UNIVERZITET U NIŠU  
ELEKTRONSKI FAKULTET  
KATEDRA ZA RAČUNARSTVO

**OPTIMIZACIJA UPITA KOD MONGODB BAZE PODATAKA**

Student:  
Dimitrije Stamenković 1495

**Sadržaj:**

[Uvod 2](#_Toc1805981894)

[Uticaj hardvera na performanse 3](#_Toc1747587609)

[Indeksi 5](#_Toc2054089542)

[B stablo 5](#_Toc477924995)

[Indeksi nad jednim poljem 6](#_Toc442981385)

[Funkcija explain 7](#_Toc1120586805)

[Kompleksni indeksi 11](#_Toc2001155161)

[Prefiks indeksa 13](#_Toc1162661701)

[Sortiranje sa indeksiranjem 13](#_Toc2067936318)

[Indeksi nad poljima nizova 14](#_Toc195404718)

[Parcijalni indeksi 15](#_Toc1726527241)

[Različite strategije za optimizaciju upita 17](#_Toc1350295964)

[Projekcije 18](#_Toc1226873817)

[Limitiranje vraćenih dokumenata 19](#_Toc315692108)

[Zaključak 20](#_Toc1586126773)

[Literatura 21](#_Toc1488835051)

# Uvod

Optimizacija upita je proces pronalaženja najefikasnijeg načina izvršavanja upita na bazi podataka. Cilj optimizacije upita je da se smanji vreme potrebno za obradu upita, smanji potrošnja resursa i poboljša performanse baze podataka u celini. Proces optimizacije upita uključuje analizu upita i izbor najboljeg načina izvršavanja. To podrazumeva analizu strukture baze podataka, indeksa, statistike podataka, raspoloživih resursa sistema i drugih faktora koji mogu uticati na performanse.

Postoje različiti pristupi optimizaciji upita, kao što su heuristički pristupi, optimizacija zasnovana na pravilima, upitna optimizacija i optimizacija zasnovana na troškovima. Svaki pristup ima svoje prednosti i mane, a izbor zavisi od specifičnosti baze podataka, količine podataka, složenosti upita i drugih faktora. Uglavnom, optimizacija upita je ključni faktor u poboljšanju performansi baze podataka, a dobro optimizovani upiti omogućavaju brže izvršavanje i bolju iskorišćenost raspoloživih resursa sistema.

MongoDB je popularna baza podataka koja se razvija u open-source okruženju. Ova baza podataka koristi dokument-orijentisani pristup, što znači da se podaci čuvaju u obliku dokumenata, slično kao u JSON formatu. Dokumenti su organizovani u kolekcije, koje su slične tabelama u relacionim bazama podataka. MongoDB nudi visoku skalabilnost, performanse i fleksibilnost. Ti atributi čini je popularnom za različite vrste aplikacija. Ona podržava replikaciju i shardovanje, što omogućava lako skaliranje baze podataka. U radu ćemo se objasniti nekoliko ključnih poglavlja koja imaju veliki značaj za efikasnost MongoDB baze.

# Uticaj hardvera na performanse

Hardver kao faktor igra bitnu ulogu kada je reč o performansama baze podataka. MongoDB baza podataka može biti jako zahtevna za hardver, posebno za veće i kompleksnije aplikacije. Efikasnost i performanse MongoDB baze podataka direktno su povezane sa hardverskim resursima, uključujući procesor, RAM, disk i mrežu.

Neki od hardverskih faktora koji utiču na performanse su:

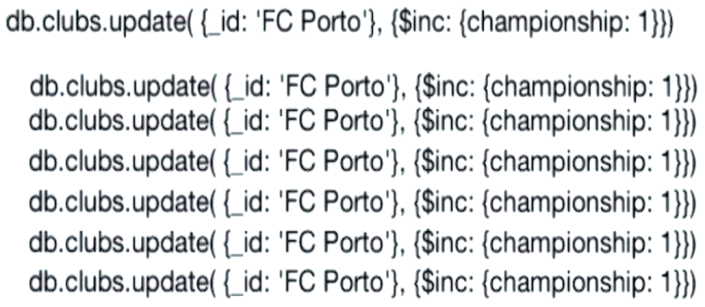
**Procesor:** Procesor ima veliki uticaj na performanse MongoDB baze podataka, posebno na velikim opterećenjima. MongoDB se oslanja na procesorsku snagu za obradu velikog broja zahteva u kratkom vremenu. Preporučuju se procesori sa visokom brzinom takta i sa više jezgara.

**RAM**: RAM je takođe važan faktor za performanse MongoDB baze podataka, jer se koristi za keširanje podataka. Više RAM-a omogućava MongoDB-u da kešira više podataka, što dovodi do bržeg pristupa podacima i poboljšane efikasnosti.

**Disk**: Brzina diska je takođe važan faktor za performanse MongoDB baze podataka, jer se MongoDB oslanja na brzi pristup podacima sa diska. Solid State Drive (SSD) je obično bolji izbor od klasičnog Hard Disk Drive (HDD) zbog bržeg pristupa podacima.

**Mreža**: Kvalitet mreže može imati uticaj na performanse MongoDB baze podataka, posebno u raspoređenim okruženjima. Preporučuju se mrežni adapteri sa visokom brzinom i niskom latencijom za poboljšane performanse.

Povećavanje broja jezgara CPU-a neće uvek dovesti do povećanja performanski same baze. Na primeru datom na slici (Slika 1: Više uzastopnih operacija), ukoliko se radi o više uzastopnih upisa nad istim dokumentom doći se do zaključavanja tog dokumenta pa će se tako te operacije izvršiti u istoj niti.

Slika 1: Više uzastopnih operacija

Tip hard diska isto može imati veliki uticaj na performanse baze, jedinica koje se koja se koristi za meru je IOPS ( broj ulazno-izlaznih operacija u sekundi) i praktično što veći IOPS diska to je baza brža. U sledećoj tabeli vidimo kolika može biti razlika u brzinama različitih tipova diska, u ekstremnim slučajevima razlika može ići i do 100x .

|  |  |
| --- | --- |
| Tip | IOPS |
| 7200 rpm SATA | 75-100 |
| 15000 rpm SAS | 175-210 |
| SSD Intel X25-E (SLC) | 5000 |
| SSD Intel X25-M G2 (MLC) | 8000 |
| Amazon EBS | 100 |
| Amazon EBS Provisioned | 3000 |
| FusionIO | 135000 |
| Violin Memory 6000 | 1 000 000 |

Tabela 1: IOPS različitih tipova diskova

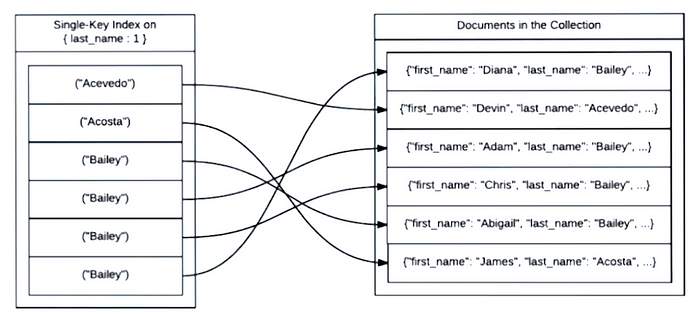
Kada je reč o hard diskova važno je pomenuti i RAID arhitekturu. RAID (*Redundant Array of Independent Disks*) je tehnologija koja se koristi za poboljšanje performansi i pouzdanosti baze podataka. U MongoDB bazi, RAID arhitektura se može koristiti na različite načine u zavisnosti od ciljeva performansi i pouzdanosti. Postoje različite vrste RAID arhitektura, kao što su RAID 0, RAID 1, RAID 5, RAID 6 i RAID 10. U MongoDB bazi, najčešće se koristi RAID 10 koji kombinuje RAID 1 i RAID 0.

Slika 2: RAID 10 arhitektura

RAID 1 je tehnologija koja koristi dupliranje podataka na dva različita diska. Ovo poboljšava pouzdanost sistema jer ako jedan disk otkaže, drugi disk sadrži identične podatke i samim tim sistem i dalje nastavlja da funkcioniše. RAID 0, s druge strane, koristi deljenje (*striping*) podataka, što znači da se podaci dele između više diskova. Ovo poboljšava performanse sistema jer se podaci mogu čitati i pisati sa više diskova istovremeno.

# Indeksi

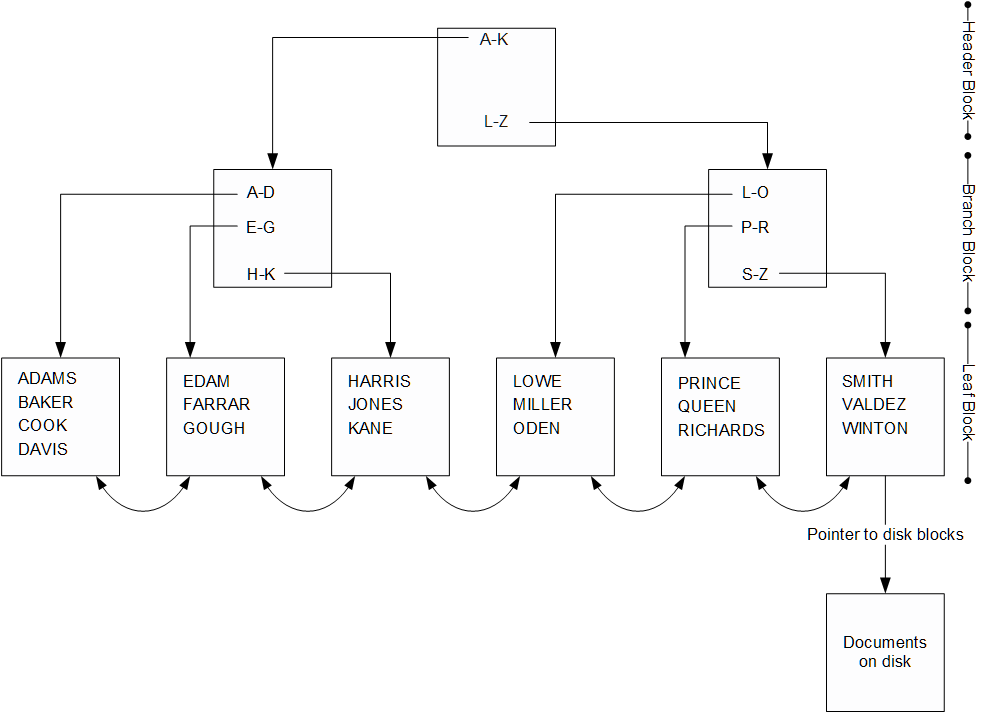
Indeks u MongoDB bazi predstavlja mehanizam koji omogućava znatno brže operacije (pretraga,sortiranje i filtriranje) nad kolekcijom. Indeksiranje radi po principut B stabla, tako da kao rezultat možemo direktno locirati podatak preko jednog ili više polja O(1) umesto skeniranja cele kolekcija gde bi optimalnost bila reda O(n).

Slika 3: Primer indeksa na atributu last\_name

Inicijalni prilikom kreiranja dokumenta u kolekciji postoji samo jedan indeks, to je indeks na poljem ***\_id***. Moguće definisati više indeksa u kolekciji, ali je preporučeno kreirate indekse samo za one atribute koji će se koristiti u upitima. Razlog tome leži u činjenici da indeksima ubrzavamo samo čitanje, ali usporavamo upis. MongoDB koristi B stablo za skladištenje indeksa, pa se tako prilikom dodavanja/brisanja dokumenta stablo mora ponovno izbalansirati kako bi reference pokazivale na sve dokumente.

## B stablo

B-stablo je struktura podataka u informatici. Karakteristike su mu potpuna balansiranost, sortiranje podataka po vrednosti ključa i čuvanje određenog broja elemenata u jednom čvoru stabla. Operacije nad podacima stabla se obavljaju u amortizovano logaritamskom vremenu.

Slika 4: B stablo

Na slici iznad vidimo jednostavan primer B stabla. Radi se o indeksiranju imena, pa tako root poseduje dva pokazivača jedan na čvor u kome se nalaze imena koja počinju slovima [A-K], a drugi [L-Z]. Čvor na koji pokazuje [A-K] sadrži tri pokazivača [A-D],[E-G],[H-K]. [A-D] pokazivač vodi do imena koje počinju slovima od A do D uključujući i A i D. Tu vidimo imena poput Adams, Baker, Cook, Davis, na kraju svako ime sadrži pokazivač do lokacije na disku.

## Indeksi nad jednim poljem

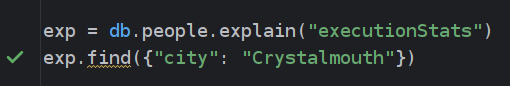
Indeksi nad jednim poljem predstavljaju najjednostavnije indeksi. Kreiramo ih kao na slici 5. ***db*** označava bazu podataka kojoj pristupamo, ***people*** označava kolekciju i sama komanda ***createIndex***. Argument unutar zagrada **(*city : 1*)** znači da ćemo kreirati indeks unutar kolekcije ***people*** nad poljem ***city*** i to u u rastućem redosledu ( ***1 – rastući, -1 opadajući).***

Slika 5: Kreiranje indeksa

Kako bi objasnili efikasnost indeksa, koristićemo MongoDB funkciju ***explain.*** Sem te funkcije koristićemo i funkciju ***hint***, o njoj će biti više u poglavlju Kompleksni indeksi.

### Funkcija explain

Funkcija explain je jako korisna funkcija koja se koristi za analizu i optimizaciju performansi upita. Kada se uz sam upit doda i explain() funkcija ili kao na slici 6, dobijamo informacije o tome kako je upit izvršen.

Slika 6: Explain funkcija i upit uz explain funkciju

Tu možemo videti da li su koriščeni indeksi prilikom upita, kako je tekao sam tok upita, koliko je trebalo vremena za upit i mnoge druge infromacije. Nekoliko ključnih stavki koje na funkcija explain pruža su:

* **"executionStats"**: Sekcija koja nam pruža detaljne informacije o vremenu izvršavanja upita, broju pronađenih dokumenata, broju obrađenih dokumenata, korišćenju indeksa, broju skeniranih indeksa i drugim statistikama.
* **"queryPlanner"**: Ova sekcija pruža informacije o planeru upita, kao što su korišćeni indeksi, broj dokumenata koji su obrađeni u planiranju upita, ocena plana izvršenja, kao i koji su alternativni planovi razmotreni.
* **"winningPlan"**: Sekcija koja nam pruža informacije o pobedničkom planu izvršenja koji je MongoDB odabrao za upit. To se odnosi na korišćene indekse, operacije koje su izvršene, broj dokumenata koji su skenirani, i druge povezane informacije.
* **"executionStages"**: Ova sekcija pruža informacije o fazama izvršenja upita. Neke od faza koje je bitno pomenuti su:   
  *COLLSCAN* – Ova faza se koristi kada se vrši skeniranje cele kolekcije dokumenata.   
  *IXSCAN* – Koristi se kada se vrši skeniranje putem indeksa.   
  *FETCH* – Izvršava se kada se vrše dodatne operacije za pribavljanje dokumenata koji su pronađeni u nekim od prethodnim fazama. Na primer, ako je indeks korišćen za filtriranje dokumenata, ali su stvarni podaci i dalje pribavljeni iz kolekcije.

U konkretnom primeru imamo kolekciju people koja u sebi sadrži 50474 dokumenata. Prvo ćemo privaviti slučajan dokument bez korišćenja indeksa. Na slici 7 pribavljamo dokument po mejlu osobe i atribut ***email*** nije indeksiran. Obzirom da nema indeksa koji može ubrzati pretragu na slici 7 vidimo da se radi *COLLSCAN* fazi i da je skeniranio 50474 (*totalDocsExamined*) dokumenata kako bi se pribavio jedan koji tražimo. Možemo videti i da je 0 indeksa skenirano (*totalKeysExamined*), što je i logično obzirom da i dalje nismo postavili nijedan indeks.

Nakon postavljivanja indeksa nad poljem ***email*** komandom datom na slici 5 ponovo pokrećemo upit gde nalazimo određenu osobu po zadatom mejlu. Sada ishod ***explain*** funkcije izgleda kao na slici 8.

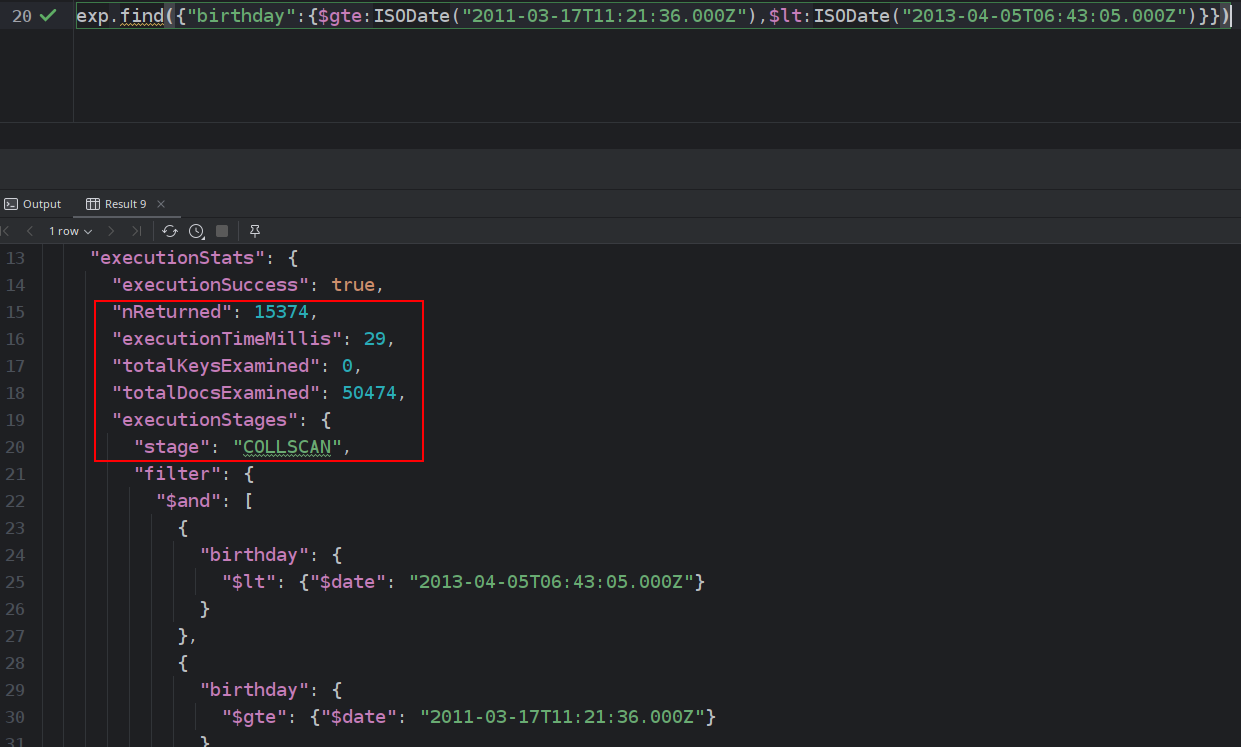
Slika 7: Upit bez indeksa

Slika 8: Upit sa indeksom

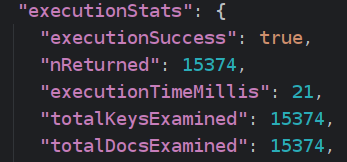
Ovde vidimo da je samo 1 dokument/indeks skeniran. Dakle uspeli smo da optimizujemo pretragu sa O(n) na O(1). Osim toga vidimo i da upit prolazi kroz fazu *IXSCAN* u kojoj se skeniraju indeksi. Ukoliko bismo izvršili pretragu nad istom kolekcijom ali putem nekog drugog polja, postojeći indeks nad poljem ***email*** nebi ubrzao pronalazak dokumenata već bi pretraga išla preko *COLLSCAN* faze.

Putem indeksiranja možemo ubrzati i pretragu opsega vrednosti. Na slici 9 vidimo filtriranje između dva datuma. Funkcija ***explain*** daje nam uvid da je za tu operaciju bilo potrebno skenirati sve dokumente preko *COLLSCAN* faze.

U skladu sa komandom datom na slici 5, postavljamo indeks nad poljem ***birthday*** i ponovo pokrećemo pretragu. Sada možemo videti da su skenirani samo oni dokumenti koji ispunjavaju uslov filtera, da je skenirano 15374 indeksa i da ne postoji više *COLLSCAN* faza izvršenja već *IXSCAN (Slika 10).*

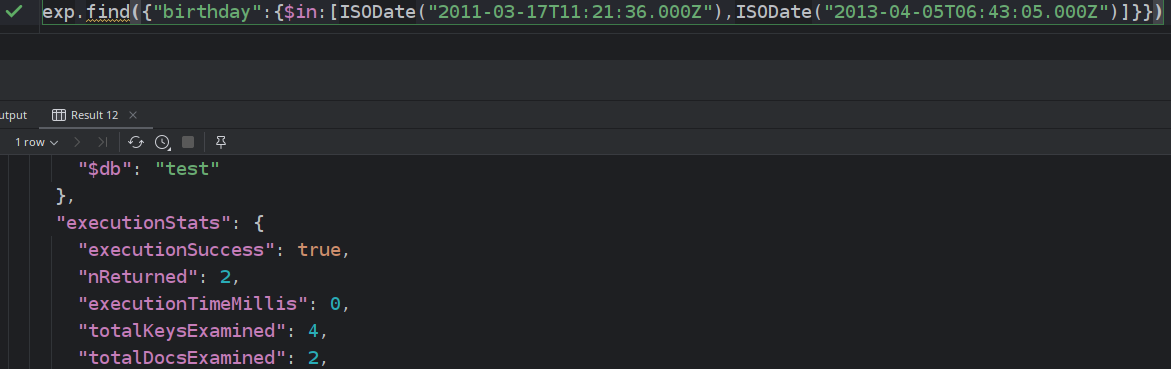


Slika 9: Pretraga opsega bez indeksiranja



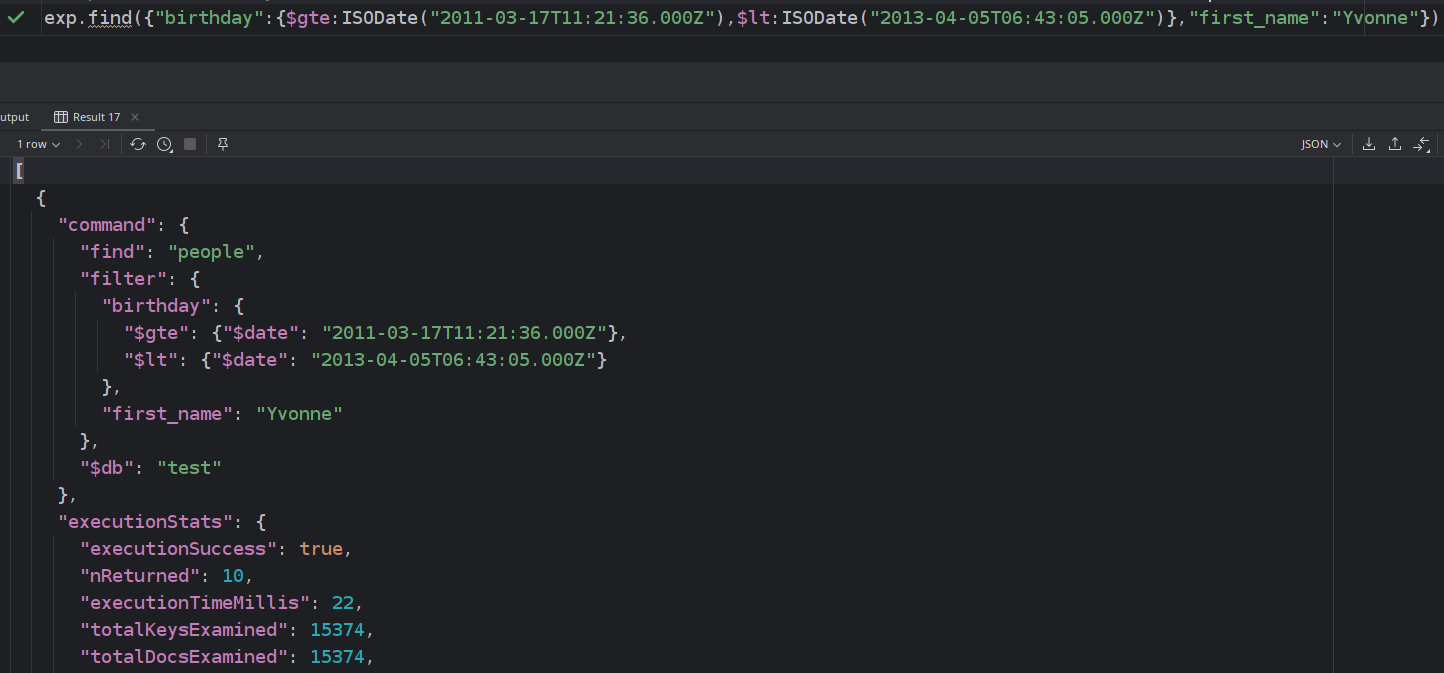
Slika 10:Explain funkcija nakon postavljanja indeksa

Sem filtera opsega vrednost indeksi utiču i na druge operacije na slici ispod vidimo operaciju ***IN*** gde tražimo dokumente čije je ***birthday*** polje poklapa sa barem jednim argumentom iz ***IN*** operacije. Na slici ispod vidimo da je za 2 vraćena dokumenta pretraženo isto 2 dokumenta dok je bilo potrebno skenirati 4 indeksa. Bilo je potrebno skenirati više indeksa zbog posledice rada algoritma za pretragu.



*Slika 11: Operacija IN uz indeksiranje*

Nekada je potrebna pretraga po više polja, indeksiranje čak i tada ubrzava pretragu. Ukoliko je na barem jedan od argumenata u pretrazi prethodno postavljen indeks, MongoDB će prvo proći kroz fazu skeniranja indeksa a potom i izvršiti pretragu po ostalim argumentima nad kojima nije postavljen indeks. Na slici možemo videti malo kompleksniji upit gde prvo filtriramo opseg datuma rođenja a potom i nalazimo tačno one dokumente kod kojih je ime „Yvonne“.

  
*Slika 12: Kompleksan upit sa poljem koje ima indeks i poljem koje nema*

## Kompleksni indeksi

Kompleksni ili složeni indeksi u MongoDB se koriste za indeksiranje polja koja su složenog tipa, kao što su polja objekata, nizova ili ugniježđenih dokumenata. Složeni indeksi omogućavaju efikasno pretraživanje i filtriranje dokumenata na temelju više polja ili svojstava.

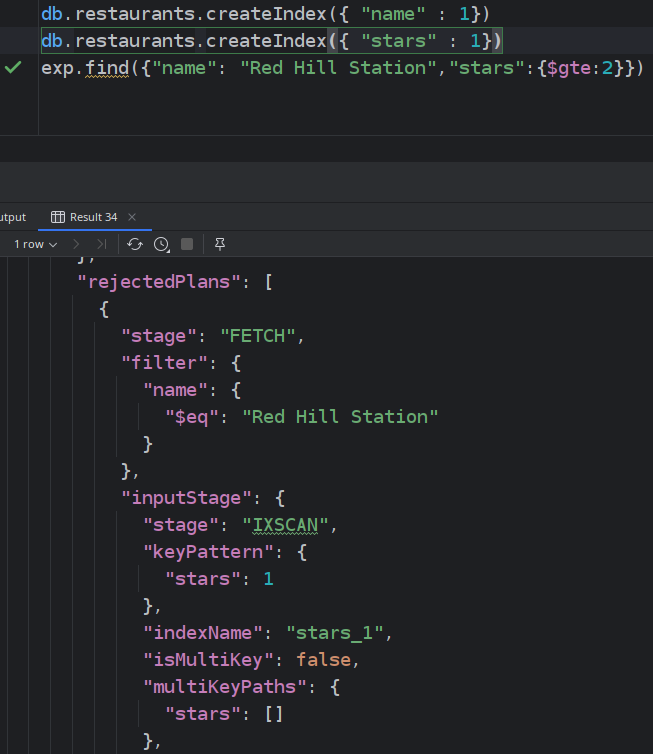
MongoDB podržava nekoliko vrsta složenih indeksa:

* **Indeksi nad poljima objekata**: Možemo indeksirati polja unutar objekata pomoću kao "imeObjekta.imePolja". Na primer, ukoliko bismo imali dokumente koji sadrže objekat "adresa" sa poljima "grad", "država" i "poštanskiBroj", mogli bismo kreirati indeks na "adresa.grad". Time bismo značajno ubrzali pretraživanje dokumenata kolekcije po parametru grada.
* **Indeksi nad poljima nizova**: Možemo indeksirati polja unutar nizova pomoću nomenklature "imeNiza.imePolja". Ako bismo imali dokumente koji sadrže niz "telefoni" s poljima "broj" i "tip", mogli bismo stvoriti indeks na "telefoni.broj" i time efikasno pretraživali dokumente po broju telefona.
* **Indeksi nad ugnježdenim dokumentima**: Moguće je indeksirati polja unutar ugnježdenih dokumenata pomoću notacije "imeDokumenta.imePolja". Na primer, ako imamo dokumente koji sadrže ugnježdeni dokument "osoba" s poljima "ime", "prezime" i "godine", možete kreirati indeks na "osoba.ime".

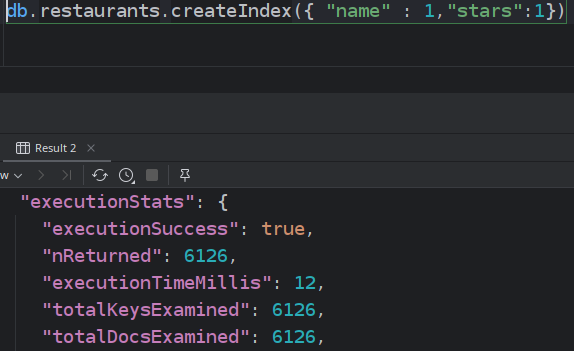
Kako bi objasnili kompleksno indeksiranje, uvodimo novu kolekciju dokumenata. Reč je o restoranima, i kolekcija sadrži 1,000,000 dokumenata. Šema dokumenta data je na slici 13.

Slika 13: Šema restoran dokumenta

Ponekad može se javiti potreba da u upitu imamo više argumenata, pri kraju prethodnog poglavlja videli smo primer u kome je jedan parametar indeksiran a drugi ne. Sada ćemo prikazati primer u kome su oba parametra indeksirana. Iako su oba parametra indeksirana MongoDB je odlučio da ipak pretragu vrši samo preko jednog indeksa (Slika 14). Ovo se desilo zato što smo zasebno indeksirali polja ***name*** i ***stars***, MongoDB je zbog toga odbio plan da vrši pretragu po drugom indeksu i zbog toga morao pretražiti 10017 dokumenata kako bi našao 6126.

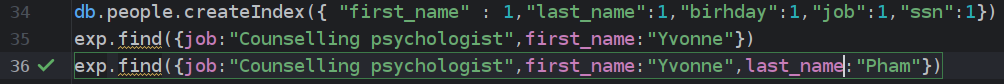
Slika 14: Pretraga kada su oba parametra posebno indeksirana

Na kraju ćemo kreirati kompleksan indeks koji sadrži u sebi 2 polja. Praktično grupišemo dva indeksa. Na slici 15, vidimo kreiranje kompleksnog indeksa. Na istoj slici možemo videti i koliko manje je dokumenata potrebno skenirati. Uspeli smo smanjiti broj skeniranih dokumenata sa **10017** na **6126**, dakle za nekih **~ 40%**.

Slika 15: Kompleksni indeks koji sadrži 2 polja

### Prefiks indeksa

Prefiks indeksa predstavlja osobinu indeksa koja se odnosi na kompozitne indekse. Na slici ispod, prvo kreiramo kompozitni indeks koji se sastoji od sledećih polja: ***first\_name***, ***last\_name***, ***job***, ***ssn***. Upit zatad u liniji 35 mora proveriti 26 indeksa kako bi našao traženi dokument, dok upit dat u liniji 36 koji sadrži i ***last\_name*** treba skenirati svega 2 indeksa. Možemo zaključiti da što je veće poklapanje između polja upita i polja indeksa to je i veća efikasnost upita.

Slika 16: Primer prefiks indeksa

Ali ukoliko bi iz upita izostavili polje kojem počinje kompozitni indeks (u našem slučaju ***first\_*name**), onda nebi ni došlo do pretrage putem indeksiranja, već bi upit prošao kroz ***COLLSCAN*** fazu. Tako da ukoliko želimo iskoristiti neki kompozitni indeks, morali bi u upitu staviti barem prvo indeksirano polje.

## Sortiranje sa indeksiranjem

MongoDB razlikuje dva načina na koje se sortiranje može obaviti, to su sortiranje u memoriji i sortiranje preko indeksa.

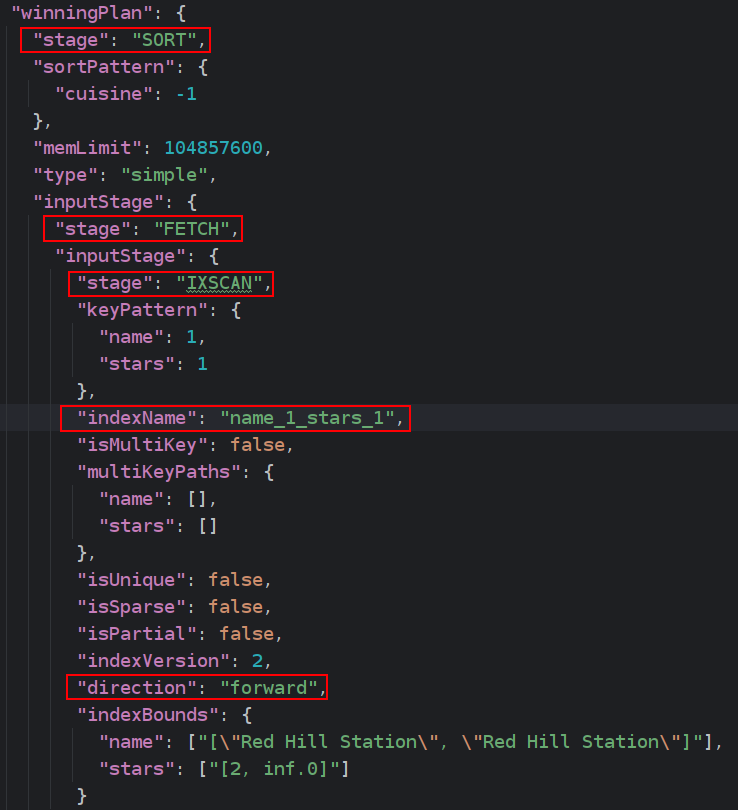
**Sortiranje u memoriji:**

Ovo se odnosi na sortiranje podataka u RAM memoriji MongoDB servera bez korišćenja indeksa. Kada se koristi sortiranje u memoriji, MongoDB učitava sve potrebne podatke u memoriju i zatim vrši sortiranje prema zadatom kriterijumu. Ovaj pristup može biti brz za manje kolekcije ili manje količine podataka, ali može biti spor za velike kolekcije ili velike količine podataka koje ne mogu stati u memoriju.

**Sortiranje preko indeksa:**

Kada se koristi sortiranje preko indeksa, MongoDB koristi indeks za direktan pristup sortiranim podacima, umesto da učitava sve podatke u memoriju i vrši sortiranje. Ovo može biti znatno brže i to posebno za velike kolekcije i kompleksne upite.

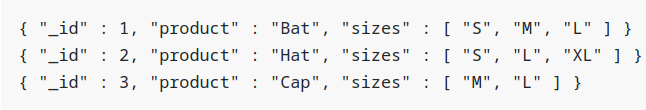
Za primer sortiranja preko indeksa iskoristićemo primer dat na slici 14, jedina promena je dodavanje ***.sort({“cuisine”:1})*** na kraju upita što znači da ćemo prethodno pribaljene redove sortirati po parametru ***cuisine.*** Na slici ispod vidimo da je sada upit prošao kroz fazu skeniranja indeksa i da je iskorišćen ***name\_1\_stars\_1***indeks. Ako se setimo, postoje zasebni indeksi i za ***name*** i za ***star*** ali su oni odbijeni zato što ne daju dovoljno dobre rezultate.

Slika 16: Plan izvršenja kod sortiranja uz prethodno postavljene indekse

Da indeksi nisu postojali redosled izvršenja bi samo počeo ***COLLSCAN*** fazom umesto ***IXSCAN***. Ukoliko prethodno postoji kompozitni indeks, a hteli bismo ga iskoristiti u svrhu sortiranja morali bi paziti na indeks prefiksa u upitu. Sort deo bi morao kao prvo polje imati prvo polje iz kompozitnog indeksa inače bi upit išao preko ***COLLSCAN*** faze.

## Indeksi nad poljima nizova

Kako bi demonstrirali indeks nad poljem nekog niza iskoristićemo kolekciju koja sarži proizvode (Slika 17).

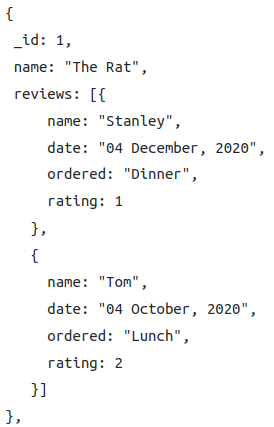


Slika 17: Kolekcija sa proizvodima

Postavljanje indeksa nad poljem se ne razlikuje od indeksiranja polja objekta, postavljamo indeks komandom ***db.products.createIndex( { "sizes": 1 } )****.* Nije potrebno posebno specificirati da se radi o indeksu nad polju niza ***(multikey)***, MongoDB će prepoznati da se radi o indeksu nad poljem. Dakle MongoDB će kreirati indeks za svaki od elemenata u nizu, u ovom slučaju za elemente ***S, M, L, XL***.

Nekada element neće biti prosta vrednost, već objekat (Slika 18). U tom slučaju MongoDB nudi opciju da postavimo indekse objektu u nizu. Kao i ranije to radimo komandom: ***db.restaurants.createIndex( { "reviews.ordered": 1, "reviews.rating": -1 } )***

db.restaurants.find( { "reviews.ordered": "Dinner" } )  
.explain() ***- IXSCAN***

Slika 18: Multikey indeks

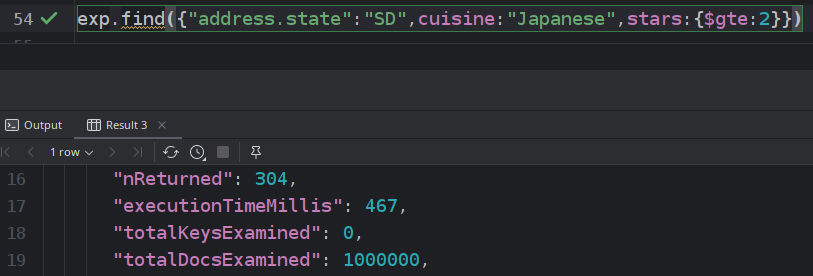
db.restaurants.find( { "reviews.ordered": "Dinner", "reviews.rating": { $gt: 3 } } ).explain() - ***IXSCAN***

db.restaurants.find( { "reviews.name": "Lisa", "reviews.rating": { $gt: 3 } } ).explain() - ***COLLSCAN***

U iznad navedena tri upita vidimo kroz koju fazu prolazi upit. U prvom slučaju naveden je samo samo prvi element obzirom da je on prvi dat i u indeksu i daje prefiks indeksa, upit ide kroz ***IXSCAN*** fazu. U drugom slučaju u upitu se koriste oba polja iz indeksa pa se tako izvršava isto ***IXSCAN*** faza. Ali u trećem slučaju MongoDB ide ***COLLSCAN ,*** to je zato što ne postoji zaseban indeks za ***reviews.name*** i ***reviews.rating,*** a nema paremetra ***reviews.ordered*** koji je indeks prefiks u prethodno kreiranom indeksu.

## Parcijalni indeksi

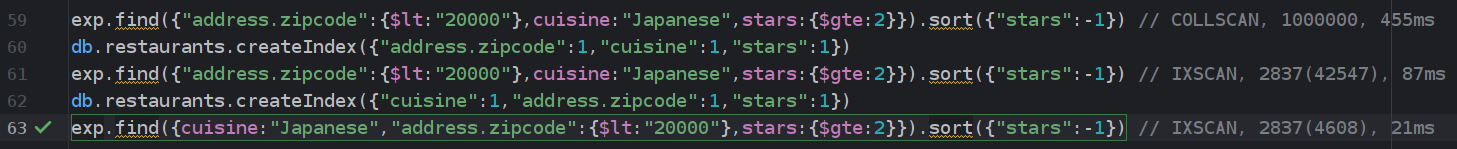
Nekada nebismo hteli kreirati indekse celoj kolekciji već samo određenim dokumentima u kolekciji. Razlog tome je zato što indeks nad celom kolekcijom može postati previše velik i postoji šansa da neće moći stati u memoriju, što bi značilo da u tom slučaju gubimo korist koju smo dobili preko indeksa. Obzirom da parcijalni indeksi targetiraju samo dokumente koji uspunjavaju neki uslov, smanjićemo uticaj indeksa na dodavanje/menjanje dokumenata jer kao što smo rekli prilikom navedenih operacija B stabla koja stoje iza indeksiranja se moraju izbalansirati, što znači sporiji odziv. Iskoristićemo kolekciju o restoranima kako bi prikzali prednosti parcijalnih indeksa.

Slika 19: Upit i efikasnost pre parcijalnog indeksiranja

Pretragom datom na slici 19, vidimo da je potrebno skenirati celu kolekciju kako bi pribavili 467 dokumenata. Postavljamo parcijalni indeks komandom ***db.restaurants.createIndex({“address.state”:1,”cuisine”:1},{partialFilterExpression:{“stars”:{$get:2}})*** . Komanda za kreiranje indeksa nam je poznata, s tim što dodajemo deo koji indeksera samo dokumente koji za vrednost u polju ***stars*** imaju veće ili jednako od 2. Ponovnim pokretanjem upita dobijamo da upit ide kroz ***IXSCAN*** fazu i da je potrebno proći kroz svega **304** indeksa/dokumenata, obzirom da toliko ima i pribavljenih dokumenata zaključujemo da se radi o efikasnosti O(1). Ali ukoliko bi promenili upit tako da bi filtrirali sve restorane koji umesto 2+ zvezdica imaju 1+ zvezdica, upit bi morao proći kroz celu kolekciju i prethodno kreirani parcijalni indeks nebi značio ništa jer smo njime indeksirali samo restorane koji imaju 2 ili više zvezdica.

# Različite strategije za optimizaciju upita

Često samo kreiranje indeksa nije dovoljno. Kada je to slučaj, rešenje treba tražiti u različitim strategijama optimizacije. Za demonstraciju korišćen je prethodno predstavljana kolekcija o restoranima.



*Slika 20: Optimizacija upita*

U upitu tražimo sve restorane koji imaju vrednost ***zipcode-a*** manji od 20000, japansku kuhinju, broj zvezdica veći ili jednak od 2 i na kraju sortiranje po broju zvezdica u opadajućem redosledu. U startu pod pretpostavkom da nijedan indeks nije kreiran, naveden upit se izvršava za **455ms** prolazi kroz ***COLLSCAN*** što znači da će morati da skenira sve dokumente kolekcije i na kraju vrši sortiranje u memoriji što predstavlja veliki udarac na performanse. Po završetku ***SORT*** faze podaci se pribavljaju.

Prvi prilaz optimizaciji bio bi uvođenjem kompozitnog indeksa nad poljima koja su data u upitu i to baš u istom redosledu. Ta strategija u mnogome poboljšava vreme izvršenja sa **455ms** na **87ms** dakle za **~ 510%**, sada je potrebno kroz svega 2837 dokumenata ali pak kroz velikih 42547 indeksa **(*IXSCAN*)**. Što i nije baš najoptimalnije a i sortiranje se i dalje obavlja u samoj memoriji. Upit prolazi kroz ***IXSCAN***, ***SORT***, ***FETCH*** fazu.

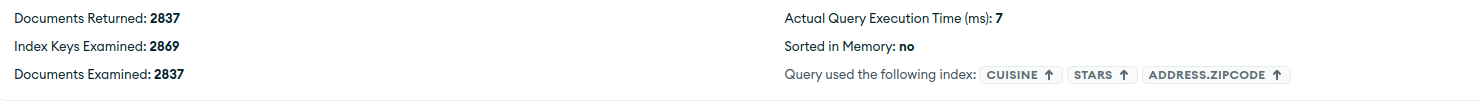
Razlog zbog čega je MongoDB morao skenirati 42547 indeksa leži u činjenici da nam je prvo polje u kompozitnom indeksu ***zipcode****.* Postoji previše različitih vrednosti za ***zipcode*** i zbog toga je potrebno proći kroz puno indeksa. Treba težiti tome da prvi indeks odnosno indeks prefiks bude što generalniji (da odstranjuje što više dokumenata), da sadrži relativno mali broj vrednosti npr. klasifikacija po nečemu. U našem slučaju možemo odabrati tip kuhinje (***cuisine***). Kreiramo novi indeks u kome ***zipcode*** i ***cuisine*** rotiraju mesta tako da ***cuisine*** polje bude prefiks indeksa. Nakon toga potrebno je rotirati ta polja i u upitu. Ovaj prilaz smanjio je vreme izvršenja sa **87ms** na **21ms** tj. za otprilike **4x** manje vreme. Smanjili smo i broj potrebnih indeksa koje trebamo skenirati u toku pretrage, ranije je bilo 42547 sada je to svega 4608 uz činjenicu da se broj skeniranih dokumenata nije promenio. Upit isto prolazi kroz ***IXSCAN***, ***SORT***, ***FETCH*** fazu i sortiranje se i dalje vrši u memoriji. Za zadnju optimizaciju pokušaćemo prebaciti sortiranje iz memorije na sortiranje prema indeksu.

Na slici 21 sada vidimo rotaciju između polja ***stars*** i ***address.zipcode*** pri čemu prefiks indeksa ostaje nepromenjen. Sada kada je polje ***stars*** na drugom mestu možemo iskoristiti sortiranje indeksa i tako pomeriti sortiranje iz memorije. Dakle MongoDB sada može pribaviti već sortirane podatke iz indeksa čime smanjujemo dodatno sortiranje posle pribavljanja dokumenata kao što je ranije bio slučaj.

/home/dimi/Desktop/FAKS/sistemi baza podataka/2023-04-15_11-58.png2023-04-15_11-58

*Slika 21: Optimizacija indeks-a i upita*

Na slici 22 vidimo i efikasnost po zaršetku svih optimizacija. Vreme smo u odnosu na prethodnu otpimizaciju smanjili sa **21ms** na **7ms** odnosno za **3x** puta. Pomerili smo sortiranje iz memorije na indekse. Sada je potrebno skenirati skoro jednak broj indeksa i dokumenata, 2869 indeksa i 2837 dokumenata. Smanjili smo broj indeksa koje je potrebno pretražiti sa 4608 na 2869.



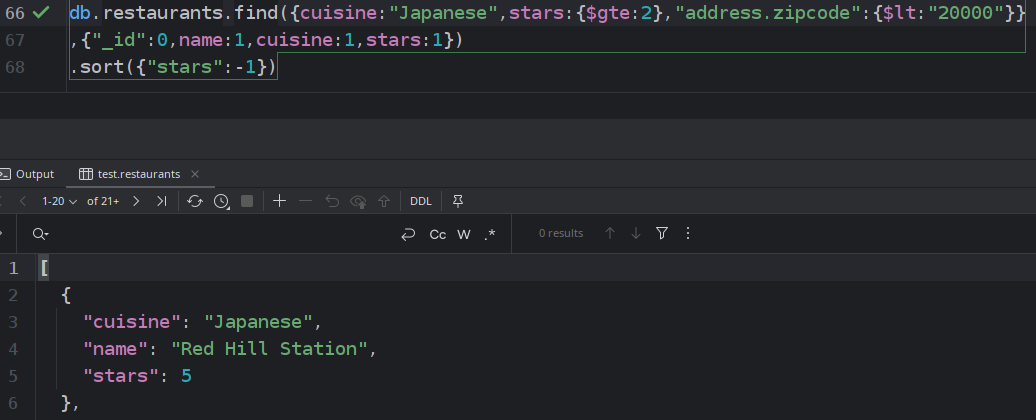
*Slika 22: Rezultat na kraju svih optimizacija*

Na kraju ukoliko bi uporedili prvi stepen gde nemamo indeks i poslednji stepen nakon svih optimizacija upitai indeksa zaključujemo:

* Smanjili smo vreme sa **455ms** na **7ms** (65x brže)
* Smanjili smo broj skeniranje dokumenata na svega 2837
* Smanjili smo broj indeksa koje je potrebno skenirati na 2869
* Prebacili smo sortiranje iz memorije na indekse

# Projekcije

Projekcije predstavljaju još jednu od funkcionalnosti MongoDB baze. Korišćenjem projekcija, možemo izabrati koja polja želimo vrati u rezultatu upita. Time smanjujemo broj vraćenih podataka i optimizujemo efikasnost upita. Projekcije definišemo u sklopu upita zajedno sa delovima za filtriranje i sortiranje. Bitnost projekcija možemo najbolje videti u slučaju kada se radi o dokumentima koji imaju kompleksnu šemu, pod time podrazumevamo puno polja ili različitih struktura u dokumentu. Za primer iskoristićemo šemu dokumenta datoj na slici 13 (restoran).

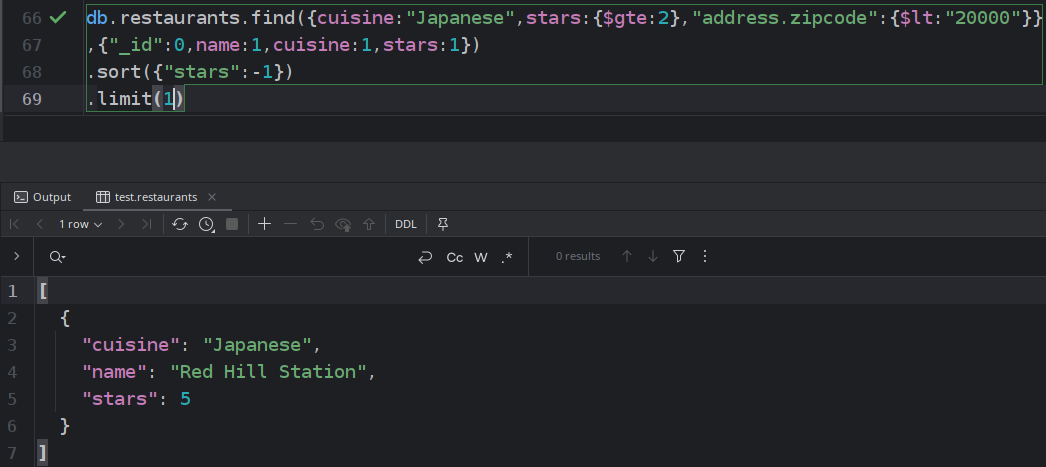


*Slika 23: Upit sa projekcijom*

Na slici 23 možemo videti modifkovan upit sa slike 21. Između ***find*** i ***sort*** dela dodali smo deo namenjen specifikaciji projekcije. Projekciju specificiramo tako što prvo navodimo polje a potom i 0 ili 1 u zavisnosti od toga da li želimo izbaciti polje iz rezultat ili ne. U našoj projekciji naveli smo polja ***cuisine, name, stars.*** Ta polja će se naći u rezultujućem filteru. Ali po default-u ***\_id*** polje se uvek vraća, samim tim ukoliko bismo hteli da izopštimo ***\_id*** polje iz rezultate dodajemo ***\_id:0*** u delu projekcije.

# Limitiranje vraćenih dokumenata

Operator ***limit*** jeste deo upita i koristimo ga kako bi ograničili broj dokumenata koje će vratiti MongoDB. Ovaj operator nam omogućava da definišemo maksimalan broj dokumenata koji će biti vraćen kao rezultat upita, bez obzira na ukupan broj dokumenata koji ispunjavaju uslove upita. Na slici 24 vidimo primer operatora limit. Ograničili smo broj pribavljenih dokumenata na jedan dokument.



*Slika 24: Funkcija limit*

Operator ***limit*** predstavljamo kao jednu od strategija zato što nije uvek potrebno pribaviti sve dokumente iz filtera, nekada je potrebno pribaviti samo prvih n dokumenata.

# Zaključak

U ovom radu istražili smo različite aspekte MongoDB baze, počevši od uticaja hardvera, preko indeksa sve do projekcije i limit operatora. Spoznali smo da su indeksi ključni za optimizaciju upita, ali sami po sebi ne daju uvek željenu optimizaciju iz prvog puta. Zato je potrebno probati optimizovati različitim tehnikama prvo indekse, a pošto upiti prate strukturu indeksa potrebno je i upite kasnije refaktorisati.Saznali smo kako se indeksi interno skladište. Videli smo kako možemo značajno ubrzati vreme izvršenja upita (65x). Summa summarum bi bila da MongoDB nudi široki spektar mogućnosti za optimizaciju performansi upita, uključujući indekse, projekcije, i limitanje broja vraćenih dokumenata. Razumevanje ovih funkcionalnosti može pomoći u efikasnom korišćenju MongoDB baze podataka i postizanju boljih performansi u aplikacijama koje koriste MongoDB kao bazu podataka.

# Literatura

1. https://sh.wikipedia.org/wiki/B-stablo