**Univerzitet u Nišu**

**Elektronski fakultet**

**Optimizacija upita**

**MongoDB**

**Student:** Jovana Nikolić 1147

**Profesor:** Doc. dr Aleksandar Stanimirović

**Predmet:** Sistemi za upravljanje bazama podataka

Sadržaj

[Uvod 2](#_Toc69343702)

[Hardverska razmatranja i konfiguracije 3](#_Toc69343703)

[Planovi za izvršenje upita 5](#_Toc69343704)

[Explain funkcija 5](#_Toc69343705)

[Hint funkcija 7](#_Toc69343706)

[Indeksi 7](#_Toc69343707)

[Indeksi nad jednim poljem 8](#_Toc69343708)

[Složeni indeksi 12](#_Toc69343709)

[Parcijalni indeksi 14](#_Toc69343710)

[Optimizacija CRUD upita 15](#_Toc69343711)

[Ograničenje broja rezultata koji se vraćaju 18](#_Toc69343712)

[Korišćenje projekcija 19](#_Toc69343713)

[Inkrement operator 20](#_Toc69343714)

[Zaključak 21](#_Toc69343715)

[Literatura 22](#_Toc69343716)

# Uvod

Optimizacija upita, i samo postojanje optimalnih upita jeste jedan od glavnih zadataka prilikom rada sa pojedinačnim DBMS-ovima. S obzirom na ekspanziju Interneta, kao i na brzine Interneta, kao i aplikacija korisnika, sve je veći pritisak ka stvaranju što optimalnijih upita.

Razlozi za optimizaciju upita su sledeći:

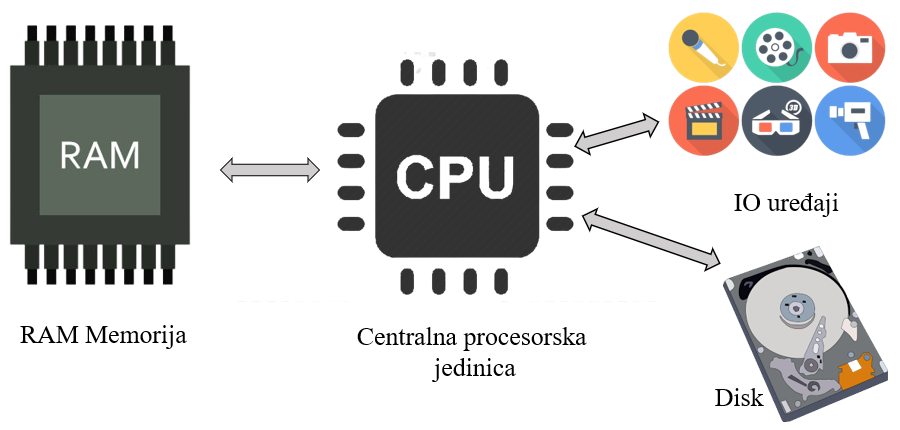
* Brže dobijanje rezultata upita, što će aplikaciju učiniti bržom korisniku.
* Omogućavanje DBMS-u da odgovori na više upita u istom vremenskom periodu
* Duže trajanje hardvera na kojima se izvršava DBMS, kao i efikasniji rad servera u pogledu potrošnje električne energije, kai i RAM-a.[1]

MongoDB baza podataka je najpopularnija i najkorišćenija NoSQL baza podataka, spada u dokumentu orjentisane NoSQL baze podatala. Koristi JSON ili BSON dokumente za smeštanje podataka. MongoDB baza podataka je postalo open-source 2009. godine, pri čemu je jedna od njenih glavnih karakteristika performanse. MongoDB je nerelaciona baza koja je pravljena kao jako bliska relacionim bazama podataka, sa mogućnošću kreiranja sopstvene šeme, koja je jako fleksibilna [2]. Umesto korišćenja tabela za smeštanje podataka, podaci su smešteni u kolekcijama, dok umesto jasno definisanih i uređenih redova kod relacionih baza podataka, kod MongoDB baze podataka, podaci se smeštaju u dokumentima, koja se sastoje od key-value parova, ime key-value para predstavlja polje, koje predstavlja pandan kolonama kod relacionih baza podataka [3].

MongoDB obezbeđuje uglavnom optimizaciju upita kroz postojanje indeksa i manipulacijom istih. Osim toga rad će se osvrnuti i na neke hardverske karakteristike sistema koje trebaju biti razmatrane prilikom rada sa MongoDB DBMS-om, kao i neke funkcije koje mogu doprineti poboljšanju performansi sistema. Imajući u vidu da upiti po pravilu ne bi trebalo da traju duže ood 100 ms, u ovom radu će biti prikazani i praktični primeri optimizacije upita. Za prikaz informacija o trajanju upita, kao i drugih informacija o izvršenju biće korišćena konzola, kao i alat MongoDB Compass. Upiti će se izvršavati nad dve kolekcije, kolekcijom *ljudi* i *restorani,* koje imaju 50 500 i 1 000 000 dokumenata, respektivno. Kolekcije su pronađene na izvoru [4].

# Hardverska razmatranja i konfiguracije

Kada se govori o performansama ne sme se isključiti uticaj hardverske konfiguracije na bazu podataka i samu aplikaciju. MongoDB spada u visokoperformansne baze podataka, ali je za postizanje dobrih performansi neohodan i odgovarajući hardver. Ukoliko se kao primer uzme Von Neuman-ova arhitektura koja je prikazana na slici , možemo uočiti RAM memoriju, SSD memoriju, disk, kao i IO jedinicu uz CPU.



*Slika 1 – Von Neumann arhitektura*

MongoDB koristi CPU za izračunavanje i izvršenje, RAM memoriju za izvršenje, disk i IO jedinicu za perzistenciju i komunikaciju između servera. RAM memorija je 25 puta brža od SSD memorije, takođe njena cena je značajno opala poslednjih godina, i iz tog razloga MongoDB ima storage engine-e koji su visoko zavisno od RAM memorije, ili čak potpuno orjentisani izvršenju u memoriji za sve svoje operacije upravljanja memorijom. Veliki broj operacija kao što su agregacija pipeline operacija, pretraga indeksa, se prevashodno oslanjaju na rad u RAM memoriji. Engine za upite se oslanja na RAM memoriju za dobijanje podataka. Takođe RAM memorija se koristi za smeštanje informaciju o konekcijama, otprilike 1Mb po uspostavljenoj konekciji. Shodno tome, performanse aplikacije će biti umnogome bolje sa više dostupne RAM memorije. CPU se koristi od strane svih aplikacija na računaru, kod MongoDB-a korišćenje CPU-a je povezano sa storage engine-om koji se koristi, kao i nivoom konkurencije, koji je moguć za odgovarajuću instancu MongoDB-a. Po difoltu MongoDB se trudi da iskoristi sva dostupna jezgra CPU-a za obradu zahteva. MongoDB podržava nezaključavajući mehanizam kontrole konkurencije, što znači da ukoliko dođe do istovremenih zahteva za uzajamno neblokirajuće operacije, kao što su upis u različite dokumente ili read operacije, MongoDB će imati bolje performanse sa više CPU resursa. Ono što treba imati u vidu jeste da nisu sve operacije upisa neblokirajuće operacije. Višestruki upis u isti dokument je blokirajuća operacija, te se sledeći upis može desiti tek nakon što se prethodni upis završi, odnosno kada write operacija otključa dokument. U ovakvim situacijama CPU sa više jezgara neće doprineti performansama. Primer predstavlja sledeća slika:



*Slika 2 - Primer write operacije*

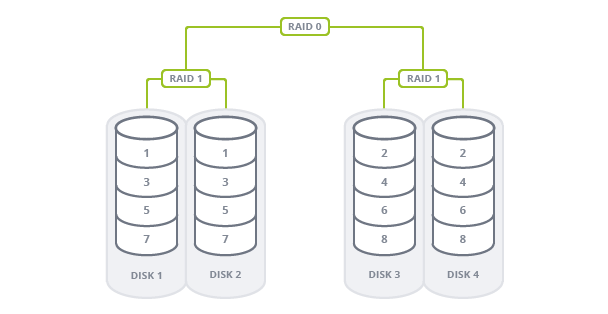
Bez obzira na broj jezgara, istovremeni zahtevi za ovom operacijom će se izvršavati sekvencijano, multi-core procesori neće davati nikakvu prednost u performansama.

Za perzistenciju podataka MongoDB koristi diskove, te visoka vrednost IOPS-a (ulazno-izlaznih operacija u sekundi) obezbeđuje bolje performanse kod upita. Tip diska ima veliki uticaj na peformanse, kao što je prikazano u tabeli. Upoređeni su HDD, SSD, EBS diskovi [5].

|  |  |
| --- | --- |
| Tip | IOPS |
| 7200 rpm SATA | 75-100 |
| SSD Intel X25-E(SLC) | 175-210 |
| Amazon EBS | 100 |
| FusionIO | 135 000 |
| Violin Memory 6000 | 1 000 000 |

*Tabela 1- IOPS različitih diskova*

Ono što se preporučuje jeste korišćenje RAID-a za redundansu prilikom read i write operacija, ali samo pojedinih arhitektura RAID-a. Preporučuje se korišćenje RAID 10 arhitekture. RAID 10 arhitektura predstavlja arhitekturu koja sadrži kombinaciju nivoa 1 (mirroring-a) i 0 (striping-a) [6]. RAID 10 arhitektura je prikazana na slici.



*Slika 3 - RAID 10 arhitektura*

RAID 10 arhitektura obezbeđuje brzinu i sigurnost istovremeno, sigurnost jer su svi podaci duplicirani, a brzinu time što su podaci podeljeni u trake i rasporostranjeni na različite diskove, te velike količine podataka mogu biti istovremeno upisane na disk ili pročitane sa istog [6]. Ne preporučuje se korišćenje RAID 5 ili RAID 6 arhitekture jer ne obezbeđuju dovoljne performanse. RAID 0 arhitektura iako ima odlične performanse prilikom operacije upisa, nije preporučena jer je obezbeđuje ograničenu dostupnost, što može izazvati manjak u performansama prilikom operacije čitanja [5].

U zavisnosti od servera, može biti omogućeno korišćenje više diskova, što je podržano od strane MongoDB DBMS-a, i omogućava raspoređivanje podataka u bazama, indeksa i propratnih fajlova po različitim diskovima. Takođe ulogu u performansama ima i hardver mreže, što je veći opseg mreže, to će performanse biti bolje. Na performanse utiču i mrežni svičevi, load balanseri podataka, zaštitni zidovi(firewall-ovi), rastojanje između klastera, načini povezivanja sa drugim data centrima. Svi ovi parametri utiču na performanse i shodno aplikaciji i arhitekturi data centara i klastera, treba se obratiti pažnja na iste [5].

# Planovi za izvršenje upita

MongoDB DBMS sadrži ugrađeni optimizator upita. Za svaki upit u MongoDB DBMS, optimizator upita bira i kešira najefikasniji plan na osnovu postojećih indeksa. Plan zapravo predstavlja niz koraka koji će biti izvedeni zarad izvršenja upita. Prilikom dolaza novog upita, server će prvo pogledati koji su sve indeksi nad kolekcijom dostupni. Zatim server proverava koji indeksi zadovoljavaju potrebe upita, ti indeksi, predstavljaju kandidate indekse Od indeksa kandidata se kreiraju planovi kandidata. MongoDB koristi empirijski planer za upite, pri čemu se svaki plan kandidata izvršava određeno vreme, planer upita određuje koji je imao najbolje rezulate, u MongoDB-u najbolji rezultat može imati upit koji prvi vrati sve rezultate, ili određeni broj rezulatat za najkraći vremenski period. U zavisnosti od upita planer za izvršenje upita bira šta će koristiti kao strategiju za izbor pobedničkog plana [7]. S obzirom na to da mnogi upiti imaju isti oblik MongoDB koristi keš za smeštanje informacija o tome koji plan se koristi shodno obliku upita, koji se zove *planCash*. Vremenom se kolekcija menja, kao i postojeći indeksi, te postojeći planovi mogu biti izbačeni iz keša. Planovi mogu biti izbačeni iz keša u sledećim situacijama:

* Server je restartovan
* Prvi deo upita prevazilazi količinu rada koja je bila neophodna za izvršenje pobedničkog plana
* Prilikom rebuild-ovanja indeksa
* Prilikom kreiranja ili brisanja indeksa [8]

## Explain funkcija

Explain je jako bitna funkcija kod optimizacije upita. Može biti dodata na kraju svakog upita jednostavnim dodavanjem poziva funkcije na kraj upita, ili klikom u MongoDB Compass alatu. Pozivom funkcije se može videti da li upit koristi očekivani indeks, da li indeks obezbeđuje sortiranje, koliko je selektivan indeks, kao i najbitnije, koji deo upita najviše oduzima performanse. MongoDB dozvoljava kreiranje takozvanog *explainable* objekta, nad kojim se mogu testirati upiti, bez samog uticaja na bazu podataka, što predstavlja odlično rešenje ukoliko je serevr preopterećen ili se samo vrši testiranje upita. Explain funkciji mogu biti zadata tri parametra, pri čemu explain funkcija se ponaša različito i vraća različite rezultate:

* *queryPlanner* – daje podatke o planu izvršenja koji je izabran od strane optimizatora upita, bez samog izvršenja upita, predstavlja difoltno ponašanje, on se izvršava ukoliko se ne prosledi nijedan parametar metodi
* *executionStats* – izvršava pobednički upit, koji je izabran od strane optimizatora upita, i daje statističke podatke o upitu
* *allPlansExecution* – izvršava pobednički upit, i daje informacije o svim alternativnim planovima koji su razmatrani za taj upit

Rezultat koji vraća explain funkcija predstavlja planove za izvršenje funkcija u okviru stabla faza. Neke od mogućih operacija su:

* COLLSCAN – koja prestavlja skeniranje kolekcije
* IXSCAN – predstavlja skeniranje ključeva indeksa
* FETCH – označava pribavljanje dokumenata

Explain funkcija vraća informacije o queryPlaner-u kao i o detaljima izvršenja, odnosno executionStats, ukoliko je funkcija u modu koji zahteva izvršenje funkcije. Na sledećim slikama su prikazani delovi informacija koje su vraćene.



*Slika 4 – Prikaz Query planer polja*

QueryPlaner sadrži informacije o selekciji upita od strane optimizarora upita, *namespace* polje specificira ime baze ili kolekcije nad kojom se vrši upit, winningPlan polje sadrži detaljne informacije o planu koji je izabran od strane optimizatora upita, plan izvršenja je u okviru faza koje su predstavljene strukturom stabla, stage polje stage sadrži ime faze. RejectedPlans polje sadrži niz planova koji su razmatrani i odbijeni od strane optimizatora upita, ukoliko nema alternativnih planova, ovo polje je prazno.

ExecutionStats polje vraća detalje o izvršenju pobedničkog plana, ukoliko je naravno došlo do izvršenja naredbe. Na sledećoj slici su su prikazane dodatne informacije koje se dobijaju prilikom izvršenja execute fukcije, kada se i sam upit izvršava.

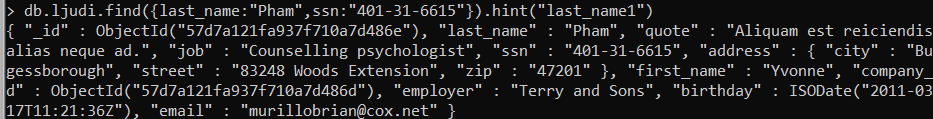


*Slika 5 – Prikaz Execution stats polja*

Bitno je posmatrati polje nReturned, koje označava broj dokumenata koji odgovara upitu, executionTimeMillis, koje označava ukupno vreme u milisekundama koje je neophodno za planiranje i izvršenje upita. TotalKeysExamined polje označava broj ključeva indeksa koji je skeniran, totalDocsExamined, koji govori o broju dokumenata koji su ispitani, dok polje executionStages, sadrži detaljne informacije o svim fazama izvršenja pobedničkog plana [9].

## Hint funkcija

Glavna uloga planera upita jeste izbor indeksa koji će biti korišćen za upit, postoje situacije kada planer upita ne uspeva da izabere indeks koji je najoptimalniji, najčešće posledica postojanja previše indeksa u kolekciji. U tim situacijama može se koristiti hint funkcija, pri čemu se kao parametar funkcije navodi indeks nad kojim bi upit trebalo da se izvrši, kao što je prikazano na primeru na Slici 6.

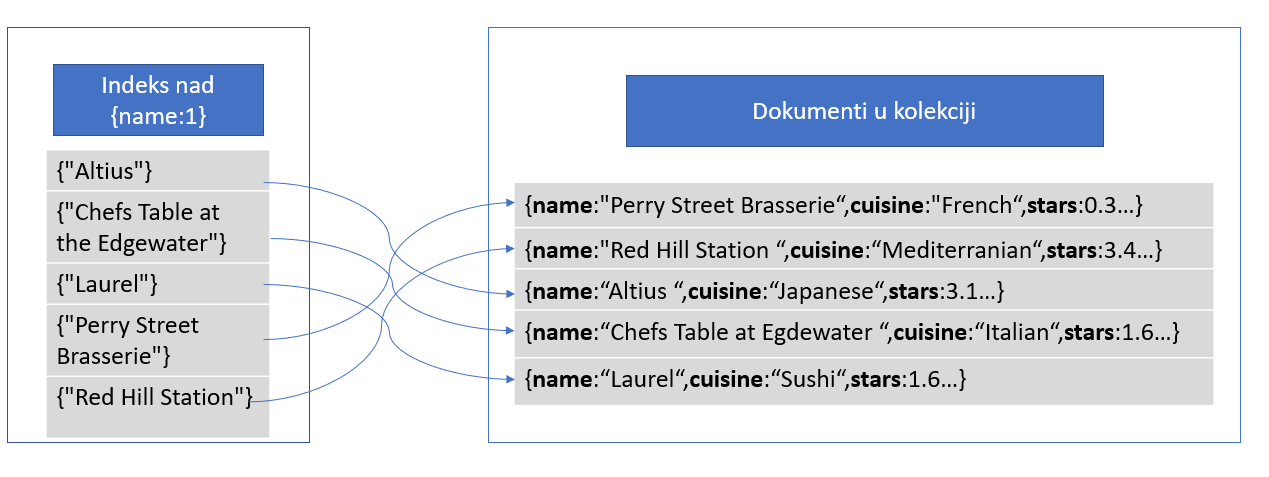


*Slika 6 – Prikaz hint funkcije*

Hint funkcija se treba oprezno koristiti, jer ukoliko dođe do potrebe za korišćenjem ove funkcije, treba prvo ispitati da li su optimalno kreirani indeksi, i da li nije došlo do gomilanja istih, pre korišćenja ove funkcije [10].

# Indeksi

Prvo, i najbitnije rešenje za problem sporih upita kod MongoDB-a jesu indeksi, koji i predstavljaju strukture koje se kreiraju radi ubrzavanja sporih upita. Indeksi predstavljaju svojeručni rečnik vrednosti nad određenim poljem, i omogućavaju ubrzanu pretragu za vrednošću na osnovu svoje strukture. Ukoliko, na primer vršimo pretragu za dokumentom koji za vrednost polja ime restorana ima određenu vrednost, MongoDB će pretraživanje vršiti po svim dokumentima, izvršavajući takozvani sken kolekcije, te će se sa porastom kolekcije, povećati i vreme koje je neophodno za vraćanje traženih dokumenata, odnosno za izvršavanje upita. Vreme koje je potrebno za izvršenje vremena je reda *O(n),* te je linearno proporcionalno od broju dokumenata u kolekciji. Postojanjem indeksa se pretraga dokumenata za vrednošću određenog atributa svodi na prolaz kroz niz ključeve koji čuvaju referencu na dokument u kome se nalaze, kao što je prikazano na slici na primeru baze restorana.



*Slika 7 – Prikaz indeksiranja na primeru kolekcije restorana*

Indeks se može kreirati nad jednim ili više polja, nad poljem \_id automatski postoji indeks. Moguće je imati više indeksa u okviru iste kolekcije, i ima ih smisla praviti ukoliko postoje upiti nad različitim poljima. MongoDB koristi B stablo za smeštanje indeksa. Ono što treba imati na umu da iako indeksi daju bolje performanse pri čitanju, oni produžavaju vreme koje je potrebno za upis, jer se možda sama struktura B stabla menja dodatim dokumentom, takođe izmenom ili brisanjem dokumenta jedan ili više stabala trebaju biti ponovo izbalansirana [11].

## Indeksi nad jednim poljem

Indeksi nad jednim poljem su najjednostavniji indeksi koji mogu biti kreirani u okviru MongoDB-a, da bi se kreirao indeks nad nekim poljem u kolekciji, neophodno je izvršenje samo jedne naredbe, *createIndex* u okviru konzole, gde je neophodno navesti ime polja i specificirati da li indeks treba biti sortiran u rastucem ili opadajućem redosledu, ili grafički, samo klikom u okviru MongoDB Compass alata, izborom polja iz selection box-a. Vrednost 1 specificira sortiranje u rastućem smeru, dok vrednost -1 specificira sortiranje u opadajućem smeru [12]. Indeks nad jednim poljem omogućava pronalaženje jedne ili više vrednosti, kao i opsega vrednosti.

Kao primer ubrzanja koje se može dobiti korišćenjem ovakvog indeksa, izvršen je upit za pretragom osoba po broju njihovog socijalnog osiguranja nad prethodno pomenutom kolekcijom ljudi. Inicijalno je izvršen upit nad kolekcijom bez indeksa, upit je prikazan na slici 8.



*Slika 8 – Upit nad kolekcijom ljudi pre kreiranja indeksa*

Upit treba da vrati sve dokumente koji sadrže traženu vrednost za polje *ssn*. Takođe da bi se dobilo više informacija o izvršenju upitu dodata je funkcija explain koja će prikazati u konzoli detalje izvršenja ovog upita. Na slici 9 i 10 su prikazana polja *queryPlanner*, odnosno *executionStats.*



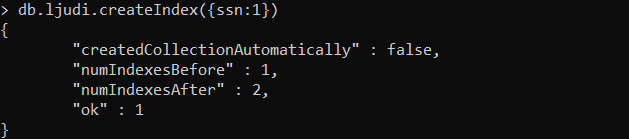
*Slika 9 – Query planner polje pre kreiranja indeksa*

Query planner ukazuje da se radi skeniranje cele kolekcije, što znači da se vrši provera kolekcije dokument po dokument. Nakon toga u konzoli se nalaze podaci o izvršenju u okviru *executionStats* polja.



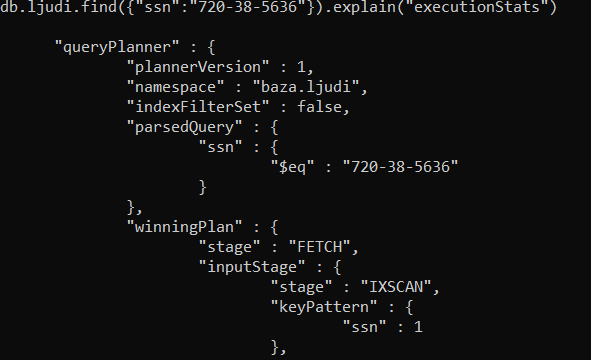
*Slika 10 – Execution stats polje pre kreiranja indeksa*

Može se uočiti da je 50474 dokumenata skenirano, što i predstavlja ukupan broj dokumenata u kolekciji, što je u skladu sa tim da se radi skeniranje cele kolekcije. Pregledom svih 50 474 dokumenta pronađen je i vraćen samo jedan dokument. Polje *totalKeysExamined* ukazuje da nijedan indeks nije pretražen, što je i logično jer indeks nije kreiran. Performanse u ovom slučaju su jako loše. Ovaj rezulat će se uporediti sa rezulatom u slučaju kada se indeks kreira nad poljem *ssn*.



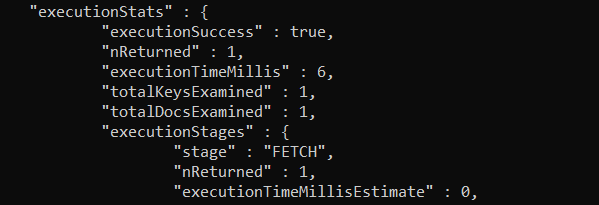
*Slika 11 – Kreiranje indeksa nad poljem ssn*

Kreiran je indeks čije su vrednosti ključeva poređane u rastućem redosledu. Ukoliko ne postoji *ssn* polje u dokumentu vrednost ključa će imati vrednost NULL. Zatim, ukoliko se ponovi isti upit dobijaju se sledeći rezultati.



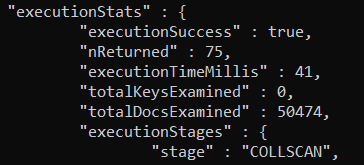
*Slika 12 – Query planner polje nakon kreiranja indeksa*

Može se uočiti da je u ovom slučaju *winningPlan* je skeniranje indeksa(*IXSCAN*). Pogledom na *executionStats* polje na Slici 13 može se uočiti poboljšanje u performansama.



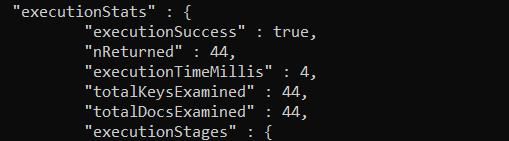
*Slika 13 – Execution stats polje nakon kreiranja indeksa*

Umesto pregleda preko 50 000 dokumenata, vidi se da je pregledan samo jedan dokument, prilikom čega je jedan dokument vraćen, ovaj rezultat predstavlja odličan odnos između broja pretraženih dokumenata i pronađenih. Indeks nad jednim poljem je u ovom slučaju imao smisla jer se upit izvršava samo nad poljem *ssn.* Ukoliko se izvrši upit nad istom kolekcijom za pretragu nekog drugog polja, postojeći indeks nad poljem *ssn* neće ubrzati pronalazak dokumenta, kao što se može videti na primeru ispod.



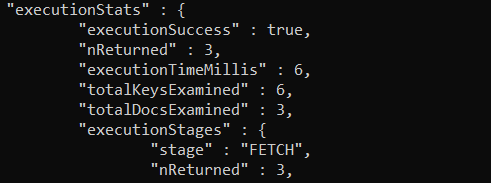
*Slike 14,15 – Upit za koji nije kreiran indeks i rezultati*

Kao što se može videti sa prethodne slike, svi dokumenti u kolekciji su bili pretraženi. Osim pronalaženja samo egzaktno zadatih vrednosti u upitu, indeksi daju bolje performanse i prilikom pretraživanja opsega vrednosti [13].



*Slike 16,17 – Upit za opsegom vrednosti*

Može se uočiti da se pretražilo samo onoliko dokumenata, koliko je vraćeno, nije bilo potrebe za pretragom drugih dokumenata, što predstavlja odličan rezultat. Osim pretrage opsega može se vršiti i pretraga za dokumentima koji imaju za vrednost *ssn* neku vrednost od skupa zadatih vrednosti, što je prikazano na slikama 18 i 19.



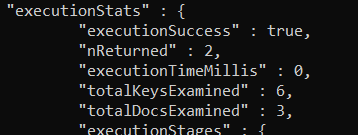
*Slike 17,18 – Upit za vrednošću iz skupa vrednosti*

Takođe kod ovakvih vrsta upita se može videti poboljšanje u performansama, jer se umesto cele kolekcije pretražilo samo 3 dokumenta, pri čemu je jednaki broj i vraćen. Može se uočiti da je i pristupano većem broju indeksa, ali je to posledica algoritma koji prebacuje vrednosti.

Poboljšanje performansi neće biti vidljivo samo prilikom upita nad vrednostima polja za koje postoji indeks, poboljšanje će se javiti i ukoliko se izvršava kompleksniji upit čija se pretraga zasniva na polju za koje je kreiran indeks, što se može videti na sledećem primeru. Indeks postoji nad poljem *ssn,*dok se traže dokumenti koji imaju određene vrednosti za *ssn,* kao i prezime prezime koje počinje sa M ili slovom koje se nalazi kasnije u abecedi.





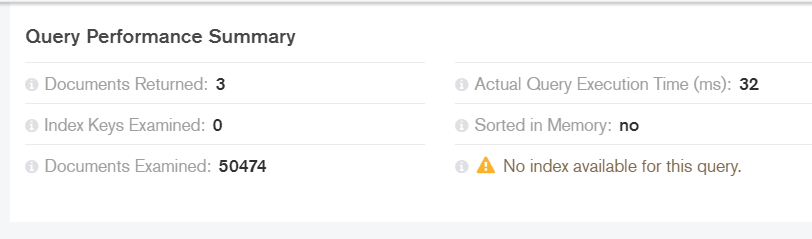


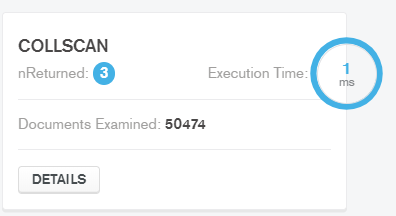
*Slike 19,20 i 21 – Efikasnost indeksa nad jednim poljem u kompleksnom upitu*

MongoDB DBMS u ovakvim slučajevima prvo filtrira dokumente na osnovu polja za koje postoji indeks, i zatim vrši pretragu po dokumentima po ostalim poljima koji se nalaze u upitu [14].

## Složeni indeksi

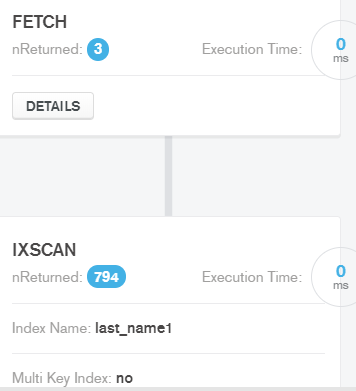
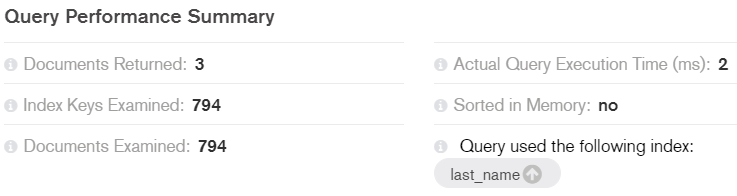
Nekada upiti zahtevaju pretragu na osnovu vrednosti više polja, što je prethodni primer prikazao, pri čemu je izabrano rešenje bilo korišćenje postojećeg indeksa za pronalaženje skupa dokumenta koji će biti skenirani, složenim indeksima se ovakav upit dodatno optimizuje. Složeni indeksi omogućavaju postojanje ključa koji je složen i koji se sastoji od navedenih polja u redosledu koji je naveden pri kreiranju indeksa. Indeksi su sortirani u rastućem ili opadajućem redosledu, u odnosu na način na koji je definisan indeks, sa time što se prvo vrši sortiranje u okviru najlevijeg polja. Na primeru kolekcije *ljudi*, je prikazana optimizacija upita korišćenjem složenih indeksa. Izvršavaće se pretraga dokumenata na osnovu imena i prezimena osoba. Na slikama 22, 23 i 24 su prikazani rezultati upita bez indeksa.





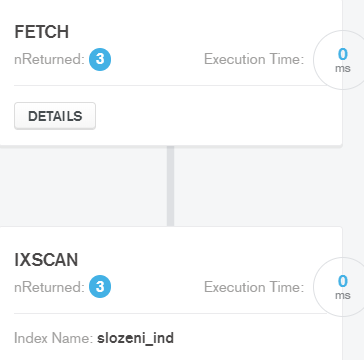
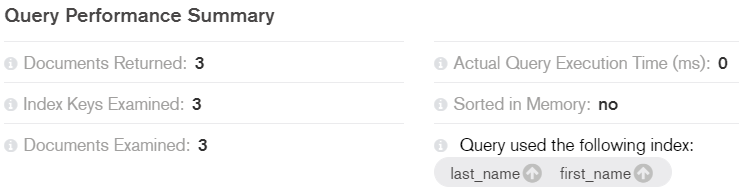
*Slike 22,23,24 – Pretraga za osobom na osnovu imena i prezimena bez indeksa*

Može se uočiti da je broj skeniranih dokumenata jednak broju dokumenata u kolekciji, i da je izvršena operacija skeniranja cele kolekcije, odnos između skeniranih i vraćenih podataka je jako veliki, te je ovo naznačava da se treba razmotriti upotreba indeksa. Na slikama 25, 26 je prikazan rezultat koji se dobija ukoliko se kreira indeks samo na jednom polju, polju prezimena.



*Slika 25,26 – Performanse pri upitu pretrage za osobom na osnovu imena i prezimena sa indeksom na polju prezime*

Ono što se odmah može uočiti jeste smanjenje vremena za izvršenje, koje se smanjilo 16 puta, ali još bolji prikaz optimizacije koja se dobija indeksiranjem, jeste poređenjem broja skeniranih i vraćenih dokumenata, ovaj odnos je znatno skladniji, ali složeni indeksi nude još bolje rešenje u ovoj situaciji. Na slikama 27,28 su prikazane performanse za isti upit, nakon kreiranja složenog indeksa nad poljima prezime i ime.



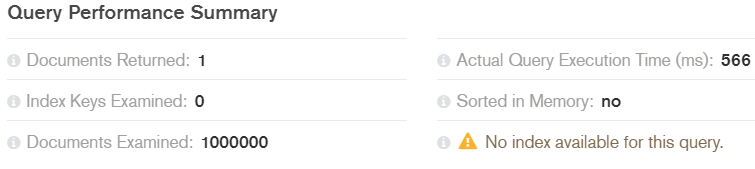
*Slika 27,28 – Performanse pri upitu pretrage za osobom na osnovu imena i prezimena sa složenim indeksom nad poljima prezime i ime redom*

U ovom slučaju je odnos između ispitanih dokumenata i vraćenih dokumenata 1-1, pri čemu je i broj ispitanih ključeva takođe samo tri. Bitno je napomenuti da vreme koje je potrebno za izvšenje nije 0ms, već je reda veličine manje od reda veličine ms. Takođe query planer je pronašao i opciju prikazanu iznad, za pretragu na osnovu jednog indeksa, ali je ona odbijena od strane optimizatora upita, kao manje optimalna.

U poslednjem primeru je prikazano kreiranje složenog indeksa nad poljima prezime i ime, i bitno je napomenuti, da će taj isti indeks poboljšati performanse u slučaju pretrage osoba samo po prezimenima, ali da se isto neće desiti u slučaju pretrage za imenom, jer su indeksi sortirani sleva na desno. Stoga prilikom kreiranja složenih indeksa treba razmisliti koje sve upite aplikacija podržava, i na osnovu toga odrediti redosled indeksa pri kreiranju indeksa [15].

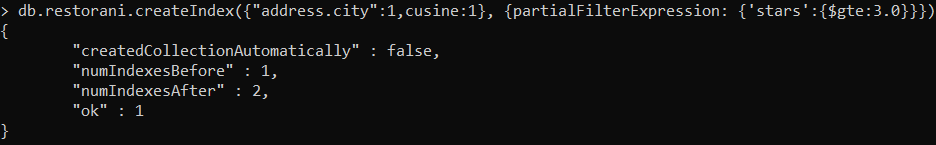
## Parcijalni indeksi

U nekim situacijama, nije potrebno kreiranje indeksa za celu kolekciju, ili je indeks koji se dobije kreiranjem indeksa nad celom kolekcijom, previše veliki, te ne može stati u memoriju, samim tim bi deo indeksa onda bio na disku, čime bi se izgubile prednosti u performansama dobijene kreiranjem indeksa. Takođe kreiranjem parcijalnih indeksa se smanjuje i uticaj dodavanja i menjanja dokumenata u kolekciju [16]. Parcijalni indeksi indeksiraju samo dokumente u kolekciji koji zadovoljavaju određeni uslov. Na sledećem primeru su prikazane prednosti kreiranja parcijalnih indeksa. Slika 30 prikazuje performanse upita nad kolekcijom restorani, prilikom pretrage po gradu, kuhinji i oceni.

S

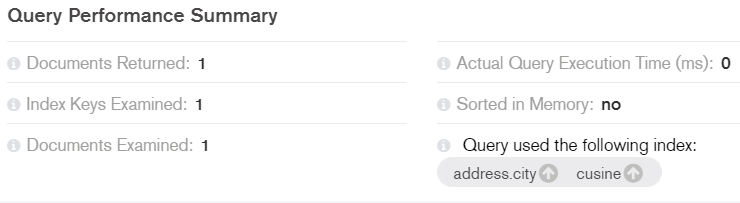
*Slike 29,30 - Performanse upita nad kolekcijom restorani, prilikom pretrage po gradu, kuhinji i oceni pre kreiranja parcijalnog indeksa*

Za ovaj upit se dobijaju jako loše performanse, zbog veličine same kolekcije, koja ima milion dokumenata, pri čemu je broj ispitanih dokumenata 1 000 000, jer se vrši skeniranje cele kolekcije. Ali kreiranje indeksa nad celom kolekcijom bi zauzimalo značajni deo memorije, pri čemu bi se potencijalno izgubile performanse koje bi bile dobijene kreiranjem indeksa, zbog nedovoljno mesta u memoriji za skladištenje indeksa. Shodno tome kreiran je parcijalni indeks nad poljem adresa, i tipa kuhinja, samo za restorane koji su ocenjeni sa više od tri zvezdi (jer ujedno i očekujemo češću pretragu nad većim vrednostima polja broja zvezdi) , čije je kreiranje prikazano na slici 31.



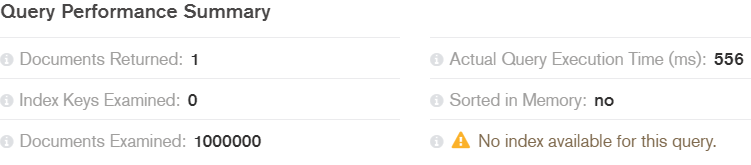
*Slika 31 – Kreiranje parcijalnog indeksa*

Na sledećoj slici se nalaze performanse pri istom upitu, nakon kreiranja parcijalnog indeksa.



*Slike 32,33 - Performanse upita nad kolekcijom restorani, prilikom pretrage po gradu, kuhinji i oceni nakon kreiranja parcijalnog indeksa*

Poboljšanje u performansama je vidljivo, jer se umesto skeniranja cele kolekcije, vrši samo pretraga na osnovu indeksa. Poboljšanje u performansama je vidljivo i u vremenu koje je neophodno za izvršenje upita. Ono što je bitno naglasiti jeste, da indeksi postoje samo za dokumente koje za polje stars imaju vrednost veću od 3.0. Ukoliko isti upit izvršimo, samo za restorane koji imaju vrednost za polje stars veće od 1.5, MongoDB će izvršiti skeniranje cele kolekcije, te neće doći do željenog poboljšanja u performansama, kao što se može videti na slikama 34 i 35 [17].

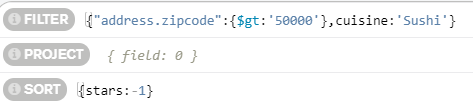


*Slike 34,35 - Performanse upita nad kolekcijom restorani, prilikom pretrage po gradu, kuhinji i oceni nakon kreiranja parcijalnog indeksa, nad poljima koja nisu obuhvaćena filterom*

# Optimizacija CRUD upita

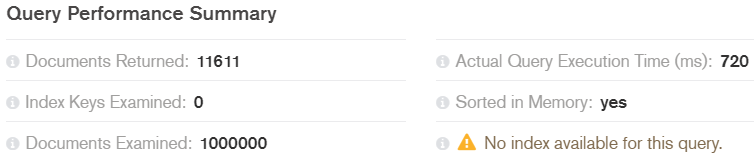
Navedeni su neki od načina za kreiranje indeksa, ali samo kreiranje indeksa nije dovoljno za postizanje optimalnih performansi. Iako MongoDB DBMS podržava kreiranje velikog broja indeksa, to ne znači da kreiranje indeksa nad svim poljima predstavlja optimalno rešenje, jer povećava memoriju koja je neophodna za smeštanje istih, kao i dužu proveru prilikom traženja optimalnog rešenja, prilikom čega se mogu javiti greške kod optimizatora upita. Prilikom pisanja optimizovanih upita treba se držati preporuke *Jednakost, sortiranje, opseg*, koja je preporučena od strane MongoDB projektanata. To znači da za kompleksne upite, prilikom kreiranja indeksa se prvo navode polja koja se u upitu ispituju na jednakost, zatim polja u odnosu na koja se vrši sortiranje u upitu (ukoliko postoji sortiranje), i tek na kraju polja čiji se opseg ispituje u upitu. U nastavku će biti prikazan tabelarni rezultat performansi, prilikom kreiranja različitih indeksa, kao i rezultat bez upotreba indeksa, na upitu koji sadrži i proveru jednakosti, sortiranje, kao i proveru opsega.

Na slici 36 je prikazan upit, u kojem se traže dokumenti koji sadrže restorane kojima je zip kod veći od 50 000, kuhinja “Sushi”, koji su sortirani u opadajućem redosledu u odnosu na broj zvezda.



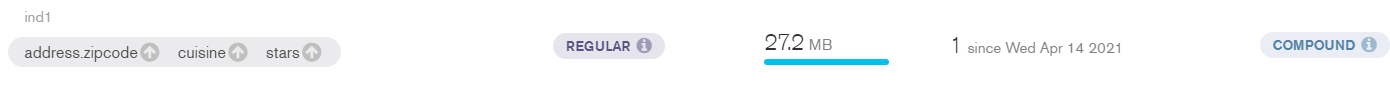
*Slika 36 – Upit koji pretražuje kolekciju restorana na osnovu polja adresa, i kuhinja i vrši sortiranje na osnovu polja stars*

Performanse pri ovom upitu su kao što se može videti na slici 37 su jako loše, gde osim skeniranja cele kolekcije, se vrši sortiranje u okviru same memorije, što samo po sebi predstavlja veoma skupu operaciju. Osim toga vreme koje je potrebno za izvršenje ovakvog upita je preveliko.



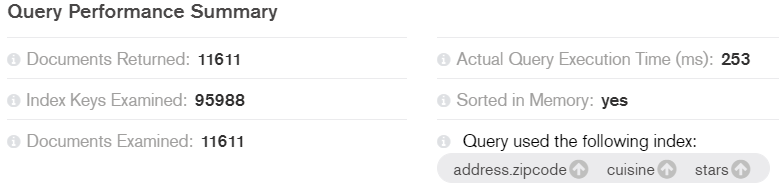
*Slika 37 – Performanse upita koji pretražuje kolekciju restorana na osnovu polja adresa, i kuhinja i vrši sortiranje na osnovu polja stars*

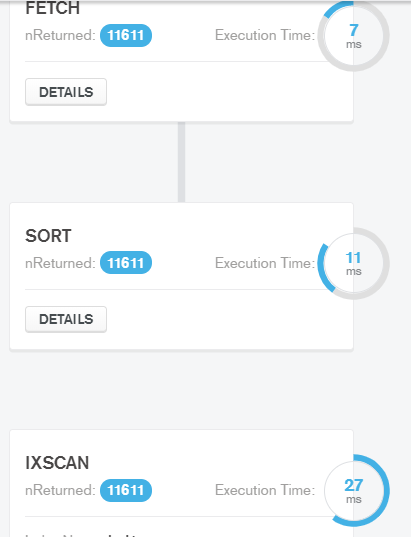
Ukoliko je ovakav upit jako čest za aplikaciju, treba napraviti indeks, ali indekse ne treba kreirati generički, već treba ispitati, koliko su efikasni indeksi koji se naprave. U sledećem primeru je prikazan primer kreiranja indeksa samo na osnovu upita, bez praćenja pravila *Jednakost, sortiranje, opseg*. Kreiran je indeks nad, redom sleva na desno *address.zipcode, cusine,* *stars* poljima, kao što je prikazano na slici.



*Slika 38 – Kreiranje indeksa nad poljima address.zipcode, cusine, stars redom*

Zatim, nakon izvršavanja upita, može se uočiti smanjenje broja dokumenta koji su ispitani, ali i jako veliki broj indeksa koji su su ispitani, u odnosu na broj vraćenih dokumenata, takođe vidljivo je da se ponovo izvršava sortiranje u okviru memorije, što dosta utiče na performanse. Ali je vreme izvršenja upita zanačajno smanjeno u odnosu na vreme koje je bilo potrebno pre bez postojanja indeksa. Informacije o upitu nakon kreiranja indeksa se nalaze na slickama 39 i 40.

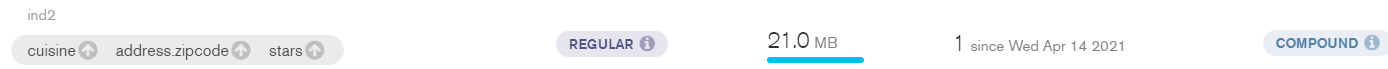




*Slike 39,40 – Performanse nakon kreiranja indeksa nad poljima address.zipcode, cusine, stars redom*

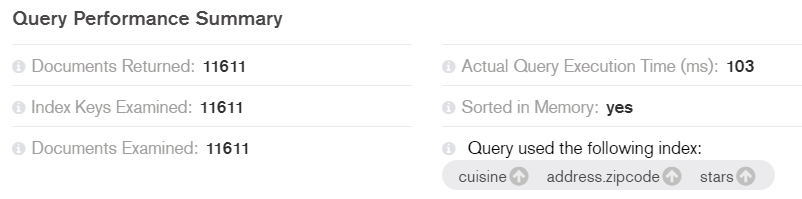
Upit prvo vrši skeniranje na osnovu indeksa, zatim sortiranje u memoriji, nakon čega se pribavljaju dokumenti i vraćaju kao rezultat.

Glavni problem u prethodnom rezultatu jeste broj indeksa koji su bili ispitani, a on je veliki, jer je prvobitni indeks nad poljem *address.zipcode*, ima veliki opseg vrednosti koji dolazi u obzir. Najbolje je prvo imati polje koje je jako selektivno (isključuje veliki broj rezultata), što je u sledećem primeru urađeno. Kreiran je indeks sličan prethodnom, samo sa razlikom da je *cusine* prvo polje u kreiranom indeksu, a *address.zipcode*, drugo.



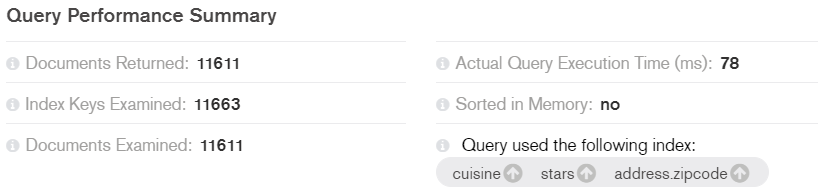
*Slika 41 – Kreiranje indeksa nad poljima cusine, address.zipcode, stars redom*

Zatim se posmatraju dobijene performanse. Vidljivo je poboljšanje u vremenu koje je potrebno za izvršenje upita, kao i u broju ispitanih indeksa, koji je jednak broju vraćenih dokumenata, ali se sortiranje idalje izvršava u memoriji, kao što se može videti na slici ispod.



*Slika 42– Performanse nakon kreiranja indeksa nad poljima cusine, address.zipcode, stars redom*

Razlog za to je što MongoDB vraća sortirane rezulate pri filtriranju i sortiranju ukoliko se ispred polja u odnosu na koje želimo sortirane rezulatate nalaze samo filtriranje na osnovu jednakosti, te je poslednje poboljšanje zamena mesta pri kreiranju indeksa poljima stars i address.zipcode.



*Slika 43– Performanse nakon kreiranja indeksa nad poljima cusine, stars, address.zipcode redom*

Ovakav indeks je doprineo najboljem rešenju, što je vidljivo na osnovu vremena izvršenja upita, pregledano je više indeksa, ali nije izvršeno sortiranje u memoriji. Pretraga većeg broja ključeva jeste tradeoff za manje vreme izvršenja i izbegavanje sortiranja u memoriji. Na sledećoj slici su tabelarno prikazane performase pri različitim indeksima.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Indeks** | **Bez indeksa** | ***{***  ***address.zipcode*:1*,***  ***cusine*:1*,***  ***stars*:1**  ***}*** | ***{***  ***cusine*:1*,***  ***address.zipcode*:1*,***  ***stars*:1**  ***}*** | ***{***  ***cusine*:1*,***  ***stars*:1**  ***address.zipcode*:1*,***  ***}*** |
| **Vreme izvršenja (ms)** | 720 | 253 | 103 | 78 |
| **Broj vraćenih dokumenata** | 11611 | 11611 | 11611 | 11611 |
| **Broj ispitanih dokumenata** | 1 000 000 | 95988 | 11611 | 11611 |
| **Broj ispitanih ključeva** | 0 | 11611 | 11611 | 11663 |
| **Sortiranje u memoriji** | da | da | da | ne |

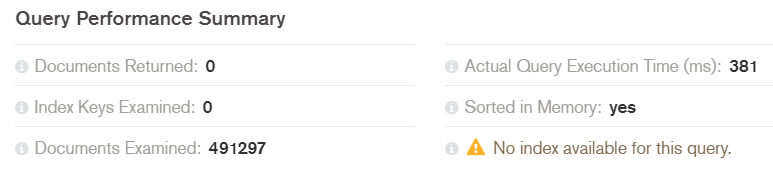
*Tabela 2 – Prikaz performansi bez indeksa, kao i sa različitim redosledima indeksa*

Jako je bitno detaljno pogledati, i isprobati različite indekse, posmatrajući parametre koji se nalaze u tabeli radi pronalaženja najoptimalnijeg rešenja za kreiranje indeksa u okviru upita [18].

# Ograničenje broja rezultata koji se vraćaju kao rezultat

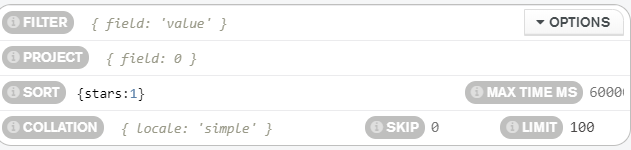
Jedan od vidova povećavanja performansi prilikom korišćenja MongoDB baze podataka jeste limitiranje broja rezultata koji se vraćaju, ukoliko nisu svi rezultati neophodni, kako bi se smanjilo opterećenje mreže, i samim tim povećale performanse upita. Ukoliko je poznat broj rezultata koji se vraća, broj željenih rezultata se označava u okviru funkcije limit. Često se koristi nakon sortiranja gde je najčešće potreban samo određeni broj elemenata nakon sortiranja [10]. Limit metodi može biti kao parametar zadata vrednost između do . Za vrednost 0, se smatra da ne postoji limit, dok negativne vrednosti označavaju slanje ekvivalentnog broja dokumenata, samo u okviru jednog batch-a. Ukoliko svi podaci nisu poslati u tom batch-u ostali podaci neće biti poslati [19]. U nastavku je dat prikaz poređenja samog sortiranja bez korišćenja metode limit nad kolekcijom restorani.

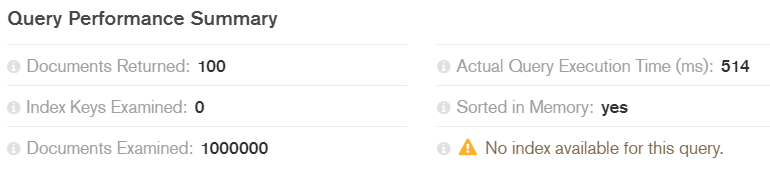




*Slike 44,45 – Performanse bez korišćenja limit metode*

Dalje je prikazan rezultat korišćenja metoda limit nakon sortiranja u kolekciji restorani. Može se uočiti da zapravo zahteva više vremena nego klasično sortiranje, ali treba imati u vidu da je prednost korišćenja ove funkcije u smanjenju zagušenja mrežnog saobraćaja.





*Slike 46,47 – Performanse prilikom korišćenja limit metode*

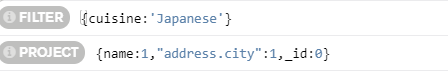
# Korišćenje projekcija

Bolje performanse se mogu dobiti vraćanjem samo onih delova dokumenata koji su neophodni [10]. MongoDB po difoltu vraća sva polja pronađenih rezultata. Da bi se kreirala projekcija, neophodno je: navođenje projekcija u MongoDB Compass-u ili navođenje u okviru vitičastih zagrada polja koja trebaju biti prisutna sa oznakom 1 u okviru komandne linije. Po difoltu se polje *\_id* uvek vraća, može se isključiti slanje polja *\_id*, time što se navede *\_id:0*, u okviru naredbe za projekciju. Na sledećoj slici je prikazan rezultat vraćanja svih polja dokumenta koji imaju vrednost “Japanese” za polje *cusine.* Performanse se neće ogledati u samom izvršenju na serveru već, smanjenju mrežnog saobraćaja i količine podataka koja se šalje.



*Slika 48 – Ceo dokument jedne osobe*

Ukoliko je neophodno samo nekoliko polja iz ovog dokumenta, a broj pronađenih dokumenata je veliki, bespotrebno će se mreža opteretiti, te klijent će duže čekati na tražene podatke. Na slici ispod je prikazana projekcija, kao i primer polja koja se šalju iz dokumenta.



*Slika 49 – Projekcija nad poljima ime i address.city*



*Slika 50 – Rezultat nakon projekcije nad poljima ime i address.city*

Razlika u količini poslatih podataka samo na nivou jednog dokumenta je veoma vidljiva, sa većim brojem dokumenta koji trebaju biti vraćeni, će se više videti benefiti korišćenja projekcija [10].

# Inkrement operator

MongoDB sadrži operator za inkrementiranje i dekrementiranje *$inc.* Operator inkrementira ili dekrementira vrednost polja na serverskoj strani, umesto selekcije dokumenta, modifikacije istog, i zatim upisivanja celog dokumenta na server. Osim štednje u performansama *$inc* operator omogućava izbegavanja uslova trke, prilikom inkrementa istog dokumenta od strane dva korisnika [10].

# Zaključak

MongoDB DBMS je sam po sebi poprilično oprimizovan, i shodno svojoj arhitekturi, kao i strukturi svojih šema nudi odlične alate za optimizaciju upita. Optimizacija upita kod MongoDB DBMS-a se najviše svodi na kreiranje indeksa. Tu mogućnost pruža već ugrađeni optimizator koji procenjuje koji je indeks najbolje iskoristiti pre izvršenja bilo koje naredbe. Ne sme se zaboraviti ni fleksibilnost koju nudi MongoDB DBMS, koji daje mogućnost override-ovanja optimizatora korišćenjem hint funkcije. Iako deluje jako lako izvršiti optimizaciju upita, samo kreiranjem indeksa, u radu su prikazani neki od indeksa koje MongoDB DBMS nudi, kao i koliko je različitih planova izvršenja moguće kreirati na osnovu njih, i koliko se međusobno razlikuju rezulati u performansama samo zbog zamene mesta dva polja pri kreiranju indeksa. Treba paziti na količinu kreiranih indeksa, jer se gube benefiti kreiranja indeksa ukoliko postoji preveliki broj indeksa, ili ukoliko indeks ne staje u RAM memoriju. Takođe rad se osvrnuo i na neke hardverske karakteristike na koje treba obratiti pažnju ukoliko se radi sa MongoDB DBMS-om, pri čemu je možda najbitnija karakteristika MongoDB DBMS-a, njegova fokusiranost ka RAM memoriji, te RAM memorija predstavlja možda najbitniju hardversku karakteristiku koja se treba posmatrati prilikom pokretanja servera. Takođe rad prikazuje na koji način smanjenje nepotrebnih podataka može povećati performanse upita.

# Literatura

1. Query Optimization in Relational Algebra - <https://www.geeksforgeeks.org/query-optimization-in-relational-algebra/> (Pristup 10.04.2021)
2. Liberios Vokorokos, Matúš Uchnár, Ľubor Leščišin Performance optimization of applications based on non-relational databases - 2016 International Conference on Emerging eLearning Technologies and Applications (ICETA), ISBN: 978-1-5090-4701-7
3. What is MongoDB? Introduction, Architecture, Features & Example - <https://www.guru99.com/what-is-mongodb.html> (Pristup 10.04.2021)
4. M201 Course MongoDB University - <https://university.mongodb.com/mercury/M201/2021_March_30/chapter/Chapter_1_Introduction/lesson/58ab5626d280e42669e6a604/lecture> (Pristup 10.04.2021)
5. Hardware Considerations & Configurations Part 1 -https://university.mongodb.com/mercury/M201/2021\_March\_30/chapter/Chapter\_1\_Introduction/lesson/58ab5626d280e4266a1881c5/lecture (Pristup 10.04.2021)
6. RAID 10 – What's RAID 10 And Why Should I Use It? - [https://www.acronis.com/en-us/articles/whats-raid10-and-why-should-i-use-it/](https://www.acronis.com/en-us/articles/whats-raid10-and-why-should-i-use-it/%20) (Pristup 10.04.2021)
7. Query Plans -[https://university.mongodb.com/mercury/M201/2021\_March\_30/chapter/Chapter\_3\_Index\_Operations/lesson/58b4c2b4d280e431d08fffcf/lecture](https://university.mongodb.com/mercury/M201/2021_March_30/chapter/Chapter_3_Index_Operations/lesson/58b4c2b4d280e431d08fffcf/lecture%20) (Pristup 11.04.2021)
8. Query Plans - <https://docs.mongodb.com/manual/core/query-plans/?jmp=university> (Pristup 11.04.2021)
9. Understanding Explain -[https://university.mongodb.com/mercury/M201/2021\_March\_30/chapter/Chapter\_2\_MongoDB\_Indexes/lesson/58b4c2b4d280e431d13d608b/lecture](https://university.mongodb.com/mercury/M201/2021_March_30/chapter/Chapter_2_MongoDB_Indexes/lesson/58b4c2b4d280e431d13d608b/lecture%20) (Pristup 11.04.2021)
10. Optimize Query Performance – [https://docs.mongodb.com/manual/tutorial/optimize-query-performance-with-indexes-and-projections/](https://docs.mongodb.com/manual/tutorial/optimize-query-performance-with-indexes-and-projections/%20) (Pristup 11.04.2021)
11. Introduction to indexes - <https://university.mongodb.com/mercury/M201/2021_March_30/chapter/Chapter_2_MongoDB_Indexes/lesson/58ab5626d280e426708ee3da/lecture> (Pristup 11.04.2021)
12. Single Field Indexes - [https://docs.mongodb.com/manual/core/index-single/](https://docs.mongodb.com/manual/core/index-single/%20) (Pristup 11.04.2021)
13. Single Field Indexes Part 1 -[https://university.mongodb.com/mercury/M201/2021\_March\_30/chapter/Chapter\_2\_MongoDB\_Indexes/lesson/58ab5626d280e426729e10d9/lecture](https://university.mongodb.com/mercury/M201/2021_March_30/chapter/Chapter_2_MongoDB_Indexes/lesson/58ab5626d280e426729e10d9/lecture%20) (Pristup 11.04.2021)
14. Single Field Indexes Part 2 -[https://university.mongodb.com/mercury/M201/2021\_March\_30/chapter/Chapter\_2\_MongoDB\_Indexes/lesson/58ab5626d280e4267361330e/lecture](https://university.mongodb.com/mercury/M201/2021_March_30/chapter/Chapter_2_MongoDB_Indexes/lesson/58ab5626d280e4267361330e/lecture%20) (Pristup 11.04.2021)
15. Querying on Compound Indexes Part 1 - <https://university.mongodb.com/mercury/M201/2021_March_30/chapter/Chapter_2_MongoDB_Indexes/lesson/58ab5627d280e42675e8b835/lecture> (Pristup 11.04.2021)
16. Partial Indexes - <https://docs.mongodb.com/manual/core/index-partial/?jmp=university> (Pristup 11.04.2021)
17. Partial Indexes - <https://university.mongodb.com/mercury/M201/2021_March_30/chapter/Chapter_2_MongoDB_Indexes/lesson/58ab5627d280e42679d3b956/lecture> (Pristup 11.04.2021)
18. Optimizing your CRUD Operations -[https://university.mongodb.com/mercury/M201/2021\_March\_30/chapter/Chapter\_4\_CRUD\_Optimization/lesson/58b4c2b4d280e431d8e9f296/lecture](https://university.mongodb.com/mercury/M201/2021_March_30/chapter/Chapter_4_CRUD_Optimization/lesson/58b4c2b4d280e431d8e9f296/lecture%20) (Pristup 12.04.2021)
19. Cursor.limit() - <https://docs.mongodb.com/manual/reference/method/cursor.limit/#mongodb-method-cursor.limit>