

UNIVERZITET U BANJALUCI
PRIRODNO-MATEMATIČKI FAKULTET

NoSQL BAZE PODATAKA
SEMINARSKI RAD

Ime i prezime studenta:

Jelena Komljenović

Mart, 2020. godina

SADRŽAJ

UVOD	1
1.1 NoSQL (nerelacione) baze podataka.....	1
NoSQL modeli distribuiranja podataka	4
Map Reduce koncept	9
Model podataka	14
4.1 Ključ-vrijednost baze podataka	15
4.1.1 Redis baza podataka	19
4.1.2 DynamoDB baza podataka.....	34
4.1.3 Oracle NoSQL baza podataka	35
4.1.4 Riak baza podataka	36
4.2 Model familija kolona	36
4.2.1 Apache Cassandra baza podataka.....	37
4.3 Graf baze podataka.....	42
4.3.1 Neo4j baza podataka	45
4.4 Dokument baze podataka	52
4.4.1 MongoDB baza podataka	54
Nedostaci NoSQL baza	64
IZVORI	65

UVOD

U poslednjih par decenija tehnologija se razvija velikom brzinom, a kao posljedica javlja se to da relacione baze podataka nisu uvijek odgovarajuće za skladištenje i obradu podataka i u tome leži motivacija za nastanak NoSQL baza. Kako bismo shvatili pojam NoSQL baza podataka, potrebno je definisati šta su to zapravo baze podataka i čemu one služe.

“Baza podataka je način organizovanja i pretraživanja uzajamno povezanih podataka. Baza podataka pomaže u organizovanju informacija u logičkom smislu kako bismo imali mogućnost da brže dođemo do željenih podataka. Ona može sadržati podatke u raznim oblicima, od jednostavnog teksta (kao što je e-mail adresa) do kompleksnih struktura koje uključuju sliku, zvuk i slično.”

Postoje dva glavna tipa baza podataka, a to su: SQL i NoSQL, odnosno relacione i nerelacione baze podataka. Razlika je u tome kako su građene, tipu podataka koje pohranjuju te kako ih pohranjuju. Relacione baze su osmišljene kao telefonski imenik koji sadrži brojeve i adrese korisnika, dok su nerelacione baze više kao dokument, odnosno kao mape datoteka koje sadrže sve podatke o osobi od njegove adrese i broja telefona pa do toga šta ta osoba voli kupovati, jesti itd.

1.1 NoSQL (nerelacione) baze podataka

NoSQL baze podataka nemaju strogu definiciju. NoSQL baza podataka je dizajnirana za pohranu, distribuciju i pristup podacima i to čini pomoću metoda koje se razlikuju od relacionih baza. Naziv NoSQL nastao je 1998. godine upravo zato što je SQL simbol relacionih baza podataka. NoSQL, ne isključuje SQL, zapravo NoSQL znači: “Ne samo SQL” (“Not only SQL”). Bez obzira na bukvalno značenje, NoSQL se danas koristi kao opšti termin za sve baze podataka i skladišta podataka koji ne prate popularne i dobro utvrđene principe relacionih baza podataka i često se odnose na velike skupove podataka kojima je potrebno brzo i efikasno pristupiti i menjati ih na Vebu. To znači da NoSQL nije jedinstveni proizvod, čak ni jedinstvena tehnologija nego da predstavlja klasu proizvoda i koncepata u vezi skladištenja podataka i njihovog upravljanja.

NoSQL baze su nastale iz zahtjeva za većom fleksibilnošću i boljim performansama u pohrani i obradi velike količine podataka, uglavnom zbog

popularnosti interneta i internetskih tehnologija i sve veće količine podataka. Za razliku od relacionih baza podataka kod kojih se podaci organizuju u skup relacionih tabela (relacija), gdje svaka relacija mora da ima primarni ključ, odnosno skup atributa pomoću kojeg se identifikuje svaka n-torka, nerelacione baze pohranjuju podatke na drugačiji način. NoSQL baze podataka rade sa dokumentima umjesto sa strogo definisanim tabelama podataka. Baze usmjerene na dokumente prikupljaju podatke iz dokumenata i omogućavaju da se oni dohvate u organizovanom obliku. Na ovaj način nestrukturisani podaci kao što su slike, videozapisi ili novinski članci mogu biti spremljeni u jedan dokument koji se lako može pronaći bez kategorizacije polja kao u relacionim bazama. Ovakva organizacija podataka zahtijeva dodatne napore obrade te veći prostor za pohranu nego što je to kod visoko organizovanih SQL podataka, ali zato omogućavaju bržu obradu određenih zahtjeva. Prilikom pohrane podataka u NoSQL bazu nije potreban veliki rad na dizajnu. Moguće je početi s kodiranjem, pohranom i dohvatanjem podataka bez prethodnog znanja kako baza sprema podatke ili kako ona radi. Prethodno nepoznavanje šeme je možda i jedna od najznačajnijih razlika između nerelacionih i relacionih baza podataka. Veliki problem relacionih baza je to što svi entiteti unutar jedne relacije moraju imati iste attribute. Ovo može biti prednost, ukoliko su podaci s kojima radimo zaista uvijek jednako strukturisani pa, već na nivou baze, možemo osigurati poštovanje pravila. Ali, u stvarnom svijetu, izuzeci su česti. Zamislimo da želimo zapisati neku vrstu zapažanja vezanu za nekog programera. Iako nam zapažanje treba samo za taj jedan entitet, prisiljeni smo dodati niz kolona "zapažanja" nad cijelom tabelom. Osim što je spora, naredba ALTER TABLE zaključaće sve podatke u toj i svim povezanim tabelama, dok se ne izvrši u potpunosti. Za to vrijeme, podaci neće biti dostupni ni za čitanje. Veliki servisi s milionima korisnika takvu situaciju ne smiju dopustiti, jer nedostupnost podacima znači finansijski gubitak. Prednost NoSQL baza je što one ne zahtijevaju šemu podataka pa se brzo i lako prilagođavaju novim poslovnim zahtjevima.

Postoje zajednička obilježja koja se odnose na sve NoSQL baze podataka:

- Nisu relacione – ne koriste relacioni model niti SQL jezik
- Podaci se čuvaju u posebne strukture podataka
- Većina je otvorenog koda (eng. Open source)
- Visoke performanse u radu sa velikom količinom podataka
- Vrlo dobro rade kao distribuirane baze – rad na više čvorova (računara) sa više baza

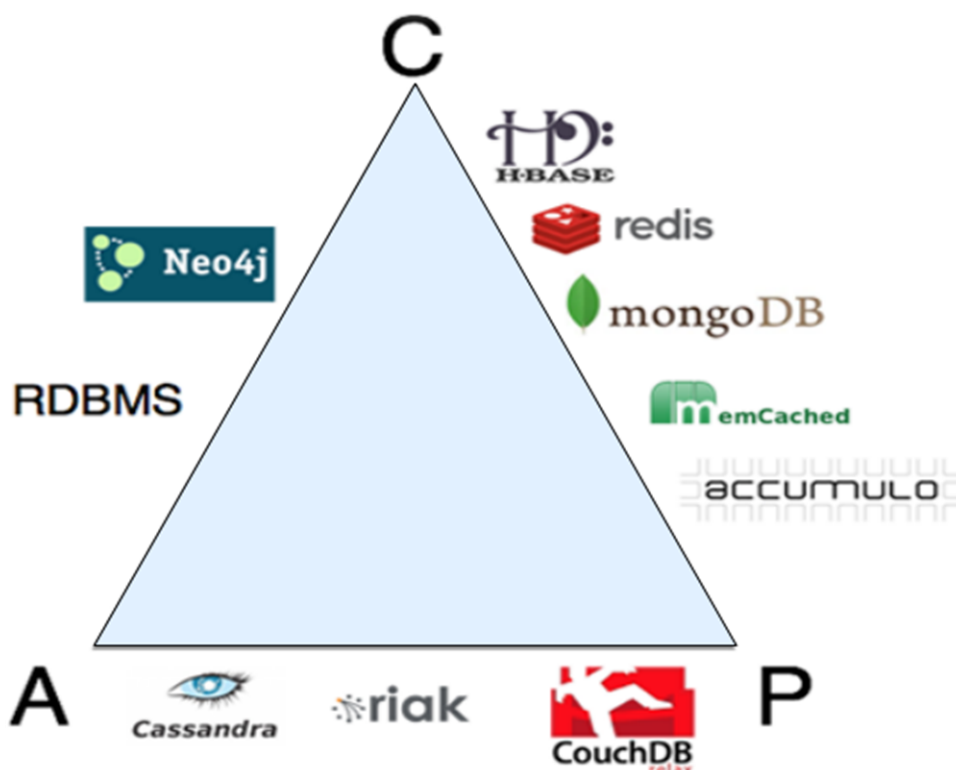
- Tolerancija na prekide u mreži
- Baze 21. vijeka – „NoSQL“ se ne odnosi na baze koje su postojale prije SQL-a niti na objektno orijentisane baze podataka
- Nemaju šemu (eng. Schema-less) - kada se kaže da ovakve baze podataka nemaju šemu, ta izjava nije u potpunosti tačna. Sama baza možda nema strogo određenu šemu, ali to ne znači da aplikacija vezana za bazu takođe nema šemu. Dakle, postoji tzv. implicitna šema. S jedne strane možemo mijenjati bazu, dodavati i uklanjati podatke po želji bez definisanja strukturnih promjena, ali opet moramo pratiti podatke i način na koji se njima upravlja i pronaći implicitnu šemu da bismo bili u mogućnosti manipulirati podacima.

Za ove baze podataka često se vežu i BASE svojstva, za razliku od SQL baza za koje se vežu ACID (Atomicity, Consistency, Isolation, Durability) svojstva:

- Basically Available – Garantovana dostupnost
- Soft state - Stanje baze se može promijeniti, čak i u slučaju kada nemamo eksplicitne upite nad bazom podataka. Razlog tome je što ažuriranje može doći s ažuriranjem čvora kojem pripada baza podataka.
- Eventually consistent - Sistem će postati konzistentan s vremenom

Osim BASE svojstava, za NoSQL baze podataka često se veže i CAP teorema koja govori da je moguće ostvariti samo dva od sljedeća tri aspekta u isto vrijeme (Weber):

- Consistency (konzistentnost) - svako čitanje iz baze podataka kao rezultat ima najnoviju verziju podataka.
- Availability (dostupnost) - Podaci uvijek moraju biti dostupni.
- Partition Tolerance (tolerancija razdvojenosti) - Baza podataka radi normalno i u slučaju prekida u mreži ili na računaru.



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

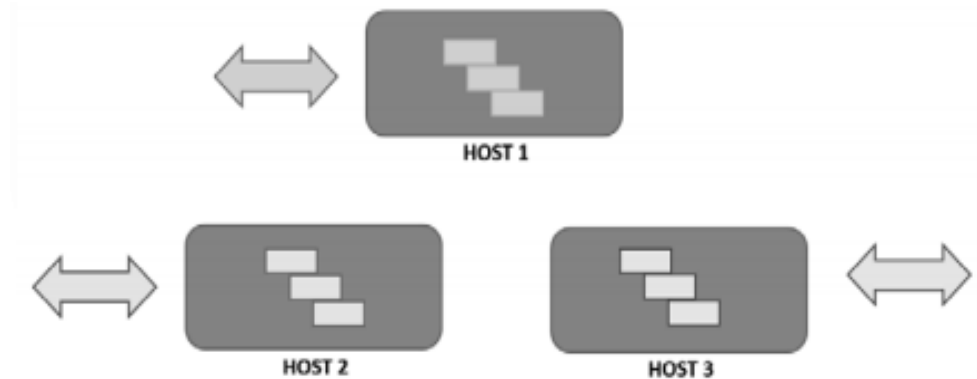
NoSQL modeli distribuiranja podataka

Jedan od najvažnijih zadataka koje bi trebalo da ispunjavaju NoSQL sistemi je održavanje podataka distribuiranih u klasteru. Dakle hostovi, koji čuvaju elementarne podatke i koji učestvuju u davanju podataka, obrazuju logičku grupu nazvanu klaster. Postoje različiti načini distribuiranja:

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

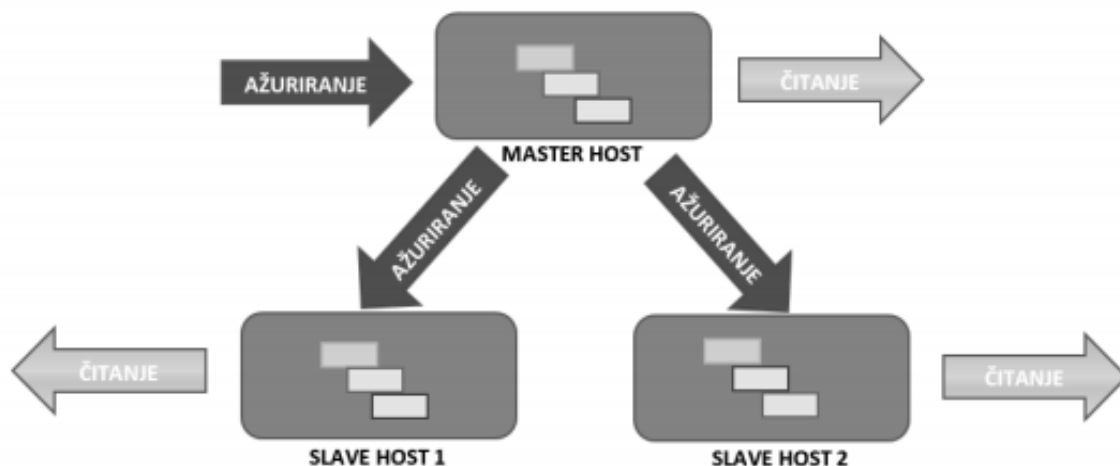
Potpuno dijeljenje je model kod kojeg se na svakom hostu u klasteru nalaze različiti podaci. Ne postoje kopije podataka na drugim hostovima tako da je konzistentnost podataka lako održiva. Ovaj model je pogodan u slučaju da aplikacije koje pristupaju lokalnim NoSQL DBMS hostovima takođe imaju lokalni

karakter. Drugim riječima one uopšte ne traže, ili veoma rijetko traže podatke sa drugih hostova u klasteru. Kod potpunog dijeljenja svaki host je odgovoran za razmjenu (čitanje, upis, ažuriranje) samo podataka koje skladišti.



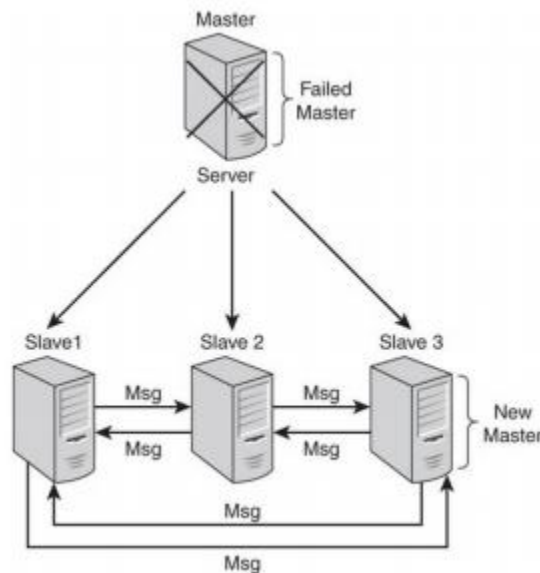
Slika 2.1. Potpuno dijeljenje, izvor: [2]

Master-slave replikacije su model koji predstavlja potpunu suprotnost prethodnom načinu distribuiranja i kod kojeg se u svim hostovima u klasteru nalaze isti podaci. To znači da je izmjena podataka moguća samo na master hostu, dok je na ostalim hostovima (koji imaju slave ulogu) dozvoljeno samo čitanje, ali ne i promjena podataka. Iz ovakve podjele uloga slijedi još jedan zadatak mastera, a to je održavanje konzistentnosti podataka, odnosno ažuriranje replika podataka na svim slave hostovima. Uslov za uspješnost primjene ovog modela je pouzdana mrežna infrastruktura i velika brzina protoka podataka. Slave hostovi preuzimaju i primaju zahtjeve samo ako master otkáže. Prednost ove replikacije je jednostavnost koja se ogleda u tome da svaki čvor u oblaku, osim master hosta koji komunicira sa svima, mora komunicirati samo s jednim hostom – master hostom.



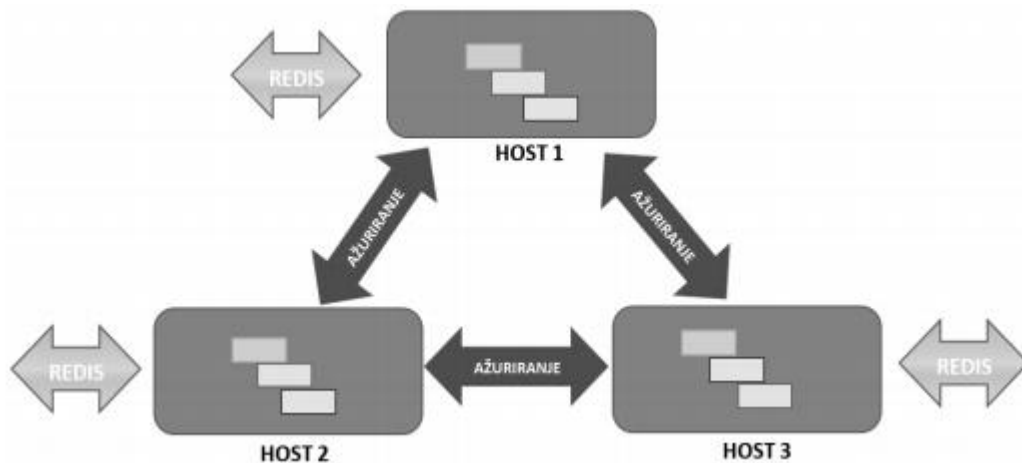
Slika 2.2. Master-slave replikacija, izvor: [2]

U slučaju da se master host pokvari, slave hostovi pokreću protokol kojim postavljaju jedan slave host za master host, koji će onda prihvatati operacije za čitanje i pisanje.



Slika 2.3. Nakon kvara master hosta, slave hostovi pokreću proceduru biranja novog master hosta, izvor: [3]

Replikacije na ravnopravnim (peer to peer) hostovima su model koji se razlikuje u odnosu na prethodni po tome što svi hostovi u klasteru mogu da čitaju i da mijenjaju podatke (označeno kao REDIS – engl. Read Edit Delete Insert Select). Ovaj model znatno otežava održavanje konzistentnosti podataka u klasteru, jer promjena podatka na nekom od hostova utiče i na ostale hostove. U tom slučaju NoSQL sistem mora da podrži transakcionu obradu, odnosno da se promjene na svim hostovima odigraju u jednoj transakciji. Može se zaključiti da su ovakvi sistemi ugroženi u slučaju gubitka komunikacionog kanala za ažuriranje pošto postoji mogućnost distribucije netačnih podataka korisničkim aplikacijama. Stalno testiranje linkova prije i u toku transakcije radi prihvatanja ili odbijanja transakcije nije dovoljna mjera da bi se uspostavila pouzdanost ovakvih distribucionih sistema.



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

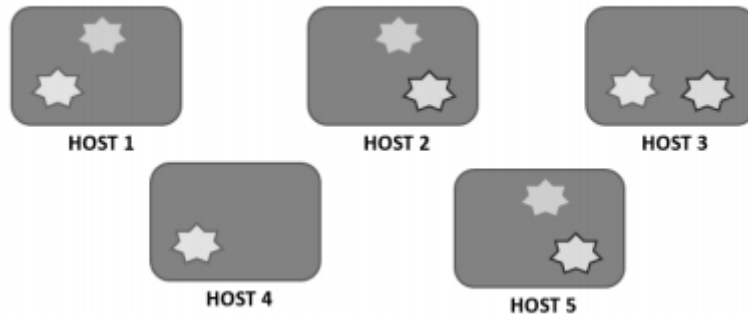
Mogućnost pojave nekonzistentnih podataka u gore opisanim situacijama doveo je do pojave hibridnog modela koji je nastao kao kombinacija dijeljenja i replikacije. Postoje 2 varijante ovog modela. U prvoj varijanti (slika 2.5.) hostovi mogu da budu u isto vrijeme masteri za neke podatke (skladište njihove originale i ažuriraju njihove replike), a za druge podatke da imaju ulogu slave hosta (samo skladište kopije). Takođe u prvoj varijanti hostovi mogu da budu samo master, ili samo slave hostovi. Pravilo distribuiranosti je da u klasteru postoji jedan original podatka i da se zna na kojem hostu se nalazi. Samo tako može da se implementira uspješno ažuriranje podataka.



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

Na slici host 1 ima ulogu mastera za 2 podatka (podaci koji su prikazani zvijezdama), a slave hosta za treći. Host 2 je master za jedan podatak, a slave za 2, dok je host 3 slave host.

Druga varijanta kombinacije dijeljenja i replikacije podrazumijeva ravnopravnost hostova (peer to peer hostovi). To znači da svi hostovi imaju ulogu mastera i da replika podataka nema, već da sve instance podataka predstavljaju originale.



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

Prilikom promjene podataka na bilo kom hostu, on preuzima odgovornost za ažuriranje tog podatka na svim ostalim hostovima. Drugim riječima svaki host mora da zna za hostove u klasteru na kojima je skladišten taj isti podatak. U praksi korištenja NoSQL podaci su distribuirani faktorom tri, što znači da je svaki distribuiran na 3 čvora u mreži (hosta u klasteru). Na taj način je povećana otpornost sistema na padove mreže, odnosno klaster se dijeli na dijelove koji se dalje pretvaraju u prvobitne, nakon uspostavljanja normalnog transfera podataka. Da bi se ovo omogućilo NoSQL DBMS-ovi pamte vremena promjene svih podataka koje skladište u klasteru, što rezultira konzistentnom rekonstrukcijom podataka prilikom novog uspostavljanja komunikacionog kanala. Za skladišta sa familijama kolona je karakteristično da koriste ovaj način distribuiranja podataka.

Map Reduce koncept

Map Reduce koncept je nastao kao proizvod potrebe da se podaci predstavljeni agregacionim modelom efikasno razmjenjuju između hostova. Implementacija ovog koncepta sadrži dvije faze: mapiranje i redukciju.

Prva faza, mapiranje, odgovorna je za izvlačenje, formatiranje i filtriranje podataka. Za efikasno izvršavanje ove faze, bitno je da su podaci dostupni procesoru koji obrađuje map funkciju. Ona čita podatke iz baze, razlaže zahtjev u niz nezavisnih funkcija, koje se mogu izvršiti posebno i raspoređuje te funkcije na različite procesore. Rezultat rada mapiranja svih procesora je serija ključ–vrijednost. Mapiranje se vrši pomoću funkcije koja kao ulazni parametar ima agregiran podatak, a kao izlaz daje niz podataka predstavljenih kao parovi 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

Svega: 7192

Plaćanje:
AMEX
14803-9465-01270-43

U ovim tabelama se nalazi račun koji predstavlja agregiran podatak i koji je ulazni parametar funkcije za mapiranje, a rezultat njenog izvršavanja će biti pojedinačne stavke računa predstavljene po ključ-vrijednost principu. Ključ svake stavke je naziv proizvoda, a vrijednost je struktura koja ima 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

S obzirom da se radi o podacima koji predstavljaju agregate, drugim riječima, koji su sastavljeni od distribuiranih podataka na različitim hostovima, funkcija mapiranja se u realnosti izvršava istovremeno na svim hostovima na kojima se nalaze podaci od interesa. Ovakav pristup omogućava veliku brzinu procesiranja podataka kako zbog paralelizma obrade, tako i zbog lokalnog karaktera podataka koji se obrađuju. Drugim riječima, host obrađuje u ovoj fazi samo podatke koje skladišti.

Funkcija za mapiranje se izvršava na jednom zapisu – u prethodnom primjeru to je jedan račun. Ako pretpostavimo da je cilj obrade da se prikupe podaci kupovine određenog proizvoda, funkcija za mapiranje će se izvršiti za sve račune svih kupovina. U tom slučaju kao rezultat se dobijaju ključ-vrijednost parovi čiji broj može da se redukuje u skladu sa zadatkom obrade. Redukcija parova ključ-vrijednost predstavlja drugu fazu Map Reduce koncepta. Redukcija prima parove ključ-vrijednost kao ulaz i izvršava tražene operacije. Tako dobijeni podaci mogu biti sortirani, grupisani ili sažeti. Obrađeni podaci se vraćaju procesoru koji ih je podijelio te ih on spaja, kako bi se dobio završni rezultat koji odgovara

klijentovom zahtjevu. Kao rezultat redukcije umjesto velikog broja ključ-vrijednost parova, host emituje samo jedan par, koji predstavlja konačan rezultat obrade.

Jabuka	
Cijena	2.45
Količina	5

Jabuka	
Cijena	37.24
Količina	76

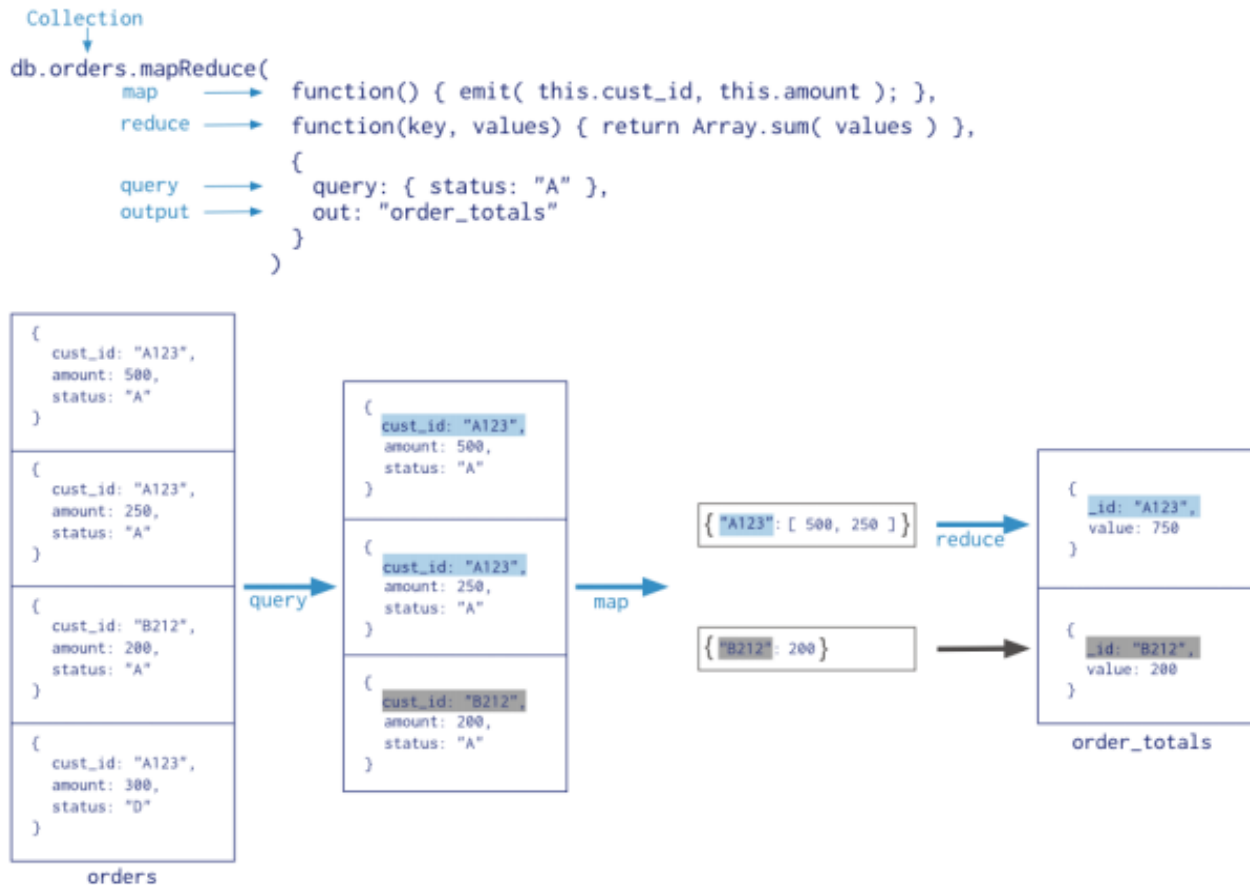
Jabuka	
Cijena	9.31
Količina	19

Jabuka	
Cijena	20.58
Količina	42



Jabuka	
Naplaćeno	69.58
Količina	142

Još jedan primjer na kojem se može uočiti kako funkcioniše Map Reduce koncept je primjer u MongoDB sistemu:



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

U ovom primjeru je vidljivo da sistem primjenjuje *map* funkciju na svaki dokument koji zadovoljava upit (upit vraća sve dokumente koji imaju status "A"). *Map* funkcija vraća ključ-vrijednost parove. Za one ključeve koji imaju više vrijednosti, sistem primjenjuje i *reduce* funkciju, koja u ovom primjeru daje zbir svih vrijednosti. *Reduce* funkcija prikuplja, sažima i agregira podatke dobijene kroz *map* funkciju. Na kraju, sistem sprema rezultate u "order_totals" collection. Opcionalno je korištenje i *finalize* funkcije, koja rezultate *reduce* funkcije dodatno može sažeti. Generalno, *map-reduce* funkcije u MongoDB sistemu mogu kao ulaz primati dokumente iz jednog collection-a i napraviti bilo kakva sortiranja ili limite prije započinjanja *map* funkcije. *Map-reduce* može vratiti rezultat kao dokument, ili može zapisati rezultat u collection, kao u primjeru.

Jedan od najpoznatijih i najčešće korištenih algoritama za Map Reduce model je Apache-ov Hadoop.

Hadoop je radni okvir (engl. framework) otvorenog koda, dakle, besplatan je i dostupan svima za upotrebu. Njegovi zadaci su biranje procesora za mapiranje, raspoređivanje ključ-vrijednost parova na procesore za redukciju i osiguravanje uspješnog izvršavanja zadatka, čak i kod greške u radu sistema. To su ujedno i najveći opšti problemi map i reduce operacija, pa na programeru aplikacije ostaju samo najjednostavniji zadaci. Hadoop se koristi u većini poznatih NoSQL sistema, osim MongoDB-a, koji ima vlastitu implementaciju ovog modela.

Memcache sistem je takođe otvorenog koda. Uprkos tome što su distribuirani sistemi daleko ubrzali i pojeftinili rad računara sa velikim količinama podataka, problem veličine RAM-a nije bio u potpunosti riješen. Iako je zbir memorije RAM-a bio i do nekoliko puta veći nego kod jednog super-računara, on je se na ovakvim sistemima dosta manje mogao iskoristiti. Naime, velika prednost RAM-a je mogućnost keširanja rezultata čestih upita i držanje “pri ruci” najkorištenijih podataka. Budući da u distribuiranom sistemu svaki procesor ima svoj vlastiti RAM, često se dešavalo da su podaci spremljeni u RAM-ovima različitih procesora jednaki. Takođe, moglo se desiti i da čvor izvršava ispočetka složen upit ili dohvata veliku količinu podataka iz baze, ne znajući da njegov susjedni čvor već ima rezultate spremne u svom RAM-u. Zbog svega navedenog, globalno popularne web stanice su gotovo na dnevnom nivou morale dodavati nove čvorove u mrežu, kako bi osigurale najbolji korisnički doživljaj velikom broju posjetilaca. Upravo iz tog razloga, inženjeri blog sistema LiveJournal su krenuli tražiti rješenje. Prvo su uveli specifičnu notaciju upita kako bi cijeli upit mogli spojiti u kratki hash string. Zatim su razvili sistem komunikacije između čvorova u mreži, koji su nazvali Memcache. Kada čvor dobije upit koji treba izvršiti, on prvo putem protokola pošalje spomenuti hash upit svim drugim čvorovima. Ako neki od čvorova koji primi hash, ima spremljen rezultat u RAM-u, on taj rezultat šalje nazad čvoru koji je poslao upit. Originalni čvor sada ima rezultate svog upita, bez da je uopšte komunicirao sa bazom. Na taj način je Memcache omogućio dijeljenje RAM-a između procesora u mreži i dodatno poboljšao mogućnosti distribuiranih sistema.

Lucene je moćan paket otvorenog koda koji se bavi indeksiranjem i potpunom pretragom tekstualnih dokumenata. Originalan paket je napisan u programskom jeziku Java, ali danas postoje integracije za mnoge druge programske jezike. Kao posrednik za komunikaciju s Lucene-om, često se koristi Apache-ov web servis Solr. Prednost korištenja posrednika je korištenje standardnog HTTP protokola, kojeg Lucene sam po sebi ne podržava. Takođe, Solr obavlja poslove replikacije,

keširanja, prevođenja dokument bilješki (ako postoje) i slično. Zbog uske povezanosti, ova dva proizvoda se nerijetko zamjenjuju, ali njihov odnos može se dobro opisati sljedećom metaforom. Solr možemo zamišljati kao jako dobar auto, a Lucene kao snažan motor koji ga pokreće. Lucene i Solr primaju tekstualne dokumente. Ti dokumenti mogu biti u gotovo bilo kakvom formatu – JSON, PDF, Word, HTML, itd. U našem kontekstu, baza će kao dokumente Solr-u slati vrijednosti koje želi indeksirati. Solr ih zatim šalje Lucene-u, koji razbija dokument na riječi i pravi indekse za njih. Svaki indeks za riječ sadrži jedinstven ID riječi, broj dokumenata u kojima se riječ nalazi i pozicije riječi u tim dokumentima. Svi upiti koje baza dobija, opet se šalju na Solr, koji zatim koristi Lucene za pretragu i vraća rezultate. Stvaranje ovakvih indeksa veliki je teret kod dodavanja novih dokumenata i ažuriranja, ali budući da taj teret nije direktno na bazi, nije problem prihvatiti ga. Solr omogućava vrlo jednostavno i brzo pronalaženje tražene riječi u velikoj količini dokumenata. Ukoliko pretraga sadrži više riječi, traži se presjek skupa dokumenata koji sadrže tražene riječi. Lucene koristi različite algoritme kako bi rangirao rezultate. Na primjer, kod pretrage jedne riječi, najrelevantnijim rezultatom možemo smatrati onaj dokument koji sadrži najviše ponavljanja te riječi, pa će on biti vraćen na prvom mjestu.

Model podataka

Model podataka je zapravo glavna stvar po kojoj se NoSQL sistemi za upravljanje bazama podataka razlikuju od relacionih sistema. Model podataka predstavlja način na koji prikazujemo podatke i upravljamo njima. Kod baze podataka, on opisuje interakciju sa podacima u bazi, odnosno organizaciju podataka u istoj. NoSQL baze možemo podijeliti upravo po tom modelu podataka na četiri različite kategorije od kojih svaka ima svoja obilježja i zato svaka odgovara specifičnim zahtjevima i tipu projekta za koji se koriste:

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

4.1 Ključ-vrijednost baze podataka

Ključ-vrijednost baze podataka predstavljaju najjednostavniji tip NoSQL baza podataka. Dobro je poznato da je jedini "jezik" koji računari razumiju mašinski jezik, čija je abeceda binarna. To znači da računar svaki podatak koji spremamo u njega pretvara u niz nula i jedinica. Ključ-vrijednost baze funkcionišu na sličan način. Ovakve strukture najlakše je zamisliti kao tabele sa dva atributa – ključ i vrijednost. Za razliku od relacionog modela, ovdje vrijednosti koje spremamo nemaju definisan domen. One mogu biti gotovo bilo šta – objekat, tekstualni dokument, slika, video, zapis zvuka, itd. Prije spremanja, potrebno je datu vrijednost prevesti u skup binarnih podataka i dodijeliti joj zadani ključ. Ovako spremljene binarne podatke nazivamo BLOB, od engl. Binary Large Objects, u prevodu veliki binarni objekti. Bazi nije poznat tip vrijednosti niti sadržaj koji ona predstavlja. Klijent sam mora brinuti o tom dijelu koda. Ključ je proizvoljan string – hash generisan iz vrijednosti adresa web stranice, SQL upita ili slično, koji ne prevazilazi maksimalnu dužinu. Dopustena dužina zavisi od implementacije. Veličina BLOB-a je gotovo proizvoljna i ograničena jedino fizičkim prostorom za pohranu, koji baza ima na raspolaganju. Operacije koje ove baze pružaju su dodavanje novog para ključa i vrijednosti te ažuriranje, dohvaćanje i brisanje vrijednosti preko ključa:

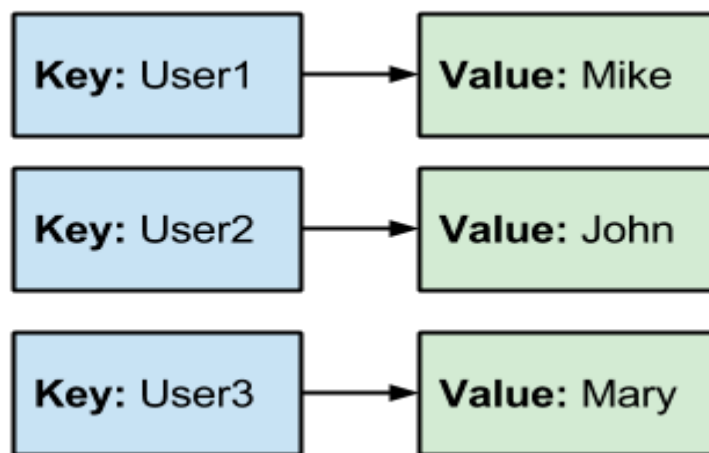
- PUT(ključ, vrijednost) – Dodaje novi par (ključ, vrijednost) ili pridružuje novu vrijednost postojećem ključu.
- GET(ključ) – Vraća vrijednost pridruženu zadanom ključu.
- DELETE(ključ) – Uklanja par (ključ, vrijednost), ako ključ ne postoji može javiti grešku.

Pored gore navedenih operacija, za ovakve baze podataka u opštem slučaju vrijede dva pravila:

- Različiti ključevi – Svaki ključ u tabeli mora biti jedinstven.
- Nema upita na osnovu vrijednosti – Upit može biti baziran samo na ključu, a ne na vrijednosti. Drugim riječima, u ovakvoj bazi podataka ne mogu se postavljati upiti kojima se traži određena vrijednost, nego samo upit kojim se traži određeni ključ. Moguće je definisati i "rok trajanja" ključa, nakon čijeg isteka se ključ i vrijednost koja mu je dodijeljena brišu iz baze. Ovaj model je najprikladniji za spremanje korisničkih podataka, odnosno

razmjene podataka, jer je, u originalu, napravljen upravo za tu svrhu. Čest primjer upotrebe je i implementacija rječnika. Razlog tome su očigledne sličnosti tih struktura podataka. Kod rječnika su riječi ključevi, a njihov izgovor, definicije i sinonimi su vrijednosti. Riječi su posložene po abecedi pa ih lako pretražujemo, na sličan način kao što baza pretražuje stablo indeksa. Zbog fleksibilnosti vrijednosti, ovaj model se često koristi i za spremanje multimedijalnog sadržaja, koji predstavlja problem za standardni relacioni model. Takođe, zbog brzine čitanja, čak i neki relacioni sistemi koriste ovaj model kako bi ubrzali pretraživanja, tako što spremaju rezultate prošlih upita (engl. query cache).

Na sljedećoj slici je prikazan jednostavan primjer tabele sa ključem i odgovarajućom vrijednosti:



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

rukovanje podacima je jednostavna. Potrebno je definisati imenski prostor (engl. namespace), koji je najčešće ime baze, te ključ koji pokazuje na operaciju koja se želi primijeniti na paru ključ-vrijednosti. Kada se navede samo imenski prostor i određeni ključ, baza će vratiti vrijednost za taj ključ. Ako je potrebno ažurirati vrijednost za određeni ključ, potrebno je navesti imenski prostor, ključ i novu vrijednost za taj ključ. Ova baza podataka je takođe fleksibilna i pruža mogućnost greške. Ako se greškom unese pogrešan tip podatka, na primjer, realan broj umjesto cijelog broja, baza podataka neće izbaciti grešku. Ovo svojstvo je korisno kada se tip podataka mijenja ili kada je potrebno podržati dva ili više tipa podataka za isti atribut. Jedna od prednosti jednostavnih struktura baza podataka je da omogućavaju veću brzinu izvršavanja operacija.

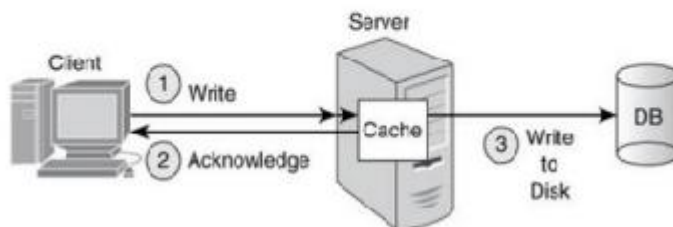
2. Distribuiranost

Distribuiranost je važna kako bi se osigurala mogućnost upravljanja novim korisnicima i podacima kako raste aplikacija i poslovanje. Kod ključ-vrijednost baza podataka postoje dva načina distribuiranja:

1. Master-slave replikacija - jedan od načina da se održi korak s rastućim brojem read operacija je da se doda host koji može odgovoriti na upite.
2. Master – master replikacija, koja je prethodno objašnjena

3. Brzina

Ključ-vrijednost baze podataka su poznate po svojoj brzini. Jednostavnim asocijativnim nizom strukture podataka te svojstvima dizajna za optimizaciju performansi, ključ-vrijednost baze podataka omogućavaju visoku propusnost te intenzivne operacije podataka. Jedan od načina za održavanje brzine rada baze podataka je zadržavanje podataka u memoriji. Čitanje i pisanje u RAM-u (engl. Random Access Memory) je puno brže nego pisanje na disk. S obzirom da RAM nije trajna memorija, prilikom nestanka napajanja, sadržaj RAM-a će se izgubiti. Kada program promijeni vrijednost povezanu s ključem, baza može ažurirati unos u RAM i poslati poruku programu da je ažurirana vrijednost spremljena, dok program može nastaviti s drugim operacijama, a za to vrijeme baza može zapisati ažuriranu vrijednost na disk. Pogledati sliku 4.2.



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

Drugi način održavanja velike brzine rada s podacima je SSD (engl. Solid State Drive). SSD je super brza flash memorija za pohranu velikih sinhronih zapisa tako da RAM može biti rezervisan za nove podatke. Neke baze podataka podržavaju SSD pohranu kako bi se osigurala maksimalna propusnost. Postoje situacije u kojima nije dobro koristiti ključ-vrijednost baze podataka. Neke od njih su sljedeće:

- Kada bi trebalo povezati različite setove podataka ili povezati podatke između različitih skupova ključeva.
- Kada bi trebalo vratiti neku od operacija zbog prethodno neuspješnog spremanja ključeva.
- Kada bi trebalo pronaći ključ na temelju vrijednosti iz ključ-vrijednost para.
- Kada bi trebalo upravljati sa više ključeva.

Međutim postoje slučajevi u kojima je korištenje ključ-vrijednost baze podataka vrlo efikasno, a neki od njih su:

- Spremanje podataka kod korisničkih profila, informacija o sesijama i elektronske pošte
- Spremanje podataka kod Internet trgovine kao što su sadržaji korpe, kategorije proizvoda, detalji o proizvodu te recenzije proizvoda.
- Povjerljivost podataka, tabele za prosljeđivanje internet protokola, itd.

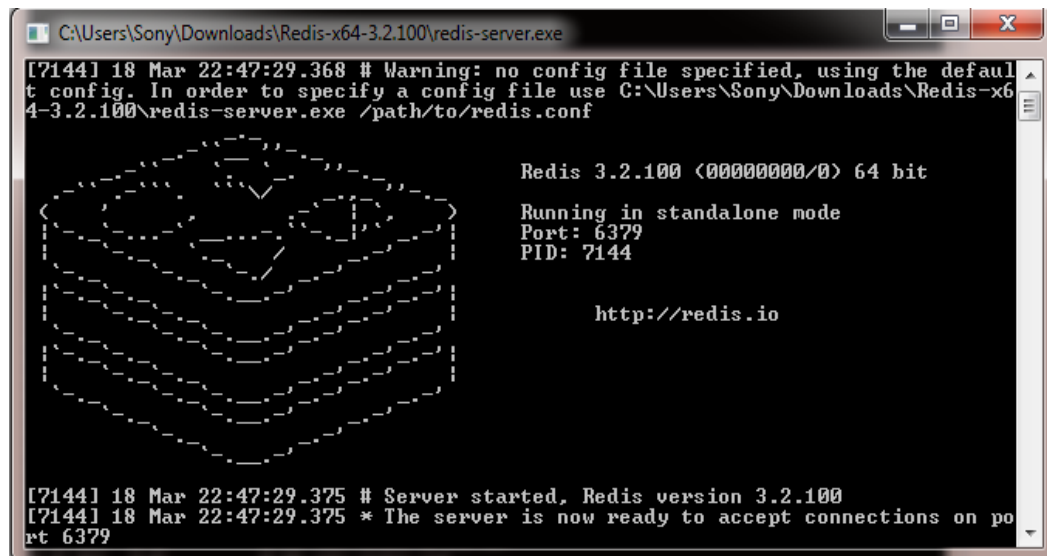
Predstavници koji su bazirani na ovom konceptu su: DynamoDB, Riak, Redis, Oracle NoSQL i mnogi drugi.

4.1.1 Redis baza podataka

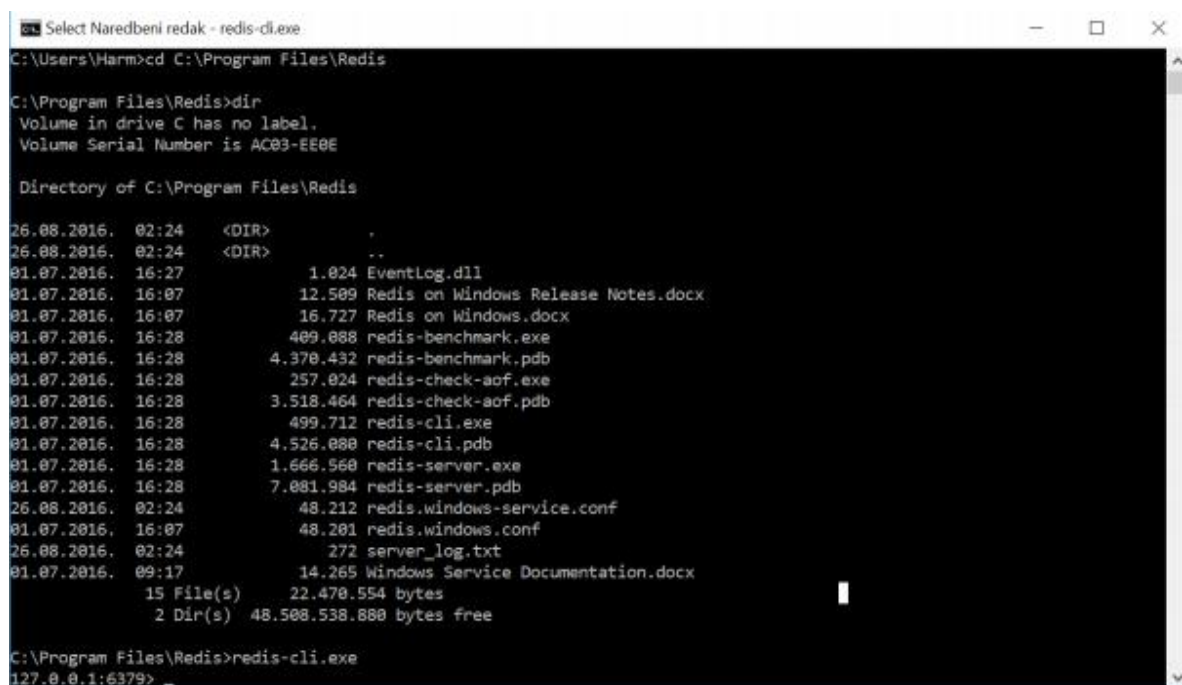
Redis je jedna od najpoznatijih ključ-vrijednost baza podataka. To je open-source memorijska jednonitna baza podataka koja podržava apstraktne strukture podataka kao što su nizovi, liste, mape, hash tabele, bitmape itd. Redis omogućava veliku brzinu izvođenja operacija pisanja tako što ograničava veličinu setova podataka koji ne mogu biti veći od memorije. U slučaju nedostatka memorije Redis proces će biti prekinut, tačnije Redis će se srušiti sa javljenom greškom ili će usporiti. S obzirom na to da je jednonitna baza podataka, ne može se koristiti za paralelno izvršavanje zadataka kao što su pohranjene procedure. Redis se može konfigurisati za spremanje podataka nakon određenog broja sekundi. Na primjer, može se naznačiti da se spremaju podaci nakon 100 promjena i najmanje 5 sekundi. S obzirom da se podaci spremaju asinhrono, ako se sistem sruši posljednjih nekoliko promjena može biti izgubljeno. Zbog toga Redis podržava master-slave replikaciju od prve verzije. S obzirom da Redis drži podatke u memoriji, to je jedna od najbržih baza podataka ovog tipa i pogodan je za aplikacije gdje povremeni manji gubitak podataka nije veliki problem.

- **Instalacija Redis-a**

Redis je prvobitno namijenjen Linux operativnom sistemu, ali postoji tim nazvan MSOpenTech koji radi sa tehnologijama otvorenog koda i prilagođava Microsoft operativnom sistemu. Da bismo preuzeli Redis za Microsoft Windows, moramo otići na stranicu Github-a: <http://github.com/MSOpenTech/redis/> i preuzeti najnoviju verziju. Podržana je i postoji samo 64-bitna verzija. Verzija za Windows je malo drugačija nego za Linux operativni sistem zbog interakcije sa pristupačnim tačkama i načina na koji koristi portove, ali naredbe su iste tako da nema problema sa korištenjem bilo kojeg operativnog sistema. Standardna putanja do Redis-a je C:\Program Files\Redis i u tome direktorijumu se nalaze server i klijent koji se moraju pokrenuti. Postoji konfiguracijska datoteka u kojoj se može mijenjati port, IP adresa koja se sluša, TCP keepalive za održavanje veze i ostale postavke. Pošto se Redis često koristi kao predmemorija, nema smisla da se svi podaci čuvaju trajno, pa se u konfiguraciji mogu podešavati postavke koliko često da se pohranjuje i slično, u zavisnosti od vremena ili od toga koliko je se ključeva promijenilo.



Slika 4.3. Pokretanje servera, vlastiti izvor



Slika 4.4. Pokretanje klijenta

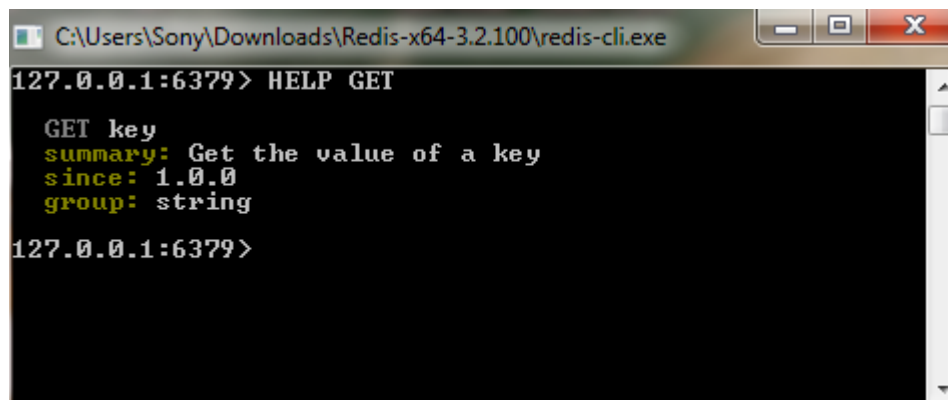
Naredbe u Redis-u

Za pisanje Redis naredbi postoje dvije opcije. Ako se želi samostalno nešto kreirati, potrebno je pokrenuti server i klijent kako bismo mogli to koristiti. Druga opcija je njihov interaktivni vodič na web stranici gdje je podržano pisanje svih naredbi što je dovoljno za vježbu i shvatanje rada Redis-a. Taj vodič nalazi se na sljedećoj web adresi: <http://try.redis.io/>. Pokretanjem toga vodiča, korisnik se

interaktivno vodi kroz najvažnije naredbe uz objašnjenja i primjere. Vodič je vrlo korisna stvar za početnike koji se ranije nisu susretali sa takvom vrstom baza podataka. Naredbe su podijeljene u nekoliko cjelina, koje su srodne, kako bi se jednostavnije objasnile.

- **HELP**

Redis sadrži mnogo naredbi koje su vrlo slične, pa može doći do zamjene ili pogrešne sintakse pisanja naredbi. Zbog toga postoji help, odnosno pomoć koja se može pozvati prilikom pisanja naredbi na način da se upiše „HELP“ te naredba za koju je potrebna sintaksa i pomoć. Primjer se može vidjeti na slici ispod. Na slici se može vidjeti kako izgleda „HELP“ te koje sve stvari sadrži. Vidi se da je prva sintaksa naredbe koja u ovom slučaju glasi: „GET key“, gdje je GET naredba, a „key“ ključ koji je postavljen pomoću naredbe SET i pohranjivanjem vrijednosti. Nakon sintakse dolazi opis naredbe, u ovom slučaju GET vraća vrijednost traženog ključa. Ispod opisa dolazi od koje se verzije Redis-a koristi spomenuta naredba, a nakon toga dolazi u koju grupu naredba spada, u ovom slučaju se vidi da naredba spada u grupu stringova.



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

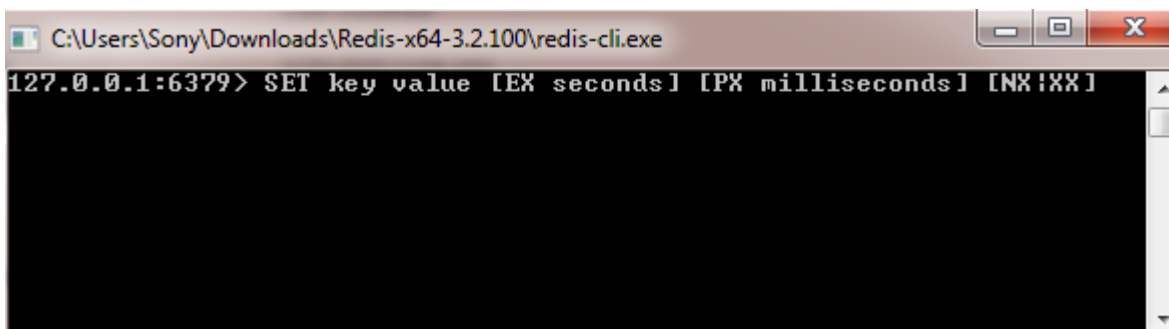
GET key
summary: Get the value of a key
since: 1.0.0
group: string
127.0.0.1:6379>
```

Slika 4.5. Pomoć u Redis-u, vlastiti izvor

- **Povratna poruka**

Nakon svakog izvršavanja naredbe, klijent vrati neku poruku. Kod Redis-a je povratna poruka malo drugačije interpretirana. Ona može biti tipa: 0, 1, OK. Poruka koja će biti vraćena zavisi od naredbe koja se koristi i od toga da li je naredba izvršena ili nije. „OK“ naravno govori da je naredba izvršena i da je sve u redu te se koristi kod bool operacija i naredbi. Ako se radi nešto komplikovanije, vratiće 0 ili 1. Nakon izvršavanja GET naredbe biće vraćena poruka, odnosno vrijednost (ako postoji) koja se nalazi spremljena pod tim ključem. Pisanje naredbi

je mnogo olakšano novijom verzijom Redis-a gdje pri pisanju naredbe, klijent nudi sintaksu te naredbe kao što se može vidjeti na slici ispod:



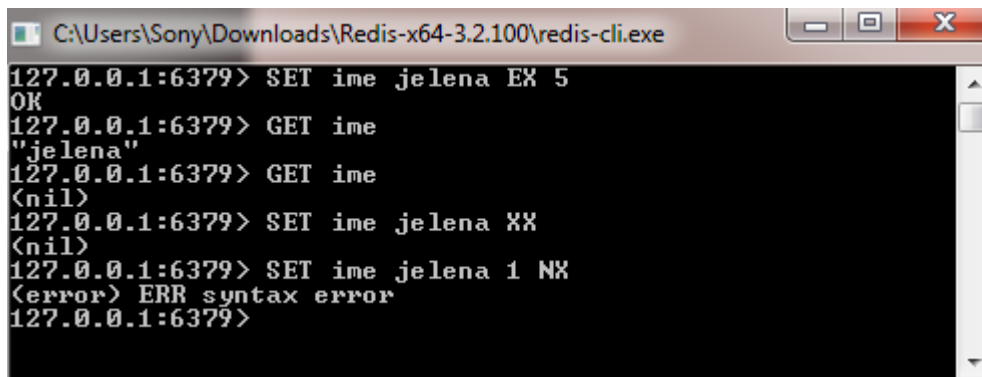
Slika 4.6. Pomoć pri pisanju komandi, vlastiti izvor

• Naredba SET

Sintaksa SET naredbe je SET „ključ“ „vrijednost“. Ta naredba služi za pohranjivanje neke vrijednosti u neki ključ. Vrijednost se pohranjuje kao niz. Ako taj ključ već sadrži neku vrijednost, ta vrijednost će biti zamijenjena novom vrijednošću bez obzira kojeg je tipa. Od verzije 2.6.12. , naredba SET podržava nekoliko dodatnih opcija:

1. EX sekundi – postavlja vrijeme isticanja u sekundama
2. PX milisekundi – postavlja vrijeme isticanja u milisekundama
3. NX – postavlja vrijednost samo ako taj ključ ne sadrži neku vrijednost
4. XX – postavlja vrijednost samo ako taj ključ sadrži neku vrijednost

Pošto ove opcije mogu zamijeniti naredbe SETNX, SETEX, PSETEX, moguće da će te naredbe u nekoj budućnosti biti obustavljene i uklonjene. To je dobra ideja, jer će se smanjiti broj naredbi koje se moraju pamtit i imaju raznovrsne sintakse, koje je teško upamtiti, a ovo su vrlo jednostavne opcije već postojećih naredbi. Isto tako broj riječi i linija se smanjuje ovim dodatnim opcijama naredbi. Na slici ispod vidi se korištenje tih naredbi. U prvoj naredbi postavljeno je isticanje na 5 sekundi, te se nakon toga vrijednost toga ključa briše. Ako navedeni ključ ne postoji, javlja se poruka „nil“, što znači da je taj ključ prazan i da ne sadrži nikakvu 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>
```

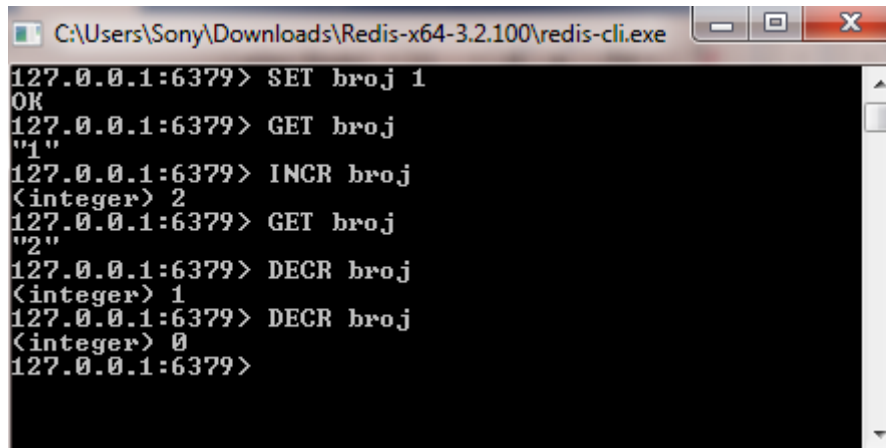
Slika 4.7. Primjer korištenja GET, SET (sa opcijama NX, XX), vlastiti izvor

- **Naredba GET**

Sintaksa naredbe GET je GET „ključ“. GET naredba služi da bi se vratila vrijednost određenog ključa. Ako taj ključ ne sadrži vrijednost, vraća se poruka „nil“. Primjer kako izgleda ta naredba u korištenju može se vidjeti na prethodnoj slici.

- **Naredba INCR**

Sintaksa naredbe INCR je INCR “ključ“. Ova naredba povećava vrijednost broja, koji je spremljen u ključu za jedan. Ako ključ ne postoji, postavlja vrijednost ključa na 0. Ako vrijednost ključa nije broj, tada javlja grešku. Iako je ovo operacija nad brojem, to je i operacija nad nizom pošto naredba SET sprema sve vrijednosti kao niz. Na slici 4.8. može se vidjeti primjer korištenja naredbi INCR i DECR. Takođe se može vidjeti da se nakon izvršavanja naredbe INCR vraća broj uvećan za 1. Zanimljiva stvar je to što iako SET sprema i vraća broj kao niz, svejedno ga prepoznaje kao broj i može ga uvećati ili umanjiti. Primjer korištenja ove naredbe je kod brojanja slanja zahtjeva. Recimo da se ograniči korisnicima da u sekundi mogu poslati maksimalno N zahtjeva kako ne bi došlo do zatrpavanja servera ili slično. Ako korisnik pređe tu granicu, pojavi se greška i korisniku se blokira račun i šalje mail da provjeri da li je računar ili neki drugi uređaj zaražen štetnim softverom kojem je cilj prenatrpati neki od servera. Tu je potrebna i naredba EXPIRE koja bi brisala vrijednost ključa odnosno brojača svake sekunde ili više, zavisno od konfiguracije.



```
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>
```

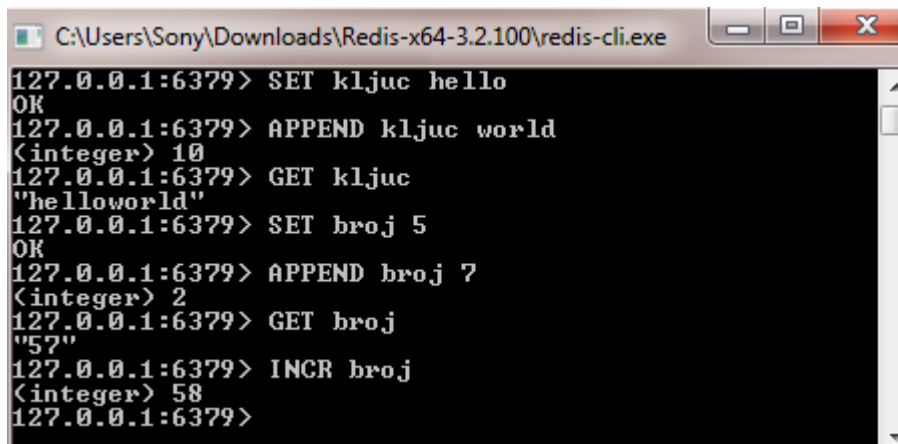
Slika 4.8. Naredbe INCR i DECR, vlastiti izvor

- **Naredba DECR**

Sintaksa naredbe DECR je DECR „ključ“. Ova naredba radi suprotno od INCR, ona smanjuje brojčanu vrijednost nekog ključa. Primjer korištenja DECR naredbe može se vidjeti također na slici 4.8. INCRBY, INCRBYFLOAT, DECRBY i DECRBYFLOAT su varijacije INCR i DECR naredbi koje smanjuju/povećavaju broj za unesenu veličinu.

- **APPEND**

Sintaksa naredbe APPEND je APPEND „ključ“ „vrijednost“. Naredba APPEND dodaje unesenu vrijednost na već postojeću vrijednost, ili kreira novu praznu vrijednost. Nakon izvršavanja ove naredbe, klijent vraća broj znakova te vrijednosti. Ukoliko je vrijednost brojčana i dodamo joj još neku brojčanu vrijednost, brojevi se ne sabiraju već dodaju. Primjer ove naredbe može se vidjeti na slici 4.9. Na slici se takođe vidi da ukoliko se doda broj 7 na broj 5, dobije se broj 57. Ako se nakon toga izvrši naredba INCR (ili DECR) ta vrijednost će se tretirati kao broj pa će se naredba izvršiti ili uvećati za 1.



```
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>
```

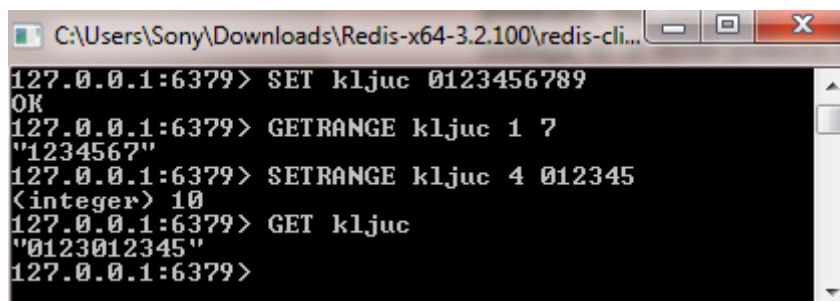
Slika 4.9. Naredba APPEND, vlastiti izvor

- **Naredba GETRANGE**

Sintaksa naredbe GETRANGE je GETRANGE „početak“ „kraj“. Ova naredba vraća znakove u navedenom rasponu što može biti korisno ako je poznato šta je zapisano u tom ključu i potreban je neki određeni raspon. Primjer je JMBG (jedinstveni matični broj građana), poznato je da se on sastoji od nekoliko podataka, tipa datuma rođenja i slično. Ako je potrebno iz JMBG-a očitati datum rođenja, to se može pomoću ove naredbe.

- **Naredba SETRANGE**

Sintaksa naredbe SETRANGE je SETRANGE „ključ“ „odstupanje“ „vrijednost“. Ova naredba zapisuje određenu vrijednost u neki ključ s početkom koji je unesen kao odstupanje. Odnosno, ako je vrijednost 0123456789 kao u primjeru na slici 4.10. i izvrši se naredba „SETRANGE kljuc 4 012345“, vrijednost 012345 zapisat će se u ključ „ključ“ s početkom na četvrtom mjestu i tada će vrijednost zapisana u tom ključu biti 0123012345.



```
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>
```

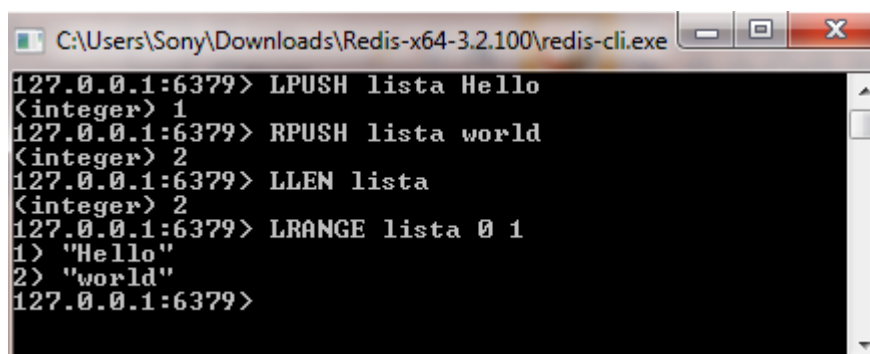
Slika 4.10. Naredbe GETRANGE i SETRANGE, vlastiti izvor

- **Liste**

Redis podržava listu nizova i te liste podržavaju nasumični pristup bilo kojem indeksu. Pomoću lista se mogu napraviti redovi i mogu se umetati u njih vrijednosti, te izbacivati vrijednosti na određenim indeksima. Postoji mnogo naredbi za upravljanje listama u Redis-u koje će biti pojedinačno objašnjene u nastavku:

- **Naredba LPUSH**

Sintaksa naredbe LPUSH je LPUSH „ključ“ „vrijednost,„. Ona dodaje unesenu vrijednost u ključ na početak liste, odnosno na lijevu stranu. Ako navedeni ključ ne postoji, prvo kreira praznu listu i dodaje vrijednost na početak te liste. S verzijom 2.4., moguće je dodati i više vrijednosti odjednom na način da se vrijednosti pišu jedna za drugom. Posljednja vrijednost postavlja se na početak, pretposljednja na desno mjesto uz početnu vrijednost i tako dalje. Nakon izvršavanja te naredbe, vraća se broj elemenata 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 je naredba ista kao i LPUSH, jedino što ona dodaje elemente liste s desne strane, odnosno na kraj navedene liste. Primjer korištenja može se vidjeti na prethodnoj slici.

- **Naredba LLEN**

Sintaksa ove naredbe je LLEN „ključ“. Ova naredba vraća veličinu liste, odnosno broj elemenata koji se nalaze u listi. Primjer korištenja može se takođe vidjeti na slici 4.11.

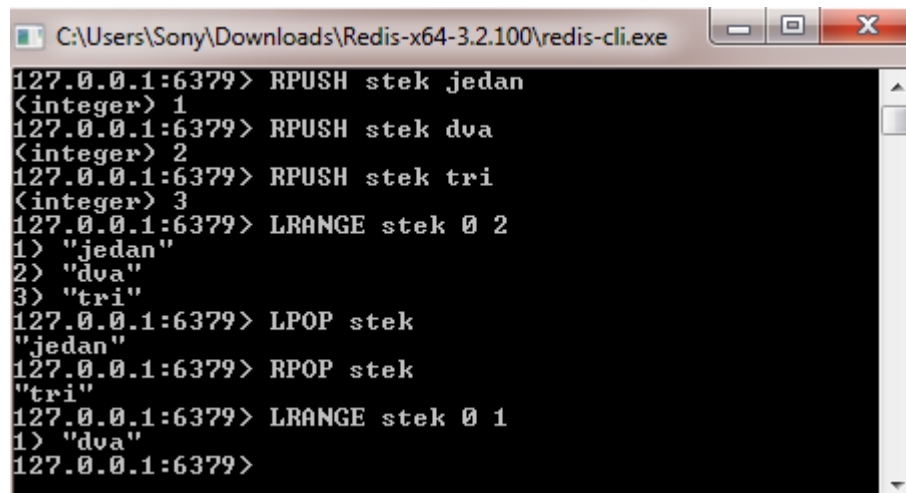
- **Naredba LRANGE**

Sintaksa ove naredbe je LRANGE „ključ“ „početak“ „kraj“. Naredba LRANGE vraća elemente liste koji se nalaze na indeksima unesenim kao

početak i kraj, kao i između njih. Kao što se vidi na slici 4.11., naredba „LRANGE lista 0 1“ vraća elemente liste na nultom i prvom indeksu.

➤ **Naredbe LPOP, RPOP**

Na primjeru ovih komandi može se vidjeti kako se gradi red, odnosno stek. Pomoću ove dvije naredbe izbacuje se i ispisuje vrijednost iz liste s lijeve ili desne strane. Primjer korištenja i načina rada ovih naredbi može se vidjeti na slici 4.12.



```
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“ „vrijednost“. Ova naredba vraća vrijednost na određenom indeksu. Ako se ne zna broj elemenata liste, a želi se dohvatiti posljednji element, može se napisati -1 kao indeks. -2 vraća element prije posljednjeg i tako dalje.

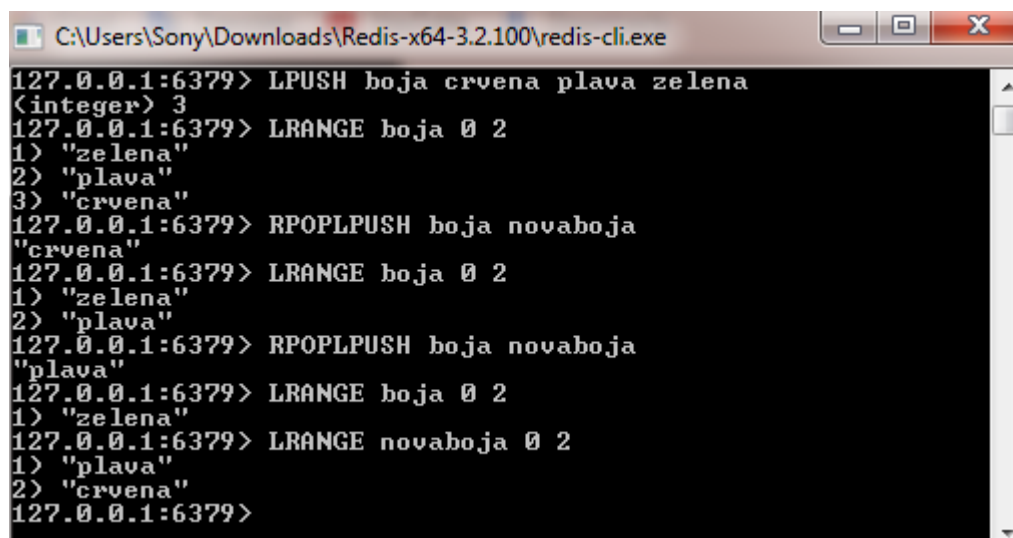
➤ **Naredba LTRIM**

Sintaksa naredbe je LTRIM „ključ“ „početak“ „kraj“. Naredba LTRIM briše elemente liste. Pošto se unose početna i završna vrijednost, odnosno indeks, moguće je brisati jedan element, a moguće je obrisati i cijelu listu. Kod ove naredbe mogu se takođe koristiti negativne vrijednosti kao što je već objašnjeno kod naredbe LINDEX.

➤ **Naredba RPOPLPUSH**

Sintaksa naredbe je RPOPLPUSH „izvor“ „destinacija“. Ovom naredbom uzima se posljednji element iz liste (izvora) i stavlja se u drugu listu (destinaciju) na prvo mjesto, odnosno početak liste. Ako izvor ne postoji, vraćena je vrijednost „nil“. Ako su izvor i destinacija isti, odnosno ako se radi o istoj listi, element će se samo prebaciti s posljednjeg na prvo mjesto pa se to može smatrati naredbom rotiranja elemenata liste. Redis se često

koristi kao server za slanje i primanje poruka te njihovo obrađivanje. Tu se koristi već spominjani stek, jer se poruke stavljaju u listu od strane kreatora kako bi se obradile, te se čeka na njih od strane potrošača. Za to se koristi naredba RPOP, ali ona nije dovoljno pouzdana, jer poruke mogu biti izgubljene ako dođe do nestanka interneta ili električne energije za vrijeme kada je poruka u obradi, a nije još stigla do potrošača. Zato tu dolazi naredba RPOPLPUSH, jer tu potrošač prima poruku i stavlja je u listu za obradu. Nakon što je poruka obrađena, naredbom LREM će se maknuti poruka iz liste za obradu. To je primjer korištenja ove naredbe kao pouzdanog steka. Kako se koristi i koji je rezultat korištenja ove naredbe može se vidjeti na slici 4.13.

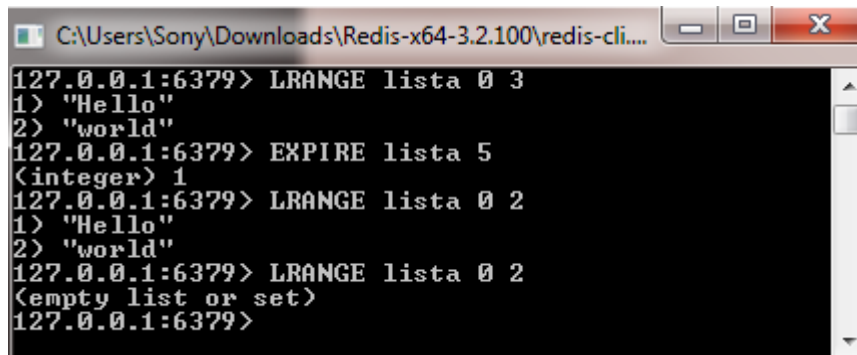


```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe
127.0.0.1:6379> LPUSH boja crvena plava zelena
(integer) 3
127.0.0.1:6379> LRANGE boja 0 2
1) "zelena"
2) "plava"
3) "crvena"
127.0.0.1:6379> RPOPLPUSH boja novaboja
"crvena"
127.0.0.1:6379> LRANGE boja 0 2
1) "zelena"
2) "plava"
127.0.0.1:6379>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

Već je nekoliko puta spomenuto da se Redis često koristi kao stek, ali koristi se i kao predmemorija. Podaci spremljeni u predmemoriju su samo privremeno tamo spremljeni, a za to pomažu naredbe EXPIRE, TTL i PERSIST. Parovi ključ-vrijednost ističu i brišu se nakon nekog određenog vremena. Sintaksa naredbe je EXPIRE „ključ“ „sekundi“. Na ovoj naredbi se temelji uglavnom predmemorija. Pošto su podaci tamo samo privremeno zapisani, pomoću EXPIRE metode se oni brišu nakon nekog određenog vremena. Ako se preimenuje lista ili skup, vrijeme isticanja se nasljeđuje. Primjer korištenja može se vidjeti na slici ispod.

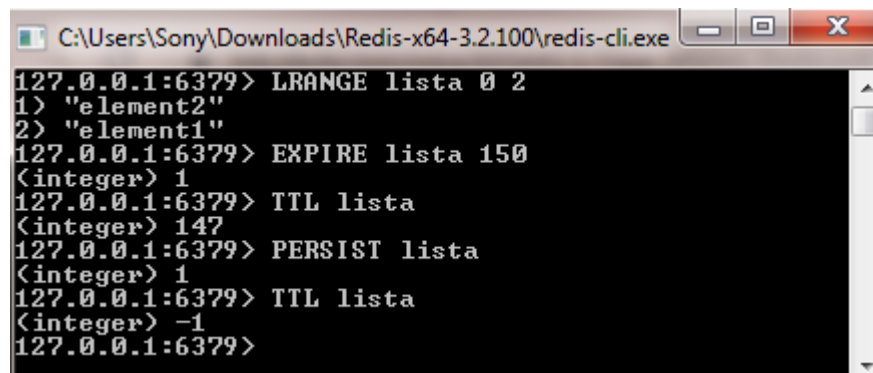


```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe
127.0.0.1:6379> LRANGE lista 0 3
1) "Hello"
2) "world"
127.0.0.1:6379> EXPIRE lista 5
(integer) 1
127.0.0.1:6379> LRANGE lista 0 2
1) "Hello"
2) "world"
127.0.0.1:6379> LRANGE lista 0 2
(empty list or set)
127.0.0.1:6379>
```

Slika 4.14. Naredba EXPIRE, vlastiti izvor

➤ Naredbe TTL i PERSIST

TTL (eng. Time To Live-vrijeme postojanja) naredba vraća vrijeme nakon kojeg će neka lista ili neki ključ biti obrisani. PERSIST naredba uklanja vrijeme isticanja. Na slici se takođe vidi da ukoliko određena lista ili skup nemaju postavljeno vrijeme isticanja, TTL naredba vraća negativnu vrijednost, odnosno -1.



```
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

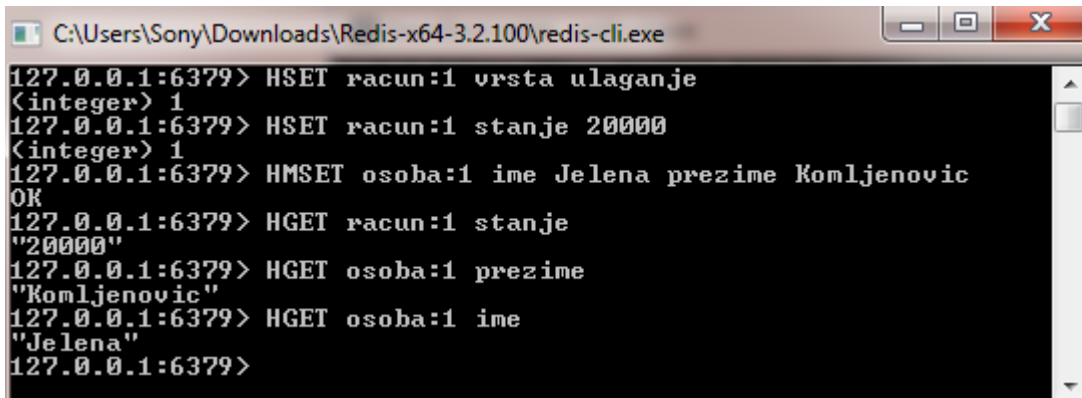
• Sadržaj

Redis ne pohranjuje samo obične tipove podataka, već može pohranjivati već spomenute liste, skupove, ali može pohranjivati i određene sadržaje. Sadržaji su u suštini kolekcije parova ključ-vrijednost. Najkorisniji su kod reprezentacije objekata i tabličnih podataka. Sadržaj može pohraniti mnogo elemenata-parova, preko 4 milijarde, a zauzima jako malo memorijskog prostora što je velika prednost.

➤ Naredba HSET

Sintaksa naredbe je HSET „ključ“ „polje“ „vrijednost“. Naredba HSET postavlja vrijednost polja. Na slici 4.16. se može vidjeti korištenje te

naredbe. Tom naredbom je rečeno: kreiraj objekt račun s ID-em 1 tipa vrsta i vrijednosti ulaganje. Odnosno druga naredba kaže: kreiraj objekt računa s ID-em 1, a stanje tog računa iznosi 20 000. Može se vidjeti i naredba HMSET koja dozvoljava da se unese više vrijednosti odjednom.



```
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>
```

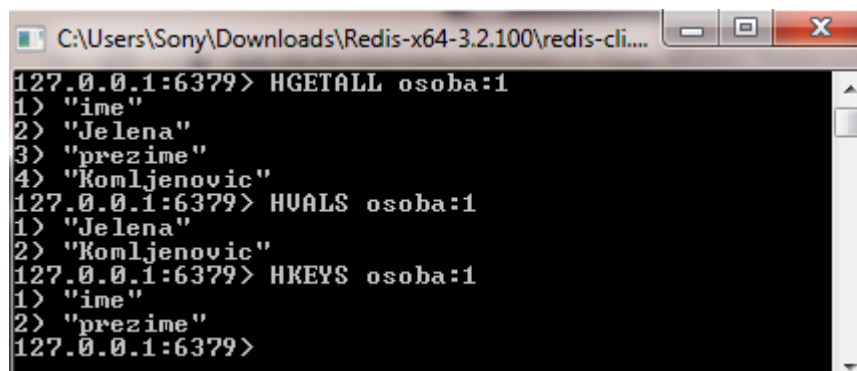
Slika 4.16. Naredbe HSET, HMSET i HGET, vlastiti izvor

➤ Naredba HGET

Sintaksa naredbe je HGET „ključ“ „polje“. Ova naredba vraća vrijednost polja određenog ključa. Kao što se vidi na slici 4.16. naredba „HGET osoba:1 ime“ vraća ime osobe s ID-em 1.

➤ Naredba HGETALL

Sintaksa naredbe je HGETALL „ključ“. HGETALL vraća sve parove ključ-vrijednosti u nekom ključu. Ova naredba vraća vrijednosti kao listu koja je tipa polje pa ispod toga vrijednost. Neće nam se prikazati kao par ključ – vrijednost zato što Redis sve pohranjuje kao niz što je već spomenuto. Ako se želi saznati samo vrijednost ili samo koja su polja, za to postoje naredbe HVALS i HKEYS koje se takođe mogu vidjeti na slici 4.17.



```
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

Naredba HINCRBY uvećava broječanu vrijednost nekog polja. Redis sadržaji podržavaju naredbe iste kao i Redis nizovi koji su navedeni ranije, a to su: HINCRBYFLOAT, HLEN, HEXISTS, HDEL, EXPIRE, TTL i slično. Te naredbe su praktično iste pa se neće dodatno objašnjavati.

• Skupovi

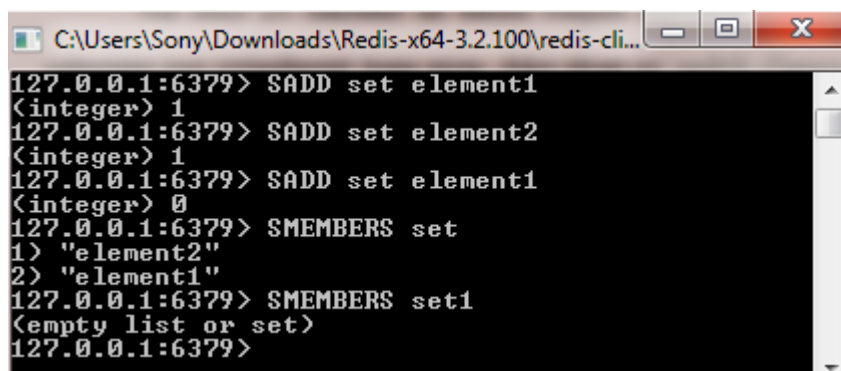
Skupovi (eng. Set) su neuređene kolekcije nizova. Razlika između skupa i niza je ta što skup ne dozvoljava ponavljanje članova. Zato se mogu koristiti kod nekih složenijih stvari, recimo ako se vodi evidencija klijenata u banci. Da ne bi došlo do ponavljanja računa, sve račune stavlja u set što garantuje da se neće ponoviti dva ista računa što nije slučaj ukoliko se koristi običan niz.

➤ Naredba SADD

Sintaksa naredbe je SADD „ključ“ „vrijednost“. Naredba SADD dodaje vrijednost u određeni skup. Ako ta vrijednost postoji već u tom skupu, vraća se poruka 0 odnosno false, jer kao što je već rečeno, skup sadrži samo jedinstvene vrijednosti. Korištenje te naredbe može se vidjeti na slici 4.18.

➤ Naredba SMEMBERS

Na slici 4.18. se takođe može vidjeti naredba SMEMBERS koja vraća sve članove, odnosno sve vrijednosti toga niza. Ako skup ne sadrži članove, javlja se greška da je skup prazan. Isto tako, skupovi podržavaju međusobne unije, razlike i presjeke. Da ponovimo, unija je spoj dva skupa, odnosno u uniji se nalaze svi (jedinstveni) elementi oba skupa. U presjek spadaju svi jedinstveni elementi koji se nalaze i u prvom skupu i u drugom skupu. Razlika vraća elemente koji se nalaze u prvom skupu, a ne nalaze u drugom skupu i obratno.



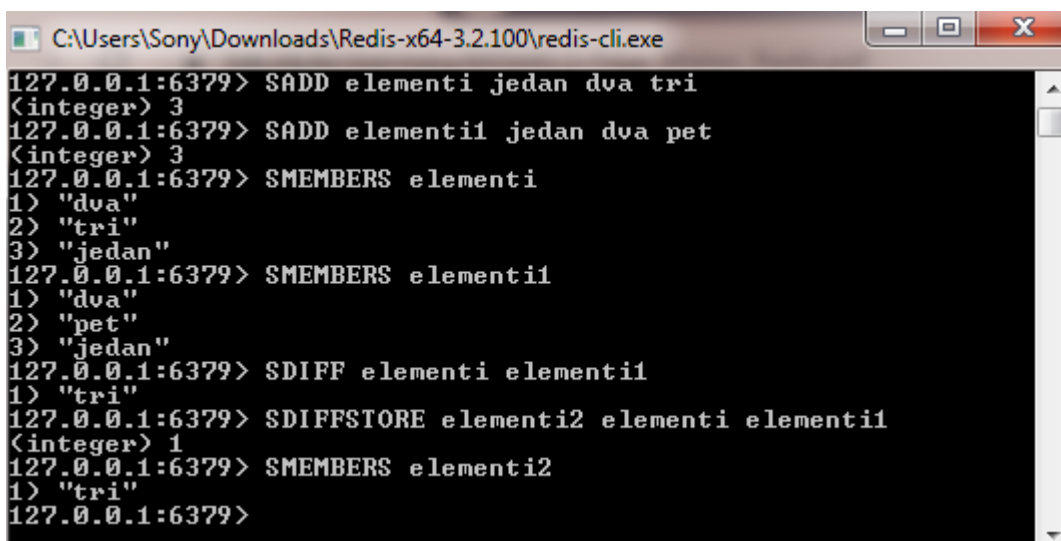
```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli...
127.0.0.1:6379> SADD set element1
(integer) 1
127.0.0.1:6379> SADD set element2
(integer) 1
127.0.0.1:6379> SADD set element1
(integer) 0
127.0.0.1:6379> SMEMBERS set
1) "element2"
2) "element1"
127.0.0.1:6379> SMEMBERS set1
(empty list or set)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

Slika 4.18. Naredbe SADD i SMEMBERS

➤ Naredbe SDIFF i SDIFFSTORE

Naredba SDIFF vraća razliku između dva skupa. Na slici 4.19. se može vidjeti da se u prvom skupu nalaze jedan, dva i tri, a u drugom skupu jedan, dva i pet. Razlika prvog i drugog skupa daje elemente koji se nalaze u prvom skupu, a ne nalaze se u drugom skupu što je u ovom slučaju tri.

SDIFFSTORE radi isto što i SDIFF, pronalazi razliku između dva skupa, ali ova naredba takođe te elemente sprema u treći skup kako bi se kasnije oni mogli obrađivati ili slično, zavisno od aplikacije.

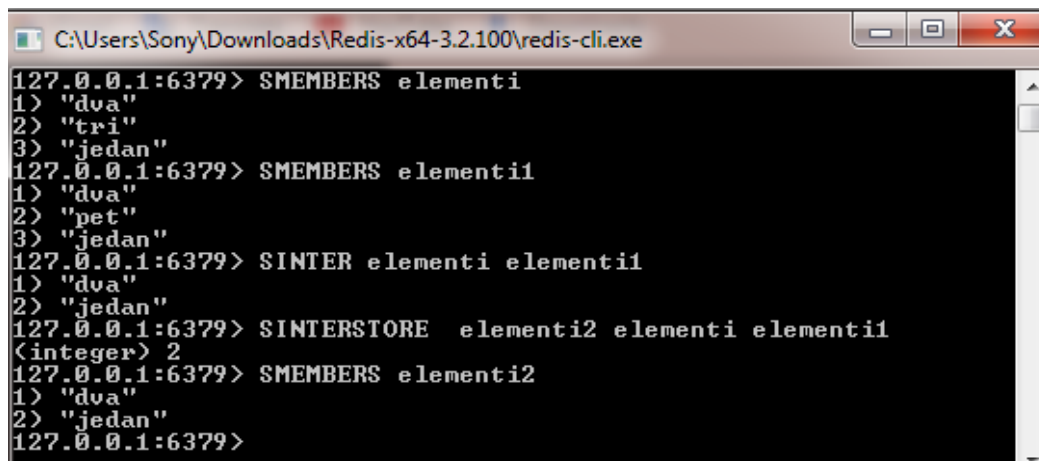


```
C:\Users\Sony\Downloads\Redis-x64-3.2.100\redis-cli.exe
127.0.0.1:6379> SADD elementi jedan dva tri
(integer) 3
127.0.0.1:6379> SADD elementi1 jedan dva pet
(integer) 3
127.0.0.1:6379> SMEMBERS elementi
1) "dva"
2) "tri"
3) "jedan"
127.0.0.1:6379> SMEMBERS elementi1
1) "dva"
2) "pet"
3) "jedan"
127.0.0.1:6379>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

SINTER traži presjek dva skupa i vraća elemente koji se nalaze i u jednom i u drugom skupu. SINTERSTORE radi isto to, samo te elemente pohranjuje u treći skup.



```
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

- **Sortiran skup**

Sortirani skupovi su slični običnim skupovima, samo što je kod sortiranih skupova članovima dodijeljena pozicija. Ovo je jako korisno kod nekih lista s pozicijama u igrama, ali isto tako se u računarskom svijetu koristi i kao lista zadataka s prioritetima. Članovi moraju biti jedinstveni, ali rang ne mora, jer prioriteti mogu biti jednaki. Ova vrsta skupa sadrži algoritam koji odmah sortira članove po rangui pa korisnik ne mora voditi brigu o tome. Tako u primjeru ako se koristi kao lista zadataka s prioritetima, svakom zadatku daje se neki rang, a redis sam sortira. Ako dva ili više elementa imaju isti rang, sortirani su abecedno međusobno. Radi tog algoritma koji je cijelo vrijeme uključen i sortira, dodavanje, brisanje i ažuriranje elemenata je složenosti $O(\log(n))$, što je vrlo brzo. Sortiran skup ima mogućnost vraćanja elemenata po poziciji na kojoj se nalazi ili rangui što će se kasnije pokazati. Isto tako, podržava operacije unije, razlike i presjeka kao i običan skup. Naredbe su vrlo slične kao i kod običnog skupa.

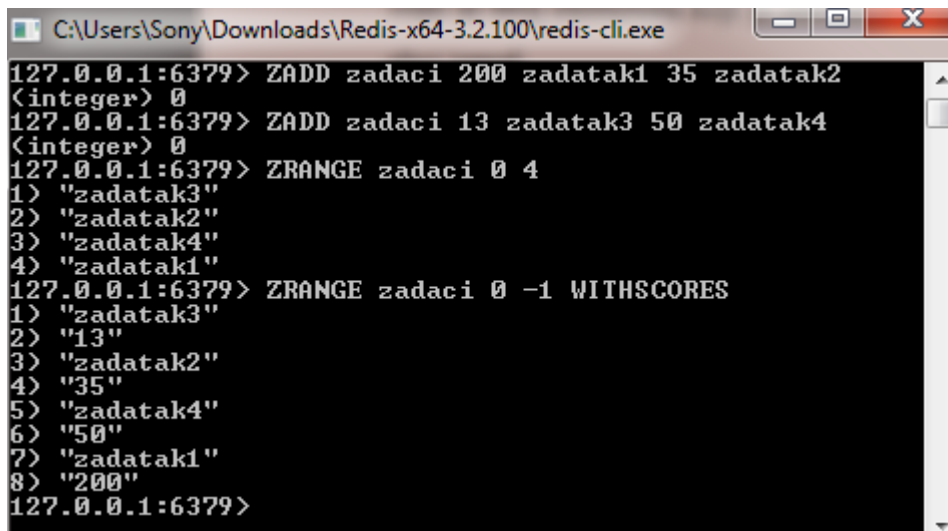
- **Naredba ZADD**

Sintaksa naredbe je ZADD „ključ“ „rang“ „element“. Ova naredba je skoro identična naredbi SADD običnog skupa, samo što se kod ove naredbe mora dodati i rang elementu kako bi on mogao biti sortiran. Nakon izvršavanja ove naredbe, vrati se broj elemenata koji je unesen u skup. Na slici se također može vidjeti da se algoritam sortiranja radi svaki put nakon što se doda, obriše ili ažurira element skupa. Naredba ZRANGE WITHSCORES vraća sve elemente s pripadajućim rangom koji su također sortirani. Postoji još mnogo naredbi koje se vrše nad sortiranim skupovima poput ZCOUNT, ZINCRBY, ZINTERSCORE, ZRANK, ZREM, ZUNIONSTORE koje su praktično identične naredbama nad običnim skupom i koje su već prethodno objašnjene i pokazane. (redis.io/commands)

- **HyperLogLog**

HyperLogLog algoritam dizajniran je da bi prebrojavao jedinstvene elemente. U računarskom svijetu je to vrlo bitno, pogotovo na internetu gdje se to koristi kao mrežna analiza. Na primjeru web stranica, pomoću HyperLogLog algoritma broji se koliko korisnika je posjetilo stranicu, jer svaki korisnik ima jedinstvenu IP adresu. Obično brojanje takvih elemenata zahtjeva veliku memoriju, jer bi se morali spremati svi korisnici koji su posjetili stranicu, ali Redis to radi na drugačiji način. Redis ima algoritme kojima nije potrebno toliko memorije kao ostalim

tehnologijama, već koristi konstantnu memoriju.



```
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

4.1.2 DynamoDB baza podataka

DynamoDB je prva u nizu ključ-vrijednost baza podataka i po uzoru na nju su nastale mnoge druge baze ovog tipa, mada postoje i neke ključ-vrijednost baze podataka koje se dosta razlikuju od nje kao što su Redis i Oracle NoSQL.

DynamoDB je napravljen od strane Amazona sa ciljem da podrži njihovu veliku online prodavnicu. Infrastruktura DynamoDB se sastoji od desetina hiljada servera i mrežnih komponentni lociranih u mnogim centrima podataka širom svijeta.

DynamoDB tabele nemaju fiksnu šemu dok njeni objekti mogu imati proizvoljan broj atributa. Prednost ove baze podataka je što raste paralelno sa podacima, a za poboljšanje performansi je dovoljno dodati nove servere. U DynamoDB, tabela predstavlja kolekciju stavki, a svaka stavka predstavlja kolekciju ključ-vrijednost parova koji predstavljaju attribute. U DynamoDB postoji podrška i za primarne ključeve koji su unutar tabele jedinstveni i služe da jedinstveno odrede stavku.

Primarni ključ je predstavljen putem heša ili putem heša sa opsegom. Heš ključ koristi jedan atribut od koga se kreira heš indeks. Da bi se pronašla stavka sa ovim ključem, neophodno je znati njegovu tačnu vrijednost. Na primjer, ako bi postojala tabela korisnika koja koristi korisnički UserId kao primarni heš ključ, do podataka o korisniku bi bilo moguće doći korištenjem UserId vrijednosti. Da bi se kreirali snažniji indeksi koriste se dva atributa: heš i opseg vrednosti. Pomoću ova dva atributa moguće je obaviti pretragu opsega od neke početne tačke. Na primjer u

tabeli gdje se čuvaju poruke korisnika može se koristiti UserId kao heš ključ, a za opseg vremenska oznaka pa bi na taj način bilo moguće doći do poruka datog korisnika koje su novije od zadate vremenske tačke. Na slici 4.22. dat je prikaz tabele Korisnici u DynamoDB. Tip podataka list u DynamoDB može da čuva uređenu kolekciju vrijednosti različitih tipova.

DynamoDB tabela Korisnici				
UserId	Ime	Prezime	e-mail	Pol
markom	Marko	Marković	marko@domen.com	muški
petarp	Petar	Petrović	petar@domen.com	muški
jovanaj	Jovana	Jovanović		

Slika 4.22. Primjer tabele u DynamoDB, izvor: [7]

4.1.3 Oracle NoSQL baza podataka

U najjednostavnijem obliku, ovakva baza podataka implementira se pomocu heš mape gdje korisnik definiše ključeve koji moraju biti niske karaktera. Aplikacije ne vode računa o usklađivanju nekomaptibilnih verzija jer postoji jedinstven master čvor koji vrši replikaciju i on uvijek ima najnoviju verziju ključeva, dok ostale kopije koje služe samo za čitanje podataka mogu da imaju malo starije verzije. Aplikacija može koristiti i brojeve verzija podataka kako bi obezbijedila konzistenstnost operacija čitanja, pisanja i mijenjanja podataka. U ovakvoj organizaciji ključevi igraju veliku ulogu, jer se na osnovu njih pronalazi lokacija podataka. Ključevi mogu imati glavne i sporedne komponente. Vrijednosti se čuvaju kao nizovi bitova i Oracle NoSQL baze nemaju interne pretpostavke o strukturi podataka smještenim u niz bajtova. Preslikavanje niza bajtova u strukture podataka je prepusteno aplikaciji.



Slika 4.23. Primjer organizacije NoSQL baze, izvor: [6]

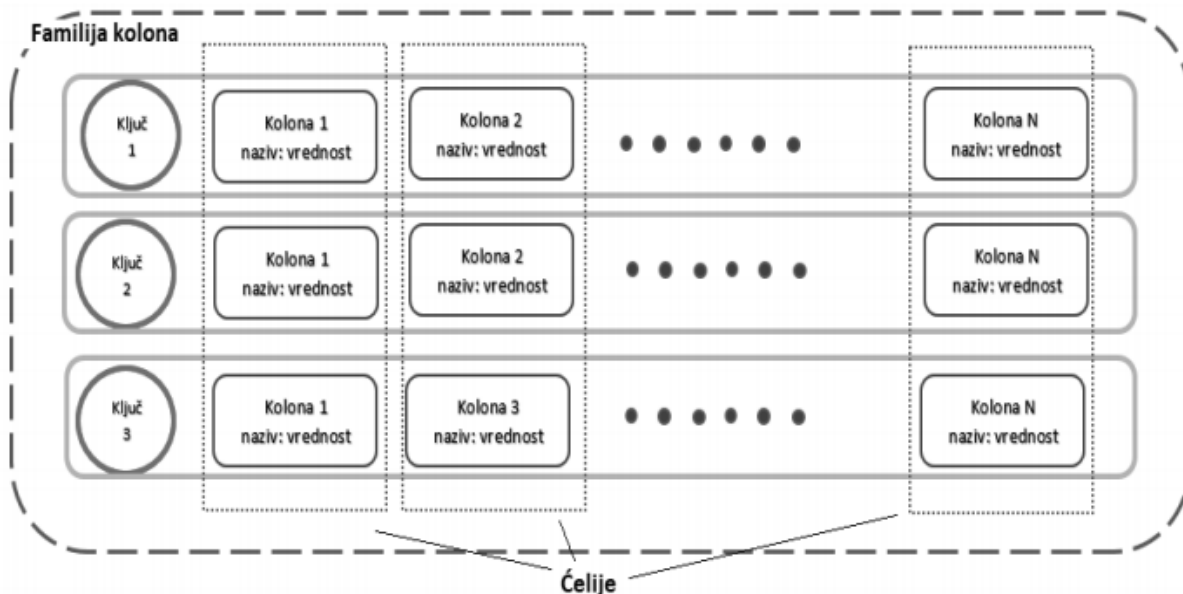
4.1.4 Riak baza podataka

Riak je distribuirana ključ-vrijednost baza podataka gdje vrijednosti mogu biti bilo šta – XML, JSON, slike, zvuk i sl. S obzirom da je Riak napravljen da radi u internetskom okruženju, komande se šalju putem HTTP protokola. Prednosti ove baze podataka su u podršci za sisteme gdje je neophodna stalna raspoloživost, kao što su kupovine preko Interneta (na primjer, Amazon). Riak također podržava nekoliko programskih jezika, a s obzirom da je napisan u Erlangu može se u tom jeziku proširivati po potrebi. Za potrebe kompleksnijih struktura podataka ili ako distribuiranost sistema nije neophodna Riak vjerovatno nije najbolji izbor.

4.2 Model familija kolona

Za razliku od ključ-vrijednost baza podataka, u kolonskim bazama podataka podacima se pristupa po kolonama, a ne po vrijednostima. Obrada po kolonama prilikom paralelnog procesiranja doprinosi boljim performansama. Za čuvanje podataka se koriste kolonske familije, odnosno multidimenzionalne sortirane mape. Kolone imaju ime i pamte veći broj vrijednosti po vrsti koje su identifikovane vremenskom oznakom, tako da predstavljaju trodimenzionalnu strukturu i do svake vrijednosti se može doći korištenjem ključ-vrste, ključ-kolone i vremenske oznake. Te vremenske oznake koriste se za rješavanje konflikata, određivanje isteka valjanosti podataka, za automatsko brisanje i slično. Svrha formiranja familije kolona je da se izvrši grupisanje podataka kojima se često pristupa. Na primjer, u tabeli građana zapisi koji predstavljaju članove iste porodice, ili zaposlene u istoj firmi su u potpunosti izolovani, dok susjedni zapisi gotovo u svim slučajevima nemaju ništa zajedničko. Familija kolona predstavlja NoSQL skladište kojim se

rješava upravo ovaj problem. To znači da u konceptualnom modelu koji je predstavljen na slici ključevi 1, 2 i 3 jesu jednoznačni, ali su redovi koje identifikuju uzajamno povezani. Na primjer, NoSQL može da formira familiju kolona od naučnika koji su objavili više zajedničkih naučnih radova, muzičara koji su imali zajedničke nastupe, osoba koje imaju slična interesovanja i sl. Primjer ovakve baze podataka je Cassandra koja predstavlja distribuiranu bazu podataka i koristi se prilikom obrade velike količine brzorastućih podataka.



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

4.2.1 Apache Cassandra baza podataka

Apache Cassandra je otvorenog koda i to je distribuiran sistem za pohranu podataka te se koristi za upravljanje vrlo velikim količinama podataka raširenih širom svijeta. Ona pruža učinkovite usluge bez ikakve greške. Glavni dijelovi Cassandre su:

- Čvor (eng. Node) – mjesto gdje su podaci spremljeni.
- Središte podataka (eng. Data center) – kolekcija povezanih čvorova.
- Grupa (eng. Cluster) – komponenta koja sadrži jedan ili više središta podataka.
- Dnevnik izvršenja (eng. Commit log) – mehanizam za oporavak od pada sistema, svaka operacija zapisivanja je zapisana u dnevnik izvršenja.

- Memorijska tabela (eng. Mem-table) – tablica u kojoj su smještene strukture podataka. Nakon dnevnika izvršenja, podatak će biti zapisan u memorijsku tabelu.
- SSTablica (eng. SortedStructure Table) –datoteka na disku u koju je podatak prebačen iz memorijske tabele kada njen sadržaj dostigne najveću dopuštenu vrijednost.
- eng. Bloom filter–brz, neodređen algoritam koji provjerava da li je element dio nekog kompleta. On je posebna vrsta predmemorije kojoj se pristupa poslije svakog upita.

Kolekcijski tipovi podataka su:

1. Lista – kolekcija jednog ili više poredanih elemenata.
2. Mapa –kolekcija parova ključ-vrijednost.
3. Set – kolekcija jednog ili više elemenata.

Uz te tipove podataka, 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) je jezik koji je sintaksom sličan SQL-u, i služi za pokretanje različitih operacija u Cassandra bazi podataka. Kod relacionih sistema prva stvar koju moramo uraditi je kreirati bazu podataka, a kod sistema Cassandra to je zapravo keyspace. Za svaki keyspace moramo definisati strategiju replikacije. Neophodno je napomeniti da se naredbe završavaju sa “;”.


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

Ovim smo kreirali keyspace. Kako bismo se prebacili u novokreirani resurs, koristimo sljedeću naredbu:

```
use keyspaceName;
```

Osim kreiranje keyspace-a ukoliko je potrebno možemo i ažurirati keyspace. U sljedećem primjeru koristi se faktor replikacije 1 koji označava da se baza podataka ne replicira, ali u stvarnim sistemima se može definisati kompleksna replikacija.

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

Kada smo kreirali i odabrali „bazu podataka“, možemo kreirati familiju kolona u našem keyspace-u. Kreiramo tabelu korisnici sa kolonama korisnicko_ime koje je ujedno i primarni ključ, zatim lozinku 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;
```

Ukoliko želimo saznati informacije o tabeli možemo koristimo sljedeću komandu:

```
SHOW TABLE korisnici;
```

Vidjeli smo da do sada CQL uveliko podsjeća na SQL jezik. Slična situacija je i prilikom unošenja podataka u keyspace. U tabelu korisnici unosimo redom vrijednosti za korisnicko_ime, lozinku i email:

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

Postavljanje upita je takođe identično načinu koji se koristi kod SQL jezika. Naredba SELECT može označiti koji redovi se dohvataju. Ako želimo dohvatiti sve redove šaljem oznaku * (zvjezdica).

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

```
korisnicko_ime | lozinka | email
```

```
-----
```

```
jelenak | sw56ghz | jelenakom@gmail.com
```

Ako želimo dohvatiti samo određene redove potrebno je navesti imena redova u naredbi SELECT i odvojiti ih zarezom.

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

```
korisnicko_ime
```

```
-----
```

```
jelenak
```

Isto tako korištenjem naredbe WHERE možemo naglasiti šta tačno tražimo. Na primjer želimo pronaći email osobe za koju imamo samo korisnicko_ime:

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

```
email
```

```
-----
```

```
jelenakom@gmail.com
```

U mogućnosti smo koristiti i mnoge druge naredbe koje se koriste i kod SQL jezika, gdje se pored naredbe WHERE može koristiti i naredba ORDER BY, itd..

Ažuriranje podataka se vrši pomoću naredbe UPDATE. Uz to je potrebno navesti naredbu SET kojom se definiše šta se mijenja i koja je nova vrijednost.

U ovom slučaju mijenjamo email adresu novom vrijednošću tamo gdje je korisnicko_ime 'jelenak'.

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

Ukoliko želimo izbrisati neki red, to radimo koristeći naredbu DELETE. Ako naredbi zadamo naziv jednog ili više redova ona će se obrisati samo vrijednost u tim redovima.

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

Ako imena redova nisu definisana, DELETE briše cijeli red.

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

CQL podržava upite kojima možemo dobiti više informacija o keyspace-u, klasteru i ostalim resursima u sistemu. Ukoliko želimo saznati više o „keyspace-u“ koristimo:

```
DESCRIBE KEYSPEC;
```

Takođe, možemo saznati informacije o verziji samog CQL jezika, verziji baze Cassandra i slično. Npr. upit za prikazivanje tačne verzije CQL jezika:

```
cqlsh:keyspacename> SHOW VERSION
```

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

Takođe možemo koristiti tracing opciju u sistemu. Kada omogućimo tracing, sistem će nam ispisivati šta tačno radi, npr. šta radi kada odgovara na neki upit. Naredba kojom možemo uključiti tracing je sljedeća:

```
TRACING on;
```

Sistem naravno podržava indekse. Dodatne indekse možemo kreirati na sljedeći način.

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

Indeks možemo izbrisati na sljedeći način:

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

CQL jezik naravno ima još naredbi i mogućnosti, kao i sam sistem Cassandra. Ovo su bile neke od osnovnih, kako bismo vidjeli kako ovaj sistem funkcioniše.

4.3 Graf baze podataka

Graf baze pohranjuju entitete i njihove međusobne veze. Graf predstavlja jasnu vezu između podataka u bazi. Veze omogućavaju povezivanje pohranjenih podataka te u mnogim slučajevima omogućavaju njihovo dohvaćanje samo jednom operacijom. Čvor u grafu predstavlja entitet (npr. osoba) i svaka veza predstavlja vezu između dva čvora. Svaki čvor je definisan jedinstvenim identifikatorom tj. setom izlaznih i/ili ulaznih veza i setom svojstava koja se nazivaju ključ-vrijednost parovima. Oni su otprilike ekvivalentni zapisu odnosa ili reda u relacionoj bazi podataka ili dokumentu u dokument orijentisanoj bazi podataka. Veze su definisane jedinstvenim identifikatorom tj. početnim i/ili završnim čvorom i setom svojstava tj. one su linije koje povezuju čvorove. One su ključan koncept graf baza podataka predstavljajući apstrakciju koja nije direktno implementirana u drugim sistemima. Svojstva su informacije o čvoru. Na primjer, ako YouTube predstavlja jedan čvor, onda bi on mogao biti povezan sa svojstvom kao što je internetska stranica ili sa riječima koje počinju slovom „Y“.

Graf baze podataka su prikladne za analizu međusobnih povezanosti te zbog toga postoji veliki interes za korištenje graf baza podataka za rudarenje podataka iz društvenih mreža. Takođe su korisne za rad s podacima u poslovnim disciplinama koje uključuju složene odnose i dinamičke šeme, kao što je upravljanje lancem nabavljanja, identifikovanje izvora problema IP telefonije itd. Za razliku od relacionih baza podataka, graf baze podataka pohranjuju veze između zapisa. Umjesto e-mail adrese koja se pronalazi preko primarnog ključa korisnika u relacionim bazama, u graf bazama korisnički zapis ima pokazivač na zapis e-mail adrese. To znači da, kada je korisnik odabran, pokazivač pokazuje na njegovu e-mail adresu te nema potrebe tražiti tabelu e-mail adresa kako bi se pronašao traženi zapis. Takva pretraga može olakšati i ubrzati pretraživanje podataka. Za razliku od većine ostalih NoSQL sustava, graf baze poštuju principe ACID modela transakcija. Kod tih transakcija, najvažnije je ne dopustiti poveznice koje imaju samo jedan čvor. Početni i završni čvor uvijek moraju postojati, a čvor se može

obrisati tek kada su obrisane sve njegove veze. Zbog važnosti poveznica između čvorova, graf baze moraju držati blisko povezane čvorove zajedno. Prednost ovakvog pristupa je gore spomenuta efikasnost upita, ali ako je želimo održati, moramo se odreći fragmentacije. Naime, ukoliko su fragmenti raspoređeni na različite klastere, koji su potencijalno na različitim lokacijama, morali bismo osigurati da veze ne prelaze iz jednog klastera u drugi, kako bi se brzo mogao dohvatiti cijeli graf. Ako bismo i uspjeli u tom u nekom trenutku, takva situacija bila bi brzo narušena dodavanjem novih veza. Zato se kod ovakvih baza najčešće koristi samo jedan računar. Ukoliko je distribucija nužna, jer jedan računar ne može efikasno rješavati probleme, koristi se master–slave replikacija koja je prije opisana. Razlika u odnosu na klasičan model master–slave je što je ovdje slave čvorovima dopušteno i pisanje, ali oni su to dužni odmah prijaviti master čvoru. Master čvor zatim odlučuje hoće li to prihvatiti i ako to prihvati proslijeđuje novu vrijednost svim drugim slave čvorovima. Slave čvor ne smije prihvatiti novu vrijednost ni od koga drugog osim od master čvora. Budući da se ovaj proces odvija u transakciji, osigurana je visoka konzistentnost. Ali, budući da su dozvoljena čitanja s čvorova i za vrijeme trajanja transakcije, takođe imamo i visoku dostupnost. Drugi način za postizanje efikasne distribucije je distribuirati sistem na aplikacijskom nivou. Ovakva distribucija ostvarljiva je ukoliko bazu mozemo logički podijeliti na cjeline koje nemaju potrebu međusobno komunicirati. Takav slučaj može se javiti ako na primjer imamo aplikaciju distribuiranu u različitim državama i podatke možemo odvojiti po državama. Samo neke od prednosti distribucije jedne baze, u odnosu na izradu posebne baze za svaku državu su lakše održavanje i generisanje izvještaja na globalnom nivou. Ipak, ovakva distribucija je složenija opcija od master–slave, jer u ovom slučaju arhitekta aplikacije mora sam osigurati da se svaki zahtjev šalje na odgovarajući klaster. Obrada prirodnog jezika i danas je veliki izazov za računare. Srećom, graf baze su korisne za povezivanje i pretraživanje velike količine tekstualnih dokumenata. Korištenjem procesa obrade prirodnog jezika, takozvanog prepoznavanja entiteta, servis može skeniranjem datih dokumenata relativno brzo identifikovati ključne pojmove u dokumentu. Zatim se na temelju rezultata, mogu izgraditi poveznice među dokumentima u kojima se nalaze isti pojmovi. Nad ovakvom strukturom spremljenom u graf bazi lako ćemo izvesti i komplikovanije pretrage. Prepoznavanje entiteta, uz pomoć dodatnih alata, možemo iskoristiti i za izvlačenje cijelih tvrdnji iz datih dokumenata. Tako detektovane tvrdnje, pogodno je spremati u graf baze, koje, zbog dopuštenih svojstava poveznica, mogu pružiti i kontekst tvrdnjama te tako olakšati aplikaciji razumijevanje novih dokumenata. Jedna od

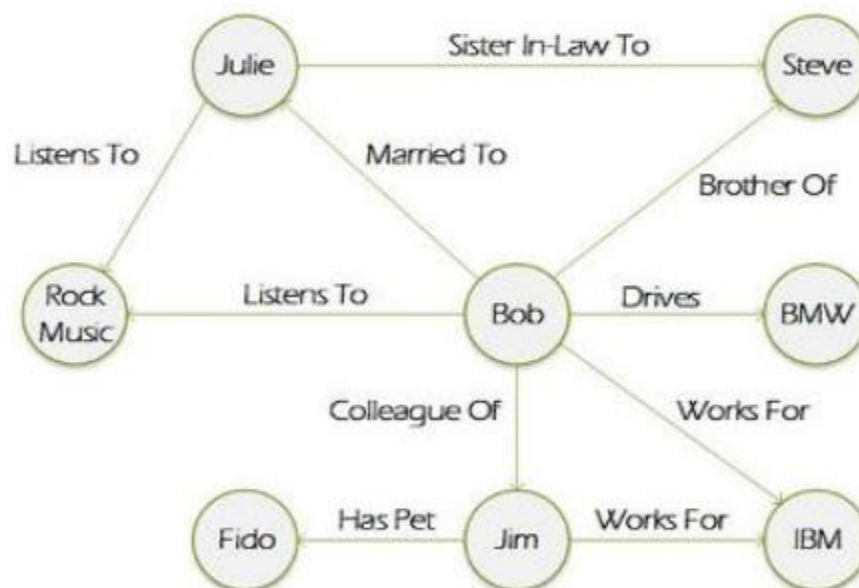
najpoznatijih graf baza podataka je Neo4j, čiji će način rada biti prikazan i objašnjen u nastavku.

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

1. Kod aplikacija za navigaciju koje služe za pronalazak najkraće rute
2. Kod društvenih mreža (facebook, twitter)
3. Za određivanje najdjelotvornijeg puta za usmjeravanje prometa preko mreže podataka
4. U pronalasku organizovanog kriminala

Graf baze podataka se takođe koriste kako bi se sugerisali određeni proizvodi kupcima na osnovu sličnih proizvoda koje su kupili drugi ljudi.

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



Slika 4.25. Primjer grafa, izvor: [3]

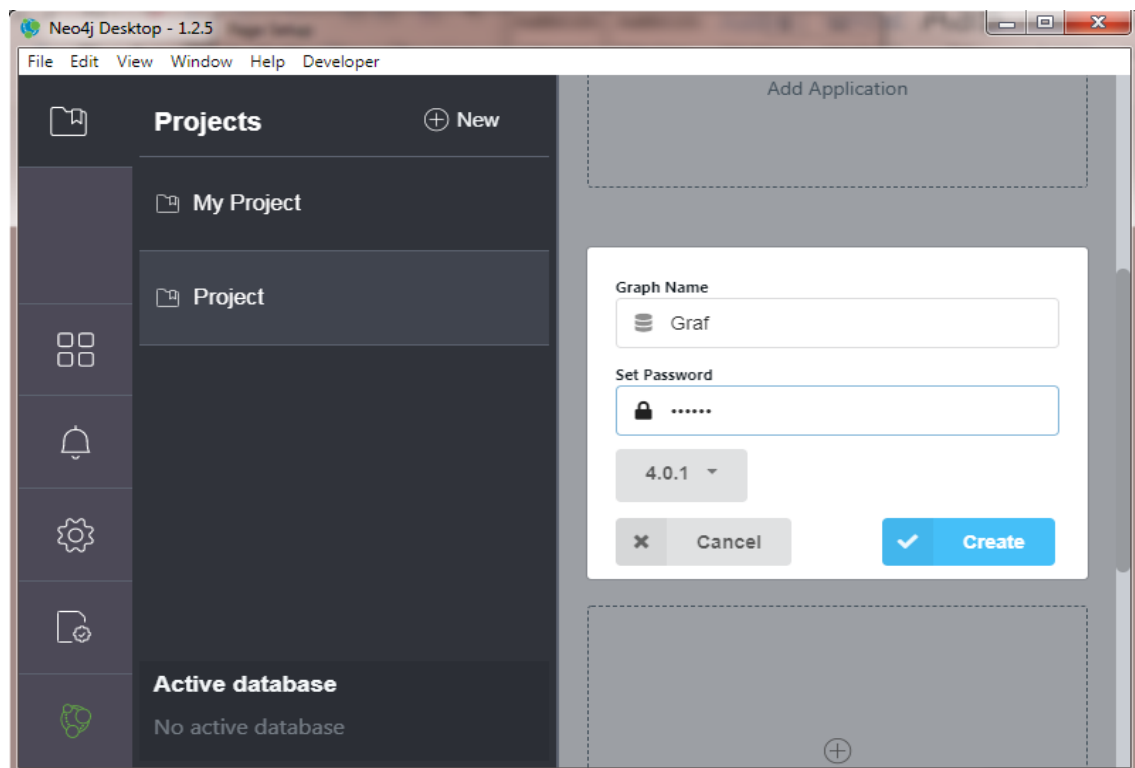
4.3.1 Neo4j baza podataka

Neo4j je najpoznatija open-source graf baza podataka razvijena od strane Neo4j, Inc. Ona omogućava spremanje čvorova i veza između njih te njihovo obilježavanje. Neo4j je razvijen u Java programskom jeziku i omogućava korištenje Cypher Query Language upitnog jezika za sofisticirane upite. Prilikom dobijanja čestih upita za iste podatke, poželjno je spremiti te podatke u cache memoriju, jer je to brže nego stalno dohvatati podatke s diska.

Neo4j omogućava dva načina keširanja podataka:

- File buffer cache: Spremanje podataka na disk u istom, efikasno zbijenom formatu.
- Object cache: Pohranjivanje čvorova, veza i njihovih svojstava u format za efikasno prebacivanje u memoriju.

Sada će biti ukratko objašnjena izrada grafa te jednostavni upiti nad podacima u graf bazi podataka. Prvi korak je kreiranje novog projekta odabirom na **New** u opciji **Projects**. Nakon toga se odabere **New Graph** unutar novog projekta. Potom se odabere **Create a Local Graph**. Zatim se proizvoljno odabere ime za naziv baze podataka te se proizvoljno postavi lozinka za novu bazu podataka. Na kraju se odabere **Create**.



Slika 4.26. Kreiranje baze podataka, vlastiti izvor

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

- Kreiranje čvorova se izvršava pomoću naredbe CREATE

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

Slika 4.27. Kreiranje čvora, vlastiti izvor

Ovaj upit kreira novi čvor označen kao *prvi* tipa *Osoba* i sadrži podatak sa ključem *ime* čija je vrednost „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"
}
```

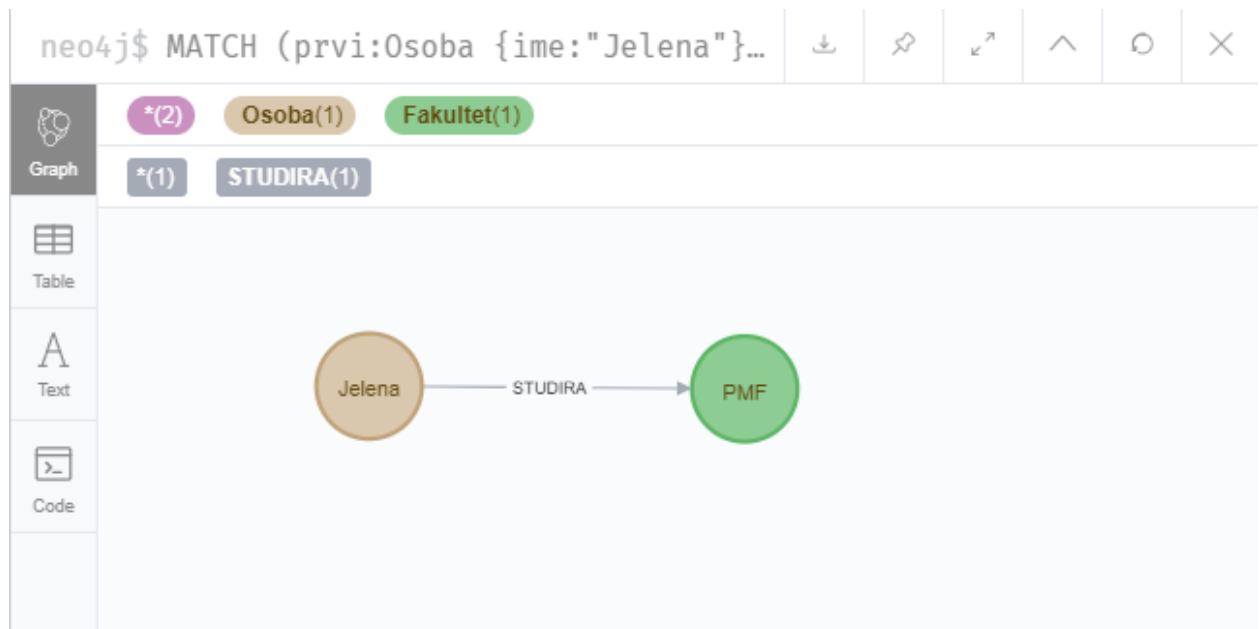
Slika 4.29. Tabelarni prikaz ključa i vrijednosti, vlastiti izvor

➤ Kreiranje relacija

```
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 relacije, vlastiti izvor

Naredbom *MATCH* pronalazimo, odnosno dohvatamo podatak tipa *Osoba* kome se vrijednost *ključa* poklapa sa zadatom vrijednosti „*Jelena*“ i dodjeljuje mu oznaku *prvi* preko koje će se referencirati na taj podatak. Naredbom *CREATE* kreira se relacija *STUDIRA* ka drugom podatku tipa *FAKULTET* sa podatkom koji za ključ *ime* ima vrijednost „*PMF*“. Grafički prikaz izgleda ovako:



Slika 4.31. Grafički prikaz kreirane relacije, vlastiti izvor

Ova komanda kreira nove podatke tipa *Osoba* sa imenima navedenim u *FOREACH* dijelu i prema svima kreira vezu tipa *PRIJATELJ*.

```

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

```

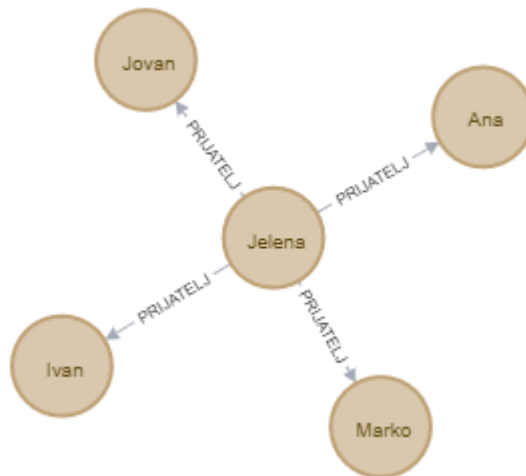
1 MATCH (prvi {ime:"Jelena"})-[:PRIJATELJ]→
  (prijatelji)
2 RETURN prvi,prijatelji

```

Slika 4.33. Pretraživanje prijatelja, vlastiti izvor

Ovaj upit kao rezultat vraća sve podatke prema kojima postoji veza tipa *PRIJATELJ* od podatka kome za ključ *ime* odgovara zadata 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

Ovim smo stvorili vezu tipa *PRIJATELJ* između vrijednosti Ana i Dragana i za vrijednost Dragana kreirali relaciju *STUDIRA_NA* ka podatku koji za ključ ima vrijednost “*pmf*”.

➤ Nalaženje najkraćeg puta

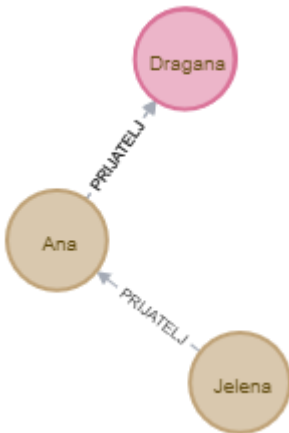
```
1 MATCH (prvi {ime:"Jelena"})
2 MATCH (expert)-[:STUDIRA_NA]→(fax:Fakultet
3 {ime:"PMF"})
4 MATCH path = shortestPath( (prvi)-[:PRIJATELJ*..5]- (expert) )
5 RETURN prvi,expert,path
```

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

Ovaj upit prolazi rekurzivno kroz sve naše prijatelje maksimalno do dužine 5 zaključno sa osobom koja je *expert*.

Kao što možemo videti **Neo4j** ima ugrađenu funkciju *shortestPath* koja u pozadini koristi *Dijkstra algoritam* za pronalaženje najkraćeg puta između čvorova. U poređenju sa relacionim bazama podataka, ovakva pretraga kod graf baza podataka traje neuporedivo kraće.

Grafički prikaz:



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

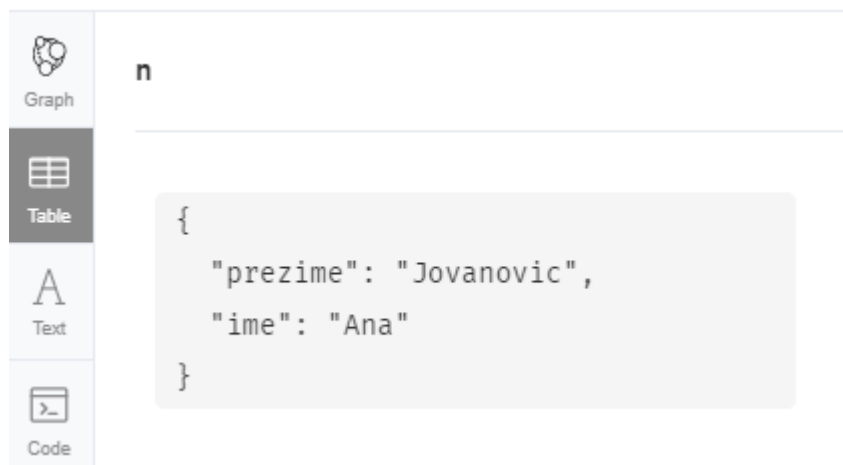
➤ Izmjena i dodavanje atributa

```
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 prethodno dodati 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

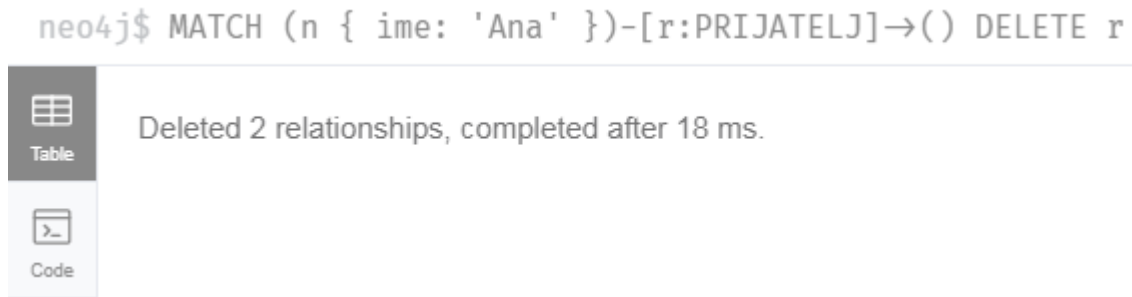
➤ Brisanje relacija

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

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

Slika 4.41. Brisanje veza određenog čvora, vlastiti izvor

Rezultat ovog upita je:



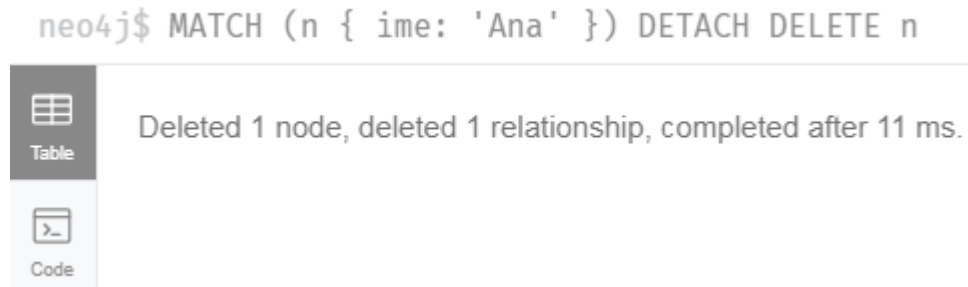
Slika 4.42. Rezultat brisanja svih veza jednog čvora, vlastiti izvor

➤ Brisanje čvora i svih njegovih relacija :

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

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

Rezultat upita:



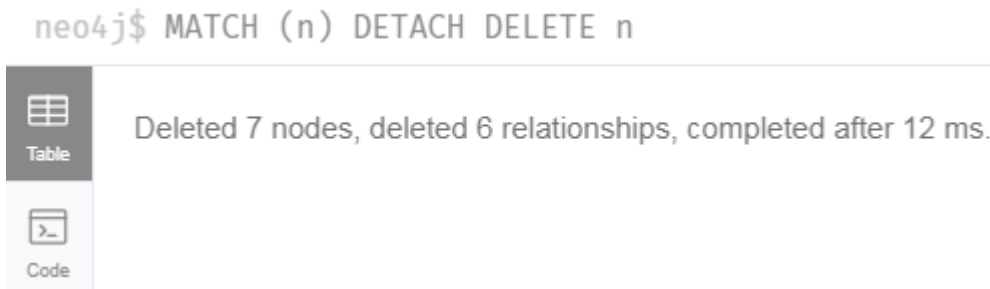
Slika 4.44. Rezultat brisanja čvora i njegovih relacija, vlastiti izvor

- 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 najfleksibilniji model NoSQL baza podataka, jer omogućavaju da se u istu bazu podataka skladište podaci koji se uzajamno razlikuju kako u veličini tako i u struktuiranosti. Drugim riječima, skladišta dokumenata nemaju nikakvo znanje o sadržaju koji se skladišti niti koriste bilo kakvu šemu za skladištenje. Dokument orijentisane baze podataka koriste dokumente kao strukturu za pohranu i izvršavanje upita. Dokument se može odnositi na Microsoft Word ili PDF dokument, ali je najčešće to XML (engl. Extensible Markup Language) i JSON (engl. JavaScript Object Notation) . Umjesto kolona sa redovima i tipom podataka koji se koriste u relacionim bazama, dokumenti sadrže opis tipa podatka i vrijednost za taj podatak. Prilikom dodavanja novog tipa podataka u dokument orijentisanu bazu nije potrebno mijenjati cijelu šemu baze podataka kao kod relacionih baza, nego se podaci mogu jednostavno dodati dodavanjem objekata u bazu podataka. Svi objekti, pa čak i oni iz iste klase, mogu se potpuno međusobno razlikovati. Pohranjivanje dokumenata omogućava pohranu različitih tipova dokumenata, omogućava opcionalna polja unutar dokumenta te dopušta kodiranje pomoću različitih sistema za kodiranje. Na primjer, jedan dokument može biti kodiran u JSON formatu, dok drugi dokument može biti XML.

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>
```

Dva prethodna dokumenta imaju neke zajedničke strukturne, ali takođe imaju i jedinstvene elemente. Za razliku od relacionih baza gdje svaki zapis sadrži ista polja i gdje neiskorištena polja ostaju prazna, kod dokument orijentisanih baza nema praznih polja. Ovakav pristup dozvoljava dodavanje novih informacija bez zahtijevanja da svaki zapis u bazi ima istu strukturu. Da bi klijent znao šta je spremljeno u kojem dokumentu, uvode se kolekcije. Za svaku kolekciju

definišemo tip dokumenata koje čuva, ali samo na logičkom nivou, nevezano za njihovu strukturu. Na primjer, jedna kolekcija može sadržati knjige, a druga studente. Iako šema podataka u prethodna dva dokumenta nije jednaka, dokumenti i dalje mogu pripadati istoj kolekciji za razliku od relacionih baza podataka gdje svaki red u tabeli mora imati istu šemu. Dokument baze u opštem slučaju su napravljene s ciljem distribucije podataka na više servera. Sistemi sami vode brigu o raspodjeli podataka i povezivanju s drugim serverima. 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. Ova odluka može se temeljiti na lokaciji čvora ili količini RAM-a koju posjeduje. Jedna od najpoznatijih dokument baza podataka je MongoDB, koja će biti detaljno objašnjena u nastavku.

4.4.1 MongoDB baza podataka

Mongo predstavlja sredinu između velikih mogućnosti upita relacionih baza i distribuirane prirode servisa za čuvanje podataka. *Mongo* je JSON dokument baza, mada su tehnički gledano podaci smješteni u binarnu formu JSON-a, poznatiju kao BSON. *Mongo* dokument može biti posmatran kao red u tabeli relacione baze bez šeme, čije vrijednosti mogu biti ugnježdene do proizvoljne dubine. *mongo* osim osnovnih operacija dodavanja, izmjenjivanja, brisanja i dohvaćanja nudi i neke jedinstvene operacije koje većina drugih baza podataka nema ili nisu na nivou koji *mongo* pruža.

Sekundarni indeksi omogućavaju brže izmjenjivanje, dohvaćanje i brisanje podataka, a indeksi koje *mongo* podržava su jedinstveni, kombinovani, tekstualni i geoprostorni indeksi. Sakupljanje je još jedna jedinstvena operacija *mongo* baze. Ovo svojstvo omogućava složena sakupljanja podataka i njihovo oblikovanje pomoću cjevovoda sakupljanja (engl. aggregation pipeline) iz jednostavnih skupova podataka u kompleksnije skupove. Specijalne kolekcije su još jedno od bitnih svojstava. Specijalne kolekcije nazivaju se ograničene kolekcije (engl. capped collections). Ove kolekcije su kolekcije ranije određenih veličina, kreiraju se eksplicitno i služe za brisanje starijih podataka, koji se brišu automatski kada kolekcija pređe neku ranije definiranu veličinu u fizičkoj memoriji računara ili kada se pređe maksimalan broj dokumenata po kolekciji koji isto mora biti ranije

definisan. Mongo sam po sebi teži ostvarivanju mnogih mogućnosti SQL baza podataka na efikasniji način.

Sljedeći pojmovi su bitni za razumijevanje MongoDB sistema:

- **_id** - Ovo je obavezno polje za svaki dokument u MongoDB bazi. Predstavlja jedinstvenu vrijednost po kojoj razlikujemo dokumente u bazi. Pošto je polje obavezno, ukoliko pokušamo da napravimo novi dokument bez njega, biće automatski dodato.
- **Kolekcija** - Predstavlja grupisane dokumente. Kolekcija postoji unutar jedne baze. Kolekcije nemaju definisanu strukturu, svaki dokument može biti različit.
- **Kursor** - Pokazivač na rezultujući skup našeg upita. Klijenti mogu iterirati kroz ovaj skup kako bi dobili rezultate.
- **Baza podataka** - Skladište za kolekcije. Svaka baza ima svoj skup datoteka.
- **Dokument** - Jedan zapis u kolekciji. Sastoji se od naziva polja i vrijednosti.
- **Polje** - Par (*ime, vrijednost*) jednog dokumenta. Dokument može imati 0 ili više polja.
- **JSON** - Notacija za predstavljanje strukturiranih podataka u čitljivom formatu.

Instalacija MongoDB sistema

Kako bi rad sa MongoDB bio moguć potrebno je preuzeti instalaciju [14], odabrati odgovarajuću verziju, odgovarajući operativni sistem i paket.

Korišćenjem MongoDB shell programa možemo se povezati sa bazom i izvršavati različite upite nad kolekcijama koje sadrži. Potrebno je pokrenuti shell skript koji dolazi uz mongo server.

MongoDB shell program ne sadrži operaciju za kreiranje nove baze podataka.

Umesto toga, baza se automatski kreira kada se u nju trajno zapiše prvi dokument.

Tako, na primjer, ukoliko MongoDB SUBP nije imao bazu podataka mojabaza, nakon izvršavanja naredne dve naredbe, biće kreirana nova baza

podataka mojabaza, koja će sadržati jednu kolekciju novakolekcija, koja će sadržati jedan dokument:

```

> 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

Također, vrijedno je napomenuti da su opisane naredbe primjenljive **samo u MongoDB shell programu**, a ne u drugim radnim okvirima, drajverima i sl. U prethodnoj sekciji smo pomenuli da su dokumenta u kojima se podaci čuvaju JSON (zapravo BSON), što znači da smo dodali novi dokument u JSON formatu, gdje vitičaste zagrade {...} označavaju objekat sa ključevima i vrijednostima, a uglaste zagrade [...] označavaju niz.

Komanda *show collections* služi za pregled kolekcija, a pomoću nje možemo vidjeti da kolekcije sada postoje:

```

> show collections
studenti
>

```

Slika 4.48. Prikaz komande show collections, vlastiti izvor

Komanda **find()** se koristi za prikaz sadržaja kolekcije. Dodatak **pretty()** u primjerima je funkcija koja formatira ispis objekata, tako da svaki par ključa i vrijednosti vraća u posebnom redu konzole.

```

> 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

Komanda `count()` broji sve objekte u kolekciji `studenti`:

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

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

Do sada smo koristili funkciju **`find()`** bez parametara kako bismo prikazali sve dokumente. Da bismo pristupili tačno određenom dokumentu potrebno je da funkciji prosledimo `_id` dokumenta tipa **`ObjectId`**. Napomenuli smo da se `_id` dodaje automatski ukoliko 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

Funkcija **`find()`** takođe prihvata opcioni drugi parametar, objekat polja kojim možemo definisati koja polja želimo da budu prikazana. Ukoliko ne želimo da u rezultatu dobijemo sva polja za dokumente, možemo iskoristiti *projekciju* (engl. *projection*) rezultata. Polja u projekciji predstavljaju polja koja se nalaze u dokumentu, a vrednosti u projekciji mogu biti:

- 0 ili `false`: polje neće biti obuhvaćeno u rezultatu
- 1 ili `true`: polje će biti obuhvaćeno u rezultatu

Podrazumijevano, ukoliko ne navedemo projekciju, sva polja iz dokumenta će biti dohvaćena. Ukoliko ipak navedemo projekciju, tada će biti dohvaćena samo ona polja koja su eksplicitno navedena da budu uključena (vrednost 1 ili `true` u projekciji), dok će ostala polja biti isključena iz rezultata. Specijalno, polje `_id` će se uvijek naći u rezultatu, osim ako eksplicitno ne navedemo `_id: 0` (ili `id_: false`) u objektu projekcije.

Ukoliko želimo da pored `_id`-a vidimo samo naziv, potrebno je kao drugi parametar da prosledimo **`ime`** sa vrijednošću 1 ili `true`:

```
> 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

Da bismo prikazali sve osim imena, umesto 1 postavimo 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

Nekada će nam biti potrebno da pronađemo dokumente sa vrijednostima koje su manje ili veće od zadate, koje su u nekom intervalu, itd. Definisana su posebna svojstva koja možemo pisati u upitu, a koja predstavljaju sljedeća ograničenja:

- \$gt - pronalazi vrijednosti koje su veće od zadate
- \$gte - pronalazi vrijednosti koje su veće ili jednake zadatoj
- \$lt - pronalazi vrijednosti koje su manje od zadate
- \$lte - pronalazi vrijednosti koje su manje ili jednake zadatoj
- \$ne - pronalazi vrijednosti koje nisu jednake zadatoj
- \$eq - pronalazi vrijednosti koje su jednake zadatoj
- \$in - pronalazi vrijednosti koje su jednake nekoj iz zadatog niza vrijednosti
- \$nin - pronalazi vrijednosti koje nisu jednake nijednoj iz zadatog 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

Pored toga, možemo koristiti i svojstva koja imaju ulogu logičkih operatora:

- \$and - pronalazi sve dokumente koji su ispunili uslove oba upita
- \$or - pronalazi sve dokumente koji su ispunili uslove bar jednog od upita
- \$not - pronalazi sve dokumente koji nisu ispunili uslove upita
- \$nor - pronalazi sve 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

U prethodnim upitima vrijednosti polja su poređenje sa niskama onakve kakve su zadate. Nekada je potrebno provjeriti da li vrijednost polja počinje ili završava nekom niskom, ili da li sadrži neku nisku.

- Ukoliko želimo da vrijednost nekog polja počinje nekom niskom, onda tu nisku navodimo između /^ i /.
- Ukoliko želimo da vrijednost nekog polja završava nekom niskom, onda tu nisku navodimo između / i \$/.
- Ukoliko želimo da vrijednost nekog polja sadrži neku nisku, onda tu nisku navodimo između / i /.

```
> 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 zatom niskom, vlastiti izvor

Osim dodavanja i pretraživanja, dokumente, naravno, možemo i ažurirati. MongoDB ima dvije osnovne metode ažuriranja. Prva je update() kojoj prosljeđujemo 3 objekta. U prvom zadajemo uslov za dokumente koje zelimo ažurirati. U drugom dokumentu navode se svi atributi i nove vrijednosti koje zelimo ažurirati. Ako ne naglasimo drugačije, baza će ažurirati samo prvi

dokument koji odgovara zadanom kriteriju. Ukoliko želimo promijeniti to ponasanje, u trećem dokumentu možemo proslijediti ključ multi sa vrijednošću true. Sledećim 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

Takođe, ova metoda dozvoljava i neke dodatne opcije, osim postavljanja vrijednosti. Možemo preimenovati postojeću vrijednost ili obrisati cijelo svojstvo i slično. Ova metoda je posebno korisna za rad s nizovima objekata. Druga metoda je save(). Ona je puno manje fleksibilna od update te prima samo jedan objekat. Taj objekat mora sadržati objekat identifikator i novi dokument, koji će se spremirati pod tim identifikatorom. Stari dokument će biti u potpunosti zamijenjen novim. Ova metoda djeluje na samo jedan dokument, jer je identifikator 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

Izmjena imena polja ili brisanje polja isto je moguće u mongo-u. Operatori koji se koriste za ovo su \$unset i \$rename. Sada ćemo polju preziva se promijeniti ime na preziva se:

```
> db.studenti.update( { ime : "Milana" }, { $rename : { preziva_se : "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 koristi se kada se želi obrisati polje dokumenta kao na slici 4.59., u ovom slučaju studentu Milana obrisano je polje preziva_se, ali da bi ovaj operator funkcionisao potrebno mu je dodijeliti neku vrijednost inače neće uspjeti operacija, tako da je moguće samo dodati prazne navodne znake da bi se određeno polje obrisalo.

```
> 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

Sada ćemo objasniti dva operatora procjene:

- \$regex – operator koji dohvata dokumente koji sadrže izraz naveden u operaciji
- \$expr – operator koji omogućava dohvaćanje dokumenta, korištenjem izraza 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

Da bismo objasnili primjenu operatora \$push i \$pop, dodaćemo ime više zanimanja studenti sa imenom Milana.

```
> 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

Dodavanje novih vrijednosti u polje reda vrijednosti, ali da pri tome ostanu stare vrijednosti se ostvaruje preko operatora \$push, za razliku od operatora \$set koji će obrisati postojeće vrijednosti i dodati nove. Na isti način s istom sintaksom koristi se operator \$pull koji služi za brisanje određene vrijednosti iz polja, isto pravilo vrijedi i za operator \$addToSet samo što će ovaj operator dodati vrijednost ukoliko navedena vrijednost ne postoji, dok operator \$push će dodati vrijednosti koje već postoje što može dovesti do vrijednosti koje se ponavljaju.

```
> 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

Operator \$pop se koristi kada je potrebno izbaciti prvu ili zadnju vrijednost iz reda vrijednosti. Polju reda vrijednosti dodjeljujemo vrijednost 1 kada želimo izbrisati zadnju vrijednost kao na slici 4.64., a kada želimo izbrisati prvu vrijednost dodjeljuje se 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

I za kraj, pogledajmo kako možemo obrisati dokumente iz kolekcije. Prvo ćemo dodati novog studenta, a 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

Sada 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

NoSQL baze podataka prisutne su tek nekoliko godina te su donijele mnoštvo prednosti i novosti u informatičkom svijetu vezanom za pohranu podataka. Uz sve prednosti, postoje i neki nedostaci NoSQL baza, a to su:

- Nedovoljna zrelost – relacione baze podataka koriste se već više od 20 godina i s godinama su se mnogo razvile i dovele na visok nivo. NoSQL baze podataka postoje tek nekoliko godina i nisu još postigle tu stabilnost kao relacione baze.
- Manja podrška – relacione baze imaju administracijsku podršku 24 sata dnevno i oni rješavaju problem kada neki od sistema upravljanja padne. Za NoSQL baze podataka postoji tek nekoliko firmi koje se time bave, a to su sve firme koje su tek počele sa radom i one nemaju podršku koja se odnosi na cijeli svijet.
- Poslovna inteligencija i analitika – NoSQL baze podataka su razvijene prema zahtjevima web 2.0 modernim aplikacijama. Prevazilaze standardne „insert-readupdate-delete“ cikluse tipičnih aplikacija i nude nekoliko svojstava analiza. Jednostavni upiti zahtijevaju određen nivo znanja u programiranju i većina alata na koje se oslanjaju ne pružaju povezivanje na NoSQL baze podataka.
- Administracija – krajnji cilj NoSQL baza podataka je da neće biti potrebna administracija, ali u stvarnosti je to drugačije i zahtijevaju se tehničke sposobnosti kao što su instalacija i održavanje.
- Nedovoljno naprednih eksperata – pošto su NoSQL baze podataka još relativno nove, većina ljudi koji se time bave još uvijek uče i nisu eksperti. To će se s vremenom naravno promijeniti.

IZVORI

- [1] 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 (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 (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 (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 (pristup 19.03.2020)
- [18] [https://www.fer.unizg.hr/download/repository/6.NoSQL\(1.od3\).pdf](https://www.fer.unizg.hr/download/repository/6.NoSQL(1.od3).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 (pristup 13.03.2020)
- [21] 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 (pristup 18.03.2020)

[25]

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

(pristup 14.03.2020)