Priručnik za NoSQL - MongoDB kroz primjere

Matijević, Marko

Sveučilište u Rijeci – Odjel za informatiku

Informacijski i komunikacijski sustavi

Marko Matijević

Priručnik za NoSQL - MongoDB kroz primjere

Diplomski rad

Mentor: doc. dr. sc. Danijela Jakšić

Rijeka, prosinac, 2020.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 9.6.2020.

Zadatak za diplomski rad

Pristupnik: Marko Matijević

Naziv diplomskog rada: Priručnik za NoSQL - MongoDB kroz primjere

Naziv diplomskog rada na eng. jeziku: NoSQL tutorial - MongoDB through examples

Sadržaj zadatka:

U ovome radu bit će opisan pojam NoSQL baze podataka, zašto se koristi, koja joj je svrha, razlika između relacijskih baza podataka i nerelacijskih baza podataka. U praktičnom dijelu rada bit će opisan MongoDB kroz praktične primjere. MongoDB jedna je od NoSQL baza podataka koja pohranjuje podatke u fleksibilne dokumente slične JSON-u, što znači da polja mogu varirati od dokumenta do dokumenta i struktura podataka može se mijenjati s vremenom. Uz izrađeni priručnik za rad s MongoDB, bit će navedene pozitivne i negativne strane rada u ovome sustavu te dano vlastito mišljenje o MongoDB-u.

Mentor:

Doc. dr. sc. Danijela Jakšić

Komentor:

Voditeljica za diplomske radove:

Izv. prof. dr. sc. Ana Meštrović

Zadatak preuzet: 1.7.2020.

(potpis pristupnika)

Sažetak

NoSQL izvorno se odnosi na „ne-SQL“ ili nerelacijska baza podataka koja pruža mehanizam za pohranu i dohvaćanje podataka koji se modeliraju na druge načine osim tabelarnih odnosa koji se koriste u relacijskim bazama podataka. U relacijskoj bazi podataka, zapis se često rastavlja i pohranjuje u zasebne tablice, a odnosi su definirani primarnim i stranim ključnim ograničenjima. U NoSQL bazi podataka, zapis se obično pohranjuje kao JSON dokument. NoSQL baze podataka dolaze u različitim vrstama na temelju njihovog modela podataka. Glavne su vrste dokument, ključ-vrijednost, široki stupac i graf. Oni pružaju fleksibilnu shemu i laganu razmjeru s velikim količinama podataka i rješenjima kod velikih opterećenja od strane korisnika.

MongoDB jedna je od NoSQL baza podataka koja pohranjuje podatke u fleksibilne dokumente slične JSON, što znači da polja mogu varirati od dokumenta do dokumenta i struktura podataka može se mijenjati s vremenom. Model dokumenta preslikava na objekte u kodu vaše aplikacije što olakšava rad s podacima. Upiti, indeksiranje i združivanje u stvarnom vremenu pružaju snažne načine pristupa i analize vaših podataka. MongoDB je jezgra distribuirane baze, sa velikom dostupnošću te je zbog toga olakšano skaliranje.

Ključne riječi

MongoDB, NoSQL, JSON, SQL, BSON

Sadržaj

1. Uvod

Danas velike većine aplikacija, bilo izrađivali web, mobilne ili desktop aplikacije koriste relacijski sustav baza podataka kao primarno spremište podataka. Nema sumnje da se način na koji se aplikacije bave podatcima značajno promijenio tijekom posljednjeg desetljeća. Prikuplja se sve više podataka i više korisnika istodobno im pristupa, više nego ikad prije. To znači da su skalabilnost i performanse veći izazov nego ikad ranije za relacijske baze podataka koje se temelje na shemi i zbog toga mogu biti teže skalirani.

Problem skalabilnosti SQL prepoznale su tvrtke poput Googlea, Amazona i Facebooka s ogromnim, rastućim potrebama za podatcima i infrastrukturom. Osmislili su vlastita rješenja problema - tehnologije poput BigTable, DynamoDB i Cassandre. Ovo sve veće zanimanje rezultiralo je brojnim NoSQL sustavima za upravljanje bazama podataka (DBMS-i), s naglaskom na performanse, pouzdanost i dosljednost. Uspjeh ovih zaštićenih sustava pokrenuo je razvoj niza sličnih sustava baze podataka otvorenog koda i vlasničkih baza podataka. Jedna od kreiranih nerelacijskih baza podataka je MongoDB. MongoDB je baza otvorenog koda bazirana na modelu dokumenta.

MongoDB je sustav za upravljanje bazama podataka dizajniran za web, mobilne i desktop aplikacije. Model podataka izgrađen je za visoke protoke čitanja, pisanja i jednostavnog skaliranja. Bez obzira zahtijeva li aplikacija samo jedan čvor baze podataka ili desetke njih, MongoDB može pružiti iznenađujuće dobre performanse. U ovom radu objasniti će se nerelacijski sustav baza podatka, koje su njegove karakteristike, vrste nerelacijskih baza podataka te razliku između relacijskih i nerelacijskih baza podataka. U trećem poglavlju biti će obrađen MongoDB, njegove značajke te osnovne MongoDB koncepte. Također je napravljen proces kreiranja baze podataka za dostavu jela i pića restorana pomoću MongoDB ljuske te operacije kreiranja, brisanja, ažuriranja i upita kroz primjere koji će biti prikazani pomoću alata NoSQLBooster.

2. NoSQL – Nerelacijski sustav baza podataka

Kraticu NoSQL prvi je put upotrijebio Carlo Strozzi 1998. godine dok je imenovao svoju „relacijsku“ bazu podataka otvorenog koda koja nije koristila SQL. Pojam NoSQL se ponovno pojavilo 2009. godine kada su je Eric Evans i Johan Oskarsson koristili za opis nerelacijskih baza podataka. Pojam NoSQL može imati 2 značenja: uobičajeniji prijevod "Ne samo SQL" ili "Nema SQL sustava", kako bi se naglasila činjenica da neki sustavi mogu podržavati jezike upita sličnih SQL-u. Većina se slaže da su baze podataka NoSQL baze podataka koje podatke pohranjuju u formatu različitom od relacijskih tablica. Uobičajena zabluda je da NoSQL baze podataka ili nerelacijske baze podataka ne pohranjuje dobro relacijske podatke. NoSQL baze podataka mogu pohraniti relacijske podatke, točnije oni ih jednostavno spremaju drugačije nego što to čine relacijske baze podataka. Zapravo, u usporedbi s SQL bazama podataka, mnogi smatraju da je modeliranje podataka odnosa u NoSQL bazama podataka lakše nego u SQL bazama podataka, jer povezani podatci ne moraju biti podijeljeni između tablica. (Foote, 2018)

Krajem 2000-ih pojavile su se NoSQL baze podataka jer su troškovi pohrane dramatično opali. Prošla je potreba za stvaranjem složenog, teško upravljanog modela podataka, samo u svrhu smanjenja dupliciranja podataka. Programerima je postojao primarni trošak razvoja softvera, pa su baze podataka NoSQL optimizirane za produktivnost programera.

2.1. Karakteristike NoSQL-a

NoSQL je generički pojam koji se koristi za baze podataka koje ne ovise o relacijskom modelu. Podatci ne moraju imati uobičajenu strukturu niti strogu shemu SQL tablice. Podatci se najčešće agregiraju kao parovi ključ - vrijednost, JSON dokumenti, bazirani na gradu ili stupčasto orijentirani. Korištenjem NoSQL baza podataka možete pohraniti goleme količine nestrukturiranih podataka i kasnije ih strukturirati.

Pojam NoSQL se odnosi na određenu skupinu baze podataka koja dijeli određene karakteristike, poput sljedećih:

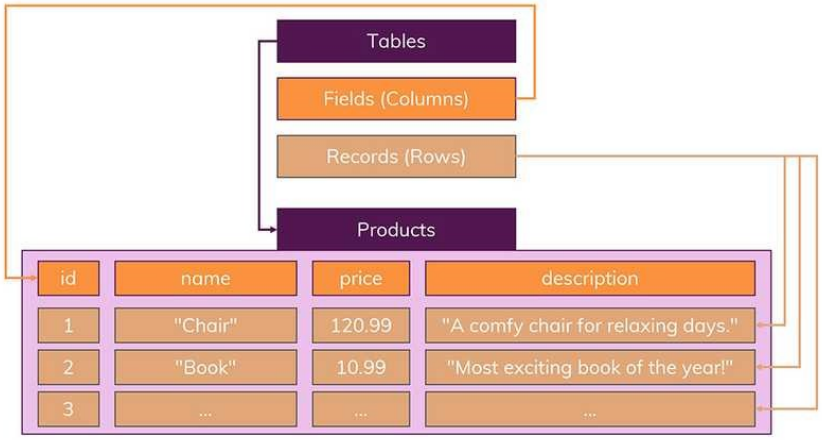
1. Nerelacijska baza podataka (eng. *Non-Relational*)
2. Bez sheme (eng. *Schema-Less*)
3. Vodoravno skalabilne (eng. *Horizontally Scalable*)
4. Manjak pridržavanja načela ACID (eng. *Lack of Adherence to ACID Principles*)
5. Nestandardizirani jezik upita (eng. *No Standard Query Language*)
6. Otvorenog koda (eng. *Open Source*)
   (What is NoSQL?, 2016)

1. Nerelacijska baza podataka

Većina baza podataka može se kategorizirati kao:

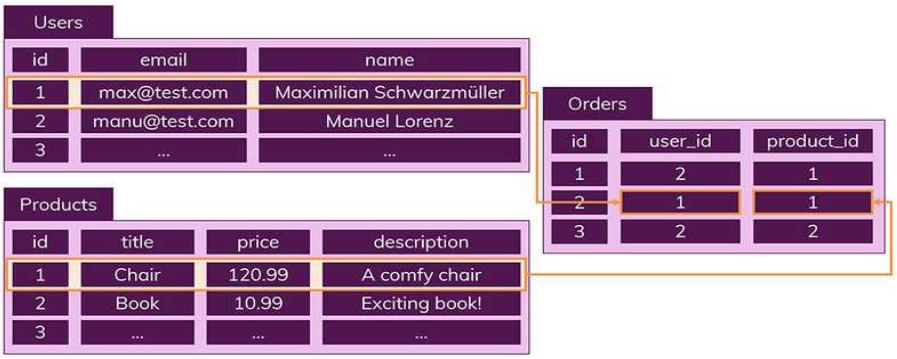
- Relacijske baze podataka
- Nerelacijske baze podataka

U relacijskim bazama podataka, podatci se pohranjuju kao zapisi u tablicama, a svaka tablica ima jasno definiranu strukturu - skup polja koja definira koji podatci mogu ući u tablicu, a koji ne. Struktura je definirana s obzirom na nazive polja kao i tipove podataka. Relacijske baze podataka koriste se strukturiranim jezikom upita (SQL). Kada se dodaju novi podatci, novi se zapisi ubacuju u postojeće tablice ili se dodaju nove tablice. (Schwarzmüller, 2018)



Slika 1. Prikaz relacijske baze podataka *(Maximillian, 2018)*

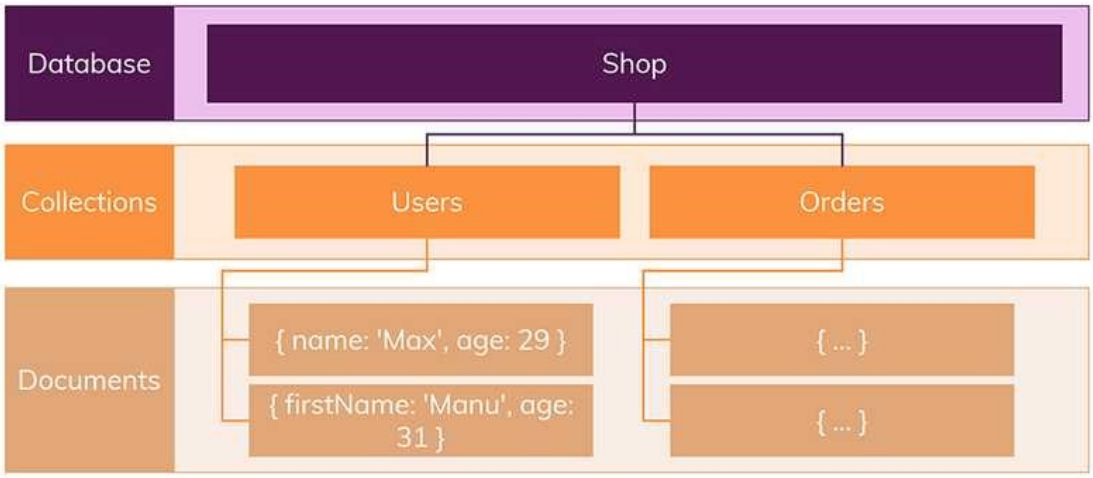
Drugi važan dio baza podataka temeljenih na SQL-u su relacije, odnosno dijeljenje podatka u više tablica kako biste izbjegli dupliciranje podataka. Na slici 2. možemo vidjeti tri tablice: Korisnici, proizvodi i narudžbe (eng. *Users, Products i Orders*). Svaka će tablica sadržavati samo podatke koji nisu pohranjeni u bilo kojoj drugoj tablici. (Schwarzmüller, 2018)



Slika 2. Prikaz relacija u relacijskim bazama podataka *(Maximillian, 2018)*

Ova jasna struktura može imati prednosti, a to znači da nećete imati netočne podatke u jednoj od svojih tablica, dok ćete u svim ostalim imati točne. (Schwarzmüller, 2018)

Glavna razlika između relacijskih i nerelacijskih baza podataka je način na koji one pohranjuju svoje podatke. Nerelacijska baza podataka pohranjuje ih u ne tabličnom obliku i nastoji biti fleksibilnija od tradicionalnih relacijskih struktura baza podataka temeljenih na SQL-u. Ne slijedi tradicionalne sustave sa relacijskim modelom koji pružaju upravljanja relacijskim bazama podataka. (What is a Non-Relational Database?, n.d.) Umjesto toga, podatke strukturirate u kolekcije (eng. *collections*) točnije kao tablice u SQL svijetu. Red u tablici se sada naziva dokumentom. NoSQL baze podataka ne uspostavljaju veze između pojedinih zapisa. Jedan se zapis obično pohranjuje kao pojedinačni JSON dokument i replicira na više čvorova u klasteru[1].



Slika 3. Prikaz nerelacijske baze podataka *(Maximillian, 2018)*

Nerelacijske baze podataka često se koriste kada treba organizirati velike količine složenih i raznolikih podataka. (Schwarzmüller, 2018) Primjerice, jedna velika trgovina ima bazu podataka gdje svaki kupac ima vlastiti dokument koji sadrži sve njegove podatke, imena, adrese, povijesti narudžbi, podataka o kreditnoj kartici i sl. Unatoč različitim formatima, svaki od tih podataka može se pohraniti u isti dokument.

Također, često se izvode brže jer upit ne mora pregledavati nekoliko tablica da bi se dobio odgovor, kao što to često čine relacijski skupovi podataka. Nerelacijske baze podataka stoga su idealne za pohranu podataka koji se mogu često mijenjati ili za aplikacije koje obrađuju mnoge različite vrste podataka. Mogu podržati brzo razvijajuće programe kojima je potrebna dinamička baza podataka koja se može brzo mijenjati i prilagoditi velikim količinama složenih, nestrukturiranih podataka. (What is a Non-Relational Database?, n.d.)

[1] Računalni klaster je skup povezanih računala (čvorova) mrežama velike brzine, koji rade zajedno kao da su jedan stroj.

2. Bez sheme (eng. *Schema-less*)

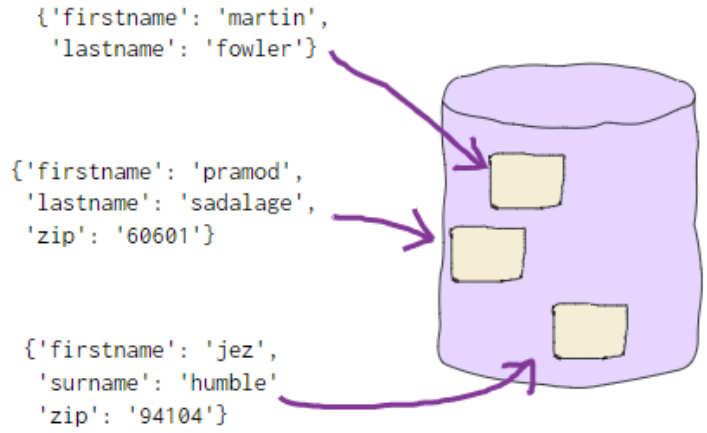
U relacijskoj shemi definira se koji se stupci pojavljuju u tablici, njihova imena i njihove vrste podataka kao što je to prikazano na slici 4.

```
create table customers (id int, firstname text, lastname text)
insert into customers (firstname, middlename, lastname) values (...)
```

Slika 4. Primjer kreiranja i unosa podataka u Relacijsku tablicu – Vlastoručni rad

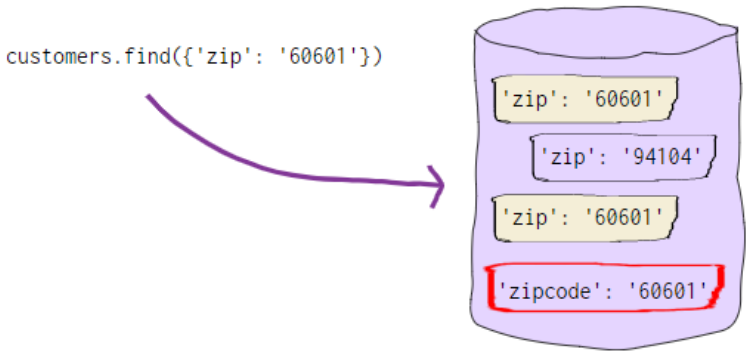
Ukoliko bi u tablici htjeli unesti podatke koji se ne nalaze u shemi dobili bi grešku. Baza podataka bez sheme omogućuje pohranjivanje podataka, strukturiranih s pojedinačnim poljima i strukturama.

Koristeći bazu podataka bez sheme smanjujete vrijeme pripreme jer se ne mora kreirati shema, a također povećava se i fleksibilnost (možete pohraniti sve vrste podataka bez prethodne definicije). Slika 5. prikazuje primjer baze podataka bez sheme.



Slika 5. Prikaz baze podataka bez sheme *(Fowler, 2013)*

Strukture bez sheme dalje imaju implicitnu shemu kao što je to prikazano na slici 6. Bilo koji kod koji manipulira podacima mora iznijeti neke pretpostavke o njegovoj strukturi, kao što je naziv polja. S bilo kojim podatcima koji se ne uklapaju u implicitnu shemu neće se pravilno manipulirati, što dovodi do pogrešaka.



Slika 6. Prikaz implicitne sheme *(Fowler, 2013)*

3. Manjak pridržavanja načela ACID

Treba li baza podataka otkazati operaciju i osigurati konzistentnost podataka u slučaju mrežnog kvara? Ili bi baze podataka trebale riskirati nekonzistentnost podataka kako bi osigurale visoku dostupnost?

Relacijske baze podataka karakteriziraju četiri svojstva transakcije koji čine akronim ACID, a to su:

- Atomičnost (eng. ***A**tomic*) - Transakcija je logična cjelina rada koja se mora dovršiti sa svim svojim izmjenama podataka ili se niti jedna ne izvršava.
- Konzistentnost (eng. ***C**onsistent*) - Na kraju transakcije svi podatci moraju ostati u konzistentnom stanju.
- Izolirano (eng. ***I**solated*): Izmjene podataka izvršene transakcijom moraju biti neovisne o drugoj transakciji. Ako se to ne dogodi, ishod transakcije može biti pogrešan.
- Trajno (eng. ***D**urable*): Kada je transakcija dovršena, učinci izmjena koje je izvršila transakcija moraju biti trajni u sustavu. (Kulkarni, n.d.)

Iako se ova svojstva čine logičnima za većinu aplikacija, nisu prikladne za vodoravno skaliranje (u slijedećem poglavlju detaljnije opisano), performanse, visoku konzistentnost i toleranciju grešaka. Primarni fokus NoSQL-a je održavanje dostupnosti nudeći eventualnu konzistentnost. (Kulkarni, n.d.) Pojam eventualna konzistentnost znači imati preslike podataka na više računala kako bi se dobila velika dostupnost i skalabilnost. Dakle, promjene izvršene na bilo kojoj stavci podataka na jednom stroju moraju se proširiti na druge replike. Replikacija podataka možda neće biti trenutna, jer će se neke preslike ažurirati odmah, a druge s vremenom. Te se preslike mogu međusobno kombinirati, ali s vremenom postaju konzistentne.

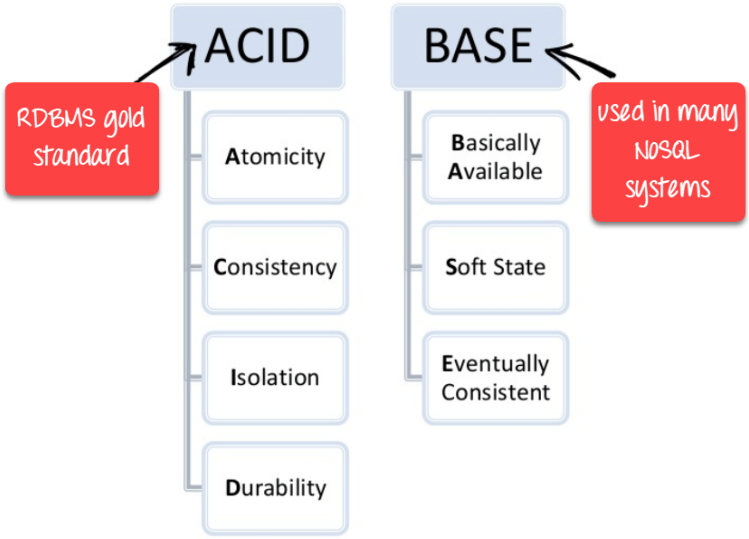
Alternativa ACID-u je BASE načelo, što slijede baze podataka NoSQL:

- Dostupnost (eng. ***B**asically **A**vailable*) - Zajamčeno je da će sustav biti dostupan u slučaju kvara.

- Meko stanje (eng. ***S**oft State*) - stanje podataka moglo bi se promijeniti bez interakcije aplikacije zbog moguće konzistentnosti.
- Eventualna konzistentnost (eng. ***E**ventual Consistency*) - Sustav će na kraju biti konzistentan nakon unosa aplikacije. Podaci će se replicirati na različite čvorove i na kraju će doseći dosljedno stanje. Ali dosljednost nije zajamčena na razini transakcije. (Pore, 2018)

BASE sustavi omogućuju vodoravno skaliranje, toleranciju kvarova i visoku dostupnost po cijenu konzistentnosti. Dakle, ako vaša aplikacija zahtijeva visoku konzistentnost i skalabilnost, mogla bi biti prikladna NoSQL baza podataka izgrađena na BASE svojstvima. (Pore, 2018)

Većina baza podataka NoSQL donekle dopušta ograničenja ACID-a. To je zato što je većina NoSQL rješenja razvijena u svrhu pružanja visoke dostupnosti i skalabilnosti u klaster okruženju. Kada osiguravamo visoku dostupnost u klaster okruženju, obično se mora žrtvovati ili postojanost ili trajnost (ili pronaći ravnotežu između sve tri). Opuštanjem dosljednosti može se postići veća dostupnost. NoSQL pristup prepoznaje da postoje slučajevi u kojima nedosljedni podatci nisu prepreka za daljnji rad te je uredu predstavljati podatke koji nisu sto posto dosljedni sto posto vremena. (NoSQL Tutorial: Types of NoSQL Databases, What is & Example, n.d.) Međutim, neki NoSQL sustavi, poput MongoDB-a, podržavaju ACID transakcije.



Slika 7. ACID i BASE načela *(NoSQL Tutorial: Types of NoSQL Databases, What is & Example, n.d.)*

4. Horizontalna skalabilnost

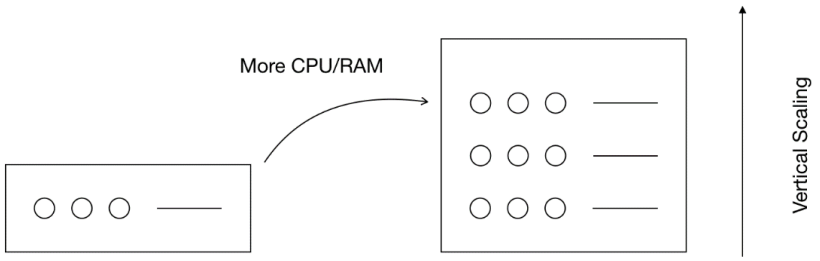
Skalabilnost je sposobnost sustava da se proširi u skladu s vašim poslovnim potrebama. Primjerice, skaliranje web aplikacije svodi se na to da više ljudi koristi vašu aplikaciju. Koncept NoSQL baza podataka postao je popularan kod internetskih divova poput Googlea, Facebooka, Amazona itd. koji se bave ogromnim količinama podataka. Ako podatci nisu dostupni, aplikacije se ne mogu

pokretati. Ukoliko aplikacije ne mogu biti pokrenute ili rade sporo, firme gube poslove. Zato je jako bitno osigurati da su baze podataka dostupne i da rade.

Postoje dva načina skaliranja:

- Vertikalno skaliranje i
- Horizontalno skaliranje

Vertikalno skaliranje postupak je poboljšanja procesorske snage poslužitelja, povećanjem njegovog hardverskog kapaciteta (npr. CPU, RAM). (Mullins, 2018). No u slučaju nerelacijskih baza podataka to nije nabolje rješenje.



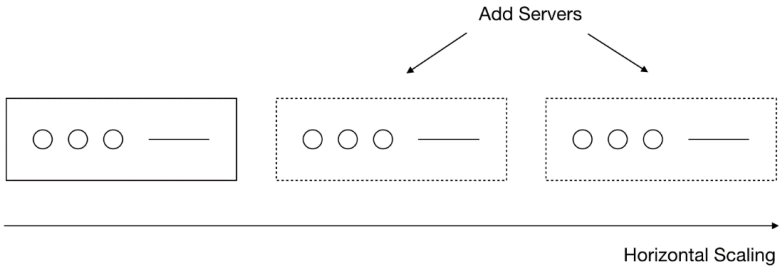
Slika 8. Vertikalno skaliranje *(Lim, How to Scale SQL and NoSQL Databases, 2020)*

Većina NoSQL baza podataka ističe se u klaster okruženjima. Ovdje se podaci dijele na više računala tako da svako računalo može izvršiti određeni zadatak neovisno od ostalih. Svaki procesor može izvršiti svoj zadatak bez potrebe za dijeljenjem memorije ili prostora na disku s drugima. Iako se relacijske baze podataka također mogu postaviti unutar klastera, relacijski pristup baza podataka otežava postavljanje od NoSQL baza podataka. (What is NoSQL?, 2016)

Primjerice za veliku organizaciju poput IBM-a koja ima urede po cijelom svijetu baza podataka o zaposlenicima bila bi znatno velika. Ako se relacijska baza podataka želi horizontalno skalirati, različite bi se tablice nalazile na različitim poslužiteljima. Ako želimo saznati podatke o jednom zaposleniku, trebali bismo spojiti 3 tablice (minimalno) u savršeno normaliziranu bazu podataka. Sada su spajanja na različitim poslužiteljima izuzetno učinkovita, pogotovo kada je količina podataka ogromna.

NoSQL baze podataka dizajnirane su za učinkoviti rad na distribuiranim sustavima koji se brzo horizontalno skaliraju. Horizontalno skalirati znači dodati više čvorova u sustav, poput dodavanja novog računala u distribuiranu softversku aplikaciju. (Kulkarni, n.d.) Distribuirani sustav ima dodatnu prednost pružanja stalne visoke dostupnosti. Više kopija zapisa čuva se na poslužiteljima i policama, a kvar hardvera ne utječe na dostupnost podataka. Možete sigurno upotrebljavati hardver srednje razine umjesto skupih vrhunskih poslužitelja za upravljanje velikom brzinom podataka. Skupljeni podaci u jednom zapisu ne mogu se povezati s agregiranim podacima u

drugom zapisu. Svaki relevantni zapis u bazi podataka treba ažurirati ako želite dodati atribut. NoSQL baze podataka su najkorištenije za velike količine podataka za koje naknadno nije potrebno strukturiranje ili povezivanje. (Kaplarevic, What is NoSQL database, 2020)



Slika 9. Horizontalno skaliranje *(Lim, How to Scale SQL and NoSQL Databases, 2020)*

5. Nestandardizirani jezik upita

Ne postoji standardni jezik upita koji podržava sve baze podataka NoSQL. Neke NoSQL baze podataka imaju vlastiti jezik upita, dok drugi podržavaju razne jezike kao što su JSON, XQuery, SPARQL itd. (What is NoSQL?, 2016)

6. Otvorenog koda

Većina NoSQL baza podataka su otvorenog koda. Iako postoje i mnoge relacijske baze podataka koji su otvorenog koda, NoSQL naginje prema projektima otvorenog koda, pri čemu mnoge organizacije doprinose razvojnim naporima za jedinstveno rješenje.

2.2. Vrste pohrane podataka u NoSQL-u

NoSQL Baze podataka uglavnom su kategorizirane u četiri vrste:

1. Ključ – vrijednost baze podataka (eng. *Key-value databases*)
2. Stupčasto orijentirani (eng. *Column-oriented*)
3. Graf-bazirani (eng. *Graph based*)
4. Dokumentno orijentirani (eng. *Document oriented*)

Svaka kategorija ima svoje jedinstvene atribute i ograničenja te bi korisnici trebali odabrati bazu podataka na temelju potreba aplikacije.

1. Ključ – vrijednost baze podataka

U Ključ – vrijednost baze podataka koristi se **hash tablica** u kojoj jedinstveni ključ pokazuje na vrijednost. Ključevi se mogu organizirati u grupe ključeva, samo što ključevi trebaju biti jedinstveni u vlastitoj grupi. Sljedeća tablica prikazuje primjer ključ – vrijednost gdje su ključevi: **Ime**, **Prezime** i **Datum rođenja**, a vrijednosti: **Marko**, **Matijević**, **08.12.1995**.

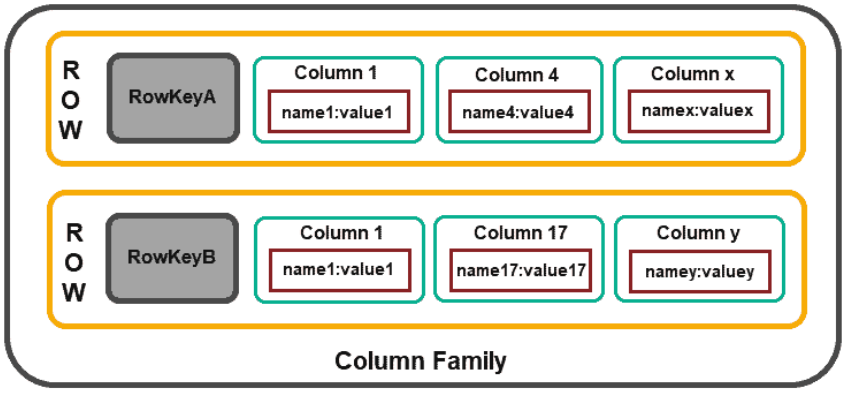
Tablica 1. Ključ vrijednost baze podataka (Vlastoručni rad)

Ključ	Vrijednost
Ime	**Marko**
Prezime	**Matijević**
Datum rođenja	**08.12.1995.**

Postoje implementacije ključ - vrijednost baze podataka koje pružaju mehanizme „pred-memoriranja“ (eng. *caching*), koji uvelike poboljšavaju njihovu izvedbu. Podatci se pohranjuju u obliku niza, JSON ili BLOB (binarni veliki objekt). Najpoznatija baza podataka NoSQL koja se gradi na spremištu ključnih vrijednosti je Amazon-ova DynamoDB.

2. Stupčasto orijentirani

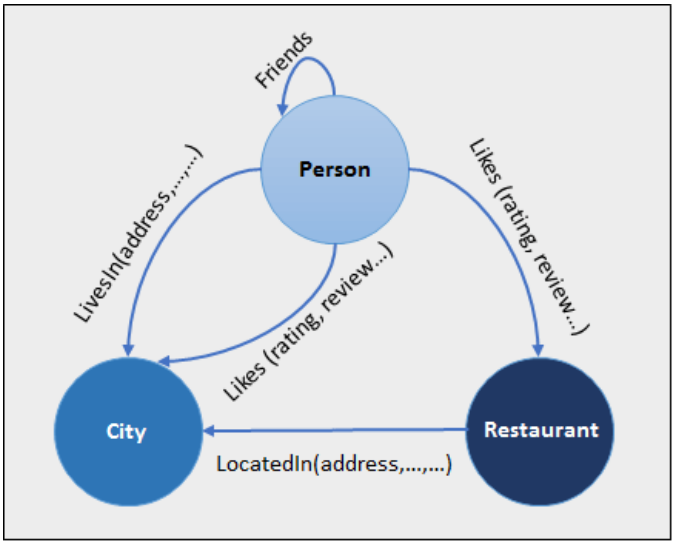
Pohrana usmjerena na stupac omogućuje učinkovito pohranjivanje podataka. Izbjegava trošenje prostora prilikom spremanja *null* vrijednosti sa jednostavnim nepohranjivanjem stupca kada vrijednost za taj stupac ne postoji. Svaka jedinica podataka može se smatrati skupom parova ključ - vrijednost, pri čemu se sama jedinica identificira uz pomoć primarnog identifikatora, često nazivanog i primarnim ključem. Oni pružaju visoku izvedbu za upite za agregiranje kao što su SUM, COUNT, AVG, MIN itd. Jer su podatci lako dostupni u stupcu. (Christine, 2020)



Slika 10. Stupčasto orijentirane nerelacijske baze podataka *(Kaplarevic, What is NoSQL database, 2020)*

3. Graf-bazirana baza podataka

Podaci u bazama podataka bazirani na grafu organizirani su u **čvorove**, dok **rubovi** uspostavljaju **veze** između tih čvorova podataka. Bez striktne sheme, baza podataka bazirana na grafu može rasti uz mala ograničenja. Na slici 11. vidimo 3 čvora: Osoba, grad i restoran (eng. *Person, City, Restaurant*). Primjer jedne relacije je da je čvor restoran lociran na nekoj adresi u gradu.



Slika 11. Graf-bazirana baza podataka *(NoSQL Tutorial: Types of NoSQL Databases, What is & Example, n.d.)*

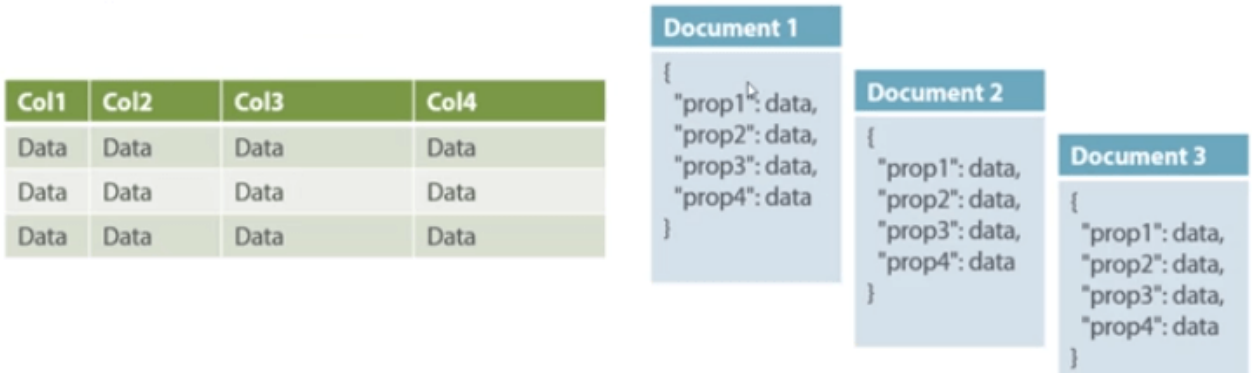
U usporedbi s relacijskom bazom podataka u kojoj su tablice slabo povezane, baza podataka grafova je multi-relacijske prirode. Odnos prelaska je brz jer su već zarobljeni u bazi podataka i nema potrebe za izračunavanjem. Baza podataka grafikona koja se uglavnom koristi za društvene mreže, logistiku, prostorne podatke. (NoSQL Tutorial: Types of NoSQL Databases, What is & Example, n.d.)

4. Dokumentno orijentirani

Dokumentno orijentirane baze podataka pohranjuju djelomično strukturirane podatke u dokumente. Možete koristiti bilo koji format kao što je JSON[2], BSON koji koristi MongoDB ili XML za pohranu podataka u bazu podataka dokumenata. Zbog rasprostranjenosti *JavaScript*-a u razvoju web aplikacija, format dokumenta JSON stekao je na popularnosti.

[2] JSON skraćenica od eng. *JavaScript Object Notation* je standardni format datoteke i čovjeku čitljivi format razmjene podataka.

Na slici 12. možete vidjeti da imamo tablicu sa redcima i stupcima te bazu podataka dokumenata koja ima strukturu sličnu JSON-u. Za relacijsku bazu podataka, morate znati koje stupce imate. Međutim, za bazu podataka dokumenata imate spremište podataka poput JSON objekta. Ne morate definirati što ga čini fleksibilnim. Podaci u dokumentu su polu-strukturirani[3] kako bi pružili veću fleksibilnost prilikom upita. Za razliku od osnovnih spremišta ključ-vrijednost, više ne morate dohvaćati cijeli zapis, već samo relevantni dio dokumenta. Također možete stvoriti dodatne indekse na temelju polja definiranih u dokumentima. Dokumentno orijentirane baze podataka uglavnom se koriste za CMS sustave, blog platforme, analitiku u stvarnom vremenu i aplikacije za e-trgovinu. (NoSQL Tutorial: Types of NoSQL Databases, What is & Example, n.d.)



Slika 12. Struktura Tablice i Dokumenta u bazi podatka *(NoSQL Tutorial: Types of NoSQL Databases, What is & Example, n.d.)*

2.3. MySQL vs NoSQL

Kad se govori o odabiru moderne baze podataka, jedna je od najvećih odluka odabir relacijske (SQL) ili nerelacijske (NoSQL) baze podataka. Iako su oba izbora dobra, između njih postoje ključne razlike koje korisnici moraju imati na umu prilikom donošenja odluke.

Relacijske baze podataka koriste strukturirani jezik upita (SQL) za definiranje i manipulaciju podacima. SQL je jedna od najsvestranijih i najčešće korištenih dostupnih opcija, što ga čini sigurnim izborom i posebno izvrsnim za složene upite. S druge strane, to može biti restriktivno. SQL zahtijeva da koristite unaprijed definirane sheme za određivanje strukture podataka prije nego što s njima radite. Uz to, svi vaši podaci moraju slijediti istu strukturu. To može zahtijevati značajnu pripremu unaprijed te da bi promjena strukture bila teška za cijeli sustav. SQL baze podataka vertikalno su skalabilne, što znači da možete povećati opterećenje na jednom poslužitelju povećavanjem hardvera.

[3] Polu-strukturirani podaci oblik su strukturiranih podataka koji se ne baziraju tabličnoj strukturi modela podataka, ali sadržavaju oznake za razdvajanje semantičkih elemenata te provode hijerarhiju zapisa i polja unutar podataka.

NoSQL baze podataka, s druge strane imaju dinamičke sheme za nestrukturirane podatke, a podaci se pohranjuju na mnogo načina: stupčasto orijentirani, dokumentno orijentirani, temeljeni na grafovima ili organizirani kao par ključ/vrijednost. To znači da možete stvoriti dokumente bez potrebe da prethodno definirate njihovu strukturu, svaki dokument može imati svoju jedinstvenu strukturu, sintaksa se može razlikovati od baze podataka do baze podataka i možete dodavati polja direktno pri radu. NoSQL baze podataka su vodoravno skalabilne, što znači da obrađujete više informacija dodavanjem više poslužitelja u NoSQL bazu podataka.

U tablici 2. prikazana je razlika SQL i NoSQL baze podataka kroz različite čimbenike.

Tablica 2. Razlika SQL i NoSQL baze podataka (Vlastoručni rad)

	SQL baze podataka	NoSQL baze podataka
Model za pohranu podataka	**Tablice s fiksnim redcima i stupcima**	**- Dokument: JSON dokumenti,** **- Ključ - vrijednost: parovi ključ - vrijednost,** **- Stupčasto orijentirani: tablice s redovima i dinamičkim stupcima,** **- Grafovi: čvorovi i rubovi**
Glavna svrha	**Opća namjena**	**- Dokument: opća namjena,** **- Ključ - vrijednost: velike količine podataka s jednostavnim upitima pretraživanja,** **- Stupčasto orijentirani: velike količine podataka s predvidljivim obrascima upita,** **- Grafovi: analiza i prelazak odnosa između povezanih podataka**
Shema	**Striktna**	**Fleksibilna**
Skalabilnost	**Vertikalna skalabilnost (povećanje performanse servera)**	**Horizontalna skalabilnost (rad na više servera)**
ACID transakcije	**Podržane**	**Jedini MongoDB podržava ACID transakcije od svih NoSQL baza podataka**
Primjeri	**Oracle, MySQL, PostgreSQL, Microsoft SQL Server**	**- Dokument: MongoDB i CouchDB,** **- Ključ - vrijednost: Redis i DynamoDB,** **- Stupčasto orijentirani: Cassandra i HBase,** **- Grafovi: Neo4j i Amazon Neptune**

3. MongoDB

MongoDB spada u dokument orijentirane NoSQL baze podataka. MongoDB aktivno razvija 10gen, koji je kasnije preimenovan u MongoDB I.inc. MongoDB je otvorenog koda, a njegov je kod dostupan na raznim platformama poput *GitHub*-a. Jedan od najvažnijih razloga popularnosti MongoDB-a je taj što je JSON-friendly baza podataka. To znači da se dokumenti pohranjuju i preuzimaju iz MongoDB kao *JavaScript* objekti. Ovi se JSON podaci interno pretvaraju u BSON formatu prilikom spremanja u sustavu. Dakle, ovo daje krajnju fleksibilnost gdje možemo koristiti isti format podataka od klijenta do poslužitelja i na kraju do baze podataka.

Sljedeća važna značajka MongoDB-a je priroda bez sheme. U relacijskim bazama podataka, dužni ste definirati (prije vremena) točnu strukturu podataka koji se pohranjuju. U MongoDB-u dokumente koje pohranjujete u bazu podataka ne moraju slijediti bilo koju shemu, osim ako je programer ne provodi na razini aplikacije. Ovo čini MongoDB izvrsnom pogodnošću za razvoj temeljen na *Agile*-u[4], kao što biste mogli brzo provesti izmjene na aplikacijskoj shemi.

MongoDB je napravljen za horizontalno skaliranje. Model podataka dokumentno orijentiran olakšava podjelu podataka na više poslužitelja. MongoDB se automatski brine o uravnoteženju podataka i učitavanju preko klastera, automatski preraspodjeljujući dokumente i usmjeravajući čitanje i pisanje na ispravne poslužitelje.

MongoDB je baza podataka koja osim kreiranja, čitanja, ažuriranja i brisanja podataka podržava opcije:

- indeksiranja,
- agregacije,
- posebne kolekcije i
- datotečni sustav.

sve to bez utjecanja na brzinu rada. MongoDB koristi onoliko RAM-a koliko može i njegova pred memorija te pokušava automatski odabrati ispravne indekse za upite. Također sadrži značajke iz relacijskih sustava. Primjerice obrada i logika se prebacuje na klijentsku stranu (njima upravljaju upravljački programi ili korisnički kod aplikacije). (Shannon BradShaw, 2019)

[4] Agilan razvoj softvera odnosi se na razvoj softvera temeljen na iterativnom razvoju, gdje se rješenja razvijaju suradnjom između organizirajućih višefunkcionalnih timova.

3.1. Osnovni koncepti MongoDB-a

U ovom poglavlju biti će opisani osnovni koncepti MongoDB-a:

- **Dokument** je osnovna jedinica podataka u MongoDB-u i skoro je ekvivalenta redu iz relacijskih baza podataka.
- **Kolekcija** je skup dokumenata za koju možemo reći da je tablica sa dinamičnom shemom.
- Jedna instanca MongoDB-a može sadržavati više neovisnih **baza podataka**.
- Svaki dokument sadrži poseban ključ, **_id** koji je jedinstven unutar kolekcije.
- MongoDB dolazi sa jednostavnom *Javascript* ljuskom (eng. *shell*), koja je korisna za administraciju MongoDB instanci i manipulaciju podataka. (Shannon BradShaw, 2019)

U tablici 3. prikazana je razlika relacijskih baza podataka i MongoDB-a.

Tablica 3. Razlika relacijskih baza podataka i MongoDB-a, izrađeno prema *(MongoDB - Overview, n.d.)*

Relacijske baze podataka	Nerelacijske baze podataka
Baza	**Baza**
Tablica	**Kolekcija**
Red	**Dokument**
Stupac	**Polje (ključ)**
Join tablice	**Ugniježđeni dokumenti**
Primarni ključ	**Primarni ključ (zadani ključ *_id* koji osigura sam MongoDB)**

3.1.1. Dokument

MongoDB sadrži zapise podataka kroz BSON dokumente. BSON je binarni prikaz JSON dokumenta, koji sadrži više tipova podataka od običnog JSON dokumenta (detaljnije u poglavlju 3.3). MongoDB dokument je uređena lista ključeva sa pripadajućim vrijednostima. Struktura dokumenta u različitim programskim jezicima varira, primjerice u *Javascript*-u objekti se reprezentiraju kao:

```
{ "ime" : "Marko" }
```

Ovaj dokument sadrži jedan ključ **ime** i vrijednost **Marko**. Dokumenti su većinom slučaja kompleksniji pa jedan dokument može sadržavati više parova ključa – vrijednosti:

```
{ "ime": "Marko", "prezime": "Matijević" }
```

Iz prethodno navedenog primjera vidimo da dokumenti mogu biti od različitih tipova podataka: vrijednost **ime** je *string*, a **godište** *integer*. Ključevi u dokumentu su *string-ovi*. Bilo koji UTF-8 znak je dopušten kao ime ključa osim:

- znaka **\0** tzv. *Null* znak koji se koristi za označavanje završnog *string* ključa
- znakove **.** jer je . notacija za elemente u nizu i **$** koji se koristi za projekciju koji vraća prvi odgovarajući element iz niza na temelju uvjeta
- Ime polja **_id** rezervirano je za upotrebu kao primarni ključ; njegova vrijednost mora biti jedinstvena u kolekciji, nepromjenjiva je i može biti bilo koje vrste osim niza. (Shannon BradShaw, 2019)

MongoDB je *type-sensitive* i *case-sensitive* što znači da:

```
{ "cijena": 1995 }
{ "cijena": "1995" }
```

nisu jednaki kao:

```
{ "cijena": 1995 }
{ "Cijena": 1995 }
```

Dokumenti u MongoDB-u ne smiju sadržavati duplicirane ključeve. Dokument prikazan na slijedećem primjeru je ne validan.

```
{ "ime": "Marko" ,  "ime": "Ivan" }
```

MongoDB čuva redoslijed polja dokumenata nakon operacija pisanja, osim u sljedećim slučajevima:

- Polje **_id** je uvijek prvo polje u dokumentu.
- Ažuriranja koja uključuju preimenovanje polja rezultiraju u sortiranju polja u dokumentu.

Pri generiranju dokumenta u MongoDB-u, kao polje dokumentu dodaje se primarni ključ **_id**. U MongoDB-u svaki dokument pohranjen u kolekciji zahtijeva jedinstveno polje **_id** koje djeluje kao primarni ključ. Ako umetnuti dokument izostavlja polje **_id**, pokretački program MongoDB automatski generira **ObjectId** za polje **_id**. (Shannon BradShaw, 2019)

MongoDB koristi znak **točke** (eng. *dot notation*) nad elementima niza i za pristup poljima ugniježdenog dokumenta. Ukoliko bi htjeli doći do vrijednosti **Primorska 20** (prvog elementa u nizu na slijedećem primjeru), koristili bi **adrese.1**.

```
{ "adrese": ["Prvomajska 10", "Primorska 20", "Omladinska 40"] }
```

Isto tako vrijedi i za ugniježdene dokumente. Ako želimo ispisat **id** artikla koristili bi **kosarica.artikl_id.**

```
{ "kosarica": { "artikl_id": "10", "kolicina": "1", "neto_cijena": 50 } }
```

3.1.2. Kolekcija

Kolekcija je grupa dokumenta, ona je analogna tablici iz relacijskih baza podataka. Kolekcija postoji unutar jedne baze podataka. Kolekcije ne provode shemu. Dokumenti unutar kolekcije mogu imati različita polja. Tipično su svi dokumenti u kolekciji slične ili srodne namjene.

Kolekcije sadrže dinamične sheme odnosno dokumenti unutar jedne kolekcije mogu imati različita polja (ključeve i vrijednosti) kao što je dolje prikazano.

```
{ "ime": "Marko" }
{ "godište": 1995 }
```

Sudeći po pravilima dinamične sheme, sve dokumente možemo staviti u jednu kolekciju. Iz više razloga bi bilo bolje da postoje više kolekcija:

- Stavljanje različitih dokumenata u kolekciju može napraviti pomutnju programerima. Programerima je potrebno da svaki upit nad bazom vraća dokumente istog tipa jer bi se posebno trebalo odvojiti vrijeme za implementaciju dodatne logike što je nepotrebno.
- Prilikom upita jednoj velikoj kolekciji, MongoDB-u bi trebalo više vremena da vrati jedan ili više podatka.
- Počinjemo nametati neku strukturu našim dokumentima pomoću *indeksa*. Indeksi se kreiraju po kolekciji te ukoliko stavimo dokumente istog tipa podataka u jednu kolekciju, možemo efikasnije indeksirati kolekciju (detaljnije u poglavlju 3.9.)

Kolekcija je identificirana po imenu. Imena kolekcija može biti UTF-8 *string*, sa par restrikcija:

- Prazan *string* („“) je ne validno ime kolekcije
- Kolekcija ne smije sadržavati **\0** jer to označava kraj imena kolekcije
- Kolekcija ne smije početi sa riječi **system**, jer je prefiks zauzet za interne kolekcije MongoDB-a.
- Kolekcija ne smije sadržavati **$** jer sistemske kolekcije MongoDB-a koriste znak **$** te se isključivo može ako se pristupa tim kolekcijama (Shannon BradShaw, 2019)

3.1.3. Baza podataka

MongoDB grupira kolekcije u baze podataka. Jedna instanca MongoDB-a može imati više baza, od kojih svaka grupira nula ili više kolekcija. Svaka baza ima svoje dozvole koje su spremljene u posebnim podatcima na disku. Dobra konvencija je spremanje podataka za jednu aplikaciju u jednu bazu podataka. Odvojene baze podataka korisne su prilikom pohrane podataka za nekoliko aplikacija ili korisnika na istu MongoDB server. Poput kolekcije, baze podataka identificiraju se poimence. Nazivi baze podataka mogu biti bilo koji UTF-8 niz, uz sljedeća ograničenja:

- Prazan *string* („“) nije važeće ime baze podataka.
- Ime baze podataka ne može sadržavati nijedan od ovih znakova: /, \,., ", *, <,>,:, |,?, $, (Jedan razmak) ili \0 (*null* znak). Treba se držati alfanumeričkih ASCII znakova
- Imena baza podataka su *case-sensitive*.

- Imena baza podataka ograničena su na najviše 64 bajta.
- Bazu se ne smije nazvati *admin*, *local* i *config* koje su sistemske baze koje služe za dozvole, različitih akcija nad bazama (npr. gašenje servera):
    - *admin* je korijen baze. Ukoliko je korisnik dodan u *admin* bazu, on ima dozvole za sve baze primjerice, prikaz svih baza ili gašenje servera.
    - - *local* baza podataka nije vidljiva za replikaciju: kolekcije u lokalnoj bazi podataka se ne repliciraju.
    - *config* baza podataka služi za mijenjanje postavka baze npr. particioniranje (Shannon BradShaw, 2019)

3.2. Model podataka za MongoDB

Podaci u MongoDB-u imaju fleksibilnu shemu. Za razliku od SQL baza podataka, gdje morate odrediti i deklarirati tablice shemu prije umetanja podataka, kolekcije MongoDB-a ne provode strukturu dokumenata. Ova fleksibilnost olakšava preslikavanje dokumenata na entitet ili objekt. Svaki dokument može se podudarati s podatkovnim poljima predstavljenog entiteta, čak i ako podatci imaju značajne varijacije. Međutim, u praksi dokumenti u kolekciji imaju sličnu strukturu.

Ključni izazov u modeliranju podataka je uravnoteženje potreba aplikacije. Kada dizajnirate podatkovne modele, uvijek uzmite u obzir upotrebu aplikacije podataka (tj. upiti, ažuriranja i obrada podataka), kao i naslijeđena struktura samih podataka. (MongoDB, 2016)

3.2.1. Struktura dokumenta

Ključna odluka u dizajniranju modela podataka za MongoDB aplikacije vrti se oko strukture dokumenata i načina na koji aplikacija predstavlja odnose između podataka preko ugniježdenih dokumenata ili korištenje referenci.

Ugniježđeni modeli podataka

S MongoDB-om možete ugraditi povezane podatke u jednu strukturu ili dokument. Te su sheme općenito poznate kao de-normalizirani modeli. Razmotrite sliku 13. Ugniježđeni modeli podataka omogućuju aplikacijama pohranu srodnih podataka u isti zapis baze podataka.

Slika 13. Prikaz ugniježdenog dokumenta *(Data Modeling Introduction, n.d.)*

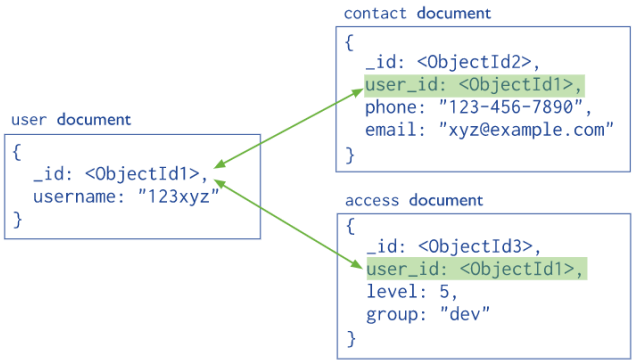
Primjer toga su ugniježdeni dokumenti koji bilježe odnose između podataka spremanjem povezanih podataka u jednu strukturu dokumenta. MongoDB dokumenti omogućuju ugrađivanje struktura dokumenata u polje ili niz unutar dokumenta. Ovi de-normalizirani modeli podataka omogućuju aplikacijama dohvaćanje povezanih podataka i upravljanje njima u jednoj operaciji baze podataka. (Data Modeling Introduction, n.d.) Kao rezultat toga, aplikacije će možda trebati kreirati manje upita i ažuriranja da bi dovršile uobičajene operacije.

Ugniježdene modele podataka koristimo kada:

- postoje odnose između entiteta.
- imate odnos jedan prema više između entiteta. U tim su odnosima dokumenti pojavljuju u odnosu „roditelj – dijete“ (eng. *parent – child relation*) (Data Modeling Introduction, n.d.)

Normalizirani modeli podataka

Normalizirani modeli podataka opisuju odnose pomoću referenci između dokumenata koje možemo vidjeti na slici 14.



Slika 14. Prikaz referenci među dokumentima *(Data Modeling Introduction, n.d.)*

Normalizirane modele podataka koristimo:

- kada bi ugnježđivanje rezultiralo dupliciranjem podataka, ali ne bi pružalo veće performanse prilikom čitanja za nadmašiti implikacije dupliciranja.
- predstavljati složenije odnose više-prema više (eng. *many-to-many*).
- modelirati velike hijerarhijske skupove podataka. (Data Modeling Introduction, n.d.)

Reference pružaju veću fleksibilnost od ugradnje. Međutim, aplikacije na strani klijenta moraju izdavati naknadne upite za rješavanje referenci. Drugim riječima, normalizirani modeli podataka mogu zahtijevati više povratnih putovanja do poslužitelja. (MongoDB, 2016). Ako uzmemo kao primjer sliku 14., ukoliko bismo željeli dobiti podatke o kontaktu ali nas zanima koje pod kojim je korisničkim imenom traženi kontakt trebat ćemo poslati 2 upita.

3.3. Vrste Podataka u MongoDB-u

MongoDB pohranjuje dokumente u format **BSON**, što je binarno kodirani format JSON-a. U osnovi, sam naziv BSON dolazi od binarno kodiranog JSON-a. JSON sadrži limitirane vrste podatke tipa: *object, array, string, numeric, boolean, null.*

BSON format podataka pruža razne vrste koje se koriste kada *JavaScript* objekte pohranjujemo u binarni oblik. Svi tipovi podataka BSON podržani su u MongoDB-u. Svaki MongoDB tip podataka odgovara jedinstvenom broju koji se koristi za njihovu identifikaciju u **$type** naredbi (MongoDB Data Types – 16 Various Data Types in MongoDB, 2018) MongoDB podržava dodatne tipove podataka, pritom zadržava esencijalnu ključ/vrijednost logiku JSON-a. Tipovi podataka koji se koriste u MongoDB-u su:

- **Null** – može reprezentirati null vrijednosti i nepostojeće polje,
- **Boolean** – koristi se za vrijednosti *true* ili *false,*
- **Number** – 64-bitni brojevi s pomičnim zarezom,
- **String** – svaki znak UTF-8 karaktera se reprezentira kao *string,*
- **Date** – datumi se spremaju u milisekundama od početka epohe tzv. UNIX epoch[5]. Vremenska zona nije spremljena u datumu,
- **Regularni izrazi** – upiti nad bazom mogu sadržavati regularne izraze koristeći *Javascript* sintaksu,
- **Array** – Lista vrijednosti reprezentirana kao *array,*
- **Ugniježđeni dokument** – dokumenti mogu sadržavati cijele dokumente ugniježđene kao vrijednosti dokumenta u kojem se nalazi i

[5] Unix epoch je broj sekundi koji je protekao od 1. siječnja 1970 minus prijestupne sekunde.

- **ObjectId** - 12-bitni znak koji se sastoji od: milisekunda od UNIX epohe, nasumično generiranog broja te brojača koji je inicijaliziran na slučajnu vrijednost. ObjectId se koristi za pohranu ID-a dokumenta. (BSON Types, n.d.)

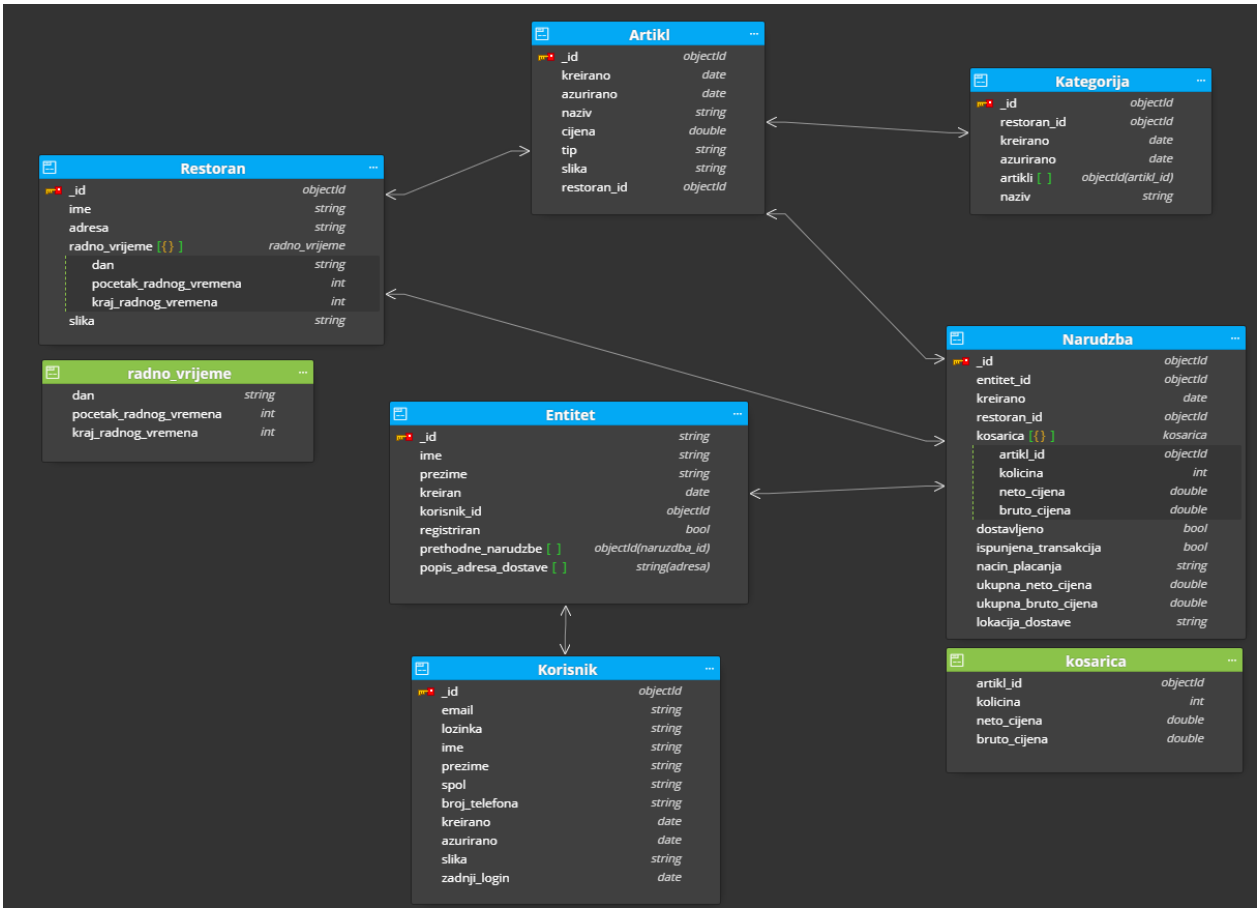
3.4. Primjer modela baze MongoDB-a

U ovom radu, kroz slijedeća potpoglavlja detaljnije će biti opisan MongoDB kroz osobno kreiranu bazu podataka. Kao primjer baze podataka izabrana je Aplikacija za narudžbu hrane tzv. „Food App“. Zadnjih nekoliko godina trend aplikacija za narudžbu hrane je eskalirao pa se kroz djelomično kreiranu bazu u nastavku detaljnije objašnjava MongoDB.

Aplikacija se sastoji od ponuda restorana u kojoj se nalaze hrana i piće (u bazi nazvan artikl) svrstani pod različitim kategorijama. Korisnik može naručiti jedan ili više artikla unutar radnog vremena jednog restorana. U aplikaciji korisnik može biti registriran (nije obavezno) jer aplikacija također prima narudžbe bez registracije. Ukoliko je korisnik registriran spremaju se njegovi podatci. Nakon što je korisnik odabrao artikle i došao do košarice, kreira se njegov entitet. Ako je registriran korisnik njegov entitet bude proširen s podatcima korisnika te prikaz njegovih prošlih narudžbi i adresa. Zadnji korak je kreiranje narudžbe u kojemu se radi suma svih artikla te spaja sa povezanim entitetom.

Baza se sastoji od 6 kolekcija:

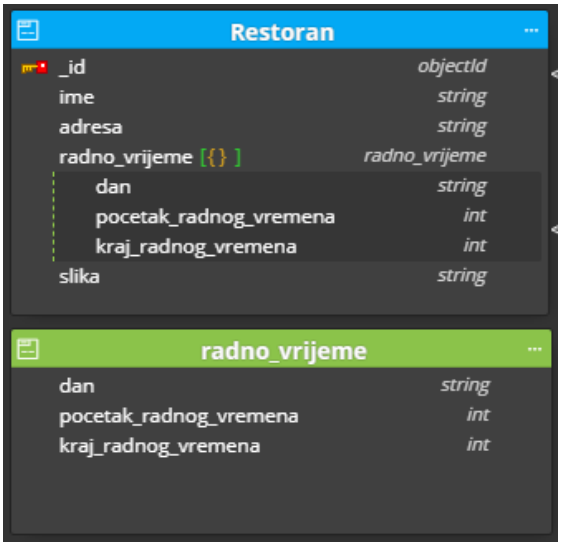
1. Restoran
2. Kategorija
3. Artikl
4. Korisnik
5. Entitet
6. Narudžba



Slika 15. Dijagram „FoodApp“ baze podataka

1. Restoran

U aplikaciji, restoran je mjesto gdje se mogu naručivati jela. Svaki restoran je različit te ga u bazi označavamo sa unikatnim **id**-em**.** Restoran se sastoji od imena, slike te adrese na kojoj posluje. Unutar radnog vremena restorana moguće je napraviti narudžbe. Restoran se sastoji od atributa: **_id**, **ime**, **adresa**, **radno_vrijeme** i **slika**. Atribut **radno_vrijeme** je ugnježđeniji dokument koji sadrži atribute: **dan**, **pocetak_radnog vremena** i **kraj_radnog_vremena.**



Slika 16. Kolekcija Restoran

2. Kategorija

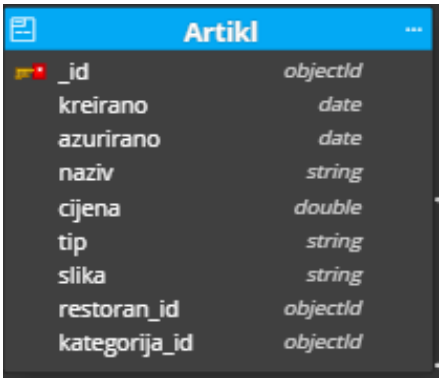
Kategorije su podijeljene po restoranima što znači da svaki restoran ima vlastite kategorije u kojoj sprema artikle. Unutar aplikacije svi artikli su podijeljeni unutar kategorija tako da krajnjem korisniku možemo lakše predočiti različite vrste artikla. Prilikom kreiranja kategorija, u bazi spremamo vrijeme kad se kreiranje izvršilo. Također se vrijeme sprema prilikom ažuriranja kategorije. Kategorija se sastoji od atributa: **_id**, **restoran_id**, **naziv**, **kreirano**, **azurirano**, **artikli.**

Slika 17. Kolekcija Kategorija

3. Artikl

Artikl je jelo ili piće restorana dostupno za narudžbu. Jedan artikl se nalazi u jednoj kategoriji restorana. Svaki artikl je spremljen u bazi sa unikatnim **id**-em. Artikl sadrži naziv, cijenu, tip (jelo ili piće), slike te **id** restorana i kategorije kojemu artikl pripada. Prilikom kreiranja artikla, u bazi spremamo vrijeme kad se kreiranje izvršilo. Također se vrijeme sprema prilikom ažuriranja artikla.

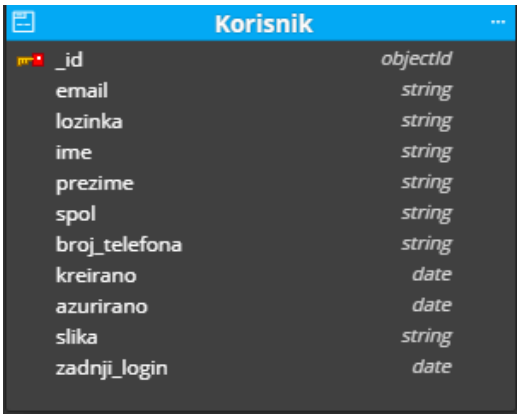
Artikl se sastoji od atributa: **_id, kreirano, azurirano, naziv, cijena, tip, slika, restoran_id i kategorija_id.**



Slika 18. Kolekcija Artikl

4. Korisnik

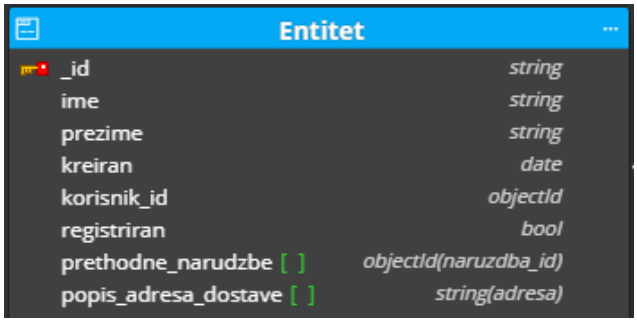
Planirano je ukoliko je korisnik registriran u aplikaciji spremati će se njegovi podatci unutar kolekcije korisnik. Kako bi korisnik bio registriran potrebni su mu e-mail i lozinka. Podatke koje će korisnik u aplikaciji unijeti su ime, prezime, spol, broj telefona te sliku. Prilikom kreiranja korisnik, u bazi spremamo vrijeme kad se kreiranje izvršilo. Također se vrijeme sprema prilikom ažuriranja korisnika te kad se korisnik zadnji put ulogirao u račun. Korisnik se sastoji od atributa: **_id, email, lozinka, ime, prezime, spol, broj_telefona, kreirano, azurirano, slika, zadnji_login**.



Slika 19. Kolekcija Korisnik

5. Entitet

Entitet je odvojen u zasebnu kolekciju jer na njemu će biti zapisani svi podatci vezani s (registriranom ili neregistriranom) korisnikom i narudžbe. U kolekciji postoji atribut **registriran** koji nam govori ako je korisnik registriran. U kolekciji zapisujemo ime korisnika ukoliko korisnik nije registriran, da znamo ime i prezime naručitelja. U bazi se spremaju podatci kad je entitet kreiran, popis bivših narudžba te popis adresa dostave. Ukoliko je korisnik registriran, sprema se referenca putem korisnikovog **id**-a. Korisnik se sastoji od atributa: **_id, ime, prezime, kreiran, korisnik_id, registriran, prethodne_naruzbe, popis_adresa_dostave**.



Slika 20. Kolekcija Entitet

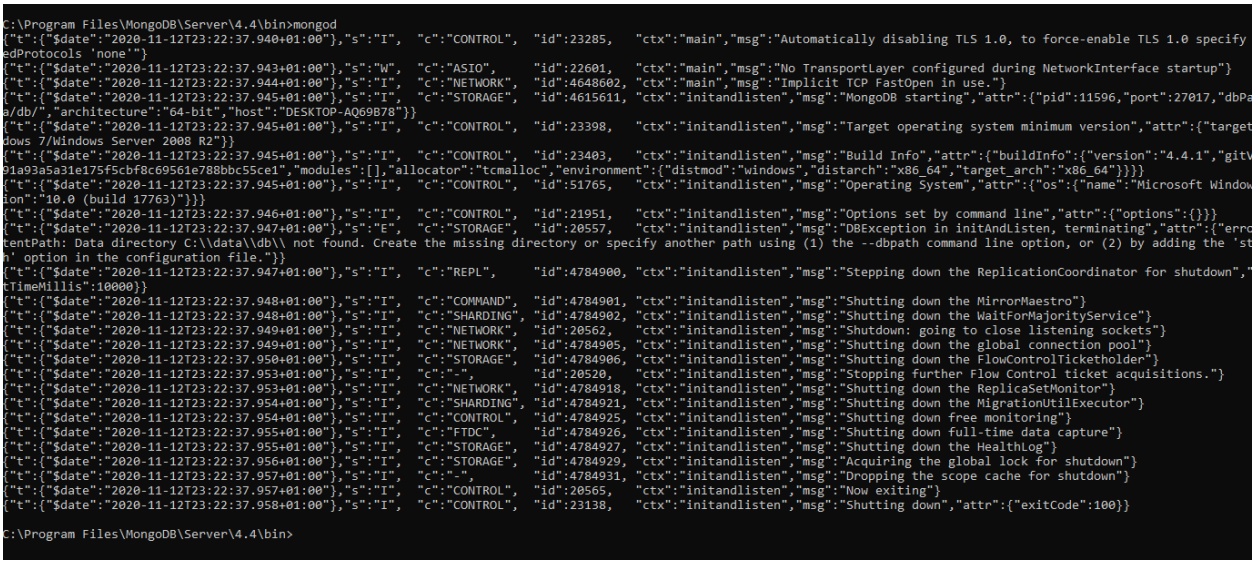
6. Narudžba

Prilikom kreiranja narudžbe kreiramo **id** narudžbe, zatim **id** entiteta koji naručuje, **id** restorana iz kojeg se naručuje te košarica u kojoj imamo podatke o naručenim artiklima. Atribut košarica je ugniježđeni dokument u kojemu imamo zapis o: **aritkl_id, kolicina, neto_cijena, bruto_cijena**. Također u narudžbu se zapisuje datum kreiranja, ukupna bruto i neto cijena, način plaćanja te da li je narudžba dostavljena. Narudžba se sastoji od atributa: **_id, entitet, restoran_id, kosarica, dostavljeno, ispunjena_transakcija, nacin_placanja, ukupna_neto_cijena, ukupna_bruto_cijena i lokacija_dostave.**

Slika 21. Kolekcija Narudžba

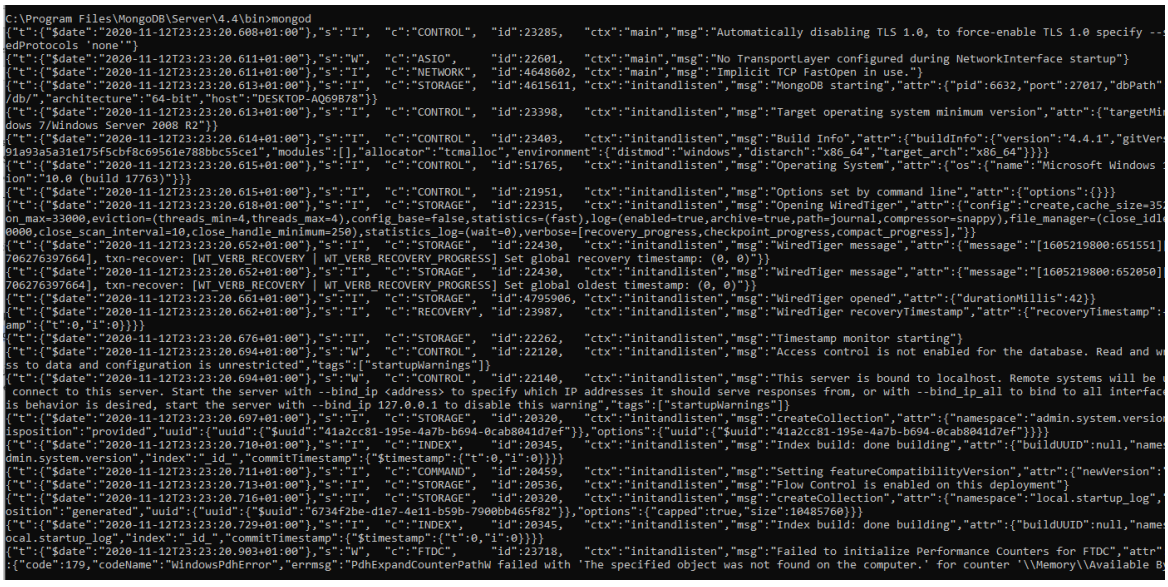
3.5. Kreiranje baze pomoću MongoDB i MongoDB Ljuske

MongoDB je uvijek pokrenut na mrežnom serveru na kojem se klijenti mogu spojiti i raditi operacije. Na stranici https://docs.mongodb.com/manual/mongo/ možete preuzeti MongoDB te vidjeti upute pri instalaciji. Verzija koja je preuzeta je 4.4.1 na Windows operacijskom sustavu. Nakon instalacije Mongo DB-a možete se spojiti na MongoDB server. Ukoliko pokrećete Mongo na Windows operacijskom sustavu potrebno se pozicionirati u direktorij MongoDB ljuske **C:\Program Files\MongoDB\Server\4.4\bin** te u komandnoj liniji upisati **mongod**. Za Linux i Mac OS dovoljno je i u kućnom direktoriju pokrenuti **mongod**.



Slika 22. Pokretanje mongod bez zadanog direktorija

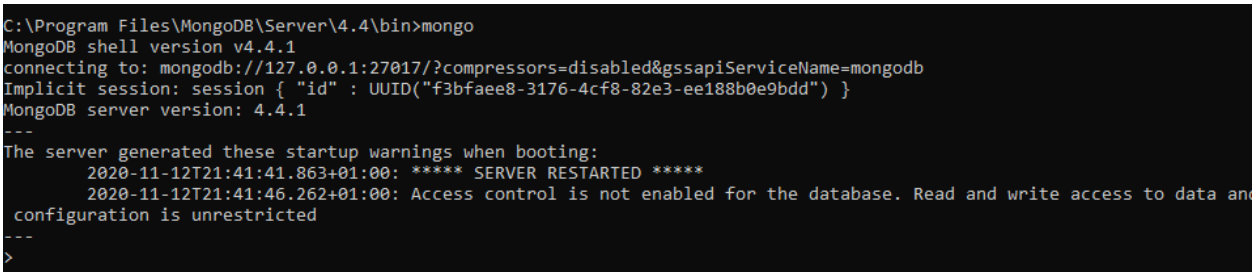
Prilikom pokretanja bez argumenta, **mongod** će koristiti zadani direktorij Linux i Mac OS **/data/db**, a Windows **\data\db**. Nakon toga potrebno je kreirati direktorij te da korisnik ima dozvole za pisanje u direktoriju prije ponovnog pokretanja MongoDB-a.



Slika 23. Pokretanje mongod sa zadanim direktorijem

Mongod postavlja *http* server koji je dostupan na portu većeg od 10000. Glavni port je na 27017. On je primarni *daemon*[6] proces za sustav MongoDB. Obrađuje zahtjeve za konekciju nad bazom, zahtjevima za podacima, upravlja pristupom podacima i obavlja operacije upravljanja pozadinom. (Shannon BradShaw, 2019)

Ako u istom direktoriju pokrenemo skriptu **mongo** otvoriti će nam se MongoDB ljuska.



Slika 24. Pokretanje MongoDB ljuske

Bitna napomena je ukoliko želimo koristiti MongoDB ljusku potrebno je imati u drugoj komandnoj liniji upaljen **mongod.**

Pri instalaciji MongoDB-a dobivamo *Javascript* ljusku pomoću koje komuniciramo s MongoDB-om iz komandne linije. Ljusku koristimo za provođenje administrativnih radnji nad bazom, praćenje pokrenute Mongo instance te drugih radnji.

Pokretanjem MongoDB-a, njegovo izvršavanje se izvodi na portu 28017 na kojemu možemo vidjeti administrativne informacije o bazi podataka. Ukoliko bismo htjeli promijeniti *port* izvođenja, primjerice izvršavanje MongoDB ljuske na portu 28015 koristili bismo naredbu:

```
mongo --port 28015
```

Također MongoDB ljuska dopušta spajanje na MongoDB instancu na udaljenom računalu (eng. *remote host machine*).

```
mongo "mongodb://baza.primjer.com:28015"
```

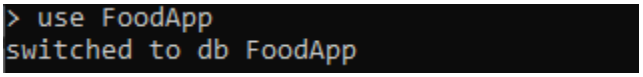
[6] Daemon je kompjuterski program koji radi kao pozadinski proces

Kreiranje baze podataka

Prilikom pokretanja, ljuska se spaja na testnu bazu podataka pod imenom test na MongoDB serveru te dodjeljuje konekciju do baze kroz varijablu **db**. Varijabla **db** služi za primarni pristup MongoDB serveru kroz ljusku. Ukoliko želimo vidjeti na kojoj smo bazi trenutno dodijeljeni, potrebno je u ljusku upisati **db.**

Slika 25. Ispis trenutne baze podataka

Za promjenu baze potrebno je upisati naredbu **use <ime baze podataka>**.

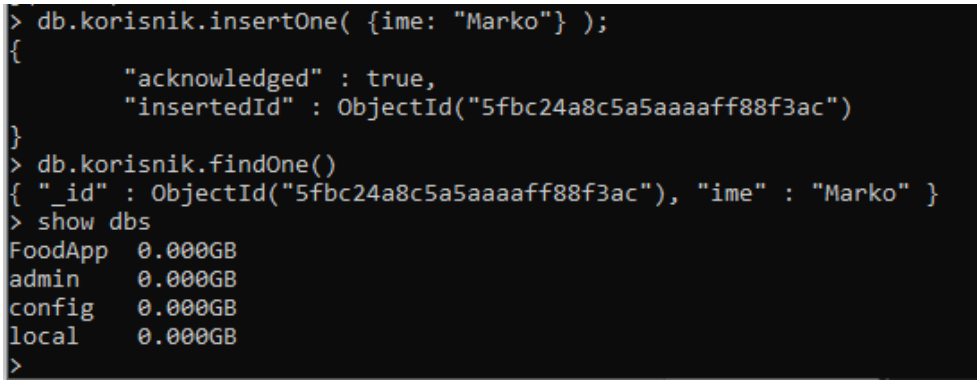


Slika 26. Pozicioniranje u trenutnu bazu podataka

Da biste ispisali sve baze podataka na MongoDB serveru, potrebno je upisati naredbu **show dbs.**

Slika 27. Prikaz svih baza podataka pomoću naredbe show dbs

Prethodno kreirana baza podataka „FoodApp“ nije prisutna na popisu. Da biste prikazali novu bazu podataka, u nju morate umetnuti barem jedan dokument. Prilikom unosa dokumenta automatski će se kreirati nova kolekcija. U MongoDB-u je zadana baza podataka *test*. Ako niste stvorili nijednu bazu podataka, tada će se kolekcije pohraniti u bazu podataka *test*.

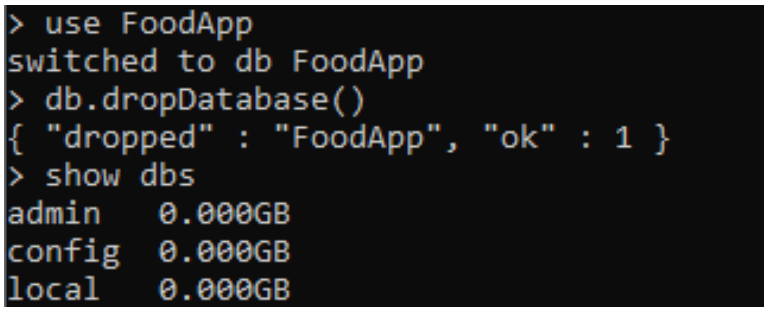


Slika 28. Unos podataka u kolekciju

Pomoću **db.korisnik.insertOne()** funkcije na trenutnoj bazi podataka („FoodApp“) kreirali kolekciju **korisnik** te unesli dokument sa imenom **„**Marko“**,** ljuska nam je vratila obavijest da je unesen dokument u kolekciju (o funkciji **insertOne()** detaljnije u poglavlju 3.7.1.). Nakon toga unesena je naredba **show dbs** koja je ispisala sve baze podataka uključujući novokreiranu bazu podataka „FoodApp“**.** Također prilikom unosa podataka u bazu podataka, npr. stvaranjem kolekcije, MongoDB kreira novu bazu podataka.

Brisanje baze podataka

Ukoliko želite izbrisati bazu, potrebno se pozicionirati u željenu bazu za brisanje te pokrenuti naredbu **db.dropDatabase()**. Na slici 29. možete vidjeti da ljuska vraća obavijest da je baza izbrisana te je nema u popisu baza na MongoDB serveru.



Slika 29. Brisanje baze podataka

Kreiranje kolekcije

Ukoliko bismo željeli kreirati kolekciju potrebno se pozicionirati u željenu bazu te unijeti **db.createCollection(ime_baze, opcije)** gdje:

- prvi parametar stavljamo ime kolekcije tipa *string*,
- drugom parametru definiramo opcije (opcionalan parametar) od kojih su:
  - **capped** – tipa *boolean* – ukoliko je vrijednost *true,* kreira se ograničena kolekcija koja je fiksne veličine. Ograničena korekcija zapisuje najprije najstarije unose sve dok ne dosegne maksimalnu veličinu.
  - **autoIndexId –** tipa *boolean* – ukoliko je vrijednost *true* kreira se indeks na polju **_id**.
  - **size –** tipa *number* - unos maksimalne veličine u bajtovima za ograničenu kolekciju.
  - **max -** tipa *number* - Određuje maksimalni broj dokumenata dopuštenih u ograničenoj kolekciji. (Shannon BradShaw, 2019)

Na slici 30. možemo vidjeti kreiranje kolekcije **korisnik**. Nakon unosa naredbe, Mongo ljuska nam je vratila status da je uspješno kreirana kolekcija. Za prikaz svih kolekcija unutar baze koristimo naredbu **show collections.**

```
> use FoodApp
switched to db FoodApp
> db.createCollection("korisnik")
{ "ok" : 1 }
> show collections
korisnik
```

Slika 30. Kreiranje kolekcije

Brisanje kolekcije

Naredba **db.ime_kolekcije.drop()** briše kolekciju iz baze. Potrebno je provjeriti ako kolekcija postoji. Ukoliko postoji možemo izbrisati kolekciju.

Na slici 31. provjeravamo listu kolekcija. Trenutno postoji samo jedna kolekcija **korisnik** te smo ju sa **db.korisnik.drop()** izbrisali. MongoDB ljuska je *true* što znači da je kolekcija izbrisana. Potom smo ispisali sve kolekcije pomoću **show collections** gdje vidimo da naša baza nema niti jednu kolekciju.

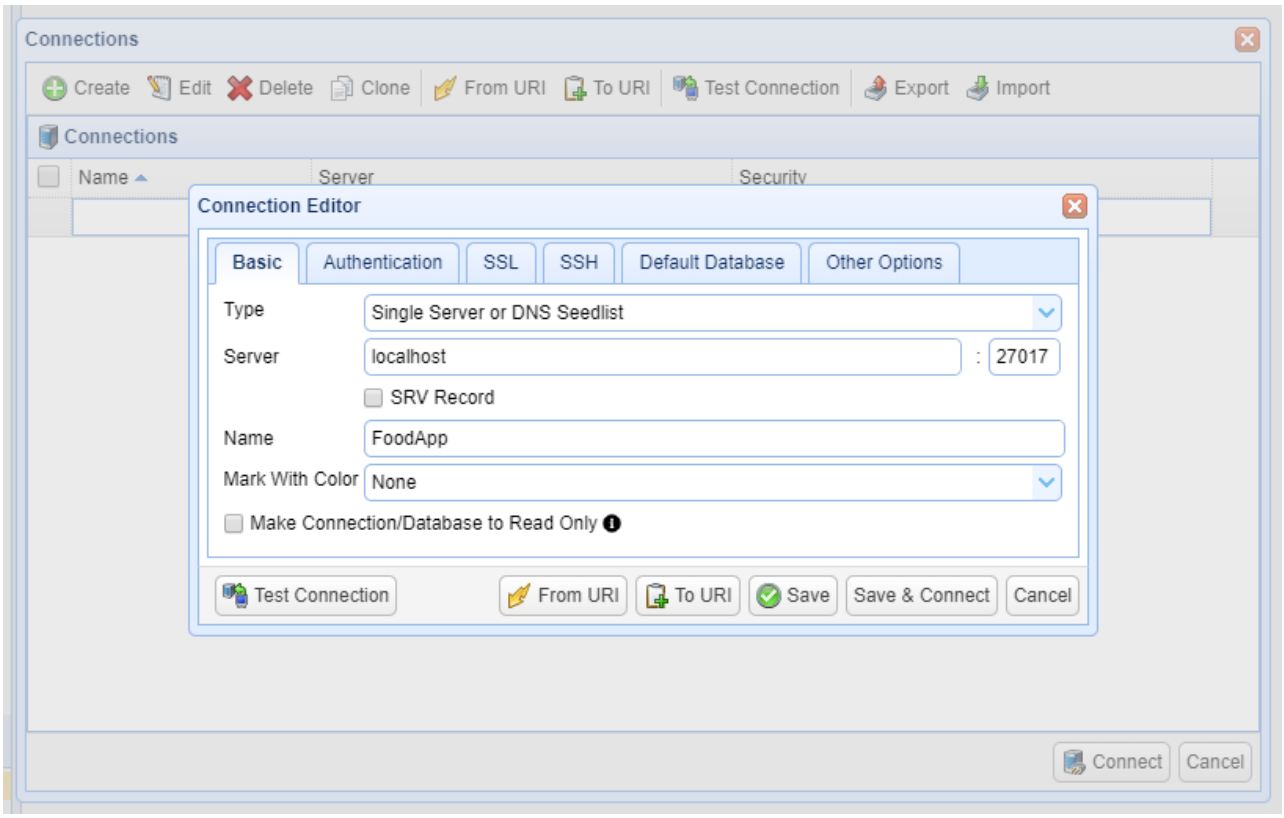
```
> show collections
korisnik
> db.korisnik.drop()
true
> show collections
>
```

Slika 31. Brisanje kolekcije

3.6. NoSQLBooster

Kroz slijedeća poglavlja biti će korišten alat **NoSQLBooster** za MongoDB radi korištenja „FoodApp“ baze podataka sa sučeljem kako bi se na lakši način prikazale radnje nad bazom te prikaz same baze. NoSQLBooster za MongoDB nudi istinsko iskustvo IntelliSense[7] koji se pojavljuje prilikom tipkanja. Ugrađena jezična usluga poznaje sva moguća dovršavanja, naredbe, svojstva, varijable, ključne riječi, čak i nazive kolekcija MongoDB, nazive polja i operatore. Također u NoSQLBooster-u moguće je spremanje skripti na kojima su rađene akcije kreiranja, brisanja, pretraživanja i brisanja što ubrzava rad nad bazom.

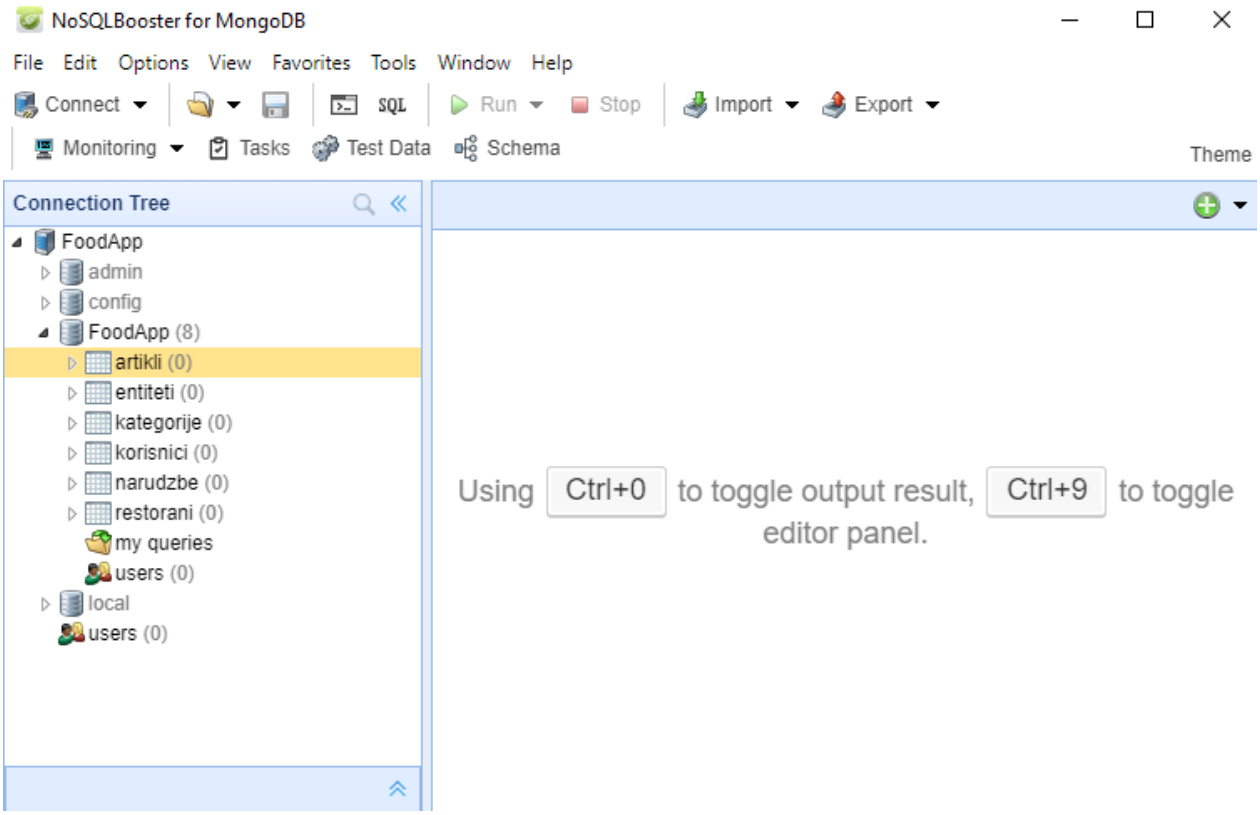
Za spajanje MongoDB baze podataka u NoSQLBooster potrebno je imati pokrenut MongoDB server. Slijedeći korak je kliknuti *Connect*, te kreirati novu konekciju sa bazom. MongoDB server radi na **localhost:27017** (ukoliko nisu promijenjene postavke unutar MongoDB servera) te ime konekcije, u ovom primjeru korišteno je „FoodApp“. Nakon što su svi podatci ispunjeni potrebno je kliknuti *Save & Connect*. Na slici 32. prikazan je prikaz spajanja MongoDB-a sa NoSQLBooster-om.



Slika 32. Postavljanje konekcije MongoDB baze podataka sa NoSQLBooster-om

[7] Intellisense je jedan od niza alata koji omogućuju inteligentno dovršavanje koda ili inteligentno dovršavanje teksta na različitim platformama.

Na slici 33. možete vidjeti prikaz uspješne konekcije sa „FoodApp“ bazom podataka te pripadajućim kolekcijama.



Slika 33. Prikaz uspješno postavljenje MongoDB baze podataka na NoSQLBooster-u

3.7. Operacije unosa, brisanja i ažuriranja dokumenta

U ovom poglavlju opisane će biti akcije unosa, brisanja i ažuriranja dokumenta nad „FoodApp“ bazom podataka.

3.7.1. Unos dokumenta

MongoDB nudi sljedeće naredbe za unos dokumenata u kolekciju:

1. **db.ime_kolekcije.insertOne(dokument)** – unos jednog dokumenta u kolekciju. Na slici 34. vidimo primjer unošenja dokumenta u restoran kolekciji. Ova naredba dodaje **_id** ključ dokumentu (ukoliko **_id** nije dodan) i sprema u bazu podataka. Nakon uspješnog unosa u

bazu, MongoDB vraća *boolean* ukoliko je dokument uspješno unesen te **_id** vrijednost unesenog dokumenta.

```
db.restorani.insertOne({
    ime : "Restoran 1",
    adresa : "Radnička 10",
    radno_vrijeme : [
        {
            "dan" : "Ponedjeljak",
            "pocetak_radnog_vremena" : 9,
            "zavrsetak_radnog_vremena" : 22,
        },
        {
            "dan" : "Utorak",
            "pocetak_radnog_vremena" : 9,
            "zavrsetak_radnog_vremena" : 22,
        },
        {
            "dan" : "Srijeda",
            "pocetak_radnog_vremena" : 9,
            "zavrsetak_radnog_vremena" : 22,
        },
        {
            "dan" : "Cetvrtak",
            "pocetak_radnog_vremena" : 9,
            "zavrsetak_radnog_vremena" : 22,
        },
        {
            "dan" : "Petak",
            "pocetak_radnog_vremena" : 9,
            "zavrsetak_radnog_vremena" : 22,
        }
    ]
})
```

Slika 34. Unos jednog dokumenta u kolekciju

2. **db.ime_kolekcije.insertMany(lista_dokumenta)** – višestruki unos dokumenata u kolekciju. Na slici 35. vidimo primjer unos različitih artikala unutar kolekcije artikli. Kao i prethodna naredba za svaki dokument se dodaje **_id** ukoliko nije definiran. Također vraća *boolean* ukoliko je lista dodana te lista naknadno dodanih **_id-**a.

```
db.artikli.insertMany([{
    kreirano: new Date(),
    azurirano: new Date(),
    naziv: "Margherita",
    cijena: 50,
    tip: "hrana",
    slika:"javno/slike/margherita.jpg",
    kategorija_id: ObjectId("5fc51827622223112c1e3fc5"),
    restoran_id: ObjectId("5fc7ac37eb25491138b89755")
},
{
    kreirano: new Date(),
    azurirano: new Date(),
    naziv: "Capriciosa",
    cijena: 60,
    tip: "hrana",
    slika:"javno/slike/capriciosa.jpg",
    kategorija_id: ObjectId("5fc51827622223112c1e3fc5"),
    restoran_id: ObjectId("5fc7ac37eb25491138b89755")
},
{
    kreirano: new Date(),
    azurirano: new Date(),
    naziv: "Slavonska",
    cijena: 65,
    tip: "hrana",
    slika:"javno/slike/slavonska.jpg",
    kategorija_id: ObjectId("5fc51827622223112c1e3fc5"),
    restoran_id: ObjectId("5fc7ac37eb25491138b89755")
},
{
    kreirano: new Date(),
    azurirano: new Date(),
    naziv: "4 sira",
    cijena: 60,
    tip: "hrana",
    slika:"javno/slike/4-sira.jpg",
    kategorija_id: ObjectId("5fc51827622223112c1e3fc5"),
    restoran_id: ObjectId("5fc7ac37eb25491138b89755")
},
])
```

Slika 35. Unos više dokumenta u kolekciju

3. **db.ime_kolekcije.insert(dokument | lista_dokumenta)** – unos jednog ili više dokumenata u kolekciju. Ukoliko je prvi parametar objekt unosi se jedan dokument, a ako je lista onda se unosi više dokumenta u kolekciju.

Također dokumenti se mogu unijeti pomoću naredba:

- **db.ime_kolekcije.update()**,
- **db.ime_kolekcije.updateOne()**,
- **db.ime_kolekcije.updateMany().**

kad je postavljen **upsert: true.** O parametru **upsert** detaljno u poglavlju 3.7.3.

3.7.2. Brisanje dokumenta

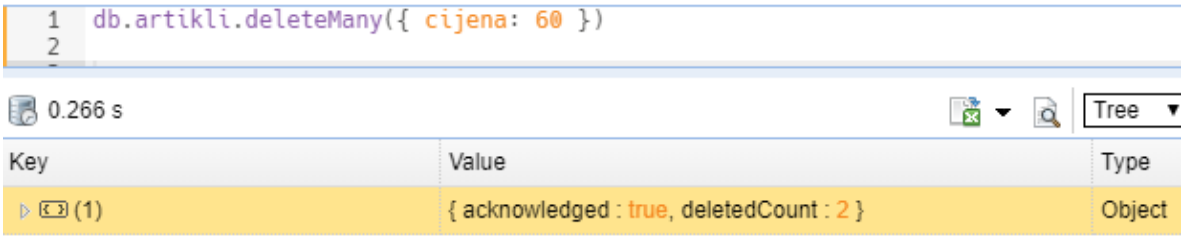
MongoDB nudi sljedeće naredbe za brisanje dokumenata u kolekciju:

1. **kolekcija.deleteOne(kriterij)** – briše prvi pronađen dokument iz kolekcije koji se podudara sa kriterijem. Pomoću kriterija pronalazimo dokument za brisanje. Na slici 36. vidimo primjer ukoliko želimo izbrisati artikl prema **id**-u iz kolekcije **artikli**.

Slika 36. Brisanje dokumenta iz kolekcije

Ukoliko smo postavili kriterij prema nazivu npr. „Margherita“ mogli bismo obrisati sve pizze „Margherita“ (iz različitih restorana) unutar baze. Stoga je najbolja praksa brisati dokumente prema **_id-u.**

2. **kolekcija.deleteMany(kriterij) –** briše sve dokumente iz kolekcije koji su se podudaraju sa danim kriterijem. Na slici 37. vidimo primjer ukoliko želimo izbrisati artikle sa cijenom 60 kn. Nakon što je pokrenuta naredba, MongoDB vraća obavijest da je brisanje uspješno te da su obrisana 2 dokumenta.



Slika 37. Brisanje više dokumenta iz kolekcije

3. **kolekcija.remove(kriterij) –** briše sve dokumente u kolekciji ako je pozvana bez parametra. Ukoliko je prisutan, kriterij briše dokumente koji su jednaki danom kriteriju.

3.7.3. Ažuriranje dokumenta

U MongoDB-u koriste se 2 načina za ažuriranje dokumenta:

1. Zamjenom dokumenta,
2. Pomoću operatora ažuriranja.

1. Zamjena dokumenta

Da bi se izvršila zamjena dokumenta, potrebno je stvaranje zamjenskog dokumenta koji se sastoji od polja i vrijednosti koje želite ažurirati te naredbe:

- **db.ime_kolekcije.replaceOne(kriterij, dokument)** - Zamjenjuje jedan dokument u kolekciji na temelju kriterija. Na slici 38. pronašli smo artikl po **_id** parametru (pizza „Istriana“) te smo ga spremili u varijablu **istriana**. Naredba **replaceOne()** vraća dokument tipa objekt pa smo pomoću sintakse **istriana.cijena** promijenili cijenu te sa **istriana.azurirano** vrijeme ažuriranja. MongoDB vraća obavijest da je ažuriranje uspješno, točnije da je pronađen dokument te da se dogodila zamjena dokumenta.

```
1  var istriana = db.artikli.findOne({ _id: ObjectId("5fc55cca622223112c1e3fcb") })
2  istriana.cijena = 90
3  istriana.azurirano = new Date
4  db.artikli.replaceOne({ _id: ObjectId("5fc55cca622223112c1e3fcb")}, istriana)
5
```

0.433 s

Key	Value	Type
(1)	{ acknowledged : true, matchedCount : 1, modifiedCount : 1 }	Object

Slika 38. Ažuriranje pomoću zamjene dokumenta

Uobičajena pogreška je podudaranje više od jednog dokumenta s kriterijima, a zatim stvaranje duplicirane vrijednosti **_id** s drugim parametrom. Baza podataka će izbaciti pogrešku zbog toga i nijedan dokument neće biti ažuriran. Stoga je preporučeno da se podudaranje radi na jednom dokumentu.

2. Operatori ažuriranja

Navedeni su neki od poznatijih operatora, dostupni za upotrebu u operacijama ažuriranja.

- Polja u dokumentu
  - **$currentDate** - Postavlja vrijednost polja na trenutni datum, bilo kao datum ili u milisekundama od UNIX epohe.
  - **$inc** - Povećava vrijednost polja za navedeni iznos.

- **$min** - polje ažurira samo ako je navedena vrijednost manja od vrijednosti postojećeg polja.
- **$max** - polje ažurira samo ako je navedena vrijednost veća od postojeće vrijednosti polja.
- **$mul** - množi vrijednost polja s navedenim iznosom.
- **$rename** - preimenuje polje.
- **$set** - postavlja vrijednost polja u dokumentu.
- **$unset** - uklanja navedeno polje iz dokumenta.

- Lista
  - **$pop** -uklanja prvu ili zadnju stavku niza.
  - **$pull** - uklanja sve elemente niza koji se podudaraju s navedenim upitom.
  - **$push** - dodaje stavku u niz.
  - **$pullAll** - uklanja sve odgovarajuće vrijednosti iz niza. (Update Operators, n.d.)

Naredbe koji koriste operatore ažuriranja:

1. **db.ime_kolekcije.updateOne(kriterij, ažuriranje)** – Ažurira najviše jedan dokument koji se podudara s navedenim kriterijem, iako se više dokumenata može podudarati s navedenim kriterijem. Na slici 39. korišten je prvi parametar **_id** za pronalazak korisnika te operator **$set** za postavljanje novog emaila. MongoDB vraća obavijest da je ažuriranje uspješno, točnije da je pronađen dokument i da se ažurirao dokument.

```
1  db.korisnici.updateOne(
2      {_id: ObjectId("5fc685d94eb5d20f188baacb")},
3      { $set : { email: "markomatijevic12345@gmail.com"}}
4  )
5
```

0.285 s

Key	Value	Type
(1)	{ acknowledged : true, matchedCount : 1, modifiedCount : 1 }	Object

Slika 39. Ažuriranje dokumenta pomoću operatora ažuriranja $set

Na slici 40. prikazan je primjer ažuriranja ugniježdenog dokumenta koji je dio liste **radno_vrijeme**. Pomoću naredbe operatora **$push** dodan je još jedan element u listu, to znači da restoran radi i subotom od 9-14 sati.

```
db.restorani.updateOne(
    { _id : ObjectId("5fc51827622223112c1e3fc5") },
        { $push : {
            radno_vrijeme: {
                dan: "Subota",
                pocetak_radnog_vremena: 9,
                kraj_radnog_vremena: 14,
            }
        }
    }
)
```

0.426 s

Key	Value	Type
▷ (1)	{ acknowledged : true, matchedCount : 1, modifiedCount : 1 }	Object

Slika 40. Ažuriranje dokumenta pomoću operatora ažuriranja $push

2. **db.ime_kolekcije.updateMany(kriterij, ažuriranje)** – Ažurira sve dokumente koje se podudaraju sa danim kriterijem. Primjerice, restoran je odlučio da za cijene veće od 60 kn daju popust od 10%. Na slici 41. vidimo da pomoću operatora **$gte** nalazimo vrijednosti veće i jednake od 60 te pronađene vrijednosti pomnožimo sa 0.9. MongoDB vraća obavijest da su pronađena 4 ažurirana dokumenta.

```
db.artikli.updateMany(
    { cijena: { $gte: 60 }},
    { $mul: { cijena: 0.9 }}
)
```

0.551 s

Key	Value	Type
▷ (1)	{ acknowledged : true, matchedCount : 4, modifiedCount : 4 }	Object

Slika 41. Ažuriranje više dokumenta pomoću updateMany()

Kao jedna od posebnih naredba ažuriranja: **db.collection.update(kriterij, ažuriranje)** je naredba u kojoj je moguće korištenje zamjena dokumenta ili korištenje modifikatora.

Upsert

Sve naredbe ažuriranja imaju treći parametar **upsert**. Kada dokument prilikom ažuriranja nije pronađen, novi dokument se kreira sa danim podatcima za ažuriranje. Ako je dokument pronađen biti će ažuriran. Na slici 42. nam MongoDB daje obavijest da nije uspješan pronalazak korisnika sa danim **_id**-em. Stoga je dodan novi dokument u kolekciju.

```
db.korisnici.updateOne(
    {_id: ObjectId("5fc685d94eb5d20f188baacf")},
        { $set : {
            ime: "Pero"
            prezime: "Perić"
            email: "pero-peric54321@gmail.com"
            lozinka: "UGVybyBwZXJpxIcgbG96asddW5rYQQW"
            spol: "muško"
            broj_telefona: "385433244422"
            kreirano: new Date()
            azurirano: new Date()
            slika: null
            zadnji_login: new Date()
        }
    }
    ,{ upsert: true }
)
```

0.285 s

Key	Value	Type
(1)	{ acknowledged : true, matchedCount : 0, modifiedCount : 0, upsertedId : ObjectId("5fc685d94eb5d20f188baacf") } (4 fields)	Object

Slika 42. Primjer dodavanja upserta prilikom korištenja naredbe updateOne()

Ako imamo bazu sa velikim brojem podataka, upit za ažuriranje će se usporiti ako ne pronađete dokument na temelju vašeg polja **_id**. MongoDB mora provjeriti da li dokument postoji sa istim **_id-**em, pa se tek onda može unesti ili ažurirati, ovisno ako dokument postoji ili ne. Stoga se preporučuje korištenje **upsert-a** sa kriterijem različitim od **_id-**a.

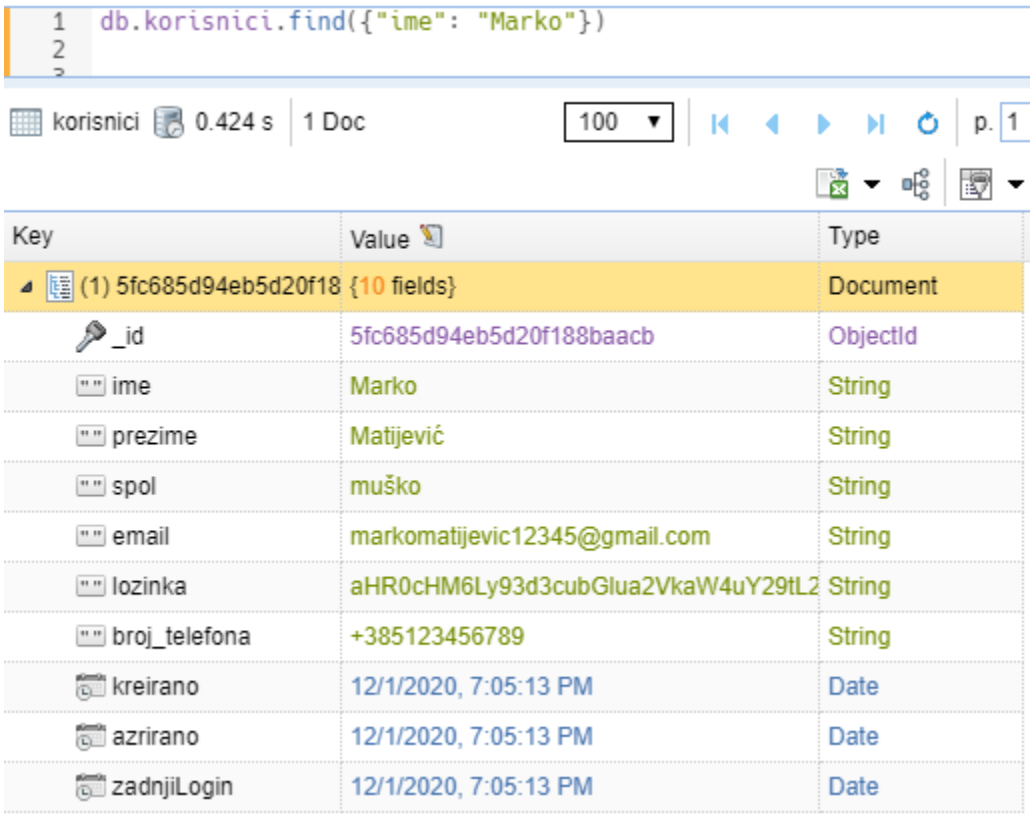
3.8. Upiti

Upiti su akcije gdje pregledavamo ili izmjenjujemo podatke koji se nalaze u bazi podataka. U MongoDB-u pomoću naredbe **find()** rade se upiti u MongoDB-u. Upiti vraćaju dokumente u kolekciji, od niti jednog dokumenta pa do cijele kolekcije. Prvi parametar naredbe **find()** je kriterij po kojim se traže dokumenti u kolekciji.

Prazan kriterij upita dokumenta npr. {} vraća sve dokumente u kolekciji. Ukoliko parametar kriterija nije dodan, **find()** naredba traži po praznim kriterijima {}.

```
db.artikli.find()
```

Ukoliko želimo pronaći korisnika sa imenom „Marko“ moguće je kao kriterij dodati kao ključ/vrijednost kao što je to napravljeno na slici 43.

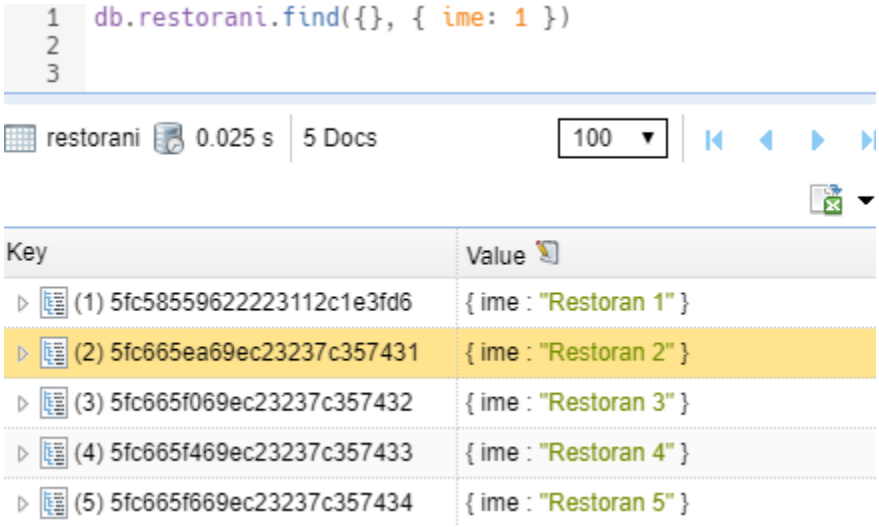


Slika 43. Upit pomoću naredbe find()

Moguće je dodavanje više uvjeta koje se interpretira kao *uvijet1* **I** *uvijet2* **I … I** *uvijetN*. Na slijedećem primjeru jer prikazan upit nad svima artiklima u koji su tipa hrana i gdje je cijena 50 kn.

```
db.artikli.find({tip: "hrana", cijena: 50 })
```

Kao drugi parametar naredbe **find** možemo postaviti koje vrijednosti upita želimo dobiti. S time možemo smanjiti vrijeme slanja podatka i izbjegnemo redundantnost podataka koje želimo pokazati prema klijentskoj strani. Na slici 44. vidimo upit nad kolekcijom restoran. Prvi parametar je prazan, što znači da prolazimo kroz sve dokumente, a kao drugi parametar postavili smo **{ ime: 1 }** kako bi nam upit vratio samo imena restorana.



Slika 44. Upit nad kolekcijom restorani prema imenima restorana

Prilikom kreiranja upita, moguće je na drugom parametru postaviti ključ/vrijednost parove koje ne želimo da nam upit vrati. Primjerice ako ne želimo da nam se prikaže broj telefona korisnika trebali bi upisati:

```
db.korisnici.find({}, { broj_telefona : 0 })
```

Upiti pomoću selektora

Upiti mogu sadržavati kompleksnije kriterije pomoću raspona vrijednost, logičkog OR uvjeta te negacije. Pri kreiranju kompleksnijih upita koristimo selektore upita. Neki od najpoznatijih selektora upita su:

- **$eq** – vraća vrijednosti koje su jednake navedenoj vrijednosti.
- **$gt** - vraća vrijednosti koje su veće od navedene vrijednosti.
- **$gte** - vraća vrijednosti koje su veće ili jednake navedenoj vrijednosti.
- **$in** - vraća bilo koju vrijednost navedenu u polju.
- **$lt** - vraća vrijednosti koje su manje od navedene vrijednosti.
- **$lte** - vraća vrijednosti koje su manje ili jednake navedenoj vrijednosti.
- **$ne** - vraća sve vrijednosti koje nisu jednake navedenoj vrijednosti.
- **$nin** - ne podudara se ni s jednom vrijednošću navedenom u polju i pridružuje se drugim uvjetima upita sa logičkim „I“ te vraća sve dokumente koji odgovaraju uvjetima upita.
- **$not** – vraća dokumente koji se ne podudaraju s uvjetom upita.
- **$or** - pridružuje upita s logičkim ILI i vraća sve dokumente koji odgovaraju uvjetima upita. (Query and Projection Operators, n.d.)

Primjerice ukoliko želimo pronaći korisnike koji su kreirani prije 02.12.2020. možemo napisati:

```
const datum = new Date("12/02/2020")
db.korisnci.find({ kreirano: { $lt: datum }})
```

Napomena je da su u MongoDB-u datumi zapisani tipa mjesec/dan/godina.

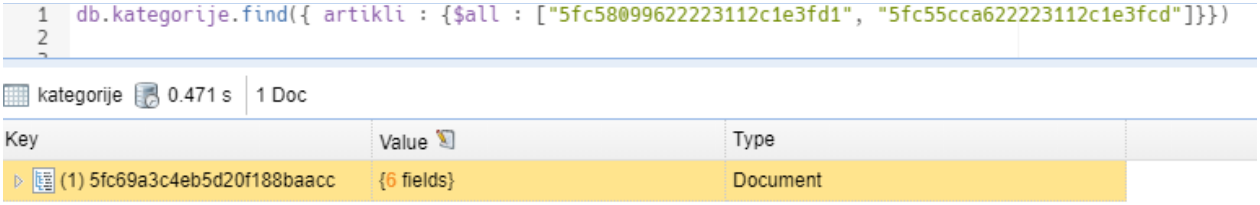
Ukoliko želimo kreirati upite da zadovoljava jedan od navedenih upita koristiti ćemo **$or** selektora. Primjerice ukoliko želimo ispisati kategorije „pizze“ ili „salate“ možemo napisati:

```
db.kategorije.find({$or : [{ naziv : "Pizze" }, { naziv: "Salate" }]})
```

Upiti nad elementima liste

Jedne od najkorištenijih operatora radi upita nad elementima liste su:

- **$all** – ukoliko se barem jedan element iz liste podudara sa navedenom vrijednošću, upit vraća sve elemente iz te liste. Na slici 45. vidimo primjer ako želimo vratiti popis svih artikla u kategoriji gledajući primjerice dva artikla, možemo koristiti:



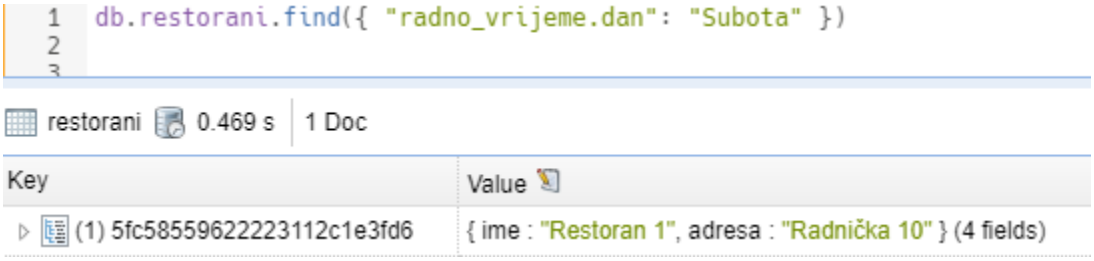
Slika 45. Upit nad elementima liste pomoću operatora $all

- **$size** – daje opciju upita nad listom prema danoj veličini liste. Primjerice odlučeno da će se nagraditi korisnike ukoliko imaju 9 nagrada. Pomoću naredbe **$size** pronaći ćemo korisnike koji imaju 9 narudžba.

```
db.entiteti.find({ prethodne_narudzbe : { $size : 9 }})
```

Upiti nad ugniježdenim dokumentima

Dokument **Restoran** sadrži ugniježdeni dokument **radno_vrijeme.** Za pristupanje elementima ugniježdenog dokumenta koristimo znak točke. Ukoliko želimo vidjeti koji restoran radi subotom upisat ćemo kao što je to na slici 46. prikazano.



Slika 46. Upit nad ugniježđenim dokumentom

Dodatne opcije upita

Najkorištenije opcije upita su limitiranje rezultata, preskok broj rezultata i sortiranje. Sve navedene operacije moraju biti dodane u upit prije nego što se šalje na bazu.

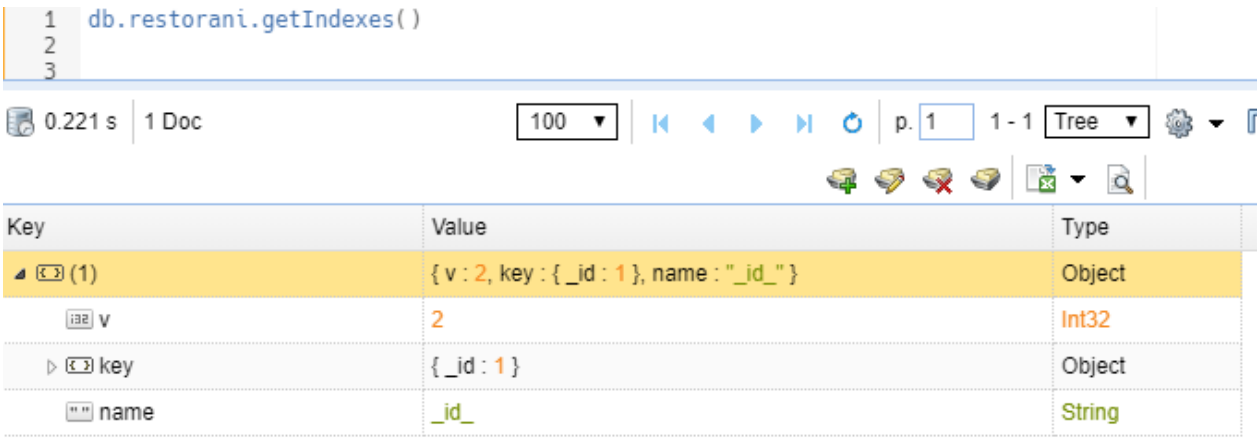
- **limit(n)** – vraća maksimalni broj rezultata ovisno o danom broju. Ukoliko je manji broj rezultata od danog broja limita, vratiti će podatke koje podudaraju određeni kriterij. Limit postavlja samo gornju granicu broja rezultata.
- **skip(n)** – radi preskok prva **n** elemenata i vraća ostatak rezultata. Ukoliko je broj dokumenta manji od broja **n**, naredba vraća prazno polje.
- **sort(1 | -1)** - naredba prihvaća dokument koji sadrži popis polja zajedno s redoslijedom sortiranja. Za određivanje redoslijeda sortiranja koriste se **1** i **-1**. **1** se koristi za uzlazni poredak, dok se **-1** koristi za silazni poredak.

Sve gore navedene naredbe mogu se koristiti u paginaciji. Ako na klijentskoj strani želimo prikazati sve artikle, a u bazi ih imamo primjerice 1000. Na svakom pozivu prema bazi morali bi vraćati svih 1000 artikala što je jako skup proces s obzirom na performanse baze i vrijeme upita. Tada paginacija dolazi kao rješenje gdje prikazujemo primjerice 50 artikla po stranici. Kad korisnik klikne da dohvati slijedeću stranicu artikla, radi se ponovni upit gdje se preskače prvih 50 artikala i vraća slijedećih 50 artikla, čime se opterećenje nad bazom i vrijeme upita smanjuje. Na slijedećem primjeru možemo vidjeti implementaciju paginacije i sortiranja rezultata.

```
db.arikli.find().limit(50).skip(50).sort({ naziv : 1 })
```

3.9. Indeksiranje

Indeksi poboljšavaju vrijeme izvođenja upita. Bez indeksa, MongoDB mora izvršiti skeniranje cijele kolekcije (eng. *collection scan*) tj. skenirati svaki dokument u kolekciji, kako bi odabrali dokumente koji odgovaraju kriterijima upita. Ako za upit postoji odgovarajući indeks, MongoDB može koristiti indeks da ograniči broj dokumenata koje mora pregledati. Svaka kolekcija ima zadani unikatni indeks, a to je **_id** indeks koji je {**_id**: 1}. To znači da su vrijednosti u indeksu **_id** spremljeni u uzlaznom poretku.



Slika 47. Prikaz _id indeksa

Na Slici 47. vidimo prikaz **_id** indeksa pomoću naredbe **db.ime_kolekcije.getIndexes()**. Ime ovog indeksa je **_id_**, što znači da nije moguće staviti ime indeksa sa istim **_id**-em jer MongoDB blokira tu operaciju. Također nije moguće brisanje **_id** indeksa za svaku kolekciju jer se on postavlja kao zadani indeks.

Indeksi spremanju vrijednosti polja u posebnom poretku. Kreiranje indeksa radi se pomoću naredbe **db.ime_kolekcije.createIndex(ključ : [-1 | 1], opcije)**. Svaki indeks se kreira po ključu. Ključ se sastoji od para ključ/vrijednost, gdje je ključ ime polja, a vrijednost 1 ili -1 (uzlazni ili silazni poredak). Drugi parametar indeksa ima slijedeće opcije:

- **background (**tipa *boolean)* – Ako želite napraviti na velikoj kolekciji indeks i ako se kolekcija aktiva i ne želite prekinuti procese na kolekciji, možete postaviti opciju da se kreiranje indeksa radi u pozadini.
- **unique (**tipa *boolean)* - kreirati će unikatni indeks, što znači da će ovaj indeks biti unikatan za cijelu kolekciju.
- **name** (tipa *string*) – daje novo generiranom indeksu navedeno ime.

```
1  db.restorani.createIndex({ ime: 1 })
2
3
4
```

0.424 s

Key	Value	Type
(1)	{ createdCollectic	Object
createdCollectionAutomatically	false	Bool
numIndexesBefore	1	Int32
numIndexesAfter	2	Int32
ok	1	Int32

Slika 48. Kreiranje indeksa

Na slici 48. kreiran je indeks prema imenu restorana u uzlaznom poretku. MongoDB vraća da je kreiran indeks na broju 2, broj indeksa prije novokreiranog indeksa je 1 (**_id** indeks) te da je akcija uspješno kreirana.

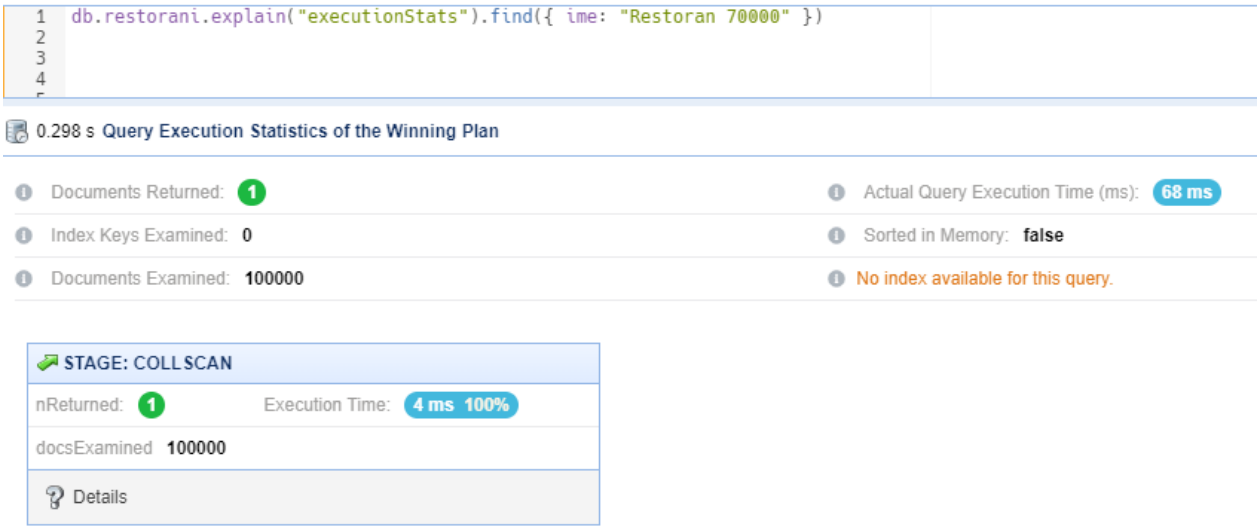
Prilikom kreiranja indeksa MongoDB unutar kolekcije uzima polje **ime** iz svih kolekcija i poreda u ulaznom poretku. Umjesto da se pregledavaju dokumenti, MongoDB pretražuje po indeksu i kad je vrijednost pronađena, dobijemo pokazivač prema dokumentu, što znači da možemo brzo pristupiti dokumentima. Kad se indeks kreira sprema se s podatcima u kolekciji na MongoDB serveru.

Kako bi dokazali poboljšanje performansi pomoću indeksa u kolekciju restorani dodali smo unos od 100 000 restorana pomoću funkcije **kreirajRestorane()**, kako je to prikazano na slici 49.

```
function kreirajRestorane() {
    const restorani = [];
    for (let i = 0; i < 100000; i++) {
        restorani.push({
            ime : `Restoran ${i+1}`,
            adresa : `Radnička ${i+1}`,
            radno_vrijeme : [
                {
                    "dan" : "Ponedjeljak",
                    "pocetak_radnog_vremena" : 9,
                    "zavrsetak_radnog_vremena" : 22,
                },
                {
                    "dan" : "Utorak",
                    "pocetak_radnog_vremena" : 9,
                    "zavrsetak_radnog_vremena" : 22,
                },
                {
                    "dan" : "Srijeda",
                    "pocetak_radnog_vremena" : 9,
                    "zavrsetak_radnog_vremena" : 22,
                },
                {
                    "dan" : "Cetvrtak",
                    "pocetak_radnog_vremena" : 9,
                    "zavrsetak_radnog_vremena" : 22,
                },
                {
                    "dan" : "Petak",
                    "pocetak_radnog_vremena" : 9,
                    "zavrsetak_radnog_vremena" : 22,
                }
            ],
        });
    }

    return restorani
}

const kreiraniRestorani = kreirajRestorane()

db.restorani.insertMany(kreiraniRestorani)
```

Slika 49. Funkcija za unos 100 000 restorana

Zatim smo koristili naredbu **explain("executionstats").** Na slici 50. vidimo prikaz upita koji nam vraća objekt prilikom izvršavanja upita sa odgovorom da je napravljeno skeniranje na 100000 dokumenta.



Slika 50. Prikaz izvođenja upita bez indeksa

Potom je pokrenut upit sa kreiranim indeksom na polju **ime** te na slici 51, vidimo prikaz izvođenja upita sa indeksom. U JSON prikazu rezultata vidimo polje **docsExamined**, što znači da je skeniranje napravljeno na 1 dokumentu.

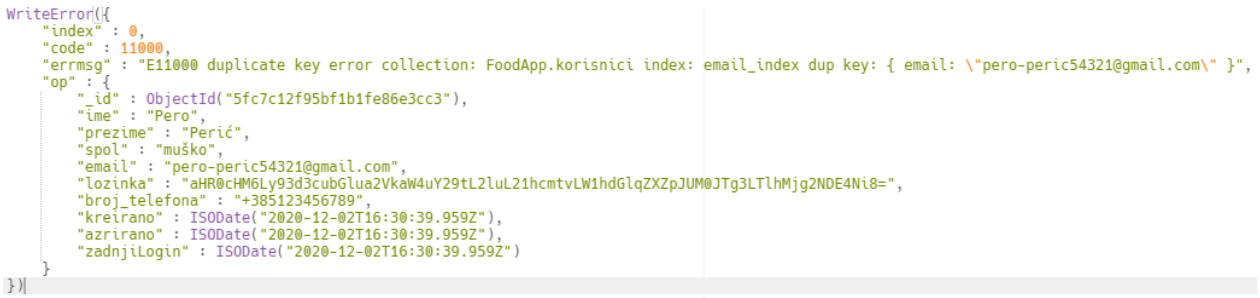
Slika 51. Prikaz izvođenja upita pomoću indeksa nad poljem ime

Kreiranje unikatnog indeksa

Ukoliko želimo kreirati unikatni indeks na određenom polju dokumenta, prilikom novog unosa dokumenta, provjeravati će ako postoji ta vrijednost u bazi. Ukoliko postoji, MongoDB će blokirati unos dokumenta. U dokumentu **korisnik** postoji polje **email**. Dva korisnika ne mogu imati isti email račun stoga koristimo unikatni indeks kako bi spriječili kreiranje dvostrukih email adresa kao što je prikazano na slijedećem primjeru.

```
db.korisnici.createIndex(
        { email: 1},
        { unique: true, name: "email_index"}
)
```

Pomoću gore navedene naredbe kreiramo unikatni **email_index** prema uzlaznom poretku. Na slici 52. vidimo prilikom unosa novog korisnika sa postojećim emailom u bazi, MongoDB nam vraća grešku i odbija dodati dokument u bazu.



Slika 52. Greška prilikom kreiranja duplicirane vrijednosti sa indeksom

Napomena, ukoliko kolekcija već sadrži dupliciranu vrijednost na polju gdje želite kreirati unikatni indeks, MongoDB vraća grešku prilikom kreiranja unikatnog indeksa te vraća popis dupliciranih vrijednosti.

Brisanje indeksa

Za brisanje indeksa koriste se dvije naredbe:

1. **db.ime_kolekcije.dropIndex(ime_indeksa)** – briše indeks u kolekciji po imenu indeksa.
2. **db.ime_kolekcije.dropIndexes()** – briše sve indekse unutar kolekcije osim zadanog **_id** indeksa.

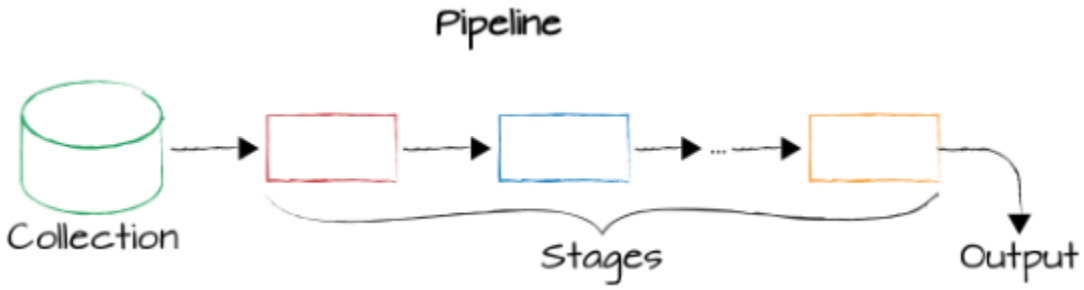
3.10. Agregacija

Nakon što su podatci spremljeni u MongoDB, možda želite učiniti više od samog dohvaćanja podataka. Ovo potpoglavlje govori o operacijama agregacije koje MongoDB pruža, a to su:

- Agregacija pomoću cjevovoda i
- Osnovne naredbe za agregaciju.

Agregacija pomoću cjevovoda

Agregacija dopušta transformaciju i spajanje dokumenta u kolekciju. Agregacija se temelji na konceptu cjevovoda. Pomoću cjevovoda za agregaciju uzimamo ulaz iz MongoDB kolekcije i dodajemo dokumente iz kolekcije koji prolaze jedan ili više faza, od kojih svaka faza izvršava različite operacije na ulazu (slika 53.). Svaka faza uzima kao ulaz bilo koju fazu prije nego što je proizvedena kao izlaz. Ulazi i izlazi za sve faze su dokumenti.

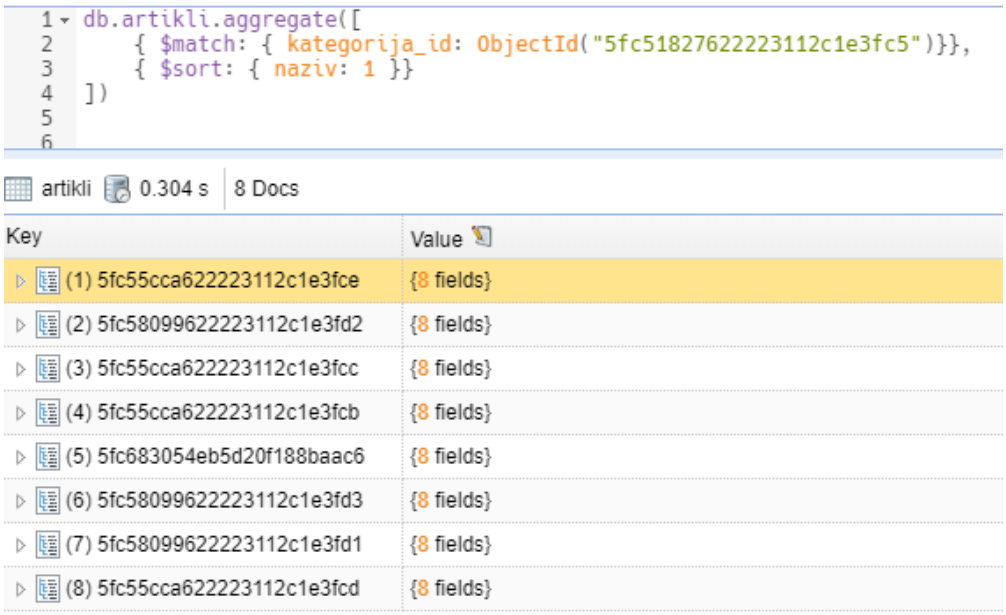


Slika 53. Agregacijski cjevovod u MongoDB-u *(Shannon BradShaw, 2019)*

Za korištenje agregacije koristi se naredba: **db.ime_kolekcije.aggregate (operacija_agreagcije).** Neki od poznatijih operacija agregacije su:

- **$match** - Filtrira dokumente za prosljeđivanje samo dokumenata koji odgovaraju navedenim uvjetima u sljedeću fazu cjevovoda.
- **$project -** Dodaje dokumentima navedena polja te ih šalje u sljedeću fazu u cjevovodu. Navedena polja mogu biti postojeća polja iz ulaznih dokumenata ili novo uračunata polja.
- **$group -** Grupira ulazne dokumente prema navedenom **_id** izrazu i za svako posebno grupiranje daje dokument. Polje **_id** svakog izlaznog dokumenta sadrži jedinstvenu vrijednost grupe. Izlazni dokumenti također mogu sadržavati izračunata polja koja sadrže vrijednosti nekog izraza akumulacije (suma, minimum, maksimum i sl.).
- **$sort –** Sortira sve ulazne dokumente i vraća ih u cjevovod poredanim redoslijedom.
- **$skip –** Preskače navedeni broj dokumenata koji se unesu u fazu i prosljeđuje preostale dokumente u sljedeću fazu cjevovoda.
- **$limit –** Ograničava broj dokumenata proslijeđenih u sljedeću fazu cjevovoda.

Na slici 54. napravljen je primjer korištenja agregacije za prikaz artikla iz jedne kategorije, te su rezultati sortirani uzlazno.



Slika 54. Primjer korištenja agregacije sa naredbama $match i $sort

Kao sljedeći primjer korištenja agregacije koristili smo dodavanje artikla u košaricu. Ako pogledamo sliku 55. vidimo da je prvo kreiran prazan dokument narudžbe. Zatim smo u varijablama spremili **_id**-eve od restorana, entiteta, kategorije i narudžbe. Pomoću metode **aggregate** i operacije **$match** ispisali smo sve artikle koji se nalaze u jednoj kategoriji restorana te pomoću **toArray()** rezultati su postavljeni u listu. Zatim je napravljena provjera u *if* petlji ako postoji artikl, koristeći *Javascript* metodu **for of**, prolazimo kroz artikle te u svakoj iteraciji radimo ažuriranje narudžbe pomoću **updateOne** gdje koristimo:

- **$set** za postavljanje vrijednosti u narudžbi,
- **$push** za dodavanje artikla u košarici te,
- **$inc** za povećavanje ukupne bruto i neto vrijednosti.

```
db.narudzbe.insertOne({
    entitet_id: "",
    kreirano: new Date(),
    restoran_id: "",
    kosarica: [],
    dostavljeno: false,
    ispunjena_transakcija: false,
    nacin_placanja: "",
    ukupna_neto_cijena: 0,
    ukupna_bruto_cijena: 0,
    lokacija_dostave: "",
})

const restoran_id = ObjectId("5fc7ac37eb25491138b89755");
const kategorija_id = ObjectId("5fc69a3c4eb5d20f188baacc");
const naruzdba_id = ObjectId("5fc91cfee43836216cf3e94d");
const entitet_id = ObjectId("5fc8415095bf1b1fe86e3cc6");

const artikli = db.artikli.aggregate([{$match: { restoran_id }, $match: { kategorija_id }}]).toArray();

if (artikli && artikli.length) {
    for (let artikl of artikli) {
        const bruto_cijena = artikl.cijena * 1.25;
        db.narudzbe.updateOne({ _id: naruzdba_id }, {
            $set: {
                entitet_id,
                restoran_id,
            },
            $push: {
                kosarica: {
                    artikl_id: artikl._id,
                    kolicina: 1,
                    neto_cijena: artikl.cijena,
                    bruto_cijena,
                }
            },
            $inc: {
                ukupna_neto_cijena: artikl.cijena,
                ukupna_bruto_cijena: bruto_cijena,
            }
        })
    }
}
```

Slika 55. Kreiranje narudžbe pomoću agregacije

Na slici 56. vidimo dokument kao rezultat prethodne akcije kreiranja narudžbe.

(1) 5fc91cfee43836216cf3e94d	{11 fields}	Document
_id	5fc91cfee43836216cf3e94d	ObjectId
entitet_id	5fc8415095bf1b1fe86e3cc6	ObjectId
kreirano	12/3/2020, 6:14:38 PM	Date
restoran_id	5fc7ac37eb25491138b89755	ObjectId
kosarica	Array[4]	Array
0	{ artikl_id : ObjectId("5fc8469195bf1b1fe86e3cc7"), kolicina : 1, neto_cijena : 50, bruto_cijena : 62.5 } (4 fields)	Object
1	{ artikl_id : ObjectId("5fc8469195bf1b1fe86e3cc8"), kolicina : 1, neto_cijena : 60, bruto_cijena : 75 } (4 fields)	Object
2	{ artikl_id : ObjectId("5fc8469195bf1b1fe86e3cc9"), kolicina : 1, neto_cijena : 65, bruto_cijena : 81.25 } (4 fields)	Object
3	{ artikl_id : ObjectId("5fc8469195bf1b1fe86e3cca"), kolicina : 1, neto_cijena : 60, bruto_cijena : 75 } (4 fields)	Object
dostavljeno	false	Bool
ispunjena_transakcija	false	Bool
nacin_placanja		String
ukupna_neto_cijena	235	Double
ukupna_bruto_cijena	293.75	Double
lokacija_dostave		String

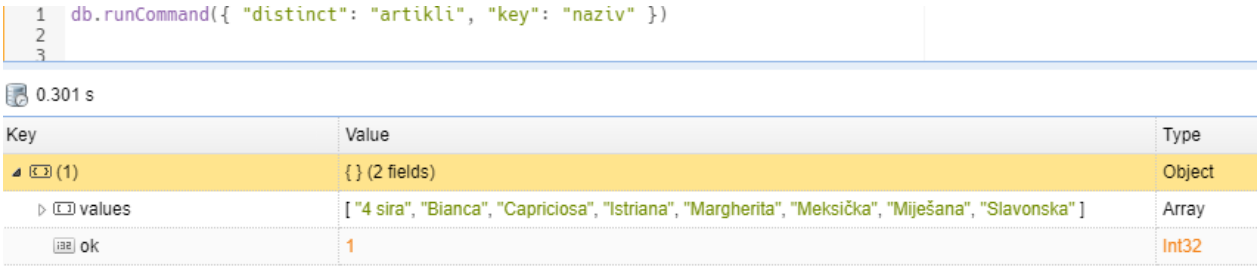
Slika 56. Prikaz dokumenta nakon dodavanja artikala u košaricu

Osnovne naredbe za agregaciju

Postoji nekoliko naredbi koje MongoDB nudi za osnovnu agregaciju kolekcije. Te naredbe dodane su prije agregacijskog cjevovoda stoga su velikom slučaju manje korištenije. Neke od najpoznatijih naredbi su:

- **db.ime_kolekcije.count() –** vraća broj dokumenta unutar kolekcije,
- **db.ime_kolekcije.distinct() -** pronalazi različite vrijednosti za određeno polje u jednoj kolekciji i vraća rezultate u polju.

Na slici 57. vidimo primjer korištenja **distinct()** naredbe koju koristimo pomoću **db.runCommand(naredba)** naredbe koja nam služi za izvršavanje zadanih naredbi baze podataka. Prvi parametar **runCommand-a** bila je naredba **distinct** nad kolekcijom artikli, zatim postavljen je ključ **naziv** po kojem dobivamo listu svih naziva artikla.



Slika 57. Korištenje naredbe distinct

Zaključak

NoSQL baze podataka kreirane su iz razloga kako bi se riješili glavni problemi kao što su skalabilnost i fleksibilnost koje su SQL baze podataka imali. Vertikalno skaliranje nije bilo rješenje jer su se povećali troškovi radi dolaska sve većeg broja podataka.

2009. godine Eric Evans i Johan Oskarson prvi su koristili opis nerelacijskih baza podataka kako bi naglasili činjenicu da se podatci mogu pohranjivati u bazu podataka u formatu različitom od relacijskih tablica. NoSQL baze pokazale su se kao učinkovite za rad na distribuiranim sustavima koji se brzo horizontalno skaliraju te rješavaju problem skaliranja, fleksibilnosti, brzi razvoj, dostupnost podataka bez utjecaja kvara hardvera i sl.

Danas, postoje nekoliko različitih vrsta NoSQL baza podataka, no u ovom radu je opisana samo jedna baza podataka, a to je MongoDB. Podatci u MongoDB-u prikazani su u obliku JSON-a, tzv. „dokumenata“. MongoDB koristi fleksibilni podatkovni model, znači da ne postoji unaprijed definirana shema, a dokument može sadržavati bilo koji skup vrijednosti na temelju bilo kojeg ključa. Uz dobro dokumentiranu dokumentaciju, osnovna instalacija i postavljanje baze za rad je brz proces. Izrada upita jedna je od prednosti MongoDB-a. Jezik upita MongoDB-a vrlo je izražajan i lak je za razumijevanje. MongoDB pohranjuje većinu izvodljivih podataka u RAM memoriji. Svi se podaci zadržavaju na tvrdom disku, ali tijekom upita ne dohvaćaju podatke s tvrdog diska, nego iz lokalnog RAM-a i prema tome može poslužiti podatke mnogo brže. Također je skalabilan te rad nad podatcima moguć na više poslužitelja. Neki od nedostataka MongoDB-a su: kreiranje modela gdje dokument ima velikih broj podobjekata što otežava pretraživanja i sortiranja, kreiranje indeksa. Za spajanje dva dokumenta potrebno je kreiranje više upita. Loše implementirani indeksi mogu rezultirati sa sporom bazom podataka. Moguće je postojanje dupliciranih podataka jer MongoDB ne podržava dobro definirane relacije.

U ovom radu MongoDB je bio korišten kao baza podataka za model dostave jela i pića restorana. MongoDB nema unaprijed definiranu shemu, ali jako je bitno kreirati dobar model kako bi se baza podataka lako mogla skalirati. Neke od dobrih praksa pri radu s MongoDB-om su: kreiranje UML dijagrama za lakši rad sa nestrukturiranim podatcima, spremanje svih podataka za zapis u jedan dokument, izbjegavanje kreiranja velikih dokumenata te nepotrebno duga imena polja i korištenje indeksa za kreiranje bržih upita.

Nerelacijske baze podataka imaju svoje snage i slabosti, kao i relacijske baze podataka. Puno je odluka koje treba donijeti kada razmišljate o svojoj pohrani podataka. Jedna od najvažnijih odluka je hoće li se kao primarna baza podataka koristiti SQL ili NoSQL baza podataka. Morati ćete razmisliti o tome kako izgledaju vaši podaci, kako ćete postavljati upite za podatke i skalabilnost koja će vam trebati u budućnosti. SQL baze podataka pružaju velike pogodnosti za transakcijske podatke čija se struktura ne mijenja često (ili se uopće ne mijenja) i gdje je cjelovitost podataka najvažnija. NoSQL baze podataka pružaju mnogo veću fleksibilnost i skalabilnost, što omogućuje brz razvoj i ponavljanje.

Popis literature

Christine, S. (16. Siječanj 2020). *What is NoSQL and is it the next big trend in databases?* Dohvaćeno iz https://www.tutorialspoint.com/what-is-nosql-and-is-it-the-next-big-trend-in-databases

Foote, K. D. (18. Lipanj 2018). *A Brief History of Non-Relational Databases*. Dohvaćeno iz Dataversity: https://www.dataversity.net/a-brief-history-of-non-relational-databases/#

Fowler, M. (2013). Dohvaćeno iz https://martinfowler.com/articles/schemaless/#schemaless-insert

Fowler, M. (2013). Dohvaćeno iz https://martinfowler.com/articles/schemaless/#implicit-schema

Kaplarevic, V. (17. Lipanj 2020). *What is NoSQL database*. Dohvaćeno iz https://phoenixnap.com/kb/what-is-a-nosql-database?fbclid=IwAR0mPAf5Ce2FAAKxE6M8kaBirYi_oMWXlfrWPJaELRpqkLlO2IFSj8gY8nY

Kaplarevic, V. (17. Lipanj 2020). *What is NoSQL database*. Dohvaćeno iz phoenixnap.com: https://phoenixnap.com/kb/wp-content/uploads/2020/05/column-based-nosql-family.png

Kulkarni, C. (n.d.). *NoSql Databases - Overview*. Dohvaćeno iz coreviewsystems.com: https://coreviewsystems.com/analysis-of-nosql-databases-part-1/

Lim, Z. H. (28. Travanj 2020). *How to Scale SQL and NoSQL Databases*. Dohvaćeno iz Medium: https://miro.medium.com/max/700/1*8P7rhC6zIIF3iCoF06McwQ.png

Lim, Z. H. (28. Travanj 2020). *How to Scale SQL and NoSQL Databases*. Dohvaćeno iz Medium: https://miro.medium.com/max/700/1*ryBTJ5J1wD-CEIdvhriAuw.png

Maximillian, S. (2018). Dohvaćeno iz https://d33wubrfki0l68.cloudfront.net/d3ed3e6fe7aace15cac669bc1aaf26afd2969088/a69e6/static/c6c8b088e9d9dd4722a965cde6b76e0d/4b190/sql-schema.jpg

Maximillian, S. (2018). Dohvaćeno iz https://d33wubrfki0l68.cloudfront.net/bdf99b183760347cf85e29a5435ae59ce996ca1c/563ea/static/986e1a479cf96ceec8a23a0386111fe9/4b190/nosql-no-schema.jpg

Maximillian, S. (2018). Dohvaćeno iz https://academind.com/static/5df24f0f34a3d98feb531b5fc7776f72/a2510/sql-relations.jpg

MongoDB Data Types – 16 Various Data Types in MongoDB. (5. Lipanj 2018). Dohvaćeno iz https://data-flair.training/: https://data-flair.training/blogs/mongodb-data-types/

Mullins, C. S. (15. Veljača 2018). *What Do We Mean by Database Scalability?* Dohvaćeno iz dzone.com: https://dzone.com/articles/what-do-we-mean-by-database-scalability

NoSQL Tutorial: Types of NoSQL Databases, What is & Example. (n.d.). Dohvaćeno iz guru99.com: https://www.guru99.com/nosql-tutorial.html

NoSQL Tutorial: Types of NoSQL Databases, What is & Example. (n.d.). Dohvaćeno iz guru99.com: https://www.guru99.com/images/1/101818_0537_NoSQLTutori9.png

NoSQL Tutorial: Types of NoSQL Databases, What is & Example. (n.d.). Dohvaćeno iz guru99.com: https://www.guru99.com/images/1/101818_0537_NoSQLTutori10.png

NoSQL Tutorial: Types of NoSQL Databases, What is & Example. (n.d.). Dohvaćeno iz guru99.com: https://www.guru99.com/images/1/101818_0537_NoSQLTutori8.png

Pore, A. (13. Travanj 2018). *How to Choose the Right NoSQL Database for Your Application?* Dohvaćeno iz dataversity.net: https://www.dataversity.net/choose-right-nosql-database-application/

Schwarzmüller, M. (25. 7 2018). *SQL vs NoSQL*. Dohvaćeno iz https://academind.com/learn/web-dev/sql-vs-nosql/

What is a Non-Relational Database? (n.d.). Dohvaćeno iz MongoDb: https://www.mongodb.com/non-relational-database

What is NoSQL? (20. Lipanj 2016). Dohvaćeno iz database.guide: https://database.guide/what-is-nosql/

Popis slika

Popis tablica