Usporedba relacijskih i NoSQL baza podataka

Čerkez, Katarina

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Katarina Čerkez

USPOREDBA RELACIJSKIH I NOSQL BAZA PODATAKA

ZAVRŠNI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Katarina Čerkez

Matični broj: 44852/16-R

Studij: Poslovni sustavi

USPOREDBA RELACIJSKIH I NOSQL BAZA PODATAKA

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Kornelije Rabuzin

Varaždin, rujan 2020.

Katarina Čerkez

Izjava o izvornosti

Izjavljujem da je moj završni/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/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovom radu objašnjene su osnove relacijskih baza podataka, što uključuje pojmove entiteta, veza i atributa te ERA modela kao i pojam referencijalnog integriteta. Na primjeru sustava knjižnice prikazan je ERA model te nekoliko upita i okidač nad tom bazom podataka kao demonstracija dohvaćanja podataka u relacijskim bazama, a opisani su i najpoznatiji sustavi za rad s relacijskim bazama podataka.

Također je objašnjen pojam NoSQL baze podataka i opisane su četiri najzastupljenije vrste istih: ključ-vrijednost baze podataka, grafovske baze podataka, „dokumentne baze podataka“ i „stupčaste baze podataka“. Navedeni su primjeri za svaku od njih, a za grafovske baze podataka napravljena je jednostavna baza i nekoliko upita u alatu Neo4j, kao i aplikacija koja radi sa Neo4j bazom podataka.

Na kraju su navedene prednosti i nedostaci NoSQL baza podataka i razlozi rasta popularnosti istih.

Ključne riječi: baza podataka; relacijska baza podataka; SQL; NoSQL; skalabilnost; ključ-vrijednost sustavi; grafovska baza podataka; dokumentna baza podataka; stupčasta baza podataka

Sadržaj

1. Uvod

Relacijske baze podataka vodeći su sustavi za rad s podacima već dugi niz godina i zasnovane su na relacijskom modelu podataka. Međutim, s novim tehnološkim izazovima i potrebama javile su se i nove vrste baza podataka koje se ne temelje na relacijskom modelu, već na slobodnijem pristupu. One su tako dinamičke i imaju otvoren tip i broj atributa nekog entiteta, što znači da se po potrebi mogu mijenjati bez da to utječe na druge podatke. Takve baze podataka su NoSQL baze podataka.

NoSQL baze podataka kao najveću prednost imaju mogućnost pohrane i obrade jako velikih količina podataka, zbog čega su u današnje doba sve popularnije jer se javlja potreba za takvom pohranom podataka (društvene mreže i sl.). S obzirom da ovakva tehnologija uzima sve više maha, poželjno je istražiti kakve vrste NoSQL baza podataka postoje i kako funkcioniraju te koje su njihove razlike u odnosu na već tradicionalni pristup relacijskih baza podataka.

Navodeći prednosti i nedostatke NoSQL-a nad relacijskim bazama podataka, u ovom radu pokušat će se odrediti koji od ova dva pristupa je u konačnici bolji odabir i je li nagla popularnost NoSQL-a znači da će relacijske baze podataka postati zastarjeli pristup.

2. Metode i tehnike rada

Za izradu ovog rada korištena je i istraživana literatura i analizirani članci navedeni u popisu na kraju rada. Za opisivanje sustava ili korištenje istih, istraživane su službene stranice s dokumentacijom i internetski članci.

ERA model baze podataka za knjižnicu napravljen je u alatu Oracle SQL Modeler, a prikazani upiti i okidač u Oracle SQL Developer. Graf baza podataka napravljena je u alatu Neo4j, a aplikacija koja koristi tu bazu u Visual Studio razvojnoj okolini. Za sve opisane vrste baza podataka naveden je primjer alata koji se koriste za implementaciju istih. Za ključ-vrijednost sustave, primjer je napravljen u Redisu, a za stupčaste u Cassandri, odnosno u DataStax Astra.

3. Relacijske baze podataka

„Baza podataka je kolekcija podataka, ograničenja i operacija koja reprezentira neke aspekte realnog svijeta.“ [1]

Ti podaci pohranjeni su u vanjskoj memoriji računala i dostupni su raznim korisnicima i aplikacijskim programima. Upravljanje podacima, tj. upisivanje, čitanje, promjenu i brisanje obavlja poseban softver – sustav za upravljanje bazom podataka (DBMS – Database Managament System). On oblikuje fizički prikaz baze na temelju logičke strukture i obavlja operacije s podacima, brine za sigurnost podataka te automatizira administrativne poslove s bazom. [2]

Logička struktura podataka izgleda onako kako prikazuje model podataka, odnosno skup pravila koji čini osnovu za oblikovanje baze. Postoji više modela podataka: [2]

- Relacijski model – zasnovan na matematičkom pojmu relacije; podaci i veze prikazuju se tablicama koje imaju retke i stupce
- Mrežni model – baza je prikazana kao mreža koja se sastoji od čvorova i usmjerenih lukova; čvorovi predstavljaju slogove podataka, a lukovi veze među podacima
- Hijerarhijski model – slučaj mrežnog modela; baza je prikazana kao stablo ili skup stabala, a svako stablo sastoji se od čvorova i veza „nadređeni-podređeni“ između čvorova
- Objektni model – inspiriran objektno-orijentiranim jezicima; baza je prikazana kao skup trajno pohranjenih objekata, svaki objekt pripada nekoj klasi, a između klasa postoje veze

Od osamdesetih godina pa sve do danas prevladavaju relacijske baze podataka, dok su mrežni i hijerarhijski modeli bili popularni prije toga. [2]
U nastavku ću govoriti upravo o relacijskim bazama podataka i demonstrirati kako one funkcioniraju.

Relacijska baza podataka je baza u kojoj su podaci spremljeni u (obično) povezanim relacijama i strukturirani tako da se osigura: [3]

- Ažurnost pohranjenih podataka
- Sigurnost i nadzor pristupa podacima
- Mogućnost dnevnog sigurnosnog arhiviranja
- Mogućnost kontrole i administriranja s jednog mjesta

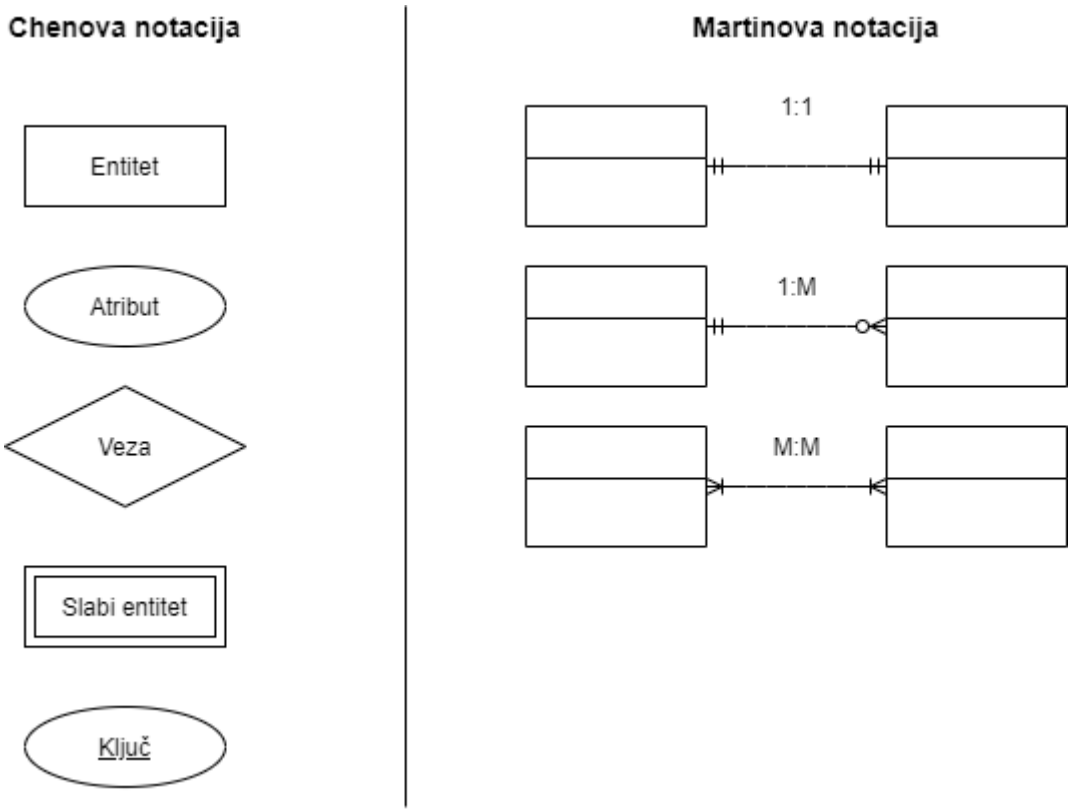
- Mogućnost postavljanja više upita s više kriterija za izradu analize i sintezu podataka
- Najmanja redundancija
- Postojanost i točnost pohranjenih podataka
- Trajno očuvanje integriteta pohranjenih podataka

Ona se bazira na relacijskom modelu koji organizira podatke u jednu ili više tablica (relacija) koje se sastoje od stupaca i redaka s jedinstvenim ključem kao identifikatorom za svaki red. Redovi se također nazivaju slogovima podataka, a stupci se nazivaju atributima. Svaka tablica, odnosno relacija, predstavlja tip entiteta, redovi predstavljaju instance tog entiteta, a stupci vrijednosti dodijeljene toj instanci, odnosno njezine atribute. [4]

3.1. ERA model

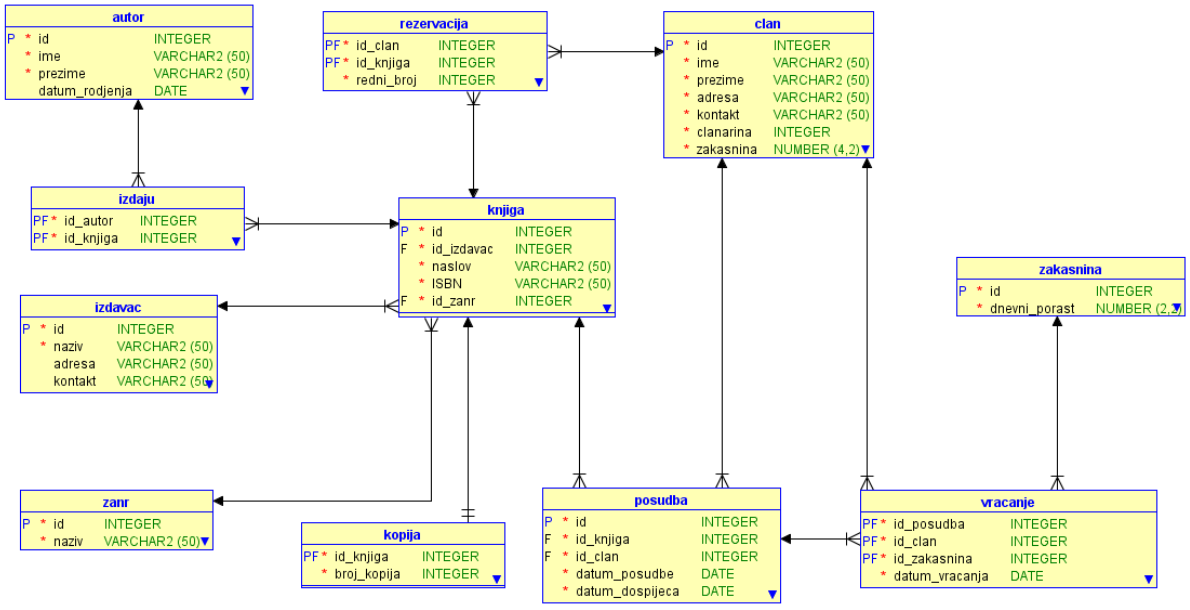
ERA model (eng. *Entity Relationship Attribute*) je konceptualni model podataka u kojem su prikazani entiteti sa svojim atributima i veze između njih. Prema ERA modelu implementira se baza podataka. Otac ERA modela, Peter Chen, napisao je: „ER model prihvaća prirodno stajalište da se stvarni svijet sastoji od entiteta i veza. Sadrži neke od važnih semantičkih informacija o stvarnom svijetu“. [6]

Postoji više notacija koje se koriste pri izradi ERA dijagrama, a najzastupljenije su Chenova i Martinova notacija. U Chenovoj notaciji entiteti su prikazani pomoću pravokutnika, atributi pomoću elipsa, a veze pomoću rombova, dok su u Martinovoj notaciji entiteti tablice sa listom atributa, a između tablica se crtaju veze kako je prikazano na slici 1.



Slika 1: Chenova i Martinova notacija

Kao primjer ERA modela slijedi model za knjižnicu:



Slika 2: ERA model za knjižnicu

Ovaj model služi za implementaciju baze podataka za knjižnicu koja služi knjižničaru da bilježi knjige koje knjižnica posjeduje, autore i izdavače istih, članove knjižnice, njihove rezervacije i posudbe knjiga kao i podatke o vraćenim knjigama i eventualnoj zakasnini. Možemo vidjeti da svaka tablica ima ID kao primarni ključ, ali i vanjske ključeve koji omogućuju povezivanje entiteta. Također vidimo više slabih entiteta, uključujući i već spomenuti između tablice *autor* i *knjiga*, a to je entitet *izdaju*, a svaki atribut ima svoju domenu koja je naznačena kao tip podatka u bazi.

3.2. Entiteti i atributi

Entitet predstavlja objekt iz stvarnog svijeta, a podaci koji ga opisuju i koji se pohranjuju zovu se atributi [5]. Primjerice, entitet autora opisan je imenom, prezimenom i datumom rođenja.

Tablica 1: Autor

ID	**Ime**	**Prezime**	**Datum_Rodjenja**
1	Jo	Nesbo	29.03.1960.
2	Meša	Selimović	26.04.1910.
3	Dan	Brown	22.06.1964.

Svrha pohranjivanja podataka u bazu je ta da ih kasnije uvijek možemo dohvatiti. To znači da moramo razlikovati entitete kako bismo bili sigurni da dohvaćamo prave podatke. Stoga mora postojati jedinstveni identifikator, kao što je atribut ID u tablici *Autor*. Tablica ne mora imati eksplicitno atribut naziva ID ako već ima neki atribut koji je jedinstven za svaku instancu entiteta. Primjerice, u entitetu *Osoba* jedinstveni identifikator može biti OIB. Važno je da bez problema možemo dohvatiti svaki redak u tablici i da sačuvamo konzistentnost i točnost podataka. [5] Relacije se također mogu zapisivati na sljedeći način:

naziv (primarni ključ, atribut…)
Autor (ID, Ime, Prezime, Datum_Rodjenja)

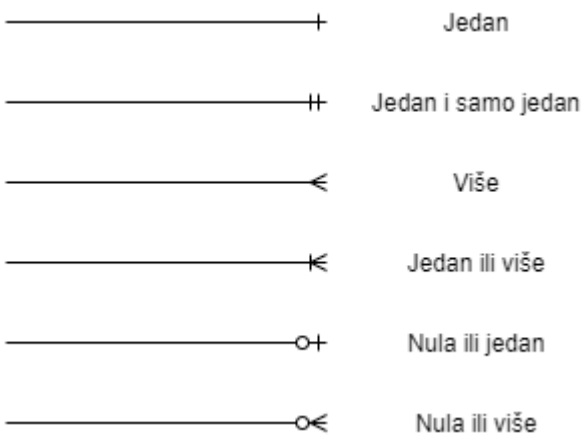
3.3. Domena

Svaki atribut ima svoju domenu, odnosno raspon dozvoljenih vrijednosti za taj atribut. Sustav za upravljanje bazom podataka provjerava je li unesena vrijednost atributa u njegovoj domeni. Većina sustava koji koriste SQL kao upitni jezik omogućuje niz tipova podataka koji se pridružuju atributu kao njegova domena. Neki od njih su [5]:

- CHAR
- VARCHAR
- INT
- DECIMAL i NUMERIC
- DATE
- TIME
- DATETIME
- BOOLEAN

3.4. Veze među entitetima

Postoje tri osnovna tipa veze među entitetima: jedan na jedan, jedan na više i više na više. U bazi podataka veze povezuju instance entiteta. Kada prikazujemo veze među entitetima na ERA dijagramu, pokazujemo moguće veze koje su dozvoljene u bazi podataka. Osim ako ne specificiramo da je veza obavezna, nije nužno da svaka instanca svakog entiteta bude povezana s drugom instancom [5]. Primjerice, u sustavu knjižnice, član ne mora nužno u svakom trenutku posuditi neku knjigu.



Slika 3: Tipovi veza

Veze jedan na jedan su rjeđe u svakodnevnom poslovanju, te treba pažljivo promotriti je li zbilja u pitanju veza jedan na jedan ili je to samo poseban slučaj veze jedan na više ili bi pak dva entiteta koja želimo povezati trebalo spojiti u jedan entitet.
Najčešći tip veze je jedan na više i većina relacijskih baza podataka građena je od brojnih veza jedan na više i rijetkih veza jedan na jedan.

Veze više na više su također česte, međutim pri prikazivanju baze na ERA dijagramu, ako postoji veza više na više, moramo uvesti takozvani slabi entitet kako bi baza funkcionirala. [5] U sustavu knjižnice, autor ne ovisi o knjizi, odnosno možemo upisati autora u bazu bez obzira je li ima izdane knjige. Međutim, obrnuta situacija ne vrijedi. Ako postoji knjiga u bazi, ona mora imati svog autora. Stoga između ta dva entiteta uvodimo slabi entitet, „međutablicu“, koja povezuje prethodna dva entiteta vezama jedan na više.

3.5. Primarni i vanjski ključevi

Primarni ključ omogućuje da se svaki red u tablici može jednoznačno odrediti.
Da bi se u relacijskim bazama pronašao traženi podatak, potrebno je ime tablice, ime stupca i primarni ključ reda. Stoga je jako važno da je primarni ključ jedinstven za svaki redak u tablici i tada smo sigurni da baza vraća točno onaj podatak koji smo zatražili.
Osim što mora biti jedinstven, primarni ključ ne smije poprimiti *null*. Sustavi za upravljanje bazom podataka neće dopustiti da se ne unese vrijednost za primarni ključ.
Stoga je poželjno da primarni ključ neke tablice bude vrijednost za koju je mala vjerojatnost da ikad može biti *null* te da se ta vrijednost nikad ne mijenja. [5]

Stupac u tablici koji odgovara primarnom ključu neke tablice naziva se vanjski ključ i predstavlja vezu između entiteta. Vanjski ključevi mogu biti dijelom primarnog ključa, ali ne moraju. [5]

izdavac (<u>id</u>, naziv, adresa, kontakt)
knjiga (<u>id</u>, *id_izdavac*, naslov, ISBN)

Relacija knjiga ima vanjski ključ na relaciju izdavač, odnosno na njezin primarni ključ i zapisuje se u kurzivu. Na ERA modelu sa slike 3 primarni ključ ima oznaku P (eng. *primary key*), a vanjski ključ F (eng. *foreign key*).

3.6. Referencijalni integritet

Referencijalni integritet je ograničenje koje zahtijeva da svaka vrijednost vanjskog ključa koja nije *null* mora odgovarati postojećoj vrijednosti primarnog ključa. Ovo ograničenje je iznimno važno, ako ne i najvažnije od svih ograničenja u relacijskim bazama podataka jer osigurava konzistentnost veza među tablicama. Sustav za upravljanje bazom podataka automatski primjenjuje referencijalni integritet. Svaki put kad korisnik unosi ili mijenja podatke, sustav provjerava ograničenja i ako promjene ne zadovoljavaju ograničenje, neće dopustiti promjene u bazi. [5]

3.7. SQL

SQL (eng. *Structured Query Language*) standardni je upitni jezik koji se koristi u relacijskim bazama podataka. Edgar F. Codd definirao je relacijski model baze podataka 1969. godine, a nekoliko godina nakon toga IBM na temelju tog modela počeo je raditi na jeziku koji će se u njima koristiti. Nazvali su ga SEQUEL (eng. *Structured English Query Language*), ali nakon nekoliko izmjena, napokon je nazvan SQL, kako ga danas poznajemo. [7]

SQL se može podijeliti na tri 'podjezika':

1. DDL (eng. *Data Definition Language*) za kreiranje i modificiranje baze podataka
2. DML (eng. *Data Manipulation Language*) za CRUD (eng. *Create, Read, Update, Delete*) operacije s podacima
3. DCL (eng. *Data Control Language*) za kontrolu pristupa podacima u bazi

DML naredbe uključuju klauzule INSERT, SELECT, UPDATE i DELETE koje čine osnovu rada s podacima. [8]

Na primjeru baze podataka za knjižnicu čiji ERA model je prikazan na slici 3, pokazat ću kako se mogu raditi jednostavniji upiti nad podacima te definirati okidači (eng. *trigger*) koje pokreću događaji u bazi podataka. Za dohvaćanje podataka iz baze koristi se naredba SELECT. Želimo li dohvatiti sve autore koji se nalaze u bazi podataka, upit je vrlo jednostavan. Kako bi definirali iz koje tablice želimo uzeti podatke, koristimo klauzulu FROM.

```
SELECT * FROM autor;
```

Query Result

SQL | All Rows Fetched: 5 in 0,003 seconds

	ID	IME	PREZIME	DATUM_RODJENJA
1	1	Jo	Nesbo	29.03.60
2	2	Dan	Brown	22.06.60
3	3	Meša	Selimović	26.04.19
4	4	Lisa	Genova	22.11.19
5	5	Hrvoje	Hitrec	14.07.19

Slika 4: Primjer SQL upita koji dohvaća sve podatke iz tablice

Također možemo specificirati koje atribute iz kojih tablica koje su međusobno povezane želimo prikazati pomoću klauzule WHERE.

```
SELECT ime, prezime, naslov
FROM clan, knjiga, posudba
WHERE posudba.id_clan = clan.id
AND posudba.id_knjiga = knjiga.id;
```

Query Result

SQL | All Rows Fetched: 9 in 0,016 seconds

	IME	PREZIME	NASLOV
1	Mate	Matic	Crvendać
2	Mate	Matic	Velika obmana
3	Mate	Matic	Jos uvijek Alice
4	Ivo	Ivic	Crvendać
5	Ana	Anic	Crvendać
6	Ana	Anic	Tvrđava

Slika 5: Primjer SQL upita s WHERE klauzulom

Moguće je raditi i kompleksnije upite nad podacima pri čemu je vrla važna JOIN klauzula pomoću koje združujemo više tablica i dohvaćamo podatke. U sljedećem primjeru kombiniramo podatke iz četiri tablice.

```
SELECT ime, prezime, naslov, datum_posudbe, datum_vracanja
FROM posudba
LEFT JOIN clan ON posudba.id_clan = clan.id
LEFT JOIN vracanje ON vracanje.id_posudba = posudba.id
AND vracanje.id_clan = clan.id
LEFT JOIN knjiga ON posudba.id_knjiga = knjiga.id;
```

Query Result

SQL | All Rows Fetched: 9 in 0,008 seconds

	IME	PREZIME	NASLOV	DATUM_POSUDBE	DATUM_VRACANJA
1	Pero	Peric	Tvrđava	20.01.19	04.02.19
2	Ivo	Ivic	Crvendać	14.01.19	19.01.19
3	Mate	Matic	Jos uvijek Alice	17.01.19	19.01.19
4	Ana	Anic	Tvrđava	10.01.19	27.01.19
5	Mate	Matic	Velika obmana	17.01.19	03.02.19
6	Marko	Maric	Velika obmana	08.01.19	24.01.19

Slika 6: Primjer SQL upita s JOIN klauzulom

Okidači su također vrlo korisni za kontrolu rada s podacima, ažuriranja i slično. U spomenutoj bazi podataka za knjižnicu, napravljen je okidač koji se pokrene kada član knjižnice vrati knjigu, odnosno kad se dogodi unos u tablicu *vracanje*. Tada se u slučaju kašnjenja izračuna iznos zakasnine i ažurira stupac *zakasnina* u tablici *clan*.

```
CREATE OR REPLACE TRIGGER RACUNAJ
AFTER INSERT ON VRACANJE FOR EACH ROW

DECLARE
    porast INT;
    broj_dana INT;
    zakasnjeno INT;

BEGIN

SELECT (:NEW.datum_vracanja-datum_dospijeca), dnevni_porast, zakasnina
INTO broj_dana, porast, zakasnjeno FROM posudba, zakasnina, clan
WHERE posudba.id=:NEW.id_posudba AND zakasnina.id=:NEW.id_zakasnina AND clan.id=:NEW.id_clan;

IF (broj_dana > 0) THEN zakasnjeno:=zakasnjeno+(broj_dana*porast);

END IF;

UPDATE clan SET zakasnina=zakasnjeno WHERE clan.id=:NEW.id_clan;

END;
```

Slika 7: Primjer okidača

3.8. Najpoznatiji sustavi za upravljanje relacijskim bazama podataka

Sustav za upravljanje bazom podataka je softver koji pohranjuje, organizira i dohvaća podatke kako bi aplikacija mogla njima upravljati. Sustavi koji se baziraju na relacijskom modelu razlikuju logičke i fizičke operacije. Kod logičkih operacija, aplikacija definira koji sadržaj se zahtijeva, a kod fizičkih operacija, sustav određuje na koji način će se operacije izvršiti. Primjerice, aplikacija zatraži podatak iz tablice, a sustav se pobrine da pronađe red u tablici kako bi korisnik dobio traženi podatak. On pohranjuje i dohvaća podatke tako da fizičke operacije budu transparentne logičkima. [9]

Najpoznatiji i najkorišteniji sustavi za upravljanje relacijskim bazama podataka su Oracle, MySQL, Microsoft SQL Server i PostgreSQL.

3.8.1. Oracle

Larry Ellison, Bob Miner i Ed Oates 1977. godine osnovali su Software Development Laboratories koji je prerastao u tvrtku Relational Software, Inc. (RSI), a 1983. napokon u Oracle Corporation. RSI je 1979. godine ostavio bitan trag u povijesti relacijskih baza podataka

tako što je predstavio Oracle V2 (Version 2)– prvi komercijalno dostupan sustav za upravljanje relacijskim bazama podataka zasnovan na SQL-u. [9]

Slika 8: Oracle Database logo

Oracle Version 3, koji je predstavljen 1983. godine, bio je prva relacijska baza podataka koja je radila na osobnim računalima. Svaka sljedeća verzija donosila je inovacije i poboljšanja, od distribuiranosti baze podataka, skalabilnosti, mogućnosti sigurnosnog kopiranja i oporavka podataka do prve verzije PL/SQL-a, proširene verzije SQL upitnog jezika. Kasnije je PL/SQL imao i ugrađene procedure i okidače. [9]

Oracle Database 11g predstavljen je 2007. godine s novim značajkama koje programerima i administratorima omogućuju brzu prilagodbu promjenjivim poslovnim zahtjevima tako što se pojednostavila informacijska infrastruktura i uvela automatizacija gdje god je bilo moguće. [9] Spomenuti PL/SQL ima dosta prednosti, a jedna od njih je da pohrani aplikacijsku logiku u samu bazu podataka. Oracle Database također može pohraniti programske jedinice pisane u Javi. Postojeći PL/SQL programi mogu se pozvati iz Jave, a Java programi mogu se pozvati iz PL/SQL-a. [9]

Oracle Database jamči konzistentnost i sigurnost podataka te radi na hardverima u različitim operacijskim sustavima, uključujući Windows Server, Unix i razne distribucije GNU/Linux-a. Također, aplikacija koja se povezuje s Oracle bazom podataka i sama baza ne moraju biti na istoj platformi. Recimo, aplikacija pokrenuta na Windowsu se može povezati s Oracle bazom podataka koja radi na Unixu. Sve su to prednosti Oracle-a i razlozi zašto je godinama jedan od najkorištenijih SUBP-ova. [10]

3.8.2. MySQL

MySQL je sustav za upravljanje relacijskim bazama podataka koji je otvorenog koda i baziran je na SQL-u. Kao i Oracle, dostupan je na raznim operacijskim sustavima (Windows, Linux, Solaris). Kreiran je od strane švedske tvrtke MySQL AB 1995. godine. Isprva su se koncentrirali prvenstveno na brzinu i produktivnost, ali s novim verzijama dodane su i druge bitne značajke. Popularnost je stekao jer je brz, jednostavan za korištenje i instalaciju i koristi

upitni jezik koji je lako razumljiv. Sun Microsystem kupio je MySQL AB 2008. godine, a Oracle je kupio Sun Microsystem 2009. godine, tako da MySQL trenutno razvija Oracle. [11]

MySQL može se koristiti na raznim platformama, ali posebno je korišten u području web aplikacija. O njegovoj popularnosti govori i to da stoji iza Facebooka, Twittera i YouTubea. [12]

Slika 9: MySQL logo

MySQL baziran je na modelu klijent-server, a MySQL server sadrži instrukcije i obavlja naredbe u bazi podataka. Dostupan je kao odvojen program i kao biblioteka (eng. *library*) koja se može koristiti u nekoj aplikaciji. MySQL napravljen je kako bi pri velikoj brzini upravljao velikim bazama podataka. Obično se instalira na jednom računalu, ali moguće je preko MySQL klijenta pristupiti bazi podatka s više lokacija. Naredbe se šalju MySQL serveru pa se prikažu rezultati. [12]

MySQL napisan je u C i C++ programskim jezicima i dostupan je na preko 20 platformi. Podržava jako velike baze podataka i ima mogućnost replikacije podataka i particioniranja tablica za bolje performanse i izdržljivost. [12]

3.8.3. Microsoft SQL Server

Microsoft razvija SQL Server više od 20 godina, i dok je prije radio isključivo na Windows operacijskim sustavima, a od 2016. godine dostupan je i na Linux operacijskim sustavima. Također je baziran na SQL upitnom jeziku, a vezan je i uz Transact-SQL ili T-SQL, Microsoftovu implementaciju SQL-a koja ima dodatne programske konstrukcije. [13]

Pomoću SQL Servera moguće je stvarati baze podataka i njima upravljati, ali i analizirati podatke koristeći SQL Server Analysis Services i generirati izvještaje koristeći SQL Server Reporting Services. Kao i prethodni sustavi, baziran je na modelu klijent-server. [14]

Slika 10: SQL Server logo

3.8.4. PostgreSQL

PostgreSQL je moćan objektno-relacijski sustav otvorenog koda koji koristi i proširuje SQL upitni jezik. Nastao je 1986. godine kao dio POSTGRES projekta na Sveučilištu u Kaliforniji. Svoju reputaciju između ostalog duguje arhitekturi, pouzdanosti, integritetu podataka posvećenosti otvorenom kodu. PostgreSQL također radi na svim važnim operacijskim sustavima, a budući da je besplatan i otvorenog koda, nudi mnoštvo mogućnosti za izradu kvalitetnih aplikacija kao što su definiranje vlastitih tipova podataka, izradu vlastitih funkcija, čak i pisanje koda u različitim programskih jezika bez da to utječe na bazu podataka.

Slika 11: PostgreSQL logo

PostgreSQL teži prilagodbi SQL standardima. Otkad je u listopadu 2019. godine izdana verzija 12, PostgreSQL je u skladu sa 160 od 179 obaveznih značajki standarda SQL:2016 Core, više nego ijedan drugi sustav za upravljanje relacijskim bazama podataka. [15]

4. NoSQL baze podataka

NoSQL (eng. *Not only SQL*) baze podataka su nerelacijske baze podataka koje se značajno razlikuju od tradicionalnih relacijskih baza podataka. Namijenjene su za pohranu velikih količina podataka (npr. Google ili Facebook koji dnevno pohranjuju terabajte podataka). Ova vrsta pohrane podataka ne mora zahtijevati fiksnu shemu, distribuirana je, a skalabilnost je horizontalna. [16]

NoSQL baze podataka su dinamičke, što znači da se tip i broj atributa nekog entiteta može mijenjati bez da to utječe na podatke. Zbog toga je interakcija između aplikacije i baze podataka jednostavnija budući da je jednostavnije preslikati strukturu podataka u nekom programskom jeziku da odgovara strukturi baze podataka. Kad kažemo da su ovakve baze podataka distribuirane, to znači da su napravljene tako da se raspoređuju na više računala gdje svaki radi sa svojom skupinom podataka, dok se kod relacijskih baza radi na jednom računalu. Ovime se ubrzava prihvat i obrada podataka, što je prednost u odnosu na relacijske baze podataka. [17]

Termin NoSQL smislio je Carlo Strozzi 1998. godine kako bi nazvao svoju bazu koja nije imala SQL sučelje. Rane 2009. Eric Evans je ponovno upotrijebio taj termin za baze koje nisu relacijske i iste godine na konferenciji u Atlanti u SAD-u, puno se govorilo i raspravljalo o NoSQL-u. Nakon toga, NoSQL bazama podataka je popularnost počela rasti. [16]

Četiri su glavne vrste NoSQL baza podataka: [17]

1. Ključ-vrijednost baze podataka(eng. *Key-value databases*)
2. Grafovske baze podataka (eng. *Graph databases*)
3. Dokumentne baze podataka (eng. *Document databases*)
4. Stupčaste baze podataka (eng. *Columnar databases*)

4.1. Skalabilnost

Skalabilnost označava sposobnost sustava da se proširuje, odnosno da izdrži dodavanje resursa tako da zadovolji potrebe poslovanja. Primjerice, skaliranje web aplikacije za cilj ima da omogući što više ljudi da tu aplikaciju koristi. Postoje dvije vrste skalabilnosti: [16]

- Vertikalna skalabilnost – podrazumijeva dodavanje (ili uklanjanje) resursa serveru na kojem su pohranjeni podaci, a veći broj elemenata povećava složenost upravljanja
- Horizontalna skalabilnost – podrazumijeva dodavanje (ili uklanjanje) servera sustavu, npr. dodavanje novog računala i pohranu podataka na više različitih servera, fizičkih ili virtualnih

Kod NoSQL baza podataka skalabilnost je horizontalna što im je prednost nad relacijskim kod kojih je vertikalna, budući da se lakše upravlja podacima kad se skalira horizontalno.

4.2. Ključ-vrijednost baze podataka

Ova vrsta temelji se na modelu podataka koji se sastoji od parova oblika (ključ, vrijednost) tako da je jednom tekstualnom ključu pridružena vrijednost bilo kojeg tipa. Ovakve baze podataka nemaju upitni jezik već samo omogućavaju da se na osnovu ključa parovi pronalaze, dodaju i uklanjaju, tako da podržavaju tri operacije: [17]

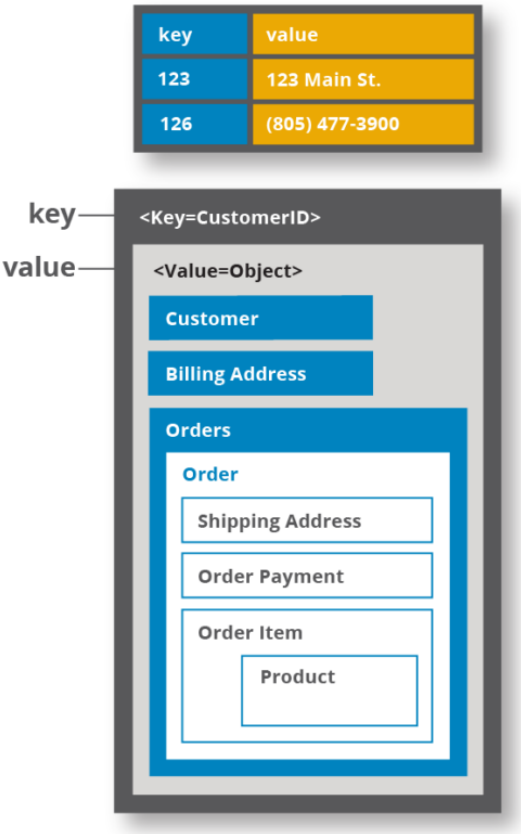
- GET (ključ) – vraća vrijednost koja je pridružena zadanom ključu
- PUT (ključ, vrijednost) – dodaje novi par ili pridružuje novu vrijednost postojećem
- DELETE (ključ) – uklanja par

Za ovakve baze podataka vrijede pravila da svaki ključ u tablici mora biti jedinstven i da upit mora biti baziran na ključu, a ne na vrijednosti.
Budući da da ove baze podataka imaju jednostavne operacije, imaju veliku skalabilnost što je velika prednost, jer se može maksimalno iskoristiti efikasnost sustava. [17]

Postoji više sustava koji rade s ovom vrstom baza podataka, a među poznatijima su Riak i Redis koji su otvorenog koda. U Riak bazi vrijednost može biti bilo što - slike, zvuk, XML, JSON i sl. Komande se šalju putem HTTP protokola jer je napravljena da radi u internetskom okruženju, podržava nekoliko programskih jezika, a može ga se proširivati u Erlangu u kojem je napisan. Riak ima veliku prednost u podršci za sustave gdje je stalna raspoloživost neophodna, kao što je kupovina preko Interneta, ali nije najbolji izbor za korištenje ako distribuiranost sustava nije neophodna ili ako su potrebne kompleksnije strukture podataka. [17]

Redis cijeli skup podataka drži u memoriji te ga povremeno upisuje na disk. On se može konfigurirati da sprema podatke nakon određenog broja sekundi, ali ako se sustav sruši, posljednje promjene mogu biti izgubljene. Međutim, ovo je jedan od najbržih sustava ovog tipa

budući da podatke drži u memoriji te je vrlo koristan u slučajevima gdje manji gubitak podataka ne predstavlja problem. [17] . Zato je dobar odabir za pohranu sesija, statistike i pričuvno memoriranje (eng. *cache*), odnosno za pohranu podataka u stvarnom vremenu, koji se često ažuriraju. [18]



Slika 12: Primjer ključ-vrijednost sustava (Izvor: https://hazelcast.com/glossary/key-value-store/)

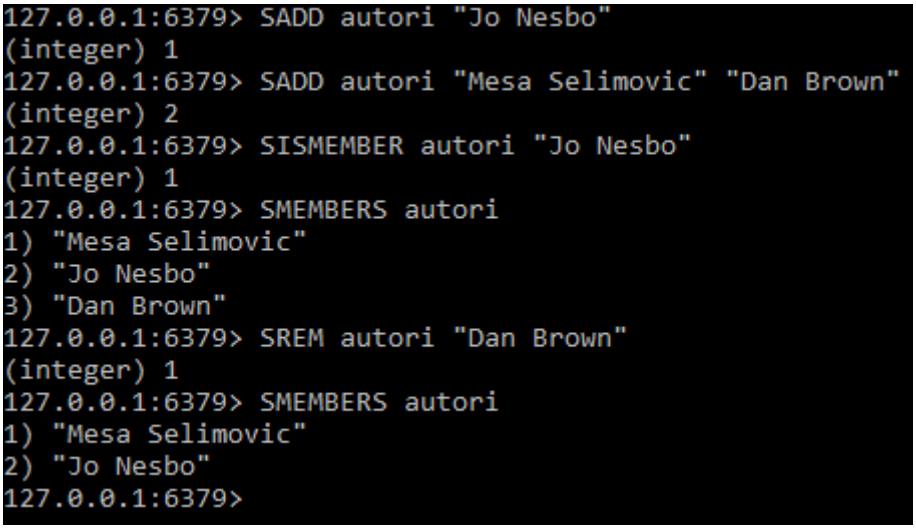
Dodavanje, vraćanje i brisanje parova u Redisu vrlo je jednostavno:

```
127.0.0.1:6379> SET 1 "Jo Nesbo"
OK
127.0.0.1:6379> GET 1
"Jo Nesbo"
127.0.0.1:6379> DEL 1
(integer) 1
127.0.0.1:6379> GET 1
(nil)
127.0.0.1:6379>
```

Slika 13: Osnovne operacije u Redisu

Par ključ-vrijednost dodaje se tako što se napiše naredba SET pa zatim ključ i vrijednost. Vidimo da je Redis potvrdio unos tako što je vratio OK. Želimo li dohvatiti vrijednost, koristimo naredbu GET i navedemo ključ čiju vrijednost želimo dohvatiti, a brišemo par naredbom DEL i navodeći ključ para kojeg želimo obrisati. Redis nas tada obavijesti da smo obrisali jedan par, a pokušamo li ponovno dohvatiti tu vrijednost, vidimo da ona više ne postoji.

Vrijednosti možemo pohraniti i u setove koristeći naredbu SADD [ime seta] [član seta]. Možemo ih dodavati jednog po jednog ili više njih odjednom. Naredbom SISMEMBER ispitujemo postoji li neki član seta, a naredbom SMEMBERS, ispisuju se svi članovi nekog seta. Sa SREM uklanja se član iz seta.



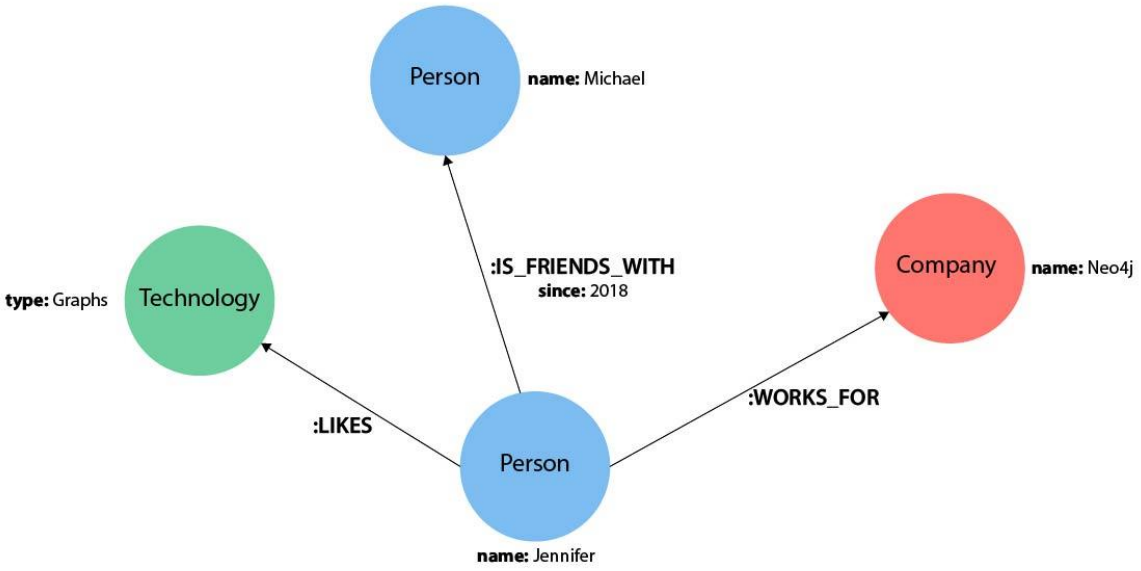
Slika 14: Rad sa setom

4.3. Grafovske baze podataka

Grafovske baze podataka dizajnirane su tako da tretiraju veze između podataka jednako važno kao i same podatke. Zbog toga su jako korisne u aplikacijama gdje su odnosi među podacima iznimno važni. U relacijskim bazama podataka odnosi su implicitni i nisu fleksibilni, dok su u grafovskim bazama podataka odnosi definirani eksplicitno. [17], [19]

U grafovskim bazama podataka entiteti su čvorovi (eng. *nodes*) koji predstavljaju instancu objekta u aplikaciji i imaju svoja svojstva (eng. *properties*) koja su skup ključ-vrijednost parova. Odnosi se nazivaju lukovima (eng. *edges*) koji također mogu imati svojstva i imaju smjer, tip te početni i završni čvor. Organizacija grafa omogućava da se podaci pohrane jednom, a onda se mogu interpretirati na više načina s obzirom na odnose. [19]

Prednost ovako organiziranih podataka u odnosu na relacijski model je da nije potrebno koristiti *join* u upitima i komplicirati iste kako bismo dohvatili tražene podatke jer su u graf modelu te veze već pohranjene s podacima. Grafovske baze podataka upravljaju složenim upitima neovisno o ukupnoj veličini skupa podataka te samo pomoću obrasca i polaznih točaka istražuju susjedne podatke oko tih točaka i prikupljaju i objedinjuju informacije iz milijuna čvorova i veza, a sve podatke koji nisu objedinjeni pretraživanjem ostavljaju netaknutima. [19]



Slika 15: Primjer grafa baze podataka (Izvor: https://neo4j.com/developer/cypher/syntax/)

Postoji mnogo grafovskih sustava, a jedan od glavnih predstavnika je Neo4j. To je grafovska baza podataka otvorenog koda čiji je razvoj započeo 2003. godine, ali je javno dostupna od 2007, a izvorni kod napisan je u jezicima Java i Scala. U Neo4j sustavu podaci su pohranjeni točno onako kako u grafu modeliramo, a baza koristi pokazivače za kretanje po grafu. Ono što Neo4j čini popularnim jesu: [19]

- Cypher – deklarativni upitni jezik sličan SQL-u, ali optimiziran za grafove
- Konstantno kretanje u velikim grafovima za učinkovit prikaz čvorova i odnosa; moguće je skaliranje do milijardu čvorova
- Fleksibilna shema svojstava grafa koja se može kroz vrijeme prilagođavati, što omogućuje dodavanje novih veza kad su potrebne promjene u poslovanju
- Upravljački programi za popularne programske jezike, uključujući Javu, JavaScript, .NET, Python itd.

Neo4j danas se koristi u mnogim organizacijama i mnogim industrijama: energija, proizvodnja, maloprodaja, tehnologija, financijske usluge itd. [19]

4.3.1. Primjer

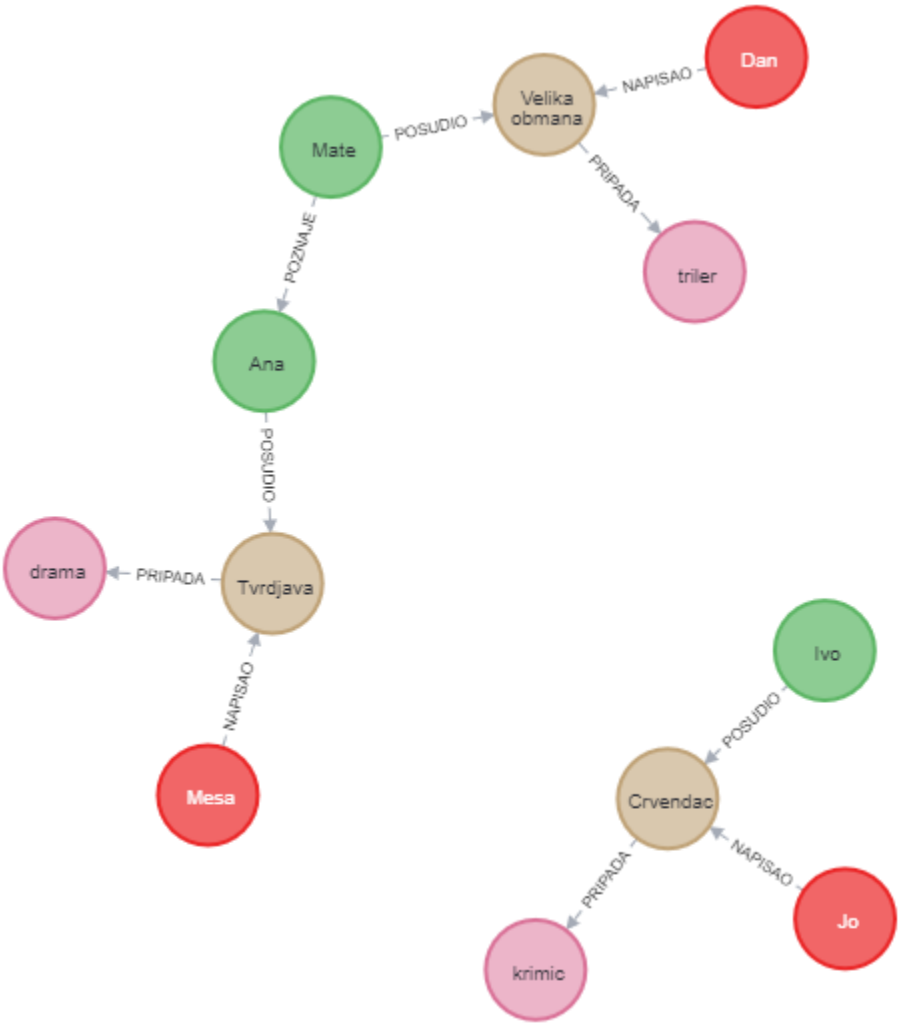
Za demonstraciju rada s grafovskom bazom podataka u alatu Neo4j izrađena je jednostavna baza podataka koja sadrži knjige, autore, žanrove i članove koji su čvorovi sa pripadajućim svojstvima a između njih postoje veze *NAPISAO*, *PRIPADA*, *POSUDIO*, *POZNAJE*.
Čvorovi sa svojim svojstvima kreiraju se na sljedeći način:

```
CREATE (nesbo:Autor {ime:'Jo', prezime:'Nesbo'})
CREATE (crvendac:Knjiga {naslov:'Crvendac'})
```

Autor i *Knjiga* su nazivi čvorova, a ispred njih nalazi se labela koja nije obavezna, ali je korisna za lakše razumijevanje. Ime i prezime, odnosno naslov su svojstva čvora kojima se pridruži vrijednost.
Veze između čvorova implementiraju se na sljedeći način:

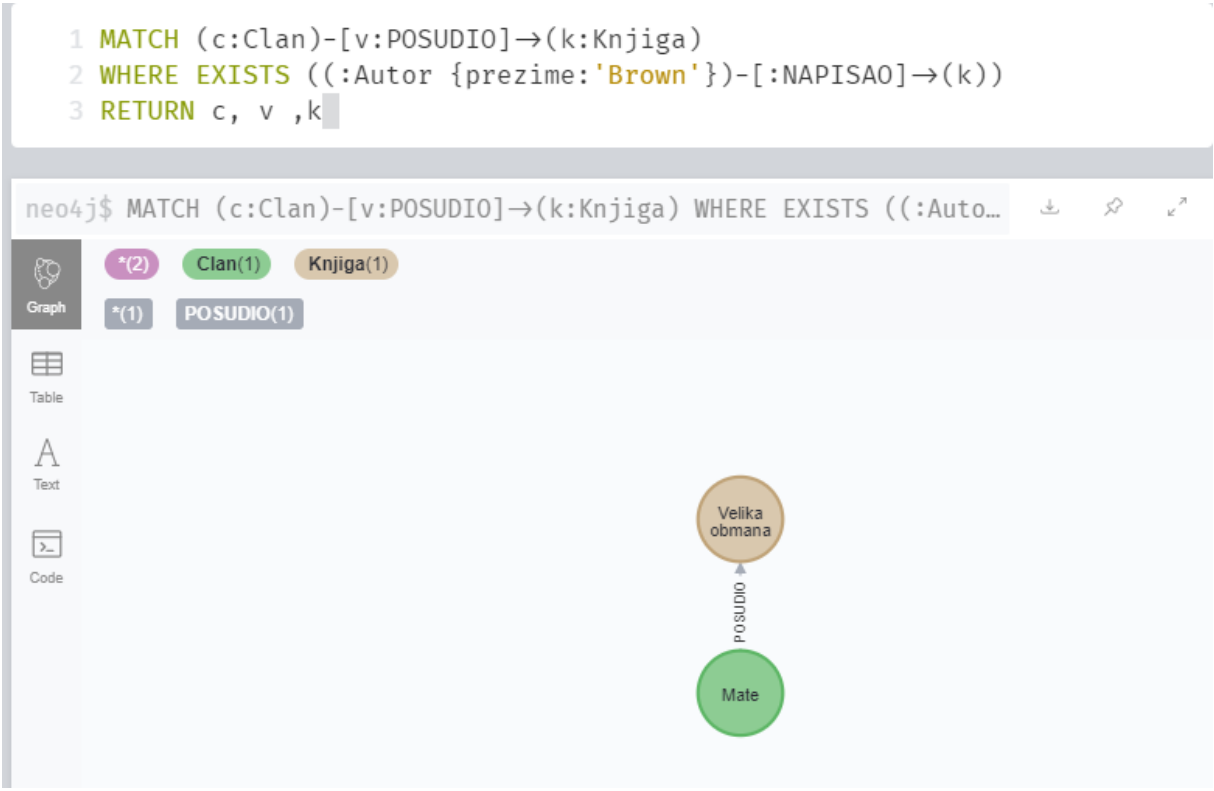
```
CREATE (nesbo)-[:NAPISAO]->(crvendac)
```

Na taj način kreirani su i ostali čvorovi sa svojim svojstvima i definirane veze između njih te u konačnici graf izgleda ovako:



Slika 16: Primjer graf vizualizacije baze podataka u Neo4j

U Cypheru se podaci dohvaćaju pomoću klauzule MATCH, a kao i kod SQL-a, postoji klauzula WHERE. Želimo li pronaći članove knjižnice koji su posudili knjigu koju je napisao Dan Brown, upit i rezultat upita izgleda ovako:



Slika 17: Primjer upita u Neo4j

Sad se ovom grafu mogu jednostavno dodati novi čvorovi i veze bez da se na bilo koji način poremete već pohranjeni podaci. Primjerice, ako želimo prikazati da Ivo voli trilere, pomoću MATCH klauzule dohvatimo ta dva čvora te definiramo novu vezu između njih.

```
MATCH (ivo:Clan {ime:'Ivo'})
MATCH (triler:Zanr {naziv:'triler'})
CREATE (ivo)-[:VOLI]->(triler)
```

Na sličan način podaci se mogu brisati i modificirati. Pomoću MATCH naredbe dohvatimo željene podatke, a onda jednostavno izvršimo naredbu nad tim podacima.

```
MATCH (ivo:Clan {ime:'Ivo'})
DELETE ivo
```

4.4. Dokumentne baze podataka

Dokumentne baze podataka najpopularnija su NoSQL vrsta, ponajviše jer je model podataka kod ovakvih baza intuitivan. Osnovni element takvog modela je uređeni skup ključeva s pridruženim vrijednostima koji se zove dokument. Dokumenti se smještaju u

kolekcije, koje su gotovo ekvivalentne tablicama u relacijskim bazama podataka pri čemu jedan dokument odgovara jednom redu tablice. [17]

Ono što čini razliku između dokumentnih baza podataka i relacijskih baza podataka između ostalog su: [20]

- Intuitivni model podataka – dokumenti mapiraju objekte u kodu pa je s njima prirodnije raditi. Nema potrebe dekomponirati podatke u tablice, podaci kojima se zajednički pristupa, zajedno su i pohranjeni.
- Fleksibilna shema – shema je dinamička te svaki dokument u kolekciji može imati drugačiju strukturu koja se može modificirati po potrebi, što poboljšava performanse.
- Univerzalnost – JSON (eng. *JavaScript Object Notation*) je standard za pohranu i izmjenu podataka
- Distribuiranost – za razliku od monolitnih, vertikalno skaliranih relacijskih baza podataka, dokumentne baze podataka su u osnovi distribuirani sustavi, skalirani horizontalno. Dokumenti su neovisne jedinice što olakšava njihovu distribuciju na više servera.

Dokumentni sustavi podržavaju niz mogućnosti koje olakšavaju upravljanje podacima, poput indeksiranja (za bržu obradu upita), replikacije (čuvanje kopija podataka na više servera), particioniranja (raspodjele podataka na više servera), agregacije (za kombiniranje i transformaciju dokumenata), specijalnih tipova kolekcija i spremišta za datoteke. [17]

Jedan od najznačajnijih predstavnika dokumentnih sustava je MongoDB. On koristi JavaScript notaciju u kojoj bi dokument izgledao ovako:

```
{
      autor: „Dan Brown“,
      knjige: [{ naslov: „Velika obmana“,
                 godina: 2001},
             { naslov; „Da Vincijev kod“,
                       godina: 2003}]
      zanrovi: [„Triler“,
               „Misterija“]
}
```

U ovom dokumentu *autor* je ključ kojem je pridružena vrijednost „Dan Brown“, *knjige* je ključ kojem su pridružena dva dokumenta kojima su *naslov* i *godina* ključevi s pridruženim vrijednostima. JavaScript se u MongoDB koristi i kao jezik za rad s podacima, pa bi upit kojim tražimo autora prezimena Brown izgledao ovako:

```
db.autori.find({prezime: „Brown“})
```

Dodavanje novog dokumenta:

```
db.autori.insert({ime: „Jo“, prezime: „Nesbo“})
```

Brisanje dokumenta:

```
db.autori.remove({prezime: „Nesbo“})
```

Brisanje kolekcije:

```
db.autori.remove()
```

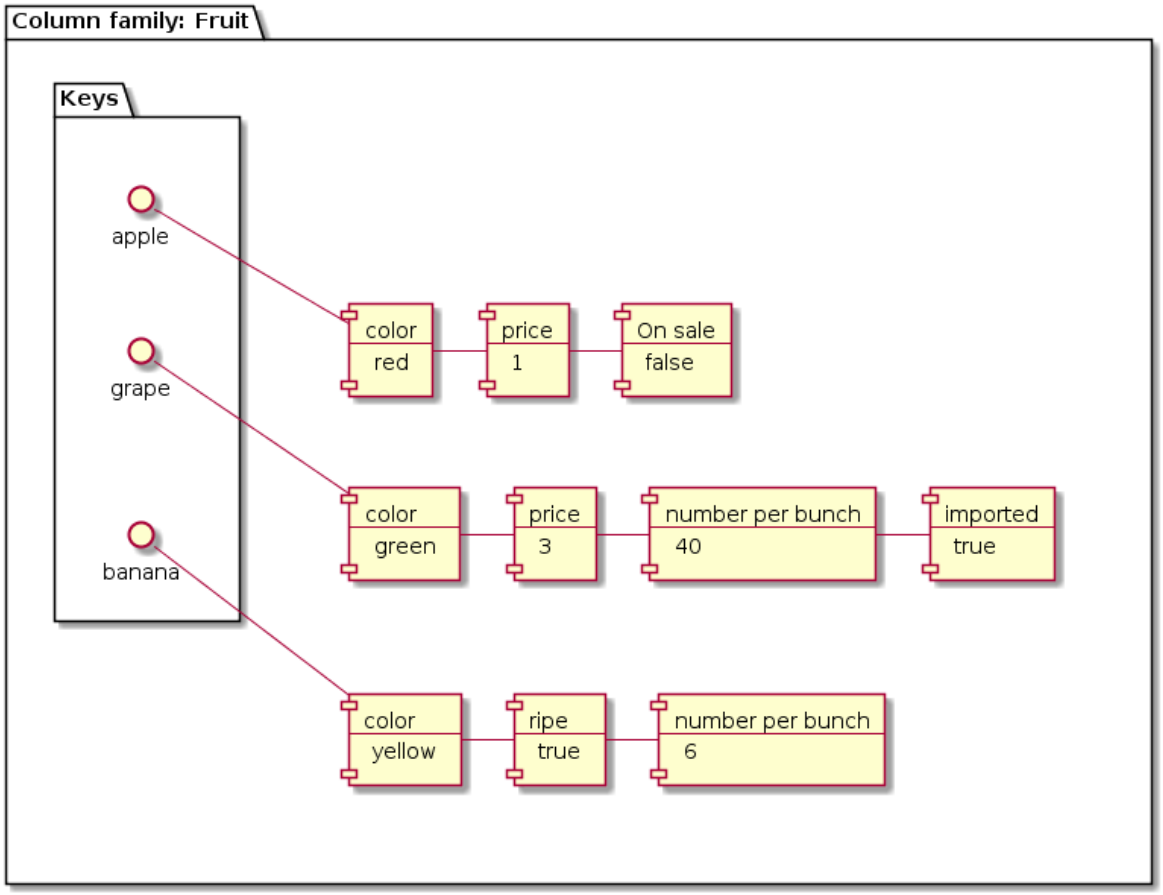
4.5. Stupčaste baze podataka

Kod stupčastih baza podataka, stupci se promatraju slično kao stupci u tablicama, međutim, za razliku od relacijskih baza podataka, u stupčastim bazama podataka može se pristupati pojedinačnim stupcima a da se ne utječe na ostale. U tome leži jedna od glavnih prednosti ovakvih sustava. Iz stupaca se mogu sastaviti i redovi ukoliko je to potrebno, kao kod uobičajenih tablica. Stupčasti sustavi su iznimno efikasni po pitanju memorijskog prostora, jer su vrijednosti koje pripadaju jednom stupcu smještene na jednom cjelovitom prostoru. Ako tablica sadrži više stupaca od kojih se upit odnosi samo na jedan od njih, nije potrebno učitavati cijelu tablicu, nego samo traženi stupac pa se u memoriji zauzima onoliko prostora koliko taj stupac zahtijeva, a ne cijela tablica. [17]

Osnovna prednost stupčastih sustava je ta da su pogodni za obradu jako velikih količina podataka. Učitavanje i modificiranje podataka jednog stupca obavlja se jako efikasno, ali učitavanje svih stupaca u svrhu stvaranja jednog reda i modifikacija tih redova je ipak efikasnija kod relacijskih baza podataka. [17]

Model podataka u stupčastim sustavima je zapravo sličan relacijskom modelu, jer iz stupaca možemo dobiti redove pa često stupčane baze podataka podržavaju relacijski model. Glavna razlika je ta da se kod relacijskih modela radi s redovima, odnosno vrijednosti su grupirane po redovima, a kod stupčastih baza podataka radi se sa stupcima, odnosno vrijednosti su grupirane po stupcima. Model podataka temelji se na djelomično popunjenoj tablici. Stupac se sastoji od naziva i vrijednosti, a stupci se nalaze u redcima te se mogu kombinirati u super-stupce – sortirani niz stupaca. Redak koji se sastoji samo od stupaca zove se porodica stupaca koja se koristi za podatke koji su povezani, a ako se u retku nalaze super-stupci, onda se to naziva porodica super-stupaca. [17]

Stupčasti sustavi mogu se usporediti sa dokumentnim sustavima, ali razlika je u tome što dokumentni sustavi imaju ključ kojim dolazimo do dokumenta i nema pojmova redaka i stupaca, a kod stupčastih sustava ključ se sastoji od retka i stupca. [17]



Slika 18: Primjer modela stupčane baze podataka (Izvor: https://dataguide.prisma.io/intro/comparing-database-types)

Jedan od najznačajnijih predstavnika ovakvih sustava je Cassandra, koja ima i svoj upitni jezik CQL (eng. Cassandra Query Language) koji je jako sličan SQL-u. Za sljedeći primjer korišten je DataStax Astra, koji omogućuje rad u Cassandra sustavu na cloudu, bez potrebe za ikakvim instalacijama.

Kreirane su tri tablice: *autori*, *knjige* i *autori_knjige*. Kod stupčastih baza podataka tablice se ne povezuju kao kod relacijskih baza podataka pomoću vanjskih ključeva. Također ne postoje ni *join* operacije. U ovom slučaju, napravila sam tablice za autore i knjige gdje se nalaze neki podaci o istima, te posebnu tablicu u kojoj se nalaze autori i njihove knjige.

```
CREATE TABLE autori (
    ime text,
    prezime text PRIMARY KEY,
    datum_rodjenja date);

CREATE TABLE knjige (
    isbn text PRIMARY KEY,
    naslov text,
    godina_izdanja int);

CREATE TABLE autori_knjige (
    id uuid PRIMARY KEY,
    autor text,
    knjiga text);
```

Slika 19: Kreiranje tablica

Podaci u tablice unose se kao i u SQL-u. Primarni ključ u tablici *autori_knjige* je tipa *uuid*, koji se može unijeti kao poziv istoimene funkcije koja ne prima nikakve argumente.

```
INSERT INTO autori (ime, prezime, datum_rodjenja) VALUES ('Jo', 'Nesbo', '1960-03-29');
INSERT INTO autori (ime, prezime, datum_rodjenja) VALUES ('Dan', 'Brown', '1964-06-22');

INSERT INTO knjige (isbn, naslov, godina_izdanja) VALUES ('978-3-16-148410-0', 'Crvendac', 2000);
INSERT INTO knjige (isbn, naslov, godina_izdanja) VALUES ('578-3-18-147490-0', 'Velika obmana', 2001);
INSERT INTO knjige (isbn, naslov, godina_izdanja) VALUES ('921-5-16-649412-0', 'Da Vincijev kod', 2003);

INSERT INTO autori_knjige (id, autor, knjiga) VALUES (uuid(), 'Jo Nesbo', 'Crvendac');
INSERT INTO autori_knjige (id, autor, knjiga) VALUES (uuid(), 'Dan Brown', 'Velika obmana');
INSERT INTO autori_knjige (id, autor, knjiga) VALUES (uuid(), 'Dan Brown', 'Da Vincijev kod');
```

Slika 20: Unos podataka u tablice

Upiti se također izvršavaju kao i u SQL-u. Želimo li podatke o naslovu i godini izdanja svih knjiga, upit je sljedeći:

```
SELECT naslov, godina_izdanja FROM knjige;
```

index ▲	naslov	godina_izdanja
0	Da Vincijev kod	2003
1	Velika obmana	2001
2	Crvendac	2000

Slika 21: Upit koji vraća podatke o knjigama

Želimo li pronaći sve knjige koje je napisao Dan Brown, koristimo klauzulu WHERE kao i u SQL-u, ali moramo još dodati ALLOW FILTERING jer bez toga upit neće raditi zato što je moguć nepredvidiv ishod, na što nas sustav upozorava.

```
SELECT autor, knjiga FROM autori_knjige WHERE autor = 'Dan Brown' ALLOW FILTERING;
```

index ▲	autor	knjiga
0	Dan Brown	Velika obmana
1	Dan Brown	Da Vincijev kod

Slika 22: Upit koji vraća knjige određenog autora

5. Prednosti i nedostaci NoSQL baza podataka

Iako su NoSQL sustavi sve popularniji i korisni su za obradu velike količine podataka, važno je naglasiti koje su prednosti, a koji nedostaci istih u odnosu na tradicionalne relacijske baze podataka, kao i usporediti performanse jednih i drugih.

5.1. Prednosti NoSQL baza podataka

Jedna od najvećih prednosti NoSQL baza podataka je sposobnost pohrane i obrade jako velike količine podataka pri velikoj brzini. One su skalirane horizontalno, što je prednost u odnosu na relacijske baze podataka koje su skalirane vertikalno pa zahtijevaju puno memorije i procesora kako bi performanse bile na razini. S druge strane, NoSQL baze zaživjele su u novije vrijeme i vrijeme Interneta pa postižu skalabilnost tako što ne opterećuju jedno računalo, nego podatke raspodijele na više servera i dodaju nove ako je potrebno. To je vrlo korisno kod pohrane velike količine podataka. Relacijske baze podataka teže postižu tu razinu skalabilnosti pri čemu su troškovi veliki. [21]

Dok se u relacijskim bazama podataka model mora dizajnirati prije nego se po istom tom modeli podaci pohranjuju u bazu podataka, odnosno obavezna je predefinirana shema podataka, kod NoSQL-a to nije slučaj. Nerelacijske baze podataka dopuštaju 'slobodnije' načine pohrane podataka koje je lakše razumjeti i koji su bliži načinima na koje aplikacija te podatke i koristi. S druge strane, kod relacijskih baza podataka, podaci pohranjeni u obliku tablica dohvate se koristeći SQL pa se moraju transformirati u oblik podataka kojeg aplikacija koristi. Bez obzira jesu li podaci strukturirani ili nestrukturirani, kod NoSQL baza lakše se pohranjuju i dohvaćaju. [21]

NoSQL baze podataka sve su popularnije jer pohranjuju podatke na intuitivan način i model je lako prilagoditi promjenama i modificirati. Primjerice, kod dokument baza podataka propisana struktura podataka kao takva uopće ne postoji, pa se vrlo lako dodaju novi dokumenti i vrijednosti. Kod stupčanih i ključ-vrijednost sustava, novi stupci/vrijednosti dodaju se lako bez utjecaja na trenutnu strukturu, a kod graf baza novi čvor sa sasvim novim svojstvima i vezama jednostavno se doda na graf ili se promijene postojeći bez negativnog utjecaja. [21]

NoSQL baze podataka jako su zaživjele među developerima, jer imaju više kontrole nad samom strukturom podataka nego što je to slučaj kod relacijskih baza. Recimo, dokument baze podataka koriste JSON kako bi rad s podacima približili samom kodiranju. Nadalje, budući da se podaci pohranjuju više-manje onakvi kakvi se u aplikaciji i koriste, nije potrebno

raditi dodatne transformacije oblika podataka kad su pohranjeni u bazi i kad ih aplikacija koristi. To uvelike olakšava posao developerima. [21]

Ponajviše zbog sposobnosti pohrane velike količine podataka i njihove brze obrade, NoSQL baze podataka jako su popularne kod razvoja društvenih mreža (Facebook), Internet kupovine (Amazon) i sl.

5.2. Nedostaci NoSQL baza podataka

Bitan nedostatak kojeg imaju NoSQL baze podataka je nepostojeći standardni upitni jezik kao što je slučaj sa relacijskim bazama i SQL-om. Iako svaka nerelacijska baza podataka ima vlastiti upitni jezik ili operacije kojima manipulira podacima, standardni upitni jezik uvelike bi olakšao rad s ovakvim sustavima jer bi za svaki vrijedila ista pravila kod manipuliranja s podacima. Stoga relacijske baze podataka imaju prednost na tom području. [22]

Također se kao nedostatak javlja sigurnost, odnosno potencijalna nesigurnost podataka. Iako horizontalna skalabilnost jača performanse, u pitanje se dovodi sigurnost i privatnost podataka koji su razmješteni na više servera. Većina NoSQL baza podataka nema sigurnu klijent-server komunikaciju koja će garantirati apsolutnu sigurnost podataka. S druge strane, relacijske baze podataka lako postižu sigurnost podataka jer su oni iznimno strukturirani. Kod NoSQL baza podataka pohranjuju se i nestrukturirani podaci, i to na više servera pa je to teže postići te baza podataka može postati nesigurna. Na tom polju NoSQL tehnologije zasigurno imaju prostora za poboljšanje. [22]

Nadalje, dok je jedna od odlika relacijskih baza podataka najmanja redundancija budući da su podaci strukturirani, normalizirani i jasno modelirani, kod NoSQL baza podataka može se javiti problem redundancije jer su podaci smješteni u kolekcije bez jasno definiranih veza i normalizacije. [22]

5.3. Usporedba performansi

NoSQL sustavi jako brzo obrađuju informacije, dok je relacijskim bazama podataka potrebno više vremena. Eksperimenti su pokazali da MongoDB ima bolju izvedbu čitanja i promjene podataka te osnovnih upita nego relacijske baze podataka. [23] Slijedi tablica sa usporedbom performansi pojedinih sustava u milisekundama prilikom izvršavanja osnovnih upita u bazama sa 10 000 zapisa.

Tablica 2: Peformanse baza sa 10 000 zapisa

Operacija	Oracle	MySQL	Mongo	Redis	Cassandra
Insert	0.076	0.093	0.005	0.009	0.011
Update	0.077	0.058	0.008	0.013	0.013
Delete	0.059	0.025	0.01	0.021	0.018
Select	0.025	0.093	0.009	0.016	0.014

(Izvor: Čerešnak i Kvet, 2019.)

Vidi se kako NoSQL sustavi uglavnom brže izvode operacije nego sustavi za relacijske baze podataka. Njihova brzina ne smanjuje se ni nakon što se doda još 90 000 zapisa, dok je, primjerice Oracleu potrebno više milisekundi da obradi operacije u bazi podataka sa 100 000 zapisa, što je vidljivo u sljedećoj tablici.

Tablica 3: Performanse baza sa 100 000 zapisa

Operacija	Oracle	MySQL	Mongo	Redis	Cassandra
Insert	0.091	0.038	0.005	0.010	0.011
Update	0.092	0.068	0.009	0.013	0.014
Delete	0.119	0.045	0.015	0.021	0.019
Select	0.062	0.067	0.009	0.015	0.014

(Izvor: Čerešnak i Kvet, 2019.)

Razlog zbog kojeg su rezultati ovakvi je način pohrane podataka. U relacijskim bazama podataka podaci su pohranjeni tako da se izbjegne njihovo dupliciranje. Stoga se normaliziraju i dijele u više tablica, a sve to koči procesiranje podataka i izvođenje operacija pa je zato potrebno više vremena. S druge strane, u NoSQL sustavima podaci se dupliciraju, podatak se ne dijeli, odnosno ne particionira, nego se pohranjuje u obliku entiteta, a operacije se brže izvode nad jednim entitetom i lakše mu je pristupiti. [24]

5.4. Usporedba upitnih jezika

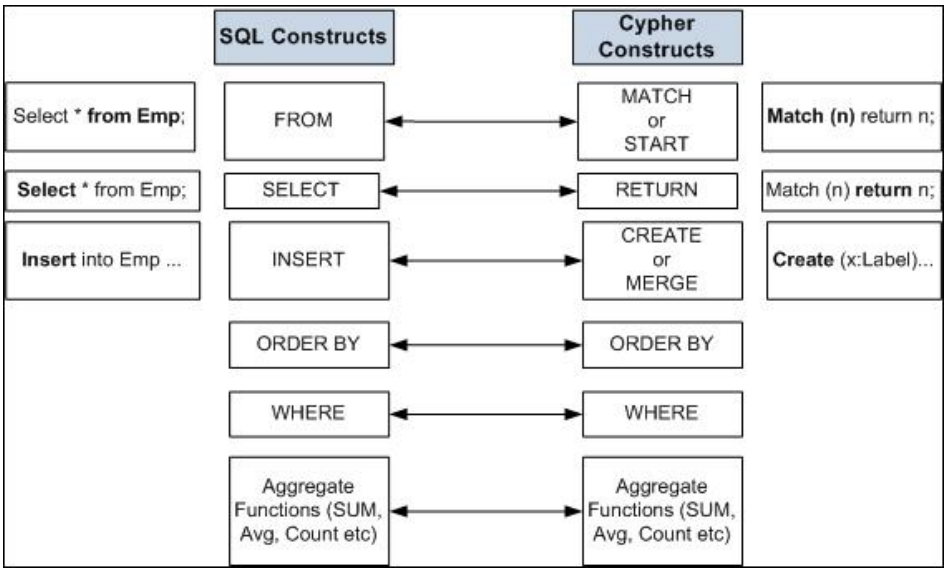
Relacijske baze podataka koriste SQL kao standardni upitni jezik, dok NoSQL sustavi, barem zasad, nemaju standardni upitni jezik, već je on različit ovisno o sustavu koji se koristi. Neo4j koji se koristi za implementaciju grafovskih baza podataka koristi upitni jezik Cypher, kako ja navedeno i demonstrirano ranije. Također je već spomenuto kako NoSQL sustavi ne koriste operacije spajanja, dok su one jako korištene i važne u SQL upitnom jeziku kako bismo dohvatili podatke iz više tablica. Na sljedećoj slici prikazan je isti upit napisan u SQL-u i u Cypheru, gdje je očito da je Cypher upit kraći i jednostavniji nego SQL upit.

```
select s.code
,      s.title
,      sp.speaker_name
,      a2.attendee_name "suggested by"
from   people p1
       join
       attendance a1
       on (p1.name = a1.attendee_name)
       join attendance a2
       on (a2.session_code = a1.session_code)
       join speaker_liking sl
       on (sl.attendee_name = a2.attendee_name)
       join speakers sp
       on (sl.speaker_name = sp.speaker_name)
       join sessions s
       on (sp.session_code = s.code)
where  p1.name = 'Lucas Jellema'
```

```
match (lucas:Person {name:'Lucas Jellema'})
  - [:ATTENDS] -> (session)
  <- [:ATTENDS] - (recommender)
  - [:VALUES] -> (speaker)
  - [:PRESENTS] -> (recommended_session)
RETURN recommended_session
,      speaker.name
,      recommender.name
```

Slika 23: Usporedba SQL i Cypher upita (Izvor: https://technology.amis.nl/2018/11/20/building-a-conference-session-recommendation-engine-using-neo4j-graph-database/)

SQL i Cypher nisu toliko različiti što se tiče nekih klauzula pa se nije teško snaći u Cypheru ako već razumijemo SQL.



Slika 24: Usporedba sintakse SQL-a i Cyphera (Izvor: https://www.packtpub.com/product/building-web-applications-with-python-and-neo4j/9781783983988)

Vidimo da su neke klauzule potpuno jednake, kao i agregirajuće funkcije.

MongoDB koristi JavaScript notaciju za izvršavanje upita, pa usporedba osnovnih upita u MongoDB u odnosu na MySQL izgleda ovako:

MySQL	MongoDB
INSERT	
INSERT INTO account (`A/c number`, `first name`, `last name`) VALUES ('12345746352', 'Mark', 'Jacobs');	db.account.insert({ A/c number: "12345746352", first name: "Mark", last name: "Jacobs" });
UPDATE	
UPDATE account SET contact number = 9426227364 WHERE A/c number = '12345746352'	db.account.update({ A/c number: '12345746352' }, { $set: {contact number: 9426227364 } });
DELETE	
DELETE FROM account WHERE e-mail address = 'jv1994@gmail.com';	db.account.remove({ "E-mail address": "jv1994@gmail.com" });

Slika 25: Usporedba upita u MySQL-u i MongoDB-u (Izvor: https://www.simform.com/mongodb-vs-mysql-databases/)

6. Primjer aplikacije za knjižnicu

Aplikacija je napravljena kao Windows Forms aplikacija u Visual Studio razvojnoj okolini i programskom jeziku C#, a baza podataka napravljena je u Neo4j sustavu. Za izradu aplikacije potrebno je instalirati Neo4jClient i Neo4jDriver kao pakete za Visual Studio kako bi se mogli povezati na bazu i pisati Cypher upite u C#.
Baza podataka sadrži čvorove *Autor*, *Knjiga*, *Zanr* i *Clan*, a veze su *NAPISAO*, *PRIPADA* i *POSUDIO*.

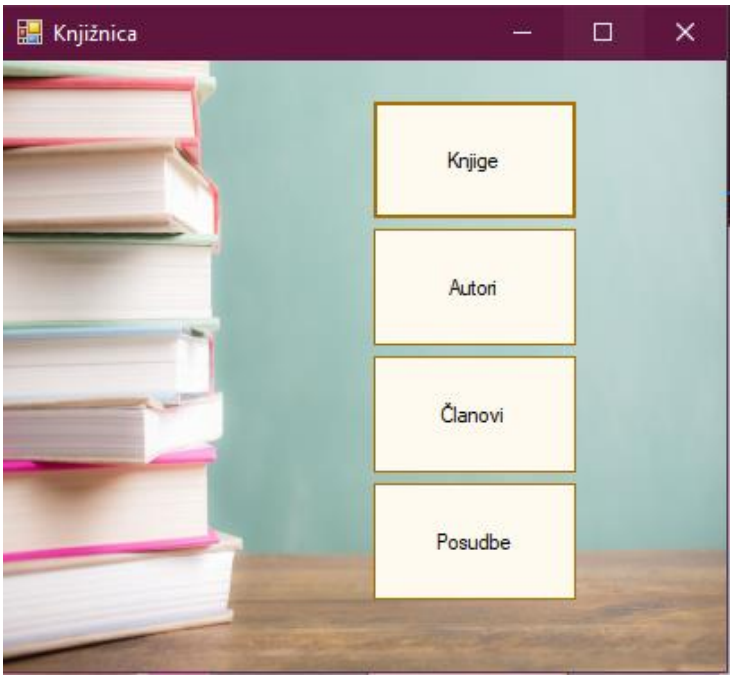
Povezivanje na bazu implementirano je na sljedeći način:

```
IDriver driver = GraphDatabase.Driver("bolt://localhost:7687",
    AuthTokens.Basic("neo4j", "katarina123"),
    Config.Builder.WithEncryptionLevel(EncryptionLevel.None).ToConfig());

BoltGraphClient client;
client = new BoltGraphClient(driver);
client.Connect();
```

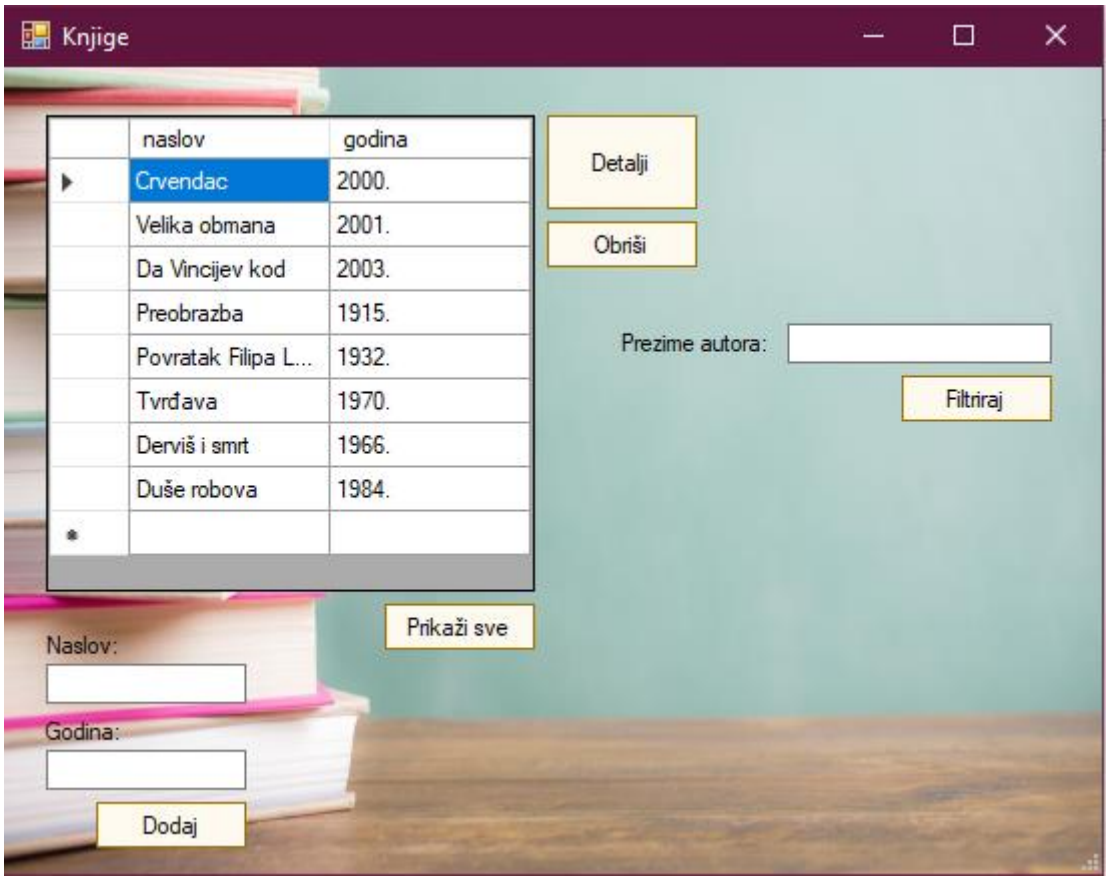
Slika 26: Povezivanje na bazu

Početni zaslon aplikacije ima četiri gumba gdje se može odabrati želimo li manipulirati podacima o knjigama, autorima, članovima ili posudbama.



Slika 27: Početni zaslon aplikacije

Odaberemo li knjige, otvara se forma sa popisom svih knjiga na kojoj se mogu dodati nove knjige, obrisati knjige, filtrirati prema prezimenu autora ili vidjeti detalji o knjizi.



Slika 28: Forma za knjige

Za prikaz svih knjiga poziva se metoda *DohvatiSveKnjige*.

```
public IEnumerable<Knjiga> DohvatiSveKnjige()
{
    return client.Cypher.Match("(k:Knjiga)")
        .Return(k => k.As<Knjiga>())
        .Results.ToList();
}
```

Slika 29: Metoda za dohvaćanje svih knjiga

Klikom na gumb Dodaj poziva se metoda *DodajKnjigu* koja sadrži Cypher upit kojim se dodaje novi čvor tipa *Knjiga* u bazu podataka, koristeći naredbu MERGE.

```
public Knjiga DodajKnjigu(Knjiga knjiga)
{
    return client.Cypher.Merge
        ("(k:Knjiga{naslov: '" + knjiga.naslov + "', godina: '" + knjiga.godina + "'})")
        .Return(k => k.As<Knjiga>()).Results.SingleOrDefault();

}
```

Slika 30: Metoda za dodavanje knjige

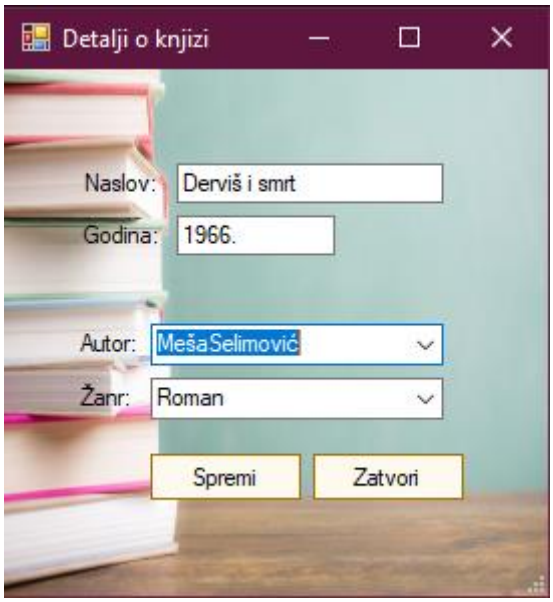
Želimo li obrisati knjigu iz baze, klikom na gumb Obriši poziva se metoda *ObrisiKnjigu*.

```
public void ObrisiKnjigu(Knjiga knjiga)
{
    client.Cypher.Match("(k:Knjiga {naslov: '" + knjiga.naslov + "'})")
        .Delete("k")
        .ExecuteWithoutResults();
}
```

Slika 31: Metoda za brisanje knjiga

Na isti način su ove metode implementirane i za čvorove *Autor* i *Clan*.

Otvorimo li formu s detaljima o nekoj knjizi, možemo joj dodijeliti autora i žanr iz padajućeg izbornika, za što su napravljene metode *DodijeliZanrKnjizi* i *DodijeliAutoraKnjizi* u kojima se kreira pripadajuća veza između čvorova *Knjiga* i *Zanr*, odnosno *Knjiga* i *Autor*.



Slika 32: Forma za detalje o knjizi

Klikom na gumb Spremi, pozivaju se gore navedene metode.

```
public void DodijeliZanrKnjizi(Knjiga knjiga, Zanr zanr)
{
    client.Cypher.Match("(k:Knjiga)", "(z:Zanr)")
        .Where((Knjiga k) => k.naslov == knjiga.naslov)
        .AndWhere((Zanr z) => z.naziv == zanr.naziv)
        .Create("(k)-[:PRIPADA]->(z)")
        .ExecuteWithoutResults();
}
```

Slika 33: Metoda za dodjeljivanje žanra

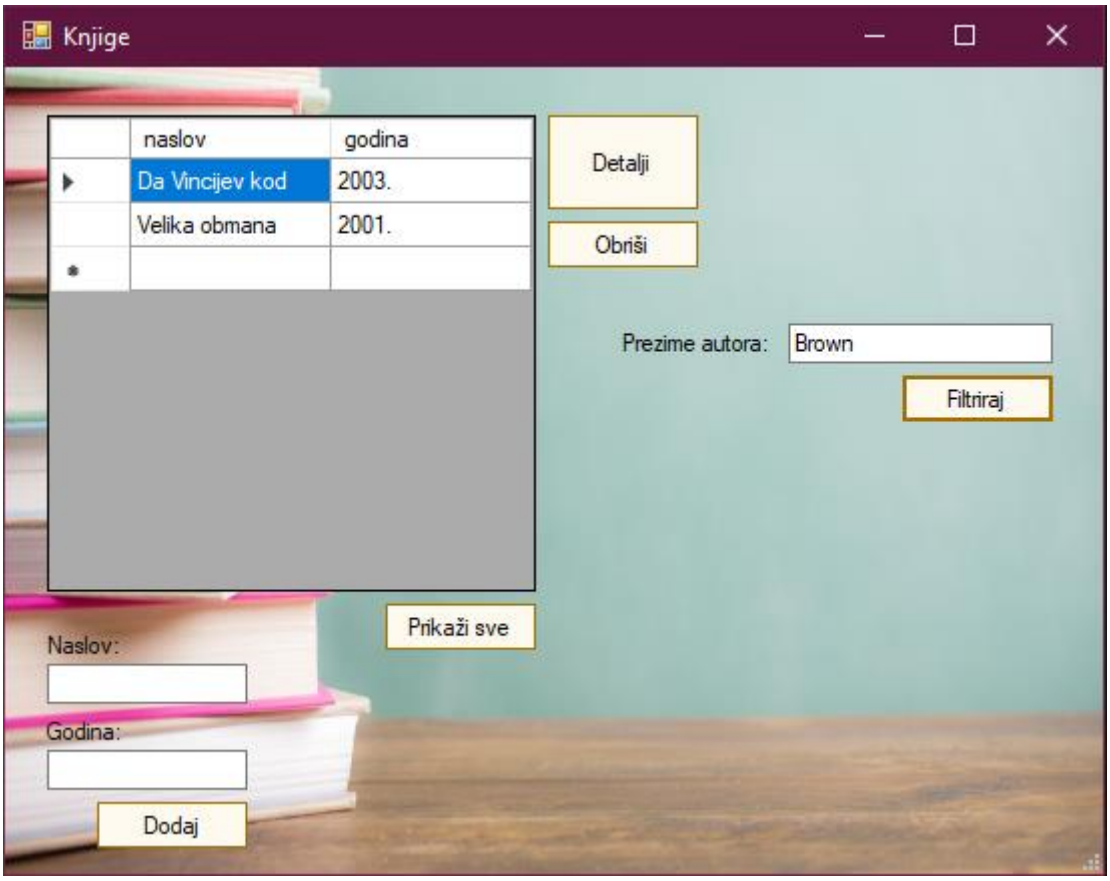
```
public void DodijeliAutoraKnjizi(Knjiga knjiga, Autor autor)
{
    client.Cypher.Match("(k:Knjiga)", "(a:Autor)")
        .Where((Knjiga k) => k.naslov == knjiga.naslov)
        .AndWhere((Autor a) => a.prezime == autor.prezime)
        .Create("(a)-[:NAPISAO]->(k)")
        .ExecuteWithoutResults();
}
```

Slika 34: Metoda za dodjeljivanje autora

Želimo li pretražiti knjigu prema prezimenu autora, klikom na gumb Filtriraj poziva se metoda *DohvatiKnjigePremaAutoru* u kojoj pomoću Cypher upita i klauzule MATCH pronađemo čvorove između kojih postoji veza *NAPISAO* preko unesenog prezimena autora i te knjige vratimo kao listu tipa *Knjiga*.

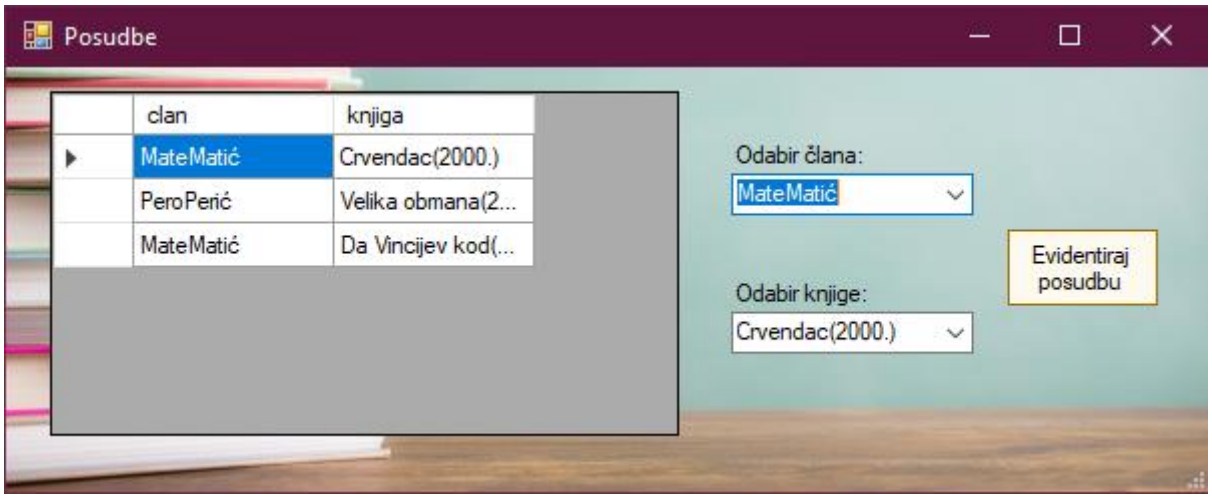
```
public List<Knjiga> DohvatiKnjigePremaAutoru(Autor autor)
{
    return
        client.Cypher.Match("(a:Autor)-[:NAPISAO]->(k:Knjiga)")
        .Where((Autor a) => a.prezime == autor.prezime)
        .Return(k => k.As<Knjiga>())
        .Results.ToList();
}
```

Slika 35: Metoda za pretraživanje knjiga prema autoru



Slika 36: Pretraživanje prema autoru

U formi *Posudbe* prikazani su članovi i knjige koje su posudili, a posudba se evidentira tako što se iz padajućeg izbornika odabere član i knjiga koju će posuditi.



Slika 37: Forma za posudbe

Za prikaz posudbi napravljen je malo drugačiji upit nego za dohvaćanje podataka u drugim formama, budući da moramo prikazati podatke koji nisu tipa samo jednog čvora, već dva.

```
var query = client.Cypher
    .Match("(c:Clan)-[:POSUDIO]->(k:Knjiga)")
    .Return((c, k) => new
    {
        clan = c.As<Clan>(),
        knjiga = k.As<Knjiga>()
    });

var results = query.Results.ToList();
```

Slika 38: Upit za prikaz posudbi

Za evidentiranje nove posudbe, napravljena je metoda *EvidentirajPosudbu* u kojoj je upit kojim se kreira nova veza tipa *POSUDIO* između čvorova *Clan* i *Knjiga*.

```
public void EvidentirajPosudbu(Knjiga knjiga, Clan clan)
{
    client.Cypher.Match("(k:Knjiga)", "(c:Clan)")
        .Where((Knjiga k) => k.naslov == knjiga.naslov)
        .AndWhere((Clan c) => c.prezime == clan.prezime)
        .Create("(c)-[:POSUDIO]->(k)")
        .ExecuteWithoutResults();
}
```

Slika 39: Metoda za evidentiranje posudbe

7. Zaključak

Iako popularnost NoSQL sustava raste, to ne znači da će relacijske baze podataka izaći iz upotrebe. Ne postoji odgovor na pitanje koji od ova dva pristupa je bolji, jer to ovisi o potrebama sustava čije podatke baza podataka treba pohraniti i obraditi. NoSQL sustavi su efikasniji kad se pohranjuju i obrađuju velike količine podataka zbog horizontalne skalabilnosti i distribuiranosti. Baza podataka se lako može modificirati zbog fleksibilne sheme podataka. S druge strane, relacijske baze podataka su kvalitetnije po pitanju sigurnosti podataka i imaju standardni upitni jezik koji vrijedi za sve relacijske baze podataka što olakšava upotrebu i razumijevanje istih.

I jedan i drugi pristup imaju kvaliteta i prostora za poboljšanje. Idealno rješenje bilo bi osmisliti pristup koji objedinjuje relacijske i NoSQL baze podataka uzimajući prednosti i svodeći nedostatke na minimum kako bi se maksimizirala efikasnost pohrane i obrade podataka.

Pojava relacijskog pristupa modeliranju podataka bila je velik uspjeh i olakšala je prikupljanje, pohranu i obradu informacija. NoSQL sustavi imaju potencijal da načine još jedan veliki korak u tom polju, ali ne tako da izbace već ukorijenjen i provjeren pristup iz uporabe već da ga nadopune i poboljšaju.

Popis literature

[1] M. Maleković i K. Rabuzin, Uvod u baze podataka, Varaždin: Fakultet organizacije i informatike, 2016.

[2] R.Manger, Baze podataka, Zagreb: Element, 2012. [Na internetu] Dostupno: https://element.hr/artikli/file/1710/baze-podataka/13576 [Pristupano 4.8.2020.]

[3] "Relacijska baza podataka", (bez dat.) u Wikipedia, the Free Encyclopedia. Dostupno: https://hr.wikipedia.org/wiki/Relacijska_baza_podataka [Pristupano 4.8.2020.]

[4] "Relational database", (bez dat.) u Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/Relational_database [Pristupano 4.8.2020.]

[5] Jan L. Harrington, Relational Database Design and Implementation, Burlington, MA, USA: Morgan Kauffman. 2009.

[6] "Entity-relationship model", (bez dat.) u Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model [Pristupano 5.8.2020.]

[7] P. Pickett, "What Is SQL?", 2020. [Na internetu]. Dostupno: https://www.thebalancecareers.com/what-is-sql-and-uses-2071909 [Pristupano 20.8.2020.]

[8] "SQL Notes for Proffesionals", (bez dat.) [Na internetu]. Dostupno: https://books.goalkicker.com/SQLBook/ [Pristupano 20.8.2020.]

[9] „Introduction to Oracle Database“, (bez dat.) [Na internetu]. Dostupno: https://docs.oracle.com/cd/E11882_01/server.112/e40540/intro.htm#CNCPT001 [Pristupano 1.9.2020]

[10] „What is Oracle Database“, (bez dat.) [Na internetu]. Dostupno: https://www.oracletutorial.com/getting-started/what-is-oracle-database/ [Pristupano 1.9.2020.]

[11] „What is MySQL?“, (bez dat.) [Na internetu]. Dostupno: https://www.atlantic.net/dedicated-server-hosting/what-is-mysql/ [Pristupano 2.9.2020]

[12] M. Rouse, „MySQL“, (bez dat.) [Na internetu]. Dostupno: https://searchoracle.techtarget.com/definition/MySQL [Pristupano 2.9.2020.]

[13] „What is SQL Server“, (bez dat.) [Na internetu]. Dostupno: https://www.sqlservertutorial.net/getting-started/what-is-sql-server/ [Pristupano 2.9.2020.]

[14] „MS SQL Server – Overview“, (bez dat.) [Na internetu]. Dostupno: https://www.tutorialspoint.com/ms_sql_server/ms_sql_server_overview.htm [Pristupano 2.9.2020.]

[15] „About PostgreSQL“, (bez dat.) [Na internetu]. Dostupno: https://www.postgresql.org/about/ [Pristupano 2.9.2020.]

[16] "NoSQL" (26.2.2020.) [Na internetu]. Dostupno: https://www.w3resource.com/mongodb/nosql.php [Pristupano 5.8.2020.]

[17] A. Stojanović, "Osvrt na NoSQL baze podataka - Četiri osnovne tehnologije", Polytechnic & Design, 2016. [Na internetu]. Dostupno: Hrcak, https://hrcak.srce.hr/. [Pristupano 9.6.2020.]

[18] A. Sharma, „When to use Redis Database and Why“, 2018. [Na internetu]. Dostupno: https://www.dignitasdigital.com/blog/when-to-use-redis-database/ [Pristupano 2.9.2020.]

[19] "What is a graph database?", (bez dat.) [Na internetu]. Dostupno: https://neo4j.com/developer/graph-database/ [Pristupano 17.8.2020.]

[20] "What is a Document Database?", (bez dat.) [Na internetu]. Dostupno: https://www.mongodb.com/document-databases [Pristupano 17.8.2020.]

[21] "Advantages of NoSQL Databases", (bez dat.) [Na internetu]. Dostupno: https://www.mongodb.com/nosql-explained/advantages [Pristupano 20.8.2020.]

[22] K. Sahatqija, J. Ajdari, X. Zenuni, B. Raufi, F. Ismaili, "Comparison between relational and NOSQL databases", South East European University, Tetovo, Macedonia. 2018. [Na internetu]. Dostupno: https://www.researchgate.net/publication/326699854_Comparison_between_relational_and_NOSQL_databases [Pristupano 20.8.2020.]

[23] D. Kunda, H. Phiri, „A Comparative Study of NoSQL and Relational Database“, Zambia (ICT) Journal, 2017. [Na internetu] Dostupno: https://ictjournal.icict.org.zm/index.php/zictjournal/article/view/8 [Pristupano 3.9.2020.]

[24] R. Čerešnak, M. Kvet, „Comparison of query performance in relational a non-relational databases“, University of Zilina, 2019. [Na internetu] Dostupno: https://www.sciencedirect.com/ [Pristupano 3.9.2020.]

Popis slika

Popis tablica