

UNIVERZITET U BANJALUCI  
PRIRODNO-MATEMATIČKI FAKULTET

**NoSQL BAZE PODATAKA**  
**SEMINARSKI RAD**

Mentor:  
Dragan Matić

Ime i prezime studenta:  
Jelena Komljenović

Mart, 2020. godina

## SADRŽAJ

1. UVOD .....	1
1.1 NoSQL (nerelacione) baze podataka .....	1
2. NoSQL modeli distribuiranja podataka .....	4
3. Map Reduce koncept.....	8
4. Model podataka .....	14
4.1 Ključ-vrijednost baze podataka .....	14
<b>4.1.1 Redis baza podataka.....</b>	<b>19</b>
4.2 Model familija kolona.....	34
<b>4.2.1 Apache Cassandra baza podataka .....</b>	<b>35</b>
4.3 Graf baze podataka .....	40
<b>4.3.1 Neo4j baza podataka .....</b>	<b>42</b>
4.4 Dokument baze podataka.....	50
<b>4.4.1 MongoDB baza podataka .....</b>	<b>52</b>
5. Nedostaci NoSQL baza .....	62
6. IZVORI .....	63

# 1. UVOD

Razvojem tehnologije javljaju se i nove potrebe kako vezano za skladištenje tako i za obradu podataka. To dovodi do posljedice da SQL baze nisu uvijek najbolji izbor, jer ne mogu da ispune sve očekivane uslove. Upravo iz tog razloga dolazi do nastanka NoSQL baza podataka koje nastaju kao rješenje za sve ono što SQL baze nisu u mogućnosti da izvrše. Da bismo lakše shvatili određene pojmove koji se vežu za same NoSQL baze potrebno je prvo objasniti pojam baze podataka. Možemo reći da su baze podataka određena skladišta podataka. Pomoću njih imamo mogućnost da organizujemo podatke koji su međusobno povezani što nam omogućava da prije dođemo do tih podataka. Ono što je karakteristično za baze podataka je sistem za upravljanje bazom podataka tj. Data Base Management System DBMS pomoću kojeg korisnik može da obavlja sve operacije sa podacima. Podatke možemo pretraživati, sortirati, ažurirati, upoređivati, mijenjati ali i brisati. Postoje dva glavna tipa baza podataka: SQL i NoSQL, tačnije relacione i nerelacione baze podataka. Ono što ih razlikuje je način na koji su građene kao i to koje tipove podataka skladište. Relacione baze su najsličnije imeniku u kojem se nalaze samo osnovni podaci, dok su nerelacione baze više kao dokument koji pored osnovnih sadrži i sve ostale podatke.

## 1.1 NoSQL (nerelacione) baze podataka

Ne postoji još uvijek adekvatna definicija za NoSQL baze. Međutim možemo reći da su to baze koje su nastale iz potrebe da se prevaziđu nedostaci relacionih baza. Sam naziv NoSQL je nastao 1998. godine upravo kao nešto što je suprotno od SQL (relacionih) baza. Međutim teorija je da NoSQL baze ne isključuju SQL baze i da NoSQL zapravo znači: “Not Only SQL” (“Ne samo SQL”). Ne uzimajući u obzir njihovo pravo značenje NoSQL baze se danas odnose na sve baze podataka koje ne prate principe relacionih baza i često se odnose na velike količine podataka s kojima je potrebno upravljati na Vebu. Drugim riječima rečeno, NoSQL nije jedinstven, nego se sastoji od mnoštva drugih proizvoda, koji omogućavaju upravljanje podacima. Ove baze su nastale prvenstveno zbog potrebe za većom fleksibilnošću i boljim mogućnostima prilikom obrade veće količine podataka. Za razliku od relacionih baza podataka kod kojih se podaci skladište u skup relacionih tabela, gdje svaka tabela mora imati svoj primarni ključ, tačnije attribute pomoću

kojih će se razlikovati od ostalih, NoSQL baze rade sa dokumentima tačnije podaci se prikupljaju iz dokumenata. Prednost toga je što i nestrukturisani podaci poput slika ili videozapisa mogu biti spremljeni u dokument koji se vrlo lako može pronaći. Iako je kod ovako organizovanih podataka brža obrada, koristi se dosta veći prostor za čuvanje podataka, što nije slučaj kod SQL baza. Jedna od glavnih razlika između relacionih i nerelacionih baza je upravo to što NoSQL baze nemaju šemu. Prednost toga je što ne moramo prethodno biti upoznati sa samim bazama podataka i njihovim radom kako bismo bili u mogućnosti da upravljamo podacima i izvršavamo određene operacije nad njima. Zahvaljujući odsustvu šeme NoSQL baze se brže i lakše prilagođavaju određenim zahtjevima, drugim riječima ukoliko je potrebna izmjena određenog atributa, to je moguće uraditi bez uticanja na druge podatke. Kod SQL baza to ne bismo mogli uraditi, jer ukoliko bismo željeli dodati ili izmijeniti određeni atribut bili bismo prinuđeni da dodamo niz drugih kolona koristeći naredbu ALTER TABLE, koja bi sve dok se u potpunosti ne izvrši onemogućila ne samo da upravljamo ostalim podacima nego i da vidimo ostale podatke u tabeli i u svim ostalim tabelama koje su povezane sa tom. Neki servisi koji imaju veliki broj korisnika bi u tom slučaju bili u ogromnim gubicima, jer ne bi imali pristup podacima, što je nedopustivo.

Postoje zajedničke karakteristike koje se vežu za sve NoSQL baze podataka:

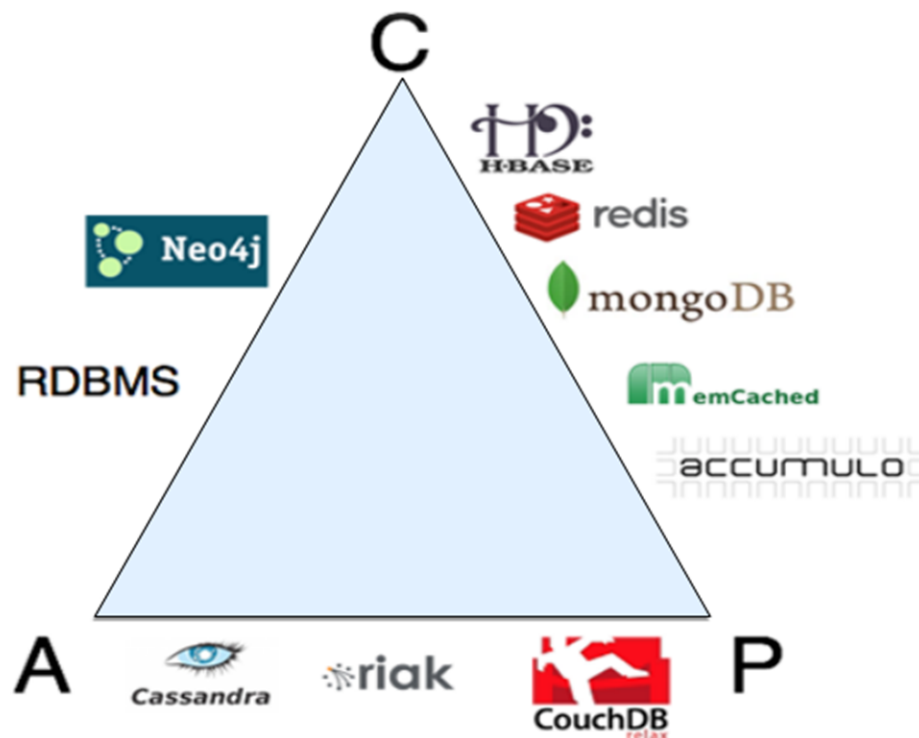
- To su nerelacione baze podataka– Ne koriste ni relacioni model ni SQL upite
- Podaci se čuvaju u posebno određene strukture podataka
- Baze su većinom otvorenog koda (eng. Open source)
- Bolje performanse prilikom rada sa velikom količinom podataka
- Distribuirane baze – Postoji mogućnost rada na više računara sa više baza
- Dobro podnose prekid u mreži- Nastavljaju da rade i ukoliko dođe do prekida u mreži.
- Baze 21. vijeka – NoSQL baze nisu povezane ni sa jednim od prethodnih
- Odsustvo šeme (eng. Schema-less) – ne možemo reći da ove baze uopšte nemaju šemu. Tačno je da sama baza nema strogo određenu šemu, ali ne možemo reći da aplikacija koja je vezana za bazu nema šemu. Tačnije, prisutna je tzv. implicitna šema. Moguće je mijenjati bazu, dodavati i brisati podatke bez definisanja promjena, ali isto tako potrebno je pratiti podatke i njihov način upravljanja kako bismo pronašli implicitnu šemu uz pomoć koje bismo bili u mogućnosti raditi sa podacima.

Jako je bitno naglasiti da za razliku od SQL baza gdje se primjenjuju ACID svojstva, što je skraćeno od atomicity- nedjeljivost transakcije, consistency- konzistentnost, isolation- izolacija i durability- izdržljivost, za NoSQL baze važe BASE svojstva:

- Basically Available – Garantovana dostupnost (CAP teorema)
- Soft state - Stanje u bazi se može promijeniti, iako ne unosimo ništa u bazu podataka, zato što može doći do ažuriranja čvora kojem pripada baza podataka.
- Eventually consistent - Sistem će s vremenom postati konzistentan

Pored BASE svojstava, ono što je veoma važno kada govorimo o NoSQL bazama podataka je CAP (Brewerova) teorema, koja govori da je od navedene tri opcije nemoguće istovremeno ostvariti više od dvije opcije:

- Consistency (konzistentnost) – Svaki čvor ima istu verziju podataka.
- Availability (dostupnost) – Garantuje da će svaki zahtjev dobiti odgovor bez obzira na to da li je bio uspješan.
- Partition Tolerance (tolerancija razdvojenosti) - Baza podataka nastavlja da radi i kada dođe do prekida između čvorova.



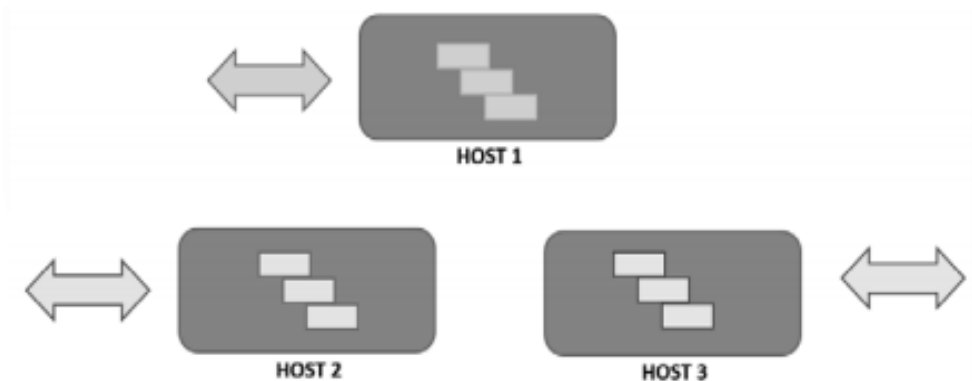
Slika 1.1. CP, AP, CA baze podataka, izvor: [1]

## 2. NoSQL modeli distribuiranja podataka

Kada govorimo o stvarima koje bi trebalo da ispunjavaju NoSQL sistemi među najvažnijim se nalazi održavanje podataka distribuiranih u klasteru. Postoje hostovi, koji su zaduženi za čuvanje podataka i koji učestvuju u slanju podataka. Ti hostovi obrazuju logičku grupu koja se zove klaster. Postoji nekoliko različitih načina distribuiranja i to su:

- Potpuno dijeljenje
- Master-slave (gospodar-rob) replikacije
- Replikacije na ravnopravnim (peer to peer) hostovima
- Kombinacija dijeljenja i replikacije

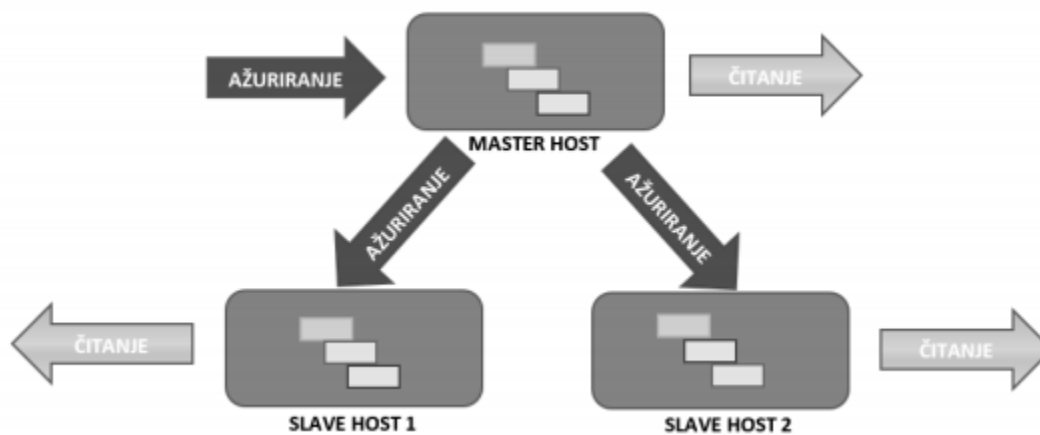
Ono što je karakteristično za model *potpunog dijeljenja* je to što se ni na jednom hostu u klasteru ne nalaze isti podaci, tj. svi podaci na hostovima su u potpunosti različiti. Prednost toga je što se konzistentnost podataka lako održava, zbog odustva kopije podataka. Ovaj model je dobar izbor u slučaju da su aplikacije koje pristupaju lokalnim NoSQL hostovima lokalnog karaktera. Uopšte se ne traže, ili se veoma rijetko traže podaci sa nekih drugih hostova kada je riječ o klasteru. Kod potpunog dijeljenja svaki host je odgovoran samo za razmjenu odnosno čitanje, upis, ažuriranje i slično onih podataka koje skladišti.



Slika 2.1. Potpuno dijeljenje, izvor: [2]

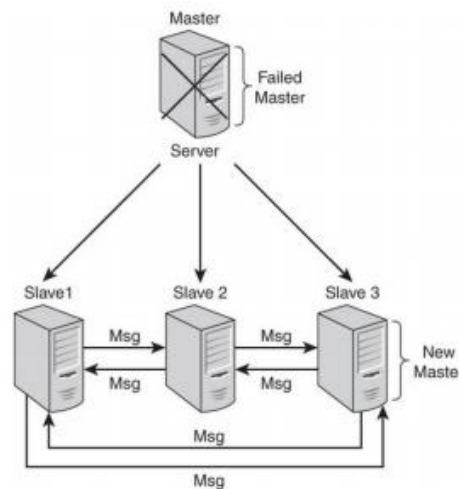
*Master-slave replikacija* funkcioniše potpuno suprotno u odnosu na model potpunog dijeljenja. Dok se kod modela potpunog dijeljenja na svim hostovima nalaze različiti podaci, ovdje se na svakom hostu u klasteru nalaze isti podaci. Upravo iz tog razloga postoji master (gospodar) host na kojem je jedino moguća izmjena podataka. Ostali hostovi koji imaju slave ulogu nemaju mogućnost

izmjene podataka, ali im je dozvoljeno čitanje podataka. U ovom slučaju je neophodno da master host održava konzistentnost podataka tj. ažurira podatke na svim slave hostovima, kako bi svi hostovi uvijek imali dostupnu istu kopiju podataka. Kako bi ovaj model bio u potpunosti uspješan neophodna je pouzdana mrežna infrastruktura i velika brzina protoka podataka. Slave hostovi ne preuzimaju i neprimaju zahtjeve, osim u slučaju ako dođe do kvara na master hostu. Ono što je prednost kod ove replikacije je njena jednostavnost za koju je zaslužno to što svaki host u klasteru, osim master hosta koji komunicira sa svim hostovima, ima mogućnost da komunicira samo s jednim hostom odnosno master hostom.



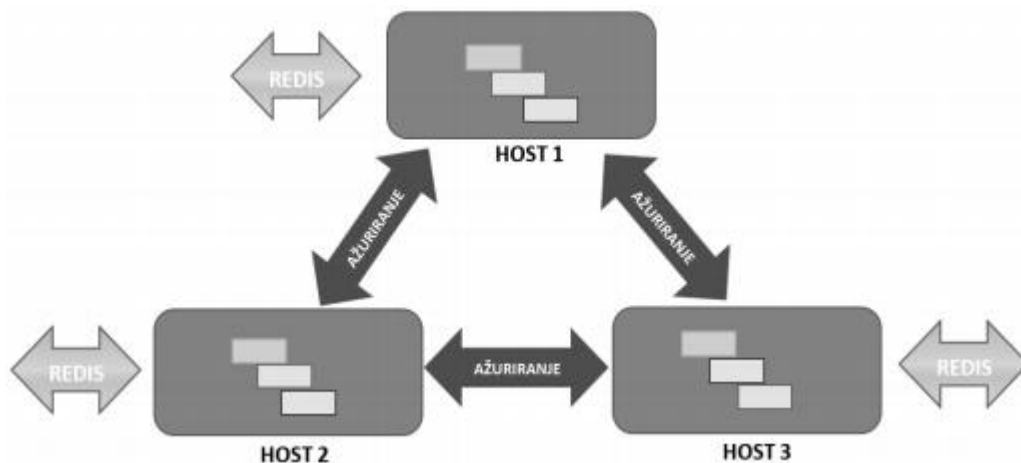
Slika 2.2. Master-slave replikacija, izvor: [2]

Kao što je već napomenuto, postoji mogućnost da dođe do kvara na master hostu. Kako bi se to uspješno otklonilo slave hostovi pokreću protokol čiji je rezultat postavljanje jednog od slave hostova za master hosta, koji će nastaviti da prihvata sve prethodno navedene operacije.



Slika 2.3. Slave hostovi nakon kvara master hosta biraju novog master hosta, izvor: [3]

*Replikacije na ravnopravnim (peer to peer) hostovima* imaju dosta sličnosti sa prethodno objašnjenim modelom. Ono što predstavlja veliku razliku u odnosu na model master-slave replikacije je to što je ovdje svim hostovima u klasteru dozvoljeno da pored čitanja podataka mijenjaju te iste podatke, i to svojstvo je označeno kao REDIS (engl. Read, Edit, Delete, Insert, Select). Kod ovog model konzistentnost podataka je u klasteru je znatno otežana, jer ukoliko dođe do promjene podataka na jednom od hostova to će uticati i na sve ostale hostove. Upravo zbog toga NoSQL sistem mora podržavati obradu transakcija, tj. mora omogućiti da se promjene odigraju u isto vrijeme i u jednoj transakciji na svim hostovima. Ono što može predstavljati problem je gubitak komunikacionog kanala koji je zaslužan za ažuriranje podataka zbog kojeg bi sistemi mogli biti ugroženi, jer postoji mogućnost raspodjele netačnih podataka korisničkim aplikacijama. Čak i ako se linkovi stalno testiraju i prije i tokom samog trajanja transakcije radi samog prihvatanja ili odbijanja transakcije, to nije sigurna garancija uspostavljanja pouzdanosti ovih sistema.



Slika 2.4. Peer to peer dijeljenje podataka, izvor: [2]

Upravo ta mogućnost pojave netačnih podataka u prethodno opisanim situacijama bila je neophodna da dođe do pojave novog modela koji je nastao kao kombinacija potpunog dijeljenja i replikacije. Možemo reći da je to hibridni model. Ovaj model ima dvije različite varijante. Za prvu varijantu (slika 2.5.) je karakteristično da hostovi mogu biti u isto vrijeme i master hostovi i slave hostovi tj. hostovi mogu da budu masteri za neke podatke kao što su skladištenje njihovih originala i ažuriranje njihovih kopija, dok za druge podatke imaju ulogu slave hosta odnosno



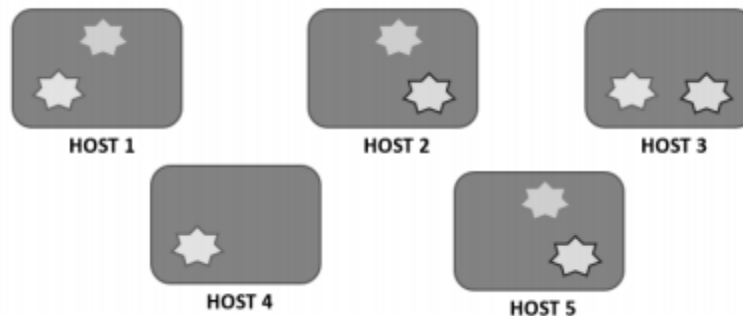
imaju mogućnost da samo skladište kopije. Takođe u prvoj varijanti je moguće i da hostovi budu samo master ili samo slave hostovi. Kako bi se uspješno izvršilo ažuriranje podataka postoji pravilo distribuiranosti koje nalaže da u klasteru mora postojati jedan original podataka i da se mora znati na kojem hostu se on nalazi.



Slika 2.5. Kombinacija dijeljenja i replikacije - 1. varijanta, izvor: [2]

Na slici 2.5. možemo vidjeti da host 1 ima ulogu mastera za 2 podatka koja su prikazana zvijezdama, dok ulogu slave hosta ima za treći podatak koji je prikazan kao pravougaonik. Isto tako host 2 je master za jedan podatak i slave za 2 podatka, dok je host 3 u potpunosti slave host.

Druga varijanta kombinacije dijeljenja i replikacije se predstavlja pomoću ravnopravnosti hostova (peer to peer hostovi). Za razliku od prve varijante ovdje svi hostovi imaju ulogu mastera i ne postoje kopije podataka, tj. na svim hostovima se nalaze originali podataka.



Slika 2.6. Kombinacija dijeljenja i replikacije – 2. varijanta, izvor: [2]

Ukoliko dođe do promjene podataka na bilo kom hostu, taj host je zadužen za ažuriranje tih podataka na svim ostalim hostovima. Drugim riječima svaki host mora da zna u kojim se sve drugim hostovima u klasteru nalazi taj isti podatak,

kako bi uspješno mogao da ažurira promjenu podataka. Kako bi to bilo uspješno u praksi NoSQL podaci su distribuirani sa faktorom tri, odnosno svaki podatak je distribuiran na 3 hosta u klasteru. Zahvaljujući tome povećana je otpornost sistema na padove koji su mogući u klasteru, odnosno klaster se dijeli na dijelove koji se vraćaju u prvobitne, nakon što dođe do uspostavljanja normalnog protoka podataka. Kako bi ovo bili moguće NoSQL sistemi pamte tačno vrijeme promjene svih podataka koji su skladišteni u klasteru, što za rezultat daje vraćanje najkasnije ažuriranih podataka kada dođe do ponovnog uspostavljanja komunikacionog kanala. Ovaj način distribuiranja podataka najviše koristi model familija kolona, koji će biti detaljno objašnjen u nastavku.

## Map Reduce koncept

*Map Reduce* koncept je nastao kako bi podaci predstavljeni agregacionim modelom bili u mogućnosti da se razmjenjuju između hostova. Postoje dvije faze koje je potrebno ispuniti kada je riječ o Map Reduce konceptu i to su: mapiranje i redukcija.

U prvoj fazi, mapiranju, izvršava se izvlačenje, formatiranje i filtriranje podataka. Kako bi se ova faza mogla efikasno izvršiti, neophodno je da podaci budu dostupni procesoru koji je zadužen za obradu map funkcije. Ova funkcija prvo čita podatke iz baze, zatim razlaže zahtjev u niz nezavisnih funkcija, koje se mogu posebno izvršavati i raspoređuje te iste funkcije na različite procesore. Kao rezultat mapiranja procesora dobijaju se serija ključ-vrijednost parova. Da bi se mapiranje izvršilo potrebno je da funkcija za ulazni parametar uzme agregiran podatak, kako bi se kao izlaz dobio niz podataka koji su predstavljeni parovima ključ-vrijednost.

Šifra	12789500317
Kupac	Jelena Komljenović

Jabuka	5	2.45	3500
Kafa	1	1.00	2000
Voda	2	2.00	1456
Čokolada	2	4.25	152
Limun	4	3.85	84

Ukupno: 7192

U ovim tabelama se nalazi agregiran podatak u obliku računa koji je ujedno i ulazni parametar map funkcije, a u tabelama ispod vidimo da je rezultat izvršavanja ove funkcije predstavljen u obliku pojedinačnih stavki računa po ključ-vrijednost principu. Naziv proizvoda predstavlja ključ, dok se vrijednost sastoji od dva ključ-vrijednost para: za cijenu i za količinu.

Rezultat mapiranja:

Jabuka	
Cijena	2.45
Količina	5

Kafa	
Cijena	1.00
Količina	1

Voda	
Cijena	2.00
Količina	2

Čokolada	
Cijena	4.25
Količina	2

Limun	
Cijena	3.85
Količina	4

Pošto je riječ o podacima koji predstavljaju aggregate, odnosno o podacima koji su sastavljeni od distribuiranih podataka koji se nalaze na različitim hostovima, map funkcija se izvršava istovremeno na svim hostovima na kojima se nalaze zahtjevani podaci. Prednost ovakvog pristupa je velika brzina procesiranja podataka ne samo zbog obrade koja se dešava istovremeno na svim hostovima, nego i zbog lokalnog karaktera podataka nad kojim se izvršava obrada. Drugačije rečeno, host u ovoj fazi obrađuje samo one podatke koji se u njemu skladište.

Ono što je važno naglasiti je to da se funkcija za mapiranje izvršava samo na jednom zapisu što smo imali priliku da vidimo u prethodnom primjeru gdje je račun predstavljao zapis. U slučaju da želimo prikupiti podatke koji su vezani za

kupovinu određenog proizvoda (npr. koliko je ukupno zaređeno novca od prodaje tog proizvoda ili u kojoj količini je taj proizvod prodat), funkcija za mapiranje se onda izvršava nad računima svih kupovina. Tada će se kao rezultat dobiti ključ-vrijednost parovi koji mogu da se redukuju u zavisnosti od potrebe. Redukcija parova ključ-vrijednost je ujedno i druga faza Map Reduce koncepta. Redukcija za ulazne parametre uzima parove ključ–vrijednost, a zatim nad njima izvršava potrebne operacije. Dobijeni podaci se mogu sortirati, grupisati ili predstaviti u sažetom obliku. Kada se podaci obrade vraćaju se na spajanje procesoru koji ih je podijelio, čime se dobija i završni rezultat koji se zahtjeva od klijenta. Kako bi se za rezultat dobio samo jedan par ključ-vrijednost, tj. kako bi se spriječilo pojavljivanje velikog broja parova ključ-vrijednost, host šalje samo jedan par, koji predstavlja konačan rezultat obrade koji je ujedno zbir svih pojedinačnih parova.

Jabuka	
Cijena	2.45
Količina	5

Jabuka	
Cijena	37.24
Količina	76

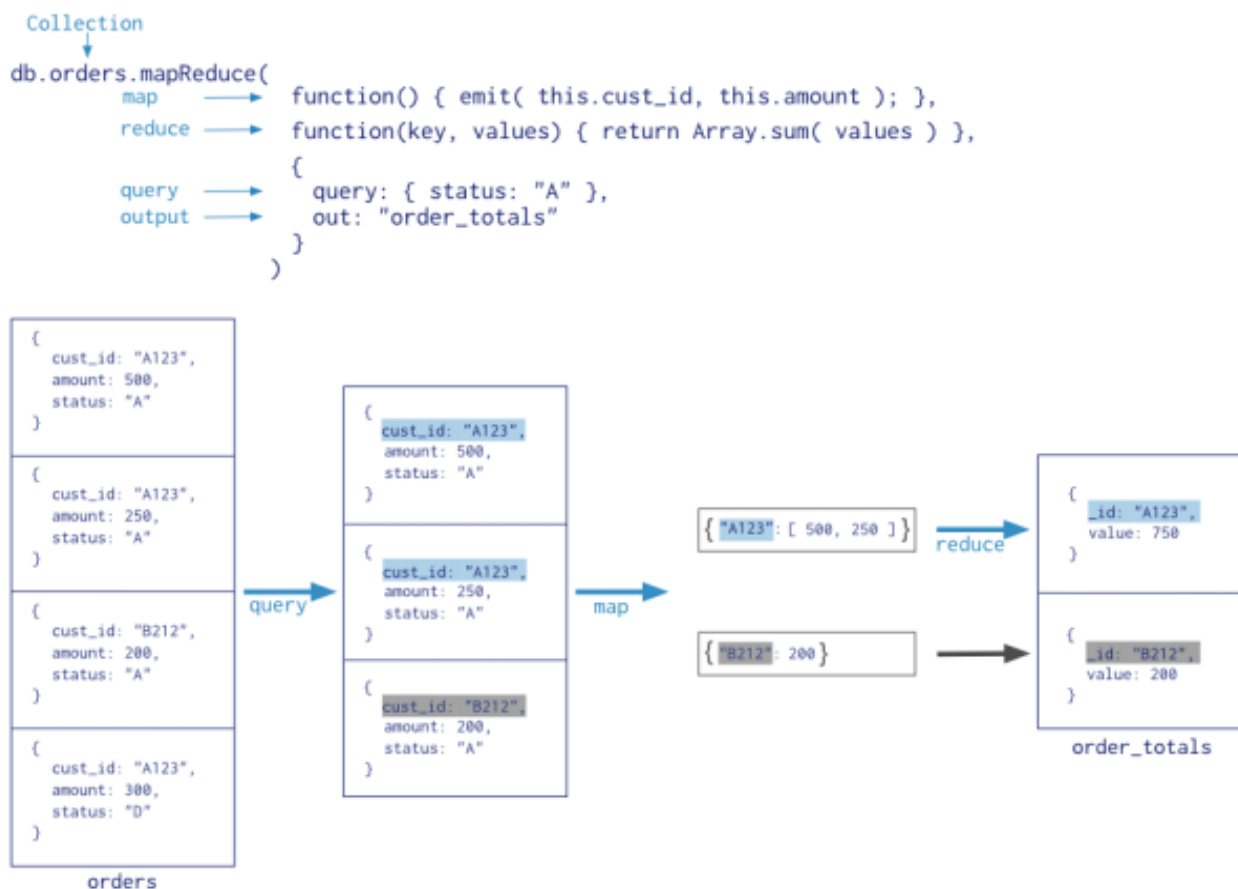
Jabuka	
Cijena	9.31
Količina	19

Jabuka	
Cijena	20.58
Količina	42



Jabuka	
Naplaćeno	69.58
Količina	142

Sada ćemo prikazati kako konkretno funkcioniše Map Reduce koncept u ovom slučaju je to u MongoDB sistemu:



Slika 3.1. Map Reduce koncept u MongoDB sistemu, izvor: [5]

U ovom primjeru možemo vidjeti da se *map* funkcija primjenjuje na svaki dokument koji ispunjava postavljene upite. U ovom slučaju upit će vratiti sve dokumente kojima je dodijeljen status "A". Napomenuli smo da za one ključeve koji imaju više vrijednosti, sistem primjenjuje i drugu fazu *Map Reduce* koncepta tj. *reduce* funkciju, koja u ovom primjeru daje zbir svih vrijednosti. Kada *reduce* funkcija prikupi, agregira i prikaže dobijene podatke u sažetom obliku sistem sprema te rezultate u "order\_totals" kolekciju. Postoji i mogućnost korištenja i *finalize* funkcije, koja već sažete rezultate *reduce* funkcije može dodatno sažeti. Može se desiti da *Map-reduce* vrati dokument kao rezultat, ali isto tako se rezultat može zapisati u kolekciju, što je ovdje slučaj.

Postoje i određeni algoritmi koji se koriste za Map Reduce model, a jedan od najpoznatijih je Apache-ov Hadoop.

**Hadoop** je radni okvir koji je otvorenog koda što znači da je besplatan i da svi imaju mogućnost da ga koriste. Zadaci koje bi trebalo da ispuni su biranje procesora za mapiranje, zatim raspodjela ključ-vrijednost parova na procesore za redukciju i garantovanje da će se zadaci uspješno izvršiti, čak i ako postoje greške u radu sistema. Svi ti prethodno navedeni zadaci su zapravo i najveći problemi sa kojima se susreću map i reduce operacije, pa su programeru aplikacije ostavljeni samo oni najjednostavniji zadaci. Većina NoSQL sistema koristi Hadoop algoritam. Izuzetak je MongoDB sistem, koji ima sopstvenu implementaciju ovog modela.

**Memcache** sistem je još jedan sistem otvorenog koda. Problem sa veličinom RAM memorije nije mogao biti u potpunosti otklonjen, iako su distribuirani sistemi ne samo dosta ubrzali nego i pojeftinili rad računara kada je u pitanju velika količina podataka. I pored toga što je zbir RAM memorije bio i do nekoliko puta veći u odnosu na jedan super-računar, on se nije mogao mnogo iskoristiti na ovakvim sistemima. Velika prednost RAM-a se ogleda u keširanju rezultata upita koje se vrlo često pojavljuju kao i u mogućnosti čuvanja onih podataka koji se najviše koriste. Znalo se dešavati da se u različitim RAM-ovima susjednih čvorova, spremaju isti podaci, zato što je u distribuiranom sistemu svaki procesor imao svoj RAM. Takođe, dešavalo se i da čvor ispočetka izvršava složene upite kao i to da dohvata veliku količinu podataka iz baze, upravo zbog toga što ne zna da je njegov susjedni čvor obradio i spremio te iste rezultate u svoj RAM. Kako bi se to izbjeglo neke od popularnih web stanica su gotovo svakodnevno morale dodavati nove čvorove u mrežu, kako bi zadovoljile potrebe velikog broja posjetilaca. To je bio jedan od glavnih razloga koji je podstakao inženjere blog sistema LiveJournal da krenu tražiti rješenje. Prvi korak je bio uvođenje specifične notacije upita zahvaljujući kojoj se cijeli upit mogao spojiti u kratki hash string. Nedugo zatim su uspješni razvili sistem komunikacije između čvorova u mreži, koji je nazvan Memcache. To je funkcionisalo na način da se putem protokola pošalje string upit svim drugim čvorovima čim dobije upit koji je potrebno izvršiti. Ukoliko neki od čvorova ima već spremljen rezultat upita u RAM-u, on taj rezultat šalje onom čvoru koji je poslao upit, čime se rješava problem spremanja istih podataka na susjedne čvorove. Nakon toga se u početni čvor spremaju rezultati upita i samim tim on uopšte nema potrebu da komunicira sa bazom. Zahvaljujući Memcache-ovoj mogućnosti da se dijeli RAM između procesora u mreži došlo je do velikog poboljšanja u funkcionalnostima distribuiranih sistema.

**Lucene** je veoma jak paket otvorenog koda čija je uloga indeksiranje i pretraživanje tekstualnih dokumenata. Iako su danas dostupni Lucene-ovi paketi u

mnogim programskim jezicima, originalni paket je napisan u programskom jeziku Postoji i Apache-ov web servis koji je posrednik prilikom komunikacije sa Lucene-om i on se naziva Solr. Ono što nam omogućava upotreba posrednika je korištenje standardnog HTTP protokola, koji nije podržan od strane Lucene-a. Pored toga, Solr može da obavlja i druge operacije kao što su replikacija, keširanje, i prevođenje dokument bilješki ukoliko postoje. Upravo zbog uske povezanosti, često dolazi do zamjene između ova dva proizvoda. Kada bismo opisivali njihov odnos nekom metaforom Solr bismo mogli predstaviti kao automobil, a Lucene kao motor bez kojeg automobil ne bi mogao da se pokrene. Lucene i Solr podržavaju tekstualne dokumente koji mogu biti u bilo kojem formatu, među kojima su i JSON, HTML, Word, PDF i slični. Gledano sa naše strane, baza šalje Solr-u vrijednosti koje je neophodno indeksirati u obliku dokumenta. Te dokumente Solr najprije šalje Lucene-u, koji rastavlja dokument na riječi i pravi indeks za svaku riječ. Ono što svaki indeks za riječ sadrži je ID riječi, koji je jedinstven, zatim broj dokumenata u kojima se ta riječ nalazi kao i same pozicije riječi u dokumentima. Svaki upit koji baza primi, neophodno je proslijediti na Solr, koji dalje uz pomoć Lucene-a vrši pretraživanje nakon kojeg vraća rezultate. Iako stvaranje ovakvih indeksa predstavlja veliki teret kada se radi o dodavanju novih dokumenta i ažuriranju, moguće ga je prihvatiti bez problema zato što taj teret ne izvršava sama baza. Solr nudi vrlo efikasno i brzo pronalaženje određeni riječi ukoliko se radi o većoj količini dokumenata. Ukoliko se desi da pretraga sadrži mnogo riječi, pronalazi se presjek skupa dokumenta u kojima se nalaze tražene riječi. Lucene se služi različitim algoritmima da bi klasifikovao rezultate npr. prilikom pretraživanja jedne riječi, kao najrelevantniji rezultat uzima onaj dokument u kojem se ta riječ najviše puta ponavlja pa će se zbog toga on nalaziti na prvom mjestu.

## Model podataka

Još jedna veoma bitna stvar koja ujedno predstavlja značajnu razliku u poređenu sa relacionim bazama podataka je NoSQL model podataka. Model podataka se može definisati kao način upravljanja podataka i njihove interpretacije.

Kada govorimo o bazama podataka, on se može predstaviti kao interakcija sa podacima u bazi, tj. organizacija podataka u bazi. NoSQL baze se prema modelu podataka dijele na četiri različita modela pri čemu svaki ima svoje karakteristike i svaki odgovara drugačijim zahtjevima u zavisnosti od toga za koju potrebu se koristi. Postoje:

1. Model ključ-vrijednost
2. Model familija kolona
3. Grafovske baze podataka
4. Dokument baze podataka

### 4.1 Ključ-vrijednost baze podataka

Jedan od najjednostavnijih modela NoSQL baza podataka je upravo ovaj model. Ono što je opšte poznato je da računari jedino razumiju mašinski jezik odnosno binarni kod. Drugačije rečeno, računar svaki podatak koji spremamo prevodi u niz nula i jedinica. Princip funkcionisanja je sličan i kod ključ–vrijednost baza. Ovakve strukture je moguće najlakše predstaviti pomoću tabele čiji su atributi ključ-vrijednost parovi(slika 4.1.), koje mogu biti bilo šta od objekta, tekstualnog dokumenta do slika, videozapisa, zapisa zvuka i sličnih stvari.

Ono što je neophodno je da se prije spremanja data vrijednost prevede u skup binarnih podataka nakon čega joj je potrebno dodijeliti zadani ključ. Binarne podatke koji su spremljeni na ovaj način nazivamo BLOB (engl. Binary Large Objects) što u prevodu znači veliki binarni objekti. Pošto bazi nije poznat tip vrijednosti kao ni sadržaj koji ona predstavlja, klijent mora samostalno da brine o kodu koji se nalazi u tom dijelu. Ključ možemo definisati kao proizvoljan string (hash) koji je generisan pomoću vrijednosti adresa web stranice ili SQL upita i koji ne prelazi maksimalnu dozvoljenu dužinu. Dopustena dužina se razlikuje u zavisnosti od implementacije. Iako je veličina BLOB-a proizvoljna, ona je ipak ograničena fizičkim prostorom za pohranu, koji je bazi na raspolaganju. Ove baze



izvršavaju i određene operacije kao što su dodavanje novog para ključa i vrijednosti, zatim ažuriranje, dohvaćanje i brisanje vrijednosti isključivo preko ključa:

- PUT(ključ, vrijednost) – dodaje par (ključ, vrijednost), a u slučaju da ključ već postoji onda se tom ključu pridružuje nova vrijednost.
- GET(ključ) – vraća vrijednost ključa.
- DELETE(ključ) – uklanja par (ključ, vrijednost). Moguće je da se javi greška u slučaju da ključ ne postoji.

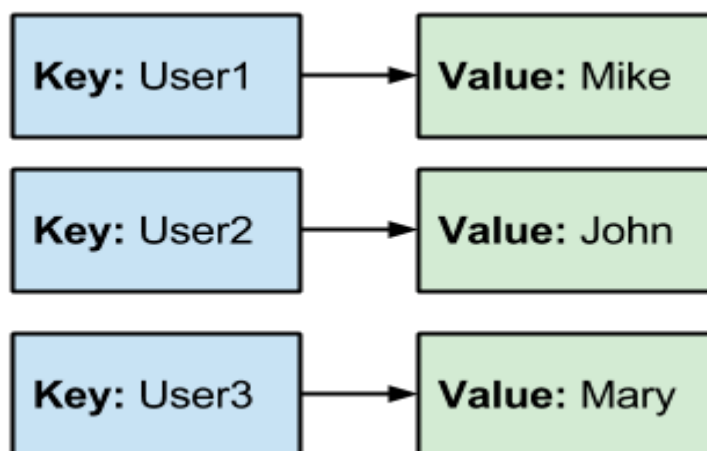
Postoje dva glavna pravila koja vrijede za ove baze podataka:

- Svaki ključ u tabeli mora biti različit.
- Nemoguće je postaviti upit na osnovu vrijednosti – Upit je moguće postaviti samo ukoliko on nije baziran na vrijednosti nego na ključu. Drugim riječima, u ovoj bazi podataka nije dozvoljeno postavljati upite pomoću kojih se traži određena vrijednost, nego samo upite na osnovu kojih se traži određeni ključ. Postoji i mogućnost definisanja roka trajanja ključa, gdje se nakon isteka roka brišu i ključ i vrijednost koja mu je dodijeljena.

Ukoliko je potrebno spremati neke korisničke podatke onda je ovaj model pravi izbor za to, jer je upravo napravljen za svrhu razmjene podataka. Često se ovaj model može objasniti implementacijom rječnika, zato što imaju velike sličnosti. Kod rječnika bi same riječi predstavljale ključeve, dok bi njihove definicije tj. objašnjenja predstavljale vrijednosti.

Znamo da su u rječniku riječi razvrstane po abecedi pa ih lako pretražujemo, što je slično načinu na koji baza pretražuje stablo indeksa. Ovaj model je veoma fleksibilan pa se zbog toga i koristi za spremanje multimedijalnog sadržaja kao što su slike, videozapisi i sl., jer je to čest problem sa kojim se susreću relacioni modeli. Ono što je još karakteristično za ovaj model je brzina čitanja, čak se i određeni relacioni sistemi služe ovim modelom kako bi mogli ubrzati pretraživanja, tako što spremaju rezultate prošlih upita (engl. query cache).

Na sljedećoj slici je prikazan primjer tabele gdje je svakom ključu pridružena odgovarajuća vrijednost:



Slika 4.1. Ključ-vrijednost tabela, izvor: [4]

Važno je pomenuti još neke karakteristike ključ-vrijednost baza podataka, a to su:

### 1. Jednostavnost

Ključ-vrijednost baze podataka odlikuje korištenje minimalne strukture podataka, tj. ukoliko je neophodno implementirati bazu u kojoj će se pohranjivati podaci o korisnicima neke kompanije ili bilo čega drugog, moguće je koristiti relacionu bazu, međutim bolja opcija je korištenje ključ-vrijednost baze podataka, iz razloga što ovaj kod ovog model nije potrebna šema. Pored toga, za attribute koji se unose nije potrebno definisati tipove podataka, što je još jedna olakšica. Ukoliko se javi potreba za unošenjem novih atributa nakon što je program već završen, moguće je jednostavno dodati programski kod koji će dodati te attribute, bez bilo kakve promjene baze. Kod ključ-vrijednost baza podataka pored toga što se koristi vrlo jednostavan model podataka, sama sintaksa za rukovanje podacima je jednostavna. Ono što je potrebno je da se definiše imenski prostor (engl. namespace), koji se odnosi ime baze, zatim ključ koji predstavlja operaciju koja će biti primijenjena na paru ključ-vrijednosti. Navođenjem samo imenskog prostora i određenog ključa, baza će kao rezultat vratiti vrijednost za taj ključ. U slučaju da je potrebno samo ažurirati vrijednost za određeni ključ, navodi se ime baze (imenski prostor), ključ i novu vrijednost koju želimo dodijeliti tom ključu. Ono što predstavlja nedostatak kod ove baze podataka je njena fleksibilnost zbog koje su moguće česte greške. Tačnije, ako se desi da greškom unesemo pogrešan tip

podatka, na primjer, cijeli broj umjesto decimalnog broja, baza podataka neće izbacivati nikakvu grešku, čime može doći do gubljenja podataka, a da toga nismo ni svjesni. Ali postoje i situacije kada ovo svojstvo može biti od koristi kao npr. kada dolazi do promjene tipa podataka ili kada je jednom atributu potrebno dodijeliti dva ili više tipova podataka. Samim tim što ovaj model spada u jednostavnije strukture baza podataka on omogućava da se operacije izvršavaju dosta brže.

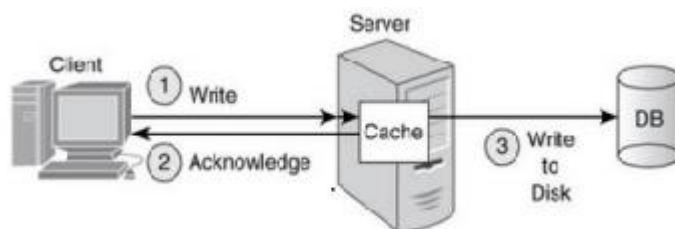
## 2. Distribuiranost

Distribuiranost je važna kako bismo mogli osigurati upravljanje novim korisnicima i podacima kada dođe do rasta aplikacije i poslovanja. Kod ključ-vrijednost baza podataka postoje dva načina distribuiranja, koji su već prethodo objašnjeni u drugom poglavlju:

1. Master–slave replikacija
2. Master – master replikacija

## 3. Brzina

Već je prethodno napomenuto da su ključ-vrijednost baze podataka karakteristične po svojoj brzini. Upravo time što imaju jednostavan niz strukture podataka kao i svojstva dizajna koja služe za poboljšavanje performansi, ključ-vrijednost baze podataka omogućavaju veliku brzinu prilikom izvođenja intenzivnih operacija podataka. Jedan od glavnih načina za održavanje brzine rada baze podataka je da se podaci zadrže u memoriji, jer je čitanje i pisanje u RAM-u (engl. Random Access Memory) mnogo brže u odnosu na pisanje na disk. RAM memorija nije trajna memorija što znači da ukoliko dođe do nestanka napajanja automatski će biti izgubljen sadržaj RAM-a. Kada u program dođe do promjene vrijednosti ključa, baza to ažurira u RAM i šalje poruku programu u kojem ga obavještava da je nova vrijednost sačuvana, a program nastavlja sa drugim operacijama, dok za to vrijeme baza zapisuje novu, ažuriranu vrijednost na disk.



Slika 4.2. Pisanje i ažuriranje podataka prvo u RAM memoriju, a zatim na disk, izvor: [3]

Drugi način na koji je moguće postići veliku brzinu prilikom rada sa podacima je SSD (engl. Solid State Drive) memorija. To je veoma brza flash memorija, koja služi za pohranu velikih sinhronih zapisa, što je dosta korisno jer u tom slučaju RAM može biti iskorišten za neke nove podatke. Veliki broj baza podataka podržava SSD pohranu kako bi se osigurao maksimalni protok podataka . Uprkos velikim prednostima postoje i situacije u kojima je bolje izbjeći korištenje ključ-vrijednost baze podataka. Neke od njih su sljedeće:

- Ukoliko bi trebalo povezati skupove podataka, koji se dosta razlikuju ili povezati podatke koji se nalaze između različitih skupova ključeva.
- Kada je potrebno vratiti neku od operacija kada dođe do neuspješnog spremanja ključeva.
- Kada je neophodno pronaći ključ na osnovu upita koji je baziran na vrijednosti iz ključ-vrijednost para.
- Kada postoji potreba da se upravlja većim brojem ključeva.

Međutim postoje i slučajevi u kojima je korištenje ključ-vrijednost baze podataka jedan od najboljih izbora, a neki od njih su:

- Ukoliko je potrebno spremiti podatke korisničkih profila, informacije o određenim sesijama kao i podatke elektronske pošte
- Kada je potrebno spremiti podatke npr. u slučaju Internet trgovine (sadržaji korpe, kategorije proizvoda, detalji o proizvodu, recenzije proizvoda).
- Kada je potrebno osigurati povjerljivost podataka
- Kod internet protokola, itd.

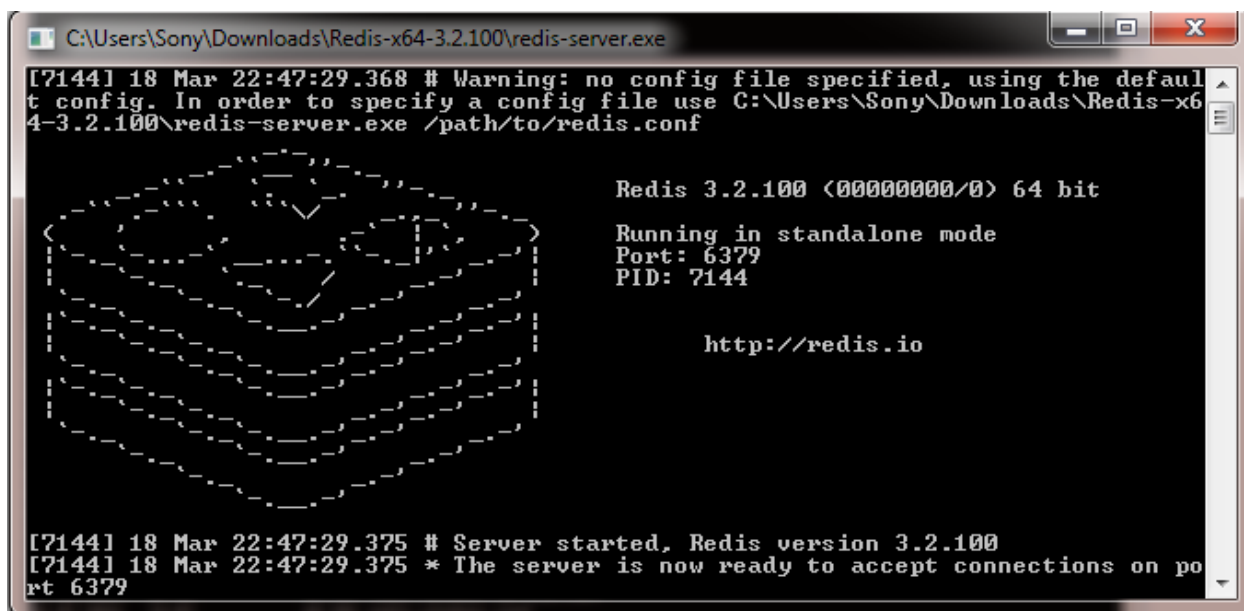
Predstavnici koji su bazirani na ovom modelu su: DynamoDB, Riak, Redis, Oracle NoSQL i mnogi drugi.

### 4.1.1 Redis baza podataka

Kada govorimo o ključ-vrijednost bazama možemo reći da je Redis jedna od najpoznatijih open-source baza podataka ovog tipa. Ova baza radi i sa apstraktnim strukturama podataka poput nizova, lista, mapa, hash tabele itd. Kod Redis-a se postiže velika brzina prilikom izvođenja operacija pisanja tako što se ograničava veličina podataka tj. oni ne mogu biti veći u odnosu na memoriju. Kada dođe do nedostatka memorije Redis proces se prekinut, odnosno moguće je da Redis javi grešku o memoriju i da se sruši ili da jednostavno uspori. Ova baza je jednonitna baza, što znači da sije moguće paralelno izvršavati zadatke. U Redis postoji mogućnost da se podaci spremaju nakon određenog broja sekundi npr. moguće je premanje podataka nakon 500 promjena i najmanje 10 sekundi. Posljedica asinhronog spremanja podataka je to da postoji mogućnost gubljenja poslednjih promjena ukoliko dođe do pada sistema. Kako bi se to spriječilo Redis koristi master-slave replikaciju još od samog nastanka. Ono što je važno je da ova baza drži sve podatke u memoriji što je čini najbržom bazom ovog tipa. Zbog mogućih gubitaka pogodana je za one aplikacije u kojima manji gubitak podataka ne predstavlja veliki problem.

- **Instalacija Redis-a**

Iako je Redis na početku bio namijenjen samo za korisnike Linux operativnog sistema, zahvaljujući timu pod nazivom MSOpenTech čiji je zadatak rad sa tehnologijama otvorenog tipa, danas postoji mogućnost korištenja ovog sistema i na Microsoft operativnom sistemu. Ukoliko želimo da preuzmemo Redis za Microsoft Windows, potrebno je otići na [stranicu Github-a](#) ovog tima, a zatim preuzeti najnoviju verziju. Postoji mogućnost preuzimanja samo 64-bitna verzije. Iako je verzija za Windows malo drugačija u odnosu na verziju Linux operativnog sistema između ostalog zbog portova koje koriste, nema razlike prilikom korištenja samih naredbi. Kako bi se pokrenuo Redis potrebno je pokrenuti i server i klijent koji se nalaze u direktorijumu C:\Program Files\Redis. Tu postoji i datoteka u kojoj se može mijenjati port, IP adresa, TCP keepalive kao i ostala podešavanja. Zbog činjenice da se Redis često koristi kao predmemorija, nije potrebno podatke čuvati trajno, pa je u konfiguraciji moguće podesiti koliko često se pohranjuje podaci i slično.



Slika 4.3. Pokretanje servera, vlastiti izvor

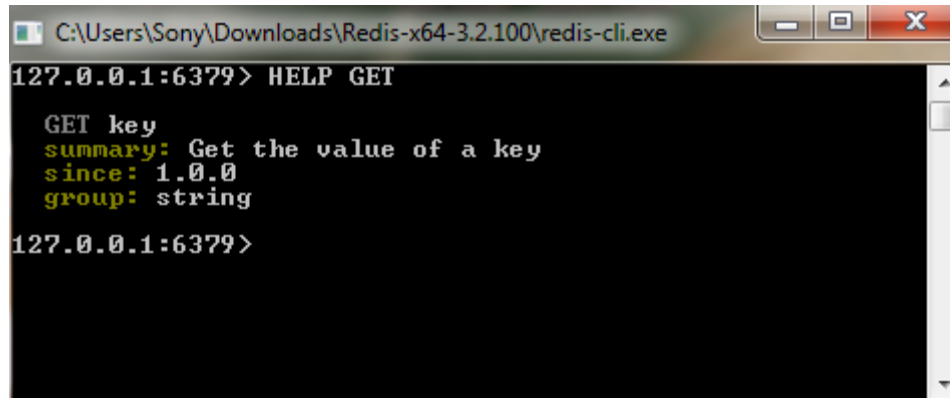
## Naredbe u Redis-u

Kada je riječ o pisanju Redis naredbi imamo dvije mogućnosti. Moguće je samostalno kreirati nešto, što je moguće uraditi jedino ukoliko izvršimo instalaciju a zatim pokrenemo i klijent i server u prethodno navedenom direktorijumu. Ali isto tako postoji i opcija pisanja naredbi uz pomoć njihovog [interaktivnog vodiča](http://redis.io) na web stranici gdje je omogućeno pisanje svih naredbi što je veoma dobar izbor ukoliko neko želi samo da se upozna sa osnovnim radom ove baze. Nakon pokretanja tog vodiča, korisnik ima mogućnost da pristupi primjerima i objašnjenjima komandi, koje su najvažnije za shvatanje ovog sistema. Ono što je dobra stvar je sama organizacija naredbi. One su podijeljene u srodne cjeline kako bi početnici imali mogućnost da se lakše snađu.

### • HELP

Kako bi se izbjegla zamjena u korištenju naredbi koje su veoma slične kao i pogrešna sintaksa prilikom pisanja, Redis nudi pomoć (help) prilikom pisanja samih naredbi. Pomoć se poziva tako što se ukuca "HELP", a zatim naredba za koju želimo vidjeti sintaksu. Na slici ispod se može vidjeti kako izgleda „HELP“ i koje sve opcije sadrži. Ukoliko izaberemo pomoć za naredbu GET vidimo da prvo dobijemo sintaksu naredbe: „GET key“, gdje GET označava naredbu, a „key“ ključ koji je prethodno postavljen pomoću naredbe SET koja će biti objašnjena u

nastavku. Pored sintakse postoji i opis naredbe “summary”, pa vidimo da se u ovom slučaju pomoću GET dobija vrijednost traženog ključa. Poslije opisa imamo i mogućnost da vidimo od koje verzije Redis-a se koristi zadata naredba, a nakon toga možemo vidjeti i kojoj grupi naredba pripada, što je u ovom slučaju grupa stringova.



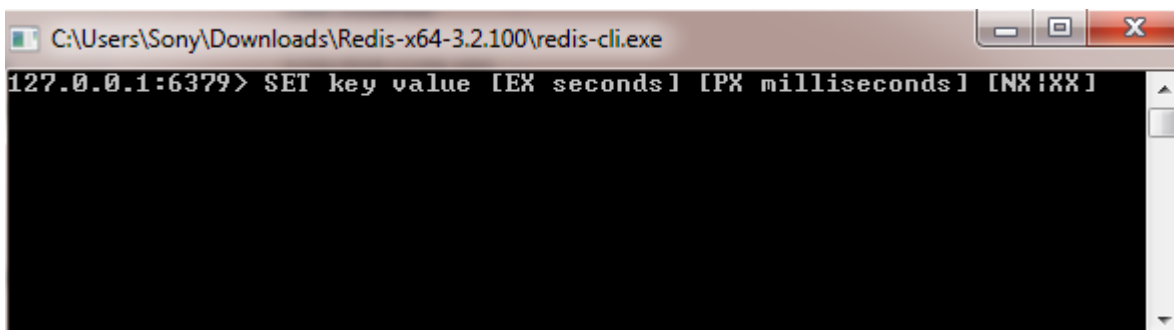
```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe
127.0.0.1:6379> HELP GET

GET key
summary: Get the value of a key
since: 1.0.0
group: string
127.0.0.1:6379>
```

Slika 4.5. Pomoć u Redis-u, vlastiti izvor

## • Povratna poruka

Ono što je još karakteristično za ove baze jeste povratna poruka, koju vraća klijent poslije svakog izvršavanja naredbe. Kod Redis-a se povratna poruka interpretira u obliku 0,1 ili OK. Poruku dobijamo u zavisnosti od toga koju smo naredbu koristili kao i od toga da li je ta naredba uspješno izvršena. Ukoliko je sve proteklo kako bi trebalo tj. ukoliko je naredba uspješno izvršena klijent će vratiti „OK“. “OK” će najčešće biti vraćeno u povratnoj poruci kada se koriste bool (true/false) operacije i naredbe. U slučaju da se radi o nečemu komplikovanijem, biće vraćena nula ili jedinica. Svaki put kada se izvrši naredba GET klijent će vratiti poruku u vidu vrijednosti (u slučaju da postoji vrijednost) koja je spremljena pod ključem. Velika prednost koja je omogućena novom verzijom je pomoć prilikom pisanja naredbi, gdje odmah pri kucanju naredbe klijent nudi sintaksu same naredbe:



```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe
127.0.0.1:6379> SET key value [EX seconds] [PX milliseconds] [NX|XX]
```

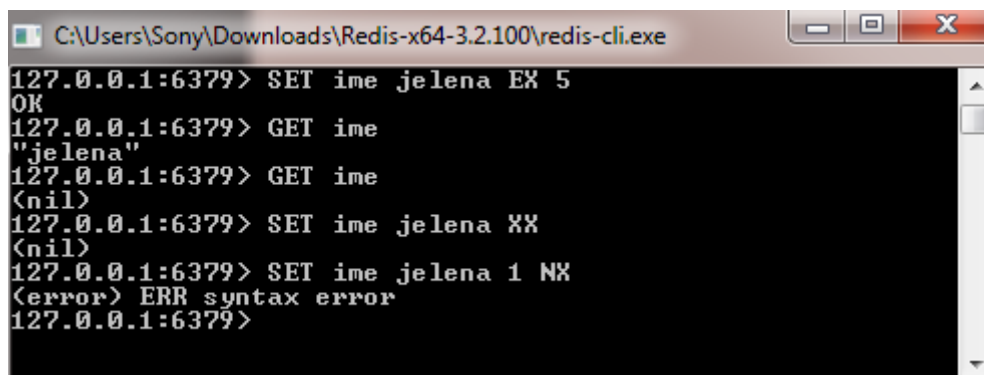
Slika 4.6. Pomoć pri pisanju komandi, vlastiti izvor

## • Naredba SET

Sintaksa naredbe SET je SET „ključ“ „vrijednost“ i ona se koristi ukoliko želimo dodijeliti neku vrijednost zadatom ključu. Vrijednost će biti sačuvana u obliku niza. Ako se desi da taj ključ već ima određenu vrijednost, onda dolazi do zamjene te vrijednosti novom vrijednošću koja je navedena. Moguće je da dođe do zamjene i ukoliko se radi o drugom tipu vrijednosti. Verzijom 2.6.12. , uspostavljene su dodatne opcije naredbe SET:

1. EX sekundi – koristimo kada želimo postaviti vrijeme isticanja
2. PX milisekundi – vrijeme isticanja u milisekundama
3. NX – pridružuje vrijednost ključu ukoliko ključ već ne sadrži vrijednost
4. XX – pridružuje vrijednost ključu ukoliko ključ već sadrži neku vrijednost

Ove opcije nisu neophodne, te se mogu i ukloniti zato što postoje naredbe SETNX, SETEX, PSETEX, koje rade isto što i prethodne, samo na dosta jednostavniji način, a u isto vrijeme smanjuje se broj riječi koji je neophodno iskoristiti. Na slici 4.7. implementirane su prethodno objašnjene naredbe. Prvom naredbom postavljamo vrijednost ključa ime sa isticanjem od 5 sekundi, nakon kojeg će dodijeljena vrijednost biti automatski obrisana. Isto tako ukoliko navedeni ključ više ne postoji, dobijamo povratnu poruku „nil“, čime se označava da ključu nije dodijeljena nikakva vrijednost.



```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe
127.0.0.1:6379> SET ime jelena EX 5
OK
127.0.0.1:6379> GET ime
"jelena"
127.0.0.1:6379> GET ime
<nil>
127.0.0.1:6379> SET ime jelena XX
<nil>
127.0.0.1:6379> SET ime jelena 1 NX
<error> ERR syntax error
127.0.0.1:6379>
```

Slika 4.7. Primjer korištenja GET, SET (sa opcijama NX, XX), vlastiti izvor

## • Naredba GET

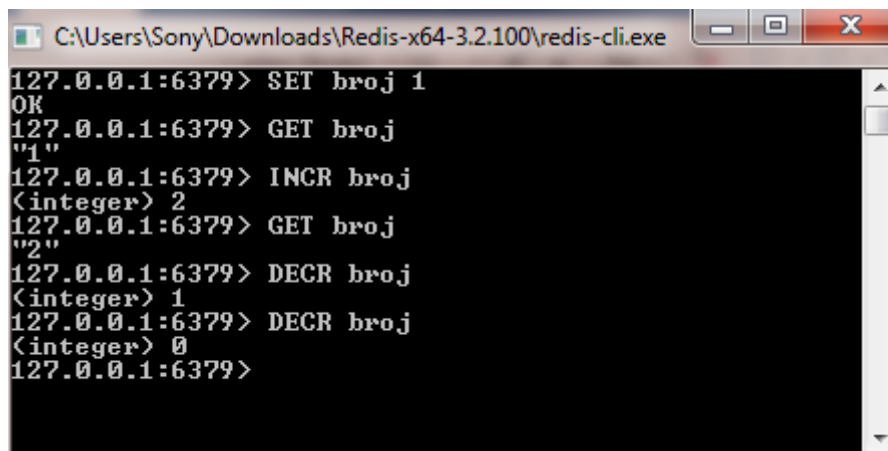
Sintaksa naredbe GET je GET „ključ“ i ona služi za dobijanje vrijednosti određenog ključa. Kao što je prethodno napomenuto u slučaju da tom ključu nije



dodijeljena nikakva vrijednost, klijent će vratiti povratnu poruku „nil“. Primjer i implementacija ove naredbe prikazani su na prethodnoj slici.

- **Naredba INCR**

Sintaksa naredbe INCR je INCR „ključ“. Ovu naredbu koristimo ukoliko je potrebno vrijednost broja koji se skladišti u ključu povećati za 1. Moguće je da traženi ključ ne postoji, pa se u tom slučaju ključu dodijeljuje vrijednost 0. Redis će vratiti grešku ukoliko se ova naredba primijeni na vrijednost koja nije broj. Ovo predstavlja i operaciju nad nizom, jer se vrijednosti čuvaju u niz. Na slici 4.8. prikazan je primjer naredbi INCR i DECR. Vidimo da se nakon primjenjivanja naredbe INCR nad vrijednošću jedan dobija vrijednost dva. Važno da napomenuti da SET uprkos tome što sprema broj kao niz ima mogućnost da ga prepozna pa ga je samim tim moguće uvećati ili umanjiti u zavisnosti od potrebe. Ova naredba može biti veoma korisna kada je potrebno izbrojati zahtjeve koje je potrebno poslati. Ona omogućava da se ograniči broj zahtjeva koji korisnici mogu poslati u sekundi kako bi se spriječilo preopterećenje servera. Moguće je da korisnik pređe to ograničenje, ali u tom slučaju mu se javlja greška i dolazi do blokiranja korisnikovog računa. Pored ove naredbe neophodno je koristiti i naredbu EXPIRE koja bi bila zadužena za brisanje vrijednosti dodijeljene ključu nakon određenog broja sekundi.



```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe
127.0.0.1:6379> SET broj 1
OK
127.0.0.1:6379> GET broj
"1"
127.0.0.1:6379> INCR broj
(integer) 2
127.0.0.1:6379> GET broj
"2"
127.0.0.1:6379> DECR broj
(integer) 1
127.0.0.1:6379> DECR broj
(integer) 0
127.0.0.1:6379>
```

Slika 4.8. Naredbe INCR i DECR, vlastiti izvor

- **Naredba DECR**

Sintaksa naredbe DECR je DECR „ključ“. Za razliku od naredbe INCR, koja uvećava vrijednost ključa za jedan ova naredba umanjuje vrijednost ključa za 1.

Rad naredbe DECR može se vidjeti također na prethodnoj slici. Postoje i varijacije prethodne dvije naredbe u obliku INCRBY, INCRBYFLOAT, DECRBY i DECRBYFLOAT koje isto služe za smanjivanje ili povećavanje određenog broja za zadatu veličinu.

- **Naredba APPEND**

Sintaksa naredbe APPEND je APPEND „ključ“ „vrijednost“. Pomoću ove naredbe moguće je dodati određenu vrijednost na već postojeću. Ukoliko se radi o brojevima određena vrijednost se dodaje ne u smislu sabiranja, nego spajanja brojeva. Kada se ova naredba izvrši klijent će vratiti novonastalu vrijednost. Korištenje ove naredbe prikazano je na slici 4.9. na kojoj se može vidjeti da se nakon dodavanja broja 7 broju 5 dobije broj 57. Nakon toga je moguće primijeniti naredbe za umanjevanje ili uvećavanje vrijednosti, jer će ta vrijednost biti posmatrana u obliku broja što se vidi i u priloženom:



```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe
127.0.0.1:6379> SET kljuc hello
OK
127.0.0.1:6379> APPEND kljuc world
(integer) 10
127.0.0.1:6379> GET kljuc
"hello world"
127.0.0.1:6379> SET broj 5
OK
127.0.0.1:6379> APPEND broj 7
(integer) 2
127.0.0.1:6379> GET broj
"57"
127.0.0.1:6379> INCR broj
(integer) 58
127.0.0.1:6379>
```

Slika 4.9. Naredba APPEND, vlastiti izvor

- **Naredba GETRANGE**

Sintaksa naredbe GETRANGE je GETRANGE „početna vrijednost“ „krajnja vrijednost“. Ova naredba se koristi ukoliko je potrebno dobiti vrijednosti u određenom rasponu što može biti veoma korisno ukoliko unaprijed znamo nešto što se nalazi u tom ključu. Kao primjer možemo uzeti JMBG jer se on sastoji od većeg broja podataka poput datuma rođenja. Ukoliko je iz JMBG-a potrebno saznati datum rođenja to je moguće tako što ćemo kao raspon navesti interval brojeva JBMG-a iz kojeg je moguće zaključiti o kojem datumu se radi.

- **Naredba SETRANGE**

Sintaksa naredbe SETRANGE je SETRANGE „ključ“ „početak“ „vrijednost“. Ova naredba uzima zadatu vrijednost i zapisuje je u ključ počevši od pozicije koja je navedena kao početak. Na primjer, ako imamo vrijednost ključa 0123456789 kao u primjeru na slici 4.10. i ako izvršimo naredbu „SETRANGE ključ 4 012345“, vrijednost 012345 biće zapisana u vrijednost ključa počevši od četvrtog mjesta i tada ćemo dobiti novu vrijednost 0123012345 koje će biti zapisana u tom ključu.



```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli...
127.0.0.1:6379> SET kljuc 0123456789
OK
127.0.0.1:6379> GETRANGE kljuc 1 7
"1234567"
127.0.0.1:6379> SETRANGE kljuc 4 012345
(integer) 10
127.0.0.1:6379> GET kljuc
"0123012345"
127.0.0.1:6379>
```

Slika 4.10. Naredbe GETRANGE i SETRANGE, vlastiti izvor

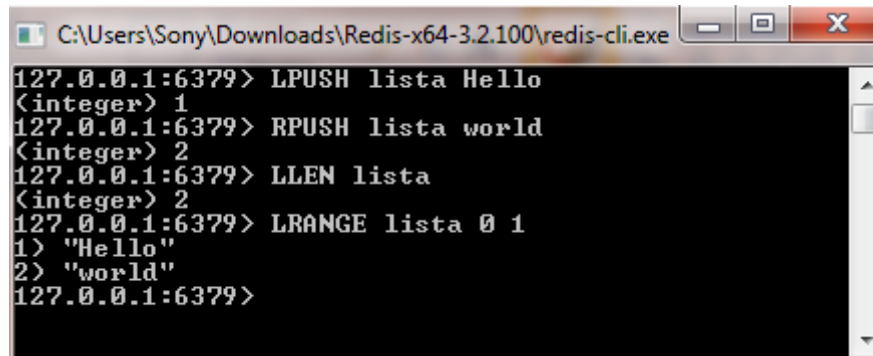
- **Liste**

Ono što je još karakteristično za Redis je to da rade sa listama nizova. Moguće je izvršavati operacije koje su opšte poznate kada je riječ o nizovima, a neke od njih su dodavanje i uklanjanje vrijednosti koje se nalaze na određenim indeksima.

Naredbe koje su najvažnije za rad sa listama biće detaljno objašnjene i implementirane u nastavku:

- **Naredba LPUSH**

Sintaksa naredbe LPUSH je LPUSH „ključ“ „vrijednost,„. Njena uloga je da dodaje vrijednost ključa na početak liste. Ako se desi da ključ ne postoji, napraviće se prazna lista pa će se tek onda dodati vrijednost na sam početak liste. Od verzije 2.4. pa na dalje, dozvoljeno je unošenje više vrijednosti koje se smještaju jedna za drugom i te vrijednosti se unose na identičan način samo što se umjesto jedne vrijednost unosi više vrijednosti. Gledajući sa lijeva na desno posljednja vrijednost se smješta na početak tj. na početno mjesto lijeve strane, a onda se svaka sljedeća smješta jedna za drugom, u redoslijedu u kojem su i navedene. Kada se naredba u potpunosti izvrši kao rezultat dobijamo broj elemenata koji se nalazi u listi.



```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe
127.0.0.1:6379> LPUSH lista Hello
(integer) 1
127.0.0.1:6379> RPUSH lista world
(integer) 2
127.0.0.1:6379> LLEN lista
(integer) 2
127.0.0.1:6379> LRANGE lista 0 1
1) "Hello"
2) "world"
127.0.0.1:6379>
```

Slika 4.11. Naredbe LPUSH, RPUSH, LLEN, LRANGE, vlastiti izvor

➤ **Naredba RPUSH**

Ova naredba funkcionise na isti način kao i LPUSH, jedina razlika je u tome što ona umjesto sa početka dodaje elemente na kraj liste tj. sa desne strane. Način funkcionisanja ove naredbe prikazan je na prethodnoj slici.

➤ **Naredba LLEN**

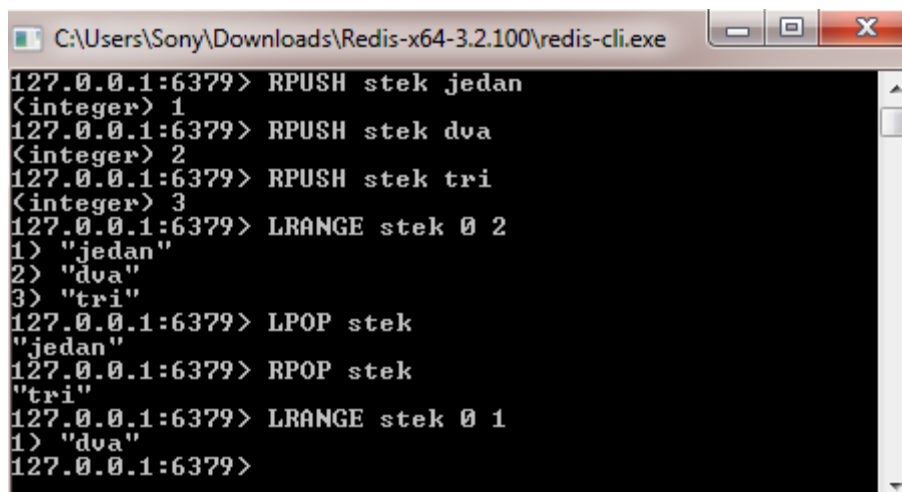
Sintaksa ove naredbe je LLEN „ključ“. Pomoću ove naredbe je moguće saznati broj elemenata koji se nalaze u samoj listi odnosno veličinu liste.

➤ **Naredba LRANGE**

Sintaksa ove naredbe je LRANGE „ključ“ „početna vrijednost“ „krajnja vrijednost“. Ova naredba kao rezultat vraća elemente liste koji se nalaze na početnim i krajnjim indeksima, ali i elemente koji se nalaze između njih. Kao što se može vidjeti na slici 4.11., naredba „LRANGE lista 0 1“ će vratiti elemente liste koji se nalaze na nultom i prvom indeksu, naravno u ovom slučaju ne postoje elementi koji se nalaze između ove dvije vrijednosti pa će zbog toga biti vraćene isključivo navedene dvije vrijednosti.

➤ **Naredbe LPOP, RPOP**

Ove komande implementiraju način građenja steka odnosno reda. Pomoću naredbe LPOP „ključ“ ispisuje se vrijednost ključa koja se nalazi na početnoj poziciji gledano sa lijeve strane dok se pomoću naredbe RPOP ispisuje posljednja vrijednost. Na slici je prikazano formiranje liste stek u koju se redom ubacuju vrijednosti jedan, dva i tri, a zatim se ispisuje dužina liste kao i početna i krajnja vrijednost u listi.

A screenshot of a Windows command prompt window titled "C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe". The window shows a series of Redis commands and their outputs. The commands are: 1. "RPUSH stek jedan" which returns "(integer) 1". 2. "RPUSH stek dva" which returns "(integer) 2". 3. "RPUSH stek tri" which returns "(integer) 3". 4. "LRANGE stek 0 2" which returns a list: "1) 'jedan'", "2) 'dva'", "3) 'tri'". 5. "LPOP stek" which returns "'jedan'". 6. "RPOP stek" which returns "'tri'". 7. "LRANGE stek 0 1" which returns a list: "1) 'dva'". The prompt "127.0.0.1:6379>" is visible at the start of each command line.

```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe
127.0.0.1:6379> RPUSH stek jedan
(integer) 1
127.0.0.1:6379> RPUSH stek dva
(integer) 2
127.0.0.1:6379> RPUSH stek tri
(integer) 3
127.0.0.1:6379> LRANGE stek 0 2
1) "jedan"
2) "dva"
3) "tri"
127.0.0.1:6379> LPOP stek
"jedan"
127.0.0.1:6379> RPOP stek
"tri"
127.0.0.1:6379> LRANGE stek 0 1
1) "dva"
127.0.0.1:6379>
```

Slika 4.12. Naredbe LPOP, RPOP, vlastiti izvor

#### ➤ **Naredba LINDEX**

Sintaksa ove naredbe je LINDEX „ključ“ „indeks“. Naredba LINDEX kao rezultat vraća vrijednost koja se nalazi na zadatom indeksu. Naravno postoji mogućnost i da korisnik ne zna od koliko se elemenata sastoji lista, pa je u tom slučaju za dohvaćanje posljednjeg elementa dozvoljeno koristiti -1, -2 za dohvaćanje elementa koji se u listi nalazi prije posljednjeg itd.

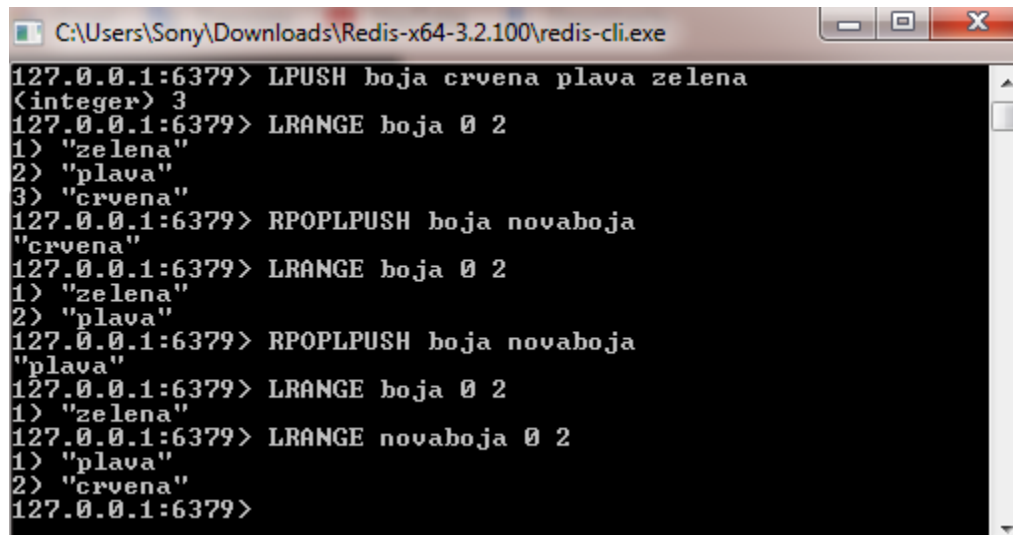
#### ➤ **Naredba LTRIM**

Sintaksa naredbe je LTRIM „ključ“ „početni indeks“ „krajnji indeks“. Ova naredba briše elemente iz liste krećući od početnog pa sve do krajnjeg indeksa. Pomoću nje je moguće obrisati kako jedan element tako i kompletnu listu. I ovdje je dozvoljeno korištenje negativnih indeksa kao što je to bio slučaj i kod prethodne naredbe.

#### ➤ **Naredba RPOPLPUSH**

Sintaksa naredbe je RPOPLPUSH „lista1“ „lista2“. Ova naredba uzima posljednji element iz liste1 i postavlja ga na prvo mjesto u listi2. Ukoliko lista1 ne postoji klijent vraća vrijednost „nil“. Takođe, u slučaju da navedemo istu listu i za lista1 i za lista2 onda će se uz pomoć ove naredbe uzeti posljednji element iz te liste i prebaciti na prvo mjesto, čime možemo zaključiti da ova naredba zapravo rotira elemente. Redis je moguće koristiti i u svrhu servera čija bi uloga bila slanje i primanje poruka, kao i njihova obrada. Na tom primjeru možemo objasniti način funkcionisanja steka, jer se poruke spremaju u listu, a zatim čekaju da dođu na red kako bi se obradile.

Iako se za to može koristiti naredba RPOP, ona nije preporučljiva jer nije dovoljno pouzdana, zato što se poruke, koje nisu stigle da se obrade, gube u slučaju nestanka interneta ili napona. Kako se to ne bi dešavalo koristi se naredba RPOPLPUSH, jer tu u tom slučaju prima poruka i postavlja u listu. Onda kada se obrada u potpunosti završi naredba LREM uklanja tu poruku iz liste, što je mnogo pouzadiniji način od korištenja naredbe RPOP. Primjer korištenja kao i sam rezultat prikazani su na slici 4.13.



```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe
127.0.0.1:6379> LPUSH boja crvena plava zelena
(integer) 3
127.0.0.1:6379> LRANGE boja 0 2
1) "zelena"
2) "plava"
3) "crvena"
127.0.0.1:6379> RPOPLPUSH boja novaboja
"crvena"
127.0.0.1:6379> LRANGE boja 0 2
1) "zelena"
2) "plava"
127.0.0.1:6379> RPOPLPUSH boja novaboja
"plava"
127.0.0.1:6379> LRANGE boja 0 2
1) "zelena"
127.0.0.1:6379> LRANGE novaboja 0 2
1) "plava"
2) "crvena"
127.0.0.1:6379>
```

Slika 4.13. Naredba RPOPLPUSH, vlastiti izvor

### ➤ Naredba EXPIRE

Pomenuto je da se Redis koristi kao stek, međutim pored steka on se koristi i kao predmemorija. Napomenuli smo da se podaci koji su sačuvani u predmemoriji, ne čuvaju trajno nego privremeno, a kako bi se to efikasno odradilo koriste se naredbe EXPIRE, TTL i PERSIST. Zahvaljujući ovim naredbama parovi ključ-vrijednost se mogu obrisati nakon što istekne zadato vremena. Sintaksa ove naredbe je EXPIRE „ključ“ „broj predstavljen u sekundama“. Kada se izvrši ova naredba podaci koji su bili zapisani u predmemoriji će biti obrisani nakon što istekne vrijeme koje je predstavljeno u sekundama. Moguće je ovu naredbu primijeniti i na listama i u tom slučaju se vrijeme nasljeđuje. Prikaz rada ove komande se može vidjeti na slici ispod. Ukoliko primijenimo ovu naredbu i sačekamo određen broj sekundi dobićemo poruku da je lista kojoj smo pokušali pristupiti prazna, što znači da je naredba efikasno izvršena.

```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli...
127.0.0.1:6379> LRANGE lista 0 3
1) "Hello"
2) "world"
127.0.0.1:6379> EXPIRE lista 5
(integer) 1
127.0.0.1:6379> LRANGE lista 0 2
1) "Hello"
2) "world"
127.0.0.1:6379> LRANGE lista 0 2
(empty list or set)
127.0.0.1:6379>
```

Slika 4.14. Naredba EXPIRE, vlastiti izvor

### ➤ Naredbe TTL i PERSIST

TTL (eng. Time To Live) odnosno vrijeme postojanja je naredba pomoću koje možemo vidjeti za koliko sekundi će lista ili ključ biti obrisani. Ukoliko želimo da uklonimo vrijeme koristimo naredbu PERSIST. Na slici se vidi detaljan princip funkcionisanja ovih komandi. Negativna vrijednost (-1) će biti vraćena u slučaju da vrijeme nije postavljeno.

```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe
127.0.0.1:6379> LRANGE lista 0 2
1) "element2"
2) "element1"
127.0.0.1:6379> EXPIRE lista 150
(integer) 1
127.0.0.1:6379> TTL lista
(integer) 147
127.0.0.1:6379> PERSIST lista
(integer) 1
127.0.0.1:6379> TTL lista
(integer) -1
127.0.0.1:6379>
```

Slika 4.15. Naredbe TTL i PERSIST, vlastiti izvor

## • Sadržaj

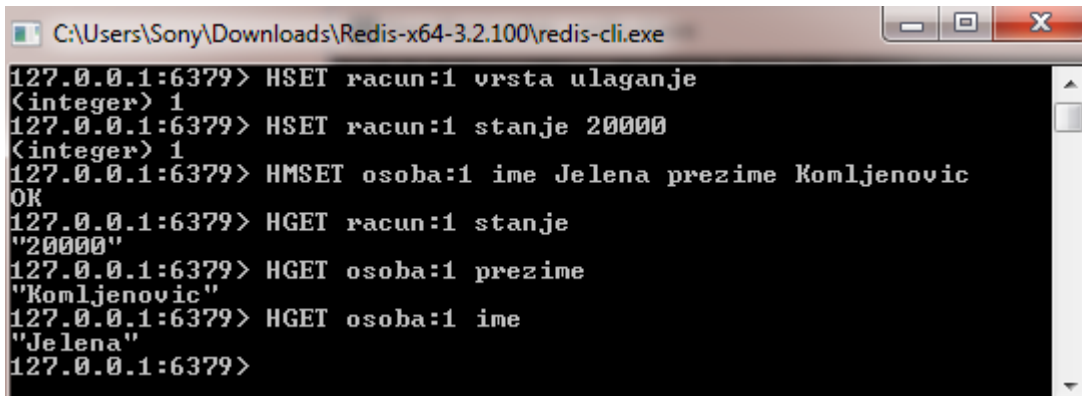
Redis pohranjuje i dosta kompleksnije tipove podataka za razliku od listi, a jedan od tih tipova su i sadržaji. Sadržaji možemo definisati kao kolekcije parova ključ-vrijednost. Najviše se koriste kada se radi o objektima ili nekim tabličnim podacima. Pored toga što zauzima veoma malo memorijskog prostora, on može da skladišti ogroman broj elemenata čak i preko 4 milijarde. U nastavku će biti objašnjene osnovne komande kada je riječ o sadržaju:

### ➤ Naredba HSET

Sintaksa naredbe je HSET „ključ“ „polje“ „vrijednost“. Uz pomoć ove naredbe polju se dodjeljuje određena vrijednost.



Na slici 4.16. se može da je uz pomoć ove naredbe kreiran objekat račun sa ID-em 1 tipa vrsta čija je vrijednosti ulaganje, dok je uz pomoć naredbe HSET kreiran objekat račun sa ID-em 1, čije stanje iznosi 20 000. Ukoliko je potrebno unijeti veliki broj vrijednosti odjednom to se može učiniti korištenjem naredbe HMSET.



```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe
127.0.0.1:6379> HSET racun:1 vrsta ulaganje
(integer) 1
127.0.0.1:6379> HSET racun:1 stanje 20000
(integer) 1
127.0.0.1:6379> HMSET osoba:1 ime Jelena prezime Komljenovic
OK
127.0.0.1:6379> HGET racun:1 stanje
"20000"
127.0.0.1:6379> HGET osoba:1 prezime
"Komljenovic"
127.0.0.1:6379> HGET osoba:1 ime
"Jelena"
127.0.0.1:6379>
```

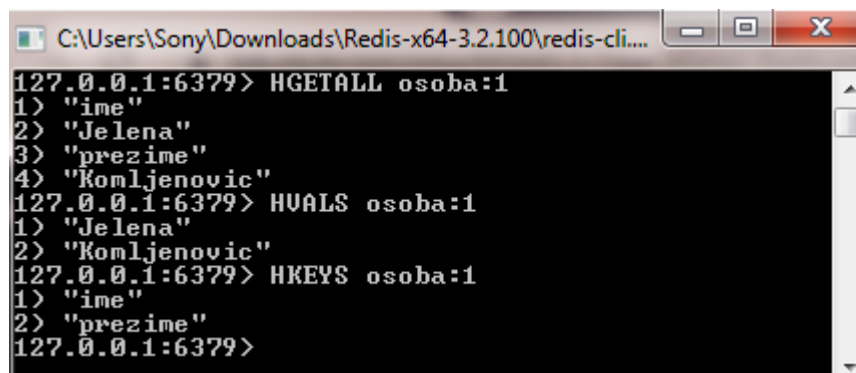
Slika 4.16. Naredbe HSET, HMSET i HGET, vlastiti izvor

#### ➤ **Naredba HGET**

Sintaksa ove naredbe je HGET „ključ“ „polje“. Ova naredba kao rezultat vraća vrijednost polja navedenog ključa. Kao što se vidi na slici 4.16. naredba „HGET osoba:1 ime“ će vratiti ime osobe čiji je ID 1.

#### ➤ **Naredba HGETALL**

Sintaksa naredbe HGETALL je HGETALL „ključ“. HGETALL će redom vratiti sve parove ključ-vrijednosti navedenog ključa. Ova naredba će vratiti vrijednosti jednu ispod druge zato što Redis sve pohranjuje kao niz. U slučaju da je potrebno da se saznaju samo ključevi ili samo vrijednosti onda se koriste naredbe HVALS (za vrijednosti) i HKEYS (za ključeve).



```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe
127.0.0.1:6379> HGETALL osoba:1
1> "ime"
2> "Jelena"
3> "prezime"
4> "Komljenovic"
127.0.0.1:6379> HVALS osoba:1
1> "Jelena"
2> "Komljenovic"
127.0.0.1:6379> HKEYS osoba:1
1> "ime"
2> "prezime"
127.0.0.1:6379>
```

Slika 4.17. Naredbe HGETALL, HVALS i HKEYS, vlastiti izvor



➤ **Naredba HINCRBY**

Naredbom HINCRBY se uvećava vrijednost polja. Kod sadržaja su prisutne i sve ostale naredbe koje su prethodno objašnjene kod lista, pa zbog toga nema potrebe da se iznova objašnjavaju i implementiraju. Neke od njih su: HINCRBYFLOAT, HLEN, HEXISTS, HDEL, EXPIRE, TTL itd.

• **Skupovi**

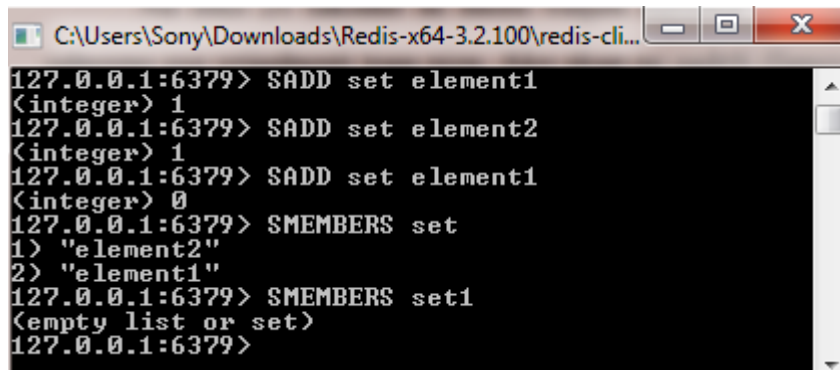
Još jedan tip jesu skupovi (eng. Set) za koje možemo reći da su kolekcije nizova, ali one koje nisu uređene. Glavna razlika između skupa i niza je to što kod skup nije dozvoljeno ponavljanje članova, što je velika prednost, jer ne postoji mogućnost da se isti podatak unese više puta. Zato su skupovi pogodni kada se radi o složenijim poslovima kao što su određene evidencije klijenata. Kako bi se izbjeglo ponavljanje elemenata, svi elementi se stavljaju u set, čime je onemogućeno da se jedan te isti element javi više puta, što nije slučaj sa nizovima.

➤ **Naredba SADD**

Sintaksa naredbe je SADD „ključ“ „vrijednost“. Ova naredba je zadužena za dodavanje vrijednost u skup. Ako se desi da vrijednost već postoji biće vraćena poruka tipa 0 ili false, jer kao što je već napomenuti, skup ne dozvoljava ponavljanje vrijednosti. Način na koji se koristi naredba SADD prikazan je na slici 4.18.

➤ **Naredba SMEMBERS**

Na slici 4.18. je takođe prikazana naredba SMEMBERS čiji je zadatak vraćanje svih članova niza. Ukoliko se desi da skup ne sadrži ni jedan član, javlja se greška. Ono što je opšte poznato je da kod skupova, posmatrano iz matematičkog aspekta, postoji unija, razlika i presjek. Znamo da unija predstavlja elementi i jednog i drugog skupa. Presjeku pripadaju samo oni elementi koji se nalaze i u jednom i u drugom skupu, dok razlika predstavlja elemente koji se nalaze u jednom skupu, a u drugom ne nalaze. Iste te operacije je moguće izvršavati i u Redis-u uz pomoć specijalno određenih komandi.

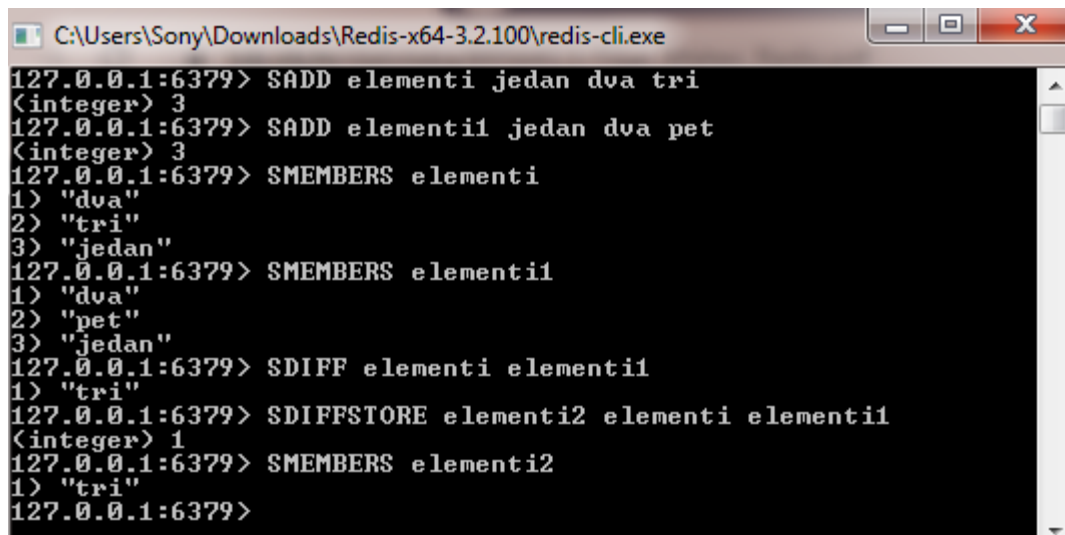


```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli...
127.0.0.1:6379> SADD set element1
(integer) 1
127.0.0.1:6379> SADD set element2
(integer) 1
127.0.0.1:6379> SADD set element1
(integer) 0
127.0.0.1:6379> SMEMBERS set
1) "element2"
2) "element1"
127.0.0.1:6379> SMEMBERS set1
(empty list or set)
127.0.0.1:6379>
```

Slika 4.18. Naredbe SADD i SMEMBERS

### ➤ Naredbe SDIFF i SDIFFSTORE

Ukoliko želimo pronaći razliku između dva skupa tada koristimo naredbu SDIFF. Na slici 4.19. se može vidjeti da se u prvom skupu nalaze elementi jedan, dva i tri, a u drugom skupu jedan, dva i pet. Kao razliku ova dva skupa dobijamo element tri, jer se on za razliku od jedan i dva ne nalazi u drugom skupu. Naredba SDIFFSTORE funkcioniše isto kao i SDIFF, samo što kod ove naredbe postoji još jedan parametar kada je u pitanju sintaksa, a to je treći skup u koji će biti smještene vrijednosti koje se dobiju kao rezultat razlike ova dva skupa.

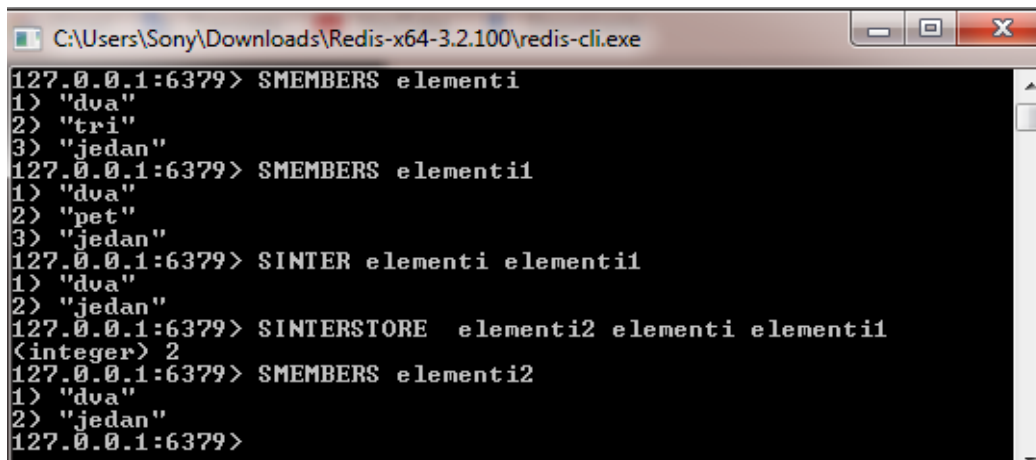


```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe
127.0.0.1:6379> SADD elementi jedan dva tri
(integer) 3
127.0.0.1:6379> SADD elementi1 jedan dva pet
(integer) 3
127.0.0.1:6379> SMEMBERS elementi
1) "dva"
2) "tri"
3) "jedan"
127.0.0.1:6379> SMEMBERS elementi1
1) "dva"
2) "pet"
3) "jedan"
127.0.0.1:6379> SDIFF elementi elementi1
1) "tri"
127.0.0.1:6379> SDIFFSTORE elementi2 elementi elementi1
(integer) 1
127.0.0.1:6379> SMEMBERS elementi2
1) "tri"
127.0.0.1:6379>
```

Slika 4.19. Naredbe SDIFF i SDIFFSTORE, vlastiti izvor

### ➤ Naredbe SINTER i SINTERSTORE

Naredba SINTER pronalazi presjek dva skupa. Vidimo na slici da su kao presjek vraćeni elementi dva i jedan, jer se oni nalaze i u jednom i u drugom skupu. Naredba SINTERSTORE radi isto, samo što elemente kao i naredba SDIFFSTORE smješta u treći skup koji navodimo kao parametar.



```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe
127.0.0.1:6379> SMEMBERS elementi
1) "dva"
2) "tri"
3) "jedan"
127.0.0.1:6379> SMEMBERS elementi1
1) "dva"
2) "pet"
3) "jedan"
127.0.0.1:6379> SINTER elementi elementi1
1) "dva"
2) "jedan"
127.0.0.1:6379> SINTERSTORE elementi2 elementi elementi1
(integer) 2
127.0.0.1:6379> SMEMBERS elementi2
1) "dva"
2) "jedan"
127.0.0.1:6379>
```

Slika 4.20. Naredbe SINTER i SINTERSTORE, vlastiti izvor

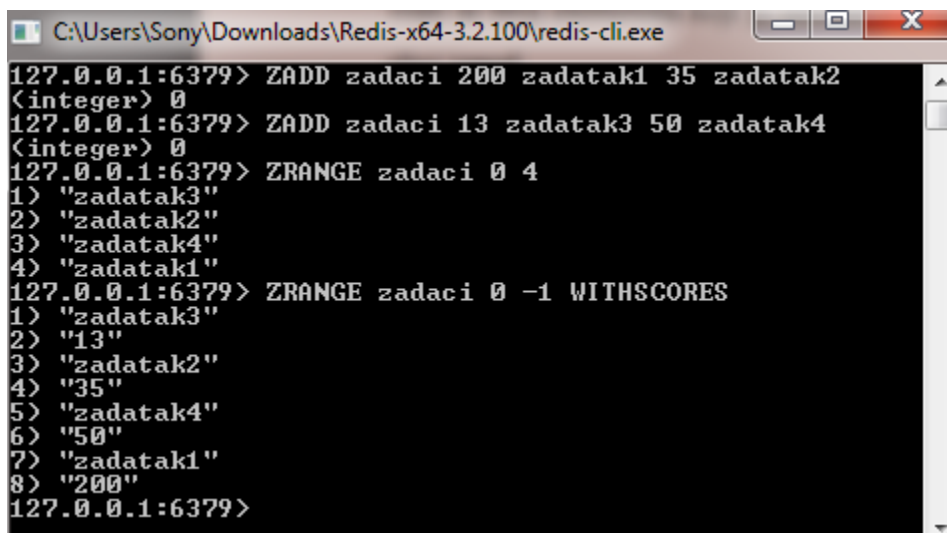
### • Sortirani skupovi

Sve ono što važi kod običnih skupova važi u kod sortiranih skupova. Jedina razlika je u tome što je kod sortiranih skupova članovi imaju svoju poziciju i to može biti veoma korisno kada se radi o nekim listama. Isto tako se računarskom svijetu to može iskoristiti u obliku liste zadataka koji imaju određene prioritete. Ono što je važno je da se članovi ne smiju ponavljati, ali sam prioritet može, jer je veoma moguće da postoje zadaci koji imaju iste prioritete. Još jedna od prednosti korištenja ovakvih skupova je ugrađen algoritam koji automatski sortira članove po prioritetu, čime je korisnik oslobođen velikog dijela posla. Ukoliko imamo neku listu zadataka sa prioritetima, svakom zadatku će se dodijeliti određen prioritet, nakon čega će se izvršiti sortiranje uz pomoć algoritma.

Ako se desi da više elemenata ima isti prioritet onda će oni biti sortirani po novom kriterijumu, tj. sortiraće se po abecedi. Složenost ovog algoritma je  $O(\log(n))$  što je veoma brzo s obzirom da ovaj algoritam dodaje, briše, ažurira i sortira elemente. Dodatne prednosti sortiranog skupa su vraćanje elemenata na osnovu pozicije na kojoj se nalazi ili na osnovu prioriteta. Naravno ovi skupovi podržavaju i operacije unije, razlike i presjeka, što je već objašnjeno.

### ➤ Naredba ZADD

Sintaksa naredbe je ZADD „ključ“ „prioritet“ „element“. Ova naredba se može uporediti sa naredbom SADD koja se koristi kod običnog skupa, samo što je kod ove naredbe potrebno dodati i prioritet elementu na osnovu čega će kasnije biti i sortiran. Nakon što se naredba izvrši kao rezultat se dobije broj elemenata koji je unesen. Isto tako, algoritam sortiranja će biti izvršen svaki put kada se nešto doda, ukloni ili ažurira što je moguće vidjeti i na slici 4.21. Bitna je i naredba ZRANGE WITHSCORES vraća sve elemente onako kako su i sortirani. Neke od naredbi koje se mogu izvršavati su: ZCOUNT, ZINCRBY, ZINTERSCORE, ZRANK, ZREM, ZUNIONSTORE. Pošto su već objašnjene nema potrebe da se ponovo implementiraju, međutim sve ove komande je moguće pronaći na sljedećem [linku](#).



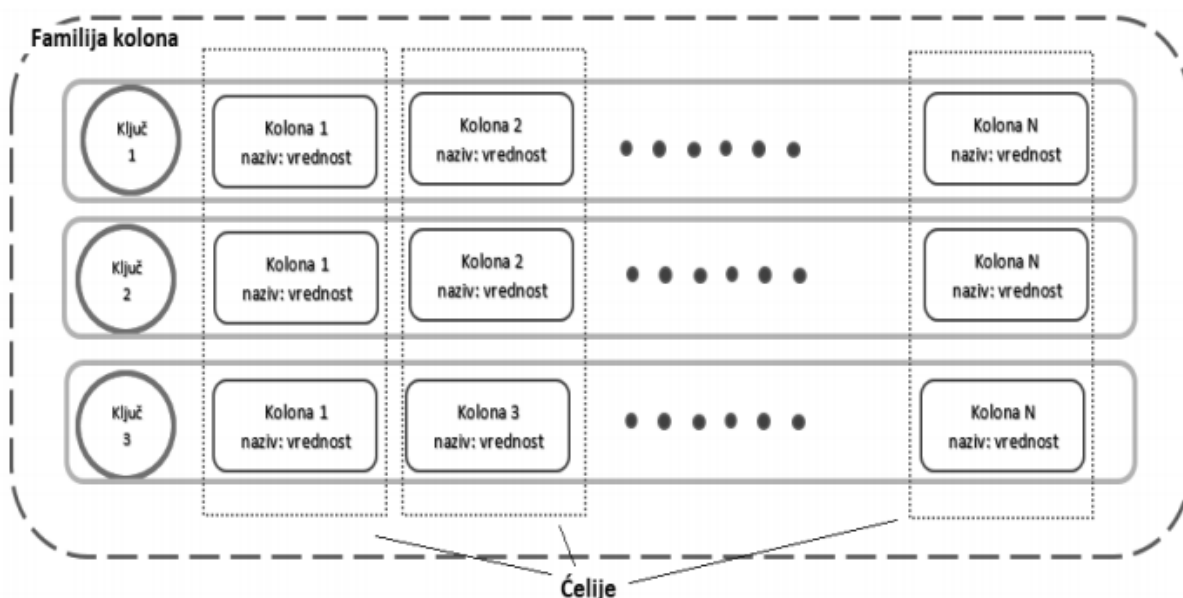
```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe
127.0.0.1:6379> ZADD zadaci 200 zadatak1 35 zadatak2
(integer) 0
127.0.0.1:6379> ZADD zadaci 13 zadatak3 50 zadatak4
(integer) 0
127.0.0.1:6379> ZRANGE zadaci 0 4
1) "zadatak3"
2) "zadatak2"
3) "zadatak4"
4) "zadatak1"
127.0.0.1:6379> ZRANGE zadaci 0 -1 WITHSCORES
1) "zadatak3"
2) "13"
3) "zadatak2"
4) "35"
5) "zadatak4"
6) "50"
7) "zadatak1"
8) "200"
127.0.0.1:6379>
```

Slika 4.21. Naredbe ZADD i ZRANGE, vlastiti izvor

## 4.2 Model familija kolona

Glavna razlika u odnosu na ključ-vrijednost baze podataka je što se kod ovog modela podacima ne pristupa na osnovu vrijednosti, nego po kolonama. Obrada se izvršava paralelno po kolonama iz čega potiču i bolje mogućnosti. Kako bi se čuvali podaci potrebno je koristiti kolonske familije tj. multidimenzione sortirane mape. Ono što je karakteristično za kolone je da imaju ime i da mogu pamtit veći broj vrijednosti po vrsti. S obzirom da te vrijednosti imaju vremensku oznaku, moguće je do svake vrijednosti doći uz pomoć ključ-vrste, ključ-kolone i vremenske oznake. Ono za šta se vremenske oznake najviše koriste je rješavanje

određenih konflikata, automatsko brisanje podataka i slično. Familije kolona se najčešće formiraju kada je potrebno grupisanu onu vrstu podataka kojoj se često pristupa. Na primjer, ukoliko imamo tabelu ljudi koji rade u istoj firmi zapisi, oni ni na koji način nisu povezani tj. susjedni zapisi nemaju apsolutno ništa zajedničko. Upravo taj problem se rješava skladištem familija kolona. To znači da u konceptualnom modelu koji možemo vidjeti na slici ključevi 1, 2 i 3 jesu jedinstveni, ali su redovi u kojima se oni nalaze povezani. Na primjer, zahvaljujući ovom modelu moguće je formirati familiju kolona od muzičara koji imaju više od 15 albuma ili od naučnika koji su dobili više od 2 vrijedne nagrade. Primjer ovakvog modela je Apache Cassandra koja se koristi kada je neophodno raditi sa velikom količinom podataka.



Slika 4.24. Konceptualni model familija kolona, izvor: [2]

### 4.2.1 Apache Cassandra baza podataka

Apache Cassandra je kao i Redis otvorenog koda, ali je instalacija malo komplikovanija u odnosu na ostale baze. To je distribuiran sistem koji se pokazao kao odličan kada se radi o skladištenju ogromnih količina podataka. Ova baza pruža usluge skoro bez ikakve mogućnosti da dođe do nekih grešaka. Postoje glavni dijelovi od kojih je sastavljena Cassandra i to su:

- Čvor (eng. Node) – Mjesto gdje se spremaju podaci .

- Središte podataka (eng. Data center) – Kolekcija čvorova koji su međusobno povezani.
- Grupa (eng. Cluster)– Komponenta koja se sastoji od jednog ili više središta podataka.
- Dnevnik izvršenja (eng. Commit log) – Svaka operacija se zapisuje u dnevnik izvršenja kako bi se mogla vratiti u slučaju da dođe do pada sistema.
- Memorijska tabela (eng. Mem-table) – Tabela u koju se smještaju strukture podataka nakon dnevnika izvršenja.
- SSTablica (eng. SortedStructure Table) – Datoteka na disku u koju se prebacuje podatak iz memorijske tabele kada njegov sadržaj dosegne najveću vrijednost.
- eng. Bloom filter– Veoma brz algoritam koji provjerava da li je određeni element dio kompleta. Ovaj filter predstavlja specijalnu vrstu predmemorije čiji je pristup omogućen poslije svakog upita.

Postoje i kolekcijski tipovi podataka i to su:

1. Lista – Kolekcija koja se sastoji od jednog ili više poredanih elemenata.
2. Mapa – Kolekcija koja se sastoji od parova ključ-vrijednost.
3. Set – Kolekcija koja se sastoji od jednog ili više elemenata.

Ono što je interesantno je činjanica da korisnik može stvoriti mijenjati i obrisati svoj tip podatka.

Neki od ugrađenih tipova podataka se nalaze u sljedećoj tabeli:

Tip podatka	Opis
Boolean	Logički tip
Int	Cjelobrojni tip
Double	Decimalni tip
Float	Decimalni tip
Varchar	Znakovni tip

U nastavku će biti objašnjene neke od osnovnih komandi u Apache Cassandra bazi.

- **CQL**

CQL (Cassandra Query Language) jezik možemo definisati kao jezik koji je sintaksom veoma sličan SQL jeziku. Ono što se razlikuje jesu sami pojmovi. Ono što je neophodno uraditi kada želimo izvršavati upite u relacionim sistemima je kreiranje same baze podataka. Isto to je potrebno uraditi i ovdje samo što ovdje umjesto same baze kreiramo keyspace. Neophodno je napomeniti da se naredbe završavaju obavezno završavaju sa “;”.

```
cqlsh> CREATE KEYSPACE keyspaceName WITH REPLICATION = { 'class':  
'NetworkTopologyStrategy' };
```

Na ovaj način kreiran je keyspace. Kako bismo mogli da radimo sa podacima neophodno je da se prebacimo u novokreirani resurs, što se radi na sljedeći način:

```
use keyspaceName;
```

Takođe pored samog kreiranja keyspace-a moguće je i vršiti ažuriranje keyspace-a. Ako se koristi faktor replikacije 1 to znači da u bazi podataka ne dolazi do replikacije.

```
cqlsh> ALTER KEYSPACE keyspaceName WITH REPLICATION = { 'class':  
'SimpleStrategy', 'replication_factor': 1 };
```

Nakon kreiranja „baze podataka“ u našem slučaju keyspace-a, kreiramo familiju kolona koja se nalazi u našem keyspace-u. U sljedećem primjeru kreirana je tabela korisnici sa kolonama korisnicko\_ime koje je ujedno i primarni ključ, zatim lozinka i email kojima dodjeljujemo znakovni tip podatka:

```
cqlsh:keyspaceName > CREATE TABLE korisnici (  
    korisnicko_ime varchar PRIMARY KEY,  
    lozinka varchar,  
    email varchar  
);
```

Kako bismo pogledali dostupne familije kolona možemo koristimo sljedeću naredbu:

```
SHOW TABLES;
```

Kako bismo saznali bitne informacije o tabeli koristimo sljedeću komandu SHOW TABLE:

```
SHOW TABLE korisnici;
```

Može se zaključiti da CQL ima velike sličnosti sa SQL jezik. Identično je i prilikom unošenja podataka jer se koristi naredba INSERT. U ovom slučaju u tabelu korisnici se unose vrijednosti za korisnicko\_ime, lozinku i email:

```
cqlsh:keyspacename> INSERT INTO korisnici (korisnicko_ime, lozinka, email)
VALUES ('jelenak', 'sw56ghz', 'jelenakom@gmail.com');
```

Postavljanje upita se vrši uz pomoć naredbe SELECT, isto kao što je to slučaj sa SQL jezikom. Naredbom SELECT biramo koje redove želimo da dohvatimo, u slučaju kada želimo dohvatiti sve redove koristimo \* (zvjezdicu).

```
cqlsh:keyspacename> SELECT * FROM korisnici;
```

korisnicko_ime	lozinka	email
jelenak	sw56ghz	jelenakom@gmail.com

Dok u slučaju kada želimo dohvatiti određene redove dovoljno je navesti imena redova (u slučaju da ih je više, moraju se odvojiti zarezom).

```
cqlsh:keyspacename> SELECT korisnicko_ime FROM korisnici
```

korisnicko_ime
jelenak

Ukoliko nam je poznat jedan podatak, a želimo pronaći ostale, to je moguće uz pomoć naredbe WHERE. Na primjer ako želimo pronaći email osobe za koju imamo samo korisnicko\_ime:

```
cqlsh:keyspacename> SELECT email FROM korisnici WHERE korisnicko_ime=
'jelenak' ;
```

email
jelenakom@gmail.com



Dozvoljeno je koristiti i ostale naredbe koje su podržane od strane SQL jezika gdje se pored naredbe WHERE može koristiti i naredba ORDER BY.

Pored dodavanja podataka, podatke je mogući i ažurirati. Za to se koristi naredba UPDATE, uz koju je neophodno i korištenje naredbe SET kojom naglašavamo šta želimo izmijeniti i koja je nova vrijednost podatka.

U ovom slučaju mijenjamo email adresu novom vrijednošću tamo gdje je korisnicko\_ime 'jelenak'.

```
cqlsh:keyspace> UPDATE korisnici SET email='jelenakom@jelenanew.com'
WHERE korisnicko_ime='jelenak' ;
```

Naredbu DELETE koristimo kada želimo da izbrišemo određeni red ili čak više njih. Kako bi naredba obrisala redove koje želimo neophodno je prilikom navođenja same naredbe navesti i imena redova koje želimo obrisati.

```
cqlsh:keyspace> DELETE email FROM korisnici WHERE korisnicko_ime =
'jelenak' ;
```

Ako ne definišemo imena redova, onda će biti obrisani cijeli red.

```
cqlsh:keyspace> DELETE FROM korisnici WHERE korisnicko_ime =
'jelenak' ;
```

Kod CQL-a je moguće dobiti i informacije o samom keyspace-u korištenjem naredbe DESCRIBE KEYSPACE:

```
DESCRIBE KEYSPACE;
```

Još jedna od mnogobrojnih mogućnosti predstavljaju informacije kako o verziji samog CQL jezika tako i o verziji baze Cassandra i slično. Npr. upit koji će nam dati sve informacije vezane CQL jezika je:

```
cqlsh:keyspace> SHOW VERSION
```

```
[cqlsh 4.1.1 | Cassandra 2.0.9 | CQL spec 3.1.1 | Thrift protocol 19.39.0]
```

Ukoliko želimo da budemo upućeni u ono što sistem radi možemo uključiti tracing opciju, nakon koje će nam biti tačno ispisano šta sistem u kojem trenutku radi. Takođe možemo koristiti tracing opciju u sistemu. Naredba kojom se to izvršava je sljedeća:

```
TRACING on;
```

Sistem podržava i operacije sa indeksima. Indekse kreiramo na sljedeći način:

```
cqlsh:keyspacename> CREATE INDEX emailindex ON korisnici(email) ;
```

Isti taj indeks je moguće i obrisati:

```
cqlsh:keyspacename> DROP INDEX emailindex;
```

U CQL jeziku postoji još mnogo naredbi i mogućnosti kao i u samom sistemu, međutim objašnjene su samo one koje su presudne za razumijevanje funkcionisanja samog sistema.

### 4.3 Graf baze podataka

Graf možemo definisati kao neku vrstu veze između podataka u bazi. Zahvaljujući vezama imamo mogućnost da povežemo podataka koji se nalaze u bazi. Pojam čvora u grafu se može predstaviti kao entitet (npr. jedna osoba može predstavljati čvor), dok se veza definiše kao povezanost dva čvora. Ono što karakteriše svaki čvor je set izlaznih i ulaznih veza kao i set ključ-vrijednost parova. Veze su najvažnija odlika grafa, jer se upravo zbog njih ovaj sistem razlikuje od svih ostalih NoSQL sistema. Veoma bitan pojam za razumijevanje funkcionisanja ovih baza jesu svojstva koja predstavljaju informacije o čvoru. Na primjer, ako određena riječ predstavlja jedan čvor, onda bi ta riječ sigurno mogla biti povezana sa riječima koje počinju na isto slovo kao ta riječ.

Graf baze podataka su veoma korisne kada je potrebno analizirati neke povezanosti podataka. Ono za šta se još koriste je rudarenje podataka koji se nalaze na društvenim mrežama. Ono što je veoma bitna razlika između relacionih i graf baza podataka je to da graf baze skladište veze između podataka tj. umjesto e-mail adrese koja bi se skladištila u SQL bazama i koja bi se mogla pronaći uz pomoć primarnog ključa korisnika, u graf bazama podataka korisnički podatak sadrži pokazivač na e-mail adresu koja mu je dodijeljena. I upravo to je ogromna prednost, jer samim tim što korisnik sadrži pokazivač na e-mail adresu dovoljno je samo odabrati korisnika i izbjeći nepotrebne upite i pretraživanja. Ono po čemu se graf baze podataka znatno razlikuju u odnosu na ostale NoSQL baze je poštovanje ACID modela transakcije. Napomenuli smo da su Base svojstva karakteristična za NoSQL baze. Upravo zbog ovih transakcija nije moguće povezivanje ukoliko

postoji samo jedan čvor. Neka od pravila koja je neophodno poštovati prilikom rada sa graf bazama su:

- Neophodno je postojanje početnog i završnog čvora
- Čvor nije moguće obrisati ukoliko se ne obrišu sve njegove veze

Kod ovih baza podataka se najčešće koristi jedan računar. Međutim ukoliko postoji potreba za distribucijom, jer jedan računar ne može na pravi način da izvrši upt koristi se master-slave replikacija koja je opisana u drugom poglavlju. Bitna razlika u odnosu na običan model master-slave je to što u ovom slučaju slave čvorovi imamo mogućnost pisanja, ali obavezno to moraju da prijave glavnom čvoru, jer on odlučuje da li će nova vrijednost biti prihvaćena ili neće.

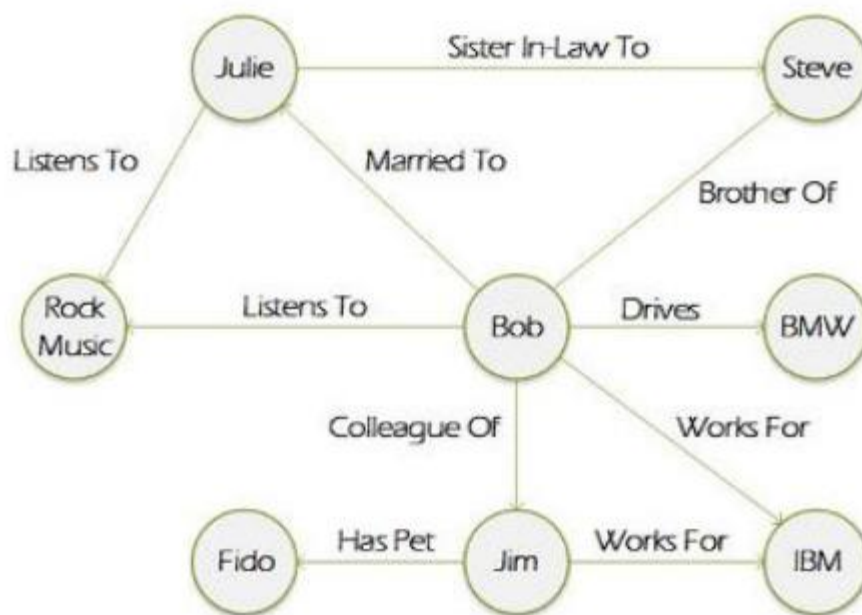
Jedna od najpoznatijih graf baza podataka je Neo4j, čiji će način rada biti prikazan i objašnjen u nastavku.

Najčešća upotreba graf baza podataka je kada pronalazimo najkraći puta između dva čvora, npr. :

1. Kod određenih aplikacija koje služe za pronalazak najkraće rute npr. navigacija.
2. Kod društvenih mreža (facebook, twitter)
3. Za određivanje najdjelotvornijeg puta za usmjeravanje prometa preko mreže podataka
4. Kada je potrebno otkriti organizovani kriminal

Ono za šta se graf baze podataka još mogu koristiti, a pri tom su vrlo efikasne u tome, je sugerisanje proizvoda kupcima prilikom kupovine na osnovu proizvoda koji su najviše kopovani od strane drugih kupaca i slično.

Postoji ugrađen algoritam koji se najčešće koristi za rješavanje ovakvih problema i to je Dijkstrin algoritam koji je dobio ime po Edsgeru Dijkstri. Algoritam radi tako što prvo analizira svaku pojedinačnu vezu od izvora do odredišta pri tom pamteći najkraću rutu. One rute koje su gude odbacuje sve dok ne pronađe najkraću rutu.



Slika 4.25. Primjer grafa, izvor: [3]

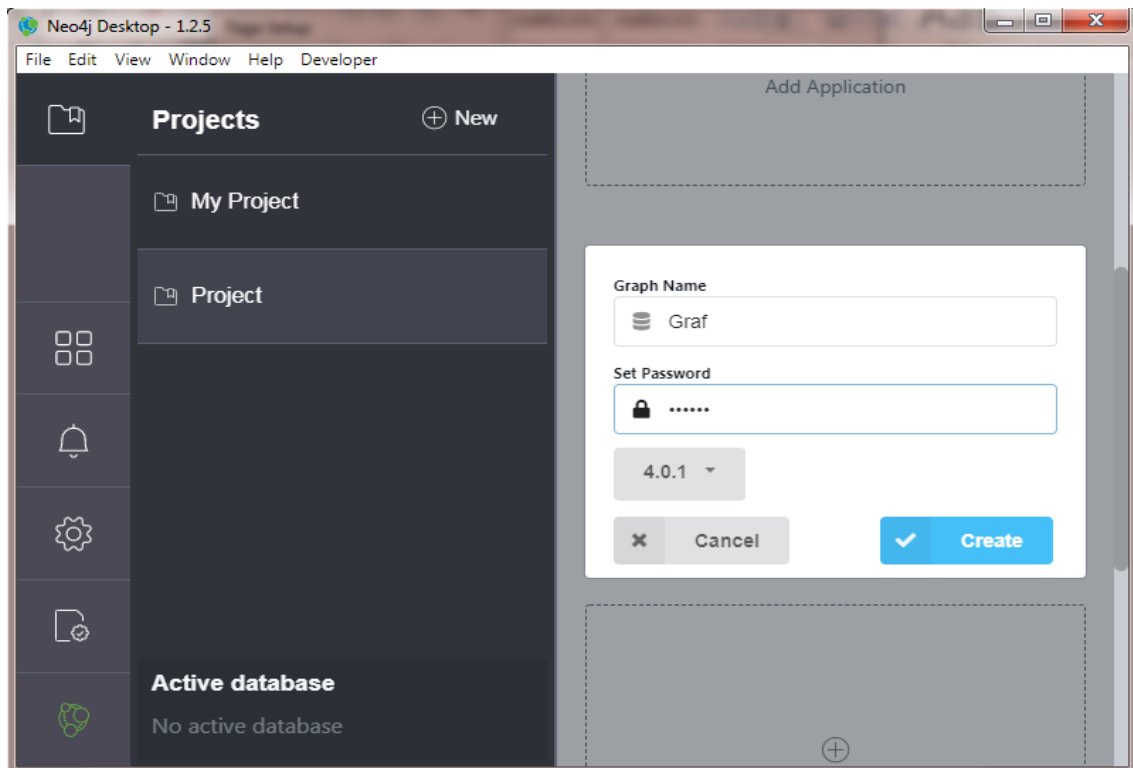
### 4.3.1 Neo4j baza podataka

Neo4j je najpoznatija graf baza podataka i ona je otvorenog koda kao i prethodno opisane baze. Ono što je novo je to da ova pruža mogućnost čuvanja čvorova i veza između njih. Neo4j je napisan u programskom jeziku Java i pruža mogućnost korištenja CQL upitnog jezika. Ukoliko je potrebno često izvršavati upite za iste podatke moguće je to podatke spremiti u keš memoriju kako bismo bili u mogućnosti da ih prije dohvatimo.

Kod Neo4j baza postoje dva načina keširanja podataka:

- File buffer cache- Spremanje podataka na disk u sažetom formatu.
- Object cache- Spremanje čvorova, veza i svih njihovih svojstava u format koji služi za efikasno prebacivanje u memoriju.

Kako bismo krenuli sa radom u Neo4j bazi podataka, potrebno je prvo preuzeti [instalaciju](#), a zatim kreirati novi projekat. Novi projekat se kreira odabirom opcije **New** u opciji **Projects**. Zatim je potrebno odabrati opciju **New Graph** unutar novog projekta. Onda **Create a Local Graph**, a zatim se odabere proizvoljno ime za naziv baze kao i lozinka za novu bazu podataka. I konačno kako bismo kreirali novu bazu idemo na **Create**.



Slika 4.26. Kreiranje baze podataka, vlastiti izvor

Sada kada smo kreirali bazu podataka pomoću Neo4j Browsera objasnićemo neke od osnovnih naredbi koje se najviše koriste:

- Čvorove kreiramo pomoću naredbe CREATE

```
1 CREATE (prvi:Osoba {ime:"Jelena"})
2 RETURN prvi
```

Slika 4.27. Kreiranje čvora, vlastiti izvor

Ovaj upit kreira novi čvor *prvi* tipa *Osoba* i sadrži podatak sa ključem *ime* čija je vrijednost „Jelena“. Ako odaberemo grafički prikaz, to onda izgleda ovako:



Slika 4.28. Grafički prikaz kreiranog čvora, vlastiti izvor

Imamo mogućnost i tabelarnog prikaza, koji izgleda ovako:

```
{
  "ime": "Jelena"
}
```

Slika 4.29. Tabelarni prikaz ključa i vrijednosti, vlastiti izvor

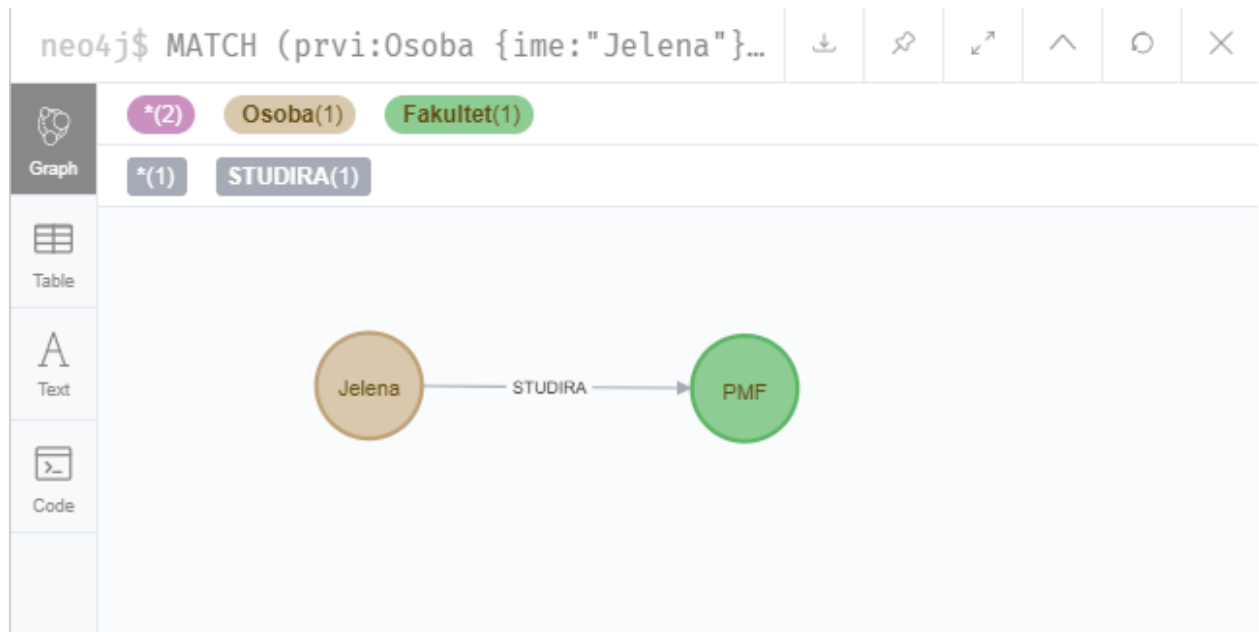
➤ Kreiranje veza

```
1 MATCH (prvi:Osoba {ime:"Jelena"})
2 CREATE (prvi)-[studira:STUDIRA]→
  (pmf:Fakultet
3 {Ime:"PMF" })
4 RETURN prvi,studira,pmf
```

Slika 4.30. Primjer kreiranja veze, vlastiti izvor

Pomoću naredbe *MATCH* pronalazimo podatak *Osoba* čija se vrijednost *ključa* poklapa sa zadatom vrijednošću „*Jelena*“ i dodjeljujemo mu oznaku *prvi* preko koje će se referencirati na taj podatak. Naredbom *CREATE* kreiramo vezu *STUDIRA* prema podatku tipa *FAKULTET* sa podatkom koji za ključ *ime* ima vrijednost „*PMF*“.

Grafički prikaz izgleda ovako:



Slika 4.31. Grafički prikaz kreirane veze, vlastiti izvor

Ukoliko želimo da kreiramo nove podatke i da od jednog podatka kreiramo vezu *PRIJATELJ* prema svim tim podacima onda koristimo naredbu *FOREACH*, tj. u dijelu *FOREACH* naredbe navodimo imena prema kojima želimo kreirati vezu.

```

1 MATCH (prvi:Osoba {ime:"Jelena"})
2 FOREACH (ime in
  ["Jovan","Marko","Ana","Ivan"] |
3 CREATE (prvi)-[:PRIJATELJ]→(:Osoba
  {ime:ime}))

```

Slika 4.32. Naredba FOREACH, vlastiti izvor

➤ Pretraga prijatelja

```

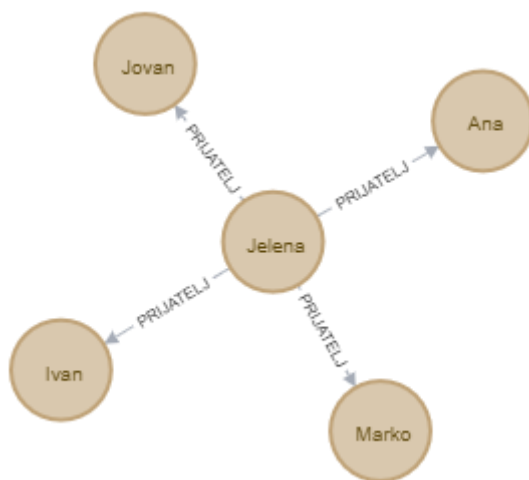
1 MATCH (prvi {ime:"Jelena"})-[:PRIJATELJ]→
  (prijatelji)
2 RETURN prvi,prijatelji

```

Slika 4.33. Pretraživanje prijatelja, vlastiti izvor

Pomoću ovog upita ćemo kao rezultat dobiti sve podatke prema kojima postoji veza *PRIJATELJ* od podatka kojem za ključ *ime* odgovara dodijeljena vrijednost „Jelena“.

Grafički prikaz:



Slika 4.34. Grafički prikaz rezultata pretraživanja, vlastiti izvor



➤ Kreiranje prijatelja našeg prijatelja

```
1 MATCH (pmf:Fakultet {ime:"PMF"})
2 MATCH (ana:Osoba {ime:"Ana"})
3 CREATE (ana)-[:PRIJATELJ]→(:Osoba:Expert
4 {ime:"Dragana"})-[:STUDIRA_NA]→(pmf)
```

Slika 4.35. Kreiranje prijatelja od već kreiranog prijatelja, vlastiti izvor

Na ovaj način smo kreirali vezu *PRIJATELJ* između vrijednosti Ana i Dragana i za vrijednost Dragana smo kreirali relaciju *STUDIRA\_NA* prema podatku koji za ključ ima vrijednost “*pmf*”.

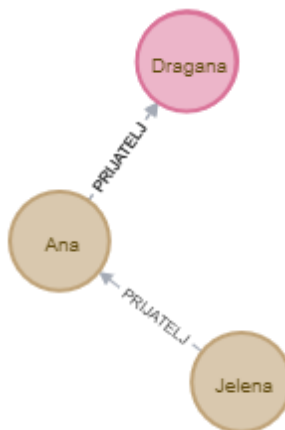
➤ Nalaženje najkraćeg puta

**Neo4j** ima ugrađenu funkciju pod nazivom *shortestPath* koja koristi *Dijkstra algoritam*, koji pronalazi najkraći put između čvorova.

Pomoću ovog upita rekurzivno prolazimo kroz sve naše prijatelje do maksimalne dužine koja je zadana. Maksimalna dužina u ovom slučaju je 5 uključujući i osobu koja je *expert*.

```
1 MATCH (prvi {ime:"Jelena"})
2 MATCH (expert)-[:STUDIRA_NA]→(fax:Fakultet
3 {ime:"PMF"})
4 MATCH path = shortestPath( (prvi)-[:PRIJATELJ*..5]- (expert) )
5 RETURN prvi,expert,path
```

Slika 4.36. Pronalaženje najkraćeg puta korištenjem ugrađene funkcije *shortestPath*, vlastiti izvor



Slika 4.37. Grafički prikaz rezultata ugrađene funkcije *shortestPath*, vlastiti izvor

### ➤ Izmjena i dodavanje atributa

```
1 MATCH (n { ime: 'Ana' })
2 SET n.prezime = 'Jovanovic'
3 RETURN n
```

Slika 4.38. dodavanje novog atributa koristeći naredbu SET, vlastiti izvor

Ovaj upit pronalazi ključ ime koji ima vrijednost “Ana” i dodaje mu novi atribut prezime korištenjem naredbe SET.

Rezultat dodavanja atributa izgleda ovako:



Slika 4.39. Rezultat dodavanja novog atributa, vlastiti izvor

### ➤ Uklanjanje atributa

Kako bismo uklonili određeni atribut koristimo naredbu REMOVE:

```
1 MATCH (n { ime: 'Ana' })
2 REMOVE n.prezime
3 RETURN n
```

Slika 4.40. Uklanjanje atributa naredbom REMOVE, vlastiti izvor

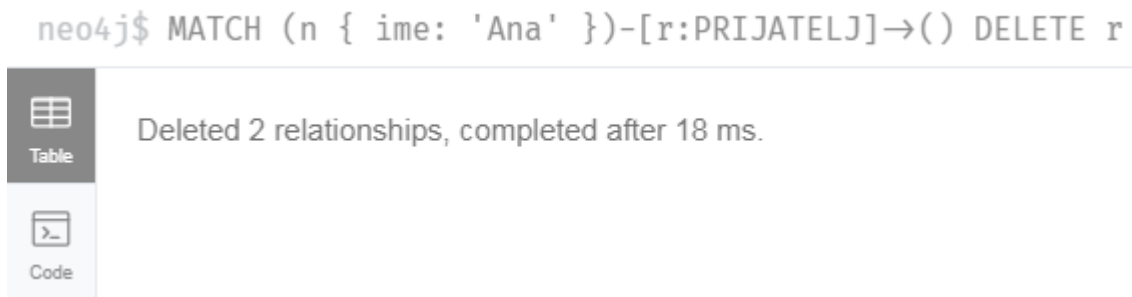
### ➤ Brisanje veza

U Neo4j bazama imamo mogućnost da obrišemo ne samo čvorove i njihova svojstva nego i veze koje se nalaze između njih. Ono što je važno napomenuti je to da je čvor nemoguće obrisati ukoliko se prvo ne obrišu sve njegove veze sa ostalim čvorovima. Brisanje veza određenog čvora se vrši na sljedeći način:

```
1 MATCH (n { ime: 'Ana' })-[r:PRIJATELJ]→()
2 DELETE r
```

Slika 4.41. Brisanje veza određenog čvora, vlastiti izvor

Rezultat ovog upita je:



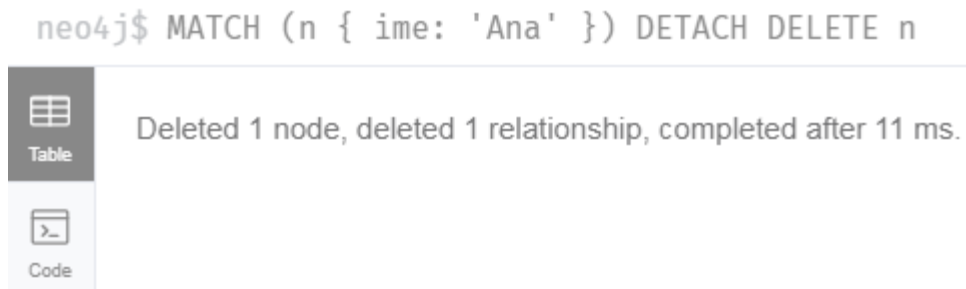
Slika 4.42. Rezultat brisanja svih veza jednog čvora, vlastiti izvor

➤ Brisanje čvora i svih njegovih veza :

```
1 MATCH (n { ime: 'Ana' })
2 DETACH DELETE n
```

Slika 4.43. Brisanje čvora i svih njegovih veza, vlastiti izvor

Rezultat upita:



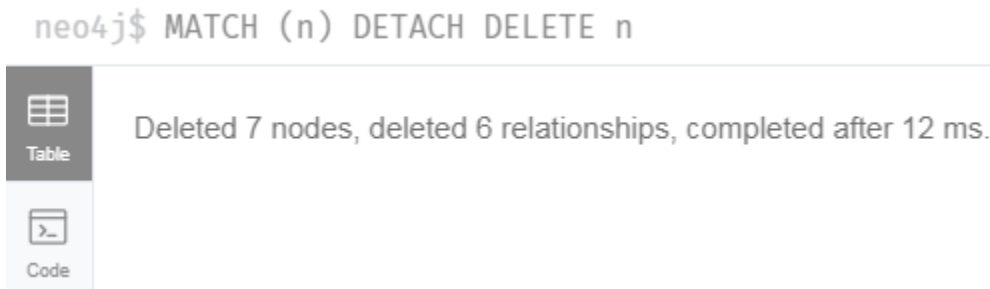
Slika 4.44. Rezultat brisanja čvora i njegovih veza, vlastiti izvor

- Brisanje svih čvorova:

```
neo4j$ MATCH (n) DETACH DELETE n
```

Slika 4.45. Brisanje svih postojećih čvorova, vlastiti izvor

Rezultat upita:



Slika 4.46. Rezultat brisanja svih čvorova, vlastiti izvor

## 4.4 Dokument baze podataka

Skladišta dokumenta su možda jedan od najfleksibilniji modela NoSQL baza podataka, jer je zahvaljujući njima moguće skladištiti različite vrste podataka, upravo iz razloga što dokument baze podataka ne znaju koji sadržaj se u njima čuva. Ove baze ne koriste nikakvu šemu za skladištenje podataka.

Dokument baze podataka kao što i sama riječ kaže rade sa dokumentima i nad njima se izvršavaju različiti upiti. Dokument može biti u različitim formatima. Dokument se može biti Microsoft Word ili PDF dokument, ali najčešće se radi o dokumentima tipa XML (engl. Extensible Markup Language) i JSON (engl. JavaScript Object Notation). Kao što relacione baze koriste kolone koje imaju svoj tip podatka tako i dokumenti sadrže opis tipa podatka kao i vrijednost koja je dodijeljena tom podatku. Ukoliko je potrebno dodati novi podatak u ovaj tip baze neophodno je samo dodati potrebni objekat bez da se mijenja sama šema baze. Samim tim što ove baze omogućavaju čuvanje različitih tipova podataka, dozvoljeno je i kodiranje pomoću različitih sistema.

Drugim riječima, ne moraju svi dokumenti biti kodirani u jednom sistemu, jedan dokument može biti kodiran u JSON formatu, a drugi u XML formatu ili obrnuto.

Primjer dokumenta u JSON formatu izgleda ovako:

```
{  
  
  "Ime": "Jelena",  
  "Adresa": "Kralja Petra I",  
  "Fakultet": "PMF",  
  "Kolekcija_knjiga": [ "Ana Karenjina", "Zločin i kazna", "Tvrđava" ]  
  
}
```

Primjer dokumenta u XML formatu:

```
<Osoba>  
  <Ime>Jelena</Ime>  
  <Prezime>Komljenovic</Prezime>  
  <Godine>20</Godine>  
  <Adresa>  
    <Ulica>Kralja Petra I</Ulica>  
    <Grad>Banja Luka</Grad>  
    <Drzava>BiH</Drzava>  
  </Adresa>  
</Osoba>
```

Ono što je veoma bitno je da za razliku od relacionih baza gdje polja mogu biti prazna, dokument baze to ne dozvoljavaju, nije moguće ostaviti neiskorištena polja. Možemo vidjeti da postoji sličnost u strukturi kod prethodna dva dokumenta. Ovakav pristup dozvoljava dodavanje novih informacija bez zahtijevanja da svaki zapis u bazi ima istu strukturu. Kod dokument baza postoji i pojam kolekcije koji

se uvodi kako bi klijent bio upućen u ono što se nalazi u samom dokumentu. Kod svake kolekcije neophodno je definisati tip dokumenta koji će ta kolekcija da čuva, ali to se odnosi samo na logički nivo ne i na samu strukturu dokumenta. Na primjer, u jednoj kolekciji se mogu nalaziti imena osoba dok se u drugoj kolekciji mogu nalaziti vrste voća. Dokumenti mogu pripadati istoj kolekciji iako šema ovih podataka nije ista, što nije moguće kod relacionih podataka, jer one zahtijevaju da svaki red u tabeli ima tačno određenu šemu. Prevažni cilj dokument baza je bio mogućnost raspodjele podataka na više različitih servera, gdje sistemi sami brinu o kako o samoj raspodjeli podataka tako i o načinu povezivanja sa serverima. Za ove baze, konkretno za MongoDB bazu je karakterističan master-slave model distribuiranja podataka. Ovaj model je efikasan, jer ostali čvorovi mogu da nastave sa radom čak i ukoliko dođe do kvara master čvora na način da izaberu novog vođu. Kako bi se od ponuđenih čvorova izabrao najadekvatniji moguće je svim čvorovima dodijeliti određene prioritete. Distribucija se najčešće ostvaruje replikacijom i fragmentacijom prema master-slave principu. MongoDB se, takođe, oslanja na ovaj princip. Ukoliko dođe do kvara master čvora, čvorovi između sebe biraju novog vođu. Kako bismo osigurali da će u tom slučaju biti odabran najprikladniji sljedeći čvor, možemo im dodijeliti prioritete. Jedna od najpoznatijih dokument baza podataka je MongoDB, koja će biti detaljno objašnjena u nastavku.

#### 4.4.1 MongoDB baza podataka

Ukoliko tražimo nešto što ima sličnosti sa relacionim bazama a opet ima sve odlike NoSQL sistema onda je MongoDB baza podataka najbolji izbor. Mongo je dokument baza JSON formata (tj. podaci su smješteni u binarnu formu JSON-a pod nazivom BSON). *Mongo* se može predstaviti kao polje u tabeli relacione baze koja nema šemu. Pored velikog broja standardnih operacija (dodavanja, ažuriranja, brisanja) MongoDB nudi i ostale operacije koje ostale baze ne samo da ne posjeduju nego nemaju mogućnost da razviju.

MongoDB posjeduje indekse koji mogu tekstualni, kombinovani pa čak i geoprostorni. Još jedna od veoma bitnih operacija je operacija sakupljanja. Zahvaljujući ovom svojstvu moguće je složenije sakupljanje podataka kao i njihovo oblikovanje koje će od najjednostavnijih skupova načiniti složene skupove. Postoje i specijalne kolekcije koje su jednako bitno svojstvo MongoDB. Kolekcije koje su ograničene (engl. capped collections) nazivamo specijalnim

kolekcijama. Svrha ovih kolekcija je brisanje starijih podataka koji će biti obrisani čim kolekcija pređe unaprijed definisanu veličinu u memoriji računara ili ukoliko se prevaziđe broj dokumenata kolekcije.

Možemo reći da MongoDB podržava veliki broj mogućnosti SQL baza podataka, ali na mnogo efikasniji način.

Postoje određeni pojmovi koji su presudni za razumijevanje funkcionisanja MongoDB sistema, a to su:

- `_id` - Obavezno polje, koje predstavlja jedinstvenu vrijednost za svaki dokument u MongoDB bazi. Pomoću njega razlikujemo dokumente u bazi, koji imaju identične određene ključeve i vrijednosti. Ukoliko napravimo dokument bez `_id`-a, on će biti automatski dodan.
- Kolekcija - Predstavlja dokumente koji su grupisati i ona postoji unutar jedne baze. Svaki dokument može da bude drugačiji, jer kolekcije nemaju tačno definisanu strukturu.
- Baza podataka - Skladište kolekcija.
- Dokument - Zapis u kolekciji koji se sastoji od polja i njihovih vrijednosti.
- Polje - Par (*ime, vrijednost*) dokumenta. Moguće je da dokument ima 0 ili više polja.
- JSON – Format u kojem se predstavljaju dokumenti u kojima se nalaze podaci.

## Instalacija MongoDB sistema

Ukoliko želimo da radimo sa MongoDB sistemom moramo prvo preuzeti [instalaciju](#) (odabrati odgovarajuću verziju, operativni sistem i paket). Kako bismo bili u mogućnosti izvršavati različite upite nad kolekcijama koje baza sadrži korist ćemo MongoDB shell program, za čije korištenje je potrebno pokrenuti shell skript koji dolazi uz mongo server.

Ono što je zanimljivo je činjenica da MongoDB shell program nema komandu za kreiranje nove baze podataka. Baza će se automatski kreirati kada u nju zapišemo prvi dokument. Na primjer, ukoliko u MongoDB shell programu nije postojala baza `mojabaza`, nakon izvršavanja naredbi prikazanih na slici 4.47., biće kreirana nova baza podataka `mojabaza`, koja sadrži jednu kolekciju pod imenom `novakolekcija`, koja sadrži dokument sa ključevima: `ime`, `prezime`, `godine` i `fakultet`. Tim ključevima su dodijeljene određene vrijednosti:

```

> use fakultet
switched to db fakultet
> db.studenti.insertMany < [ {
...                               "ime" : "Jelena",
...                               "prezime" : "Komljenovic",
...                               "godine" : "20",
...                               "fakultet" : "PMF"
...                               } ] >
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId<"5e7a42e5a1c08e67171625b4">
  ]
}
>

```

Slika 4.47. Kreiranje baze podataka i kolekcije, koja sadrži dokumente, vlastiti izvor

Važno je napomenuti da su komande, koje će u nastavku biti objašnjene primjenljive samo i isključivo u **MongoDB shell programu**.

Pomenuli smo da se dokumenti u kojima se nalaze podaci čuvaju u JSON (BSON) formatu, što znači da smo prilikom kreiranja baze dodali novi JSON dokument, pa samim tim vitičaste zagrade {...} označavaju objekat sa ključevima i vrijednostima, dok uglaste zagrade [...] označavaju niz.

Komanda *show collections* kao što i samo ime kaže služi za pregled kolekcija, i pomoću možemo vidjeti koje kolekcije trenutno postoje:

```

> show collections
studenti
>

```

Slika 4.48. Prikaz komande show collections, vlastiti izvor

Da bismo prikazali sadržaj kolekcije koristimo komandu *find()*. Kako bismo, zbog preglednosti, izbjegli ispis svih objekata u istom redu, koristimo funkciju *pretty()*, koja formatira ispis i ispisuje svaki par ključ-vrijednost u posebnom redu.

```

> db.studenti.find().pretty()
{
  "_id" : ObjectId<"5e7a42e5a1c08e67171625b4">,
  "ime" : "Jelena",
  "prezime" : "Komljenovic",
  "godine" : "20",
  "fakultet" : "PMF"
}
{
  "_id" : ObjectId<"5e7a7a02a1c08e67171625b5">,
  "ime" : "Ana",
  "prezime" : "Markovic",
  "godine" : "17",
  "fakultet" : "PMF"
}
>

```

Slika 4.49. Prikaz sadržaja kolekcije, vlastiti izvor



Ukoliko želimo da dobijemo broj svih dokumenata u kolekciji tada koristimo komandu `count()`:

```
> db.studenti.count()
2
>
```

Slika 4.50. Komanda `count()`, vlastiti izvor

Funkciji **find()** nije potrebno prosljeđivati nikakve parametre ukoliko želimo samo da prikazemo sadržaj kolekcije tj. da prikazemo sve dokumente. Međutim ako želimo pristupiti tačno određenom dokumentu funkciji je potrebno proslijediti **\_id** dokumenta. Prethodno smo napomenuli da se **\_id** dodaje automatski u slučaju da ga mi ne dodamo:

```
> db.studenti.find( { _id : ObjectId <"5e7a42e5a1c08e67171625b4"> } ).pretty()
{
  "_id" : ObjectId<"5e7a42e5a1c08e67171625b4">,
  "ime" : "Jelena",
  "prezime" : "Komljenovic",
  "godine" : "20",
  "fakultet" : "PMF"
}
```

Slika 4.51. Pristupanje određenom dokumentu pomoću **id**-a, vlastiti izvor

Ukoliko želimo da samo određena polja dokumenta budu prikazana tada funkciji **find()** dodjeljujemo drugi(opcioni) parametar. Tačnije ukoliko ne želimo da kao rezultat dobijemo sva polja dokumenta, koristimo projekciju rezultata. Polja u projekciji predstavljaju polja dokumenta, dok vrijednosti u projekciji mogu biti:

- 0 ili false: ukoliko ne želimo da polje bude prikazano
- 1 ili true: ukoliko želimo da polje bude prikazano

Ukoliko ne navedemo ništa, tada će sva polja iz dokumenta biti prikazana. A u slučaju da navedemo, tada će biti prikazana samo ona polja kojima je dodijeljena vrijednost 1 ili true, dok sva ostala polja neće biti prikazana. Polje **\_id** je izuzetak i ono će uvijek biti prikazano, osim ako mu ne dodijelimo vrijednost 0 ili false. Ako pored **\_id**-a želimo da prikazemo još samo ime onda ćemo to uraditi na sljedeći način :

```
> db.studenti.find( { _id : ObjectId <"5e7a42e5a1c08e67171625b4"> }, {ime : 1 } ).pretty()
{ "_id" : ObjectId<"5e7a42e5a1c08e67171625b4">, "ime" : "Jelena" }
```

Slika 4.52. Prikaz samo onih elemenata koji su označeni sa 1 (true), vlastiti izvor

Ukoliko želimo da prikazemo sve osim imena, imenu umjesto 1 dodjeljujemo vrijednost 0 (ili *false* ili *null*):

```
> db.studenti.find( { _id : ObjectId <"5e7a42e5a1c08e67171625b4"> }, { ime : 0 } )
> .pretty()
{
  "_id" : ObjectId<"5e7a42e5a1c08e67171625b4">,
  "prezime" : "Komljenovic",
  "godine" : "20",
  "fakultet" : "PMF"
}
```

Slika 4.53. Prikaz svih elemenata osim onih koji su označeni sa 0 (false ili null), vlastiti izvor

Ako je potrebno pronaći dokumente sa vrijednostima koje su manje ili veće od neke vrijednosti ili vrijednosti koje se nalaze u nekom intervalu tada u upitu koristimo posebno definisana svojstva, što je posebno korisno kada imamo veliki broj dokumenata, a neka od njih su:

- \$gt - pronalazi one vrijednosti koje su veće od navedene vrijednosti
- \$gte – pronalazi vrijednosti koje su veće ili jednake navedenoj vrijednosti
- \$lt - pronalazi one vrijednosti koje su manje od navedene vrijednosti
- \$lte - pronalazi vrijednosti koje su manje ili jednake navedenoj vrijednosti
- \$ne - pronalazi one vrijednosti koje nisu jednake navedenoj vrijednosti
- \$eq - pronalazi one vrijednosti koje su jednake navedenoj vrijednosti
- \$in - pronalazi one vrijednosti koje su jednake nekoj od niza vrijednosti

Ukoliko želimo da pronađemo sve studente koji imaju ispod 25 godina koristićemo sljedeći upit:

```
> db.studenti.find( { godine: { $lt : "25" } } ).pretty()
{
  "_id" : ObjectId<"5e7a42e5a1c08e67171625b4">,
  "ime" : "Jelena",
  "prezime" : "Komljenovic",
  "godine" : "20",
  "fakultet" : "PMF"
}
{
  "_id" : ObjectId<"5e7a7a02a1c08e67171625b5">,
  "ime" : "Ana",
  "prezime" : "Markovic",
  "godine" : "17",
  "fakultet" : "PMF"
}
```

Slika 4.54. Prikaz funkcionisanja svojstva \$lt, vlastiti izvor

Na isti način je moguće koristiti i svojstva koja imaju ulogu logičkih operatora i to su:

- \$and - prikazuje dokumente koji su ispunili uslove oba upita
- \$or - prikazuje dokumente koji su ispunili uslove bar jednog upita
- \$not - prikazuje dokumente koji nisu ispunili uslove upita
- \$nor - prikazuje dokumente koji nisu ispunili uslove nijednog upita

```
> db.studenti.find( { $and : [ { godine : "17" }, { fakultet : "PMF" } ] } ).pretty()
{
  "_id" : ObjectId("5e7a7a02a1c08e67171625b5"),
  "ime" : "Ana",
  "prezime" : "Markovic",
  "godine" : "17",
  "fakultet" : "PMF"
}
```

Slika 4.55. Prikaz funkcionisanja logičkog operatora \$and, vlastiti izvor

Postoje i upiti koji provjeravaju da li se u vrijednosti nekog polja nalazi određena niska. Tačnije moguće je provjeriti da li vrijednost određenog polja počinje ili se završava nekom niskom, kao i da li to polje sadrži neku nisku.

- Nisku navodimo između /^ i / kada želimo da provjerimo da li vrijednost nekog polja počinje tom niskom.
- Nisku navodimo između / i \$/ kada želimo da provjerimo da li se vrijednost nekog polja završava tom niskom.
- Nisku navodimo između / i / kada želimo provjeriti da li vrijednost nekog polja sadrži tu nisku.

```
> db.studenti.find( { prezime : /^K/ } ).pretty()
{
  "_id" : ObjectId("5e7a42e5a1c08e67171625b4"),
  "ime" : "Jelena",
  "prezime" : "Komljenovic",
  "godine" : "20",
  "fakultet" : "PMF"
}
```

Slika 4.56. Provjeravamo da li vrijednost nekog polja počinje zadatom niskom, vlastiti izvor

Pored standardnih opcija dodavanja i pretraživanja dokumenata, dokumente je moguće i ažurirati. Postoje dvije mogućnosti ažuriranja. Prva mogućnost je pomoću metoda update(), kome prosljeđujemo 3 argumenta. Prvi argument predstavlja dokument koji želimo ažurirati, drugi atribut i nove vrijednosti koje

želimo da ažuriramo. Ukoliko ne navedemo drugačije baza će ažurirati samo prvi dokument koji pronađe, a koji ispunjava prethodne uslove. To se može spriječiti tako što ćemo trećem argumentu proslijediti ključ *multi* koji ima vrijednost *true*. Sljedećim upitom ažuriramo sve vrijednosti fakulteta studenata koji imaju manje od 23 godine. Znamo da je prethodna vrijednost fakulteta bila “PMF”, dok je nakon ažuriranja “ETF”.

```
> db.studenti.update( { godine : { $lt : "23" } }, { $set : { fakultet : "ETF" } }, { multi : true } )
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })
> db.studenti.find().pretty()
{
  "_id" : ObjectId<"5e7a42e5a1c08e67171625b4">,
  "ime" : "Jelena",
  "prezime" : "Komljenovic",
  "godine" : "20",
  "fakultet" : "ETF"
}
{
  "_id" : ObjectId<"5e7a7a02a1c08e67171625b5">,
  "ime" : "Ana",
  "prezime" : "Markovic",
  "godine" : "17",
  "fakultet" : "ETF"
}
```

Slika 4.57. Ažuriranje vrijednosti koristeći metod update(), vlastiti izvor

Uz pomoć metoda update() imamo mogućnost ne samo da ažuriramo vrijednosti nego i da preimenujemo već postojeće vrijednosti kao i da obrišemo određena svojstva, što je vrlo korisno ukoliko se radi sa nizovima objekata. Druga, ali dosta manje fleksibilna metoda je metoda save(). Ova metoda prima samo jedan argument. Taj argument mora sadržati id objekta i novi dokument, pomoću kojeg će se zadate vrijednosti spremi pod tim identifikatorom. Samim tim stari dokument biva zamijenjen sa novim. Nedostatak ove metode je to što ova metoda može da ažurira samo jedan dokument, jer je id okarakterisan kao jedinstven.

```
> db.studenti.save( { _id : ObjectId<"5e7a42e5a1c08e67171625b4">, ime : "Milana", prezime : "Jovanovic" } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.studenti.find().pretty()
{
  "_id" : ObjectId<"5e7a42e5a1c08e67171625b4">,
  "ime" : "Milana",
  "prezime" : "Jovanovic"
}
{
  "_id" : ObjectId<"5e7a7a02a1c08e67171625b5">,
  "ime" : "Ana",
  "prezime" : "Markovic",
  "godine" : "17",
  "fakultet" : "ETF"
}
```

Slika 4.58. Ažuriranje vrijednosti koristeći metod save(), vlastiti izvor

Ukoliko želimo izmijeniti ili obrisati određeno polje koristimo operatore \$rename i \$unset. Sada ćemo uz pomoć operatora \$rename polju prezime promijeniti ime na preziva\_se:

```
> db.studenti.update( { ime : "Milana" }, { $rename : { prezime : "preziva_se" } }
>
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.studenti.find().pretty()
{
  "_id" : ObjectId("5e7bf3a5dad6f9f1161232d3"),
  "_id" : ObjectId("5e7a42e5a1c08e67171625b4"),
  "ime" : "Milana",
  "preziva_se" : "Jovanovic"
}
```

Slika 4.59. Primjena operatora \$rename, vlastiti izvor

Operator \$unset se koristi kada želimo obrisati određeno polje dokumenta kao na slici 4.59. U ovom slučaju studentu Milana obrisano je polje preziva\_se. Možemo vidjeti da je polju preziva\_se dodijeljena vrijednost praznih navodnika, zato što operator ne funkcionira ukoliko mu se ne dodijeli neka vrijednost, pa je iz toga razloga dovoljno dodati prazne navodnike kako bismo uspješno obrisali određeno polje.

```
> db.studenti.update( { ime: "Milana" }, { $unset: { preziva_se : "" } }
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.studenti.find().pretty()
{
  "_id" : ObjectId("5e7bf3a5dad6f9f1161232d3"),
  "_id" : ObjectId("5e7a42e5a1c08e67171625b4"),
  "ime" : "Milana"
}
```

Slika 4.60. Primjena operatora \$unset, vlastiti izvor

Takođe postoje i dva osnovna operatora procjene, a to su:

- Operator \$regex – za rezultat daje sve one dokumente koji sadrže određeni izraz koji se navodi između dvije kose crte ( "/" ). Na slici 4.61. možemo vidjeti sve dokumente u čijem se imenu nalazi izraz 'ana'.
- Operator \$expr – dohvata dokumente, korištenjem upita koji uključuje dva ili više polja.

```
> db.studenti.find( { ime : { $regex : /ana/ } } ).pretty()
{
  "_id" : ObjectId("5e7bf3a5dad6f9f1161232d3"),
  "_id" : ObjectId("5e7a42e5a1c08e67171625b4"),
  "ime" : "Milana"
}
```

Slika 4.61. Primjena operatora procjene \$regex, vlastiti izvor

MongoDB baza omogućava i primjenu operatora \$push i \$pop, koje ćemo objasniti nakon što dodamo više zanimanja studentu sa imenom Milana koristeći operator \$set, kako bismo mogli primijeniti ove operatore.

```
> db.studenti.update( { ime: "Milana" }, { $set : { zanimanje : [ "sportista", "kolekcionar" ] } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.studenti.find().pretty()
{
  "_id" : ObjectId("5e7cbd47a4b63ab3e93fb4ee"),
  "ime" : "Milana",
  "zanimanje" : [
    "sportista",
    "kolekcionar"
  ]
}
```

Slika 4.62. Dodavanje više vrijednosti koristeći \$set, vlastiti izvor

Ukoliko se javi potreba da dodamo novu vrijednost određenom polju, a da pri tome ne uklonimo stare vrijednosti tada koristimo operator \$push. Operator \$set se koristi ukoliko želimo obrisati stare, a u isto vrijeme dodati nove vrijednosti. Isto tako ukoliko želimo obrisati određenu vrijednost iz polja vrijednosti koristimo operator \$pull. Nedostatak operatora \$push je to što postoji mogućnost da se doda vrijednost koja već postoji. Ukoliko želimo to da izbjegnemo možemo koristiti operator \$addToSet, koji će dodati vrijednost u polje vrijednosti samo ukoliko se ta vrijednost već ne nalazi u polju.

```
> db.studenti.update( { ime: "Milana" }, { $push : { zanimanje : "novinar" } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.studenti.find().pretty()
{
  "_id" : ObjectId("5e7cbd47a4b63ab3e93fb4ee"),
  "ime" : "Milana",
  "zanimanje" : [
    "sportista",
    "kolekcionar",
    "novinar"
  ]
}
```

Slika 4.63. Dodavanje novih vrijednosti koristeći operator \$push, vlastiti izvor

Kada želimo da izbacimo prvu ili posljednju vrijednost iz polja vrijednosti koristimo operator \$pop. Određenom polju dodjeljujemo vrijednost 1 ukoliko želimo da obrišemo zadnju vrijednost polja što se može vidjeti i na slici 4.64., dok se prilikom brisanja prve vrijednosti polju dodjeljuje vrijednost - 1.

```

> db.studenti.update( { ime: "Milana" }, { $pop : { zanimanje : 1 } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.studenti.find().pretty()
{
  "_id" : ObjectId("5e7cbd47a4b63ab3e93fb4ee"),
  "ime" : "Milana",
  "zanimanje" : [
    "sportista",
    "kolekcionar"
  ]
}
1
>
>

```

Slika 4.64. Izbacivanje vrijednosti koristeći operator \$pop, vlastiti izvor

Koristeći naredbu remove možemo obrisati dokumente iz kolekcije, koji zadovoljavaju zadati uslov. Prvo ćemo dodati novog studenta pod imenom Jovana, a zatim obrisati sve studente koji imaju ime “Jovana”:

```

> db.studenti.find().pretty()
{
  "_id" : ObjectId("5e7cbd47a4b63ab3e93fb4ee"),
  "ime" : "Milana",
  "zanimanje" : [
    "sportista",
    "kolekcionar"
  ]
}
1
>
{ "_id" : ObjectId("5e7cc152a4b63ab3e93fb4ef"), "ime" : "Jovana" }
> db.studenti.remove( { ime : "Jovana" } )
WriteResult({ "nRemoved" : 1 })
> db.studenti.find().pretty()
{
  "_id" : ObjectId("5e7cbd47a4b63ab3e93fb4ee"),
  "ime" : "Milana",
  "zanimanje" : [
    "sportista",
    "kolekcionar"
  ]
}
1
>
>

```

Slika 4.65. Brisanje dokumenata iz kolekcije, vlastiti izvor

I za kraj ćemo obrisati sve dokumente, tj. cijelu kolekciju:

```

> db.studenti.drop()
true
> db.studenti.find().pretty()
>

```

Slika 4.66. Brisanje cijele kolekcije, vlastiti izvor

## Nedostaci NoSQL baza

Iako smo naveli veliki broj prednosti koje se vežu za ove baze, postoje i određeni nedostaci koji su prouzrokovani time što su ove baze počele da se koriste tek prije nekoliko godina i samim tim nisu dovoljno istražene kao što je to slučaj sa SQL bazama. Neki od nedostataka su:

- Nedovoljna razvijenost – činjenica je da se relacione baze podataka koriste dosta više i da su sa godinama sve više i više unaprijeđene, za razliku od NoSQL baza podataka koje su se otkrile dosta kasnije pa samim tim nisu dovoljno istražene i razvijene kao što je to slučaj sa SQL bazama.
- Nepostojanje stranog ključa – za razliku od relacionih baza koje imaju primarni i strani ključ na osnovu kojih se kreiraju veze između tabela, NoSQL baze nemaju takvu mogućnost, upravo zato što nemaju strani ključ.
- Nemaju šemu
- Nemaju standardni jezik za upite- potrebno je za svaki model NoSQL baza iznova učiti jezik, zato što NoSQL baze nemaju standardni jezik
- Administracija – neophodne su tehničke sposobnosti u vidu instalacije i održavanja.
- Mali broj obučenih stručnjaka – napomenuli smo da su se NoSQL baze pojavile relativno skoro što ima za posljedicu nedovoljan broj obučenih programera, koji imaju iskustva sa ovakvim sistemima zbog čega su moguće česte greške.
- Nedostatak sigurne kopije- MongoDB je primjer nedostatka sigurne kopije. Postoje određeni alati pomoću kojih se može izraditi sigurna kopija podataka međutim oni nisu dovoljno razvijeni za razliku od SQL alata.
- Uklanjanje duplih podataka



## IZVORI

- [1] [http://uni-mo.sum.ba/~goran/nastava/8\\_SBP\\_NoSQL.pdf](http://uni-mo.sum.ba/~goran/nastava/8_SBP_NoSQL.pdf) (pristup 14.03.2020)
- [2] <file:///C:/Users/Sony/Downloads/US%20-%20Baze%20podataka%20-%202018%20-%20Singipedia.pdf> (pristup 15.03.2020)
- [3] <https://repozitorij.etfos.hr/islandora/object/etfos%3A1734/datastream/PDF/view> (pristup 17.03.2020)
- [4] <http://scraping.pro/where-nosql-practically-used/> (pristup 13.03.2020)
- [5] <https://bib.irb.hr/datoteka/714321.1-maperokov-primjenaNoSQLnaDMS.pdf> (pristup 15.03.2020)
- [6] [http://poincare.matf.bg.ac.rs/~vladaf/Courses/Matf%20MNSR/Prezentacije%20Individualne/Mijalkovic\\_NoSQL\\_baze\\_podataka.pdf](http://poincare.matf.bg.ac.rs/~vladaf/Courses/Matf%20MNSR/Prezentacije%20Individualne/Mijalkovic_NoSQL_baze_podataka.pdf) (pristup 19.03.2020)
- [7] <http://postel.sf.bg.ac.rs/simpozijumi/POSTEL2018/RADOVI%20PDF/Telekomunikacioni%20saobracaj,%20mreze%20i%20servisi/11.JankovicMladenovicUzelacZdravkovic.pdf> (pristup 15.03.2020)
- [8] <https://prezi.com/xljtbtdkhrbo/nosql-baze-podataka-za-socijalne-mreze/> (pristup 19.03.2020)
- [9] [file:///C:/Users/Sony/Downloads/06\\_Stojanovic\\_Vol4No1\\_2016.pdf](file:///C:/Users/Sony/Downloads/06_Stojanovic_Vol4No1_2016.pdf) (pristup 19.03.2020)
- [10] <https://repozitorij.pmf.unizg.hr/islandora/object/pmf%3A5549/datastream/PDF/view> (pristup 14.03.2020)
- [11] [https://bib.irb.hr/datoteka/718470.1-ivpusic\\_diplomski.pdf](https://bib.irb.hr/datoteka/718470.1-ivpusic_diplomski.pdf) (pristup 14.03.2020)

- [12] <https://repozitorij.etfos.hr/islandora/object/etfos%3A916/datastream/PDF/view>  
(pristup 16.03.2020)
- [13] <https://repozitorij.pmf.unizg.hr/islandora/object/pmf%3A3234/datastream/PDF/view>  
(pristup 15.03.2020)
- [14] <https://www.mongodb.com/download-center/community?jmp=docs>  
(pristup 19.03.2020)
- [15] <https://matfuvit.github.io/UVIT/vezbe/knjiga/Poglavlja/MongoDB/>  
(pristup 20.03.2020)
- [16] <https://repozitorij.unipu.hr/islandora/object/unipu%3A3771/datastream/PDF/view>  
(pristup 18.03.2020)
- [17] [http://www.acs.uns.ac.rs/sites/default/files/7\\_BP\\_Alternativni\\_Pristupi\\_Izgradnji\\_SBP.pdf](http://www.acs.uns.ac.rs/sites/default/files/7_BP_Alternativni_Pristupi_Izgradnji_SBP.pdf) (pristup 19.03.2020)
- [18] [https://www.fer.unizg.hr/download/repository/6\\_NoSQL\\_\(1\\_od\\_3\).pdf](https://www.fer.unizg.hr/download/repository/6_NoSQL_(1_od_3).pdf)  
(pristup 17.03.2020)
- [19] <http://www.milanpopovic.me/redis-moc-jednostavnosti/>  
(pristup 15.03.2020)
- [20] [http://www.racunarstvo.matf.bg.ac.rs/MasterRadovi/2018\\_03\\_24\\_AnaSimijonovic/rad.pdf](http://www.racunarstvo.matf.bg.ac.rs/MasterRadovi/2018_03_24_AnaSimijonovic/rad.pdf) (pristup 13.03.2020)
- [21] [http://poincare.matf.bg.ac.rs/~vladaf/Courses/Matf%20MNSR/Prezentacije%20Individualne%20Stare/Veljkovic\\_NoSql\\_baze\\_podataka.pdf](http://poincare.matf.bg.ac.rs/~vladaf/Courses/Matf%20MNSR/Prezentacije%20Individualne%20Stare/Veljkovic_NoSql_baze_podataka.pdf) (pristup 15.03.2020)
- [22] <https://repozitorij.unipu.hr/islandora/object/unipu%3A2945/datastream/PDF/view>  
(pristup 20.03.2020)

[23]

<https://repozitorij.etfos.hr/islandora/object/etfos%3A838/datastream/PDF/view>

(pristup 16.03.2020)

[24] [https://bib.irb.hr/datoteka/895641.1-Tiljar\\_Mateo\\_Redis.pdf](https://bib.irb.hr/datoteka/895641.1-Tiljar_Mateo_Redis.pdf)

(pristup 18.03.2020)

[25]

<https://repozitorij.etfos.hr/islandora/object/etfos%3A970/datastream/PDF/view>

(pristup 14.03.2020)