**Univerzitet u Nišu**

**Elektronski fakultet**

**Katedra za računarstvo**

**Seminarski rad**

**Interna struktura i organizacija skladišta podataka kod MongoDB skladišta podataka**

**Mentor: Student:**

**Aleksandar Stanimirović Andrea Popović 1475**

**Niš, 2023**

Contents

[1. Uvod 3](#_Toc132661949)

[2. NoSQL baze podataka 4](#_Toc132661950)

[3. Dokument orjentisane baze podataka 6](#_Toc132661951)

[4. Interna struktura MongoDB-ja 7](#_Toc132661952)

[4.1. Osnovni koncepti 7](#_Toc132661953)

[4.2. Karakteristike MongoDB-ja 11](#_Toc132661955)

[5. Skladištenje podataka 12](#_Toc132661956)

[5.1. BSON format 12](#_Toc132661957)

[5.2. Ograničenja veličine BSON fajlova 15](#_Toc132661958)

[5.3. GridFS API 16](#_Toc132661959)

[5.4. Mehanizmi za skladištenje podataka 18](#_Toc132661960)

[5.4.1. WiredTiger Storage Engine (podrazumevano) 18](#_Toc132661961)

[5.4.1.2. Journal 19](#_Toc132661962)

[5.4.1.3. Kompresija 19](#_Toc132661963)

[5.4.2. In-Memori Storage Engine 21](#_Toc132661964)

[6. Praktični deo 22](#_Toc132661965)

[6.1. Struktura dokumenata 25](#_Toc132661966)

[6.2. Embedded vs Referenced 26](#_Toc132661967)

[6.3. Embedded dokumenti 27](#_Toc132661968)

[6.4. Referentni dokumenti 28](#_Toc132661969)

[6.5. Agregacija 31](#_Toc132661970)

[7. Zaključak 32](#_Toc132661971)

[8. Literatura 33](#_Toc132661972)

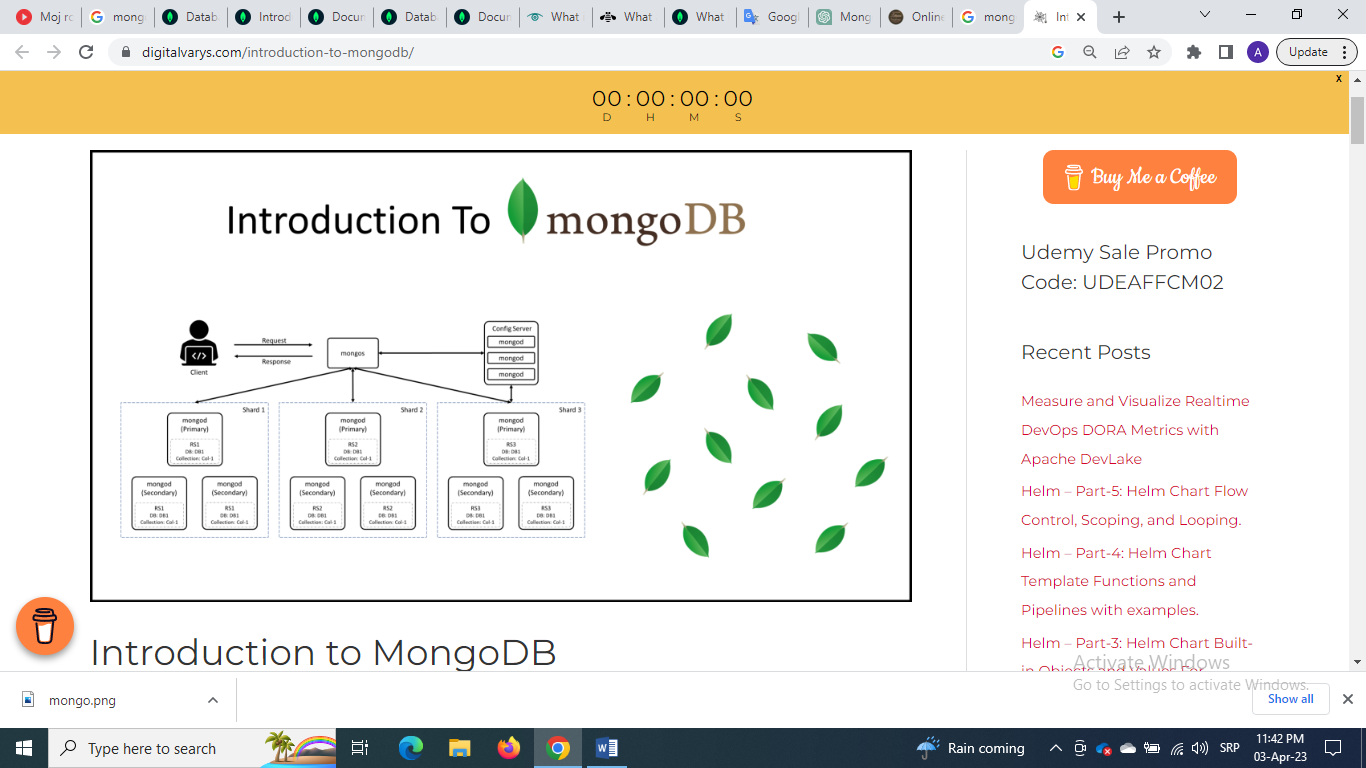
# Uvod

MongoDB predstavlja izuzetno popularno rešenje za upravljanje NoSQL bazom podataka. NoSQL (Not Only Sql) se koristi kao alternativa tradicionalnim relacionim bazama podataka. NoSQL baze podataka su veoma korisne za rad sa velikim skupovima distribuiranih podataka. MongoDB je alatka koja može da upravlja informacijama orijentisanim na dokumente, čuva informacije ili ih preuzima.

MongoDB se koristi za skladištenje podataka velikog obima, pomažući organizacijama da skladište velike količine podataka, a da pritom i dalje rade brzo i efikasno. Organizacije takođe koriste MongoDB za svoje ad-hoc upite, indeksiranje, balansiranje opterećenja, agregaciju i druge funkcije.

Structured Query Language (SQL) je standardizovani programski jezik koji se koristi za upravljanje relacionim bazama podataka. SQL normalizuje podatke kao šeme i tabele, a svaka tabela ima fiksnu strukturu. Problem kod ovih podataka jeste skladištenje velike količine podataka, jer sa povećanjem količine podataka perfomanse se smanjuju. Zbog toga one SQL baze podataka nisu najbolje rešenje za web aplikacije jer njhova osnovna karakteristika skladištenje velike količine podataka. Takodje, šema nije fleksibilna, a to je još jedan od zahteva web aplikacija.

Umesto korišćenja tabela i redova kao u relacionim bazama podataka, kao NoSQL baza podataka, MongoDB arhitektura se sastoji od kolekcija i dokumenata. Dokumenti se sastoje od parova ključ/vrednost -- MongoDB-ova osnovna jedinica podataka. Kolekcije, ekvivalentne SQL tabelama, sadrže skupove dokumenata. MongoDB nudi podršku za mnoge programske jezike, kao što su C, C++, C#, Go, Java, Python, Ruby i Swift.



Slika 1. Uvod u MongoDB

# NoSQL baze podataka

NoSQL baze podataka obezbeđuju mehanizam za skladištenje i pronalaženje podataka koji je modelovan na način koji nije tabelarni odnos koji se tradicionalno koristi u relacionim bazama podataka. Takve baze podataka postoje od kasnih 1960-ih, ali je naziv „NoSQL“ skovan tek početkom 21. veka. Javile su se kao potrebe potrebama web kompanija. NoSQL baze podataka se sve više koriste za skladištenje velike količine podataka i u web aplikacijama u realnom vremenu. NoSQL sistemi se takođe ponekad nazivaju ,,Not Only SQL“ da bi se naglasilo da oni mogu da podržavaju jezike upita slične SQL-u.

Umesto tipične tabelarne strukture relacione baze podataka, NoSQL baze podataka sadrže podatke unutar jedne strukture podataka, kao što je na primer JSON dokument. Pošto ovaj nerelacioni dizajn baze podataka ne zahteva šemu, on nudi brzu skalabilnost za upravljanje velikim i tipično nestrukturiranim skupovima podataka.

NoSQL je takođe tip distribuirane baze podataka, što znači da se informacije kopiraju i čuvaju na različitim serverima, koji mogu biti udaljeni ili lokalni. Ovo obezbeđuje dostupnost i pouzdanost podataka. Ako neki od podataka odu van mreže, ostatak baze podataka može nastaviti da radi.

Danas kompanije moraju da upravljaju velikim količinama podataka pri velikim brzinama sa mogućnošću brzog povećanja radi pokretanja modernih web aplikacija u skoro svakoj industriji. U ovoj eri rasta u okviru oblaka, velikih podataka i mobilnih i web aplikacija, NoSQL baze podataka pružaju tu brzinu i skalabilnost, što ih čini popularnim izborom za njihove performanse i lakoću korišćenja.

Iako je NoSQL predstavlja alternativu SQL-u, ovaj napredak ni u kom slučaju nije zamenio SQL baze podataka. Na primer, recimo da upravljate maloprodajnim porudžbinama u kompaniji. U relacionom modelu, pojedinačne tabele bi upravljale podacima o klijentima, podacima o porudžbini i podacima o proizvodu odvojeno, i bile bi spojene zajedno preko jedinstvenog, zajedničkog ključa, kao što je ID klijenta ili ID porudžbine. Iako je ovo odlično za brzo skladištenje i preuzimanje podataka, zahteva značajnu memoriju. Kada želite da dodate više memorije, SQL baze podataka mogu da se skaliraju samo vertikalno, a ne horizontalno, što znači da je vaša mogućnost dodavanja više memorije ograničena na hardver koji imate. Rezultat je da vertikalno skaliranje na kraju ograničava skladištenje i preuzimanje podataka vaše kompanije.

Za poređenje, NoSQL baze podataka nisu relacione, što eliminiše potrebu za povezivanjem tabela. Njihove mogućnosti visoke dostupnosti olakšavaju horizontalno skaliranje. Ako jedan server baze podataka nije dovoljan za skladištenje svih vaših podataka ili rukovanje svim upitima, radno opterećenje se može podeliti na dva ili više servera, omogućavajući kompanijama da horizontalno skaliraju svoje podatke.

Iako svaki tip baze podataka ima svoje prednosti, kompanije obično koriste i NoSQL i relacione baze podataka u jednoj aplikaciji. Baza se bira samo na osnovu krajnjih ciljeva i svrhe korišćenja iste.

Vremenom su se pojavila četiri glavna tipa NoSQL baza podataka:

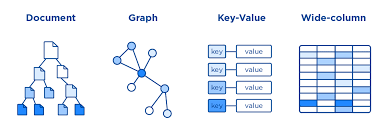
* dokument baze podataka,
* baze podataka ključ-vrednost,
* wide-column baze i
* baze podataka grafova.

Baze podataka dokumenata čuvaju podatke u dokumentima sličnim JSON (JavaScript Object Notation) objektima. Svaki dokument sadrži parove polja i vrednosti. Vrednosti obično mogu biti različitih tipova uključujući stvari kao što su stringovi, brojevi, logičke vrednosti, nizovi ili objekti. MongoDB je predstavnik ovog tipa baze podataka pa će o njemu biti više reči u nastavku.

Baze podataka ključ/vrednost su jednostavniji tip baze podataka gde svaka stavka sadrži ključeve i vrednosti. Dakle ovaj model podataka bez šeme je organizovan u rečnik parova ključ/vrednost. Ključ je sličan jedinstvenom ključu u SQL bazama, dok je vrednost niz podataka objekta koji ima taj ključ. Obično se koristi za keširanje i čuvanje informacija o sesiji korisnika. Redis predstavlja najpoznatiji primer ove baze.

Wide-column baze podataka čuvaju podatke u tabelama, redovima i dinamičkim kolonama. Ove baze podataka čuvaju informacije u kolonama, omogućavajući korisnicima da pristupe samo određenim kolonama koje su im potrebne bez dodeljivanja dodatne memorije na nerelevantne podatke. Apache HBase i Apache Cassandra su primeri wide-column baza podataka.

Graf baze podataka čuvaju podatke u čvorovima i ivicama. Čvorovi obično čuvaju informacije o ljudima, mestima i stvarima, odnosno objektima dok ivice čuvaju informacije o odnosima između čvorova, odnosno njihovim vezama. Neo4j predstavlja predstavnika ovog tipa baze podataka.



Slika 2. NoSQL

# Dokument orjentisane baze podataka

Kako MongoDB pripada ovoj grupi baza podataka, potrebno je pomenuti i neke opšte karakteristike ove grupe. Baza podataka orijentisana na dokumente ili skladište dokumenata je računarski softver i sistem za skladištenje podataka dizajniran za skladištenje, preuzimanje i upravljanje informacijama orijentisanim na dokumente, takođe poznatim kao polustrukturirani podaci.

Kao što je već pomenuto, baze podataka orijentisane na dokumente su jedna od glavnih kategorija NoSQL baza podataka. XML baze podataka su podklasa ove grupe baza podataka koje su optimizovane za rad sa XML dokumentima. Graf orjentisane baze podataka su slične, ali dodaju još jedan sloj, odnos, koji im omogućava da povežu dokumente radi brzog prelaska.

Karakteristično za ovu vrstu baza podataka je to što ona predstavlja kolekciju dokumenata. Dakle, sve informacije iz dokumenta, nalaze se u samom dokumentu, odnosno nema nikakvih povezanih baza i nema definisane šeme podataka. Takodje, dokumenti ne moraju imati istu strukturu. Ukoliko je potrebno da se doda novo polje, ono će se dodati bez uticaja na ostala polja ili ostale dokumente. Objekti se čuvaju kao dokumenti i medjusobno su nezavisni.

Otvoreni formati su :

* XML,
* JSON,
* BSON(Binary JSON),
* YALM..

Tri ključna faktora razlikuju baze podataka dokumenata od relacionih baza podataka:

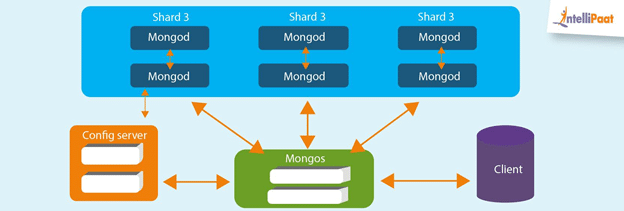
1. Intuitivnost modela podataka: Dokumenti se mapiraju na objekte u kodu, tako da je mnogo prirodnije raditi sa njima. Nema potrebe za dekomponovanjem podataka po tabelama. Podaci kojima se pristupa zajedno se čuvaju zajedno, tako da programeri imaju manje koda za pisanje, a krajnji korisnici imaju bolje performanse.

2. Sveprisutnost JSON dokumenata: JSON je postao ustaljeni standard za razmenu i skladištenje podataka. JSON dokumenti su lagani, nezavisni od jezika i čitljivi. Dokumenti su nadskup svih drugih modela podataka, tako da programeri mogu strukturirati podatke na način na koji je potreban njihovim aplikacijama — bogati objekti, parovi ključ-vrednost, tabele, geoprostorni podaci i podaci vremenskih serija ili čvorovi i ivice grafikona.

3. Fleksibilnost šeme: Šema dokumenta je dinamična i samoopisna, tako da programeri ne moraju prvo da je unapred definišu u bazi podataka. Polja se mogu razlikovati od dokumenta do dokumenta. Programeri mogu da izmene strukturu u bilo kom trenutku, izbegavajući ometajuće migracije šeme.

# Interna struktura MongoDB-ja

Kao što je već rečeno MongoDB je dokumentno orijentisana baza podataka koja čuva podatke u kolekcijama, koje se sastoje od pojedinačnih dokumenata. Interna arhitektura MongoDB-a je dizajnirana da bude visoko skalabilna, fleksibilna i pouzdana i zasnovana je na brojnim ključnim komponentama. Šema arhitekture MongoDB-ja prikazana je na sledećoj slici, a u nastavku teksta biće opisani osnovni koncepti ove baze.

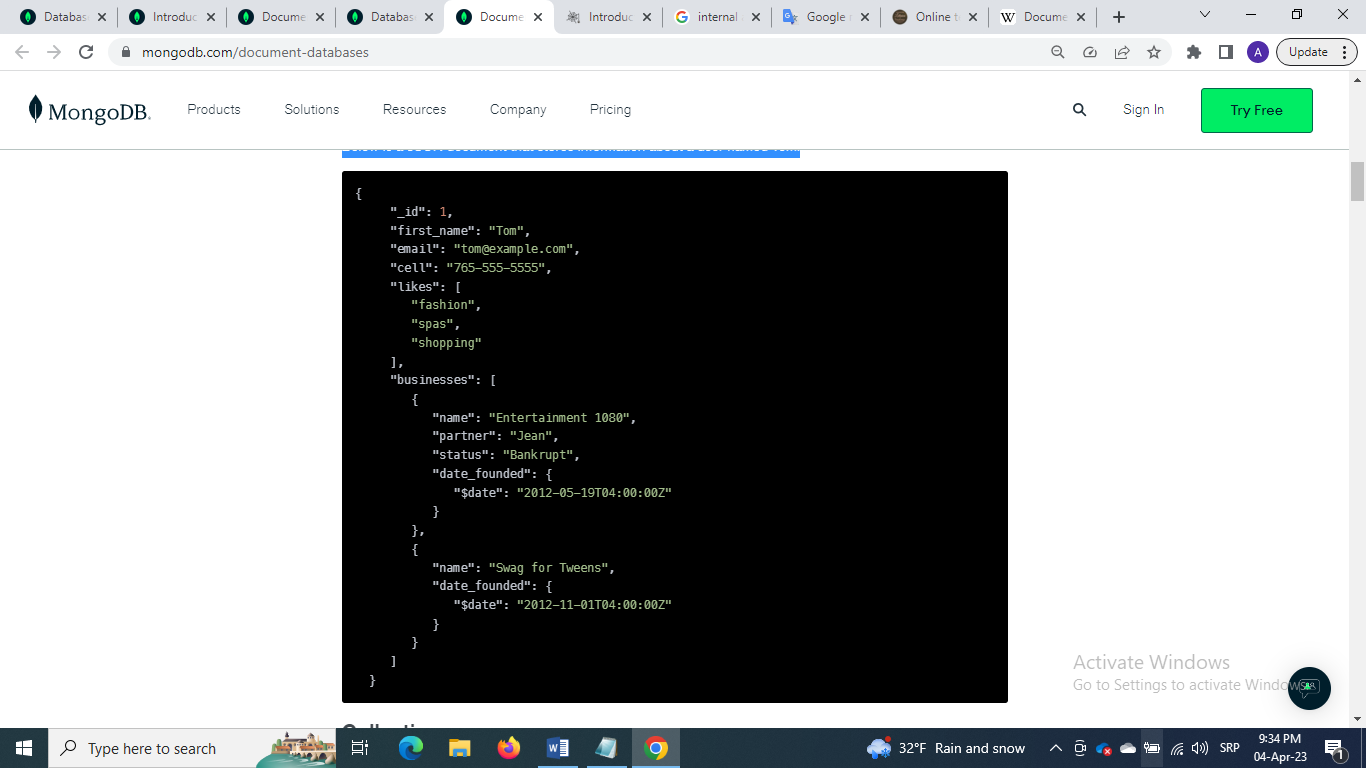


Slika 3. Šematski prikaz interne strukture MongoDB-ja

## Osnovni koncepti

Kao što je već rečeno centralni koncept dokumentno orijentisane baze podataka, samim tim i MongoDB baze je pojam dokumenta. Iako se svaka implementacija baze podataka orijentisana na dokumente razlikuje samo u nekim sitnim detaljima , generalno, sve one pretpostavljaju da dokumenti inkapsuliraju i kodiraju podatke (ili informacije) u nekom standardnom formatu ili kodiranju. Kodiranja koja se koriste uključuju XML, YAML, JSON, kao i binarne forme kao što je BSON.

Primer JSON dokumenta o korisniku po imenu Tom prikazan je na sledećoj slici.

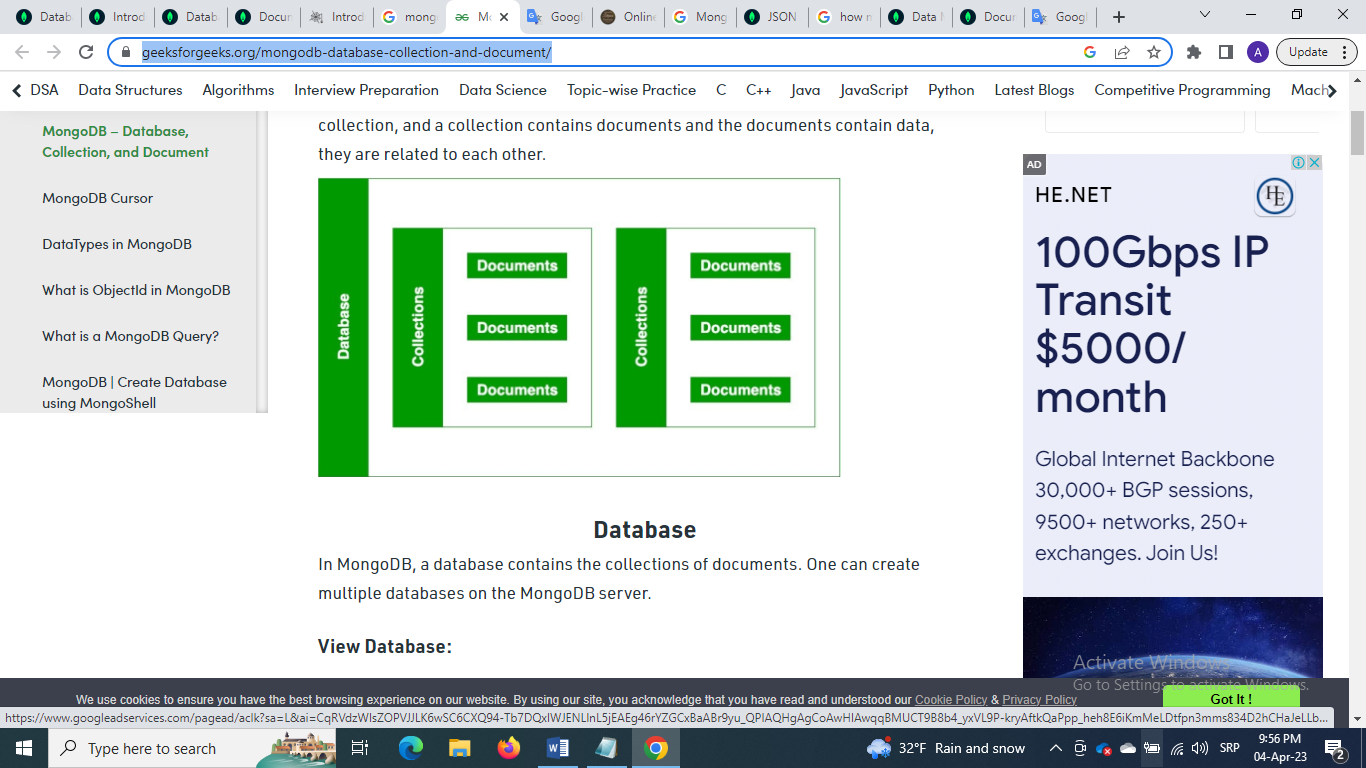


Slika 4. JSON dokument

Pored dokumenta, drugi važan koncept MongoDB-ja jeste kolekcija. Kolekcija predstavlja grupu dokumenata. U jednoj bazi podataka je dozvoljeno skladištenje više kolekcija. Kolekcije obično čuvaju dokumente koji imaju sličan sadržaj. Ne moraju svi dokumenti u kolekciji da imaju ista polja, jer baze podataka dokumenata imaju fleksibilnu šemu. Dakle, u kolekciji nije neophodno da je šema jednog dokumenta slična drugom dokumentu.

Nastavljajući sa gornjim primerom, dokument sa informacijama o Tomu mogao bi da se uskladišti u kolekciji pod nazivom korisnici. Više dokumenata bi se moglo dodati u kolekciju korisnika kako bi se dopunile informacije o drugim korisnicima. Na primer, dokument ispod koji čuva informacije o drugom korisniku mogao bi da se doda u kolekciju korisnika.

Na kraju, u MongoDB-ju, baza podataka sadrži zbirke dokumenata. Može se kreirati više baza podataka na MongoDB serveru. Jednostavnim rečima, može se nazvati fizičkim kontejnerom za podatke. Svaka od baza podataka ima svoj skup datoteka u sistemu datoteka sa više baza podataka koje postoje na jednom MongoDB serveru. Šema prethodno objašnjenih pojmova prikazana je na sledećoj slici.



Slika 5. Struktura baze

# Pored pored baze podataka, kolekcija i dokumenata priča o MongoDB-ju ne bi bila potpuna bez pomena indeksa.

Indeksi podržavaju efikasno izvršavanje upita u MongoDB-u. Bez indeksa, MongoDB mora da izvrši skeniranje kolekcije, tj. da skenira svaki dokument u kolekciji, da bi izabrao one dokumente koji odgovaraju naredbi upita. Ako postoji odgovarajući indeks za upit, MongoDB može koristiti indeks da ograniči broj dokumenata koje mora da pregleda.

Kada kreirate indeks u MongoDB-u, server gradi strukturu podataka B-stabla ili novije verzije B+ stabla za taj indeks. Svaki čvor u B-stablu sadrži opseg ključeva i referencu na svoje podređene čvorove. Na nivou lista B-stabla, svaki čvor sadrži pokazivač na stvarni dokument ili dokumente u kolekciji.

Kada izvršite upit koji može da koristi indeks, server prelazi preko B-stabla da pronađe odgovarajuće dokumente. Pošto su ključevi uskladišteni u sortiranom redosledu, server može da koristi algoritam binarne pretrage da brzo pronađe relevantne čvorove u stablu.

U nastavku biće opisani primarni i sekundarni indeksi.

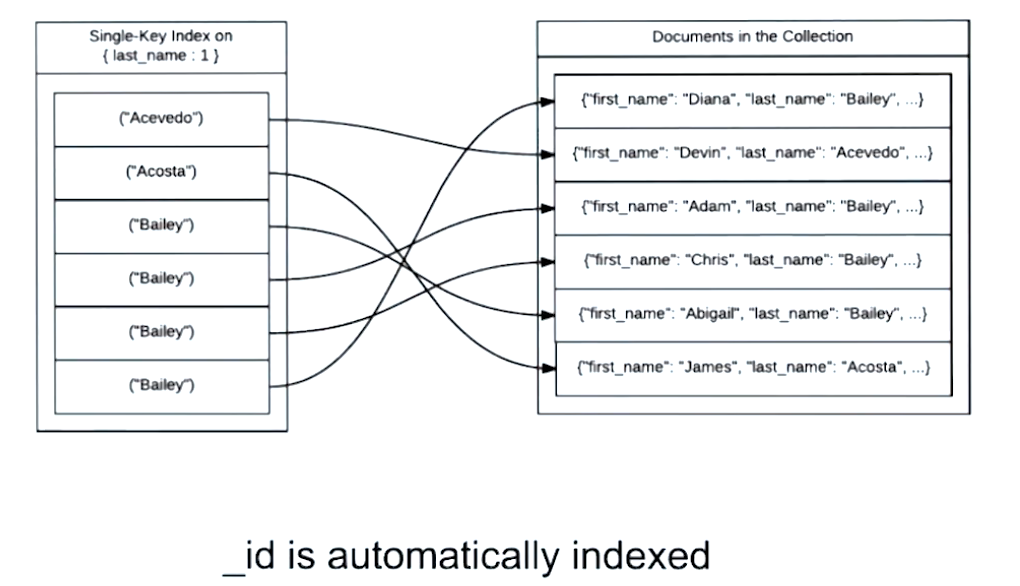
Što se tiče primarnog indeksa, kada se kreira kolekcija u Mongo-u, primarni ključ \_id koji predstavlja ID dokumenta kreira se pored indeksa B+Tree tako da je pretraga optimalna. \_id jedinstveno identifikuje dokument i može se koristiti za pronalaženje dokumenta.

Tip \_id je objectId i to je polje od 12 bajtova. Razlog zašto je velika jer ga Mongo koristi za jedinstvenu identifikaciju dokumenta na mašinama ili delovima radi skalabilnosti. Takođe korisnik može zameniti polje \_id sa vrednošću po svom izboru što bi moglo učiniti ključ još većim.

Primarni indeks \_id se koristi za mapiranje \_id-a u BSON dokument kroz B+Tree strukturu. Samo mapiranje je prošlo kroz faze kako se MongoDB razvijao.

Što se tiče sekundarnog indeksa, korisnici mogu da kreiraju sekundarne B+Tree indekse na bilo kom polju u kolekciji koje zatim upućuje na BSON dokumente koji zadovoljavaju indeks. Ovo je veoma korisno za omogućavanje brzog prelaska koristeći različita polja u dokumentu, a ne samo podrazumevano polje \_id. Bez sekundarnih indeksa, Mongo mora da izvrši kompletno skeniranje kolekcije tražeći polja dokumenta jedno po jedno.

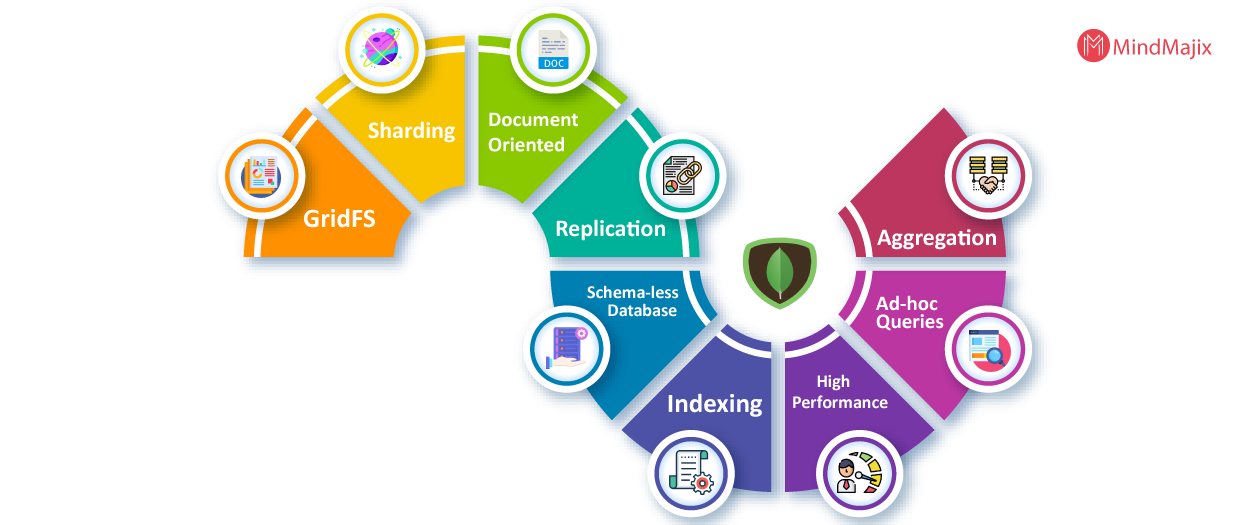
Veličina sekundarnog indeksa zavisi od dve stvari, veličine ključa koja predstavlja veličinu polja koje se indeksira i veličine pokazivača dokumenta.



Slika 6. Primer za indeksiranje

## Karakteristike MongoDB-ja

Na samom kraju priče o internoj strukturi i arhitekturi MongoDB-ja potrebno je spomenuti i ukratko opisati sve njegove karakteristike od kojih su neke od njih već detaljno objašnjene ili će tek biti pomenute i objašnjene u nastavku.



Slika 7. Karakteristike MongoDB-ja

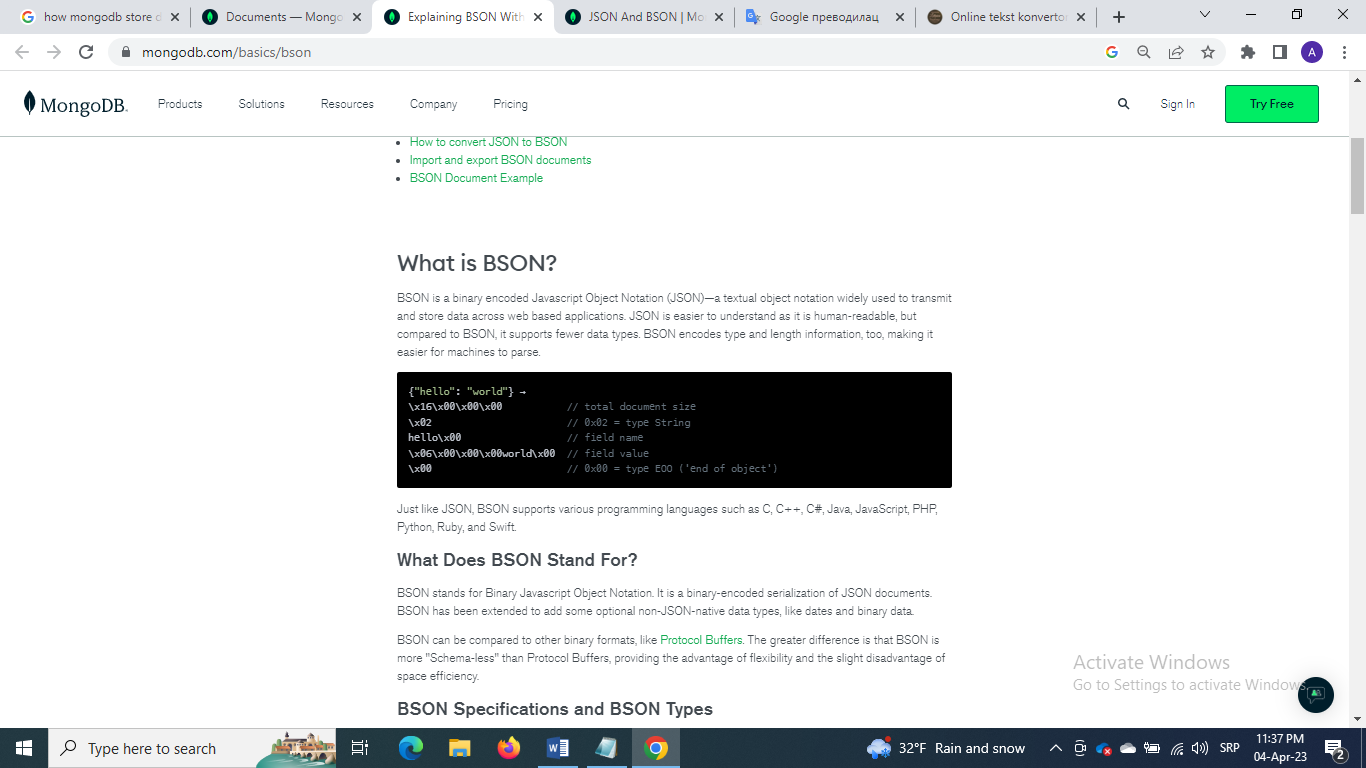
* Podrška za upite: MongoDB podržava ad-hoc upite i upite zasnovane na dokumentima.
* Indeksna podrška: Sva polja u dokumentu su indeksirana.
* Replikacija: MongoDB poseduje Master-Slave replikaciju. Koristi matičnu aplikaciju za sprečavanje zastoja baze podataka održavanjem više kopija podataka.
* Više servera: Duplirani podaci koji se čuvaju u bazi podataka se pokreću na više servera kako bi se izbegla šteta uzrokovana kvarom hardvera.
* Automatsko deljenje: Podaci se distribuiraju između nekoliko fizičkih particija poznatih kao delovi. MongoDB ima ugrađenu funkciju koja se zove automatsko balansiranje opterećenja.
* MapReduce: To je fleksibilan alat za agregaciju koji podržava funkciju MapReduce.
* Rukovanje greškama: U MongoDB-u, efikasno radi u slučaju kvarova kao što su kvarovi više mašina, kvarovi data centra tako što štiti podatke i čini ih dostupnim.
* GridFS: Ova funkcija će omogućiti da se datoteke podele na manje delove i pohranjuju u različite dokumente bez komplikovanja steka.
* Baza podataka bez šema: MongoDB je baza podataka bez šema programirana na jeziku C++.
* Skladištenje orijentisano na dokumente: Koristi BSON format koji je sličan JSON-u.
* Procedure: MongoDB JavaScript radi bolje od procedura jer baze podataka više koriste jezik nego procedure.

# Skladištenje podataka

Na mnogim sajtovima može se pronaći informacija o tome da MongoDB skladišti podatke u JSON formatu, medjutim to nije tako. MongoDB za razliku od sistema koji čuvaju JSON kao vrednosti kodirane u stringovima ili binarno kodirane blobove, MongoDB koristi BSON da ponudi moćne funkcije indeksiranja i upita.

## BSON format

BSON je binarno kodirana Javascript objektna notacija (JSON) — notacija tekstualnih objekata koja se široko koristi za prenos i skladištenje podataka preko web aplikacija. JSON je lakši za razumevanje jer je čitljiv ljudima, ali u poređenju sa BSON-om, podržava manje tipova podataka. BSON takođe kodira informacije o tipu i dužini.



Slika 5. Primer BSON fajla

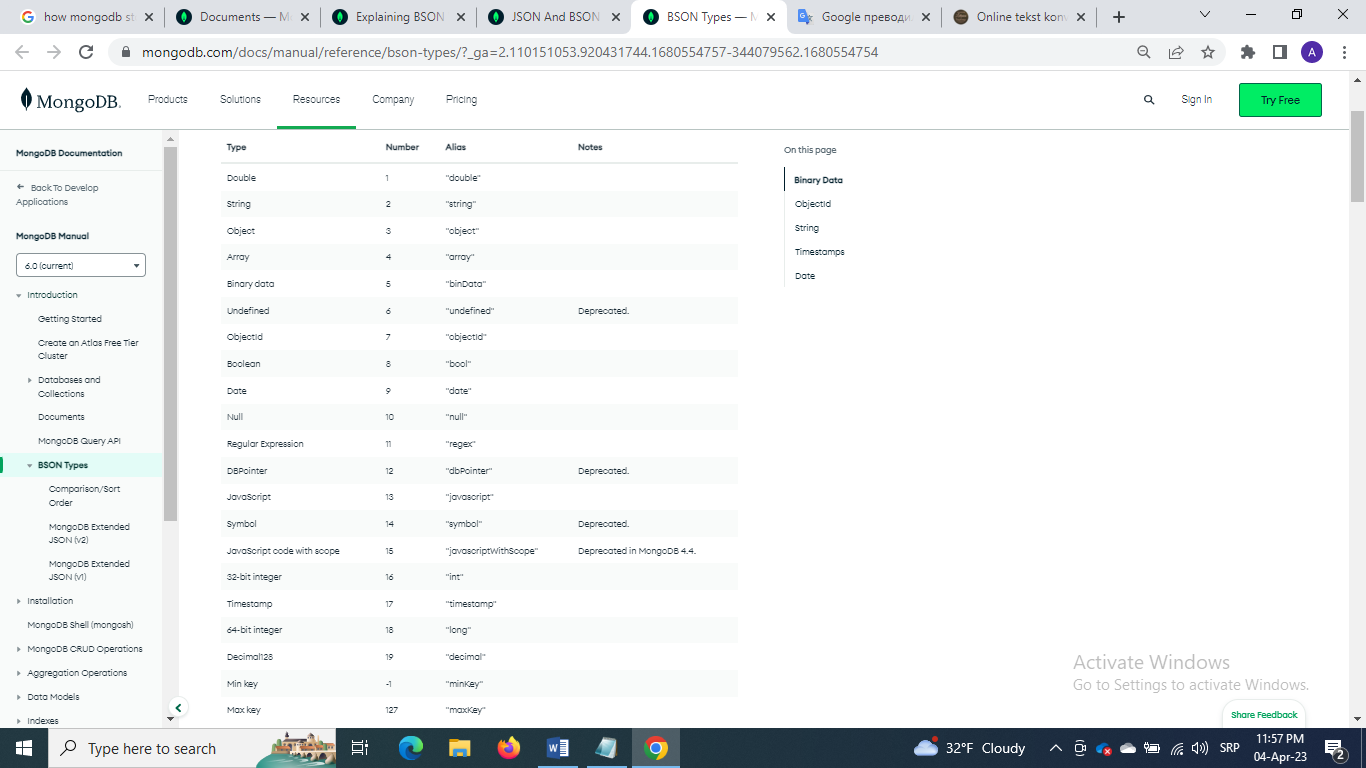
Kada se koristi MongoDB drajver za odredjeni programski jezik, radi se sa izvornim strukturama podataka za taj jezik. Drajver će se pobrinuti za konvertovanje podataka u BSON i nazad kada postavlja upit bazi podataka.

JSON i BSON su izuzetno slični po dizajnu. BSON je dizajniran kao binarni prikaz JSON podataka, sa specifičnim proširenjima za šire aplikacije i optimizovan za skladištenje. Baš kao i JSON, BSON podržava ugrađivanje objekata i nizova.Jedan poseban način na koji se BSON razlikuje od JSON-a je njegova podrška za neke naprednije tipove podataka. JSON, na primer, ne pravi razliku između celih brojeva i brojeva sa pokretnim zarezom (koji imaju decimalnu preciznost do različitih stepeni).

BSON dokumenti mogu da sadrže datum ili binarne objekte koji nisu izvorno predstavljeni u čistom JSON-u. Drugo, svaki programski jezik ima sopstvenu semantiku objekata. JSON objekti imaju uređene ključeve, na primer, dok su Python rečnici neuređeni. Treće, BSON podržava različite numeričke tipove koji nisu izvorni za JSON, a mnogi jezici ih predstavljaju drugačije.

Baš kao i JSON, BSON podržava različite programske jezike kao što su C, C++, C#, Java, JavaScript, PHP, Python, Ruby i Swift.

Tipovi podataka koje podržava BSON prikazani su na sledećoj slici.



Slika 6. Tipovi podataka koje BSON podržava

Kao što se može videti BSON zaista podržava ogroman broj različitih tipova podataka, što zaista i jeste jedan od njegovih velikih prednosti kao što je već rečeno. Medjutim, može se primetiti da pored standarnih tipova, ovaj format podržava i neke specifične tipove koji nisu karakteristični za druge formate. Neki od karakterističnih tipova su ObjectId i timestamp.

**ObjectId**-ovi su male veličine, jedinstveni su, brzo se generišu i poređaju. Vrednosti ObjectId-a su dužine 12 bajtova, koje se sastoje od:

* Vremenska oznaka od 4 bajta, koja predstavlja kreiranje ObjectId-a, mereno u sekundama od Unix epohe.
* 5-bajtna nasumična vrednost koja se generiše jednom po procesu. Ova nasumična vrednost je jedinstvena za mašinu i proces.
* Brojač sa povećanjem od 3 bajta, inicijalizovan na slučajnu vrednost.

Ako se vrednost celog broja koristi za kreiranje ObjectId-a, ceo broj zamenjuje vremensku oznaku.

U MongoDB-u, svaki dokument uskladišten u kolekciji zahteva jedinstveno polje \_id koje deluje kao primarni ključ. Ako umetnuti dokument izostavi polje \_id, MongoDB drajver automatski generiše jedan.

MongoDB klijenti treba da dodaju polje \_id sa jedinstvenim ObjectId-om. Korišćenje ObjectIds-a za polje \_id pruža sledeće dodatne prednosti:

* Moguće je pristupiti vremenu kreiranja ObjectId-a, koristeći metod *ObjectId.getTimestamp()*,
* sortiranje na polju \_id koje čuva ObjectId vrednosti je otprilike ekvivalentno sortiranju prema vremenu kreiranja.

**Vremenske oznake (time stamps)** su BSON poseban tip vremenske oznake za internu upotrebu MongoDB-a i nije povezan sa regularnim tipom datuma. Ovaj interni tip vremenske oznake je 64-bitna vrednost gde:

* najznačajnija 32 bita su vrednost time\_t (sekunde od Unix epohe)
* najmanje značajna 32 bita su rastući redni broj za operacije unutar date sekunde.

Vremenska oznaka MongoDB izgleda slično tipu podataka Datum, gde ima istu 64-bitnu vrednost. Ali takođe ima nekoliko aspekata u kojima se razlikuje od datuma. Vremenska oznaka MongoDB se prilično koristi za internu svrhu, a sa svakom pojedinačnom instancom mongodb-a, vrednosti generisane za vremensku oznaku su jedinstvene.

Pored raznovrsnog asortimana tipova podataka sa kojima mogu raditi još jedna od velikih atrakcija za programere koji koriste baze podataka sa BSON modelima podataka je dinamička i fleksibilna šema koju pružaju u poređenju sa krutim, tabelarnim modelima podataka koje koriste relacione baze podataka.

Prvo, MongoDB dokumenti su polimorfni — polja mogu da variraju od dokumenta do dokumenta unutar jedne kolekcije (analogno tabelama u relacionoj bazi podataka). Ova fleksibilnost olakšava modeliranje podataka bilo koje strukture i prilagođavanje modela kako se zahtevi menjaju.

Drugo, nema potrebe da se deklariše struktura dokumenata u bazi podataka – dokumenti se sami opisuju.

Treće, ako dokumentu treba dodati novo polje, ono se može kreirati bez uticaja na sve ostale dokumente u kolekciji, bez ažuriranja centralnog sistemskog kataloga i bez isključivanja baze podataka. Kada treba da unesete izmene u model podataka, baza podataka dokumenata nastavlja da skladišti ažurirane objekte bez potrebe za izvođenjem skupih ALTER TABLE operacija — ili još gore, redizajniranje šeme od nule.

Kroz ove prednosti, fleksibilnost modela podataka dokumenta je dobro prilagođena zahtevima savremenih praksi razvoja aplikacija.

Iako je fleksibilna šema moćna karakteristika, postoje situacije u kojima je potrebno više kontrole nad strukturom podataka i sadržajem dokumenata. Većina baza podataka dokumenata gura primenu ovih kontrola nazad na programera da ih primeni u kodu aplikacije. Međutim, naprednije baze podataka dokumenata obezbeđuju validaciju šeme, koristeći pristupe kao što je standard IETF JSON šeme koji je usvojio MongoDB.

## Ograničenja veličine BSON fajlova

Iako je do sada više puta napomenuto da se MongoDB koristi za skladištenje ogromnih količina podataka, ipak postoi ograničenje veličino BSON fajla. Maksimalna veličina BSON dokumenta je 16 megabajta. Takodje, postoji i ograničenje i za ugnježdenu dubinu. MongoDB podržava ne više od 100 nivoa ugnežđenja za BSON dokumente. Svaki objekat ili niz dodaje nivo.

Maksimalna veličina dokumenta pomaže da se osigura da jedan dokument ne može da koristi preveliku količinu RAM-a ili, tokom prenosa, preveliku količinu propusnog opsega. Medjutim, naravno da postoji mehanizam koji omogućava skladištenje dokumenata ili kolekcija koji prevazilaze ovu veličinu. Za skladištenje dokumenata većih od maksimalne veličine, MongoDB obezbeđuje GridFS API.

## GridFS API

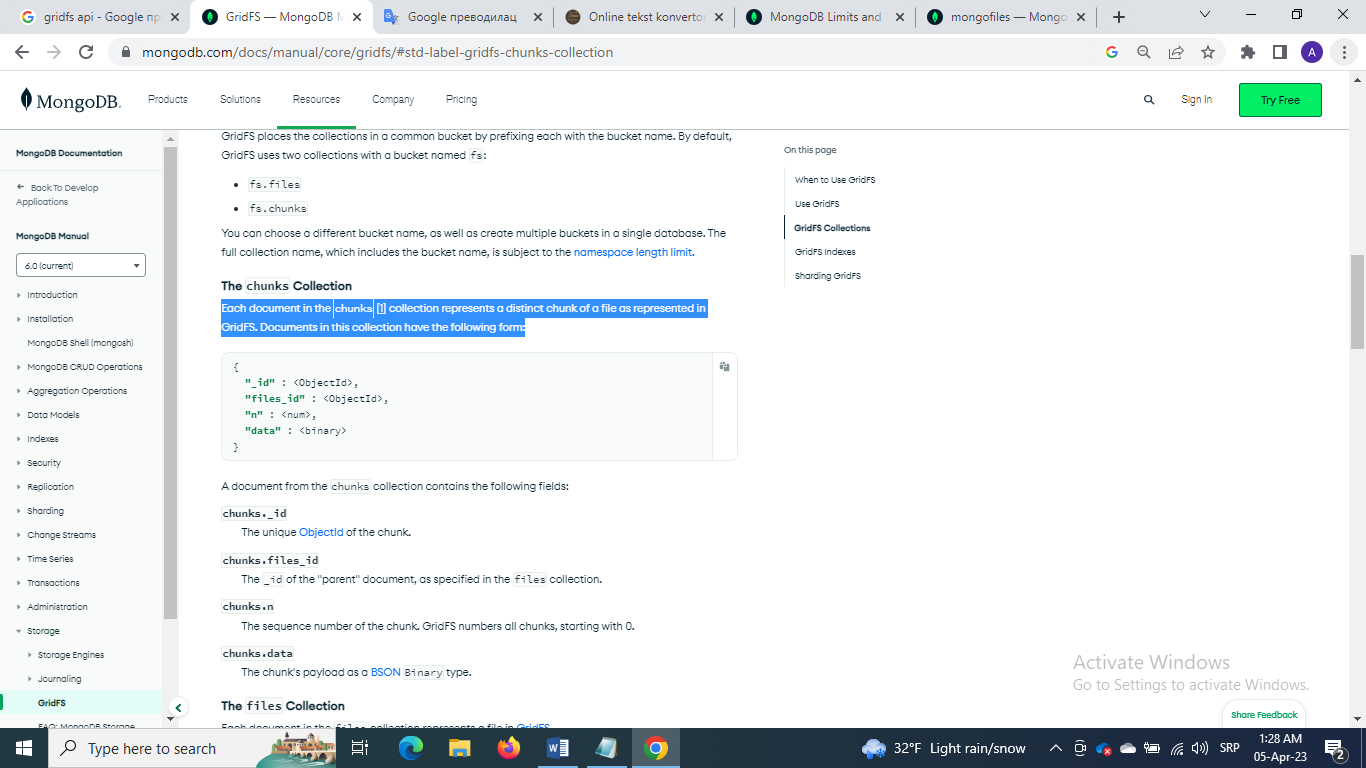
GridFS je specifikacija za čuvanje i preuzimanje datoteka koje premašuju ograničenje veličine BSON dokumenta od 16 MB.

Umesto da skladišti datoteku u jedan dokument, GridFS deli datoteku na *files* i *chunks* i čuva svaki deo kao poseban dokument. GridFS koristi podrazumevanu veličinu komada od 255 kB, to jest, GridFS deli datoteku na delove od 255 kB sa izuzetkom poslednjeg dela. Poslednji de je samo onolike veličine koliko je potrebno. Slično tome, datoteke koje nisu veće od veličine komada imaju samo završni komad, koristeći samo onoliko prostora koliko je potrebno plus neke dodatne metapodatke.

GridFS koristi dve kolekcije za skladištenje datoteka. Jedna kolekcija čuva delove datoteka, a druga čuva metapodatke datoteka.

GridFS je koristan ne samo za čuvanje datoteka koje prelaze 16 MB, već i za čuvanje svih datoteka kojima želite da pristupite bez potrebe da učitavate celu datoteku u memoriju. Takode, ako vaš sistem datoteka ograničava broj datoteka u direktorijumu, korišćenje GridFS-a može da pomogne da se uskladišti samo onoliko datoteka koliko je potrebno.

Kao što je rečeno GridFS deli datoteku na dva dela. Prvi deo jeste chunks. Svaki dokument u kolekciji chunks predstavlja poseban komad datoteke kao što je predstavljeno u GridFS. Dokumenti u ovoj kolekciji imaju sledeći oblik:

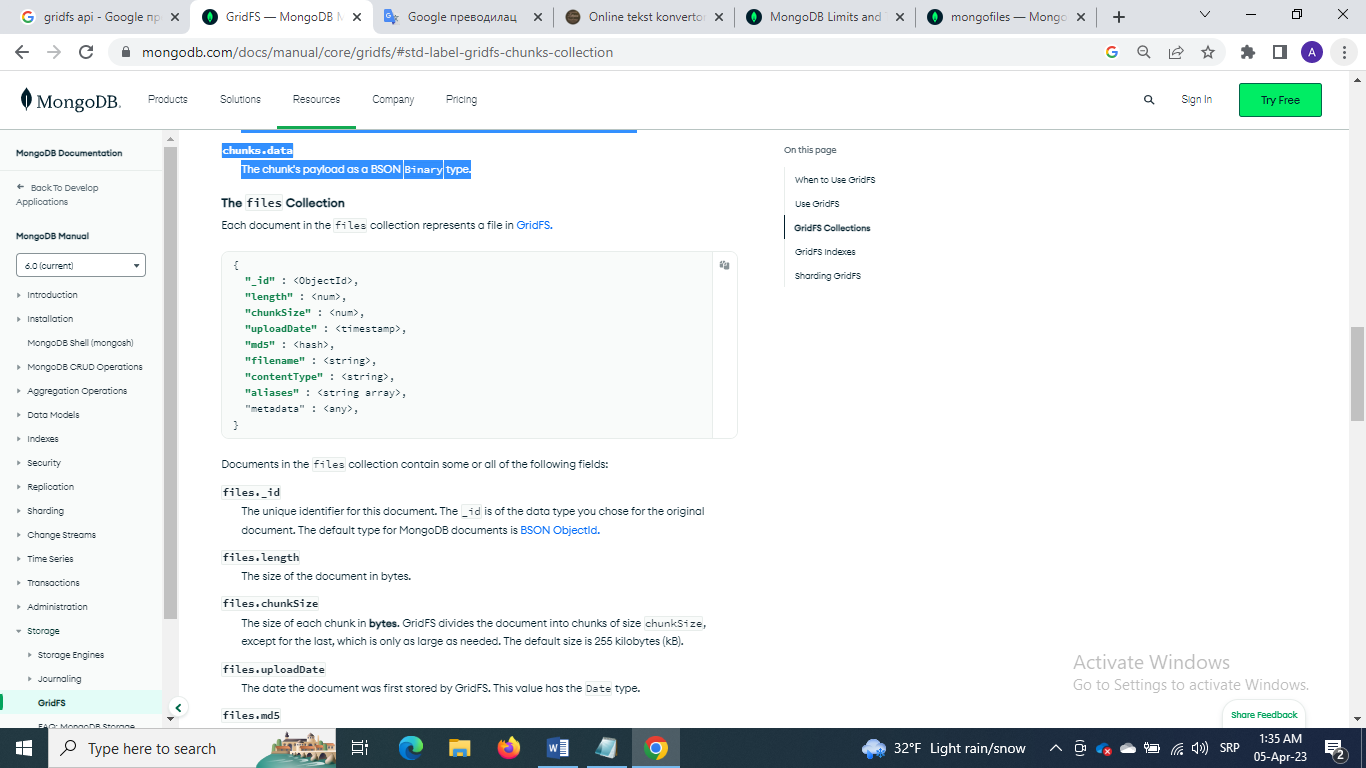


Slika 7. Struktura chunks kolekcije

Dokument iz kolekcije komada sadrži sledeća polja:

* chunks.\_id-Jedinstveni ObjectId komada.
* chunks.files\_id-\_id "roditeljskog" dokumenta, kako je navedeno u kolekciji datoteka.
* chunks.n-Redni broj chunk-a. GridFS numeriše sve delove, počevši od 0.
* chunks.data-Podaci kao BSON binarni tip.

Druga kolekcija jeste files i njena struktura je prikazana na sledećoj slici.



Slika 8. Struktura files kolekcije

Dokumenti u kolekciji files sadrže neka ili sva sledeća polja:

* files.\_id-Jedinstveni identifikator za ovaj dokument. \_id je tipa podataka koji ste izabrali za originalni dokument. Podrazumevani tip za MongoDB dokumente je BSON ObjectId.
* files.length-Veličina dokumenta u bajtovima.
* files.chunkSize-Veličina svakog komada u bajtovima. GridFS deli dokument na delove veličine chunkSize, osim poslednjeg, koji je samo onoliko koliko je potrebno. Podrazumevana veličina je 255 kilobajta (kB).
* files.uploadDate-Datum kada je dokument prvi put sačuvao GridFS. Ova vrednost ima tip datuma.
* files.md5- (Zastarelo) MD5 algoritam je zabranjen prema FIPS 140-2. MongoDB drajveri odbijaju podršku za MD5 i ukloniće MD5 generaciju u budućim izdanjima. Aplikacije koje zahtevaju sažetak datoteke treba da ga implementiraju izvan GridFS-a i da ga čuvaju.
* files.metadata -MD5 heš kompletne datoteke koju vraća naredba filemd5. Ova vrednost ima tip String.
* files.fileName-Opcionalno. Čovjeku čitljivo ime za datoteku GridFS.
* files.contentTipe- Zastarelo.Opciono. Važeći MIME tip za datoteku GridFS. Samo za aplikaciju.
* files.aliases-(Zastarelo)Opciono. Niz nizova alijasa. Samo za aplikaciju.
* files.metadata- Opciono. Polje metapodataka može biti bilo kog tipa i može da sadrži sve dodatne informacije koje želite da sačuvate. Ako želite da dodate dodatna proizvoljna polja dokumentima u kolekciji datoteka, dodajte ih objektu u polju metapodataka.

## Mehanizmi za skladištenje podataka

Mehanizam za skladištenje je primarna komponenta MongoDB-a odgovorna za upravljanje podacima. MongoDB pruža različite mehanizme za skladištenje, omogućavajući izbor onog koji najviše odgovara aplikaciji i primeni.

Mehanizam za skladištenje je komponenta baze podataka koja je odgovorna za upravljanje načinom na koji se podaci čuvaju, kako u memoriji tako i na disku. MongoDB podržava više mehanizama za skladištenje, jer različiti motori rade bolje za određena radna opterećenja. Odabir odgovarajućeg mehanizma za skladištenje za odredjeni slučaj upotrebe može značajno uticati na performanse aplikacija.

### WiredTiger Storage Engine (podrazumevano)

WiredTiger je podrazumevani mehanizam za skladištenje koji počinje u MongoDB 3.2. Pogodan je za većinu radnih opterećenja i preporučuje se za nove primene. WiredTiger između ostalih funkcija pruža model istovremenosti na nivou dokumenta, kontrolne tačke i kompresiju.

WiredTiger koristi kontrolu konkurentnosti na nivou dokumenta za operacije pisanja. Kao rezultat toga, više klijenata može istovremeno da menja različite dokumente kolekcije.

Dakle, ovaj mehanizam koristi optimistične algoritme kontrole paralelnosti. Ovo izbegava usko grlo centralizovanog menadžera zaključavanja i znači da transakcijske operacije ne blokiraju: čitanje ne blokira upisivanje i obrnuto. Dalje, upisi ne blokiraju upisivanje, ali upisivanje u različite dokumente. Medjutim logično je da istovremene transakcije koje ažuriraju istu vrednost neće uspeti i nisu dozvoljene jer izazivaju takozvani *deadlock*. Neke aplikacije mogu imati koristi od sinhronizacije na nivou aplikacije kako bi se izbegli ponovljeni pokušaji vraćanja i ažuriranja iste vrednosti..

#### Checkpoints

WiredTiger koristi MultiVersion Concurrenci Control (MVCC). Na početku operacije, WiredTiger obezbeđuje snimak podataka u trenutku operacije. Snimak predstavlja konzistentan prikaz podataka u memoriji. WiredTiger podržava kontrolne tačke kao statički prikaz samo za čitanje jednog ili više izvora podataka.

Kada piše na disk, WiredTiger upisuje sve podatke u snimku (snapshot) na disk na konzistentan način u svim datotekama sa podacima. Trajni podaci deluju kao kontrolna tačka u datotekama podataka. Kontrolna tačka osigurava da su datoteke podataka konzistentne do i uključujući poslednju kontrolnu tačku; odnosno kontrolni punktovi mogu delovati kao tačke oporavka.

Počevši od verzije 3.6, MongoDB konfiguriše WiredTiger za kreiranje kontrolnih tačaka (tj. zapisivanje snimaka podataka na disk) u intervalima od 60 sekundi. U ranijim verzijama, MongoDB postavlja kontrolne tačke da se pojavljuju u WiredTiger-u na korisničkim podacima u intervalu od 60 sekundi ili kada je upisano 2 GB podataka dnevnika, šta god se prvo dogodi.

Tokom pisanja nove kontrolne tačke, prethodna kontrolna tačka je i dalje važeća. Kao takav, čak i ako MongoDB prekine ili naiđe na grešku tokom pisanja nove kontrolne tačke, nakon ponovnog pokretanja, MongoDB se može oporaviti od poslednje važeće kontrolne tačke.

Nova kontrolna tačka postaje dostupna i trajna kada se WiredTigerova tabela metapodataka atomski ažurira da bi se referencirala na novu kontrolnu tačku. Kada je nova kontrolna tačka dostupna, ViredTiger oslobađa stranice sa starih kontrolnih tačaka.

### Journal

WiredTiger koristi journal za upisivanje unapred u kombinaciji sa kontrolnim tačkama kako bi se osigurala trajnost podataka.

WiredTiger održava u journal-u sve izmene podataka između kontrolnih tačaka. Ako MongoDB izađe između kontrolnih tačaka, koristi dnevnik da ponovo reprodukuje sve podatke izmenjene od poslednje kontrolne tačke. Za informacije o učestalosti sa kojom MongoDB upisuje podatke dnevnika na disk.

ViredTiger časopis je komprimovan pomoću biblioteke brze kompresije. Metode kopresije je moguće promeniti, a više informacija o kompresiji biće u nastavku.

### Kompresija

Sa WiredTiger-om, MongoDB podržava kompresiju za sve kolekcije i indekse. Kompresija minimizira upotrebu skladišnog prostora na račun dodatnog CPU-a.

Podrazumevano, koristi se kompresija blokova sa bibliotekom brze kompresije za sve kolekcije i kompresija prefiksa za sve indekse.

Za kolekcije su dostupne i sledeće biblioteke kompresije blokova:

* zlib
* zstd (dostupno počevši od MongoDB 4.2)
* snappy

Kada se podaci zapisuju na disk, MongoDB ih komprimuje pomoću specificiranog metoda kompresije bloka i zatim ih zapisuje na disk. Kada se ovaj blok podataka pročita, on ga dekompresuje u memoriji i predstavlja dolaznom zahtevu.

Kompresija blokova je vrsta kompresije koja komprimuje podatke u blokove umesto da kompresuje ceo skup podataka odjednom. Blok kompresija može poboljšati performanse omogućavajući čitanje i pisanje podataka u manjim komadima.

MongoDB podrazumevano obezbeđuje brz metod kompresije blokova za skladištenje i mrežnu komunikaciju.

*Snappy* kompresija je dizajnirana da bude brza i efikasna u pogledu korišćenja memorije, što je čini dobrom za radna opterećenja MongoDB. Snappy je biblioteka kompresije koju je razvio Google.

Prednosti Snappy kompresije u MongoDB-u:

* Velike brzine kompresije i dekompresije,
* Niska upotreba CPU-a,
* Format koji se može strimovati i koji omogućava brzu obradu,
* Minimalni uticaj na performanse upita.

*Zstandard Compression* ili zstd, još jedan noviji metod kompresije blokova koji obezbeđuje MongoDB počevši od v4.2, obezbeđuje veće stope kompresije. Zstd je biblioteka kompresije koju je razvio Facebook.

Zstd obično nudi veći odnos kompresije od prethodno pomenutog metoda, što znači da može efikasnije kompresovati podatke i postići manju komprimovanu veličinu za iste ulazne podatke.

Prednosti zstd kompresije u MongoDB:

* Veći stepen kompresije od Snappija,
* Visoko konfigurabilni nivoi kompresije,
* Velike brzine kompresije i dekompresije,
* Minimalni uticaj na performanse upita.

### In-Memori Storage Engine

In-Memory Storage Engine je dostupan u MongoDB Enterprise. Umesto da čuva dokumente na disku, on ih zadržava u memoriji radi predvidljivijih kašnjenja podataka. Osim nekih metapodataka i dijagnostičkih podataka, mehanizam za skladištenje u memoriji ne održava nikakve podatke na disku, uključujući konfiguracione podatke, indekse, korisničke akreditive itd.

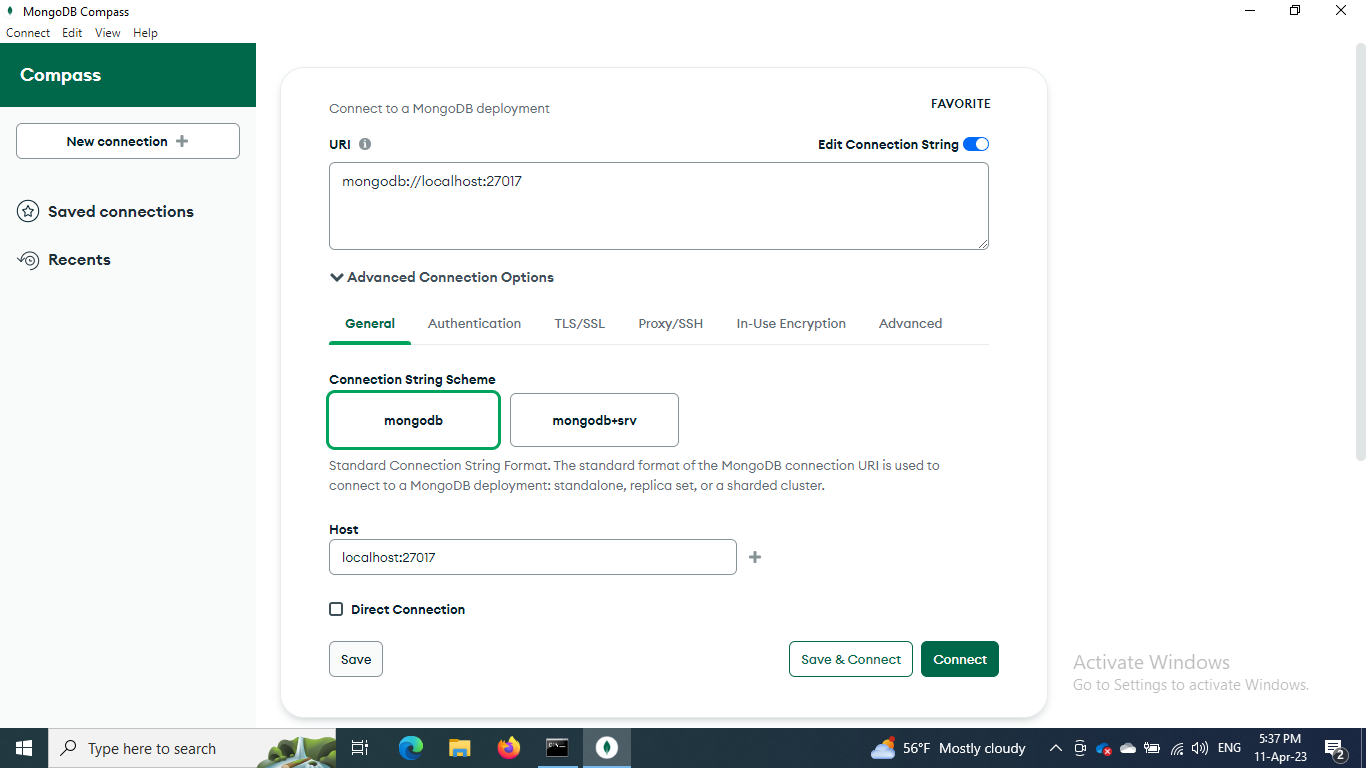
Mehanizam za skladištenje u memoriji koristi kontrolu istovremenosti na nivou dokumenta za operacije pisanja. Kao rezultat toga, više klijenata može istovremeno da menja različite dokumente kolekcije.

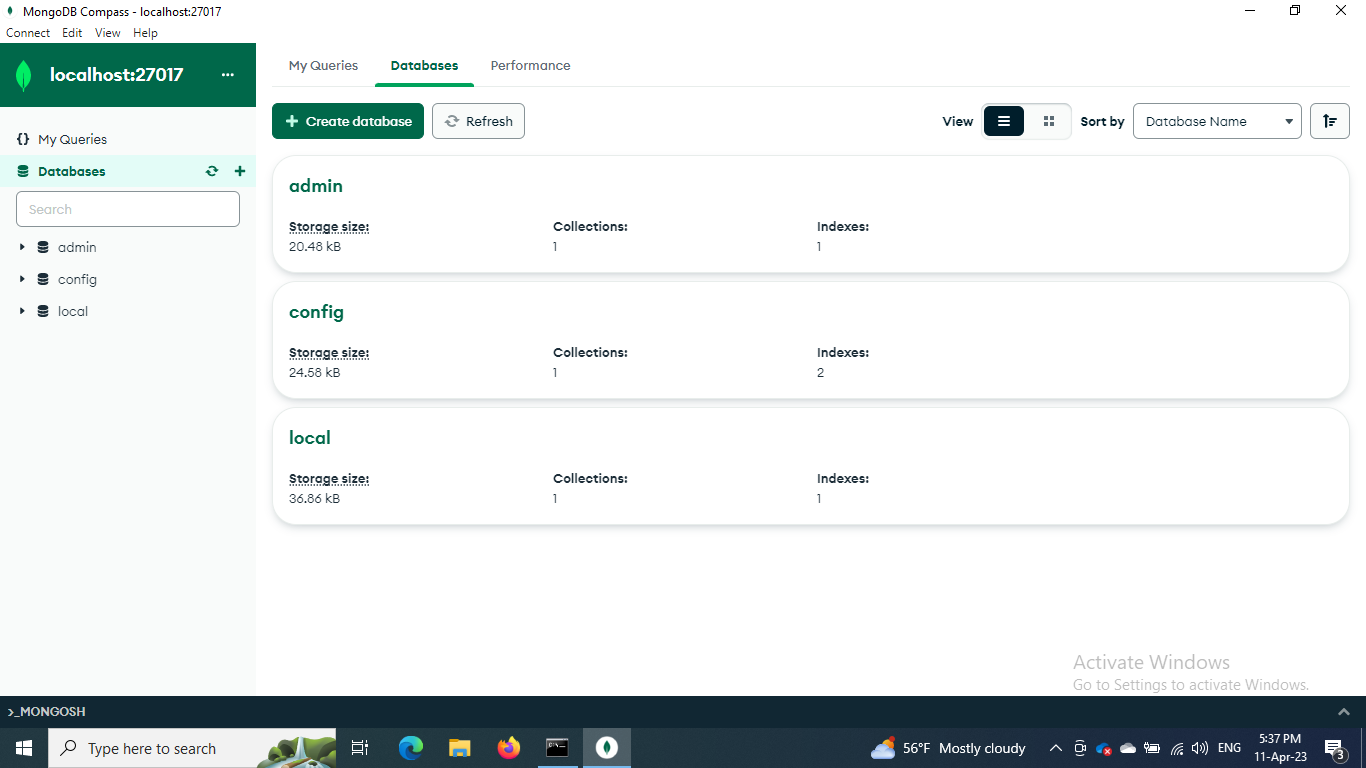
Podrazumevano, mehanizam za skladištenje u memoriji koristi 50% fizičke RAM memorije minus 1 GB. Ako bi operacija pisanja prouzrokovala da podaci premaše navedenu veličinu memorije, MongoDB prijavljuje grešku.

# Praktični deo

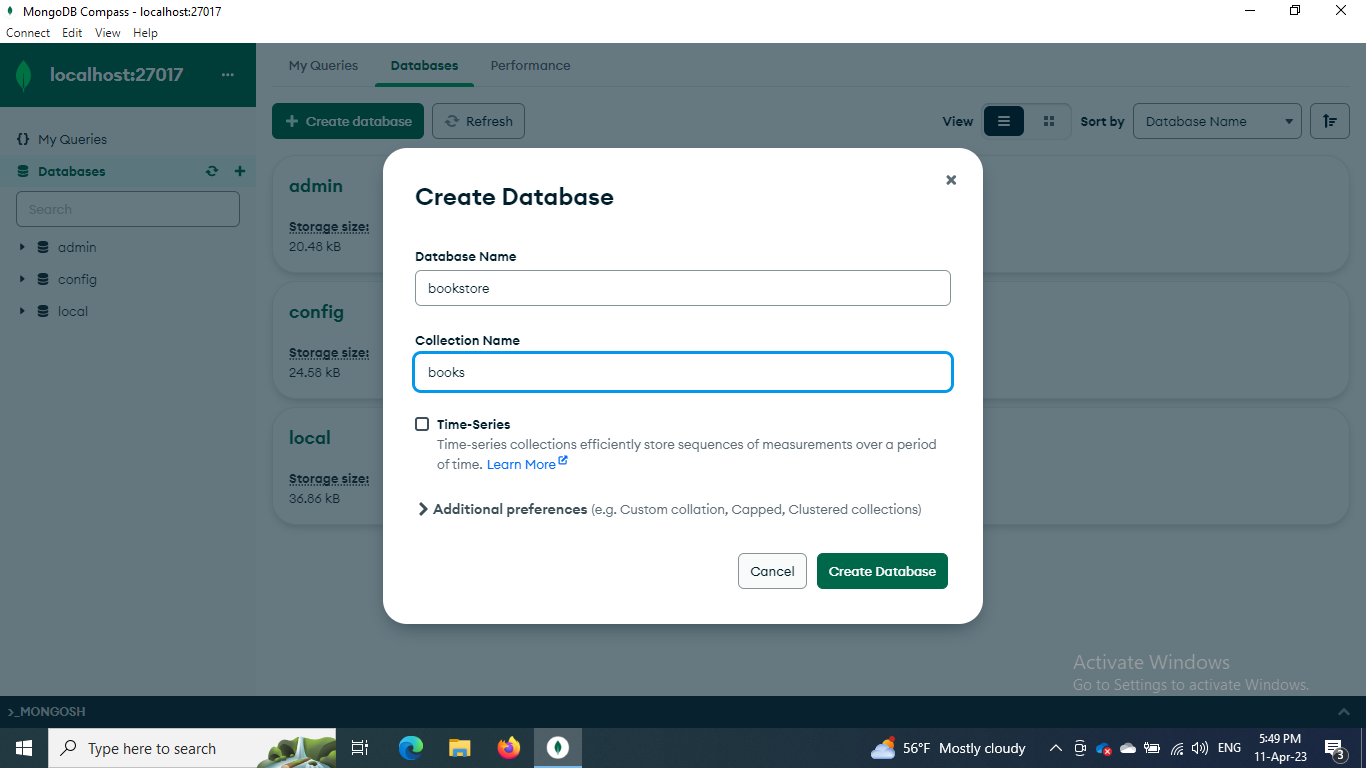
Za praktični deo projekta odlučila sam da pravim bazu podataka za prodavnicu knjiga. Kako bih prikazala način pravljenja i rada sa bazom, kolekcijama i dokumenatima kao i svim bitnim detaljima koristila sam dva načina za manipulaciju nad njima. Prvi jeste MongoDB Compass koji predstavlja besplatnu interaktivnu alatku za ispitivanje, optimizaciju i analizu MongoDB podataka. Compass pruža sve, od analize šeme do optimizacije indeksa do pipeline agregacije u jednom, centralizovanom interfejsu. Drugi jeste Mongo Shell. Mongo Shell je interaktivni JavaScript interfejs za MongoDB koji se koristi za upite i ažuriranje podataka, kao i za obavljanje administrativnih operacija.

Da bi kreiranje baze bilo moguće potrebno je pre svega kreirati konekciju, a potom kreirati bazu. MongoDB Compass omogućava izuzetno jednostavno kreiranje konekcije i baze, što je prikazano na sledećim slikama.



Slika 9. Kreiranje konekcije u MongoDB Compass-u 

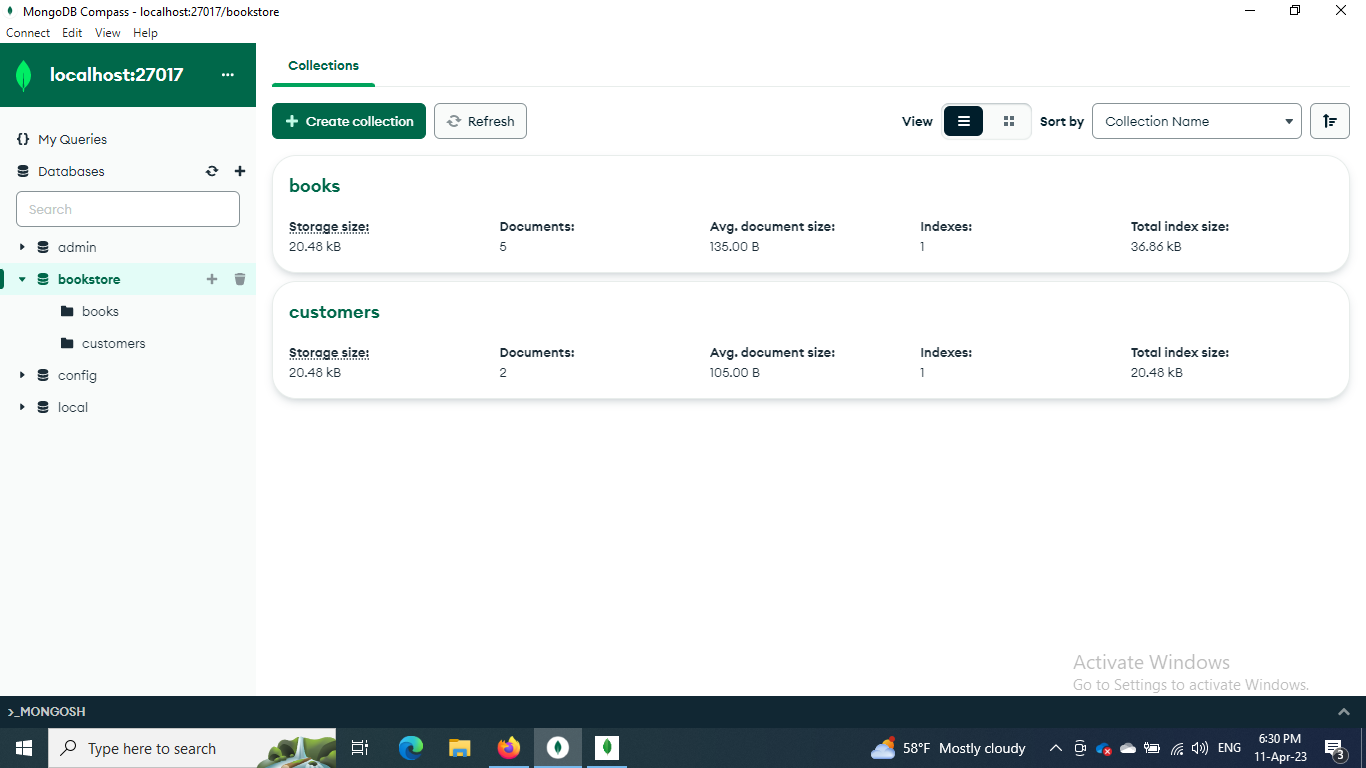
Slika 10. Postojeće baze podataka i dugme za kreiranje nove



Slika 11. Kreiranje nove baze podataka “bookstore” i kolekcije “books”

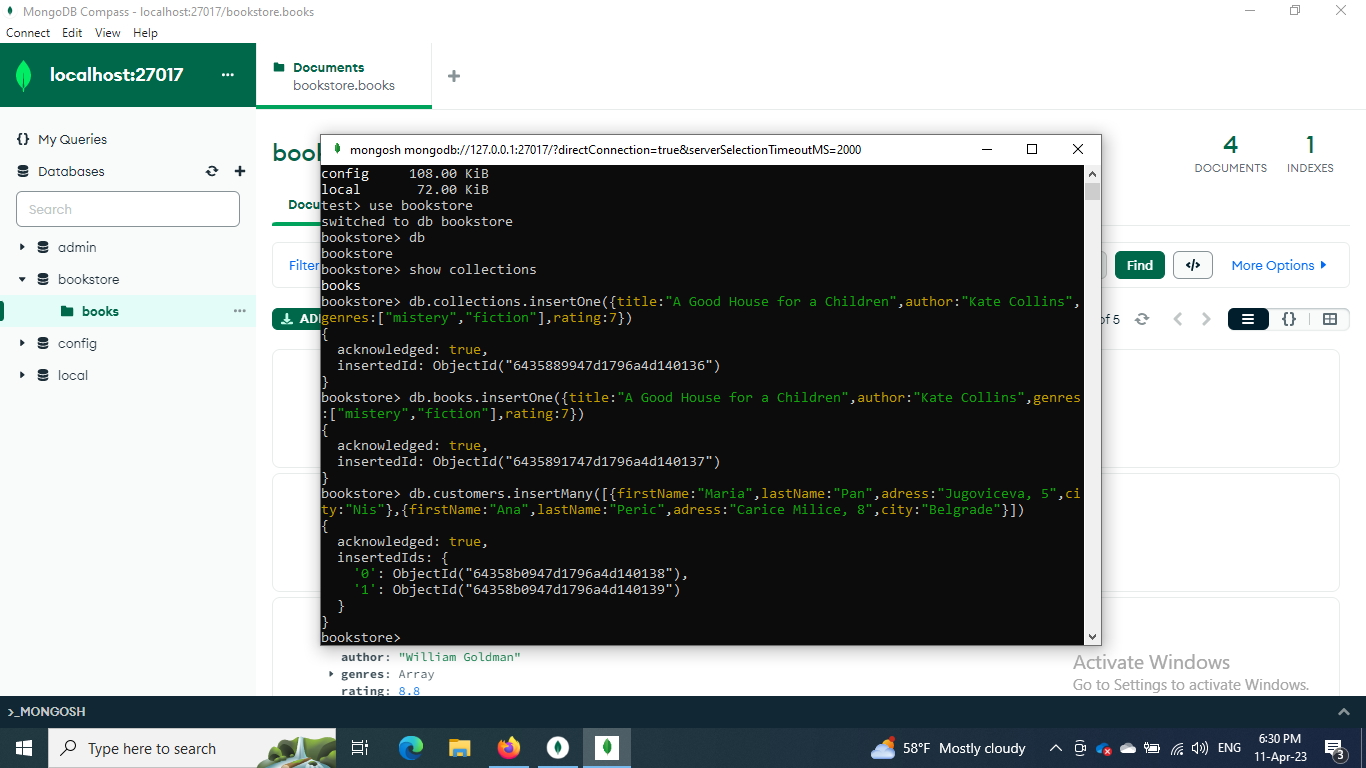
Nakon kreiranja baze podataka, sledeći korak jeste kreiranje kolekcija. Kako sam kreirala bazu za prodavnicu knjiga, potrebne kolekcije koje sam kreirala su : *books, customers, orders,ordersStatus*. Prva kolekcija kreira se istovremeno sa bazom upisom njenog imena u drugo polje. Na sledećim slikama prikazan je način kreiranja ostalih kolekcija.

Što se tiče MongoDB Compass-a kreiranje je izuzetno jednostavno. Klikom na sekciju *collection,* a zatim i na dugme *Create collection* možemo dodati proizvoljan broj kolekcija u bazi.

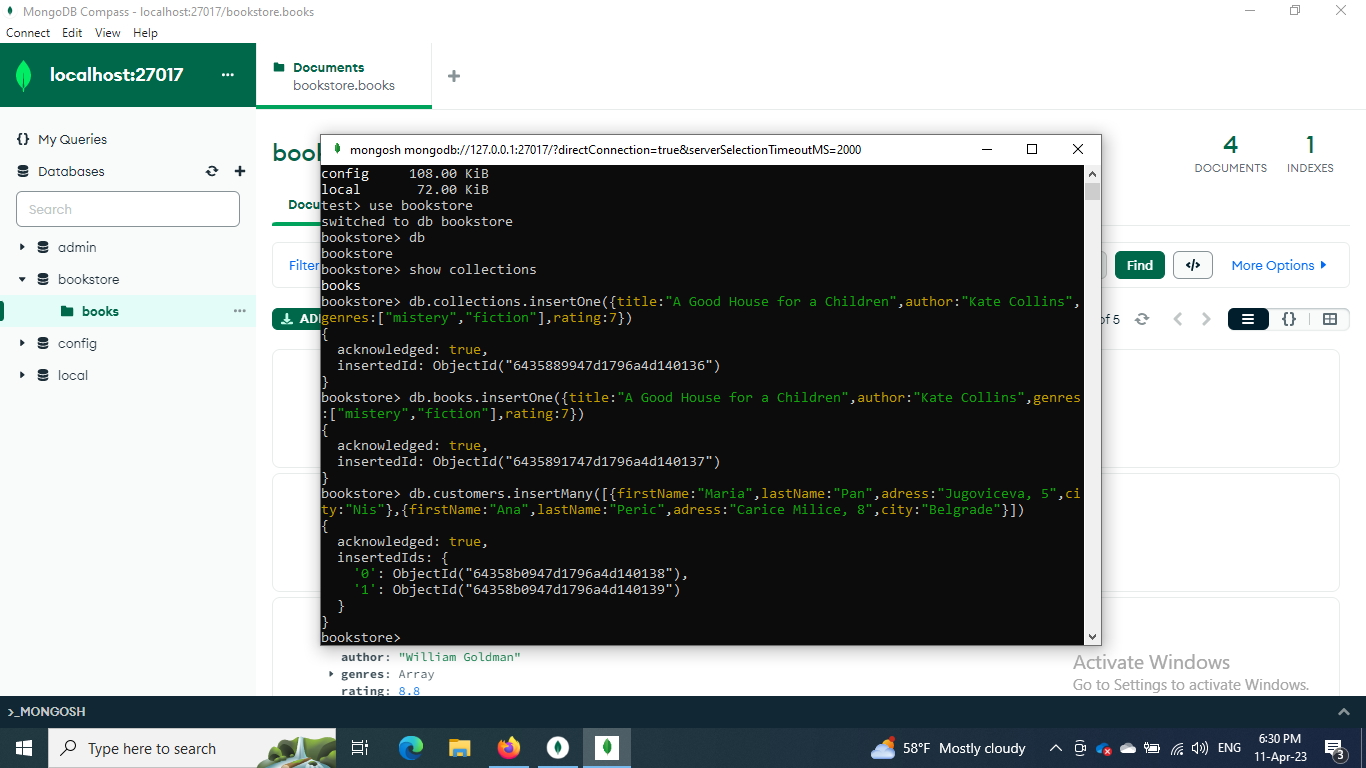


Slika 12. Kreiranje nove kolekcije u MongoDB Compass-u

MongoDB Shell omogućava dva načina dodavanja novih kolekcija. Jedan od načina jeste korišćenjem komande ***db.createCollection(“CollectionName”).*** Drugi metod predstavlja automatsko dodavanje dokumenta u kolekciju koja još uvek nije kreirana. Ova prečica na taj način omogućava i dodavanje nove kolekcije i novog dokumenta pomoću samo jedne komande. Ako istu komandu koristimo za dodavanje dokumenta u postojeću kolekciju, dodaće se samo dokument, odnosno neće se kreirati nova kolekcija.



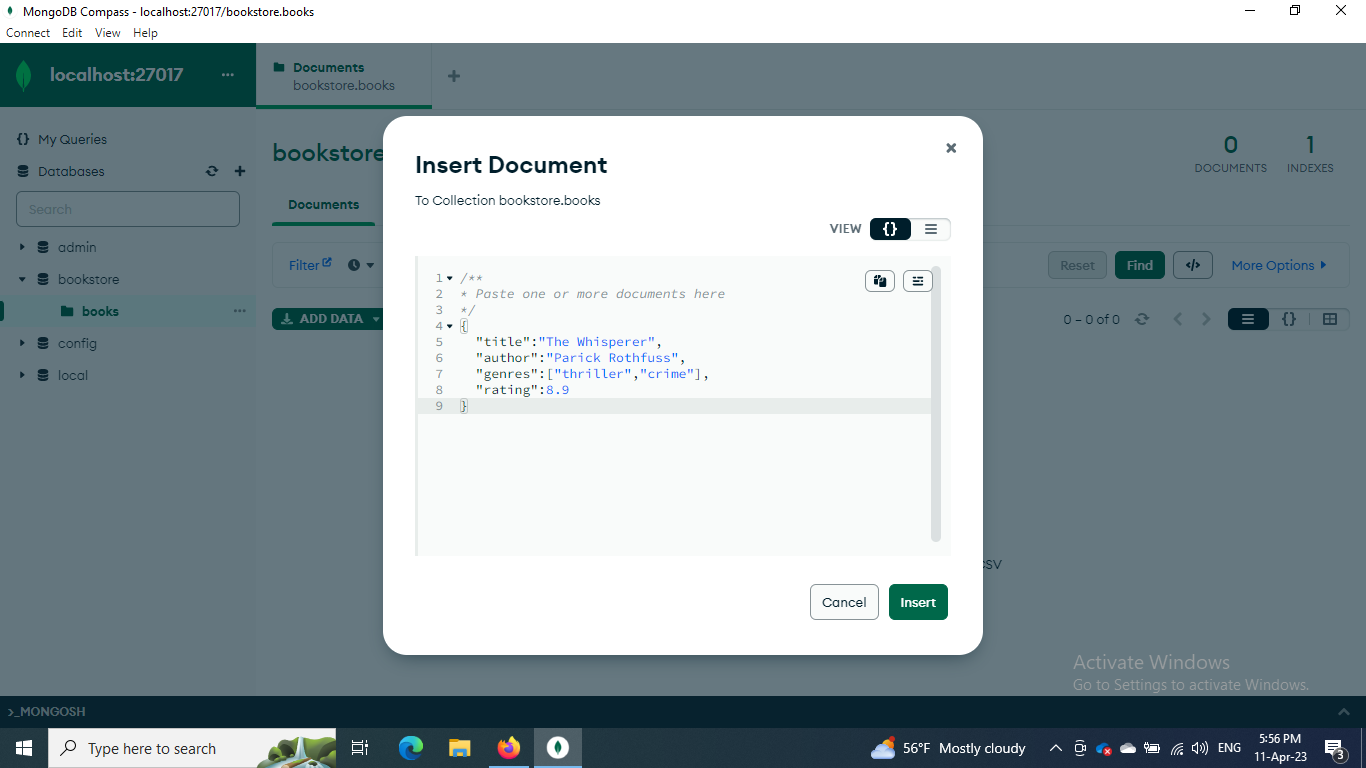
Slika 13. Mongo shell i komanda kojom se prikazuju sve kolekcije



Slika 14. Komanda kojom se kreiraju i novi dokumenti i nova kolekcija

Iako *customers* kolekcija još uvek nije kreirana i ne postoji u bazi, na ovaj način pomoću *insertMany* komande u isto vreme kreirana je i nova kolekcija, kao i novi dokumenti koji se automatski dodaju u tu kolekciju.

Kao što može da se vidi na prethodnoj slici dokumenti predstavljaju json fajlove. Medjutim kako MongoDB skladišti podatke kao bson, automatski se u pozadini vrši konverzija json fajla u bson fajl.

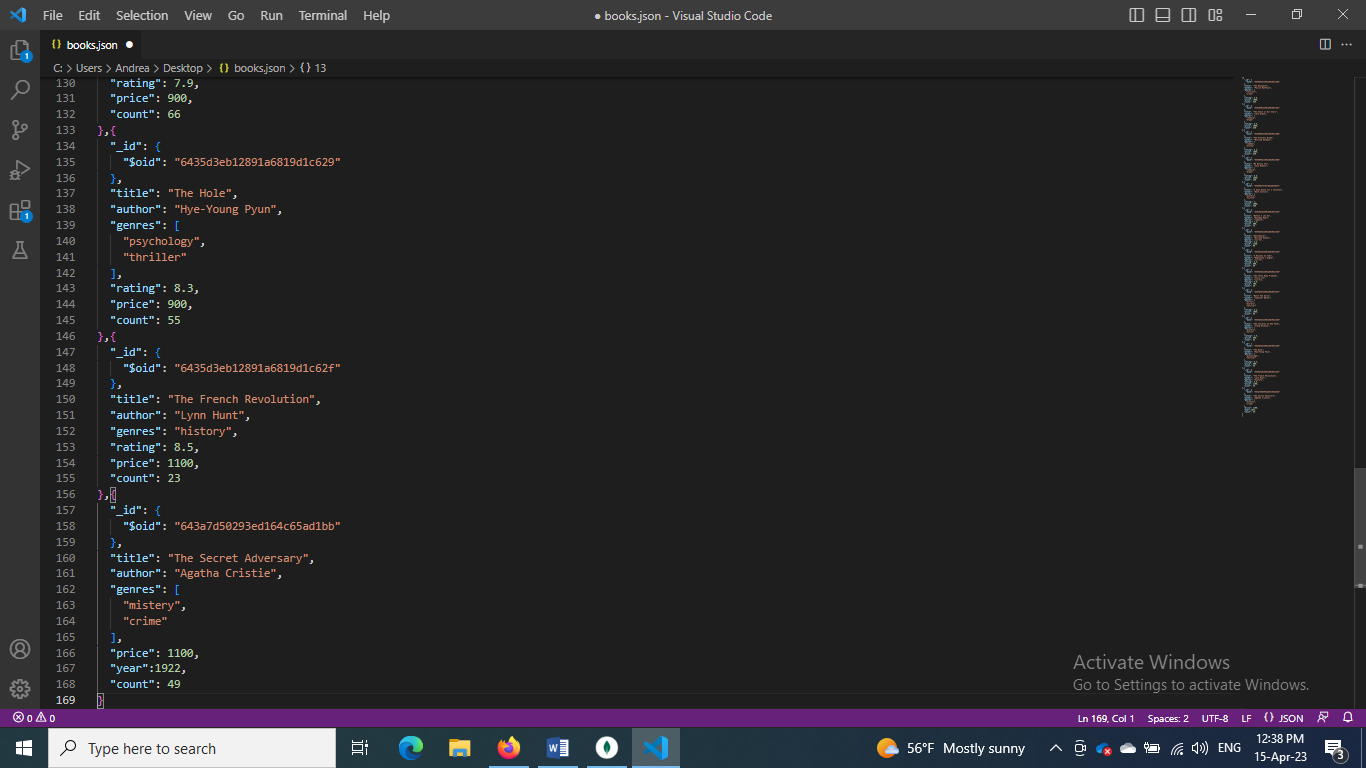


Slika 15. Prikaz unosa novog dokumenta

## Struktura dokumenata

Bitna karakteristika MongoDB-ja jeste ta da dokumenti u istoj kolekciji ne moraju imati istu strukturu. Dokumenti u jednoj kolekciji ne moraju da imaju isti skup polja, a tip podataka za polje može da se razlikuje među dokumentima unutar kolekcije. Medjutim, preporuka je da u slučaju aplikacije koja na globalnom nivou i na isti način pristupa svim dokumentima ipak kreiramo dokumente kolekcije koji će imati istu strukturu. Primera radi, ako kreiramo dokumente koji će imati različita polja, ili će vrednosti polja biti različitog tipa samo okruženje i server neće javiti grešku i dokumenti će se kreirati i dodati u kolekciju bez ikakvih problema. Medjutim, u slučaju da je u aplikaciji koja koristi bazu potrebno izvršiti pretragu na osnovu nekog konkretnog polja ili sortiranje na osnovu vrednosti odredjenog polja, u slučaju da svi dokumenti nemaju to konkretno bolje javiće se greška ili će se ignorisati odredjeni dokumenti koji su možda baš oni koji se traže.

Eksportovana šema kolekcije *books* prikazana je na sledećoj slici.



Slika 16. Eksportovana šema kolekcije books

Kao što se može videti na slici iznad, nisu svi dokumenti isti. Pre svega razlikuje se tip vrednosti polja „genres“. U nekim slučajevima vrednost ovog polja je samo string, dok u nekim slučajevima, kada knjiga pripada većem broju žanrova, vrednost polja je niz. Takodje, poslednji dokument na slici ima i dodatno polje koje govori o godini kada je knjiga izdata, dok nema polje koje predstavlja ocenu. U tom slučaju, ako se vrši sortiranje ili pretraga na osnovu ocene, ova knjiga se neće pojaviti kao rezultat. Iako MongoDB dozvoljava kreiranje ovakve šeme i to zaista jeste korisno kada je potrebno grupisati različite podatke, ali tada moramo voditi računa o upitima i uglavnom nećemo moći da dobijemo sve željene podatke jednim upitom.

Takodje na slici iznad se može videti da svaki dokument sadrži polje ***\_id*** koje je tipa *ObjectId*, dok na prethodnim slikama koje prikazuju metod kreiranja dokumenta to polje nije prikazano. Razlog tome jeste taj što MongoDB obezbedjuje automatsko kreiranje tog primarnog indeksa jer on uvek mora da postoji. Dakle, korisnik ga može i sam kreirati, a u slučaju da ne želi, MongoDB će sam odraditi taj posao. Bitno je napomenuti da identifikator mora biti jedinstven, pa kada se automatski kreira, taj zahtev će sigurno biti ispunjen a ako pak odlučimo da ga mi sami menjamo i prilagodjavamo potrebama mora se dosta voditi računa o tome.

## Embedded vs Referenced

Jedno od ključnih pitanja na koje treba odgovoriti prilikom dodavanja novih kolekcija jeste da li da uskladištite povezane dokumente kao ugrađene dokumente (embedded) ili da ih referencirate u zasebnoj kolekciji.

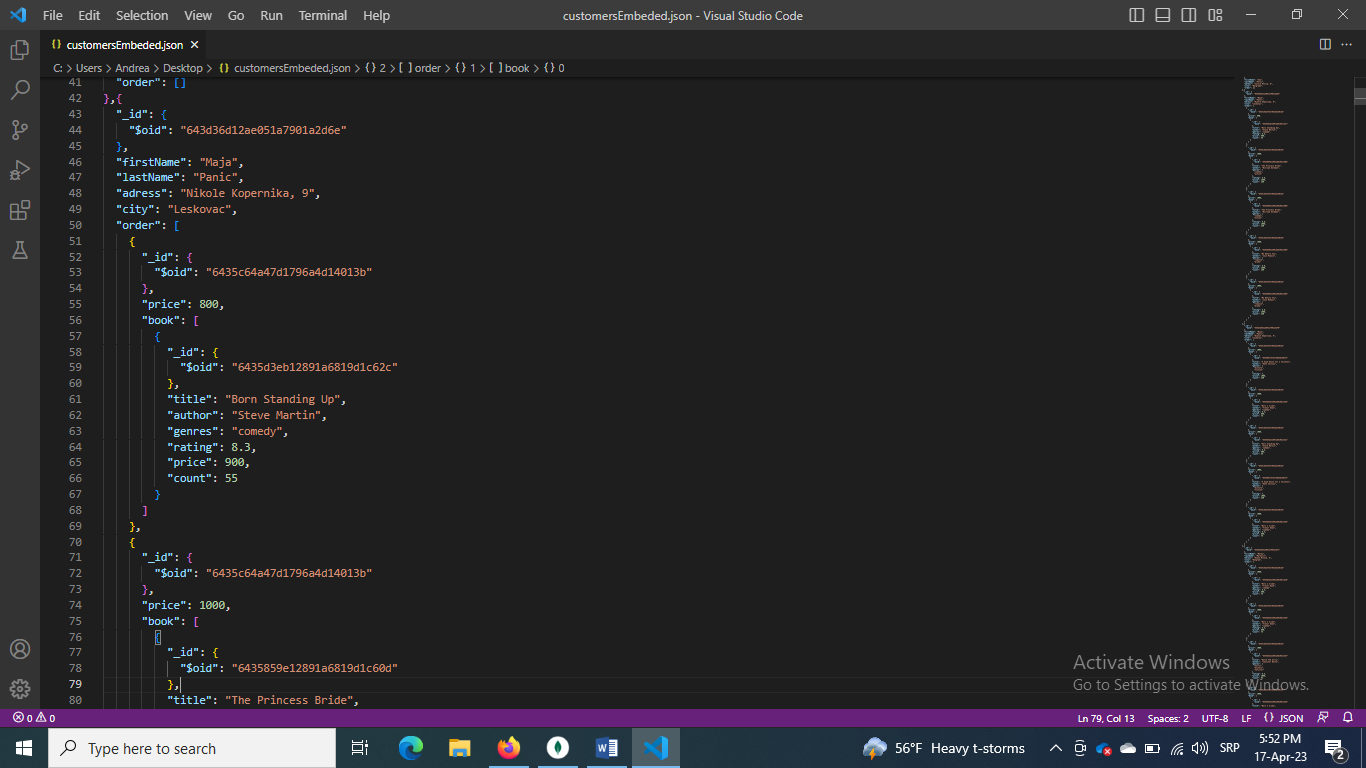
Donošenje pogrešne odluke može imati značajne implikacije na performanse — pa koju opciju treba izabrati?

Slično relacionoj bazi podataka, MongoDB omogućava korišćenje šeme za definisanje polja i tipova podataka unutar kolekcija. Iako su šeme opcione, toplo se preporučuju za lako razumevanje formata dokumenata. Šeme takođe osiguravaju da imate visok kvalitet podataka tako što osiguravaju da su obavezna polja prisutna i da su sva polja u skladu sa svojim tipovima. Jedna od moćnih opcija šeme dostupnih u MongoDB-u je povezivanje dokumenata jedan sa drugim, bilo putem ugrađivanja ili referenciranja.

Ove mehanizme iskoristila sam kako bih proverila prednosti i mane ovih načina povezivanja dokumenata. Kako bih izvršila testiranje kreirala sam dve kolekcije koje sadrže dokumente koji predstavljaju informacije o korisnicima. Dokumenti iz jedne kolekcije sadrže reference na dokumente koji predstavljaju porudžbine, dok dokumenti druge kolekcije sadrže ugradjene informacije o porudžbinama.

## Embedded dokumenti

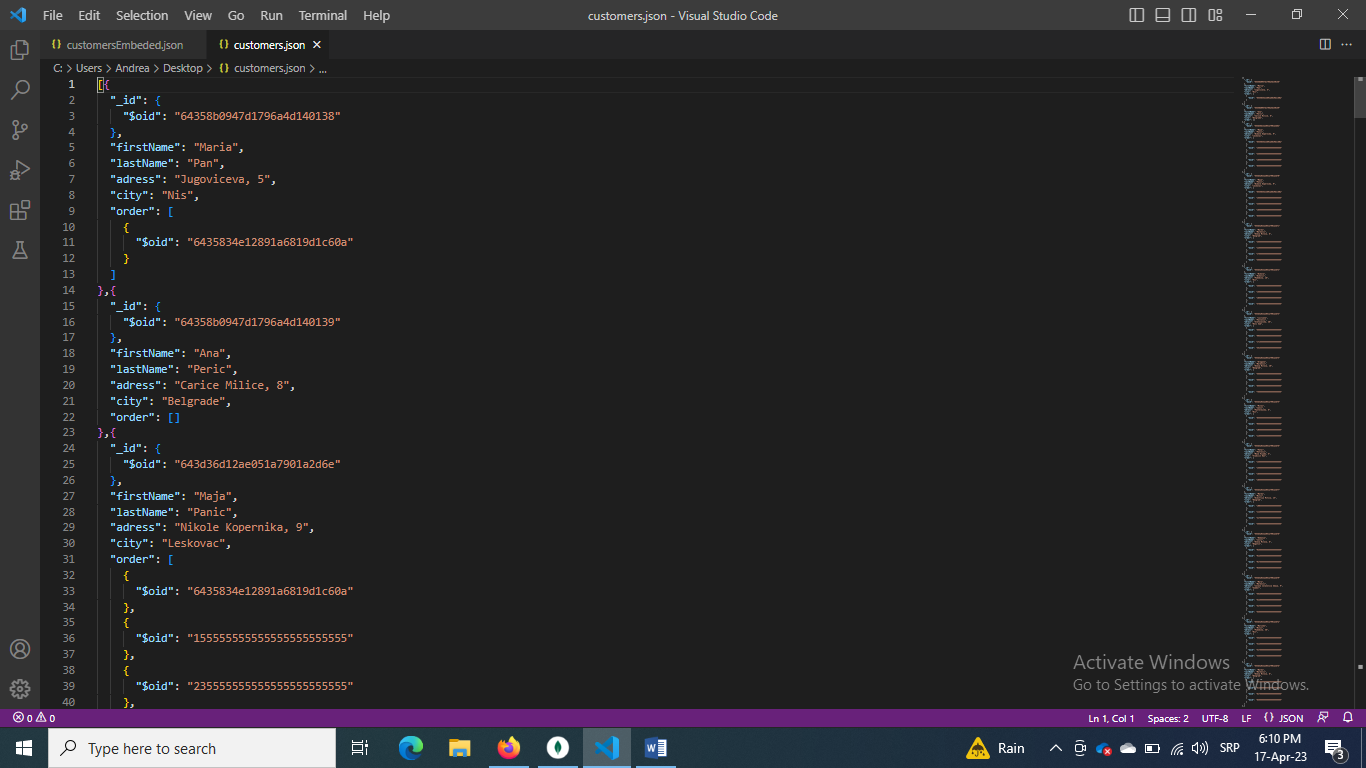
Ugrađeni dokumenti se čuvaju kao deca unutar roditeljskog dokumenta. To znači da su svi uskladišteni u jednoj kolekciji i kad god preuzmete nadređeni dokument, takođe preuzimate i sve njegove ugrađene dokumente. Dakle u ovom slučaju, ako se preuzme dokument o nekom korisniku, odmah će se preuzeti i biti vidljive i sve informacije o njegovim porudzbinama i izabranim knjigama. U JSON formatu, eksportovana šema ove kolekcije izgleda ovako:



Slika 17. Embedded document

## Referentni dokumenti

Za razliku od ugrađenih dokumenata, referentni dokumenti se čuvaju u posebnoj kolekciji u odnosu na njihov roditeljski dokument. Zbog toga je moguće preuzeti roditeljski dokument bez preuzimanja nijednog njegovog referentnog dokumenta. Kao referenca može se izabrati bilo koje polje drugog dokumenta, iako je najlakše izabrati primarni indeks. Razlog tome je taj što je sigurno da je on jedinstven u celoj kolekciji. Upravo ovo polje je izabrano kao referenca u primeru koji je već objašnjen. Eksportovana šema kolekcije koji sadrži dokumente sa informacijama o korisnicima koji sadrže reference na dokumente o porudžbinama prikazana je u nastavku.



Slika 18. Referencirani document

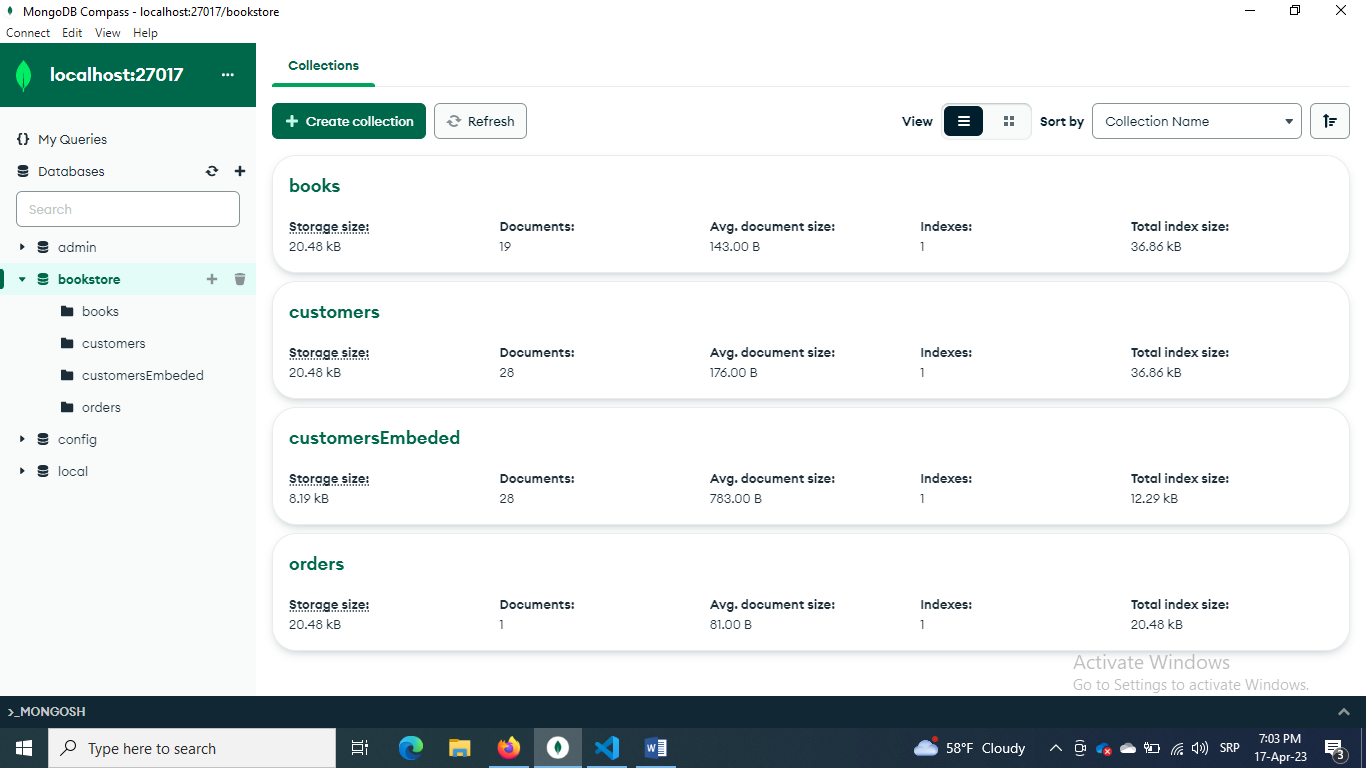
U čuvanju informacija o kupcima koristeći reference, čuvamo 3 odvojena dokumenta. Ako želimo da preuzmemo porudzbine korisnika zajedno sa korisnikom, moramo efektivno da pokrenemo 2 upita: prvi da preuzmemo korisnika, drugi da preuzmemo njegove porudžbine. Takodje, ako pak želimo da proverimo i koju knjigu je poručio korisnik biće nam potreban i treći upit. To je zato što MongoDB, za razliku od SQL-a, nema performansni koncept pridruživanja. Omogućava operaciju traženja, ali obično nije dovoljno efikasna da bi se koristila u proizvodnom okruženju u realnom vremenu i u pogledu brzine i resursa.

Umesto toga, trebalo bi da tražiti bolji dizajn šeme tako da odgovara MongoDB-u. Vredi napomenuti da je moguće izabrati način korišćenja reference. Možemo odlučiti, kao što smo već uradili, da sačuvamo listu referenci direktno u dokumentu. Ako je verovatno da će lista referenci postati velika ili da će sam dokument sadržati previse informacija, obično je bolje da sačuvati samo ID nadređenog dokumenta koji će sadržati liste referenci porudzbina ili same porudžbine, naravno sve u zavisnosti od količine podataka. Kako odabrati između ugrađenih ili referenciranih dokumenata

Uopšteno govoreći, trebalo bi da gledamo da koristimo ugrađene dokumente kada se i roditeljski dokument i povezani dokumenti čitaju ili pišu u isto vreme. Štaviše, trebalo bi da odredimo prioritete na osnovu toga da li je kolekcija teška za čitanje ili pisanje. Dakle koji ćemo način izabrati zavisi isključivo od slučaja upotrebe.

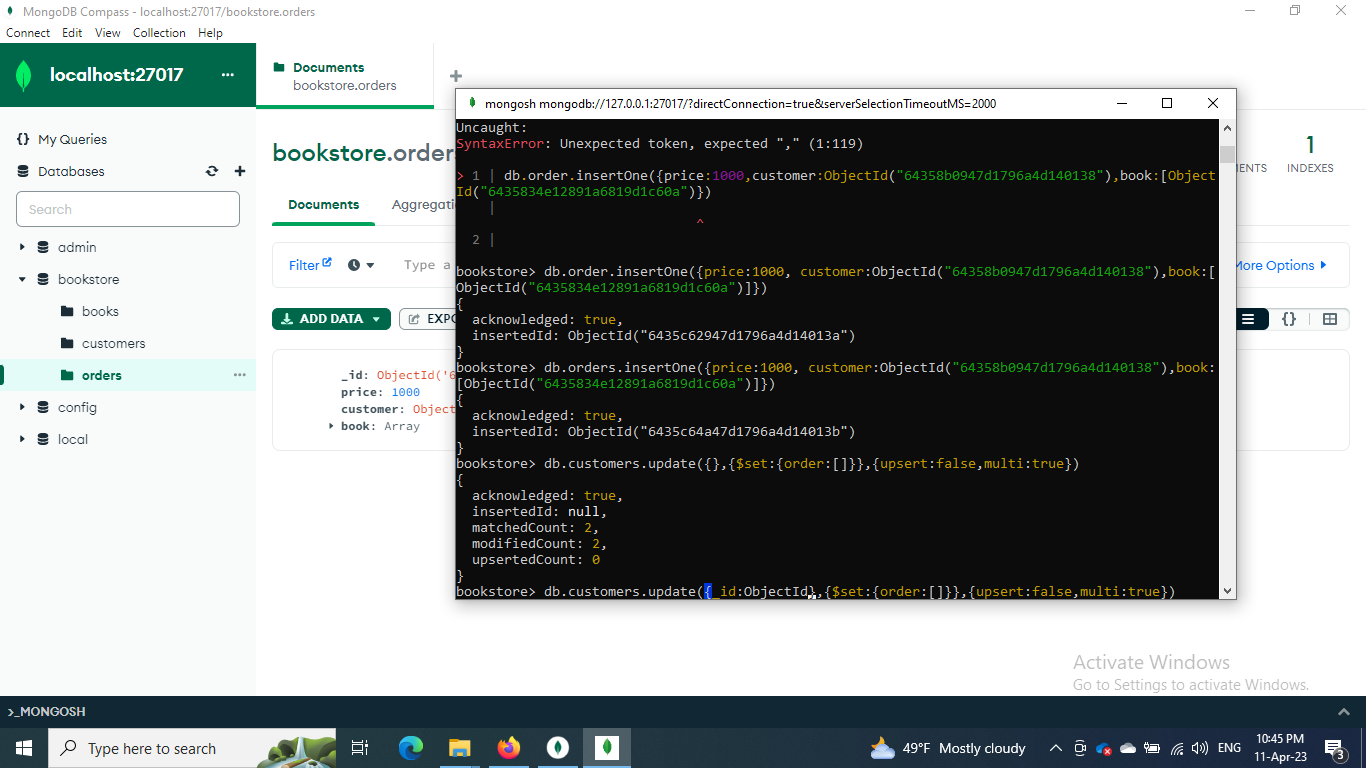
Imamo dva dokumenta: kupci, porudžbine.Treba li ih ugraditi? Sve ovo zavisi od toga kolika je količina podataka koji se čuvaju i da li se često pristupa dokumentima radi čitanja i upisa.

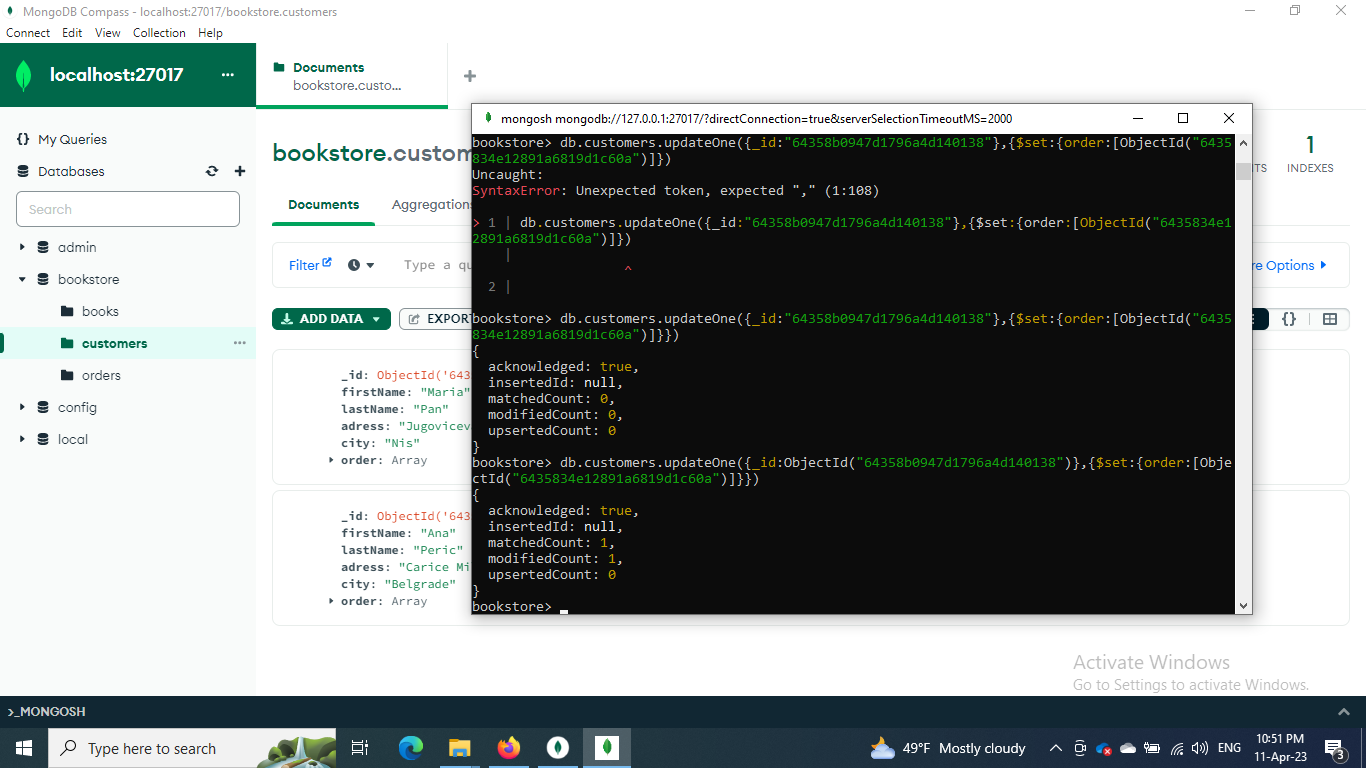
Ako je potrebno stalno upisivati mnogo podataka, bez potrebe za čitanjem ili menjanjem celog nadređenog dokumenta, bolje je koristiti reference. U slučaju skladištenja stotine dokumenata kao ugrađenih dokumenata, može doći do upisivanja koje će trajati više od jedne sekunde (uglavnom zbog potrebe da se svaki put serijalizuje/deserijalizuje ceo dokument). Naravno problem je i memorija. Već je rečeno da je maksimalna veličina dokumenata 16MB. Na ovaj način može se lako prekoračiti ova granica. Na sledećoj slici prikazane su dve pomenute kolekcije o kupcima koje sam kreirala. Podaci su u potpunosti isti sem toga što *customers* kolekcija sadrži dokumente sa referencama, a *customersEmbeded* ugnježdene. Kao što se može primetiti kolekcija sa ugnježdenim dokumentima zauzima čak 6 puta više memorije.



Slika 19. Embedded vs Referenced

U slučaju da je potrebno vršiti čitanje, menjanje i korišćenjenje većine podataka u dokumentu i povezanim dokumentima, bolje je sačuvati ih zajedno kao ugrađene dokumente. Čuvanje istih kao reference samo će nepotrebno povećati vreme odgovora zbog toga što se izvršava više upita nego što je potrebno. Na sledećim slikama prikazana su 2 upita koja je potrebno izvršiti u slučaju dodavanja nove porudžbine. Da su u pitanju ugnježdeni dokumenti potreban bi bio samo jedan upit.





Slika 20. Upiti potrebni za dodavanje novih

Dakle, koji način će više odgovarati implementaciji zavisi od količite podataka koje je potrebno skladištiti, kao i od dinamičnosti upita koji se koriste.

## Agregacija

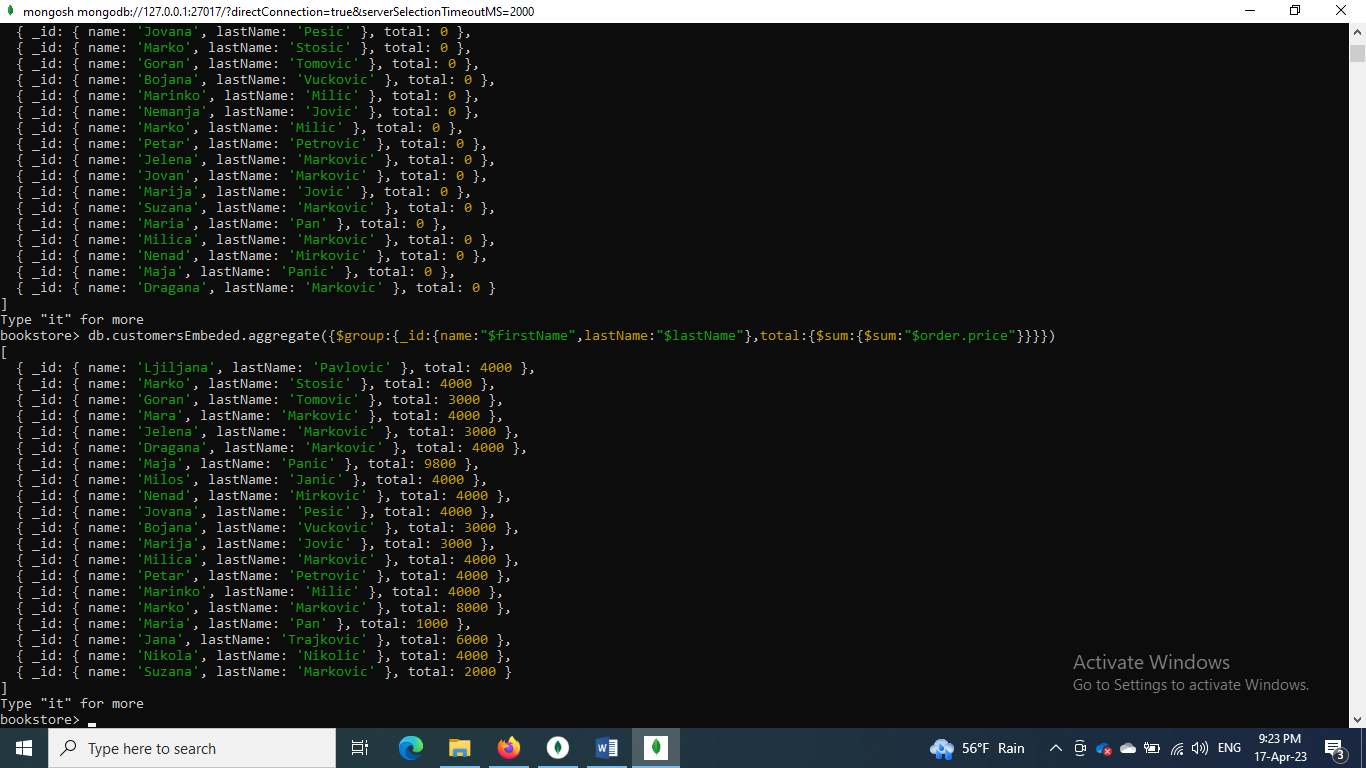
Agregacija je način obrade velikog broja dokumenata u kolekciji putem njihovog prolaska kroz različite faze. Faze čine ono što je poznato kao pipeline. Faze u pipeline-u mogu da filtriraju, sortiraju, grupišu, preoblikuju i modifikuju dokumente koji prolaze kroz pipeline.

Jedan od najčešćih slučajeva upotrebe agregacije je izračunavanje zbirnih vrednosti za grupe dokumenata. Ovo je slično osnovnom agregiranju dostupnom u SQL-u sa klauzulom GROUP BY i funkcijama COUNT, SUM i AVG. MongoDB agregacija ide dalje i takođe može da obavlja spajanja nalik na relacije, preoblikuje dokumente, kreira nove i ažurira postojeće kolekcije itd.

Iako postoje i druge metode za dobijanje zbirnih podataka u MongoDB-u, okvir agregacije je preporučeni pristup za većinu posla.

Postoje metode koje se nazivaju jednonamenskim metodama kao što su count() i different() koje se dodaju upitu find() što ih čini brzim za korišćenje, ali ograničenim po obimu.

U ovom konkretnom primeru agregacija se koristi da se izračuna ukupna suma novca koju je korisnik potrošio na kupovinu knjiga. Primer takvog upita i rezultata agregacije prikazan je na sledećoj slici.



Slika 21. Agregacija

# Zaključak

U ovom radu pokušala sam da na slikovit način prikažem načine korišćenja i rukovanja sa MongoDB kao i sve mehanizme skladištenja koje ovo rešenje nudi. Iako MongoDB možda nije savršeno rešenje za svaki slučaj upotrebe, on nudi mnoge prednosti koje ga čine jakim kandidatom za razvoj modernih aplikacija. Njegove prednosti u rukovanju nestrukturiranim podacima i horizontalnom skaliranju učinile su ga popularnim izborom među programerima koji žele da izgrade moderne aplikacije. Takodje čuvanje većine podataka u RAM-u što omogućava brže performanse prilikom izvršavanja upita predstavlja još jednu veliku prednost MongoDB-ja.

MongoDB radi 100 puta brže od drugih relacionih baza podataka i pruža visoke performanse, a pored toga nudi i jednostavnu sintaksu upita koju je mnogo lakše shvatiti nego SQL. Još jednu od prednosti predstavlja jednostavno okruženje. Instalacija, podešavanje i izvršavanje za MongoDB su brzi i jednostavni. Brži je i lakši za podešavanje od RDBMS-a i nudi moderne JavaScript okvire. Nikako ne smemo izostaviti i fleksibilnost ovog rešenja. MongoDB šema nije unapred definisana. To znači da ima dinamičku šematsku arhitekturu koja radi sa nestrukturiranim podacima i skladištem, a često je jako važno imati fleksibilan model baze podataka koji bi se mogao prilagoditi ovim promenama. Takodje, skalabilnost je jedna od najvažnijih prednosti MongoDB-a. Kao što se vidi, MongoDB koristi „sharding“, što proširuje kapacitet skladištenja. Za razliku od SQL baza podataka koje koriste vertikalnu skalabilnost, dijeljenje omogućava MongoDB-u da koristi horizontalnu skalabilnost.

Pored brojnih prednosti, naravno da postoje i mane ovog rešenja. MongoDB koristi ACID (atomičnost, konzistentnost, izolaciju i izdržljivost) transakcije sa više dokumenata. Većina aplikacija ne zahteva transakcije, iako postoji nekoliko kojima će možda biti potrebna za ažuriranje više dokumenata i kolekcija. Ovo je jedno od glavnih ograničenja za MongoDB jer može dovesti do oštećenja podataka. Drugi veliki nedostatak je spajanje dokumenata jer to može biti veoma naporan zadatak zbog toga što MongoDB ne podržava spojeve kao relacione baze podataka. Još jedan nedostatak jeste ograničena veličina dokumenata. Kao što je već rečeno MongoDB dozvoljava ograničenu veličinu dokumenta od samo 16 MB. Još jedno od glavnih ograničenja MongoDB-a je velika upotreba memorije. MongoDB zahteva veliku količinu skladišta zbog nedostatka funkcionalnosti spajanja koje dovode do dupliciranja podataka. Povećava se redundantnost podataka koja zauzima nepotreban prostor u memoriji.

Stručnjaci smatraju da brojne prednosti MongoDB-a prevazilaze njegove nedostatke i to je razlog zašto mnoga preduzeća traže njegova rešenja za poboljšanje svojih proizvoda, usluga i aplikacija. Popularnost ovog rešenja poslednjih godina konstantno raste i kako timovi stručnjaka konstantno rade na popravljanju navedenih nedostataka smatram da će se popularnost MongoDB-ja još dugo menjati samo u pozitivnom smeru.

# Literatura

[1] https://digitalvarys.com/introduction-to-mongodb/

[2] https://www.techtarget.com/searchdatamanagement/definition/MongoDB

[3] https://www.mongodb.com/nosql-explained

[4] https://www.mongodb.com/docs/manual/introduction/

[5] https://en.wikipedia.org/wiki/NoSQL

[6] https://en.wikipedia.org/wiki/Document-oriented\_database

[7] https://medium.com/@hnasr/mongodb-internal-architecture-9a32f1403d6f

[8] https://intellipaat.com/blog/what-is-mongodb/#no3

[9] https://www.geeksforgeeks.org/mongodb-database-collection-and-document/

[10] https://www.mongodb.com/docs/manual/core/wiredtiger/

[11] https://www.mongodb.com/docs/manual/core/gridfs/

[12] http://source.wiredtiger.com/2.0.1/checkpoints.html

[13] https://source.wiredtiger.com/2.9.1/tune\_compression.html

[14] https://betterprogramming.pub/embedded-vs-referenced-documents-in-mongodb-how-to-choose-correctly-for-increased-performance-d267769b8671

[15] https://medium.com/geekculture/mongodb-relationships-embedding-vs-linking-pro-and-cons-5f7583e655ab

[16] https://www.mongodb.com/basics/embedded-mongodb

[17] https://www.mongodb.com/docs/manual/aggregation/