UNIVERZITET U NIŠU

ELEKTRONSKI FAKULTET

KATEDRA ZA RAČUNARSTVO

Optimizacija korišćenja MongoDB-a

Studijsko-istraživački rad

Mentor:

Prof. dr Leonid Stoimenov

Student:

Natalija Mitić 1046

Sadržaj

1 Uvod

Baze podataka pružaju informacije sačuvane u hijerarhijskim i srodnim strukturama, što omogućava da se sadržaj izdvoji i lako uredi. Postoji veliki broj baza podataka koje se mogu odabrati za određeni sistem. Svaka od njih pripada određenom tipu skladišta i implementirana je s ciljem da optimizuje određene aspekte korišćenja. Zbog različitog načina skladištenja i organizovanja podataka, načina obrade upita, mogućnosti, performansi itd., ne postoji idealan izbor baze podataka koji je primenljiv za svaki sistem.

Međutim, svaka baza podataka se može dodatno optimizovati i na taj način se prilagoditi potrebama korisnika u pogledu poboljšanja performansi. Optimizacija baze podataka predstavlja strategiju smanjenja vremena odziva sistema baze podataka. Postoji niz koraka koji se preporučuju radi optimizacije, pa je zbog toga važno proučiti ih pre nego se počne s implementacijom rešenja.

Cilj ovog rada je da prikaže različite načine optimizacije MongoDB baze podataka. U okviru rada su navedeni načini optimizacije i slučajevi korišćenja. Dati su, takođe, primeri testiranja različitih načina optimizacije i prikazani su rezultati.

Nakon uvoda, u drugom poglavlju je dat opis strukture MongoDB baea podataka, kao i za koju namenu je optimizovana. U trećem poglavlju je predstavljena optimizacija MongoDB baze podataka. Navedeni su načini optimizacije i sprovedena su testiranja koja pokazuju koliko koji način doprinosi performansama. U četvrtum poglavlju je dat zaključak, a u petom je navedena korišćena literatura.

2 MongoDB

MongoDB je objektno orijentisana, dinamična i skalabilna NoSQL baza podataka. Zasnovana je na modelu skladišta baziranih na dokumentima.

Ovakav tip baza podataka je dizajniran za upravljanje podacima uskladištenim u dokumentima koji koriste različite standardne formate, kao što su XML ili JSON. Zbog raznolikosti tipova vrednosti polja u dokumentima i moćnih jezika upita, baze podataka bazirane na dokumentima imaju širok spektar primena i mogu se koristiti kao baze podataka opšte namene. Mogu se horizontalno skalirati, kako bi se prilagodile velikim količinama podataka. Najčešća primena je kod rada sa velikim količinama dokumenata koji se mogu sačuvati u strukturirane datoteke, poput tekstualnih dokumenata, e-maila, XML dokumenata ili CMS[1] i CRM[2] sistema.

Objekti podataka, kod MongoDB-a, se čuvaju kao zasebni dokumenti unutar kolekcije, umesto čuvanja u kolonama i redovima kao kod tradicionalne relacione baze podataka. Motivacija

[1] CMS(Content Management System) - sistem za dinamičko upravljanje digitalnim sadržajem.
[2] CRM (Customer relationship management) – pristup koji omogućava kompaniji da analizira svoje interakcije sa trenutnim, bivšim i potencijalnim kupcima.

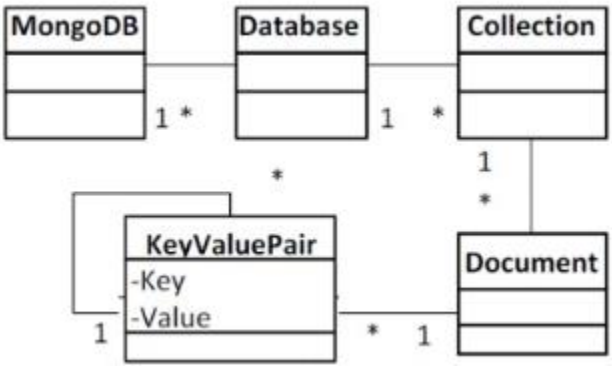
MongoDB-a je implementiranje skladišta podataka koje pruža visoke performanse, visoku dostupnost i automatsko skaliranje.

Da bi se odabrala baza podataka koja bi bila prikladnija za određeno poslovanje, važno je razumeti njene glavne karakteristike. Svaka baza podataka pruža različite mehanizme za čuvanje i preuzimanje podataka, što direktno utiče na performanse. Svaka baza podataka, takođe, ima mogućnost različitih načina optimizacije, što rezultuje različitim vremenima izvršenja operacija čitanja, pisanja i ažuriranja podataka.

MongoDB se sastoji od više kolekcija, a svaka kolekcija se sastoji od različitih vrsta objekata, što je ekvivalent tabelama kod relacionih baza podataka. Objekat se naziva dokumentom. MongoDB koristi BSON[3] dokumente za čuvanje podataka. Dokument se sastoji od strukture parova ključ / vrednost koji su osnovna jedinica podataka u MongoDB-u. Vrednost u dokumentu može biti jednog od sledećih tipova:

- Primitivni JSON tip (number, string, boolean)
- Primitivni BSON tip (datetime, ObjectID, UUID, regex)
- Niz
- Objekat sastavljen od parova ključ/vrednost
- Null

Struktura dokumenata je približnija načinu na koji se konstruišu klase i objekti u programskim jezicima, nego struktura tabela. Model MongoDB sistema je prikazan na slici 1.



Slika 1 – Model MongoDB sistema

Dokumenti ne moraju prethodno imati definisanu šemu. Umesto toga, polja se mogu kreirati u letu. Dinamička šema znači da dokumenti u istoj kolekciji ne moraju imati isti skup polja ili strukturu, a dokumenti kolekcije mogu sadržati različite tipove podataka. MongoDB podržava pretragu po vrednosti polja, upite koji se odnose na opseg i pretrage pomoću regularnih izraza.

U ovom radu je korišćenja verzija 4.4.1 MongoDB-a i MongoDB Compass, grafički interfejs za praćenje i izvršenje upita.

[3] BSON (Binary JSON) - binarno kodirana serializacija dokumenata sličnih JSON-u, koja omogućava njegovo brže parsiranje. JSON dokument je čitljiv tekst za predstavljanje struktuiranih podataka.

3 Optimizacija MongoDB baze podataka

3.1 Modeliranje podataka

Prvi korak u optimizaciji performansi je razumevanje obrazaca upita koje će aplikacija zahtevati, kako bi se u skladu s tim dizajnirao model podataka, kao i odgovarajući indeksi. Prilagođavanje modela podataka prema obrascima upita aplikacije uzrokuje efikasnije upite, povećava protok operacija upisivanja i ažuriranja i efikasnije raspoređuje radno opterećenje u klasteru. Optimalni dizajn šeme određuje se obrascima upita za svaku aplikaciju pojedinačno. Ne postoji jedinstveno optimalno rešenje koje se može uvek primenjivati. Glavna prednost JSON dokumenata je u tome što omogućuju fleksibilnost za modeliranje podataka na bilo koji način koji je potreban aplikaciji. Ugnježdavanje nizova i dokumenata čini dokumente vrlo moćnim u modeliranju složenih odnosa između podataka.

Kod MongoDB-a svaki zapis (dokument) sadrži i podatke i šemu. Svaki dokument može sadržati ugneždene dokumente, što znači da je ceo dokument stablo, a ne vektor.

Dokumenti na slikama 2 i 3 sadrže istu količinu informacija, ali drugi dokument je znatno manji. Veličina dokumenata je prikazana na slici 4. S obzirom na to da je šema uskladištena u svakom dokumentu, treba koristiti kraća imena za polja. Ovakav način imenovanja neće smanjiti samo veličinu dokumenata, već i veličinu upita. U mnogim slučajevima to može biti izuzetno bitan faktor kod poboljšanja performansi.

```
{
  "v1": {
    "count": { "offline_conversion:add_to_wishlist": 5 },
    "value": { "offline_conversion:add_to_wishlist": 50.0 }
  },
  "v7": {
    "count": { "offline_conversion:add_to_wishlist": 15 },
    "value": { "offline_conversion:add_to_wishlist": 300.0 }
  },
  "c1": {
    "count": { "offline_conversion:add_to_wishlist": 6 },
    "value": { "offline_conversion:add_to_wishlist": 50.0 }
  },
  "c7": {
    "count": { "offline_conversion:add_to_wishlist": 13 },
    "value": { "offline_conversion:add_to_wishlist": 250.0 }
  }
}
```

Slika 2 – Primer većeg dokumenta

```
{
  "offline_conversion:add_to_wishlist": {
    "count": {
      "v1": 5,
      "v7": 15,
      "c1": 6,
      "c7": 13
    },
    "value": {
      "v1": 50.0,
      "v7": 300.0,
      "c1": 50.0,
      "c7": 250.0
    }
  }
}
```


Slika 3 – Primer manjeg dokumenta

Collection Name	Documents	Avg. Document Size	Total Document Size
bigdoc	1	474.0 B	474.0 B
smalldoc	1	151.0 B	151.0 B

Slika 4 – Poređenje veličine dokumenata

3.2 Povezivanje podataka

U MongoDB-u postoje dve opcije za modeliranje veza između podataka - ugrađivanje podataka i referenciranje dokumenata. Ugrađivanje je postupak spajanja dva ili više entiteta u jedan entitet pomoću hijerarhije, što predstavlja denormalizaciju podataka. Referenciranje je slično kreiranju stranog ključa u RDBMS-u koji služi kao pokazivač iz jednog entiteta (kolekcije) na drugi entitet (kolekciju) i predstavlja normalizaciju podataka. Kao i većina stvari u modeliranju, ne postoji jedinstveno rešenje za donošenje odluke o tipu povezivanja koji će biti primenjen. Pristup načinu modeliranja je specifičan za kontekst, jer zavisi od toga kako aplikacija komunicira sa podacima. Postoji pet heuristika koje su predložene za odlučivanje o ugradnji ili referenciranju: [4]

1. Podaci različitih entiteta koji se često pretražuju istovremeno mogu se ugraditi u jedan dokument.
2. Entiteti koji se smatraju zavisnim entitetima mogu se ugraditi u jedan entitet.
3. Ako postoji veza 1:1 između dva entiteta, treba ugraditi jedan od entiteta u drugi entitet.
4. Entiteti koji imaju sličnu brzinu menjanja(umetanja, ažuriranja i brisanja) mogu biti ugrađeni zajedno.
5. Entiteti koji nisu ključni entiteti, ali imaju veze sa ključnim entitetima, mogu se referencirati.

Glavni razlog za ugrađivanje dokumenata je poboljšanje performansi čitanja koji je povezan sa samom prirodom načina na koji su računarski rotirajući diskovi izrađeni. Kada se pretražuje određeni zapis, potrebno je izvesno vreme da ga disk locira (velika latencija), ali nakon toga, pristup bilo kom dodatnom bajtu se dešava brzo (velika propusnost). Dokument je kontinualno uskladišten u memoriji. Zbog toga, kolociranje srodnih informacija ima smisla, jer se mogu preuzeti istovremeno. Drugi aspekt je taj što se smanjuje broj pristupa bazi podataka koji bi bio potreban za pretraživanje odvojenih kolekcija. Takođe, ugrađivanje ima uticaj i na operacije ažuriranja koje su u tom slučaju atomične.

S druge strane, dokument koji se često čita, ali bez potrebe za čitanjem i povezanih podataka, je dobar kandidat za referenciranje. Operacije agregacije (aggregation pipeline) za izvršenje složenih upita, kao što je uključivanje samo određenih polja u rezultat, smanjuju performanse. Takođe, ako se povezani dokument često ažurira, trebalo bi koristiti referenciranje. Kada se određeno polje ažurira, zadatak je da se pronađu sve instance koje treba ažurirati. To rezultuje sporom obradom upita, posebno ako su dokumenti ugnježđeni. S obzirom na maksimalnu veličinu dokumenta od 16 MB, treba referencirati one dokumente ili nizove koji imaju tendenciju rasta.

3.2.1 Tipovi veza između podataka

Važan faktor kod određivanja optimalnog modela podataka je kardinalnost veze između podataka. Na osnovu broja entiteta jedne vrste koji su povezani sa određenim brojem entiteta druge vrste, zavisi koji tip povezivanja predstavlja optimalnije rešenje.

1:1 veza

Veza „jedan prema jedan“ je vrsta kardinalnosti koja opisuje odnos između dva entiteta, gde je jedan zapis iz entiteta A povezan sa jednim zapisom u entitetu B. Može se modelirati na dva načina: ugradnjom u vidu poddokumenta (slika 5) ili referenciranjem sa dokumentom u zasebnoj kolekciji (slike 6 i 7). Prilikom referenciranja se ne primenjuju ograničenja stranog ključa, tako da veza postoji samo na nivou aplikacije. Vrsta povezivanja zavisi od toga kako aplikacija pristupa podacima, koliko često, kao i od životnog ciklusa skupa podataka (npr. ako je entitet A izbrisan, da li entitet B i dalje mora da postoji?).

Ako objektu B treba pristupiti samostalno, odnosno izvan konteksta roditeljskog objekta A, tada treba koristiti referenciranje, u suprotnom treba koristiti ugrađivanje. Ugrađivanje pruža bolje performanse za operacije čitanja, zbog mogućnosti traženja i preuzimanja povezanih podataka u jednoj internoj operaciji baze podataka, umesto traženja dokumenata uskladištenih u različitim kolekcijama. Ugrađeni modeli podataka, kao što je prethodno spomenuto, omogućavaju ažuriranje povezanih podataka u jednoj atomskoj operaciji upisivanja, jer je upisivanje pojedinačnih dokumenata transakciono.

```
{
  name: "Peter Wilkinson",
  age: 27,
  address: {
    street: "100 some road",
    city: "Nevermore"
  }
}
```

Slika 5 – Ugnježđeni dokument

```
{
  _id: 1,
  name: "Peter Wilkinson",
  age: 27
}
```

Slika 6 – Referencirani dokument

```
{
  user_id: 1,
  street: "100 some road",
  city: "Nevermore"
}
```

Slika 7 – Referencirajući dokument

1:N veza

Veza „jedan prema više" predstavlja odnos između dva entiteta, pri čemu jedan entitet može imati jednu ili više veza sa drugim, dok drugi entitet može imati samo jednu vezu sa prvim entitetom. Poput veze 1:1, veza 1:N se, takođe, može modelirati korišćenjem ugrađivanja ili referenciranja. Metod modeliranja zavisi od veličine dokumenta i poddokumenata. Maksimalna veličina svakog dokumenta je 16MB. Ako se povećava veličina dokumenta, potrebno je alocirati još memorije, a takođe indeksi moraju biti ažurirani, što utiče na performanse upisa. U slučaju velikih dokumenata treba koristiti referenciranje. Međutim, kompromis je da će dva upita morati da budu izvršena da bi se dobili detalji o povezanom entitetu, tako da to utiče na performanse čitanja.

Prilikom dizajniranja MongoDB šeme, uticaj na metod ima kardinalnost veze. 1:N veze se mogu podeliti u tri grupe: [5]

- jedan prema nekoliko
- jedan prema više
- jedan prema puno

U zavisnosti od kardinalnosti, koristi se različit format za modeliranje veza. Nizovi ne bi trebalo da rastu beskonačno. Neka je dokument A povezan sa dokumentom B. Ako je na B strani manje od par stotina kratkih dokumenata, dokumenti se mogu ugraditi. Ako je na B strani više od par stotina dokumenata, trebalo bi izvršiti referenciranje tako što se kreira niz referenci ID-eva koji ukazuju na dokumente B tipa. Ako na B strani postoji više od nekoliko hiljada dokumenata, treba koristiti referencu na roditeljski dokument A u B dokumentima.

Modeliranje veze jedan prema nekoliko

Na slici 8 je dat primer za vezu jedan prema nekoliko, gde je za adrese osoba korišćen princip ugradnje. Ako aplikacija zahteva potpune podatke o osobi, ugradnja predstavlja optimalnije rešenje.

```
> db.person.findOne()
{
  name: 'Kate Monster',
  ssn: '123-456-7890',
  addresses : [
     { street: '123 Sesame St', city: 'Anytown', cc: 'USA' },
     { street: '123 Avenue Q', city: 'New York', cc: 'USA' }
  ]
}
```

Slika 8 - Modeliranje veze jedan prema nekoliko pomoću ugnježdavanja

Prednost ovog dizajna je što se ne mora izvršiti zaseban upit da za dobijanje ugrađenih adresa. S druge strane, nedostatak je taj što ne postoji način da se pristupi ugrađenim detaljima kao samostalnim entitetima.

Modeliranje veze jedan prema više

Kao primer za vezu „jedan prema više“ dati su delovi proizvoda u sistemu za naručivanje rezervnih delova. Svaki proizvod može imati do nekoliko stotina rezervnih delova, ali nikada više od nekoliko hiljada. U ovom slučaju dobra praksa je korišćenje referenciranja. ID-evi delova su implementirani kao niz u dokumentu proizvoda (slika 10). Svaki deo ima svoj dokument, što je prikazano na slici 9.

```
> db.parts.findOne()
{
    _id : ObjectID('AAAA'),
    partno : '123-aff-456',
    name : '#4 grommet',
    qty: 94,
    cost: 0.94,
    price: 3.99
```

Slika 9 – Veza jedan prema više – referencirani dokument

```
> db.products.findOne()
{
    name : 'left-handed smoke shifter',
    manufacturer : 'Acme Corp',
    catalog_number: 1234,
    parts : [     // array of references to Part documents
        ObjectID('AAAA'),    // reference to the #4 grommet above
        ObjectID('F17C'),    // reference to a different Part
        ObjectID('D2AA'),
        // etc
    ]
```

Slika 10 – Veza jedan prema više – referencirajući dokument

Modeliranjem podataka korišćenjem referenci, svaki deo je samostalni dokument, pa ih je lako pretraživati i samostalno ažurirati. S druge strane, za pretraživanje delova određenog proizvoda, potrebno je izvršiti dva upita.

Iako je na ovaj način modelirana veza 1:N, šema omogućava da delovi postoje pojedinačno. Delovi, u tom slučaju, mogu biti korišćeni od strane više proizvoda, pa je dobijena veza N:M, bez potrebe za promenom šeme.

Modeliranje veze jedan prema puno

Kao primer za vezu „jedan prema puno“ dat je sistem evidentiranja događaja koji prikuplja poruke logova za različite mašine. Bilo koji host može da generiše dovoljno poruka da prekorači maksimalnu veličinu dokumenta od 16 MB. U ovom slučaju je kreiran dokument roditelj koji se

odnosi na hosta (slika 11), a za svaki log zapis se kreira poseban dokument (slika 12) koji sadrži referencu na roditeljski dokument (ID hosta).

```
> db.hosts.findOne()
{
    _id : ObjectID('AAAB'),
    name : 'goofy.example.com',
    ipaddr : '127.66.66.66'
}
```

Slika 11 – Veza jedan prema puno – roditelj dookument

```
>db.logmsg.findOne()
{
    time : ISODate("2014-03-28T09:42:41.382Z"),
    message : 'cpu is on fire!',
    host: ObjectID('AAAB')       // Reference to the Host document
}
```

Slika 12 – Veza jedan prema puno – dokument koji referencira roditelja

N:M veza

Veza „više prema više" predstavlja vezu između dva entiteta A i B, pri čemu obe strane mogu imati jednu ili više veza sa drugom. U relacionim bazama podataka ovi slučajevi su modelirani tabelom spojeva, međutim u MongoDB-u se koristi dvosmerno ugrađivanje.

Primer dvosmernog ugrađivanja je dat na slikama 13 i 14, gde autor ima niz ID-eva knjiga i knjiga ima niz ID-eva autora. Prilikom izvršenja pretraživanja, potrebno je obaviti dva upita (u oba smera). Prvo je potrebno pronaći knjigu, a zatim izvršiti drugi upit za pronalaženje autora ili obrnuto. Primer upita je dat na slici 15.

```
{
  _id: 1,
  name: "Peter Standford",
  books: [1, 2]
}
{
  _id: 2,
  name: "Georg Peterson",
  books: [2]
}
```

Slika 13 – Dokument o autorima koji referencira knjige – dvostruko referenciranje

```
{
  _id: 1,
  title: "A tale of two people",
  categories: ["drama"],
  authors: [1, 2]
}
{
  _id: 2,
  title: "A tale of two space ships",
  categories: ["scifi"],
  authors: [1]
}
```

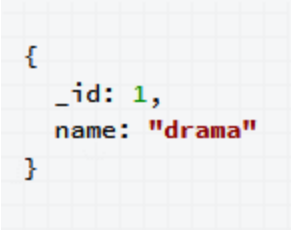
Slika 14 – Dokument o knjigama koji referencira autore – dvostruko referenciranje

```
var book = booksCollection.findOne({title: "A tale of two space ships"});
var authors = authorsCollection.find({_id: {$in: book.authors}}).toArray();
```

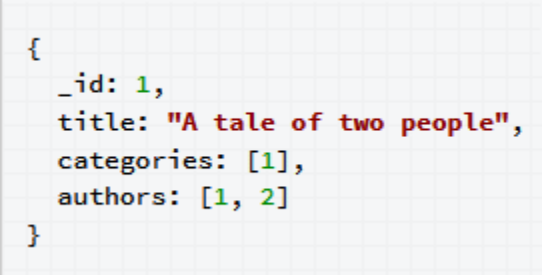
Slika 15 – Upiti koji izvlače autore određene knjige

Kod dvosmernog ugrađivanja složenost proizlazi iz uspostavljanja ravnomerne ravnoteže između dva tipa entiteta, jer se prag od 16MB može prevazići. U ovim slučajevima se preporučuje jednosmerno ugrađivanje.

Strategija jednosmernog ugrađivanja optimizuje performanse čitanja N:M veze ugrađivanjem referenci u jednu stranu veze. Za primer je uzet odnos kategorija i knjiga. Nekoliko knjiga spada u nekoliko kategorija, a svakoj kategoriji pripada više knjiga. Kategorija je izdvojena u zaseban dokument, kao na slici 16. Svaka knjiga sada ima niz ID-eva kategorija.



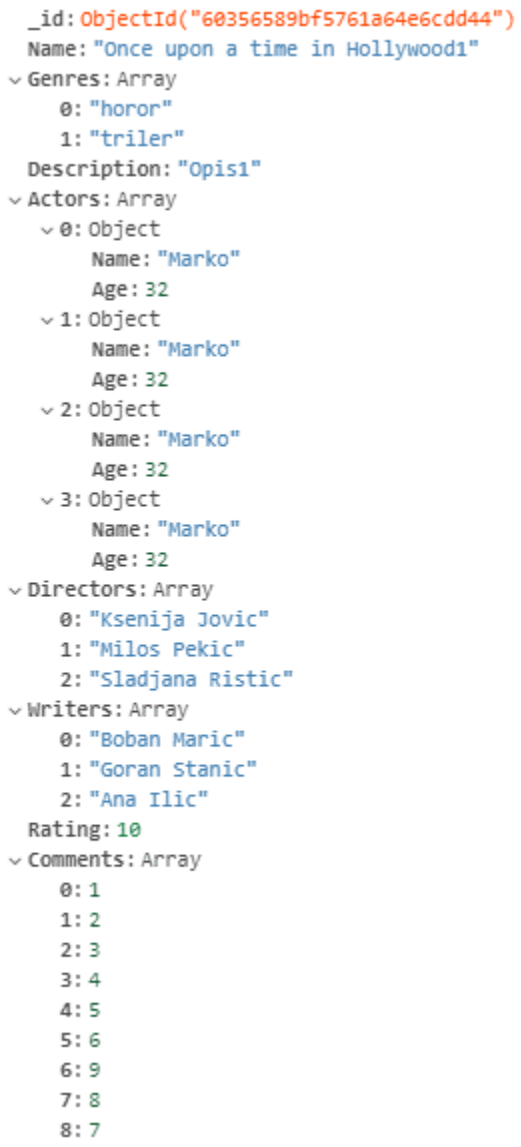
Slika 16 – Izdvojen dokument kategorije – jednosmerno ugrađivanje



Slika 17 – Dokument koji referencira kategorije – jednosmerno ugrađivanje

3.2.2 Testiranje ugradnje i referenciranja dokumenata

Za testiranje performansi ugradnje i referenciranja uzet je primer dokumenta, prikazanog na slici 18. Kolekcija sadrži 40 000 dokumenata. Komentari su skladešteni u posebnoj kolekciji i postoji 101 000 dokumenata o komentarima.



Slika 18 – Dokument o filmovima

Za glumce je korišćena tehnika ugradnje, a za komentare tehnika referenciranja (navedeni su ID-evi komentara u dokumentima o filmovima). Vršen je upit na osnovu koga je potrebo izdvojiti dokumente sa određenim nazivom. Ukoliko je potrebno izdvojiti glumce, to se može izvršiti pomoću istog upita, jer su podaci o njima u istom dokumentu. U skupu od 40 000 dokumenata, prosečno vreme izvršenja iznosilo je 15,3 ms, ako se izdvajaju celi dokumenti, odnosno 19,8 ms ako se koristi projekcija za izdvajanje samo podataka o glumcu (Projekcije su opisane u poglavlju 3.4.2 - Projekcija). Ukoliko je potrebno naći podatke o komentarima za određeni film, to zahteva izvršenje dva upita: prvi izdvaja film, a drugi komentare, koji se nalaze u posebnoj kolekciji. Za izdvajanje samo ID-eva komentara je bilo poterbno 19,6 ms, dok je za izdvajanje samih komentara bilo potrebno prosečno 11,6 ms uz korišćenje indeksa nad poljem *_id*.

3.2.3 Smanjenje količine potrebne memorije za smeštanje podataka na disku

Uklanjanje nepotrebnih podataka

Neki podaci se lako mogu ukloniti iz baze podataka kao što je brisanje celih kolekcija. Međutim, uklanjanje nepotrebnih podataka može biti prilično teško za sprovođenje. Takvi podaci su, na primer, suvišni podaci koji su prisutni u svakom dokumentu. Stoga, svaki dokument treba ažurirati. Takvo ažuriranje može potrajati duži vremenski period. Takođe, ukoliko postoje replike, ažuriranje će morati da se izvrši za svaku repliku posebno. Kod brisanja dokumenata, važnu ulogu imaju TTL indeksi (opisani u poglavlju 3.3.1 – TTL indeks) koji služe za automatsko uklanjanje dokumenata iz kolekcije nakon određenog vremena.

Izvođenje podaka

Ukoliko se podaci mogu izvesti agregacijom drugih podataka, sačuvanih u bazi podataka, nije potrebno skladištiti ih. Neki podaci mogu zauzeti dosta prostora, pa se izvođenjem štedi memorija. Količina prostora, koja se štedi, dakle, može biti prilično značajna.

3.3 Indeksi

Indeksi su posebne strukture podataka koje čuvaju mali deo skupa podataka kolekcije u obliku koji je lako pretraživati. Indeks čuva vrednost određenog polja ili skupa polja, poređanih po vrednosti tog polja za svaki dokument. Uređenje indeksa omogućava efikasno izvršavanje upita koji sadrže pretraživanje po jednakosti ili se odnose na pretraživanje opsega vrednosti. Takođe, indeksi se koriste za ubrzanje sortiranja rezultata, zbog sortiranja u samom indeksu.

Bez indeksa, vrši se skeniranje cele kolekcije za svaki upit koji se izvrši. Skeniranje kolekcije predstavlja proveru svakog dokumenta u kolekciji kako bi se utvrdilo da li je zadovoljava dati upit. Kada indeks podržava upit, MongoDB skenira samo indeks, a ne i dokumente, stoga dolazi do poboljšanja performansi.

MongoDB indeksi koriste strukturu B-stabla koja sadrže deo skupa podataka. Indeksi su definisani na nivou kolekcije i čuvaju vrednosti određenog polja ili skupa polja kolekcije, pri čemu su vrednosti sortirane. Bilo koja vrsta polja (i ugnježdenog polja) u dokumentu se može indeksirati.

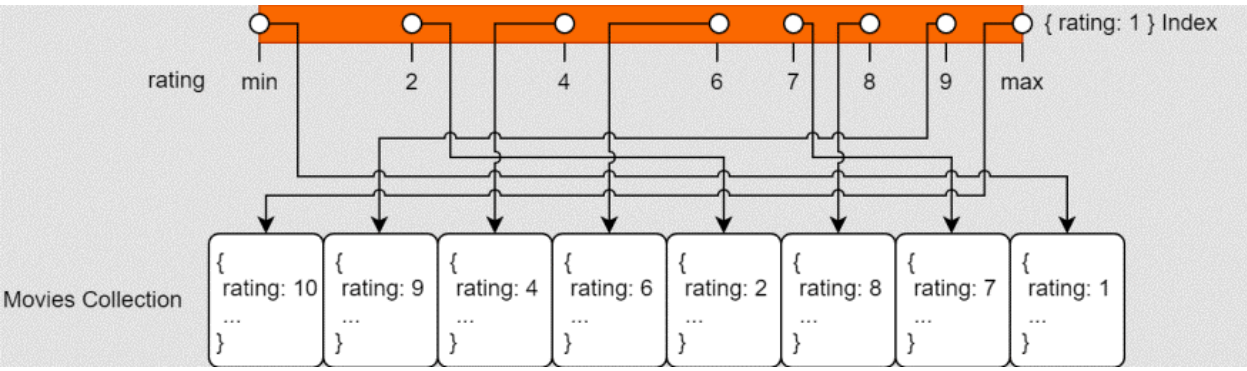
Iako kreiranje indeksa povećava performanse izvršenja upita, sa svakim dodatnim indeksom, smanjuje se brzina upisa za kolekciju. Smanjenje brzine se dešava zbog strukture B-stabla. Naime, prilikom ažuriranja ili brisanja dokumenta, možda će biti potrebno uravnotežiti jedno ili više B-stabala. Preporuka je da ne postoji previše nepotrebnih indeksa u kolekciji, jer bi došlo do prekomernog gubitka performansi upisa, ažuriranja i brisanja.

Prilikom kreiranje kolekcije, automatski se kreira i jednistveni indeks nad poljem *_id*. Ovaj indeks sprečava dodavanje dva dokumenta sa istom vrednošću za polje *_id* i ne može se obrisati.

MongoDB podržava šest vrsta indeksa. Svaki od njih je namenjen za određenu vrstu polja ili upita.

Indeks pojedinačnog polja (Single field index)

Indeks pojedinačnog polja je indeks dodeljen jednom određenom polju dokumenta. On predstavlja sortiranu strukturu ovog polja koja ukazuje na dokumente u kolekciji. U primeru na slici 19 je kreiran rastući indeks nad poljem koje označava ocenu {rating:1}[4]. Ovaj indeks stvara strukturu, sortiranu prema polju za *rating*, gde svaka stavka u ovoj strukturi pokazuje na povezani dokument u kolekciji. Indeks pojedinačnog polja se, takođe, može kreirati nad celom ugrađenom dokumentu. Da bi se koristio ovaj indeks, upit morati da sadrži sva polja poddokumenta u istom redosledu kao što je navedeno u indeksu i u dokumentima u kolekciji.



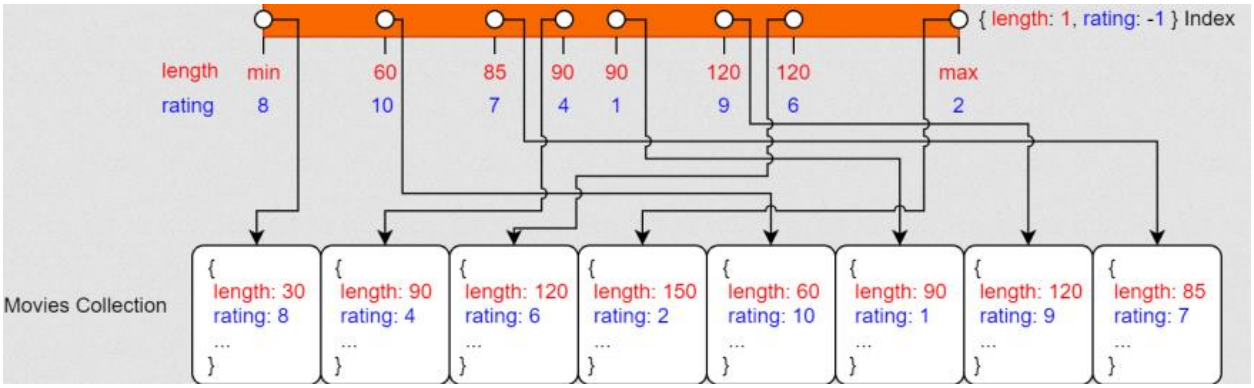
Slika 19 – Indeks pojedinačnog polja

Složeni indeks (Compound index)

Složeni indeks je indeks koji sadrži više polja dokumenta istovremeno (do 32 polja). Indeks podržava upite kod kojih se sva navedena polja pojavljuju i u indeksu. Prilikom kreiranja složenog indeksa, svako polje može imati drugačiji redosled sortiranja. Prvo polje određuje početni redosled dokumenata, drugo polje određuje redosled svih dokumenata koji nemaju jedinstvenu vrednost prvog polja (vrednost prvog polja pojavljuje se u više dokumenata). Sledeća polja indeksa čine isto za naredne „slojeve sortiranja“. Na primer, ako je kreiran indeks nad poljima dužine i ocene filmova {length: 1, rating: -1} (slika 20), indeks će biti poređan od najkraćeg filma do najdužeg. Ako nekoliko filmova ima istu dužinu, oni će biti poređani prema svojoj oceni, od najveće do najniže.

[4] Za sortiranje u rastućem redosledu navodi se 1, dok se za sortiranje u opadajućem redosledu navodi -1.

Pored upita koji se poklapaju po svim poljima sa indeksom, složeni indeksi podržavaju i upite koji se poklapaju po prefiksu indeksa. Prefiksi indeksa su početni podskupovi indeksiranih polja. U datom primeru složeni indeks se može koristiti za upite nad poljem *length*, kao prefiksom indeksa.



Slika 20 – Složeni indeks

Indeks sa više ključeva (Multikey index)

Indeks sa više ključeva se koristi za indeksiranje nizova. Ukoliko se indeksira polje čija je vrednost niz, MongoDB kreira poseban ulaz (eng. data entry) za svaki element niza. Indeksi sa više ključeva se mogu kreirati nad nizovima koji sadrže skalarne vrednosti (brojevi, stringovi itd.) ili nad nizovima koji sadrže ugneždene dokumente. Ukoliko niz sadrži ugnježdene dokumente kao vrednosti, indeks se, takođe, može kreirati i nad nekim od polja ugnježdenih dokumenata. Indeksi sa više ključeva podržavaju upite koji selektuju dokumente koji sadrže nizove poklapanjem jednog ili više elemenata niza. Mogući su složeni indeksi sa više ključeva, međutim, samo jedno od polja može biti niz (u određenom dokumentu). Ako je indeks već kreiran, novi dokumenti koji se dodaju, a ne zadovoljavaju ograničenja indeksa, (2 ili više polja indeksa su nizovi) neće biti dodati u kolekciju.

Tekstualni indeks (Text index)

Tekstualni indeksi podržavaju upite za pretraživanje teksta u stringovima. Mogu sadržati polja tipa string ili niz stringova. Svaka kolekcija sme imati samo jedan tekstualni indeks, ali ovaj indeks može biti kreiran nad više polja. Može se kreirati i složeni tekstualni indeks nad tekstualnim poljem i drugim poljima.

Tekstualni indeksi mogu imati dodeljenu težinu. Težine se mogu dodeliti svakom polju u tekstualnom indeksu. Na taj način im se dodeljuje prioritet nad ostalim poljima prilikom pretraživanja teksta. Za svako indeksirano polje u dokumentu, MongoDB množi broj poklapanja sa težinom i sumira rezultate. Korišćenjem sume, MongoDB računa *skor* dokumenta.

Geoprostorni indeks (Geospatial index)

Namena geoprostornog indeksa je efikasno izvršenje upita nad podacima koji predstavljaju geoprostorne koordinate. Postoje dva tipa geoprostornih indeksa: *2d* i *2dsphere*. *2d* indeksi podržavaju upite koji izračunavaju geometrije na dvodimenzionalnoj ravni i koriste planarnu geometriju prilikom vraćanja rezultata. *2dsphere* indeksi podržavaju upite koji izračunavaju geometrije na zemljinoj sferi i koriste sfernu geometriju za vraćanje rezultata.

Heširani indeks (Hashed index)

Da bi podržao raščlanjivanje (eng. sharding) zasnovano na hešu, MongoDB koristi heširani tip indeksa, koji indeksira heš vrednosti polja. Heširani indeksi koriste heš funkciju da izračunaju heš vrednost indeksiranog polja. Ovi indeksi imaju slučajniju distribuciju vrednosti duž svog opsega, ali podržavaju samo traženje po jednakosti i ne mogu podržati upite zasnovane na opsegu. Ukoliko je vrednost ugnježđeni dokument, heš vrednost se računa za ceo dokument. Može se kreirati heširani indeks i indeks pojedinačnog polja nad istim poljem. MongoDB onda koristi skalarni indeks za pretragu po opsegu.

3.3.1 Svojstva indeksa

Jedinstveni indeks (Unique index)

Svojstvo jedinstvenosti indeksa se odnosi na to da MongoDB odbija duple vrednosti za indeksirano polje. Podrazumevani indeks nad *_id* poljem je jedinstven. MongoDB ne može da kreira jedinstveni indeks nad određenim poljima, ukoliko kolekcija već sadrži podatke koji ne zadovoljavaju ograničenje jedinstvenosti.

Parcijalni indeks (Partial index)

Parcijalni indeksi indeksiraju samo dokumente u kolekciji koji ispunjavaju navedeni izraz filtera. Indeksiranjem podskupa dokumenata u kolekciji, parcijalni indeksi imaju niže zahteve za skladištenjem i smanjene troškove performansi za kreiranje i održavanje indeksa.

Parcijalni indeks se može specificirati za bilo koju vrstu indeksa. Nad *_id* poljem se ne može kreirati parcijalni indeks i ne može se kreirati više indeksa koji se razlikuju samo po izrazu za filtriranje. Ukoliko je parcijalni indeks i jednistveni indeks, onda se ograničenje jedinstvenosti odnosi samo na dokumente koji zadovoljavaju uslov filtriranja.

Retki indeks (Sparse index)

Osobina retkih indeksa osigurava da indeks sadrži samo unose za dokumente koji imaju indeksirano polje. Indeks preskače dokumente koji nemaju indeksirano polje. Ukoliko korišćenje retkih indeksa rezultuje nekompletnim rezultatima upita, MongoDB neće koristiti taj indeks ukoliko to nije eksplicitno navedeno.

Retki indeks se može kombinovati sa jedinstvenim indeksom. Geoprostorni i tekstualni indeksi su uvek retki. Složeni retki indeksi koji sadrže samo pojedinačna polja će indeksirati dokument ukoliko sadrži barem jedno od navedenih polja. Ukoliko se radi o geoprostornim ili tekstualnim složenim indeksima, dokument se indeksira samo ako sadrži geoprostorno ili tekstualno polje.

TTL indeks

TTL indeksi su posebni indeksi koje MongoDB koristi za automatsko uklanjanje dokumenata iz kolekcije nakon određenog vremena. Ovi indeksi su namenjeni za određene vrste informacija kao što su mašinski generisani podaci o događajima, evidencije i informacije o sesijama koje u bazi podataka treba da postoje samo određeno vreme.

Skriveni indeks (Hidden index)

Skriveni indeksi nisu vidljivi planeru upita i ne mogu se koristiti za izvršavanje upita. Namena sakrivanje indeksa od planera je da korisnici mogu proceniti potencijalni uticaj uklanjanja indeksa bez stvarnog brisanja indeksa. Ako je uticaj negativan, korisnik može otkriti indeks, umesto da ponovo kreira obrisani indeks. Budući da se vrši održavanje indeksa dok su sakriveni, indeksi su odmah dostupni za upotrebu nakon što budu otkriveni. Svi indeksi se mogu sakriti, osim indeksa nad *_id* poljem.

3.4 Optimizacija upita

MongoDB, kao i većina drugih baza podataka, ima planer upita i optimizator upita. Ove dve komponente zajedno preuzimaju upite od korisničkih programa i čine ih što efikasnijim i bržim. Planer upita uzima optimizovan upit i određuje kako da preuzme podatke iz mehanizma za skladištenje baze podataka (eng. Storage engine). On određuje pojedinosti o tome koji koraci će biti preduzeti za preuzimanje podataka. To može uključivati korake poput transformacije, preuzimanja ili sortiranja podataka. Optimizator upita, pre nego što planer izvrši upit, traži načine za poboljšanje upita. Postavlja pitanja poput: „Može li se koristiti indeks?, Koji indeks se može koristiti?, Koji je plan najbolji?“. Optimizatori zasnovani na troškovima, najčešće, procenjuju onoliko planova koliko je vremena određeno za dati upit, a zatim biraju najjeftiniji od njih.

Metoda optimizacije upita kod MongoDB-a je relativno jednostavna. Kada se obrađuje zahtev za upit, sistem bira plan najbržeg izvršenja, a funkcije *skip* i *limit* se ne uzimaju u obzir prilikom izvršavanja plana upita. MongoDB optimizator koristi empirijsku metodu: ako nema keširanog plana, kreiraju se svi mogući planovi izvršenja, na osnovu dostupnih indeksa. MongoDB će proći kroz upit onoliko puta koliko je planova generisano i oceniće ih. Zatim će odabrati onaj s najboljim performansama. Nakon toga, plan se kešira. Plan koji je keširan koristiće se i za sledeće upite istog oblika, umesto ponovnog proveravanja planova kandidata. Keširani plan se evaluira i ukoliko u nekom trenutku njegove performanse padnu ispod određenog praga, on će biti izbačen iz keša i ponovo će se proći kroz fazu testiranja planova kandidata.

Prilikom ocenjivanja planova, oni se testiraju po *round-robin* sistemu, skuplja se metrika izvršenja, računa konačni skor, nakon čega se planovi rangiraju i bira se najbolji. Evaluacija se bazira na broju radnih jedinica (eng. work units, works) koje plan izvrši kada planer procenjuje kandidate.

Počevši od verzije 4.2, za planove u kešu se vezuju sledeća stanja:

- Nepostojeći (eng. Missing) – ne postoji unos za upit ovog oblika u kešu. Kada nema unosa, planovi kandidati se evaluiraju i bira se najbolji plan. On se zatim dodaje u keš u stanju *Neaktivan,* zajedno sa svojom *works* vrednošću.
- Neaktivan (eng. Inactive) – postoji unos u kešu, tj. planer je video ovakav oblik upita i izračunaoje cenu (works) i rezervisao je mesto, ali se oblik upita ne koristi za generisanje planova upita. Ukoliko je za određeni oblik upita, stanje plana u kešu *Neaktivan,* planovi kandidati se evaluiraju i bira se najbolji. Zatim se upoređuju vrednosti *works* izabranog najboljeg plana i onog koji je neaktivan. Ukoliko izabrani plan ima manju ili vrednost jednaku neaktivnom planu, izabrani plan će zameniti neaktivni unos i biće upisan sa stanjem *Aktivan*. U suprotnom će na tom mestu ostati neaktivni unos, pri čemu će mu se *works* vrednost inkrementirati. Ukoliko treba zameniti planove, a pre nego što se zamena izvrši, neaktivni unos promeni stanje u *Aktivan* (do čega može doći usled izvršenja još nekog upita), zamena će se obaviti samo ako je *works* vrednost novog aktivnog plana veća od izabranog najboljeg plana.
- Aktivan (eng. Active) – unos u kešu je najbolji plan i planer ga koristi kako bi generisao plan za upit. Planer pored toga posmatra i evaluira performanse tog unosa. Ukoliko u nekom momentu njegova *works* vrednost ne zadovoljava definisane kriterijume, plan će preći u neaktivno stanje, i doći će do ponovnog planiranja.

3.4.1 Optimizacija korišćenjem indeksa

Almog Vagman Ciprut [10] navodi u svom istraživanju tri upita za testiranje performansi indeksa. Korišćena je aplikacija za filmove koja ima tri glavne karakteristike:

1. Pretraživanje filmova po nazivu, opisu, žanrovima, scenaristima, rediteljima i glumcima
2. Sortiranje liste filmova prema njihovom žanru i rejtingu
3. Nalaženje spiska filmova u kojima je određeni glumac igrao

Izvršeno je testiranje sa po 20 faza za svaki upit, gde svaka faza predstavlja različitu količinu dokumenata u kolekciji filmova. U svakoj fazi je dodato još 10 000 dokumenata u kolekciju filmova, a zatim je svaki upit izvršen 200 puta, 100 puta sa relevantnim indeksom i 100 puta bez indeksa.

Prvi upit

Prvi upit treba da pronađe filmove kada se pretražuju pomoću stringa. Upit traži zadati string u poljima naziva, opisa, žanrova, scenarista, reditelja i glumaca.

Prilikom pretraživanja bez korišćenja indeksa, nije moguće koristiti tekstualno pretraživanje, već je potrebno koristiti regularne izraze, što je prikazano na slici 21.

```
db.movies.find({
  $or: [
    { name: { $regex: 'Human', $options: "i" } },
    { description: { $regex: 'Human', $options: "i" } },
    { genres: { $regex: 'Human', $options: "i" } },
    { writers: { $regex: 'Human', $options: "i" } },
    { directors: { $regex: 'Human', $options: "i" } },
    { actors: { $regex: 'Human', $options: "i" } }
  ]
});
```

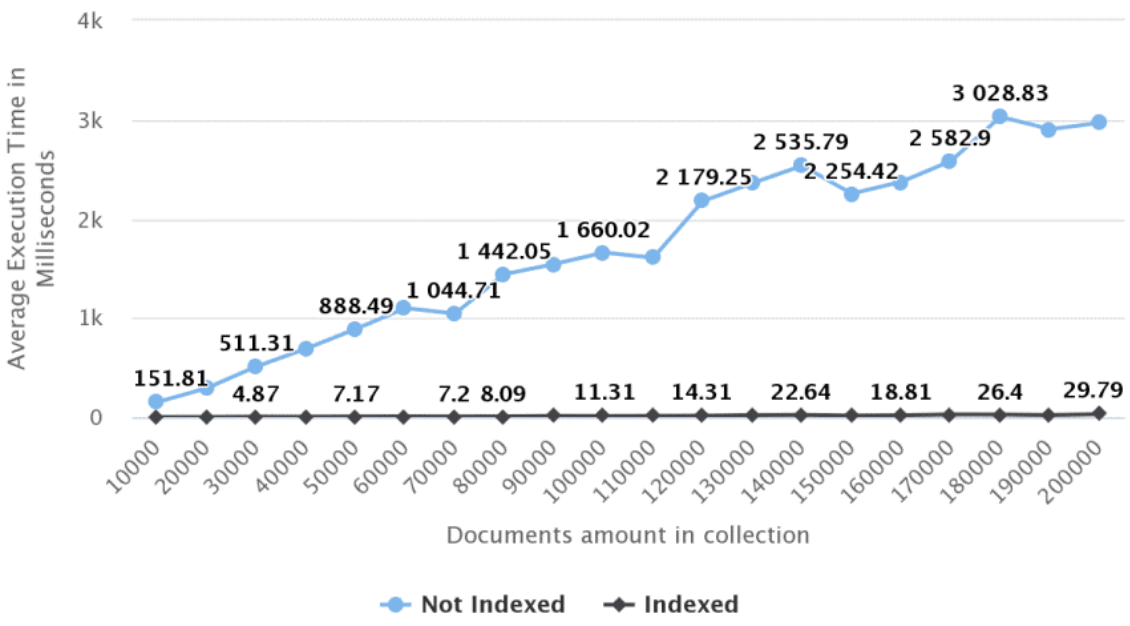
Slika 21 – Prvi upit kada ne postoji indeks

Tekstualni indeks je kreiran nad svim potrebnim poljima (name, description, genres, writers, direstors i actors). Ovakav indeks omogućava tekstualno pretraživanje kao na slici 22.

```
db.movies.find({
  $text: {
    $search: "Human"
  }
});
```

Slika 22 – Prvi upit kada postoji indeks

Rezultati izvršenja upita sa i bez korišćenja indeksa su dati na slici 23.



Slika 23 – Prosečno izvršenje prvog upita

Lako se može zaključiti da je tekstualni indeks znatno doprineo povećanju brzine izvršenja upita, što je s povećanjem broja dokumenata u kolekciji još uočljivije. Broj dokumenata koji je vraćen je 9 868 u oba slučaja, dok broj skeniranih dokumenata iznosi 200 000 prilikom skeniranja kolekciije, a u slučaju korišćenja indeksa iznosi 9 868. Ovi podaci se odnose na poslednju fazu testiranja.

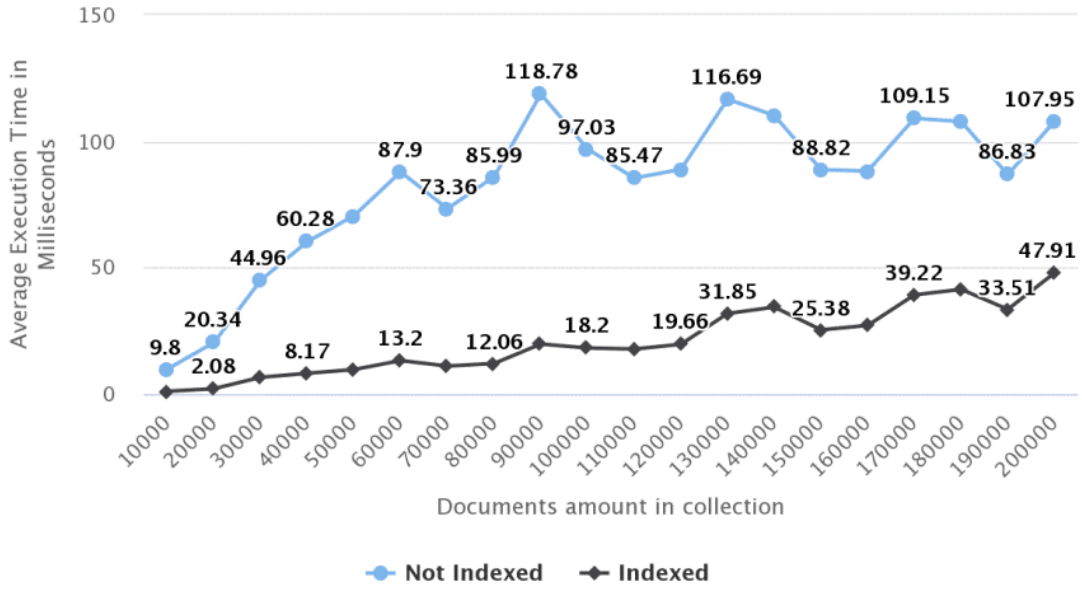
Drugi upit

Drugi upit se odnosi na izdvajanje filmova određenog žanra i sortiranje rezultata po rejtingu (slika 24).

```
db.movies.find({genres: "Fantasy"}).sort({rating: -1});
```

Slika 24 – Drugi upit

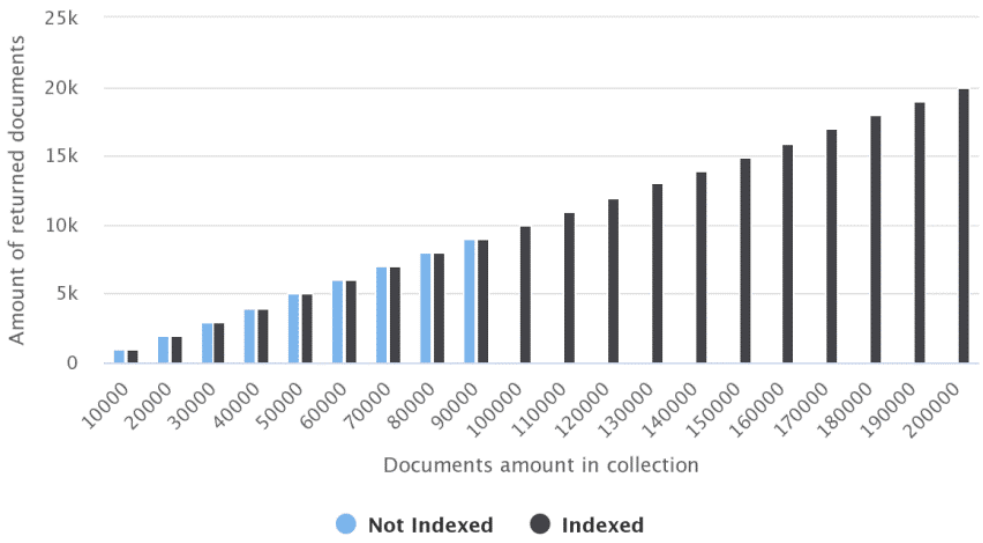
Za potrebe ovog upita, indeks koji je korišćen je složenog tipa i kreiran je nad poljima *genres* i *rating*. Rezultati izvršenja upita sa i bez korišćenja indeksa su dati na slici 25.



Slika 25 – Prosečno izvršenje drugog upita

Broj dokumenata koji je vraćen prilikom skeniranja kolekcije je 0, dok broj dokumenata vraćenih korišćenjem indeksa iznosi 19 957. Broj dokumenata skeniranih izvršenjem upita bez indeksa iznosi 91 605, dok sa indeksom iznosi 19 957. U ovom slučaju, indeks, takođe, doprinosi skeniranju manjeg broja dokumenata i povećanju brzine izvršenja upita. Međutim, uočava se da broj vraćenih dokumenata nije isti u oba slučaja. To se dešava, jer je prilikom sortiranja bez korišćenja indeksa potrebno da podaci ne prelaze limit od 32 MB (Više o sortiranju je dato u poglavlju 3.4.1 - Sortiranje pomoću indeksa). U ovom slučaju, limit je prekoračen, pa se upit ne izvršava do kraja.

Na slici 26 je prikazan broj vraćenih dokumenata u zavisnosti od veličine kolekcije u slučajevima kada se koristi indeks i kada se ne koristi. Uočava se da sortiranje postaje previše zahtevna operacija kada kolekcija ima preko 100 000 dokumenata, a ne koristi se indeks.



Slika 26 – Broj vraćenih dokumenata prilikom sortiranja pomoću indeksa i bez

Treći upit

Treći upit, koji vrši pretragu filmova po glumcima koji igraju u njima, dat je na slici 27.

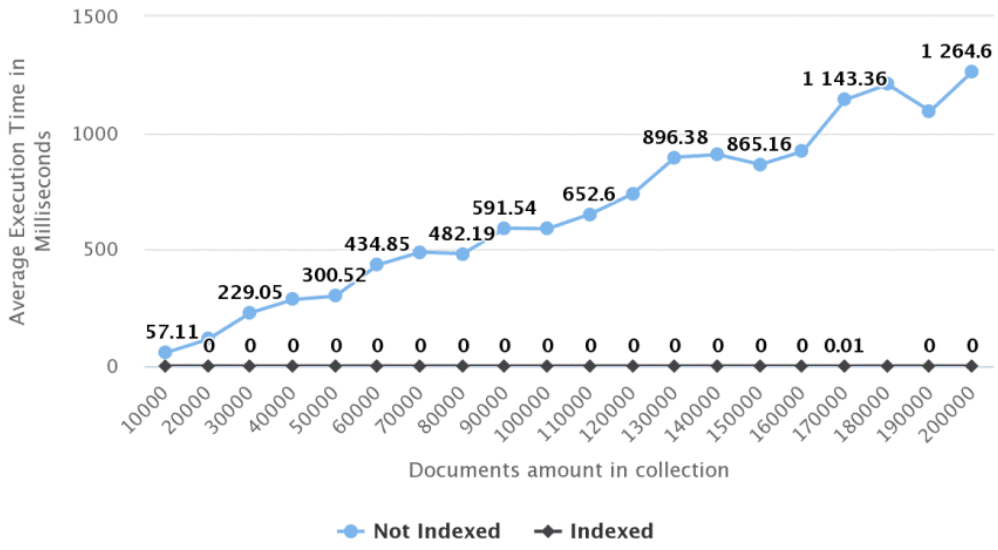
```
db.collection.find({actors: 'Jules Strosin'});
```

Slika 27 – Treći upit

S obzirom na to da je polje *actors* niz stringova, korišćen je indeks sa više ključeva.

Kod ovog upita se uočava da je indeks imao ogroman uticaj na performanse. Izvršenje upita korišćenjem indeksa traje kraće od milisekunde, dok izvršenje bez indeksa prelazi jednu sekundu u kolekcijama koje sadrže preko 170 000 dokumenata (slika 28).

Broj vraćenih dokumenata u oba slučaja iznosi 11, dok broj skeniranih dokumenata iznosi 200 000 bez korišćenja indeksa, a sa korišćenjem indeksa iznosi 11.



Slika 28 – Prosečno izvršenje trećeg upita

Radi upoređivanja performansi, testirala sam navedena tri upita nad istom šemom dokumenata. Svaki upit je izvršen po 20 puta nad kolekcijama koje sadrže 10 000, 50 000, 100 000, 150 000 i 200 000 dokumenata. Rezultati upita su prikazani u tabeli 1.

Broj dokumenata u kolekciji	Prvi upit bez indeksa (ms)	Prvi upit sa indeksom (ms)	Drugi upit bez indeksa (ms)	Drugi upit sa indeksom (ms)	Treći upit bez indeksa (ms)	Treći upit sa indeksom (ms)	Broj dokumenata rezultata
10 000	21,0	4,2	6,5	1,0	7,1	0	1 000
50 000	107,3	9,9	30,1	5,1	30,2	0	5 000
100 000	214,4	19,5	61,5	12,9	55,9	0	10 000
150 000	309,4	29,7	91,8	25,5	82,6	0	20 000
200 000	370,0	85,2	147,3	63,7	108,7	0	50 000

Tabela 1 – Prosečna vremena izvršenja tri upita

Mogu se uočiti različiti rezultati izvršenja u odnosu na rezultate iz studije, što je rezultat nedostatka informacija o broju dokumenata koji zadovoljavaju upit za svaku fazu, kao i nedostatak samih dokumenata koji se nalaze u bazi na osnovu koje je istraživanje sprovedeno u studiji. Na slikama 42 i 43 je prikazano koliko memorije zauzimaju kolekcije i indeksi u oba slučaja, gde se može uočiti razlika. Takođe, važno je napomenuti da je u novim verzijama

MongoDB-a limit za sortiranje u memoriji 100 MB, pa je kod testiranja drugog upita, svaki put vraćen celokupan skup dokumenata koji zadovoljavaju upit. Pored razlika u brzini izvršenja, i u ovom slučaju se zaključuje da indeksi doprinise efikasnosti izvršenja upita, kako kod pretraživanja, tako i prilikom sortiranja. Najveća razlika je kod prvog upita, što ukazuje na veliku prednost tekstualnog indeksa u odnosu na korišćenje logičkog $or operatora za pretraživanje teksta u svim tekstualnim poljima.

Selektivnost indeksa

Za kreiranje što korisnijeg indeksa, potrebno je da on odvaja podatke što je više moguće. Na primer, pretpostavimo da ocena (rating) u kolekciji filmova može imati vrednost između 1 i 3, što znači da će indeks, ako se kreira nad poljem *rating,* odvojiti dokumente samo u 3 odeljka (rating: 1, rating : 2, rating: 3). Međutim, ako se indeks kreira nad poljem *releaseDate*, podaci će biti razdvojiti u više malih odeljaka. Svaki odeljak predstavljata sve filmove objavljene tog dana.

Efikasan upit treba da poveća selektivnost. Selektivnost se može definisati kao sposobnost upita da suzi rezultat pomoću indeksa. Upiti treba da ograniče broj mogućih dokumenata sa indeksiranim poljem. Selektivnost je uglavnom povezana sa složenim indeksom koji uključuje polje niske selektivnosti i zatim druga polja. Na primer, neka postoje sledeći podazi u bazi:

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

Upit koji pretražuje polje a, a zatim polje b - {a: 7, b: “cd”} će skenirati dva dokumenta i vratiti jedan koji zadovoljava upit.

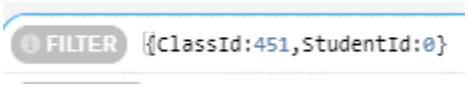
Međutim, neka su vrednosti polja a jednako raspoređene:

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

Upit {a: 7, b: “cd”} će skenirati jedan dokument i vratiti taj jedan dokument. S obzirom na to da je pretražen manji broj dokumenata, potrebno je manje vremena za izvršenje upita.

Indeks nad pojedinačnim poljem i složeni indeks

Izvršen je upit kao na slici 29. Ukoliko posotji indeks samo nad poljem *ClassId*, prosečno vreme izvršenja upita iznosi 7,3 ms. Pretraženo je 2032 indeksa i 2032 dokumenta, dok je vraćen samo jedan. Ako se kreira indeks nad poljem *StudentId*, pretražuje se po 20 indeksa i dokumenata, a vreme izvršenja je 0 ms. Ovo je bolje rešenje zbog velike selektivnosti polja *StudentId*. Međutim, ako se kreira složeni indeks nad oba polja, biće pretražen samo 1 indeks i 1 dokument, a prosečno vreme izvršenja iznosi 0 ms. U složenom indeksu nije bitan redosled polja, ako upit pretražuje polja po jednakosti.

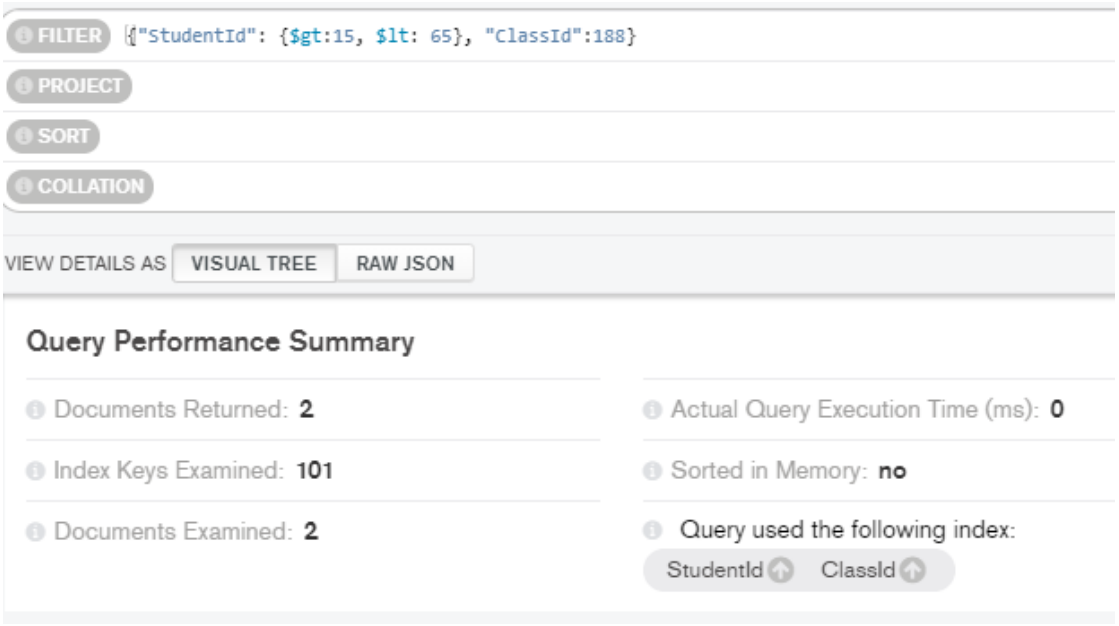


Slika 29 – Upit jednakosti nad dva polja

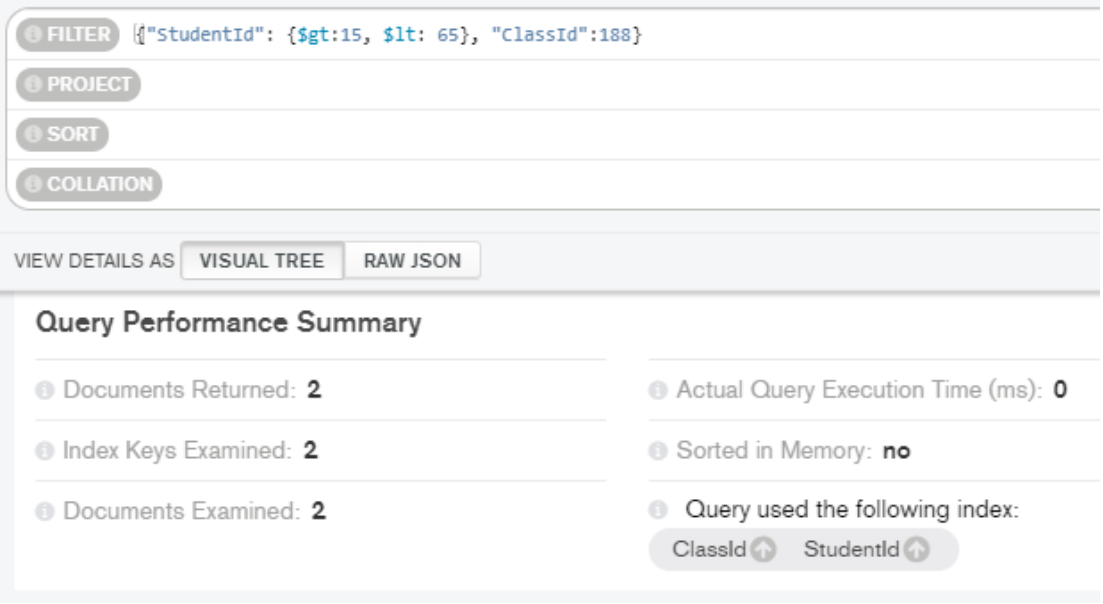
Upit koji pretražuje opseg (Range query)

Kada upit sadži pretraživanje opsega, važno je obratiti pažnju na redosled polja prilikom kreiranja složenog indeksa.

Na slikama 30 i 31 je prikazano izvršenje upita koji izdvaja dokumente gde su ID-evi studenata iz opsega 15 do 65, a ID predmeta je 188. U prvom slučaju je kreiran složeni indeks nad poljima *StudentID* i *ClassID*, a u drugom indeks sa poljima obrnutog redosleda. Uočava se razlika u broju ključeva indeksa koji su pretraženi. To se dešava, jer u prvom slučaju se pretražuje opseg polja *StudentID*, a zatim se filtriraju oni sa zadatim brojem za *ClassID*. Međutim, kada je u pitanju skeniranje opsega, MongoDB može da izvrši prvo traženje po jednakosti (ClassID), ako je to polje prvo u indeksu, a zatim da skenira opseg.



Slika 30 – Izvršenje upita korišćenjem indeksa nad poljima StudentId i ClassId



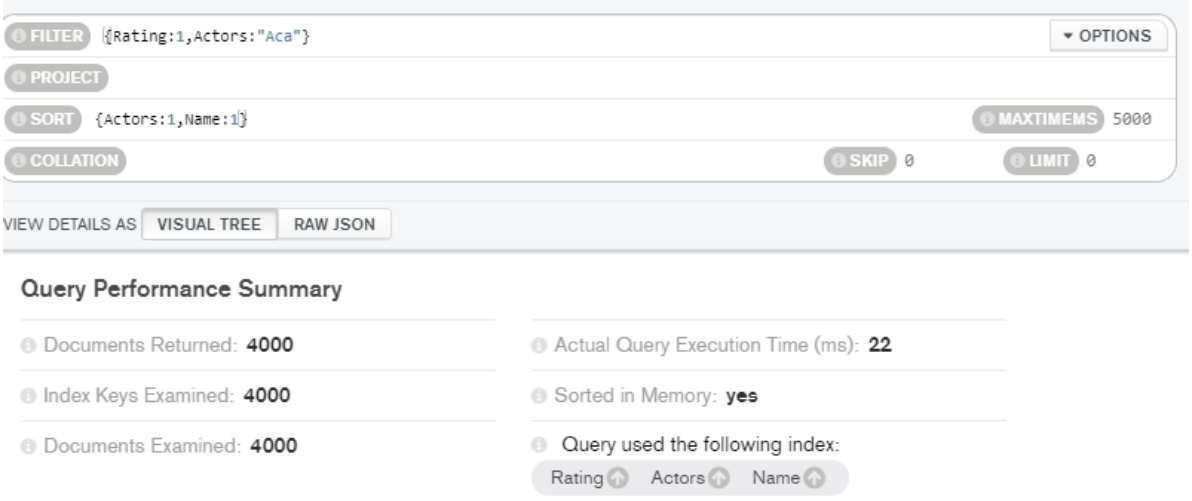
Slika 31 - Izvršenje upita korišćenjem indeksa nad poljima ClassId i StudentId

Sortiranje pomoću indeksa

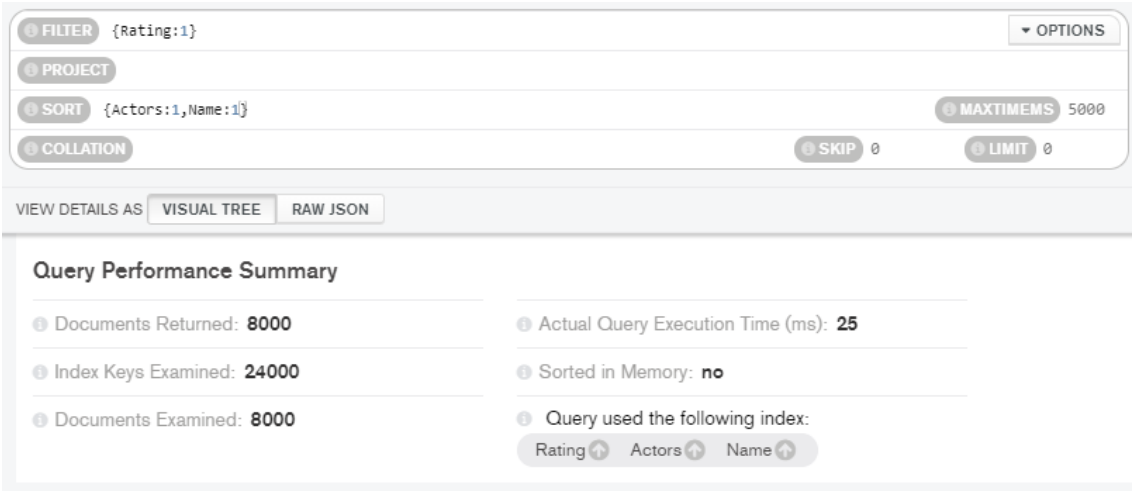
Indeks podržava sortiranje kada je redosled polja po kojima se sortira jednak je redosledu polja u indeksu. Prilikom korišćenja operacije sortiranja na pojedinačnom polju, gde je to polje indeksirano pomoću indeksa pojedinačnog polja, indeks će uvek podržati sortiranje. Složeni indeks može podržati sortiranje na više polja. Da bi podržao sortiranje, složeni indeks mora da ima definisana polja u istom redosledu kao i sortirana polja. Takođe, smer sortiranja (opadajući ili rastući) mora biti ili potpuno isti ili suprotan za svako polje navedeno u indeksu. Prilikom korišćenja sortiranja polja koja su prefiks indeksa, prefiks mora poštovati prethodno navedena pravila. Sortiranje nad poljima koja nisu prefiks je moguće, samo ako upit sadrži pretraživanje polja po jednakosti koja čine prefiks indeksa, a prethode poljima koja su navedena kao uslov sortiranja.

Ako se sortiranje ne može izvršiti pomoću indeksa, onda se izvršava u memoriji. Sortiranje korišćenjem indeksa ima bolje performanse od izvođenja sortiranja u memoriji (bez indeksa). Pored toga, sortiranje u memoriji ima svoja ograničenja. Ako se ne koristi indeks, sortiranje će se prekinuti, ako je za tu operaciju potrebno više od 100 MB memorije (odnosno 32 MB kod verzije 4.2 i starijih). U tom slučaju MongoDB neće nužno vratiti grešku, već samo prazan skup zapisa.

Kreiran je indeks nad poljima *Rating, Actors* i *Name*. Na slici 32 se može videti da se indeks ne koristi za sortiranje ako polja u filteru ne prethode svim poljima u *sort* delu, dok se na slici 33 može videti da se indeks koristi za sortiranje kada su polja pravilno navedena.



Slika 32 – Sortiranje u memoriji



Slika 33 – Sortiranje korišćenjem indeksa

Retki i parcijalni ideksi

Kao primer poređenja ova dva indeksa, kreirana su oba nad poljem *Rating*. Na slici 35 je prikazano kreiranje retkog indeksa, a na slici 34 parcijalnog.

```
> db.Films.createIndex({ "Rating": 1 }, {partialFilterExpression: { "Rating": { $exists: true } }})
```

Slika 34 – Kreiranje parcijalnog indeksa

```
> db.Films.createIndex({ "Rating": 1 }, { sparse: 1 })
```

Slika 35 – Kreiranje retkog indeksa

Retki indeksi zauzimaju manje prostora od parcijalnih. U datom primeru retki indeks zauzima 73,7 kB, dok parcijalni indeks zauzima 131,1 kB. Vreme izvršenja upita je u oba slučaja iznosilo 2 ms. Parcijalni indeksi određuju unose indeksa na osnovu navedenog filtera, koji se ne mora odnositi samo na postojanje nekog polja. Filter može sadržati i polja koja nisu indeksni ključevi. S obzirom na ove karakteristike, preporučuje se korišćenje parcijalnog indeksa.

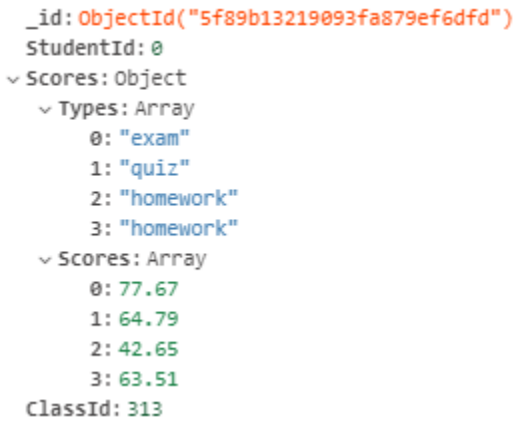
3.4.2 Dodatna optimizacija upita

Kada je reč o efikasnosti upita, pored kreiranja indeksiranih upita i korišćenja selektivnosti upita, kao što je prethodno opisano, postoje i drugi koncepti čija primena daje doprinos efikasnosti.

Pokriveni upiti

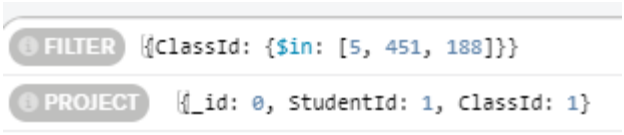
Pokriven upit (eng. Covered query) je upit koji se u potpunosti može ispuniti korišćenjem indeksa i ne mora da pretražuje dokumente. Stoga, pokriveni upit treba da sadrži sva polja kao deo indeksa, a samim tim i rezultat treba da sadrži sva ta polja.

Za testiranje pokrivenih upita, korišćen je dokument o podacima bodova studenata na različitim predispitnim obavezama, prikazan na slici 36.[13]



Slika 36 – Dokument o studentu

U kolekciji postoji 1 000 100 dokumenata. Izvršen je upit koji izdvaja ID-eve studenata na predmetima sa ID-evima 5, 451 i 188 (slika 37). U rezultat su uključeni samo ID studenta i ID predmeta.



Slika 37 – Pokriveni upit

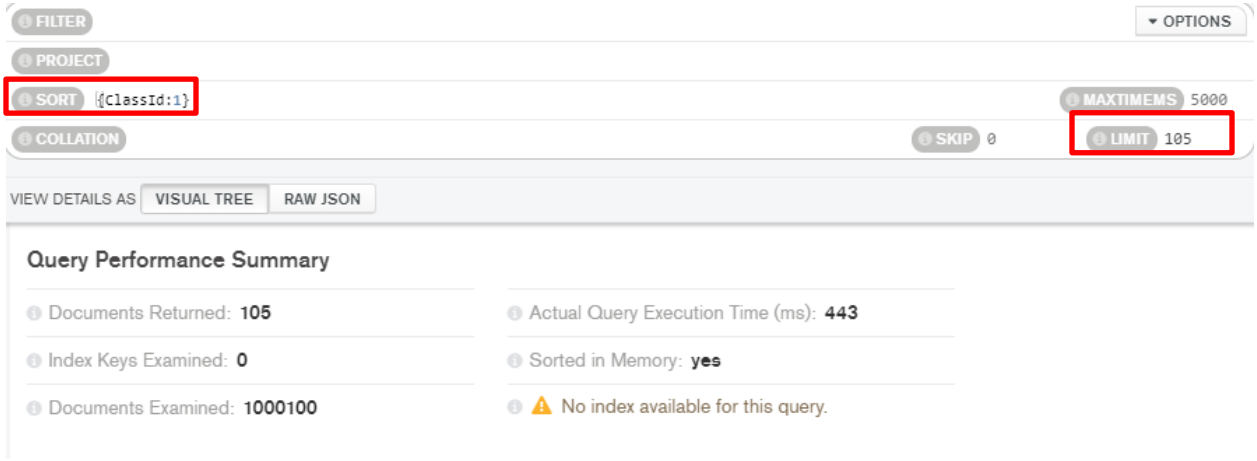
Svaki upit je izvršen 10 puta. Prilikom izvršenja upita bez korišćenja indeksa, vrši se pretraživanje cele kolekcije, iako je vraćeno 5926 dokumenata. Prosečno vreme izvršenja iznosi 369,4 ms.

Zatim je izvršen isti upit, ali je prethodno kreiran indeks nad poljima *ClassID* i *StudentID*. Na ovaj način je kreiran pokriveni indeks koji pokriva dati upit, jer sva polja koja upit vraća se nalaze u indeksu. Prosečno vreme izvršenja je 5,1 ms, što znači da je postignuto ubrzanje od preko 70 puta. Takođe, izvršenjem korišćenjem indeksa, pretraženo je 5929 dokumenata, a ne cela kolekcija kao u prethodnom slučaju.

Ukoliko se kreira indeks samo nad poljem *ClassID,* prosečno vreme izvršenja iznosi 13,3 ms, što dokazuje prednost pokrivenih upita.

Ograničenje broja rezultata

MongoDB kursori vraćaju rezultate u grupama od više dokumenata. Ukoliko je poznat broj rezultata koji je neophodan, može se iskoristiti metoda *limit* kako bi se smanjili zahtevi. Ova metoda se uglavnom koristi u kombinaciji sa *sort* operacijom i može doprineti ako ne postoji indeks koji se koristi za sortiranje. Na ovaj način je omogućeno sortiranje u memoriji, dok sortiranje cele kolekcije ne bi bilo moguće, jer prevazilazi kapacitet memorije. Na slici 38 je prikazano limitiranje rezultata na 105 dokumenata.



Slika 38 – Limitiranje rezultata

Projekcija

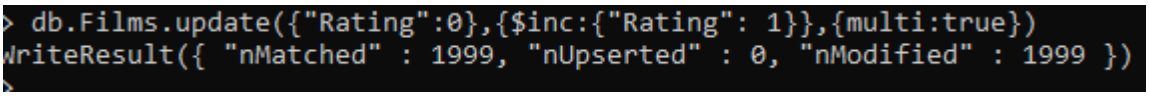
Ukoliko nisu potrebni svi podaci iz dokumenta, bolje performanse se mogu postići ako se vrate samo neophodna polja u okviru rezultata. Projekcija eksplicitno uključuje polja setovanjem vrednosti polja na 1 u projekcionom dokumentu. *_id* je podrazumevano polje i može se ukloniti iz rezultata tako što će se u projekciji setovati na nulu (_id:0).

Izvršenje upita sa slike 37 iznosi 369,4 ms bez korišćenja ideksa i sa korišćenjem projekcije. S druge strane izvršenje istog upita bez projekcije iznosi u proseku 353,1 ms. Dakle, pretraživanje celih dokumenata je efikasnije.

Međutim, korišćenje projekcije za ograničavanje polja rezultata upita može poboljšati druge performanse kao što su: uklanjanje nepotrebnih polja iz rezultata upita, što rezultuje uštedom na mrežnom saobraćaju, jer je rezultat manji; ograničavanje polja rezultata za postizanje pokrivenog upita (vraćanje indeksiranih dokumenata bez pretraživanja cele kolekcije). Već je pokazano da se pokriveni upiti brže izvršavaju. Za ovakav upit sa korišćenjem indeksa je potrebno 5,1 ms. Što se tiče veličine dokumenta u datom primeru, ona iznosi 32 B po dokumentu sa primenom projekcije, odnosno 176 B po celom dokumentu (bez projekcije). To znači da rezultujući skup dokumenata uz primenu projekcije iznosi 185 MB, dok bez projekcije iznosi 1018 MB.

Operator inkrementiranja

Upotreba $inc operatora za inkrementiranje ili dekrementiranje vrednosti u dokumentima je još jedna opcija za optimizaciju upita. Operator inkrementira vrednost polja na serverskoj strani, što je alternativa za selektovanje dokumenta, kreiranje par modifikacija na klijentu, a zatim upis celog dokumenta na server. Ovaj operator može pomoći i u izbegavanju uslova trke, do kog dolazi kada dve instance aplikacije pošalju upit za dokument, zatim inkrementiraju polje i sačuvaju ceo dokument u isto vreme. Na slici 39 je prikazan primer inkrementiranja polja *Rating* za 1 i to onih polja kod kojih je *Rating* iznosio 0.



Slika 39 – Korišćenje operatora inkrementiranja

3.5 **Veličina memorije**

Pored modeliranja podataka i izbora indeksa, važno razmatranje u optimizaciji performansi je određivanje veličine radnog seta. Kao i većina baza podataka, MongoDB najbolje funkcioniše kada se radni skup aplikacije (indeksi i podaci kojima se najčešće pristupa) staju u memoriju. Izvršenje operacija u memoriji je brže od izvršenja operacija na disku. Druge vrste optimizacije neće značajno poboljšati performanse baze podataka, ako nema dovoljno RAM memorije. Ako odnos cena i performanse ima veći prioritet od samih performansi, upotreba brzih SSD-ova, kao

dopune manjim količinama RAM-a, je koristan izbor za dizajn. Kada se radni set aplikacije uklopi u RAM, aktivnost čitanja sa diska biće mala, što utiče na performanse.

Na slici 40 je prikazano sortiranje 476600 dokumenata u memoriji. Izvršenje iznosi 212 ms. Na slici 41 je prikazano sortiranje 476700 dokumenata, čija veličina prelazi 100MB RAM-a, pa je uključena opcija allowDiskUse(), kako bi se sortiranje izvršilo na disku. Izvršenje iznosi 542 ms, što je veliki skok koji dokazuje prednost izvršenja operacija u memoriji u pogledu brzine.

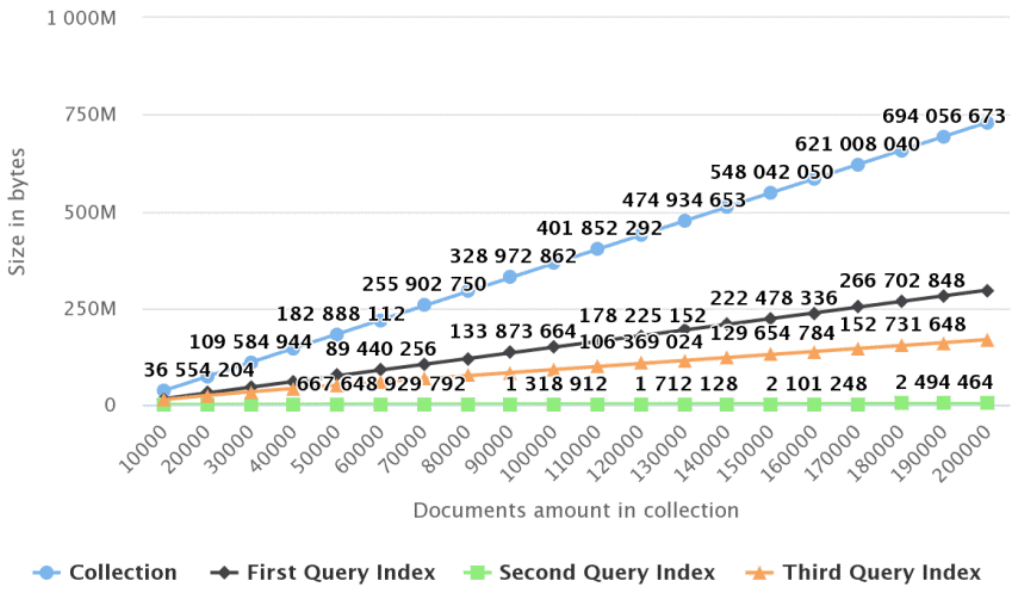
```
> db.Students.find({"StudentId": {$gt:9}}).sort({"StudentId":1}).limit(476600)
```

Slika 40 – Sortiranje u memoriji

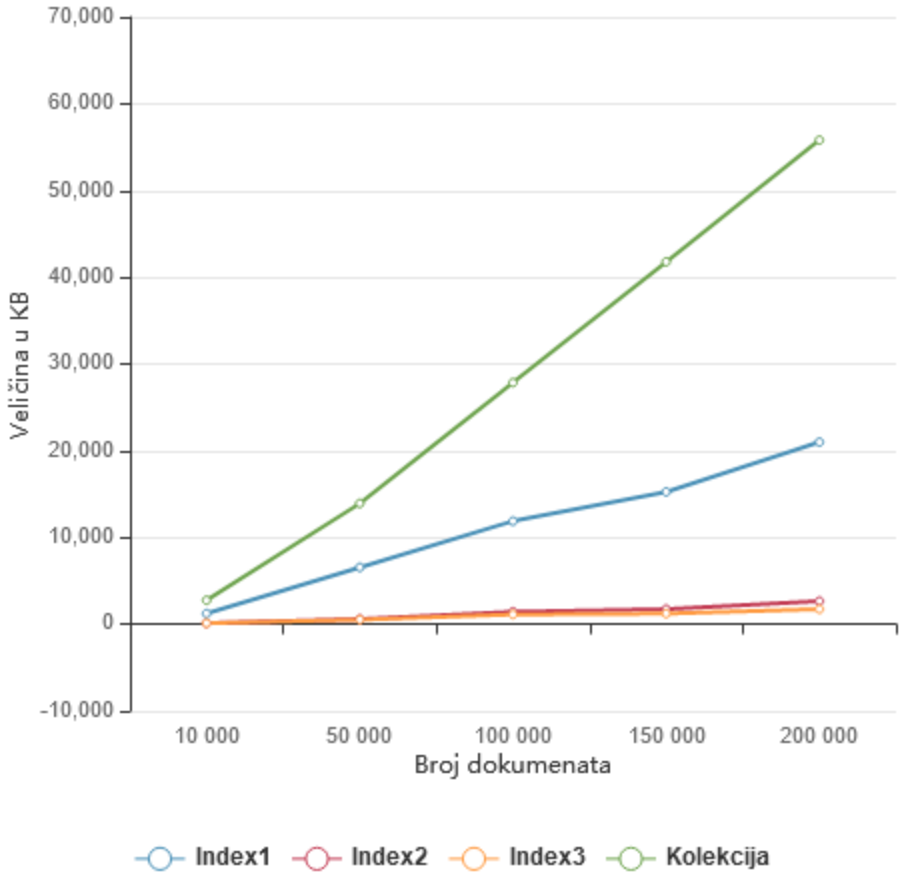
```
> db.Students.find({"StudentId": {$gt:9}}).sort({"StudentId":1}).limit(476700).allowDiskUse()
```

Slika 41 – Sortiranje na disku

Kao primer veličine sistema koji ima od 10 000 do 200 000 dokumenata, uzeta je kolekcija iz poglavalja 3.3.1. Na slici 42 je prikazana veličina indeksa korišćenih za optimizaciju svakog od tri prethodno opisana upita u studiji (poglavlje 3.4.1), kao i količina memorije koju zauzima kolekcija dokumenata. Na slici 43 je prikazana veličina kolekcije i indeksa koje sam koristila za testiranje upita iz navedene studije.



Slika 42 – Veličine kolekcija i indeksa u studiji



Slika 43 – Veličine kolekcija i indeksa

3.6 Pregled performansi

Profajler (MongoDB Profiler)

MongoDB poseduje ugrađeni profajler koji pruža uvid u performanse svake operacije. Namenjen je skupljanju podataka i monitoringu baze podataka. Zbog čuvanja dodatnih informacija, preporučuje se da bude isključen tokom normalnog rada. Profajler upisuje podatke u kolekciju *system.profile*, koja je ograničene veličine, a podrazumevana je 1MB. Može se koristiti da se lociraju svi upiti ili operacije upisa koje su spore. Te informacije se mogu koristiti za optimizaciju, npr. za definisanje indeksa. Profajler skuplja informacije o komandama koje se izvršavaju na *mongod* instanci. Ovo uključuje CRUD operacije, kao i konfiguracione i administrativne komande.

Profajler poseduje tri različita nivoa profajliranja:

- 0 – isključen profajler, ne skuplja nikakve podatke. Ovo je podrazumevani nivo.

- 1 – sakuplja podatke za operacije čije izvršenje traje duže od vrednosti *slowms*[5].
- 2 – sakuplja podatke za sve operacije.

Na slici 44 je prikazano uključivanje profajlera sa nivoom 2.

```
> db.setProfilingLevel(2)
{ "was" : 0, "slowms" : 100, "sampleRate" : 1, "ok" : 1 }
```

Slika 44 – Uključivanje profajlera

Nad profajlerom se mogu postavljati različiti upiti kako bi se utvrdile performanse i pronašli delovi koji se mogu optimizovati. Na primer, na slici 45 su prikazani upiti koji izvršavaju skeniranje kolekcije, jer ne postoji odgovarajući indeks. Za svaki vraćeni upit, profajler vraća sve podatke o upitu. Crvenom bojom su obeleženi podaci o tome koji je upit u pitanju i koliko je dokumenata pretraženo. Takođe, prikazan je i plan upita (slika 46).

```
db.system.profile.find({"planSummary":{$eq:"COLLSCAN"}, "op" : {$eq:"query"}}).sort({millis:-1}).limit(1).pretty()
{
        "op" : "query",
        "ns" : "StudentsDB.Students",
        "command" : {
                "find" : "Students",
                "filter" : {
                        "StudentId" : {
                                "$gt" : 15,
                                "$lt" : 65
                        },
                        "ClassId" : 188
                },
                "limit" : 20,
                "maxTimeMS" : 5000,
                "returnKey" : false,
                "showRecordId" : false,
                "lsid" : {
                        "id" : UUID("3ca87f7e-a5ae-4c99-b116-fffc4f58f0e8")
                },
                "$db" : "StudentsDB"
        },
        "keysExamined" : 0,
        "docsExamined" : 1000100,
        "cursorExhausted" : true,
        "numYield" : 1000,
        "nreturned" : 2,
        "queryHash" : "A4F2F79C",
        "planCacheKey" : "A4F2F79C",
```

Slika 45 – Rezultat profajlera

[5] *slowms* se izražava u milisekundama

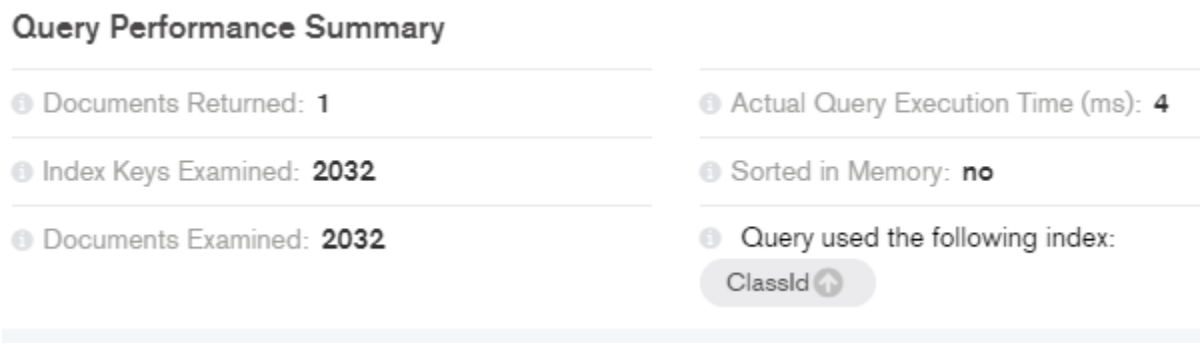
Slika 46 - Rezultat profajlera

Explain metoda

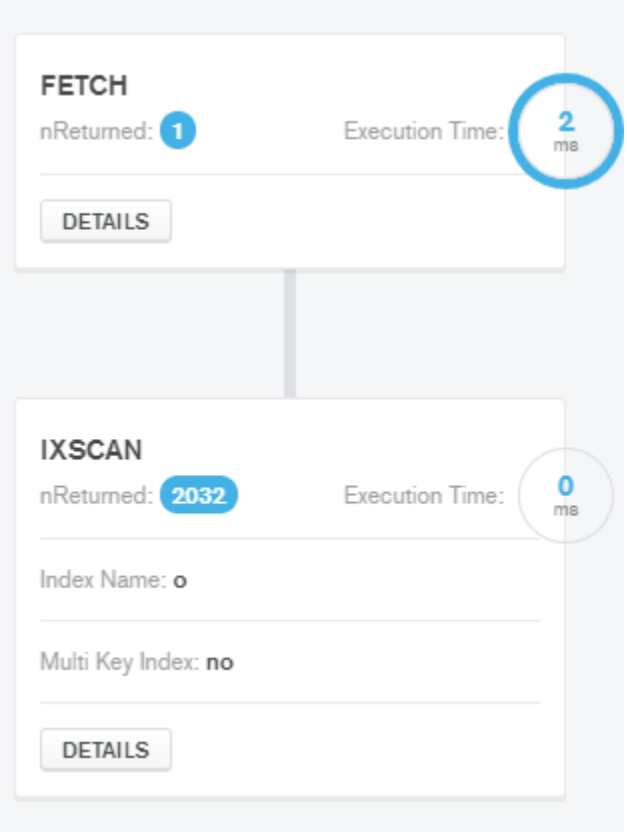
Slično kao i kod profajlera, metoda *explain* pruža podatke za pregled informacija o planovima upita i statistici izvršenja planova. Ova metoda se može pozvati nad pojedinačnim upitom. Namena profajlera je da identifikuje sve upite koji se mogu optimizovati, a zatim se koristi metoda *explain* za pružanje informacija o izvršenju jednog upita. Rezultati predstavljaju planove upita kao stablo faza. Svaka faza prosleđuje svoje rezultate (dokumente ili ključeve indeksa) roditeljskoj fazi. Listovi pristupaju kolekciji dokumenata ili indeksima. Interni čvorovi manipulišu dokumentima ili ključevima indeksa koji dolaze od čvorova dece. Koren stabla je finalna faza odakle MongoDB uzima rezultujući skup.

Na slikama je prikazan izgled *explain* metode u MongoDB Compass-u. Izvršen je upit db.Students.find({“ClassId”:451,”StudentId”:0}).sort({“ClassId”:1}).

Na početku se prikazuju osnovne informacije o celom upitu (slika 47). Ispod je prikazano stablo izvršenja (slika 48), gde se mogu videti detalji svake faze (slika 49). Na slici 49 su označeni: ime faze, ime indeksa, smer sortiranja i ime roditeljske faze.



Slika 47 – Osnovni podaci o upitu



Slika 48 – Stablo izvršenja upita

```
{
 "stage": "IXSCAN",
 "nReturned": 2032,
 "executionTimeMillisEstimate": 0,
 "works": 2033,
 "advanced": 2032,
 "needTime": 0,
 "needYield": 0,
 "saveState": 2,
 "restoreState": 2,
 "isEOF": 1,
 "keyPattern": {
  "ClassId": 1
 },
 "indexName": "o",
 "isMultiKey": false,
 "multiKeyPaths": {
  "ClassId": []
 },
 "isUnique": false,
 "isSparse": false,
 "isPartial": false,
 "indexVersion": 2,
 "direction": "forward",
 "indexBounds": {
  "ClassId": [
   "[451, 451]"
  ]
 },
 "keysExamined": 2032,
 "seeks": 1,
 "dupsTested": 0,
 "dupsDropped": 0,
 "parentName": "FETCH"
}
```


Slika 49 – Detalji IXSCAN faze

4 Zaključak

U ovom radu su prikazani načini optimizacije MongoDB baze podataka. Optimizacija se u najvećoj meri odnosi na modeliranje podataka u skladu sa zahtevima aplikacije i pravilno korišćenje indeksa za optimizaciju upita. Najvažnije pitanje, prilikom projektovanja, je kakvi će se upiti izvršavati nad bazom podataka. Na osnovu toga se može odrediti da li je optimalnije ugnezditi dokumente, ako se često preuzimaju zajedno, ili ih referencirati, ako im se pristupa odvojeno. Takođe i izbor indeksa zavisi od upita koji se izvršavaju. S obzirom na to da indeksi mogu zauzeti dosta prostora, treba kreirati samo potrebne indekse. U zavisnosti od potrebnih performansi, može se odrediti u kojim slučajevima je bitnije da se poboljša brzina izvršenja, a gde je potrebno uštedeti memoriju. Pored toga, praćenje veličine podataka kolekcije i indeksa i uvid u to da li oni prelaze ograničenja RAM memorije, ima značajnu ulogu kod velikih baza podataka, zbog toga što su operacije koje se izvode u memoriji brže od onih koje se izvode na disku. Za praćenje performansi upita, MongoDB poseduje funkciju *explain*, koja pruža detaljan prikaz izvršenja upita, što doprinosi zaključivanju kako se upit može optimizovati. Pored ove funkcije, profajler se može koristiti za identifikovanje sporih ili drugih upita sa određenim karakteristikama, kako bi se oni kasnije optimizovali.

Optimizacija je jako važna prilikom kreiranja modela podataka i upita koji se izvršavaju nad bazom podataka, kako bi se zadovoljile potrebe korisnika u pogledu brzine izvršenja i uštede resursa. Iako postoje preporuke za različite načine optimizacije, ne postoji idealno rešenje za sve vrste sistema. Zbog toga je važno odrediti, pre projektovanja baze podataka, kakvi će se podaci pamtiti, koji će se upiti izvršavati i koji su zahtevi za performansama.

5 Literatura

[1] – NoSQL Tutorial, learn NoSQL features, types, what is, adventages - *Guru99*
https://www.guru99.com/nosql-tutorial.html

[2] - MongoDB zvanična dokumantacija
https://docs.mongodb.com/manual/

[3] – Schema basics – *Learn MongoDB the hard way*
http://learnmongodbthehardway.com/schema/schemabasics/

[4] - NoSQL Databases and Data Modeling Techniques for a Document-oriented NoSQL - *Database Robert T. Mason*
http://proceedings.informingscience.org/InSITE2015/InSITE15p259-268Mason1569.pdf

[5] – 6 Rules of Thumb for MongoDB Schema Design –*William Zola*
https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-1

[6] – Design patterns for MongoDB – *Semi Koen*
https://towardsdatascience.com/design-patterns-for-mongodb-894767315905

[7] - 5 Successful Tips You Need to Optimize MongoDB – *Vaclav Dedik*
https://www.roihunter.com/blog/5-successful-tips-you-need-to-optimize-mongodb

[8] – MongoDB applied design patterns – *Rick Copeland*
https://www.oreilly.com/library/view/mongodb-applied-design/9781449340056/ch01.html

[9] – Introduction to MongoDB indexes – *Dvir Arad*
https://medium.com/swlh/introduction-to-mongodb-indexes-cdb216f54f80

[10] - MongoDB Optimization - Indexes and when to use them – *Almog Vagman Ciprut*
https://dev.to/almogvc/mongodb-optimization-indexes-and-when-to-use-them-38kg

[11] – Optimizing MongoDB compound indexes – *A. Jesse Jiryz Davis*
https://emptysqua.re/blog/optimizing-mongodb-compound-indexes/

[12] – Query planner – *Migual Angen Nieto*
https://www.slideshare.net/miguelangelnieto/query-planner-88498371

[13] - MongoDB Query Optimization: Covered Query – *Hafidz Jazuli Luthfi*
https://dev.to/bladefidz/mongodb-query-optimization-covered-query--34d2

[14] - How to Use the MongoDB Profiler and explain() to Find Slow Queries – *Studio 3T*
https://studio3t.com/knowledge-base/articles/mongodb-query-performance/