Univerzitet u Nišu
Elektronski Fakultet
Katedra za računarstvo

Seminarski rad

MongoDB

Skladištenje i indeksiranje;
Procesiranje upita i optimizacija

Srđan Ognjanović | Sistemi za upravljanje bazama podataka

Profesori: Leonid Stoimenov, Aleksandar Stanimirović

Sadržaj

Opis

U seminarskom radu biće teorijski obrađene izabrane teme sa opisom praktične implementacije na primeru tehnologije MongoDB. Zbog uske povezanosti odabrane su teme *Skladištenje i indeksiranje* (*Storage and indexing*); *Procesiranje upita i optimizacija* (*Query processing and optimization*).

Teorijski deo

Indeksiranje podataka

Osnovni pojmovi

Definicije

Poznato je da se podaci čuvaju u obliku zapisa. Svaki zapis ima ključno polje koje mu pomaže da bude prepoznatljivo. Indeksiranje je tehnika strukture podataka koja efikasno preuzima zapise iz datoteka baze podataka na osnovu nekih atributa nad kojima je izvršeno indeksiranje. Indeksiranje u sistemima baza podataka je slično onom koje vidimo u knjigama. [1]

Indeksiranje se koristi za optimizaciju performansi baze podataka tako što se minimizira broj pristupa disku potrebnih kada se upit obradi. Indeks je vrsta strukture podataka i koristi se za brzo pronalaženje i pristup podacima u tabeli baze podataka. [2]

Struktura indeksa

Indeksi se mogu kreirati pomoću nekih kolona baze podataka.

Search key	Data Reference

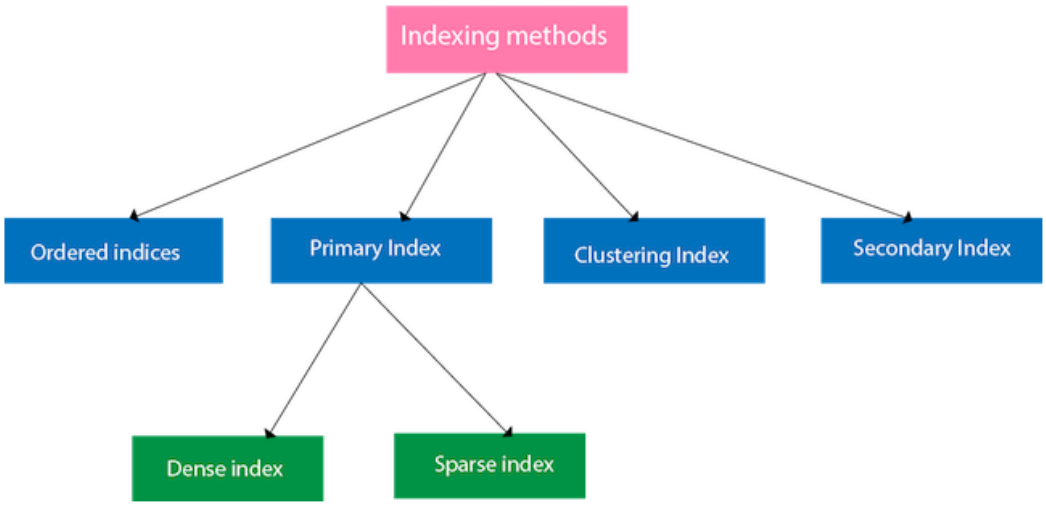
Slika 1 - Struktura indeksa

Prva kolona baze podataka je ključ za pretraživanje, koji sadrži kopiju primarnog ključa ili ključa kandidata tabele. Vrednosti primarnog ključa čuvaju se u sortiranom redosledu tako da se odgovarajućim podacima može lako pristupiti. Druga kolona baze podataka je referenca podataka. Sadrži skup pokazivača koji drže adresu bloka diska gde se može pronaći vrednost određenog ključa. [2]

Tipovi i metode indeksiranja

Indeksiranje je definisano na osnovu njegovih atributa indeksiranja i može biti sledećih tipova:

- **Uređeni indeks** (***Ordered Indices***) - indeksi se obično sortiraju kako bi pretraživanje bilo brže, a indeksi koji se sortiraju su poznati kao uređeni indeksi
- **Primarni indeks** (***Primary Index***) - primarni indeks definisan je na uređenoj datoteci podataka. Datoteka sa podacima je uređena na ključnom polju. Ključno polje je generalno primarni ključ odnosa
- **Sekundarni indeks** (***Secondary Index***) - sekundarni indeks može biti generisan iz polja koje je ključ kandidata i ima jedinstvenu vrednost u svakom zapisu, ili ne-ključ sa dupliciranim vrednostima
- **Grupisani indeks** (***Clustering Index***) - grupisani indeks (klaster) definisan je na uređenoj datoteci podataka. Datoteka sa podacima je uređena na ne-ključnom polju



Slika 2 - Metode indeksiranja

Uređeni indeks (Ordered Indices)

Primer: Data je tabela zaposlenih sa hiljadama zapisa, od kojih je svaki dug 10 bajta. Ako njihovi ID-ovi počinju sa 1, 2, 3, i tako dalje, a potrebno je pretražiti studenta sa ID-jem 543. [2]

- U slučaju baze podataka bez indeksa, mora se pretražiti disk blok od početka do rednog broja 543. DBMS će čitati zapis i nakon očitavanja iznosiće 543 * 10 = 5430 bajta
- U slučaju indeksa, pretraživaće se pomoću indeksa, a DBMS će pročitati zapis nakon očitavanja 542 * 2 = 1084 bajta, što je veoma manje u odnosu na prethodni slučaj pri čemu se vreme čitanja znatno smanjuje

Primarno indeksiranje (Primary Index)

Ako je indeks kreiran na osnovu primarnog ključa tabele, onda je poznat kao primarno indeksiranje. Ovi primarni ključevi su jedinstveni za svaki zapis i sadrže

odnos 1:1 između zapisa. Kako su primarni ključevi smešteni u sortiranom redosledu, performanse operacije pretraživanja su jako efikasne. [1]

Primarni indeks sačinjavaju dva osnovna tipa:

- **Gusto indeksiranje** (***Dense Index***)
- **Retko indeksiranje** (***Sparse Index***)

Višestruko indeksiranje odnosno ***multilevel index*** je specijalni tip indeksiranja gde se indeksiranje vrši u više nivoa, što se može zaključiti iz samog naziva.

Gusto indeksiranje (Dense Index)

Kod gustog indeksiranja, postoji indeksni zapis za svaku vrednost ključa pretrage u bazi podataka. To čini pretraživanje bržim, ali zahteva više prostora za pohranjivanje samih indeksnih zapisa. Zapisi indeksa sadrže vrednost ključa za pretragu i pokazivač na stvarni zapis na disku.

Slika 3 - Gusto indeksiranje

Retko indeksiranje (Sparse Index)

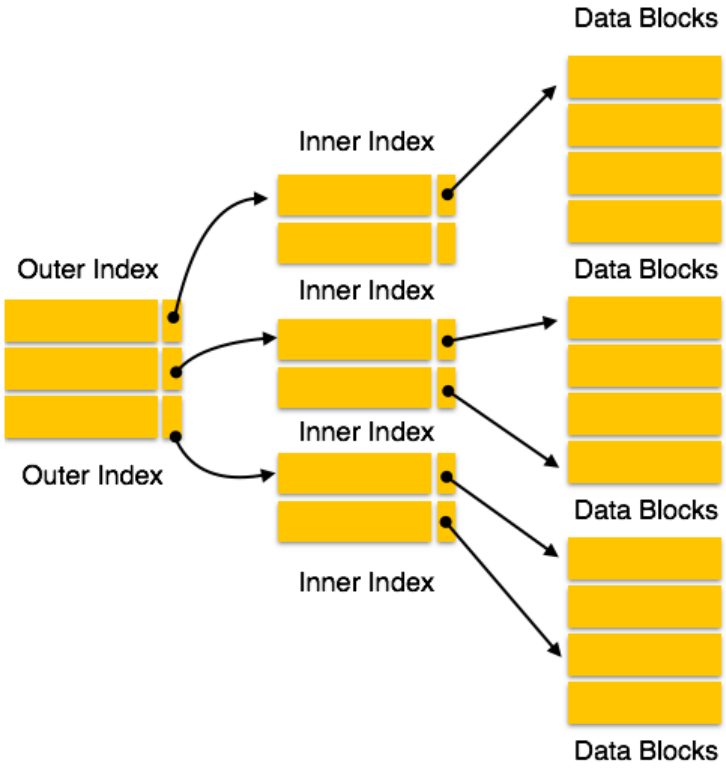
U slučaju retkog indeksiranja, zapisi indeksa nisu kreirani za svaki ključ za pretragu. Zapis indeksa sadrži ključ za pretragu i stvarni pokazivač na podatke smeštene na disku. Da bi se pretražila stavka (*record*), prvo se pristupa indeksnom zapisu i dolazi do stvarne lokacije podatka. Ako podaci koji se traže nisu tamo gde se direktno dolazi prateći indeks, onda sistem započinje sekvencijalno pretraživanje dok se ne pronađu željeni podaci.

Slika 4 - Retko indeksiranje

Indeksiranje u više nivoa (Multilevel Index)

Kod *multilevel* indeksiranja zapisi indeksa sadrže vrednosti ključa za pretraživanje i pokazivače podataka. Višestruki indeks se skladišti na disku zajedno sa stvarnim datotekama baze podataka. Kako veličina baze raste, tako i veličina indeksa.

Postoji ogromna potreba da se indeksni zapisi čuvaju u glavnoj memoriji kako bi se ubrzale operacije pretraživanja. Ako se koristi indeks na jednom nivou, onda se indeks velike veličine ne može čuvati u memoriji što dovodi do više pristupa disku.

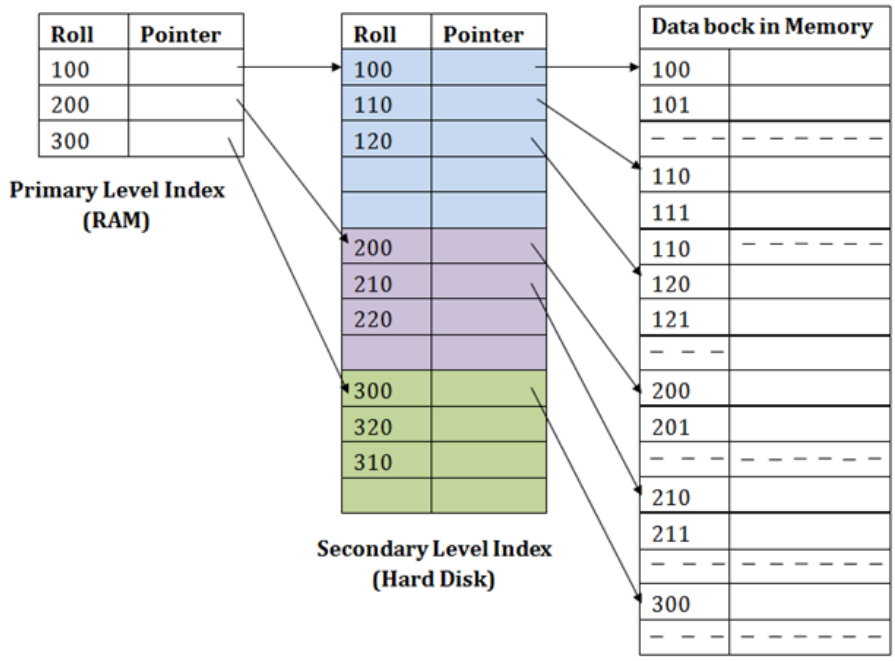


Slika 5 – Indeksiranje u više nivoa

Indeksiranje u više nivoa pomaže u razbijanju indeksa u nekoliko manjih indeksa kako bi najudaljeniji nivo bio tako mali da se može sačuvati u jednom bloku diska, koji se lako može smestiti bilo gde u glavnoj memoriji.

Sekundarno indeksiranje (Secondary Index)

Kod retkog indeksiranja, kako veličina tabele raste, veličina mapiranja takođe raste. Ova mapiranja se obično čuvaju u primarnoj memoriji tako da bi dohvat (*fetch*) adrese trebao biti brži. Zatim sekundarna memorija pretražuje stvarne podatke na osnovu adrese dobijene od mapiranja. Ako veličina mapiranja raste, onda pribavljanje (*fetching*) same adrese postaje sporije. U ovom slučaju, retki indeks neće biti efikasan. Da bi se ovaj problem prevazišao, uvedeno je sekundarno indeksiranje.



Slika 6 - Sekundarno indeksiranje

Kod ovog tipa indeksiranja, da bi se smanjila veličina mapiranja, uvodi se drugi nivo indeksiranja. U ovoj metodi, veliki raspon za kolone se bira u početku tako da veličina mapiranja prvog nivoa postaje mala. Zatim se svaki opseg dalje deli na manje opsege. Mapiranje prvog nivoa se čuva u primarnoj memoriji, tako da je dohvat adrese brži. Mapiranje drugog nivoa i stvarnih podataka se čuva u sekundarnoj memoriji (na hard disku). [2]

Primer:

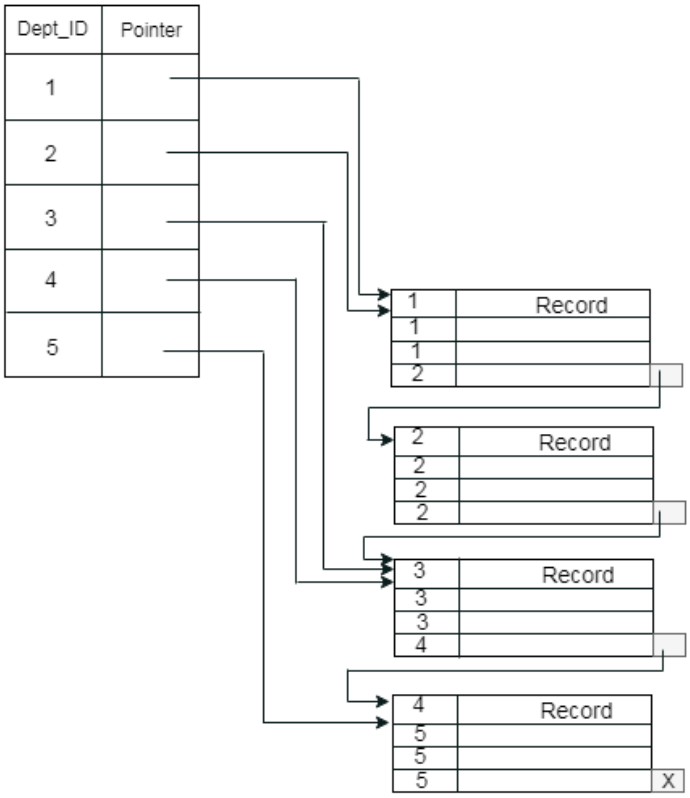
- Ako je potrebno pronaći zapis (roll) 111 u dijagramu, on će pretražiti najviši unos koji je manji ili jednak 111 u indeksu prvog nivoa. Dobiće se 100 na ovom nivou
- Zatim, u drugom nivou indeksa, opet se vrši pretraga kao *max* (111) <= 111 i dobija 110. Sada koristeći adresu 110, prelazi u blok podataka i počinje da traži svaki zapis dok ne dobije 111
- Ovim načinom se vrši pretraga. Unošenje, ažuriranje ili brisanje se takođe vrši na isti način

Grupisano indeksiranje (Clustering Index)

Grupisani indeks se može definisati kao uređena datoteka podataka. Ponekad se indeks kreira na kolonama koje nisu primarni ključ i koje možda nisu jedinstvene za svaki zapis. U ovom slučaju, kako bi se brže identifikovali zapisi, grupisaće se dve ili više kolona kako bi se dobila jedinstvena vrednost i kreirao indeks od njih. Ovaj metod se zove indeks klastera. Zapisi koji imaju slične karakteristike su grupisani, a indeksi su kreirani za ove grupe. [2]

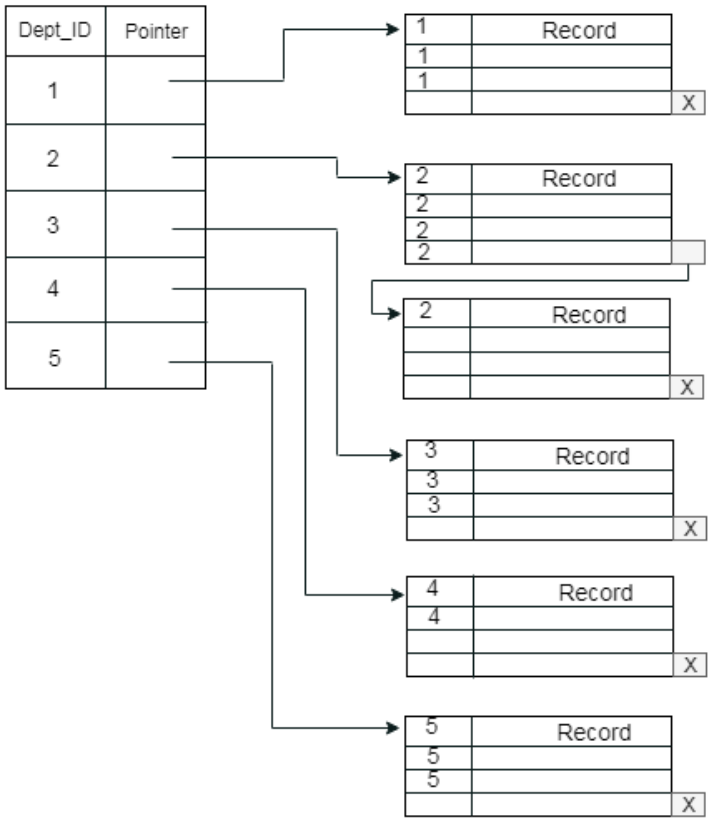
Primer: Pretpostavimo da kompanija ima nekoliko zaposlenih u svakom odeljenju. Pretpostavimo da koristimo indeks klastera, gde se svi zaposleni koji

pripadaju istom Dept_ID-u razmatraju u okviru jednog klastera, a indeksi ukazuju na klaster kao celinu. Ovde Dept_Id je ne-jedinstveni ključ.



Slika 7 - Grupisano indeksiranje

Prethodna šema je malo zbunjujuća jer je jedan disk blok deljen zapisima koji pripadaju različitim klasterima. Ako koristimo poseban disk blok za odvojene klastere, onda se to naziva boljom tehnikom.



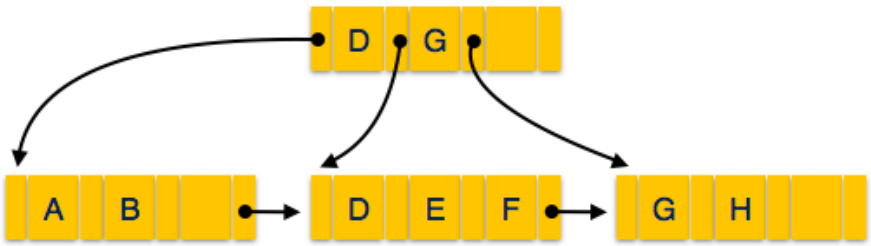
Slika 8 - Grupno indeksiranje

B+ stabla

B+ stablo je uravnoteženo binarno stablo pretrage koje prati format indeksa na više nivoa. Čvor listova B+ stabla označava stvarne pokazivače podataka. B+ stablo osigurava da svi čvorovi listova ostanu na istoj visini, tako da su uravnoteženi. Pored toga, čvorovi lista su povezani pomoću liste veza; stoga, B+ stablo može podržavati nasumični pristup kao i sekvencijalni pristup. [1, 2]

Struktura B+ stabla

Svaki čvor listova je na istoj udaljenosti od korenskog čvora. B+ stablo je reda *n*, gde je *n* fiksno za svako B+ stablo.



Slika 9 - Struktura B+ stabla

Interni čvorovi

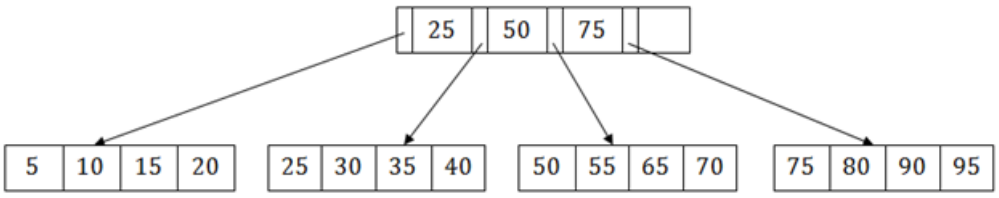
- Unutrašnji (ne-listni) čvorovi sadrže bar **⌈n / 2⌉** pokazivače, osim korenskog čvora
- Najviše, unutrašnji čvor može da sadrži **n** pokazivača

Čvorovi listova

- Čvorovi listova sadrže bar **⌈n / 2⌉** pokazivače za snimanje i **⌈n / 2⌉** ključne vrednosti
- Najviše, čvor lista može da sadrži **n** pokazivačkih zapisa i **n** ključnih vrednosti
- Svaki čvor lista sadrži jedan pokazivač bloka **P** da bi ukazao na sledeći čvor lista i formirao povezanu listu

Pretraga elementa

Primer: Pretpostavimo da je potrebno pretražiti element 55 u strukturi ispod. Prvo, najpre će se pribaviti posredni čvor koji će se usmeriti na čvor lista koji može sadržati zapis za 55. Dakle, u posrednom čvoru, naći će se grana između čvorova sa vrednostima 50 i 75. Onda će na kraju biti preusmereni na treći čvor lista. Ovde će *DBMS* izvršiti sekvencijalno pretraživanje da bi pronašao 55.



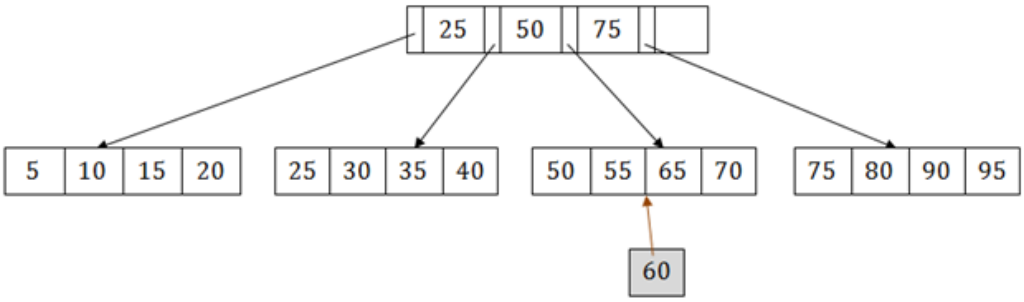
Slika 10 - Pretraga elemenata

Dodavanje elemenata

- B+ stabla se popunjavaju od dna i svaki unos se vrši na čvoru lista.
- Ako se čvor listova preliva (*overflows*):
  - Razdvaja se čvor na dva dela
  - Particija na **i = ⌊ (m + 1) / 2⌋**
  - Prvih **i** unosa se skladište u jednom čvoru
  - Ostali unosi (**i + 1** nadalje) se premeštaju u novi čvor
  - **i**-ti ključ je dupliran kod roditelja lista
- Ako se node od ne-lista ne preliva (*overflows*):
  - Razdvojiti čvor na dva dela
  - Razdelite čvor na **i = ⌈ (m + 1) / 2⌉**
  - Stavke do **i**-tog se čuvaju u jednom čvoru
  - Ostali unosi se premeštaju u novi čvor

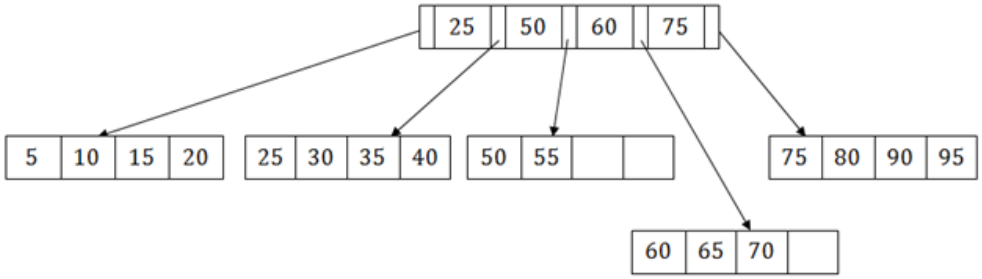
Primer: Pretpostavimo da je potrebno ubaciti zapis 60 u strukturu ispod. On će preći na treći čvor posle 55. To je balansirano stablo, a čvor lista ovog stabla je već pun,

tako da ne možemo da ubacimo 60 tamo. U ovom slučaju, potrebno je razdvojiti čvor lista, tako da se može umetnuti u stablo bez uticaja na faktor popunjavanja, balans i red.



Slika 11 - Dodavanje elemenata

Čvor trećeg lista ima vrednosti (50, 55, 60, 65, 70) i njegov trenutni korenski čvor je 50. Podelimo čvor lista stabla u sredini tako da se njegov balans ne menja. Tako možemo grupisati (50, 55) i (60, 65, 70) u 2 lista čvorova. Ako ova dva moraju biti listni čvorovi, srednji čvor ne može da se odvaja od 50. Trebalo bi da mu se doda 60, a onda možemo imati pokazivače na novi čvor lista.



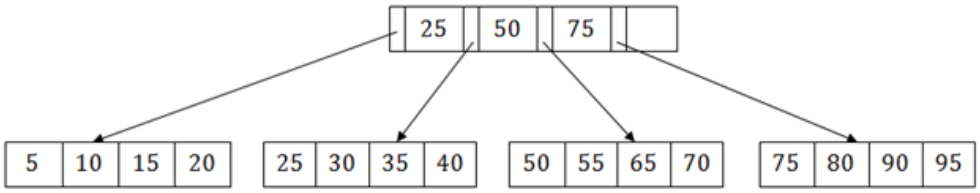
Slika 12 - Stanje nakon dodatog elementa

Uklanjanje elemenata

- B+ stabla se brišu u čvorovima lista
- Ciljni unos se pretražuje i briše
  - Ako je to unutrašnji čvor, izbriše i zamenite unos iz leve pozicije
- Nakon brisanja, testira se *underflow*
  - Ako dođe do prelivanja, distribuiraju se unose iz čvorova koji su mu ostavljeni
- Ako distribucija nije moguća sa leve strane, onda se distribuiraju iz čvorova pravo na njega
- Ako distribucija nije moguća sa leve ili desne strane, onda se spajaju čvorovi sa leve i desne strane

Primer: Pretpostavimo da je potrebno obrisati čvor sa vrednošću 60 iz gore navedenog primera. U ovom slučaju, potrebno je ukloniti 60 iz srednjeg čvora, kao i iz čvora 4. lista. Ako se ukloni iz srednjeg čvora, stablo neće zadovoljiti pravilo B+ stabla.

Zato se mora modifikovati da bi ostalo uravnoteženo stablo. Nakon brisanja čvora 60 iznad stabla B+ i ponovnog raspoređivanja čvorova, on će se prikazati na sledeći način:



Slika 13 - Uklanjanje elementa

MongoDB

Tehnologija

MongoDB je baza podataka za opštu upotrebu. Njena dinamička šema i objektno-orijentisana struktura, čine je pravim izborom za analitiku u realnom vremenu, kao i za *e-mailing*, mobilne aplikacije, arhiviranje i slično. Poznati slučajevi korišćenja MongoDB-a obuhvataju "*big data*" podatke, upravljanje sadržajem, mobilnu i društvenu infrastrukturu kao i korišćenje MongoDB-a za *Business Intelligence* modele.

Slika 14 - MongoDB logo

Neke od osnovnih karakteristika MongoDB-a su:

- Smeštanje usmereno na dokumente — *JSON* dokumenta sa dinamičkim šemama nude jednostavnost i snagu
- Apsolutna podrška indeksiranju — indeksiranje bilo kog atributa
- Replikacija i visoka dostupnost
- Automatsko skaliranje — horizontalno skaliranje bez ugrožavanja funkcionalnosti. Horizontalno skaliranje distribuira jedan logički sistem baze podataka između skupa mašina
- Upiti — moćni upiti, bazirani na dokumentima
- Brzi *update*
- Mapiranje/redukovanje — fleksibilna agregacija i obrada podataka
- *GridFS* — čuva datoteke bilo koje veličine bez komplikacija. Umesto čuvanja datoteke u pojedinačnom dokumentu, *GridFS* deli datoteku na delove, ili blokove i čuva svaki od tih blokova kao poseban dokument
- *Online shell* pruža mogućnost korišćenja odnosno isprobavanja MongoDB-a bez instalacije
- *Ad hoc* upiti — MongoDB podržava pretragu po polju, upite po opsegu i pretrage po regularnim izrazima. Upiti mogu da vrate određena polja dokumenta, kao i da obuhvate korisnički definisane *JavaScript* funkcije

- Izvršavanje *JavaScript* koda na strani servera — *JavaScript* se može koristiti u upitima, agregatnim funkcijama (kao što je *MapReduce*) i da se prosledi direktno bazi podataka na izvršavanje

Indeksiranje

Uvod u indeksiranje

Tipično, indeksi su strukture podataka koje mogu da skladište skupove podataka u obliku koji je lak za obilaženje. Upiti se efikasno izvršavaju uz pomoć indeksa u MongoDB-u. Indeksi pomažu MongoDB-u da pronađe dokumente koji odgovaraju kriterijumima upita bez potrebe skeniranja kolekcije. Ako upit ima odgovarajući indeks, MongoDB koristi indeks i ograničava broj dokumenata koje ispituje. Indeksi čuvaju vrednosti polja u redosledu vrednosti. Redosled kojim se unose indeksi prave operacije podrške, kao što su podudaranja jednakosti i upiti zasnovani na opsegu. MongoDB sortira i vraća rezultate pomoću sekvencijalnog reda indeksa. Indeksi MongoDB-a su slični indeksima u bilo kojoj drugoj bazi podataka. MongoDB definiše indekse na nivou zbirke za upotrebu u bilo kojem polju ili potpolju. [3]

Vrste indeksa

MongoDB podržava sledeće tipove indeksa za upite:

- **Indeks podrazumevanog _id-ja** (***Default _id Index***)
- **Indeks jednog polja** (***Single Field Index***)
- **Složeni indeks** (***Compound Index***)
- **Višestruki-ključ indeks** (***Multikey Index***)
- **Geoprostorni indeks** (***Geospatial Index***)
- **Heširani indeks** (***Hashed Index***)

Indeks podrazumevanog _id-ja (Default _id Index)

Svaka MongoDB kolekcija sadrži indeks na podrazumevanom polju _id (čita se "donja crta" id). Ako nijedna vrednost nije navedena za _id, jezički upravljački program (drajver) ili mongod (čita se kao mongo D) kreira _id polje i obezbeđuje ObjectId (čita se kao Object ID) vrednost.

Indeks jednog polja (Single Field Index)

Za pojedinačni indeks i sortiranje, redosled sortiranja indeksnih ključeva nije važan tako da MongoDB može da iterira preko indeksa u rastućem ili opadajućem redosledu.

MongoDB podržava indekse na bilo kom dokumentu koji se nalazi u kolekciji. Podrazumevano, polje _id u svim kolekcijama ima indeks. Štaviše, aplikacije i korisnici dodaju indekse za pokretanje upita i izvršavanje operacija. MongoDB podržava oba, pojedinačna polja ili više polja indeksa na osnovu operacija koje indeksni tip obavlja.

```
db.items.createIndex( { “item” : 1 } )
```

Navedena naredba se koristi za kreiranje indeksa u polju stavke za zbirku stavki. U sledećem odeljku opisan je način kako kreirati pojedinačne indekse polja na ugrađenim dokumentima.

Indeks jednog polja na dokumentu (Single Field Index on Embedded Document)

Mogu se indeksirati polja najvišeg nivoa u dokumentu. Slično tome, mogu se kreirati indeksi unutar polja ugrađenog dokumenta.

```
{ "_id" : 3, "item" : "Book", "available" : true, "soldQty" : 144821, "category" : "NoSQL",
"details" : { "ISDN" : "1234", "publisher" : "XYZ Company" }, "onlineSale" : true }
```

Gore prikazana struktura odnosi se na dokument koji se nalazi u kolekciji. Polje sa detaljima u dokumentu prikazuje ugrađeni dokument koji ima dva ugrađena polja - ISDN i izdavač.

```
db.items.createIndex( {details.ISDN: 1 } )
```

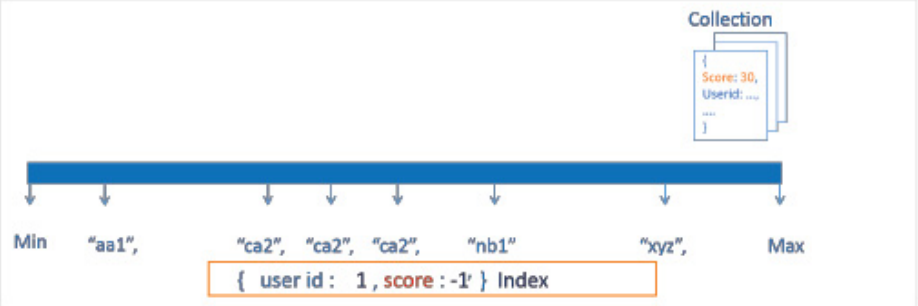
Da bi se kreirao indeks na ISDN polju i ugrađeni dokument koji se zove "detalji", potrebno je izvršiti gore prikazane upite.

Složeni indeks (Compound Index)

Za više polja, MongoDB podržava korisnički definisane indekse, kao što su složeni indeksi. Sekvencijalni redosled polja u indeksu složenosti je značajan u MongoDB-u. Složeni indeks sadrži više indeksa pojedinačnih polja odvojenih zapetom.

```
db.products.createIndex( { "item": 1, "stock": 1 } )
```

Gore prikazana naredba je primer složenog indeksa na dva polja.



Slika 15 - Složeni indeks

Ovaj dijagram prikazuje složeni indeks za polja, *user id* i *score*. Dokumenti su prvo organizovani od strane *user id*-a i unutar svakog *user id*-a, rezultati su organizovani po opadajućem redosledu. Redosled sortiranja polja u složenom indeksu je presudan. Dokumenti se prvo sortiraju po vrednosti polja stavke, a zatim, u okviru svake vrednosti polja stavke, dalje se sortiraju po vrednostima polja zaliha. Za složeni indeks, MongoDB ograničava polja na maksimalnu vrednost 31.

Višestruki-ključ indeks (Multikey Index)

Za indeksiranje podataka iz niza, MongoDB koristi *multikey* indekse. Kada indeksirate polje sa vrednosti niza, MongoDB pravi odvojene unose indeksa za svaki element niza.

Kada se vrši indeksiranje polje koje sadrži vrednost niza, MongoDB kreira odvojene unose indeksa za svaku komponentu niza. Ovi *multikey* indeksi u upitima odgovaraju elementima matrice sa dokumentima koji sadrže nizove i selektuju ih. Mogu se napraviti *multikey* indeksi za nizove koji sadrže skalarne vrednosti, kao što su stringovi, brojevi i ugnežđeni dokumenti.

```
db.coll.createIndex( { : < 1 or -1 > } )
```

Da bi se kreirao *multikey* indeks, može se koristiti metod *db.collection.createIndex()*. Ako indeksirano polje sadrži niz, MongoDB automatski odlučuje da li će kreirati *multikey* indeks ili ga neće kreirati. Ne morate eksplicitno navesti tip *multikey*.

Složeni višestruki-ključ indeks (Compound MultiKey Index)

U složenim *multikey* indeksima, svaki indeksirani dokument može imati maksimalno jedno indeksirano polje sa vrednošću niza. Ako više od jednog polja ima vrednost niza, ne možete da kreirate složeni *multikey* indeks.

```
{ _id: 1, product_id: [ 1, 2 ], retail_id: [ 100, 200 ], category: "both fields are arrays" }
```

Primer strukture dokumenta je prikazan iznad. U ovoj kolekciji, polja *product_id* i *retail_id* su polja. Prema tome, ne može se kreirati složeni *multikey* indeks.

```
Note that a shard key index and a hashed index cannot be a multikey index.
```

Geoprostorni indeks (Geospatial Index)

Za upit geoprostornih podataka, MongoDB koristi dva tipa indeksa:

- 2d indekse (čita se kao dva D indeks) i 2d sferu (čita se kao dva D sfera) indeksa
- Tekstualni indeksi - ovi indeksi u MongoDB-u pretražuju string podatak u kolekciji

Uz povećano korišćenje ručnih uređaja, geoprostorni upiti postaju sve učestaliji za pronalaženje najbližih tačaka podataka za datu lokaciju. MongoDB pruža geoprostorne indekse za koordinaciju takvih upita. Pretpostavimo da je potrebno pronaći najbliži kafić sa Vaše trenutne lokacije. Potrebno je kreirati poseban indeks da bi se efikasno izvršio takav upit, jer treba vršiti pretragu u dve dimenzije - dužina i širina. Geoprostorni indeks se kreira pomoću funkcije *createIndex*. Ona prosleđuje "2d" ili "*2dsphere*" kao vrednost umesto 1 ili -1. Da bi se tražili geoprostorni podaci, neophodno je kreirati geoprostorni indeks.

```
db.collection.createIndex( { : "2dsphere" } )
```

U dokumentu specifikacije indeksa za metodu *db.collection.createIndex()*, kao što je prikazano iznad, navedeno je polje za lokaciju kao ključ indeksa i naveden je *string* literal "*2dsphere*" kao vrednost. Složeni indeks može da sadrži ključ indeksa *2dsphere* u kombinaciji sa ključevima ne-geoprostornog indeksa.

Heširani indeks (Hashed Index)

MongoDB podržava indeksiranje zasnovano na heš funkcijama (*hash-based sharding*), koje daje heš indeks. Time se indeksiraju heš vrednosti polja.

Slede karakteristike heš funkcije:

- Heš funkcija kombinuje sve ugrađene dokumente i izračunava heševe za sve vrednosti polja.
- Heš funkcija ne podržava više ključeva.
- Heš indeksi podržavaju *sharding*, koriste heš *shard* ključ za *shard* kolekcije, čime osiguravaju ravnomernu distribuciju podataka.
- Heš indeksi podržavaju upite jednakosti (*equality queries*), međutim, upiti raspona (*range queries*) nisu podržani.

Ne može se kreirati jedinstveni ili složeni indeks uzimajući polje čiji je tip heširan. Međutim, mogu se kreirati heširani i ne-heširani indeks za isto polje. MongoDB koristi skalarni indeks za opseg upita.

```
db.items.createIndex( { item: "hashed" } )
```

Mogu se kreirani indeksi koristeći operaciju koja je data iznad. Ovo će kreirati heširani indeks za kolekciju stavki u polju stavke.

Svojstva indeksa

Kod MongoDB-a postoje dva svojstva indeksiranja:

- **Jedinstvo indeksiranje** (***Unique indexes***)
- **Retko indeksiranje** (***Sparse indexes***)

Kao specijalni tip indeksiranja javlja se takozvano "totalno vreme za život" odnosno ***TTL* indeksiranje** (***Time-To-Live indexes***).

Unique indexes

Jedinstvena svojstva MongoDB indeksa osiguravaju da su duple vrednosti za indeksirano polje budu odbijene. Osim toga, jedinstveni indeksi mogu se funkcionalno zameniti sa drugim MongoDB indeksima.

Da bi se kreirao jedinstveni indeks, koristi se metodu *db.collection.createIndex()* i postavlja se *unique* opciju na *true*.

```
db.items.createIndex( { "item": 1 }, { unique: true } )
```

Na primer, da bi se kreirao jedinstveni indeks u polju stavke kolekcije stavki, izvršava se operacija prikazaau iznad. Podrazumevano, *unique* parametar je postavljen na *false*.

Sparse indexes

Ovo svojstvo osigurava da se upitima pretražuju stavke dokumenta koje imaju indeksirano polje. Dokumenti bez indeksiranih polja su preskočeni tokom upita. Retko indeksiranje i jedinstveno indeksiranje mogu se kombinovati kako bi se odbacili dokumenti sa dupliciranim vrednostima polja i ignorisali dokumenati bez indeksiranih ključeva.

Retki indeksi upravljaju samo onim dokumentima koji imaju indeksirana polja, čak i ako to polje sadrži *null* vrednosti. *Sparse* indeks zanemaruje one dokumente koji ne sadrže polje indeksa. Ne-retki indeksi ne ignorišu ove dokumente i čuvaju *null* vrednosti za njih. Da bi se kreirao retki indeks, koristi se metoda *db.collection.createIndex()* i postavlja se sparse opciju na *true*.

```
db.addresses.createIndex( { "xmpp_id": 1 }, { sparse: true } )
```

U gore navedenom primeru, operacija u mongo *shell*-u kreira retki indeks na polju stavke u kolekciji stavki. Ako retki indeks vrati nekompletan indeks, onda MongoDB ne koristi taj indeks osim ako nije naveden u metodi hinta (*hint method*).

```
{ x: { $exists: false } }
```

Na primer, druga gore navedena naredba neće koristiti retki indeks na polju *x* osim ako ne primi eksplicitne savete. Indeks koji kombinuje i retko i jedinstveno ne dozvoljava kolekciji da uključi dokumente koji imaju duple vrednosti polja za jedno polje. Međutim, on omogućava višestruke dokumente koji izostavljaju ključ.

Total time to live – TTL indexes

TTL su specijalni tipovi indeksa u MongoDB, koji se koriste za automatsko brisanje dokumenata iz kolekcije nakon određenog vremenskog perioda. Ovo je idealno za brisanje informacija, kao što su podaci generisani od strane mašina (*machine-generated data*), evidencije događaja (*event logs*) i podaci o sesiji (*session data*) koji moraju biti u bazi podataka za kraće trajanje.

TTL indeksi automatski brišu mašinski generisane podatke. *TTL* indeks Mogu se može kreirati kombinovanjem metode *db.collection.createIndex()* sa opcijom *expireAfterSeconds* na polju čija je vrednost ili datum ili niz koji sadrži vrednosti datuma.

```
db.eventlog.createIndex( { "lastModifiedDate": 1 }, { expireAfterSeconds: 3600 } )
```

Na primer, da bi se kreirao *TTL* indeks na polju *lastModifiedDate* (datum poslednje izmene) kolekcije *eventlog*, koristi se operacija prikazana iznad u mongo *shell*-u. *TTL* pozadinska nit radi na primarnim i sekundarnim čvorovima. Međutim, dokument se briše samo iz primarnog čvora.

TTL indeksi imaju sledeća ograničenja:

- Nisu podržani složenim indeksima koji ignorišu *expireAfterSeconds*
- Polje _id ne podržava *TTL* indekse.
- *TTL* indeksi se ne mogu kreirati na ograničenoj kolekciji jer MongoDB ne može da izbriše dokumente iz ograničene kolekcije.
- Ne dozvoljava metodi *createIndex()* da promeni vrednost *expireAfterSeconds* postojećeg indeksa.

Ne može se kreirati *TTL* indeks za polje ako ne postoji *TTL* indeks za isto polje. Ako je potrebno promeniti ne-*TTL* indeks jednog polja u *TTL* indeks, prvo se briše (*drop*) indeks, a potom ponovo kreira (*create*) indeks sa opcijom *expireAfterSeconds*.

Agregacija

Osnovni pojmovi

Operacije koje obrađuju skupove podataka i povratne izračunate rezultate nazivaju se agregacije. MongoDB obezbeđuje agregaciju podataka koje ispituju skupove podataka i izvršavaju izračune na njima. Agregacija se izvodi na *mongod* instanci kako bi se pojednostavili kodovi aplikacija i ograničili zahtevi resursa.

Slično kao i upiti, operacije agregacije u MongoDB-u koriste kolekcije dokumenata kao ulazne podatke i vraćaju rezultate u obliku jednog ili više dokumenata.

Dokumenti prolaze kroz višefazne *pipeline*-ove i pretvaraju se u agregirani rezultat. Najosnovnija faza *pipeline*-a u okviru agregacije obezbeđuje filtere koji funkcionišu kao upiti. Takođe obezbeđuje transformacije dokumenata koje modifikuju izlazni dokument.

Operacije *pipeline*-a grupišu i sortiraju dokumente po definisanim poljima ili poljima. Pored toga, oni izvršavaju agregaciju na nizovima. Faze *pipeline*-a mogu koristiti operatore za izvršavanje zadataka kao što su izračunavanje proseka ili povezivanje niza. *Pipeline* koristi prirodne operacije unutar MongoDB-a kako bi omogućio efikasnu agregaciju podataka i predstavlja omiljeni metod za agregaciju podataka.

Pomoću funkcije agregacija mogu se izvršiti složene operacije agregacije, kao što je pronalaženje ukupnog iznosa transakcije za svakog klijenta. [3]

Pipeline Operators and Indexes

Kombinacija agregata u MongoDB funkciji na jednoj kolekciji, logički prenosi kolekciju kroz *pipeline* agregacije. Mogu se optimizovati operacije i izbegavati skeniranje cele kolekcije korišćenjem *$match*, *$limit* i *$skip* faza.

Možda će se zahtevati samo podskup podataka iz kolekcije da bi se izvršila operacija združivanja. Stoga, koriste se faze *$match*, *$limit* i *$skip* za filtriranje dokumenata. Kada se postavi na početak *pipeline*-a, operacija *$match* skenira i bira samo odgovarajuće dokumente u kolekciji.

Primer: Navedena operacija agregacije vraća sve države sa ukupnim brojem stanovnika većim od 10 miliona.

```
db.zipcodes.aggregate( [{ $group: { _id: "$state", totalPop: { $sum: "$pop" } } },

{ $match: { totalPop: { $gte: 10*1000*1000 } } }] )
```

Ovaj primer opisuje da *pipeline* agregacije sadrži *$group* fazu nakon čega sledi faza *$match*.

U ovoj operaciji, *$group* faza odrađuje tri celine:

- Grupiše dokumente za zip kod u polje stanja
- Izračunava polje *TotalPop* (ukupna populacija) za svaku državu
- Vraća izlazni dokument za svako jedinstveno stanje

Novi dokumenti po-stanju sadrže dva polja: polje *_id* i polje *totalPop*. U ovoj komandi se koristi agregatni *pipeline*. Faza *$sort* sortira ove dokumente i *$group* faza primenjuje operaciju *sum* na polja iznosa tih dokumenata.

```
db.users.aggregate([{ $project :{month_joined : { $month : "$joined" },name :
"$_id",_id : 0}},{ $sort : { month_joined : 1 } } ])
```

Druga operacija agregacije koja je prikazana iznad vraća korisnička imena sortirana prema mesecu njihovog pridruživanja. Ova vrsta agregacije može pomoći u generisanju obaveštenja o obnavljanju članstva.

Optimizacija performansi

Odlične performanse baze podataka su važne kada se razvijaju aplikacije pomoću MongoDB-a. Ponekad se celokupni proces posluživanja podataka može degradirati zbog niza razloga, od kojih neki uključuju:

- Neodgovarajući obrasci dizajna šeme
- Neodgovarajuća ili ne upotrebljena strategija indeksiranja
- Neodgovarajući hardver
- Kašnjenje replikacije
- Loše izvođenje tehnika upita

Neke od ovih prepreka mogu primorati korisnike da povećaju hardverske resurse, dok drugi ne. Na primer, loše strukture upita mogu dati rezultate u vidu dugog procesiranja upita, uzrokujući zaostajanje replike i možda čak i gubitak podataka. U

ovom slučaju, može se pomisliti da memorija za skladištenje podataka možda nije dovoljna, i da je verovatno potrebno njeno proširenje. Neke od naprikladnijih procedura koje se mogu koristiti za poboljšanje performansi date su u nastavku. [4]

Dizajn šeme

U suštini, dva najčešće korišćena tipa veza u dizajnu šema (***Schema Design***) su:

- Jedan-na-nekoliko (*One-to-few*)
- Jedan-na-više (*One-to-many*)

Iako je najefikasniji dizajn šeme odnos jedan prema više, svaki od njih ima svoje prednosti i ograničenja.

One-to-few

U ovom slučaju, za dato polje postoje ugrađeni dokumenti, ali oni nisu indeksirani sa identitetom objekta. Evo jednostavnog primera:

```
{
      userName: "Brian Henry",
      Email : "example@gmail.com",
      grades: [
            {subject: 'Mathematics',  grade: 'A'},
            {subject: English,  grade: 'B'},
      ]
}
```

Jedna od prednosti korišćenja ovog tipa veze jeste da mogu da se dobiju ugrađeni dokumenti sa samo jednim upitom. Međutim, sa stanovišta upita, ne može se pristupiti jednom ugrađenom dokumentu. Dakle, ukoliko se ne referencira na ugrađene dokumente odvojeno, ovaj dizajn šeme je optimalan.

One-to-many

Za ovaj tip veze podaci u jednoj bazi podataka se odnose na podatke u drugoj bazi podataka. Na primer, može postojati baza podataka za korisnike, a druga za postove. Dakle, ako korisnik napravi post, on se snima sa *ID*-jem korisnika.

Šema za korisnike

```
{
    Full_name: "John Doh",
    User_id: 1518787459607.0
}
```

Šema za postove

```
{
    "_id" : ObjectId("5aa136f0789cf124388c1955"),
    "postTime" : "16:13",
    "postDate" : "8/3/2018",
    "postOwnerNames" : "John Doh",
    "postOwner" : 1518787459607.0,
    "postId" : "1520514800139"
}
```

Prednost ovog dizajna šeme je u tome što se dokumenti smatraju samostalnim (mogu se odabrati zasebno). Još jedna prednost je da ovaj dizajn omogućava

korisnicima različitih *ID*-jeva da dele informacije iz šeme sa porukama (otuda ime *One-to-Many*), a ponekad mogu biti „N-na-N“ šeme - u suštini bez korišćenja tabele. Ograničenje kod ovog dizajna šeme je to što se moraju napraviti najmanje dva upita za pribavljanje ili odabir podataka u drugoj kolekciji. Način modelovanja podataka zavisi od obrasca pristupa aplikacije. Osim toga, potrebno je razmotriti dizajn.

Tehnike optimizacije za dizajna šeme

- Koristiti ugrađivanje dokumenata koliko god je to moguće jer smanjuje broj upita koji su potrebni za određeni skup podataka
- Ne koristite denormalizaciju za dokumente koji se često ažuriraju. Ako će se polje često ažurirati, onda je zadatak pronaći svie instance koje treba ažurirati. Ovo će dati spore rezultate prilikom obrade upita, pa će stoga biti nadjačane čak i zasluge povezane s denormalizacijom
- Ako postoji potreba da se dokument odvojeno pribavi, onda nema potrebe da se koristi ugrađivanje pošto kompleksni upiti, kao što je agregatni *pipeline*, zahtevaju više vremena za izvršenje
- Ako je niz dokumenata koje treba ugraditi dovoljno veliki, ne vršiti ugrađivanje istih. Rast polja trebalo bi da ima bar ograničenu granicu

Pravilno indeksiranje

Pravilno indeksiranje (***Proper Indexing***) predstavlja kritičniji deo ugađanja performansi i zahteva da neko ima sveobuhvatno razumevanje upita za aplikacije, odnos čitanja sa zapisima kao i predstavu o količini slobodne memorije koju sistem poseduje. Ako se koristi indeks, upit će skenirati indeks, a ne kolekciju. Odličan indeks je onaj koji uključuje sva polja skenirana upitom. Ovo se naziva složeni indeks, koji je opisan u sekciji *Compound index*.

Kreiranje jednog indeksa za polja se vrši na sledeći način:

```
db.collection.createIndex({“fields”: 1})
```

Kreiranje složenog indeksa se vrši na sledeći način:

```
db.collection.createIndex({“filed1”: 1, “field2”:  1})
```

Pored bržeg načina zadavanja upita korišćenjem indeksiranja, postoji i dodatna prednost drugih operacija kao što su sortiranje, uzorci i ograničenja. Na primer, ako je šema dizajnirana kao {f: 1, m: 1} može se napraviti dodatna operacija pored pretrage:

```
db.collection.find( {f: 1} ).sort( {m: 1} )
```

Čitanje podataka iz RAM-a je efikasnije od čitanja istih podataka sa diska. Iz tog razloga, uvek se savetuje da se obezbedi da se indeks u potpunosti uklapa u RAM. Da bi se dobio trenutni *indexSize* kolekcije, izvršiti naredbu:

```
db.collection.totalIndexSize()
```

Dobija se vrednost otprilike od 36864 bajta. Ova vrednost takođe ne bi trebalo da uzima veliki procenat ukupne veličine RAM-a, pošto treba da zadovolji potrebe celog radnog skupa servera.

Efikasan upit takođe treba da poboljša selektivnost (*selectivity*). Selektivnost se može definisati kao sposobnost upita da suzi rezultat koristeći indeks. Zbog sigurnosti, upiti bi trebali ograničiti broj mogućih dokumenata sa indeksiranim poljem. Selektivnost je uglavnom povezana sa složenim indeksom koji uključuje polje niske selektivnosti i drugo polje. Na primer, ako su dati sledeći podaci:

```
{ _id: ObjectId(), a: 6, b: "no", c: 45 }
{ _id: ObjectId(), a: 7, b: "gh", c: 28 }
{ _id: ObjectId(), a: 7, b: "cd", c: 58 }
{ _id: ObjectId(), a: 8, b: "kt", c: 33 }
```

Upit {a: 7, b: "cd"} će se skenirati kroz dva dokumenta da bi vratio jedan odgovarajući dokument. Naravno, u slučaju da su podaci za traženu vrednost ravnomerno raspoređeni.

```
{ _id: ObjectId(), a: 6, b: "no", c: 45 }
{ _id: ObjectId(), a: 7, b: "gh", c: 28 }
{ _id: ObjectId(), a: 8, b: "cd", c: 58 }
{ _id: ObjectId(), a: 9, b: "kt", c: 33 }
```

Upit {a: 7, b: "cd"} će se skenirati kroz jedan dokument i vratiti ovaj dokument. Stoga će za to biti potrebno kraće vreme od prve strukture podataka.

Obezbeđivanje resursa

Neadekvatna memorija za skladištenje, RAM i drugi operativni parametri mogu drastično smanjiti performanse MongoDB-a. Za pravilan i efikasan rad potrebno je obezbediti odgovarajuće resurse (***Resources Provisioning***). Na primer, ako je broj korisničkih veza veoma veliki, to će ometati sposobnost serverske aplikacije da blagovremeno obradi zahteve. Jedna od ključnih stvari za nadgledanje u MongoDB-u, kojom se može dobiti pregled o tome koji ograničeni resursi su na raspolaganju i kako ih se mogu prilagoditi specifikacijama. Za veliki broj istovremenih zahteva za aplikacijama, sistem baze podataka će biti nadjačan u skladu sa zahtevima.

Efikasne tehnike upita

Pored kreiranja indeksiranih upita i korišćenja selektivnosti upita (*Query Selectivity*), kao što je gore razmotreno, postoje i drugi koncepti koji se mogu koristiti da bi se učvrstili i učinili upite efikasnim (***Efficient Query Techniques***).

Korišćenje pokrivenog upita

Pokriveni upit je onaj koji je uvek potpuno zadovoljan indeksom, stoga ne treba ispitivati nijedan dokument. Pokriveni upit bi stoga trebao imati sva polja kao deo indeksa i za posledicu bi rezultat trebao sadržati sva ova polja.

Na primer:

```
{_id: 1, product: { price: 50 }
```

Ukoliko se kreira indeks za ovu kolekciju:

```
{“product.price”: 1}
```

Imajući u vidu operaciju pronalaženja, ovaj indeks će pokriti ovaj upit i vratiti samo polje proizvoda i vrednosti.

```
db.collection.find( {“product.price”: 50}, {“product.price”: 1, _id: 0}  )
```

Notacija tačaka

Za ugrađene dokumente koristi se notacija tačaka (.). Oznaka tačka pomaže u pristupu elementima niza i poljima ugrađenog dokumenta. Dat je sledeći primer.

Pristupanje nizu:

```
{
   prices: [12, 40, 100, 50, 40]
}
```

Da se navede četvrti element niza koristi se komanda:

```
“prices.3”
```

Pristupanje nizu objekata se vrši naredbom:

```
{

   vehicles: [{name: toyota, quantity: 50},
              {name: bmw, quantity: 100},
              {name: subaru, quantity: 300}
}
```

Da bi se odredio naziv polja koristi se sledeća naredba:

```
“vehicles.name”
```

Provera pokrivenog upita

Provera da li je pokriven upit se vrši funkciojm *db.collection.explain()*. Ova funkcija će pružiti informacije o izvršavanju drugih operacija. Na primer: *db.collection.explain().aggregate()*.

Zaključak: Uopšteno, vrhunska tehnika što se tiče upita je korišćenje indeksa. Upit samo indeksa je mnogo brži od upita dokumenata izvan indeksa. Oni se mogu uklopiti u memoriju, dakle u RAM, a ne u disku. To ih čini dovoljno lakim i brzim da se pribavljaju iz memorije.

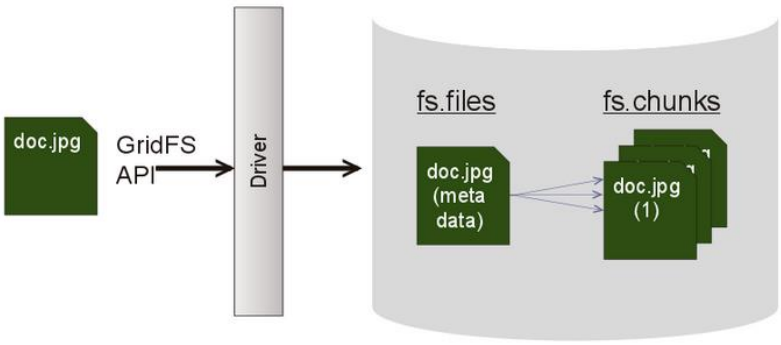
Skladištenje

Mehanizam za skladištenje je primarna komponenta MongoDB-a koja je odgovorna za upravljanje podacima. MongoDB obezbeđuje razne mehanizme za skladištenje, omogućavajući da se izaberu oni koji najviše odgovaraju aplikaciji. Dnevnik (*journal*) je *log* koji pomaže bazi podataka da se oporavi u slučaju tvrdog isključivanja (*hard shutdown*). Postoji nekoliko opcija koje se mogu konfigurisati i koje omogućavaju dnevniku da uspostavi ravnotežu između performansi i pouzdanosti koja funkcioniše za određeni slučaj korišćenja. *GridFS* je svestrani sistem za skladištenje

koji je pogodan za rukovanje velikim datotekama, kao što su one koje prelaze ograničenje veličine dokumenta od 16 MB. [5]

MongoDB GridFS

GridFS omogućava skladištenje više podataka u jednom MongoDB dokumentu nego što je dozvoljeno *BSON* specifikacijom od 16MB. Ovo omogućava da se zapis baze podataka podeli na odvojene datoteke na disku i može omogućiti neke prilično specifične koristi pored onih koje su navedene u *GridFS* dokumentaciji. [6]



Slika 16 – GridFS

Na primer, sa replikacijom, moguće je da se datoteke razmeštaju na više objekata i da se koriste funkcije za sinhronizaciju MongoDB-a. Međutim, mnogi sistemi datoteka sami na sebe računaju.

Praktični deo

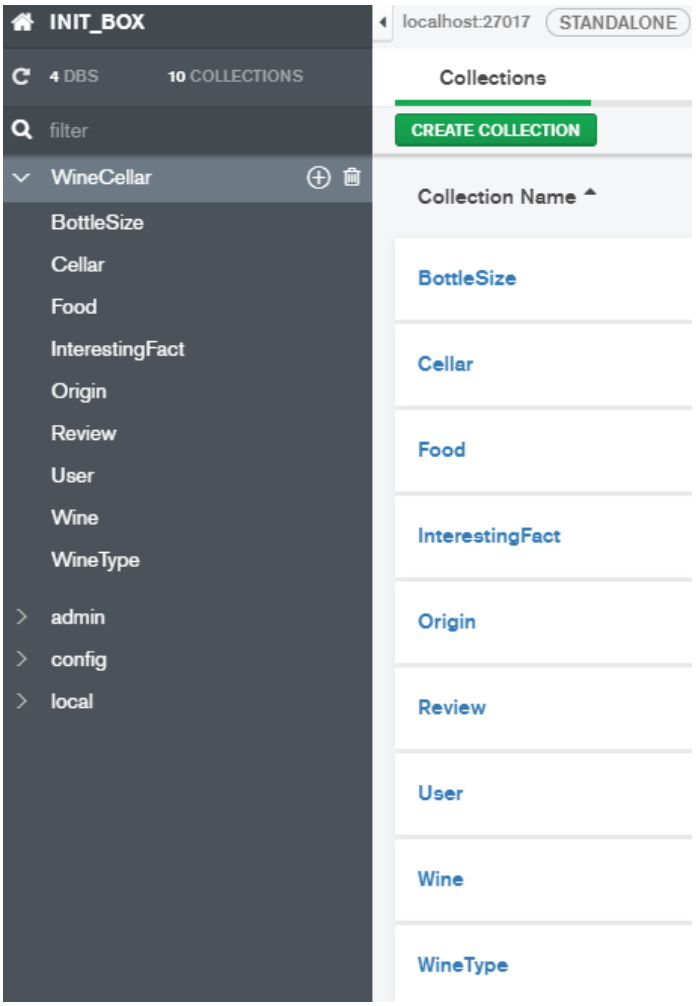
U nastavku slede primeri nad lokalnom bazom podataka *WineCellar*, koja skladišti podatke o podrumu vina. Zbog boljeg prikaza podataka i kvalitetnije vizuelizacije, pored *mongod shell*-a, kao pomoćno okruženje korišćen je *MongoDB Compass Community*.

Primeri

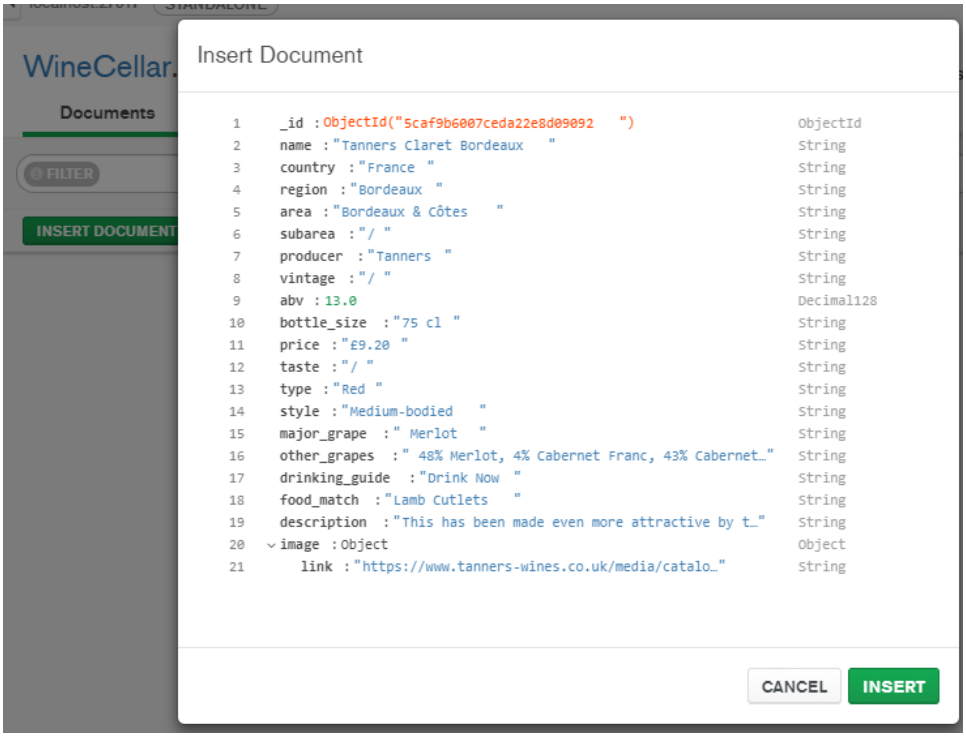
Primeri paralelno prate rad u *MongoDB shell* konzolnoj aplikaciji i *MongoDB Compass* okruženju. Najčešće biće korišćene kolekcije *Wine* i *WineType*, koje čuvaju podatke o vinu odnosno o tipu vina, respektivno. Rad je ilustrovan slikama, pri čemu uz svaku sliku stoji odgovarajući tekst u vidu opisa. Ukoliko je potrebno dodatno objašnjenje, biće dato iznad ili ispod odgovarajuće slike.

```
> use WineCellar
switched to db WineCellar
> show collections
BottleSize
Cellar
Food
InterestingFact
Origin
Review
User
Wine
WineType
```

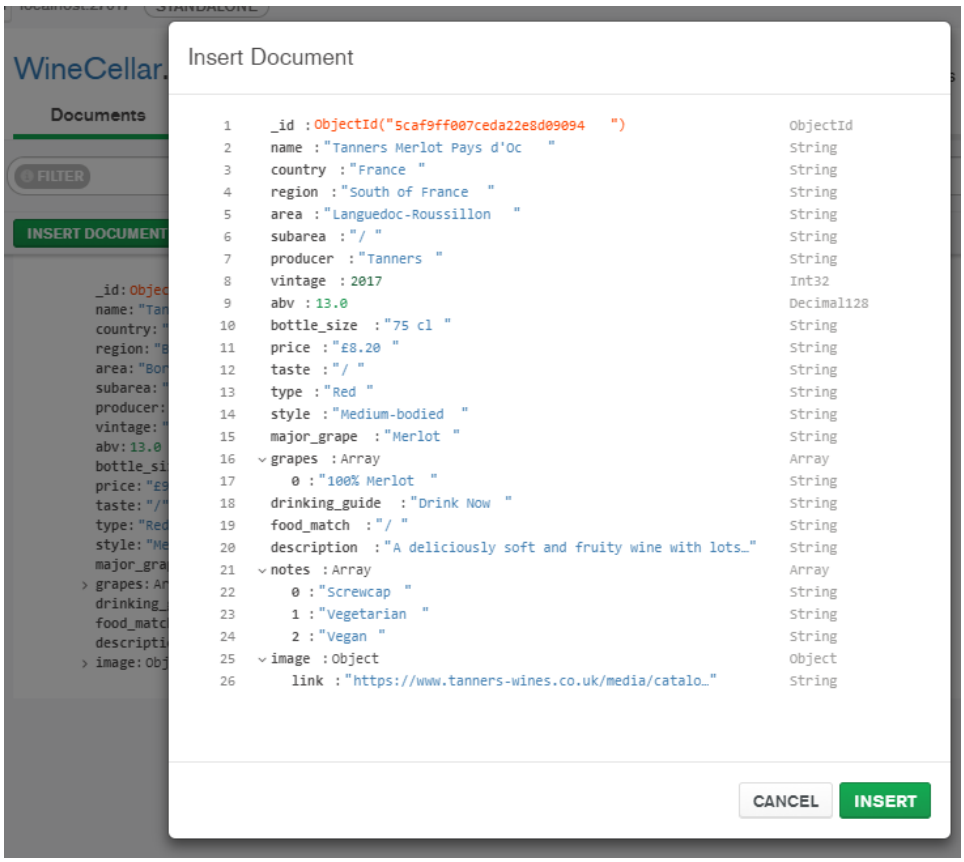
Slika 17 - Postavljanje baze za rad i pregled kolekcija



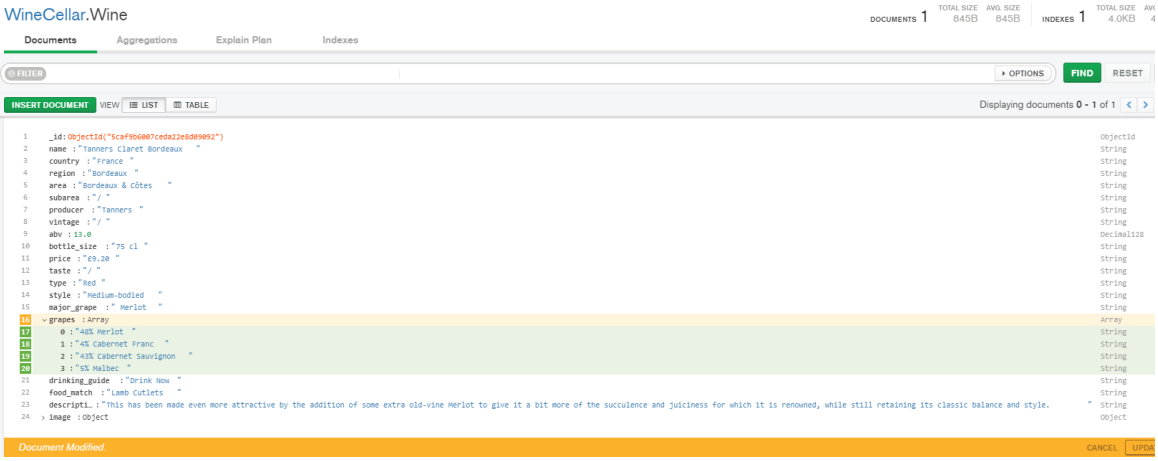
Slika 18 - Instanca WineCellar i njene kolekcije



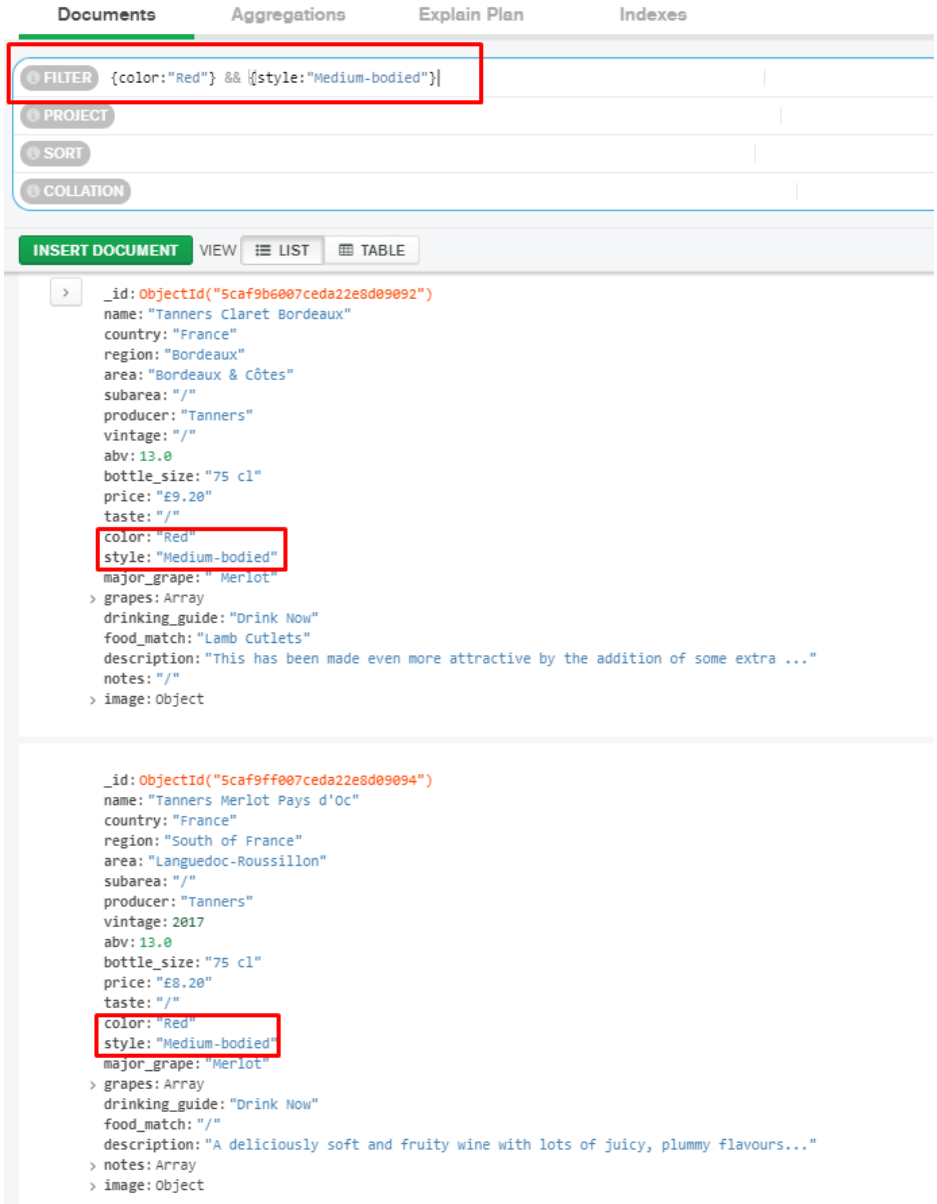
Slika 19 - Dodavanje dokumenta u kolekciju Wine



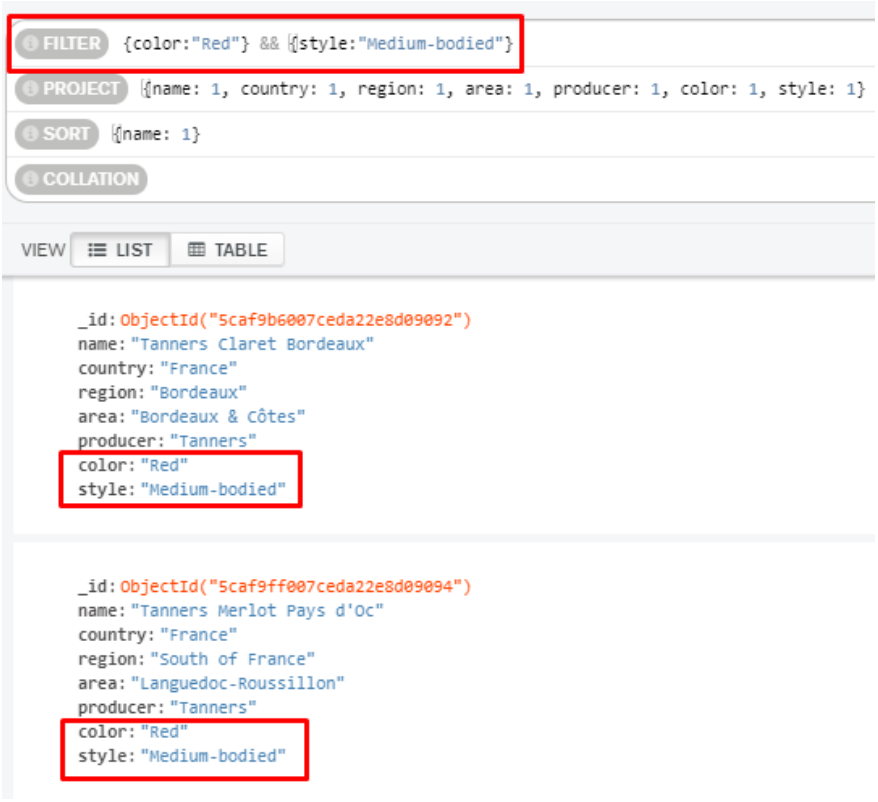
Slika 20 - Dodavanje drugog dokumenta u kolekciju Wine



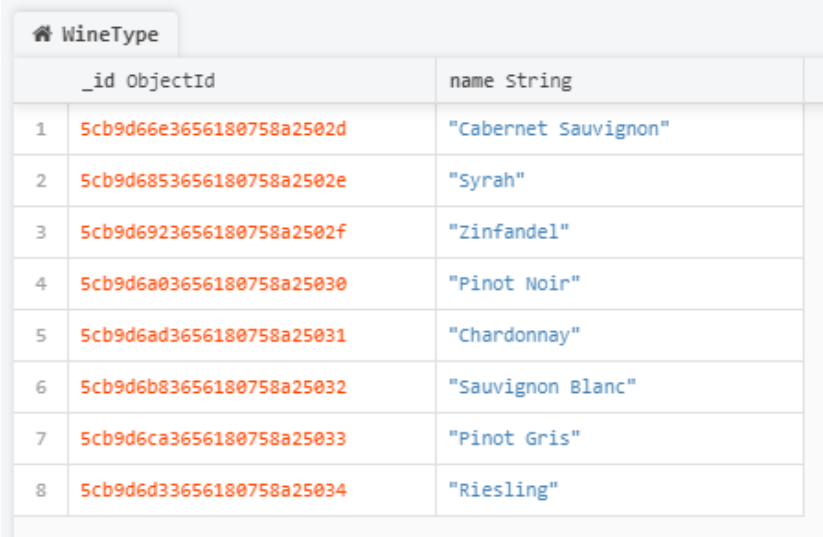
Slika 21 - Ažuriranje dokumenta u kolekciji Wine



Slika 22 - Jednostavan upit primenom filter opcije nad kolekcijom Wine



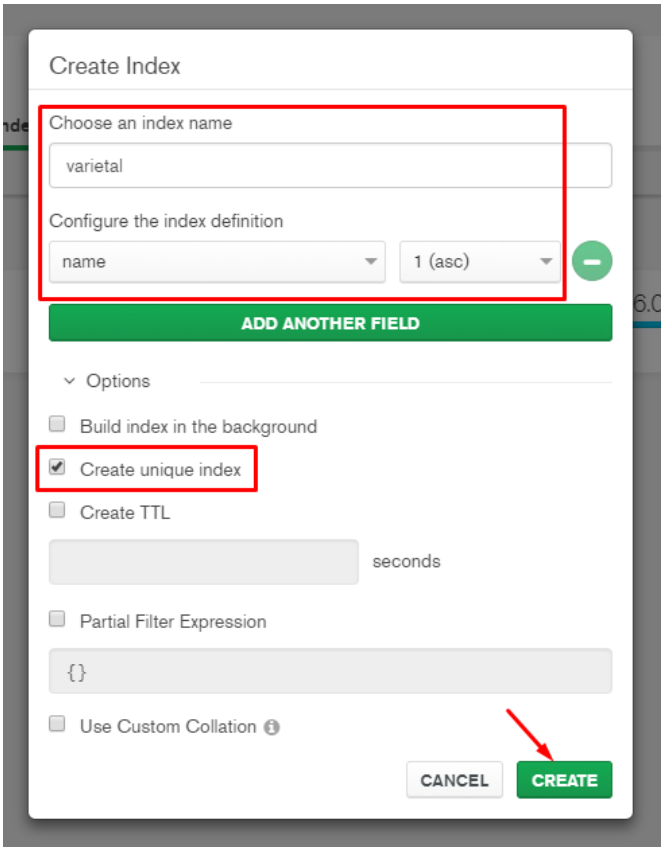
Slika 23 - Složeni upit primenom filter, project i sort opcija nad kolekcijom Wine



WineType

	_id ObjectId	name String
1	5cb9d66e3656180758a2502d	"Cabernet Sauvignon"
2	5cb9d6853656180758a2502e	"Syrah"
3	5cb9d6923656180758a2502f	"Zinfandel"
4	5cb9d6a03656180758a25030	"Pinot Noir"
5	5cb9d6ad3656180758a25031	"Chardonnay"
6	5cb9d6b83656180758a25032	"Sauvignon Blanc"
7	5cb9d6ca3656180758a25033	"Pinot Gris"
8	5cb9d6d33656180758a25034	"Riesling"

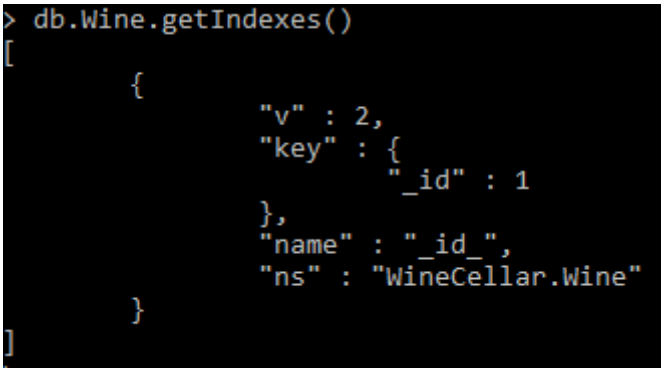
Slika 24 - Pregled kolekcije WineType



*Slika 25 - Dodavanje **unique** indeksa u kolekciju WineType*

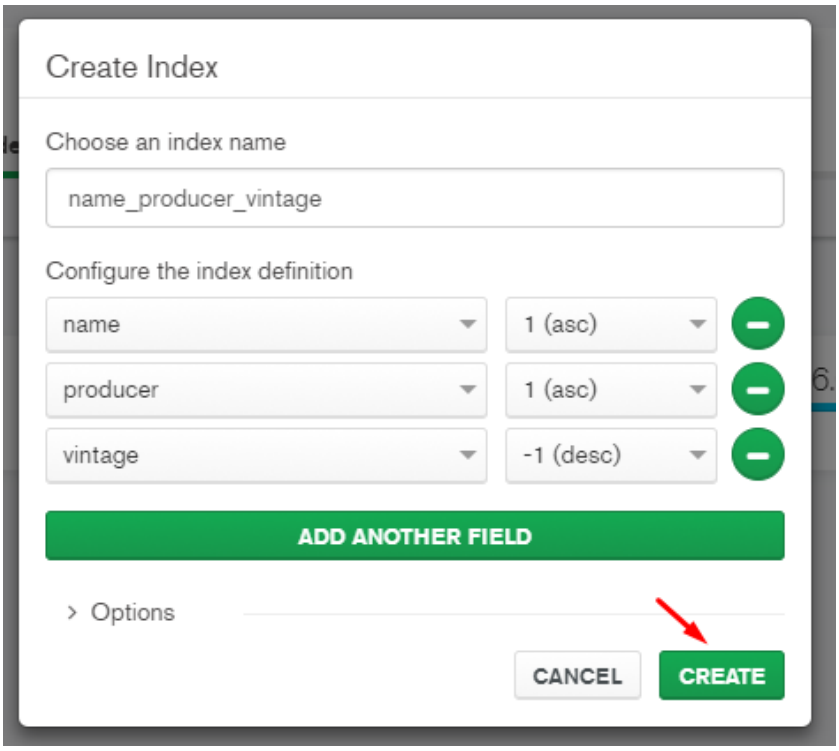
```
> db.WineType.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "WineCellar.WineType"
        },
        {
                "v" : 2,
                "unique" : true,
                "key" : {
                        "name" : 1
                },
                "name" : "varietal",
                "ns" : "WineCellar.WineType"
        }
]
```

Slika 26 - Izlistinani indeksi u kolekciji WineType



Slika 27 - Indeksi u kolekciji Wine pre dodavanja novog indeksa

Slika 28 - Indeksi u kolekciji Wine pre dodavanja novog indeksa

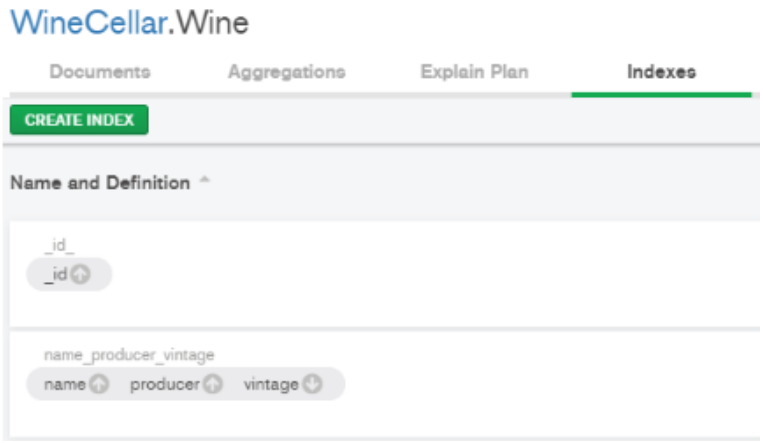


*Slika 29 - Dodavanje **compound** indeksa u kolekciji Wine*

```
> db.Wine.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "WineCellar.Wine"
        },
        {
                "v" : 2,
                "key" : {
                        "name" : 1,
                        "producer" : 1,
                        "vintage" : -1
                },
                "name" : "name_producer_vintage",
                "ns" : "WineCellar.Wine"
        }
]
```

Slika 30 - Indeksi u kolekciji Wine nakon dodavanja novog indeksa

Vrednost 1 označava da se sortiranje vrši u rastućem redosledu, dok vrednost -1 ukazuje na obrnuto odnosno na opadajući redosled.

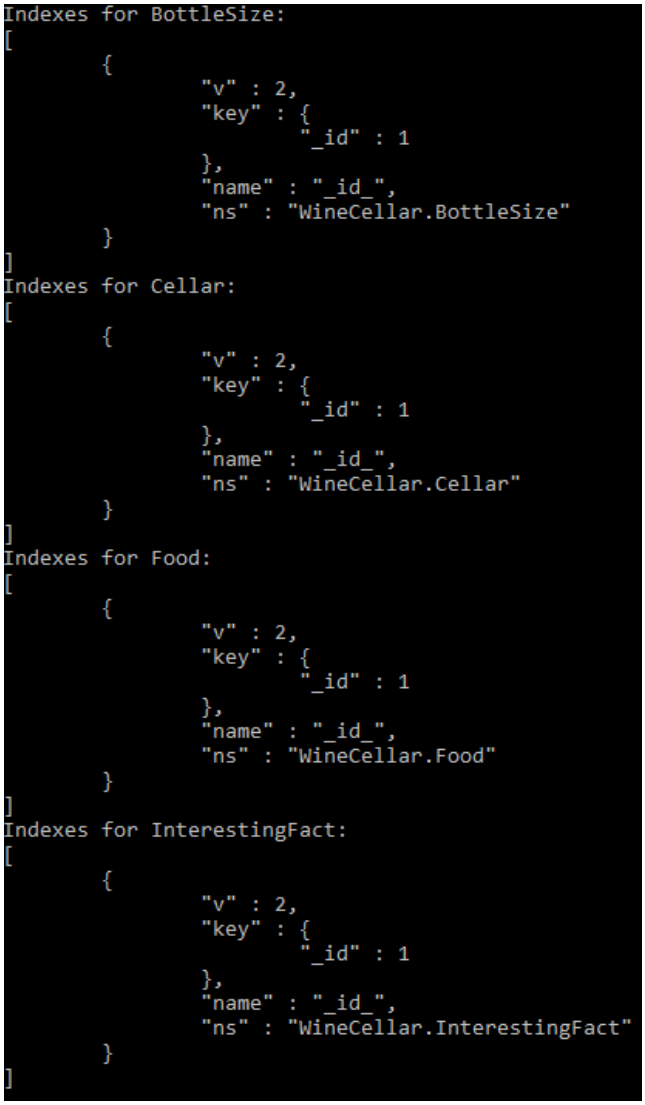


Slika 31 - Indeksi u kolekciji Wine nakon dodavanja novog indeksa

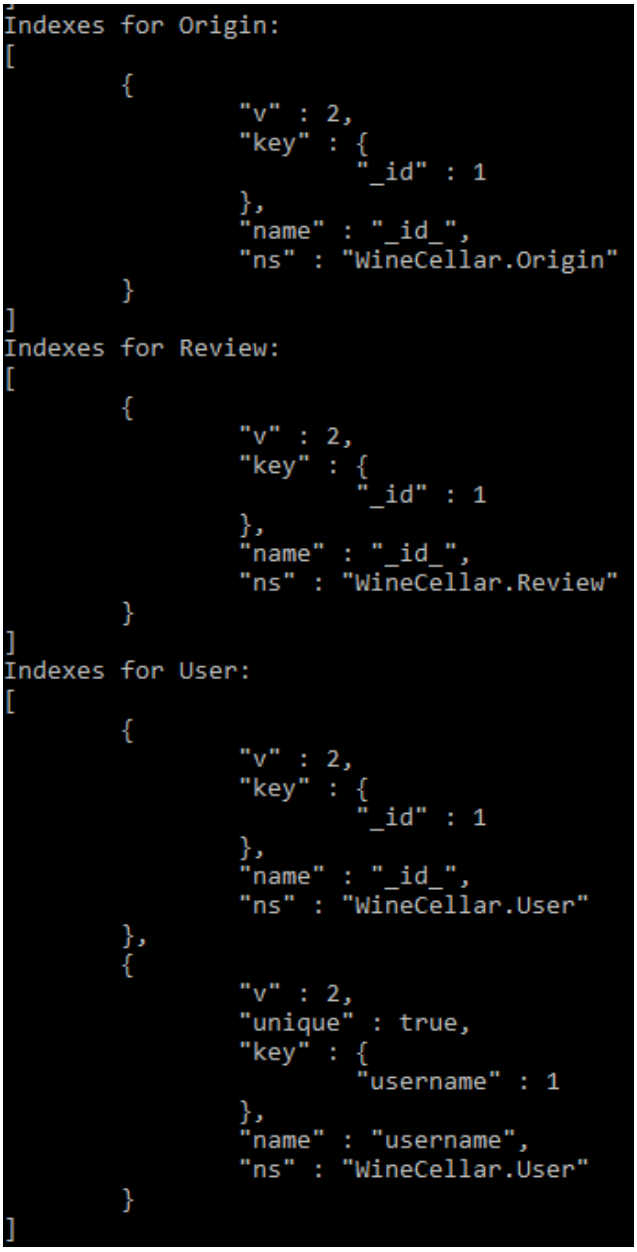
```
> db.getCollectionNames().forEach(function(collection) {
...     indexes = db[collection].getIndexes();
...     print("Indexes for " + collection + ":");
...     printjson(indexes);
... });
```

Slika 32 - Operacija kojom se izlistavaju svi indeksi u bazi

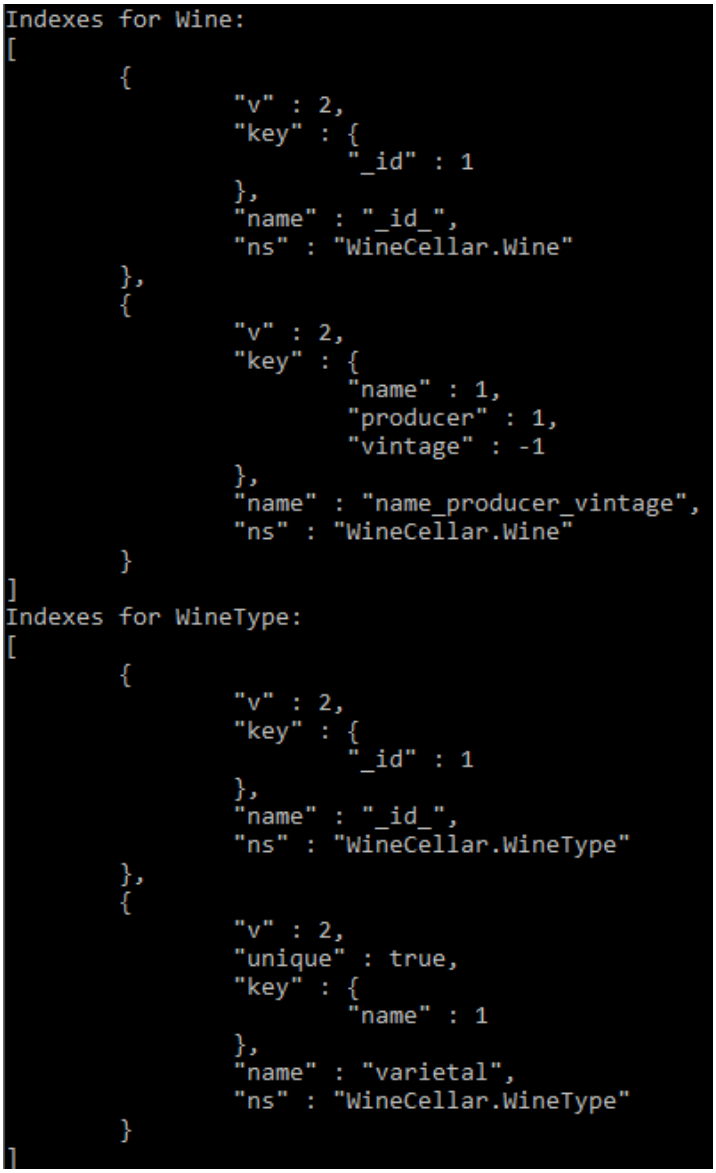
Rezultat je sledeći:



Slika 33 - Indeksi



Slika 34 - Indeksi



Slika 35 – Indeksi

Sledi primer pretraživanja kolekcije *Wine* korišćenjem polja *vintage* zadavanjem konkretne vrednosti. Retultat upita je jedan dokument prikazan na slici. Nakon toga je odrađen isti upit zadavanjem *explain* funkcije, koja prikazuje dodatnu statistiku, gde je važna stavka *executionTimeMills*. Bitno je obratiti pažnju na sledeći faktor, a to je da u ovom slučaju za pretragu nije korišćen indeks, kao i to da kolekcija sadrži samo tri dokumenta. Vreme za izvršenje ovog upita je 1ms.

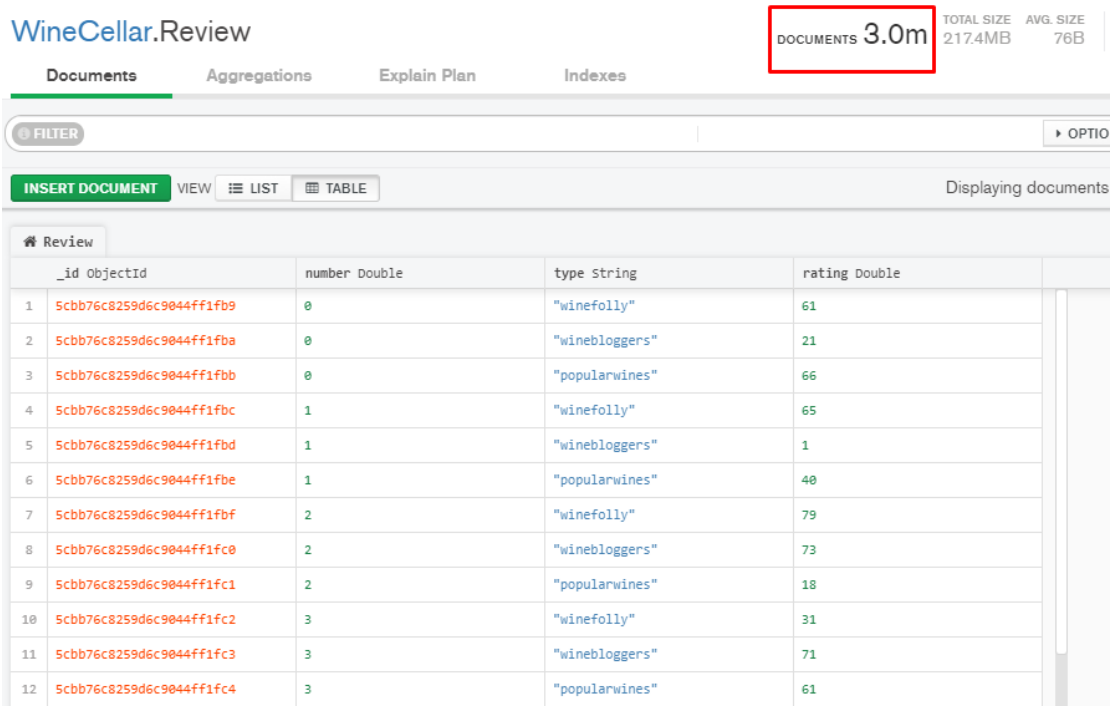
Slika 36 – Pretraživanje pomoću find funkcije

Slika 37 – Pretraživanje pomoću find funkcije uz dodatni prikaz statistika performansi

Obzirom da u prethodnom primeru postoje samo tri dokumenta, poređenja radi, u kolekciju *Review* biće uneto 3 miliona dokumenata. Kreirana je skripta kojom se popunjavaju dokumenti. [7]

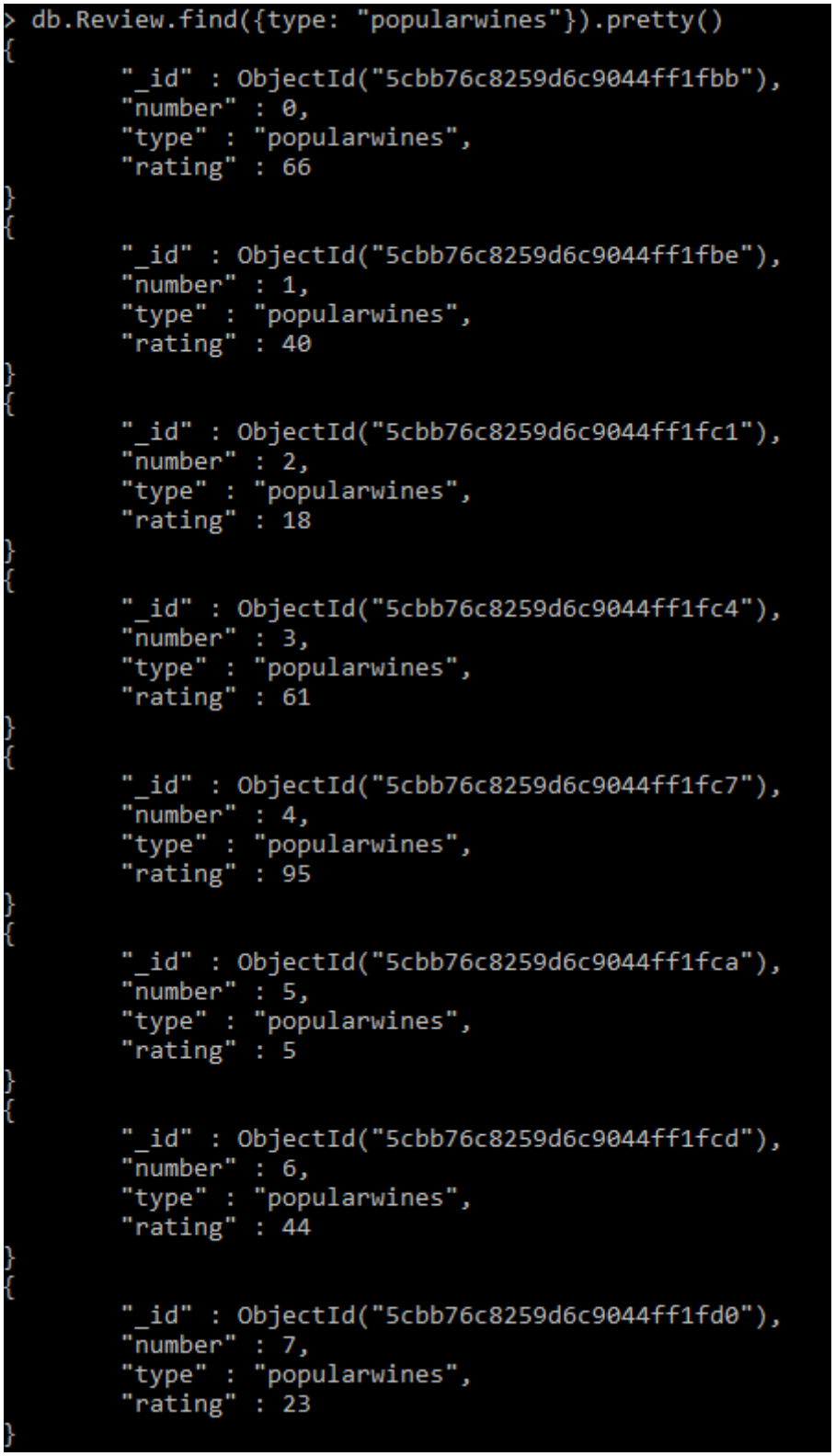
```
> var reviews = [];
> for (var i = 0; i < 1000000; i++) {
...    var types = ["winefolly", "winebloggers", "popularwines"];
...    for (var j = 0; j < 3; j++) {
...      reviews.push({
...        number : i,
...        type   : types[j],
...        rating   : Math.round(Math.random()*100)
...      });
...    }
... }
3000000
> db.Review.insert(reviews)
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 3000000,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
```

Slika 38 - Dodavanje se vrši izvršavanjem datog dela koda



	_id ObjectId	number Double	type String	rating Double
1	5cbb76c8259d6c9044ff1fb9	0	"winefolly"	61
2	5cbb76c8259d6c9044ff1fba	0	"winebloggers"	21
3	5cbb76c8259d6c9044ff1fbb	0	"popularwines"	66
4	5cbb76c8259d6c9044ff1fbc	1	"winefolly"	65
5	5cbb76c8259d6c9044ff1fbd	1	"winebloggers"	1
6	5cbb76c8259d6c9044ff1fbe	1	"popularwines"	40
7	5cbb76c8259d6c9044ff1fbf	2	"winefolly"	79
8	5cbb76c8259d6c9044ff1fc0	2	"winebloggers"	73
9	5cbb76c8259d6c9044ff1fc1	2	"popularwines"	18
10	5cbb76c8259d6c9044ff1fc2	3	"winefolly"	31
11	5cbb76c8259d6c9044ff1fc3	3	"winebloggers"	71
12	5cbb76c8259d6c9044ff1fc4	3	"popularwines"	61

Slika 39 - Izgled kolekcije Review nakon dodatih 3 miliona dokumenata



Slika 40 - Pretraga izvršavanjem upita pomoću find i pretty metoda

Na slici je prikazan samo deo rezultata, obzirom da postoji mnoštvo duplikata u kolekciji. Izvršavanjem naredbe *it* moguće je izlistati ostale dokumente iz rezultata.

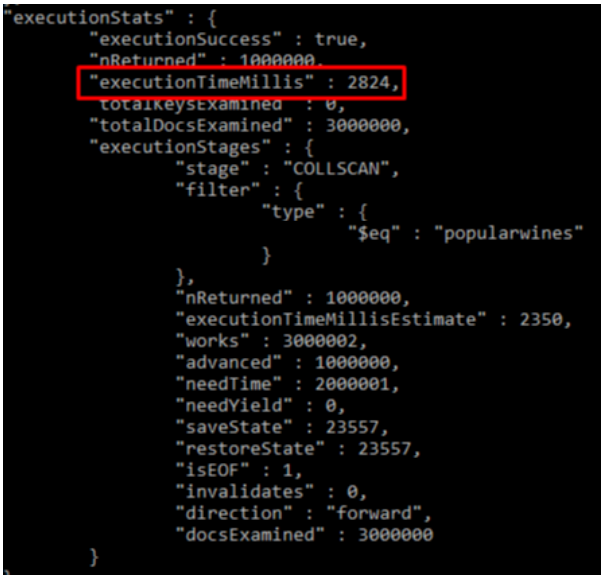
Slika 41 - Izvršavanje upita pomoću find i explain funkcija

Na sledećim slikama, poređenja radi, dati su rezultati izvršavanja istog upita. Ono što je uočljivo i bitno jeste da se vreme za obilazak 3 miliona dokumenata u kolekciji *Review* razlikuje prilikom svakog izvršenja istog upita.

Slika 42 - Vreme 1

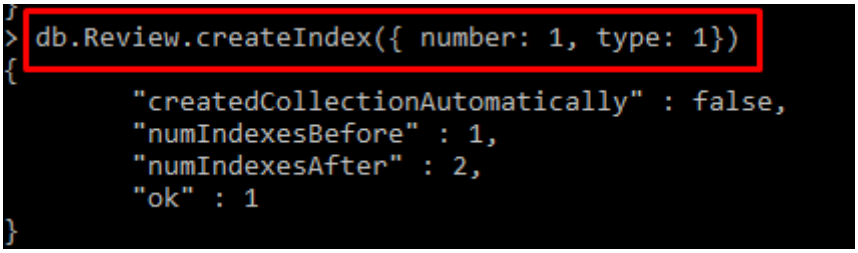
Slika 43 - Vreme 2

Slika 44 - Vreme 3

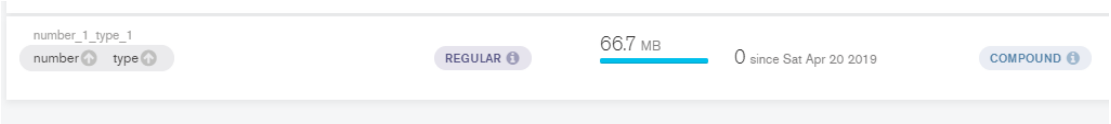


Slika 45 - Vreme 4

Zatim je kreiran složeni indeks (*compound*) koji grupiše polja *number* i *type*.



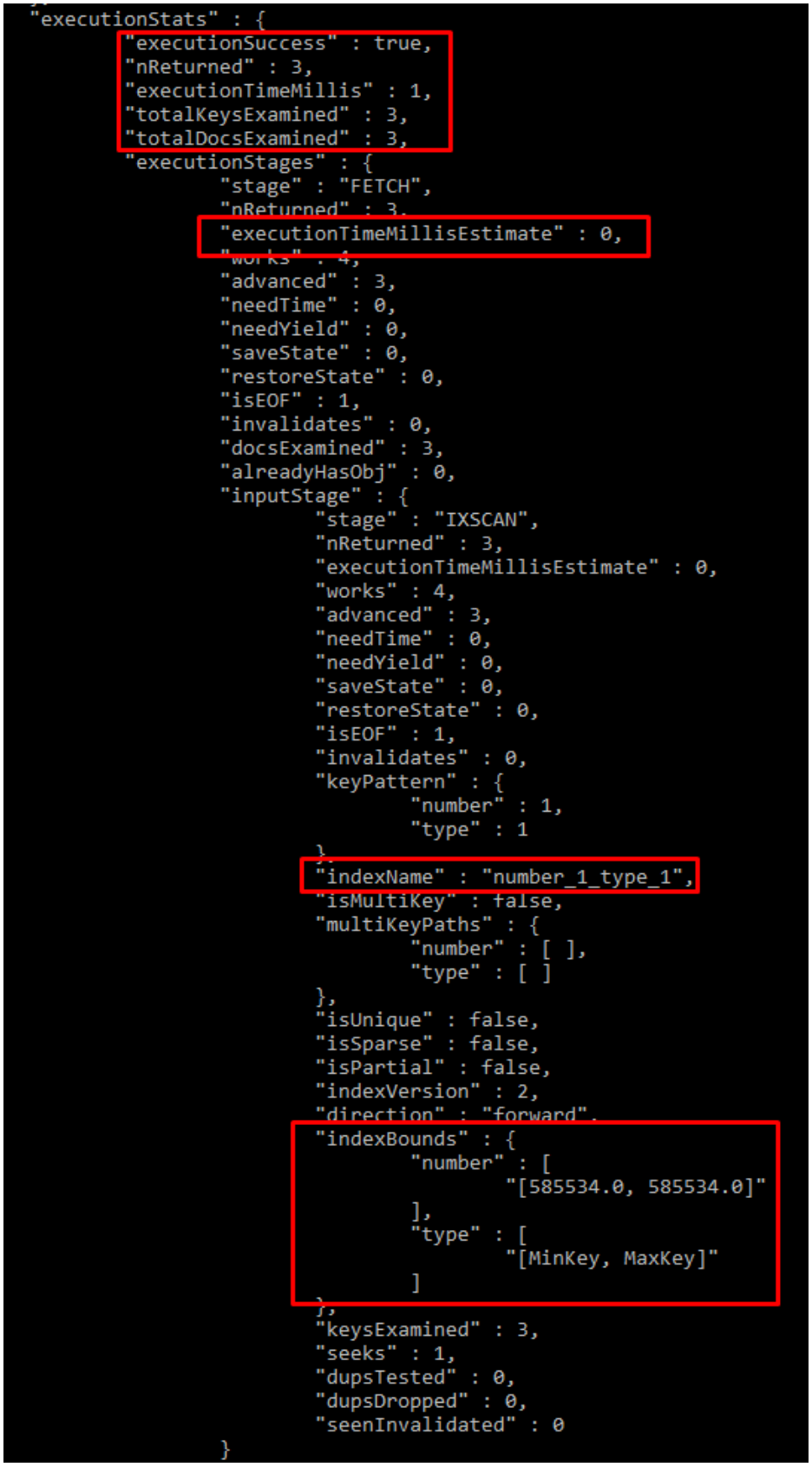
*Slika 46 - Kreiranje **compound** indeksa u kolekciji Review*



Slika 47 - Provera da li je indeks kreiran u kolekciji Review

Zatim slede izvršenje upita korišćenjem indeksa. Pretraga sa zadatim poljem *number* koje sadrži vrednost *585534*. Iz slika se može uočiti da je *executionTimeMillsEstimate* jednako nuli. U kontekstu, ne može biti 0 milisekundi, ali je približno nuli. Izvršavanjem istog upita par puta mogu da se dobiju slični rezultati koji su skoro ravni nuli. Bez obzira na sve to, suština je da se izvršavanje upita odvija mnogo brže korišćenjem indeksa, što znatno poboljšava performanse!

Slika 48 - Izvršavanje upita pretrage pomoću explain metode korišćenjem indeksa - prvi deo



Slika 49 - Izvršavanje upita pretrage pomoću explain metode korišćenjem indeksa - drugi deo

Literatura

[01] https://www.tutorialspoint.com/dbms/dbms_indexing.htm

[02] https://www.javatpoint.com/indexing-in-dbms

[03] https://www.simplilearn.com/indexing-and-aggregation-mongodb-tutorial-video

[04] https://severalnines.com/blog/how-optimize-performance-mongodb

[05] https://docs.mongodb.com/manual/storage/

[06] https://www.kenwalger.com/blog/nosql/mongodb/storing-documents-mongodb-database/

[07] https://itnext.io/indexing-and-mongodb-query-performance-a8a6a64c4308