



UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET
Katedra za računarstvo



INTERNA STRUKTURA I ORGANIZACIJA INDEKSA KOD APACHE CASSANDRA BAZE PODATAKA

Predmet: Sistemi za upravljanje bazama podataka
-Seminarski rad-

Student:

Marijana Cvetković, br. ind. 1431

Mentor:

Doc. dr Aleksandar Stanimirović

Niš, april 2022. godina

Sadržaj

1. Uvod	3
2. Apache Cassandra	4
2.1. Interna struktura	5
2.2. Koncept indeksiranja	7
2.2.1. Primarni i sekundarni indeksi.....	9
2.2.2. Korišćenje sekundarnog indeksa	9
2.2.3. Korišćenje više indeksa	11
2.2.4. Indeksiranje kolekcija	11
2.2.5. SASI indeks.....	14
2.2.6. Indeksiranje vezano za skladištenje (SAI).....	22
2.2.7. Problemi sa korišćenjem indeksa	23
2.2.8. Primeri primene	23
3. Praktična primena	26
3.1. Pokretanje Cassandra baze podataka	26
3.2. Skup podataka.....	27
3.3. Unos i upotreba podataka	28
4. Zaključak	33
5. Literatura.....	34

1. Uvod

Apache Cassandra je NoSQL distribuirana baza podataka otvorenog koda. Predstavlja particionisani model skladištenja širokih kolona sa konačno doslednom semantikom. Po dizajnu NoSQL baze podataka su lagane, otvorenog koda, nerelacione i u velikoj meri distribuirane. Cassandra baze podataka lako se skaliraju kada je aplikacija pod velikim stresom, a distribucija takođe sprečava gubitak podataka usled kvara hardvera bilo kog data centra. Distribuirano znači da Cassandra može da radi na više mašina dok se korisnicima čini kao jedinstvena celina. Koncept indeksiranja kod Cassandre predstavlja poseban način pristupa podacima, pristup onim atributima koji predstavljaju kolone koji nisu deo particionog ključa. Na taj način je moguće pristupiti, pretražiti podatke po određenom uslovu i prednost takvog traženja su efikasnost i brzina.

U ovom radu biće detaljno opisana interna struktura Cassandra baze podataka, definicija i organizacija indeksa, prednosti i mane njihovog korišćenja, kao i dati primeri upotrebe. Biće navedeni primeri u kojim slučajevima je pogodno koristiti indekse, a kada je to nemoguće učiniti. Biće objašnjena razlika između primarnog i sekundarnog indeksa i navedeni načini njihove upotrebe i načina kreiranja. Moguća je upotreba indeksiranja nad različitim skupovima podataka, kao što su kolekcije, skupovi i liste, što će detaljno biti objašnjeno u daljem radu. Na kraju drugog poglavlja se nalaze problemi koji mogu nastati pri korišćenju indeksa.

U praktičnom delu ovog rada biće prikazana upotreba indeksa nad skupom podataka koji se odnose na zakazivanje koncerata, prikazan način kreiranja i dodavanja podataka. Zatim će biti prikazani primeri u kojima se mogu koristiti indeksi, a kada ne i iz kojih razloga, kao i prednosti i mane takvog načina upotrebe.

Na kraju, u poslednjem poglavlju, nalazi se zaključak, izveden na osnovu teorijskog i praktičnog dela rada. Biće pomenute prednosti i mane korišćenja ove baze podataka i rada sa njom.

2. Apache Cassandra

Cassandra je open source distribuirani sistem za upravljanje bazom podataka. Ova baza podataka podržava upravljanje velikom količinom podataka. Razvijena je od strane Facebook-a, za potrebe inbox search-a, koristeći arhitekturu vođenu događajima za implementaciju kombinacije Amazanovih Dinamo tehnika distribuiranog skladištenja i replikacije Google-vog BigTable modela podataka i mehanizma za skladištenje podataka. I Dinamo i BigTable razvijeni su da zadovolje nove zahteve za skalabilnim, pouzdanim i visoko dostupnim sistemima za skladištenje podataka, ali svaki je imao oblasti koje bi se mogle poboljšati. [1]

Cassandra je dizajnirana kao najbolja kombinacija oba sistema u klasi kako bi se zadovoljile potrebe za skladištenjem podataka velikih razmera u nastajanju, kako u pogledu otisaka podataka, tako i u obimu upita. Sistemi kao što je Cassandra dizajnirani su i traže ciljeve kao što su: fleksibilna šema, skaliranje na robnom hardveru, globalna dostupnost uz malo kašnjenje, particionisani upiti orijentisani na ključ i ostalo. [1]

Cassandra je 2008. godine objavljena kao open source rešenje. Danas je deo Apache fondacije i jedno je od najpopularnijih NoSQL rešenja. Na slici 1 prikazan je logo Apache Cassandra baze podataka.



Slika 1: Cassandra logo

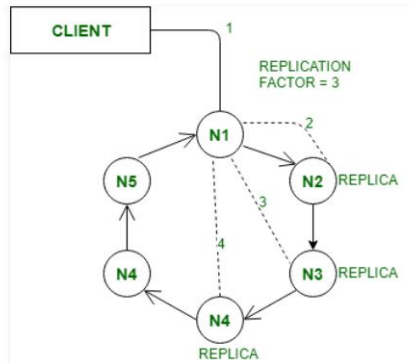
Cassandra baza podataka koristi svoj upitni jezik koji se naziva 'Cassandra Query Language (CQL)'. Cassandra nudi model sličan SQL-u, za kreiranje i ažuriranje šeme baze podataka i pristup podacima. Podaci se čuvaju u tabelama koje sadrže redove kolona.

U Apache Cassandri ne postoji arhitektura master-klijent. Ima peer to peer arhitektutu. Mogu se kreirati više kopija podataka u vreme kreiranja prostora ključeva, može se jednostavno definisati strategija replikacije i RF(faktor replikacije) za kreiranje više kopija podataka. Primer kreiranja se vidi na slici 2: [2]

```
CREATE KEYSPACE Example
WITH replication = {'class': 'NetworkTopologyStrategy',
                    'replication_factor': '3'};
```

Slika 2: Faktor replikacije

U ovom primeru je definisan faktor replikacije 3 što jednostavno znači da se kreiraju 3 kopije podataka preko više čvorova u smeru kazaljke na satu, što je prikazano na slici 3:



Slika 3: Faktor replikacije

Za razliku od modela relacione baze podataka u kojem upiti koriste spajanje tabela za dobijanje podataka iz više tabela, Cassandra ne podržava spajanja tako da sva obavezna polja (kolone) moraju biti grupisana zajedno u jednu tabelu. Pošto svaki upit podržava tabelu, podaci se dupliraju u više tabela u procesu poznatom kao denormalizacija. Dupliciranje podataka i velika propusnost pisanja se koriste za postizanje visokih performansi čitanja. [1]

2.1. Interna struktura

Interna struktura baze podataka predstavlja kako se podaci fizički skladište i organizuju (npr. indeksi, putanja pristupa,...). Odnosi se na organizaciju datoteka i njihov fizički zapis na disku. Interni (fizički) nivo obuhvata sadržaj cele baze podataka i definiše način fizičke organizacije na medijumima gde su podaci smešteni (formati zapisa u memoriji, organizacija i drugo).

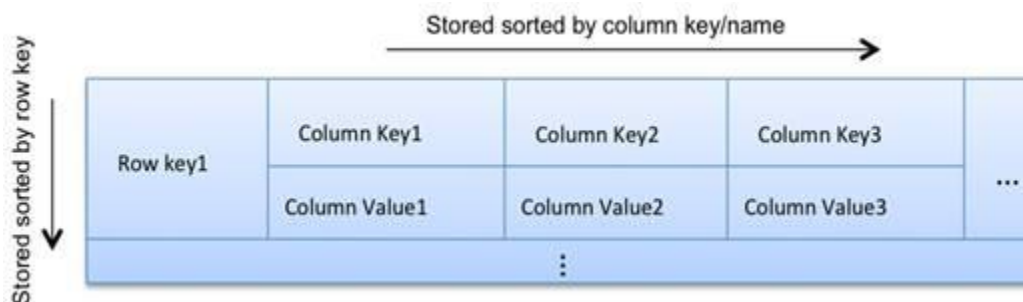
Cassandra model podataka je šema-opcioni model podataka orijentisan na kolone. To znači da, za razliku od relacione baze podataka, ne mora unapred da se modeluju sve kolone koje zahteva neka aplikacija, jer svaki red ne mora da ima isti skup podataka.

Cassandra model podataka sastoji se od prostora ključeva (keyspaces, analogno bazama podataka), porodica kolona (column families, analogno tabelama u relacionom modelu), ključeva (keys) i kolona (columns). [3] U tabeli 1 nalazi se prikaz pojmova koji se koriste u Cassandri u poređenju sa poznatim pojmovima kod relacionih baza podataka:

Tabela 1: Model podataka

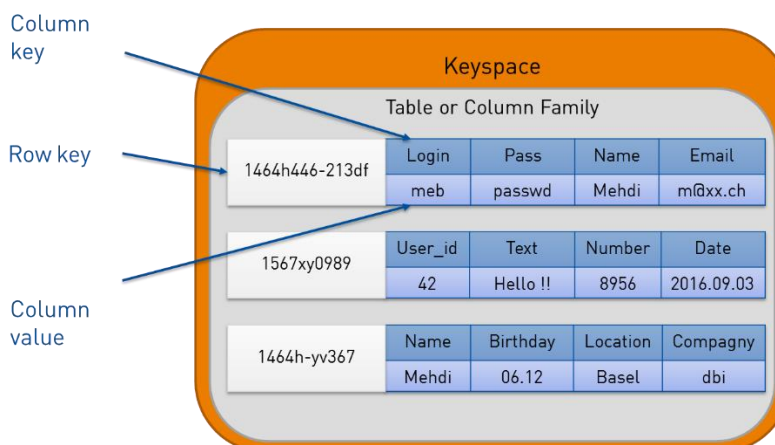
Relacioni model	Cassandra model
Database	Keyspace
Table	Column Family(CF)
Primary key	Row key
Column name	Column name/key
Column value	Column value

Svaka porodica kolona ne može se porediti sa relacionom tabelom. Nešto približnije poređenje je ugnježdjena struktura podataka mape. Interna struktura podataka u Cassandra bazi podataka prikazana je na slici 4:



Slika 4: Interna struktura podataka

Mapa omogućava efikasno traženje ključeva, a sortirana priroda daje efikasna skeniranja. U Cassandri broj ključeva kolona je neograničen, tj. mogu postojati dugački redovi. Ključ može da sadrži vrednost kao deo imena ključa, što znači da može postojati kolona bez vrednosti. Na slici 5 je prikazan izgled DataModela i dat primer nekim sa podacima:



Slika 5: DataModel

Apache Cassandra je distribuirana baza podataka koja skladišti podatke u klasteru čvorova. Particioni ključ se koristi za particionisanje podataka među čvorovima. Cassandra deli podatke preko čvorova za skladištenje koristeći varijantu doslednog heširanja za distribuciju podataka. Heširanje je tehnika koja se koristi za mapiranje podataka pomoću kojih heš funkcija generiše heš vrednost (ili jednostavno heš) koja se čuva u heš tabeli. Ključ particije se generiše iz prvog polja primarnog ključa. Podaci podeljeni u heš tabele pomoću particionih ključeva omogućavaju brzo traženje. Manje particija koje se koriste za upit je brže vreme odgovora na upit.

Recimo imamo primer particionisanja, tabela *tablee* u kojoj je *id* jedino polje u primarnom ključu:

```
CREATE TABLE tablee (
  id int,
  k int,
  v text,
```

```
PRIMARY KEY (id)
```

```
);
```

Particioni ključ se generiše iz primarnog ključa *id* za distribuciju podataka po čvorovima u klasteru. Varijacija tabele *tablee* koja ima dva polja koja čine primarni ključ da bi se napravili složeni ili složeni primarni ključ:

```
CREATE TABLE tablee (
```

```
  id int,
```

```
  c text,
```

```
  v text,
```

```
  PRIMARY KEY (id,c)
```

```
);
```

U ovom slučaju postoji složeni primarni ključ, gde se prvo polje *id* koristi za generisanje ključa particije, a drugo polje *c* je za grupisanje koje se koristi za sortiranje unutar particije. Particionisanje podataka poboljšava efikasnost čitanja i pisanja. Ostala polja koja nisu deo primarnog ključa, mogu se zasebno indeksirati kako bi se dodatno poboljšale performanse upita. Dizajn primarnog ključa je izuzetno važan, jer će odrediti koliko podataka će biti uskladišteno na svakoj particiji i kako su ti podaci organizovani na disku, što će zauzvrat uticati na brzinu čitanja Cassandra procesa.

Upit za podatke iz podataka vrši se uz pomoć SELECT naredbe. Naredba select čita jednu ili više kolona za jedan ili više redova u tabeli. Vraća skup rezultata redova koji odgovaraju zahtevu, gde svaki red sadrži vrednosti za izbor koji odgovara upitu. Primer jedne select naredbe je sledeći:[1]

```
SELECT time, value
```

```
FROM events
```

```
WHERE event_type='myEvent'
```

```
AND time>= '2012-01-01'
```

2.2. Koncept indeksiranja

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 tabel 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.

Postoje dva tipa mehanizma organizacije datoteka koje prate metode indeksiranja za skladištenje podataka i to su sekvencijalna organizacija datoteka i heš organizacija fajla. Kod sekvencijalne organizacije datoteke indeksi su zasnovani na sortiranom redosledu vrednosti. Ovakva organizacija može čuvati podatke u gostom ili razređenom formatu. Kod heš organizacije indeksi se zasnivaju na vrednostima koje su ravnomerno raspoređene u rasponu grupa. Heš funkcija se primenjuje na kolone ili attribute da bi se dobila adresa bloka. Generisana heš funkcija u koloni koja se smatra ključem, naziva se heš ključ, a ako je generisana nad kolonom koja nije ključ, kolona se naziva heš kolona.

Postoje tri metode indeksiranja: klasterizovano, neklasterizovano i indeksiranje na više nivoa. Klasterizovano indeksiranje predstavlja način kada je dva zapisa uskladišteno u istoj datoteci. Ovaj pristup omogućava smanjenje troškova pretraživanja, jer se na istom mestu nalaze podaci koji se odnose na istu stvar. Indeks klastera je definisan na uređenoj datoteci podataka. Ovakva datoteka je uređena po polju koja nije ključ. Primarno indeksiranje je podrazumevani format indeksiranja kod sekvencijalnih datoteka i to je vrsta klasterizovanog indeksiranja. Sa druge strane, neklasterizovani indeks govori gde leže podaci, daje listu referenci na lokaciju gde su podaci skladišteni. Podaci se ne čuvaju po redosledu indeksa, oni su prisutni u krajnjim čvorovima, tj. listovima stabla. Pošto se indeks čuva u glavnoj memoriji, vremenom skladištenje velikoj broja indeksa i podataka može zauzeti veliki prostor. Iz tog razloga nastali su i indeksi na više nivoa, gde se glavni blok odvaja na manje blokove.

Podacima kod Cassandre baze podataka se može pristupiti korišćenjem atributa koji imaju ključ particije. Na primer `Emp_id` ime kolone za tabelu `Employee` i ako je to particioni ključ te tabele, onda možemo da filtriramo ili pretražimo podatke uz pomoć particionog ključa. Klauzula `where` se koristi da se definiše uslov nad atributom kako bi se pretražili određeni podaci. Kod Cassandre, ukoliko postoji kolona koja nije deo particionog ključa, a mi želimo da pretražimo takvu tabelu i filtriramo, pretražimo i pristupimo podacima koristeći `where` klauzulu po toj koloni, takav upit neće biti uspešno izvršen i daće grešku. Kako bi moglo da se pristupa podacima koristeći attribute koji nisu deo particionog ključa za brzo i efikasno traženje podataka koji odgovaraju datom uslovu, onda mora da se definiše indeks. Indeks se može koristiti u različite svrhe, kao što su kolekcije, statičke kolone, kolone za prikupljanje i bilo koje druge kolone osim kolona brojača. Prednost je brzo i efikasno traženje podataka koji odgovaraju datom uslovu. U stvari, ukoliko nema indeksa na normalnoj koloni, čak nije dozvoljeno ni uslovno upiti po kolonama.

Indeks indeksira vrednosti kolona u zasebnoj, skrivenoj porodici kolona (tabeli) od one koja sadrži vrednosti koje se indeksiraju. Podaci indeksa su samo lokalni, što znači da neće replicirati vrednosti koje se indeksiraju. Ovo takođe znači da za upit podataka po indeksiranoj koloni, zahtevi moraju biti prosleđeni svim čvorovima, čekajući sve reakcije, a zatim se rezultati spajaju i vraćaju. Dakle, ukoliko postoje više čvorova, odgovor na upit se usporava kako se više mašina dodaje u klaster. [4]

Kada se koristi indeks:

- Ugrađeni indeksi su najbolja opcija u tabeli koja ima mnogo redova i koji redovi sadrže indeksiranu vrednost.
- U određenoj koloni koja kolona ima više jedinstvenih vrednosti u tom slučaju može se koristiti indeksiranje.
- Tabela koja ima više troškova zbog nekoliko razloga kao što je kolona koja ima više unosa nego u tom slučaju može se koristiti indeksiranje.
- Za ispitivanje i održavanje indeksa može se koristiti indeksiranje koje je u tom slučaju uvek dobra opcija. [5]

Indeks se ne koristi u situacijama:

- Na kolonama visokog kardinaliteta, jer se onda postavljaju upiti velikom broju zapisa za mali broj rezultata
- U tabelama koje koriste kolonu brojača

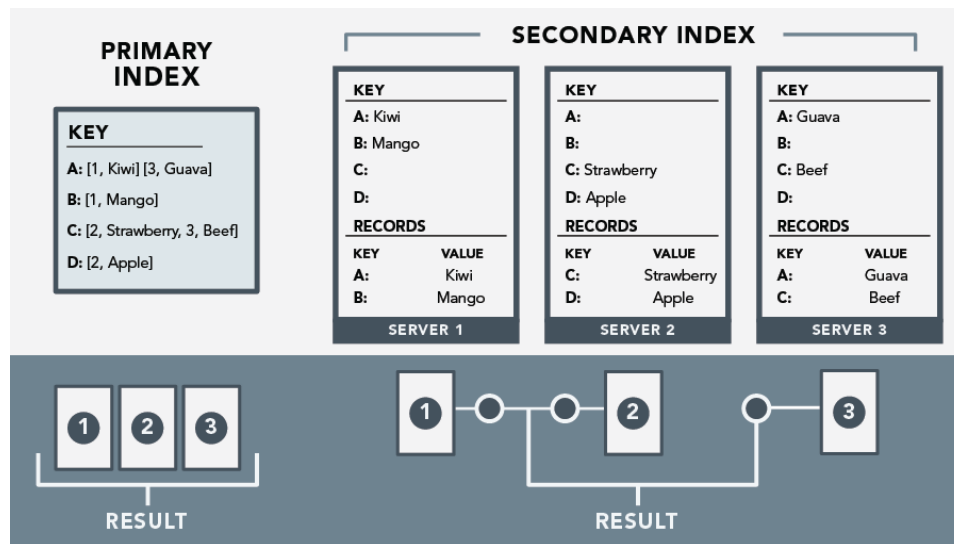
- U koloni koja se često ažurira i briše
- Da bi se pretražio red u velikoj particiji, osim ako nije usko upitan. [4]

2.2.1. Primarni i sekundarni indeksi

Primarni indeks je poznat kao jedinstveni ključ, ili u Cassandra rečniku, particioni ključ. Kao primarni metod pristupa bazi podataka, Cassandra koristi particioni ključ da identifikuje čvor koji drži podatke, a zatim datoteku podataka koja čuva particiju podataka.

Imamo tabelu korisnika, koji sadrži e-poštu korisnika. Primarni indeks bio bi korisnički ID, tako da ukoliko je potrebno pristupiti e-pošti određenog korisnika, mogu se pretražiti po njihovom ID-u. Ukoliko je potrebno izvršiti inverzni upit – da se preko email-a dobije korisnički ID, neophodno je koristiti sekundarni indeks.

Primarni indeks je globalni, dok je sekundarni indeks lokalni. Na slici je dat primer primarnih i sekundarnih indeksa nekog sistema.



Slika 6: Primarni i sekundarni indeksi [6]

Ukoliko se koristi Cassandra na prstenu od pet mašina, sa primarnim indeksom korisničkih ID-jeva i sekundarnim indeksom korisničkih imejlova. Ako bi se tražio korisnik po ID-u ili po primarnom indeksiranom ključu-bilo koja mašina u prstenu bi znala koja mašina ima zapis tog korisnika. Međutim, ukoliko bi upit bio preko e-pošte ili sekundarno indeksirane vrednosti svaka mašina mora da upita svoj sopstveni zapis korisnika. Jedan upit, pet čitanja sa diska. Problem je što se povećava time odnos buke i signala i ukupna efikasnost čitanja opada-u nekim slučajevima do isteka vremena za API pozive.

2.2.2. Korišćenje sekundarnog indeksa

Koristeći CQL, može se kreirati indeks na koloni nakon definisanja tabele. Takođe može se indeksirati kolona kolekcije. Sekundarni indeksi se koriste za ispitivanje tabele pomoću kolone koja se inače ne može ispitivati.

Sekundarni indeksi su teški za korišćenje i mogu značajno uticati na performanse. Tabela indeksa se čuva na svakom čvoru u klasteru, tako da upit koji uključuje sekundarni indeks može brzo postati noćna mora performansi ako se pristupi više čvorova. Opšte pravilo je da se indeksira kolona sa malom kardinalnošću od nekoliko vrednosti. Pre kreiranja indeksa, treba biti svestan kada i ne treba kreirati indeks, što je opisano u poglavlju iznad. [7]

CQL podržava kreiranje sekundarnih indeksa na tabelama, omogućavajući upitima u tabeli da koriste te indekse. Sekundarni indeks je identifikovani imenom definisanim sa:

```
index_name ::= re('[a-zA-Z_0-9]+')
```

Kreiranje sekundarnog indeksa na tabeli koristi CREATE INDEX naredbu:

```
create_index_statement ::= CREATE [ CUSTOM ] INDEX [ IF NOT EXISTS ] [ index_name ]  
    ON table_name '(' index_identifier ')'  
    [ USING string [ WITH OPTIONS = map_literal ] ]  
index_identifier ::= column_name  
    | ( KEYS | VALUES | ENTRIES | FULL ) '(' column_name ')'
```

Na primer:

```
CREATE INDEX userIndex ON NerdMovies (user);  
CREATE INDEX ON Mutants (abilityId);  
CREATE INDEX ON users (keys(favs));  
CREATE CUSTOM INDEX ON users (email)  
    USING 'path.to.the.IndexClass';  
CREATE CUSTOM INDEX ON users (email)  
    USING 'path.to.the.IndexClass'  
    WITH OPTIONS = { 'storage': '/mnt/ssd/indexes/' };
```

Naredba CREATE INDEX koristi se za kreiranje novog (automatskog) sekundarnog indeksa za datu (postojeću) kolonu u tabeli. Ime za sam indeks može se navesti ispred ON ključne reči, ako se želi. Ako podaci postoje već za kolonu, oni će biti asinhrono indeksirani. Nakon kreiranja indeksa, novi podaci za kolonu se automatski indeksiraju u vreme umetanja.

Pokušaj kreiranja već postojećeg indeksa će vratiti grešku osim ako se IF NOT EXISTS opcija ne koristi. Ako se koristi, izjava će biti no-op ako indeks već postoji. Cassandra 3.4 (i novije verzije) podržava indeksiranje statičkih kolona. Cassandra odbija pokušaj kreiranja indeksa i za ključ i za vrednost. Pre samog kreiranja indeksa treba biti svestan kada je moguće koristiti ih, jer utiče na performanse. U Cassandra 3.4 i novijim verzijama dodat je novi prilagođeni SASI indeks.

Kada se kreira indeks na maps<maps>, mogu se indeksirati ili ključevi ili vrednosti. Ako je identifikator kolone postavljen unutar keys() funkcije, indeks će biti na ključevima mape, što omogućava da se koriste CONTAINS KEY u WHERE klauzulama. U suprotnom, indeks će biti na vrednostima karte.

Brisanje sekundarnog indeksa se vrši uz pomoć DROP INDEX:

```
drop_index_statement ::= DROP INDEX [ IF EXISTS ] index_name
```

DROP INDEX se koristi za brisanje postojećeg indeksa. Argument naredbe je ime indeksa, koje može opcionalno da specificira prostor ključeva indeksa. Ako indeks ne postoji, naredba će vratiti grešku, osim ako IF EXISTS se ne koristi u kom slučaju je opcija bez opcije. [8]

2.2.3. Korišćenje više indeksa

Indeksi se mogu kreirati na više kolona i koristiti u upitima. Opšte pravilo o kardinalnosti važi za sve indeksirane kolone. U stvarnoj situaciji, određene kolone možda nisu dobar izbor, u zavisnosti od njihove kardinalnosti.[9]

Primeri:

- Tabela cycling.cyclist_alt_stats može dati statistiku o biciklistima:

```
cqlsh> CREATE TABLE cycling.cyclist_alt_stats ( id UUID PRIMARY KEY, lastname text, birthday timestamp, nationality text, weight text, height text );
```

- Kreiranje indeksa na kolonama rođendan i nacionalnost:

```
cqlsh> CREATE INDEX birthday_idx ON cycling.cyclist_alt_stats ( birthday );  
  
CREATE INDEX nationality_idx ON cycling.cyclist_alt_stats ( nationality );
```

- Upit za sve bicikliste sa određenim rođendanom iz određene zemlje:

```
cqlsh> SELECT * FROM cycling.cyclist_alt_stats WHERE birthday = '1982-01-29' AND nationality = 'Russia';
```

```
cqlsh:cycling> SELECT * FROM cycling.cyclist_alt_stats WHERE birthday = '1982-01-29' AND nationality = 'Russia';  
InvalidRequest: code=2200 [Invalid query] message="Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING"
```

Slika 7: Rezultat greška

- Indeksi su kreirani na odgovarajućim kolonama niske kardinalnosti, ali upit i dalje ne uspeva. Odgovor je u ključu particije, koji nije definisan. Kada se pokuša sa potencijalno skupim upitom, kao što je pretraživanje niza redova, Cassandra zahteva direktivu ALLOW FILTERING. Greška nije zbog više indeksa, već zbog nedostatka definicije ključa particije u upitu. [9]

```
cqlsh> SELECT * FROM cycling.cyclist_alt_stats WHERE birthday = '1990-05-27' AND nationality = 'Portugal' ALLOW FILTERING
```

Sada se dobija rezultat koji je očekivan (slika):

id	birthday	height	lastname	nationality	weight
1ba0417d-62da-4103-b710-de6fb227db6f	1990-05-27 00:00:00-0700	null	PAULINHO	Portugal	null

Slika 8: Rezultat

2.2.4. Indeksiranje kolekcija

Kolekcije se mogu indeksirati i ispitivati da bi se pronašla kolekcija koja sadrži određenu vrednost. Skupovi i liste su indeksirani malo drugačije od mapa, s obzirom na prirodu ključ/vrednost mapa.

Skupovi i liste mogu indeksirati sve vrednosti pronađene indeksiranjem kolone kolekcije. Mape mogu indeksirati ključ mape, vrednost karte ili unos karte koristeći metode prikazane u nastavku. Više indeksa se može kreirati na istoj koloni mape u tabeli, tako da se mogu ispitivati ključevi mape, vrednosti ili unosi. Pored toga, zamrznute kolekcije mogu se indeksirati pomoću indeksiranje punog sadržaja zamrznute kolekcije.[10]

Sva upozorenja o korišćenju sekundarnih indeksa odnose se na kolekcije indeksiranja.

- Za kolekcije skupova i lista, treba napraviti indeks na imenu kolone. Napraviti indeks na skupu da bi se pronašli svi biciklisti koji su bili u određenom timu. Primer:

```
CREATE INDEX team_idx ON cycling.cyclist_career_teams ( teams );
```

```
SELECT * FROM cycling.cyclist_career_teams WHERE teams CONTAINS 'Nederland bloeit'; [9]
```

Gde bi rezultat bio recimo kao na slici:

id	lastname	teams
5b6962dd-3f90-4c93-8f61-eabfa4a803e2	VOS	{'Nederland bloeit', 'Rabobank Women Team', 'Rabobank-Liv Giant', 'Rabobank-Liv Woman Cycling Team'}

Slika 9: Rezultat

- Mapa povezuje jednu stavku sa drugom parom ključ-vrednost. Za svaki ključ može postojati samo jedna vrednost, a duplikati se ne mogu čuvati. I ključ i vrednost su označeni istim tipom podataka. Za kolekcije mapa, treba napraviti indeks na ključu mape, vrednosti karte ili unosu karte. Napraviti indeks na ključu mape da bi se pronašle sve kombinacije biciklista/tim za određenu godinu. Primer:

```
CREATE INDEX team_year_idx ON cycling.cyclist_teams ( KEYS (teams) );
```

```
SELECT * From cycling.cyclist_teams WHERE teams CONTAINS KEY 2015;
```

Gde bi rezultat bio kao na slici:

id	firstname	lastname	teams
cb07baad-eac8-4f65-b28a-bddc06a0de23	Elizabeth	ARMITSTEAD	{2011: 'Team Garmin - Cervelo', 2012: 'AA Drank - Leontien.nl', 2013: 'Boels-Dolmans Cycling Team', 2014: 'Boels-Dolmans Cycling Team', 2015: 'Boels-Dolmans Cycling Team'}
5b6962dd-3f90-4c93-8f61-eabfa4a803e2	Marianne	VOS	{2011: 'Nederland bloeit', 2012: 'Rabobank Women Team', 2013: 'Rabobank-Liv Giant', 2014: 'Rabobank-Liv Woman Cycling Team', 2015: 'Rabobank-Liv Woman Cycling Team'}
e7cd5752-bc0d-4157-a80f-7523add8dbcd	Anna	VAN DER BREGGEN	{2009: 'Team Flexpoint', 2012: 'Sengers Ladies Cycling Team', 2013: 'Sengers Ladies Cycling Team', 2014: 'Rabobank-Liv Woman Cycling Team', 2015: 'Rabobank-Liv Woman Cycling Team'}

Slika 10: Rezultat2

- Pravljenje indeksa na usnosima na mapi i nalaženje biciklista koji su istih godina. Korišćenje indeksa ENTRIES važi samo za mape:

```
CREATE TABLE cycling.birthday_list (cyclist_name text PRIMARY KEY, blist map<text,text>);
```

```
CREATE INDEX blist_idx ON cycling.birthday_list (ENTRIES(blist));
```

```
SELECT * FROM cycling.birthday_list WHERE blist['age'] = '23';
```

Rezultat:

cyclist_name	blist
Claudio HEINEN	{'age': '23', 'bday': '27/07/1992', 'nation': 'GERMANY'}
Laurence BOURQUE	{'age': '23', 'bday': '27/07/1992', 'nation': 'CANADA'}

Slika 11: Rezultat3

- Korišćenjem indeksa, pronalaženje biciklista iz iste zemlje:

```
SELECT * FROM cyclist.birthday_list WHERE blist['nation'] = 'NETHERLANDS';
```

cyclist_name	blist
Luc HAGENAARS	{'age': '28', 'bday': '27/07/1987', 'nation': 'NETHERLANDS'}
Toine POELS	{'age': '52', 'bday': '27/07/1963', 'nation': 'NETHERLANDS'}

Slika 12: Rezultat

- Pravljenje indeksa na vrednostima karte i nalaženje biciklista koji imaju određenu vrednost koja se nalazi na navedenoj mapi. Korišćenje indeksa VALUE važi samo za mape. Primer:

```
CREATE TABLE cycling.birthday_list (cyclist_name text PRIMARY KEY, blist
```

```
map<text,text>;
```

```
);CREATE INDEX blist_idx ON cycling.birthday_list (VALUES(blist));
```

```
SELECT * FROM cycling.birthday_list CONTAINS 'NETHERLANDS';
```

```
lorina@cqlsh:cycling> SELECT * FROM cycling.birthday_list WHERE blist CONTAINS 'NETHERLANDS';
```

cyclist_name	blist
Luc HAGENAARS	{'age': '28', 'bday': '27/07/1987', 'nation': 'NETHERLANDS'}
Toine POELS	{'age': '52', 'bday': '27/07/1963', 'nation': 'NETHERLANDS'}

Slika 13: Rezultat

- Pravljenje indeksa na punom sadržaju FROZEN mape. Tabela u ovom primeru čuva broj Pro pobeda, Grand Tour traka i Klasičnih trka u kojima se biciklista takmičio. Naredba SELECT pronalazi svakog biciklistu koji ima 39 pobeda u Pro trkama, 7 startova Grand Toura i 14 klasičnih startova:

```
CREATE TABLE cycling.race_starts (cyclist_name text PRIMARY KEY, rnumbers FROZEN<LIST<int>>);
```

```
CREATE INDEX rnumbers_idx ON cycling.race_starts (FULL(rnumbers));
```

```
SELECT * FROM cycling.race_starts WHERE numbers = [39,7,14];
```

cyclist_name	numbers
John DEGENKOLB	[39, 7, 14]

Slika 14: Rezultat

2.2.5. SASI indeks

SASI indeks, ili skraćeno SASI, je implementacija Cassandrinog Index interfejsa koja se može koristiti kao alternativa postojećim implementacijama. SASI-jevo indeksiranje i upiti poboljšavaju postojeće implementacije tako što ga posebno prilagođavaju Cassandrinim potrebama. SASI ima superiorne performanse u slučajevima kada bi upiti prethodno zahtevali filtriranje. Da bi postigao ove performanse, SASI ima za cilj da bude značajno manje intezivan prema resursima od postojećih implementacija, u upotrebi memorije, diska i CPU-a. Pored toga, SASI podržava prefiks i sadrži upite na stringovima. [1]

Upotreba SASI indeksa biće prikazana na primeru. Prvo je potrebno kreirati keyspace *demo*:

```
cqlsh> CREATE KEYSPACE demo WITH replication = {  
    ... 'class': 'SimpleStrategy',  
    ... 'replication_factor': '1'  
    ... };  
cqlsh> USE demo;
```

Svi primeri biće izvedeni na *sasi* tabeli:

```
cqlsh:demo> CREATE TABLE sasi (id uuid, first_name text, last_name text,  
    ... age int, height int, created_at bigint, primary key (id));
```

Za kreiranje SASI indeksa koristi CREATE CUSTOM INDEX naredbu:

```
cqlsh:demo> CREATE CUSTOM INDEX ON sasi (first_name) USING 'org.apache.cassandra.in  
dex.sasi.SASIIndex'  
    ... WITH OPTIONS = {  
    ... 'analyzer_class':  
    ... 'org.apache.cassandra.index.sasi.analyzer.NonTokenizingAnalyzer',  
    ... 'case_sensitive': 'false'  
    ... };  
  
cqlsh:demo> CREATE CUSTOM INDEX ON sasi (last_name) USING 'org.apache.cassandra.ind  
ex.sasi.SASIIndex'  
    ... WITH OPTIONS = {'mode': 'CONTAINS'};  
  
cqlsh:demo> CREATE CUSTOM INDEX ON sasi (age) USING 'org.apache.cassandra.index.sas  
i.SASIIndex';  
  
cqlsh:demo> CREATE CUSTOM INDEX ON sasi (created_at) USING 'org.apache.cassandra.in  
dex.sasi.SASIIndex'  
    ... WITH OPTIONS = {'mode': 'SPARSE'};
```

Kreirani indeksi imaju određene opcije koje prilagođavaju njihovo ponašanje i potencijalno

performanse. Indeks uključen *first_name* je bez obzira na velika i mala slova. NonTokenizingAnalyzer vrši analizu teksta. Svaki indeks ima režim: PREFIX, CONTAINS ili SPARSE, prvi je podrazumevani. Indeks *last_name* je kreiran u režimu CONTAINS koji odgovara terminima samo na sufiksima umesto na prefiksu. *Created_at* kolona kreirana sa režimom postavljenim na SPARSE, što je namenjeno poboljšanju performansi upita velikih, gustih opsega brojeva kao što su vremenske oznake za podatke koji se ubacuju svake milisekundi. Indeks age je kreiran u podrazumevanom PREFIX režimu i nisu navedene opcije osetljivosti i na velika i mala slova ili nalize teksta pošto je polje numeričko.

Nakon umetanja sledećih podataka i izvođenja nodetool flush, SASI koja vrši ispiranje indeksa na disk može se videti u Cassandrinim evidendcijama-iako direktan poziv za ispiranje nije potreban:

```
cqlsh:demo> INSERT INTO sasi (id, first_name, last_name, age, height, created_at)
... VALUES (556ebd54-cbe5-4b75-9aae-bf2a31a24500, 'Pavel', 'Yaskevich', 27,
181, 1442959315018);

cqlsh:demo> INSERT INTO sasi (id, first_name, last_name, age, height, created_at)
... VALUES (5770382a-c56f-4f3f-b755-450e24d55217, 'Jordan', 'West', 26, 173
, 1442959315019);

cqlsh:demo> INSERT INTO sasi (id, first_name, last_name, age, height, created_at)
... VALUES (96053844-45c3-4f15-b1b7-b02c441d3ee1, 'Mikhail', 'Stepura', 36,
173, 1442959315020);

cqlsh:demo> INSERT INTO sasi (id, first_name, last_name, age, height, created_at)
... VALUES (f5dfcabe-de96-4148-9b80-a1c41ed276b4, 'Michael', 'Kjellman', 26
, 180, 1442959315021);

cqlsh:demo> INSERT INTO sasi (id, first_name, last_name, age, height, created_at)
... VALUES (2970da43-e070-41a8-8bcb-35df7a0e608a, 'Johnny', 'Zhang', 32, 17
5, 1442959315022);

cqlsh:demo> INSERT INTO sasi (id, first_name, last_name, age, height, created_at)
... VALUES (6b757016-631d-4fdb-ac62-40b127ccfbc7, 'Jason', 'Brown', 40, 182
, 1442959315023);

cqlsh:demo> INSERT INTO sasi (id, first_name, last_name, age, height, created_at)
... VALUES (8f909e8a-008e-49dd-8d43-1b0df348ed44, 'Vijay', 'Parthasarathy',
34, 183, 1442959315024);

cqlsh:demo> SELECT first_name, last_name, age, height, created_at FROM sasi;
```

first_name	last_name	age	height	created_at
Michael	Kjellman	26	180	1442959315021
Mikhail	Stepura	36	173	1442959315020
Jason	Brown	40	182	1442959315023
Pavel	Yaskevich	27	181	1442959315018
Vijay	Parthasarathy	34	183	1442959315024
Jordan	West	26	173	1442959315019
Johnny	Zhang	32	175	1442959315022

(7 rows)

SASI podržava sve upite koje CQL već podržava, uključujući nardebe LIKE za pretrage PREFIX,

CONTAINS i SUFFIX:

```
cqlsh:demo> SELECT first_name, last_name, age, height, created_at FROM sasi
... WHERE first_name = 'Pavel';
```

first_name	last_name	age	height	created_at
Pavel	Yaskevich	27	181	1442959315018

(1 rows)

```
cqlsh:demo> SELECT first_name, last_name, age, height, created_at FROM sasi
... WHERE first_name = 'pavel';
```

first_name	last_name	age	height	created_at
Pavel	Yaskevich	27	181	1442959315018

(1 rows)

```
cqlsh:demo> SELECT first_name, last_name, age, height, created_at FROM sasi
... WHERE first_name LIKE 'M%';
```

first_name	last_name	age	height	created_at
Michael	Kjellman	26	180	1442959315021
Mikhail	Stepura	36	173	1442959315020

(2 rows)

Naravno, slučaj upita nije bitan za first_name kolonu zbog opcija koje se pružaju u vreme kreiranja indeksa.

```
cqlsh:demo> SELECT first_name, last_name, age, height, created_at FROM sasi
... WHERE first_name LIKE 'm%';
```

first_name	last_name	age	height	created_at
Michael	Kjellman	26	180	1442959315021
Mikhail	Stepura	36	173	1442959315020

(2 rows)

- **Složeni upiti**

SASI podržava upite sa više predikata, međutim, zbog prirode implementacije podrazumevanog indeksiranja, CQL zahteva od korisnika da navede ALLOW FILTERING da se uključi u potencijalne zamke performansi takvog upita. Sa SASI-jem, iako ostaje zahtev za uključivanje ALLOW FILTERING, da bi se smanjile modifikacije gramatike, zamke performansi ne postoje jer se filtriranje ne vrši.

```
cqlsh:demo> SELECT first_name, last_name, age, height, created_at FROM sasi
... WHERE first_name LIKE 'M%' and age < 30 ALLOW FILTERING;
```

first_name	last_name	age	height	created_at
------------	-----------	-----	--------	------------

Michael	Kjellman	26	180	1442959315021
---------	----------	----	-----	---------------

(1 rows)

- **Sufiksni upiti**

Sledeći primer prikazuje CONTAINS režim na last_name koloni. Korišćenjem ovog režima, predikati mogu da traže bilo koji niz koji sadrži string za pretragu kao podniz. U ovom slučaju nizovi koji sadrže a' ' or an''.

```
cqlsh:demo> SELECT * FROM sasi WHERE last_name LIKE '%a%';
```

id	age	created_at	first_name	height
last_name				
f5dfcabe-de96-4148-9b80-a1c41ed276b4	26	1442959315021	Michael	180
Kjellman				
96053844-45c3-4f15-b1b7-b02c441d3ee1	36	1442959315020	Mikhail	173
Stepura				
556ebd54-cbe5-4b75-9aae-bf2a31a24500	27	1442959315018	Pavel	181
Yaskevich				
8f909e8a-008e-49dd-8d43-1b0df348ed44	34	1442959315024	Vijay	183
Parthasarathy				
2970da43-e070-41a8-8bcb-35df7a0e608a	32	1442959315022	Johnny	175
Zhang				

(5 rows)

```
cqlsh:demo> SELECT * FROM sasi WHERE last_name LIKE '%an%';
```

id	age	created_at	first_name	height
last_name				
f5dfcabe-de96-4148-9b80-a1c41ed276b4	26	1442959315021	Michael	180
Kjellman				
2970da43-e070-41a8-8bcb-35df7a0e608a	32	1442959315022	Johnny	175
Zhang				

(2 rows)

- **Izrazi na neindeksiranim kolonama**

SASI takođe podržava filtriranje neindeksiranih kolona kao što je height. Izraz može samo suziti postojeći upit koristeći AND.

```
cqlsh:demo> SELECT * FROM sasi WHERE last_name LIKE '%a%' AND height >= 175 ALLOW FILTERING;
```

id	age	created_at	first_name	height
last_name				
f5dfcabe-de96-4148-9b80-a1c41ed276b4	26	1442959315021	Michael	180
Kjellman				
2970da43-e070-41a8-8bcb-35df7a0e608a	32	1442959315022	Johnny	175
Zhang				

f5dfcabe-de96-4148-9b80-a1c41ed276b4 Kjellman	26	1442959315021	Michael	180
556ebd54-cbe5-4b75-9aae-bf2a31a24500 Yaskevich	27	1442959315018	Pavel	181
8f909e8a-008e-49dd-8d43-1b0df348ed44 Parthasarathy	34	1442959315024	Vijay	183
2970da43-e070-41a8-8bcb-35df7a0e608a Zhang	32	1442959315022	Johnny	175
(4 rows)				

- **Analiza tokenizacije zasnovana na razgraničavanju**

Jednostavna analiza teksta je tokenizacija zasnovana na razgraničenju. Ovo pruža alternativu indeksiranju kolekcija, pošto se tekst razdvojen graničnicima može indeksirati bez dodatnih troškova CONTAINS režima niti upotrebe PREFIX ili SUFIX upita.

```
cqlsh:demo> ALTER TABLE sasi ADD aliases text;
cqlsh:demo> CREATE CUSTOM INDEX ON sasi (aliases) USING 'org.apache.cassandra.index.sasi.SASIIndex'
... WITH OPTIONS = {
... 'analyzer_class': 'org.apache.cassandra.index.sasi.analyzer.DelimiterAnalyzer',
... 'delimiter': ',',
... 'mode': 'prefix',
... 'analyzed': 'true'};
cqlsh:demo> UPDATE sasi SET aliases = 'Mike,Mick,Mikey,Mickey' WHERE id = f5dfcabe-de96-4148-9b80-a1c41ed276b4;
cqlsh:demo> SELECT * FROM sasi WHERE aliases LIKE 'Mikey' ALLOW FILTERING;
```

id	age	aliases	created_at
first_name height last_name			
f5dfcabe-de96-4148-9b80-a1c41ed276b4 1 Michael 180 Kjellman	26	Mike,Mick,Mikey,Mickey	1442959315021

- **Analiza teksta (tokenizacija i stiming)**

Na kraju, da bi se demonstrirala analiza teksta potrebna je dodatna kolona u tabeli. Njegova definicija, indeks i izjave za ažuriranje redova prikazni su ispod:

```
cqlsh:demo> ALTER TABLE sasi ADD bio text;
cqlsh:demo> CREATE CUSTOM INDEX ON sasi (bio) USING 'org.apache.cassandra.index.sasi.SASIIndex'
... WITH OPTIONS = {
... 'analyzer_class': 'org.apache.cassandra.index.sasi.analyzer.StandardAnalyzer',
... 'tokenization_enable_stemming': 'true',
... 'analyzed': 'true',
... 'tokenization_normalize_lowercase': 'true',
... 'tokenization_locale': 'en'
... };
cqlsh:demo> UPDATE sasi SET bio = 'Software Engineer, who likes distributed systems , doesnt like to argue.' WHERE id = 5770382a-c56f-4f3f-b755-450e24d55217;
```

```
cqlsh:demo> UPDATE sasi SET bio = 'Software Engineer, works on the freight distribu
tion at nights and likes arguing' WHERE id = 556ebd54-cbe5-4b75-9aae-bf2a31a24500;
cqlsh:demo> SELECT * FROM sasi;
```

id	created_at	first_name	height	age	bio	last_name
f5dfcabe-de96-4148-9b80-a1c41ed276b4			26			
null	1442959315021	Michael	180			Kjellman
96053844-45c3-4f15-b1b7-b02c441d3ee1			36			
null	1442959315020	Mikhail	173			Stepura
6b757016-631d-4fdb-ac62-40b127ccfbc7			40			
null	1442959315023	Jason	182			Brown
556ebd54-cbe5-4b75-9aae-bf2a31a24500			27		Software Engineer, works on the freight distribution at nights and likes arguing	
1442959315018		Pavel	181			Yaskevich
8f909e8a-008e-49dd-8d43-1b0df348ed44			34			
null	1442959315024	Vijay	183			Parthasarathy
5770382a-c56f-4f3f-b755-450e24d55217			26		Software Engineer, who likes distributed systems, doesnt like to argue.	
1442959315019		Jordan	173			West
2970da43-e070-41a8-8bcb-35df7a0e608a			32			
null	1442959315022	Johnny	175			Zhang

(7 rows)

Indeksni termini i stringovi za pretragu upita su poveazni sa bio kolonom jer je konfigurisana da koristi StandardAnalyzer i analyzed postavljena je na true. Tokenization_normalize_lowercase je slično case_sensitive svojstvu, ali za StandardAnalyzer. Ovi upiti demonstriraju stiming koji primenjuje StandardAnalyzer.

```
cqlsh:demo> SELECT * FROM sasi WHERE bio LIKE 'distributing';
```

id	created_at	first_name	height	age	bio	last_name
556ebd54-cbe5-4b75-9aae-bf2a31a24500			27		Software Engineer, works on the freight distribution at nights and likes arguing	
1442959315018		Pavel	181			Yaskevich
5770382a-c56f-4f3f-b755-450e24d55217			26		Software Engineer, who likes distributed systems, doesnt like to argue.	
1442959315019		Jordan	173			West

(2 rows)

```
cqlsh:demo> SELECT * FROM sasi WHERE bio LIKE 'they argued';
```

id	created_at	first_name	height	age	bio	last_name
----	------------	------------	--------	-----	-----	-----------

```

-----+-----+-----+-----+
-----+-----+-----+-----+
-----
556ebd54-cbe5-4b75-9aae-bf2a31a24500 | 27 | Software Engineer, works on the freig
ht distribution at nights and likes arguing | 1442959315018 | Pavel | 181 |
Yaskevich
5770382a-c56f-4f3f-b755-450e24d55217 | 26 | Software Engineer, who likes
distributed systems, doesnt like to argue. | 1442959315019 | Jordan | 173 |
West

(2 rows)

cqlsh:demo> SELECT * FROM sasi WHERE bio LIKE 'working at the company';

id | age | bio
| created_at | first_name | height | last_name
-----+-----+-----+-----+
-----
556ebd54-cbe5-4b75-9aae-bf2a31a24500 | 27 | Software Engineer, works on the freig
ht distribution at nights and likes arguing | 1442959315018 | Pavel | 181 |
Yaskevich

(1 rows)

cqlsh:demo> SELECT * FROM sasi WHERE bio LIKE 'soft eng';

id | age | bio
| created_at | first_name | height | last_name
-----+-----+-----+-----+
-----
556ebd54-cbe5-4b75-9aae-bf2a31a24500 | 27 | Software Engineer, works on the freig
ht distribution at nights and likes arguing | 1442959315018 | Pavel | 181 |
Yaskevich
5770382a-c56f-4f3f-b755-450e24d55217 | 26 | Software Engineer, who likes
distributed systems, doesnt like to argue. | 1442959315019 | Jordan | 173 |
West

(2 rows)

```

- **Detalji implementacije**

SASI je jednostavno implementacija Index interfejsa, u svojoj srži postoji nekoliko struktura podataka i algoritama koji se koriste da ga zadovolje. Interfejs Index deli odgovornost implementatora na dva dela: indeksiranje i postavljanje upita. Cassandra omogućava da se te odgovornosti podele na komponente memorije i diska. SASI koristi prednost Casandrinog nepromenljivog, uređenog modela podataka za jednokratno upisivanje za izgradnju indeksa zajedno sa ispiranjem memtable na disk.

Strukture podataka SASI indeksa se ugrađuju u memoriju dok se SSTable upisuje i ispuštaju se na disk pre nego što se upisivanje SSTable završi. Za pisanje svake indeksne datoteke potrebno je samo uzastopno upisivanje na disk. U nekim slučajevima se izvode delimične ispune, a kasnije se ponovo spajaju, kako bi se smanjila upotreba memorije. Ove strukture podataka su optimizovane

za ovaj slučaj upotrebe.

Koristeći prednost Cassandrinog uređenog modela podataka, u vreme upita, indeksi kandidata se sužavaju za pretragu, minimizirajući količinu obavljenog posla. Pretraživanje se zatim vrši korišćenjem efikasnog metoda koji prenosi podatke sa diska po potrebi.

Indeksiranje

Prema SSTable-u, SASI upisuje indeksnu datoteku za svaku indeksiranu kolonu. Podaci za ove datoteke se ugrađuju u memoriju pomoću OnDiskIndexBuilder. Kada se prenesu na disk, podaci se čitaju pomoću OnDiskIndex klase. Oni se sastoje od bajtova koji predstavljaju indeksirane termine, organizovane za efikasno pisanje ili pretragu. Ključevi i vrednosti koje drže predstavljaju tokene i pozicije u SSTable-u i oni se čuvaju po indeksiranim terminima u TokenTreeBuilders za pisanje i TokenTrees za postavljanje upita. Ove indeksne datoteke se mapiraju u memoriju nakon što su zapisane na disk, radi bržeg pristupa. Za indeksiranje podataka u memtable, SASI koristi svoju IndexMemtable

OnDiskIndex(Builder)

Svaka OnDiskIndex je instanca modifikovane strukture podataka niza sufiksa . Sastoji se OnDiskIndex od blokova veličine stranice sortiranih termina i pokazivača na podatke povezane sa terminima, kao i samih podataka, takođe uskladištenih u jednom ili više blokova veličine stranice. Strukturiran OnDiskIndex je kao stablo nizova, gde svaki nivo opisuje termine u donjem nivou, a poslednji nivo su sami termini. Konačni nivo i njegov s.PointerLevel's and their 'PointerBlock's contain terms and pointers to other blocks that end with those terms. The 'DataLevelDataBlock's contain terms and point to the data itself, contained in 'TokenTree

Termini upisani u OnDiskIndex variraju u zavisnosti od njegovog režima" može se konfigurisati po koloni u vreme kreiranja indeksa.mode": either PREFIX, CONTAINS, or SPARSE. In the PREFIX and SPARSE cases, terms' exact values are written exactly once per OnDiskIndex. For example, when using a PREFIX index with terms Jason, Jordan, Pavel, all three will be included in the index. A CONTAINS index writes additional terms for each suffix of each term recursively. Continuing with the example, a CONTAINS index storing the previous terms would also store ason, ordan, avel, son, rdan, vel, etc. This allows for queries on the suffix of strings. The SPARSE mode differs from PREFIX in that for every 64 blocks of terms a TokenTree is built merging all the TokenTrees for each term into a single one. This copy of the data is used for efficient iteration of large ranges of e.g. timestamps. The index

TokenTree(Builder)

Ovo TokenTree je implementacija dobro poznatog B+-stabla koje je modifikovano da bi se optimizovalo za svoj slučaj upotrebe. Konkretno, optimizovano je da poveže tokene, duge, sa skupom pozicija u SSTable, takođe duge. Dozvoljavanjem skupa dugih vrednosti prihvata se mogućnost heš kolizije u tokenu, ali struktura podataka je optimizovana za malo verovatnu mogućnost takve kolizije.

Da bi se optimizovao za svoje okruženje za jednokratno upisivanje, TokenTreeBuilder potpuno učitava svoje unutrašnje čvorove dok se drvo gradi i koristi dobro poznati algoritam optimizovan za masovno učitavanje strukture podataka.

TokenTrees obezbeđuju sredstva za iteraciju preko tokena i pozicija datoteke, koji se podudaraju sa datim terminom, i za preskakanje unapred u toj iteraciji, operaciji koja se uveliko koristi u vreme upita.

IndexMemtable

IndexMemtable Ručke koje indeksiraju podatke u memoriji koji se nalaze u memtable . IndexMemtable Zauzvrat upravlja ili a ili TrieMemIndex po SkipListMemIndex koloni. Izbor tipa indeksa koji se koristi zavisi od podataka. TrieMemIndex Koristi se za literalne tipove . AsciiTypei UTF8Typepodrazumevano su literalni tipovi, ali bilo koja kolona može da se konfiguriše kao literalni tip korišćenjem is_literalopcije u vreme kreiranja indeksa. Za neliteralne tipove SkipListMemIndex se koristi. Ovo TrieMemIndex je implementacija koja može efikasno da podrži upite sa prefiksom na podacima sličnim znakovima. , SkipListMemIndexnaprotiv, je pogodniji za druge tipove podataka Cassandra kao što su brojevi.

Napravljen TrieMemIndex je pomoću paketa ConcurrentRadixTreeili ConcurrentSuffixTreeiz com.googlecode.concurrenttrees paketa. Izbor između ova dva se vrši na osnovu režima indeksiranja, PREFIXodnosno drugih režima, odnosno CONTAINSrežima.

Izgrađen SkipListMemIndex je na vrhu java.util.concurrent.ConcurrentSkipListSet

- **Klasa SASIIndex**

SASIIndex klasa implementira Index i instancira se po tabeli koja sadrži SASI indekse. Upravlja svim indeksima za tabelu preko *sasi.conf.DataTracker* i *sasi.conf.view.View* komponenti, kontroliše pisanje svih indeksa za SSTable preko svog *PerSSTableIndexWriter* i pokreće pretrage sa *Searcher*. [1]

2.2.6. Indeksiranje vezano za skladištenje (SAI)

Indeksiranje vezano za skladište je visoko sklabilan mehanizam indeksiranja podataka dostupan za baze podataka DataStek Astra i DataStek Enterprice. Mogu se definisati jedan ili više SAI indeksa u bilo kojoj koloni, a zatim koristiti upiti opsega (samo numeričke), sematiku CONTAIN i upite za filtriranje.

SAI skladišti pojedinačne indeksirane datoteke za svaku kolonu i sadrži pokazivač na pomak izvornih podataka u SSTable-u. Jednom kada se ubace podaci u ineksiranu kolonu, oni će prvo biti upisani u memoriju. Kad god Cassandra izbaci podatka iz memorije na disk, ona upisuje indeks zajedno sa tabelom podataka.

Ovaj pristup poboljšava propusnost za 43% i kašnjenje za 230%. U poređenju sa SASI koristi znatno manje prostora na disku za indeksiranje, ima manje tačaka kvara i dolazi sa pojednostavljenom arhitekturom.

Primer deifnisanja ideksa korišćenjem SAI-a:

```
CREATE CUSTOM INDEX ON company (employee_age) USING 'StorageAttachedIndex' WITH OPTIONS =  
{'case_sensitive': false, 'normalize': false};
```

Opcija normalizacije transformiše specijalne znakove u njihov osnovni karakter. Na primer, može se normalizovati nemački znak o u obično o, omogućavajući podudaranje upita bez kucanja specijalnih znakova. [13]

2.2.7. Problemi sa korišćenjem indeksa

- **Problemi sa korišćenjem indeksa kolone visoke kardinalnosti**

Ukoliko se kreira indeks na koloni visoke kardinalnosti, koji ima mnogo različitih vrednosti, upit između polja će izazvati mnogo traženja za vrlo malo rezultata. U tabeli sa milijardu pesama, traženje pesama po piscu (vrednost koja je tipično jedinstvena za svaku pesmu) umesto po izvođaču verovatno će biti veoma neefikasno. Verovatno bi bilo efikasnije ručno održavati tabelu kao oblik indeksa umesto da se koristi ugrađeni indeks Cassandra. Za kolone koje sadrže jedinstvene podatke, ponekad je dobro u pogledu performansi koristiti indeks radi pogodnosti, sve dok je obim upita za tabelu koja ima indeksiranu kolonu umeren i nije pod konstantnim opterećenjem.

Nasuprot tome, kreiranje indeksa na koloni izuzetno niske kardinalnosti, kao što je logička kolona, nema smisla. Svaka vrednost u indeksu postoji jedan red u indeksu, što rezultira ogromnim redom za sve lažne vrednosti, na primer. Indeksiranje mnoštva indeksiranih kolona koje imaju `foo==true` i `foo==false` nije korisno.

- **Problemi sa korišćenjem indeksa u koloni koja se često ažurira ili briše**

Cassandra čuva nadgrobnih spomenika u indeksu sve dok granica nadgrobnih spomenika ne dostigne 100.000 ćelija. Nakon prekoračenja ograničenja nadgrobnih spomenika, upit koji koristi indeksiranu vrednost neće uspeti.

- **Problemi sa korišćenjem indeksa za traženje reda u velikoj particiji osim ako se ne pitaju usko**

Upit za indeksiranu kolonu u velikom klasteru obično zahteva sređivanje odgovora sa više particija podataka. Odgovor na upit se usporava kako se više mašina dodaje u klaster. Može se izbeći smanjenje performansi kada se traži red u velikoj particiji sužavanjem pretrage. [11]

2.2.8. Primeri primene

Postoji tabela za unos utakmica u kriket sa milion unosa za igrače u stotinama mečeva i potrebno je pretražiti rang igrača prema broju odigranih utakmica. Mnogi rangovi igrača će deliti istu vrednost kolone za godinu utakmice. Kolona `match_year` je dobra opcija za indeks:

Ispod na slici je data sintaksa za kreiranje indeksa:

```
CREATE INDEX [ IF NOT EXISTS ] index_name
ON [keyspace_name.]table_name
([ ( KEYS | FULL ) ] column_name)
(ENTRIES column_name);
```

Slika 15: Sintaksa za kreiranje indeksa

Za kreiranje tabele koristi se keyspace1 kao prostor ključeva i Task kao ime tabele. Na slici 2 je dat prikaz kreiranja tabele sa odgovarajućim atributima i tipovima podataka:

```
CREATE TABLE keyspace1.Task
(
    Task_id text,
    Task_name text,
    Task_time timestamp,
    T_location text,
    PRIMARY KEY (Task_id, Task_name)
);
```

Slika 16: Kreiranje tabele

Pošto je Cassandra distribuirana i decentralizovana baza podataka sa podacima organizovanim po ključu particije, u opštem slučaju, upiti klauzule WHERE moraju uključiti particioni ključ. Na slici 3 dat je primer jedne SELECT naredbe sa određenim where uslovom:

```
SELECT *
FROM Task
WHERE Task_id = 'T210' AND Task_name; 'set alarm';
```

Slika 17: Primer select naredbe

Kao što se vidi sa slike broj 5 Task_id i Task_name su delovi primarnog ključa i upit sa prethodne slike bi dobro funkcionisao. Ukoliko bi se na primer napisao upit kao na slici, takav ne bi dobro funkcionisao, jer kao što se i vidi Task_time nije deo primarnog ključa.

```
SELECT * FROM Task WHERE Task_time= '2019-09-30 15:02:56';
```

Slika 18: Primer upita

Da bi se rešila ovakva vrsta grešaka, mogu se kreirati indeksi na koloni za grupisanje. Potrebno je definisati tabelu koja ima kompozitni particioni ključ, a zatim da se kreira indeks na koloni za grupisanje. Na slici se vidi primer kreiranja indeksa i novog keyspace-a.


```
CREATE TABLE keyspace1.Task (
  Task_id text,
  Task_name text,
  Task_time timestamp,
  T_location text,
  PRIMARY KEY ((Task_id, Task_name), Task_time)
);

CREATE INDEX ON keyspace1.Task(Task_time);
```

Slika 19: Kreiranje tabele i indeksa

Sada bi prethodni upit, sa slike *** bio uspešno izvršen i vraćao bi dobre rezultate.

Potrebno je znati, da kreiranje sekundarnih indeksa ne znači da će biti povećana brzina upita u Cassandri. Jedna od važnih prednosti sekundarnih indeksa pomaže pristupu podacima koji jednostavno mogu učiniti tako da se klauzula where koje upućuju na vrednosti u koloni izvan primarnih i kolona za grupisanje mogu pokrenuti.

Povećanje brzine upita u Cassandri može se postići kreiranjem tabele posebno za upit. Primer je dat na slici:

```
CREATE TABLE Student_record
(
  Stu_state text,
  Stu_zip text,
  Stu_address text,
  PRIMARY KEY(Stu_state, Stu_zip)
);
```

Slika 20: Kreiranje

U ovoj tabeli Stu_state i Stu_zip mogu biti isti, tako da, da bi se definisao jedinstveni zapis u tabeli, može se dodati Stu_id kao primarni ključ kako bi jedinstveno definisao zapis. To se može postići vršenjem izmena u postojećoj tabeli korišćenjem komande ALTER u CQL-u na sledeći način (slika):

```
ALTER TABLE Student_record ADD Stu_id int PRIMARY KEY;
```

Slika 21: Naredba alter

Nakon izvršenja ove naredbe, izgled tabele biće kao na slici:

column 1	column 2	column 3	column 4
Stu_state	Stu_zip	Stu_address	Stu_id

Slika 22: Evidencija učenika

Kako bi se proverila ova kreirana tabela u Cassandra, može se napisati sledeći upit(slika):

```
SELECT *  
FROM Student_record  
WHERE Stu_id = '107';
```

Slika 23:

A primer izlaza se vidi na slici:

	column 1	column 2	column 3	column 4
ROW 1	Stu_state	Stu_zip	Stu_address	Stu_id
	UP	12345	noida	107

Slika 24: Izlaz [12]

3. Praktična primena

U ovom poglavlju redom biće opisan način pokretanja Cassandra baze podataka i alata koji se koristi za lakše upravljanje baze. Nakon toga biće pomenut skup podataka koji se koristi, dati upiti za kreiranje, indeksi koji su kreirani i mogu se koristiti.

3.1. Pokretanje Cassandra baze podataka

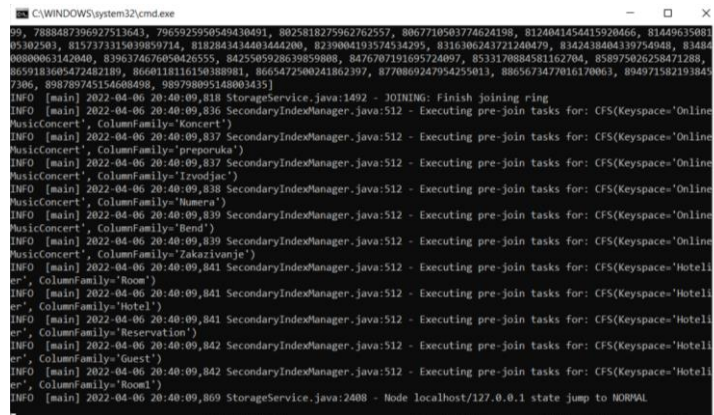
Da bi Cassandra bila pokrenuta na računaru, neophodno je njeno preuzimanje sa: https://cassandra.apache.org/_/download.html. Trenutno dostupna najnovija verzija je 4.3.0. U ovom radu je korišćena verzija 3.11.9. Nakon uspešno preuzimanja i raspakivanja, neophodno je podesiti promenljive okruženja na svom računaru. To su (slika):

Korisničke promenljive za Zeljko

Promenljiva	Vrednost
CASSANDRA_HOME	D:\apache-cassandra-3.11.9-bin\apache-cassandra-3.11.9
COMPOSE_CONVERT_WIN...	true
HADOOP_HOME	C:\Hadoop
JAVA_HOME	C:\Program Files\Java\jre1.8.0_241
MOZ_PLUGIN_PATH	C:\Program Files (x86)\Foxit Software\Foxit Reader\plugins\

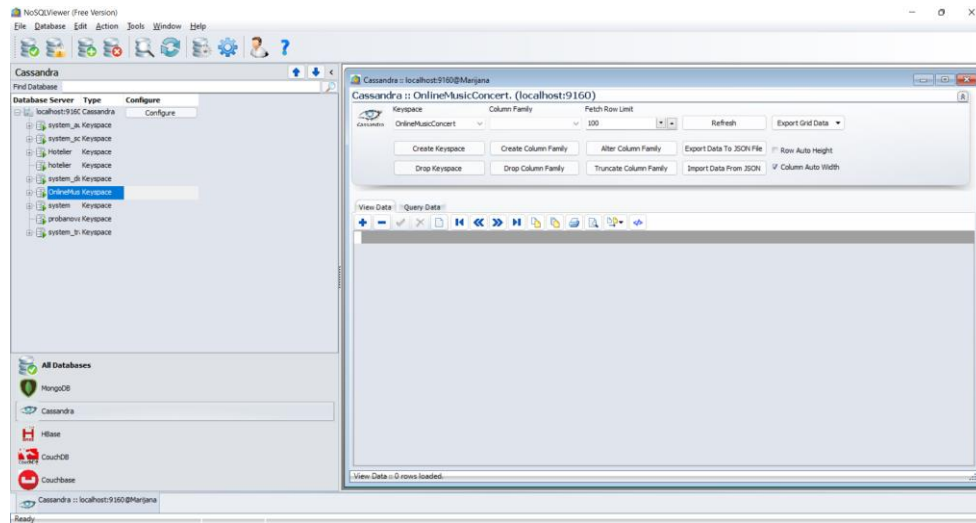
Slika 25: Promenljive okruženja

Nakon završenog podešavanja, može se Cassandra se može pokrenuti klikom na .bat fajl koji se nalazi u folderu preuzete instalacije na putanji ..\apache-cassandra-3.11.9-bin\apache-cassandra-3.11.9\bin. Kao rezultat uspešnog pokretanja biće prikaz kao na slici:



Slika 26: Cassandra

Postoje alati za upravljanje NoSQL bazom podataka, koji pomažu da se poboljša produktivnost. Svaki od alata ima korisnički interfejs, koji pruža korisničko iskustvo u razvoju. Neki od njih su: Compass, NoSQL Manager, NoSQL Booster, Robo Mongo, QueryAssist,... Za potrebe izrade ovog rada korišćen je NoSQL Viewer. To je besplatan inovativan i moćan softverski proizvod za popularne NoSQL baze podataka, kao što su Apache Cassandra, MongoDB, Hbase i druge. Izgled ovog okruženja pri pokretanju se vidi na slici:



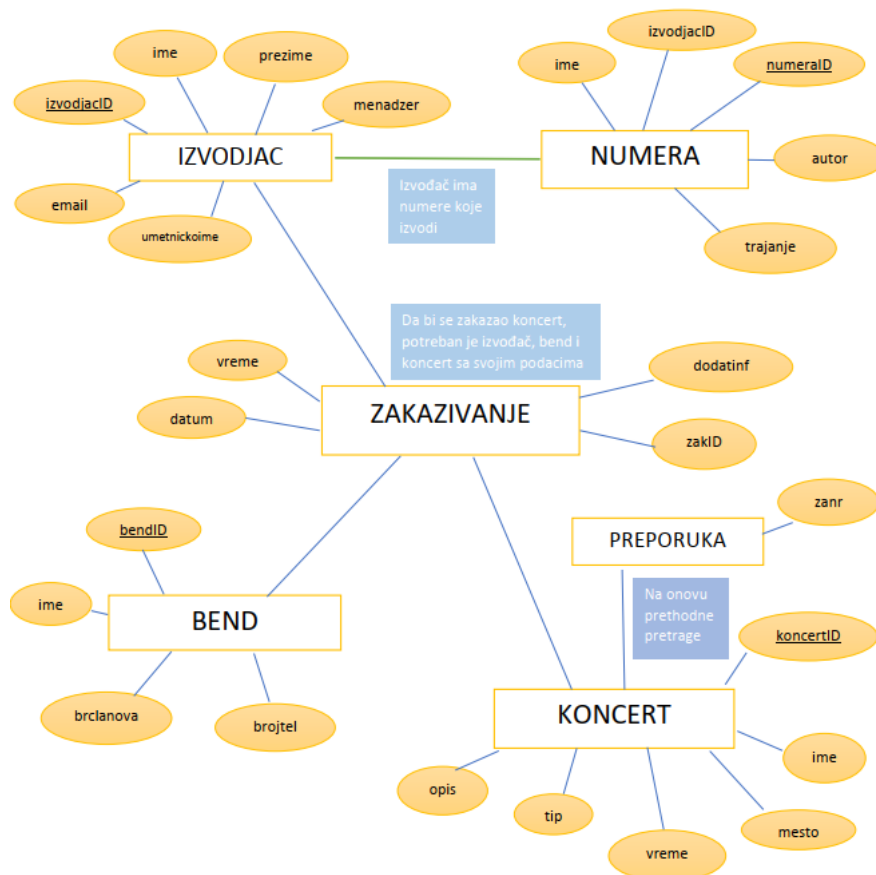
Slika 27: NoSQL Viewer

Sa leve strane prozora prikazana je baza podataka na koju smo povezani, u ovom slučaju je to Cassandra, zatim svi keyspace-ovi koji su već kreirani sa svojim column family-jama i njihovim nazivima. U manjem prozoru, sa desne strane, vidi se prostor u kome je moguće napisati neki od upita i izvršiti ga ukoliko je ispravan.

3.2. Skup podataka

Skup podataka odnosi se na zakazivanje online koncerata, gde se nalaze podaci o koncertima, izvođačima, bendovima, numerama koje izvođači izvode i zakazivanju. Na slici je prikazan dijagram koji prikazuje sve entitete sa atributima, gde su naznačeni atributi koji

predstavljaju primarne ključeve.



Slika 28: Dijagram

3.3. Unos i upotreba podataka

- Prvi korak je kreiranje keyspace. To se čini na sledeći način:

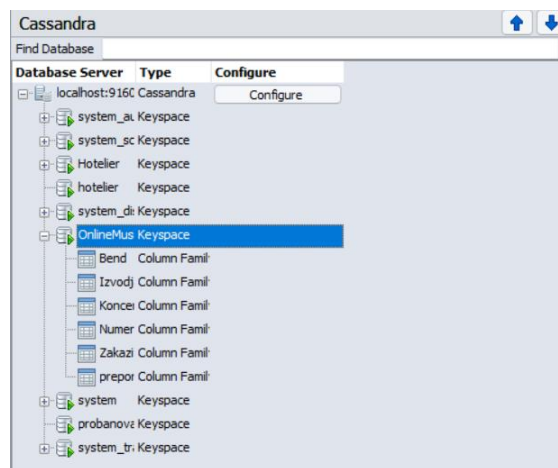
```
create keyspace OnlineMusicConcert;
```

- Kada je keyspace uspešno kreiran, mogu se kreirati column family unutar njega. Neophodno je kreirati Koncert, Izvodjac, Bend, Numera, Zakazi, preporuka. I to na sledeći način:

```
CREATE TABLE "Koncert" (
  "koncertID" text,
  ime text,
  opis text,
  organizator text,
  sponzor text,
  tip text,
  PRIMARY KEY ("koncertID")
)
```

```
CREATE TABLE "Zakazivanje" (
    "zakID" text,
    "izvodjacID" text,
    "bendID" text,
    "koncertID" text,
    vreme text,
    datum text,
    dodatinf text,
    PRIMARY KEY("zakID")
)
```

Na isti način kreiraju se i ostale tabele. Kada su sve uspešno kreiranje u okviru keyspace-a OnlineMusicConcert u NoSQL Viewer-u može se sa leve strane videti struktura kao na slici:



Slika 29: OnlineMusicConcert keyspace

- Kada su uspešno kreirane tabele, mogu se dodati podaci. Ispod se nalaze primeri nekih od njih:

```
insert into "Koncert" ("koncertID",ime,opis,organizator,sponzor,tip) values
('122','Moderna','Dobrodolsi','Grad Nis','Bmw','20-te');
```

```
insert into "Koncert" ("koncertID",ime,opis,organizator,sponzor,tip) values ('133','UvekZabava','Dobar
provod.','Firma Anoo','Stark','Pop');
```

```
insert into "Koncert" ("koncertID",ime,opis,organizator,sponzor,tip) values ('144','Nikad bolje','Dodjite da
se zabavimo','Srednjoskolci','Restoran In','Razno');
```

- Naredbom `select * from "Koncert";` želimo da prikazemo sve podatke koji su uneti. Na slici se vidi rezultat ovog upita:

koncertID (text)	ime (text)	opis (text)	organizator (text)	sponzor (text)	tip (text)
144	Nikad bolje	Dodajte da se	Srednjoskolski	Restoran In	Razno
133	UvekZabava	Dobar	Firma Anoo	Stark	Pop
122	Moderna	Dobrodola	Grad Nis	Bmw	20-te

Slika 30: Rezultat upita

Na potpuno isti način vrši se dodavanje podataka u ostalim tabelama. Primeri takvih upita nalaze se u pratećem dokumentu, koji se dostavlja uz ovaj rad, pod nazivom 'Create and insert.docx'.

- Kada su dodate sve tabele i podaci za svaku od njih, kako bi se podaci pretraživali i po kolonama koje nisu deo primarnog ključa, kreiraćemo indekse. Primer jednog od njih je sledeći:

```
CREATE INDEX ON "Koncert"(tip);
```

To znači da je kreiran indeks u tabeli 'Koncert' nad kolonom tip. Kada je uspešno kreiran, moguće je pretraživati neophodne podatke nad tom kolonom. Recimo potrebno su nam koncerti čiji je tip 'Pop' i to se može naći na sledeći način:

```
select * from "Koncert" where tip='Pop';
```

Rezultat ovog upita je sledeći (slika 31):

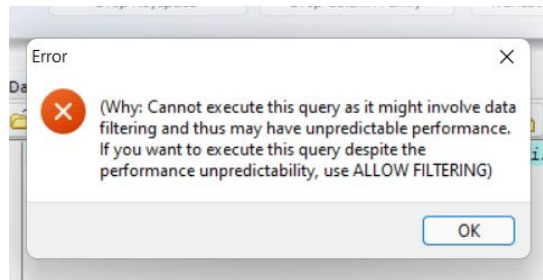
koncertID (text)	ime (text)	opis (text)	organizator (text)	sponzor (text)	tip (text)
133	UvekZabava	Dobar	Firma Anoo	Stark	Pop

Slika 31: Rezultat upita

Ukoliko upit izgleda ovako:

```
select * from "Koncert" where sponzor='Grad Nis';
```

Nastaće problem. Nije moguće pretražiti po koloni 'sponzor' i rezultat ovog upita biće greška (slika):



Slika 32: Greška

Greška je iz razloga što nije kreiran indeks nad tom kolonom i nije moguće pretraživati podatke na taj način. Kada se kreira indeks i nad kolonom sponzor naredbom : **CREATE INDEX ON "Koncert"(sponzor)**; moguće je izvršiti sledeći upit, koji predstavlja korišćenje indeksa nad više kolona:

Primer korišćenja indeksa nad kolonama u ovom skupu podataka je:

```
SELECT * FROM "Koncert" WHERE tip = 'Pop' AND sponzor = 'Stark' ALLOW FILTERING;
```

- **Kreiranje indeksa na skupu ili kolekciji liste:**

U već postojećoj tabeli dodaćemo dodatnu kolonu sa nazivom *phones* koja predstavlja kolekciju telefonskih brojeva kako bi se indeksirali podaci u telefonskom skupu i to na sledeći način:

```
ALTER TABLE "Koncert" ADD phones set<text>;
CREATE INDEX ON "Koncert" (phones);
```

Nakon uspešnog kreiranja indeksa nad kolonom phones, mogu se dodati podaci kao:

```
insert into "Koncert" ("koncertID",ime,opis,organizator,phones,sponzor,tip) values ('144','Nikad bolje','Dodajte da se zabavimo','Srednjoskolci',{'123455','148855','125452'},'Restoran In','Razno');
```

Izvršenjem sledećeg upita:

```
SELECT * FROM "Koncert" WHERE phones CONTAINS '12312';
```

Nalaze se svi koncerti kod kojih je kao kontakt telefon naveden broj '12312' (slika 33):

koncertID (text)	ime (text)	opis (text)	organizator (text)	phones (text)	sponzor (text)	tip (text)
144	Nikad bolje	Dodajte	Srednjoskola	<input type="checkbox"/>	Restoran In	Razno
164	Nikad nije	Dodajte da se zabavimo	Skola	<input type="checkbox"/>	Restoran In	Pop

Slika 33: Rezultat

- Kao što je i navedeno u tekstu iznad, indeksiranje mapa je nešto drugačije, te iz tog razloga, radi poređenja krećemo tabelu 'Dogadjaj' koja će imati tip podataka map koji će predstavljati par ključ-vrednost koji se odnosi na godina-opis i to se kreira sledećim upitom:

```
CREATE TABLE "Dogadjaj"( id text PRIMARY KEY, ime text, opis text, održavanja  
map<int,text> );
```

Nakon kreiranja, unecemo podatka u novu tabelu:

```
INSERT INTO "Dogadjaj" (id,ime,opis,odrzavanja) VALUES ('111','Club8','Klub  
zabave',{2011:'Srednjoskolci',2013:'Deca ispod 18 godina',2014:'Maturanti'});
```

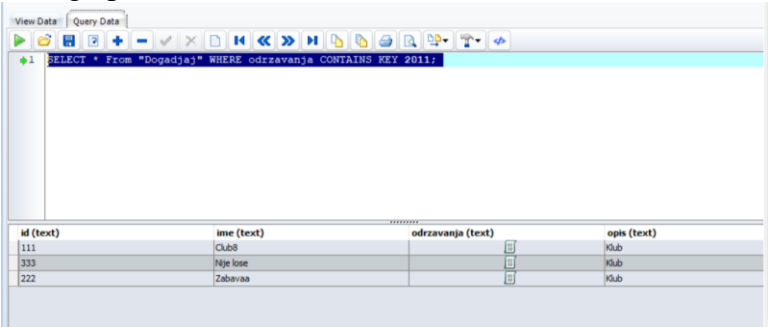
Kreiranje indeksa nad kolonom koja predstavlja kolekciju mapa održavanja nekog događaja:

```
CREATE INDEX doginx ON "Dogadjaj" ( KEYS (odrzavanja) );
```

Upit kojim se pretražuju događaji koji su održani 2011. godine vrši se na sledeći način:

```
SELECT * From "Dogadjaj" WHERE održavanja CONTAINS KEY 2011;
```

Rezultate prethodnog upita vidimo na slici 34:



id (text)	ime (text)	održavanja (text)	opis (text)
111	Club8		Klub
333	Nije lose		Klub
222	Zabavaa		Klub

Slika 34: Rezultat upita

Korišćenje ovih upita može naći primenu u aplikaciji koja bi mogla biti kreirana prema potrebama korisnika i na taj način bi, u zavisnosti od zahteva, mogle da se koriste indeksne strukture Cassandra baze podataka i izvlače neophodni podaci.

4. Zaključak

U radu je detaljno opisana Apache Cassandra baza podataka. U prvom delu su date opšte karakteristike ove NoSQL baze podataka. Detaljno su opisane interna struktura i koncept indeksiranja, kao i dati neki primeri primene. Opisan je način kreiranja indeksa nad različitim strukturama podataka, kao što su kolekcije, skupovi, data je razlika između priarnih i sekundarnih indeksa, kao i opisan specifičan indeks SASI, njegova upotreba i značaj u Cassandra bazi podataka. Navedene su prednosti i mane korišćenja indeksa i dati primeri kada je poželjno koristiti ih, a kada to nije dobro i zašto. Data je i lista problema do kojih dovodi korišćenje indeksa u određenim situacijama.

U trećem poglavlju opisan je način pokretanja Cassandra baze podataka na računaru, kao i alata za lakše korišćenje. Nakon toga je opisan skup podataka koji se koristi i dat način kreiranja tabele i unosa podataka u Cassandra bazu podataka. Reč je o tabelama koje se tiču zakazivanja koncerata, gde je moguće zakazati bend, izvođača, mesto gde će se održati koncert. U istom poglavlju su dati primeri kreiranja indeksa nad skupom podataka i uočene razlike između toga kada je moguće to učiniti, kada ne i zbog čega.

Zaključak je da korišćenje indeksa u sistemima koji nemaju preveliki skup podataka, može biti koristan kako bi se došlo do podataka koji su neophodni, a kojima korišćenjem samo primarnog indeksa ne bi bilo moguće pristupiti. Ukoliko je skup podataka prilično veliki, primena indeksa može dovesti do toga da se efikasnost prilično smanji i dođe do toga da je potrebno dosta vremena kako bi se potrebni podaci dobili.

5. Literatura

- [1] Apache Cassandra, dostupno na: https://cassandra.apache.org/_/index.html
- [2] Apache Cassandra (NoSQL database) <https://www.geeksforgeeks.org/apache-cassandra-nosql-database/?ref=rp>
- [3] Internal Data Structure, dostupno na: https://teddyma.gitbooks.io/learncassandra/content/model/internal_data_structure.html
- [4] Indexing, dostupno na: <https://teddyma.gitbooks.io/learncassandra/content/model/indexing.html>
- [5] Concept of indexing in Apache Cassandra, dostupno na: https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/usePrimaryIndex.html
- [6] Cassandra at Scale:The Problem with Secondary Indexes, dostupno na: <https://pantheon.io/blog/cassandra-scale-problem-secondary-indexes>
- [7] Using a secondary index, dostupno na: https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useSecondaryIndex.html
- [8] Secondary Indexes, dostupno na: <https://cassandra.apache.org/doc/latest/cassandra/cql/indexes.html>
- [9] Using multiple indexes, dostupno na: https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useMultIndexes.html
- [10] Indexing a collection, dostupno na: https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useIndexColl.html
- [11] When to use an index, dostupno na; https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useWhenIndex.html
- [12] Concept of indexing in Apache Cassandra, dostupno na: <https://www.geeksforgeeks.org/concept-of-indexing-in-apache-cassandra/>
- [13] Secondary Indexes in Cassandra, dostupno na: <https://www.baeldung.com/cassandra-secondary-indexes>