Sistemi za upravljanje bazama podataka

Optimizacija upita kod MongoDB baze podataka

Mentor:
prof. dr Aleksandar Stanimirović

Student:
Marija Milošević 1045

Sadržaj

1. Uvod

U današnje vreme se ne može zamisliti sistem koji za funkcionisanje ne koristi neku od dostupnih baza podataka. Od svojih početaka sedamdesetih godina prošlog veka, baze podataka karakterišu se kao skupovi povezanih podataka i način na koji su oni organizovani. Upiti su zahtevi za određenim podacima iz baze, čija složenost varira. Budući da su u današnjim sistemima baze jako kompleksne, tako će od kompleksnosti upita zavisiti i vreme neophodno za pribavljanje svih neophodnih podataka iz baze. Optimizacija upita je deo obrade upita gde će sistem za upravljanje bazom podataka uporediti različite strategije i odabrati najefikasniju za njegovo izvršenje. Ovo je automatizovan proces koji zavisi od više faktora i u obzir uzima različite strategije i mere prema kojima bira najoptimalniji način za izvršenje upita. U velikim sistemima je jako bitno primeniti najoptimalnija rešenja kako bi se posao mogao obavljati neometano i bez zadrški.

U ovom radu će kratko biti predstavljeni osnovni koncepti optimizacije upita, nakon čega će se preći na konkretnu bazu – u ovom slučaju, MongoDB. Biće razmotreni relevantni koncepti koji su neophodni za razumevanje funkcionisanja optimizacije (vrste upita, indeksi), a zatim i sam proces optimizacije i pregleda izvršenja upita.

2. Osnovno o optimizaciji

Optimizacija upita je proces koji pokušava da odredi najefikasniji način za izvršavanje određenog upita, uzimajući u obzir moguće planove za izvršenje. U suštini, korisnici ne mogu da pristupe direktno optimajzeru upita: kada se upit prosledi serveru, on će biti parsovan pa prosleđen optimajzeru gde će do optimizacije i doći. Međutim, neke baze podataka dozvoljavaju neposredno uključivanje korisnika korišćenjem hintova.

Uglavnom se dolazi do kompromisa između vremena neophodnog za nalaženje najboljeg plana i kvaliteta samog izbora; optimajzer možda neće samostalno odabrati najbolji odgovor. Različiti sistemi za upravljanje bazama podataka će na drugačije načine balansirati ove kriterijume. Optimajzeri bazirani na ceni će evaluirati resurse planova i koristiti to kao osnovu za selekciju. Oni dodeljuju „cenu“ svakom mogućem planu upita i biraju plan s najmanjom cenom. Te cene se koriste kako bi odredili cenu evaluacije upita, kada je reč o broju zahtevnih I/O operacija, dužini procesorskog puta, količini prostora na disku i vremena zauzeća na istom, i ostalim faktorima koje definiše sistem. Skup planova za ispitivanje se formira ispitivanjem mogućih pristupnih puteva (npr. pristup primarnim ili sekundarnim indeksom, puno skeniranje fajlova) i različitih tehnika spajanja tabela (kada je reč o relacionim bazama podataka). Prostor za pretragu može biti jako veliki u zavisnosti od složenosti samog upita. Optimizacija se svodi na logičku, koja generiše niz relacione algebre za rešavanje upita, i fizičku, koja određuje na koji će se način operacija izvesti. U praksi, pretpostavke optimajzera mogu biti sklone greškama, ali svakako nije očekviano da on odabere neki preterano loš plan.

Većina optimajzera predstavlja planove kao stablo „čvorova plana“. Jedan čvor plana predstavlja jedinstvenu operaciju neophodnu za izvršenje upita. Čvorovi su uređeni u stablo, pri čemu međurezultati idu od dna stabla ka vrhu. Svaki čvor ima nula ili više potomaka – izlaz čvorova potomaka dolazi do roditeljskog čvora. Na primer, čvor spoja (*join node*) će imati dva potomaka koja predstavljaju dva operanda spoja, dok će čvor sortiranja imati samo jednog potomka, koji predstavlja ulaz koji treba sortirati. Listovi stabla su čvorovi koji kreiraju rezultate na osnovu skeniranja diska, izvršavanjem skeniranja indeksa ili sekvencijalnog skeniranja.

O načinu rada optimajzera i planera kod MongoDB baze podataka više reči će biti u poglavlju 4.1.

3. Osnovni koncepti MongoDB baze podataka

MongoDB je moćna, fleksibilna i skalabilna baza podataka opšte namene. Od prvog izdanja 2009. godine, MongoDB je postala najpopularnija nerelaciona baza podataka.

Primarni razlog za udaljavanje od relacionog modela je visok nivo skalabilnosti. Ovakva baza menja koncept „reda“ fleksibilnijim modelom - „dokumentom“. Pored toga, nema predefinisanih šema: ključevi i vrednosti dokumenata nisu fiksnih tipova ili veličina. Bez fiksne šeme, dodavanje i uklanjanje polja postaje lakše. U suštini, ovo ubrzava razvoj, ali je i lakše eksperimentisati: mogu se isprobati različiti modeli za podatke, pa odabrati najprikladniji. Takođe, manje problema nastaje iz korisničkog ugla ukoliko je neophodna promena u modelu podataka.

MongoDB je „document-oriented“ baza - ima dokument kao osnovnu jedinicu podataka. Dokumenti se predstavljaju u JSON notaciji, i sadrže uređene skupove ključeva sa odgovarajućim vrednostima . Kolekcije su grupe dokumenata i slične su tabeli u relacionom modelu, dok je dokument pandan zapisu u tabeli (jednom redu). Kolekcije pripadaju bazi u MongoDB-u. Dokumenti su serijalizovani u BSON formatu. Na ovaj način, uključeni su i dodatni tipovi poput int, long, date, floating point. Ovo olakšava aplikacijama obradu, sortiranje i upoređivanje podataka. Jedan od razloga što se timovi opredeljuju za baze sa JSON i BSON modelima podataka je dinamična i fleksibilna šema koju pružaju nasuprot striktnim tabelarnim modelama u relacionim bazama. JSON dokumenti su polimorfni – polja mogu varirati od dokumenta do dokumenta unutar jedne kolekcije, za razliku od relacionog modela gde se mora znati struktura tabele. Na ovaj način se elegantno rukuje podacima bilo koje strukture. Pored toga, ukoliko treba dodati novo polje u dokument, ono se može dodati bez ikakvog uticaja na ostale dokumente u kolekciji i bez modifikacije centralnog sistemskog kataloga. Kada je neophodno promeniti model podataka, „document-oriented“baza nastavlja da smešta modifikovane objekte bez potrebe da se izvede operacija ALTER TABLE (koja dosta košta) ili redizajnira cela šema.

Još jedna od karakteristka „document-oriented“ baza je da nude API ili upitni jezik koji omogućava korisniku da nađe dokumente na osnovu njihovog sadržaja ili metapodataka – upotreba upitnog jezika, sekundarnih indeksa i agregacionog pajplajna omogućava nalaženje bilo kakvih podataka – od jednostavnih upita preko sofisticiranih pajplajnova za analizu podataka, sve do obrade geoprostornih podataka.

3.1. Upiti

Za izvršavanje upita u MongoDB-u koristi se *find* metoda. Kao rezultat dobijaju se skupovi dokumenata u kolekciji – u zavisnosti od samog upita, rezultat ne mora sadržati nijedan dokument ili pak može sadržati celu kolekciju. Koji će dokumenti biti uključeni u rezultat zavisiće od kriterijuma upita koji su zadati.

Upiti variraju od jednostavnih do kompleksnih i mogu uključivati različite operatore. Pored toga, upiti se mogu vršiti nad ugneždenim dokumentima, nizovima (na primer, pretraga po dužini niza, sa više kriterijuma, itd.), a mogu se i generisati projekcije – gde se selektuju samo određena polja koja će biti uključena u rezultat.

3.2. Indeksi

Indeksi omogućavaju efikasno izvršavanje upita kod MongoDB-a. Bez njih, MongoDB bi morao da izvrši tzv. „collection“ skeniranje – skeniranje svakog dokumenta u kolekciji kako bi bili odabrani oni koji se podudaraju sa kriterijumima upita. Ukoliko postoji odgovarajući indeks za upit, MongoDB ga može koristiti kako bi ograničio broj dokumenata koje skenira.

Indeksi su posebne strukture podataka koje skladište mali deo podataka iz kolekcije, a za indekse se u MongoDB-u koristi struktura B stabla. Indeks čuva vrednost određenog polja ili skupa polja uređenih po vrednosti polja. Upravo to uređenje po vrednosti omogućava efikasnu pretragu po jednakosti i operacije u upitima koje se vezuju za određeni opseg. Pored toga, MongoDB može vratiti sortirane rezultate koristeći uređenje indeksa. Indeksi mogu da se kreiraju nad poljem dokumenta, nad delom polja ili nad više polja (kompozitni indeksi). Indeksi ubrzavaju operacije pretrage ali usporavaju operacije ažuriranja podataka.

MongoDB kreira podrazumevani *unique* index nad *_id* poljem tokom kreiranja kolekcije. Ovaj indeks sprečava unošenje dva dokumenta sa istom vrednošću _id polja, i ne može se izbrisati. Ovo polje može sadržati bilo koju vrednost (korisnički definisanu) a najčešće se koristi ObjectId vrednost.

Pomenućemo ovde i par tipova indeksa koji će se pominjati u poglavlju 4:

- Indeks nad jednim poljem – korisnički definisan rastući ili opadajući indeks nad jednim poljem dokumenta.
- Kompozitni indeks – korisnički definisan indeks na više polja.
- Multikey indeks – koriste se za indeksiranje sadržaja nizova. Ovo omogućava upitima da nađu dokumente koji sadrže nizove tako što će se poklopiti s jednim ili više elemenata niza.
- Geoprostorni indeks – indeksi kreirani za podršku geoprostornim informacijama – koriste planarnu ili sfernu geometriju pri vraćanju rezultata.

Kada upit sadrži samo indeksirana polja, MongoDB vraća rezultate direktno iz indeksa bez skeniranja dokumenata ili njihovog uvođenja u memoriju. Ovo su tzv. pokriveni upiti i mogu biti jako efikasni. Primer pokrivenog upita biće dat u poglavlju 4.2.3.

4. Optimizacija kod MongoDB baze podataka

4.1. Optimajzer, generisanje i keširanje planova

Optimajzer upita predstavlja proces koji generiše planove za upite. Za svaki upit, optimajzer generiše plan koji povezuje upit s indeksom i koji najoptimalnije vraća rezultate. Cilj je svakako naći najefikasniji plan za upit, plan s najmanjom cenom.

MongoDB optimajzer koristi empirijsku metodu: ako nema keširanog plana, svi mogući planovi izvršenja na osnovu dostupnih indeksa se kreiraju. 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[1], umesto ponovnog proveravanja planova kandidata. Keširani plan se evaluira i ukoliko u nekom momentu njegove performanse padnu ispod određenog praga, on će biti izbačen iz keša i ponovo će se proći kroz fazu testiranja planova kandidata.

Kada je reč o ocenjivanju 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. Da bi se videle informacije o planu za dat upit, može se koristiti *db.collection.explain()* ili *cursor.explain().*

Počev od verzije 4.2, koja je trenutno aktuelna, za planove u kešu se vezuju i 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 izračunao je cenu (vrednost *works*) i rezervisao je mesto, ali se oblik upita ne koristi za generisanje planova upita. Ukoliko je ovo stanje u pitanju, treba evaluirati planove kandidate i odabrati najbolji plan. Sada 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. Treba napomenuti da, ukoliko treba zameniti planove, a pre nego što se zamena izvrši, neaktivni unos promeni stanje u aktivan (do čega može doći zbog 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

[1] Oblik upita (*eng. Query Shape*) – kombinacija strukture predikata upita, sortiranja i projekcije.

momentu njegova *works* vrednost ne zadovoljava definisane kriterijume, plan će preći u neaktivno stanje, i doći će do ponovnog planiranja.

Keš planova za upite ne perzistira ako se server restartuje ili ugasi. Korisnici takođe mogu ručno očistiti keš metodom *PlanCache.clear()* ili *PlanCache.clearPlansByQuery()*.

Kako bi se identifikovali spori upiti istog oblika, počev od verzije 4.2, svaki oblik upita povezan je sa jednom heš vrednošću – *queryHash*. To je heširana vrednost oblika upita i zavisi samo od njega. Kao i sa bilo kojom heš funkcijom, dva različita oblika upita mogu dati istu vrednost sažetka – međutim, nije verovatna kolizija između različitih oblika upita.

Za pružanje boljeg uvida u keširane planove, verzija 4.2 uvodi *planCacheKey*. To je heš ključa za keširani plan vezan za neki upit. Iako ovo zvuči isto kao *queryHash, planCacheKey* je funkcija koja u obzir uzima i oblik upita i trenutno postojeće indekse za taj oblik. To znači da će se njegova vrednost promeniti ukoliko se neki indeksi za taj oblik upita dodaju ili uklone, dok će u istom slučaju vrednost za *queryHash* ostati nepromenjena.

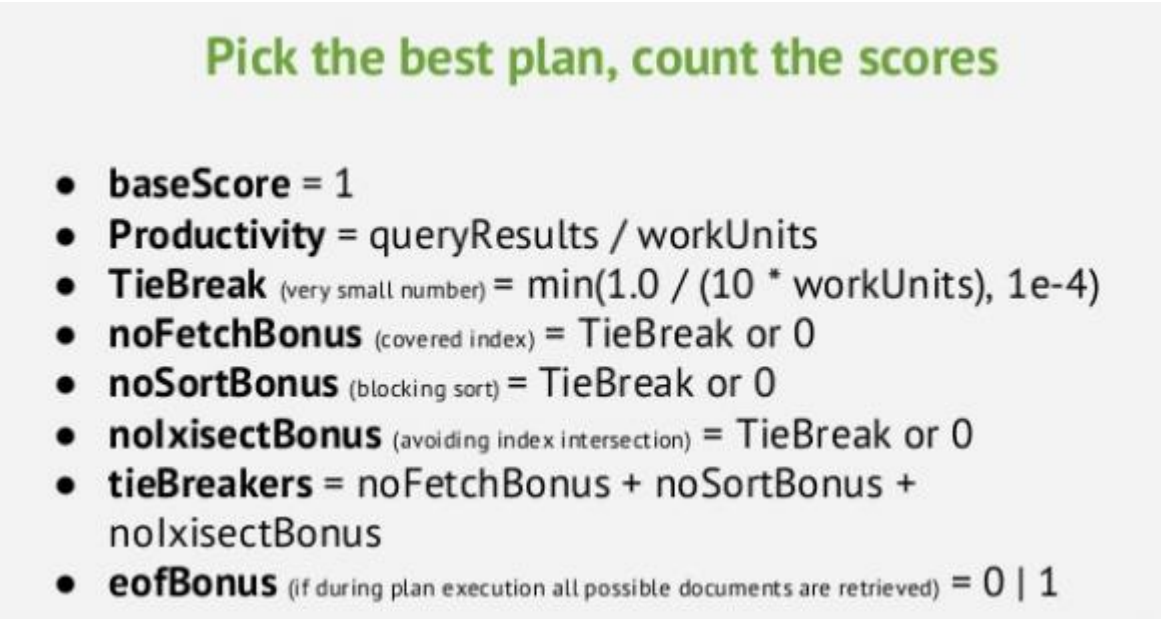
Kao što je rečeno, optimajzer je kreiran tako da krajnji korisnici ne mogu da utiču na njegov rad. Međutim, MongoDB nudi opcije za upravljanje njegovim radom. Jedno je *hint()* metoda koja forsira upotrebu definisanog indeksa za dati upit. S druge strane, postoje i filteri indeksa - ako za dati oblik upita postoji filter indeksa, optimajzer razmatra samo indekse specificirane u tom filteru. Kada za oblik upita postoji filter indeksa, MongoDB će ignorisati *hint()* metodu. Da bi se videlo da li je MongoDB primenio filter, treba proveriti polje *indexFilterSet()* u rezultatu metode *db.collection.explain()*. Filteri utiču na optimajzer, ali samo na indekse koje on evaluira – on i dalje može odabrati skeniranje cele kolekcije kao najbolji plan za dati oblik upita. Filteri postoje samo dok je server aktivan i s njegovim gašenjem se gube. Treba ih koristiti pažljivo, budući da na neki način nadjačavaju očekivano ponašanje optimajzera i *hint()* metodu. Filteri se mogu podesiti tipom naredbe prikazanoj na slici 1. Filter se primenjuje nad kolekcijom, ali samo za navedeni oblik upita – koga kao parametre u komandi definišu *query, sort* i *projection.*

```
db.runCommand(
   {
      planCacheSetFilter: <collection>,
      query: <query>,
      sort: <sort>,
      projection: <projection>,
      indexes: [ <index1>, <index2>, ...]
   }
)
```

Slika 1. Kreiranje filtera indeksa.

U zvaničnoj dokumentaciji za MongoDB, pojam *works* se pominje kao glavna stavka oko procenjivanja najboljeg plana za izvršenje upita. Pored toga, postoje i stavke *advanced* i *needTime* koje su usko povezane sa *works.* One će biti bliže objašnjene u poglavlju 4.3.1, pri pominjanju

parametara koje vraća metoda *explain()*. Na slici 2 dato je par faktora koji utiču na skor plana i odabir istog, a koje razmatraju i upotrebu pokrivenih upita, sortiranja, intersekcija indeksa, itd. Skor se dobija kao zbir stavki *baseScore, productivity, tieBreakers* i *eofBonus.*



Slika 2. Računanje skora za plan.

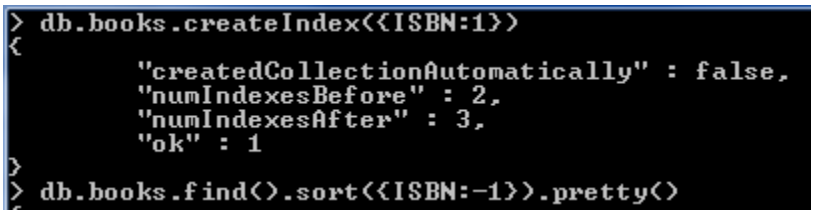
4.2. Načini optimizacije upita

4.2.1. Upotreba indeksa

Indeksi poboljšavaju efikasnost operacija čitanja tako što smanjuju količinu podataka koja se treba obraditi. Ovo pojednostavljuje proces izvršavanja upita u MongoDB-u. Ukoliko su u aplikaciji prisutni upiti vezani za određeno polje ili skup polja, tada bi indeks za to polje ili kompozitni indeks za skup polja mogao da spreči skeniranje čitave kolekcije za rezultate.

Pored optimizacije operacija čitanja, indeksi pomažu i kod operacija sortiranja i omogućavaju efikasnije iskorišćavanje memorije. Treba reći da je odabir rastućeg ili opadajućeg uređenja za indekse nad jednim poljem nebitno, dok je kod kompozitnih indeksa ovo jako bitna stavka.

Skeniranje indeksa je mnogo brže od skeniranja kolekcije. Strukture indeksa su manje od dokumenata i uređene su. Indeksi poboljšavaju i efikasnost upita koje se rutinski sortiraju na datom polju. Indeksi podržavaju upite, operacije modifikacije i neke faze agregacionog pajplajna. Na slici 3 dat je primer kreiranja indeksa za polje na kome se rutinski sortira upit. Naredba *pretty()* korisitće se i u narednim primerima kako bi se rezultati prikazali u čitljivijoj formi.



Slika 3. Kreiranje indeksa po kome se sortira upit.

Svaki indeks u kolekciji dodaće malo opterećenja performansama upisa. Za svaku insert ili delete operaciju upisa u kolekciji, MongoDB ili unosi ili uklanja odgovarajući ključ za dokument iz svakog indeksa posmatrane kolekcije. Update operacija može rezultovati u modifikaciji indeksa kolekcije, u zavisnosti od ključa na koji se operacijom utiče.

U suštini, performanse koje indeksi donose u operacijama čitanja značajnije su od toga što se dešava pri operacijama pisanja. Svakako, kako bi se performanse bile što bolje, treba pažljivo kreirati nove indekse i evaluirati postojeće kako bi se osiguralo da ih upiti zaista i koriste.

MongoDB ne može kombinovati više indeksa – za dati upit, koristi se samo jedan indeks. Upravo zbog toga cilj treba biti kreiranje dobro promišljenih kompozitnih indeksa. Određivanje grupe polja koja se treba zajedno indeksirati svodi se na razumevanje upita koji se izvršavaju.

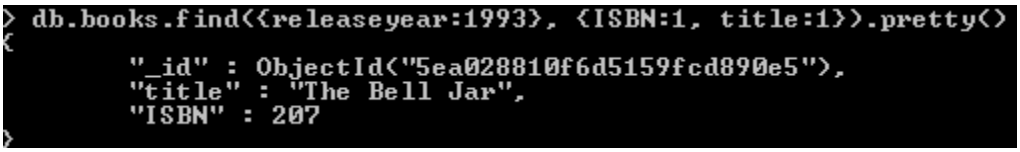
4.2.2. Ograničiti broj rezultata

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. Ovo se uglavnom koristi u kombinaciji sa *sort* operacijom. Na slici 4 dat je primer koji ograničava rezultate na 5 dokumenata iz kolekcije knjiga.

Slika 4. Upotreba metode limit().

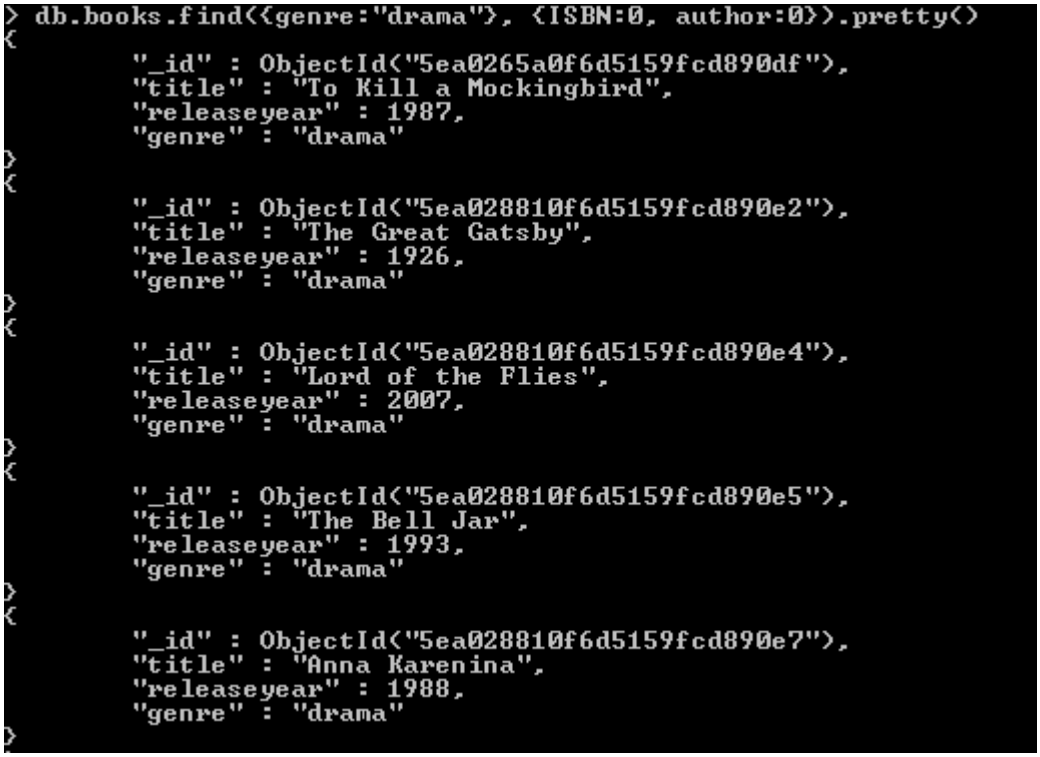
4.2.3. Upotreba projekcija

Kada je neophodan samo podskup polja 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. U primeru na slici 5, upit će vratiti sve dokumente koji se poklapaju sa uslovom *releaseyear*:1993, ali će skup rezultata sadržati samo polja *ISBN*, *title* i *_id*, kao podrazumevano polje. Ono se može ukloniti iz rezultata tako što će se u projekciji setovati na nulu (_id:0).



Slika 5. Korišćenje projekcije s uključivanjem polja.

Umesto navođenja polja koja će rezultat vratiti, projekcija može i isključiti specifična polja, kao u primeru 6, gde će rezultat vratiti nađene dokumente sa svim poljima sem polja *ISBN* i *author*. Treba napomenuti da se ne mogu kombinovati isključenja i uključenja polja, sem za *_id* polje.



Slika 6. Isključivanje polja projekcijom.

Mogu se uključiti ili isključiti i specifična polja ugneždenih dokumenata. Za polja koja sadrže nizove, MongoDB pruža specijalne projekcione operatore: *$elemMatch, $slice* i *$.* Ovi operatori su jedini način da određeni elementi iz niza vrate u okviru rezultata, ne može se npr. napisati indeks elementa u nizu.

4.2.4. Pokriveni upiti

Pokriven upit (*eng. Covered query*) je upit koji se u potpunosti može ispuniti korišćenjem indeksa i ne mora da pretražuje dokumente. Kaže se da indeks pokriva upit kada važe sledeće stavke: sva polja u upitu su deo indeksa i sva polja koja rezultat vraća su u istom indeksu. Pored toga, nijedno polje u upitu nije jednako *null*. Na slici 7 kreiran je indeks nad poljima *title* i *genre*, nakon čega je izvršen upit za ta polja, pri čemu je prikazano samo polje *title.* Kako bi specificirani indeks pokrio upit, projekcija mora eksplicitno označiti *_id: 0* kako bi isključio to polje iz rezultata, jer ono nije uključeno u kreirani indeks.

```
> db.books.createIndex({title:1, genre:1})
```

```
> db.books.find({title:"1984", genre:"dystopian"}, {title:1, _id:0}).pretty()
{ "title" : "1984" }
```

Slika 7. Primer pokrivenog upita.

Od verzije 3.6, indeks može da pokriva upit na poljima unutar ugrađenih (*eng. embedded*) dokumenata. Pošto indeks sadrži sva polja koja upit zahteva, MongoDB ispunjava uslove upita i vraća rezultate samo korišćenjem indeksa, a ovo je uglavnom dosta brže od pretraživanja dokumenata van indeksa.

Postoje i određena ograničenja kod pokrivanja upita: geoprostorni indeksi ne mogu pokriti upit. Multikey indeksi ne mogu pokriti upite sa nizom polja. Postoje i restrikcije vezane za sharded kolekcije.

4.2.5. Upotreba *hint*-a

Najčešće će optimajzer odabrati optimalni indeks za neku operaciju. Međutim, MongoDB se može forsirati da upotrebi određeni indeks korišćenjem *hint()* metode. Koristi se kao podrška testiranju performansi, ili na nekim upitima gde se mora selektovati polje uključeno u više indeksa. Moguće je i forsiranje skeniranja kolekcije, tako što se *hint()* metodi prosledi *$natural:1*. Na slici 8 prikazana je upotreba hint metode kojom se forsira upotreba indeksa nad poljem *ISBN*.

```
> db.books.find({ISBN:207, releaseyear:1993}).hint({ISBN:1}).pretty()
{
        "_id" : ObjectId("5ea028810f6d5159fcd890e5"),
        "title" : "The Bell Jar",
        "author" : {
                "name" : "Sylvia",
                "surname" : "Plath"
        },
        "ISBN" : 207,
        "releaseyear" : 1993,
        "genre" : "drama"
}
>
```

Slika 8. Upotreba hint metode.

4.2.6. Upotreba operatora inkrementiranja

Upotreba *$inc* operatora za inkrementiranje ili dekrementiranje vrednosti u dokumentima je još jedna opcija. 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, ručno inkrementiraju polje i sačuvaju ceo dokument u isto vreme.

4.2.7. Selektivnost upita

Ovaj pojam se odnosi na to koliko dobro upit isključuje i filtrira dokumente iz kolekcije. Ovako se može utvrditi da li upiti mogu koristiti indekse efikasno. Selektivniji upiti odnose se na manji procenat dokumenata – tako je podudaranje na *_id* polju visoko selektivno jer se poklapa samo s jednim dokumentom. Manje selektivni upiti ne mogu koristiti indekse efikasno ili ih ne mogu koristiti uopšte. Na primer, operatori nejednakosti *$nin* i *$ne* nisu preterano selektivni jer se najčešće podudaraju sa većim delom indeksa. Kao rezultat se često dolazi do zaključka da ovakvi upiti s indeksom ne funkcionišu ništa bolje od upita s operatorima nejednakosti koji su lišeni indeksa, tj. moraju skenirati sve dokumente u kolekciji.

4.3. Pregled performansi

4.3.1. Metoda *explain*

Za pregled informacija o planovima upita i statistici izvršenja planova, MongoDB pruža *explain()* metodu. Rezultati predstavljaju planove upita kao stablo faza, kao što je i rečeno u poglavlju 2. Svaka faza prosleđuje svoje rezultate (dokumente ili ključeve indeksa) roditeljskom čvoru. 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/dobija rezultujući skup.

Neke od mogućih faza u stablu plana su:

- COLLSCAN – podrazumeva skeniranje cele kolekcije.
- IXSCAN – podrazumeva skeniranje ključeva indeksa.
- FETCH – vraćanje rezultata.
- SHARD_MERGE – skupljanje rezultata iz delova (shardova).
- SHARDING_FILTER – filtriranje dokumenata iz shardova.

Metoda *explain()* ima tri različita parametra koja određuju opširnost informacija koje će biti obuhvaćene u rezultatu: *queryPlanner, executionStats, allPlansExecution.* Podrazumevani

parametar je *queryPlanner* i on prikazuje detalje plana koji je optimajzer odabrao i daje listu odbijenih planova. S druge strane, *executionStats* donosi informacije i detalje o izvršenju najboljeg plana. Za operacije upisa, izvršenje upita se odnosi na ono što bi bilo izvršeno, ali se modifikacije ne primenjuju na bazu podataka. Na kraju, *allPlansExecution* sadrži parcijalne informacije o izvršenju koje su zabeležene tokom faze biranja plana, za najbolji ali i za odbijene planove.

Pomenućemo samo neka od bitnih polja koje rezultat metode *explain()* vraća a koje ćemo razmatrati u kasnijem primeru.

- explain.queryPlanner.indexFilterSet – boolean koji definiše da li je primenjen filter indeksa za taj oblik upita. Na ovaj način proveravamo da li je filter iskorišćen, ukoliko on za taj oblik upita postoji.
- explain.queryPlanner.queryHash – heksadecimalni string koji je heš oblika upita i zavisi samo od njega. Može da pomogne u pronalaženju sporih upita s istim oblikom.
- explain.queryPlanner.planCacheKey – heš ključa za unos keširanog plana povezanog s upitom. Ovaj i prethodni parametar već su detaljnije pominjani u poglavlju 4.1.
- explain.queryPlanner.winningPlan – dokument s detaljima najefikasnijeg plana koji je odabrao optimajzer. MongoDB prikazuje plan kao stablo faza: faza može imati jednu ili više ulaznih faza. Struktura ovog parametra u okviru rezultata prikazana je na slici 9. Parametar *stage* se odnosi na naziv faze i svaka faza sadrži specifične informacije – na primer, IXSCAN faza će uključivati granice indeksa, zajedno s drugim podacima koji su bitni za skeniranje indeksa. Ukoliko faza ima faze potomke, postojaće i parametri *inputStage/inputStages*.

```
"winningPlan" : {
    "stage" : <STAGE1>,
    ...
    "inputStage" : {
        "stage" : <STAGE2>,
        ...
        "inputStage" : {
            ...
        }
    }
},
```

Slika 9. Struktura parametra winningPlan.

- explain.queryPlanner.rejectedPlans – Niz planova kandidata koji su razmatrani i odbijeni od strane optimajzera. Niz može biti i prazan ako nije bilo drugih kandidata planova.
- explain.executionStats.nReturned – broj dokumenata koji zadovoljvaju uslove upita.
- explain.executionStats.executionTimeMilis – vreme u ms za biranje i izvršenje plana.
- explain.executionStats.totalKeysExamined – broj unosa indeksa koji je skeniran.

- explain.executionStats.totalDocsExamined – broj skeniranih dokumenata. Ovo se odnosi na broj ispitanih, ne na broj vraćenih dokumenata. Ukoliko je neki dokument ispitan više puta, to će se računati i uticaće na ovaj broj.
- explain. executionStats.executionStages – detalji izvršenja najboljeg plana, dati u vidu niza izvršenih faza.
- explain. executionStats.executionStages.inputStage.docsExamined – broj dokumenata skeniran tokom faze izvršenja upita. Prisutan u COLLSCAN fazi i u fazama koje pribavljaju dokumente iz kolekcije, npr. FETCH.

Sada je bitno pomenuti detaljnije metriku koja se posmatra za biranje najefikasnijeg plana. One se vezuju za priču iz poglavlja 4.1, kada je razmatrano generisanje i keširanje planova.

- explain. executionStats.executionStages.works – broj work unita koje izvršava ova faza. Posao se deli na manje jedinice – to može biti ispitivanje jedinstvenog ključa indeksa, pribavljanje dokumenta iz kolekcije, primena projekcije na dokument, itd.
- explain. executionStats.executionStages.advanced – broj međurezultata koji je vraćen i koji je napredovao do roditeljske faze. Ovo znači da, ukoliko plan može da obezbedi zahtevani dokument, odgovara sa *advanced*.
- explain. executionStats.executionStages.needTime – broj radnih ciklusa koji nisu poslali međurezultate roditeljskoj fazi.
- explain. executionStats.executionStages.isEOF – definiše da li je faza izvršenja došla do kraja toka. Ako je vrednost 1, došlo se do kraja toka i svi očekivani rezultati su pribavljeni. Ako je vrednost 0, moguće je da treba vratiti još rezultata.

Treba napomenuti par stvari. Kada indeks pokriva upit, rezultat metode *explain()* imaće IXSCAN fazu koja nije potomak FETCH faze, a broj ukupno ispitanih dokumenata (executionStats.totalDocsExamined) biće jednak nuli. Ako MongoDB koristi indekse za $*or* izraz, rezultat će uključivati OR fazu (koja nije navedena u osnovnim fazama) sa nizom ulaznih faza s detaljima o indeksima (ima slika). Ako MongoDB može da koristi indeks da dobije zahtevano sortiranje, rezultat neće uključivati SORT fazu. U suprotnom, ako se indeks ne može koristiti, postojaće SORT faza.

4.3.2. *Database Profiler*

MongoDB obezbeđuje profajler koji pokazuje karakteristike performansi svake operacije. 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 za kreiranje. Profajler skuplja informacije o komandama koje se izvršavaju na mongod instanci. Ovo uključuje CRUD operacije, kao i konfiguracione i administrativne komande. On je podrazumevano isključen, a može se uključiti na nivou baze ili instance, na više različitih nivoa.

Profajler nudi tri različita nivoa profajliranja:

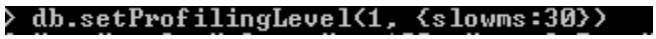
- 0 – isključen profajler, ne skuplja nikakve podatke. Ovo je podrazumevani nivo.

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

Za uključivanje profajlera, koristi se naredba *db.setProfilingLevel(<nivo>)* gde se navodi nivo 1 ili 2. Kao rezultat, vratiće se dokument koji pokazuje prethodni nivo profajliranja, vrednost praga *slowms* i potvrdu da je novi nivo podešen, što se može videti na slici 10.

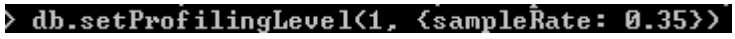
Slika 10. Uključivanje profajlera.

Podrazumevani prag sporih operacija, *slowms*, je 100ms. To se može promeniti na više načina, a jedan od njih je i specificiranje praga pri podešavanju nivoa profajliranja, kao što je prikazano na slici 11. Treba napomenuti da se nivo profajliranja može podesiti za određenu bazu ili instancu MongoDB-a, ali se prag uvek primenjuje na sve baze mongo instance. Koristiće ga i profajler i loger za dijagnostiku i pri podešavanju treba biti podešen na najvišu upotrebljivu vrednost, kako bi se izbegle degradacije performansi.



Slika 11. Podešavanje parametra slowms.

Ukoliko želimo da samo jedan nasumični deo sporih operacija bude profajliran, parametar *sampleRate* treba podesiti na vrednost između 0 i 1. On je podrazumevano 1, što znači da će obuhvatiti ceo skup sporih operacija pri profajliranju. Ukoliko želimo da samo 35% sporih operacija bude obuhvaćeno, može se zadati naredba kao na slici 12:



Slika 12. Podešavanje parametra sampleRate.

Treba napomenuti da uključivanje profajlera ima uticaja na upotrebu diska i performanse, posebno kada je uključen nivo 2 ili nivo 1 sa niskim pragom *slowms*. Zbog toga bi ga trebalo isključiti ukoliko se ne nastavlja s njegovim korišćenjem.

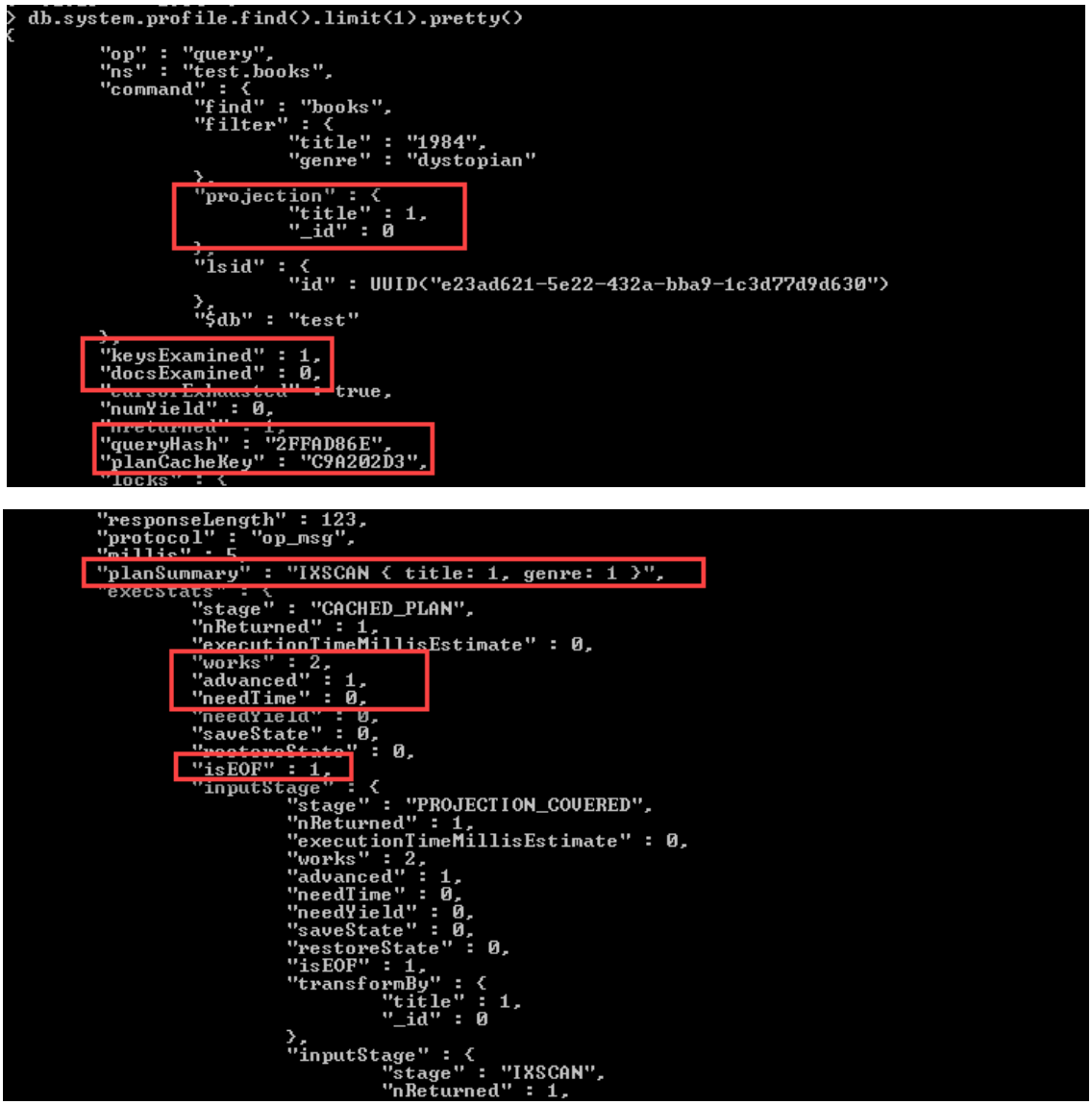
Profajler upisuje podatke u kolekciju *system.profile*, koja je ograničene veličine – podrazumevana je 1MB. Ukoliko je to neophodno, veličina se može i ručno podesiti. Za dobijanje zapisanih informacija, treba sprovesti upit nad ovom kolekcijom.

Na slici 13 dati su samo neki od upita koji se mogu sprovesti nad *system.profile* kolekcijom. Prvi upit će vratiti 5 najsporijih upita iz kolekcije, a drugi vraća sve upite kojima treba više od 30 sekundi za izvršenje, tj. one upite koji prelaze prethodno postavljenu vrednost parametra *slowms*.

Slika 13. Upiti nad kolekcijom system.profile

Slika 14 prikazuje deo jednog od dokumenata iz kolekcije *system.profile*. Bitno je naglasiti par stvari: dokument definiše za koji je upit izvršen plan, koliko je dokumenata pretraženo, koje su

faze plana i da li je keširan ili nije. Konkretno, ovo je upit sa slike 7. Bitne stavke su uokvirene na slici. Profajler ima slične stavke u dokumentima kao i rezultati koje vraća *explain* metoda. Sa slike se vidi da postoji projekcija, da je broj ispitanih dokumenata 0 – što znači da se nije radilo skeniranje kolekcije, već indeksa. To će potvrditi i faza u parametru *planSummary* – IXSCAN – indeks nad poljima *title* i *genre.* Takođe vidimo i metrike *works, advanced, needTime* i *isEOF.*



Slika 14. Deo dokumenta iz kolekcije system.profile.

4.3.3. Analiza performansi

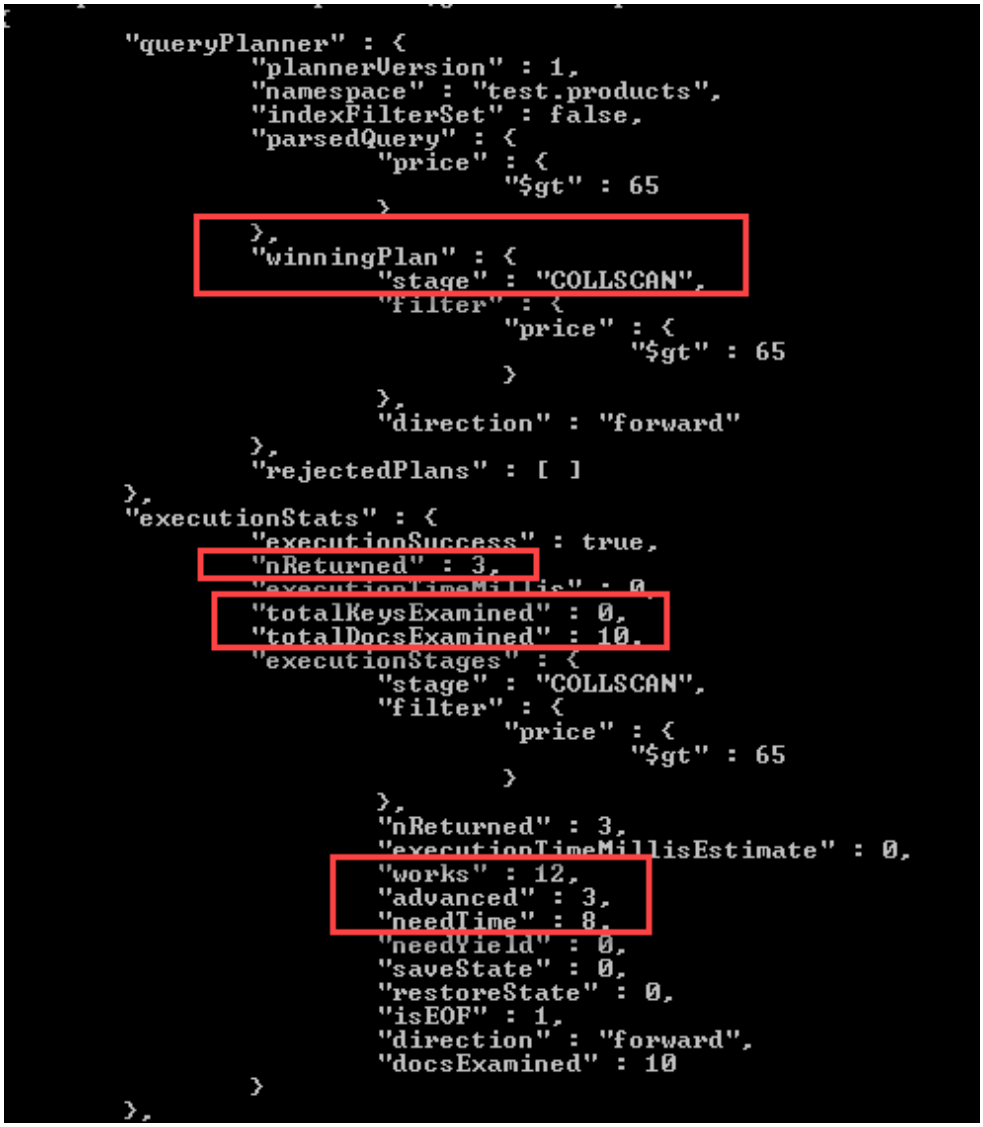
U poglavlju 4.2 pomenuto je nekoliko tehnika koje se koriste kako bi se upiti optimizovali. Ipak, treba to i dokazati – upravo zato su u poglavljima 4.3.1. i 4.3.2. objašnjeni *explain()* i profajler kao moćni alati koji ne samo da pružaju statistiku, već i olakšavaju uočavanje upita koji se dalje mogu optimizovati.

Razmotrićemo rezultat metode *explain()* nad upitom bez upotrebe indeksa i sa kreiranim indeksom. Nad kolekcijom jednostavnih proizvoda obavićemo upit sa slike 15, uz dodavanje metode *explain* s parametrom *executionStats.*

```
> db.products.find({price:{$gt:65}}).explain("executionStats")
```

Slika 15. Upit nad kolekcijom products uz metodu explain.

Rezultati su dati na slici 16. Prikazan je deo dokumenta (bez *serverinfo* infrormacija). Uočava se da za *winningPlan* kao faza stoji COLLSCAN, što znači da je odrađeno skeniranje cele kolekcije, tj. skeniranje 10 dokumenata koliko je u kolekciji (što potvrđuje i *totalDocsExamined*). Broj dokumenata koji bi se upitom vratili kao rezultat je 3 – *nReturned.* Takođe se uočava da nije bilo ispitanih indeksa (*totalKeysExamined: 0*), a da je vrednost parametra *works 12*.



Slika 16. Dobijeni rezultat explain metode.

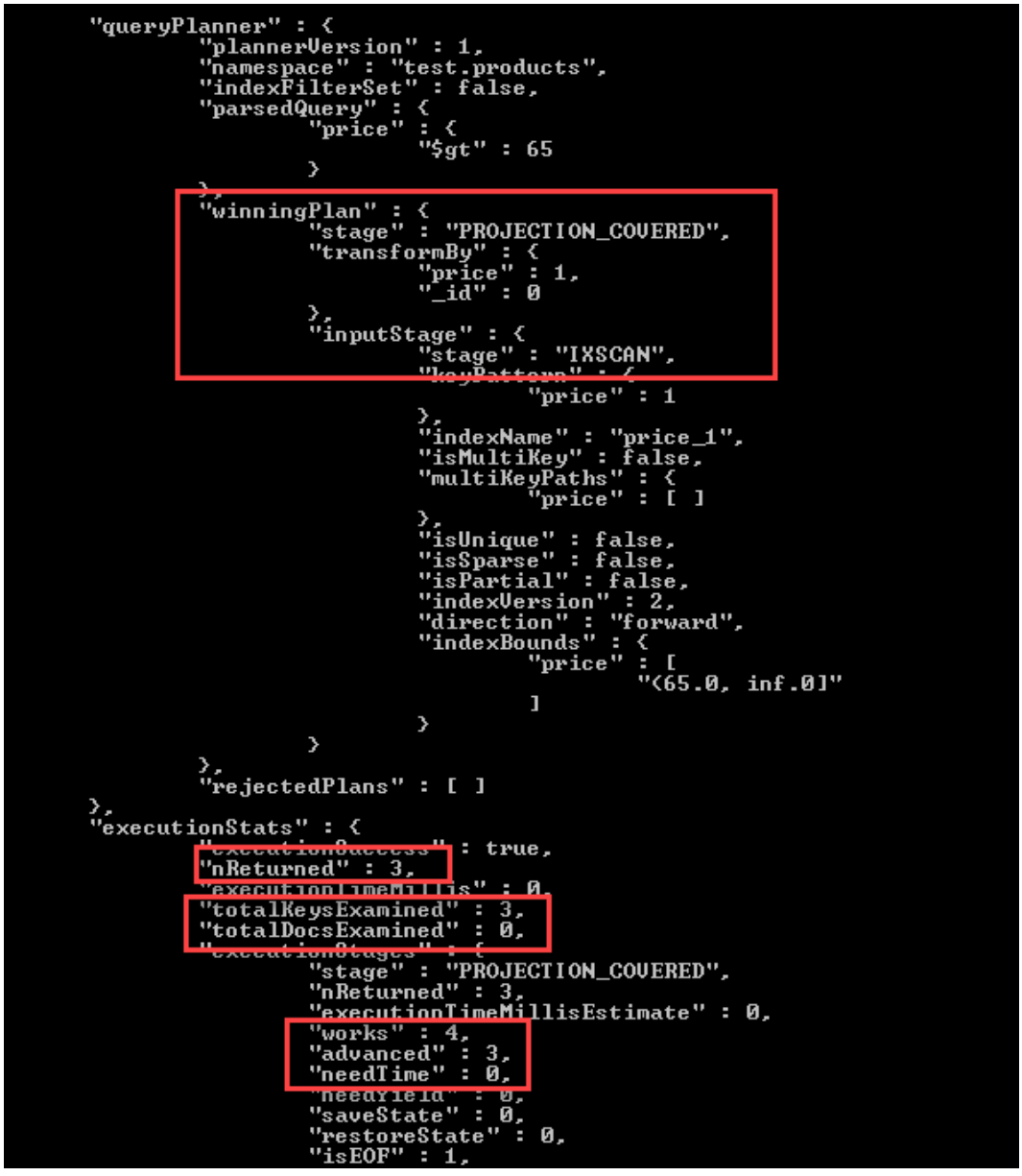
Razlika između broja skeniranih i broja vraćenih dokumenata (što su u ovom slučaju brojevi 10 i 3) može sugerisati da bi se poboljšala efikasnost upotrebom indeksa. Upravo to ćemo isprobati, kreiranjem indeksa nad poljem *price*. Deo rezultata je prikazan na slici 17. Ovog puta, *winningPlan* prikazuje fazu FETCH sa ugneždenom, ulaznom fazom IXSCAN, što znači da je indeks iskorišćen. Pored toga, broj skeniranih dokumenata je sa 10 smanjen na 3, koliki je i broj skeniranih indeksa. Vrednost parametra *works* smanjen je sa 12 na 4, što jeste napredak. Naravno, broj vraćenih dokumenata (*nReturned*) ostao je isti.

Slika 17. Dobijeni rezultat explain metode uz korišćenje indeksa.

Poslednja stavka koju ćemo isprobati na ovom primeru je testiranje pokrivenog upita. Slika 18 pokazuje izvršeni upit: uključivanjem samo polja *price* u rezultate, a isključivanjem polja *_id* dobili smo pokriven upit. Na slici 19 se nalazi deo rezultata. Prvo što treba uočiti je da je broj skeniranih dokumenata 0 – ovog puta, nema potrebe skenirati ih jer je ceo upit pokriven i radi se samo s indeksom. Pored toga, kao što je već rečeno u poglavlju 4.3.1., kada se radi o pokrivenom upitu, faza IXSCAN neće biti potomak FETCH faze: ovde se vidi da je aktivna faza za *winningPlan* PROJECTION_COVERED. Naravno, broj dokumenata u rezultatu ostaje isti – 3.

```
> db.products.find({price: {$gt: 65}}, {price:1, _id:0}).explain("executionStats")
```

Slika 18. Testiranje pokrivenog upita.



Slika 19. Rezultati metode explain nad pokrivenim upitom.

Ovde se mogu upoređivati i efikasnosti kompozitnih indeksa. Ako, na primer, kreiramo dva kompozitna indeksa na način kao na slici 20, možemo upoređivati njihove performanse tako što ćemo forsirati optimajzer da koristi neki od ta dva, upotrebom metode *hint()*.

```
> db.supermarket.createIndex({quantity:1, type:1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
> db.supermarket.createIndex({type:1, quantity:1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "ok" : 1
}
```

Slika 20. Kreiranje kompozitnih indeksa.

Na slikama 21 i 22 dati su rezultati upita koji su forsirali upotrebu ova dva indeksa, respektivno. Sada su prikazani samo delovi koje želimo da uporedimo, a to je broj skeniranih i vraćenih dokumenata, budući da će u oba slučaja pobednički plan skenirati indeks. Vidi se da je drugi kreirani kompozitni indeks, sa redosledom polja *type-quantity* efikasniji, jer skenira manje ključeva indeksa.

```
"executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 2,
        "executionTimeMillis" : 24,
        "totalKeysExamined" : 6,
        "totalDocsExamined" : 2,
        "executionStages" : {
                "stage" : "FETCH",
                "nReturned" : 2,
                "executionTimeMillisEstimate" : 25,
                "works" : 6,
                "advanced" : 2,
                "needTime" : 3,
```

```
"executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 2,
        "executionTimeMillis" : 23,
        "totalKeysExamined" : 2,
        "totalDocsExamined" : 2,
        "executionStages" : {
                "stage" : "FETCH",
                "nReturned" : 2,
                "executionTimeMillisEstimate" : 24,
                "works" : 3,
                "advanced" : 2,
                "needTime" : 0,
```

Slike 21 i 22. Rezultati upotrebe kompozitnih indeksa.

Upravo zbog ovakvih stvari je korisno upotrebljavati neku od metoda za posmatranje performansi, kako bi se utvrdilo šta se može poboljšati, ali i gde treba promeniti pristup.

5. Zaključak

Optimizacija je neizostavan deo bilo koje baze podataka, i ukoliko se neko dublje bavi određenom bazom, neophodno je znati korake i način optimizacije kako bi se sve ono što sistem pruža moglo iskoristiti na najbolji način. MongoDB dozvoljava generisanje jako kompleksnih upita, a to u kombinaciji sa nepostojanjem predefinisane šeme koja se prati, omogućava veliku količinu eksperimentisanja u cilju dobijanja najboljeg modela podataka i najefikasnijih upita.

Kreiranje raznovrsnih indeksa je još jedna od odlika MongoDB-a, s tim što je neophodno znati kako njima upravljati da bi se postigli najbolji rezultati. Kao što je u prethodnom poglavlju prikazano, nije uvek dovoljno samo kreirati indeks – neophodno je i testirati ga i evaluirati njegove performanse ne bi li se utvrdilo pomaže li on zapravo i dovodi li do optimizacije.

U ovom radu prikazani su osnovni koncepti MongoDB baze podataka, s posebnom pažnjom usmerenom na optimizaciju upita. Prikazano je na koji način funkcioniše optimajzer i šta sve uzima u obzir pri biranju najefikasnijeg plana za izvršenje upita, predloženo je par načina za optimizaciju upita, a zatim su neki od njih i analizirani na osnovu parametara performansi koje MongoDB obebeđuje.

LITERATURA

[1] K. Chodorow, *MongoDB: The Definitive Guide*, O'Reilly Media, 2013.

[2] W. da Rocha Franca, *MongoDB Data Modelling*, Packt Publishing, 2015.

[3] A. Doan, A. Halevy, Z.Ives, *Principles of Data Integration*, Elsevier Inc, 2012.

[4] MongoDB dokumentacija: https://docs.mongodb.com/manual/core/query-optimization/

[5] Query Optimization, Wikipedia: https://en.wikipedia.org/wiki/Query_optimization

[6] Query Planner, Miguel Angel Nieto: https://www.slideshare.net/miguelangelnieto/query-planner-88498371

[7] Understanding Queries, openmymind.net: https://www.openmymind.net/Speedig-Up-Queries-Understanding-Query-Plans/