UNIVERZITET U NIŠU

ELEKTRONSKI FAKULTET

**Fizičko projektovanje MongoDB baze podataka i optimizacija podataka**

Seminarski rad iz predmeta

Sistemi za upravljanje bazama podataka

Student: Mentor:

Marta Đorđević, 1490 prof. dr Aleksandar Stanimirović

Niš, 2023. godina

Sadržaj

[Uvod 3](#_Toc132765284)

[NoSQL 4](#_Toc132765285)

[Tipovi NoSQL baza podataka 5](#_Toc132765286)

[Key/Value store 6](#_Toc132765287)

[Wide column store 7](#_Toc132765288)

[Graph store 7](#_Toc132765289)

[Document store 8](#_Toc132765290)

[Razlike između SQL-a i NoSQL-a 10](#_Toc132765291)

[MongoDB 12](#_Toc132765292)

[Pokretanje MongoDB-a 13](#_Toc132765293)

[Struktura MongoDB baze podataka 16](#_Toc132765294)

[Optimizacija podataka 21](#_Toc132765295)

[Indeksi 21](#_Toc132765296)

[GridFS 23](#_Toc132765297)

[Mehanizam replikacije 25](#_Toc132765298)

[Sharding 27](#_Toc132765299)

[Map/Reduce 28](#_Toc132765300)

[Agregacija 29](#_Toc132765301)

[Zaključak 30](#_Toc132765302)

[Literatura 31](#_Toc132765303)

# Uvod

MongoDB je popularna document-oriented NoSQL baza podataka čije su glavne odlike to što pruža skalabilnost, fleksibilnost i performanse. Fizičko projektovanje MongoDB baze podataka je važan proces koji direktno utiče na performanse sistema. Posebno treba obratiti pažnju na veličinu dokumenta kao i količinu podataka koji se čitaju ili pišu.

Optimizacija podataka je jako važan deo fizičkog projektovanja i odnosi se na razne tehnike kako bi se ubrzao rad i smanjili troškovi rada. Optimalan dizajn šeme, korišćenje odgovarajućih indeksa i prikladnih tehnika za pisanje upita samo su neke od stvari koje mogu poboljšati performanse.

U ovom radu biće pre svega opisane NoSQL baze podaka, a nakon toga pažnja će biti posvećena MongoDB-u, njegovoj strukturi i objasniti fizičko projektovanje ove baze.

Nakon toga, biće reči o tehnikama koje se primenjuju u optimizaciji podataka.

# NoSQL

NoSQL (Not Only SQL) baze podataka su tip sistema za upravljanje bazom podataka koji ne koristi tradicionalni SQL (Structured Query Language) za skladištenje i preuzimanje podataka. Obezbeđuju mehanizam koji je modelovan na način koji nije tabelarni odnos koji se koristi u relacionim bazama podataka. Nastao je krajem 1960-ih godina, ali je naziv dobio početkom 21. veka. Dizajnirane su tako da rukuju velikim količinama nestruktuiranih ili polustruktuiranih podataka, kojima je teško upravljati tradicionalnim relacionim bazama podataka. Umesto tipične tabelarne strukture relacione baze podataka, NoSQL baze podataka sadrže podatke unutar jedne strukture podataka. Takođe, NoSQL je tip distribuirane baze podataka, gde se informacije kopiraju i čuvaju na različitim serverima, koji mogu biti udaljeni i lokalni, što obezbeđuje dostupnost i pouzdanost podataka. Ako neki od podataka ode sa mreže, ostatak baze podataka može nastaviti neometano sa radom. Dizajnirane su tako da se mogu skalirati na horizontalan način (dodavanje novih čvorova u bazu podataka kako bi se lakše obradili podaci).

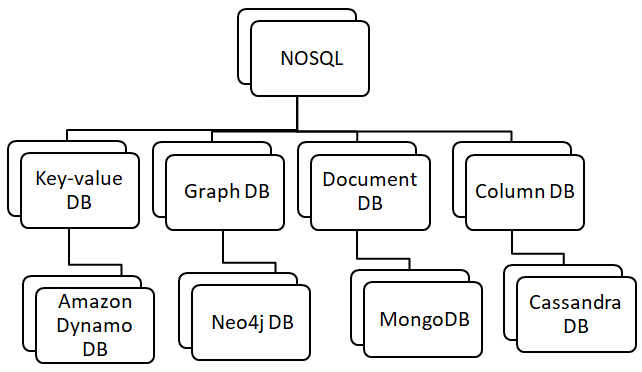
NoSQL baze podataka su napravljenje sa namerom da odgovaraju specifičnim modelima podataka i imaju fleksibilne šeme za izgradnju modernih aplikacija. Poznate su po lakoći razvoja, funkcionalnosti i performansama u velikom obimu. Koriste razne modele podataka za pristup podacima i upravljanje njima. Posebno su optimizovane za aplikacije koje zahtevaju veliki obim podataka, nisko kašnjenje i fleksibilne modele podataka. Prednosti korišćenja NoSQL baza podataka su:

* Fleksibilnost – NoSQL baze podataka pružaju fleksibilne šeme omogućavajući brži i iterativniji razvoj, što ih čini idealnim za polustruktuirane i nestruktuirane podatke. Horizontalno skaliranje i fleksibilni model podataka takođe znače da NoSQL baze podataka mogu da adresiraju velike količine podataka koji se brzo menjaju, što ih čini odličnim za agilan razvoj i brze iteracije.
* Skalabilnost – NoSQL baze podataka su generalno dizajnirane da skaliraju korišćenjem distribuiranih klastera hardvera umesto povećanja dodavanjem skupih i robusnih servera.
* Isplativost – NoSQL baze podataka su često isplativije od tradicionalnih relacionih baza podataka, jer su uglavnom manje složene i ne zahtevaju veliki skup hardvera i softvera.
* Performanse – kako su dizajnirane da rukuju velikim količinama podataka i saobraćaja, mogu ponuditi poboljšane performanse u poređenju sa relacionim bazama podataka.
* Pojednostavljuju razvoj aplikacija, posebno one koje koriste REST API i Web usluge.
* Visoka dostupnost – mogućnost automatske replikacije čini ih veoma dostupnim jer se u slučaju bilo kakve greške ili gubitka podaci repliciraju u prethodno konzistentno stanje.
* Brzina – omogućavaju brže skladištenje i obradu za sve korisnike, što ih čini boljim za moderne, složene Web ili mobilne aplikacije.
* Visoko funkcionalne – NoSQL baze podataka pružaju visoko funkcionalne API-je i tipove podataka koji su namenski napravljeni za svaki od njihovih modela podataka.

Takođe, NoSQL baze podataka poseduju i sledeće nedostatke:

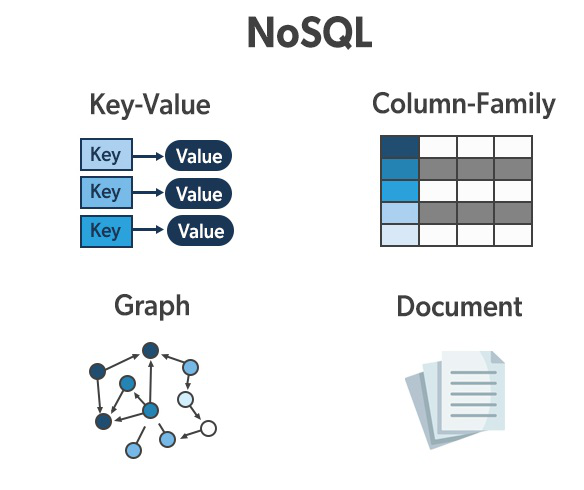
* Nedostatak standardizacije – s obzirom na to da NoSQL baze podataka nisu standardizovane kao SQL baze podataka, to može dovesti do problema u integraciji podataka.
* Nedostatak podrške za složene upite – nisu dizajnirane za rukovanje složenim uputima, što znači da nisu pogodne za aplikacije koje zahtevaju složenu analizu podataka.
* Kako su NoSQL baze podataka novije, ne postoje sveobuhvatni industrijski standardi kao kod relacionih baza podataka.
* Jedan od najvećih izazova NoSQL baza podataka je održavanje konzistentnosti podataka. U tradicionalnim bazama podataka, konzistentnost se održava transakcijama, što znači da se sve promene u bazi moraju izvršiti u okviru transakcije. NoSQL baze podataka često ne podržavaju transakcije, pa se događaji u bazi obično izvršavaju kao pojedinačni događaji što podatke neretko čini nedostupnim.

## Tipovi NoSQL baza podataka



Slika 1 – Tipovi NoSQL baza podataka i njihovi predstavnici

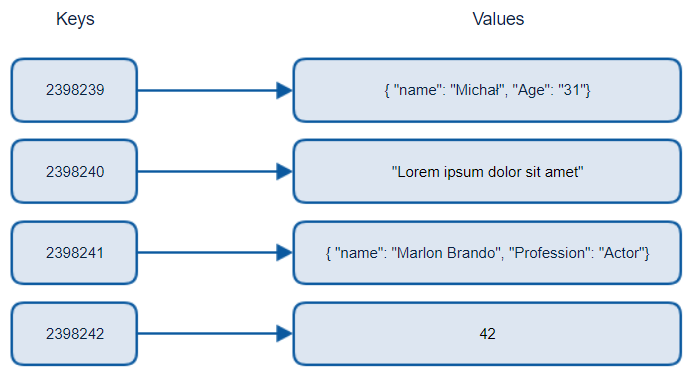
Postoje četiri popularna tipa NoSQL baza podataka, gde svaki od njih koristi drugačiji tip modela podataka, što rezultira značajnim razlikama između svakog tipa.



Slika 2 – Tipovi NoSQL baza podataka

### Key/Value store

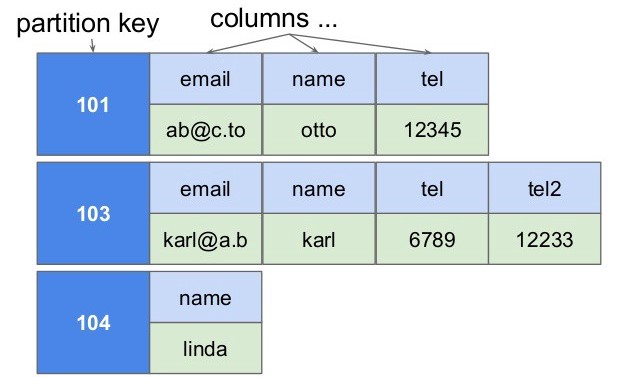
Takođe poznate i kao baze podataka ključ/vrednost, ovi sistemi predstavljaju jednostavan model podataka koji uparuje jedinstveni ključ sa pridruženom vrednošću. Organizovan je kao rečnik parova ključ/vrednost, gde svaka stavka ima ključ i vrednost. Svaka vrednost se čuva pod odgovarajućim ključem i kada je potrebno pristupiti određenoj vrednosti, potrebno je navesti njen ključ. Ovaj tip baza podataka je dobro koristiti za keširanje i upravljanje sesijama u Web aplikaciji. Kako je ovaj model prilično jednostavan, može se korisiti za razvoj visoko skalabilnih i efikasnih aplikacija. Ovaj model je jednostavan i brz za čitanje i upisivanje podataka, pretraga po ključu je efikasna i štedi se memorija tako što se prikazuju samo atributi čije vrednosti postoje. Međutim, ovaj model nije dobro koristiti za kompleksnije zahteve i upite koji uključuju složenije strukture i relacije između podataka. Neki od najpoznatijih predstavnika ovog tipa NoSQL baza podataka su Redis i DynamoDB.



Slika 3 – Primer key/value modela podataka

### Wide column store

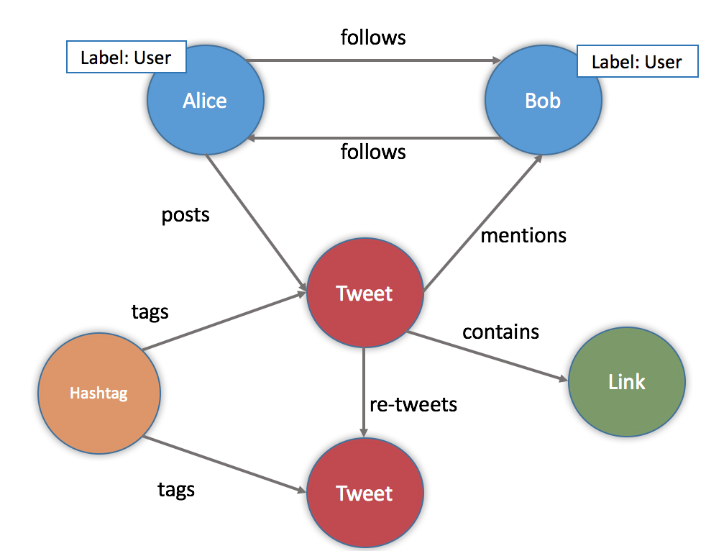
Ovaj tip baza podataka čuva informacije u kolonama, omogućavajući korisnicima da pristupe samo određenim kolonama koje su im potrebne bez dodeljivanja dodatne memorije na nerelevantne i nepotrebne podatke. Koriste se tabele, kolone i vrste poput relacionih baza podataka, ali je razlika u tome što se imena kolona mogu razlikovati od redova u jednoj tabeli, gde se svaka kolona čuva zasebno na disku. Poznati predstavnik ovog tipa je Cassandra.



Slika 4 – Primer wide column modela podataka

### Graph store

Obično sadrže podatke iz grafa znanj a, elementi podataka se čuvaju kao čvorovi, ivice i svojstva. Čvorovi se koriste za reprezentaciju entiteta, svojstva su podaci koji se upisuju u čvorove, a ivice predstavljaju relacije između čvorova. Dobro je rešenje za velike količine podataka jer ne zahtevaju skupe operacije spoja. Zahvaljujući fleksibilnoj šemi pogodne su za razvoj aplikacija koje brzo evaluiraju. Predstavnik ovog tipa je Neo4j.



Slika 5 – Primer graf modela podataka

### Document store

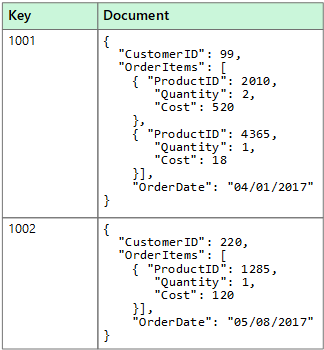
Document baze podataka predstavljaju jedan od tipova NoSQL baza podataka kompjuterski softver čija je osnovna namena čuvanje, pribavljanje i upravljanje document-oriented ili polustruktuiranim podacima, dok se podaci obično čuvaju u JSON, XML ili BSON formatima. One podatke skladište u dokumentima koji se čuvaju u kolekcijama. Svaki dokument se sastoji od parova ključ/vrednost, gde vrednost predstavlja sam dokument. Dokumenti se u bazi podataka adresiraju preko jedinstvenog ključa, koji predstavlja taj dokument. Ključ predstavlja identifikator (ID) koji je uglavnom string, URI ili putanja i može se koristiti za preuzimanje dokumenata iz baze podataka, dok je u nekim slučajevima ključ potreban za kreiranje ili umetanje dokumenata u bazu podataka. Osim jednostavnog traženja, ovaj tip baze podataka nudi API ili jezik upita koji omogućava korisnicima da preuzmu dokumente na osnovu sadržaja. Skup API-ja ili dostupnih funkcija jezika upita značajno varira u zavisnosti od implementacije.

Ovaj tip baza podataka uglavno obezbeđuje neki mehanizam za ažuriranje ili uređivanje sadržaja dokumenta, zamenom celog dokumenta ili pojedinačnih struktuiranih delova dokumenta. Za organizaciju dokumenata koriste se sledeće tehnike:

* Kolekcije – ovo je najčešći oblik organizacije, u nju se smeštaju grupe dokumenata koji su slični ili istog tipa, jedan dokument se može nalaziti u jednoj ili više kolekcija.
* Mehanizam tagova – dokumentima se dodeljuju jedan ili više tagova i oni su na osnovu toga organizovani.
* Hijerarhije direktorijuma – grupe dokumenata su organizovane u strukturu nalik stablu, odgovara organizaciji file sistema.
* Nevidljivi metapodaci – pridružuju se vidljivim podacima (npr. datum kreiranja, poslednja izmena...).

Osim što se dokumentima može pristupiti korišćenjem ključeva, većina document baza podataka ima API ili upitni jezik koji omogućava pretraživanje dokumenata i izvršenje CRUD operacija. Iako ne postoji standardni API, većina baza obezbeđuje sopstvenu implementaciju.

* Kreiranje (Create) – dokumenti se mogu kreirati i svaki dokument ima jedisntveni identifikator.
* Čitanje (Read) – dokumenti se mogu čitati iz baze podataka, omogućeno je pretraživanje dokumenata koristeći identifikatore ili vrednosti polja. Kako bi se povećala efikasnost pretraživanja, moguće je dodati indekse.
* Ažuriranje (Update) – postojeći dokumenti se mogu ažurirati u celosti ili delimično.
* Brisanje (Delete) – moguće je brisanje dokumenata iz baze podataka.



Slika 6 – Primer document-oriented baze podataka

Dobre strane document store baza podataka:

* U današnje vreme programeri smatraju da je rad sa podacima u dokumentima dosta lakši i intuitivniji od rada sa podacima iz relacionih baza podataka koji su smešteni u tabelama. Dokumenti se mapiraju u strukture podataka i programeri ne moraju da vode računa o tome.
* Fleksibilnost šeme – svaki dokument može imati drugačiju strukturu, pa se podaci mogu dodavati u dokumente bez menjanja cele šeme baze podataka. Na ovaj način dodavanje je brže i jednostavnije.
* Document store baze podataka su distribuirane i omogućavaju horizontalno skaliranje.
* Prisutnost JSON dokumenata – JSON je postao ustaljeni standard za razmenu i čuvanje podataka, oni su laki za razumevanje i nezavisni od jezika.
* Korišćenjem indeksa dokumenti se mogu efikasno pretražiti po ključnim rečima ili nekim drugim kriterijumima, što omogućava brzo i lako pronalaženje potrebnih podataka.

Pored navedenih prednosti, postoje i mane:

* Nedostatak standardizacije – koriste različite formate za čuvanje podataka što može dovesti do problema pri razmeni podataka između aplikacija koje koriste različite baze podataka.
* Mogu biti manje efikasne u radu sa podacima koji zahtevaju kompleksne agregacije, kao što su relacijski podaci.
* Nema podrške za ACID transakcije.

Document baze podataka su dobre za upotrebu u situacijma gde se podaci prirodno grupišu u dokumente kao što su društvene mreže, e-prodavnice ili IoT sistemi. Takođe, mogu biti korisne u sistemima gde je potrebna obrada velike količine podataka i kod dinamičkih podataka, gde korisnici objekte mogu menjati i prilagođavati sopstvenim potrebama. Glavni predstavnik ovog tipa je MongoDB o kome će u daljem tekstu biti više reči.

## Razlike između SQL-a i NoSQL-a

Pre relacionih baza podataka, kompanije su koristile hijerarhijski sistem za baze podataka sa strukturom nalik stablu kao tabele podataka. Ovakvi sistemi za upravljanje bazama podataka omogućili su korisnicima da organizuju velike količine podataka, međutim bili su dosta složeni i ograničeni. Zbog ovih ograničenja dovela su do toga da se razvije novi sistem za upravljanje bazama podataka – SQL. On je obezbedio interfejs za interakciju sa relacionim bazama podataka, koje podatke uređuju u tabelama. Nakon nekog vremena, zahtevi u vezi sa brzinom i većom količinom podataka postajali su sve veći i značajniji i programerima je trebalo nešto fleksibilnije od SQL baza podataka. Rešenje za to postao je NoSQL.

NoSQL je predstavljao alternativu za SQL, ali ovaj napredak nije mogao zameniti SQL baze podataka. Na visokom nivou SQL baze podataka su opšte namene, dok su NoSQL baze podataka projektovane za specifične slučajeve upotrebe. Glavne razlike između SQL-a i NoSQL-a mogu se sumirati u sledeće četiri kategorije:

* Model podataka – dok se kod SQL baza podataka podaci modeluju kao tabele sa fiksnim redovima i kolonama, kod NoSQL-a to nije slučaj. Umesto toga, u zavisnosti od baze, podaci se mogu modelovati kao JSON dokumenti, grafovi sa čvorovima i ivicama ili parovi ključ-vrednost.
* Šema – šema za NoSQL baze podataka je fleksibilna, što znači da nema fiksne strukture podataka, tipove podataka i dužine elemenata podataka. Podaci se čuvaju u slobodnom obliku. Ovakav pristup programerima nudi veću fleksibilnost, što im može dosta olakšati u radu. Kod SQL-a šema je fiksna, sa tačno utvrđenim tipovima podataka i dužinama za svaku kolonu, dok svaki red mora odgovarati definisanom izgledu i strukturi kolone.
* Prilagodljivost – NoSQL baze podataka uglavnom primenjuju horizontalno skaliranje (odnosi se na dodavanje dodatnih čvorova u NoSQL bazi podataka kako bi se povećala skalabilnost i kapacitet, postiže se dodavanjem novih čvorova koji rade zajedno i dele opterećenje baze), a SQL obično vertikalno skaliranje (odnosi se na dodavanje hardverskih resursa pojedinačnom čvoru u bazi kako bi se povećao kapacitet baze podataka)
* Integritet podataka – SQL i NoSQL koriste različite pristupe za zaštitu integriteta podataka dok se kreiraju, čitaju, ažuriraju i brišu aplikacije i korisnici. Većina NoSQL baza podataka upravlja integritetom podataka poznatim kao BASE (Basically Available, Soft State, Eventual consistent) tako da podaci mogu biti nekonzistentni tokom određenog vremenskog perioda, ali replikacija baze podataka na kraju ažurira sve kopije kako bi bile konzistentne. Pristup koji koristi SQL je ACID, gde svaki od njegova četiri obeležja (atomičnost, konzistentnost, izolovanost i trajnost) doprinose sposobnost transakcije da osigura integritet podataka. Koristeći ACID, svaka transakcija kada se izvrši sama u konzistentnom stanju će se ili izvršiti dajući ispravne rezultate ili prekinuti bez ikakvog efekta.

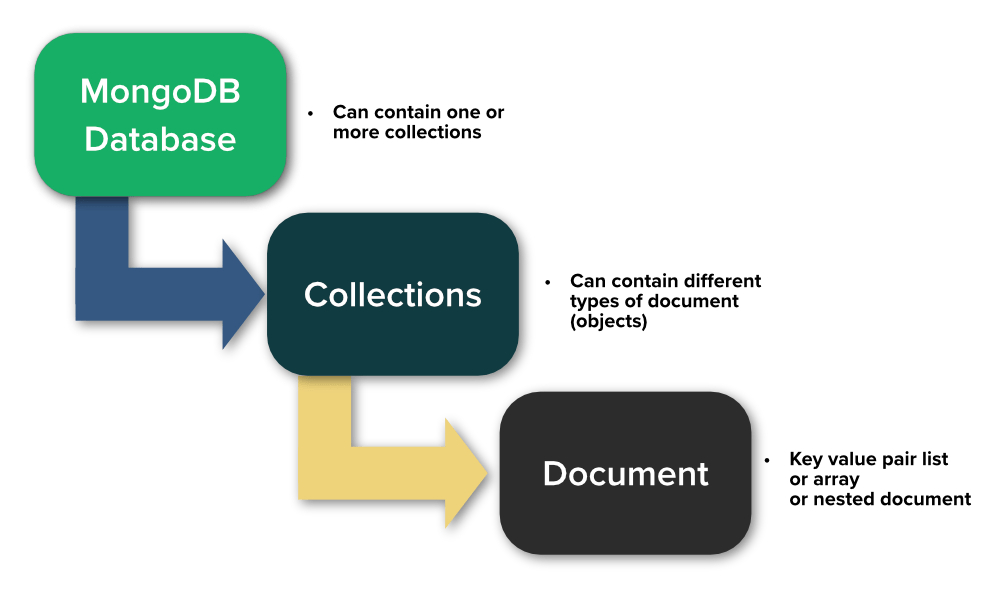
Iako svaki tip baze ima svoje prednosti i mane, kompanije obično koriste i relacione i NoSQL baze podataka u jednoj aplikaciji. Današnji dobavljači cloud-a mogu podržavati SQL i NoSQL baze podataka, a koja baza će se koristiti zavisi od ciljeva.

|  |  |  |
| --- | --- | --- |
|  | NoSQL | SQL |
| Model podataka | Različit, u zavisnoti od tipa NoSQL baze podataka | Tabele sa fiksiranim vrstama i kolonama |
| Šema | Fleksibilna | Nije fleksibilna |
| Integritet podataka | BASE | ACID |
| Skalabilnost | Horizontalno skaliranje | Vertikalno skaliranje |
| API | SQL nije obavezan | SQL obavezan |
| Transakcije | Podržava jednostavne | Podržava kompleksne |
| Promene u bazi podataka | Omogućava lake i česte promene u bazi podataka | Teško izvršava promene kada je baza definisana |

Tabela 1 – Razlike između NoQSL i SQL

# MongoDB

MongoDB je document-oriented NoSQL baza podataka koja se koristi za skladištenje i manipulaciju podacima velikog obima. Za razliku od tradicionlanih relacionih baza podataka kod kojih se koriste tabele, MongoDB koristi kolekcije i dokumente i podaci se čuvaju u JSON (JavaScript Object Notation) formatu. Dokumenti se sastoje od parova ključ/vrednost. Kolekcije sadrže grupe dokumenata i slične su tabelama relacionih baza podataka. Svaki dokument može biti drugačiji, veličina i sadržaj polja mogu se razlikovati jedan od drugog.



Slika 7 – Organizacija MongoDB baze podatka

Razlozi zbog čega se koristi MongoDB:

* Skladište – MongoDB može da skladišti velike struktuirane i nestruktuirane količine podataka. Za poboljšanje performansi pretrage koriste se indeksi.
* Integracija podataka
* Load balancing – MongoDB koristi koncept sharding-a za horizontalno skaliranje, tako što deli podatke na više instanci, MongoDB može da radi na više servera, balansirajući opterećenje kako bi se sistem mogao održati u funkciji u slučaju kvara hardvera.

Prednosti korišćenja MongoDB-a:

* Horizontalno skaliranje – MongoDB ima ovu mogućnost, što znači da se više čvorova može dodati u mrežu kako bi se povećala sposobnost obrade podataka i takođe omogućava lakše povećanje kapaciteta baze podataka sa povećanjem broja podataka koji se skladište.
* Fleksibilna struktura podataka – ne zahteva fiksnu šemu podataka, pa se podaci mogu čuvati u različitim strukturama.
* Visoka dostupnost – MongoDB se može kofigurisati tako da automatski otklanja greške čime se postiže visoka dostupnost podataka.
* Brzina – može jako brzo da čita i upisuje podatke i iz tog razloga se može koristiti za obradu velike količine podataka.
* Integracija sa drugim alatima – lako se integriše sa drugim alatima, što omogućava lakše upravljanje i obradu podataka.
* Replikacija – više instanci MongoDB-a koje se koriste za obezbeđivanje visoke dostupnosti.

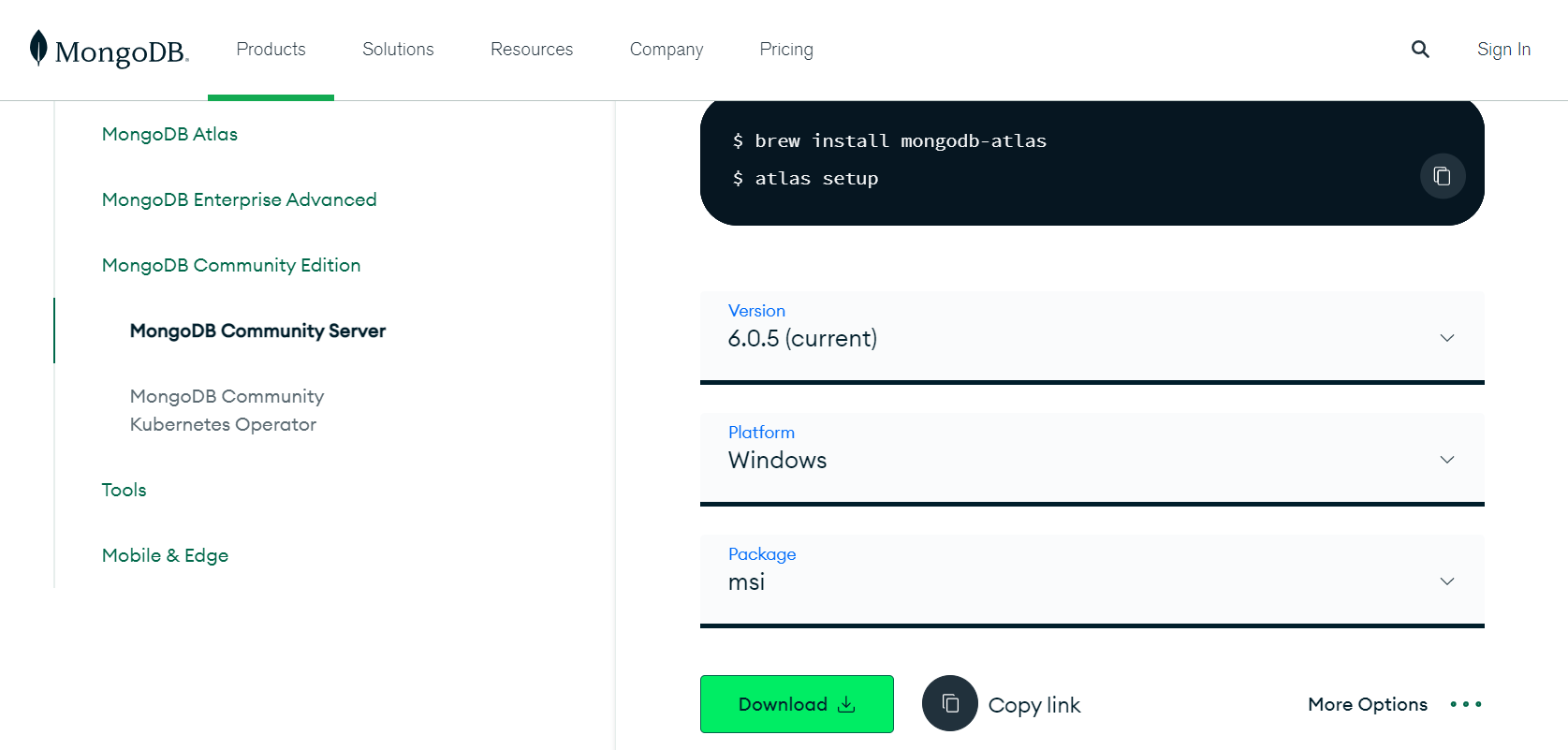
Iako postoji puno prednosti MongoDB-a, postoje i nedostaci:

* Nedostatak transakcija – MongoDB ne podržava transakcije sa više dokumenata u više kolekcija, što može otežati održavanje konzistentnosti u složenim aplikacijama.
* Nema podrške za join – ne modržava join operaciju, što može otežati izvođenje složenih upita
* Ne postoji ACID usklađenost – nije kompatibilan sa ACID-om, pa iz tog razloga nije pogodan za aplikacije koje zahtevaju strogu konzistentnost podataka.
* Složenost podešavanja – konfiguracija i podešavanje MongoDB-a može biti složeno i zahtevno, posebno ako je reč o složenim arhitekturama, postoje mnogi parametri koje je potrebno razmotriti kako bi se postigla optimalna performansa i stabilnost sistema.
* Visoka cena upravljanja memorijom – MongoDB koristi fleksibilnu strukturu za čuvanje podataka, te zahteva više memorije što dovodi do dodatnih troškova.

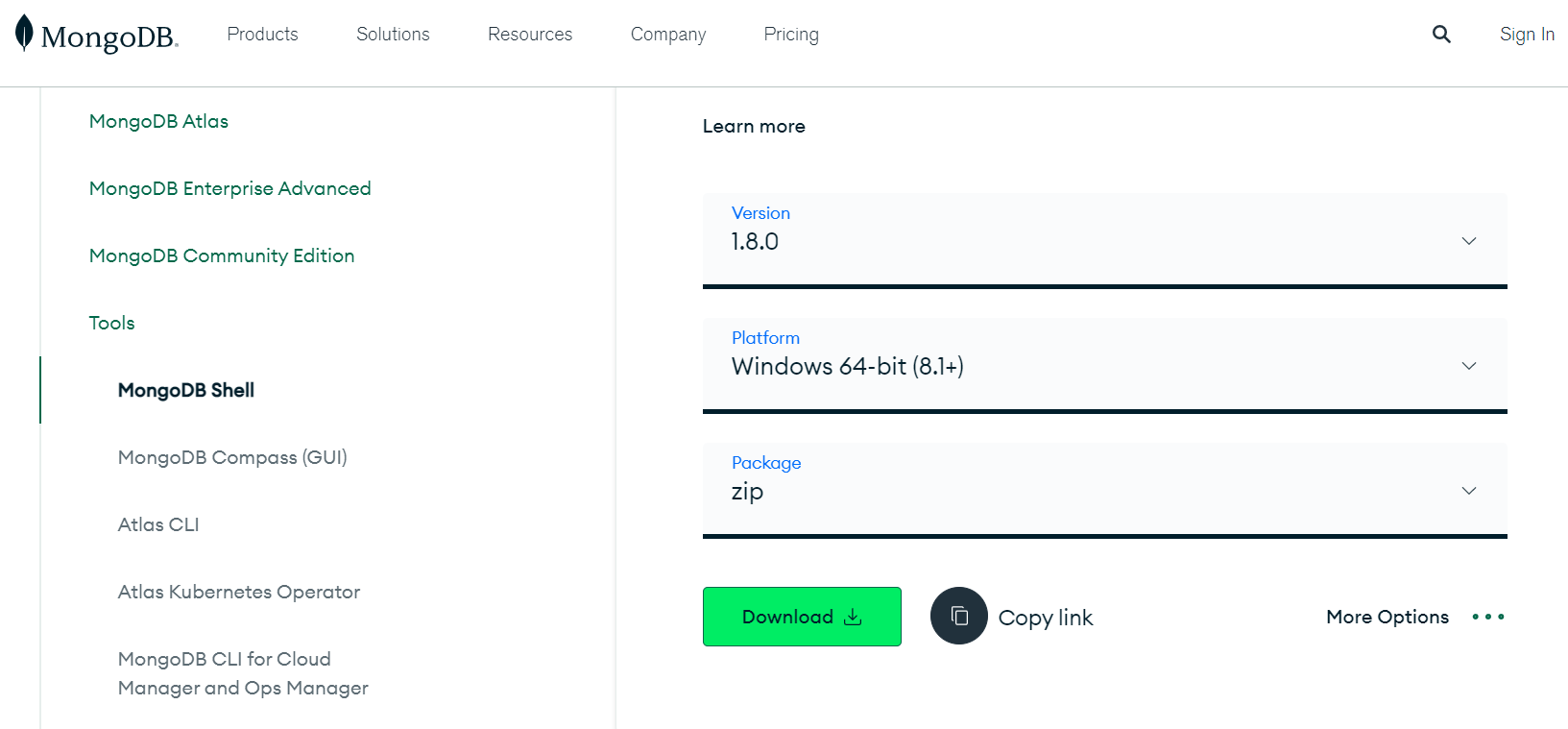
Pored svih ovih mana, MongoDB i dalje prestavlja popularnu bazu podataka za rad sa dokumentima, posebno u velikim i složenim aplikacijama.

## Pokretanje MongoDB-a

Prvo je potrebno preuzeti datoteku *MongoDB Community Server* sa zvanične stranice MongoDB-a i preuzeti alat *MongoDB Shell* (najnovije verzije u oba slučaja).

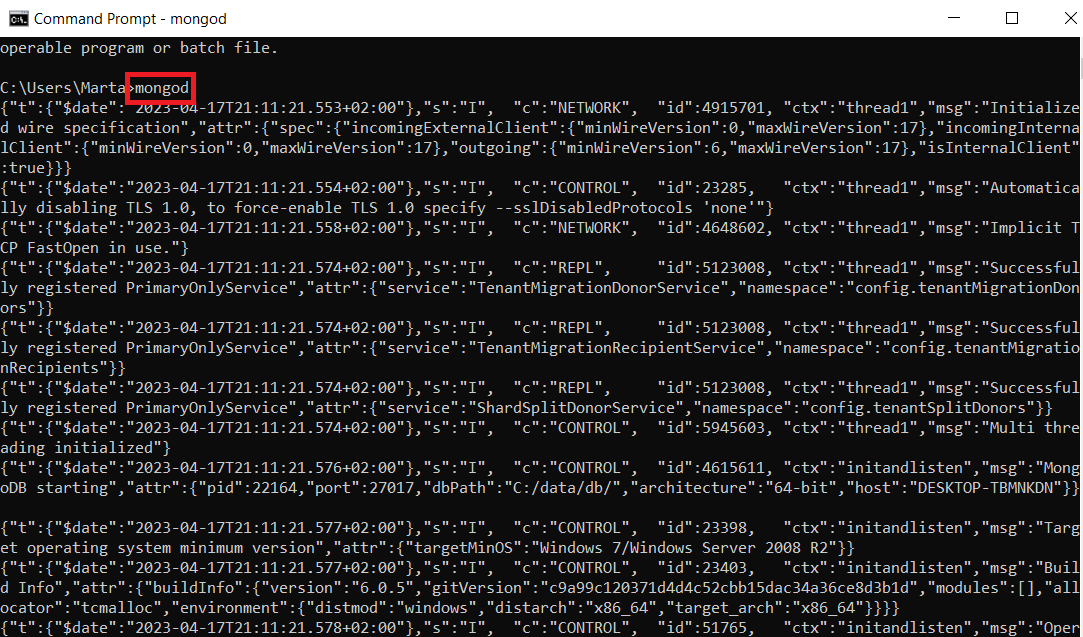


Slika 8 – Preuzimanje *MongoDB Community Server-*a



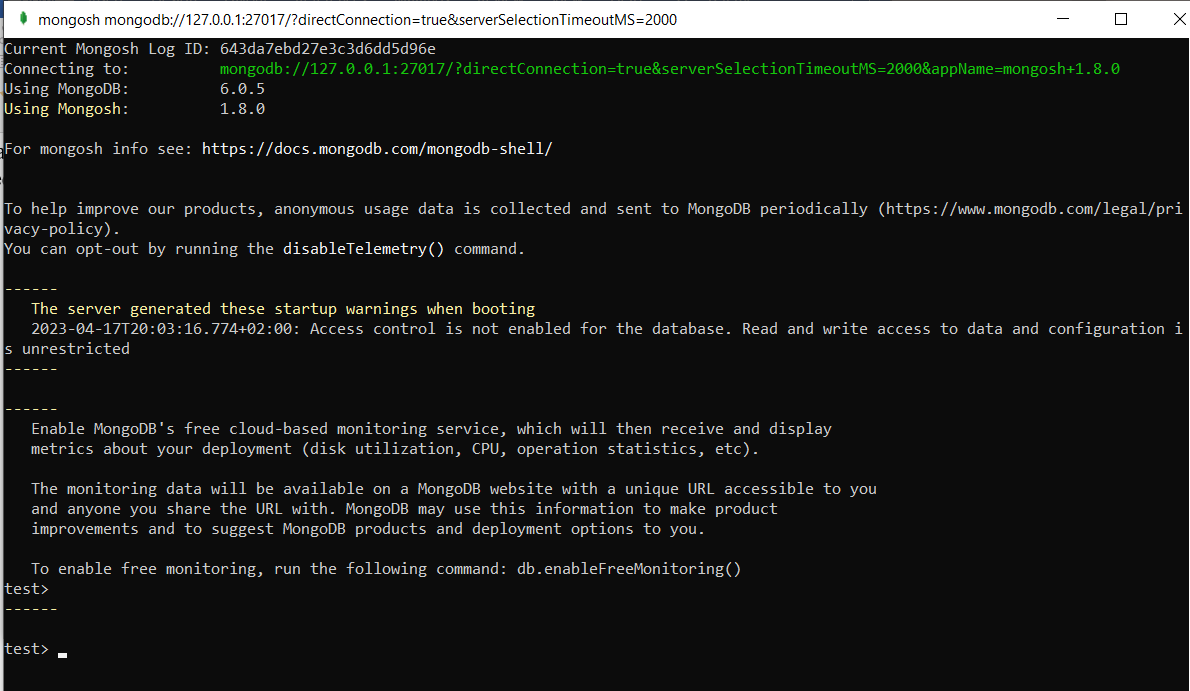
Slika 9 – Preuzimanje *MongoDB Shell-*a

Nakon uspešne instalacije potrebno je kreirati folder pod nazivom *db,* a zatim u Command Prompt-u uneti naredbu sa Slike 10:



Slika 10 – Pokretanje MongoDB servera

*mongod* je skraćenica za „Mongo Daemon“ i to je pozadinski proces koji koristi MongoDB. Glavna svrha mu je da upravlja svim zadacima MongoDB servera kao što su prihvatanje zahteva, odgovaranje klijentu ili upravljanje memorijom. Nakon toga potrebno je otvoriti *MongoDB Shell*:



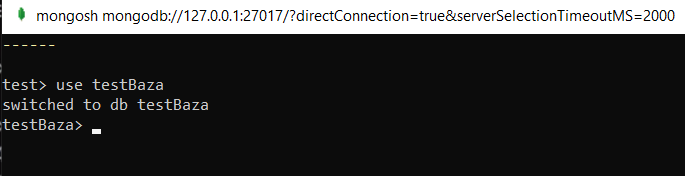
Slika 11 – MongoDB Shell

## Struktura MongoDB baze podataka

Iako MongoDB ima fleksibilnu šemu podataka, postoji podrazumevana struktura i model podataka koji bi trebalo biti ispoštovan. U nastavku će biti navedeno nekoliko uobičajenih termina u vezi sa MongoDB:

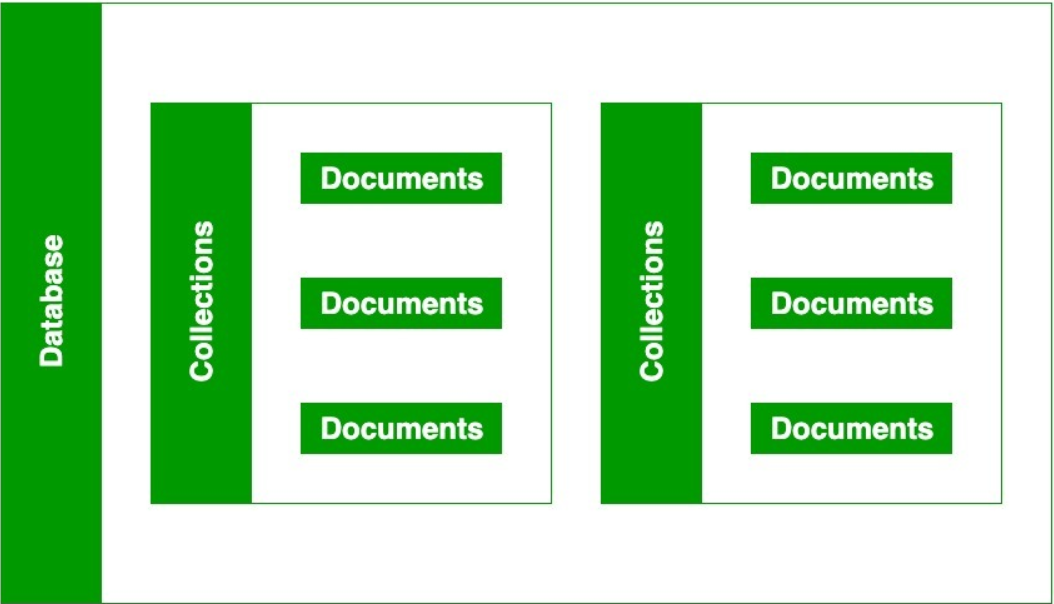
**Baza podataka** – može se nazvati fizičkim kontejnerom za podatke. Svaka baza podataka dobija svoj skup datoteka u sistemu datoteka. MongoDB server može da skladišti više baza podataka.

Za kreiranje baze podataka ne postoji klasična naredba *Create,* već je potrebno prvo prebaciti kontekst na nepostojeću bazu, koristeći komandu *use*:

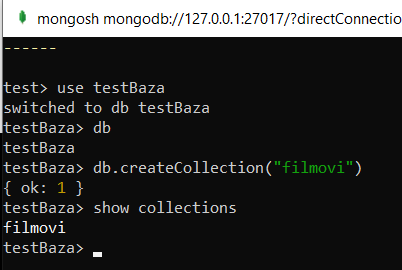


Slika 12 – kreiranje baze podatala

**Kolekcija** – ekvivalentno tabelama u bilo kom RDMS-u. Predstavlja skup dokumenata koji su grupisani na osnovu nekih kriterijuma. Ne postoji striktna šema kolekcije, unutar kolekcije nalaze se različiti dokumenti koji mogu imati različita polja, ali su uglavnom dokumenti unutar kolekcije namenjeni za istu svrhu. MongoDB kolekcije su skalabilne i fleksibilne što ih čini jako popularnim. Svaka kolekcija može sadržati neograničen broj dokumenata. Nakon što se kreira nova kolekcija, neće se odmah alocirati prostor na disku, već će se to desiti kada se u kolekciju doda prvi dokument.



Slika 13 – Kolekcije i dokumenti



Slika 14 – Kreiranje kolekcije

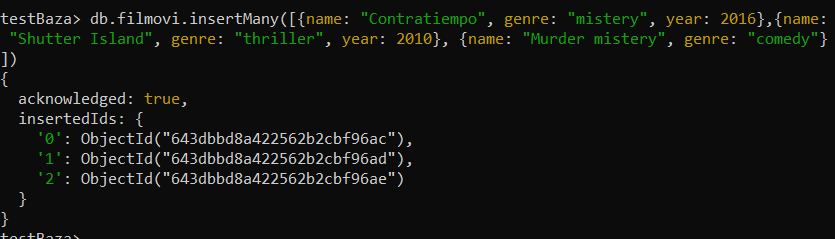
**Dokument** – predstavlja skup parova kjuč/vrednost. Dokumenti sadrže podatke koje želimo da sačuvamo u bazi podataka i oni su osnovna jedinica podataka u MongoDB-u. Polja u dokumentima su jednaka kolonama u relacionim bazama podataka. Struktura dokumenta je fleksibilna i svaki dokument može imati različit broj polja i tip podataka. Struktura dokumenta se menja dodavanjem novih ili ažuriranjem i brisanjem postojećih polja. Dokumetni su u JSON formatu, međutim dokumenti se interno skladište u BSON (Binary JSON) formatu jer je na taj način skladištenje efikasnije, a MongoDB radi prepakovanje iz BSON u JSON format, čitljiv format za čoveka. MongoDB podržava širok skup tipova podataka:

* String – mora biti UTF-8
* Ceo broj – čuva numeričku vrednost od 32 ili 64 bita u zavisnosti od servera
* Boolean – čuva vrednost *true/false*
* Double – čuva vrednosti sa pokretnim zarezom
* Nizovi – nizovi, liste ili više vrednosti sa jednim ključem
* Datum, vremenska oznaka
* Objekat – koristi se za embeded dokumente
* ID objekta
* Binarni podaci
* Null
* Regularni izraz

Prednost korišćenja dokumenata u MongoDB bazi podataka je fleksibilnost, brza obrada i učitavanje podataka, jednostavna i skalabilna struktura podataka.

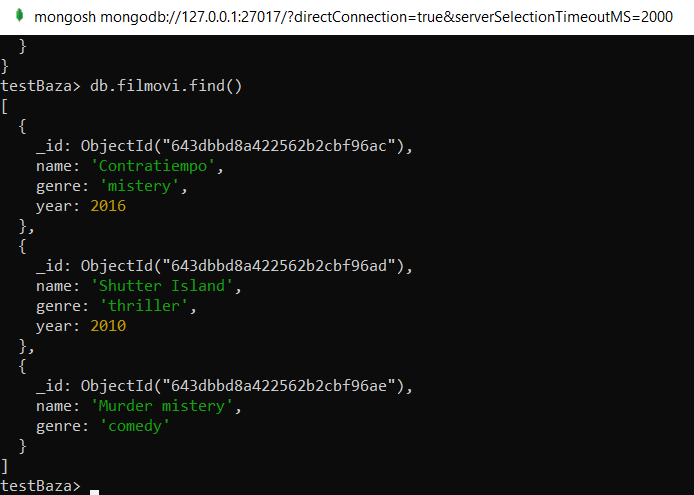
**\_id** – predstavlja jedinstvenu vrednost u dokumentu i obavezno je. Polje \_id služi kao jedinstveni identifikator i ključ za svaki dokument u kolekciji i pomaže u jedinstvenom pretraživanju, ažuriranju i brisanju dokumenta. Ovo polje može se ručno dodeliti ili se automatski generisati uz pomoć ObjectId() vrednosti. ObjectId se sastoji od 12 bajtova (4B – vremenska oznaka, 5B – nasumična vrednost koja se generiše jednom po procesu i ova vrednost je jedinstvena za proces, 3B – brojač sa povećanjem inicijalizovan na slučajnu vrednost). Ovo predstavlja podrazumevani unique indeks i ne može se obrisati.

Na Slici 15 prikazano je dodavanje dokumenata u kolekciju „filmovi“ pomoću naredbe *insertMany.* Pomoću nje je moguće dodati više dokumenata odjednom. Postoji i naredba *insertOne* kojom se dodaje jedan dokument u kolekciju. U ovom slučaju polje \_id je kreirano automatski pomoću ObjectId().



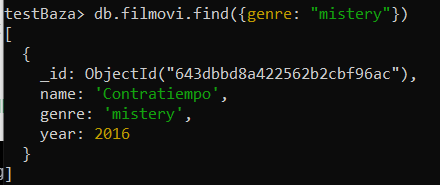
Slika 15 – Ubacivanje dokumenata u kolekciju filmovi

*find()* metoda se koristi za selektovanje dokumenata u kolekciji i vraćanje pokazivača na izabrane dokumente, jedan po jedan. Ako nije naveden nijedan parametar, onda ova metoda vraća sve dokumente i vraća prvih 20 dokumenata.



Slika 16 – find()

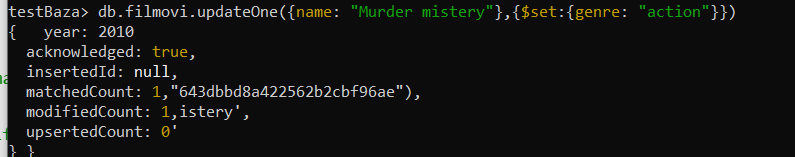
Prikazivanje dokumenata po određenom kriterijumu prikazano je na sledećoj slici:



Slika 17 – find() sa kriterijumom

Brisanje svih dokumenata vrši se metodom *deleteMany(),* a da bi se obrisao jedan dokument koristi se metoda *deleteOne()*. U oba slučaja se kao parametar može navesti kriterijum po kome se vrši navedena akcija.

Ažuriranje vrednosti nekog polja vrši se na sledeći način, korišćenjem operatora *$set* koji modifikuje vrednost određenog polja:

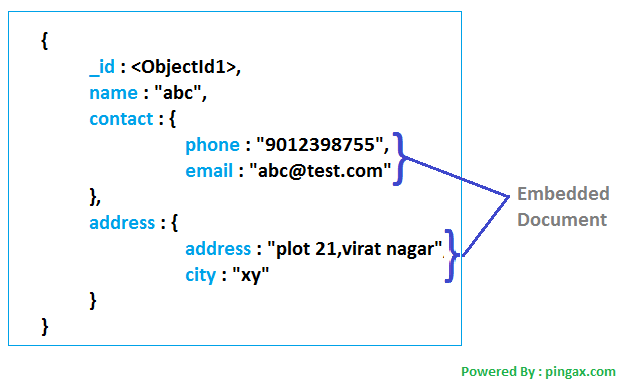


Slika 18 – ažuriranje dokumenta

Pored ove, postoji i metoda *updateMany()*, koja istovremeno ažurira vrednost više polja koja odgovaraju određenom kriterijumu.

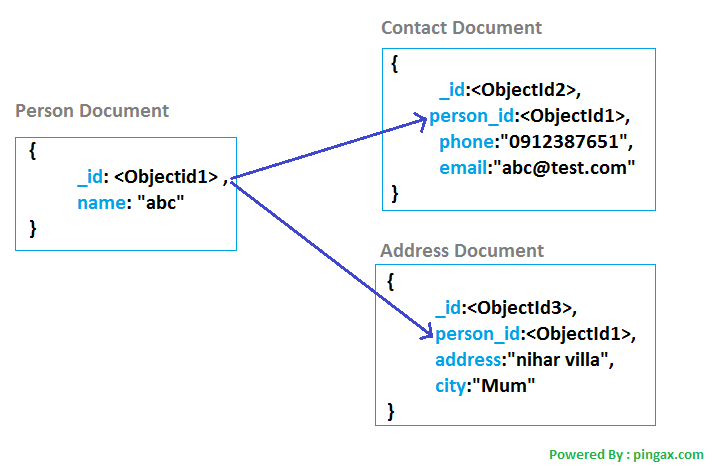
MongoDB ima dva tipa modela podataka – embedded (ugrađeni) i normalizovani model podataka. U zavisnosti od zahteva može se koristiti bilo koji od ova dva modela.

Embedded model poznat je i kao „denormalizovani“ model i podrazumeva da se svi podaci koji su povezani nalaze u istom dokumentu. Ovaj model podataka koristan je kada podaci neće biti deljeni sa drugim entitetima i kada se neće često menjati. Olakšava upite nad podacima i povećava brzinu čitanja, ali može povećati veličinu dokumenta.



Slika 19 – Embedded model podataka

U normalizovanom modelu se podaci različitih entiteta nalaze u različitim kolekcijama, povezivanje između njih vrši se pomoću ID-a. Koristi se kada postoje veze između različitih entiteta i kada se podaci često menjaju. Normalizovani model smanjuje veličinu dokumenta i omogućava fleksibilnost u izvršavanju upita, ali je čitanje podataka sporije. Za raliku od embedded modela, ovaj model sprečava dupliranje podataka jer je moguće referencirati se na drugi dokument bez dupliranja njegovog sadržaja.



Slika 20 – Normalizovani model podataka

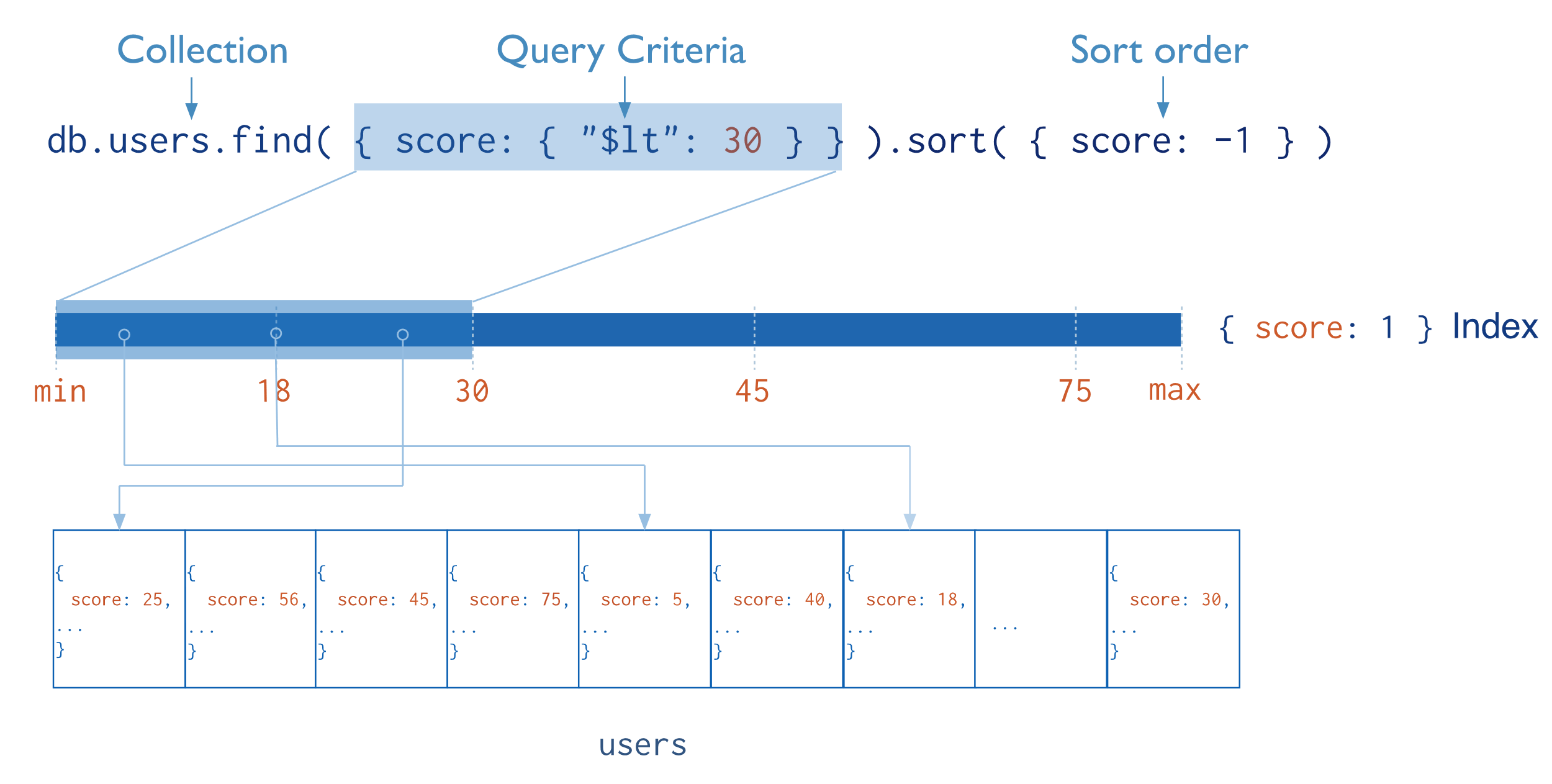
U praksi se najčešće koristi hibridni pristup, gde se koristi kombinacija ova dva modela i na taj način se dobija najbolja kombinacija fleksibilnosti i brzine.

## Optimizacija podataka

Da bi se podaci efikasno zapamtili u MongoDB, treba primeniti odgovarajuće tehnike optimizacije podataka koje će biti navedene u nastavku.

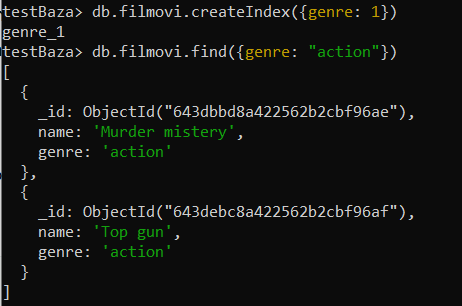
### Indeksi

Indeks je struktura koja postoji u bazi podataka radi poboljšanja performansi upita. U slučaju da se indeksi ne koriste, MongoDB mora skenirati sve dokumente kako bi proverio da li odgovaraju upitu. Sa druge strane, ako indeksi postoje, ograničen je broj upita koje će MongoDB skenirati i ovim se performanse upita poboljšavaju. U osnovi, indeksi u MongoDB-u su slični indeksima u drugim sistemima baza podataka. On definiše indekse na nivou kolekcije i podržava indekse za bilo koje polje dokumenta u kolekciji. MongoDB obezbeđuje niz različitih tipova indeksa.



Slika 21 - indeksi

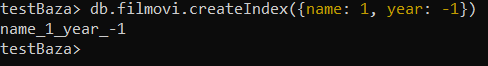
**Podrazumevani \_id indeks** – MongoDB kreira jedinstveni indeks u polju \_id prilikom kreiranja kolekcije i sprečava korisnike da ubace dva dokumenta sa istom vrednošću za polje \_id. Moguće je kreirati indeks u bilo kom polju korišćenjem naredbe *db.collection.createIndex().* Ova metoda kreira indeks samo u slučaju da ne postoji indeks iste specifikacije.



Slika 22 – kreiranje indeksa

Na Slici 22 je prikazano kreiranje indeksa nad poljem „genre“. Za polja koja čine indeks može da se specificira smer sortiranja – 1 za rastući, -1 za opadajući redosled.

**Kompozitni indeksi** – MongoDB podržava indekse definisane na više polja, odnosno složene indekse. Redosled polja navedenih u kompozitnom indeksu ima značaja, jer u slučaju da se kompozizni indeks sastoji od {userId: 1, years: -1}, indeks prvo sortira po userId-u, a zatim unutar toga sortira po rezultatu.



Slika 23 – kreiranje kompozitnog indeksa

**Indeks sa više ključeva** – MongoDB koristi indekse sa više ključeva za indeksiranje sadržaja uskladištenog u nizovima. Kreira zasebne unose indeksa za svaki element niza i ovi indeksi omogućavaju upitima da izaberu samo dokumente koji sadrže nizove uparujući elemente.

**Heširani indeksi** – MongoDB obezbeđuje heširani tip indeksa, koji indeksira heš vrednost polja. Ovaj tip indeksa podržava samo podudaranja i ne podržava opsege.

**Sparse indeksi** – dozvoljavaju indeksiranje samo onih dokumenata koji imaju indeksirano polje. Podrazumevano, MongoDB indeksi nisu sparse indeksi. Ovo može smanjiti veličinu indeksa i ubrzati upite koje koristi taj indeks.

### GridFS

Maksimalna veličina dokumenta koji MongoDB može da podrži je 16MB. GridFS je specifikacija za čuvanje i preuzimanje datoteka koje premašuju ovo ograničenje veličine dokuemnta. On deli datoteke na delove i svaki deo čuva kao poseban dokument, a svaki deo je maksimalne veličine 255KB, osim poslednjeg koji je veličine koliko je potrebno. Podrazumevano koristi dve kolekcije *fs.files* koja skladišti metapodatke datoteke i *fs.chunks* koja sadrži delove dokumenta. Kada se postavi upit GridFS-u za neku datoteku, drajver ponovo sastavlja delove kao što je potrebno.

Diagram

Description automatically generated

Slika 24 - GridFS

Kolekcija chunks minimalno mora da poseduje sledeća polja:

A picture containing graphical user interface

Description automatically generated

Slika 25 – polja chunks kolekcije

Kolekcija files treba da poseduje sledeća polja, od kojih su prvih 5 obavezna polja, a ostala su proizvoljna:

Text

Description automatically generated with medium confidence

Slika 26 - polja files kolekcije

GridFS organizuje datoteke u kantu, grupu MongoDB kolekcija koje sadrže delove datoteka i deskriptivne informacije. Kada se kreira nova kanta, upravljčki program kreira komade i kolekcije datoteka, sa prefiksom podrazumevanog naziva kante fs, osim ako se ne navede drugačije ime. Upravljački program kreira indeks za svaku kolekciju kako bi obezbedio efikasno preuzimanje datoteka i povezanih metapodataka. Svaki deo je identifikovan svojim jedinstvenim poljem \_id. Fs.files služi kao roditeljski dokument. Polje files\_id unutar chunks kolekcije povezuje deo sa njegovim roditeljem.

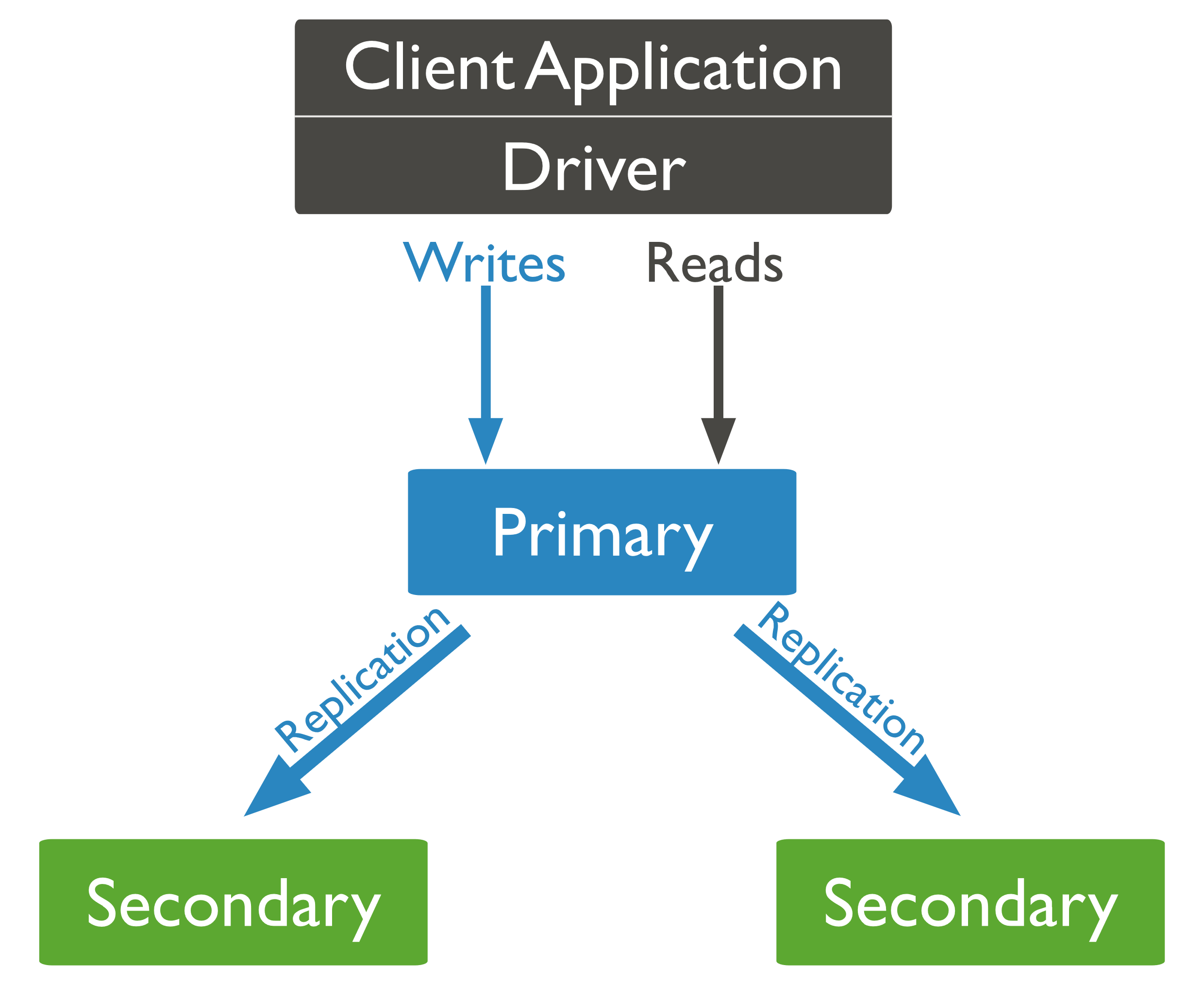
GridFS nije koristan samo za čuvanje datoteka koje prelaze 16MB, već i za čuvanje svih datoteka kojma je potrebno pristupiti bez učitavanja cele datoteke u memoriju. GridFS treba koristiti u sledećim slučajevima:

* Kada postojeći sistem datoteka ima ograničenje na broj datoteka koje se mogu uskladištiti u datom direktorijumu.
* Kada se mora pristupiti samo delu sačuvanih informacija bez pregleda celog dokumenta.
* Kada se distribuiraju datoteke i njihovi metapodaci preko geografski distribuiranih replika, omogućava da se metapodaci automatski sinhronizuju i raspoređuju na brojnim računarima.

GridFs ne treba koristiti kada se ažurira ceo sadržaj datoteke.

### Mehanizam replikacije

MongoDB mehanizam replikacije omogućava sinhronizaciju podataka između većeg broja MongoDB instanci. Skup replika je grupa mongod procesa koji održavaju isti skup podataka. Jedna instanca je primarna, a sve ostale su sekundarne. Svi klijenti upisuju podatke u primarnu instancu, a podaci se nakon toga asinhrono sinhronizuju sa sekundarnim instancama. Replikacija obezbeđuje redudantnost i povećava dostupnost podataka kod čitanja. Trajnost podataka se održava tako što se čuva više kopija ili replika tih podataka na fizički izolovanim serverima. Skup čvorova sadrži nekoliko čvorova koji imaju podatke i opciono jedan arbitarski čvor koji ne poseduje podatke. Od čvorova koji imaju podatke samo jedan može da bude primarni, dok se ostali čvorovi smatraju sekundarnim čvorovima. MongoDB obezbeđuje podršku za failover mehanizam. U slučaju da primarni čvor otkaže, mehanizmom glasanja ostale instance proglašavaju jednu od sekundarnih za novu primarnu.



Slika 27 – mehanizam replikacije

Razlozi zbog kojih je dobro korišćenje mehanizma replikacije:

* Visoka dostupnost podataka – ako primarni čvor prestane sa radom, sekundarni čvorovi mogu preuzeti funkciju primarnog čvora i osigurati dostupnost baze podataka
* Automatski oporavak od kvara – kada se primarni čvor oporavi od kvara, automatski će se sinhronizovati sa drugim čvorovima i vratiti funkciju primarnog čvora
* Bolja zaštita od gubitka podataka – replikacija osigurava da se podaci automatski kopiraju na druge čvorove u replikaciji i time se smanjuje rizik od gubitka podataka

Da bi se konfigurisala replikacija potrebno je izvršiti sledeće naredbe:

1. Pokretanje 3 MongoDB procesa, prvi proces biće primarni, a drugi i treći će biti sekundarni

mongod --replSet myReplSet --port 27017 --dbpath /data/rs1 --bind\_ip localhost

mongod --replSet myReplSet --port 27018 --dbpath /data/rs2 --bind\_ip localhost

mongod --replSet myReplSet --port 27019 --dbpath /data/rs3 --bind\_ip localhost

1. Povezivanje sa jednim od MongoDB procesa

mongosh --port 27017

1. Inicijalizacija replikacije:

rs.initiate( {

\_id : "myReplSet",

members: [

{ \_id: 0, host: "localhost:27017" },

{ \_id: 1, host: "localhost:27018" },

{ \_id: 2, host: "localhost:27019" }

]

})

1. Proveravanje statusa replikacije

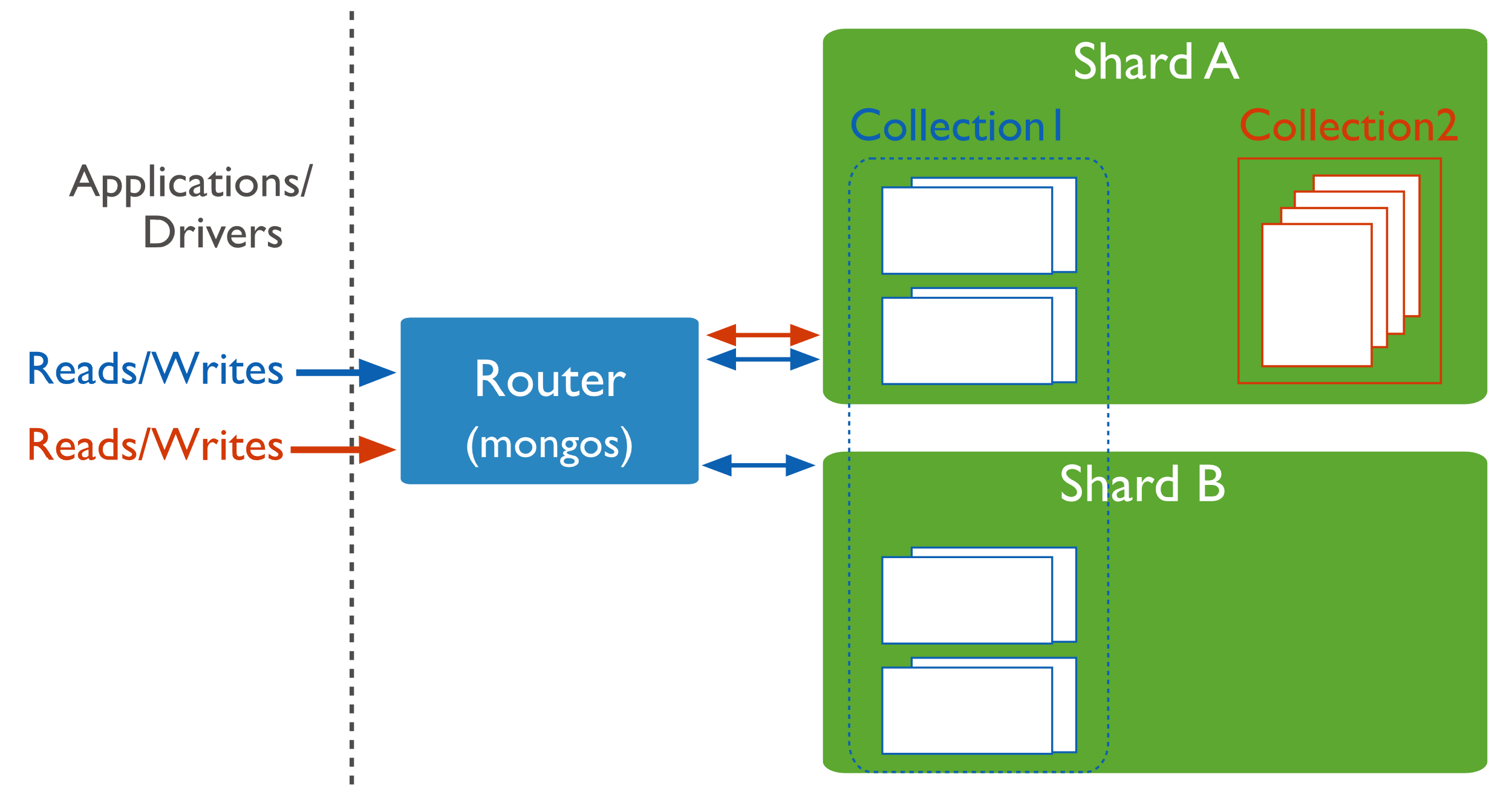
rs.status()

### Sharding

Sharding mehanizam omogućava particionisanje kolekcije dokumenata i njihovu distribuciju na veći broj instanci (shards). Njegov zadatak je da poveća kapacitet sistema. Omogućava horizontalno skaliranje koje predstavlja dodavanje mašina za deljenje skupa podataka i učitavanje. Horizontalno skaliranje pruža skoro neograničeno skaliranje za rukovanje velikim podacima i intenzivnim opterećenjima.

Shard arhitektura:

* Shard – skup replika koje sadrže podatke
* Tri MongoDB servera – sadrže metapodatke i mapiraju particije i MongoDB instance
* Veći broj mongos instanic – deluje kao ruter upita za klijentske aplikacije. Rutiraju klijentske zahteve ka odgovarajućim instancama na osnovu metapodataka. Klijenti se povezuju na mongos, a ne na pojedinačne delove.



Slika 28 – sharding mehanizam

Sharding omogućava skaliranje baze podataka kako bi se podnela opterećenja do skoro neograničenog stepena i time je povećan protok čitanja/pisanja. Takođe omogućava lako kreirane distribuirane baze podataka za podršku geografski distribuiranim aplikacijama.

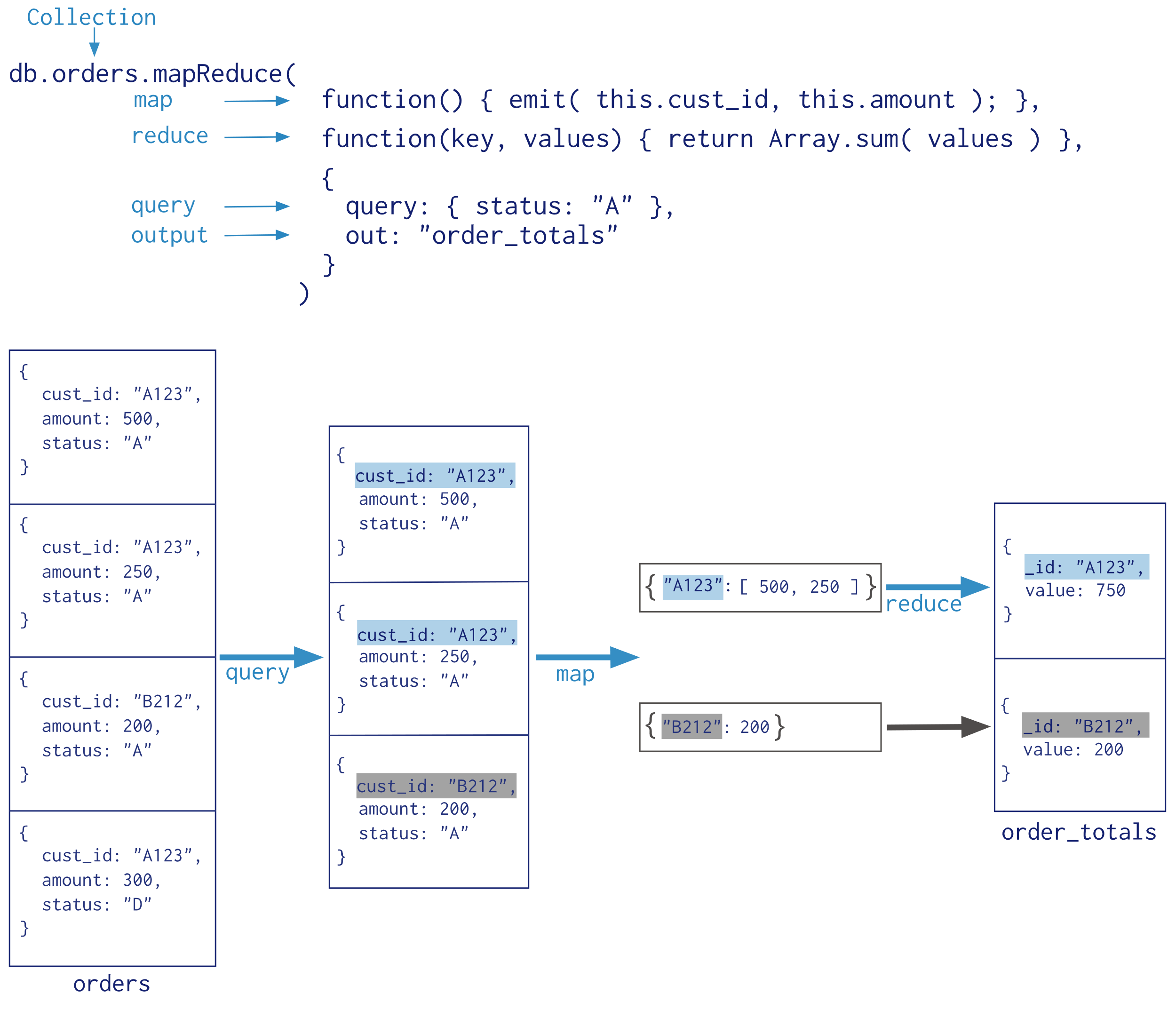
### Map/Reduce

Map/Reduce je programski model za obradu velike količine podataka. Glavna ideja je distribucija obrade na veći broj čvorova. Pruža paralelnu obradu velike količine podataka koji se nalaze u bazi podataka ili file sistemu i obrada se vrši na distribuiranim čvorovima. Odvija se u dva koraka: map i reduce.

U map koraku master čvor prikuplja ulazne podatke, koje zatim deli na manje delove i distribuira ih worker čvorovima. Worker čvor može ponoviti korak i time kreira stablo čvorova, a takođe i obrađuje problem i rezultat vraća master čvoru.

U reduce koraku master prikuplja sve rezultate koje su vratili worker čvorovi i kombinuje ih u rezultat originalnog problema. Dokumenti rezultat moraju odgovarati ograničenju veličine dokumenta, odnosno ne smeju biti veći od 16MB.

Map/Reduce je koristan u situacijama kada se podaci moraju analizirati i obrađivati u različitim koracima, jer Map/Reduce može izvršiti nekoliko iteracija nad skupom podataka da bi se dobio konačan rezultat.



Slika 29 – Map/Reduce

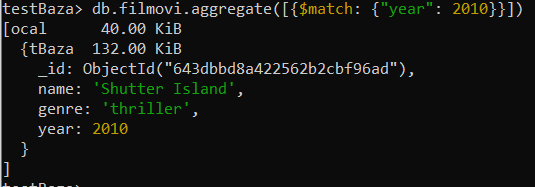
Na Slici 29 je prikazana funkcija „map“ koja emituje id, iznos i status narudžbine, a funkcija „reduce“ sabira iznose narudžbina sa istim id-jem. Nad kolekcijom „orders“ se poziva funkcija „mapReduce“ i njoj se prosleđuju funkcije „map“ i „reduce“ kao i izlaz u novu kolekciju „order\_totals“. Na ovaj način generisaće se nova kolekcija koja će sadržati ukupan iznos svake narudžbine.

### Agregacija

Proces koji se koristi za dobijanje rezultata na osnovu različitih operacija koje se primenjuju nad podacima. Pomoću agregacije postiže se efikasna i fleksibilna analiza podataka, kao što su operacije sabiranja, filtriranja, sortiranja...

Podržani operatori:

* $project – određuje koja polja iz dokumenta će biti uključena u rezultat
* $match – filtriranje dokumenta na osnovu zadatog uslova
* $skip – preskače određeni broj dokumenata na početku kolekcije
* $group – grupisanje dokumenata
* $sort – sortiranje dokumenata
* $limit – definiše broj dokumenata koji će se obraditi
* $unwid – transformisanje niza



Slika 30 – agregacija sa match operatorom

Agregacija sa slike koristi operaciju „$match“ koja filtrira dokumente na osnovu navedenog uslova.

# Zaključak

Na samom kraju, može se zaključiti da je fizičko projektovanje MongoDB baze podataka ključno za postizanje optimalnih performansi i skalabilnosti. Pored fizičkog projektovanja, podjednako bitnu ulogu ima i optimizacija podataka. U ovom radu su navedene i objašnjene tehnike koje se koriste za optimizaciju, uz prateće primere.

Prilikom dizajniranja šeme treba pažljivo obratiti pažnju na sve faktore, kao što su veličina dokumenata, količina podataka koji se čitaju i upisuju. MongoDB nudi razne opcije za optimizaciju performansi i efikasno obrađivanje podataka. Korišćenje pravilnih modela podataka, sharding-a, indeksiranja i ostalih stvari koje su opisane u radu od velike je važnosti za efikasno i uspešno fizičko projektovanje baze podataka.

Fizičko projektovanje MongoDB baze podataka i optimizacija podataka ključni su faktori za postizanje optimalnih perfomansi i povećanje efikasnosti.

# Literatura

1. <https://aws.amazon.com/nosql/>

2. <https://www.couchbase.com/resources/why-nosql/>

3. <https://www.techtarget.com/searchdatamanagement/definition/MongoDB>

4. <https://www.tutorialsteacher.com/mongodb/what-is-mongodb>

5. <https://www.mongodb.com/docs/manual/indexes/>

6. <https://www.mongodb.com/docs/manual/core/gridfs/>

7. <https://www.mongodb.com/docs/manual/replication/>

8. <https://www.mongodb.com/docs/manual/sharding/>

9. <https://www.mongodb.com/docs/manual/core/map-reduce/>

10. <https://www.mongodb.com/docs/manual/aggregation/>