# NoSQL baza podataka

**Janjić, Vedran**





**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU**

**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA**

**Sveučilišni diplomski studij Računarstva**

# *NOSQL* BAZA PODATAKA

**Diplomski rad**

**Vedran Janjić**

**Osijek, 2018.**

# Sadržaj

# 1. UVOD

U današnjoj svakodnevnici ljudi su okruženi podacima, odnosno informacijama kao nikada prije te ih na razne načine pokušavaju organizirati na način na koji bi ih kasnije mogli što lakše iskoristiti. Potrebno je znati koje su informacije od koristi i trajno ih pohraniti na što sigurniji način te ih brzo i precizno dohvatiti onda kada te informacije zatrebaju.

Ovaj rad upravo govori o jednom od sustava koji se koristi za rad s podacima. Taj sustav se zove *NoSQL* baza podataka. *NoSQL* baze podataka predstavljaju relativno novu tehnologiju u radu s podacima te se razlikuju od tradicionalnih *SQL* baza podataka o čemu će se više govoriti u nastavku rada.

Cilj ovog rada je detaljno objasniti *NoSQL* baze podataka i njihove performanse te prikazati što *NoSQL* baze podataka predstavljaju u današnjem svijetu po pitanju organizacije podataka. Također je želja prikazati upotrebu *NoSQL* baza podataka u svakodnevnici.

U radu će se prvo ukratko objasniti što su to relacijske, odnosno *SQL* baze podataka. Zatim će kratko biti opisana povijest i sadašnjost *NoSQL* baza podataka i njihova podjela. U nastavku će biti objašnjene neke od najpoznatijih *NoSQL* baza podataka kao što su *Neo4j, MongoDB, Redis* te će biti prikazana detaljna usporedba *MongoDB* i *Oracle* relacijske baze podataka te će biti istaknute prednosti i nedostatci *NoSQL* baza podataka.

## 1.1. ZADATAK DIPLOMSKOG RADA

Napraviti detaljan opis i razvrstavanje *NoSQL* baza podataka. Usporediti performanse, funkcionalnost i brzinu *NoSQL* baze podataka s klasičnim relacijskim bazama podataka (primjerice *MySQL*), te istaknuti njene prednosti i mane.

# 2. BAZA PODATAKA

U ovom poglavlju će se govoriti o tome što je to *NoSQL* (engl. „non-SQL“, „non-relational“ ili „not only SQL“) baza podataka te o njenim tipovima i performansama. No prije toga potrebno je objasniti što su to baze podataka i čemu one služe te što je to *SQL (engl. Structured query language)* baza podataka.

„Baza podataka je organizirana zbirka podataka pohranjena na sustavan način tako da računalni program može poslati upit bazi podataka na koji ona odgovara. Baze podataka služe za bolju dostupnost i razvrstavanje podataka te mogu sadržavati podatke u raznim oblicima, od jednostavnog teksta do kompleksnih struktura koje uključuju sliku, zvuk itd. [1]. “

„Baza podataka nije uobičajeno prijenosna u različite sustave za upravljanje bazom podataka (*SUBP*), ali različiti *SUBP*-ovi mogu međusobno djelovati korištenjem standarda kao što su *SQL*, *ODBC* [1] *(engl. Open Database Connectivity)* ili *JDBC* [2] (engl. Java Database Connectivity) *API* [3]-ja (engl. Application Programming Interface) i tako omogućuju da jedna aplikacija surađuje s više *SUBP*-ova [2]. “

U svijetu tehnologija za rad s bazama podataka postoje dva glavna tipa baza podataka, a to su: *SQL* i *NoSQL*, odnosno postoje relacijske i nerelacijske baze podataka. Razlika je u tome kako su građene, tipu podataka koje pohranjuju te kako ih pohranjuju. Relacijske baze su strukturirane kao telefonski imenik koji sadrži brojeve i adrese korisnika. Nerelacijske baze su više kao dokument, odnosno kao mape datoteka koje sadrže sve podatke o osobi od njegove adrese i broja telefona pa do toga što ta osoba voli kupovati, jesti itd. [3].

[1] Standard za pristupanje bazi podataka.
[2] Standard za pristupanje bazi podataka za Java programski jezik.
[3] Set protokola, alata i definicija za izradu aplikacija.

## 2.1. Relacijske baze podataka

Jedno od glavnih obilježja koje odvaja relacijske od nerelacijskih baza je način na koji strukturiraju podatke. Relacijske baze ili *SQL* baze podataka spremaju podatke na strog i strukturiran način poput telefonskog imenika. Sastoji se od dvije ili više tablica sa stupcima i redcima. Svaki red predstavlja ulaz, a svaki stupac predstavlja određenu vrstu podatka kao što je npr. ime, adresa, broj telefona. Relacija između tablica i vrsta polja se naziva shema. Shema mora biti jasno definirana prije nego se što se krenu unositi ikakvi podaci [4].

Da bi relacijska baza bila učinkovita, podaci koji se spremaju moraju biti strukturirani na vrlo organiziran način. Dobro dizajnirana shema smanjuje redundanciju i sprječava nesklad tablica što je jako važno svojstvo za mnoge poslove, posebice one koji bilježe financijske transakcije. Loše dizajnirana shema može uzrokovati velike probleme. Npr. ukoliko postoji stupac u tablici dizajniran za unos matičnog broja građana te zahtjeva 13 znamenaka to može spriječiti unos matičnog borja manjeg od 13 znamenaka, ali ukoliko nekada dođe do zahtijeva za promjenom sheme tada je potrebno mijenjati čitavu bazu podataka [4].

Programski jezik koji se koristi za dizajn relacijskih baza podataka se zove *SQL*, odnosno strukturirani jezik upita. U *SQL* bazama podataka kao što su *MySQL, Oracle* ili *Sybase, SQL* izvršava upite, dohvaća podatke i izmjenjuje ih tako što ažurira, briše ili dodaje nove zapise. Prednost *SQL*-a je ta što programeri mogu dohvaćati podatke spremljene u više tablica pomoću samo jedne naredbe, a to je naredba *JOIN* (engl. spojiti) [4].

# 3. NOSQL (NERELACIJSKE) BAZE PODATAKA

## 3.1. Kratka povijest NoSQL baza podataka

Pojam *NoSQL* baza podataka se prvi put pojavio 1998. godine od strane Carla Strozzija na sastanku u San Franciscu kao ime za *open-source* [4] relacijsku bazu „*Strozzi NoSQL*“ koja je tablice spremala kao *ASCII* datoteke. Termin *NoSQL* se koristi zato što se bazi podataka nije pristupalo koristeći *SQL* upite nego se bazom upravljalo preko *shell* skripta [5]. Korištenje ovog termina također predstavlja frustraciju prema zajednici koja je koristila *SQL*. Želja je bila naći bolji način rada s bazama podataka bez čitanja složenih *SQL* upita. Nakon sastanka u San Franciscu programeri su nastavili s traženjem alternativa za rad s bazama podataka. Jedna je bila *Java* biblioteka, koja je omogućila programerima jednostavno pozivanje funkcija i objekata bez prethodnog znanja o strukturi baze podataka. S obzirom da složeni *SQL* upiti predstavljaju problem u performansama rada i da to stvara dodatan trošak i potrebu za dodatnim resursima te da se relacijske baze ne mogu nositi s naglim rastom Interneta, pronalazak alternativa tj. biblioteka koje će smanjiti kompleksnost se čini kao dobro rješenje [5].

## 3.2. NoSQL baze podataka danas

Za razliku od relacijskih baza podataka koje pohranjuju podatke putem tabličnih relacija, nerelacijske baze pohranjuju i dohvaćaju podatke na drukčiji način. Umjesto tablica, *NoSQL* baze podataka su dokument orijentirane tj. rade s dokumentima umjesto sa strogo definiranim tablicama podataka. Baze usmjerene na dokumente prikupljaju podatke iz dokumenata i omogućavaju njihovo dohvaćanje u pretražujućem i organiziranom obliku. Na ovaj način nestrukturirani podaci kao što su slike, videozapisi ili novinski članci mogu biti spremljeni u jedan dokument koji se lako može pronaći bez kategorizacije polja kao u relacijskim bazama. Ovakva organizacija podataka zahtijeva dodatne napore obrade te veći prostor za pohranu nego što je to kod visoko organiziranih *SQL* podataka, ali zato omogućavaju bržu obradu određenih zahtjeva.

Prilikom pohrane podataka u *NoSQL* bazu nije potreban veliki prethodan rad na dizajnu. Moguće je početi s kodiranjem, pohranom i dohvaćanjem podataka bez prethodnog znanja kako baza

[4] To je softver čiji je izvorni kod dostupan javnosti, izvor: [https://hr.wikipedia.org/wiki/Otvoreni_kod]
[5] To je kompjutorski program pokretan od strane Unix operacijskog sustava, odnosno preko njegovg naredbenog retka.

sprema podatke te kako ona radi. Prethodno nepoznavanje sheme je možda i jedna od najznačajnijih razlika između nerelacijskih i relacijskih baza podataka [5].

Ovakav pristup omogućava jednostavnost dizajna, jednostavno vodoravno skaliranje te lako podnosi značajne promjene sheme. [4].

## 3.3. Vrste nerelacijskih baza podataka

Postoji više vrsta nerelacijskih baza podataka, svaka s različitim kategorijama i potkategorijama gdje se neke i preklapaju, a to su [4]:

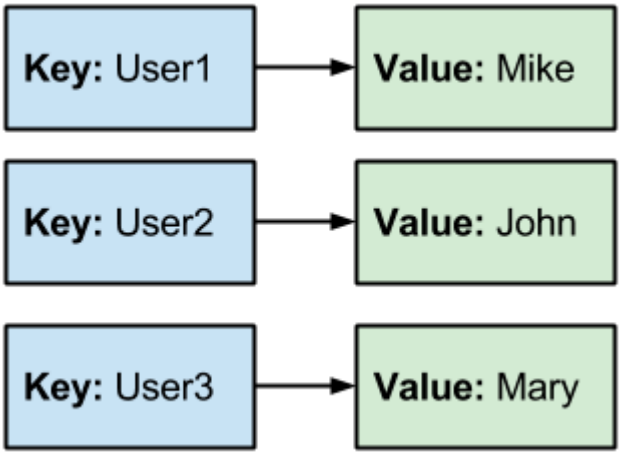
1. *Key-value model* – najjednostavnija vrsta *NoSQL*-a koja pohranjuje podatke ne koristeći shemu, ali koristi indekse i vrijednosti podatka. Neki primjeri su: Redis, *Cassandra*, *Azure*, *LevelDB* i *Riak*.
2. *Column sotre* – ili wide-column store – pohranjuje tablice podataka kao stupce, a ne kao redove. Ovako se podaci spremaju učinkovitije jer ako nema podataka za danu kolonu oni neće biti upisani dok se kod relacijskih baza upisju *null* vrijednosti.
3. *Document database* – radi po konceptu *key-value* modela ali s dodatnom složenošću. Svaki dokument u ovoj vrsti baze podataka ima svoje vlastite podatke i vlastiti jedinstveni ključ koji služi za dohvaćanje podataka. To je dobar izbor za pohranu, dohvaćanje i upravljanje podacima koji su orijentirani ka dokumentima ali ipak imaju nekakvu strukturu. Primjeri takvih baza su: *MongoDB* i *CouchDB*.
4. *Graph database* – dizajnirane su za podatke čiji su odnosi predstavljeni kao graf koji se sastoji od elemenata međusobno povezanih s konačnim brojem odnosa između njih.

## 3.4. Model parova ključeva i vrijednosti

Ključ-vrijednost model baze podataka je vrsta *NoSQL* baze podataka koja koristi jednostavnu metodu ključa i vrijednosti za spremanje podataka. Ključ-vrijednost se odnosi na činjenicu da baza podataka sprema podatke kao kolekciju parova ključeva i pripadajućih vrijednosti.

Parovi ključeva i vrijednosti su utvrđen koncept u mnogim programskim jezicima. U većini programskih jezika ključ-vrijednost se odnosi na asocijativni niz ili strukturu podataka.

Na sljedećoj slici je prikazan jednostavan primjer tablice s ključem i odgovarajućom vrijednosti.




Sl. 3.1. Ključ vrijednost tablica, izvor: [6]

Ključ u paru ključ-vrijednost mora biti jedinstven. To je jedinstveni identifikator koji omogućava pristup vrijednosti koja je povezana tim ključem. Teoretski, ključ može biti bilo što, ali može ovisiti o *SUBP*-u. Neki *SUBP*-ovi mogu imati ograničenja dok drugi nemaju. Npr. u *Redisu*, maksimalna dozvoljena veličina ključa je 512 MB. Za ključ se može koristiti bilo koji binarni slijed, od kratkog *string* tipa podataka, teksta pa do slikovne datoteke. Čak je i prazan niz valjani ključ. Međutim, zbog boljih performansi, ključevi koji su preveliki bi se trebali izbjegavati, ali i prekratki mogu dovesti do pogrešaka. Zbog toga ključevi bi trebali biti dodjeljivani prema dogovorenim konvencijama kako bi se održala konzistencija [7].

Dobar ključ omogućava jedinstveno identificiranje jedinstvenog zapisa koji odgovara na upit bez potrebe gledanja svih vrijednosti unutar tog zapisa. Loš ključ će zahtijevati da vaš aplikacijski kod interpretira vaš zapis kako bi utvrdio hoće li odgovarati upitu.

Ako ključ nije dobro dizajniran, to može dovesti da jedan poslužitelj bude pod puno većim opterećenjem od ostalih što dovodi do loših performansi [5].

Vrijednost u ključ-vrijednosti može biti bilo što, od teksta, broja, *HTML* [6]-a (engl. Hyper Text Markup Language), programskog koda, slike, itd. Vrijednost, također, može biti lista ili čak drugi par ključ-vrijednosti enkapsuliran u objekt.

Neki *SUBP*-ovi dozvoljavaju definiranje tipa podataka za vrijednost. Npr. moguće je definirati vrijednost kao cijeli broj, dok kod drugih *SUBP*-ova koji ne omogućavaju ovo funkciju, vrijednost može biti bilo kojeg tipa [7].

[6] Opisni jezik za izradu internetskih stranica, izvor: [https://hr.wikipedia.org/wiki/HTML]

Redis *SUBP* dozvoljava definiranje sljedećih tipova podataka za vrijednost [7]:

- binarni sigurni nizovi.
- liste: kolekcije nizova elemenata spremljenih prema slijedu unošenja.
- setovi: kolekcije jedinstvenih, nesortiranih nizova elemenata.
- sortirani setovi: slično kao setovi, ali svaki element niza je povezan s cjelobrojnom vrijednosti broja koja se zove „*score*".
- polja bitova.
- hrpe odnosno mape koje se sastoje od polja povezanih s vrijednostima. I polja i vrijednosti su nizovi.

Ključ-vrijednost baze podataka se mogu koristiti za spremanje podataka u raznim slučajevima, a neki od njih su [7]:

- spremanje podataka kod korisničkih profila, informacija o sesijama i elektroničke pošte
- spremanje podataka kod Internet trgovine kao što su sadržaji košarice, kategorije proizvoda, detalji o proizvodu te recenzije proizvoda
- deduplikacija podataka, tablice za prosljeđivanje internetskog protokola, itd.

Ključ-vrijednost baze podataka mogu spremiti cijele internetske stranice, koristeći *URL* [7] *(engl. Uniform Resource Locator)* kao ključ, a internetsku stranicu kao vrijednost.

Neki od primjera ključ-vrijednost baza podataka su: *Redis*, *Oracle NoSQL*, *Aerospike*, *Oracle Berkeley DB.*

- ***Redis*** je jedna od najpoznatijih ključ-vrijednost baza podataka. To je *open-source* memorijska jednonitna baza podataka koja podržava apstraktne podatkovne strukture kao što su nizovi, liste, mape, *hash* tablice, bitmape itd. Najvećim dijelom je razvijena od strane Salvatore Sanfilippa. Redis omogućava velike brzine izvođenja operacija pisanja tako što ograničava veličinu setova podataka koji ne mogu biti veći od memorije. U slučaju nedostatka memorije *Redis* proces će biti prekinut, *Redis* će se srušiti s dojavljenom pogreškom ili će se usporiti. S obzirom na to da je jednonitna baza podataka, ne može se koristiti za paralelno izvršavanje zadataka kao što su pohranjene procedure [8].

Važno je spomenuti još neke od karakteristika ključ-vrijednost baza podataka, a to su:

[7] Ujednačeni ili usklađeni lokator sadržaja, izvor: [https://hr.wikipedia.org/wiki/URL]

- **Jednostavnost**

Ključ-vrijednost baze podataka koriste minimalnu podatkovnu strukturu. Na primjer, ako je potrebno implementirati bazu za pohranu podataka o korisnicima neke Internet trgovine, može se koristiti relacijska baza, ali je jednostavnije koristiti ključ-vrijednost bazu podataka jer nije potrebno definirati shemu u *SQL*-u. Nije čak potrebno ni definirati tipove podataka za atribute koji se žele pratiti. Ako se javi potreba za praćenjem dodatnih atributa nakon što je program već napisan, može se jednostavno dodati programski kod koji će obraditi te atribute, bez promjene baze.

Ključ-vrijednost baza podataka koristi jednostavan podatkovni model. Sintaksa za rukovanje podacima je jednostavna. Potrebno je definirati imenski prostor (engl. namespace), koji je najčešće ime baze, te ključ koji ukazuje na operaciju koja se želi primijeniti na paru ključ-vrijednosti. Kada se navede samo imenski prostor i određeni ključ, baza će vratiti vrijednost za taj ključ. Ako je potrebno ažurirati vrijednost za određeni ključ, potrebno je navesti imenski prostor, ključ i novu vrijednost za taj ključ.

Ova baza podataka je također fleksibilna i pruža mogućnost pogreške. Ako se greškom unese pogrešan tip podatka, na primjer, realan broj umjesto cijelog broja, baza podataka neće izbaciti pogrešku. Ovo svojstvo je korisno kada se tip podataka mijenja ili kada je potrebno podržati dva ili više tipa podataka za isti atribut.

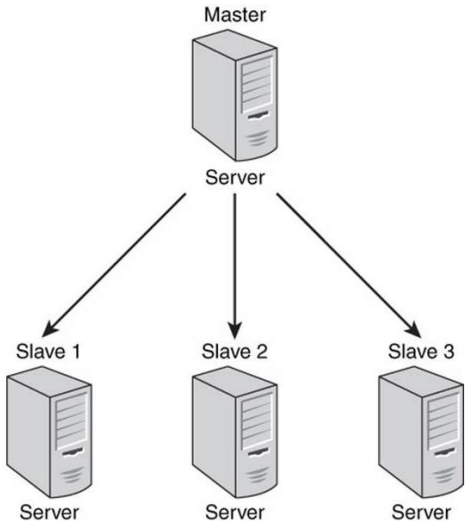
Jedna od prednosti jednostavnih struktura baza podataka je da omogućavaju veću brzinu izvršavanja operacija [9].

- **Skalabilnost**

Skaliranje je važno kako bi se osigurala mogućnost upravljanja novim korisnicima i podacima kako raste aplikacija i poslovanje. Kod ključ-vrijednost baza podataka postoje dva načina skaliranja [9]:

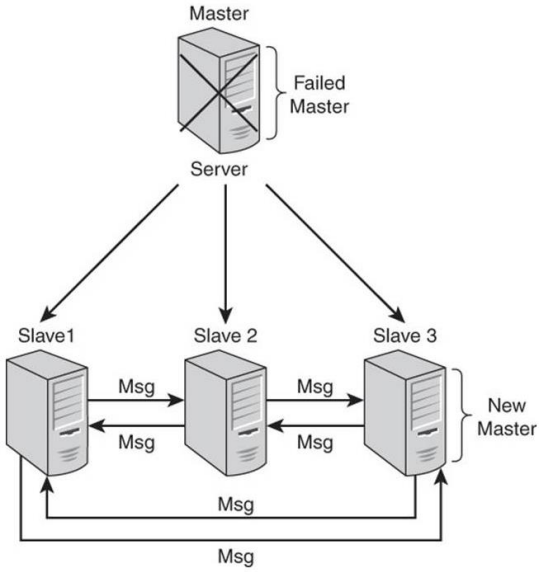
1. *Master–slave* replikacija - jedan od načina da se održi korak s rastućim brojem *read* operacija je da se doda poslužitelj koji može odgovoriti na upite. Kod aplikacija kod kojih je veći broj *read* operacija ( *web* portali, *web* stranice s vijestima) je razumljivo da postoji više poslužitelja koji odgovaraju na *read* upite nego onih koji prihvaćaju *write* operacije. Sve *read* i *write* operacije se događaju na *master* poslužitelju. Slave poslužitelji

preuzimaju i primaju zahtjeve samo ako *master* poslužitelj otkaže. Prednost ove replikacije je jednostavnost koja se ogleda u tome da svaki čvor u oblaku, osim *master* poslužitelja koji komunicira sa svima, mora komunicirati samo s jednim poslužiteljem – *master* poslužiteljem.




Sl. 3.2. *Master-slave* arhitektura prilikom izvođenja uobičajenih operacija, izvor: [9]

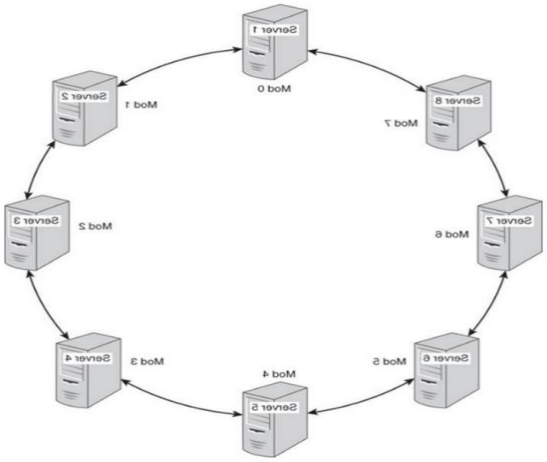
U slučaju da se *master* poslužitelj pokvari, *slave* poslužitelji pokreću protokol kojim postavljaju jednog *slave* poslužitelja za *master* poslužitelja koji će onda prihvaćati operacije za čitanje i pisanje.




Sl. 3.3. Nakon kvara *master* poslužitelja, *slave* poslužitelji pokreću proceduru odabira novog *master* poslužitelja, izvor: [9]

2. *Master – master* replikacija

*Master–slave* model replikacije s jednim poslužiteljem na kojem se odvijaju operacije pisanja ne radi dobro kada postoji veliki broj zahtjeva za operacijom pisanja stoga se u modelu *master – master* replikacija operacije čitanja i pisanja mogu odvijati na svim čvorovima koji rukuju ključem. Ne postoji koncept primarnog poslužitelja.



Sl. 3.4. Oblak od 8 poslužitelja konfiguriranih u prsten, izvor: [9]

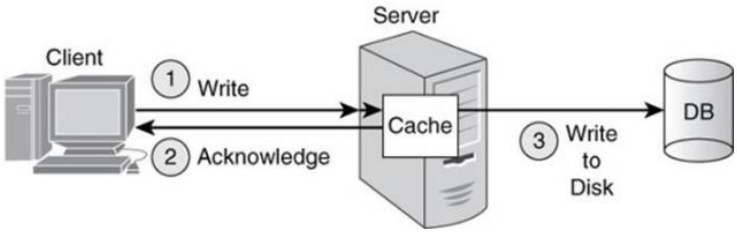
- **Brzina**

Ključ-vrijednost baze podataka su poznate po svojoj brzini. Jednostavnim asocijativnim nizom strukture podataka te svojstvima dizajna za optimizaciju performansi, ključ-vrijednost baze podataka omogućavaju visoku propusnost te podatkovno intenzivne operacije.

Jedan od načina za održavanje brzine rada baze podataka je zadržavanje podataka u memoriji. Čitanje i pisanje u *RAM* [8] (engl. Random Access Memory) je puno brže nego pisanje na disk. S obzirom da *RAM* nije trajna pohrana, prilikom nestanka napajanja, sadržaj *RAM*-a će se izgubiti.

Kada program promijeni vrijednost povezanu s ključem, baza može ažurirati unos u *RAM* i poslati poruku programu da je ažurirana vrijednost spremljena, dok program može nastaviti s drugim operacijama, a za to vrijeme baza može zapisati ažuriranu vrijednost na disk. Pogledati sliku 3.5.

[8] Memorija čijem se sadržaju može trenutno pristupiti, izvor: [https://hr.wikipedia.org/wiki/RAM ]




Sl. 3.5. Primjer pisanja i ažuriranja podataka prvo u *RAM* memoriju, a zatim na disk, izvor: [9]

Drugi način održavanja velike brzine rada s podacima je *SSD (engl. Solid State Drive)*. *SSD* je super brza *flash* memorija za pohranu velikih sinkronih zapisa tako da *RAM* može biti rezerviran za nove podatke. Neke baze podataka kao što je *Aerospike* podržavaju *SSD* pohranu kako bi se osigurala maksimalna propusnost [5, str. 84].

Postoje situacije u kojima nije dobro koristiti ključ vrijednost baze podataka. Neke od njih su sljedeće [10]:

- kada treba povezati različite setove podataka ili povezati podatke između različitih skupova ključeva
- kada treba vratiti neku od operacija zbog prethodno neuspješnog spremanja ključeva
- kada treba pronaći ključ na temelju vrijednosti iz ključ-vrijednost para
- kada treba upravljati s više ključeva

## 3.5. Stupčaste baze podataka

Stupčasto usmjerene baze podataka su baze koje spremaju podatke u stupce, a ne u redove. Podaci se spremaju pomoću ključeva povezanih s vrijednostima koje su grupirane u više obiteljskih stupaca gdje je obiteljski stupac mapa podataka. Ovakva baza omogućava veliku skalabilnost u pohrani podataka [11].

Stupčasto orijentirane baze podataka prikladne su za rudarenje podataka te za aplikacije koje služe za analizu podataka. Neki od značajnijih poslužitelja koji koriste stupčasto orijentirane baze podataka su *Google*-ov *Big Table* i *Cassandra* [10].

- ***Big Table*** je Google-ova komprimirana baza podataka visokih performansi razvijena u *C* i *C++* programskom jeziku. Pruža konzistentnost i tolerantna je na pogreške. Implementacija se sastoji od tri glavne komponente a to su: biblioteka koja je povezana sa svakim klijentom, jedan master poslužitelj i više tablet poslužitelja. Tablet poslužitelji služe za upravljanje setovima tableta( isto što i tablice u relacijskim bazama). Glavni poslužitelj rukuje promjenama sheme, izvršava zadatke poput dodjeljivanja tableta tablet

poslužiteljima, balansira opterećenje tablet poslužitelja. *Big Table* se koristi u *Google*-ovim aplikacijama kao što su *Gmail*, *YouTube* te *Google Earth* [10].

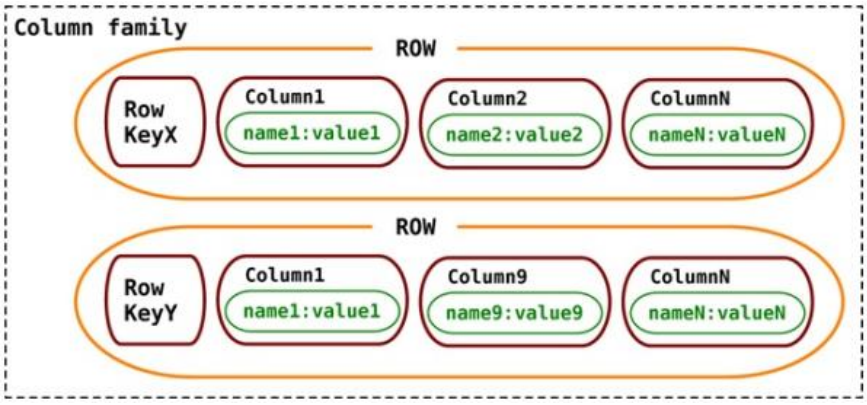
- ***Cassandra*** je razvijena u *Java* programskom jeziku. Temelji se na *Amazonovom Dynamo* modelu *Google*-ovih *Big Table*-a stoga uključuje i *key-value* pohranu kao i stupčastu pohranu podataka. Pruža visoku dostupnost, toleranciju na podjelu, dosljednost, visoku skalabilnost itd. Ima dinamičku shemu te može biti korištena od aplikacija za izradu internetskih stranica do aplikacija za rad u bankarstvu i financijama te za analizu podataka itd. Nedostatak *Cassandre* je što su operacije čitanja sporije od operacija pisanja [10].

Razlika u strukturi između relacijskih baza podataka te *Cassandre* koja predstavlja stupčaste baze podataka je prikazana na slici 3.6..

| RDBMS | Cassandra |
|---|---|
| database instance | cluster |
| database | keyspace |
| table | column family |
| row | row |
| column (same for all rows) | column (can be different per row) |

Sl. 3.6. Razlika u strukturi između relacijskih i Cassandra baze podataka, izvor: [5, str. 89]

Kao što je već spomenuto stupčaste baze podataka pohranjuju podatke u obiteljske stupce poput redova koji imaju više stupaca povezanih s ključem tog reda. Obiteljski stupci su skupine povezanih podataka kojima se najčešće pristupa zajedno. Na sljedećoj slici je prikazan jednostavan primjer obiteljskog stupca.




Sl. 3.7. Cassandrin model podataka koristeći obiteljske stupce, izvor: [5, str. 89]

Obitelj stupaca se sastoji od više redova. Svaki red može se sastojati od različitog broja stupaca u odnosu na druge redove te stupci ne moraju odgovarati drugim stupcima u drugim redovima, odnosno mogu imati različita imena stupaca, različite tipove podataka itd.

Oblak nema glavnog čvora, tako da bilo koja operacija čitanja i pisanja može biti upravljana od strane bilo kojeg čvora u oblaku [10].

Karakteristike stupčastih baza podataka su sljedeće:

- **Sažimanje podataka**

  Stupčasto orijentirane baze su jako učinkovite u sažimanju te u odvajanju podataka u particije [10].

- **Skalabilnost –** je stvar dodavanja novih čvorova u oblak. S obzirom da ne postoji master čvor, kada se doda novi čvor, povećava se kapacitet oblaka za podržavanje operacija pisanja i čitanja. Ovakvo vodoravno skaliranje omogućava maksimalno iskorištavanje vremena, budući da oblak nastavlja s obradom zahtjeva dok se novi čvorovi dodaju na oblak.
- **Brzina** učitavanja podataka je jako velika. Milijarde redova iz tablice se mogu učitati u samo nekoliko sekunda. Slanje upita i analiziranje podataka se mogu obavljati gotovo istog trena. Brzina obrade zahtjeva ovisi i o računalu. Na primjer, prosječan *Serial ATA* tvrdi disk ima prosječno vrijeme obrade zahtjeva između 16 i 22 milisekunde dok *DRAM* pristup na *Intelovom i7* procesoru traje prosječno 60 nanosekundi što je do 400000 puta brže.

  Stupčaste baze podataka povećavaju performanse smanjenjem količine podataka koja se treba učitati s diska učinkovitim sažimanjem sličnih podataka te čitanjem samo nužnih podataka koji odgovaraju upitu [12].

Situacije u kojima nije dobro koristiti stupčaste baze podataka su sljedeće [10]:

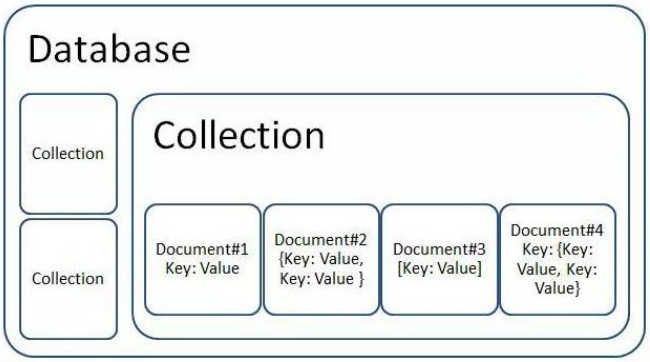
- u sustavima koji zahtijevaju atomičnost, konzistentnost, izolaciju i izdržljivost tj. *ACID* [9] *(engl. Atomicity, Consistency, Isolation, Durability)* transakcije za operacije čitanja i pisanja.
- prilikom izrade ranih prototipa

[9] Skup svojstava baze podataka kojima se jamči valjanost podataka u slučaju pogrešaka, gubitka struje itd.

## 3.6. Baze podataka usmjerene prema dokumentima

Baze podataka orijentirane prema dokumentima su baze dizajnirane za pohranu, dohvaćanje te upravljanje polovično strukturiranih podataka [10] i jedna su od glavnih kategorija *NoSQL* baza podataka te je njihova popularnost porasla s porastom popularnosti *NoSQL*-a. Za razliku od tradicionalnih relacijskih baza podataka, model podataka u dokument orijentiranim bazama podataka nije strukturiran poput tablica s redcima i stupcima. Shema varira, omogućujući tako veću fleksibilnost nego kod relacijskih baza podataka, a to znači mogućnost dodavanja bilo kakvog dokumenta bez potrebe da baza zna strukturu tog dokumenta [13].

Dokument orijentirane baze podataka koriste dokumente kao strukturu za pohranu i izvršavanje upita. Dokument se može odnositi na *Microsoft Word* ili *PDF* dokument, ali je najčešće to *XML (engl. Extensible Markup Language)* [11] i *JSON (engl. JavaScript Object Notation)* [12]. Umjesto stupaca s redcima i tipom podataka koji se koriste u relacijskim bazama, dokumenti sadrže opis tipa podatka i vrijednost za taj podatak. Prilikom dodavanja novog tipa podataka u dokument orijentiranu bazu nije potrebno mijenjati cijelu shemu baze podataka kao kod relacijskih baza, nego se podaci mogu jednostavno dodati dodavanjem objekata u bazu podataka. Svi objekti, pa čak i oni iz iste klase, mogu se potpuno međusobno razlikovati [13].




Sl. 3.8. Primjer strukture dokument orijentirane baze podataka, izvor: [14]

Pohranjivanje dokumenata omogućava pohranu različitih tipova dokumenata, omogućava opcionalna polja unutar dokumenta te dopušta kodiranje pomoću različitih sustava za kodiranje.

[10] Oblik strukturiranih podataka koji nisu u skladu s formalnom strukturom modela podataka povezanih s relacijskim bazama podataka ili drugim oblicima podatkovnih tablica, no sadrže oznake za odvajanje semantičkih elemenata i provođenje hijerarhija zapisa i polja unutar podataka.

[11] Opisni jezik za označavanje podataka, izvor: [https://hr.wikipedia.org/wiki/XML]

[12] Tip podataka lako čitvljiv čovjeku, napisan JavaScript načinom označavanja objekata.

Na primjer, jedan dokument može biti kodiran u *JSON* formatu, dok drugi dokument može biti *XML* formata kao na sljedećim primjerima:

```
{
   „Ime“: „Marko“,
   „Adresa“: „Vukovarska 100“
   „Zanimanje“: „Senior developer“
   „Kolekcija_automobila“: [ "Audi", "BMW", "Mercedes", "Fiat" ]
}
```

Sl. 3.9. Primjer dokumenta u *JSON* formatu

```
<Kontakt>
   <Ime>Petar</Ime>
   <Prezime>Perić</Prezime>
   <Broja telefona>(063) 225-883</Broj telefona>
   <Adresa>
     <Ulica>Vukovarska 112>
     <Grad>Zagreb</Grad>
     <Država>Hrvatska</Država>
     <Poštanski broj>10000</Poštanski broj>
   </Adresa>
 </Kontakt>
```

Sl. 3.10. Primjer dokumenta u *XML* formatu

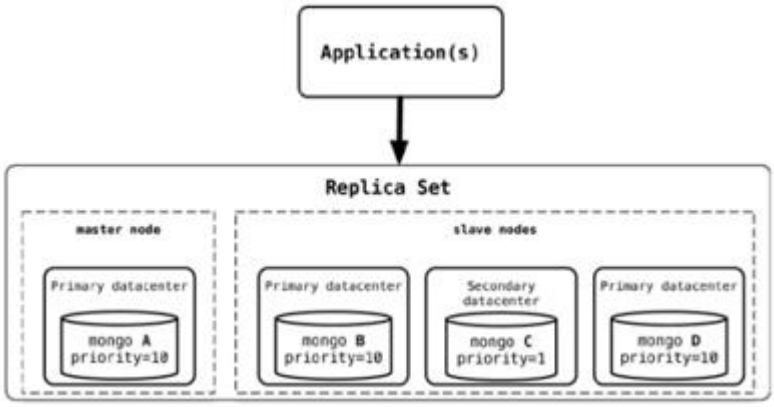
Dva prethodna dokumenta imaju neke zajedničke strukturne, ali također imaju i jedinstvene elemente. Za razliku od relacijskih baza gdje svaki zapis sadrži ista polja i gdje neiskorištena polja ostaju prazna, kod dokument orijentiranih baza nema praznih polja. Ovakav pristup dozvoljava dodavanje novih informacija bez zahtijevanja da svaki zapis u bazi ima istu strukturu.

Iako shema podataka u prethodna dva dokumenta nije jednaka, dokumenti i dalje mogu pripadati istoj kolekciji za razliku od relacijskih baza podataka gdje svaki red u tablici mora imati istu shemu.

Baze podataka usmjerene ka dokumentima također osiguravaju povezivanje i pohranu metapodataka [13] s dokumentacijskim sadržajem te na taj način pružaju organizaciju podataka i sigurnost.

- Skalabilnost

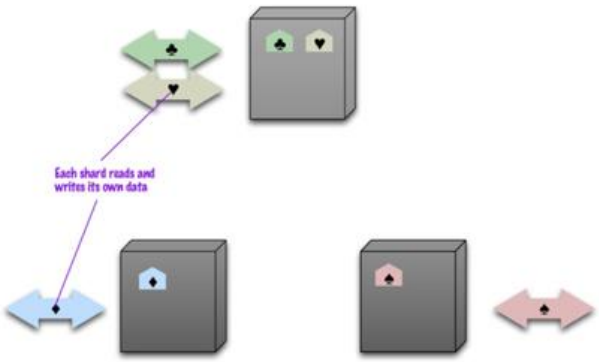
  Skaliranje za učitavanje velikog broja operacija čitanja može se postići dodavanjem slave poslužitelja za *read* operacije, tako da sva čitanja budu usmjerena na slave poslužitelje.




Sl. 3.11. Dodavanje novog *mongo D* čvora u postojeći oblak računala, izvor: [5, str. 86 ].

Dodavanjem novog čvora, on se sinkronizira s postojećim čvorovima, pridružuje se setu replika kao sekundarni čvor i započinje obrađivati zahtjeve.

Kada se želi skalirati za operacije pisanja, tada se radi *sharding* podataka. *Sharding* predstavlja horizontalnu skalabilnost tako da različite vrste podataka odvaja na različite poslužitelje, a to se radi jer često različiti ljudi pristupaju različitim skupovima podataka te tako stvaraju opterećenje.




Sl. 3.12. Razdvajanje različitih podataka na različite čvorove radi bržeg pretraživanja, izvor [5, str. 44]

[13] Podaci koji pružaju informacije o drugim podacima.

U idealnom slučaju, svaki korisnik komunicira samo s jednim poslužiteljem te dobiva brze odgovore od poslužitelja i opterećenje je dobro raspoređeno između poslužitelja. Na primjer, ako postoji 10 poslužitelja, svaki radi pod opterećenjem od 10% [10].

- Dostupnost

  Dokument baze podataka ostvaruju dostupnost koristeći master-slave postavke. Isti podaci su dostupni na različitim čvorovima i korisnici mogu pristupiti podacima čak i ako glavni čvor ne radi.

Dobre su za korištenje u sljedećim situacijama [10]:

- analiza *Weba* ili analiza u stvarnom vremenu
- za sustave koji upravljaju sadržajem
- za *Internet* trgovanje
- za praćenje događaja

Dokument orijentirane baze podataka nije dobro koristiti u sljedećim slučajevima [10]:

- prilikom složenih transakcija koje obuhvaćaju različite operacije
- prilikom stvaranja upita između tablica između kojih se mijenjaju agregacijski odnosi

## 3.7. Graf baze podataka

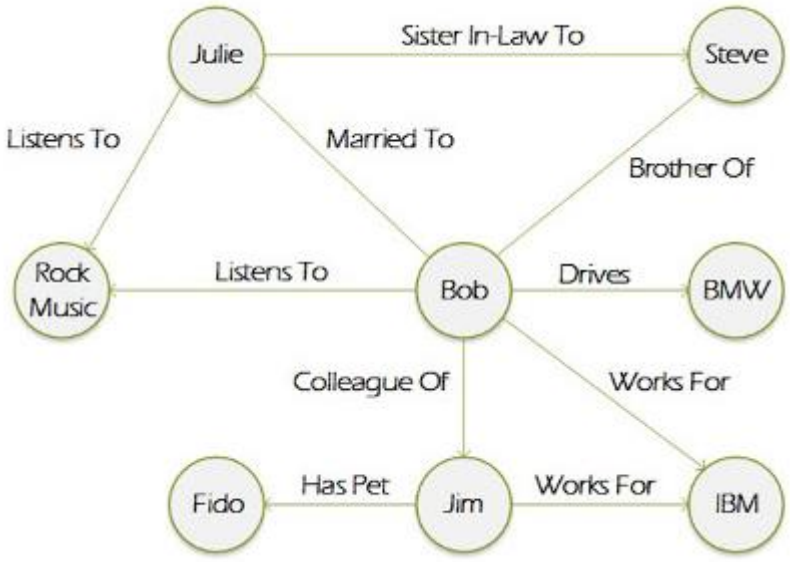
Graf baze podataka su baze koje koriste strukturu grafova za semantičke upite koristeći čvorove i veze te svojstva za prikaz i pohranu podataka [15].

Graf predstavlja izravnu vezu između podataka u bazi. Veze omogućuju izravno povezivanje pohranjenih podataka te u mnogim slučajevima omogućuju njihovo dohvaćanje samo jednom operacijom.

**Čvor** u grafu predstavlja entitet (npr. osoba) i svaka veza predstavlja vezu između dva čvora. Svaki čvor je definiran jedinstvenim identifikatorom tj. setom izlazećih i/ili ulazećih veza i setom svojstava koja se nazivaju ključ-vrijednost parovima. Oni su otprilike ekvivalent zapisa, odnosa ili retka u relacijskoj bazi podataka ili dokumentu u dokument orijentiranoj bazi podataka.

**Veze** su definirane jedinstvenim identifikatorom tj. početnim i/ili završnim čvorom i setom svojstava tj. one su linije koje povezuju čvorove. One su ključan koncept graf baza podataka predstavljajući apstrakciju koja nije direktno implementirana u drugim sustavima.

**Svojstva** su informacije o čvoru. Na primjer, ako YouTube predstavlja jedan čvor, onda bi mogla biti povezana sa svojstvom kao što je internetska stranica ili s riječima koje počinju slovom „Y“ [16].




Sl. 3.13. Primjer jednostavnog grafa, izvor: [15]

Graf baze podataka su prikladne za analizu međusobnih povezanosti te zbog toga postoji veliki interes za korištenje graf baza podataka za rudarenje podataka iz društvenih mreža. Također su korisne za rad s podacima u poslovnim disciplinama koje uključuju složene odnose i dinamičke sheme, kao što je upravljanje nabavnim lancem, identificiranje izvora problema *IP* telefonije itd. [15].

Za razliku od relacijskih baza podataka, graf baze podataka izravno pohranjuju veze između zapisa. Umjesto email adrese koja se pronalazi preko primarnog ključa korisnika u relacijskim bazama, u graf bazama korisnički zapis ima izravan pokazivač na zapis email adrese. To znači da, kada je korisnik odabran, pokazivač može izravno pokazati na njegovu email adresu te nema potrebe tražiti tablicu email adresa kako bi se pronašao traženi zapis. Takva pretraga može olakšati i ubrzati pretraživanje podataka.

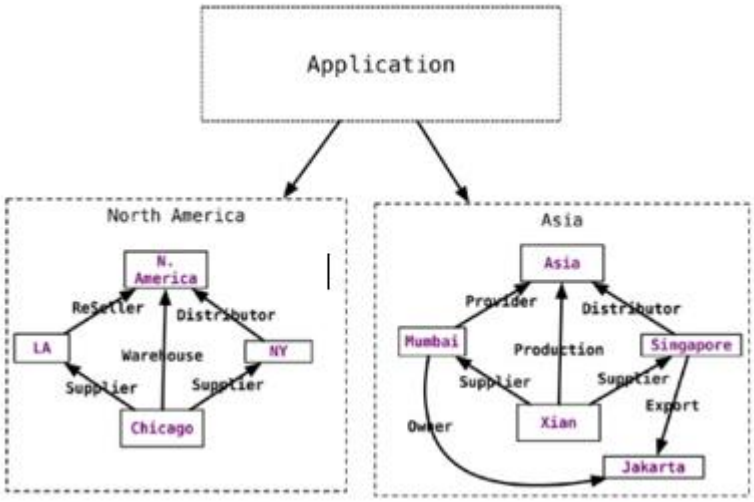
- **Skalabilnost**

  S obzirom na to da graf baze podataka nisu agregacijski, nego relacijski orijentirane, kod skaliranja graf baza podataka nije dobro raditi *sharding* metodu skaliranja. Postoje tri načina skaliranja graf baza podataka.

  Jedan od načina je dodati dovoljno *RAM* memorije poslužitelju tako da se radni skup čvorova i veza između njih u potpunosti nalazi u memoriji. Ova metoda je korisna samo ako se skup podataka s kojim se radi može uklopiti u realnu količinu *RAM*-a.

  Drugi način skaliranja je dodavanje slave poslužitelja koji služe samo za read operacije, a gdje sve operacije pisanja idu na glavni poslužitelj. Ovo je jedna od glavnih metoda skaliranja u *MySQL* oblacima i korisna je kada je skup podataka dovoljno velik da ne stane u *RAM* memoriju, a dovoljno mal kako bi se mogao replicirati na više računala odnosno poslužitelja.

  Treći način je *sharding* podataka od strane aplikacije koji se radi kada je replikacija skupa podataka nepraktična. Npr. čvorovi koji se odnose na Sjevernu Ameriku mogu biti na jednom serveru, dok su čvorovi koji se odnose na Aziju na drugom serveru.




Sl. 3.14. Razdvajanje podataka na razini aplikacije, izvor: [5, str. 104]

***Neo4j*** je najpoznatija *open-source* graf baza podataka razvijena od strane *Neo4j, Inc*. On omogućava spremanje čvorova i veza između njih te njihovo obilježavanje. Neo4j je razvijen u Java programskom jeziku i omogućava korištenje *Cypher Query Language*[14] upitnog jezika za sofisticirane upite [5].

Najčešća upotreba graf baza podataka je za pronalazak najkraćeg puta između dva čvora, npr. [5]:

- kod aplikacija za navigaciju koje služe za pronalazak najkraće rute
- u društvenim mrežama (*facebook, twitter*)
- Za određivanje najdjelotvornijeg puta za usmjeravanje prometa preko podatkovne mreže
- u pronalasku organiziranog kriminala

Graf baze podataka se također koriste kako bi sugerirali određene proizvode kupcima na osnovu sličnih proizvoda koje su kupili drugi ljudi.

Algoritam koji se najčešće koristi za rješavanje ovih slučaja je Dijkstrin algoritam nazvan po Edsgeru Dijkstri. Algoritam radi tako da analizira svaku pojedinu vezu iz izvora do odredišta pamteći najkraću rutu, a odbacujući one duže, sve dok se ne pronađe najkraća rutu.

- **Postizanje visokih performansi keširanjem**

  Prilikom dobivanja čestih upita za iste podatke, poželjno je spremiti te podatke u *cache* memoriju, jer je to brže nego stalno dohvaćati podatke s diska.
  Neo4j omogućava dva načina keširanja podataka:
  ***File buffer cache:*** Spremanje podataka na disk u istom, učinkovito komprimiranom formatu
  ***Object cache:*** Pohranjivanje čvorova, veza i njihovih svojstava u format za učinkovito prebacivanje u memoriju [5].

[14] Deklarativni jezik za rad s graf bazama podataka.

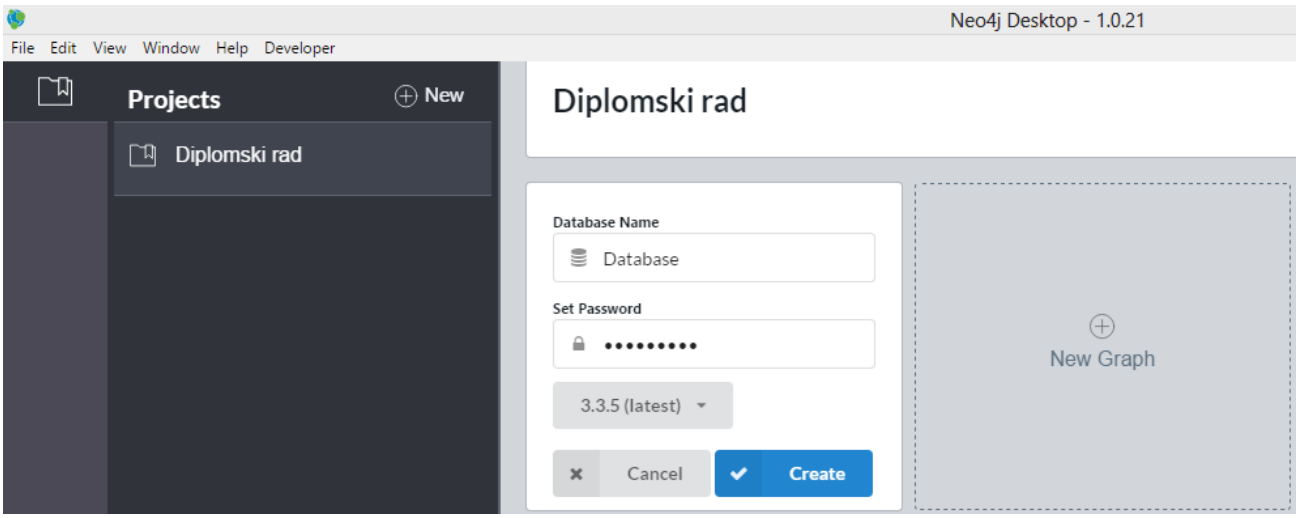
Graf baze podataka nije poželjno koristiti u sljedećim situacijama [17]:

- kada je potrebno ažurirati sve entitete ili njihove podskupove
- za upravljanje velikom listom podataka

### 3.7.1. **Upiti u Neo4j bazi podataka**

U ovom poglavlju će biti ukratko objašnjena izrada grafa te jednostavni upiti nad podacima u graf bazi podataka.

Prvi korak je stvaranje novog projekta odabirom na *New* u izborniku *Projects*. Nakon toga se odabire *New Graph* unutar novog projekta. Potom se odabere *Create a Local Graph*. Zatim se proizvoljno odabere ime za naziv baze podataka te se proizvoljno postavi zaporka za novu bazu podataka. Na kraju se odabere *Create*.




Sl. 3.15. Kreiranje baze podataka koristeći *Neo4j*

Kada je kreirana baza podataka pomoću *Neo4j Browsera* moguće je manipulirati radom baze podataka.

- **Kreiranje čvorova** se izvršava pomoću naredbe *Create*. U radu će se koristiti gotov primjer grafa koji je omogućen od strane Neo4j-a.

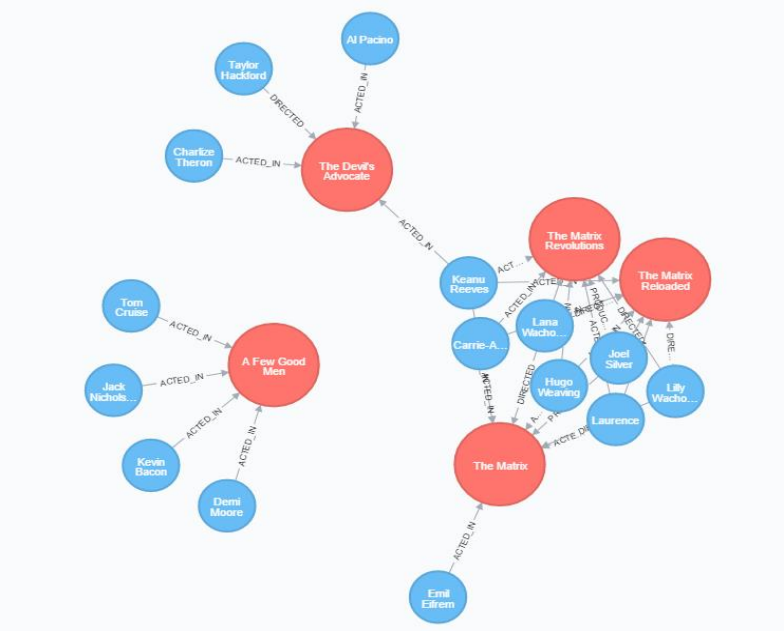
```
CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the Real World'})
CREATE (Hugo:Person {name:'Hugo Weaving', born:1960})
```

Pomoću navedenih naredaba se stvaraju čvorovi film (Movie) pod nazivom „The Matrix“ i osoba (Person) „Hugo Weaving“.

- **Definiranje veza između čvorova**
  Nakon što su definirani čvorovi, potrebno je definirati veze između njih. To se također izvršava naredbom *CREATE*.

```
CREATE  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrix),
```

Prethodnom naredbom je određeno da osoba Hugo glumi ulogu 'Agenta Smitha' u filmu „The Matrix“. Nakon definiranja čvorova i veza između njih graf ima sljedeći izgled:




Sl. 3.16. Prikaz grafa u Neo4j bazi podataka

- **Dohvaćanje** se izvršava naredbom *Match*. Osim samih čvorova, moguće je dohvatiti podatke na osnovu svojstava čvorova i njihovih veza što će biti prikazano na sljedećim primjerima.

```
MATCH (n) RETURN n LIMIT 20
```

Ovom naredbom se dohvaćaju svi čvorovi u grafu. S obzirom na velik broj čvorova u grafu i veza između njih, dohvaćeno je samo 20 čvorova funkcijom *LIMIT* 20 kako bi graf bio jasniji. Rezultat prethodne funkcije je vidljiv na slici 3.7.1.

Osim samih čvorova, moguće je dohvaćati i njihova svojstva. Tako je na primjer moguće dohvatiti uloge svih glumaca pomoću sljedeće naredbe:

```
MATCH (n) WHERE EXISTS(n.roles) RETURN DISTINCT "node" as entity, n.roles
AS roles LIMIT 25
UNION ALL MATCH ()-[r]-()
WHERE EXISTS(r.roles)
RETURN DISTINCT "relationship" AS entity, r.roles AS roles LIMIT 25
```

Table
Text
Code

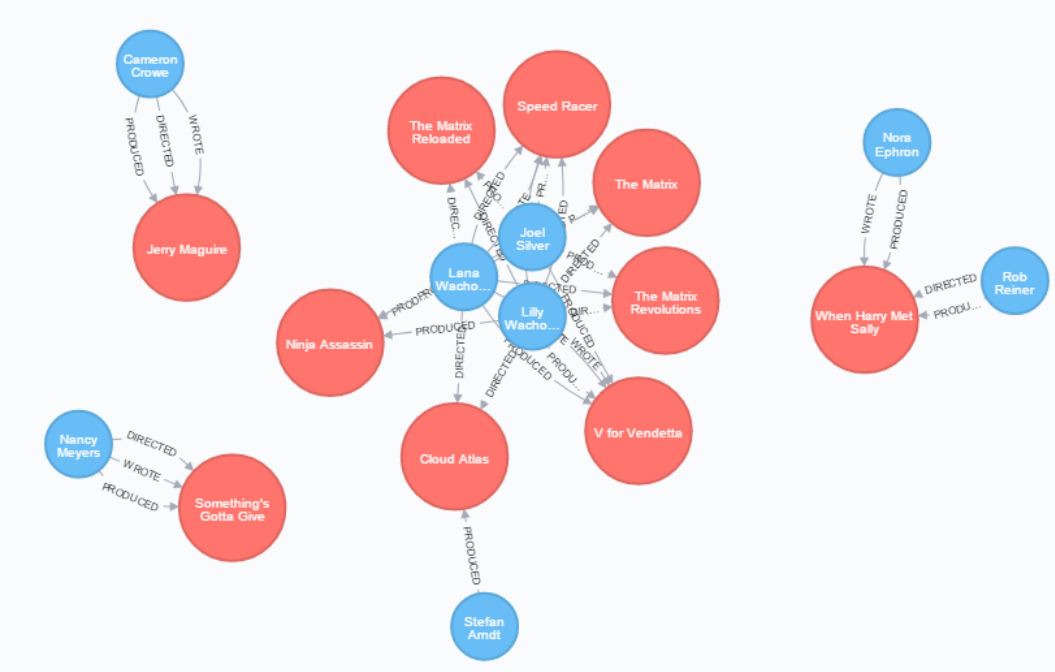
| entity | roles |
|---|---|
| "relationship" | ["Emil"] |
| "relationship" | ["Agent Smith"] |
| "relationship" | ["Morpheus"] |
| "relationship" | ["Trinity"] |
| "relationship" | ["Neo"] |
| "relationship" | ["Julian Mercer"] |
| "relationship" | ["Johnny Mnemonic"] |
| "relationship" | ["Shane Falco"] |
| "relationship" | ["Kevin Lomax"] |
| "relationship" | ["V"] |

Sl. 3.17. Rezultat dohvaćanja svih uloga glumaca

Dohvaćanje podataka na osnovu relacija se izvršava pomoću sljedeće naredbe:

```
MATCH p=()-[r:PRODUCED]->() RETURN p LIMIT 25
```

Prethodnom naredbom se dohvaćaju svi producenti i filmovi koje su producirali.

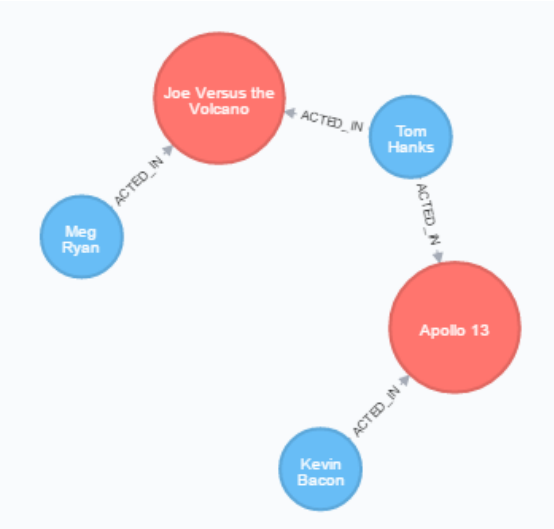



Sl. 3.18. Prikaz producenata i filmova koje su producirali

Postoje razne mogućnosti koje pruža Neo4j baza podataka, a jedna od važnijih je pronalaženje najkraće rute između 2 čvora.

```
MATCH p=shortestPath(
  (bacon:Person {name:"Kevin Bacon"})-[*]-(meg:Person {name:"Meg Ryan"})
)
RETURN p
```

Ovom naredbom je određen najkraći put između osoba „Kevin Bacon“ i „Meg Ryan“. Rezultat upita je sljedeći:

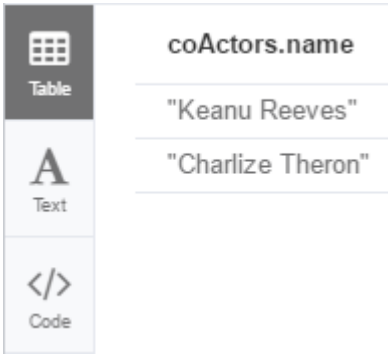



Sl. 3.19. Pronalazak najkraćeg puta između dva čvora

Neo4j omogućava i pronalaženje svih glumaca koji su zajedno glumili u nekim filmovima.

```
MATCH (al:Person {name:"Al Pacino"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors)
RETURN coActors.name
```

Tako se pomoću prethodne naredbe mogu dohvatiti svi glumci koji su glumili zajedno s Al Pacinom.



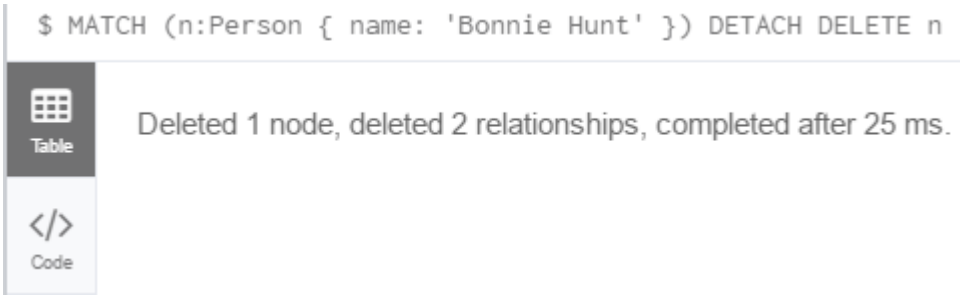

Sl. 3.20. Pronalazak svih glumaca koji su glumili zajedno s Al Pacinom

- **Brisanje**

*Neo4j* omogućava brisanje čvorova, njihovih svojstava te veza između njih. Važno je napomenuti da čvor nije moguće izbrisati ukoliko nisu izbrisane njegove veze s ostalim čvorovima. Brisanje jednog čvora i njegovih veza se izvršava pomoću sljedeće naredbe:

```
MATCH (n:Person { name: 'Bonnie Hunt' })
DETACH DELETE n
```

Rezultat ovog upita je:

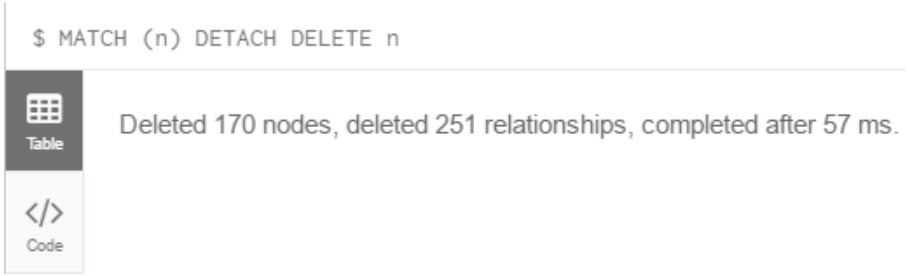



Sl. 3.21. Rezultat brisanja određenog čvora

Brisanje svih čvorova se izvršava pomoću sljedeće naredbe:

```
MATCH (n) DETACH DELETE n
```

Rezultat prethodnog upita je:

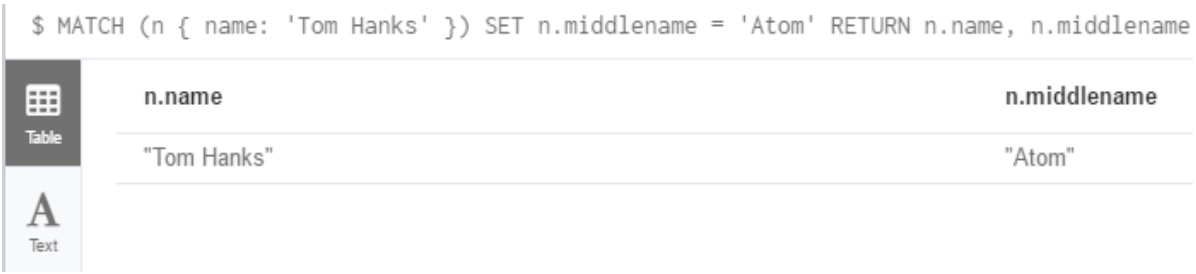



Sl. 3.22. Rezultat brisanja svih čvorova u grafu

- **Ažuriranje** se obavlja pomoću sljedeće naredbe:

```
MATCH (n { name: 'Tom Hanks' })
SET n.middlename = 'Atom'
RETURN n.name, n.middlename
```

Ovom naredbom je glumcu s imenom „Tom Hanks“ dodano srednje ime „Atom“. Rezultat upita je sljedeći:




Sl. 3.23. Rezultat ažuriranja korisnika

# 4. IZRADA NOSQL I SQL BAZA PODATAKA KORISTEĆI MONGODB ODNOSNO ORACLE

U sljedećim poglavljima će biti obrađene izrade baza podataka koristeći *MongoDB* i *Oracle* te će se vršiti usporedbe performansi spomenutih baza podataka.

## *4.1. MongoDB*

MongoDB je jedna od vodećih *NoSQL* baza podataka velike fleksibilnosti i skalabilnosti. To je baza podataka orijentirana prema dokumentima. Osnovana je 2007. godine od strane *MongoDB Inc* kao glavni dio produkta *Platform as a Service*. Postao je jedna od najpoznatijih baza podataka te se koristi kao backend softver za brojne značajne internetske stranice poput *eBaya, New York Times* itd. [18].

*MongoDB* pohranjuje podatke u binarnom prikazu, takozvanom *BSON* (*Binary JSON*). Dokumenti u *MongoDB* bazi podataka su blisko povezani sa strukturom objekata u programskim jezicima. To aplikacijama omogućava jednostavnije i brže mapiranje podataka pohranjenih u bazi podataka. Dokumenti, također, većinom sadrže sve podatke za jedan zapis, dok u relacijskim bazama informacije za određeni zapis se nalaze u više raspršenih tablica, a rezultat toga je brža izvedba i skalabilnost jer jedna read operacija može dohvatiti cijeli dokument dok je u relacijskim baza potrebno koristiti *JOIN* operacije te dohvaćati podatke iz više tablica [19].

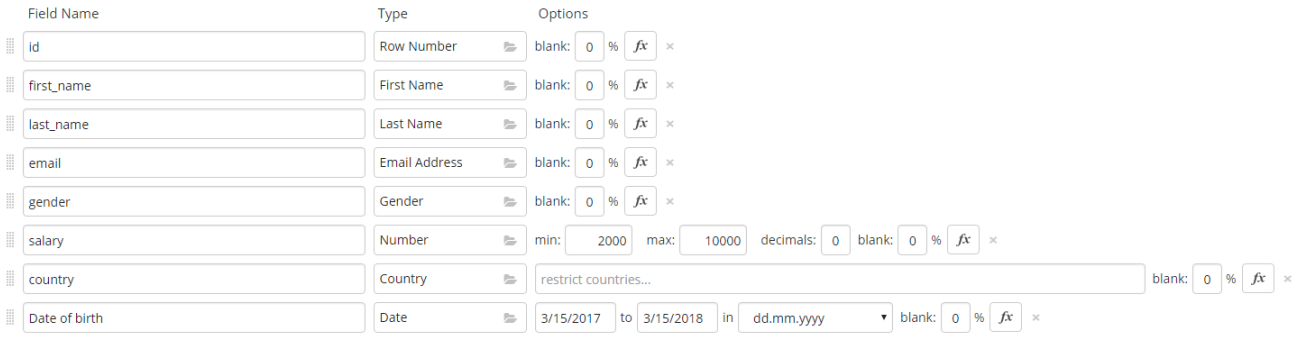
Karakteristike *MongoDB* baze podataka su sljedeće [19]:

- Spremanje podataka u fleksibilne dokumente nalik *JSON* dokumentima, a to znači da polja mogu biti različita u različitim dokumentima i da se struktura podataka može mijenjati tijekom vremena.
- Objekti u određenoj aplikaciji se mapiraju s modelom dokumenta što omogućacva lakši rad s podacima.
- Pristup i analiza podataka preko *Ad hoc* upita, indeksiranja, i agregiranje u stvarnom vremenu
- Visoka dostupnost, horizontalno skaliranje, te rasprostranjenost što omogućava njeno lako korištenje
- *MongoDB* je besplatan, *open-source* softver.

## 4.2. Izrada *MongoDB* baze

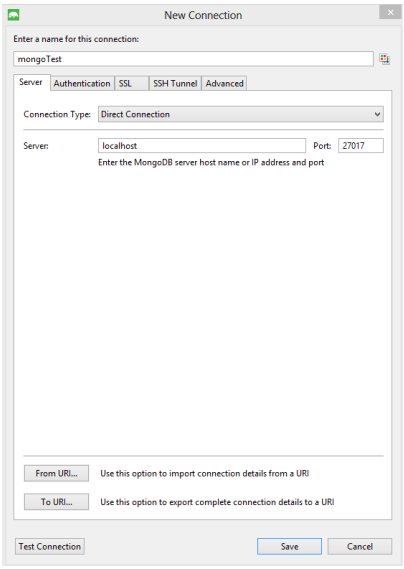
U ovom poglavlju će biti obrađena izrada *NoSQL* baze podataka koristeći *MongoDB*. U ovom radu je korištena *Mongo* baza podataka verzije 3.6. Za lakše upravljanje i manipuliranje podacima instaliran je *3T Studio* koji predstavlja grafičko korisničko sučelje za *MongoDB* bazu podataka.

Podaci za popunjavanje baze podataka su generirani korištenjem *Mockaroo* aplikacije koja omogućava generiranje stvarnih podataka u *csv*, *JSON*, *SQL*, i e*xcel* formatu što je od velike koristi prilikom testiranja i izrade baze podataka.



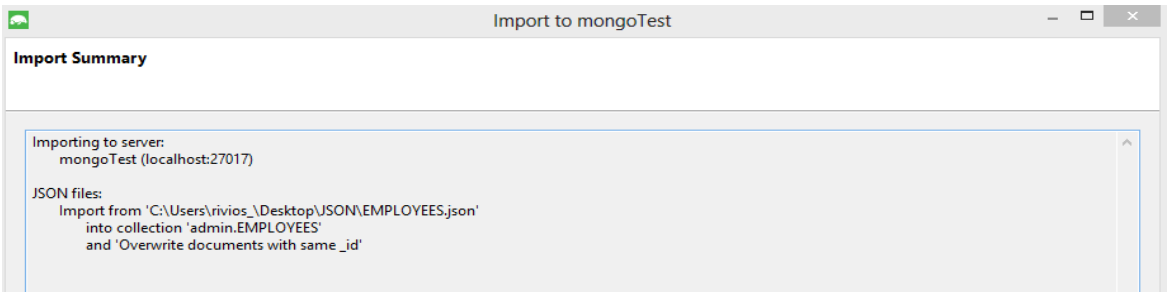

Sl. 4.1. Generiranje podataka korištenjem Mockaroo aplikacije

Nakon spremanja podataka u obliku JOSN formata potrebno je učitati podatke u bazu podataka korištenjem *3T* studija. Kako bi se koristio *3T Studio* potrebno je prvo napraviti konekciju s Mongo Serverom na portu 27017. Prilikom stvaranja nove konekcije potrebno je dati naziv konekcije, unijeti ime servera ili *IP* adresu te port na koji se spaja sa serverom.




Sl. 4.2. Stvaranje nove konekcije s *MongoDB* serverom

Nakon toga potrebno je napraviti novu bazu podataka ili koristiti postojeću admin bazu naredbom use. Zatim, naredbom „*Import Collection*“, moguće je popuniti bazu podataka s prethodno spremljenom *JSON* datotekom.




Sl. 4.3. Popunjavanje kolekcije *employees* podacima iz *JSON* datoteke

U ovom radu kolekcija pod nazivom „*employees*“ je popunjena s 1 000 000 dokumenata od kojih svaki dokument predstavlja pojedinog zaposlenika s pripadajućim atributima nad kojima će se vršiti upiti te mjeriti vrijeme potrebno za izvršenje upita. Primjer jednog dokumenta može se vidjeti na sljedećoj slici:


```
{
  "_id" : ObjectId("5aaa89718f1b0e07704fa21c"),
  "id" : NumberInt(2),
  "first_name" : "Windham",
  "last_name" : "Trulocke",
  "email" : "wtrulocke1@merriam-webster.com",
  "gender" : "Male",
  "salary" : NumberInt(2375),
  "country" : "Indonesia",
  "Date of birth" : "10.07.2017"
}
```


Sl. 4.4. Primjer dokumenta unutar kolekcije „*employees*“

Zapisi u *MongoDB* bazi podataka su dokumenti, čija se struktura sastoji od polja i vrijednosti. *MongoDB* dokumenti su slični *JSON* objektima. Moguće je da je vrijednost nekog polja novi dokument, niz, ili nizovi dokumenata.

## 4.3. Izvršavanje operacija nad podacima u kolekciji *MongoDB*

U ovom poglavlju će se obraditi jednostavni upiti nad podacima u kolekciji *MongoDB* baze podataka te će se mjeriti vrijeme potrebno za njihovo izvršavanje kako bi se dobivena vremena mogla uspoređivati sa sličnim operacijama u relacijskoj bazi podataka. Dobivena vremena su prosječna vremena deset istih uzastopnih upita.

Operacije koje će se obraditi su sljedeće:

- **Učitavanje podataka**

  Učitavanje podataka iz JSON datoteke izvodi se naredbom *Import Data*. Nakon toga se odabere dokument iz kojeg se žele učitati podaci te se opcionalno može odabrati validacija podataka. Na kraju se odabere naredba *Start Import*. Prosječno vrijeme trajanja ove operacije je 33 sekunde.

- **Dohvaćanje jednog zapisa**

```
db.employees.find({„_id“ : ObjectId(„5ab159fe8f1b0e097096897e“)})
```

  Korištenjem funkcije *find* pronalazi se dokument sa zadanim parametrima:

```
{
    „_id“ : ObjectId(„5ab159fe8f1b0e097096897e“),
    „id“ : NumberInt(3),
    „first_name“ : „Tamma“,
    „last_name“ : „Longthorn“,
    „email“ : „tlongthorn2@freewebs.com“,
    „gender“ : „Female“,
    „salary“ : NumberInt(6741),
    „country“ : „Indonesia“,
    „Date of birth“ : „06.03.2018“
}
```

Sl. 4.5. Dohvaćanje korisnika sa zadanim parametrom

Prosječno trajanje ove operacije je 3.4 milisekunde.

- **Dohvaćanje svih zapisa se obavlja pomoću sljedeće funkcije**

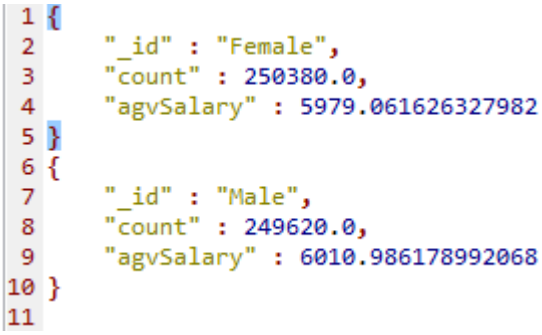
```
db.employees.find({})
```

Kao rezultat će biti dohvaćeni svi dokumenti unutar kolekcije. prosječno vrijeme potrebno za izvođenje ove funkcije je 418.8 milisekunda.

- **Dohvaćanje te računanje prosječnih vrijednosti prema određenom atributu**

```
db.employees.aggregate(
  [{
    $group:
     {
      _id: "$gender",
      count:{$sum :1},
      agvSalary: { $avg: "$salary"}
     }}])
```

Pomoću navedene naredbe iz tablice „*employees*“ se računaju prosječne plaće po spolovima te je naveden ukupan broj pripadnika ženskog i muškog spola. Rezultat upita se može vidjeti na sljedećoj slici:



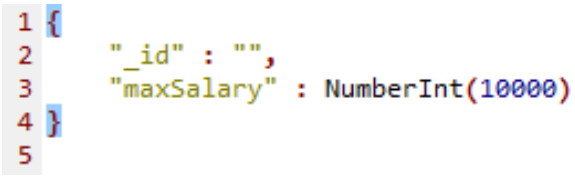

Sl. 4.6. Rezultat računanja prosječnih plaća po spolovima

Prosječno vrijeme trajanja ove operacije je 1.751 sekunda.

- **Dohvaćanje najveće odnosno najmanje plaće:**

```
db.employees.aggregate([{$group:{_id: "",maxSalary: { $max: "$salary"}}}])
```

Kao rezultat navedene agregacije dohvaćena je maksimalna plaća. Potrebno vrijeme za izvođenje ove operacije je 1.473 sekunda.



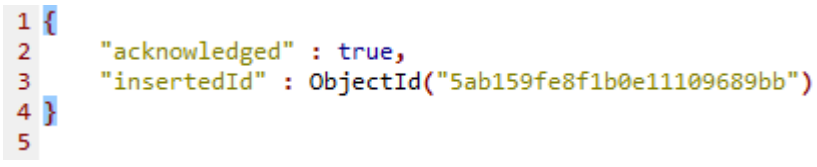

Sl. 4.7. Rezultat pronalaska najveće plaće

- **Dodavanje novog zapisa**

```
db.employees.insertOne(
  {
    "_id" : ObjectId("5ab159fe8f1b0e11109689bb"),
   "id" : NumberInt(1),
   "first_name" : "Marko",
   "last_name" : "Marković",
   "email" : "mmarkovic@dmail.com",
   "gender" : "Male",
   "salary" : NumberInt(2185),
   "country" : "Croatia",
   "Date of birth" : "18.08.2000"
  }
)
```

Prethodnom naredbom se dodaje jedan zapis u bazu. Prosječno vrijeme za ovu operaciju je 16 milisekunda.

Rezultat operacije je sljedeći:



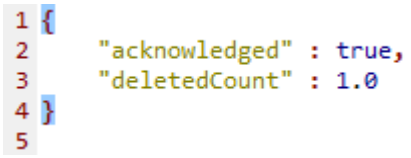

Sl. 4.8. Rezultat dodavanja novog zapisa u bazu podataka

- **Brisanje jednog  zapisa**

```
db.employees.deleteOne({“_id” : ObjectId(“5ab159fe8f1b0e11109689bb”) })
```

Prethodnom naredbom se iz baze briše korisnik čije je prezime Markovic. Potrebno vrijeme za izvođenje ove operacije je 14 milisekunda.

Rezultat operacije je sljedeći:



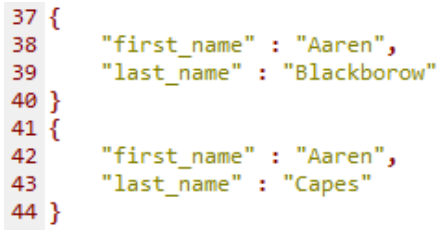

Sl. 4.9. Brisanje dokumenta iz baze podataka

- **Sortiranje podataka**

```
db.employees.find({},{"first_name": 1, "last_name": 1, "_id":0 }).sort({"first_name": 1, "last_name":1,}).limit(100)
```

Prethodnom naredbom se sortiraju podaci uzlazno prema imenu i prezimenu, odnosno od slova A prema Z.

Rezultat sortiranja je sljedeći:




Sl. 4.10. Uzlazno sortirani podaci prema imenu i prezimenu

Prosječno vrijeme potrebno za ovu operaciju je 2.312 sekunda.

- **Ažuriranje svih zapisa**

```
db.employees.updateMany( { "salary": { $lt: 13001 } }, { $set: { salary: 14000 } } )
```

Prethodnom naredbom se ažuriraju plaće svih korisnika. Potrebno vrijeme za izvršenje ove operacije je 26.05 sekunda.

- **Brisanje svih zapisa**

```
db.employees.deleteMany({})
```

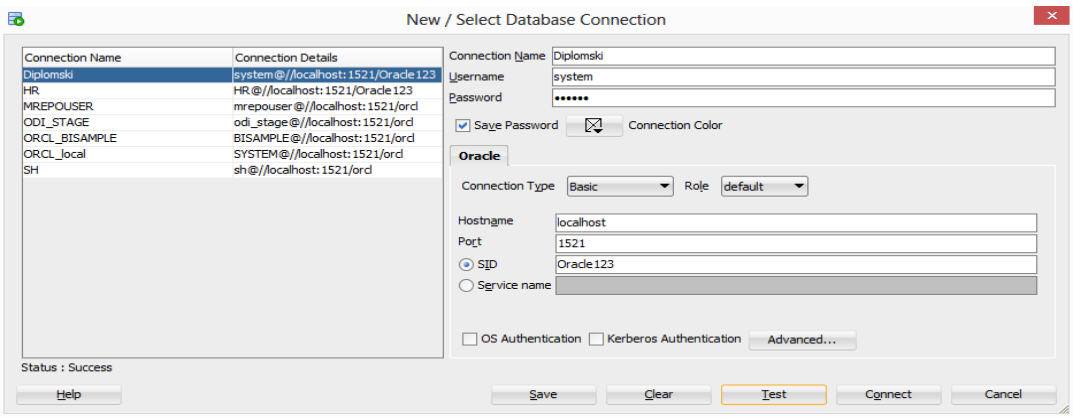
Prethodnom naredbom se brišu svi dokumenti iz kolekcije „*employees*“. Potrebno vrijeme za izvršenje ove operacije je 20.738 sekunda.

## 4.4. *Oracle*

*Oracle DB* je relacijski sustav za upravljanje bazom podataka *Oracle* korporacije. Osnovan je 1977 godine od strane Lawrenca Ellisona i ostalih, te je danas jedna od najpouzdanijih i najkorištenijih relacijskih baza podataka [20]. To je najpopularnija baza podataka na svijetu za obradu online transakcija (*engl. Online transaction protocol*, skraćeno *OLTP*), skladištenje podataka (engl. *Data Warehouse*, skraćeno *DW*) i za mješovita radna opterećenja baze podataka. *Oracle* je izgrađen u relacijskom okruženju u kojem korisnici mogu izravno pristupiti podatkovnim objektima putem strukturiranog jezika upita (*SQL*). *Oracle* je potpuno skalabilna, relacijski izgrađena baza podataka koju često koriste globalna poduzeća koja upravljaju i obrađuju podatke diljem svijeta ili unutar određenog lokalnog područja [21].

## 4.5. Izrada *Oracle* baze

U ovom poglavlju će biti obrađena izrada *SQL* baze podataka koristeći *Oracle*. Za potrebe rada korištena je *Oracle 11g* baza podataka, a za manipuliranje podacima je korišten *Oracle sqldeveloper*. Nakon pokretanja *sqldevelopera* potrebno je napraviti konekciju s *Oracle serverom* na portu 1521. Prilikom stvaranja nove konekcije potrebno je dati naziv konekcije, unijeti korisničko ime i lozinku, odrediti port te ime servisa.




Sl. 4.11. Spajanje na *Oracle* bazu podataka

Nakon toga potrebno je izraditi odnosno definirati tablice u koje će se spremati podaci. Zatim se putem *sqldevelopera* u tablicu mogu učitati podaci u *.xlsx* formatu.

| | COLUMN_NAME | DATA_TYPE | NULLABLE | DATA_DEFAULT | COLUMN_ID | COMMENTS |
|---|---|---|---|---|---|---|
| 1 | ID | VARCHAR2(60 BYTE) | No | (null) | 1 | (null) |
| 2 | FIRST_NAME | VARCHAR2(40 BYTE) | Yes | (null) | 2 | (null) |
| 3 | LAST_NAME | VARCHAR2(40 BYTE) | Yes | (null) | 3 | (null) |
| 4 | EMAIL | VARCHAR2(60 BYTE) | Yes | (null) | 4 | (null) |
| 5 | GENDER | VARCHAR2(20 BYTE) | Yes | (null) | 5 | (null) |
| 6 | SALARY | NUMBER(20,0) | Yes | (null) | 6 | (null) |
| 7 | COUNTRY | VARCHAR2(60 BYTE) | Yes | (null) | 7 | (null) |
| 8 | DATE_OF_BIRTH | VARCHAR2(60 BYTE) | Yes | (null) | 8 | (null) |

Sl. 4.12. Prikaz tablice „*EMPLOYEES*“ u koju se spremaju podaci

## 4.6. Izvršavanje operacija nad podacima u *Oracle* bazi podataka

U ovom poglavlju će se obraditi jednostavni upiti nad podacima u tablici *Oracle* baze podataka te će se mjeriti vrijeme potrebno za njihovo izvršavanje. Dobivena vremena su prosječna vremena deset istih uzastopnih upita. Operacije koje će se obraditi su sljedeće:

- **Učitavanje podataka**

  Učitavanje podataka iz *Excel* tablice izvodi se naredbom *Import data*. Nakon toga se odabere dokument iz kojeg se žele učitati podaci te se odabere metoda učitavanja podataka. Zatim se odabiru stupci koji se žele učitati te se na kraju odabere naredba *Finish*. Prosječno vrijeme potrebno za ovu operaciju kojom se učitavalo milijun zapisa je bilo 403.974 sekunda.

- **Dohvaćanje jednog zapisa** se izvodi pomoću jednostavnog *SQL* upita

```
select * from EMPLOYEES where id = '1';
```

  Prethodnom naredbom se dohvaćaju svi stupci korisnika čiji je id 1.

| ID | FIRST_NAME | LAST_NAME | EMAIL | GENDER | SALARY | COUNTRY | DATE_OF_BIRTH |
|---|---|---|---|---|---|---|---|
| 1 | Caitlin | Fulep | cfulep0@tamu.edu | Female | 197 | China | 16.05.1984 |

Sl. 4.13. Dohvaćanje svih stupaca osobe čiji je id 1.

  Prosječno vrijeme potrebno za izvođenje ove operacije je 18 milisekunda

- **Dohvaćanje svih zapisa** se obavlja korištenjem sljedećeg *SQL* upita

```
select * from EMPLOYEES;
```

  Prethodnom naredbom se dohvaćaju svi stupci svih korisnika u tablici.

  Prosječno trajanje izvođenja ove operacije je 88,95 sekunda

- **Dohvaćanje te računanje prosječnih vrijednosti prema određenom atributu**

```
select  GENDER, count(GENDER), round(avg(SALARY),2) as AVGSalary
from EMPLOYEES
group by GENDER;
```

Pomoću navedene naredbe iz tablice *employees* se računaju prosječne plaće po spolovima te je naveden ukupan broj pripadnika ženskog i muškog spola. Rezultat upita se može vidjeti na sljedećoj slici:

| | GENDER | COUNT(GENDER) | AVGSALARY |
|---|---|---|---|
| 1 | Male | 281252 | 10249.93 |
| 2 | Female | 291571 | 11472.61 |

Sl. 4.14. Računanje prosječne plaće po spolovima

Prosječno vrijeme potrebno za izvođenje ove operacije je 268 milisekunda.

- **Dohvaćanje korisnika s najvećom odnosno s najmanjom plaćom**

```
SELECT * FROM EMPLOYEES WHERE salary=(select max(salary) from EMPLOYEES);
```

Kao rezultat dohvaćanja najviše plaće dohvaćeno je 50 zapisa. Potrebno vrijeme za izvođenje ove operacije je 141 milisekunda.

SQL | Fetched 50 rows in 0.109 seconds

| | ID | FIRST_NAME | LAST_NAME | EMAIL | GENDER | SALARY | COUNTRY | DATE_OF_BIRTH |
|---|---|---|---|---|---|---|---|---|
| 1 | 6546 | Ferdinand | Fidoe | ffidoef5@hud.gov | Male | 99978 | Peru | 08.01.1978 |

Sl. 4.15. Dohvaćanje korisnika s najvećom plaćom

- **Dodavanje novog zapisa**

```
INSERT INTO EMPLOYEES VALUES ('97908775', 'Marko', 'Markovic', 'mmarkovic@gmail.com', 'Male', '5000', 'Croatia', '29.11.1992');
```

Prethodnom naredbom se dodaje jedan zapis u bazu. Prosječno vrijeme za ovu operaciju je 31.6 milisekunde.

- **Brisanje jednog  zapisa**

```
DELETE from EMPLOYEES where LAST_NAME = 'Markovic';
```

Prethodnom naredbom se iz baze briše korisnik čije je prezime Markovic. Potrebno vrijeme za izvođenje ove operacije je 31.2 milisekunde.

- **Sortiranje podataka**

```
SELECT *
FROM EMPLOYEES
ORDER BY FIRST_NAME  ASC;
```

Prethodnom naredbom se sortiraju podaci uzlazno prema imenu, odnosno od slova A prema Z.

Rezultat sortiranja je sljedeći:

Script Output × Query Result ×

SQL | Fetched 150 rows in 0.316 seconds

| | ID | FIRST_NAME | LAST_NAME | EMAIL | GENDER | SALARY | COUNTRY | DATE_OF_BIRTH |
|---|---|---|---|---|---|---|---|---|
| 1 | 346965 | Aaren | Fisher | afisherqs@github.com | Female | 624 | Botswana | 06.03.1990 |
| 2 | 342496 | Aaren | Moen | amoendr@usa.gov | Female | 7254 | Israel | 30.04.1986 |
| 3 | 336965 | Aaren | Fisher | afisherqs@github.com | Female | 624 | Botswana | 06.03.1990 |
| 4 | 332496 | Aaren | Moen | amoendr@usa.gov | Female | 7254 | Israel | 30.04.1986 |
| 5 | 326965 | Aaren | Fisher | afisherqs@github.com | Female | 624 | Botswana | 06.03.1990 |

Sl. 4.16. Uzlazno sortirani podaci prema imenu i prezimenu

Prosječno vrijeme potrebno za ovu operaciju je 657.4 milisekunda

- **Ažuriranje svih zapisa**

```
UPDATE EMPLOYEES
SET salary = round(dbms_random.value(2000,10000))
WHERE SALARY < 11;
```

Prethodnom naredbom ažurirane su plaće svih korisnika. Prosječno vrijeme trajanja ove operacije je 39.742 sekunda.

- **Brisanje svih zapisa**

```
DELETE from EMPLOYEES;
```

Prethodnom naredbom se brišu svi podaci iz tablice „EMPLOYEES“. Prosječno vrijeme potrebno za ovu operaciju je 65.015 milisekunda sekunda.

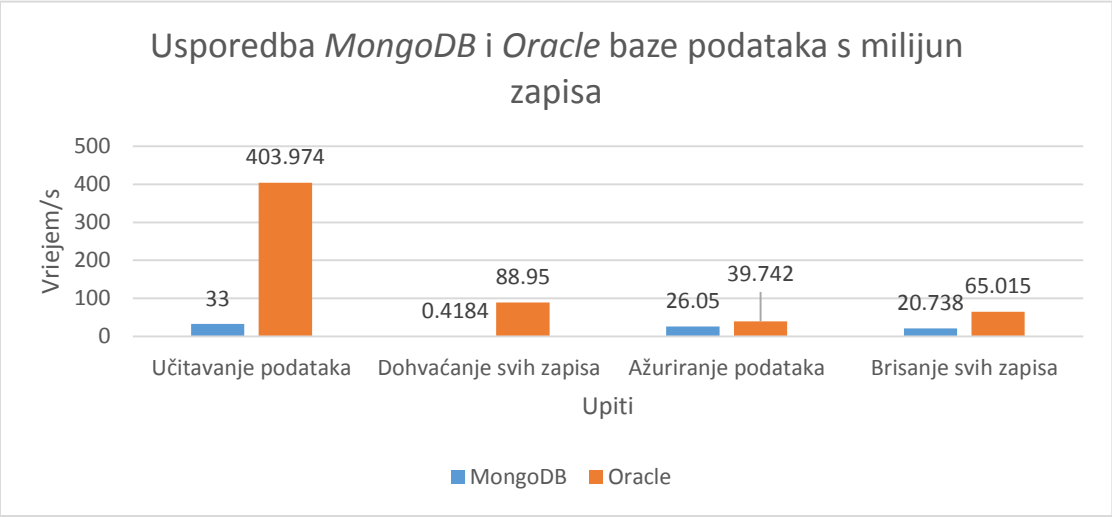
# 5. USPOREDBA NOSQL I SQL BAZE PODATAKA

U ovom poglavlju bit će napravljena usporedba *NoSQL* i *SQL* baza podataka. U tablicama će biti prikazane usporedbe izvođenja osnovnih upita kod *NoSQL MongoDB* i *SQL Oracle* baze podataka. Upiti će se raditi na kolekcijama, odnosno tablicama od 1 milijun, 5 milijuna te 10 milijuna zapisa.

- **Trajanje izvođenja osnovnih operacija na 1,000,000 zapisa**

Tab. 5.1. Usporedba prosječnog trajanja izvođenja osnovnih upita nad podacima kod *NoSQL* i *SQL* baza podataka na milijun zapisa

| Upiti | *MongoDB* | *Oracle* |
|---|---|---|
| Učitavanje podataka | 33 s | 403.974 s |
| Dohvaćanje 1 zapisa | 3.4 ms | 18 ms |
| Dohvaćanje svih zapisa | 418.8 ms | 88.95 s |
| Računanje prosječne vrijednosti | 1.751 s | 268 ms |
| Dohvaćanje najmanje/najveće vrijednosti | 1.473 s | 141 ms |
| Dodavanje novog zapisa | 16 ms | 31.6 ms |
| Brisanje jednog zapisa | 14 ms | 31.2 ms |
| Sortiranje podataka | 2.312 s | 657.4 ms |
| Ažuriranje podataka | 26.05 s | 39.742 s |
| Brisanje svih zapisa | 20.738 s | 65.015 s |




Sl. 5.1. Usporedba *MongoDB* i *Oracle* baze podataka s milijun zapisa

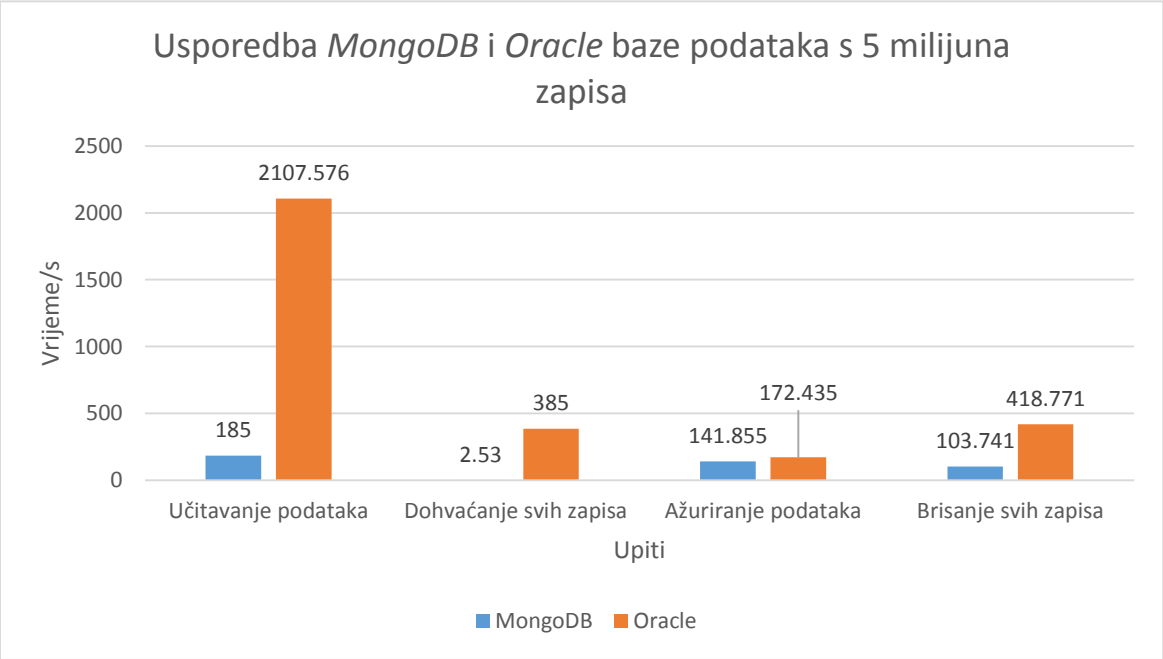
Iz tablice i grafa je vidljivo da je *MongoDB* baza podataka dosta brža u izvođenju operacija učitavanja, dohvaćanja, ažuriranja i brisanja. Kod operacija učitavanja i dohvaćanja zapisa brža je čak i preko deset puta.

- **Trajanje izvođenja osnovnih operacija na 5,000,000 zapisa**

-

Tab. 5.2. Usporedba prosječnog trajanja izvođenja osnovnih upita nad podacima kod *NoSQL* i *SQL* baza podataka na 5 milijuna zapisa

| Upiti | *MongoDB* | *Oracle* |
|---|---|---|
| Učitavanje podataka | 185 s | 2107.576 s |
| Dohvaćanje 1 zapisa | 7 ms | 15 ms |
| Dohvaćanje svih zapisa | 2.53 s | 385 s |
| Računanje prosječne vrijednosti | 9.303 s | 1.188 s |
| Dohvaćanje najmanje/najveće vrijednosti | 7.281 s | 835 ms |
| Dodavanje novog zapisa | 15 ms | 32 ms |
| Brisanje jednog zapisa | 14 ms | 31 ms |
| Sortiranje podataka | 21.98 s | 4.063 s |
| Ažuriranje podataka | 141.855 s | 172.435 s |
| Brisanje svih zapisa | 103.741 s | 418.771 s |



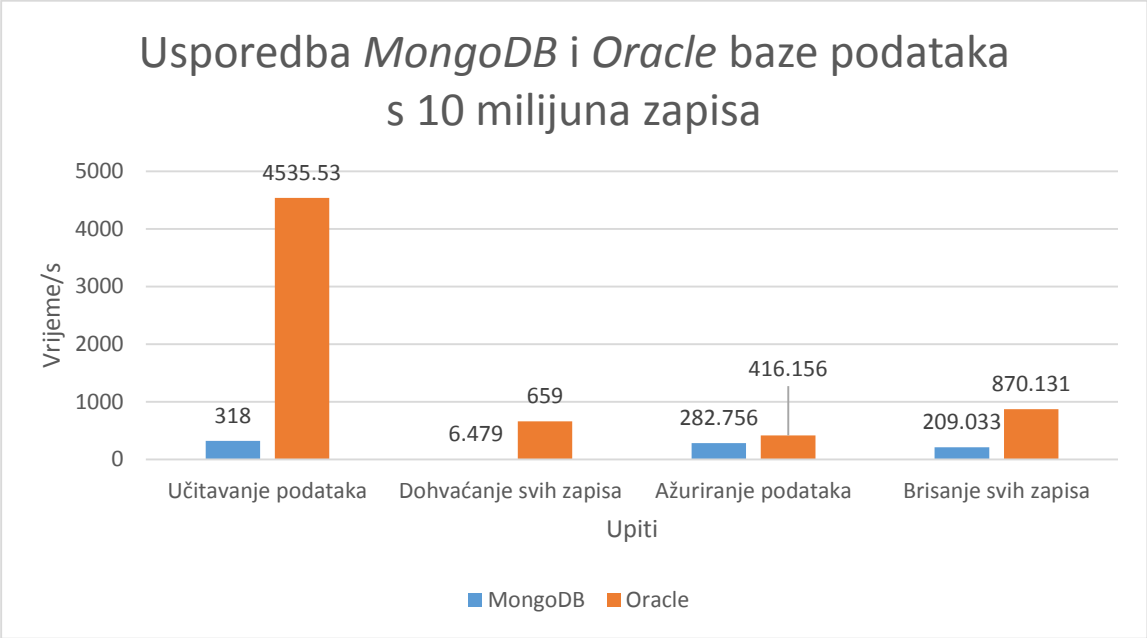

Sl. 5.2. Usporedba *MongoDB* i *Oracle* baze podataka s 5 milijuna zapisa

Povećanjem broja podataka na kojima se izvode upiti raste i vrijeme izvođenja operacija. *MongoDB* i dalje treba značajno manje vremena za izvođenje osnovnih operacija nad podacima.

- **Trajanje izvođenja osnovnih operacija na 10,000,000 zapisa**

Tab. 5.3. Usporedba prosječnog trajanja izvođenja osnovnih upita nad podacima kod *NoSQL* i *SQL* baza podataka nad 10 milijuna zapisa

| Upiti | *MongoDB* | *Oracle* |
|---|---|---|
| Učitavanje podataka | 318 s | 4535.53 s |
| Dohvaćanje 1 zapisa | 12 ms | 16 ms |
| Dohvaćanje svih zapisa | 6.479 s | 659 s |
| Računanje prosječne vrijednosti | 19.275 s | 2.516 s |
| Dohvaćanje najveće vrijednosti | 16.165 s | 2.094 s |
| Dodavanje novog zapisa | 16 ms | 32 ms |
| Brisanje jednog zapisa | 13 ms | 31 ms |
| Sortiranje podataka | 42.086 s | 6.438 s |
| Ažuriranje podataka | 282.756 s | 416.156 s |
| Brisanje svih zapisa | 209.033 s | 870.131 s |



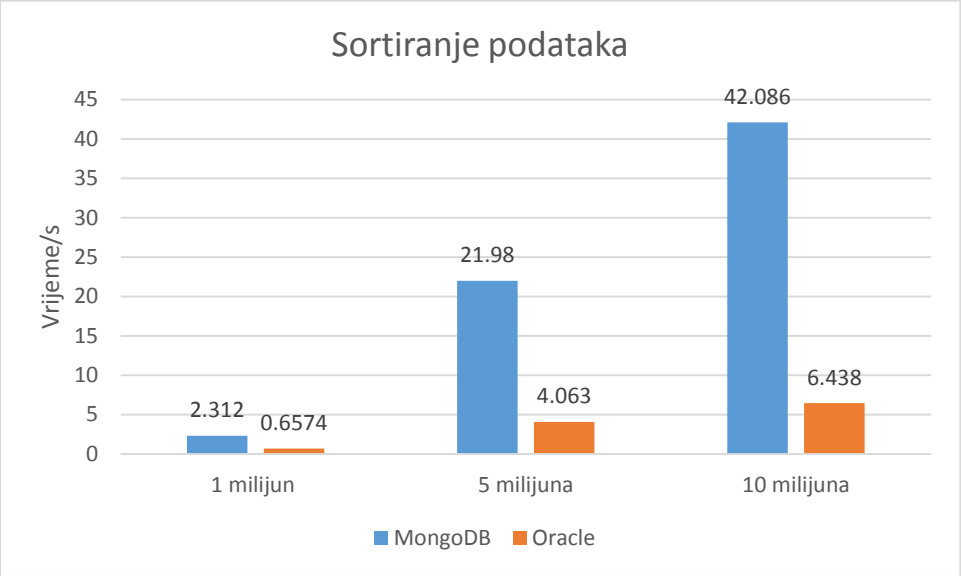

Sl. 5.3. Usporedba *MongoDB* i *Oracle* baze podataka s 10 milijuna zapisa

Prilikom rada s 10 milijuna podataka je potvrđeno da je *MongoDB* puno brži u izvođenju *CRUD* [15]*(engl. Create, Read, Update, Delete)* operacija. Još jednom je pokazano da s povećanjem broja podataka na kojima se izvode upiti raste i vrijeme izvođenja operacija. Funkcija promjene

[15] Operacije kreiranja, čitanja, ažuriranja i brisanja. To su četiri osnovne operacije u radu s podacima.

vremena u odnosu na povećanje količine podataka je linearna kod obje baze podataka što znači da brzina *MongoDB* baze podataka još više dolazi do izražaja s porastom količine podataka. Na primjer, kod rada s milijun podataka za učitavanje kod *MongoDB* baze podataka je bilo potrebno 33 sekunde, a kod *Oracle* baze podataka je bilo potrebno 403 sekunde. Kod rada s 10 milijuna zapisa, za učitavanje podataka kod *MongoDB* baze podataka je bilo potrebno 318 sekunda, a kod *Oracle* baze podataka je bilo potrebno 4535.53 sekunda. Očigledno je da porast vremena kod *MongoDB* baze podataka s 33 sekunde na 318 sekunda je neusporediv s porastom vremena kod *Oracle* baze podataka s 403 sekunde na 4535.53 sekunda iako je porast vremena linearan kod obje baze podataka.

- **Sortiranje podataka**

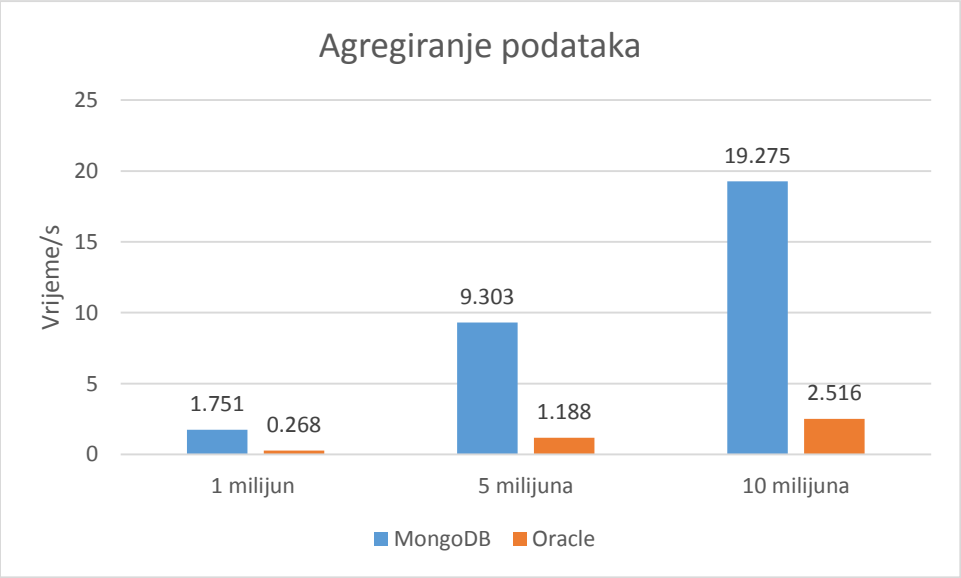



Sl. 5.4. Usporedba *MongoDB* i *Oracle* baze kod sortiranja podataka

Iz slike 5.4. je vidljivo kako je *Oracle* baza podataka značajno brža kod sortiranja podataka od *MongoDB* baze te s povećanjem broja podataka razlika u brzini sortiranja sve više ide u korist *Oracle* baze podataka.

- **Agregiranje podataka**

Kao što je vidljivo na slici 5.5. *Oracle* baza podataka je brža i u operacijama agregacije nad podacima. Kako raste količina podataka u bazi, brzina *Oracle* baze podataka sve više dolazi do izražaja.




Sl. 5.5. Usporedba *MongoDB* i *Oracle* baze kod agregiranja podataka

Vremena izvođenja operacija dohvaćanja, dodavanja i brisanja jednog zapisa ostaju nepromijenjena s porastom količine podataka, odnosno promjene nisu značajne.

- **Razlike između *NoSQL* i *SQL* baza podataka**

Tablica 5.4. Razlike između *NoSQL* i *SQL* baza podataka

| | *NoSQL* | *SQL* |
|---|---|---|
| **Struktura** | *NoSQL* baze podataka imaju dinamičku shemu za nestrukturirane podatke. One su orijentirane prema dokumentima, grafovima, prema parovima ključeva i vrijednosti ili spremaju podatke u obiteljske stupce kao kod stupčastih baza podataka. | *SQL* baze podataka su tablično orijentirane sa strogo definiranom shemom podataka. |
| **Upitni jezik** | *NoSQL* baze podataka koriste upitni jezik sličan Java skriptama s argumentima nalik *JSON*-u. | Za definiranje i manipulaciju podataka se koristi *SQL* upitni jezik. |
| **Skalabilnost** | *NoSQL* baze podataka podržavaju horizontalnu skalabilnost, a to znači rukovođenje povećanjem opterećenja dodavanjem novih poslužitelja u bazu podataka ili raspodjelom opterećenja na više poslužitelja. | *SQL* baze podataka podržavaju vertikalnu skalabilnost što omogućava povećanje opterećenja servera dodavanjem *CPU*-a, *RAM*-a ili *SSD*-a. |
| **Fleksibilnost** | Shema je dinamična što omogućava veliku fleksibilnost | Ograničena fleksibilnost jer je shema prethodno strogo definirana |
| **Brzina** | Brže pretraživanje kod baza s velikom količinom podataka jer se pretražuju samo oni podaci koji su direktno povezani sa zadanim uvjetima. | Brzina izvođenja operacija se smanjuje porastom količine podataka. Nedostatak kod pretraživanja podataka u velikim bazama je taj što je potrebno više puta izvesti *JOIN* operaciju kako bi se dohvatio određeni podatak koji zadovoljava određene uvjete. |

**Prednosti *NoSQL* baza podataka** su sljedeće [22]:

- skalabilnost
- velika fleksibilnost koja omogućava prilagodbu prema potrebama projekta
- promjena sheme bez prekida rada baze podataka
- mogućnost rada na uređajima s malom količinom resursa
- ekonomičnost – mogućnost instaliranja *NoSQL* baze na jeftin hardver te korištenje oblaka računala porastom obujma poslovnih zahtjeva. To znači mogućnost obrade i pohrane podataka s dosta manje troškova.

**Nedostatci *NoSQL* baza podataka** su sljedeći [22]:

- postoji više vrsta *NoSQL* sustava te nema dovoljno visoke razine standardiziranosti
- nedostatak konzistentnosti podataka što u nekim slučajevima može dovesti čak i do njihovog gubitka
- *NoSQL* baze podataka su relativno nova tehnologija te postoje mnoge važne značajke koje nisu još implementirane
- nedostatak alata za analizu i za testiranje performansi [23]
- nedostatak stručnjaka na polju *NoSQL* tehnologija

# 6. ZAKLJUČAK

Na temelju ovog rada se može zaključiti da se podaci mogu organizirati na razne načine. Kroz rad se moglo vidjeti da *NoSQL* baze podataka nude drukčija rješenja za rad s podacima od onih koja nude relacijske baze podataka. Prilikom podjele *NoSQL* baza podataka na dokument orijentirane, stupčaste, graf baze podataka te na parove ključeva i vrijednosti može se vidjeti da ovisno o potrebama i slučajevima se primjenjuju različita rješenja.

Kroz praktične primjere prikazano je jednostavno postavljanje upita u graf bazama podataka. Iz primjera se vidi da su graf baze podataka praktične u mnogim slučajevima svakodnevnice. Praktične su u pronalaženju najkraćih puteva između čvorova što se može primijeniti u pronalasku najkraćih relacija u sustavima navigacija te su pogodne za društvene mreže prilikom pronalaska prijatelja, ali se koriste i za predlaganje proizvoda kupcima na osnovu kupovina njihovih prijatelja.

Usporedbom *MongoDB* baze podataka, koja predstavlja dokument orijentiranu bazu podataka, i *Oracle* relacijske baze podataka, na jednostavnim primjerima se mogao vidjeti odnos brzine izvođenja osnovnih operacija nad podacima. *MongoDB* se pokazao značajno bržim u izvođenju *CRUD* operacija, čak i preko 10 puta u neki slučajevima, dok je *Oracle* baza podataka bila brža u sortiranju podataka i u izvođenju operacija agregiranja nad podacima.

Također se može zaključiti da su *NoSQL* baze podataka pogodnije za rad s nestrukturiranim podacima i pružaju veliku fleksibilnost dok relacijske baze podataka imaju strogo definiranu strukturu i pružaju veću sigurnost u očuvanju podataka.

Na kraju se može reći kako su *NoSQL* baze podataka nova tehnologija u razvoju s određenim nedostatcima koji se trebaju riješiti tijekom vremena te da će odabir *NoSQL* baza podataka u budućnosti ovisiti o potrebama korisnika.

# 7. LITERATURA


[1] https://bs.wikipedia.org/wiki/Baza_podataka (pristup 24.1.2018)

[2] https://en.wikipedia.org/wiki/Database (pristup 24.1.2018)

[3] https://en.wikipedia.org/wiki/NoSQL (pristup 24.1.2018)

[4]https://www.upwork.com/hiring/data/sql-vs-nosql-databases-whats-the-difference/ (pristup24.1.2018)

[5] A. Fowler, NoSQL For Dummies, John Wiley & Sons, 2015.

[6] http://scraping.pro/where-nosql-practically-used/ (pristup 24.1.2018)

[7] http://database.guide/what-is-a-key-value-database/ (pristup 29.1.2018)

[8] https://en.wikipedia.org/wiki/Redis (pristup 03.02.2018)

[9] http://apprize.info/data/nosql/5.html (pristup 04.02.2018)

[10] P. J. Sadalage, M. Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, Addison-Wesley, 2012.

[11] https://en.wikipedia.org/wiki/Column-oriented_DBMS (pristup 08.02.2018)

[12] http://database.guide/what-is-a-column-store-database/ (pristup 10.02.2018)

[13] http://basho.com/resources/document-databases/ (pristup 24.02.2018)

[14] https://dzone.com/articles/a-primer-on-open-source-nosql-databases (pristup 26.02.2018)

[15] https://whatis.techtarget.com/definition/graph-database (pristup 03.03.2018)

[15] https://neo4j.com/docs/developer-manual/3.3/ (pristup 03.03.2018)

[16] https://www.sitepoint.com/sql-vs-nosql-differences/ (pristup 08.03.2018)

[17] https://stackoverflow.com/questions/30133924/when-to-not-use-neo4j (pristup 10.03.2018)

[18] https://en.wikipedia.org/wiki/MongoDB (pristup 15.03.2018)

[19] https://www.mongodb.com/mongodb-architecture (pristup 15.03.2018)

[20] https://www.techopedia.com/definition/8711/oracle-database (pristup 22.03.2018)

[21] https://en.wikipedia.org/wiki/Oracle_Database (pristup 25.03.2018)

[22] https://blog.pandorafms.org/nosql-vs-sql-key-differences/ (pristup 28.03.2018)

[23] http://www.monitis.com/blog/cc-in-review-the-key-differences-between-sql-and-nosql-dbs/ (pristup 30.03.2018)

# SAŽETAK

Fokus ovog rada je opis *NoSQL* baza podataka. U radu su kratko opisane relacijske baze podataka te je kasnije pažnja posvećena podjeli *NoSQL* baza podataka na parove ključeva i vrijednosti, na stupčaste, dokument orijentirane i graf baze podataka. Detaljno su opisane karakteristike svake od spomenutih baza podataka te su navedeni primjeri kada je poželjno koristiti nerelacijske baze podataka kao i primjeri kada to nije slučaj. Kasnije je na jednostavnim primjerima prikazan rad *Neo4j* graf baze podataka. U daljnjem radu je napravljena usporedba *MongoDB* baze podataka, koja predstavlja *NoSQL* bazu, i *Oracle* relacijske baze podataka na praktičnim primjerima te su uspoređivane njihove performanse. Na samom kraju rada navedene su prednosti i nedostaci NoSQL baze podataka.

**Ključne riječi**: *MongoDB, Neo4, NoSQL, Oracle*

# ABSTRACT

## NoSQL Database

The focus of this paper is the description of the NoSQL database. The paper briefly describes the relational database and later attention is devoted to the division of the NoSQL databases into key-value databases, column stores, document databases and graph databases. The characteristics of each of the mentioned databases are described in detail, and examples are given when it is desirable to use non-relational databases as well as examples when that is not the case. Later on, the work with Neo4j graph database was presented on simple examples, and a further work was done to compare performances of MongoDB, which represents the NoSQL database, to Oracle relational database using practical examples. At the very end of the work, the advantages and disadvantages of the NoSQL database are listed.



**Keywords**: NoSQL, Neo4j, MongoDB, Oracle

# ŽIVOTOPIS

Vedran Janjić rođen je 13.05.1993. godine u Russelsheimu u Njemačkoj kao najmlađi član sedmeročlane obitelji. Od 1996. godine sa svojom obitelji je nastanjen u Vidovicama gdje se upisuje u Osnovnu školu Antuna Gustava Matoša. Osnovnu školu završava s prosječnom ocjenom 5.0 te se upisuje u Nadbiskupsku klasičnu gimnaziju u Zagrebu. Nakon jedne godine provedene u Zagrebu prebacuje se u Opću gimnaziju Fra Martina Nedića u Orašju gdje završava svoje srednjoškolsko obrazovanje. Akademske 2012. godine se upisuje na Elektrotehnički fakultet u Osijeku kao redovan student na smjeru Računarstvo te zvanje sveučilišnog prvostupnika stječe 2015. godine. Iste godine se upisuje na diplomski sveučilišni studij, smjer Računarstvo, izborni blok Programsko inženjerstvo, na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija.