo4j platforma

Budući da je riječ o SUGBP-u, Neo4j grafovska baza podataka je središnja komponenta cijele Neo4j platforme. Nadalje za pristup bazi podataka koristi se **Neo4j Desktop** aplikacija pomoću koje upravljamo projektima i bazama podataka koju je potrebno instalirati. Unutar aplikacije mogu se kreirati projekti te unutar pojedinog projekta i same baze podataka koje su lokalne. Možemo se povezati i na udaljenu bazu podataka koja se nalazi na serveru za što nam je potreban URL do te baze i podaci za prijavu. Kreiranjem lokalne baze podataka se zapravo kreira instanca **Neo4j Servera** koja tada sadrži bazu podataka, od verzije 4.x dodana je podrška za više baza podataka unutar jedne takve instance. [49], [50]

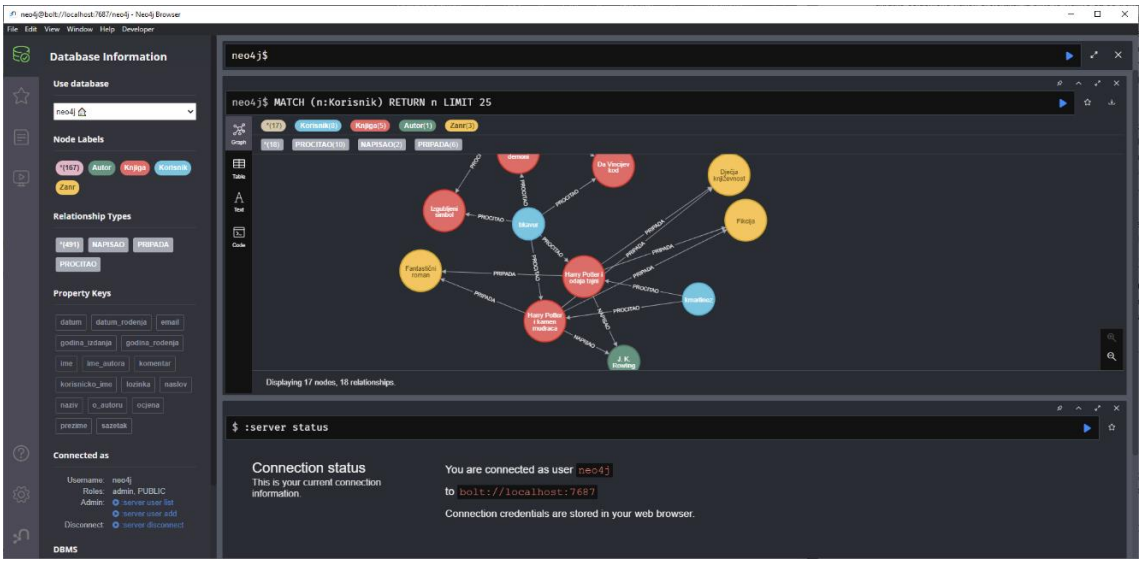


Slika 21: Neo4j Desktop aplikacija [vlastita izrada]

Prethodna slika prikazuje izgled Neo4j Desktop aplikacije. S lijeve strane imamo izbornik uokviren crvenim pravokutnikom gdje se nalaze projekti (prikazani na slici 21), verzije SUBP-a gdje možemo pronaći kreirane baze prema verziji i posljednje su aplikacije koje će biti ukratko objašnjene nakon projekata. Iz slike je vidljivo da su trenutno kreirana dva projekta i

odabran je projekt pod nazivom „Diplomski“, za kreiranje novog projekta jednostavno kliknemo na gumb „New“ te dodijelimo ime projektu. Nakon toga možemo kreirati bazu podataka klikom na gumb „Add“ i odabrati želimo li kreirati lokalnu bazu ili se povezati na udaljenu. U ovom slučaju kreirana je lokalna baza s nazivom „Diplomski“. Nakon kreiranja baze, da bi ju koristili, potrebno je istu i pokrenuti što je već napravljeno klikom na gumb „Start“ nakon čega baza postaje aktivna, tj. pokrene se kreirana instanca servera i na vrhu se prikaže aktivna baza „Active DBMS“. Kada smo gotovi s radom bazu je potrebno zaustaviti što se radi pritiskom na gumb „Stop“. S desne strane je zelenom bojom uokviren izbornik koji sadrži detalje odabrane instance baze podataka. Budući da se ovdje radi o novokreiranoj bazi (prazna je) fale detalji kao što je ID baze te broj čvorova, veza, oznaka i tipova veza koji se nalaze u bazi. Druga stavka izbornika su dodaci (engl. „plugins“) koji se mogu instalirati te će biti kasnije pojašnjeni. Posljednja opcije je nadogradnja (engl. „Upgrade“) koja omogućuje nadogradnju odabrane instance baze na noviju verziju.

Neo4j Browser je aplikacija koja nam pruža grafičko sučelje za interakciju s bazom podataka. Omogućuje nam izvršavanje Cypher upita, unos podataka u bazu i rad s podacima, vizualizaciju podataka i upravljanje korisnicima baze i njihovim ulogama. Aplikacija se može koristiti u desktop verziji ili preko web preglednika. Nakon što je baza ili server pokrenut u detaljima možemo vidjeti HTTP port na kojem je baza dostupna (vidi sliku 21), npr. http://127.0.0.1:7474. Unosom navedene adrese u web preglednik otvara se Neo4j Browser sučelje pomoću kojeg imamo pristup bazi podatak uz unos podataka za prijavu na tu instancu servera. Što se tiče vizualizacije podataka, nakon izvršavanja Cypher upita, rezultat je vidljiv u obliku grafa, tablice ili teksta te se može odabrati željeni prikaz klikom na ikone s lijeve strane okvira na kojem je prikazan rezultat upita. Rezultate možemo i izvesti u CSV i JSON formatu, a ako je riječ o grafičkom prikazu tada i kao sliku. [49], [51]



Slika 22: Neo4j Browser [vlastita izrada]

Neo4j Bloom iduća je aplikacija koja nam omogućuje grafički pregled i analizu podataka bez potrebe za poznavanjem upitnog jezika. Za pretragu se mogu pisati uzorci slični pitanjima koja bi mi postavili za dohvat određenih informacija, recimo da nas zanima tko je autor neke knjige i kojem žanru ona pripada tada jednostavno možemo napisati „Autor Knjiga Zanr“ ili „Knjiga Autor Knjiga Zanr“. Kao rezultat dobit ćemo graf koji prikazuje sve knjige, njihove autore i žanrove kojima pripadaju. Za svaki čvor i vezu možemo vidjeti detalje te ažurirati podatke ako je to potrebno. Detalji čvora, osim svojstava, uključuju i prikaz veza koje taj čvor posjeduje i susjednih čvorova s kojima je povezan. [49], [52]

Posljednja veća aplikacija je **Neo4j ETL Tool** pomoću koje možemo uvesti podatke iz relacijske baze podataka. Postoji nekoliko SURBP-ova koji su uključeni u aplikaciju, ali je moguće ručno povezati i ostale preko JDBC (engl. „Java Database Connectivity““) Driver-a. ETL proces je poznat iz skladišta podataka gdje se podaci iz raznih izvora preoblikuju i pohrane u skladište podataka. Svrha procesa je, u ovom slučaju, transformirati podatke iz relacijske baze podataka u Neo4j grafovsku bazu podataka. Kroz taj proces, alat sam napravi transformaciju relacijskog modela podataka u model označenog grafa sa svojstvima te nakon toga učita podatke u kreirani model. Tijekom procesa moguće je ručno promijeniti nazive veza i svojstava te tipove podataka ako automatski prepoznat model nije potpuno točan. [53]

Postoje i druge aplikacije koje su izradili Neo4j Labs, partneri Neo4j-a i drugi iz Neo4j zajednice. Aplikacije možemo naći unutar Neo4j Graph Apps Gallery-a. Neke aplikacije su edukacijskog tipa koje nam daju primjere baza koje su korisnici kreirali za određenu domenu, primjere algoritama temeljenih na grafovima i slično. Neke od takvih aplikacija su Graph Data Science Playground, Graph Gallery, GraphQL Architect. Druge aplikacije poput Halin i Query Log Analyser su za nadzor i analizu baze podataka. Treće služe vizualizaciji grafa i neke od njih su NeoMap Map Visualizer i Graphlytic Desktop. Postoji puno aplikacija koje se mogu naći u toj galeriji i svaka nam pruža neke nove mogućnosti. Isto tako možemo sami kreirati aplikaciju te na taj način pomoći cijeloj zajednici i unaprjeđenju Neo4j platforme. [54]

4.1.1. Dodaci ili biblioteke (engl. „Plugins“)

Ovisno o verziji baze podataka dostupni su određeni dodaci koji mogu olakšati rad s bazom podataka i proširiti dostupne funkcionalnosti, a najpopularniji su:

- **APOC** (engl. „Awesome Procedures on Cypher“) je biblioteka koja se sastoji od preko 450 funkcija i procedura koje se često koriste za integraciju podataka, konverziju podataka, izvršavanje algoritama temeljenih na grafovima i za druge namjene. Sve funkcije ili procedure pozivaju se putem Cyper upitnog jezika ili se mogu pozivati direktno iz aplikacije i skripte. Za poziv procedura koristi se klauzula ili naredba CALL. Budući da navedena biblioteka ima puno funkcija, preporuka je da se pogleda postoji

li već implementirana funkcionalnost koju trebamo prije nego što sami krenemo s implementacijom svog rješenja. [55]

- **GDS** (engl. „Graph Data Science Library“) biblioteka kreirana je s ciljem pružanja naprednih analitičkih mogućnosti vezanih uz algoritme temeljene na grafovima. Biblioteka sadrži algoritme za otkrivanje zajednica, pronalaženje sličnih čvorova, obilazak grafa i predviđanje veza. Svi algoritmi su optimizirani za rad s velikom količinom podataka i paralelno izvođenje. [56]
- **Neo4j Stremas** je isto poznatija biblioteka koja omogućuje integraciju Neo4j baze podataka s Apache Kafka platformom.

Postoje i druge biblioteke poput Neosemantics koja omogućuje korištenje RDF-a u Neo4j bazi podataka i GraphQL biblioteke (dostupna samo u Neo4j verziji 3.5) koja omogućuje izvršavanje upita pisanih u jeziku GraphQL tako da ih prevodi u Cypher upite. Sve navedene biblioteke je vrlo lako instalirati, potrebno je na odabranoj bazi otići na „Plugins“ u izborniku označenom zelenom bojom na slici 21, odabrati željenu biblioteku i kliknuti na gumb „Install“. Potrebno je voditi računa da u trenutku instalacije baza nije pokrenuta. [57]

4.2. Cypher

U svijetu baza podataka poznata su dva tipa upitnih jezika: imperativni i deklarativni upitni jezici. Imperativnim jezicima definiramo ono što želimo postići i na koji način to želimo postići, točnije detaljno „korak po korak“ definiramo postupak kojim želimo nešto postići. Takvi jezici su vrlo kompleksni i potrebno ih je dobro poznavati prije nego što ih koristimo kako bi izbjegli potencijalne pogreške. Ako smo samo jedan dio pogrešno definiramo, rezultat će biti neispravan. Neki poznati imperativni programski jezici su C, Java i Python. Što se tiče upitnih jezika za grafovske baze podataka Gremlin i Java API za Neo4j posjeduju imperativne značajke. Korištenjem imperativnih jezika imamo veću kontrolu nad izvršavanjem određenog zadatka i ako pravilno definiramo postupak kojim želimo nešto postići to će se na taj način i izvršiti, bez iznenađenja. Moramo biti vrlo pažljivi prilikom korištenja imperativnih jezika jer korisnici često rade greške u definiranju procesa izvršavanja i stoga je potrebno, nakon izvršavanja, provjeriti da li je sve onako kako smo željeli. S druge strane u deklarativnim upitnim jezicima samo definiramo što želimo postići bez potrebe da definiramo kako to želimo postići. Recimo da želimo dohvatiti neke podatke iz baze podataka, tada samo definiramo što želimo dohvatiti i prepustimo SUBP-u da se pobrine na koji način će dohvatiti podatke. Fokus deklarativnih jezika je na krajnjem rezultatu, a ne na postupku kako doći do njega. Poznati deklarativni programski jezici su Ruby, R i Haskell. Najpoznatiji deklarativni upitni jezik je SQL koji se koristi u relacijskim bazama podataka, što se tiče grafovskih baza podataka poznatiji

deklarativni upitni jezici su Cypher, SPARQL i Gremlin (koji posjeduje i imepretivne značajke). Možemo uočiti da su deklarativni jezici jednostavniji za korištenje, korisnici ga mogu lakše shvatiti i često nam daje bolje i brže rješenje od onog koje bi sami implementirali. Naravno, postoje i određeni nedostaci deklarativnih jezika i to je vrlo mala do nikakva kontrola nad procesom izvršavanja upita, ako postoji neka greška moramo se osloniti na kreatora jezika da otkloni problem i ako postoji funkcionalnost koju jezik ne podržava, a potrebna nam je tada ne možemo ništa učiniti. Ne postoji bolji jezik, već odabir ovisi o slučaju za koji ga trebamo koristiti. Ako nam je potrebna bolja kontrola i preciznost onda je imperativni jezik bolji odabir, a ako nam je u fokusu produktivnost, brzina i jednostavnost tada je bolja opcija deklarativan jezik. [29, str. 19 i 20]

Cypher je deklarativni grafovski upitni jezik, inspiriran SQL upitnim jezikom, koji koristi ASCII-Art sintaksu za opisivanje uzoraka u grafu. Razvio ga je Neo4j i optimiziran je za rad s grafovima te se koristi za pohranu, ažuriranje, brisanje i dohvat podataka iz grafosvke baze podataka. Originalno je kreiran za Neo4j grafovsku bazu podataka, ali je krajem 2015. godine otvoren za uporabu i u drugim bazama preko „openCypher“ projekta. Cilj tog projekta je standardizirati Cypher upitni jezik. [58]

Ranije smo se upoznali s modelom označenog grafa sa svojstvima i elementima od kojih se sastoji, no važno je napomenuti da ti jednostavni elementi (čvorovi i veze) zajedno čine obrazac ili uzorak (engl. „pattern“). Pomoću uzoraka mogu se opisati jednostavniji i složeniji putevi unutar grafa. Za izvršavanje upita koriste se algoritmi obilaska ili prolaska grafom te se traži uzorak koji je zadan upitom. Kao rezultat upita vraća se podgraf zadanog uzorka ili više njih u slučaju da postoji veći broj podgrafova koji odgovaraju traženom uzorku. Cypher je također baziran na uzorcima te je dizajniran sa svrhom pronalaženja raznih uzoraka u podacima. Takav pristup pretrazi podataka temeljen na definiranju uzoraka je vrlo intuitivan, kao i sam model označenog grafa sa svojstvima, što ga čini jednostavnim za korištenje i lakim za učenje. [59]

4.2.1. Sintaksa i naredbe

Da bi mogli koristiti upitni jezik potrebno je poznavati njegovu sintaksu i naredbe ili klauzule pomoću kojih se izvršavaju razne operacije. U ovom poglavlju upoznat ćemo se s osnovnim naredbama koje se koriste za čitanje (pretragu) i pisanje podataka u bazu.

Kako bi se objasnila sintaksa upita ili što upit radi mogu se koristiti komentari. Oni se dodaju tako da se na početku linije stavi // te je sav tekst nakon tih znakova komentiran. Možemo uočiti da je sam način komentiranja sličan onome u većini programskih jezika. Čvorovi su glavni elementi i reprezentiraju se unutar oblih zagrada $(čvor)$, postoji i anonimni čvor koji reprezentiramo praznim zagradama i na takav čvor se ne možemo referencirati niti ga vratiti u

rezultatu upita. Temeljem modela označenog grafa sa svojstvima svaki čvor ima svoju oznaku. Preko oznaka grupiramo čvorove i preko njih specificiramo određene tipove entiteta u grafu što su u SQL-u zapravo tablice, tj. njihovi nazivi $(:Korisnik)$. Svakom čvoru možemo dodati i varijablu kako bi se kasnije u upitu mogli referencirati na taj čvor i eventualno ga vratiti u rezultatu upita. U idućem primjeru imamo varijablu k kojom se referenciramo na čvor s oznakom $Korisnik$, $(k:Korisnik)$. Veze reprezentiramo preko strelica $-\!\!\rightarrow$ ili $\leftarrow\!\!-$ koje povezuju dva čvora. Ako nismo sigurni koji je smjer veze možemo koristiti i vezu koja nema određen smjer tako da stavimo samo dvije crtice bez označenog smjera $--$. Slično kao i čvorovi, veze mogu imati oznaku ili tip kako bi ih kategorizirali te im se može dodijeliti varijabla da bi se kasnije u upitu mogli referencirati na tu vezu. Primjere veze s varijablom r i oznakom PROCITAO je $-[r:PROCITAO]\rightarrow$. Uočimo da se veze definiraju unutar uglatih [ ] zagrada te da se njihov tip ili oznaka definira velikim slovima. Posljednji element koji je ostao iz modela označenog grafa sa svojstvima su svojstva koja mogu imati čvorovi i veze. Svojstava još nazivamo i atributima te nam služe za pohranu podataka koji detaljnije opisuju pojedini čvor i vezu. Podaci se unutar svojstava pohranjuju kao „ključ: vrijednost“ parovi te se definiraju unutar vitičastih { } zagrada. Ovisno o tipu podataka koji se pohranjuje, vrijednost se mora navesti unutar navodnika. Za primjer možemo dodati atribute prethodno navedenom čvoru Korisnik $(k:Korisnik\ \{ime:"Bojam", godina\_rodenja:1995\})$. Temeljem navedene sintakse primjer uzorka može biti: $(k:Korisnik\ \{ime:"Bojan", godina\_rodenja:1995\})-[r:PROCITAO]\rightarrow (b:Knjiga\ \{naslov:"Harry\ Potter\ i\ kamen\ mudraca"\})$. Vidimo da je primjer uzorka vrlo intuitivan te ga interpretiramo kao: korisnik Bojan je pročitao knjigu Harry Potter i kamen mudraca. Na takav jednostavan način možemo Cypher-u reći koji uzorak trebamo, ali su nam potrebne i određene naredbe kako bi znao želimo li taj uzorak i podatke spremiti u bazu ili ga pronaći unutar grafa. [59]

Kako bi mogli raditi s bazom podataka, dohvaćati i pisati podatke, potrebne su nam određene naredbe ili klauzule. Važnije naredbe za upravljane podacima preko Cypher upitnog jezika su: [60]

- **MATCH** – osnovna naredba za dohvat podataka. Koristi se za pronalazak zadanog uzorka u grafovskoj bazi podataka.
- **OPTIONAL MATCH** – slična prethodnoj naredbi MATCH, samo što koristi NULL za dijelove uzorka koji nisu pronađeni. Možemo reći da je ekvivalentan vanjskom spajanju (engl. „outer join“) u SQL-u.
- **RETURN** – definira što će se vratiti kao rezultat upita. Ponekad nam nisu potrebne sve informacije tako da možemo vratiti samo određene čvorove, veze ili neka njihova svojstva (atribute).

- **DISTINCT** – nastavno na klauzulu RETURN, klauzula DISTINCT ide nakon RETURN i vraća, identično kao i u SQL-u, samo jedinstvene vrijednosti. Znači da će se u rezultatu nalaziti jedinstvene (različite) vrijednosti i neće biti ponavljanja ili duplikata.
- **WITH** – ova klauzula omogućuje nam povezivanje više upita tako da se rezultati trenutnog upita na neki način koriste u idućem upitu. Isto tako možemo, nekim klauzulama kao što su LIMIT i ORDER BY, modificirati podatke prije nego ih prenesemo u drugi upit. Vrlo je važno napomenuti da se varijable koje nisu uključene u WITH klauzulu neće prenijeti na idući upit.
- **WHERE** – zapravo se ne smatra klauzulom za sebe, već je dodatak na klauzule MATCH, OPTIONAL MATCH i WITH. Korištenjem uz klauzulu WITH jednostavno filtrira rezultat. Nadalje, ako se koristi uz klauzule MATCH i OPTIONAL MATCH tada služi kao ograničenje nad definiranim uzorkom te se smatra dijelom uzorka (npr. ograničenje nad svojstvom da je jednako nekoj vrijednosti ili veće/manje od određene vrijednosti). Može se koristiti za provjeru tekstualnog (string) svojstva počinje li (STARTS WITH) ili završava (ENDS WITH) s određenim znakom ili sadrži li (CONTAINS) određene znakove. Nakon WHERE možemo definirati i podupite.
- **ORDER BY** – klauzula koja slijedi nakon klauzule RETURN te nam omogućuje sortiranje rezultata i na koji način ga želimo sortirati (silazno ili uzlazno). Sortiranje je moguće samo nad svojstvima čvorova i veza, ne mogu se sortirati same veze ili čvorovi. Klauzula se ne može koristiti zajedno s agregirajućim funkcijama iz razloga da se spriječi mogućnost promjene rezultata, što znači da se ovom klauzulom može mijenjati samo poredak.
- **SKIP** – definira koliko redova želimo priskočiti kod prikaza rezultata upita, tj. od kojeg reda se prikazuje rezultat. Gotovo uvijek se koristi uz klauzulu ORDER BY jer se inače ne može garantirati identičan redoslijed rezultata.
- **LIMIT** – ograničava koliko redova je vidljivo u rezultatu. Može se koristiti zajedno s klauzulom SKIP da se omogući paginacija rezultata, recimo da dohvatimo prvih 5 redova za prvu stranicu tablice (samo LIMIT), drugih 5 redova za drugu stranicu (LIMIT i SKIP) i tako dalje.
- **UNWIND** – omogućuje nam da listu rastavimo na redove te zahtjeva da dodijelimo ime tim vrijednostima korištenjem klauzule **AS**. Redove, tj. vrijednosti koje nam je vratio neki izraz možemo pretvoriti u listu korištenjem agregirajuće funkcije collect(). Primjer korištenja ove klauzule može biti kreiranje liste bez duplikata, recimo da imamo listu koja sadrži ponavljajuće podatke taka možemo napraviti UNWIND te liste, dodijeliti ime podacima, koristiti DISTINCT nad njima da maknemo duplikate te na kraju pozvati funkciju collect() nad tim podacima da ih vratimo natrag u listu.

- **CREATE** – kao što nam govori i sam naziv klauzule, koristi se za kreiranje čvorova i veza u bazi. Možemo ih kreirati zasebno ili definirati uzorak koji će tada kreirati sve definirane čvorove i veze.
- **DELETE** – briše čvorove, veze ili cijeli uzorak (put) iz grafovske baze podataka. Ovom naredbom nije moguće brisati čvor koji sadrži vezu te ih je potrebno prethodno obrisati zasebno ako se želi obrisati taj čvor.
- **DETACH DELETE** – samo DELETE ne može obrisati čvor koji sadrži veze te je zbog toga kreirana ova klauzula. Ona briše čvor i sve njegove veze.
- **SET** – služi za ažuriranje oznaka čvorova i svojstava čvorova i veza, ovom klauzulom možemo dodati i novo svojstvo vezi i čvoru. Pomoću ove naredbe možemo i maknuti (obrisati) svojstvo tako da mu vrijednost postavimo na NULL ili sva svojstva nekog čvora ili veze tako da definiramo prazne vitičaste zagrade.
- **REMOVE** – koristi se za micanje oznaka čvorovima i za micanje svojstava s čvorova i veza. Ako želimo obrisati sva svojstva tada moramo koristiti klauzulu SET.
- **MERGE** – možemo gledati kao kombinaciju klauzula CREATE i MATCH jer osigurava da traženi uzorak postoji u grafu. Što znači da će kreirati čvorove i veze iz zadanog uzorka ako one ne postoje ili ih dohvatiti u slučaju da postoje. Možemo koristiti ON CREATE i ON MATCH za finiju kontrolu izvođenja te ovisno o tome što se izvršilo definirati željena svojstva čvorova i veza.
- **UNION** – omogućava kombinaciju rezultata iz dva ili više upita te nam daje konačan rezultat koji sadrži sve rezultate pojedinih upita bez duplikata. Ako želimo baš sve rezultate pojedinih upita uključujući i duplikate tada koristimo klauzulu UNION ALL.

Postoje i brojne funkcije[2] pomoću kojim radimo s raznim tipovima podataka, učitavamo datoteke, radimo izračune i slično. Na primjer, za provjeru postoji li određeni čvor, veza ili svojstvo koristi se funkcija exists() koja je s trenutnom verzijom označena kao „deprecated“ te će se u nekoj novijoj verziji maknuti. Kao alternativa koristi se izraz IS NOT NULL nakon WHERE.

[2] Funkcije u Cypher upitnom jeziku: https://neo4j.com/docs/cypher-manual/current/functions/

4.2.2.Indeksi

U Cypher-u i Neo4j grafovskoj bazi podataka indeksi imaju istu namjenu, kao i u SQL-u i relacijskim bazama podataka. Velika količina podataka koju je potrebno obraditi prilikom izvršavanja upita može imati veliki utjecaj na performanse izvođenja istih. Indeksi dolaze kao rješenje koje može znatno ubrzati izvršavanje upita i unaprijediti performanse upita žrtvujući malo mjesta za pohranu. Naravno, treba biti oprezan s korištenjem indeksa jer oni nešto usporavaju pisanje (unos) i ažuriranje podataka čime možemo proizvesti i suprotan efekt kojim ćemo narušiti performanse ako ih nepotrebno kreiramo. Jednom kad se indeks kreira, SUBP će dalje njime upravljati i ažurirati ga. U Neo4j SUGBP-u indeksi se mogu kreirati jedino nad svojstvima čvorova i veza, može biti kreiran samo nad jednim svojstvom ili nad više njih. Kreiraju se naredbama: [61]

$$CREATE\ INDEX\ IF\ NOT\ EXIST\ FOR\ (n{:}\ OznakaČvora)\ ON\ (n.imeSvojstva)$$

$$CREATE\ INDEX\ IF\ NOT\ EXIST\ FOR\ ()-"["r{:}\ TIP\_VEZE\ "]"-()\ ON\ (r.imeSvojstva)$$

Ako se indeks kreira nad više svojstava ili atributa tada ih samo navodimo jedno iza drugog te ih odvajamo zarezom. Indeksi ubrzavaju pretraživanje jer se podaci unutar njega pohranjuju sortiranim redoslijedom te ih je iz tog razloga puno lakše i pronaći. Uglavnom se koriste B-tree indeksi koji imaju strukturu B+ stabla koje se sastoji od listova, unutarnjih čvorova i korijena od čega svaki ima određen broj elemenata. Indeksi se sastoje od ključa i reference ili pokazivača koji sadrži adresu sloga u osnovnoj datoteci gdje se može naći zapis za pripadni ključ. Možemo reći da su indeksi zapravo manje datoteke koje se koriste za pretraživanje glavne datoteke. Dakle u njih je pohranjen maji dio podataka, točnije samo oni podaci vezani uz svojstvo ili atribut nad kojim je indeks kreiran te se stoga može puno brže pretražiti nego svi zapisi, čvorovi ili veze, u glavnoj datoteci ili u bazi podataka. Ako se dohvaća samo vrijednost traženog svojstva recimo za provjeru postoji li već u bazi tada se taj podataka može dohvatiti direktno iz indeksa, ali ako trebamo i druge podatke o čvoru koji sadrži to svojstvo tada se preko pokazivača dođe do zapisa u osnovnoj datoteci gdje se može lako pronaći ostatak podataka.

5. Algoritmi temeljeni na grafovima

Ranije je navedeno da se upiti nad grafovskom bazom podataka izvršavaju tako da upitni jezik koristi neki od algoritama prolaska ili obilaska grafa te je to jedna od četiri glavne skupine u koje možemo svrstati algoritme temeljene na grafovima. U skupinu algoritama za obilazak grafa svrstavamo iduće algoritme: algoritam traženja u dubinu (engl. „Depth First Search“), algoritam traženja u širinu (engl. „Breadth First Search“), algoritmi za pronalaženje najkraćeg puta (Dijkstra, A*, Yen's i drugi), algoritmi za pronalazak minimalnog razapinjućeg stabla i algoritam nasumične šetnje grafom. Drugu skupinu čine algoritmi detekcije ili otkrivanja zajednica (engl. „Comunity Detection“), a u njih svrstavamo algoritam brojanja trokuta (engl. „Triangle Count“), algoritam koeficijenta grupiranja (engl. „Clustering Coefficient“), algoritmi povezanih i snažno povezanih komponenti i algoritam Louvianove modularnosti. Treću skupinu algoritama temeljnih na grafovima nazivamo algoritmi centralnosti u koju ubrajamo PageRank algoritam, algoritam ili mjera stupnja centralnosti (engl. „Degree Centrality“), mjera blizine čvorova (engl. „Closeness Centrality“) i njene varijacije (Harmonic Closeness Centrality, Dangalchev Closeness Centrality, Wasserman & Faust Closeness Centrality), mjera povezanosti čvorova (engl. „Betweenness Centrality“) i njene varijacije (Approximate Betweenness Centrality). Posljednju skupinu algoritama čine algoritmi sličnosti i u njih spadaju Euklidska udaljenost, algoritam Jaccardovog koeficijenta sličnosti, algoritam kosinusne sličnosti (engl. „Cosine Similarity“) i algoritam Pearsonovog koeficijenta sličnosti. [62, str. 27, 41, 82 i 117], [63, str. 8]

U sustavima za preporuke najzanimljiviji su algoritmi sličnosti jer se traže slični korisnici i sličan sadržaj onome kojem pregledavamo te se na temelju toga rade preporuke. Sličnost možemo mjeriti na razne načine, ovisno o podacima s kojima radimo. U sklopu ovog rada izrađen je sustav za preporuke knjiga, stoga se sličnost može mjeriti temeljem žanra u koji knjiga spada i autora koji ju je napisao uz to može se iskoristiti i ocjena korisnika kako bi se mogle dati kvalitetnije preporuke (kako je korisnik sličan nama ocijenio određenu knjigu).

5.1. Jaccardov indeks ili koeficijent sličnosti

Ovaj algoritam razvio je Paul Jaccard, a koristi se za mjeru sličnosti između dvaju skupova. Formula kojom se izračunava Jaccardov koeficijent sličnosti glasi: [2, str. 155], [64]

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Iz formule vidimo da se indeks sličnosti izračunava kao veličina presjeka podijeljena s veličinom unije dvaju skupova i kao rezultat dobijemo broj između 0 i 1 koji pokazuje koliko su ti skupovi slični. 1 nam govori da su skupovi identični dok nam 0 govori da ti skupovi nemaju zajedničkih elemenata. Pomoću ovog algoritma možemo izračunati sličnost korisnika na temelju knjiga koje su pročitali, ali i sličnost knjiga temeljem njihove pripadnosti žanru, autora koji ju je napisao i slično. [64]

U praktičnom dijelu rada bit će implementiran algoritam koji će mjeriti sličnost knjiga na način da će tražiti slične knjige onoj koju korisnik trenutno gleda. Dakle potrebo je pronaći sve knjige koje spadaju u isti žanr ili koje je napisao isti autor te se tada računa Jaccardov indeks za pojedinu knjigu. Računa se na način da će skup A činiti svi žanrovi i autori koji su povezani s knjigom za koju tražimo slične knjige, a skup B se mijenja za svaku pronađenu knjigu i čine ga svi žanrovi i autori koji su povezani s njom. Presjek će tada čini zajednički žanrovi i autori, a uniju svi žanrovi i autori. Recimo da je knjige koje uspoređujemo napisao isti autor, prva spada i žanrove fikcija, dječja književnost i fantastično roman, a druga spada u te žanrove, ali još i u triler i misteriju tada će presjek biti 4, a unija 6 što će nam u konačnici dati Jaccardov indeks u iznosu od 0.667 što nam govori da su te knjige u nekoj mjeri slične.

5.2. Kosinusna sličnost (engl. „Cosine Similarity“)

Ovim algoritmom izračunavamo sličnost na način da se računa kosinus kuta između dva n-dimenzionalna vektora u n-dimenzionalnom prostoru što je zapravo skalarni umnožak dvaju vektora podijeljen umnoškom njihovih duljina. Formula za izračun kosinusne sličnosti je: [2, str. 159 i 160], [65]

$$sličnost(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Kao rezultat dobivamo vrijednost između -1 i 1 gdje nam 1 označava potpunu sličnost komponenti, a -1 da nisu nimalo slični. Algoritam treba neku mjeru, kao što je ocjena, kako bi se mogla izračunati sličnost. S obzirom na to, možemo izračunati sličnost korisnika ovisno o tome kako su ocijenili pojedinu knjigu koju su čitali. [65]

Kasnije, u praktičnom dijelu rada, implementacijom ovog algoritma mjerit će se sličnost korisnika temeljem knjiga koje su pročitali i ocijenili. Pronaći će se sve knjige koje je korisnik pročitao te će se uspoređivati s drugim korisnicima koji su pročitali te iste knjige. Mjera će biti

točnija što je veći broj zajedničkih knjiga. Kao primjer možemo pretpostaviti da su dva korisnika pročitala pet istih knjiga i prvi je knjigama dao ocjene 3, 4, 5, 4, 4, a drugi korisnik je istim knjigama dao ocjene 4, 5, 5, 3, 5. Tada bi sličnost izračunali kao:

$$sličnost(A,B) = \frac{3\times4+4\times5+5\times5+4\times3+4\times5}{\sqrt{3^2+4^2+5^2+4^2+4^2}\times\sqrt{4^2+5^2+5^2+3^2+5^2}} = \frac{89}{9.056\times10} = 0.982$$

Već smo iz zadanih ocjena mogli vidjeti da su one vrlo slične, tj. da odstupaju za 1 ili su identične te smo mogli zaključiti da su ti korisnici vrlo slični što smo i pokazali prethodnim izračunom kosinusne sličnosti. Kao rezultat dobili smo iznos 0.982 što nam govori da su ti korisnici vrlo slični. Sad možemo prvom korisniku preporučiti sve one knjige koje je pročitao drugi korisnik, ali treba voditi računa da rezultat sličnosti ne bude nizak. Kako je mjera od -1 do 1, ovaj rezultat je sasvim u redu i takva preporuka je solidna.

5.3. Pearsonov koeficijent sličnosti

Pearsonov koeficijent sličnosti računa se kovarijanca dva n-dimenzionalna vektora podijeljena s umnoškom njihove standardne devijacije. Formula za izračun glasi: [2, str. 162], [66]

$$sličnost(A,B) = \frac{cov(A,B)}{\sigma_A\sigma_B} = \frac{\sum_{i=1}^{n}(A_i-\overline{A})(B_i-\overline{B})}{\sqrt{\sum_{i=1}^{n}(A_i-\overline{A})^2\ \sum_{i=1}^{n}(B_i-\overline{B})^2}}$$

Što se tiče rezultata on je između -1 i 1 i skala je jednaka kao i kod prethodnog algoritma kosinusne sličnosti te za izračun treba neku mjeru kao što je ocjena. Kroz praktični dio rada će se pomoću ovog algoritma računati sličnost korisnika temeljem knjiga koje su oni pročitali i ocijenili. Ako uzmemo identične skupove kao u za izračun kosinusne sličnosti, kao rezultat dobit ćemo 0.395 čime možemo zaključiti da ti korisnici i nisu toliko slični i da se izračun sličnosti razlikuje ovisno o tome koji algoritam koristimo.

$$sličnost(A,B) = \frac{1}{1.414\times1.789} = 0.395$$

5.4. Mjera stupnja centralnosti (engl. „Degree Centrality“)

Mjera stupnja centralnosti spada u skupinu algoritama centralnosti i smatra se jednim od najjednostavnijih algoritama. Jednostavno se računa broj ulaznih i izlaznih veza iz čvora te se može koristi za pronalaženje popularnih čvorova. Algoritam je predstavio Linton C. Freeman 1979. godine u svom radu „Centrality in Social Networks: Conceptual Clarification“[3]. [62, str. 85]

Algoritam će u ovom radu biti korišten za davanje preporuka tako da će se gledati broj veza prema onim žanrovima i autorima koji interesiraju korisnika te na temelju toga naći slične knjige koje pripadaju tim žanrovima i koje je napisao taj autor. Tako će se za pojedinu komponentu (žanr i autor) izračunati broj veza prema knjigama koje je korisnik pročitao da bi se moglo odrediti koliko je preporučena knjiga slična knjigama koje je korisnik pročitao.

[3] Rad je dostupan na poveznici [Pristupano 18.7.2021.]:
https://www.bebr.ufl.edu/sites/default/files/Centrality%20in%20Social%20Networks.pdf

6. Sustav za preporuku knjiga

Odabrani SUGBP je prethodno opisan Neo4j, a aplikacijska domena je sustav za preporuke knjiga. U idućem potpoglavlju slijedi opis odabran aplikacijske domene nakon čega slijedi opis modela podataka, tj. modela označenog grafa sa svojstvima. Nakon toga slijedi opis implementacije samog sustava od kreiranja baze podataka do davanja preporuka korisnicima. Implementacija sustava bit će rađena u programskom jeziku Java te će biti korišten Spring Boot kao programski okvir za izradu aplikacije koja će imati i pripadno web sučelje za interakciju korisnika sa sustavom.

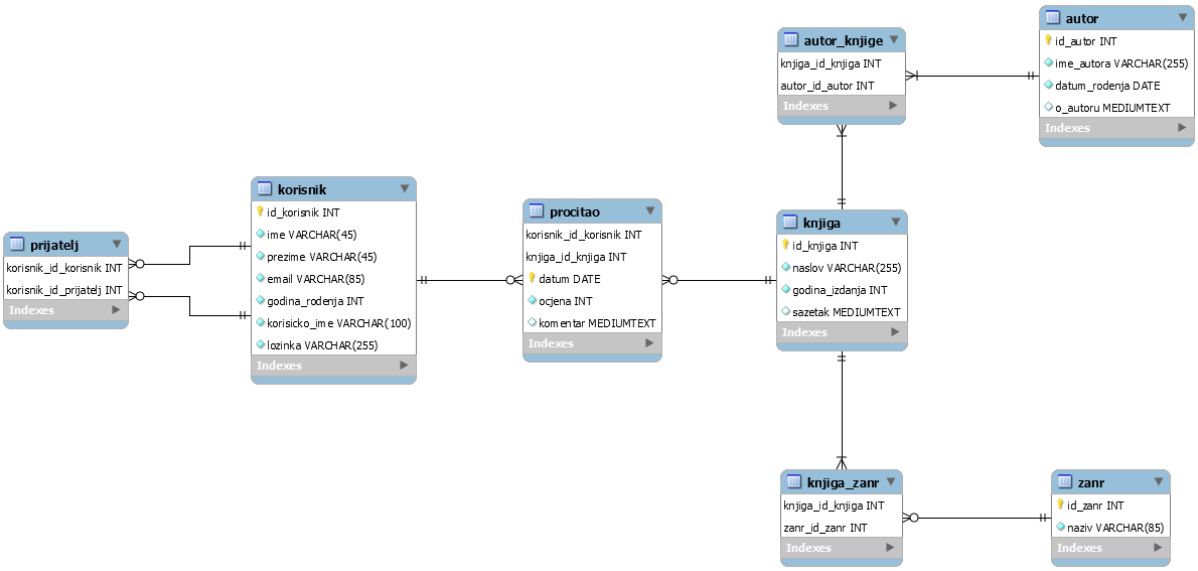
6.1. Aplikacijska domena – sustav za preporuku knjiga

Kao što je već i jasno iz samog naziva, svrha sustava je preporuka knjiga te ujedno čini i aplikacijsku domenu. Iz domene možemo uočiti da su nam glavni entiteti knjige i korisnici kojima će se knjige preporučivati. Da bi se izradila kvalitetna baza podataka potrebno je dobro poznavati aplikacijsku domenu, ali za potrebe ovog rada koristit će se osnovni podaci o korisnicima i knjigama. Aplikacija će podržavati prijavu postojećih i registraciju novih korisnika. Neregistrirani korisnici imat će mogućnost nepersonaliziranih preporuka na temelju sadržaja gdje će odabrati žanr koji voli ili svog omiljenog autora te će na temelju odabira dobiti preporuku. Uz to imat će preporuku temeljem prosječne ocjene knjige koju su dali korisnici sustava, godini izdanja knjige gdje će odabrati jedan od raspona i na kraju preporuku najčitanijih knjiga koja u obzir uzima čitanja od registriranih korisnika. Što se tiče prijavljenih korisnika, oni će imati iste preporuke samo što će se filtrirati knjige koje su već pročitali (neće im se preporučiti nešto što su već čitali). Naknadno na to, oni će imati i mogućnost pregleda svih knjiga i korisnika gdje će vidjeti koje su komentare pojedini korisnici dali za pojedinu knjigu i što su drugi korisnici čitali. Imaju mogućnost dodavanja prijatelja ako poznaju nekog korisnika te će im tada sustav moći dati preporuku temeljem knjiga koje čitaju njegovi prijatelji ili poznanici. Pregledom pojedine knjige korisnik će imati uvid u komentare i ocjene koje su korisnici ostavili za knjigu, moći će ocijeniti knjigu te će dobiti preporuke sličnih knjiga koje se temeljene na algoritmu Jaccardovog indeksa sličnosti što je preporuka na temelju sadržaja. Korisnik će imati još nekoliko preporuka na temeljenu sadržaja, od kojih će prva biti implementirana korištenjem algoritma centralnosti (mjera stupnja centralnosti) te dvije preporuke temeljem žanrova i autora knjiga koje je ocijenio (traže se svi žanrovi i autori temeljem knjiga koje je korisnik ocijenio te se preporučuju ostale knjige pojedinog autora i žanra koje još nije pročitao). Posljednje preporuke bit će temeljene na kolaborativnom filtriranju koje će koristiti algoritam kosinusne sličnosti i algoritam Pearsonovog koeficijenta sličnosti za

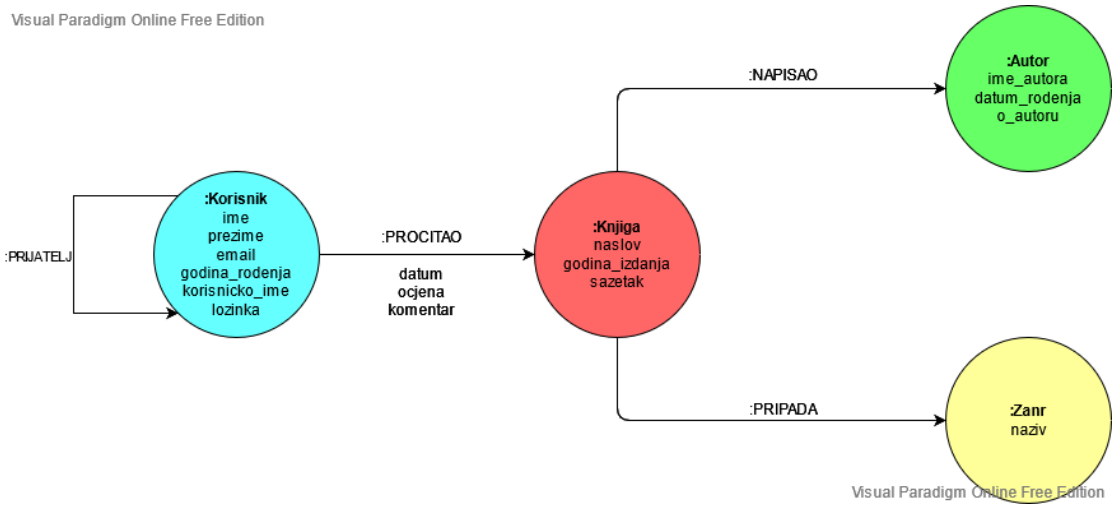
traženje sličnih korisnika i preporučiti knjige koje čitaju korisnici slični nama, tj. korisniku koji je trenutno prijavljen.

6.2. Model baze podataka

Ranije, u ovom radu, objašnjen je model označenog grafa sa svojstvima koji je najpopularniji grafovski model podataka te ga koristi i Neo4j SUGBP. Isto tako, spomenuta je i usporedba navedenog modela s relacijskim modelom podataka koji je, nakon normalizacije, nešto drugačiji nego početni model temeljen na našoj percepciji domene. Na idućim slikama prikazan je ERA model baze podataka i model označenog grafa sa svojstvima kako bi mogli uočiti razliku i vidjeti zašto je model označenog grafa sa svojstvima toliko intuitivan i jednostavan, a istovremeno i ekspresivniji.



Slika 23: ERA model [vlastita izrada]



Slika 24: Model označenog grafa sa svojstvima [vlastita izrada]

Na slici 23 vidljiv je ERA model baze podataka te možemo uočiti četiri jaka entiteta (korisnik, knjiga, autor i zanr) i četiri slaba entiteta (prijatelj, procitao, autor_knjige i knjiga_zanr). Slabi entiteti nastali su iz veza više naprema više koje povezuju navedene jake entitete. Model možemo protumačiti na način da jedan korisnik može pročitati više knjiga i jedna knjiga može biti pročitana od strane više korisnika, jedna knjiga može imati više autora i jedan autor može napisati više knjiga, jedna knjiga može pripadati u više žanrova i jednom žanru može pripadati više knjiga te jedan korisnik može imati više prijatelja. Nadalje, na slici 24 vidimo model označenog grafa sa svojstvima koji će biti korišten za implementaciju baze podataka. Možemo uočiti da je takav model stvarno vrlo jednostavan, intuitivan i pregledniji od prethodno definiranog ERA modela. Veza više naprema više se može jednostavno implementirati tako da kreiramo više veza iz pojedinog čvora, recimo da korisnik pročita više knjiga tada će iz čvora korisnik ići više veza tipa :PROCITAO na čvorove knjiga koje je on pročitao. Model je prikazan na način koji nam je prirodan za promatranje neke domene, ako bi radili skicu vrlo vjerojatno bi onda bila identična modelu koji je prikazan na slici 24. Moguća promjena je da bi vezi :NAPISAO promijenili smjer tako da ide od autora prema knjizi što je sasvim u redu, ali može se implementirati i na ovakav način. NoSQL baze podataka pa tako i grafovske baze podataka imaju fleksibilnu shemu što znači da vrlo lako možemo dodati nove atribute ako je to potrebno i proširiti postojeći model tako da dodamo novi čvor i vezu s pripadnom oznakom. Takav pothvat je kod relacijskih baza podataka vrlo kompleksan pogotovo ako baza već posjeduje veliku količinu pohranjenih podataka jer tada treba voditi računa da se podaci pravilno povežu i da se tijekom transformacije sačuvaju svi podaci.

6.3. Implementacija Neo4j grafovske baze podataka

Za izradu baze podataka korišten je Neo4j Desktop i Neo4j Browser koji su detaljnije objašnjeni u poglavlju 4.1. Prvo je kreiran novi projekt i unutar njega nova baza podataka. Nakon pokretanja, baza je dostupna na zadanim portovima i to su 7474 za HTTP, 7473 za HTTPS i 7678 za Bolt port. U postavkama je moguće promijeniti zadane portove, ako nam je neki već prethodno zauzet. Otvaranjem Neo4j Browser-a krećemo na kreiranje same baze podataka i dodavanje podataka u bazu.

6.3.1. Kreiranje čvorova, veza, indeksa i ograničenja

Čvorovi se pomoću Cypher upitnog jezika kreiraju naredbom CREATE. Sad će na konkretnom primjeru biti prikazano kako se kreiraju čvorovi i veze te kako se preko tih veza sami čvorovi i povezuju. Preporučena notacija za oznaku čvora je Camel-case s tim da je prvo slovo veliko.

```
CREATE ( bojan:Korisnik { ime:'Bojan', prezime:'Kavur',
email:'bkavur@foi.hr', godina_rodenja:1995, korisnicko_ime:'bkavur',
lozinka:'admin' } ) RETURN bojan;
```

Prethodni isječak koda kreira korisnika s navedenim atributima i njihovim vrijednostima te na kraju vraća kreiranog korisnika. Na identičan način kreira se knjiga, autor i žanr, samo što je potrebno navesti odgovarajuću oznaku čvora i atribute koje želimo dodati.

```
CREATE ( k:Knjiga {naslov:'Anđeli i demoni', godina_izdanja:2000,
sazetak:"Godinu dana prije nego što je riješio Da Vincijev kod …"} ) RETURN
k;

CREATE ( a:Autor {ime_autora:'Dan Brown', datum_rodenja:'22.06.1964.',
o_autoru:"Dan Brown (Exeter, New Hampshire, 22. lipnja 1964.) je američki
pisac triler romana, poznat po kontroverznom bestselleru Da Vincijev kod."}
) RETURN a;

CREATE ( z1:Zanr {naziv:'Kriminalistički roman'} ) RETURN z1;

CREATE ( z2:Zanr {naziv:'Triler'} ) RETURN z2;
```

Što se tiče veza, one povezuju čvorove i kreiramo ih tako da dohvatimo čvorove koje želimo povezati i između njih kreiramo novu vezu kojoj dodijelimo odgovarajuću oznaku. Za veze, preporučena notacija je da se sve piše velikim slovima i koristi donja povlaka za odvajanje riječi, npr. „:JE_NAPISAO“. Idući isječak koda Cypher upita prikazuje kreiranje veza :NAPISAO između knjige i autora, :PRIPADA između knjige i žanra i :PROCITAO između korisnika i knjige koja sadrži i dodatne atribute datum, ocjena i komentar.

```
MATCH (a:Knjiga), (b:Autor) WHERE a.naslov = 'Anđeli i demoni' AND
b.ime_autora = 'Dan Brown' CREATE (a)-[ r:NAPISAO ]->(b) RETURN a, type(r),
b;
```

```
MATCH (a:Knjiga), (b:Zanr) WHERE a.naslov = 'Anđeli i demoni' AND b.naziv =
'Kriminalistički roman' CREATE (a)-[ r:PRIPADA ]->(b) RETURN a, type(r), b;

MATCH (a:Knjiga), (b:Zanr) WHERE a.naslov = 'Anđeli i demoni' AND b.naziv =
'Triler' CREATE (a)-[ r:PRIPADA ]->(b) RETURN a, type(r), b;

MATCH (a:Korisnik), (b:Knjiga) WHERE a.ime = 'Bojan' AND b.naslov = 'Anđeli
i demoni' CREATE (a)-[ r:PROCITAO {datum:'21.05.2019.', ocjena: 5,
komentar:'Knjiga je super, preporučam svima da pročitaju!'}]->(b) RETURN a,
type(r), b;
```

Možemo definirati i uzorak pomoću kojeg u jednom koraku kreiramo korisnika, knjigu i autora te veze :PROCITAO i :NAPISAO. Tako ne moramo kasnije posebno kreirati veze već će Cypher kreirati cijeli uzorak odjednom.

```
CREATE ( katarina:Korisnik { ime:'Katarina', prezime:'Martinez',
email:'kmartinez@gmail.com', godina_rodenja:1997,
korisnicko_ime:'kmartinez', lozinka:'knjigee' } ) -[ r1:PROCITAO
{datum:'14.09.2019.', ocjena: 5, komentar:'Moja najdraža knjiga ikad!'}]->
( knjiga:Knjiga {naslov:'Harry Potter i kamen mudraca', godina_izdanja:1997
, sazetak:"Roditelji Harryja Pottera stradali su dok je još bio
jednogodišnja beba…"} ) -[ r:NAPISAO ]-> ( autor:Autor {ime_autora:'J. K.
Rowling', datum_rodenja:'31.07.1965.', o_autoru:"Joanne 'Jo' Rowling (Yate,
Gloucestershire, 31. srpnja 1965.), britanska je književnica znana po
svojoj fantastičnoj seriji romana Harry Potter…"} );
```

Na identičan način kreiran je i ostatak baze podataka koja u konačnici sadrži nešto više od 120 knjiga što je dovoljno za implementaciju sustava za preporuke. Vrlo je važno držati se sintakse prilikom korištenja Cypher upitnog jezika koja je detaljnije opisana u poglavlju 4.2.1 gdje je spomenuto koja vrsta zagrade se koristi za čvor, vezu i atribute.

Nadalje, kreirano je ograničenje UNIQUE nad korisničkim imenom što će nam osigurati da su vrijednosti jedinstvene, tj. da ne mogu postojati dva korisnika koja imaju isto korisničko ime. Ograničenje definiramo na idući način:

```
CREATE CONSTRAINT ON (k:Korisnik) ASSERT k.korisnicko_ime IS UNIQUE;
```

Kreiranjem ograničenja Neo4j SUGBP automatski kreira i jedinstveni indeks nad istim atributom. Kreirat ćemo i indeks nad naslovom knjige kako bi ubrzali pretragu knjiga jer su oni glavni entitet baze podatak i očekivano je da će se raditi brojne pretrage knjiga. U poglavlju 4.2.2 opisani su indeksi i sintaksa kako se kreiraju te se prema tome kreira kako slijedi:

```
CREATE INDEX IF NOT EXIST FOR (b:Knjiga) ON (b.naslov);
```

Za pregled kreiranih indeksa možemo pozvati proceduru db.indexes, a za pregled ograničenja proceduru db.constraints. Procedure pozivamo pomoću klauzule ili naredbe CALL, što u konačnici izgleda kao: $CALL\ db.indexes$. Drugi način, kojim se istovremeno ispišu indeksi i ograničenja je da upišemo : $schema$.

```
$ :schema
```

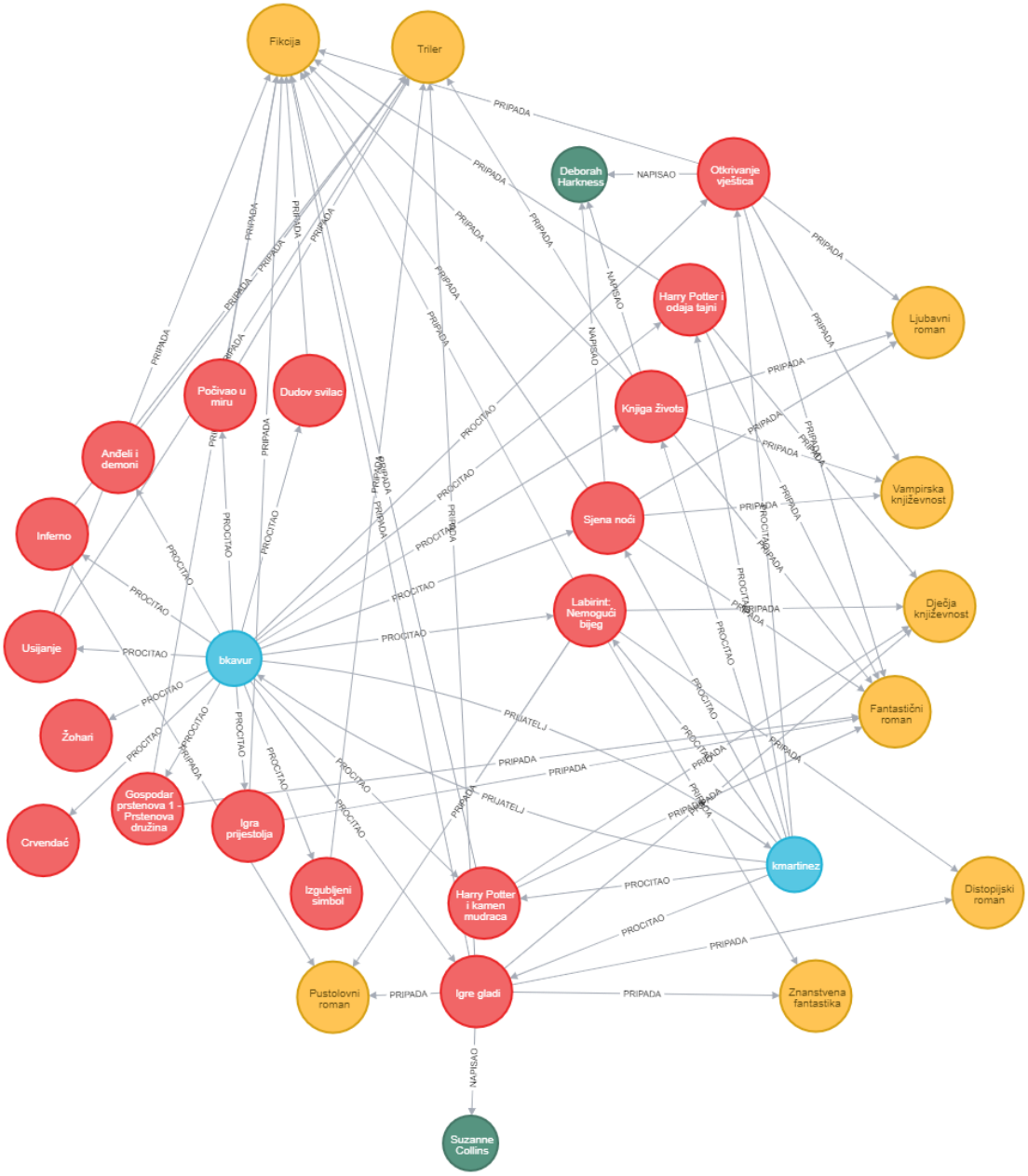
Index Name	Type	Uniqueness	EntityType	LabelsOrTypes	Properties	State
constraint_4dc18f5b	BTREE	UNIQUE	NODE	["Korisnik"]	["korisnicko_ime"]	ONLINE
index_5726608f	BTREE	NONUNIQUE	NODE	["Knjiga"]	["naslov"]	ONLINE

Constraints

ON (korisnik:Korisnik) ASSERT (korisnik.korisnicko_ime) IS UNIQUE

Slika 25: Prikaz indkesa i ograničenja [vlastita izrada]

Izgled baze podataka nakon što su kreirani svi čvorovi i veze vidljiv je na idućoj slici. Zbog veličine same baze prikazan je samo dio grafa.



Slika 26: Dio grafovske baze podataka sustava za preporuke [vlastita izrada]

6.4. Implementacija sustava za preporuke i primjer korištenja aplikacije „Book For You“

Za izradu aplikacije korišten je Java programski jezik zajedno s programskim okvirom Spring Boot. Kao alat za upravljanje projektom korišten je Maven te je potrebno dodati zavisnost na Neo4j Java Driver kako bi se mogli povezati na bazu podataka preko binarnog „Bolt“ protokola koji je dostupan na zadanom portu 7678. Moguće je za Javu koristiti i Spring Data Neo4j projekt koji nudi integraciju sa Spring programskim okvirom te se isto tako koristi za pristup Neo4j bazi podataka. Njegova prednost je u tome što nam preko raznih anotacija daje mogućnost mapiranja grafa u Java objekte (engl. „Object Graph Mapping“) tako da ih ne moramo ručno mapirati kao s Neo4j Java Driver-om koji je korišten za implementaciju sustava za preporuke. [67], [68]

```
<dependency>
	<groupId>org.neo4j.driver</groupId>
	<artifactId>neo4j-java-driver</artifactId>
	<version>4.2.3</version>
</dependency>
```

Nakon dodavanja zavisnosti potrebno je kreirati novu instancu Drivera pomoću kojeg se spajamo na bazu podataka te je potrebno pružiti podatke za spajanje. Potrebni podaci su uri do baze što je bolt port do baze, korisničko ime i lozinka. U Neo4j Desktop aplikaciji moguće je definirati korisnike i dodavati im određena prava te je dodan novi korisnik preko kojeg se spajamo na bazu. Iduća slika prikazuje isječak koda koji je korišten za spajanje na bazu.

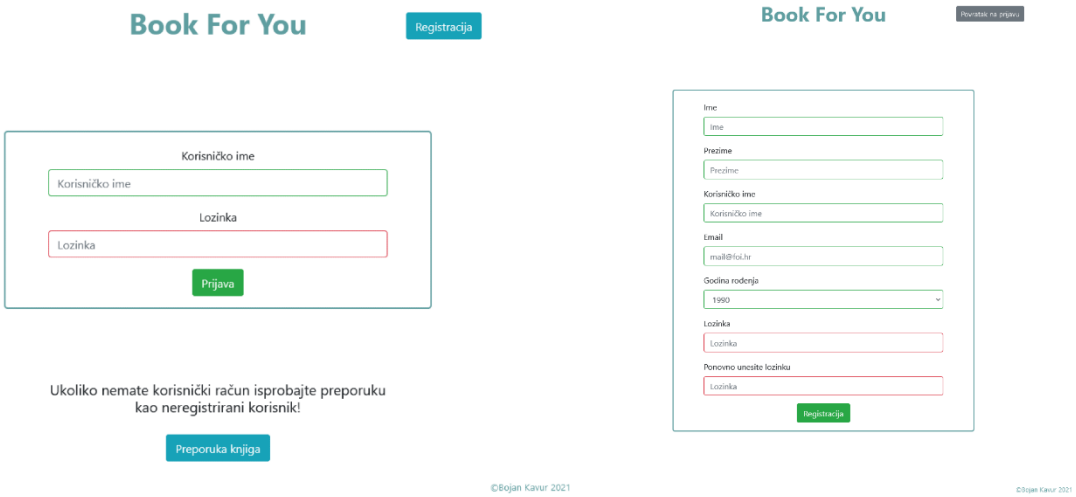
```
private static final String URI = "bolt://localhost:7687";
private static final String USER = "admin";
private static final String PASSWORD = "admin";

private final Driver driver;

public Database() { driver = GraphDatabase.driver(URI, AuthTokens.basic(USER, PASSWORD)); }

@Override
public void close() { driver.close(); }
```

Slika 27: Spajanje na bazu - Neo4j Driver [Vlastita izrada]

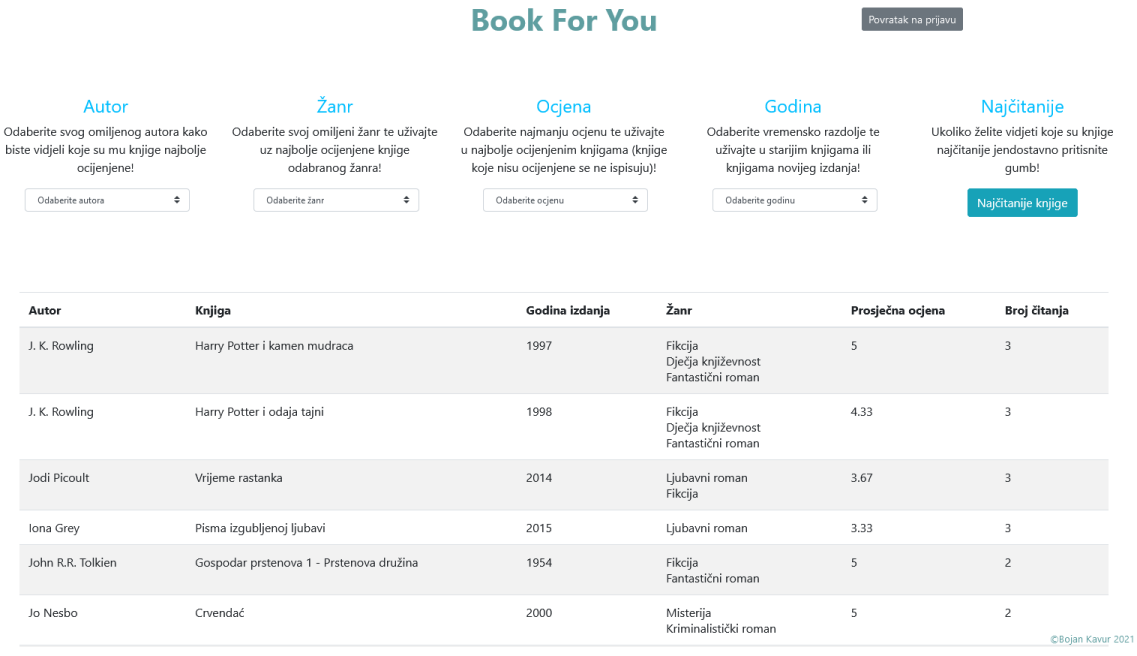
Uspješnim spajanjem na bazu preostaje nam samo otvaranje nove sesije preko kreirane instance Driver-a unutar koje možemo izvršiti željeni upit korištenjem metode run() te

je na kraju potrebno zatvoriti sesiju kad nam ona više nije potrebna. Možemo kreirati i transakcije korištenjem metode writeTransaction() nad otvorenom sesijom.

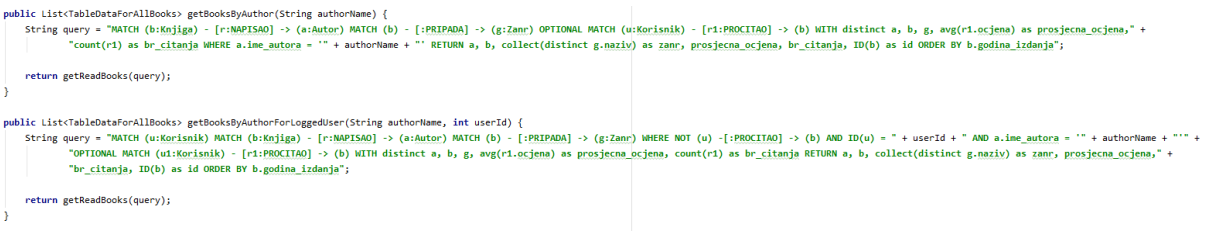
Iduća slika prikazuje početnu stranicu aplikacije „Book For You“ koja je vidljiva s lijeve strane slike te s desne strane registraciju korisnika. Možemo vidjeti i gumb „Preporuka knjiga“ koji je namijenjen neregistriranim korisnicima i njima se može dati preporuka prema autoru, žanru i godini izdanja knjige tako da odaberu omiljenog autora, žanr ili raspon godina kada je knjiga izdana te prema prosječnoj ocjeni knjige i najčitanijim knjigama temeljem ocjena koje su dali registrirani korisnici. S time da se za preporuku prema ocjeni i najčitanijim knjigama gledaju samo one knjige koje je neki korisnik pročitao i ocijenio, a broj čitanja je zapravo broj veza :PROCITAO koje idu u čvor knjige.



Slika 28: Početna stranica aplikacije (lijevo) i registracija koirsnika (desno) [Vlastita izrada]



Slika 29: Preporuka za neregistrirane korisnike (najčitanije knjige) [Vlastita izrada]



Slika 30: Preporuka prema odabranom autoru [Vlastita izrada]

Upiti za preporuku knjiga temeljem slike 29 su vrlo identični i stoga će biti prikazan samo upit za dohvat knjiga prema autoru. Možemo vidjeti da upit dohvaća sve knjige, autora koji je napisao pojedinu knjigu i žanrove kojima knjiga pripada uz uvjet da je ime autora jednako onome koje je proslijeđeno metodi što je zapravo autor kojeg korisnik odabere iz padajućeg izbornika. Uz to se opcionalno dohvate sve veze :PROCITAO tako da se na temelju atributa „ocjena“ koji pripada vezi može izračunati prosječna ocjena i broj čitanja knjige. Dohvat je opcionalan jer može biti da niti jedan korisnik nije pročitao tu knjigu što znači da ona nema niti jednu ocjenu i upit će u tom slučaju vratiti NULL što će biti prikazano kao da je prosječna ocjena 0. Isto tako možemo uočiti agregirajuće funkcije count() koja vraća ukupan broj veza :PROCITAO što predstavlja broj čitanja knjige, avg() koja izračunava prosječnu ocjenu na temelju atributa „ocjena“ i collect() koja vraća listu žanrova kojima knjiga pripada. Za prijavljenog korisnika upit je identičan samo što se još dodatno dohvati korisnik čiji je ID jednak ID-u prijavljenog korisnika (proslijeđen kao parametar metode) te se dodaje uvjet da korisnik nije pročitao knjigu koju preporučujemo. Cilj je korisnicima preporučiti knjige koje još nisu pročitali. Prilikom kreiranja čvora i veze, Neo4j SUGBP svakom dodaje jedinstveni identifikator ili ID kojeg je moguće dohvatiti preko funkcije ID() kojoj proslijedimo varijablu čvora ili veze i tada nam vraća pripadni identifikator.

Iduća slika prikazuje implementaciju metode getReadBooks() kojom izvršavamo mapiranje grafa, točnije dohvaćenih čvorova i varijabli u Java objekt. Možemo vidjeti da je korišten „try-with-resources“ za otvaranje sesije kako bi se ona sigurno zatvorila nakon što se izvrši try blok. Ovo je temeljni princip korištenja Neo4j Driver-a i sami moramo iz rezultata dohvatiti vrijednosti i mapirati ih kako nam odgovara, npr. čvor i vezu mapiramo tako da kreiramo mapu String, Object u koju se tada pohrane svi atributi pripadnog čvora kao ključ–vrijednost parovi i preko naziva atributa možemo dohvatiti njegovu vrijednost.

```java
public List<TableDataForAllBooks> getReadBooks(String query) {
    try (Session session = driver.session()) {

        Result result = session.run(query);
        List<TableDataForAllBooks> tableDataForAllBooks = new ArrayList<>();

        while (result.hasNext()) {
            TableDataForAllBooks bookData = new TableDataForAllBooks();

            Map<String, Object> author;
            Map<String, Object> book;
            double prosjecnaOcjena;
            double brCitanja;

            author = result.peek().fields().get(0).value().asMap();
            book = result.peek().fields().get(1).value().asMap();
            if (result.peek().get(3).isNull()) {
                prosjecnaOcjena = 0;
            } else {
                prosjecnaOcjena = result.peek().get(3).asDouble();
            }
            brCitanja = result.peek().get(4).asDouble();
            int id = result.peek().get(5).asInt();

            List<String> genres = new ArrayList<>();
            List<Object> genreObjects = result.peek().get(2).asList();
            genreObjects.forEach(o -> genres.add(o.toString()));

            String imeAutora = author.get("ime_autora").toString();
            String naslovKnjige = book.get("naslov").toString();
            String godinaIzdanjaKnjige = book.get("godina_izdanja").toString();

            bookData.setId(id);
            bookData.setImeAutora(imeAutora);
            bookData.setNaslovKnjige(naslovKnjige);
            bookData.setGodinaIzdanjaKnjige(godinaIzdanjaKnjige);
            bookData.setProsijecnaOcjena(prosjecnaOcjena);
            bookData.setBrCitanja(brCitanja);
            bookData.setGenres(genres);

            log.info( msg: "Id knjige: " + id);
            log.info( msg: "Ime autora: " + imeAutora);
            log.info( msg: "Naslov knjige: " + naslovKnjige);
            log.info( msg: "Godina izdanja knjige: " + godinaIzdanjaKnjige);
            log.info( msg: "Prosjecna ocjena: " + prosjecnaOcjena);
            log.info( msg: "Broj čitanja: " + brCitanja);
            log.info( msg: "Žanrovi: " + genres.toString());

            tableDataForAllBooks.add(bookData);
            result.next();
        }

        log.info( msg: "getReadBooks: " + tableDataForAllBooks.size() + " Data: " + tableDataForAllBooks.toString());

        return tableDataForAllBooks;
    } catch (Exception e) {
        log.warning( msg: "Exception: " + e);
        return null;
    }
}
```

Slika 31: Mapiranje rezultata upita [Vlastita izrada]

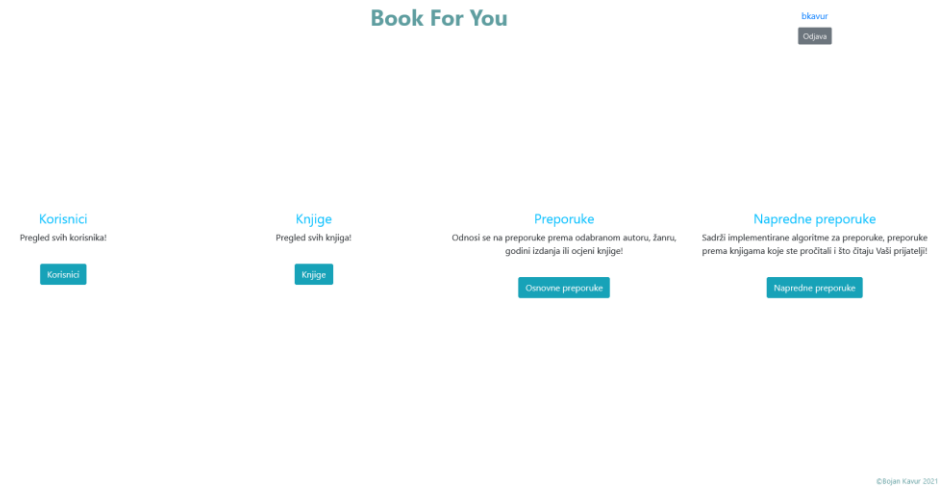
Na sličan način možemo izvršiti i registraciju novog korisnika, napravljena je provjera postoji li već korisnik s upisanim korisničkim imenom, ali kreirano je i ograničenje nad bazom da mora biti jedinstveno pa upit neće proći ako se i pokuša izvršiti CREATE. Na sljedećoj slici možemo vidjeti metodu za registraciju korisnika koja prima instancu objekta User te kreira novi čvor s oznakom Korisnik u bazi podataka. Prisjetimo se da je shema fleksibilna i potrebno je

programski odrediti koji atributi će biti obavezni, a koji ne. U ovom slučaju provjera je napravljena prije poziva metode za registraciju korisnika te je uvjetovano da će svi atributi biti obavezni za registraciju novog korisnika.

```
public boolean registerUser(User user) {
    boolean registered = false;

    String query = "CREATE (novi:Korisnik { ime:'" + user.getIme() + "', prezime:'" + user.getPrezime() + "', email:'" + user.getEmail() + "'" +
            ", godina_rodenja:" + Integer.valueOf(user.getGodinaRodenja()) + ", korisnicko_ime:'" + user.getKorisnickoIme() + "'" +
            ", lozinka:'" + user.getLozinka() + "' }) RETURN novi";

    try (Session session = driver.session()) {
        Result result = session.run(query);

        Map<String, Object> singleUser;

        singleUser = result.single().get(0).asMap();

        String ime = singleUser.get("ime").toString();
        String prezime = singleUser.get("prezime").toString();
        String korisnickoIme = singleUser.get("korisnicko_ime").toString();
        String email = singleUser.get("email").toString();
        String lozinka = singleUser.get("lozinka").toString();
        String godinaRodenja = singleUser.get("godina_rodenja").toString();

        log.info( msg: "Ime registriranog korisnika: " + ime);
        log.info( msg: "Prezime registriranog korisnika: " + prezime);
        log.info( msg: "Korisničko ime registriranog korisnika: " + korisnickoIme);
        log.info( msg: "Email registriranog korisnika: " + email);
        log.info( msg: "Lozinka registriranog korisnika: " + lozinka);
        log.info( msg: "Godina rođenja registriranog korisnika: " + godinaRodenja);

        if (ime != null && prezime != null && korisnickoIme != null && email != null && lozinka != null && godinaRodenja != null) {
            registered = true;
        }
    } catch (Exception e) {
        log.warning( msg: "Exception: " + e);
        registered = false;
    }
    return registered;
}
```

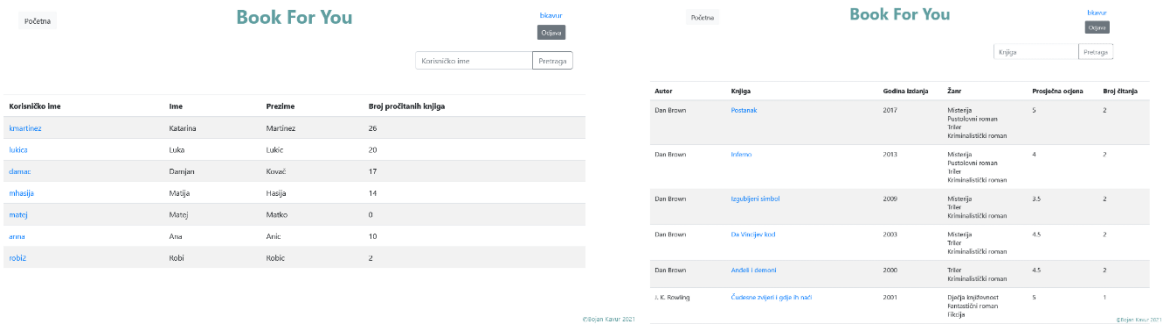
Slika 32: Registracija korisnika [Vlastita izrada]

Nakon prijave korisniku je vidljiva početna stranica (slika 33) preko koje se može dalje navigirati na pregled korisnika, knjiga i razne preporuke.



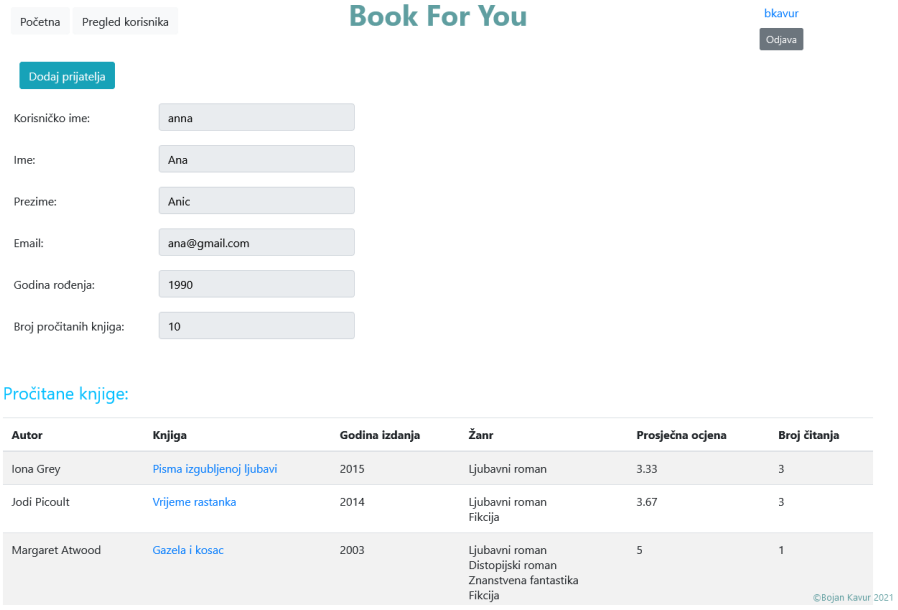
Slika 33: Početna stranica prijavljenog korisnika [Vlastita izrada]

Na pregledu korisnika i knjiga postoji mogućnost pretrage prema korisničkom imenu i nazivu knjige te je vidljiva statistika čitanja pojedinog korisnika i za knjige prosječna ocjena i broj čitanja.



Slika 34: : Pregled korisnika (lijevo) i knjiga (desno) [Vlastita izrada]

Klikom na korisničko ime jednog od korisnika možemo vidjeti detalje tog korisnika te ga dodati kao prijatelja ili poznanika ako već nismo. U detaljima su vidljive i knjige koje je taj korisnik pročitao, a ako gledamo svoj profil možemo vidjeti i popis prijatelja koje smo dodali.



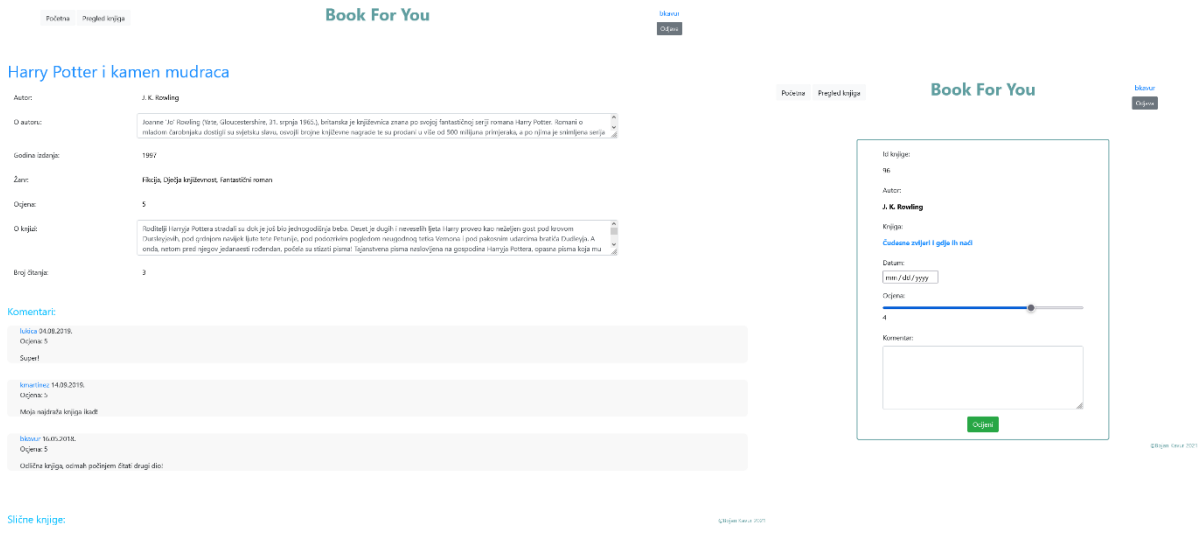
Slika 35: Detalji korisnika [Vlastita izrada]

Upit za dodavanje prijatelja samo kreira vezu tipa :PRIJATELJ između dvaju korisnika za što je potrebno dohvatiti čvorove tih korisnika i kreirati vezu između njih. Kod za kreiranje veze je vrlo jednostavan i vidljiv je na idućoj slici. Metoda vraća TRUE ako je veza uspješno kreirana, a u suprotnome FALSE.

```java
public boolean addFriend(int userId, int friendId) {
    boolean friend = false;

    String query = "MATCH (a:Korisnik), (b:Korisnik) WHERE ID(a) = " + userId + " AND ID(b) = " + friendId + " CREATE (a)-[ r:PRIJATELJ ]->(b) RETURN type(r);";

    try (Session session = driver.session()) {
        Result result = session.run(query);

        String relationshipType = result.peek().get(0).asString();

        log.info( msg: "Kreirana relacija tipa: " + relationshipType);

        if (relationshipType != null) {
            friend = true;
        }
    } catch (Exception e) {
        log.warning( msg: "Exception: " + e);
        friend = false;
    }
    return friend;
}
```

Slika 36: Dodavanje prijatelja [Vlastita izrada]

Nadalje, ako se klikne na naziv knjige tada se otvaraju detalji knjige te korisnik, ako već nije, može ocijeniti knjigu. Kroz detalje knjige vidljivi su detalji o autoru, sažetak, statistika čitanja, komentari i ocjene korisnika koji su pročitali knjigu te se na dnu stranice nalazi preporuka sličnih knjiga koja je implementirana korištenjem algoritma Jaccardovog indeksa sličnosti. Takva preporuke se temelji na sustavima za preporuke na temelju sadržaja gdje se gledaju knjige koje pripadaju istim žanrovima i koje je napisao isti autor.



Slika 37: Detalji knjige (lijevo) i davanje ocjene (desno) [Vlastita izrada]

Na prethodnoj slici s lijeve strane vidljivi su detalji knjige „Harry Potter i kamen mudraca“ te možemo vidjeti da iznad naslova knjige nema gumba „Ocijeni“ jer je trenutno prijavljeni korisnik već pročitao tu knjigu što je vidljivo i iz posljednjeg komentara. Za ocjenu se otvara nova forma, vidljiva na desnoj strani prethodne slike, gdje je potrebno odabrati datum, ocjenu i upisati komentar nakon čega se izvrši idući upit.

```
MATCH (a:Korisnik), (b:Knjiga) WHERE ID(a) = userId AND ID(b) = bookId CREATE
(a)-[ r:PROCITAO {datum:'date', ocjena: grade, komentar: 'comment'}]->(b)
RETURN r;
```

Gdje je userId identifikator korisnika, bookId identifikator knjige, date odabrani datum kada je korisnik pročitao knjigu, grade odabrana ocjena i comment korisnikov komentar na knjigu.

Slične knjige:

©Bojan Kavur 2021

Autor	Knjiga	Godina izdanja	Žanr	Prosječna ocjena	Broj čitanja	Sličnost (Jaccard)
J. K. Rowling	Harry Potter i princ miješane krvi	2005	Fikcija Dječja književnost Fantastični roman	5	1	1
J. K. Rowling	Harry Potter i red feniksa	2003	Fikcija Dječja književnost Fantastični roman	5	1	1
J. K. Rowling	Čudesne zvijeri i gdje ih naći	2001	Dječja književnost Fantastični roman Fikcija	5	1	1
J. K. Rowling	Harry Potter i plameni pehar	2000	Fikcija Dječja književnost Fantastični roman	5	1	1
J. K. Rowling	Harry Potter i zatočenik Azkabana	1999	Fikcija Fantastični roman	4.5	2	0.75
J. K. Rowling	Harry Potter i darovi smrti	2007	Triler Fikcija Misterija Dječja književnost Fantastični roman	5	1	0.67
John R.R. Tolkien	Hobit	1937	Dječja književnost Fikcija Fantastični roman	5	1	0.6
George R. R. Martin	Ples sa zmajevima	2011	Fikcija Fantastični roman	5	1	0.4
George R. R. Martin	Gozba vrana	2005	Fikcija Fantastični roman	5	1	0.4
John R.R. Tolkien	Gospodar prstenova 2 - Dvije kule	1954	Fikcija Fantastični roman	5	1	0.4
George R. R. Martin	Oluja mačeva	2000	Fikcija Fantastični roman	5	1	0.4
John R.R. Tolkien	Gospodar prstenova 3 - Povratak kralja	1955	Fikcija Fantastični roman	5	1	0.4
Nora Roberts	Magija krvi	2014	Fantastični roman Ljubavni roman Fikcija	0	0	0.33
Nora Roberts	Suočavanje s Vatrom	2002	Ljubavni roman Fikcija Fantastični roman	0	0	0.33

Slika 38: Preporuka sličnih knjiga, nastavno na sliku 37 [Vlastita izrada]

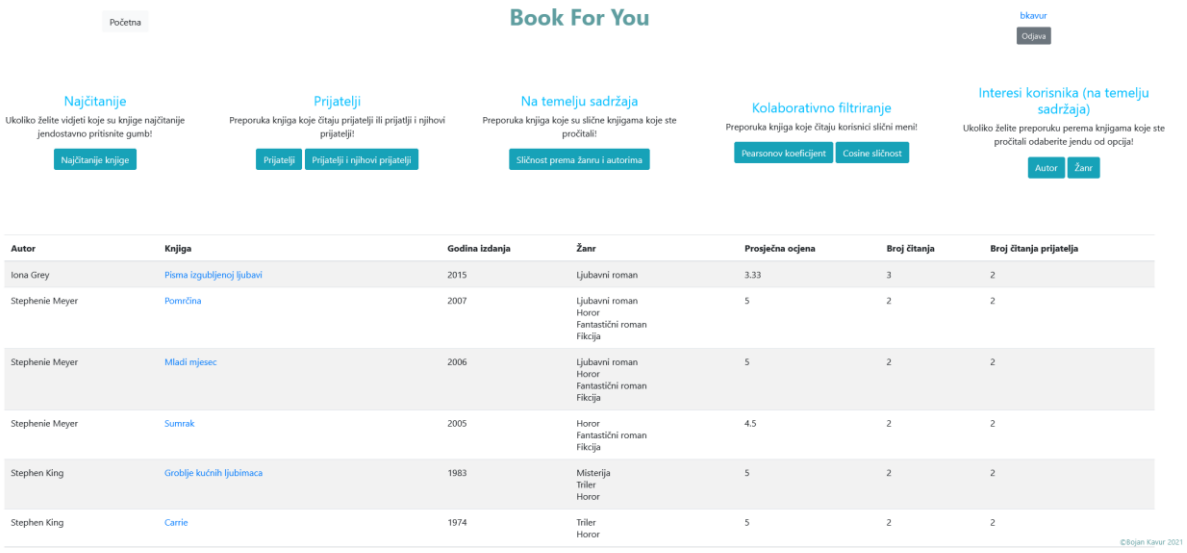
Formula za Jaccardov indeks sličnosti definirana je u poglavlju 5.1, a implementirana je korištenjem Cypher upitnog jezika.

```
MATCH (u:Korisnik) WHERE ID(u) = uderId MATCH (k:Knjiga)-[:PRIPADA|NAPISAO]-
>(j)<-[:PRIPADA|NAPISAO]-(b:Knjiga) WHERE ID(k) = book AND NOT EXISTS ((u) -
[:PROCITAO] -> (b))

WITH k, b, COUNT(j) AS intersection, COLLECT(j) AS i

MATCH (k)-[:PRIPADA|NAPISAO]-(kj)

WITH k, b, intersection, i, COLLECT(kj) AS s1

MATCH (b)-[:PRIPADA|NAPISAO]-(oj)

WITH k, b, intersection, i, s1, COLLECT(oj) AS s2, collect(distinct oj.naziv)
AS zanr

WITH k, b, intersection, s1, s2, zanr

MATCH (b)-[r:NAPISAO]->(a:Autor) OPTIONAL MATCH (:Korisnik)-[r1:PROCITAO]-
>(b)
```

```
WITH k, b, intersection, s1+[x IN s2 WHERE NOT x IN s1] AS union, s1, s2,
zanr, a, avg(r1.ocjena) as prosjecna_ocjena, count(r1) AS br_citanja

RETURN a, b, zanr, prosjecna_ocjena, br_citanja, ID(b) as id,
((1.0*intersection)/SIZE(union)) AS jaccard ORDER BY jaccard DESC LIMIT 35
```

Upit dohvaća korisnika prema identifikatoru i sve knjige koje je napisao isti autor ili pripadaju istom žanru kao i knjiga koju trenutno gledamo, a da ih korisnik još nije pročitao. Presjek se računa kao broj zajedničkih žanrova i autora te se nakon toga dohvate svi žanrovi i autori knjige koju gledamo u skup s1 i knjige koja je slična u skup s2. Potom se računa unija tako da se uzme sve iz skupa s1 i dodaju se elementi iz skupa s2 koji još nisu u skupu s1. Jaccardov indeks računa se tako da presjek podijelimo s unijom skupova uz to da se pomnoži s 1.0 kako bi se cijeli broj pretvorio u decimalan broj. Ostatak upita dohvaća žanrove kojima preporučena knjiga pripada, njenog autora te opcionalno prosječnu ocjenu i broj čitanja temeljem drugih korisnika sustava. Rezultat se u kodu mapira na identičan način kao što je prethodno opisao uz izmjenu da je dodano jedno dodatno polje u rezultat.

Ako se vratimo na početnu stranicu prijavljenog korisnika (slika 33) pod „Preporuke“ nalaze se prethodno definirane preporuke za neregistrirane korisnike samo što se u ovom slučaju ne preporučaju knjige koje je korisnik već pročitao. Zanimljivija nam je posljednja opcija „Napredne preporuke“ i prva takva preporuka je preporuka prema prijateljima.



Slika 39: Preporuka temeljem prijatelja [Vlastita izrada]

Prethodna slika prikazuje preporuku knjiga temeljem prijatelja koje je korisnik dodao putem aplikacije. U Cypher-u je takav upit vrlo jednostavan i lako se može dodati dubina veze tako da će se raditi preporuke prema prijateljima i njihovim prijateljima pa čak i do treće razine ili više. Idući upit predstavlja preporuku knjiga prema prijateljima:

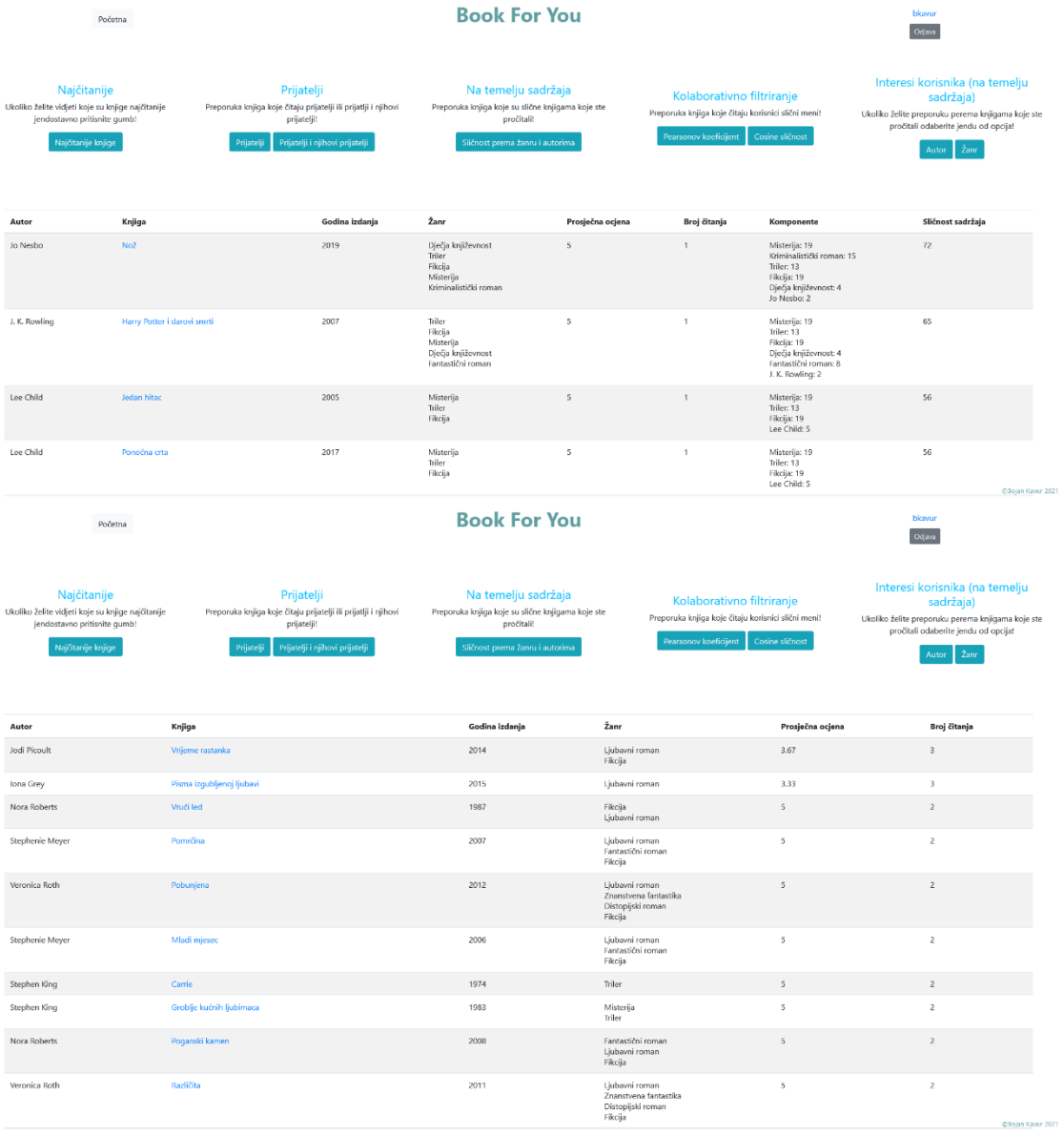
```
MATCH (k:Korisnik) - [:PRIJATELJ * 1..dubina] -> (p:Korisnik) WHERE k <> p
and ID(k) = userId WITH DISTINCT p, k

MATCH (p) - [:PROCITAO] -> (b:Knjiga) WHERE NOT (k) -[:PROCITAO] -> (b) with
b, count(b) as broj

MATCH (u:Korisnik) - [r:PROCITAO] -> (b1:Knjiga) - [:NAPISAO] -> (a:Autor)
MATCH (b1) - [:PRIPADA] -> (g:Zanr) WHERE b = b1 WITH DISTINCT a, b1, g,
avg(r.ocjena) as prosjecna_ocjena, count(r) AS br_citanja, broj

RETURN a, b1, collect(DISTINCT g.naziv) as zanr, prosjecna_ocjena,
br_citanja, ID(b1) AS id, broj ORDER BY br_citanja DESC, broj DESC LIMIT 35
```

Gdje dubina označava koliko skokova želimo ići u dubinu prateći vezu :PRIJATELJ, veza je usmjerena tako da će pratiti prijatelje koje je korisnik dodao i dalje njihove prijatelje. Da smjer nije definiran tada bi se preporuke radile i prema korisnicima koji su nas dodali kao poznanika ili prijatelja, ali mi nismo njih dodali što znači da se ne bi gledali smo naši prijatelji. Navedeni parametar (*1..X) kod veze je opcionalan prilikom dohvata podataka i njegova zadana vrijednost je 1. Operator <> označava različitost čime isključujemo da korisnik ne može sam sebi biti prijatelj što niti ne bi smjelo biti pohranjeno u bazi, ali možemo ostaviti kao provjeru u upitu. Idući dio upita dohvaća sve knjige koje su pročitali prijatelji prijavljenog korisnika uz uvjet da ih korisnik još nije pročitao (userId je identifikator prijavljenog korisnika) te se dohvaćaju ostali detalji koji su potrebni za tablični prikaz knjige. Ovdje nemamo opcionalni MATCH jer barem jedan prijatelj mora pročitati knjigu koja se preporučuje tako da će svaka preporučena knjiga imati broj čitanja minimalno 1 i prema tome i prosječnu ocjenu. Varijabla „broj“ označava broj prijatelja koji su pročitali knjigu.

Na gornjem dijelu iduće slike prikazana je preporuka prema žanru i autorima knjiga koje je korisnik pročitao ili ocijenio što predstavlja sustave za preporuke na temelju sadržaja. Preporuka se temelji na algoritmu centralnosti koji je opisan u poglavlju 5.4 i naziva se mjera stupnja centralnosti. U ovom slučaju gledao se broj izlaznih veza iz čvora Knjiga prema čvorovima tipa Žanr i Autor ili broj ulaznih veza prema Žanru i Autoru.

```
MATCH (u:Korisnik)-[r:PROCITAO]->(k:Knjiga), (k)-[:PRIPADA|NAPISAO]->(ga)<-
[:PRIPADA|NAPISAO]-(rec:Knjiga)

WHERE NOT EXISTS((u)-[:PROCITAO]->(rec)) AND ID(u) = userId

WITH rec, CASE when ga.naziv is null THEN [ga.ime_autora, COUNT(*)] ELSE
[ga.naziv, COUNT(*)] END AS scores

MATCH (rec)-[:NAPISAO]->(a:Autor) WITH DISTINCT a, rec, scores

MATCH (rec)-[:PRIPADA]->(g:Zanr) OPTIONAL MATCH (:Korisnik)-[r1:PROCITAO]-
>(rec) WITH DISTINCT a, rec, scores, COLLECT(DISTINCT g.naziv) AS zanr,
avg(r1.ocjena) as prosjecna_ocjena, count(DISTINCT r1) AS br_citanja

RETURN a, rec AS recommendation, zanr, prosjecna_ocjena, br_citanja, ID(rec)
as id, COLLECT(scores) AS scoreComponents, REDUCE (s=0,x in COLLECT(scores)
| s+x[1]) AS score ORDER BY score DESC LIMIT 35
```

Prethodni upit dohvaća sve knjige koje je korisnik pročitao i nakon toga sve knjige koje spadaju u neki od žanrova ili koje je napisao neki od autora knjiga koje je korisnik pročitao te se naravno provjerava da korisnik još nije pročitao knjigu koju će mu sustav preporučiti (userId je identifikator prijavljenog korisnika). Sličnost se računa kao broj veza koje ulaze u čvorove tipa Autor i Žanr, stavljen je uvjetni izraz CASE jer žanr ima naziv, a autor ima ime tako da se broj veza računa ovisno o tome u koji je čvor odredišni. Ostatak upita dohvaća detalje koji su potrebni za tablični prikaz gdje je opcionalan dohvat broja čitanja i prosječne ocjene. Funkcija collect() nam vraća listu komponenti koje sudjeluju u izračunu sličnosti, a funkcija reduce() računa sličnost tako da ima akumulator koji je inicijalno postavljen na 0 i prolazi kroz sve elemente komponenti te za izračun uzima drugi element iz liste koji predstavlja broj veza prema toj komponenti (prvi element je naziv žanra ili ime autora) i dodaje ga akumulatoru.



Slika 40: Preporuka na temelju sadržaja, sličnost prema žanru i autorima (gore) i knjige prema žanrovima koriniska (dolje) [Vlastita izrada]

Temeljem opisa prethodnog upita možemo pogledati prvu preporučenu knjigu na gornjem dijelu prethodne slike i vidimo da autor Jo Nesbo te da knjiga spada u žanrove dječja književnost, triler, fikcija, misterija i kriminalistički roman. Ako pogledamo komponente one se sastoje upravo od navedenih informacija i prikazuju broj veza prema pojedinoj komponenti temeljem knjiga koje je korisnik pročitao što znači da je korisnik pročitao 2 knjige od autora Jo Nesbo, 19 knjiga koje spadaju u žanr misterija i tako dalje. Sličnost je izračunata kao zbroj svih komponenti.

Donji dio prethodne slike prikazuje preporuku temeljem žanrova koje korisnik voli. Slično je i s autorima i takva preporuka isto spada u sustave za preporuke na temelju sadržaja. Cilj je pronaći žanrove knjiga koje je korisnik pročitao te preporučiti knjige koje pripadaju tim žanrovima.

```
MATCH (b:Knjiga) - [r:NAPISAO] -> (a:Autor) MATCH (b) - [r1:PRIPADA] -> (g:Zanr) MATCH (u:Korisnik) -[r2:PROCITAO] -> (b1:Knjiga) - [:PRIPADA] -> (g1:Zanr)

WHERE NOT (u) -[:PROCITAO] -> (b) AND g.naziv = g1.naziv AND ID(u) = userId WITH DISTINCT a, b, g OPTIONAL MATCH (u1:Korisnik) - [r3:PROCITAO] -> (b)

WITH distinct a, b, g, avg(r3.ocjena) as prosjecna_ocjena, count(r3) as br_citanja

RETURN DISTINCT a, b, collect(distinct g.naziv) as zanr, prosjecna_ocjena, br_citanja, ID(b) as id ORDER BY br_citanja DESC, prosjecna_ocjena DESC LIMIT 35
```

Ovaj upit je vrlo jednostavan i traži sve knjige koje je korisnik pročitao (userId je identifikator prijavljenog korisnika) i njihove žanrove te sve knjige koje spadaju u neki od žanrova čije je knjige korisnik prethodno čitao. Ostatak upita dohvaća podatke potrebne za tablični prikaz knjiga i opcionalno dohvaća broja čitanja i prosječnu ocjenu pojedine knjige. Što se tiče preporuke temeljem autora koje korisnik voli upit je identičan uz razliku u tome da se ne gleda da preporučena knjiga spada u neki od žanrova već da ju je napisao neki od autora čije je knjige korisnik prethodno pročitao i ocijenio.

Posljednje preporuke koje su ostale spadaju u sustave za preporuke na temelju kolaborativnog filtriranja. Cilj je pronaći korisnike slične prijavljenom korisniku te preporučiti knjige koje su takvi slični korisnici pročitali, a da ih prijavljeni korisnik još nije pročitao. Prva preporuka odnosi se na algoritam Pearsonovog koeficijenta sličnosti koji je objašnjen u poglavlju 5.3.

```
MATCH (u1:Korisnik)-[r:PROCITAO]->(k:Knjiga) WHERE ID(u1) = userId

WITH u1, avg(r.ocjena) AS u1_mean MATCH (u1)-[r1:PROCITAO]->(k:Knjiga)<-[r2:PROCITAO]-(u2) WITH u1, u1_mean, u2, COLLECT({r1: r1, r2: r2}) AS ratings WHERE size(ratings) > 5

MATCH (u2)-[r:PROCITAO]->(k:Knjiga) WITH u1, u1_mean, u2, avg(r.ocjena) AS u2_mean, ratings UNWIND ratings AS r

WITH sum( (r.r1.ocjena-u1_mean) * (r.r2.ocjena-u2_mean) ) AS nom,
```

```
sqrt( sum( (r.r1.ocjena - u1_mean)^2) * sum( (r.r2.ocjena - u2_mean) ^2)) AS denom, u1, u2 WHERE denom <> 0

WITH u1, u2, nom/denom AS pearson

MATCH (u2)-[r:PROCITAO]->(k:Knjiga)-[:NAPISAO]->(a:Autor) WHERE NOT EXISTS( (u1)-[:PROCITAO]->(k) ) WITH DISTINCT a, k, (pearson * r.ocjena) AS score, u2.korisnicko_ime as simUser, ID(u2) as idKor, pearson as howSimilarIsUser

OPTIONAL MATCH (:Korisnik)-[r2:PROCITAO]->(k)-[:PRIPADA]->(g:Zanr) WITH DISTINCT a, k, score, simUser, idKor, howSimilarIsUser, avg(r2.ocjena) as prosjecna_ocjena, count(DISTINCT r2) AS br_citanja,collect(DISTINCT g.naziv) AS zanr

RETURN a, k, zanr, prosjecna_ocjena, br_citanja, ID(k) as id, score, simUser, idKor, howSimilarIsUser ORDER BY score DESC LIMIT 35
```

Autor	Knjiga	Godina izdanja	Žanr	Prosječna ocjena	Broj čitanja	Sličnost	Korisnik	Sličnost korisnika
Lee Child	Ponoćna crta	2017	Misterija Triler Fikcija	5	1	1.0004	lukica	0.2001
Lee Child	Jedan hitac	2005	Misterija Triler Fikcija	5	1	1.0004	lukica	0.2001
Stephen King	Groblje kućnih ljubimaca	1983	Misterija Triler Horor	5	2	1.0004	lukica	0.2001
Stephen King	Isijavanje	1977	Fikcija Triler Horor	5	1	1.0004	lukica	0.2001
Stephen King	Carrie	1974	Triler Horor	5	2	1.0004	lukica	0.2001
J. K. Rowling	Harry Potter i zatočenik Azkabana	1999	Fikcija Fantastični roman	4.5	2	1.0004	lukica	0.2001
John R.R. Tolkien	Hobit	1937	Dječja književnost Fikcija Fantastični roman	5	1	1.0004	lukica	0.2001
John R.R. Tolkien	Gospodar prstenova 3 - Povratak kralja	1955	Fikcija Fantastični roman	5	1	1.0004	lukica	0.2001
John R.R. Tolkien	Gospodar prstenova 2 - Dvije kule	1954	Fikcija Fantastični roman	5	1	1.0004	lukica	0.2001
George R. R. Martin	Ples sa zmajevima	2011	Fikcija Fantastični roman	5	1	1.0004	lukica	0.2001
George R. R. Martin	Gozba vrana	2005	Fikcija Fantastični roman	5	1	1.0004	lukica	0.2001
George R. R. Martin	Oluja mačeva	2000	Fikcija Fantastični roman	5	1	1.0004	lukica	0.2001
Lee Child	Prošlo vrijeme	2018	Misterija Triler Fikcija	4	1	0.8003	lukica	0.2001
Stephen King	Misery	1987	Triler Horor	4	1	0.8003	lukica	0.2001
Philippa Gregory	Druga sestra Boleyn	2001	Povijesni roman Fikcija	3	1	-0.982	kmartinez	-0.3273
Suzanne Collins	Igre gladi - Balada pjevica i zmija	2020	Pustolovni roman Znanstvena fantastika Triler	3	1	-0.982	kmartinez	-0.3273

Slika 41: Preporuka na temelju kolaborativnog filtriranja - Pearsonov koeficijent sličnosti [Vlastita izrada]

Algoritam je implementiran prethodnim Cypher upitom koji dohvaća korisnike slične prijavljenom korisniku temeljem zajedničkih knjiga koje su pročitali. Ograničeno je da mora biti minimalno 5 zajedničkih knjiga, ali da bi preporuka bila kvalitetnije preporučeno je da taj broj bude veći, oko 20. Za potrebe ovog rada i manje baze podataka taj broj je ograničen na 5 knjiga. Sličnost se računa temeljem ocjene koju su korisnici dali za pojedinu knjigu. U listu „rating“ se pohrane sve ocjene koje je dao prijavljeni korisnik r1 i pojedini slični korisnik r2. Za svakog korisnika je izračunata prosječna ocjena, „u1_mean“ za prijavljenog i „u2_mean“ za pojedinog sličnog korisnika te se tada prema formuli danoj u poglavlju 5.3 računa brojnik kao kovarijanca i nazivnik kao umnožak standardne devijacije temeljem ocjena koje su korisnici

dali za knjigu. Nadalje, dohvaćaju se sve knjige koje je pročitao sličan korisnik, a da ih prijavljeni korisnik još nije pročitao i kao mjera sličnosti koristi se izračunati Pearsonov koeficijent pomnožen s ocjenom koju je sličan korisnik dao toj knjizi. Na taj način možemo dobiti kvalitetniju preporuku gdje će veću sličnost imati bolje ocijenjene knjige. Ostatak upita, kao i do sad, dohvaća podatke potrebne za tablični prikaz.

Jedini nedostatak ovakve mjere sličnosti su oni korisnici koji imaju negativnu sličnost. Prisjetimo se da je rezultat Pearsonovog koeficijenta između -1 i 1. Pretpostavimo da je sličnost korisnika -0.5 i da je taj korisnik prvoj knjizi dao ocjenu 3, a drugoj ocjenu 5. U tom slučaju ako pomnožimo -0.5 s 3 dobit ćemo -1.5 kao sličnost za prvu knjigu i -2.5 za drugu. Znamo da je -1.5 veći od -2.5 te bi ispalo da ta knjiga ima veću sličnost i preporučili bi ju prije druge knjige što naravno nije dobro. Taj problem mogli bi riješiti tako da uzimamo apsolutnu vrijednost, ali onda bi morali paziti da takav korisnik čija je sličnost (Pearsonov koeficijent) negativna nema prednost pred korisnikom čiji je koeficijent pozitivan jer bi se lako moglo desiti da se preporučuju krive knjige. U implementaciji je taj dio ostavljen kako je i za sve negativne sličnosti vrijedi da je manji broj sličniji od većeg, tj. da se knjiga s izračunatom sličnošću -2.5 preporučuje prije knjige čija je sličnost -1.5. Za pozitivne vrijednosti jednostavno se gleda veći broj.

Na posljetku implementirana je sličnost korisnika temeljem algoritma kosinusne sličnosti koji je detaljnije objašnjen u poglavlju 5.2.

```
MATCH     (u1:Korisnik)-[r:PROCITAO]->(k:Knjiga)<-[y:PROCITAO]-(u2:Korisnik)
WHERE ID(u1) = userId

WITH COUNT(k) AS brknjiga, SUM(r.ocjena * y.ocjena) AS xy,

SQRT(REDUCE(x = 0.0, a IN COLLECT(r.ocjena) | x + a^2)) AS xLength,

SQRT(REDUCE(y = 0.0, b IN COLLECT(y.ocjena) | y + b^2)) AS yLength,

u1, u2 WHERE brknjiga > 5

MATCH   (u2)-[r1:PROCITAO]->(k1:Knjiga)-[:NAPISAO]->(a:Autor)     WHERE   NOT
EXISTS(  (u1)-[:PROCITAO]->(k1)  )  WITH  DISTINCT  a,  k1,  u2,  xy,  xLength,
yLength,   ((xy / (xLength * yLength))*r1.ocjena) AS score

OPTIONAL  MATCH  (:Korisnik)-[r2:PROCITAO]->(k1)-[:PRIPADA]->(g:Zanr)  WITH
DISTINCT a, k1, u2, xy, xLength, yLength, avg(r2.ocjena) as prosjecna_ocjena,
count(DISTINCT r2) AS br_citanja, score, collect(DISTINCT g.naziv) AS zanr

RETURN  a,  k1,  zanr,  prosjecna_ocjena,  br_citanja,  ID(k1)  as  id,  score,
u2.korisnicko_ime, ID(u2) as idKor, xy / (xLength * yLength) AS sim

ORDER BY score DESC, prosjecna_ocjena DESC LIMIT 35
```

Kao i kod Pearsonovog koeficijenta sličnosti ograničeno je da korisnici moraju imati minimalno 5 zajedničkih knjiga koje su pročitali, ali preporučuje se da taj broj bude veći. Sličnost se također računa temeljem ocjena koje su dali korisnici. Brojnik je skalarni umnožak koji se računa kao suma umnoška ocjena koje je dao prijavljeni korisnik i sličan korisnik za

pojedinu zajedničku knjigu, a nazivnik je umnožak duljina vektora koji se računa kao korijen sume kvadrata ocjena za pojedinog korisnika (prijavljenog i svakog sličnog) te se tada ti korijeni pomnože da se dobije umnožak duljina. Ostatak upita dohvaća knjige koje je pročitao sličan korisnik, a da ih trenutno prijavljeni korisnik još nije pročitao te detalje potrebne za tablični prikaz koji je vidljiv na idućoj slici. Za sličnost knjige pomnožena je ocjena koju je dao sličan korisnik s kosinusnom sličnošću korisnika tako da se dobije kvalitetnije preporuka. Rezultat kosinusne sličnosti je između -1 i 1 te se javlja identičan problem s korisnicima koji imaju negativnu vrijednost sličnosti koji je prethodno objašnjen kod Pearsnovog koeficijenta sličnosti.

Autor	Knjiga	Godina izdanja	Žanr	Prosječna ocjena	Broj čitanja	Sličnost	Korisnik	Sličnost korisnika
Sidney Sheldon	Ako dočekam sutra	1985	Kriminalistički roman Pustolovni roman Triler Fikcija	5	1	4.9691	kmartinez	0.9938
Stephenie Meyer	Pomrčina	2007	Ljubavni roman Horor Fantastični roman Fikcija	5	2	4.9691	kmartinez	0.9938
Stephenie Meyer	Mladi mjesec	2006	Ljubavni roman Horor Fantastični roman Fikcija	5	2	4.9691	kmartinez	0.9938
Veronica Roth	Pobunjena	2012	Ljubavni roman Znanstvena fantastika Distopijski roman Fikcija	5	2	4.9691	kmartinez	0.9938
Veronica Roth	Različita	2011	Ljubavni roman Znanstvena fantastika Distopijski roman Fikcija	5	2	4.9691	kmartinez	0.9938
Suzanne Collins	Igre gladi - Plamen	2009	Pustolovni roman Znanstvena fantastika Triler Distopijski roman Fikcija Dječja književnost	5	1	4.9691	kmartinez	0.9938
J. K. Rowling	Čudesne zvijeri i gdje ih naći	2001	Dječja književnost Fantastični roman Fikcija	5	1	4.9691	kmartinez	0.9938
J. K. Rowling	Harry Potter i darovi smrti	2007	Triler Fikcija Misterija Dječja književnost Fantastični roman	5	1	4.9691	kmartinez	0.9938
J. K. Rowling	Harry Potter i princ miješane krvi	2005	Fikcija Dječja književnost Fantastični roman	5	1	4.9691	kmartinez	0.9938
J. K. Rowling	Harry Potter i red feniksa	2003	Fikcija Dječja književnost Fantastični roman	5	1	4.9691	kmartinez	0.9938
Stephen King	Carrie	1974	Triler Horor	5	2	4.9621	lukica	0.9924
John R.R. Tolkien	Hobit	1937	Dječja književnost Fikcija Fantastični roman	5	1	4.9621	lukica	0.9924
John R.R. Tolkien	Gospodar prstenova 3 - Povratak kralja	1955	Fikcija Fantastični roman	5	1	4.9621	lukica	0.9924
John R.R. Tolkien	Gospodar prstenova 2 - Dvije kule	1954	Fikcija Fantastični roman	5	1	4.9621	lukica	0.9924
George R. R. Martin	Ples sa zmajevima	2011	Fikcija Fantastični roman	5	1	4.9621	lukica	0.9924
George R. R. Martin	Gozba vrana	2005	Fikcija Fantastični roman	5	1	4.9621	lukica	0.9924
George R. R. Martin	Oluja mačeva	2000	Fikcija Fantastični roman	5	1	4.9621	lukica	0.9924
J. K. Rowling	Harry Potter i zatočenik Azkabana	1999	Fikcija Fantastični roman	4.5	2	4.9621	lukica	0.9924
J. K. Rowling	Harry Potter i zatočenik Azkabana	1999	Fikcija Fantastični roman	4.5	2	3.9753	kmartinez	0.9938
Sidney Sheldon	Krvna veza	1977	Pustolovni roman Triler Fikcija	4	1	3.9753	kmartinez	0.9938
Philippa Gregory	Časni trgovci	1998	Ljubavni roman Povijesni roman Fikcija	4	1	3.9753	kmartinez	0.9938
Suzanne Collins	Igre gladi - Šojka rugalica	2010	Pustolovni roman Znanstvena fantastika Triler Distopijski roman Fikcija Dječja književnost	4	1	3.9753	kmartinez	0.9938
Lee Child	Prošlo vrijeme	2018	Misterija Triler Fikcija	4	1	3.9697	lukica	0.9924
Stephen King	Misery	1987	Triler Horor	4	1	3.9697	lukica	0.9924
Iona Grey	Pisma izgubljenoj ljubavi	2015	Ljubavni roman	3.33	3	2.9814	kmartinez	0.9938

Slika 42: Preporuka na temelju kolaborativnog filtriranja - Kosinusna sličnost [Vlastita izrada]

Ako detaljnije pogledamo slike 41 i 42 možemo uočiti da je sličnost korisnika različita, ovisno o tome koji algoritam za izračun sličnosti koristimo. Samim time u nekoj mjeri se razlikuju i knjige koje sustav preporučuje, stoga je vrlo važno biti upoznat sa svim algoritmima i uzeti „bolji“ koji više odgovara domeni sustava za preporuke. Najbolje bi bilo kreirati hibridni sustav za preporuke koji će koristiti više algoritama i na kraju filtrirati rezultate pojedinog algoritma da se nađu najbolje preporuke. Isto tako treba obratiti pažnju na to da slični korisnici moraju imati najmanje 5 zajedničkih knjiga što znači da ove preporuke neće raditi za nove korisnike koji još nisu pročitali niti jednu knjigu ili su pročitali mali broj knjiga. Iako je preporučeno da taj broj bude veći, između 20 i 50 kako bi preporuke bile kvalitetnije.

7. Zaključak

U ovom radu upoznali smo se sa sustavima za preporuke i NoSQL bazama podataka. Iz navedenih primjera uspješnih sustava za preporuke i Netflix-ovog natjecanja u kojem je nudio milijun dolara timu koji izradi bolji sustava za preporuke od njihovog tadašnjeg sustava, možemo uočiti kolika je važnosti takvih sustava. Neke tvrtke koje imaju kvalitetne sustave za preporuke povećale su svoje prihode i za više od 50%, što je vrlo velik postotak te se isplati ulagati u sustave za preporuke, pogotovo u današnje vrijeme kada gotovo sve pretražujemo i kupujemo preko weba. Imamo dva glavna tipa informacija s kojima rade sustavi za preporuke, informacije o proizvodima i korisnicima te informacije dobivene interakcijom korisnika s proizvodima i samim sustavom. Prema tome postoje tri algoritma ili metode koje se koriste u sustavima za preporuke i to su sustavi za preporuke na temelju sadržaja, sustavi za preporuke na temelju kolaborativnog filtriranja i hibridni sustavi za preporuke. Svaki sustav se susreće s određenim problemima i kod sustava za preporuke najizražajniji su problem dugog repa i problem hladnog starta. Ostali problemi su nešto manje izražajni i neki od njih su skalabilnost sustava, pouzdanost preporuka, privatnost i drugi.

Drugi dio rada obuhvaća područje NoSQL baza podataka koje su nastale kao odgovor na razne vrste podataka (polustrukturirani, strukturirani i nestrukturirani) te dinamičnost i fleksibilnost koja je ključna za opstanak modernih aplikacija. Relacijske baze podataka bile su dominantna tehnologija za pohranu strukturiranih podataka te su se pojavom novih vrsta podataka susrele s brojnim izazovima i problemima. Poznata je skraćenica 3V koja označava tri glavna izazova relacijskih baza podataka: obujam, brzina rasta podataka kroz vrijeme i raznolikost podataka. NoSQL baze podataka rješavaju upravno navedene probleme i postoje četiri glavne skupine NoSQL baza podataka: ključ–vrijednost, stupčane, dokument i grafovske baze podataka. Glavna prednost NoSQL baza podataka je fleksibilnost sheme koja omogućuje dodavanje atributa po potrebi ili dinamičko dodavanje atributa što ne zahtijeva promjenu modela baze podataka ili sheme kao što je to slučaj s relacijskim bazama podataka. U radu je fokus bio na grafovskim bazama podataka koje se temelje na teoriji grafova te su u fokusu veze između podataka. One koriste grafovski model podataka za implementaciju od kojih su naj poznatiji RDF i model označenog grafa sa svojstvima. Jedna od najpoznatijih grafovskih baza podataka je Neo4j koja je, u ovom radu, korištena za izradu sustava za preporuke knjiga te ona implementira model označenog grafa sa svojstvima koji se sastoji od čvorova ili vrhova, veza ili bridova, oznaka kojima kreiramo klase čvorova i veza te svojstava koja se mogu dodijeliti čvorovima i vezama kako bi ih pobliže opisali. Neo4j posjeduje i svoj vlastiti grafovski upitni jezik Cypher koji je optimiziran za rad s grafovima. Čvorovi i veze zajedno čine uzorke koji tada opisuju jednostavnije ili složenije puteve unutar grafa.

Praktični dio rada sastoji se od izrade sustava za preporuke knjiga. Model baze podataka prikazan je preko modela označenog grafa sa svojstvima, ali i relacijskog modela podataka kako bi lakše uočili intuitivnost, jednostavnost i ekspresivnost takvog modela podataka koji je vrlo sličan ljudskoj percepciji stvarnosti odabrane domene. Nadalje, za implementaciju sustava korištena je Neo4j baza podataka koja je povezana s programskim jezikom Java preko Neo4j Java Driver-a. Aplikacija posjeduje i pripadno grafičko sučelje koje korisnicima omogućuje interakciju sa sustavom. Za davanje određenih preporuka korišteni su neki algoritmi temeljeni na grafovima, a to su algoritam ili mjera stupnja centralnosti koji spada u algoritme centralnosti te tri algoritma iz skupine algoritama sličnosti: algoritam Jaccardovog koeficijenta sličnosti, algoritam kosinusne sličnosti i algoritam Pearsonovog koeficijenta sličnosti. Postoje i drugi algoritmi od kojih su najpoznatiji algoritmi obilaska grafa koji se koriste i za izvršavanje upita nad grafovskom bazom podataka. Kao rezultat upita dobije se podgraf koji odgovara traženom uzorku. Neo4j posjeduje i razne dodatke ili biblioteke koje olakšavaju rad s bazom podataka od kojih neke imaju i implementirane razne algoritme temeljene na grafovima. U radu su korišteni jednostavniji algoritmi pa je i njihova implementacija napravljena „ručno“ bez korištenja takvih biblioteka, iako je i to moguće. Algoritmima su implementirane preporuke na temelju sadržaja (kod pregleda za pojedinu knjigu i kasnije temeljem knjiga koje je korisnik pročitao) te dvije preporuke na temelju kolaborativnog filtriranja kako bi se našli slični korisnici temeljem knjiga koje su pročitali. Zanimljiva je usporedba tih dvaju algoritama i kako svaki daje nešto drugačije preporuke što nam govori da je potrebno dobro poznavati dostupne algoritme te odabrati onog koji najviše odgovara domeni aplikacije i koji daje najbolje preporuke.

Na posljetku možemo zaključiti da su sustavi za preporuke vrlo kompleksni te su se kao najboljima za implementaciju takvih sustava pokazale grafovske baze podataka jer u fokus stavljaju veze između podataka i posjeduju fleksibilnu shemu kojom se mogu lako podržati razni podaci koji su sustavu potrebni. Isto tako preko raznih primjera uspješnih sustava za preporuke dokazano je da se isplati ulagati u njih jer kvalitetan sustav može uvelike unaprijediti poslovanje tvrtke koja ga posjeduje.

Popis literature

[1] C. Pinela, „Recommender Systems — Ranking and Classifications“, 29-lis-2017. [Na internetu]. Dostupno na: https://medium.com/@cfpinela/recommender-systems-ranking-and-classifications-3abae71d6fbf. [Pristupljeno: 06-lip-2021].

[2] K. Falk, *Practical Recommender Systems*. Manning Publications Co., 2019.

[3] „The Difference between a Recommendation and an Ad“, 2007. [Na internetu]. Dostupno na: http://bokardo.com/archives/the-difference-between-a-recommendation-and-an-ad/. [Pristupljeno: 05-lip-2021].

[4] B. Rocca, „Introduction to recommender systems“, 03-lip-2019. [Na internetu]. Dostupno na: https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada. [Pristupljeno: 06-lip-2021].

[5] N. Polatidis i C. K. Georgiadis, „Mobile recommender systems: An overview of technologies and challenges“, *2013 2nd Int. Conf. Informatics Appl. ICIA 2013*, izd. August 2013, str. 282–287, 2013, doi: 10.1109/ICoIA.2013.6650270.

[6] N. Severt, „An Introduction to Recommender Systems (+9 Easy Examples)“, 06-stu-2018. [Na internetu]. Dostupno na: https://www.iteratorshq.com/blog/an-introduction-recommender-systems-9-easy-examples/. [Pristupljeno: 06-lip-2021].

[7] S. S. Lakshmi i T. A. Lakshmi, „Recommendation Systems: Issues and challenges“, *International Journal of Computer Science and Information Technologies*, izd. 5(4). str. 5771–5772, 2014.

[8] M. Hussien Mohamed, M. Helmy Khafagy, i M. Hasan Ibrahim, „Recommender Systems Challenges and Solutions Survey“, *2019 Int. Conf. Innov. Trends Comput. Eng. ITCE*, str. 149–155, 2019.

[9] „Book Depository“. [Na internetu]. Dostupno na: https://www.bookdepository.com/. [Pristupljeno: 06-lip-2021].

[10] M. Milankovich, „The Cold Start Problem for Recommender Systems“, 2015. [Na internetu]. Dostupno na: https://medium.com/yusp/the-cold-start-problem-for-recommender-systems-89a76505a7. [Pristupljeno: 13-lip-2021].

[11] M. H. Nadimi-Shahraki i M. Bahadorpour, „Cold-start problem in collaborative recommender systems: Efficient methods based on ask-to-rate technique“, *J. Comput. Inf. Technol.*, sv. 22, izd. 2, str. 105–113, 2014, doi: 10.2498/cit.1002223.

[12] Netflix Technology Blog, „Netflix Recommendations: Beyond the 5 stars (Part 1)“,

2012. [Na internetu]. Dostupno na: https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429. [Pristupljeno: 15-lip-2021].

[13] J. Bennett i S. Lanning, *The Netflix Prize. In Proceedings of KDD Cup and Workshop*. 2007.

[14] W. Rahman, „The Netflix Prize — How Even AI Leaders Can Trip Up“, 2020. [Na internetu]. Dostupno na: https://towardsdatascience.com/the-netflix-prize-how-even-ai-leaders-can-trip-up-5c1f38e95c9f. [Pristupljeno: 15-lip-2021].

[15] „Netflix Prize: View Leaderboard“. [Na internetu]. Dostupno na: https://www.netflixprize.com/leaderboard.html. [Pristupljeno: 15-lip-2021].

[16] G. Rodriguez, „Introduction to Recommender Systems | Tryolabs Blog“. [Na internetu]. Dostupno na: https://tryolabs.com/blog/introduction-to-recommender-systems/. [Pristupljeno: 15-lip-2021].

[17] „Content Recommendations on Media Websites | RecoSense“, 2020. [Na internetu]. Dostupno na: https://recosenselabs.com/blog/content-recommendations-media-websites. [Pristupljeno: 17-lip-2021].

[18] F. O. Isinkaye, Y. O. Folajimi, i B. A. Ojokoh, „Recommendation systems: Principles, methods and evaluation“, *Egyptian Informatics Journal*, sv. 16, izd. 3. str. 261–273, 2015, doi: 10.1016/j.eij.2015.06.005.

[19] N. Mohammedali, „Recommendation System Based on Graph Database Techniques“, *Int. Res. J. Eng. Technol.*, sv. 6, izd. 10, str. 754–763, 2019.

[20] M. Madasamy, „Introduction to recommendation systems and How to design Recommendation system,that resembling the Amazon | Medium“, 2019. [Na internetu]. Dostupno na: https://madasamy.medium.com/introduction-to-recommendation-systems-and-how-to-design-recommendation-system-that-resembling-the-9ac167e30e95. [Pristupljeno: 18-lip-2021].

[21] Dataaspirant, „An Introduction to Recommendation Engines - Dataconomy“, 2015. [Na internetu]. Dostupno na: http://dataconomy.com/2015/03/an-introduction-to-recommendation-engines/. [Pristupljeno: 20-lip-2021].

[22] P. Rathle, „Driving Innovation in Retail with Graph Technology How Top Retailers Use Neo4j“, 2020.

[23] I. MacKenzie, C. Meyer, i S. Noble, „How retailers can keep up with consumers“, 2013. [Na internetu]. Dostupno na: https://www.mckinsey.com/industries/retail/our-

insights/how-retailers-can-keep-up-with-consumers. [Pristupljeno: 20-lip-2021].

[24] J. E. Solsman, „YouTube's AI is the puppet master over most of what you watch", 2018. [Na internetu]. Dostupno na: https://www.cnet.com/news/youtube-ces-2018-neal-mohan/. [Pristupljeno: 20-lip-2021].

[25] C. Dilmegani, „Recommendation Systems: Applications, Examples & Benefits", 25-stu-2020. [Na internetu]. Dostupno na: https://research.aimultiple.com/recommendation-system/. [Pristupljeno: 20-lip-2021].

[26] M. Maleković i M. Schatten, *Teorija i primjena baza podataka*. Sveučilište u Zagrebu Fakultet organizacije i informatike Varaždin, 2017.

[27] C. Strauch, *NoSQL Databases*. Hochschuleder Medien, Stuttgart, 2011.

[28] L. Schaefer, „What is NoSQL? NoSQL Databases Explained | MongoDB". [Na internetu]. Dostupno na: https://www.mongodb.com/nosql-explained. [Pristupljeno: 21-lip-2021].

[29] B. M. Sasaki, J. Chao, i R. Howard, *Graph Databases for Beginners*. Neo4j, 2018.

[30] M. Maleković i K. Rabuzin, *Uvod u baze podataka*. Fakultet organizacije i informatike, Sveučilište u Zagrebu, 2016.

[31] D. Sullivan, *NoSQL for Mere Mortals*. Addison-Wesley, 2015.

[32] Ian, „What is a Column Store Database?", 2016. [Na internetu]. Dostupno na: https://database.guide/what-is-a-column-store-database/. [Pristupljeno: 23-lip-2021].

[33] „Types of NoSQL Databases | MongoDB". [Na internetu]. Dostupno na: https://www.mongodb.com/scale/types-of-nosql-databases. [Pristupljeno: 24-lip-2021].

[34] G. Vaish, *Getting Started with NoSQL*. Packt Publishing Ltd., 2013.

[35] „Document Database | MongoDB". [Na internetu]. Dostupno na: https://www.mongodb.com/document-databases. [Pristupljeno: 24-lip-2021].

[36] A. Nayak, A. Poriya, i D. Poojary, „Type of NOSQL Databases and its Comparison with Relational Databases", *Int. J. Appl. Inf. Syst.*, sv. 5, izd. 4, str. 16–19, ožu. 2013.

[37] L. Schaefer, „NoSQL vs SQL Databases | MongoDB". [Na internetu]. Dostupno na: https://www.mongodb.com/nosql-explained/nosql-vs-sql. [Pristupljeno: 26-lip-2021].

[38] A. B. M. Moniruzzaman i S. A. Hossain, „NoSQL Database: New Era of Databases for Big data Analytics-Classification, Characteristics and Comparison", *International Journal of Database Theory and Application*, sv. 6, izd. 4. 2013.

[39] A. Hodler i M. Needham, *Graph Data Science (GDS) For Dummies*. John Wiley &

Sons, Inc., 2021.

[40] M. Besta *i ostali*, „Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries“, 2019.

[41] K. Rabuzin i M. Šestak, „Grafovske baze podataka-pregled istraživanja i budućih trendova“.

[42] I. Robinson, J. Webber, i E. Eifrem, *Graph Databases: new opportunities for connected data*. O'Reilly Media, Inc., 2015.

[43] „What is a Graph Database? - Developer Guides“. [Na internetu]. Dostupno na: https://neo4j.com/developer/graph-database/. [Pristupljeno: 02-srp-2021].

[44] „Graph Modeling Guidelines - Developer Guides“. [Na internetu]. Dostupno na: https://neo4j.com/developer/guide-data-modeling/. [Pristupljeno: 02-srp-2021].

[45] T. Cormen, D. Balkcom, i Khan Academy, „Representing graphs (article) | Algorithms | Khan Academy“. [Na internetu]. Dostupno na: https://www.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs. [Pristupljeno: 03-srp-2021].

[46] Y. Schulz, „A quick primer on graph databases | IT World Canada Blog“, 07-kol-2020. [Na internetu]. Dostupno na: https://www.itworldcanada.com/blog/a-quick-primer-on-graph-databases/434215. [Pristupljeno: 05-srp-2021].

[47] „Neo4j - an overview | ScienceDirect Topics“. [Na internetu]. Dostupno na: https://www.sciencedirect.com/topics/computer-science/neo4j. [Pristupljeno: 05-srp-2021].

[48] D. Fernandes i J. Bernardino, „Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB“, str. 373–380, 2018, doi: 10.5220/0006910203730380.

[49] „Neo4j Graph Platform - Developer Guides“. [Na internetu]. Dostupno na: https://neo4j.com/developer/graph-platform/. [Pristupljeno: 07-srp-2021].

[50] „Neo4j Desktop User Interface Guide - Developer Guides“. [Na internetu]. Dostupno na: https://neo4j.com/developer/neo4j-desktop/. [Pristupljeno: 07-srp-2021].

[51] „Neo4j Browser User Interface Guide - Developer Guides“. [Na internetu]. Dostupno na: https://neo4j.com/developer/neo4j-browser/. [Pristupljeno: 07-srp-2021].

[52] „Neo4j Bloom User Interface Guide - Developer Guides“. [Na internetu]. Dostupno na: https://neo4j.com/developer/neo4j-bloom/. [Pristupljeno: 07-srp-2021].

[53] „How-To: Neo4j ETL Tool - Developer Guides“. [Na internetu]. Dostupno na: https://neo4j.com/developer/neo4j-etl/. [Pristupljeno: 07-srp-2021].

[54] „Featured Graph Apps - Developer Guides“. [Na internetu]. Dostupno na: https://neo4j.com/developer/graph-apps/featured/. [Pristupljeno: 07-srp-2021].

[55] „Neo4j APOC Library - Developer Guides“. [Na internetu]. Dostupno na: https://neo4j.com/developer/neo4j-apoc/#apoc-intro. [Pristupljeno: 07-srp-2021].

[56] „Graph Data Science Algorithms | Graph Data Analysis | Neo4j“. [Na internetu]. Dostupno na: https://neo4j.com/product/graph-data-science-library/?ref=product. [Pristupljeno: 07-srp-2021].

[57] „Neo4j Labs - Neo4j Graph Database Platform“. [Na internetu]. Dostupno na: https://neo4j.com/labs/. [Pristupljeno: 07-srp-2021].

[58] „Cypher Query Language - Developer Guides“. [Na internetu]. Dostupno na: https://neo4j.com/developer/cypher/. [Pristupljeno: 08-srp-2021].

[59] „Getting Started with Cypher - Developer Guides“. [Na internetu]. Dostupno na: https://neo4j.com/developer/cypher/intro-cypher/. [Pristupljeno: 10-srp-2021].

[60] „Clauses - Neo4j Cypher Manual“. [Na internetu]. Dostupno na: https://neo4j.com/docs/cypher-manual/current/clauses/. [Pristupljeno: 12-srp-2021].

[61] „Indexes for search performance - Neo4j Cypher Manual“. [Na internetu]. Dostupno na: https://neo4j.com/docs/cypher-manual/current/administration/indexes-for-search-performance/. [Pristupljeno: 12-srp-2021].

[62] M. Needham i A. Hodler, *Graph Algorithms*, sv. 1. O’Reilly Media, Inc., 2019.

[63] N. Mathur, „Graph-Powered Recommendation Engines How the Power of Suggestion Drives Better Decisions & Higher Revenues Graph-Powered“, Neo4j, 2021.

[64] „Jaccard Similarity - Neo4j Graph Data Science“. [Na internetu]. Dostupno na: https://neo4j.com/docs/graph-data-science/current/alpha-algorithms/jaccard/. [Pristupljeno: 17-srp-2021].

[65] „Cosine Similarity - Neo4j Graph Data Science“. [Na internetu]. Dostupno na: https://neo4j.com/docs/graph-data-science/current/alpha-algorithms/cosine/. [Pristupljeno: 18-srp-2021].

[66] „Pearson Similarity - Neo4j Graph Data Science“. [Na internetu]. Dostupno na: https://neo4j.com/docs/graph-data-science/current/alpha-algorithms/pearson/. [Pristupljeno: 18-srp-2021].

[67] „Using Neo4j from Java - Developer Guides“. [Na internetu]. Dostupno na: https://neo4j.com/developer/java/. [Pristupljeno: 22-srp-2021].

[68] „Spring Data Neo4j - Developer Guides“. [Na internetu]. Dostupno na: https://neo4j.com/developer/spring-data-neo4j/. [Pristupljeno: 22-srp-2021].

Popis slika

Popis tablica