*Elektronski fakultet Niš*



*Sistemi za upravljanje bazama podataka* **Interna struktura i organizacija indeksa kod Mongo baze podataka**

Mentor:

prof. dr Aleksandar Stanimirović

Student:

David Bosnić 1289

Sadržaj:

[Uvod 3](#_Toc101897361)

[Indeksi 4](#_Toc101897362)

[Indeksiranje kod Mongo baze podataka 10](#_Toc101897363)

[Primer funkcionisanja indeksa kod MongoDB-a 24](#_Toc101897364)

[Zaključak 30](#_Toc101897365)

[Reference 31](#_Toc101897366)

Uvod

Sve je veći broj uređaja koji su povezani na internet i koji generišu veliku količinu podataka. Sa porastom količine podataka, nastaje problem njihovog skladištenja, pretraživanja i izvlačenja korisnih informacija iz njih.

Indeksi su jedan od najvažnijih alata koje administrator baze podataka može da koristi da bi svesno pomogao samoj bazi podataka i poboljšao njene performanse. Cilj ovog rada je istraživanje indeksa kod Mongo baze podataka i praktična realizacija i prikaz rada sa indeksima kod ove baze podataka.

MongoDB je sistem za upravljanje bazom podataka orijentisan na dokumente koji omogućava skladišćenje velike količine podataka u okviru dokumenata, koji mogu da variraju u pogledu veličine i strukture. MongoDB ima moćan mehanizam upita koji omogućava filtriranje dokumenata na osnovu specifičnih kriterijuma. Međutim, kako MongoDB kolekcija raste, traženje dokumenata može postati veoma kompleksna i spora. Fleksibilnost koju MongoDB nudi u pogledu upita može otežati bazi podataka da predvidi koje će se vrste upita najčešće koristiti. Sistem mora biti spreman za pronalaženje dokumenata bez obzira na veličinu zbirke podataka. Zbog ovoga, količina podataka koji se čuvaju u kolekciji direktno utiče na performanse pretrage. Što je veći skup podataka, to je za Mongo bazi podataka teže da pronađe dokumente koji odgovaraju upitu.

Indeksi

Indeksiranje je način da se optimizuju performanse baze podataka minimiziranjem broja pristupa disku koji su potrebni kada se upit obrađuje. Indeks baze podataka je struktura podataka koja poboljšava performanse operacija preuzimanja podataka u tabeli baze podataka po cenu dodatnog upisivanja i prostora za skladištenje. Indeksi se koriste za brzo lociranje podataka bez potrebe za pretraživanjem svakog reda u okviru neke tabele baze podataka svaki put kada se pristupa toj tabeli baze podataka. Indeksi se mogu kreirati korišćenjem jedne ili više kolona tabele baze podataka, obezbeđujući osnovu za brzo nasumično traženje i efikasan pristup uređenim podacima.

Indeksi se kreiraju pomoću nekoliko kolona baze podataka.

* Prva kolona je ključ za pretragu koji sadrži kopiju primarnog ključa ili ključa kandidata tabele. Ove vrednosti se čuvaju u sortiranom redosledu tako da se odgovarajućim podacima može brzo pristupiti.
* Druga kolona je referenca podataka ili pointer koji sadrži skup pokazivača koji drže adresu bloka diska gde se ta određena vrednost ključa može naći.



*Slika 1 – Struktura indeksa*

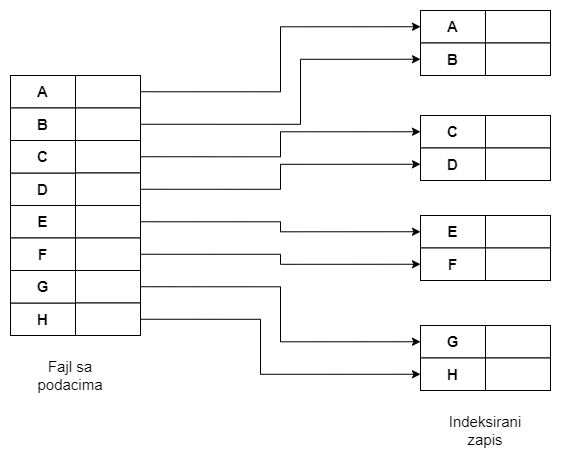
Atributi indeksiranja su:

* + **Tipovi pristupa** – Ovo se odnosi na tip pristupa kao što je pretraga zasnovana na vrednosti, pristup opsegu, itd.
  + **Vreme pristupa** – Odnosi se na vreme potrebno za pronalaženje određenog elementa podataka ili skupa elemenata.
  + **Vreme umetanja** – Odnosi se na vreme potrebno da se pronađe odgovarajući prostor i umetnu novi podaci.
  + **Dodatan prostor** – Odnosi se na dodatni prostor potreban indeksu.

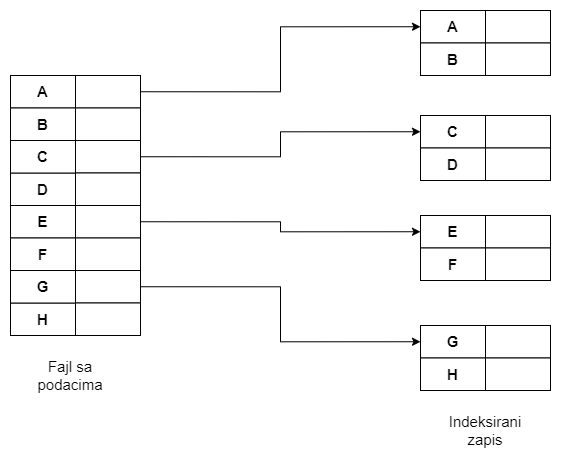
Generalno postoje dva tipa mehanizama organizacije datoteka koje prate metode indeksiranja za skladištenje podataka:

* Sekvencijalna organizacija datoteka ili uređeni indeksni fajlovi
* Heš organizacija fajla

Sekvencijalna organizacija datoteke

U ovom slučaju indeksi su zasnovani na sortiranom redosledu vrednosti. To su generalno brzi i tradicionalniji tip mehanizama za skladištenje. Ova poređana ili sekvencijalna organizacija datoteka može čuvati podatke u gustom ili razređenom formatu. Kod prvog formata, koji je prikazan na slici 2, se za svaku vrednost ključa za pretragu u datoteci podataka čuva indeksni zapis. Ovaj zapis sadrži ključ za pretragu i takođe referencu na prvi zapis podataka sa tom vrednošću ključa za pretragu. Kod drugog formata indeksni zapis se pojavljuje samo za nekoliko stavki u datoteci sa podacima. Svaka stavka pokazuje na blok kao što je i prikazano na slici 3. Da bi se locirao zapis, potrebno je da se pronađe indeksni zapis sa najvećom vrednošću ključa za pretragu koja je manja ili jednaka vrednosti ključa za pretragu koji se traži. Počinje se od zapisa na koji ukazuje zapis indeksa i prolazi se kroz pokazivače u datoteci dok se ne pronađe željeni zapis.

*Slika 2 – Čuvanje podataka u gustom formatu*

**

*Slika 3 – Čuvanje podataka u razuđenom formatu*

Heš organizacija fajla

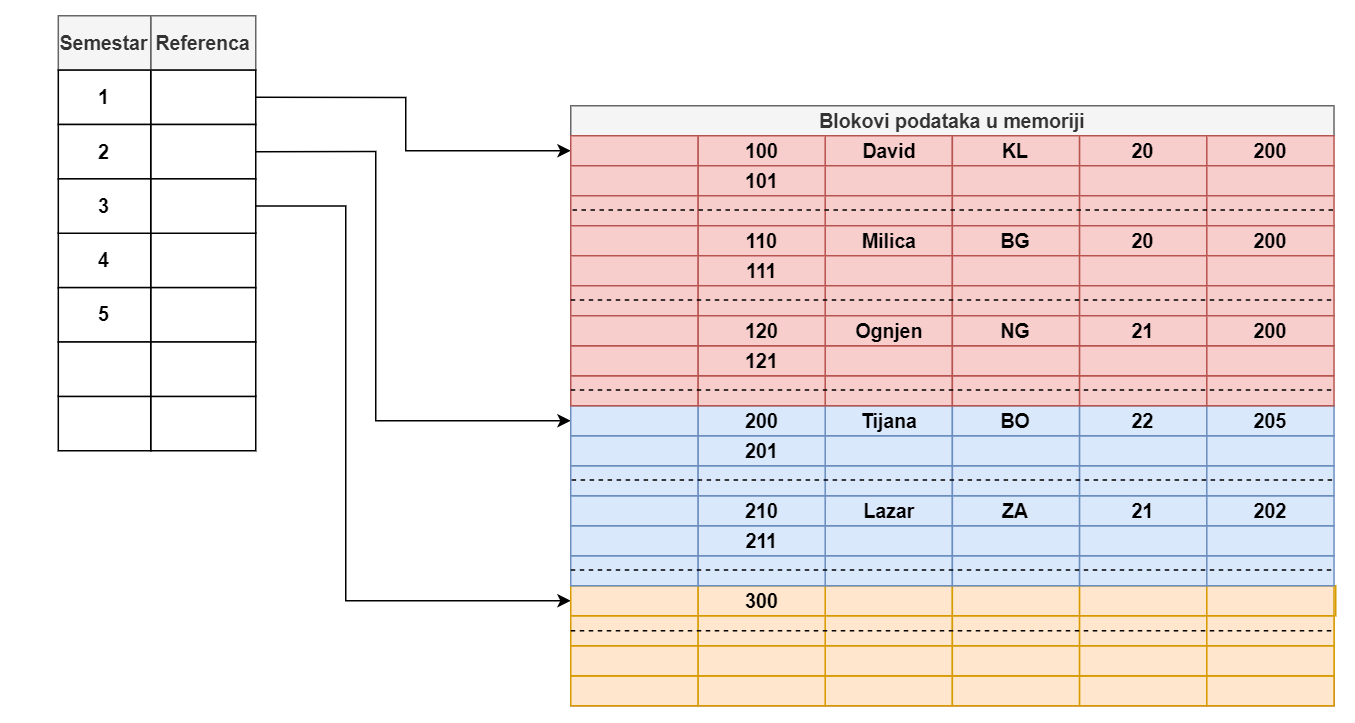
Indeksi se zasnivaju na vrednostima koje su ravnomerno raspoređene u rasponu grupa. Korpe kojima se dodeljuje vrednost određuje funkcija koja se zove heš funkcija. Heš funkcija se primenjuje na kolone ili atribute da bi se dobila adresa bloka. Zapisi se skladište nasumično. Poznata je i kao direktna ili nasumična organizacija datoteka. Ako je generisana heš funkcija u koloni koja se smatra ključem, onda se kolona nazva heš ključ, a ako je generisana heš funkcija u koloni koja se ne smatra ključem, kolona se nazva heš kolona.

Metode indeksiranja

Postoje prvenstveno tri metode indeksiranja:

* Klasterizovano indeksiranje
* Neklasterizovano ili sekundarno indeksiranje
* Indeksiranje na više nivoa

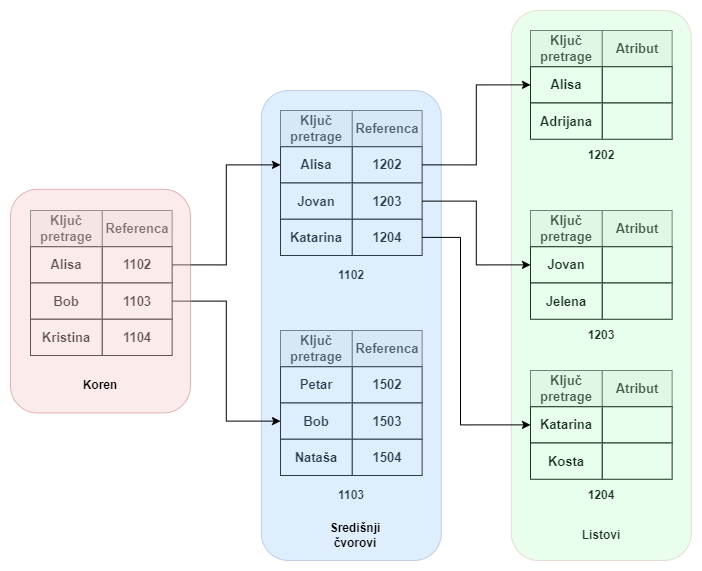
Klasterizovano indeksiranje

Kada je više od dva zapisa uskladišteno u istoj datoteci, ovi tipovi skladištenja poznati su kao indeksiranje klastera. Korišćenjem klasterizovanog indeksiranja možemo smanjiti troškove pretraživanja jer se više zapisa koji se odnose na istu stvar čuvaju na jednom mestu, a takođe daje učestalo spajanje više tabela (zapisa). Indeks klastera je definisan na uređenoj datoteci podataka. Ovakva datoteka je uređena po polju koje nije ključ. U nekim slučajevima, indeks se kreira na koloni koja nije primarni ključ, koja možda ne sadrži jedinstvene vrednosti. U takvim slučajevima, da bi se brže identifikovali zapisi, grupisaće se dve ili više kolona zajedno da bi se dobile jedinstvene vrednosti i kreirao indeks od njih. Ovaj metod je poznat kao indeks klastera. U osnovi, zapisi sa sličnim karakteristikama se grupišu i nad tim grupama se kreiraju indeksi. Primer na kome su studenti koji uče u svakom semestru grupisani zajedno (grupisani su studenti 1. semestra, 2. semestra, 3. semestra itd.) prikazan je na slici 4 .

*Slika 4 – Primer klasterizovanog indeksiranja*

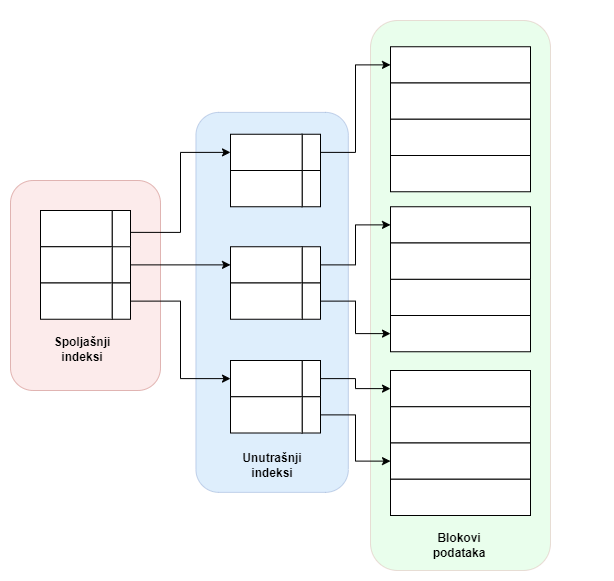
Vrsta klasterizovanog indeksiranja u kojoj se podaci sortiraju prema ključu za pretragu, a primarni ključ tabele baze podataka se koristi za kreiranje indeksa naziva se primarno indeksiranje. To je podrazumevani format indeksiranja kod sekvencijalne organizacije datoteka. Kako su primarni ključevi jedinstveni i čuvaju se na sortiran način, izvođenje operacije pretraživanja je prilično efikasno.

Neklasterizovano indeksiranje

Neklasterizovani ili sekundarni indeks nam samo govori gde leže podaci, tj. daje nam listu virtuelnih pokazivača ili referenci na lokaciju na kojoj su podaci zapravo uskladišteni. Podaci se fizički ne čuvaju po redosledu indeksa. Umesto toga, podaci su prisutni u krajnjim čvorovima, to jest listovima stabla. Jedan od primera prethodno navedenog je stranica sa sadržajem knjige. Svaki unos nam daje broj stranice ili lokaciju sačuvanih informacija. Stvarni podaci ovde (informacije na svakoj stranici knjige) nisu organizovani, ali imamo uređenu referencu (stranicu sa sadržajem) gde zapravo leže tačke podataka. Kod neklasterizovanog indeksiranja se koristi jedino gusti format čuvanja podataka, jer podaci nisu fizički organizovani u skladu sa razuđenim formatom. To zahteva više vremena u poređenju sa klasterizovanim indeksom, jer se radi izvestan dodatni posao da bi se izvukli podaci daljim praćenjem pokazivača. U slučaju klasterizovanog indeksa, podaci su direktno prisutni ispred **indeksa. Primer neklasterizovanog indeksiranja dat je na slici 5.

*Slika 5 – Primer neklasterizovanog indeksiranja*

Indeksiranje na više nivoa

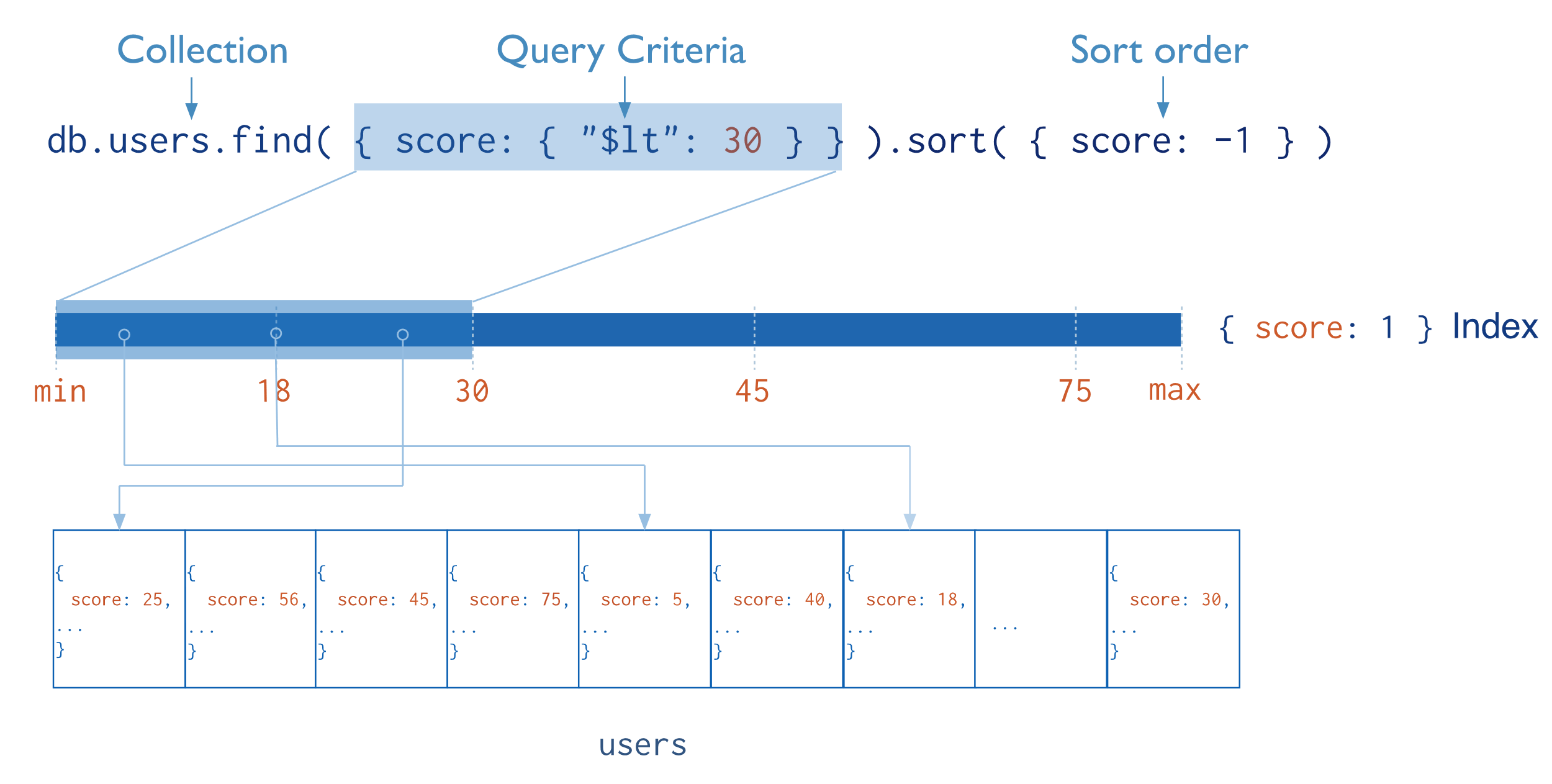
Sa rastom veličine baze podataka rastu i indeksi. Pošto se indeks čuva u glavnoj memoriji, indeks na jednom nivou može postati prevelik za skladištenje sa višestrukim pristupima disku. Indeksiranje na više nivoa odvaja glavni blok u različite manje blokove tako da se isti mogu uskladištiti u jednom bloku. Spoljni blokovi su podeljeni na unutrašnje blokove koji su zauzvrat usmereni na blokove podataka. Ovo se lako može sačuvati u glavnoj memoriji sa manje troškova. Grafički prikaz strukture ovakvog načina indeksiranja dat je na slici 6.

*Slika 6 – Grafički prikaz strukture indeksiranja na više nivoa*

Indeksiranje kod Mongo baze podataka

*MongoDB* je vodeća *NoSQL* baza podataka napisana u *C++* programskom jeziku. *NoSQL* se koristi kao alternativa tradicionalnim relacionim bazama podataka i veoma su korisne za rad sa velikim skupovima distribuiranih podataka. *MongoDB* je alatka koja može da upravlja informacijama orijentisanim na dokumente, čuva ili preuzima informacije. Visoko je skalabilan i pruža visoke performanse i dostupnost. Radi na konceptu zbirki i dokumenata. Kolekcija u *MongoDB*-u je grupa povezanih dokumenata koji su povezani zajedno. Kolekcija ne prati nijednu šemu koja je jedna od izuzetnih karakteristika *MongoDB*-a.

*MongoDB* podržava različite oblike podataka. To je jedna od mnogih tehnologija nerelacionih baza podataka koje su se pojavile sredinom 2000-ih obično za upotrebu u aplikacijama velikih podataka i drugim poslovima obrade koji uključuju podatke koji se ne uklapaju dobro u rigidni relacioni model. Umesto korišćenja tabela i redova kao u relacionim bazama podataka, *MongoDB* arhitektura se sastoji od kolekcija i dokumenata.

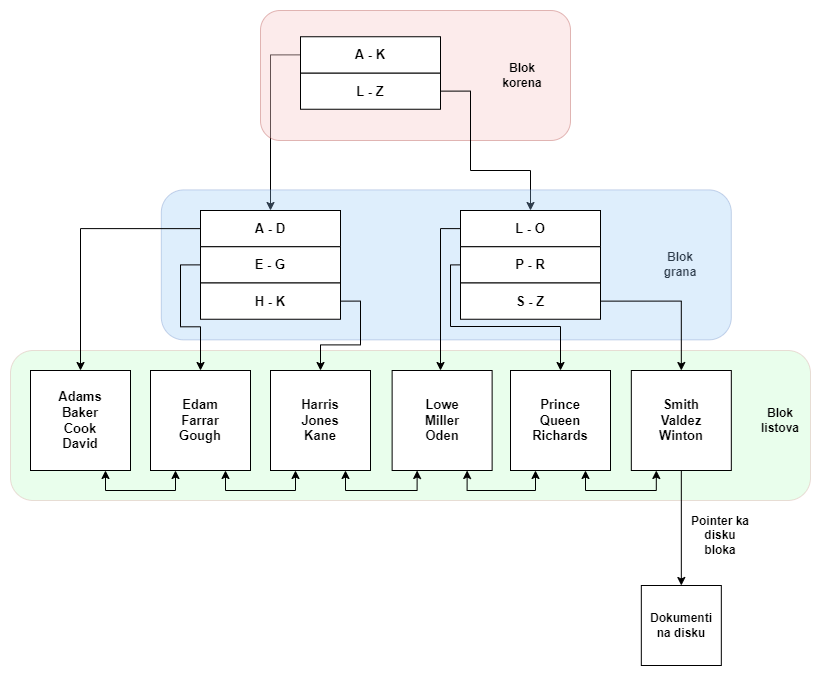
Indeksi su specijalne strukture podataka koje čuvaju mali deo skupa podataka kolekcije u obliku koji se lako može preći. *MongoDB* koristi indeksiranje kako bi obrada upita bila efikasnija. Ako nema indeksiranja, onda *MongoDB* mora da skenira svaki dokument u kolekciji i da preuzme samo one dokumente koji odgovaraju upitu. Indeksi su posebne strukture podataka koje čuvaju neke informacije vezane za dokumente tako da *MongoDB*-u postaje lako da pronađe pravu datoteku podataka. Indeksi su uređeni prema vrednosti polja navedenog u indeksu. Indeks čuva vrednost određenog polja ili skupa polja, uređenih prema vrednosti polja. Redosled unosa indeksa podržava efikasna podudaranja jednakosti i operacije upita zasnovane na opsegu. Pored toga, *MongoDB* može da vrati sortirane rezultate korišćenjem redosleda u indeksu. Dijagram na slici 7 ilustruje upit koji pronalazi i sortira odgovarajuće dokumente pomoću indeksa.

*Slika 7 – Dijagram koji ilustruje upit koji pronalazi i sortira odgovarajuće dokumente pomoću indeksa*

U osnovi, indeksi u *MongoDB* su slični indeksima u drugim sistemima baza podataka. *MongoDB* definiše indekse na nivou kolekcije i podržava indekse za bilo koje polje dokumenata u *MongoDB* kolekciji. *MongoDB* kreira jedinstveni indeks u polju (**\_id)** tokom kreiranja kolekcije. Indeks **\_id** sprečava klijente da ubace dva dokumenta sa istom vrednošću za polje **\_id**. Ne može se ukloniti indeks za ovo polje. U *shard* klasterima, ako se ne koristi polje **\_id** kao *shard* ključ, aplikacija mora da obezbedi jedinstvenost vrednosti u polju **\_id** da bi sprečila greške. Ovo se najčešće radi korišćenjem standardnog automatski generisanog *ObjectId*-a.

Struktura indeksa

B-stablo („uravnoteženo stablo“) je podrazumevana struktura indeksa *MongoDB*-a. Ispod, na slici 8, je pregled visokog nivoa strukture indeksa B-stabla.



*Slika 8 – Primer indeksiranja pomoću B-stabla*

Indeks B-stabla ima hijerarhijsku strukturu stabla. Na vrhu stabla je blok korena. Ovaj blok sadrži pokazivače na odgovarajući blok grane za bilo koji dati opseg vrednosti ključa. Blok grane će obično pokazivati na odgovarajući blok lista za specifičniji opseg ili, za veći indeks, pokazivati na drugi blok grane. Blok lista sadrži listu ključnih vrednosti i pokazivače na lokaciju dokumenata na disku.

Ako je potrebno da se pristupi zapisu za „Baker“, sistem bi prvo konsultovao blok korena. Blok korena bi rekao da se vrednosti ključeva koji počinju sa A do K čuvaju u krajnjem levom bloku grane. Pristupajući ovom bloku grane, saznaje se da su ključne vrednosti koje počinju sa A do D smeštene u krajnjem levom bloku lista. Konsultujući ovaj blok lista, pronalazi se vrednost „Baker“ i njenu pridruženu lokaciju na disku, koja bi se onda koristila za pronalaženje dotičnog dokumenta.

Blokovi lista sadrže veze i na prethodni i na sledeći blok lista. Ovo omogućava da se skeniraju indeksi u rastućem ili opadajućem redosledu i omogućava da se upiti za opseg koji koriste *$gt* ili *$lt* operatore obrađuju pomoću indeksa.

Indeksi B-stabla imaju sledeće prednosti u odnosu na druge strategije indeksiranja:

* Pošto je svaki listni čvor na istoj dubini, performanse su veoma predvidljive. U teoriji, nijedan dokument u kolekciji neće biti udaljen više od tri ili četiri I/O.
* B-stabla nude dobre performanse za velike kolekcije jer je dubina najviše četiri (jedan blok korena, dva nivoa blokova grana i jedan nivo bloka lista). Generalno, nijednom dokumentu neće biti potrebno više od četiri I/O za lociranje. U stvari, pošto će blok korena skoro uvek biti već učitan u memoriju, a blokovi grana obično učitani u memoriju, stvarni broj čitanja fizičkog diska je obično samo jedan ili dva.
* Indeks B-stabla podržava upite za opseg, kao i tačna pretraživanja. Ovo je moguće zbog veza sa prethodnim i sledećim blokovima lista.
* Indeks B-stabla pruža fleksibilne i efikasne performanse upita. Međutim, održavanje B-stabla pri promeni podataka može biti skupo. Na primer, ako se umetne dokument sa ključnom vrednošću „NIVEN“ u kolekciju sa prethodnog dijagrama. Da bi se izvršilo ubacivanje u kolekciju, mora se dodati novi unos u blok „L-O“. Ako u ovom bloku ima slobodnog prostora, onda je trošak znatan, ali možda ne prevelik. Ali ako u bloku nema slobodnog prostora dolazi do problema. Ako u bloku lista nema slobodnog prostora za novi unos, onda je potrebna podela indeksa. Mora se dodeliti novi blok i polovina unosa u postojećem bloku premestiti u novi blok. Pored toga, postoji zahtev za dodavanje novog unosa u blok grane (kako bi ukazali na novokreirani blok lista). Ako u bloku grane nema slobodnog prostora, onda se blok grane takođe mora podeliti. Ova podela indeksa je skupa operacija: novi blokovi moraju biti dodeljeni i unosi indeksa premešteni iz jednog bloka u drugi.

Svojstva indeksa

*MongoDB* nudi ogroman broj svojstava koje mogu posedovati indeksi. Jedno od generalno osnovnih svojstva indeksa je jedinstvenost. Ovo svojstvo indeksa uzrokuje da *MongoDB* odbije dupliranje vrednosti za indeksirano polje. *MongoDB* podržava takozvano parcijalno indeksiranje, tj. indeksiranje dokumenata iz kolekcije koji ispunjavaju unapred zadati specijalni izraz za filtriranje. Indeksiranjem podskupa dokumenata iz kolekciji utiče na redukovanje zahteva za skladištenjem i smanjivanje troškova performansi prilikom kreiranja i održavanja indeksa. Svojstvo razuđenosti indeksa obezbeđuje da indeks sadrži samo unose za dokumente koji imaju indeksirano polje. Indeks preskače dokumente koji nemaju indeksirano polje. Mogu se kombinovati opcija razuđenosti indeksa sa opcijom jedinstvenosti indeksa da bi se sprečilo umetanje dokumenata koji imaju duple vrednosti za indeksirana polja i preskočili dokumenti za indeksiranje kojima nedostaju indeksirana polja. Delimični indeksi nude super skup funkcionalnosti u odnosu na razuđene indekse i trebalo bi da imaju prednost u odnosu na njih. TTL indeksi su posebni indeksi koje *MongoDB* može koristiti za automatsko uklanjanje dokumenata iz kolekcije nakon određenog vremena. Ovo je idealno za određene tipove informacija kao što su podaci o događajima generisanim mašinama, evidencije i informacije o sesiji koje treba da postoje samo u bazi podataka ograničeno vreme. Skriveni indeksi nisu vidljivi planeru upita i ne mogu se koristiti za podršku upita. Sakrivanjem indeksa od planera, korisnici mogu da procene potencijalni uticaj uklanjanja indeksa bez stvarnog uklanjanja indeksa. Ako je uticaj negativan, korisnik može da otkrije indeks umesto da ponovo pravi obrisanog indeks. I pošto se indeksi u potpunosti održavaju dok su skriveni, indeksi su odmah dostupni za upotrebu kada budu otkriveni. Osim indeksa **\_id**, mogu se sakriti svi indekse.

Kreiranje indeksa

Kreiranje indeksa u *MongoDB*-u se može uraditi sledećom komandom:

*db.kolekcija.createIndex( <specifikacija ključa i tipa indeksa>, <opcije> )*

Ova metoda kreira indeks samo ako indeks iste specifikacije već ne postoji. Podrazumevano ime za indeks je spajanje indeksiranih ključeva i pravca svakog ključa u indeksu (tj. 1 ili -1) koristeći donje crte kao separator. Na primer, indeks kreiran za *{ predmet : 1, količina: -1 }* ima naziv *predmet\_1\_količina\_-1*.

Može se kreirati indeks sa prilagođenim imenom. Na primer, ako se uzme u obzir aplikaciju koja često ispituje kolekciju proizvoda da bi popunila podatke o postojećem inventaru. Metoda *createIndex*() koja bi kreirala indeks za predmet i količinu pod nazivom „inventar“ bi izgledala:

*db.products.createIndex(*

*{ predmet: 1, količina: -1 } ,{ name: "inventar" }*

*)*

Tipovi indeksa

*MongoDB* obezbeđuje niz različitih tipova indeksa za podršku specifičnim tipovima podataka i upita.

Indeksi jednog polja

*MongoDB* pruža potpunu podršku za indekse na bilo kom polju u kolekciji dokumenata. Podrazumevano, sve kolekcije imaju indeks u polju \_id, a aplikacije i korisnici mogu da dodaju dodatne indekse da podrže važne upite i operacije.

Ako se razmatra kolekcija pod nazivom „zapisi“, koja sadrži dokumente koji liče na sledeći primer dokumenta:

*{*

*"\_id": ObjectId("570c04a4ad233577f97dc459"),*

*"skor": 1034,*

*"lokacija": { država: "Srbija", grad: "Niš" }*

*}*

Sledeća komanda kreira rastući indeks u polju rezultata kolekcije zapisa.

*db.zapisi.createIndex( { skor: 1 } )*

Vrednost polja u specifikaciji indeksa opisuje vrstu indeksa za to polje. Na primer, vrednost 1 navodi indeks koji uređuje stavke u rastućem redosledu. Vrednost -1 specificira indeks koji uređuje stavke u opadajućem redosledu. Kreirani indeks će podržavati upite koji biraju na osnovu polja skor, kao što su sledeće:

*db.zapisi.find( { skor: 2 } )*

*db.zapisi.find( { skor: { $gt: 10 } } )*

Sledeća operacija kreira indeks na polju lokacija.država:

*db.zapisi.createIndex( { "lokacija.država": 1 })*

Kreirani indeks će podržavati upite nad poljem lokacija.država, kao što su sledeće:

*db.zapisi.find( { "lokacija.država": "Srbija" } )*

*db.zapisi.find(*

*{*

*"lokacija.grad": "Antananarivo",*

*"lokacija.država": "Madagaskar"*

*} )*

Takođe je moguće kreiranje indeksa nad ugrađenom dokumentu kao celini. Polje lokacije je ugrađeni dokument koji sadrži ugrađena polja grad i država. Sledeća komanda kreira indeks nad poljem lokacije u celini:

*db.zapisi.createIndex( { lokacija: 1 } )*

Sledeći upit može da koristi indeks u polju lokacije:

*db.zapisi.find( { lokacija: { grad: "Njujork", država: "SAD" } } )*

Složeni indeksi

*MongoDB* podržava složene indekse, gde jedna struktura indeksa sadrži reference na više polja unutar dokumenata kolekcije. *MongoDB* nameće ograničenje od 32 polja za bilo koji složeni indeks

Ako se razmatra kolekcija koja sadrži dokumente koji liče na sledeći primer dokumenta:

*{*

*"\_id": ObjectId(...),*

*"proizvod": "Banana",*

*"kategorija": ["hrana", "proizvod", "namirnice"],*

*"lokacija": "Prodavnica u 4. ulici",*

*"zaliha": 4,*

*}*

Sledeća operacija kreira rastući indeks na poljima *proizvod* i *zaliha*:

*db.proizvodi.createIndex( { "proizvod ": 1, "zaliha": 1 })*

Redosled polja navedenih u složenom indeksu je važan. Indeks će sadržati reference na dokumente sortirane prvo prema vrednostima polja *proizvod*, unutar svake vrednosti polja stavke, sortirane prema vrednostima polja *zaliha*. Pored toga što podržavaju upite koji se podudaraju na svim poljima indeksa, složeni indeksi mogu podržati upite koji se podudaraju na prefiksu indeksnih polja.

*db.proizvodi.find( { proizvod: "Banana" })*

*db.proizvodi.find( { proizvod: "Banana", zaliha: { $gt: 5 } })*

Redosled sortiranja

Indeksi čuvaju reference na polja u rastućem (1) ili opadajućem (-1) redosledu sortiranja. Za indekse sa jednim poljem, redosled sortiranja ključeva nije bitan jer MongoDB može da prelazi indeks u bilo kom smeru. Međutim, za složene indekse, redosled sortiranja može biti važan u određivanju da li indeks može da podrži operaciju sortiranja. Ako se razmatra kolekcija *događaji*, koja sadrži dokumente sa poljima *korisničko\_ime* i *datum*. Aplikacije mogu da izdaju upite koji vraćaju rezultate sortirane prvo rastućim vrednostima polja *korisničko\_ime*, a zatim opadajućim (tj. od novijeg do poslednjeg) vrednosti datuma, kao što su:

*db.događaji.find().sort( { korisničko\_ime: 1, datum: -1 } )*

ili upite koji vraćaju rezultate sortirane prvo po opadajućim vrednostima korisničkog imena, a zatim po rastućim vrednostima datuma, kao što su:

*db.događaji.find().sort( { korisničko\_ime: -1, datum: 1 } )*

Sledeći indeks može da podrži obe ove operacije sortiranja:

*db.događaji.createIndex( { "korisničko\_ime" : 1, "datum" : -1 })*

Međutim, gornji indeks ne može da podrži sortiranje rastućim vrednostima korisničkog imena, a zatim rastućim vrednostima datuma, kao što su sledeće:

*db.događaji.find().sort( { korisničko\_ime: 1, datum: 1 } )*

Indeksi sa više ključeva

Da bi se indeksiralo polje koje sadrži vrednost niza, *MongoDB* kreira indeksni ključ za svaki element u nizu. Ovi indeksi sa više ključeva podržavaju efikasne upite prema poljima. Indeksi sa više ključeva mogu se konstruisati preko nizova koji sadrže i skalarne vrednosti (npr. stringove, brojeve) i ugnežđene dokumente. Skalarna vrednost se odnosi na vrednost koja nije ni ugrađeni dokument ni niz.

Da bi se kreirao indeks sa više ključeva, koristi se sledeća metoda:

*db.kolekcija.createIndex( { <polje>: < 1 ili -1 > })*

*MongoDB* automatski kreira indeks sa više ključeva ako je bilo koje indeksirano polje niz; ne mora se eksplicitno navesti tip višestrukog ključa. Ako je indeks sa više ključeva, onda izračunavanje granica indeksa sledi posebna pravila. Za jedinstvene indekse, jedinstveno ograničenje se primenjuje na različite dokumente u kolekciji, a ne na jedan dokument. Pošto se jedinstveno ograničenje primenjuje na zasebne dokumente, za jedinstveni indeks sa više ključeva, dokument može imati elemente niza koji rezultiraju ponavljanjem vrednosti ključa indeksa sve dok vrednosti ključa indeksa za taj dokument ne dupliraju one iz drugog dokumenta. Za složeni indeks sa više ključeva, svaki indeksirani dokument može imati najviše jedno indeksirano polje čija je vrednost niz. Ograničenja kod ove vrste indeksa su sledeća:

* Ne može se kreirati složeni indeks sa više ključeva ako je više od jednog polja dokumenta za indeksiranje niz. Na primer, ako se razmatra kolekcija koja sadrži sledeći dokument:

*{ \_id: 1, a: [ 1, 2 ], b: [ 1, 2 ], kategorija: "AB - oba niza" }*

Ne može se kreirati složeni indeks sa više ključeva { a: 1, b: 1 } na kolekciji pošto su i a i b polja nizovi.

* Ako složeni indeks sa više ključeva već postoji, ne možete umetnuti dokument koji bi prekršio ovo ograničenje.

*{ \_id: 1, a: [1, 2], b: 1, kategorija: "A niz" }*

*{ \_id: 2, a: 1, b: [1, 2], kategorija: "B niz" }*

Složeni višeključni indeks { a: 1, b: 1 } je dozvoljen jer je za svaki dokument samo jedno polje indeksirano složenim višeključnim indeksom niz; tj. nijedan dokument ne sadrži vrednosti niza za a i b polja. Međutim, nakon kreiranja složenog indeksa sa više ključeva, ako se pokuša sa da umetanjem dokumenta gde su polja a i b nizovi, *MongoDB* to neće uspeti da realizuje.

* Ako je polje niz dokumenata, može se indeksirati ugrađena polja da bi se kreirao složen indeks.

*{ \_id: 1, a: [ { k: 5, z: [ 1, 2 ] }, { z: [ 1, 2 ] } ] }*

*{ \_id: 2, a: [ { k: 5 }, { z: 4 } ] }*

Može se kreirati složeni indeks na { "a.k": 1, "a.z": 1 }. Primenjuje se i ograničenje gde najviše jedno indeksirano polje može biti niz.

* Heširani indeksi ne mogu da budu sa više ključeva.
* Indeksi sa više ključeva ne mogu da pokriju upite preko polja(a) niza.

Geoprostorni indeks

Korišćenje *2d* indeksa je prisutno kod podataka uskladištenih kao tačke na dvodimenzionalnoj ravni.

Dvodimenzionalni indeks se koristi:

* Ako baza podataka sadrži zastarele parove koordinata iz *MongoDB* 2.2 ili ranije
* Ako ne postoji namera da skladištite podaci o lokaciji kao *GeoJSON* objekte.

Indeks *2dsphere* podržava upite koji izračunavaju geometrije na sferi nalik zemlji. 2dsphere indeks podržava sve *MongoDB* geoprostorne upite. Indeks *2dsphere* podržava podatke koji se čuvaju kao *GeoJSON* objekti i zastareli parovi koordinata. Za zastarele parove koordinata, indeks konvertuje podatke u *GeoJSON* tačku. *GeoHaistack* indeks je poseban indeks koji je optimizovan da daje rezultate na malim površinama. *GeoHaistack* indeksi poboljšavaju performanse na upitima koji koriste ravnu geometriju. Za upite koji koriste sferičnu geometriju, 2dsphere indeks je bolja opcija od *GeoHaistack* indeksa. 2dsphere indeksi omogućavaju preuređivanje polja. *GeoHaistack* indeksi zahtevaju da prvo polje bude polje lokacije. Takođe, *GeoHaistack* indeksi su upotrebljivi samo preko komandi i tako uvek vraćaju sve rezultate odjednom.

Tekstualni indeksi

Da bi se pokrenuli stari upiti za pretragu teksta, moraju postojati tekstualni indeksi u kolekcijama. *MongoDB* obezbeđuje tekstualne indekse koji podržavaju upite za pretragu teksta. Tekstualni indeksi mogu uključiti bilo koje polje čija je vrednost string ili niz stringova. Kolekcija može imati samo jedan indeks za pretragu teksta, ali taj indeks može pokriti više polja. Da bi se indeksiralo polje koje sadrži string ili niz elemenata tipa string, potrebno je u konfiguraciji kao parametar uz naziv polja staviti literal „text“, kao u primeru ispod.

*db.pregledi.createIndex( { komentari: "text" })*

Mogu se indeksirati više polja sa tekstualnim indeksom. Sledeći primer kreira tekstualni indeks u poljima „tema“ i „komentari“:

*db.pregledi.createIndex(*

*{*

*predmet: "text",*

*komentari: "text"*

*}*

*)*

Složeni indeks može uključivati ključeve tekstualnog indeksa u kombinaciji sa rastućim/opadajućim indeksnim ključevima. Za tekstualni indeks, težina indeksiranog polja označava značaj polja u odnosu na druga indeksirana polja u smislu rezultata pretraživanja teksta. Za svako indeksirano polje u dokumentu, *MongoDB* množi broj podudaranja sa težinom i zbraja rezultate. Koristeći ovu sumu, *MongoDB* zatim izračunava rezultat za dokument. Podrazumevana težina je 1 za indeksirana polja. Za podešavanje težine za indeksirana polja, treba uključiti opciju težine u metodu db.kolekcija.createIndex().

Kada se kreira tekstualni indeks na više polja, može se koristiti i džoker specifikacija ($\*\*). Uz pomoćni tekstualni indeks, *MongoDB* indeksira svako polje koje sadrži string podatke za svaki dokument u kolekciji. Sledeći primer kreira tekstualni indeks koristeći džoker specifikaciju:

*db.kolekcija.createIndex( { "$\*\*": "text" })*

Ovaj indeks omogućava pretragu teksta u svim poljima sa sadržajem tipa string. Takav indeks može biti koristan sa veoma nestrukturiranim podacima ako nije jasno koja polja treba uključiti u tekstualni indeks ili za ad-hoc upite.

Indeksi teksta zasnovani na džokerima su tekstualni indeksi na više polja. Kao takvima, mogu se dodeliti težine određenim poljima tokom kreiranja indeksa da bi se kontrolisalo rangiranje rezultata.

Tekstualni indeksi džoker znakova, kao i svi tekstualni indeksi, mogu biti deo složenih indeksa. Na primer, sledeće stvara složeni indeks u polju a, kao i specifikaciju džoker znakova:

*db.kolekcija.createIndex( { a: 1, "$\*\*": "text" })*

Kao i kod svih složenih tekstualnih indeksa, pošto „a“ prethodi ključu tekstualnog indeksa, da bi se izvršila pretraga $text sa ovim indeksom, predikat upita mora da sadrži uslove podudaranja jednakosti „a“. Tekstualni indeksi imaju sledeće zahteve za skladištenje i troškove performansi:

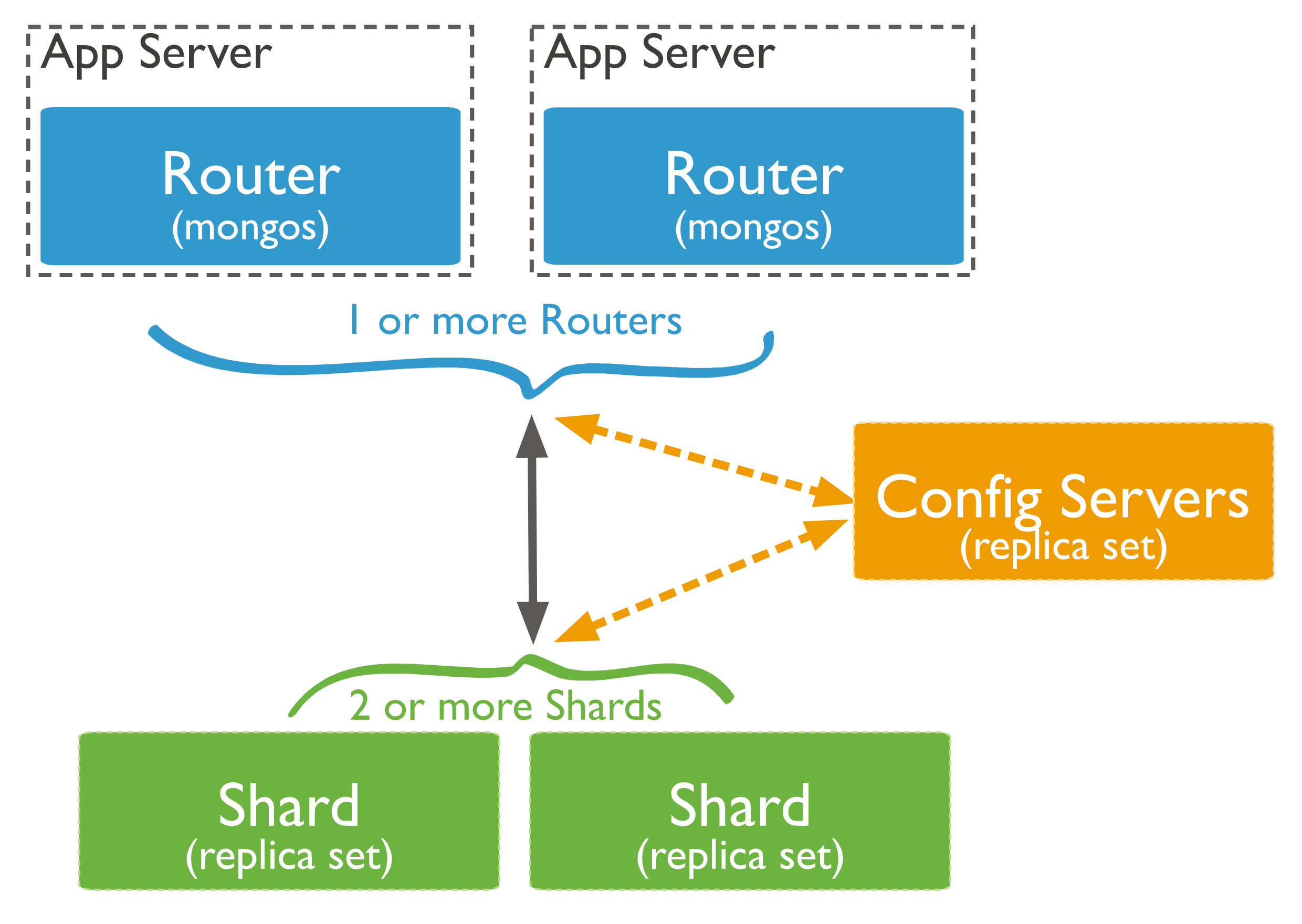
* Tekstualni indeksi mogu biti veliki. Oni sadrže po jedan indeksni unos za svaku jedinstvenu reč koja se nalazi u svakom indeksiranom polju za svaki umetnuti dokument.
* Izgradnja tekstualnog indeksa je veoma slična izgradnji velikog indeksa sa više ključeva i trajaće duže od pravljenja jednostavnog uređenog (skalarnog) indeksa na istim podacima.
* Tekstualni indeksi će uticati na propusnost umetanja jer *MongoDB* mora da doda indeksni unos za svaku jedinstvenu reč sa osnovom u svakom indeksiranom polju svakog novog izvornog dokumenta.
* Pored toga, tekstualni indeksi ne čuvaju fraze ili informacije o blizini reči u dokumentima. Kao rezultat toga, upiti fraze će raditi mnogo efikasnije kada cela kolekcija stane u RAM.

Shard indeksi

*Sharding* je metoda za distribuciju podataka na više mašina. *MongoDB* koristi *sharding* za rad sa veoma velikim skupovima podataka i operacijama velike propusnosti. Sistemi baza podataka sa velikim skupovima podataka ili aplikacijama velike propusnosti mogu da iscrpe kapacitete jednog servera. Na primer, visoke stope upita mogu iscrpiti CPU kapacitet servera. Radne grupe koje su veće od sistemske RAM memorije opterećuju I/O kapacitet disk jedinica. Postoje dve metode za rešavanje rasta sistema: vertikalno i horizontalno skaliranje. Vertikalno skaliranje podrazumeva povećanje kapaciteta jednog servera, kao što je korišćenje moćnijeg CPU-a, dodavanje više RAM-a ili povećanje količine prostora za skladištenje. Ograničenja u dostupnoj tehnologiji mogu ograničiti da jedna mašina bude dovoljno moćna za dato radno opterećenje. Pored toga, provajderi zasnovani na oblaku imaju čvrste plafone na osnovu dostupnih hardverskih konfiguracija. Kao rezultat, postoji praktičan maksimum za vertikalno skaliranje. Horizontalno skaliranje uključuje podelu sistemskog skupa podataka i opterećenja na više servera, dodavanjem dodatnih servera radi povećanja kapaciteta po potrebi. Iako ukupna brzina ili kapacitet jedne mašine možda nisu visoki, svaka mašina se nosi sa podskupom ukupnog radnog opterećenja, potencijalno pružajući bolju efikasnost od jednog servera velike brzine i velikog kapaciteta. Proširenje kapaciteta implementacije zahteva samo dodavanje dodatnih servera po potrebi, što može biti niži ukupni trošak od vrhunskog hardvera za jednu mašinu. Kompromis je povećana složenost infrastrukture i održavanja za primenu. *MongoDB* podržava horizontalno skaliranje putem *shard*-ova.

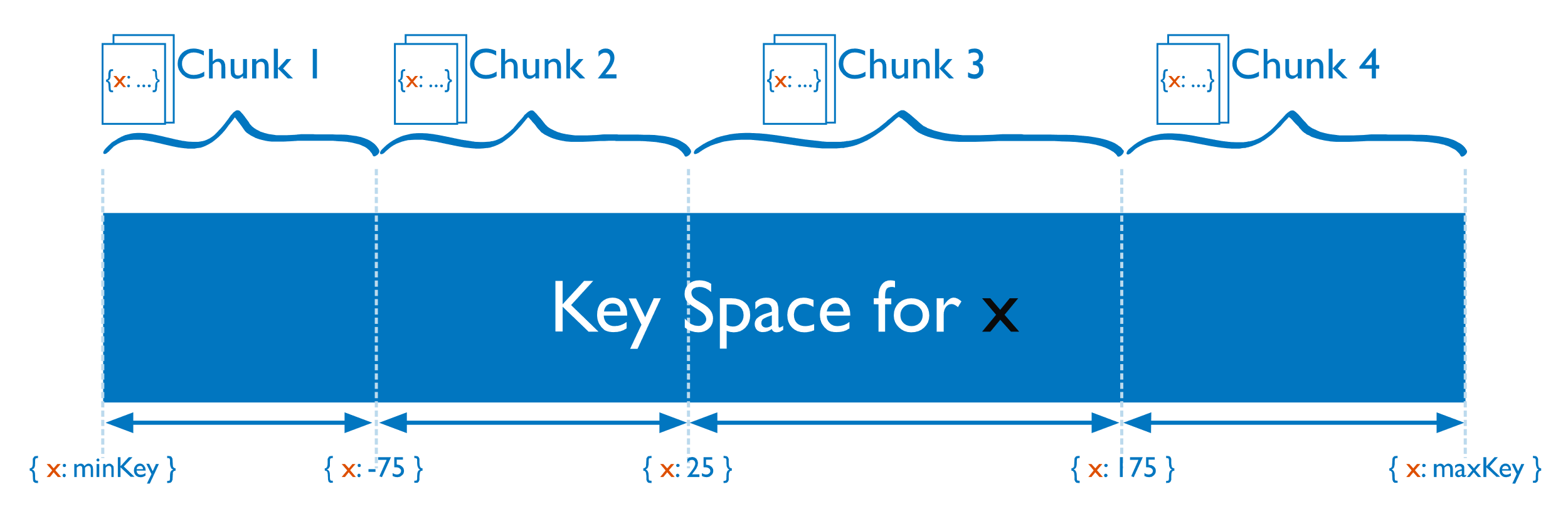
Kod *MongoDB-*a*,* *shard* klaster se sastoji od sledećih komponenti:

* *Shard* – Svaki deo sadrži podskup podeljenih podataka. Svaki deo se može primeniti kao skup replika.
* *mongos*: Mongos deluje kao ruter upita, obezbeđujući interfejs između klijentskih aplikacija i *shard* klastera. Počevši od *MongoDB*-a 4.4, *mongos* može da podrži zaštićeno čitanje kako bi se minimizirala kašnjenja.
* konfiguracioni serveri: Konfiguracioni serveri čuvaju meta podatke i podešavanja konfiguracije za klaster.

Grafikon sa slike 9 opisuje interakciju komponenti u sklopu *shard* klastera:

*Slika 9 – Interakcija komponenti u sklopu shard klastera*

*MongoDB* vrši *sharding* podataka na nivou kolekcije, distribuirajući podatke kolekcija po *shard*-ovima u klasteru. *MongoDB* koristi ključ *shard*-a za distribuciju dokumenata kolekcije. *Shard* ključ se sastoji od polja ili više polja u dokumentima. *MongoDB* deli raspon vrednosti ključeva *shard*-a (ili heširanih vrednosti ključeva *shard*-a) na opsege vrednosti ključeva *shard*-a koji se ne preklapaju (ili vrednosti heširanih vrednosti ključeva *shard*-a). Svaki opseg je povezan sa *chunk*-om, a *MongoDB* pokušava da ravnomerno rasporedi *chunk*-ove između *shard*-ova u klasteru, kao na slici 10.



*Slika 10 –Raspoređivanje opsega ključeva po chunk-ovima*

Sve *shard* kolekcije moraju imati indeks koji podržava ključ *shard*-a. Indeks može biti indeks na ključu *shard*-a ili složeni indeks gde je ključ *shard*-a prefiks indeksa. Ako je kolekcija prazna, *sh.shardCollection()* kreira indeks na ključu *shard*-a, ako takav indeks već ne postoji.

Ako kolekcija nije prazna, mora se prvo kreirati indeks pre upotrebe *sh.shardCollection().*

Strategije indeksiranja

Prilikom izbora najboljih indeksa za aplikaciju moraju se uzeti u obzir brojni faktori, uključujući vrste upita koji se koriste, odnos čitanja i upisa i količinu slobodne memorije na postojećem sistemu.

Kada se razvija strategija indeksiranja, treba imati duboko razumevanje upita aplikacije. Pre nego što se kreiraju indeksi, odrede se tipovi upita koji će se koriste tako da se na osnovu njih mogu napraviti indeksi koji upućuju na ta polja. Indeksi dolaze sa troškovima performansi, ali su više nego vredni troškova za česte upite u velikim skupovima podataka. Treba imati u vidu i relativnu učestalost svakog upita u aplikaciji i da li upit opravdava indeks.

Najbolja opšta strategija za projektovanje indeksa je profilisanje različitih konfiguracija indeksa sa skupovima podataka sličnim onima koji će se koristiti u aplikaciji, da bi se odredilo koje konfiguracije imaju najbolje rezultate. Potrebno je pregledati trenutne indekse kreirane za kolekcije da bi se utvrdilo da podržavaju trenutne i planirane upite. Ako se indeks više ne koristi, treba ga ukloniti.

Generalno, *MongoDB* koristi samo jedan indeks da ispuni većinu upita. Međutim, svaka klauzula upita *$or* može da koristi drugačiji indeks, a pored toga, *MongoDB* može da koristi presek više indeksa.

Lista komandi za indeksiranje

*MongoDB* *Shell*, ili skraćeno *mongosh*, je potpuno funkcionalno *JavaScript* i *Node*.*js* okruženje za interakciju sa *MongoDB-*jem. Može se koristiti i za testiranje upita i operacija direktno sa bazom podataka. Metode indeksiranja u *mongosh*-u su prikazane u tabeli 1.

Tabela 1:

|  |  |
| --- | --- |
| **Ime** | **Opis** |
| *db.collection.createIndex()* | Kreira indeks nad kolekcijom |
| *db.collection.dropIndex()* | Uklanja navedeni indeks iz kolekcije |
| *db.collection.dropIndexes()* | Uklanja sve indekse iz kolekcije |
| *db.collection.getIndexes()* | Vraća niz dokumenata koji opisuju postojeće indekse u kolekciji |
| *db.collection.reIndex()* | Ponovo kreira se postojeće indekse nad zadatom kolekcijom |
| *db.collection.totalIndexSize()* | Izveštaji o ukupnoj veličini koju koriste indeksi u kolekciji |
| *cursor.explain()* | Izveštaji o planu izvršenja upita za kursor |
| *cursor.hint()* | Prisiljava *MongoDB* da koristi određeni indeks za upit |
| *cursor.max()* | Određuje ekskluzivni gornji indeks koji je ograničen za kursor |
| *cursor.min()* | Određuje inkluzivni donji indeks za kursor |

Kursor je *MongoDB* kolekcija dokumenta koja se vraća nakon izvršenja metode *find*. Podrazumevano, automatski se izvršava kao petlja. Međutim, možemo eksplicitno dobiti određeni indeksni dokument iz vraćenog kursora. To je isto kao pokazivač koji pokazuje na određenu vrednost indeksa.

Komande za indeksiranje baze podataka date su u tabeli 2:

Tabela 2:

|  |  |
| --- | --- |
| **Ime** | **Opis** |
| *createIndexes* | Kreira jedan ili više indeksa za kolekciju. |
| *dropIndexes* | Uklanja indekse iz kolekcije |
| *compact* | Defragmentira kolekciju i ponovo gradi indekse |
| *reIndex* | Ponovo gradi sve indekse na kolekciji |
| *validate* | Interna komanda koja skenira podatke kolekcije i indeksira tačnost |
| *checkShardingIndex* | Interna komanda koja proverava indeks na ključu šarda |
| *setIndexCommitQuorum* | Menja minimalni broj članova koji nose podatke (tj. kvorum potvrde), uključujući primarne, koji moraju glasati da bi se potvrdila izgradnja indeksa u toku rada |

Primer funkcionisanja indeksa kod MongoDB-a

Iako je moguće uskladištiti veliki deo informacija u *MongoDB* bazi podataka, potrebna je efikasna strategija indeksiranja da bi se na brz i efikasan način dobile potrebne informacije iz nje.

Prilikom inicijalnog ubacivanja podataka u bazu, *MongoDB* kreira podrazumevani primarni ključ \_id koji je indeksiran. Međutim, baza podataka ne može da pogodi druge indekse koji su potrebni jer ne postoji način da predvidi vrstu pretraživanja, sortiranja i agregacije koje će biti primenjivani nad ovim podacima. Dakle, on samo obezbeđuje jedinstveni identifikator za svaki dokument u kolekciji, koji se zadržava u svim daljim indeksima. *MongoDB* omogućava da se kreiraju dodatni indeksi koji su slični dizajnu onih koji se nalaze u relacionim bazama podataka, a za njih je potrebna određena količina administracije. Kao i kod drugih sistema baza podataka, što je i rečeno u teorijskom delu, postoje posebni indeksi za razuđene podatke, za pretraživanje kroz tekst ili za odabir prostornih informacija. Svaki upit ili ažuriranje obično koristi samo jedan indeks ako postoji odgovarajući. Indeks obično može pomoći u performansama bilo koje operacije sa podacima, ali to nije uvek slučaj. Ukoliko se koristi takozvani *scattergun* pristup, to jest kreiranje velikog broja različitih indeksa kako bi se obezbedilo da uvek postoji jedan koji će verovatno biti prikladan, nailazi se na lošu stranu indeksa koja se ogleda u tome da svaki indeks koristi resurse i sistem mora da ga održava kad god se podaci menjaju. Ako se preteruje sa indeksima, oni će dominirati memorijskim stranicama i dovesti do okupiranja diska *I/O* operacijama. Mali broj visoko efikasnih indeksa je najbolji.

Za ilustraciju nekih od osnovnih indeksa, biće učitano 70.000 kupaca u *MongoDB* iz JSON datoteke preuzete sa sajta <https://studio3t.com/>. Ovo učitavanje, kao i sve ostale operacije biće izvršene pomoću *MongoDB* *GUI* alata *Studio 3T*. Nakon učitavanja podataka preko korisničkog interfejsa, može se pogledati šema jednostavnim ispitivanjem prvog dokumenta preko komande:

*db.Customers.find({}).limit(1);*

kao što je i prikazano na slici 11. Naredni jednostavan upit prema novokreiranoj pronalazi sve kupce čije je prezime „Johnston“, prikazujući kolone „First Name“ i „Last Name“, koje su pritom sortirane po koloni „Last Name.

*use customers;*

*db.Customers.find({*

*"Name.Last Name" : "Johnston"*

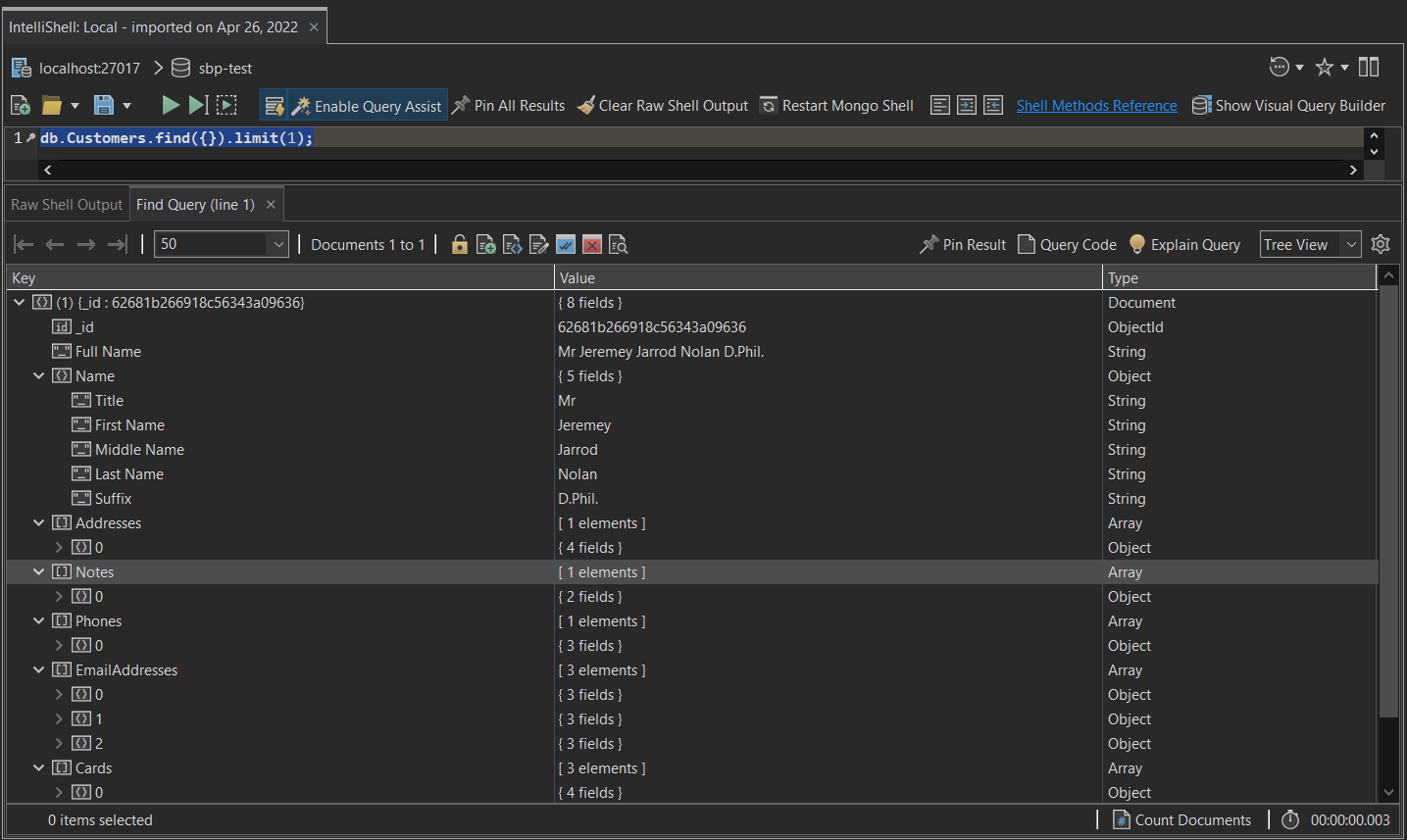
*}, {"\_id" : NumberInt(0),*

*"Name.First Name" : NumberInt(1),*

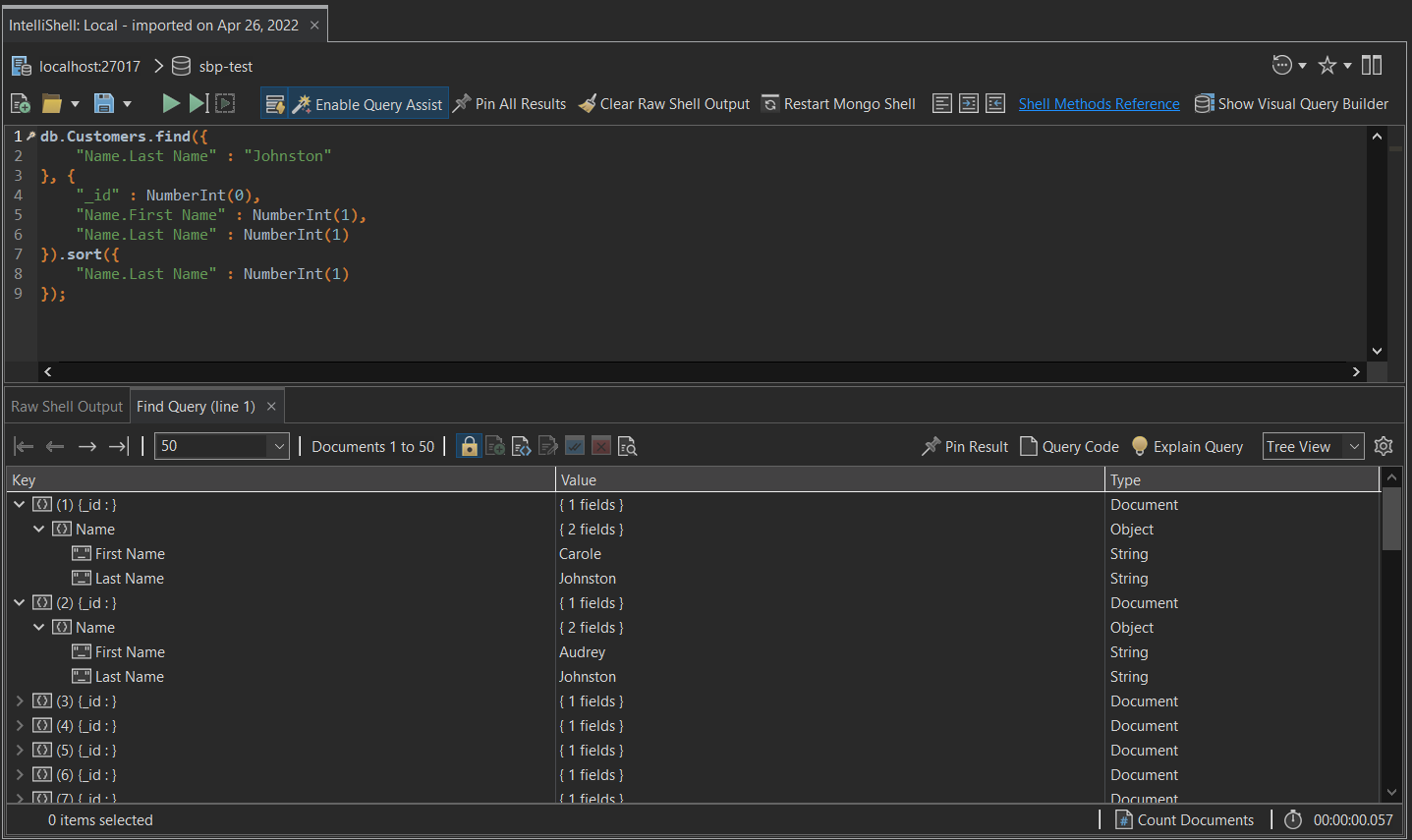
*"Name.Last Name" : NumberInt(1)*

*}).sort({ "Name.Last Name" : NumberInt(1)});*

Rezultati ovog upita prikazani su na slici 12.

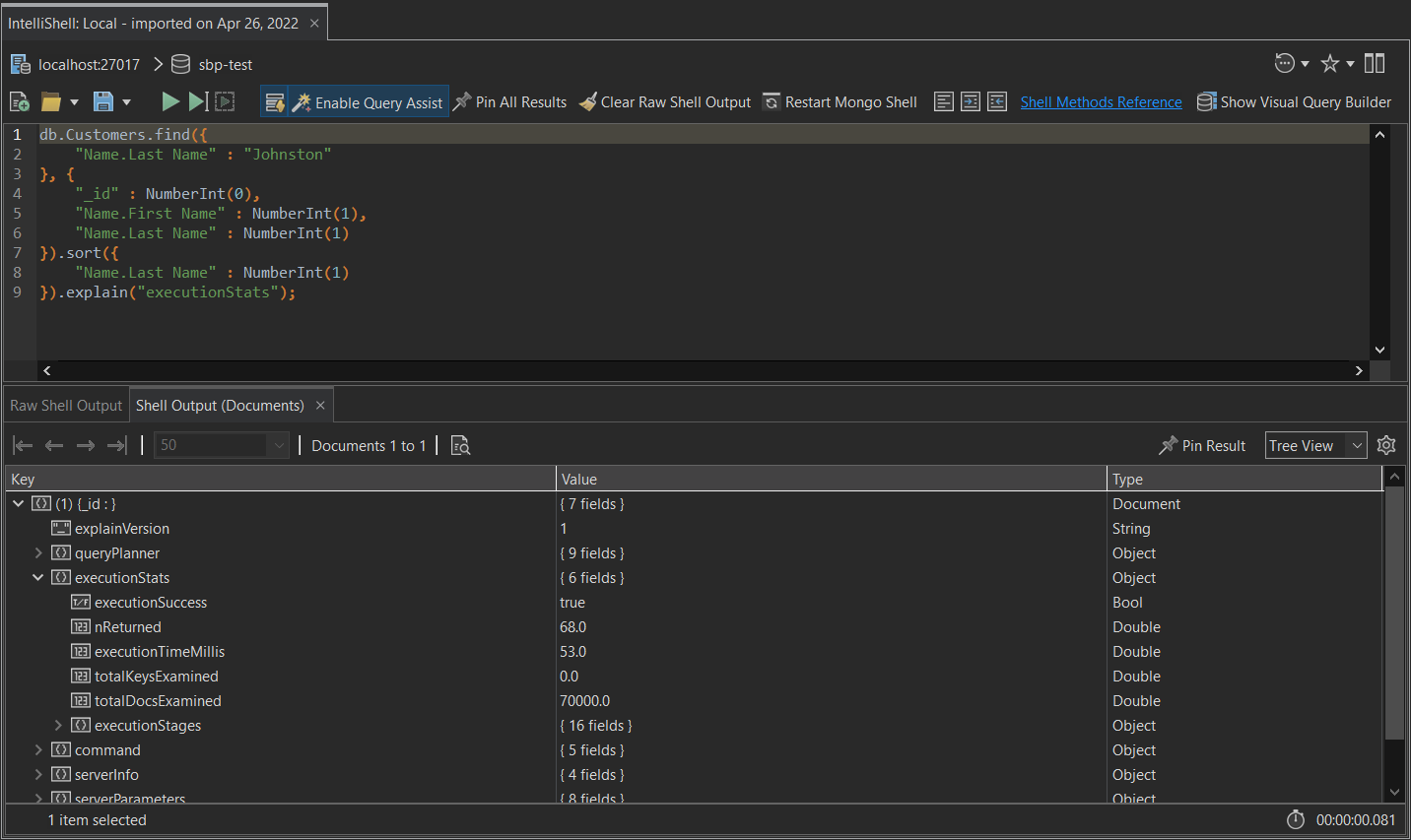


*Slika 11 – Struktura prvog dokumenta iz kolekcije*



*Slika 12 – Pribavljanje imena i prezimena*

Prema statistici izvršenja pomoću komande *explain*, kao što je i prikazano na slici 13, za prethodni upit je potrebno 53 ms (*ExecutionTimeMillis*). Ovo uključuje *collscan*, što znači da nema dostupnog indeksa, tako da *MongoDB* mora skenirati celu kolekciju. Ovo nije nužno loša stvar sa relativno malom kolekcijom, ali kako se veličina povećava i više korisnika pristupa podacima, manja je verovatnoća da će kolekcija stati u straničenu memoriju, a aktivnost diska će se povećati. Baza podataka se neće dobro skalirati ako je primorana da uradi veliki procenat *collscan*-a. Dobra je ideja minimizirati resurse koje koriste često izvršavani upiti. Pa, očigledno je da ako će indeks smanjiti potrebno vreme, verovatno će uključiti *First Name.Last Name*.



*Slika 13 – Statistika izvršenja pomoću komande explain*

Komandom *db.Customers.createIndex( {"Name.Last Name" : 1 },{ name: "LastNameIndex"} )* se postavlja rastući indeks nad kolonom *Last Name.* Ovom promenom drastično se smanjuje vreme pribavljanja podataka (3 ms) jer se vrši takozvani *ixscan*, tj. prvo skeniranje po indeksu, a potom pribavljanje samih podataka. Dodatnim ubrzavanjem dodavanjem indeksa i nad kolonom *First Name,* vreme izvršenja se svodi na 1 ms i realizovano je komandama:

*db.Customers.dropIndex("LastNameIndex")*

*db.Customers.createIndex( { "Name.Last Name" : 1,"Name.First Name" : 1 },*

*{ name: "LastNameCompoundIndex"} )*

Pošto je indeks „pokrio“ upit, *MongoDB* je uspeo da uskladi uslove upita i vrati rezultate koristeći samo ključeve indeksa, čak i bez potrebe da se pregledaju dokumenti iz zbirke da bi se vratili rezultati. Ako konačni rezultat treba da bude od naveden u opadajućem redosledu nema vidljive razlike sa ovim kratkim skupom rezultata. Ako se promeni redosled dva polja u indeksu tako da *Name.First Name* bude ispred *Name.Last Name*, vreme izvršenja dostiže 136 ms, što je ogromno povećanje. Ovo izgleda bizarno, jer je indeks zapravo usporio izvršenje tako da zauzima dvostruko više vremena koje je bilo potrebno samo sa podrazumevanim primarnim indeksom. Čini se da jednostavni upiti imaju najviše koristi od indeksa koji su uključeni u kriterijume odabira i sa istim upoređivanjem. U prethodnom primeru, ilustrovana je generalna istina o *MongoDB* indeksima: ako prvo polje indeksa nije deo kriterijuma za izbor, nije korisno izvršiti upit.

Ako postoje dva kriterijuma, od kojih jedan uključuje podudaranje stringa unutar vrednosti, kao u sledećem upitu, potrebno je 21 ms. Ako izuzmemo ime iz pretrage, vreme izvršenja se zapravo veće (48 ms). Ako se sada uvede složeni indeks koji vodi sa imenom, a zatim dodaje adresu, upit se izvrši za 1 ms. Složena pretraga treba da smanji izbor kandidata što je više moguće sa prvom stavkom na listi indeksa. Kardinalnost je termin koji se koristi za ovu vrstu selektivnosti. Polje sa niskom kardinalnošću, kao što je pol, je daleko manje selektivno od prezimena. U primeru, prezime je dovoljno selektivno da bude očigledan izbor za prvo polje koje je navedeno u indeksu, ali nije mnogo upita tako očigledno. Pretraga koju nudi prvo polje u upotrebljivom indeksu treba da bude *SARGable* (*Search ARGumentable*). U slučaju pretrage po reči „rutland“, termin za pretragu se nije direktno odnosio na ono što je bilo u indeksu i na redosled sortiranja indeksa. Bilo je moguće da se efikasno koristi samo zato što se koristio redosled indeksa da bila obezbeđena najbolja strategija za pronalaženje dvadeset verovatnih „Wigginsonovih“ u bazi podataka i zatim korišćenjem kopije pune adrese u indeksu umesto samog dokumenta. Tada je *MongoDB* mogao da pretraži tih dvadeset punih adresa veoma brzo, a da čak i ne mora da preuzme podatke iz dvadeset dokumenata. Konačno, sa primarnim ključem koji je bio u indeksu, mogao je vrlo brzo da preuzme tačan dokument iz kolekcije.

Naša kolekcija dokumenata omogućava da kupac ima jednu ili više adresa e-pošte. Oni su u ugrađenom nizu. Ako se pretražuje prezime i adresu e-pošte klijenta sa određenim prezimenom, npr. „Barker“, i određenom adresom e-pošte, npr. „Ibrahim10@Gmuul.com“ to ćemo odraditi sledećom komandom.

*db.Customers.find({*

*"Name.Last Name" : "Barker",*

*"EmailAddresses.EmailAddress" : "Ibrahim10@Gmuul.com"*

*}, {*

*"\_id" : NumberInt(0),*

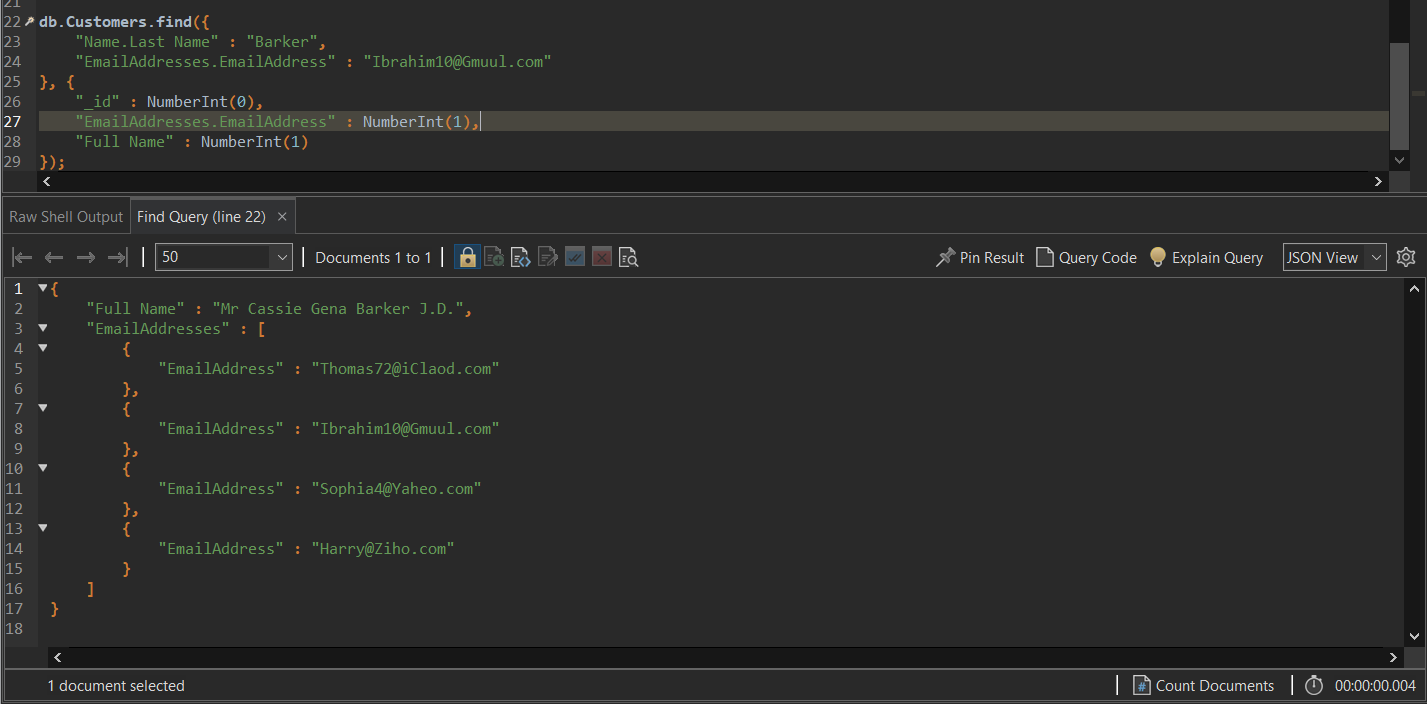
*"EmailAddresses.$.EmailAddress" : NumberInt(1),*

*"Full Name" : NumberInt(1)*

*});*

Rezultat je prikazan na slici 14. Ovo govori da je „Cassie Barker“ imala adresu e-pošte „Ibrahim10@Gmuul.com“ od 11. januara 2016. do 25. januara 2018. Kada je pokrenut upit, trebalo je 125 ms jer nije bilo korisnog indeksa. Možemo da napravimo indeks koji će pomoći u tome:

*db.Customers.createIndex( { "Name.Last Name" : 1 },{ name: "Nad"} );*



*Slika 14 – Rezultat pretrage*

Ovaj indeks je smanjio vreme izvršenja na 4 ms. Nad indeks koji je jedini bio dostupan kolekciji nalazio se samo u polju *Name.Last Name*. Za fazu unosa, korišćena je *ixscan* strategija i vrlo brzo je vratila odgovarajuća dokumenta, krećući se napred. Zatim je filtrirao te dokumente koji se podudaraju da bi dohvatio niz *EmailAddresses* za adresu koja je zatim vraćena u fazi projekcije. Jedan od razloga što pronalaženje indeksa funkcioniše tako dobro je taj što imaju mnogo bolje rezultate u kešu od skeniranja pune kolekcije. Međutim, ako cela kolekcija može da stane u keš, onda će skeniranje kolekcije raditi bliže indeksnim brzinama.

U nastavku će biti analizirano koja su najpopularnija imena na listi kupaca, koristeći agregaciju. Potreban je indeks na *Name.Last Name*. Komanda je sledeća:

*db.Customers.aggregate({$project :{"Name.Last Name": 1}},*

*{$group :{\_id: "$Name.Last Name", count : {$sum: 1}}},*

*{$sort : {count : -1}},*

*{$limit:10}*

*);*

Dakle, u deset najboljih, imamo dosta članova porodice „Snajder“:

*{ "\_id" : "Snyder", "count" : 83 }*

*{ "\_id" : "Baird", "count" : 81 }*

*{ "\_id" : "Evans", "count" : 81 }*

*{ "\_id" : "Andrade", "count" : 81 }*

*{ "\_id" : "Woods", "count" : 80 }*

*{ "\_id" : "Burton", "count" : 79 }*

*{ "\_id" : "Ellis", "count" : 77 }*

*{ "\_id" : "Lutz", "count" : 77 }*

*{ "\_id" : "Wolfe", "count" : 77 }*

*{ "\_id" : "Knox", "count" : 77 }*

Ovo je trajalo samo 8 ms uprkos tome što je urađen *collscan* jer se cela baza podataka mogla držati u keš memoriji. Ako želimo da vidimo koja imena privlače najviše beleški u svom dosijeu to možemo uraditi sledećom komandom:

*db.Customers.aggregate(*

*{$group: {\_id: "$Name.First Name", NoOfNotes: {$avg: {$size: "$Notes"}}}},*

*{$sort : {NoOfNotes : -1}},*

*{$limit:10}*

*);*

U podacima, ljudi koji se zovu „Charisse“ dobijaju najviše beleški. Ovde se vidi da je *collscan* neizbežan jer će se broj beleški menja u sistemu. Neke baze podataka dozvoljavaju indekse na izračunatim kolonama, ali to ovde ne bi pomoglo.

*{ "\_id" : "Charisse", "NoOfNotes" : 1.793103448275862 }*

*{ "\_id" : "Marian", "NoOfNotes" : 1.72 }*

*{ "\_id" : "Consuelo", "NoOfNotes" : 1.696969696969697 }*

*{ "\_id" : "Lilia", "NoOfNotes" : 1.66666666666666667 }*

*{ "\_id" : "Josephine", "NoOfNotes" : 1.65625 }*

*{ "\_id" : "Vili", "NoOfNotes" : 1.6486486486486487 }*

*{ "\_id" : "Charlton", "NoOfNotes" : 1.6458333333333333 }*

*{ "\_id" : "Kristi", "NoOfNotes" : 1.6451612903225807 }*

*{ "\_id" : "Cora", "NoOfNotes" : 1.64 }*

*{ "\_id" : "Dominic", "NoOfNotes" : 1.6363636363636365 }*

Učinak agregiranja može se poboljšati indeksom jer oni mogu pokriti agregaciju. Samo operatori cevovoda $match i $sort mogu direktno da iskoriste prednosti indeksa, i to samo ako se pojave na početku cevovoda.

Zaključak

Kada se razvija strategija indeksiranja za *MongoDB*, postoji niz faktora koje treba uzeti u obzir, kao što su struktura podataka, obrazac korišćenja i konfiguracija servera baze podataka.

*MongoDB* generalno koristi samo jedan indeks kada izvršava upit, i za pretragu i za sortiranje i ako dobije izbor strategije, uzorkovaće indekse najboljih kandidata. Većina kolekcija podataka ima neke prilično dobre kandidate za indekse, koji će verovatno jasno razlikovati dokumente u kolekciji i koji će verovatno biti popularni u obavljanju pretraga. Dobra je ideja biti štedljiv sa indeksima jer oni imaju veliku cenu u smislu resursa. Veća opasnost je da se zaboravi ono što je već tu, mada na sreću nije moguće kreirati duple indekse u *MongoDB*-u. Uvek je moguće napraviti nekoliko složenih indeksa koji su veoma bliski po svom sastavu. Ako se indeks ne koristi, najbolje je da se ukloni. Složeni indeksi su veoma dobri u podršci upitima. Oni koriste prvo polje za pretragu, a zatim koriste vrednosti u drugim poljima da vrate rezultate, umesto da moraju da uzimaju vrednosti iz dokumenata. Da bi indeksi bili efikasni za poređenja stringova, moraju da koriste isto upoređivanje. Vredi paziti na performanse upita. Često je iznenađujuće lako transformisati brzinu takvih upita pružanjem pravog indeksa.

Reference

[1] https://www.geeksforgeeks.org/indexing-in-databases-set-1/

[2] https://www.tutorialspoint.com/what-are-hashed-files-and-indexed-file-organization-dbms

[3] https://www.digitalocean.com/community/tutorials/how-to-use-indexes-in-mongodb

[4] https://www.techtarget.com/searchdatamanagement/definition/MongoDB

[5] <https://dzone.com/articles/effective-mongodb-indexing-part-1>

[6] https://studio3t.com/knowledge-base/articles/mongodb-index-strategy/#creating-a-test-database