Razvoj sustava preporuka pomoću graf baze podataka

Šuvak, Katarina

Sveučilište Josipa Jurja Strossmayera u Osijeku

Odjel za matematiku

Sveučilišni diplomski studij matematike
smjer: Matematika i računarstvo

Razvoj sustava preporuka pomoću graf baze podataka

Diplomski rad

Mentor:

izv. prof. dr. sc. Domagoj Matijević

Komentor:

dr. sc. Mateja Đumić

Kandidat:

Katarina Šuvak

Osijek, 2024.

Sadržaj

1 | Uvod

Relacijske baze podataka definirao je Edgar F. Codd prvi put sedamdesetih godina prošloga stoljeća. U relacijskim bazama podataka podaci se spremaju u retke tablica pri čemu je svaki redak jedinstven. Stupci tablica sadrže atribute podataka kojima je jednostavno uspostaviti vezu između podataka koji se nalaze u drugim tablicama. Gotovo sve relacijske baze podataka koriste strukturirani upitni jezik (engl. *Structured Query Language*, SQL) i mogu biti vrlo složene te su ograničene na rad sa strukturiranim podacima. Kako 1990-ih web aplikacije značajno dobivaju na popularnosti, postaje teže upravljati podacima koje se nalaze u relacijskim bazama. Količine podataka postaju sve veće, tipovi podataka su različitiji, broj podataka se stalno povećava, a pretraživanje se treba odviti u što kraćem vremenu. Ovakva promjena dovodi do razvoja nerelacijskih baza podataka koje se nazivaju NoSQL baze podataka. Ove baze nude brzinu i skalabilnost u radu s podacima te su jednostavne za korištenje. Zbog toga su postale preferirani izbor za zadovoljenje sve većih zahtjeva upravljanja podacima.

Jedna od vrsta NoSQL baza podataka jest graf baza. Ona vrlo jednostavno rješava probleme promjena i prilagođavanja podataka. Naglasak u graf bazama je na vezama između podataka, što omogućuje jednostavno sastavljanje upita i brze odgovore na njih. Među najpoznatijim sustavima za upravljanje graf bazom je Neo4j. Zbog svoje stabilnosti i sigurnosti Neo4j koriste velike tvrtke poput eBay Inc., Fujitsu, AWS (Amazon Web Services), Bank of America i mnoge druge.

Rad je organiziran kako slijedi: u drugom poglavlju dan je pregled NoSQL baza podataka i njihovih osnovnih karakteristika. Objašnjeno je što su NoSQL baze podataka, koje vrste postoje i kada bi bilo dobro koristiti se njima. Treće poglavlje daje pregled matematičke pozadine graf baza podataka i bitnih algoritma koji se koriste nad njima. Objašnjena je Neo4j graf baza te jezik Cypher. U četvrtom poglavlju ukratko je objašnjen sustav preporuka, vrste sustava preporuka, kako se dijele i kako se postižu koristeći graf baze podataka. Radi svrhe boljeg razumijevanja Neo4j baze podataka i Cypher upitnog jezika, u sklopu ovog rada, napravljena je web aplikacija. Web aplikacija podatke sprema u Neo4j graf bazu podataka, te koristi Cypher upitni jezik kako bi na temelju spremljenih podataka napravila preporuku knjiga korisniku. U petom poglavlju može se vidjeti implementacija samog sustava kao i web sučelja.

2 | NoSQL baze podataka

NoSQL je način dizajniranja baza podataka koji omogućuje pohranu podataka u strukture koje se razlikuju od onih u relacijskim bazama podataka. Također, NoSQL je vrsta distribuirane baze podataka zbog čega se spremljeni podaci mogu nalaziti na različitim serverima. Na taj način se povećava pouzdanost i otpornost baze podataka na greške što smanjuje mogućnost prestanka rada baze.
Osim što NoSQL sustavi mogu raditi i sa strukturiranim i nestrukturiranim podacima, mogu i brzo raditi s velikom količinom podataka (engl. *Big Data*). To je potaknulo organizacije kao što su Facebook, Twitter, LinkedIn i Google da prihvate NoSQL sustave. Korištenjem NoSQL sustava, ove tvrtke mogu na jednostavan način koristiti veliku količinu podataka, te u njima pronalaziti obrasce što im omogućuje povećanje profita.
Najbitnija svojstva prema [24] koje NoSQL baze posjeduju su:

- Kompatibilnost više podatkovnih modela: NoSQL baza nema točno definiranu shemu, tj. fiksni model podataka. Zbog toga se kaže da su NoSQL baze dinamičke. NoSQL baze koriste modele koji mogu biti orijentirani na dokumente, u obliku grafa, rječnika i dr.

- Poboljšana skalabilnost i dostupnost: NoSQL rješenja mogu imati konzistentnu arhitekturu bez servera u svim dijelovima mreže na kojoj se nalaze što ih čini prilagođenima za aplikacije u oblaku. Podaci se mogu particionirati i rasporediti na različita mjesta za pohranu.

- Podaci mogu biti distribuirani na nekoliko čvorova mreže, omogućujući čitanje i obradu podataka na svakom čvoru. To svojstvo osigurava minimalno vrijeme čekanja na odgovore upita.

- NoSQL baze su robusne i imaju minimalno vrijeme zastoja. Osim što podaci mogu biti smješteni na različitim mjestima, stvaraju se i kopije tih podataka te se one čuvaju. Ako se neka greška ili kvar dogodi na jednom mjestu, podatak ostaje dostupan na drugom te mu se može pristupiti.

2.1 CAP teorem

Eric Brewer sa Sveučilišta u Kaliforniji je 1998. godine predstavio CAP teorem koji se odnosi na nerelacijske baze podataka. Po [3] i [12] teorem tvrdi da bilo koja distribuirana baza podataka, odnosno nerelacijska baza čiji su podaci spremljeni na više od jednog mjesta, može istovremeno imati zadovoljena najviše dva od tri navedena svojstva:

1. **Konzistentnost** (engl. *Consistency*):
   Svi klijenti u isto vrijeme vide iste podatke, bez obzira na koji čvor distribuiranog sustava se povezuju. Da bi to bilo moguće, svaka promjena se primjenjuje na sve ostale dijelove sustava. Zbog toga postoji jamstvo da svaki dio sustava uspješno vraća iste, najnovije podatke.

2. **Dostupnost** (engl. *Availability*):
   Sustavu se može pristupiti u bilo kojem trenutku, uvijek je dostupan i nema zastoja. Svaki zahtjev za pisanjem ili čitanjem podataka treba biti uspješno obrađen bez poruka o grešci. Zbog ovog svojstva nema garancije da su se pojavili najnoviji ili ažurirani podaci.

3. **Tolerancija na particiju** (engl. *Partition Tolerance*):
   Sustav nastavlja raditi čak i ako dolazi do izostanka poruka u komunikaciji između različitih čvorova sustava. Poslužitelji (čvorovi sustava) mogu biti podijeljeni u više grupa koje mogu međusobno komunicirati i dijeliti podatke što može ovu komunikaciju učiniti nepouzdanom.
   Kada se dogodi kvar dijela distribuiranog sustava postoje dvije mogućnosti: otkazati operaciju i time smanjiti dostupnost, a osigurati dosljednost ili nastaviti s operacijom i osigurati dostupnost, ali riskirati nedosljednost. Stoga, ako postoji particija sustava, treba birati između dosljednosti ili dostupnosti.

NoSQL baze podataka se, s obzirom na gore navedena svojstva, dijele na tri kategorije:

- CP baze podataka (engl. *Consistency and Partition tolerance databases*): baze podataka koje imaju svojstva konzistentnosti i tolerancije na particiju, ali žrtvuju dostupnost. U ovom slučaju, kada dođe do particije, sustav čini nekonzistentni čvor nedostupnim sve dok se problem ne riješi. Primjeri CP baza podataka su MongoDB, Redis i Neo4j.

- AP baze podataka (engl. *Availability and Partition tolerance databases*): baze podataka koje nude dostupnost i toleranciju na particiju, ali ne i konzistentnost u slučaju kvara. Svi čvorovi ostaju dostupni kada dođe do particije, ali neki mogu vratiti stariju verziju podataka. Primjeri su CouchDB, Cassandra i ScyllaDB.

- CA baze podataka (engl. *Consistency and Availability*): baze podataka koje osiguravaju konzistentnost i dostupnost, ali ne mogu pružiti toleranciju na pogreške. Ovdje se CAP teorem i NoSQL baze podataka ne poklapaju jer ne

postoji sustav koji može izbjeći particioniranje, tj. raditi savršeno tako da se greške među dijelovima sustava uopće ne pojavljuju.

Relacijske baze podataka pridržavaju se ACID modela što znači da pouzdana baza mora imati četiri svojstva [1]:

- **Atomarnost** (engl. *Atomic*) - transakcija, odnosno po [4], logička jedinica naredbi koja čita podatke iz baze ili ju mijenja, mora se odviti cijela. Greška u transakciji dovodi do prekida rada i ne događaju se nikakve promjene u podacima ili na bazi.
- **Konzistentnost** (engl. *Consistent*) - nakon dovršetka transakcije baza podataka je u konzistentnom stanju, nema djelomičnih ažuriranja podataka i nepravilne promjene baze.
- **Izolacija** (engl. *Isolated*) - transakcije su međusobno izolirane, nemaju utjecaja jedne na druge, točnije rezultat jedne transakcije vidljiv je ostalima tek kada je ona u potpunosti dovršena.
- **Trajnost** (engl. *Durable*) - u slučaju kvara rezultati svih transakcija ostaju nepromijenjeni.

Održavanje ACID svojstava za NoSQL bazu podataka može usporiti njene performanse i uzrokovati velike troškove [15]. Kao rezultat toga nastaje nastaje BASE model temeljen na CAP teoremu. Ovaj model je prilagođen fleksibilnosti koju nude NoSQL i slični sustavi koji rade s nestrukturiranim podacima. BASE se sastoji od tri principa:

- **Osnovna dostupnost** (engl. *Basic Availability*):
  Umjesto održavanja jednog velikog mjesta pohrane podataka i fokusiranja na toleranciju grešaka te pohrane, NoSQL baze pohranjuju podatke na više različitih mjesta, odnosno u više čvorova distribuiranog sustava. U slučaju kvara sustava i dalje se može pristupati podacima. Sustav neće biti u potpunosti ugašen, već samo čvorovi na kojima se nalazi kvar.
- **Meko stanje** (engl. *Soft State*):
  Stanje sustava može se mijenjati tijekom vremena i bez novih unosa ili upita. Podaci se mogu mijenjati bez djelovanja korisnika ili aplikacije zbog raznih čimbenika, primjerice zbog naknadnih ažuriranja ili algoritama za optimiziranje brzine rada baze. To dovodi do nekonzistentnosti podataka među čvorovima, ali ostvaruje se fleksibilnost baze podataka. Ovaj pristup omogućuje NoSQL bazama podataka da obrađuju velike količine podataka što može biti korisno za određene slučajeve upotrebe kao što su analitika u stvarnom vremenu, mreže za isporuku sadržaja ili distribuirani sustavi za predmemoriju.
- **Konačna konzistententnost** (engl. *Eventual Consistency*):
  Iako sustav nema svojstvo konstantne konzistentnosti, to ne znači da se ona nikada neće postići. Sustav će uvijek na kraju biti konzistentan.

Općenito, BASE model pruža manje sigurnosti u usporedbi s ACID-om što znači da podaci nisu uvijek konzistentni, no zato omogućuje jednostavnije skaliranje i fleksibilnost pri radu s velikim količinama promjenjivih podataka. BASE model je prikladan za sustave koji zahtijevaju veliku dostupnost poput društvenih mreža, web stranica ili online trgovina, tj. općenito sustava koji moraju obrađivati mnogo različitih podataka.

Treba napomenuti da je konzistentnost definirana u CAP teoremu drugačija od konzistentnosti zajamčene u transakcijama baze podataka u ACID modelu. Također, neki NoSQL sustavi mogu težiti postizanju ACID svojstava.

2.2 Podjela NoSQL baza podataka

NoSQL baze sadrže veliki broj različitih modela u koje se podaci spremaju, te se mogu podijeliti prema strukturama za spremanje podataka kako slijedi [7]:

1. **Ključ - vrijednost baze podataka** (engl. *Key/value systems*)
   Ovakve baze podataka obično se smatraju najjednostavnijima. Podaci su spremljeni kao skup parova ključ-vrijednost, odnosno u obliku rječnika. Svaki ključ je jedinstven i služi kao identifikator. Ključevi i pripadne vrijednosti mogu biti raznoliki, od jednostavnih objekata poput niza podataka do objekata koji se sastoje od više dijelova, npr. datoteke. Baze podataka ključ-vrijednost mogu se particionirati i horizontalno skalirati na razini koju druge vrste baza podataka teško postižu. Ključevi se mogu sortirati po različitim kriterijima, kao što su abecedni, numerički, kronološki ili prema veličini vrijednosti koju predstavljaju. Ove baze podataka imaju fleksibilnu shemu i bolju izvedbu od ostalih NoSQL baza podataka u određenim slučajevima.
   Njihove prednosti su:

   - Skalabilnost: prilikom dodavanja novih podataka, ove baze prave više različitih mjesta pohrane za te podatke što se zove horizontalno skaliranje. Automatski distribuiraju podatke na više servera kako bi se smanjilo opterećenje na jednom serveru.
   - Jednostavnost korištenja: slično objektno orijentiranom programiranju, ključ-vrijednost baze omogućuju mapiranje objekata iz stvarnoga svijeta direktno u softverske objekte što čini pohranu podataka intuitivnijom.
   - Brzina izvođenja: za razliku od relacijskih baza podataka, ključ-vrijednost baze ne zahtijevaju operacije spajanja tablica (engl. *join*). Umjesto toga, podacima se direktno pristupa preko ključa, što rezultira bržim pristupom podacima.

   Zbog svoje jednostavnosti posjeduju i neke nedostatke.

   - Odsutnost složenih upita: podatkovne operacije se uglavnom odvijaju kroz jednostavne izraze jezika upita kao što su GET, PUT i DELETE te su operacije filtriranja i sortiranja ograničene i kompleksne.

- Loše upravljanje shemom: dizajn ovakve baze podataka ne nameće nikakvu konkretnu shemu, svatko ju može mijenjati i stoga se mora temeljito planirati kako u budućnosti ne bi nastali problemi i nedosljednosti.

Ključ-vrijednost baze podataka preporučuje se koristiti u slučaju brojnih malih i kontinuiranih čitanja i pisanja. Pristup podacima je brz i učinkovit, a promjene ne utječu na performanse. Isto tako, prikladne su za pohranjivanje osnovnih podataka nad kojima se ne izvode složeni upiti i nema mnogo ažuriranja. Često se koriste u aplikacijama s velikim brojem korisnika jer mogu brzo pohraniti i dohvatiti informacije o sesiji kao što su tokeni za autentifikaciju korisnika, korisničke postavke ili privremeni podaci. Koriste se za upravljanje sesijama igrača u mrežnim igrama za veći broj igrača. Ove igre zahtijevaju upravljanje velikim brojem sesija igrača u stvarnom vremenu pa se ove baze podataka često koriste kao dijelovi predmemorije za ubrzavanje odgovora aplikacije.
Primjeri takvih baza podataka su: Azure Cosmos DB, ArangoDB, Amazon DynamoDB, Aerospike, Couchbase, ScyllaDB.

2. **Dokument baze podataka** (engl. *Document databases*)
Kao što samo ime sugerira, dokument baze podataka spremaju podatke u dokumente koji mogu biti u JSON, YAML, BSON ili XML formatu. Ove baze ne podržavaju veze i *join* operacije. Za identifikaciju i pohranu podataka koristi se unutarnja struktura dokumenta. Dokumenti se u bazi podataka adresiraju putem jedinstvenog ključa koji predstavlja taj dokument. Ovaj ključ je obično jednostavan identifikator, kao što je niz znakova, URI (*Uniform Resource Identifier*)[1] ili putanja do dokumenta. Ključ se koristi za dohvaćanje dokumenta iz baze podataka, a obično baza podataka ima indeks na ključu kako bi se ubrzalo pronalaženje dokumenta.

   Još jedna značajka definiranja baze podataka orijentirane na dokumente je API ili jezik upita za dohvaćanje dokumenata na temelju njihovog sadržaja. Ovakve baze se mogu smatrati potklasom ključ-vrijednost baza podataka. Razlika je u načinu na koji se podaci obrađuju. Umjesto čitanja ključa pa pristupanju vrijednostima, dokument baze podataka se oslanjaju na unutarnju strukturu dokumenata kako bi se pristupilo podacima koji se čitaju iz njih. Ovakav način pohrane podataka osmišljen je kako bi se lakše radilo s jednostavnim podacima u kombinaciji s modernim tehnikama programiranja.
   U primjerima 1 i 2 možemo vidjeti dva dokumenta koji dijele neke zajedničke strukturne dijelove, ali imaju i zasebne. Te dijelove, zajedno s tekstom i ostalim podacima unutar dokumenta, zovemo sadržajem dokumenta. Sadržaju se može pristupiti putem metoda dohvaćanja ili uređivanja. Za razliku od relacijske baze podataka gdje svaki zapis sadrži ista polja, ostavljajući neiskorištena polja prazna, u dokumentima nema praznih polja. Ovaj pristup omogućuje dodavanje novih informacija nekim zapisima bez potrebe da svaki drugi zapis u bazi podataka dijeli istu strukturu. Primjer ovakve baze

[1] Jedinstvena adresa resursa dostupnog u računalnoj mreži

je MongoDB.

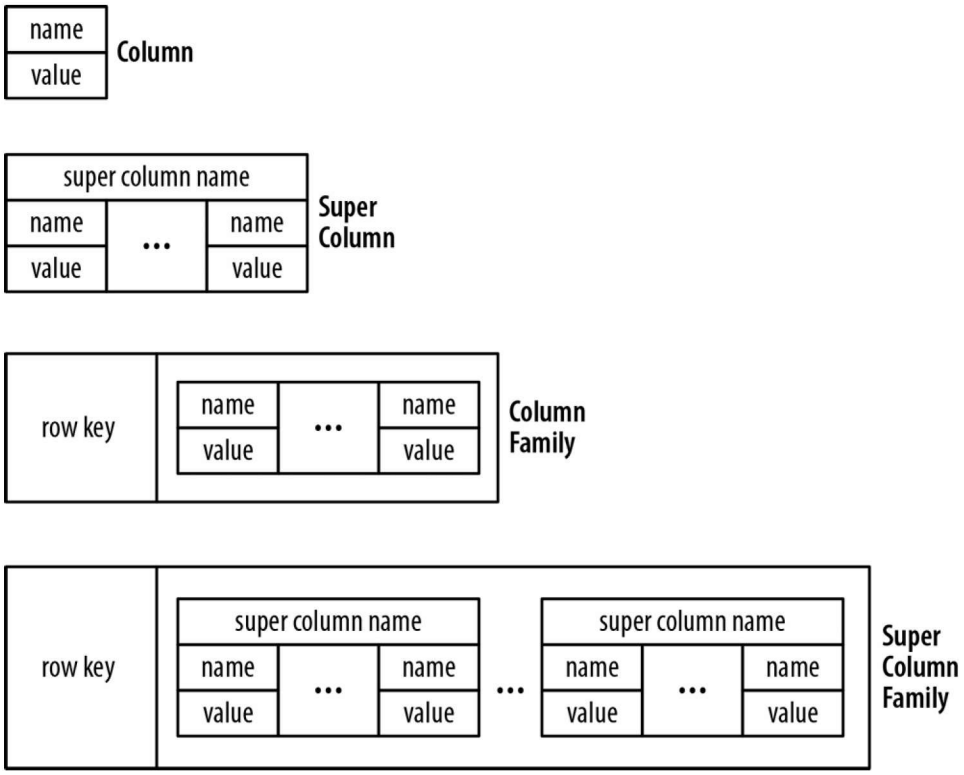
Primjer 1. *JSON dokument*

```
{
  "person": {
    "name": "John Doe",
    "age": 30,
    "address": {
      "city": "New York",
      "zipcode": "10001"
    }
  }
}
```

Primjer 2. *XML dokument*

```
<person>
  <name>John Doe</name>
  <age>30</age>
  <location>
    <city>Los Angeles</city>
  </location>
</person>
```

3. **Stupčane baze podataka** (engl. *Columnar databases*)
   Stupčane baze podataka pohranjuju podatke u fleksibilne stupce koristeći mapiranje podataka po stupcu, retku i vremenskoj oznaci. Svaki stupac sastoji se od naziva i vrijednosti. Stupci su međusobno neovisni tako da se jednom stupcu može pristupati bez referenciranja na neki drugi stupac. Stupci se mogu kombinirati u super-stupce koji se sastoje od niza stupaca. Stupac koji se sastoji od više stupaca naziva se familijom stupaca, a analogno tome, stupac sastavljen od više super-stupaca, familijom super-stupaca što možemo radi lakše predodžbe vidjeti na Slici 2.1.

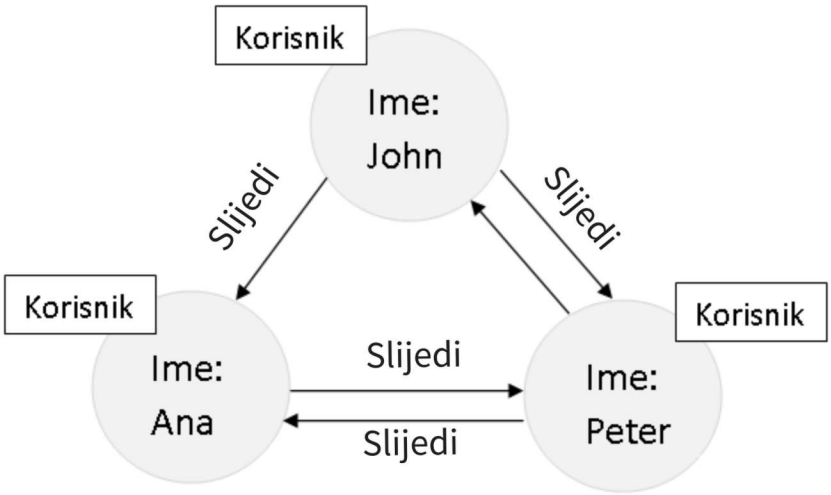


Slika 2.1: Stupci, super-stupci i pripadne familije, Izvor: [6]

Naziv i format stupca može se razlikovati po redovima čak i unutar iste tablice, dakle mogu biti različitih veličina. Sve vrijednosti koje pripadaju jednom stupcu smještene su na zasebnim memorijskim lokacijama što omogućava efikasno korištenje memorije i podržava postojanje distribuiranog sustava. Budući da su podaci pohranjeni u stupcima, upiti za određenu vrijednost brzo se izvršavaju jer se cijeli stupac može brzo učitati i pretraživati. Ovakva baza je efikasna jer se ne mora čitati cijela tablica prilikom upita. Stupčane baze podataka često podržavaju relacijski model jer iz stupaca uvijek možemo dobiti redove. U principu, stupčane baze podataka vrlo učinkovito rade kompresiju podataka, skalabilne su, pogodne su za rad s podacima koji se nalaze na više različitih mjesta te se iz njih se mogu brzo čitati podaci. Zbog jednostavnih upita koji se odvijaju na stupcima vrijeme izvršavanja agregacijskih funkcija nije veliko, ali svaki kompliciraniji upit može potrajati. Migracije podataka su zahtjevne i obično nisu optimizirane za stvaranje ponovljivih složenih struktura podataka ili veza među podacima. Primjeri ovakvih baza su Cassandra i HBase.

4. **Graf baze**(*engl. Graph databases*)
Graf baza podataka je NoSQL baza podataka koja koristi matematičku teoriju grafova za prikaz podataka i njihovih veza. Graf se sastoji od čvorova, odnosno vrhova koji su međusobno povezani bridovima. Svaki čvor ima svojstva ili atribute koji ga opisuju, dok brid predstavlja vezu između tih čvorova. Bridovi imaju svoje oznake, a u nekim slučajevima svojstva i smjer. Primjer

jednog dijela graf baze koji se sastoji od tri čvora oznake Korisnik sa svojstvom Ime i veza između njih vidimo na Slici 2.2.



Slika 2.2: Primjer povezanih čvorova u bazi (prema: [19])

Za razliku od drugih baza podataka, graf baze stavljaju naglasak na bridove, odnosno veze između podataka. Oni omogućuju direktnu povezanost podataka, ne moraju se pronalaziti niti uspostavljati kroz upite. Zbog toga su ovakve baze optimizirane za brzo kretanje kroz podatke te se u vrlo kratkom vremenu može obići velik broj čvorova. Isto tako dodavanje novih čvorova i veza je jednostavno jer se struktura baze može mijenjati. Primjeri graf baza su Neo4j, Aerospike, Amazon Neptune, ArangoDB, Azure Cosmos DB, NebulaGraph i dr.
Više o graf bazama bit će dano u idućem poglavlju.

Odluka koristiti relacijsku ili nerelacijsku bazu podataka ovisi o kontekstu, strukturi, mogućnostima, svojstvima i o potrebi. Relacijske baze podataka dobra su opcija ako je potreban strukturirani model i kada se želi osigurati konzistentnost podataka u svim tablicama. Glavne prednosti relacijskih baza i SQL-a bi bile: dokumentiranost i jednostavnost učenja, široka podržanost i razumljivost, jednostavnost za rad s agregacijskim funkcijama za veće skupove podataka kao što je izračunavanje sume ili prosjeka i osiguravanje valjanosti podataka.
Međutim, relacijske baze podataka nisu uvijek najbolji izbor u pogledu fleksibilnosti ili skaliranja. Operacije spajanja tablica i prolazak kroz spojene tablice su skupe. NoSQL je bolji izbor kada se radi o skupovima podataka s promjenjivom strukturom i veličinom, no nije bolji kada se nad podacima trebaju provoditi kompleksni upiti i kada podaci u svakom trenutku trebaju biti točni i konzistentni. Kao zaključak, razlike između relacijskih i NoSQL baza podataka možemo vidjeti u Tablici 2.1.

Relacijske baze podataka	NoSQL baze podataka
ACID svojstva	BASE svojstva
Podaci u tablicama	Razne strukture za spremanje podataka poput grafa i rječnika
Komplicirani upiti	Jednostavni, ali brzi upiti
Nisu pogodne za hijerarhijske podatke	Pogodne za hijerarhijske podatke
Fiksna i predefinirana shema	Dinamička shema
Vertikalno skalabilne - resursi za spremanje podataka se nadograđuju i povećavaju	Horizontalno skalabilne
Koriste manje prostora	Koriste više prostora zbog pamćenja svih veza
Uvijek najnoviji podaci	Ažuriranost podataka može kasniti
Sigurne, većinom besplatne i podrška na više platformi	Jednostavne za korištenje, bolje performanse, fleksibilan alat

Tablica 2.1: Usporedba relacijskih i NoSQL baza podataka.

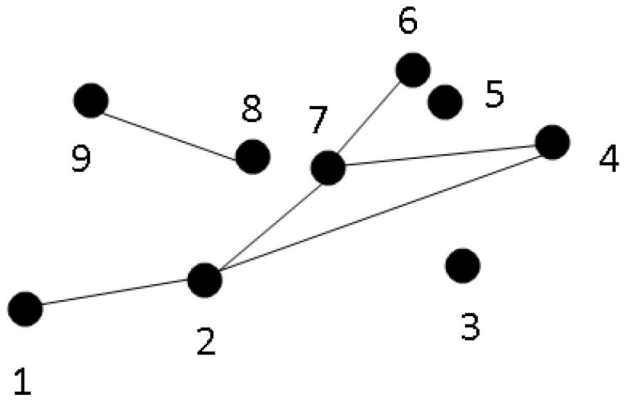
3 | Graf baze podataka

3.1 Grafovi

Kako bi lakše shvatili što su graf baze podataka i na koji način one funkcioniraju prvo moramo uvesti nekoliko osnovnih pojmova iz teorije grafova[2].
Jednostavno objašnjeno, graf je skup točaka koje zovemo vrhovi ili čvorovi zajedno sa spojnicama među vrhovima koji se nazivaju bridovi.

Definicija 1. *Graf G je uređena trojka $G = (V(G), E(G), \psi_G)$ koja se sastoji od nepraznog skupa $V = V(G)$ čiji su elementi vrhovi od G, skupa $E = E(G)$ koji je disjunktan s $V(G)$, čiji su elementi bridovi od G i funkcije incidencije ψ_G koja svakom bridu grafa G pridružuje neuređeni par ne nužno različitih vrhova od G.*

Funkcija ψ_G ne mora biti injekcija, tj. moguće je da dva različita vrha budu spojena s više bridova, odnosno da graf ima višestruke bridove. Ako su $u, v \in V(G)$ i $e \in E(G)$ tako da je $\psi_G = \{u, v\}$ kažemo da e spaja u i v i oni su krajevi od e. Za krajeve u, v brida e kažemo da su incidentni s bridom e, a za dva vrha incidentna s istim bridom kažemo da su susjedni. Primjer jednostavnog grafa G s vrhovima $V = \{1, ..., 7\}$ i skupom bridova $E = \{\{1,2\}, \{1,5\}, \{2,5\}, \{3,4\}, \{5,7\}\}$ dan je na Slici 3.1.

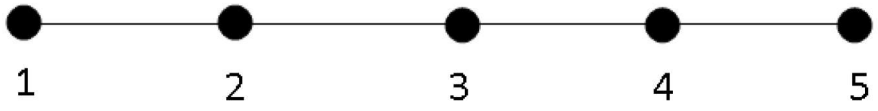


Slika 3.1: Primjer grafa

Definicija 2. *Usmjereni graf ili digraf D je uređena trojka $(V(D), A(D), \psi_D)$ koja se sastoji od nepraznog skupa $V(D)$ vrhova, $A(D)$ skupa lukova (ili usmjerenih bridova) i funkcije incidencije ψ_D koja svakom luku $\boldsymbol{a}$ pridružuje uređeni par vrhova $\boldsymbol{u}$, $\boldsymbol{v}$ spojenih s $\boldsymbol{a}$. Vrh $\boldsymbol{u}$ naziva se početni vrh, a vrh $\boldsymbol{v}$ krajnji vrh od $\boldsymbol{a}$.*

Definicija 3. *Put je jednostavan graf P_n s n vrhova definiran s:*

$$V(P_n) = \{v_1, v_2, ..., v_n\}, \;\; E(P_n) = \{v_i v_{i+1} : i = 1, ..., n-1\}.$$

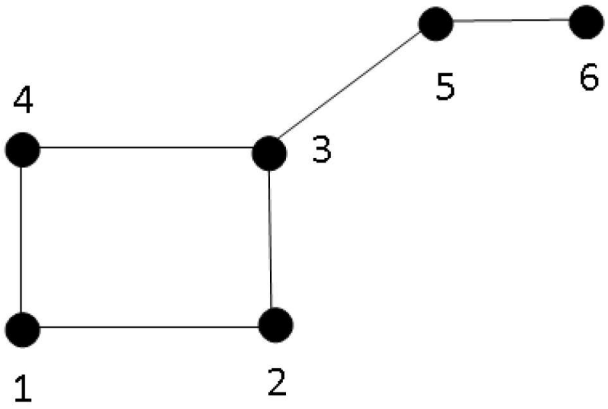


Slika 3.2: Put P_5

Definicija 4. *Ciklus s n vrhova je jednostavan graf definiran s:*

$$V(C_n) = \{v_1, v_2, ..., v_n\}, \;\; E(C_n) = \{v_1v_2, v_2v_3, ..., v_{n-1}v_n, v_nv_1\}.$$

Na slici 3.3 dan je primjer grafa koji ima ciklus.



Slika 3.3: Ciklus

Definicija 5. *Ako $e \in E$ i $v \in V$, tada je moguće da $\psi(e) = \{v, v\}$, tj. brid može spajati neki vrh sa samim sobom i zove se petlja.*

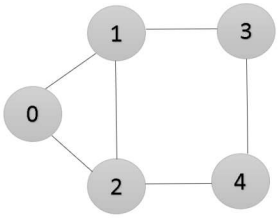
3.2 Algoritmi nad grafovima

Kvaliteta i brzina upita i pretraživanja po graf bazi ovise i o algoritmima koji se nad njom koriste. Oni optimiziraju same prolaske kroz graf i služe kako bi dobili točne podatke. Neki osnovni algoritmi za rad s grafovima bi po [10] bili:

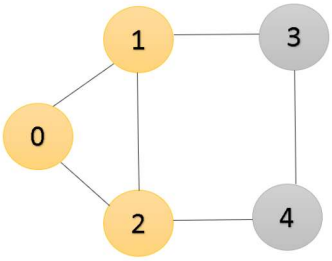
1. **Pretraživanje u širinu** (engl. *Breadth First Search - BFS*)
   Jedan od temeljnih algoritama je pretraživanje u širinu, odnosno u ovom slučaju, grafa po širini. Počevši od početnog zadanog čvora BFS istražuje sve

susjedne čvorove prije nego što krene dublje u graf. Kako bi se izbjegao ulazak u ciklus koristi se polje za bilježenje posjećenih čvorova. Za prolaz kroz graf BFS koristi red kao pomoćnu strukturu. Svi neposjećeni čvorovi koji se nalaze u trenutnom nivou stavljaju se u red, a posjećeni izlaze iz reda. [11]

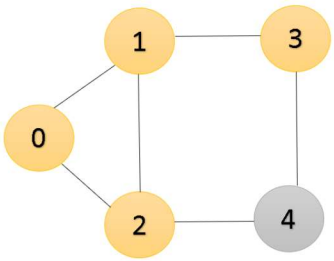
Primjer 3. *Zadan je graf sa skupom čvorova* $\{0,1,2,3,4\}$ *kao na slici ispod, polje posjećenih čvorova koje je označeno s P i red Q.*



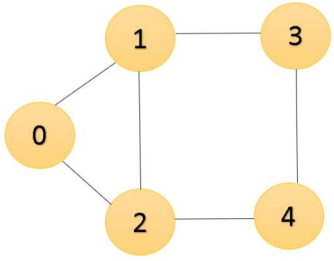
Pogledajmo nekoliko iteracija BFS algoritma:
1. Odabrali smo početni čvor 0, spremimo ga u P i Q
$P = \{0\}$
$Q = \{0\}$
2. Susjedi od 0 su 1 i 2. Njih stavljamo u polje posjećenih P i u red Q, a iz Q izbacimo 0.
$P = \{0,1,2\}$
$Q = \{1,2\}$



3. Prvi u redu Q je 1 i gledamo koji je njegov susjed. To bi bio 3 tako da 3 ubacujemo u Q i P, a 1 izbacujemo iz Q.
$P = \{0,1,2,3\}$
$Q = \{2,3\}$



4. Za kraj, idući u Q je 2 i njegov susjed je 4 kojeg stavljamo u P te iz Q izbacujemo 2.
$P = \{0, 1, 2, 3, 4\}$
$Q = \{3, 4\}$

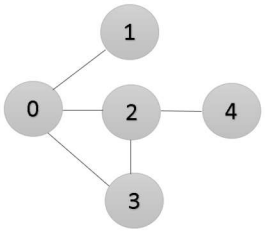


Ostaje nam još provjeriti susjeda od 3 te i njega izbacujemo iz Q. U Q ostane 4 koji nema susjede, izbacimo ga iz Q i algoritam je gotov.

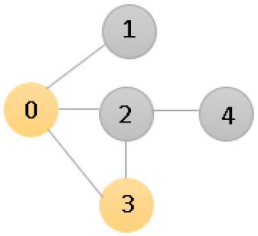
Ovaj algoritam se najčešće koristi za pronalaženje najkraćeg puta između dva čvora, pronalaženje ciklusa u grafu ili za identificiranje spojenih čvorova.

2. **Pretraživanje u dubinu** (engl. *Depth-first search - DFS*)
DFS, poput BFS-a započinje s početnim čvorom, ali umjesto vraćanja na istu razinu, ide što je dublje moguće duž svake grane prije nego se vrati. Budući da se radi o grafovima koji imaju cikluse, DFS kao i BFS koristi polje za pamćenje posjećenih čvorova. Kao pomoćnu strukturu koristi stog. [16]

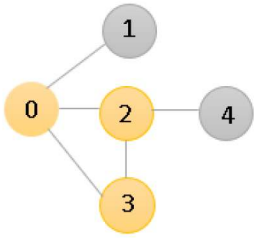
Primjer 4. *Zadan je graf sa skupom čvorova* $\{1, 2, 3, 4\}$ *kao na slici ispod, polje posjećenih čvorova P i stog S.*



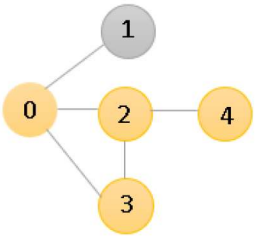
1. Krećemo iz početnog čvora 0 i njega stavljamo u polje posjećenih čvorova P, a u stog stavljamo njegove susjede 1, 2, 3.
$P = \{0\}$
$S = \{1, 2, 3\}$



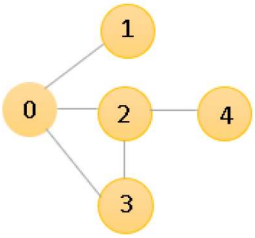
2. Izbacujemo 3 iz stoga i stavljamo ga u skup posjećenih čvorova P.
P={0, 3}
S={1, 2}



3. Idući u stogu je 2, izbacujemo ga iz S i stavljamo u P, a njegovog susjeda 4 stavimo u skup S.
P={0, 3, 2}
S={1, 4}



4. Izbacujemo 3 iz S i stavljamo u P:
P={0, 1, 2, 3}
S={4}



Algoritam je koristan za prepoznavanje ciklusa, traženje određenih čvorova ili uzoraka među skupovima čvorova.

3. **PageRank**
PageRank algoritam [5] se koristi za ocjenu važnosti nekog čvora u grafu. PageRank su razvili osnivači Googlea i koriste ga web pretraživači. Kada se postavi upit za pretraživanje, Google prvo popisuje sve web stranice povezane s traženim riječima, a zatim ih rangira prema njihovom rangu, PR(v), gdje je $v \in V(G)$. Što stranica sadrži više podatka koji se podudaraju sa zadanim unosom to je PR(v) veći.
Osim broja poveznica koje vode na određenu stranicu, važnost te stranice ovisi i o važnosti stranica koje na nju pokazuju. Stranica koja je manje važna (ima manje poveznica prema njoj) pridonosi manje u računanju PageRanka nego stranica veće važnosti (s više poveznica prema njoj). Također, važnost stranice se smanjuje ako sadrži više poveznica koje vode prema drugim stranicama. Općenito, vrijednost PageRanka stranice u možemo računati:

$$PR(u) = \sum_{\substack{v \in V \\ \forall u \neq v}} \frac{PR(v)}{L(v)}$$

pri čemu je je $L(v)$ broj stranica na koje u pokazuje.

4. **Dijkstrin algoritam**
Dijkstrin algoritam [17] radi pretraživanje u širinu, ali s dodatnom analizom kako bi pronašao najkraći put između dva čvora u grafu. Koraci algoritma su:

   1. Odaberemo početni i završni čvor. Početni čvor se dodaje skupu čvorova s poznatim najkraćim putem od početnog čvora koji je na početku prazan. Taj skup zvat ćemo skup riješenih čvorova. Tom čvoru se pridružuje vrijednost 0 jer je njegova udaljenost od samoga sebe 0.
   2. Prolazimo prvo po širini krenuvši od početnog čvora sve do njegovih najbližih susjeda i bilježimo duljinu puta do svakog susjednog čvora.
   3. Među najbližim susjedima odaberemo čvor koji ima najkraću udaljenost i njega stavljamo u skup. Ako svi susjedi imaju jednaku udaljenost, algoritam nasumično odabire čvor.
   4. Posjetimo susjede čvorova iz skupa riješenih čvorova i bilježimo udaljenosti od početnog čvora, ako su čvorovi već u skupu riješenih, njih ne trebamo jer već znamo njihove vrijednosti.
   5. Koraci 3. i 4. ponavljaju se sve dok se završni čvor ne nađe u skupu riješenih čvorova.

   Dijkstrin algoritam često se koristi za pronalaženje najkraćih puteva u stvarnom svijetu, primjerice za navigaciju i logistiku.

3.3 Graf baza

Kako je već ranije navedeno, graf baza podataka je vrsta NoSQL baze podataka koja koristi strukturu grafa za pohranu i obradu podataka putem operacija poput stvaranja, čitanja, ažuriranja i brisanja (poznatih kao CRUD [1] operacije). U ovom tipu baze, entiteti su predstavljeni kao čvorovi i svaki može imati više od jednog svojstva. Ako čvorovi u grafu imaju svojstva on se naziva graf svojstava (engl. *Property graph*). Čvorovi se mogu obilježiti jednom ili više oznaka radi grupiranja. Na primjer, oznake mogu biti zaposlenik, odjel, projekt - slično kao imena tablica u relacijskim bazama.

Veze među čvorovima predstavljene su usmjerenim bridovima, dakle jednoznačno je određeno koji je čvor početni, a koji završni. Bridovi također imaju imena i mogu imati vlastita svojstva. Naglasak se pri radu s graf bazama stavlja na veze među podacima za razliku od drugih vrsta baza koje primjerice koriste strane ključeve za stvaranje veza. Veze u grafu prirodno stvaraju puteve. Upit, odnosno obilazak grafa, uključuje prolazak i praćenje tih puteva. Zbog takvog funkcioniranja većina operacija na podatkovnom modelu ovisi o načinu na koji su podaci organizirani u bazi.
Pri radu s graf bazama bitna su dva svojstva [6]:

- Pohrana podataka: neke graf baze podataka su dizajnirane za direktno spremanje podataka, što znači da se podaci modeliraju i optimiziraju za korištenje svojstava grafova. Druge baze pak podatke grafa pohranjuju u relacijske baze, objektno-orijentirane baze ili neku drugu vrstu baze podataka.

- Način obrade podataka: neke graf baze podataka koriste susjedstvo bez indeksiranja što znači da spojeni čvorovi direktno povezani. Ovdje možemo proširiti definiciju graf baze i tako nazvati svaku koja se "ponaša" kao graf, odnosno koristi graf za CRUD operacije i ne koristi indeksiranje.

Iako graf baze podataka imaju široku primjenu i mogu modelirati mnoge situacije, ne mogu potpuno zamijeniti druge vrste podatkovnih platformi. Kako bi se takva zamjena dogodila mora postojati njena praktičnost i korisnost. Na primjer, relacijske baze podataka su izvrsne u rješavanju složenih upita koji uključuju spajanje tablica, korištenje agregacijskih funkcija i transakcija. Ako aplikacija ovisi o takvim upitima, prelazak na NoSQL bazu podataka mogao bi zahtijevati novo projektiranje baze i mogao bi utjecati na performanse. NoSQL baze podataka, pogotovo nove, nemaju istu razinu podrške kao relacijske baze podataka.
Graf baze najviše se koriste zbog njihovih sljedećih prednosti [25]:

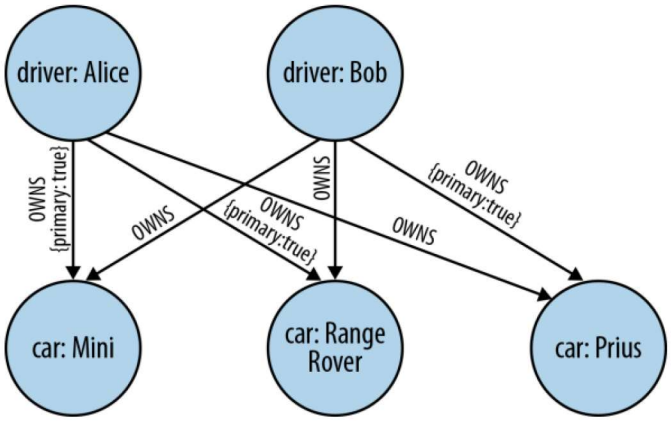
- Izvedba: jedan od razloga za odabiranje graf baze jest sama brzina izvedbe koja dolazi kao rezultat rada s povezanim podacima. U relacijskim bazama brzina upita opada s povećanjem količine spremljenih podataka, dok na graf bazi, čak i nakon dodavanja novih podataka, brzina ostaje relativno konstantna. To je zato što upiti ostaju lokalizirani na jednom dijelu grafa pa je vrijeme izvršavanja svakog upita proporcionalno veličini dijela grafa koji upit mora obraditi, a ne čitavog grafa.

[1] engl. *create, read, update, delete* - stvori, čitaj, ažuriraj, briši

- Fleksibilnost: fleksibilnost grafova omogućuje dodavanje novih čvorova, bridova, ili čak podgrafova bez narušavanja osnovnih svojstava i funkcionalnosti grafa, kao i bez potrebe za promjenom ili prilagodbom upita. Ova karakteristika olakšava upravljanje projektima, posebno onima koji su podložni čestim promjenama. Nadalje, otpornost na promjene i fleksibilna struktura grafa smanjuju potrebu za migracijama podataka, što rezultira manjim troškovima održavanja.
- Agilnost: agilnost u razvoju softvera ključna je za održavanje koraka s promjenjivim okruženjima. Graf baze podataka pružaju potporu takvoj agilnoj praksi omogućavajući kontrolirani razvoj aplikacija. Bez obveze stroge sheme podataka i s podrškom za jezike i API-jeve, graf baze omogućavaju prilagodbu modela podataka kako bi se uskladio s razvojem projekta. To omogućuje softveru da ostane prilagodljiv i konkurentan u dinamičnom okruženju.

Osim pristupa pohrani i procesuiranju podataka graf baza također ima specifični način modeliranja podataka, tj. podatkovni model. Postoji nekoliko različitih modela od kojih su tri najkorištenija: grafovi svojstava, hipergrafovi i trostruki grafovi. [6]

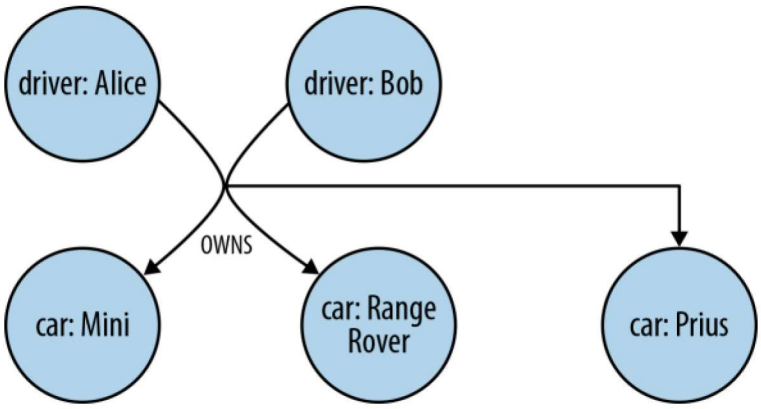
1. **Grafovi svojstava**
   Značajka grafa svojstava je ta da, osim što čvorovi imaju svojstva, svojstva imaju i veze. Svaka veza ima naziv, smjer i svojstva koja su uređena po principu ključ-vrijednost. Svojstva dopunjuju čitavu strukturu rješenja. U grafovima svojstava svaki čvor i brid imaju identifikator. Omogućuju razne primjene poput predviđanja trendova i ponašanje kupaca, prepoznavanje zajedničkih poznanika na društvenim mrežama, identifikaciju klastera podatka i sl.



Slika 3.4: Graf svojstava, Izvor: [6]

2. **Hipergrafovi**
   Hipergraf je generalizirani graf u kojem jedan brid može povezivati više od dva čvora. Takav brid naziva se hiper-brid. Najčešći primjer problema u kojemu se oni koriste su podaci povezani vezom više-prema-više. Na Slici 3.5

možemo vidjeti hiperbrid koji ilustrira da su Alice i Bob vozači triju auta.



Slika 3.5: Hipergraf, Izvor: [6]

Prilikom modeliranja hipergrafa mogu se izostaviti detalji koji opisuju veze ili čvorove. Tako se dobije manji broj bridova u grafu koji je ekvivalentan grafu svojstava kao što je slučaj s grafovima na Slikama 3.4 i 3.5. Možemo zaključiti da svaki hipergraf može biti predstavljen kao graf svojstava.

3. **Okvir za opis podataka** (engl. *Resource Description Framework - RDF*) ili **grafovi trojki**
Trojka je struktura podataka koja opisuje odnos subjekt-predikat-objekt. U kontekstu grafova, trojka se sastoji od dva čvora (subjekt i objekt) povezanih bridom (predikatom) među njima. Može se modelirati i s dodatnim čvorom koji predstavlja predikat, stvarajući skup činjenica poput "Bob vozi auto". Pojedinačno, trojke su semantički oskudne, ali zajedno daju smislenu informaciju. Za razliku od drugih graf baza podataka, grafovi trojki ne podržavaju susjedstvo bez indeksiranja i mogu se dovoljno dobro optimizirati za pohranjivanje svojstava. Iako podržavaju horizontalno skaliranje, nisu tako brzi u upitima i pretraživanju bridova. Zbog tih karakteristika, koriste se kada brzina upita nije ključna, već kada je važna analiza već pohranjenih podataka. Ovaj je model glavna komponenta u W3C semantičkoj web aktivnosti - unaprijeđenoj verziji *world wide weba* u kojoj automatizirani softver može pohranjivati, razmjenjivati i koristiti strojno čitljive podatke distribuirane diljem Weba i tako omogućujući korisnicima da učinkovitije koriste informacije s većom sigurnošću.

3.4 Primjene graf baza

Zbog svoje brzine izvedbe i jednostavnosti operacija, graf baze podataka su popularan izbor u stvarnim problemima s realnim ograničenjima. Brzina upita i responzivnost su primaran razlog odabiranja graf baze. Umjesto složenih *join* operacija, graf baze podataka rade brze obilaske grafa čime se postižu performanse

od nekoliko milisekundi bez obzira na ukupnu veličinu skupa podataka. Njihova dinamička struktura čini ih dobrim izborom za rješavanje problema poput prilagodbe novim zahtjevima i smanjenja vremena razvoja aplikacija. Graf baze podataka se najčešće koriste u idućim situacijama [6]:

- Socijalne veze:
  Društvene aplikacije omogućuju organizacijama konkurentnu prednost iskorištavanjem informacija o vezama među ljudima. Kombiniranjem informacija o pojedincima i njihovim odnosima može se olakšati njihovo povezivanje, poboljšati suradnja i predvidjeti ponašanje.

- Preporuke:
  Preporuke su najbolji primjer generiranja vrijednosti za korisnika primjenom zaključivanja i sugestije identifikacijom ljudi, proizvoda, servisa ili grupa ljudi koji imaju zajedničke interese i njihovih odnosa, odnosno veza među njima. Veze se uspostavljaju na temelju ponašanja korisnika dok kupuju, konzumiraju, ocjenjuju ili na neki drugi sličan način. Kvalitetna preporuka ovisi o razumijevanju veza između elemenata i njihovih svojstava.

- Geografski položaj:
  Geografski problem izvorni je problem na temelju kojega je i nastala teorija grafova. Eulerov problem sedam mostova u Konigsbergu jedan je od poznatijih problema u teoriji grafova i matematici općenito. Ovdje se primjene graf baza ističu u planiranju cestovnih ili željezničkih mreža, služe za traženje ruta do nekoga mjesta, pronalazak središta nekog područja i dr.

- Master data:
  Master data su podaci značajni za rad nekog poduzeća. Obuhvaćaju informacije o korisnicima, kupcima, o odjelima, dobavljačima, troškovnim centrima, lokacijama i sl. U velikim organizacijama ovi se podaci nalaze na različitim mjestima, u raznim formatima, znaju se ponavljati i biti nepročišćeni. Upravljanje ovim podacima može postati izazovno, pogotovo kad se trebaju mijenjati ili kada stižu novi podaci. Graf baze podataka ne nude potpuno rješenje za rad s ovim podacima, ali znatno ga pojednostavljuju. Primjerene su za njihovo pohranjivanje, modeliranje i postavljanje upita koji se odnose na hijerarhiju.

- Autorizacija i kontrola pristupa:
  Rješenja za autorizaciju i kontrolu pristupa pohranjuju mnoge informacije, primjerice o vrsti pristupa, ulogama, organizaciji, pravima pojedinog korisnika i njihovim resursima, mrežnim uređajima, potrebnim datotekama i još mnogo toga. Ovi kompleksni skupovi podataka često su pohranjeni kao rječnici koji sadrže raznolike podatke, a njihovo spajanje može biti dugotrajno i neefikasno. Graf baza podataka efikasno može spremiti takve složene, nagomilane i gusto povezane strukture. Upiti za traženje pristupa i prolazak po grafu s velikim brojem veza mogu se izvršavati u milisekundama, što olakšava brzu i učinkovitu autorizaciju i kontrolu pristupa.

- Otkrivanje prijevara:
  Graf baze podataka su korisne u prepoznavanju prijevara i analizi njihovih povezanosti, bilo da se radi o bankovnim prijevarama, onima u osiguranju ili e-trgovinama. Njihovim korištenjem može se u složenoj mreži odnosa prepoznati sumnjive obrasce, otkriti lažne aktivnosti i veze koje mogu ukazivati na neželjene pojave što ih čini korisnim alatom u kibernetičkoj sigurnosti.

3.5 Neo4j i Cypher

Neo4j je sustav za upravljanje graf bazom podataka koja se naziva Neo4j Graph Database. Prva verzija Neo4j objavljena je u veljači 2010. godine. Implementiran je u Javi i upotrebljava se korištenjem upita napisanih jezikom Cypher za bilo koju CRUD operaciju na bazi. [23]
Cypher je najkorišteniji i intuitivan jezik za rad s graf bazama podataka. Dizajniran je da ga lako čitaju i razumiju programeri, stručnjaci za baze podataka i ostale osobe uključene u poslovne procese. Njegova jednostavnost korištenja proizlazi iz toga što je usklađen s načinom na koji intuitivno opisujemo grafove pomoću dijagrama. Glavne značajke su mu: jednostavnost za učenje, sigurnost i pouzdanost, fleksibilnost i može se lako grafički prikazati[6]. Cypher omogućuje vizualni prikaz povezanih čvorova i bridova graf baze inspiriran ASCII-art tipom sintakse. Na primjer, u izrazu (čvor)-[:JE_POVEZAN_S]→(drugim čvorom) elementi u okruglim zagradama označuju čvorove, a u uglatim bridove između čvorova. Ako čvor neće biti korišten kasnije u upitima može se označiti i praznim zagradama.
Slični čvorovi mogu se označavati istim oznakama (engl. *label*) kako bi se mogli grupirati. Tako možemo specificirati posebnu vrstu entiteta, kao što u relacijskim bazama podataka tablice imaju svoje nazive. Oznake pomažu Cypheru da razlikuje entitete i ubrza izvršavanje upita. Bridovi se označavaju strelicama, a ako su podaci pohranjeni u jednom smjeru, Cypher će vratiti rezultate samo ako je upit u skladu s tim smjerom. U suprotnom, bolje je koristiti neusmjereni brid kako bi dobili podatke nevezane sa smjerom brida, primjerice: (o:Osoba)-[:POSJEDUJE]-(s:Stvar). Varijable bridova označuju se uglatim zagradama, npr. možemo označiti s [r], [rel] ili [rel:JE _ POVEZANA]. Ako nije navedena dvotočka ispred veze, nije uvedena varijabla koja predstavlja određenu vezu, već Cypher pretražuje sve vrste veza.
Posljednji dio podatkovna modela čine svojstva - parovi ime-vrijednost koji detaljnije opisuju čvorove i veze. Svojstva mogu imati vrijednosti različitih tipova podataka. Svojstva se pišu unutar vitičastih zagrada unutar zagrade za čvor ili vezu. Primjer navođenja svojstva unutar čvora bi bio (o:Osoba {ime: 'Sally'}). Na taj način pretražuju se svi čvorovi s oznakom Osoba kojima je ime Sally. Primjer unutar veze: -[rel:JE_ PRIJATELJICA_ SA {od: 2018}]. Tako se sve veze filtriraju po svojstvu od.
Kombiniranjem sintakse za pisanje čvorova i bridova među njima nastaju uzorci, npr. (o:Osoba {ime: "Sally"})-[relacija:VOLI]→(t:Tehnologija {vrsta: "grafovi"}). Ovaj uzorak će pretražiti sve čvorove s oznakama Osoba i Tehnologija, imena Sally i vrste grafovi, koji su povezani bridom VOLI. Koristeći tako nastale uzorke dolazimo do semantički bogatih modela.

3.5.1 Sintaksa Cyphera

Kao i većina upitnih jezika, Cypher se sastoji od klauzula. Osnovne naredbe jezika Cypher su:

- MATCH:
  Najjednostavniji upiti sastoje se samo od klauzule MATCH iza koje mora slijediti RETURN. Omogućuje kreiranje uzoraka koje će Neo4j pretraživati u bazi podataka. [21]

- RETURN:
  specificira koji čvorovi, bridovi i svojstva trebaju biti vraćeni klijentu. [6]

  Primjer 5. *Idući upit pretražuje čvor s oznakom Osoba koji ima svojstvo s vrijednošću Ana. Vrijednosti koje dobivamo ovim upitom vezane su uz identifikator **a**, on omogućuje referenciranje na čvor koji predstavlja Anu kroz čitav upit. To je početni čvor od kojega kreće pretraživanje puta koji se sastoji od tri čvora, **a**, **b** i **c**.*

  MATCH (a:Osoba {ime: 'Ana'})-[:POZNAJE] →(b)-[:POZNAJE] →(c)
  RETURN b, c

- WHERE:
  služi za filtriranje rezultata pretrage.

  Primjer 6. *U ovom primjeru vidimo upit koji vraća sve osobe koji poznaju osobu čije ime je Lucija.*
  MATCH (a:Osoba)-[:POZNAJE] →(b)-[:POZNAJE] →(c)
  WHERE b.ime = 'Lucija'
  RETURN b, c

- CREATE, CREATE UNIQUE:
  služe za stvaranje čvorova, indeksa, ograničenja, veza i dr.

  Primjer 7. *Primjer stvaranja čvorova, veza i ograničenja:*
  CREATE (n1:Osoba name: 'Ana', age: 24)-[:POZNAJE] →(n2:Osoba name: 'Lucija').

  MATCH (a:Osoba), (b:Osoba)
  WHERE a.name = 'Ana' AND b.name = 'Lucija' CREATE (a)-[:POZNAJE]→(b).

  CREATE CONSTRAINT ON (o:Osoba) ASSERT o.email IS UNIQUE

- MERGE:
kombinira funkcionalnosti od MATCH i CREATE. Pokušava pronaći uzorak u bazi i ako ne uspije, napravi ga, a ako postoji osigurava da se ne pojavljuje duplo kao što bi se to dogodilo korištenjem samo CREATE. Može ažurirati podatke ili ih ostaviti nepromijenjenima.

- DELETE:
briše čvorove, veze i svojstva iz baze.

- SET:
postavlja svojstva nekoga čvora.

  Primjer 8. *Primjer postavljanja dobi čvoru Osoba koji ima ime Ana.*
  MATCH (o:Osoba ime: 'Ana')
  SET o.dob = 24

- REMOVE:
je suprotna operacija od SET.

- UNION:
spaja rezultate više upita.

- WITH:
služi za ulančavanje dijelova upita. Rezultate jednoga upita prosljeđuje drugome.

- FOREACH:
prolazi svakim čvorom koji zadovoljava neki uvjet i ažurira.

- CASE:
koristi se za grananje i sintaksa kao u primjeru 9.

  Primjer 9. *Grananje*
  CASE n.eyes
  WHEN 'blue' THEN 1
  WHEN 'brown' THEN 2
  ELSE 3
  END

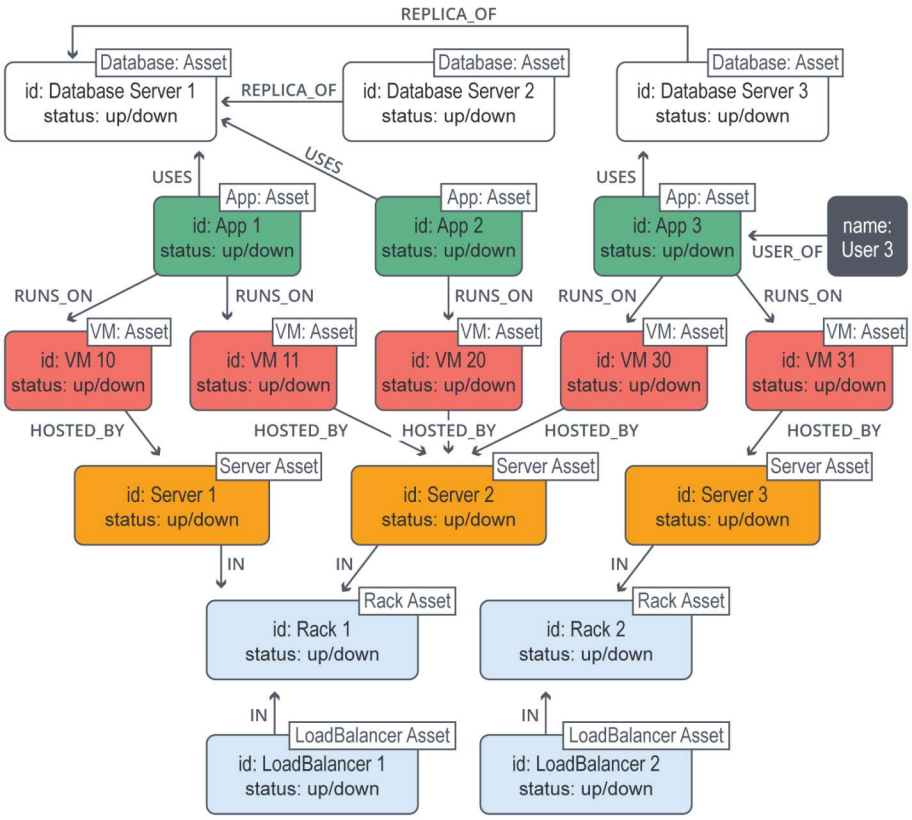
- UNWIND:
pretvara listu u niz redaka.

Broj čvorova koje će graf proći na putu označava se vitičastim zagradama na kraju upita:
((m:Person)-[:KNOWS] $\rightarrow$(n:Person) WHERE m.born $<$ n.born){1,5}.
Od operatora postoje operatori dohvaćanja: . i [] za dohvaćanje vrijednosti iz svojstava, osnovni matematički operatori, operatori za uspoređivanje, logički operatori AND, OR, XOR, NOT. Za konkatenaciju listi koristi se operator +, za regularni izraz $\sim=$ i STARTS WITH, ENDS WITH, CONTAINS za slaganje stringova. Postoji i operator DISTINCT koji vraća jedinstvene podatke

Mogu se koristiti razne ugrađene funkcije kao što su: avg, sum, max, min, count, stDev, ceil, floor, rand, isNan, round, sign, time, datetime, i mnoge druge za rad sa stringovima, listama, rječnicima, vremenom te brojevima koje se detaljno mogu pogledati na [14]. Sintaksa Cyphera može se proširiti dodavanjem raznih paketa i učitavanjem njihovih funkcija.

3.5.2 Modeliranje graf baze

Proces modeliranja graf baze može se opisati kao pokušaj stvaranja strukture grafa koji najbolje odgovara rješenju problema s kojim se suočavamo.
Najprije treba jasno definirati ciljeve projekta i korisničke zahtjeve kako bi se razumjelo što je sve potrebno za ispunjenje zadatka. Dodatno, potrebno je uskladiti zahtjeve korisnika s programerima i njihovim mogućnostima, dostupnim alatima i resursima. Nadalje, treba preoblikovati zahtjeve u pitanja na koja graf treba dati odgovor. Kada je to utvrđeno, detaljnije se razrađuju dijelovi graf baze koja će se koristiti uključujući čvorove, njihove oznake, svojstva i bridove među njima. Svaki čvor treba imati jedinstveni identifikator kako bi se izbjeglo dupliciranje podataka. Svojstva se mogu naknadno dodavati tako da nije potrebno planirati svaku moguću situaciju. Kada je graf pripremljen, pitanja izražavamo u obliku uzoraka koje koristimo u upitima. Primjer modela graf baze dan je na Slici 3.6.



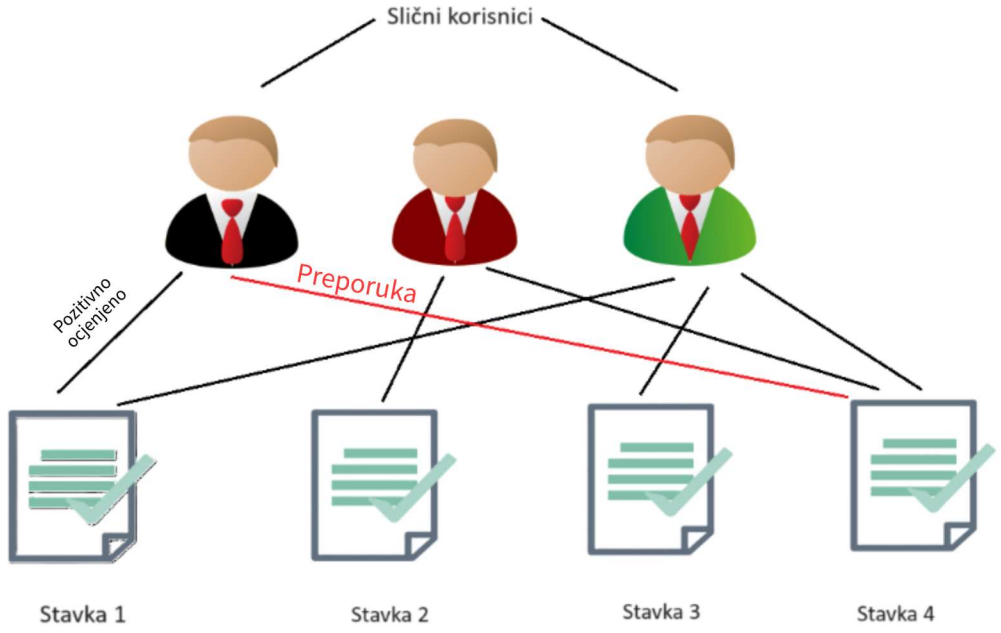
Slika 3.6: Graf model, Izvor: [6]

Općenito, entitete ne treba spremati u obliku veza, već veze treba koristiti za prenošenje informacija o tome kako su entiteti povezani i kakva je kvaliteta tih odnosa. Entiteti nisu odmah jasno vidljivi dok se slažu zahtjevi tako da treba dobro proučiti koje će se oznake za čvorove koristiti. Treba imati na umu da iako se graf može jednostavno širiti i povećavati s velikom količinom podataka, loše ga je mijenjati dok se koristi jer može krenuti rasti u neželjenom smjeru koji ne odgovara traženim zahtjevima.
Modeliranje graf baze bit će napravljeno na primjeru sustava preporuke napravljenom u sklopu praktičnog djela ovog rada.

4 | Sustav preporuka

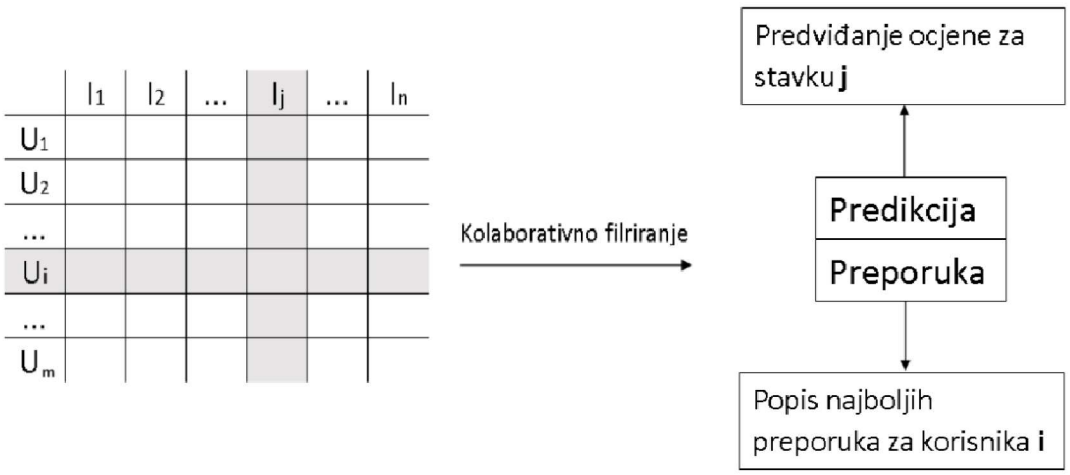
Zbog mnoštva podataka, pogotovo prisutnih na internetu, postoji potreba za filtriranjem, prioritiziranjem i učinkovitom pretragom istih. Sustav preporuka čine algoritmi koji koriste veliku količinu dostupnih podataka za uspješno personaliziranje i pronalazak proizvoda i usluga korisnicima koje inače ne bi jednostavno sami pronašli. Možemo reći da sustavi preporuka funkcioniraju kao sustavi filtriranja informacija koristeći korisničke preferencije i ponašanje kako bi pružili korisne sugestije. Također, ovi sustavi mogu predvidjeti korisničke preferencije na temelju korisničkog profila, čime unapređuju proces donošenja odluka. Sustave preporuka najčešće koriste pružatelji internetskog sadržaja i prodavači kako bi poboljšali korisničko iskustvo i povećali zadovoljstvo korisnika.
Ovisno o potrebama, ograničenjima i zahtjevima bitno je odabrati odgovarajući algoritam za preporuke. Važno je koristiti pouzdane i točne pristupe za rad sa sustavom preporuka kako bi se korisnicima pružile korisne i relevantne preporuke [9]. Prema [8] možemo podijeliti sustave preporuka na dvije osnovne vrste, odnosno dva različita načina na koja se oni mogu ostvariti:

1. **Kolaborativno filtriranje**
   Kolaborativno je filtriranje najpopularnija i najčešće korištena tehnika za preporuke. Usredotočuje se na sličnosti između različitih korisnika i stavki. Korisnici koji dijele neke zajedničke interese vrlo vjerojatno će imati slične preferencije. Ove sličnosti poboljšavaju preporuke svim korisnicima unutar skupa podataka, a algoritam nastavlja učiti s dolaskom novih proizvoda na tržište. Na primjer, ako Marko i Petar vole nogomet i Petar kupi određene tenisice, tada će postojati određena vjerojatnost da i Marko kupi te tenisice. Ova tehnika ima dvostruko djelovanje. Prvo, pomaže identificirati grupu sličnih korisnika čija se mišljenja koriste kao temelj za preporuke (filtriranje korisnika prema korisnicima). Drugo, koristi ta mišljenja kako bi stvorila širu sliku o korisniku, omogućujući bolje preporuke (filtriranje korisnika prema proizvodima). Na primjer, preporučuje proizvode i usluge temeljene na onome što je odabrao korisnik sličnih interesa. Takav algoritam zahtijeva značajne računalne resurse jer provodi usporedbe korisnika u stvarnom vremenu. Filtriranje korisnika prema proizvodima je najjednostavnija metoda filtriranja. U njoj algoritam traži slične proizvode koje je potrošač prethodno kupio ili su mu se ranije svidjeli.



Slika 4.1: Princip kolaborativnog filtriranja (prema:[9])

Tehnike za primjenu kolaborativnog filtriranja zahtijevaju obradu vrlo velikih skupova podataka i mogu se primijeniti u različitim područjima, poput financija, vremenske prognoze, e-trgovine i sl. Ove tehnike koriste podatke o stavkama koje su se već bile svidjele korisniku kako bi predvidjele druge stavke koje bi mogle zanimati korisnika. Koristi se model koji se predstavlja kao matrica preferencija/ocjena reda m × n. Označimo matricu s $A = [a_{ij}]$ gdje je m broj korisnika $(U_1, U_2, ..., U_m)$ i n broj stavki $(I_1, I_2, ..., I_n)$ koje je korisnik ocijenio kao na Slici 4.2



Slika 4.2: Korisnik-produkt matrica (matrica ocjena) (prema: [8])

Vrijednost matrice u i-tom retku i j-tom stupcu a_{ij} jednaka je ocjeni koju je i-ti korisnik dao j-tom proizvodu. Ove ocjene mogu biti implicitne (kupnja proizvoda) ili eksplicitne (povratna informacija korisnika). Korištenje matrice ima dva bitna ishoda, predviđanje vrijednosti a_{ij} i popis najboljih preporuka za korisnika.[9]
Metode za kolaborativno filtriranje možemo podijeliti na one bazirane na memoriji koje koriste odnose među korisnicima i ne zahtijevaju postojanje modela za algoritam i bazirane na modelu koje koriste matricu za ocjene podatka te kombinaciju te dvije, tzv. hibridnu metodu.
Primarna funkcija preporuke je predvidjeti korisnost artikla za korisnika. Općenito, za nekog korisnika u kojega zanima proizvod i sustav preporuka može označiti s $r(u,i)$ stupanj preferencije ili ocjene.
Predviđena preferencija aktivnog korisnika a prema proizvodu j računa se na način:

$$P_{a,j} = \overline{r_a} + k \sum_{u=1}^{m} S_{(a,u)} \times (r_{uj} - \overline{r_u}),$$

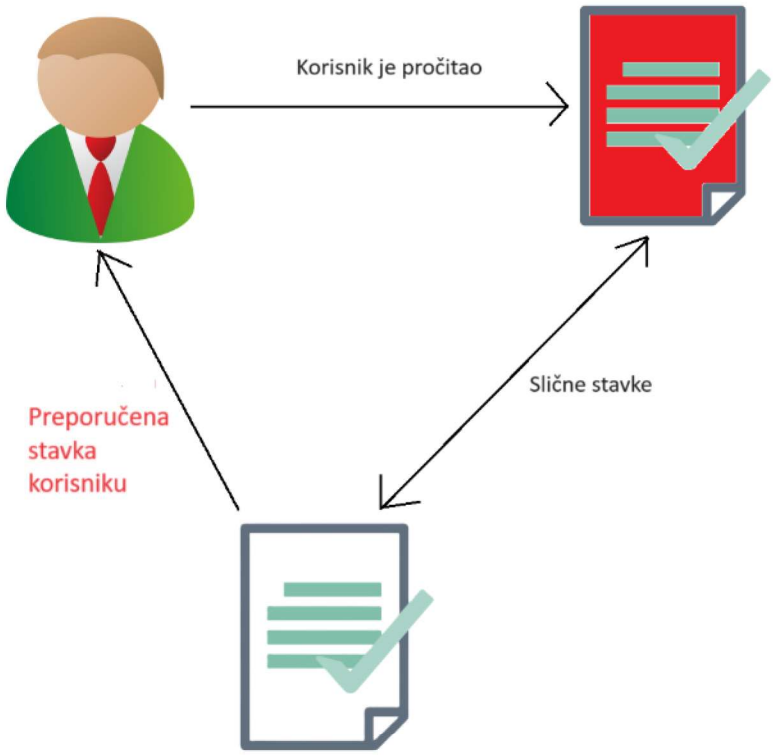
gdje je r_a srednja ocjena koju korisnik a daje, n je broj korisnika u bazi koji su dali nenegativnu ocjenu j-tom proizvodu - $r_{i,j}$ i $S_{(a,i)}$ sličnost između korisnika a i svakog drugog, zatim preostaje k: faktor normalizacije.
Za izračunavanje sličnosti između korisnika postoje mnogi načini. Dva osnovna bi bila računanje Pearsonov koeficijent korelacije za računanje korelacije između dva vektora u i v s vrijednostima u segmentu $[-1,1]$:

$$S_{Pearson}(u,v) = \frac{\sum_{I=1}^{n}(r_{u,i} - \overline{r_u}) \times (r_{v,i} - \overline{r_v})}{\sqrt{\sum_{I=1}^{n}(r_{u,i} - \overline{r_u})^2 \times (r_{v,i} - \overline{r_v})^2}},$$

i kosinus sličnosti:

$$S_{Cosine(u,v)} \frac{\sum_{I=1}^{n}(r_{u,i}) \times (r_{v,i})}{\sqrt{\sum_{I=1}^{n}(r_{u,i})^2 \times (r_{v,i})^2}}.$$

2. **Filtriranje temeljeno na sadržaju**
Preporuke temeljene na sadržaju koriste podatke koje sam korisnik daje izravno ili neizravno. Analizira se sadržaj koje je korisnik ocijenio i zatim se stvaraju zaključci o korisniku. Nakon toga, uspoređuju se proizvodi koji se mogu preporučiti korisniku s proizvodima za koje je već poznato da se sviđaju korisniku te se na temelju sličnosti korisniku prikazuje najrelevantniji proizvod. Ovaj pristup je jednostavan za korištenje u slučajevima kada su informacije o proizvodima dobro opisane, organizirane i dostupne. Prednost ove tehnike je mogućnost prilagodbe preporuka čak i kada se korisnički profil mijenja u vrlo kratkom razdoblju. Glavni nedostatak je potreba za detaljnim opisom proizvoda i njegovim metapodacima. Isto tako, može se dogoditi i suprotno: informacije o novim proizvodima gotovo su jednake postojećima u korisničkom profilu, što rezultira preporukom proizvoda koje korisnik već posjeduje ili je vidio.



Slika 4.3: Kako preporuke temeljene na sadržaju fukcioniraju (prema: [13])

4.1 Sustav preporuka i Neo4j

Graf baze podataka nadmašuju relacijske i druge NoSQL baze podataka kada je potrebno dobiti uvid u potrebe kupaca i trendove proizvoda. To je posebno vidljivo kod povezivanja mnoštva podataka o kupcima i proizvodima te drugim povezanim podacima. One su temeljna tehnološka platforma za velike tvrtke koje rade s korisničkim podacima, poput Googlea, Amazona, Facebooka i LinkedIna. Sustavi za preporuke u stvarnom vremenu zahtijevaju sposobnost praćenja i razumijevanja korisnikovih prethodnih radnji, npr. kupčevih prethodnih kupnji, brzo pretraživanje tih podataka i povezivanje sličnih. Također, potrebno je i bilježiti trenutačno stanje podataka i prikupljati informacije o novim interesima korisnika tijekom trenutne korisničke sesije. Takve radnje su vrlo jednostavne za popratiti korištenjem Neo4j sustava. Neo4j omogućuje korisnicima lakše donošenje odluka u stvarnom vremenu iskorištavajući veze između podataka. Na primjer, eBay je naveo da je zamjenom relacijskih baza s Neo4j veličina koda smanjena od 10 do 100 puta.[22]
Sustavi preporuka često se oslanjaju na algoritme temeljene na grafovima za analizu i iskorištavanje veza između korisnika, stavki i njihovih interakcija. Neo4j-ov model podataka grafova prirodno podržava ove algoritme što ga čini dobrom platformom za izgradnju sustava preporuka. Modeli preporuka temeljeni na grafovima mogu učinkovito modelirati interakcije između korisnika i proizvoda, slič-

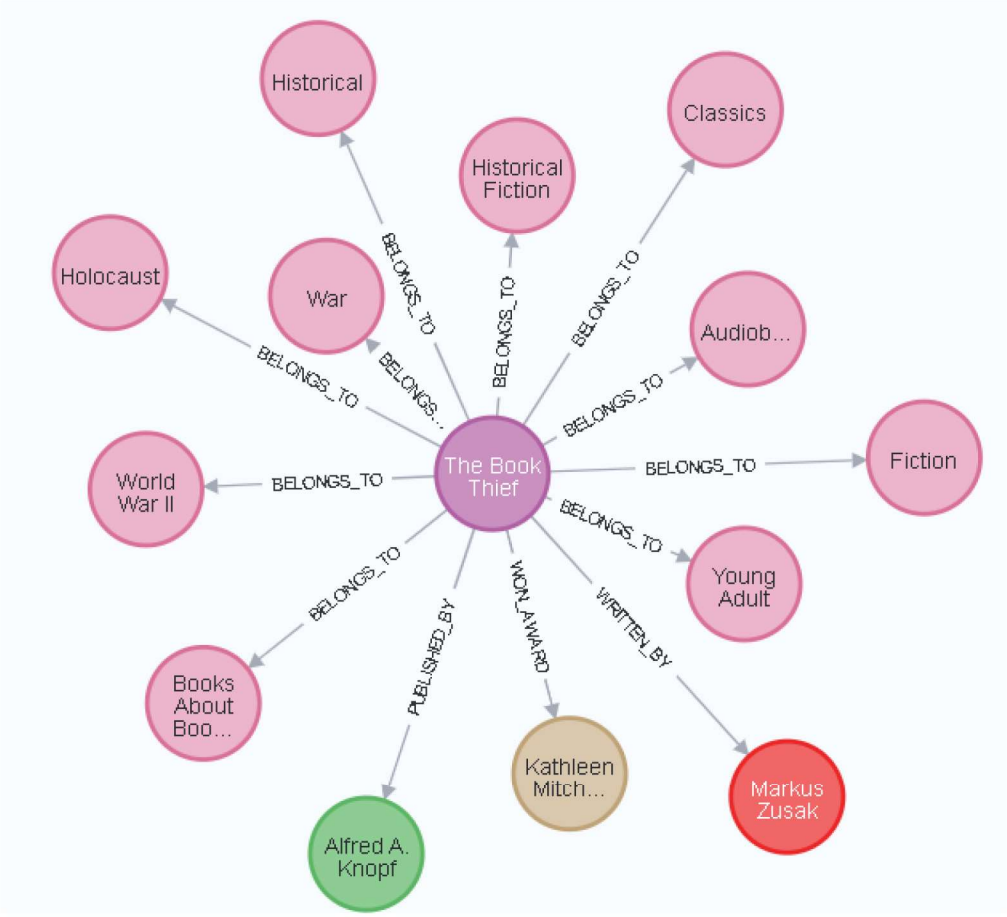
nosti između korisnika te odnosa proizvoda i proizvoda za generiranje relevantnih preporuka.
Kada bismo implementirali kolaborativno filtriranje koristili bismo različite čvorove s oznakama korisnik i čvorove s oznakama koje predstavljaju imena proizvoda. Te čvorove možemo još dodatno grupirati, tj. označiti. Na primjer, mogu se dodati veze između korisnika koje označavaju međusobne odnose, (npr. poznaju se) i veze između korisnika i proizvoda (npr. ocjena, kupnja, pretraživanje). Pomoću već ugrađenih algoritama koje Neo4j ima, možemo izračunati koeficijent korelacije koji nas zanima (npr. koristeći Pearsonov koeficijent korelacije) između korisnika i proizvoda.
Neo4j postiže filtriranje temeljeno na sadržaju spremanjem i prikazivanjem atributa i svojstava nekoga proizvoda u obliku čvorova omogućujući personalizirane preporuke temeljene na karakteristikama proizvoda. Neka svojstva i atributi proizvoda također mogu biti čvorovi što pojednostavljuje model i omogućuje brži prolazak kroz graf te ubrzava traženje veza i korelacija između svojstava. Isto tako korisnikove prethodne interakcije ili već znane preferencije mogu se spremiti kao podgraf koji je povezan s korisnikom. Preporuke se mogu generirati postavljanjem upita za proizvode koji dijele slične atribute ili karakteristike s preferiranim stavkama korisnika.

5 | Implementacija web aplikacije za preporuke korištenjem Neo4J

U nastavku možemo vidjeti primjer web aplikacije koja služi za preporuku knjiga korisniku na temelju sličnosti knjiga. Korisnik može pretraživati i odabirati knjige, a aplikacija će koristeći filtriranje temeljeno na sadržaju predlagati druge knjige koje bi se mogle svidjeti korisniku. Isto tako aplikacija može pružiti detaljan opis odabrane knjige koristeći informacije iz baze podataka. Baza podataka i upiti su rađeni korištenjem Neo4j Drivera, paketa unutar .NET-a, i Cyphera. Za korisničko sučelje korišten je Angular.
Za kreiranje baze korišten je Neo4j Desktop, lokalno okruženje za rad s Neo4j. U Neo4j Desktopu se mogu praviti lokalne baze, upravljati sustavom za upravljanje bazama, eksperimentirati s Neo4j, izrađivati vlastiti projekti i povezivati Neo4j s drugim aplikacijama.

5.1 Model podataka

Podaci o knjigama su dostupni na [18], a u .csv obliku preuzeti su s [20].
Proces modeliranja se može opisati kao pokušaj stvaranja grafa koji će najbolje odgovoriti na pitanja na koja aplikacija treba dati odgovor korisniku. Za početak to su pitanja: koje su knjige najpopularnije, koje su najbolje ocijenjene, za određenu knjigu koja bi bila slična po žanru, koju knjigu je napisao isti autor i ako korisnik želi pretraživati knjigu po njenim značajkama koji bi bio način za njihovo spremanje. Iz podataka vidimo da neka knjiga može imati više žanrova, ali i nekom žanru može pripadati više knjiga. Isto je i s autorima, nagradama i nakladnicima. Upravo zbog toga, svaki od tih podataka je čvor u modelu. Također, svaki od njih povezan je sa čvorom knjiga. Na Slici 5.1 možemo vidjeti primjer knjige, odnosno čvora s oznakom Book koji ima jedno od svojstava, naslov, "The Book Thief", kako je povezana vezom BELONGS_TO sa čvorovima oznake žanr, PUBLISHED_BY s čvorom Publisher, WRITEN_BY Author i vezom WON_AWARD s čvorom Award.



Slika 5.1: Primjer kreiranih čvorova i bridova

Čvorove smo kreirali naredbom prikazanom na Slici 5.2 u Neo4j Browser-u koji služi za pisanje Cyphera u Neo4j Destopu:

```
LOAD CSV WITH HEADERS FROM 'file:///Best_Books_Ever.csv' AS row
MERGE (b:Book {bookId: row.bookId})
ON CREATE SET
  b.title = row.title,
  b.series = row.series,
  b.rating = toFloat(row.rating),
  b.description = row.description,
  b.language = row.language,
  b.isbn = row.isbn,
  b.characters = row.characters,
  b.bookFormat = row.bookFormat,
  b.edition = row.edition,
  b.pages = toInteger(row.pages),
  b.publishDate = row.publishDate,
  b.firstPublishDate = row.firstPublishDate,
  b.numRatings = row.numRatings,
  b.ratingsByStars = row.ratingByStars,
  b.likedPercent = row.likedPercent,
  b.setting = row.setting,
  b.coverImg = row.coverImg,
  b.bbeScore = toInteger(row.bbeScore),
  b.bbeVotes = toInteger(row.bbeVotes),
  b.price = toFloat(row.price);
```

Slika 5.2: Cypher naredba za stvaranje čvora knjige

Korištenjem naredbe LOAD CSV WITH HEADERS parsira se csv datoteka s podacima i čita se kao skup redaka. Naredba MERGE pravi čvor, ako on ne postoji, tako što retke prepiše kao oznake čvora. Na takav način kreirani su svi preostali čvorovi u bazi.
Zbog sigurnosti da se ne stvaraju duplikati čvorova, prije nego što se pokrenulo njihovo stvaranje, postavljena su ograničenja na njihovu jedinstvenost što je dano Slikom 5.3.

```
CREATE CONSTRAINT bookIdConstraint FOR (book:Book) REQUIRE book.bookId IS UNIQUE;
```

Slika 5.3: Cypher naredba za stvaranje ograničenja

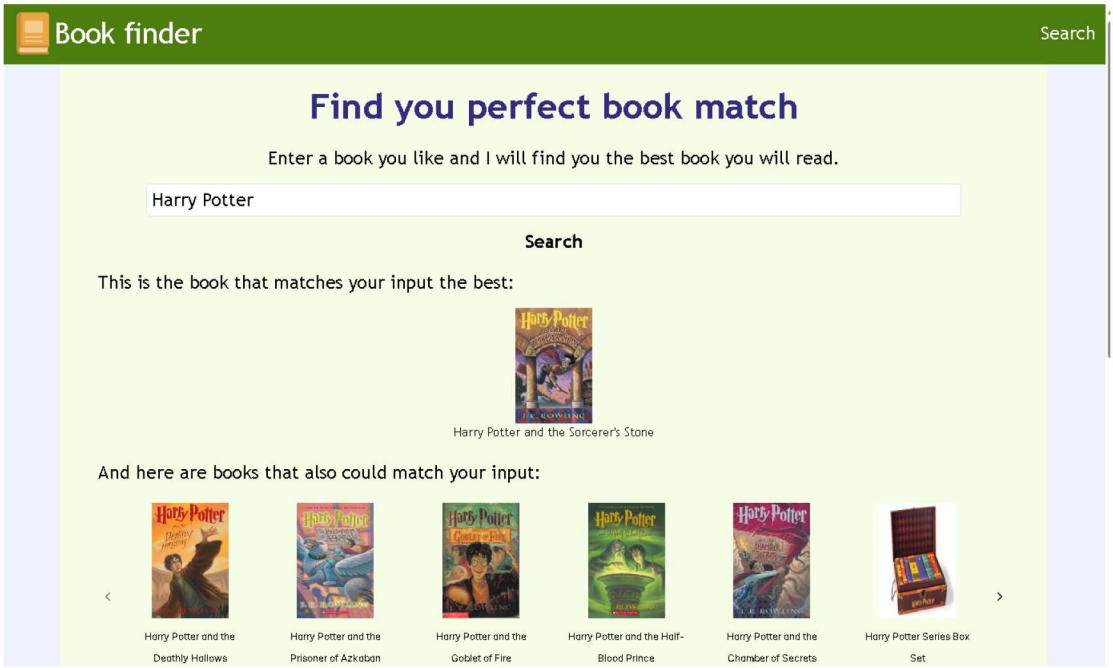
Sada kada imamo sve čvorove, potrebno ih je spojiti. Prilikom kreiranja koristit ćemo MERGE naredbu. Ovdje je razlika jedina što u podacima dobijemo listu podataka koju moramo razdvojiti na više čvorova.

```
LOAD CSV WITH HEADERS FROM 'file:///Best_Books_Ever.csv' AS row
WITH row.bookId AS bookId, row.awards AS awards
MATCH (b:Book {bookId: bookId})
WITH b, apoc.convert.fromJsonList(awards) AS bookAwards
UNWIND bookAwards AS award
MATCH (g:Award { name: award })
MERGE (b)-[:WON_AWARD]→(g);
```

Slika 5.4: Cypher naredba za stvaranje veza

5.2 Funkcionalnosti i preporuke

Pokretanje aplikacije vodi nas na početnu stranicu. Na njoj možemo unijeti naslov knjige koja nas zanima i pojavit će se knjiga koja najbolje odgovara danom unosu i ispod nje niz (karusel) knjiga koje također odgovaraju unosu, ali su manje slične, npr. imaju neku riječ više. Na Slici 5.5 možemo vidjeti kako to izgleda.



Slika 5.5: Početna stranica

Osim polja za unos, imamo po tri predložena niza knjiga. Možemo vidjeti knjige koje su najbolje ocijenjene, koje su najnagrađivanije, i one koje su se najviše svidjele čitateljima. Na njih možemo gledati kao knjige u grafu koje imaju najveću ocjenu - sortirane po svojstvu rating, po svojstvu likedPrecent i knjige koje imaju najviše veza na nagrade.
Za dohvat knjiga sortirane po ocjeni korišten je jednostavan upit koji se sastoji od MATCH klauzule koji se vidi na Slici 5.6. Ovdje se filtriraju knjige iz skupa podataka koje nemaju sliku kako se ne bi takve prikazivale na stranici. Na isti način

napisan je upit za knjige sortirane po likedPrecent.

```
MATCH (book:Book) WHERE book.coverImg <> "" AND book.coverImg <> ""
RETURN book ORDER BY book.rating DESC LIMIT 15
```

Slika 5.6: Cypher naredba za dohvat 15 najbolje ocijenjenih knjiga

Upit koji vraća sortirane knjige po veličini broja nagrada koristi klauzuli COUNT i prebrojava veze čvora jedne knjige na čvorove nagrada dan je na Slici 5.7:

```
MATCH (book:Book)-[:WON_AWARD]→(award:Award) WITH book,
COUNT(award) as awardCount
ORDER BY awardCount DESC
LIMIT 15 RETURN book
```

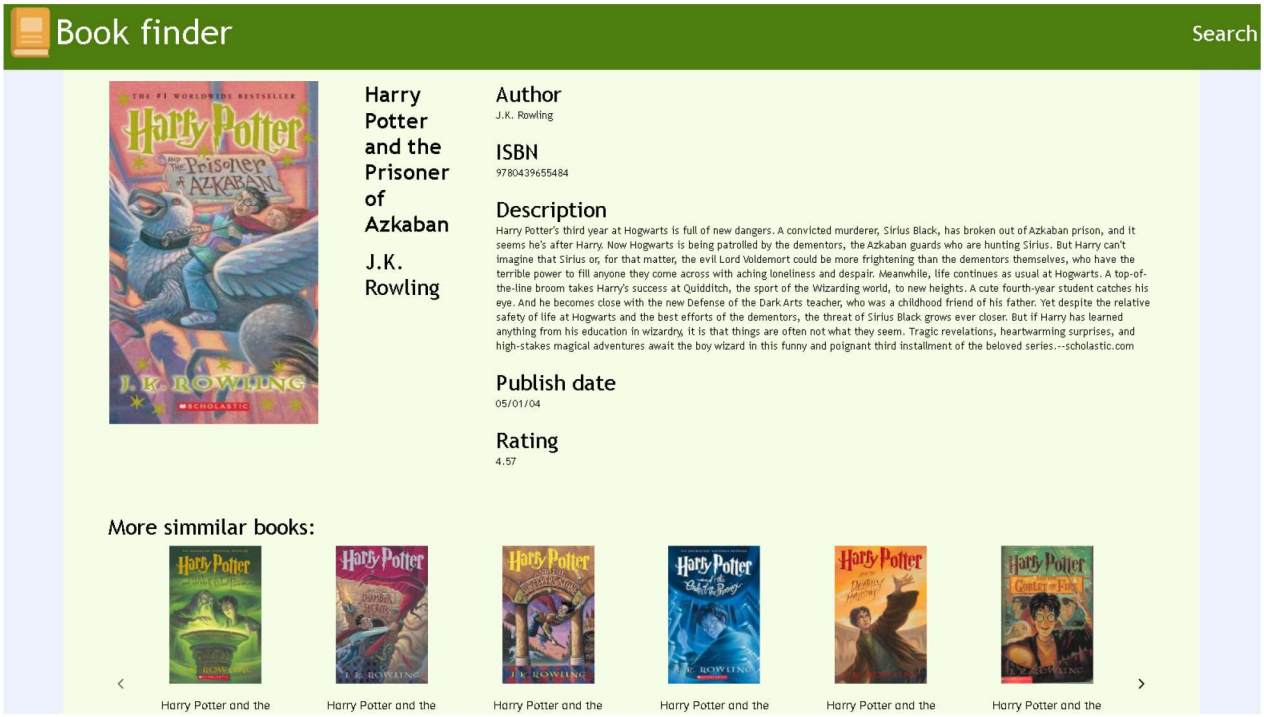
Slika 5.7: Cypher naredba za dohvat 15 knjiga s najviše nagrada

Prilikom klika na neku od knjiga odlazimo na stranicu koja prikazuje detalje o knjizi. Tu se zapravo iščitaju svojstva čvora koji ima identifikator poslan s backenda (.NET-a). Na ovoj stranici možemo vidjeti primjer filtriranja temeljenog na sadržaju. Ispod detalja knjige nalaze se primjeri knjiga koji su najsličniji odabranoj knjizi. Ovdje je korišten Cypher upit koji gleda koje knjige imaju istog autora kao i odabrana knjiga, i kako svaka knjiga može pripadati više žanrova, najveći broj zajedničkih žanrova. Upit vraća 10 takvih knjiga. U slučaju da nema svih 10 knjiga istog autora, onda nakon što je naveo knjige sa zajedničkim autorom sortira samo po broju zajedničkih žanrova. Korišteni upit dan je na Slici 5.8

```
// this part saves the data about genres of provided book
MATCH (providedBook:Book {bookId: '19063.The_Book_Thief'})-[:BELONGS_TO]→(genre:Genre)
WITH providedBook, COLLECT(DISTINCT genre) AS providedGenres

// this line is for introducing the author variable
MATCH (providedBook)-[:WRITTEN_BY]→(author:Author)

// all the books of the same author as the provided book
MATCH (otherBookByGenre:Book)-[:BELONGS_TO]→(commonGenre:Genre)
WHERE otherBookByGenre <> providedBook
AND (otherBookByGenre)-[:WRITTEN_BY]→(author)


WITH providedBook, otherBookByGenre, author,
 COLLECT(DISTINCT commonGenre) AS commonGenres, providedGenres

// Calculate and order by common genres count for the first 15 books -
// books that have same genres and author ordered by the number of genres
WITH providedBook, providedGenres, otherBookByGenre, author, commonGenres,
SIZE([genre IN commonGenres WHERE genre IN providedGenres]) AS commonGenresCount
ORDER BY commonGenresCount DESC

WITH providedBook, providedGenres,
COLLECT({book: otherBookByGenre, author: author, commonGenresCount: commonGenresCount})
AS firstPart


// Get the remaining 10 books sorted by the number of genres
MATCH (otherBookByGenreRemain:Book)-[:BELONGS_TO]→(commonGenreRemain:Genre)
WHERE otherBookByGenreRemain <> providedBook
AND NOT otherBookByGenreRemain IN firstPart
WITH providedBook, otherBookByGenreRemain,providedGenres,
COLLECT(DISTINCT commonGenreRemain) AS commonGenresRemain, firstPart

// Calculate and order by common genres count for the remaining books
WITH providedBook, otherBookByGenreRemain, commonGenresRemain,
SIZE([genre IN commonGenresRemain WHERE genre IN providedGenres])
AS commonGenresCountRemain, firstPart, providedGenres
ORDER BY commonGenresCountRemain DESC limit 10


  // Return the first books and the remaining 10
  unwind firstPart as fp
  return collect(fp.book) + collect(otherBookByGenreRemain)
```

Slika 5.8: Cypher upit za najsličnije knjige

Osim toga možemo vidjeti knjige poslagane po broju zajedničkih žanrova i knjge koje je napisao isti autor.

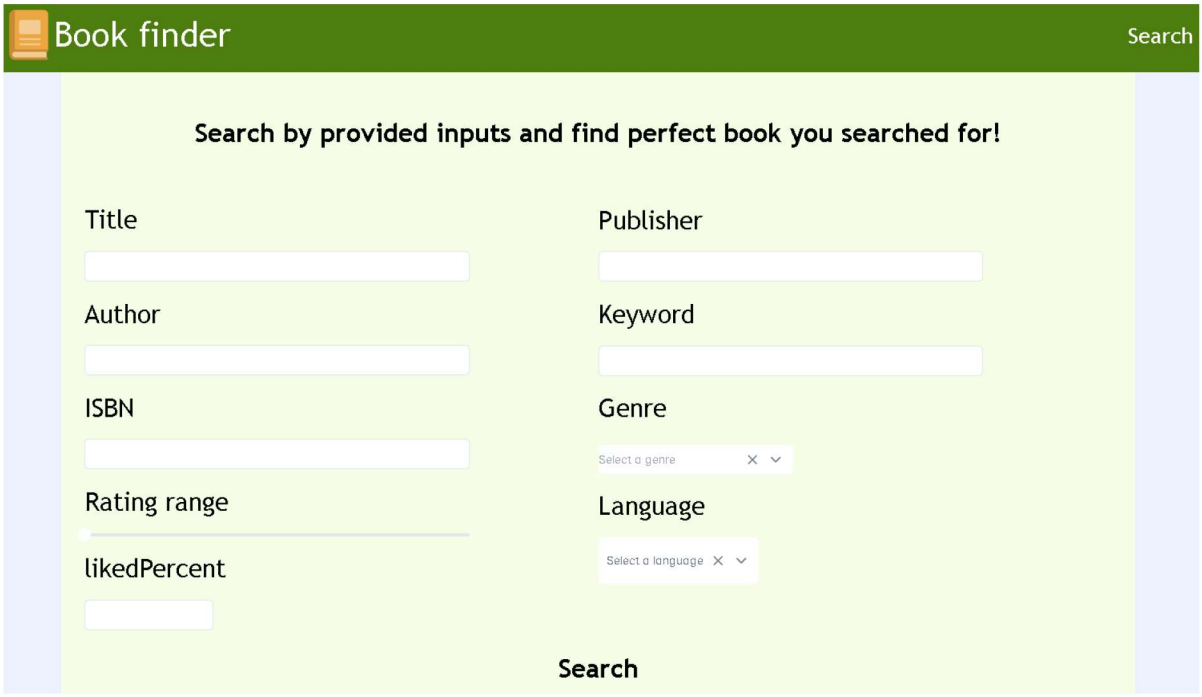


Slika 5.9: Primjerak stranice detalja knjige

Zadnje preostaje stranica za pretragu koju možemo vidjeti na Slici 5.11. Korisnik može unijeti bilo koji od ponuđenih atributa. Ovdje, osim upita za pretragu imamo dodatna dva, upit koji vraća sve jezike kojima su knjige pisane pri čemu je svaki jezik naveden samo jednom dan na Slici 5.10 i u drugi koji vraća sve jedinstvene žanrove iz baze napisan na isti način. Njihove rezultate možemo vidjeti u dva padajuća izbornika. To bi bio primjer dohvaćanja jednog svojstva umjesto čitavog čvora.

```
MATCH (book:Book) WHERE book.language IS NOT NULL AND trim(book.language) <> "" RETURN distinct book.language AS language
```

Slika 5.10: Cypher upit koji vraća sve jezike



Slika 5.11: Primjerak stranice za pretragu

Rezultat upita vraća knjigu i odvodi nas na zasebnu stranicu koja na sebi prikazuje upravo tu knjigu i ima listu preostalih knjiga koje najbolje odgovaraju korisnikovoj pretrazi. Ovaj upit je napisan koristeći backend u C#. Njegova struktura ovisi o tome koji sve unosi su odabrani. Na primjer ako korisnik ne upiše kojeg autora knjige želi, neće se generirati dio Cypher koda koji pretražuje knjige po autorima. Na Slici 5.12 možemo vidjeti primjer generiranog Cypher koda nakon unosa željenih atributa za pretraživanje:

```
MATCH (book : Book) -[:WRITTEN_BY]→(a:Author{name:'Shannon Messenger'})
MATCH(book)-[:BELONGS_TO]→(g: Genre{ title: 'Fiction'})
WHERE book.coverImg <> ""
AND book.title CONTAINS( 'everblaze')
AND book.language = 'English'
RETURN book LIMIT 15
```

Slika 5.12: Cypher upit za pretraživanje

Literatura

[1] Theo Reuter Andreas; Härder. "Principles of Transaction-Oriented Database Recovery". *ACM Computing Surveys* (1993.).

[2] Reinhard Diestel. "Graph Theory". (2000.).

[3] Nancy Gilbert Seth; Lynch. "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services". *ACM SIGACT News. Association for Computing Machinery (ACM)*. (2002.), str. 51–59.

[4] Russell K. ; Wang Ming; Chan. "Database Development Process". *Encyclopedia of Information Systems* (2003.).

[5] Ekstrom Andersson E. "Investigating Google's PageRank algorithm". (2004.).

[6] Ian Robinson, Jim Webber i Emil Eifrem. *Graph Databases*. O'Reilly Media, Inc., 2015.

[7] Aleksandar Stojanović. "Osvrt na NoSql baze podataka - četiri osnovne tehnologije". *Polytehnic design* (2016.).

[8] Sandeep K. Raghuwanshi i R. K. Pateriya. "Recommendation Systems: Techniques,Challenges, Application, and Evaluation". *Soft Computing for Problem Solving* (2019.).

[9] Utta Patel Dhruval;Chauhan i Foram; Patel. "Recommendation Systems: Types, Applications, and Challenges". *International Journal of Computing and Digital Systems* (2023.).

[10] *Algoritmi nad grafovima*. URL: https://memgraph.com/blog/what-is-a-graph-database.

[11] *BFS algoritam*. URL: https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/.

[12] *CAP theorem details*. URL: ibm.com/topics/cap-theorem.

[13] *Content-Based Recommender System Using NLP*. URL: https://medium.com/mlearning-ai/content-based-recommender-system-using-nlp-445ebb777c7a.

[14] *Cypher Cheat Sheet*. URL: https://neo4j.com/docs/cypher-cheat-sheet/5/auradb-enterprise/#_lists.

[15] *Data Consistency Models: ACID vs. BASE Explained*. URL: https://neo4j.com/blog/acid-vs-base-consistency-models-explained/5.

[16] *DFS algoritam*. URL: https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/.

[17] *Dijkstrin algoritam*. URL: https://www.geeksforgeeks.org/introduction-to-dijkstras-shortest-path-algorithm/.

[18] *Goodreads*. URL: https://www.goodreads.com/list/show/1.Best_Books_Ever.

[19] *Graph Databases for Beginners: Why Graph Technology Is the Future*. URL: https://neo4j.com/blog/why-graph-databases-are-the-future/.

[20] *Izvor podataka*. URL: https://github.com/scostap/goodreads_bbe_dataset/blob/main/Best_Books_Ever_dataset/books_1.Best_Books_Ever.csv.

[21] *Neo4j Cypher manual*. URL: https://neo4j.com/docs/cypher-manual/current/clauses/match/.

[22] *Powering Recommendations with a Graph Database: Proven Business Benefits*. URL: https://neo4j.com/blog/powering-recommendations-graph-database-proven-business-benefits/.

[23] *Welcome to Neo4j*. URL: https://neo4j.com/docs/getting-started/.

[24] *What is NoSQL*. URL: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-nosql/.

[25] *Why Graph Databases*. URL: neo4j.com/why-graph-databases/.

Sažetak

Zbog potrebe za sve bržim odgovorom aplikacija na upite koji se izvršavaju nad podacima, dolazi do razvoja NoSQL baza podataka. U ovom radu objašnjeno je što su NoSQL baze podataka, opisana su njihova svojstva i navedene su njihove prednosti. U NoSQL bazama podataka, podaci se mogu pohranjivati na različite načine što dovodi do podjele NoSQL baza, ovisno o strukturi podataka koja se u njima koristi. U radu su navedene vrste NoSQL baza i kada je najbolje koristiti koju. Kako je jedna od najkorištenijih NoSQL baza graf baza, objašnjeno je što su grafovi i predstavljeni su najpoznatiji algoritmi nad njima, a na temelju toga objašnjeno je kako se graf baza može modelirati. Osim toga, navedeni su primjeri primjene i korisnosti graf baza.
U radu je dan uvid u najpoznatiju graf bazu, Neo4j i jezik kojime se ona koristi, Cypher. Opisana je sintaksa Cyphera i objašnjene su najčešće korištene naredbe. U sklopu rada, napravljena je web aplikacija koja u svojoj pozadini koristi Neo4j za upravljanjem podacima. U radu je opisano modeliranje baze koja se nalazi u pozadini aplikacije, sučelje aplikacije, kao i naredbe koje su u njoj koriste.

Ključne riječi

NoSql, grafovi, graf baza, Neo4j, Cypher, sustav preporuka

Development of a recommendation system using a graph database

Summary

The development of NoSQL databases began due to the need for a quick response from the application to the queries executed on the data. This paper explains NoSQL databases, describes their properties, and lists their advantages. In NoSQL databases, data can be stored in different ways, so the NoSQL databases are divided depending on the data structure used. This paper lists the types of NoSQL databases and explains when to use them. Since one of the most used NoSQL databases is a graph database, it explains what graphs are and the most famous graph algorithms. Based on this, how the graph database can be modeled is explained. In addition, there are examples of the application and usefulness of the graph base. The paper provides insight into the most famous graph base, Neo4j, and the language it uses, Cypher. Cypher syntax and the most commonly used commands are described. As a part of this thesis, a web application that uses Neo4j in its background for data management was made. The thesis describes the modeling of the database in the background of the application, the interface of the application, and the commands used in it.

Keywords

NoSql, graphs, graph databases, Neo4j, Cypher, recommendation system

Životopis

Rođena sam 30.1.2000. u Osijeku gdje sam završila osnovnu i srednju školu. Nakon završetka III. Gimnazije Osijek, 2018. godine upisala sam preddiplomski studij matematike i računarstva na Odjelu za matematiku. Odmah nakon toga, nastavljam obrazovanje na diplomskom studiju smjera matematika i računarstvo. Prilikom završetka fakultetskog obrazovanja zaposlena sam u tvrtki Span gdje sam odradila stručnu praksu.