UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET

# Interna struktura i organizacija indeksa u ArangoDB bazi podataka

Master akademske studije
Seminarski rad iz predmeta Sistemi za upravljanje bazama podataka

Mentor:
Prof. dr Aleksandar Stanimirović

Student:
Sanja Milenković 1549

Niš, april 2024. godine

# 1. Sadržaj

# 2. Uvod

**“If you dont find it in the index, look very carefully through the entire catalogue”**

Sears, Roebuck, and Co, Consumer’s Guide 1897

Koncept indeksiranja već je dugo poznat i vekovima unazad koristi se u rečnicima, raznim katalozima i enciklopedijama. Katalog autora u biblioteci ili indeks korišćenih termina koje se najčešće sreću na poslednjim stranama udžbenika, zapravo su metode koje se jako često koriste i doprinose efikasnosti u radu. Kada računar želi da pronađe podatak u bazi podataka, zapravo bi trebalo da postigne istu stvar – da pronađe odgovarajući podatak u velikom setu podataka, što je efikasnije moguće, bez potrebe za prolaskom kroz sve redove.

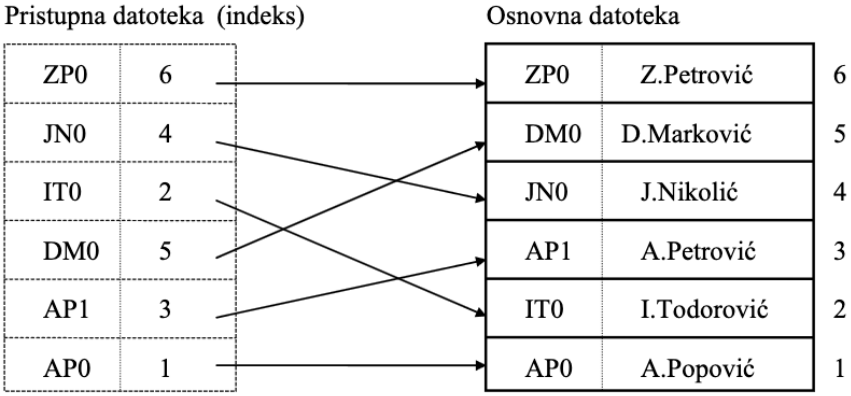
Sve veća količina podataka u bazama podataka zapravo je samo uvećala potrebu za metodama indeksiranja koje bi omogućile brz pristup određenom podskupu podataka. Često se dešava da je potrebno pročitati jako malu količinu podataka, a prolazak kroz sve redove baze podataka kako bi se pronašlo nešto čemu se može pristupiti jedinstvenom operacijom čitanja bio bi krajnje neefikasan.

Već krajem 70.-tih godina prošlog veka, indeksiranje je bilo sastavni deo relacionih baza podataka, a sam koncept implementiran je na različite načine: sekvencijalno, heširanjem, preko binarnih stabala, B-stabala, invertovanih dokumenata i drugih.

Ovaj rad ima za cilj da objasni termin indeksa uz poseban osvrt na ArangoDB bazu podataka i organizaciju indeksa u istoj.

# 3. Pojam indeksa

Najveća efikasnost pristupa dobija se uvođenjem posebnih pristupnih datoteka odnosno **indeksa**. Uz osnovnu, direktnu, datoteku koristi se i takozvana pristupna datoteka. Osnovna datoteka čuva osnovne podatke, odnosno slogove podataka od interesa za korisnike. Pristupna datoteka, kako joj i samo ime kaže, sadrži podatke koji obezbeđuju direktni i efikasan pristup osnovnim podacima. Uobičajeno je da se pristupne datoteke nazivaju **indeksnim datotekama** ili **indeksima**. Svakom slogu u osnovnoj datoteci odgovara jedan slog u indeksnoj datoteci. Ona je organizovana tako da svaki slog sadrži dva podatka: šifru osnovnog sloga i njegov redni broj u osnovnoj datoteci. Ključ optimizacije pretraživanja korišćenjem indeksa leži u ulančavanju slogova datoteka u stabla, koja su uređena po rastućoj ili opadajućoj vrednosti šifre. Na slici 1 data je ilustracija navedene organizacije, na kojoj je prikazan način zapisivanja podataka o šiframa i imenima autora naslova u biblioteci.




Slika 1. Ilustracija organizacije indeksa

Na slici je prikazan redosled slogova indeksa kakav se dobija kada se redom prolazi kroz sve slogove, a ne stvarni zapis tih slogova i njihova ulančanost u stablo. Slogovi u osnovnoj datoteci zapisani su redom, po hronologiji dodavanja. Prilikom bilo koje izmene podataka u osnovnoj datoteci (dodavanje, brisanje) uporedo se vrši i odgovarajuća izmena u indeksu.

Pojam indeksa najlakše možemo ilustrovati primerom funkcionisanja baze podataka koja ne koristi indekse. Izvršavanje najjednostavnijeg SELECT upita podrazumevao bi pretragu svih redova baze podataka, kako bi se pronašli oni koji ispunjavaju uslov zadat u WHERE delu upita. Ovaj pristup biće efikasan ukoliko baza podataka sadrži svega nekoliko redova, međutim neminovno će doći do pada performansi ukoliko govorimo o bazama podataka sa nekoliko miliona redova. Poboljšanje performansi u ovom kontekstu postiglo bi se uvođenjem indeksa, koji bi omogućili pribavljanje podataka bez pretrage svih podataka u samoj bazi podataka, jer indeksi pružaju mehanizam strukturisane pretrage umesto pretrage na nivou redova. Suštinski, indeksi predstavljaju kopiju baze podataka, gde su podaci sortirani po nekoj logici, a koja će pomoći optimizaciju upita.

Indeksi predstavljaju najbolju organizaciju direktnog pristupa, budući da nude brz direktan pristup traženim podacima i uređenost podataka po određenom kriterijumu. Indeksi uvode i neke dodatne pogodnosti. Pre svega, bitno je napomenuti da nismo ograničeni na samo jedan indeks po jednoj osnovnoj datoteci, već je moguće formirati onoliko indeksa koliko nam je kriterijuma pristupa ili uređenosti podataka potrebno. Ukoliko je nad određenom datotekom

definisano više indeksa, pre svake manipulacije bitno je naglasiti koji od indeksa je takozvani **aktivni indeks**. Takođe, prilikom kreiranja indeksa, u najvećem broju slučajeva, nismo ograničeni na proste kriterijume u smislu korišćenja jednog podatka, već je moguće kreirati složene indeksne izraze koji se sastoje od više podataka.

## 3.1 Tipovi indeksa

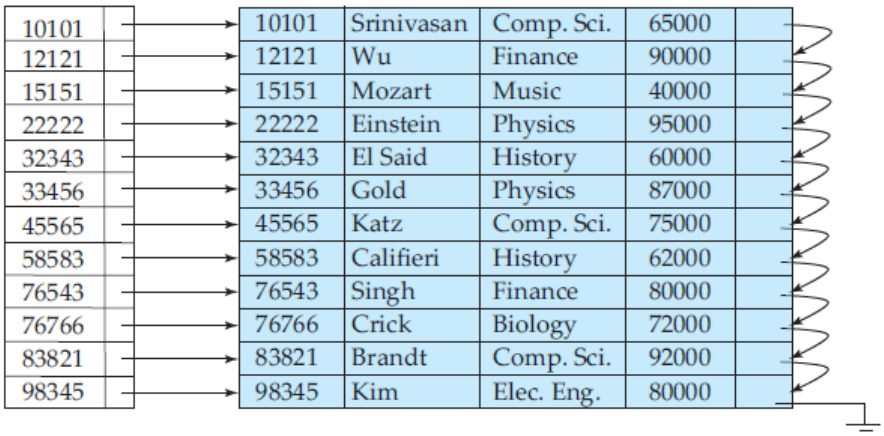
Indeks nije ništa drugo do struktura podataka koja čuva vrednosti jedne specifične kolone tabele i omogućava da se prilikom pribavljanja podataka izbegne neefikasna pretraga cele baze podataka. Dva su osnovna tipa indeksa: **uređeni indeksi**, kod kojih su ključevi pretrage sačuvani u sortiranom redosledu ili **heš indeksi**, gde su ključevi pretrage distribuirani uniformno po "baketima", na osnovu heš funkcije. Metrike koje se mogu koristiti za evaluaciju indeksa pre svega odnose se na vreme – vreme pristupa, upisivanja i brisanja ili utrošak prostora. U nastavku biće dat pregled nekih najčešće korišćenih tipova indeksa.

### 3.1.1 Primarni indeksi

Primarni indeks je uređena datoteka fiksne dužine sa dva polja. Prvo polje jedinstveno identifikuje podatak, te je nalik primarnom ključu, a drugo polje ukazuje na specifičan blok podataka.

### 3.1.2 Gusti (dense) indeksi

Gusti indeksi jesu indeksi kod kojih postoji zapis za svaku vrednost ključa pretrage u datoteci. Prednost ovih indeksa jeste u brzini pretrage koju ostvaruju, ali zahtevaju veći skladišni prostor za čuvanje svih indeksa.

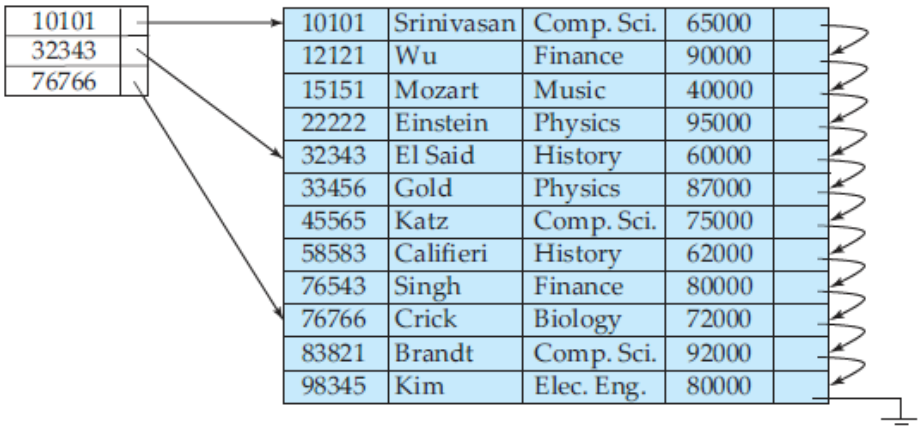



Slika 2. Primer gustog indeksa

### 3.1.3 Retki (sparse) indeksi

Retkim indeksom označavamo one indekse kod kojih zapisi postoje samo za neke vrednosti ključa pretrage u datoteci. Bitno je napomenuti da je ovakva organizacija primenljiva kada su zapisi datoteke sekvencijalno uređeni po ključu pretrage. Kako bi se locirao zapis čiji je vrednost ključa pretrage K, potrebno je najpre pronaći zapis indeksa sa najvećom vrednošću ključa pretrage koja je istovremeno manja od specificirane vrednosti K, te potom sekvencijalno pretražiti ostatak datoteke, polazeći pritom od pronađenog zapisa indeksa.

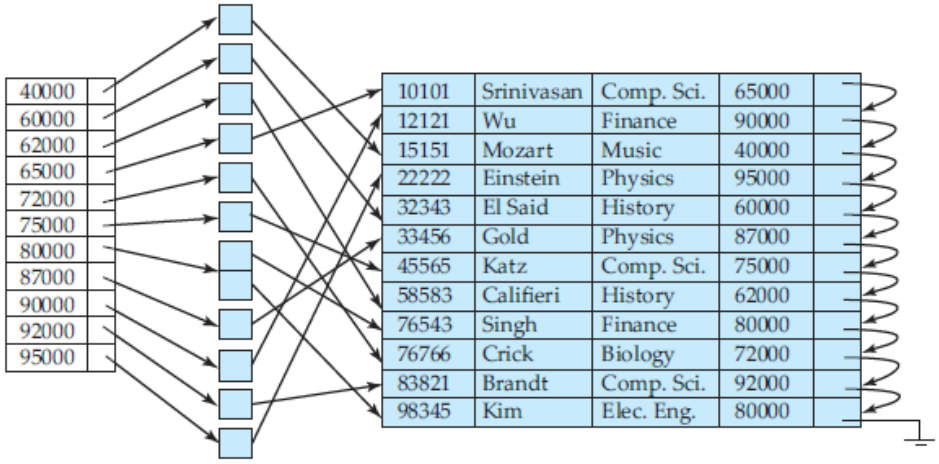
Ova vrsta indeksa, u odnosu na prethodno navedene guste indekse, zauzima manje prostora i zahteva manje održavanja pri dodavanju i brisanju zapisa, međutim postižu slabije performanse i manju brzinu prilikom pretrage zapisa.




Slika 3. Primer retkog indeksa

### 3.1.4 Sekundarni indeksi

Zapisi indeksa ukazuju na "bakete" koji sadrže pokazivače koji dalje ukazuju na sve zapise koji imaju određenu vrednost ključa pretrage. Sekundarni indeks **mora** biti gust, što znači da mora imati pripadajuću vrednost za sve vrednosti ključeva pretrage. Korišćenje sekundarnog indeksa za potrebe sekvencijalne pretrage je "skupa operacija."




Slika 4. Primer sekundarnog indeksa

## 3.2 Strukture podataka za indeksiranje

Strukture podataka koje se koriste za izgradnju indeksa oblikuju način na koji baze podataka pretražuju, ažuriraju i skladište informacije i njihov glavni cilj je da omoguće efikasno izvršenje ovih operacija. Ove strukture treba da budu dovoljno fleksibilne da podrže različite tipove upita, ali i dovoljno optimizovane da ne opterećuju previše performanse baza podataka. Među najčešće korišćenim strukturama podataka za indekse su B+ stabla i heš tabele.

### 3.2.1 $B^+$-stabla

Radi se o, po mišljenju mnogih, najvažnijem i najrasprostranjenijem tipu indeksa. Ovaj tip indeksa intenzivno se koristi, budući da uvodi značajne prednosti u odnosu na sekvencijalne

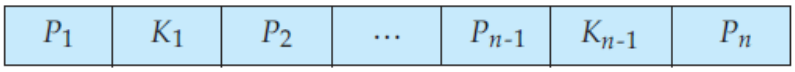
indekse. Pre svega, ovi indeksi uključuju i automatsku reorganizaciju sa malim, lokalnim promenama usled dodavanja i brisanja zapisa, međutim iako uvode i dodatan trošak vremena prilikom istih operacija, ali i trošak prostora, prednosti nadmašuju ove nedostatke. $B^+$ stabla jesu stabla koje zadovoljavaju neka specifična ograničenja:

– Sve putanje od korena do listova su iste dužine

– Svaki čvor koji nije koren ili list ima između n/2 i n dece

– Svaki list ima između (n-1)/2 i n-1 vrednosti.

Primer tipičnog čvora dat je na slici 5. Ključevi pretrage označeni su sa Ki, dok su sa Pi označeni pokazivači na decu čvorove za slučaj unutrašnjih čvorova, odnosno pokazivači na zapise ili grupe zapisa u slučaju listova. Ključevi pretrage u čvoru su uređeni, odnosno sortirani.



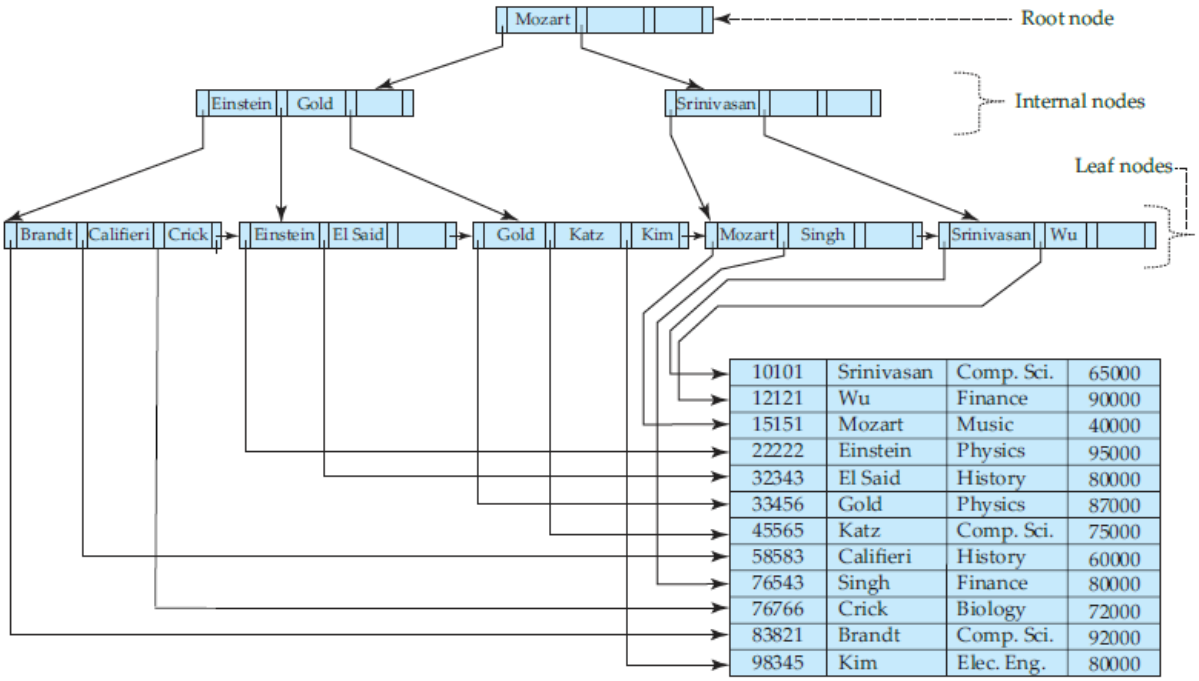

Slika 5. Primer čvora u $B^+$ stablu

Osobine listova u B+ stablu:

- Za i = 1, 2, n-1 pokazivači Pi pokazuju na zapise u datoteci sa ključem pretrage Ki

- Ako su Li, Lj listovi pri čemu i<j, onda je ključ pretrage lista Li manji ili jednak ključu pretrage lista Lj

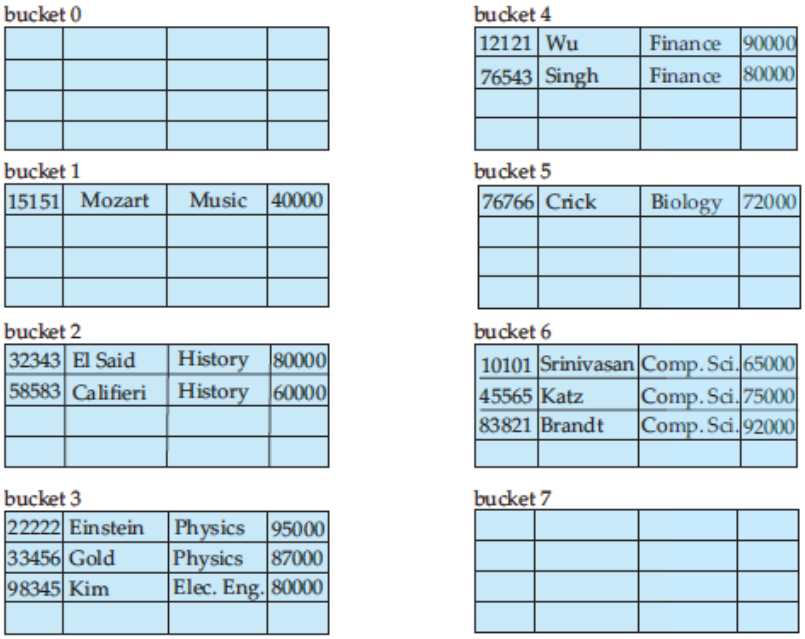
- Pn pokazuje na sledeći list u redosledu sortiranosti ključeva pretrage




Slika 6. Primer indeksa $B^+$ stabla

### 3.2.2 Heš tabele

Heš tabele jesu strukture podataka koje se koriste za čuvanje indeksa, pa se prirodno ovaj tip indeksa naziva heš indeksi. Ovde se korišćenjem hash funkcije h skup ključeva K mapira na skup svih baketa (eng. *buckets*). Baket je prostor smeštanja koji čuva jedan ili više zapisa. Za dati ulaz baket se može pronaći tako što se nad ključem pretrage za određeni ulaz primeni heš funkcija, koja se takođe koristi za lociranje zapisa prilikom pristupa, dodavanja i brisanja.




Slika 7. Primer heš tabele za čuvanje indeksa

# 4. Organizacija indeksa u ArangoDB bazi podataka

ArangoDB je **„native“, multimodel rešenje**, čija je glavna odlika da podržava rad sa različitim modelima podataka, bez potrebe da se na to obraća posebna pažnja, budući da pruža jedinstveni interfejs ka svim podacima kroz objedinjeni jezik za pisanje upita - **AQL**. Inicijalna verzija ove baze podataka objavljena je 2011. godine. Radi se o besplatnom rešenju otvorenog koda (eng. *open-source*), koje je pisano u programskim jezicima C++ i JavaScript. Aktuelna verzija je 3.12.

Time što je, po definiciji, naglašeno da se radi o native multimodel rešenju, jasno su iz ovog sistema isključeni sistemi baza podataka koji jednovremeno podržavaju samo jedan model podataka, te zahtevaju povremenu zamenu modela. U ArangoDB-ju različiti modeli podataka koegzistiraju te ih je moguće kombinovati u jedinstvenom upitu. [25] Ovo multimodel rešenje podržava tri modela podataka, te predstavlja svojevrsnu kombinaciju ključ-vrednost, dokument i graf baze podataka.

## 4.1 Interna struktura baze podataka

Osnovni model podataka u ArangoDB bazi podataka jesu **dokumenti**, koji se interno čuvaju u binarnom obliku JSON formata pod nazivom „VelocityPack.“ Dokumenti se dalje mogu organizovati u **kolekcije dokumenata**, koje okupljaju grupe dokumenata slične ili iste strukture, budući da ovde ne postoji jasno definisana šema. Ovakva organizacija značajno može poboljšati performanse upita i olakšati indeksiranje. Broj atributa u okviru dokumenata je proizvoljan, a vrednosti njima pridružene mogu biti atomične ili složene.

## 4.2 Indeksi u ArangoDB bazi podataka

Kao što je ranije navedeno, indeksi jesu mehanizam koji omogućava brz pristup dokumentima održavanjem posebnih struktura podataka i taj se kontekst ne menja ni u slučaju ArangoDB-ja. Indeksi omogućavaju brz pristup dokumentima onda kada se indeksirani atributi koriste u upitu.

Ova baza podataka, osim što nudi mogućnost kreiranja dodatnih, **nesistemskih**, indeksa nad atributima i dokumentima, automatski indeksira neke **sistemske atribute**. Ono o čemu se mora voditi računa jeste uspostavljanje pravog balansa između kreiranja indeksa za atribute koji se često koriste, radi poboljšanja performansi upita za čitanje tih podataka i troškova koje indeksi stvaraju prilikom upisa, radi njihovog održavanja.

Korisnički definisani indeksi kreiraju se na nivou **kolekcija** i najveći broj njih može se kreirati samo navođenjem indeksiranog atributa. Neki od indeksa dozvoljavaju indeksiranje samo jednog atributa, dok ima i tipova koji nude indeksiranje više atributa istovremeno.

Sistemski atributi **_id**, **_key**, **_from** i **_to** jesu automatski indeksirani od same baze podataka, bez potrebe za bilo kakvom dodatnom aktivnošću sa strane korisnika. Prva dva indeksa, _id i _key su pokriveni primarnim ključem same kolekcije, dok su indeksi _from i _to automatski pokriveni indeksom potega. Prilikom kreiranja korisničkih indeksa, postoje i određena ograničenja. Primera radi, ne mogu se indeksirati polja koja sadrže **.** u svojim nazivima, jer se one interpretiraju kao putanje do ugnježdenih atributa.

## 4.3 Tipovi indeksa

ArangoDB podržava nekoliko različitih tipova indeksa, čiji će pregled biti dat u nastavku.

### 4.3.1 Primarni indeks

Svaka novokreirana kolekcija imaće najmanje jedan indeks tzv. **primarni indeks**. On se kreira automatski kada se kreira i kolekcija i ne može se obrisati niti promeniti, niti se njegova vrednost može korisnički definisati. Njegova glavna odgovornost je da indeksira vrednost atributa _key u kolekciji, pri čemu mora da vodi računa da ključevi svih dokumenata u kolekciji zapravo budu jedinstveni. Primarni indeks će biti korišćen primarno za **upite traženja**, pomoću atributa kolekcije _key ili _id. Pored toga, primenu nalazi i u operacijama ažuriranja, zamene i brisanja dokumenata po primarnom ključu.

```
db.collection.document("<document-key>");
db._document("<document-id>");
```

Slika 8. Primarni indeks

### 4.3.2 Indeks potega

Svaka kolekcija potega će imati i **indeks potega** (eng. e*dge index*), koji se automatski kreira. On je odgovoran za indeksiranje vrednosti _from i _to, u cilju **brzog pretraživanja odlaznih ili dolaznih potega** prilikom obilaska grafa. Ovaj indeks se, takođe, ne može ukloniti niti promeniti, niti se može eksplicitno kreirati, ali se može neometano koristiti u nekim drugim korisnički definisanim indeksima. Indeksi potega nisu jedinstveni, što znači da više ivica može biti povezano sa istim dokumentom. Ovaj indeks se koristi u upitima koji pretražuju dokumente po vrednostima _from ili _to.

```
db.collection.edges("<from-value>");
db.collection.edges("<to-value>");
db.collection.outEdges("<from-value>");
db.collection.outEdges("<to-value>");
db.collection.inEdges("<from-value>");
db.collection.inEdges("<to-value>");
```

Slika 9. Indeksi potega

### 4.3.3 Indeks čvora

Kao što je pomenuto, najbitniji indeksi u grafovima jesu upravo indeksi potega koji indeksiraju atribute _from i _to u kolekcijama potega. Ovi indeksi pružaju veoma brz pristup svim odlaznim i dolaznim potezima određenog čvora, čime je omogućen pristup svim čvorovima susedima u grafu. Čest je slučaj da je potrebno izvršavati malo specifičnije upite,

poput onog koji bi među odlaznim potezima jednog čvora nalazio samo one koji ispunjavaju neki specifičan uslov, npr. čiji timestamp ima vrednost manju ili veću od definisane. Upravo ovaj efekat najlakše se postiže **indeksima čvorova**. Ovi indeksi za OUTBOUND[1] obilaske sortiraju kolekciju potega najpre po vrednostima _from, a potom i prema vrednostima ostalih atributa, odnosno najpre po vrednosti _to atributa, a potom i po ostalim atributima za INBOUND obilaske. Za obilaske koji nemaju definisan smer, tzv. ANY obilaske, potrebna su dva indeksa, jedan sa _from, a drugi sa _to kao prvim indeksiranim poljem.

Moguće je kreirati perzistentne indekse koji indeksiraju specijalne atribute potega, _from ili _to, ali i neke druge atribute kolekcije. Ovi indeksi će se kasnije koristiti prilikom obilaska grafova, kada odgovarajući FILTER iskazi budu pronađeni od strane optimizatora.

Kreiranje indeksa čvora gorenavedenog tipa postiglo bi se naredbom sledećeg tipa u okviru ArangoDB Shell-a:

```
db.edges.ensureIndex({"type": "persistent", "fields": ["_from", "timestamp"]});
```

Slika 10. Naredba kreiranja indeksa potega

Nakon implementacije ovog indeksa, upit kao na slici 10 bio bi značajno brži za situacije gde postoji mnogo potega koji polaze iz čvora V/1, ali samo par onih sa timestamp-om koji zadovoljava prosleđeni kriterijum. Bitno je napomenuti da se može desiti da optimizator iskoristi indeks potega pre indeksa čvora na osnovu estimacije troškova, iako se zapravo može desiti da indeks čvora pruži bolje performanse. Najčešća njihova primena zapravo je u grafovima koji su gusto povezani i koji imaju veliki broj potega.

```
FOR v, e, p IN 1..1 OUTBOUND "V/1" edges
  FILTER e.timestamp >= "2018-07-09"
  RETURN p
```

Slika 11. Upit obilaska potega u izlaznom smeru

### 4.3.4 Perzistentni indeks

Perzistentni indeks je sortirani indeks sa logaritamskom kompleksnošću za operacije upisa, ažuriranja i brisanja. Može se kreirati nad jednim ili više atributa dokumenata. Koristi se za **brz pronalazak dokumenata sa specifičnim vrednostima atributa** (lookups ili poređenja po vrednosti) i za pribavljanje dokumenata iz indeksa u sortiranom redosledu, ali samo za slučaj da su svi atributi iz indeksa navedeni u okviru upita ili ako je naveden samo atribut koji se prvi javlja u samoj definiciji indeksa. (leftmost atribut) Na slici 11 dat je prikaz

[1] OUTBOUND, INBOUND i ANY definišu smerove obilaska grafa. Za atribut OUTBOUND pribavljaju se potezi koji polaze iz specificiranog čvora. Za atribut INBOUND pribavljaju se potezi u kojima je specificirani čvor odredišni. Ako je naveden ANY, smer je nebitan

svih mogućih kombinacija naredbi, koristeći indeks koji je definisan korišćenjem atributa value1 i value2, poštojući navedeni redosled.

Ako se perzistentni indeks koristi za sortiranje, potreno je u SORT delu upita navesti sve atribute indeksa u istom redosledu kako su navedeni u okviru same definicije indeksa. Ovi indeksi se uvek kreiraju u rastućem redosledu, ali se mogu koristiti za pristup indeksiranim elementima kako u rastućem, tako i u opadajućem redosledu. Ako govorimo o kombinovanom indeksu (indeksu koji je definisan nad većim brojem atributa), smer sortiranja naveden u SORT delu upita mora da se poklapa za sve atribute.

```
FILTER doc.value1 == ...
FILTER doc.value1 < ...
FILTER doc.value1 > ...
FILTER doc.value1 > ... && doc.value1 < ...

FILTER doc.value1 == ... && doc.value2 == ...
FILTER doc.value1 == ... && doc.value2 > ...
FILTER doc.value1 == ... && doc.value2 > ... && doc.value2 < ...
```

Slika 12. Primer mogućih naredbi kreiranih korišćenjem perzistentnog indeksa

### 4.3.5 Invertovani indeks

Invertovani indeksi zaduženi su za mapiranje vrednosti atributa dokumenata sa njihovim konkretnim lokacijama u okviru kolekcija. Moguće je dodati proizvoljan broj atributa u jedan invertovani indeks. Uz to, za svaki pojedinačni atribut može se definisati Analyzer koji bi odradio procesiranje unosa, čime je moguće tokenizovati stringove za efikasan full-text search. Indeks se, takođe, opciono može sortirati kako bi se optimizovale kasnije operacije sortiranja za slučaj istih smerova sortiranja.

Invertovani indeksi jesu konzistentni, ali sa određenim kašnjenjem. Ažuriranje ovih indeksa odigrava se skoro istovremeno sa samom promenom dokumenata, ali se može desiti da dobijene vrednosti budu blago zastarele, dok sam indeks ne sustigne promene dokumenta. Kako bi se upiti ubrzali korišćenjem invertovanih indeksa, potrebno je navesti nagoveštaj ovih indeksa u samim upitima. Za razliku od drugih tipova indeksa, invertovani indeksi se ne koriste automatski, iako imaju sposobnost optimizacije upita.

### 4.3.6 Time-to-live indeks (TTL)

Ovaj tip indeksa se koristi za automatsko **brisanje dokumenata** koji su “istekli” iz kolekcije. TTL indeks je uspešno definisan nakon definisanja *expireAfter* vrednosti i odabira jedinstvenog atributa u okviru dokumenta koji ukazuje na tačan trenutak kreiranja dokumenta. Dokument će isteći nakon *expireAfter* sekundi od trenutka njegovog kreiranja.

Brisanje samih dokumenata koji su istekli neće se desiti odmah, već onda kada pozadinska nit, koja periodično prolazi kroz sve TTL indekse, uvidi da je došlo do isteka TTL indeksa. Ono na šta se može uticati jeste frekvencija kojom će se ova pozadinska nit "buditi", podešavajući opciju **–ttl.frequency**. Budući da se ne može uticati na tačan trenutak brisanja dokumenata, moguće je da se u rezultatu nekog upita nađe i neki istekli dokument. Ono što, sa druge strane, može biti garantovano jeste da će samo dokumenti čije je vreme života isteklo biti obrisani, čime zasigurno znamo da neće doći do gubitaka značajnih podataka.

### 4.3.7 Geo indeks

Korisnici mogu kreirati i dodatne geo indekse nad jednim ili više atributa u okviru kolekcije. Ovaj tip indeksa može optimizovati upite filtriranja i sortiranja po distanci između sačuvanih tačaka i tačaka definisanih u upitu. Geo indeks pamti dvodimenzionalne parove koordinata. Može se kreirati nad dva zasebna atributa u okviru dokumenta, poput latitude i longitude ili nad jedinstvenim atributom koji zapravo predstavlja niz sačinjen od dva prethodno navedena atributa. Latituda i longituda svakako moraju biti numeričke vrednosti.

Geo indeks se koristi za operacije pronalaska dokumenata čija je lokacija najbliža prosleđenoj lokaciji poređenja ili kako bi se pronašli dokumenti čija se lokacija nalazi u okviru definisanog radijusa oko lokacije poređenja.

### 4.3.8 Fulltext indeks

Fulltext indeks koriste se za pronalazak reči ili prefiksa u okviru dokumenata. Ovaj tip indeksa može se kreirati nad samo jednim atributom i indeksiraće sve reči koje u konkretnom dokumentu imaju tekstualnu vrednost sačuvanu u okviru navedenog atributa. Uz to, biće indeksirane samo reči čija dužina prelazi specificirani minimum i to prevedene u mala slova. Ovaj indeks podržava upite potpunog preklapanja, odnosno upite koji za cilj imaju pronalazak celih reči. Takođe, podržani su upiti za pronalaženje prefiksa, a u svakom upitu moguće je koristiti i logičke operacije radi kombinovanja parcijalnih rezultata.

Fulltext indeks je sparse tj. redak indeks, što znači da će indeksirati samo dokumente u kojima je vrednost atributa koji je korišćen za inicijalizaciju indeksa definisan i koji zasigurno nije prazan. Takođe, samo reči koje su duže od definisane minimalne dužine će biti indeksirane.

Kako bi se ovaj indeks koristio potrebno je koristiti definisane funkcije AQL-a, ali njegovo korišćenje nije moguće mimo tih upita ili uslova.

**Fulltext indeks je zastareo (eng. deprecated) počev od verzije 3.10. Preporuka je umesto ovog indeksa koristiti ArangoSearch funkcionalnost, radi postizanja naprednijih full-text pretraga.**

## 4.4 Kako ArangoDB optimizuje upite pomoću indeksa?

U najvećem broju slučajeva, ArangoDB će koristiti jedan indeks po kolekciji u zadatom upitu. Moguće je u okviru AQL upita koristiti i više od jednog indeksa po kolekciji kada se više uslova filtriranja u FILTER delu upita kombinuje logičkim operacijom OR.

Kreiranje višestrukih indeksa na različitim atributima jedne kolekcije može optimizatoru upita (eng. *query optimizer*) dati više izbora prilikom odabira indeksa. Generalno je mišljenje da je bolje kreirati indeks nad više atributa istovremeno, umesto nad pojedinačnim atributima. Što je indeks „kompleksniji“ i zavisniji od većeg broja atributa, to je selektivniji, što znači da se smanjuje broj dokumenata koje će upiti morati da obrađuju.

Veći broj indeksa u ArangoDB bazi podataka podržava i metode za **procenu selektivnosti**. Korišćenje **indexes()** metode obezbeđuje ovu procenu za konkretan indeks, dok je moguće nad već kreiranim upitom pozvati **explain()** metodu, koja će kao rezultat dodatno dati i detaljan plan izvršenja ovog upita. Ove vrednosti procene selektivnosti još se koriste i od strane optimizatora upita prilikom kreiranja plana izvršenja upita, onda kada postoji više indeksa koje potencijalno može iskoristiti u konkretnom upitu. Optimizator će iskoristiti onu kombinaciju indeksa čiji su ukupni troškovi izvršenja najmanji, ali će zapravo biti odabrani oni indeksi čija je procenjena selektivnost najveća.

# 5. Praktična primena u ArangoDB bazi podataka

Pre svega, potrebno je definisati način pristupa indeksima. Svakom definisanom indeksu u okviru baze podataka dodeljen je jedinstveni identifikator na nivou kolekcije. Ova vrednost je automatski generisana od same baze podataka. Indeks je moguće jedinstveno identifikovati i na nivou cele baze podataka, navođenjem naziva kolekcije kao prefiksa. Imajući sve navedeno u vidu, moguće je pristupiti određenom indeksu na neki od načina prikazanim na slici 9.

```
db.collection.index("<index-identifier>");
db.collection.index("<collection-name>/<index-identifier>");
db._index("<collection-name>/<index-identifier>");
```

Slika 13. Primer pristupa indeksu

## 5.1 Kreiranje indeksa

Za kreiranje indeksa koristi se metoda ensureIndex, koja se poziva na sledeći način:

**collection.ensureIndex(index-description)**

Ova naredba za cilj ima zapravo proveru postojanja indeksa sa navedenim opisom. Za slučaj da takav indeks ne postoji, biće kreiran novi. Poziv ove metode vratiće objekat indeksa, a na osnovu vrednosti atributa ***isNewlyCreated*** može se zaključiti da li je indeks već postojao, ili je novokreiran. U okviru opisa samog indeksa, koji se zadaje kroz index-description, potrebno je navesti i kojeg je tipa indeks koji se kreira (po analogiji sa tipovima indeksa koje ova baza podataka ima, podržane vrednosti su *persistent, inverted, ttl, geo* itd.), a u skladu sa zahtevima specifičnih tipova indeksa, može biti neophodno dodati i neki drugi atribut.

- fields – niz putanja atributa, koji sadrži jedan ili više atributa koje je potrebno indeksirati. Neki indeksi mogu se odnositi na samo jedan atribut, dok ima i onih koji dozvoljavaju navođenje više atributa. Bitno je napomenuti da je, ako se koristi više atributa, bitan redosled njihovog navođenja.
- storedValues – neki tipovi indeksa, poput invertovanog i perzistentnog, omogućavaju čuvanje dodatnih atributa u samom indeksu. Ovi atributi se ne mogu koristiti za lookup-ove ili sortiranje
- name – atribut imena indeksa, lokalno sačuvan u obliku stringa. Ukoliko je izostavljen, biće automatski generisan. Ideja korisnički definisanih imena indeksa omogućava njihovo lakše korišćenje u daljem radu.
- sparse – boolean vrednost. Moguće je, pomoću sparse atributa, kontrolisati gustinu indeksa za perzistentne indekse, dok su invertovani, fulltext i geo indeksi po definiciji retki (sparse) indeksi
- unique – boolean vrednost. Po default-u, svi korisnički definisani indeksi nisu jedinstveni, te je potrebno eksplicitno ovo zahtevati.
- cacheEnabled – boolean vrednost, podržana za perzistentne indekse. Ovaj atribut kontroliše to da li će dodatna keš memorija biti kreirana za konkretan indeks.

## 5.2 Primeri optimizacije upita korišćenjem indeksa

U nastavku rada biće dat pregled upita koji bi postigli svojevrsnu optimizaciju uvođenjem indeksa. Optimizacija upita biće ilustrovana korišćenjem baze podataka za sistem iznajmljivanja vozila, koja pre izrade ovog rada nije bila potkrepljena indeksima.

### Pribavljanje podataka na osnovu konkretne vrednosti atributa – lookup upit

Prvi pokušaj optimizacije pokušan je na jednostavnom upitu, čiji je cilj pribavljanje svih modela automobila specificiranog brenda. Na slici 14 dat je plan izvršenja ovog upita pre kreiranja bilo kakvog indeksa. Budući da je potrebno pronaći konkretnu vrednost atributa *Brend*, ne postoji druga opcija do prolaska kroz sve dokumente u okviru kolekcije, kojih je u datom primeru bilo 36.

```
Query String (60 chars, cacheable: true):
 FOR a IN automobili
 FILTER a.Brend == "Volkswagen"
 RETURN a


Execution plan:
 Id   NodeType                Est.   Comment
  1   SingletonNode              1   * ROOT
  2   EnumerateCollectionNode   36     - FOR a IN automobili   /* full collection scan  */   FILTER (a.`Brend` == "Volkswagen")
  5   ReturnNode                36       - RETURN a

Indexes used:
 none

Optimization rules applied:
 Id   RuleName
  1   move-filters-into-enumerate
```

9 elements 0.563 ms

Slika 14. Plan izvršenja neoptimizovanog lookup upita

Ilustracije radi, a uzimajući u obzir i čestu implementaciju na sajtovima za iznajmljivanje vozila, a to je filtriranje po brendu automobila, kreiran je **perzistentni** indeks nad atributom Brend. Rezultat planiranja ovog upita, nakon kreiranja indeksa data je na slici 15. Može se uočiti da je, iako se radi o maloj količini podataka, svakako postignuta optimizacija jer je vreme izvršenja upita sa 0.563ms smanjena na 0.430ms. Plan upita ukazuje na to da je umesto sekvencijalnog prolaska kroz 36 dokumenata, upotrebljen indeks iBrend.

```
Query String (60 chars, cacheable: true):
 FOR a IN automobili
 FILTER a.Brend == "Volkswagen"
 RETURN a


Execution plan:
 Id   NodeType        Est.   Comment
  1   SingletonNode      1   * ROOT
  6   IndexNode          2     - FOR a IN automobili   /* persistent index scan, index scan + document lookup */
  5   ReturnNode         2       - RETURN a

Indexes used:
 By   Name     Type         Collection   Unique   Sparse   Cache   Selectivity   Fields       Stored values   Ranges
  6   iBrend   persistent   automobili   false    true     false       36.11 %   [ `Brend` ]  [ ]             (a.`Brend` == "Volkswagen")

Optimization rules applied:
 Id   RuleName
  1   use-indexes
  2   remove-filter-covered-by-index
  3   remove-unnecessary-calculations-2
```

9 elements 0.430 ms

Slika 15. Plan izvršenja lookup upita nakon kreiranja perzistentnog indeksa

Na slici 16 prikazana je naredba kreiranja indeksa iBrend koji je korišćen u gorenavedenom upitu.

```
db.automobili.ensureIndex({
  name: "iBrend",
  type: "persistent",
  fields: ["Brend"],
  cacheEnabled: true,
  sparse: true
})
```

Slika 16. Kreiranje perzistentnog indeksa

## Pribavljanje potega sa konkretnim vrednostima atributa iz kolekcije potega

Naredni primer odnosi se na optimizaciju upita izvršenih nad kolekcijom potega. U samom upitu upotrebljen je atribut potega. Analogno prethodnom primeru, neoptimizovani upit bi podrazumevao sekvencijalni prolazak kroz sve podatke, u konkretnom slučaju 23.

```
Query String (87 chars, cacheable: true):
 FOR rent IN korisnikRentirao
 FILTER rent.startDate == "4/4/2020"
 RETURN rent.startDate


Execution plan:
 Id   NodeType                  Est.   Comment
  1   SingletonNode                1   * ROOT
  2   EnumerateCollectionNode     23     - FOR rent IN korisnikRentirao   /* full collection scan
  5   CalculationNode             23       - LET #3 = rent.`startDate`    /* attribute expression *
  6   ReturnNode                  23       - RETURN #3

Indexes used:
 none

Optimization rules applied:
 Id   RuleName
  1   move-calculations-up
  2   move-filters-up
  3   move-calculations-up-2
  4   move-filters-up-2
  5   move-filters-into-enumerate
  6   reduce-extraction-to-projection
```

1 elements 0.687 ms

Slika 17. Primer plana izvršenja neoptimizovanog upita pribavljanja atributa

Samo uvođenjem indeksa nad specificiranim atributom, postignuto je poboljšanje u vremenu izvršenja od preko 270ms.

```
Query String (87 chars, cacheable: true):
 FOR rent IN korisnikRentirao
 FILTER rent.startDate == "4/4/2020"
 RETURN rent.startDate


Execution plan:
 Id   NodeType          Est.   Comment
  1   SingletonNode        1   * ROOT
  7   IndexNode            7     - FOR rent IN korisnikRentirao   /* persistent index scan, index only (proj
  5   CalculationNode      7       - LET #3 = rent.`startDate`   /* attribute expression */   /* collections
  6   ReturnNode           7       - RETURN #3

Indexes used:
 By   Name         Type         Collection         Unique   Sparse   Cache   Selectivity   Fields
  7   iStartDate   persistent   korisnikRentirao   false    false    false       13.04 %   [ `startDate` ]

Optimization rules applied:
 Id   RuleName
  1   move-calculations-up
  2   move-filters-up
  3   move-calculations-up-2
  4   move-filters-up-2
  5   use-indexes
  6   remove-filter-covered-by-index
  7   remove-unnecessary-calculations-2
  8   reduce-extraction-to-projection
```

1 elements 0.410 ms

Slika 18. Primer plana izvršenja optimizovanog upita

# 6. Zaključak

Odabir broja i tipa indeksa može drastično uticati na performanse, omogućavajući pribavljanje podataka u svega par sekundi sa par ulazno/izlaznih operacija ili, sa druge strane, kreirajući upite čije će izvršenje trajati minutima, pa čak i satima sa mnoštvom izvršenih operacija čitanja i upisa. Stoga zaključujemo da je odabir optimalnog broja indeksa koji bi se mogli koristiti u upitima od kritičnog značaja i smatra se izuzetno bitnim zadatkom, koji se nipošto ne bi smeo zanemariti.

Iako indeksi poboljšavaju brzinu pretraživanja, oni dodaju dodatni sloj složenosti i mogu povećati troškove održavanja baze podataka. Previše indeksa može dovesti do smanjenja performansi tokom operacija upisa. Takođe, ne sme se zanemariti ni prostor potreban za skladištenje indeksa, što može biti posebno značajno u velikim bazama podataka.

U radu s indeksima upostavljanje pravog balansa je ključno. Optimizacija upita, odabir pravih tipova indeksa i održavanje odgovarajuće količine istih jesu zadaci od vitalnog značaja. Jedan od izazova je osigurati da se indeksi koriste na način koji doprinosi performansama, a da se istovremeno izbegava preopterećenje baze podataka preteranim indeksiranjem.

Upotreba indeksa u bazama podataka zasigurno može dovesti do poboljšanja performansi čitavog sistema. Međutim potrebno je mudro iskoristiti indekse kako bi se postiglo istinsko poboljšanje.

# 7. Literatura


[1] “ArangoDB,” .Available: https://www.arangodb.com.

[2] “About indexing and indexes”, Available: https://docs.arangodb.com/stable/index-and-search/

[3] A. Inc., “What is a multi-model database and why use it?,” Available: https://www.arangodb.com/resources/white-paper/multi-model-database/.

[4] “ArangoDB Documentation - Manual,” Available: https://www.arangodb.com/docs/stable

[5] “ArangoDB Documentation - AQL,” Available: https://www.arangodb.com/docs/stable/aql

[6] A. Inc., “Switching From Relational Databases to ArangoDB,” Available: https://www.arangodb.com/resources/white-paper/coming-from-relational/.

[7] S.Milenković, “ArangoDB baza podataka”, diplomski rad, 2022.

[8] C.Cioloca, M. Georgescu, “Increasing Database Performance using Indexes”, 2011.

[9] V. Blagojević “Relacione baze podataka”, 2006.

[10] Y. Manolopoulos, Y. Theodoridis, V. J. Tsotras, “Advanced database indexing”, 2000.