Sistemi za upravljanje bazama podataka

Seminarski rad

Obrada upita kod Apache Cassandre

Mentor:
Aleksandar Stanimirović

Student:
Teodora Novković 1426

Sadržaj

Uvod

Bilo da se radi o relacionim SQL ili NoSQL bazama podataka, pribavljanje podataka iz baze podataka izdvaja se kao jedna od osnovnih operacija. Različite baze podataka implementirale su različite načine pristupa podacima – direktni pristup, pristup preko određenog upita, pristup na osnovu određene reči ili fraze, odn. termina. Prva dva pristupa se mogu naći kod svih baza, dok se treći obično implementira samo kod NoSQL baza podatka i baza podataka koje služe kao indeksi za pretragu, poznatiji kao „pretraživači" (eng. *Search Engines*). Najbitniji i najčešće korišćen pristup podacima jeste zadavanjem upita u određenom upitom jeziku. U najvećem broju slučajeva to je SQL ili upitni jezik slične sintakse nastao iz njega (kod relacionih baza podataka), dok se kod NoSQL baza podataka izdvajaju svojstveni jezici koji mogu strukturno da se oslanjaju na redosled zadavanja upita SQL-a, a mogu biti i potpuno nezavisni. Takav je Cassandrin specifični CQL upitni jezik (eng. *Cassandra Query Language, CQL*), koji se svojom strukturom i operacijama oslanja na SQL upitni jezik.

Cassandra pripada column-store tipu baza podataka. Predstavlja skalabilnu i distribuiranu NoSQL bazu podataka otpornu na greške. U Cassandri, pisanje je „skoro besplatno" zahvaljujući njenoj arhitekturi, u poređenju sa relacionim bazama podataka. Latencija pisanja može biti u stotinama mikrosekundi i veliki proizvodni klasteri mogu podržavati milione pisanja u sekundi. Operacije čitanja su takođe veoma brze, pa zato Cassandra spada u grupu baza podataka sa visokim performansama, koju odlikuju veoma brze operacije čitanja i upisa.

Dok se kod relacionih baza podataka obično podstiče skladištitenje podatka u normalizovanom obliku radi uštede prostora na disku i obezbeđivanja integriteta podataka, u Cassandri, redundantno skladištenje u više tabela ne samo da se očekuje, već takođe ima tendenciju da bude karakteristika dobrog modela podataka.

Na početku rada biće opisan pristup na kom se bazira modelovanje podataka u Cassandri, a zatim i opisana demonstrativna baza podataka na koju će se referencirati svi primeri u radu. U drugom poglavlju, detaljnija pažnja biće posvećena samim upitima, njihovoj pozadinskoj obradi u Cassandri i njihovom planu izvršenja. U trećem pogljavlju biće obrađen CQL upitni jezik, njegove specifičnosti i upiti koji se njime mogu definisati, kao i same Cassandrine specifičnosti.

Cassandrin „Query First" pristup

U tradicionalnim bazama podataka tabele baza podataka reprezentuju entitete u realnom svetu međusobno povezanih relacijama (relaciono modeliranje). Takva spajanja tabela (eng. „joins") predstavljaju dobro rešenje kada je baza podataka podignuta na jednoj mašini. Međutim, kada je baza podataka distribuirana i kada su sami podaci distribuirani na više mašina, spajanje tabela ne predstavlja efikasnu soluciju, jer bi u tom slučaju jedan upit ka bazi podataka zahtevao povezivanje i sa ostalim mašinama u klasteru (istovremeno), pa onda ne bi postojao efikasan način za obradu veće količine ovakve vrste upita.

Cassandra dobija dosta na brzini u čitanju i pisanju samo iz činjenice da nikada ne mora da obavlja spajanja u bazi podataka, tj. u Cassandri je zapravo nemoguće izvršiti spajanje. Umesto pristupa relacionog modela, Cassandra koristi „Query First" pristup ili „prvo upiti", što znači da se tabele dizajniraju unapred za specifične upite. U ovakvom pristupu tabele predstavljaju refleksiju upita koji će se kasnije koristiti. Kao rezultat ovakvog pristupa dobija se više tabela, gde je svaka od njih prilagođena odgovarajućem upitu.

Svakako da postoje očigledne posledice ovakvog pristupa, jer se evidentno može doći u situaciju pisanja jednih te istih podataka u više različitih tabela, kako bi se zadovoljili različiti upiti. Međutim, Cassandrina arhitektura je idealna za ovakav scenario. Iz ovih razloga treba pažljivo modelovati podatke u Cassandri da bi bili sigurni da će kreirane tabele zadovoljiti upite koje unapred pretpostavljamo i očekujemo da će nam trebati. Kod ovog pristupa, treba uvek imati na umu i samu činjenicu da se kasnije (nakon definisanja tabela) može stvoriti potreba za izvršavanje upita za koje ne postoji predviđena podrška. Ali i u takvim situacijama, postoje izvodljive alternative, iako neke od njih mogu biti teško izvodljive ili ne tako teško izvodljive, ali loše sa stanovišta performansi. U nastavku rada biće detaljnije opisane pomenute alternative kojima se pribegava u opisanoj situaciji.

Čak i sa očiglednim nedostacima sa kojima se model Cassandrine baze podataka nekad mora pomiriti, Cassandra pruža ogromne dobitke u pogledu performansi i prostora za skladištenje u distribuiranom sistemu. Dok bi relacione baze podataka mogle biti bolji izbor za manje podatke na jednoj mašini ili u jednom data centru.

U nastavku biće opisana demonstrativna baza podataka modelovana na bazi ovog pristupa, sa jasnim ciljem da tabele u bazi podataka treba da budu refleksija upita koji će se koristiti.

Baza podataka MovieCritic

Za potrebe demonstracije korišćena je poslednja stabilna verzija Cassandre, Apache Cassandra 3.11.12. Kao baza podataka za demonstraciju, korišćena je baza koja se bavi recenzijama filmova, gde korisnici mogu da se detaljno informišu o filmovima, kao i njihovim ocenama, preporukama i recenzijama. Takodje korisnici mogu da beleže svoje omiljene i prethodno odgledane filmove.

Name Film Primary godina naziv zanr

column_name	data_type	kind
godina	text	clustering
naziv	text	clustering
sifra	text	regular
zanr	text	partition_key

Slika 3.1. *Tabela Film, njen partition i clustering key*

Tabela „Film" sadrzi informacije o filmovima: žanr (particioni ključ), godina (ključ za grupisanje), naziv (ključ za grupisanje), šifra filma. Kreirana je tako da se upitima efikasno može izvršiti:

- pribavljanje svih raspoloživih žanrova po kojima su filmovi kategorisani
- pribavljanje svih filmova određenog žanra (žanr od interesa izabran od strane korisnika), sortiranih po godini ili nazivu
- pribavljanje svih filmova

zanr	godina	naziv	sifra
krimi	1990	Goodfellas	1080
krimi	1967	Bonnie and Clyde	1039
horor	2019	Annabelle Comes Home	1010
horor	2018	Upgrade	1011
horor	2017	Get Out	1012
horor	2016	Split	1013
horor	2015	Green Room	1014
horor	2014	What We Do in the Shadows	1015
horor	2013	The Conjuring	1016
horor	2012	Sinister	1017
horor	2011	Take Shelter	1018
horor	1960	Psycho	1019
triler	2016	Don't breathe	1050
triler	2015	Room	1051
triler	2013	Oldboy	1052
triler	2006	Das Leben der Anderen	1054
triler	2006	The Departed	1053
triler	2001	Doni Darko	1055
triler	1999	Fight Club	1056
triler	1995	Seven	1057
triler	1995	The Usual Suspects	1058
triler	1991	The Silence of the Lambs	1059
romantika	2019	A Hidden Life	1071
romantika	2019	Judy	1073
romantika	2019	Little Women	1070
romantika	2019	Waves	1072

Slika 3.2. *Podaci u tabeli Film*

Tabela „ReziserGlumci" sadrži detaljnije informacije o filmovima: šifra filma (particioni ključ), glumci, režiser. Kreirana je za pribavljanje dodatnih informacija o filmu, koje bi korisnicima bile od interesa.

Name ReziserGlumci Primary sifra

column_name	data_type	kind
glumci	text	regular
reziser	text	regular
sifra	text	partition_key

Slika 3.3. *Tabela ReziserGlumci*

Tabela „Korisnik" sadrži informacije o korisnicima i administratorima: korisničko ime (particioni ključ), lozinka (particioni ključ), ime, prezime, e-mail, posedovanje administratorskih permisija. Kreirana je tako da se nakon unosa kredencijala u formi za prijavljivanje na sistem lako mogu pribaviti informacije o prijavljenom korisniku.

Name Korisnik Primary password username

column_name	data_type	kind
admin	text	regular
ime	text	regular
mail	text	regular
password	text	partition_key
prezime	text	regular
username	text	partition_key

Slika 3.4. *Tabela Korisnik*

Tabela „Ocene" sadrži informacije o ocenama koje su korisnici evidentirali za filmove: šifra filma (particioni ključ), korisničko ime (ključ za grupisanje), lozinika (ključ za grupisanje), ocena filma. Kreirana je tako da se za određeni film lako mogu pribaviti sve njegove ocene.

Name Ocene Primary password sifraFilma username

column_name	data_type	kind
ocena	int	regular
password	text	clustering
sifraFilma	text	partition_key
username	text	clustering

Slika 3.5. *Tabela Ocene*

Tabela „Recenzije" sadrži informacije o korisničkim recenzijama za filmove: šifra filma (particioni ključ), datum (ključ za grupisanje), recenzija (ključ za grupisanje), korisničko ime. Kreirana je tako da se za određeni film lako mogu pribaviti sve njegove recenzije, sa mogućnošću sortiranja po datumu.

Name Recenzije Primary datum recenzija sifraFilma

column_name	data_type	kind
datum	date	clustering
recenzija	text	clustering
sifraFilma	text	partition_key
username	text	regular

Slika 3.6. *Tabela Recenzije*

Tabela „Preporuke" sadrži informacije o korisničkim preporukama za filmove: šifra filma (particioni ključ), korisničko ime (ključ za grupisanje), lozinka (ključ za grupisanje), recenzija (ključ za grupisanje), preporuka. Kreirana je tako da se za određeni film mogu pribaviti sve njegove korisničke preporuke.

Name	Preporuke	Primary	password	sifraFilma	username
column_name	data_type	kind			
password	text	clustering			
preporuka	text	regular			
sifraFilma	text	partition_key			
username	text	clustering			

Slika 3.7. *Tabela Preporuke*

„OmiljeniFilmovi" i „Odgledani filmovi" sadrže informacije o tome koje filmove su korisnici evidentirali kao omiljene, odnoso odgledane: korisniko ime (particioni ključ), lozinka (particioni ključ), žanr (ključ za grupisanje), godina (ključ za grupisanje), naziv (ključ za grupisanje), šifra filma. Kreirane su tako da se za prijavljene korisnike lako može pribaviti spisak njihovih omiljenih i odgledanih filmova.

Name	OmiljeniFilmovi	Primary	godina	naziv	password	username	zanr
column_name	data_type	kind					
godina	text	clustering					
naziv	text	clustering					
password	text	partition_key					
sifraFilma	text	regular					
username	text	partition_key					
zanr	text	clustering					

Slika 3.8. *Tabela OmiljeniFilmovi*

CQL upitni jezik

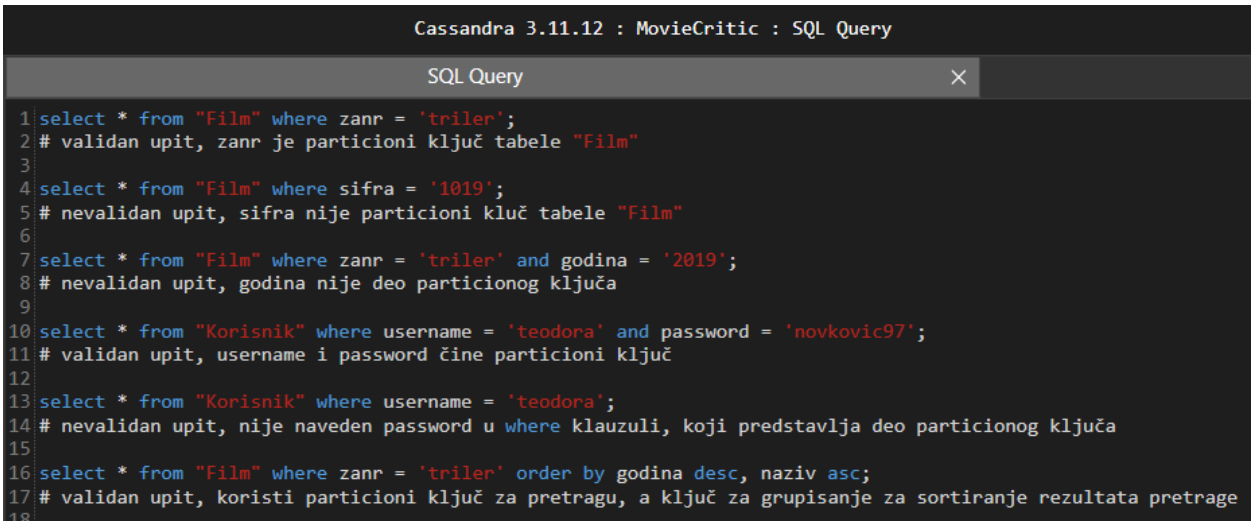
Prvi korak u obradi upita jeste samo definisanje upita. Za to se koristi CQL (eng. „Cassandra Query Language") upitni jezik koji se svojom strukturom zasniva na SQL-u.

Iako dele sličnu sintaksu, postoji mnogo razlika između CQL-a i SQL-a. Razlozi za ove razlike potiču uglavnom iz činjenice da se Cassandra bavi distribuiranim podacima i ima za cilj da spreči neefikasne upite.

Osnovni upit kao i kod SQL-a zadaje se SELECT naredbom, pri čemu kod CQL-a postoje određene restrikcije kod zadavanja upita koji sadrže WHERE klauzulu.

U Cassandri, dve vrste kolona imaju posebnu ulogu: kolone particionih ključeva i kolone za grupisanje. Ove kolone, koje odgovaraju particionom ključu (eng. „partition key") i ključu za grupisanje (eng. „clustering key"), zajedno predstavljaju primarni ključ. Kolone particionog ključa su prvi deo primarnog ključa i njihova uloga je da ravnomerno rasporede podatke po klasteru. Redovi će biti raspoređeni po klasteru na osnovu heširane vrednosti particionih ključeva. Za tu svrhu se koristi Consisten Hashing algoritam, koji od particionog ključa generiše 64-bitni token koji identifikuje čvor u klasteru u kojem će sami redovi biti sačuvani. Kolone za grupisanje se koriste za grupisanje podataka particije, omogućavajući veoma efikasno pronalaženje redova.

Zbog razlika u ulozi koju igraju, kolone particionih ključeva, kolone ključeva za grupisanje i normalne kolone podržavaju različite skupove ograničenja unutar WHERE klauzule. Jedno od ograničenja je obaveza da se usled korišćenja WHERE klauzule navedu sve kolone particionih ključeva, tj. biće odbačeni svi upiti kod kojih se u WHERE klauzuli navedu samo neke kolone particionih ključeva ili samo kolone koje nisu kolone particionih ključeva. Razlog zašto su Cassandri potrebne sve kolone particionih ključeva je da bi Cassandra mogla da izračuna heš koji će joj omogućiti da locira čvorove koji sadrže particiju.



Slika 4.1. *CQL Upiti*

U CQL-u postoji način da se prevaziđe prethodno opisano ograničenje pri zadavanju SELECT upita sa WHERE klauzulom i pretraži željena tabela po koloni koja nije deo particionog ključa. U tom slučaju neophodno je eksplicitno navesti u upitu „ALLOW FILTERING", ali se gotovo nikada ne preporučuje zadavanje ove vrste upita, jer neretko mogu biti veoma neefikasni, budući da Cassandra u tom slučaju pribavlja sve redove tabele, a zatim odbacuje one koji ne zadovoljavaju zadati uslov u upitu (neefikasno kada su odbačeni redovi u većini, posebno kada je potrebno obraditi ogromnu količinu redova). Kao alternative koje su se mnogo uspešnije pokazale od prethodno opisane su:

- pravljenje sekundarnog indeksa
- pravljenje pogleda pridruženog tabeli
- pravljenje nove redudantne tabele koja zadovoljava željene upite

Sve ove alternative i njihovi koncepti biće detaljnije opisani u nastavku rada.

Dodatna CQL restrikcija je da su nad kolonama particionog ključa dozvoljeni samo „=" i „IN", „NOT IN" operatori, ali ne i operatori opsega „>", „>=", „<=", „<" (osim ako se za particionisanje ne koristi ByteOrderedPartitioner koji održava sortiranost distriburianih podataka). Nad regularnim kolonama dozvoljeni su „=", „>", „>=", „<=", „<" operatori, ali nisu dozvoljeni "IN" i „NOT IN" operatori, dok su svi navedeni operatori dozvoljeni nad kolonama ključa za grupisanje.

Takođe CQL podržava i operatore „CONTAINS", „CONTAINS KEY" za pretragu sadržaja nad tekstualnim poljima radi ispitivanja da li se vrednost polja poklapa sa navedenom vrednošću. Pomoću „LIMIT" operatora se u CQL-u može ograničiti maksimalni broj redova koji će upit vratiti.

```
                                    Cassandra 3.11.12 : MovieCritic : SQL Query
1
2 SELECT * FROM "MovieCritic"."Film" WHERE "zanr" = 'krimi';
3 SELECT * FROM "MovieCritic"."Film" WHERE "zanr" IN ('krimi', 'triler', 'akcioni');
4 # validni upiti, korišćenje '=' i 'IN' operatora za pretragu kolone particionog ključa
5
6 SELECT * FROM "MovieCritic".OceneFilmova where "sifraFilma" = '1058' and ocena >= 9;
7 # nevalidan upit, operatori opsega se ne mogu koristiti za pretragu kolone particionog ključa
8
```

Slika 4.2. *Primer korišćenja operatora*

Za definisanje kompleksnijih upita dostupan je „AND" logički operator, kao što je i prikazano na prethodnom primeru. Međutim, Cassandra nema podršku za korišćenje „OR" operatora u WHERE klauzuli, na šta se može gledati kao na još jednu restrikciju CQL upitnog jezika.

CQL, slično SQL-u, podržava i funkcije agregacije „AVG", „MAX", „MIN", „COUNT", „SUM", a na sledećem primeru prikazan koji koristi funkciju agregacije „AVG" za računanje prosečne ocene za određeni film.

```
11 SELECT AVG("ocena") AS PROSECNA_OCENA
12 FROM "MovieCritic"."Ocene"
13 WHERE "sifraFilma" = '1058';
14 # računanje prosečne ocene za određeni film korišćenjem funkcije agregacije
```

prosecna_ocena
9

Slika 4.3. *Primer korišćenja funkcije agregacije*

CQL ima podršku upisa podatka u json formatu, kao i čitanja podatka u json formatu. Na sledećem primeru prikazan je upis podatka u tabeli „Film" u json formatu, kao i čitanje istog u json formatu.

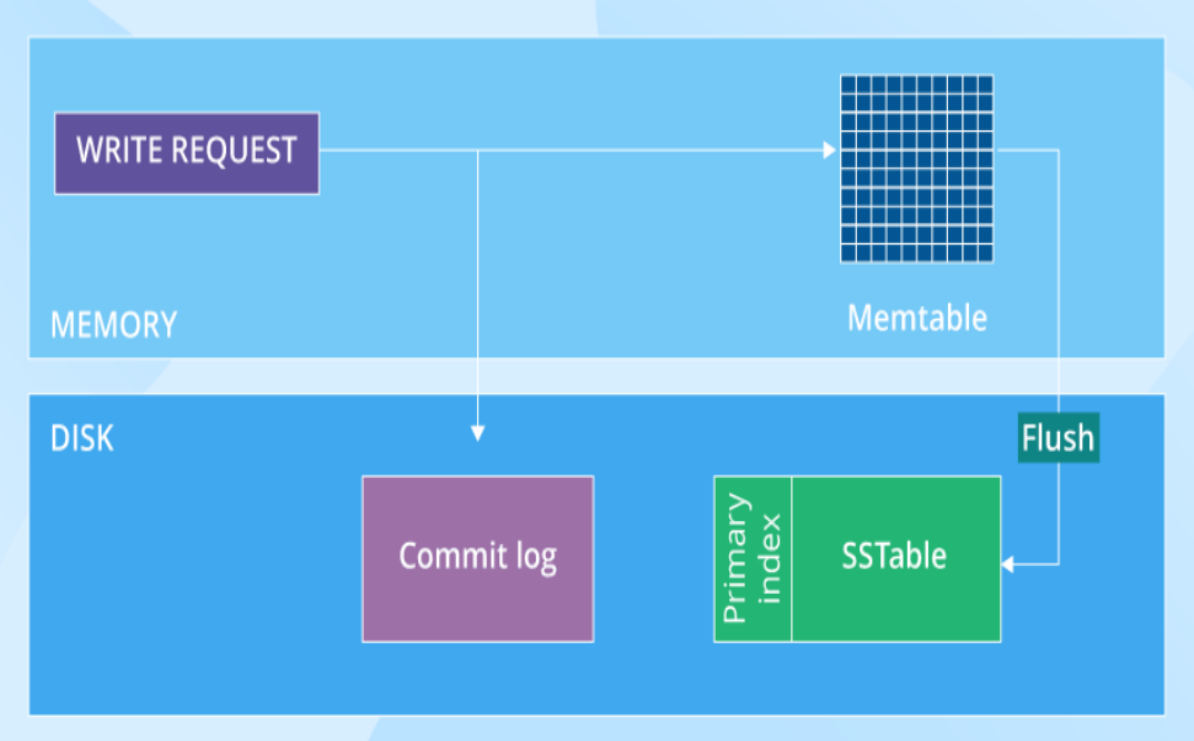
```
16 INSERT INTO "MovieCritic"."Film"
17 JSON '{
18 "zanr" : "komedija",
19 "godina" : "2021",
20 "naziv" : "Dont Look Up",
21 "sifra": "10101010"
22 }';
23
24 SELECT JSON * FROM "MovieCritic"."Film" WHERE "zanr"='komedija' and "godina"='2021' and "naziv"='Dont Look Up';
```

[json]
{"zanr": "komedija", "godina": "2021", "naziv": "Dont Look Up", "sifra": "10101010"}

Slika 4.4. *Primer korišćenja operatora*

Obrada Write upita

Proces upisa započinje tako što čvor koordinator identifikuje koji čvor je zadužen za upis podataka i od tog trentuka čvor koordinator postaje odgovaran za uspešno/neuspešno kompletiranje procesa upisa. Nakon što čvor upiše podatke, on obaveštava čvor koordinatora o uspešno završenoj operaciji. Istovetni proces upisa se obavlja i u čvorovima u kojima se upisuju replike. U kom trenutku će Cassandra potvrditi da je upis uspešno kompletiran zavisi od podešene strategije konzistentnosti („any", „one", „quorum", „local quorum", „each quorum", „all").



Slika 5.1. *Graficki pirkaz procesa obrade upisa u Cassandri*

Nakon što bude upućen na određeni čvor, zahtev za pisanje prvo stiže do „Commit log" skladišta, koje predstavlja privremeno skladište u kojem se čuvaju keširani upisi i koristi se u slučajevima neuspelih upisa ili slučajevima otkazivanja čvorova kako bi se pročitale neophodne informacije za novi pokušaj upisa i skladištenja podataka („restore" proces).

U isto vreme, podaci se upisuju i u Memtable, koji predstavlja keš memoriju u kojoj se skladište particije podataka koje Cassandra pretražuje po ključu. Cassandra smatra upis uspešno kompletiranim kada se podatak upiše i u Commit log i u Memtable, iako se realan upis na disk dešava kasnije. U trenutku kada ispunjenost Memtable memorije dostige izvestan prag (koji se može konfigurisati), Cassandra ispušta podatke iz keša na disk („flush" operacija) – u SSTables.

SSTable-ovi predstavljaju fajlove, tj. fizičko skladište. Svaka flush operacija rezultira novom SSTable-om. Ako tokom određenog vremenskog perioda naraste broj SSTable-a, kako ne bi došlo do degradiranja performansi čitanja (da ne bi moralo da se prilikom čitanja prolazi kroz više SSTable-ova), Cassandra povremeno u pozadini izvodi operaciju spajanja više SSTable-ova u jedan SSTable, radi optimizovanja operacija čitanja. Ova operacija je poznatija kao „kompakcija". U trenutku flush operacije, Commit log čisti sve svoje podatke, pošto više ne mora da pazi na odgovarajuće podatke u kešu. U Commit log-u se podaci ne čuvaju u sortiranom redosledu, kao što je to slučaj kod Memtable i SStable, već se čuvaju u onom redosledu po kojem je Cassandra obrađivala te podatke.

U slučaju otkaza nekog čvora, bilo da isti prestane da raspolaže zbog mrežnih ili hardverskih problema, bilo zbog preopterećenja samog čvora, Cassandra pribegava „Hinted Handoff" procesu. U ovom procesu koordinator čvor detektuje (putem Gossip protokola) neaktivnost čvora koji poseduje odgovarajuću repliku , a zatim skladišti „hint" podatke o toj replici u lokalnoj tabeli hint-ova. U hint-u se upisuje lokacija same replike koja više nije raspoloživa, verzija meta-podatka i sam podatak koji se upisuje. Ukoliko u odgovarajućem konfigurabilnom vremenskom intervalu koordinator čvor utvrdi ponovnu aktivnost čvora za koji on čuva hintove, na osnovu svakog hitna šalje ciljnom čvoru odgovarajuće podatke koje je potrebno upisati. Ako se čvor ne oporavi u izvesnom vremenskom periodu, čvor koordinator trajno čuva zapis. Cassandra sama brine o automatskom brisanju hintova za čvorove koji više ne postoje ili koji referenciraju obrisane tabele.

```
cqlsh> INSERT INTO "MovieCritic"."Film" (zanr, godina, naziv, sifra) values ('triler', '1995', 'The Usual Suspects', '112233');

Tracing session: 145e3930-c58e-11ec-8c80-c1e80cd50c39

 activity                                                                                                                           | timestamp                  | source    | source_e
lapsed | client
------------------------------------------------------------------------------------------------------------------------------------+----------------------------+-----------+---------
-------+----------
                                                                                                                 Execute CQL3 query | 2022-04-26 20:24:15.939000 | 127.0.0.1 |
     0 | 127.0.0.1
 Parsing INSERT INTO "MovieCritic"."Film" (zanr, godina, naziv, sifra) values ('triler', '1995', 'The Usual Suspects', '112233'); [Native-Transport-Requests-1] | 2022-04-26 20:24:15.939000 | 127.0.0.1 |
   131 | 127.0.0.1
                                                                                 Preparing statement [Native-Transport-Requests-1] | 2022-04-26 20:24:15.939000 | 127.0.0.1 |
   250 | 127.0.0.1
                                                                      Determining replicas for mutation [Native-Transport-Requests-1] | 2022-04-26 20:24:15.939000 | 127.0.0.1 |
   523 | 127.0.0.1
                                                                             Appending to commitlog [Native-Transport-Requests-1] | 2022-04-26 20:24:15.940000 | 127.0.0.1 |
   755 | 127.0.0.1
                                                                            Adding to Film memtable [Native-Transport-Requests-1] | 2022-04-26 20:24:15.940000 | 127.0.0.1 |
   835 | 127.0.0.1
                                                                                                                   Request complete | 2022-04-26 20:24:15.943413 | 127.0.0.1 |
  4413 | 127.0.0.1
```

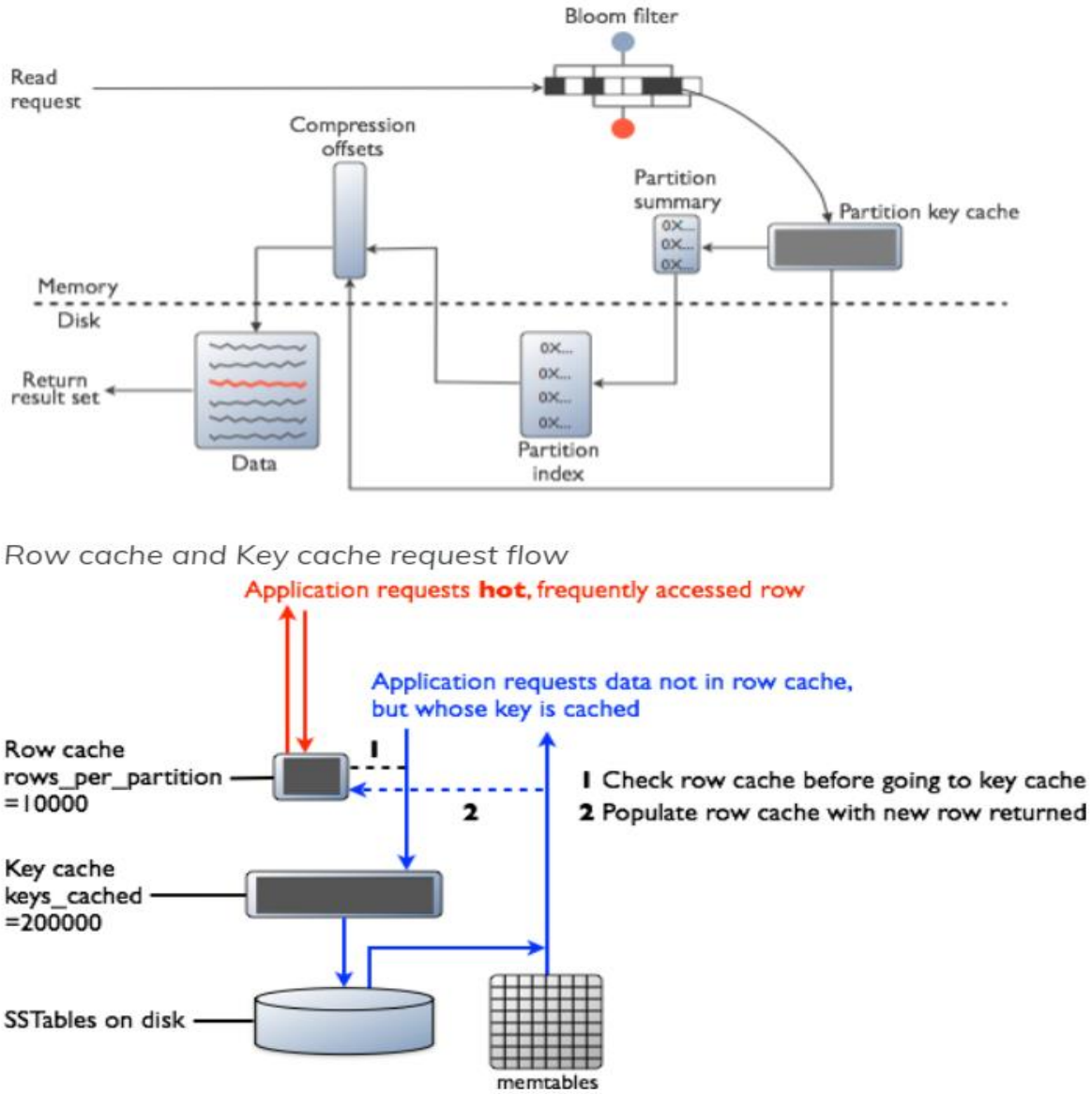
Slika 5.2. *Plan izvršenja upisa u Cassandri*

U nastavku je dat primer plana izvršenja upita za upis u Cassandru (dodavanje novog filma u tabeli „Film", u „MovieCritic" keyspace-u). Za prikaz plana izvršenja upita u radu korišćen je cqlsh command-line interfejs, posredstvom kog se korišćenjem CQL upitnog jezika interagovalo sa Cassandrom. Korišćen je Python 2.7.10 kao kompatibilna verzija za cqlsh. Radi detaljnog prikaza plana izvršenja upita u Cassandri korišćeno aktiviran je mod praćenja upita (eng. „tracing"). Na primeru iznad se može videti obrada Cassandrinog upisa u pozadini, koja prati upis u Commit-log i Memtable, kao što je prethodno i opisano u poglavlju. U trenutku kada podatak postane

upisan istovremeno i u Commit-log i u Memtable, Cassandra smatra upis uspešno kompletiranim, iako se realan upis na disk dešava kasnije.

Obrada Read upita

Kako bi Cassandra uspešno obradila odgovarajući zahtev za čitanje nekog podatka, mora kombinovati rezultate iz aktivnog MemTable-a i više potencijalnih SSTable-ova. Prilikom obrade zahteva za čitanje podatka Cassandra ne razmatra Commit log, budući da nije čitljiv (eng. „readable") i služi samo za rekreiranje izgubljenih podataka u slučaju otkazivanja čvora.



Slika 6.1. *Graficki prikaz procesa čitanja u Cassandri*

Cassandra pretragu po particionom ključu započinje najpre iz MemTable-a, pri čemu ukoliko tu nađe više rezultata pretrage, poredi ih po vremenskim oznakama (eng. „timestamps") i uzima u obzir samo podatak koji je najskoriji.

Pre nego što otpočne eventualnu pretragu po SSTable-ovima, Cassandra proverava keš redova, koji kešira cele redove za frekventno čitane redove iz SSTable-ova (ovo keširanje je podležno konfiguraciji, tj. može se uključiti/isključiti). Zatim se podatak pretražuje po SSTable-ovima, a kako bi se efikasno pretražili svi SSTable-ovi Cassandra prolazi nekoliko koraka:

- Prvo se na osnovu Bloom filtera ustanovljava koji SSTable-ovi potencijalno sadrže podatak. Svaki SSTable ima svoj Bloom filter, a oni daju informaciju o tome da li se podatak koji se traži ne nalazi u njegovom SSTable-u (sa 100% sigurnošću) ili postoji velika verovatnoja da SSTable sadrži traženi podatak. Kada se ustanove potencijalni SSTable-ovi, pristupa se traženju unutar njih.
- Za odgovarajući SSTable se proverava keš ključeva (ukoliko je uključeno keširanje ključeva), u kome se smeštaju keširani indeksi onih particija kojima se frekventno pristupa. Ukoliko se indeks particije nalazi u kešu onda se direktno preko kompresione mape ofseta nalazi konkretna lokacija bloka na disku gde se nalazi podatak.
- Ukoliko se particioni ključ ne nalazi u kešu, onda se pretražuje prostor na disku koji za sve particione ključeve poseduje indekse njihovih particija (Partition Index). Međutim, u situacijama gde postoji veliki broj particionih ključeva, traženje indeksa njihovih particija bi predstavljao jedan skup proces. Zato Cassandra koristi Partition Summary, koji za razliku od Partition Index-a koji sadrži sve particione ključeve, grupiše svakih N particionih ključeva i mapira lokaciju grupe u odgovarajućem fajlu, radi brže pretrage particionih ključeva.
- Kada se odredi indeks željene particije onda se pomoću kompresione mape ofseta nalazi sama lokacija bloka na disku gde se nalazi podatak. Kompresiona mapa ofseta čuva pokazivače na konkretne lokacije na disku gde se nalaze particije. Kada se utvrdi konkretna lokacija particije na disku, podaci sa particije se pribavljaju iz odgovarajućeg SSTable-a a zatim se sekvencijalnim skeniranjem particije dolazi do željenog podatka. Na ovaj način Cassandra efikasno izbegava sekvencijalnu pretragu kroz čitav SSTable.

Dobijene rezultate pretrage, što iz Memtable, što iz SSTable, Cassandra poredi po vremenskoj oznaci (timestamp-u) i vraća najskoriji rezultat. Na primeru ispod je prikazan plan izvršenja upita čitanja u Cassandri, gde se jasno vidi Cassandrino pretraživanje Memtable i odgovarajućih SSTable-ova.

```
cqlsh>  select * from "MovieCritic"."Film" where "zanr" IN ('krimi','triler');

 zanr   | godina | naziv                     | sifra
--------+--------+---------------------------+--------
  krimi |   2019 |      John Wick:Chapter 3  |   1032
  krimi |   2019 |                   Joker   |   1030
  krimi |   2019 |              Knives Out   |   1031
  krimi |   2018 |               Andhadhun   |   1033
  krimi |   2017 |              Wind River   |   1034
  krimi |   2016 |            Contratiempo   |   1035
  krimi |   2015 |                  Talvar   |   1036
  krimi |   2014 |            Nightcrawler   |   1037
  krimi |   1994 |            Pulp Fiction   |   1038
  krimi |   1990 |             Goodfellas    |   1080
  krimi |   1967 |        Bonnie and Clyde   |   1039
 triler |   2016 |           Don't breathe   |   1050
 triler |   2015 |                   Room    |   1051
 triler |   2013 |                 Oldboy    |   1052
 triler |   2006 |    Das Leben der Anderen  |   1054
 triler |   2006 |            The Departed   |   1053
 triler |   2001 |              Donl Darko   |   1055
 triler |   1999 |              Fight Club   |   1056
 triler |   1995 |                  Seven    |   1057
 triler |   1995 |       The Usual Suspects  | 112233
 triler |   1991 | The Silence of the Lambs  |   1059

(21 rows)

Tracing session: a0ff7d20-c595-11ec-8c80-c1e80cd50c39

 activity                                                                                                                 | timestamp                  | source    | source_elapsed | client
--------------------------------------------------------------------------------------------------------------------------+----------------------------+-----------+----------------+-----------
                                                                                                      Execute CQL3 query | 2022-04-26 21:18:18.354000 | 127.0.0.1 |              0 | 127.0.0.1
 Parsing select * from "MovieCritic"."Film" where "zanr" IN ('krimi','triler'); [Native-Transport-Requests-1] | 2022-04-26 21:18:18.354000 | 127.0.0.1 |            175 | 127.0.0.1
                                                                                   Preparing statement [Native-Transport-Requests-1] | 2022-04-26 21:18:18.354000 | 127.0.0.1 |            337 | 127.0.0.1
                                                                  Executing single-partition query on Film [ReadStage-2] | 2022-04-26 21:18:18.365000 | 127.0.0.1 |          10489 | 127.0.0.1
                                                                                       Acquiring sstable references [ReadStage-2] | 2022-04-26 21:18:18.365000 | 127.0.0.1 |          10570 | 127.0.0.1
                                  Skipped 0/0 non-slice-intersecting sstables, included 0 due to tombstones [ReadStage-2] | 2022-04-26 21:18:18.365000 | 127.0.0.1 |          10637 | 127.0.0.1
                                                                 Merged data from memtables and 0 sstables [ReadStage-2] | 2022-04-26 21:18:18.365000 | 127.0.0.1 |          10782 | 127.0.0.1
                                                                   Read 11 live rows and 0 tombstone cells [ReadStage-2] | 2022-04-26 21:18:18.365000 | 127.0.0.1 |          10893 | 127.0.0.1
                                                                  Executing single-partition query on Film [ReadStage-3] | 2022-04-26 21:18:18.367000 | 127.0.0.1 |          12092 | 127.0.0.1
                                                                                       Acquiring sstable references [ReadStage-3] | 2022-04-26 21:18:18.367000 | 127.0.0.1 |          12168 | 127.0.0.1
                                  Skipped 0/0 non-slice-intersecting sstables, included 0 due to tombstones [ReadStage-3] | 2022-04-26 21:18:18.367000 | 127.0.0.1 |          12219 | 127.0.0.1
                                                                 Merged data from memtables and 0 sstables [ReadStage-3] | 2022-04-26 21:18:18.367000 | 127.0.0.1 |          12302 | 127.0.0.1
                                                                   Read 10 live rows and 0 tombstone cells [ReadStage-3] | 2022-04-26 21:18:18.367000 | 127.0.0.1 |          12337 | 127.0.0.1
                                                                                                        Request complete | 2022-04-26 21:18:18.366518 | 127.0.0.1 |          12518 | 127.0.0.1
```

Slika 6.2. *Plana izvršenja upita čitanja u Cassandri*

Regularni sekundarni indeksi

Kako je vremenom model podataka podložan promeni i može evoluirati, može doći do situacije da su za pretragu podataka potrebne upravo one kolone koje se po Cassandrinim restrikcijama, koje su objašnjene u prethodnom delu rada, ne mogu pretraživati. U takvim situacijama, sekundarni indeksi mogu biti jedna od opcija. Druga opcija bi svakako bilo kreiranje redudantne duplikat tabele sa kolonom koju želimo pretraživati kao deo particionog ključa. Sekundarni indeksi predstavljaju zapravo lokalne indekse (skrivene tabele) koji postoje lokalno u čvoru i vidljivi su na nivou tog čvora, ali ne i na nivou klastera. Iz tog razloga se sekundarni indeksi poistovećuju sa lokalnim, a primarni inkdeksi sa globalnim indeksima. Takođe, sekundarni indeksi pružaju mogućnost i indeksiranja kolekcije kolona. Međutim, za razliku od relacionih baza podataka, ovde sekundarni indeksi pružaju potencijalne pogodnosti, ali ne i performanse.

Jedan od primera potrebe za sekundarnim indeksom se ogleda i kroz tabelu „Ocene", gde šifra filma predstavlja particioni ključ, a korisničko ime i lozinka ključ za grupisanje. Ukoliko u nekom trenutku bude neopodno iz ove tabele pribaviti podatke o konkretnim ocenama (npr. samo konkretne najviše ocene ili samo konkretne niske ocene i filmove koji su istim ocenjeni) Cassandrine restrikcije ne bi dozvolile tu vrstu upita, a „allow filtering" bi svakako bila opcija koja se zbog degradiranja performansi uvek izbegava. Sekundarni indeks nad kolonom „ocena" bi u pozadini kreirao lokalnu tabelu gde bi ta kolona predstavljala deo particionog ključa i samim tim postala pretraživa. Na narednoj slici je prikazan ovaj primer.

Cassandra 3.11.12 : MovieCritic : SQL Query

SQL Query

```
1
2 CREATE INDEX IF NOT EXISTS ON "MovieCritic"."Ocene" (ocena);
3 # kreiranje sekundarnog indeksa nad kolonom ocena
4
5 select * from "MovieCritic"."Ocene" where ocena = 9;
6 # pretraživanje tabele Ocene po samoj vrednosti ocene
```

sifraFilma	username	password	ocena
1048	nemanja	milosevic98	9
1048	tanja	novkovic77	9
1048	vladimir	janjic96	9
1031	tanja	novkovic77	9
1072	tanja	novkovic77	9

Slika 7.1. *Primer kreiranja sekundarnog indeksa*

Ono što se da primetiti sa slike je da sada i pored postojanja sekundarnog indeksa, filmovi lokalno (u čvorovima) ostaju particionisani po particionom ključu osnovne tabele, odnoso po šifri filma. To znači da pri pretrazi filmova po konkretno zadatoj oceni u select upitu, sami podaci koji su targetirani ovom pretragom bi bili distribuirani po različitim čvorovima, pa bi iz tog razloga svaki od čvorova morao biti skeniran pri obradi ovakvog upita (ukoliko uzmemo u obzir da je faktor replikacije RF=1, a u opštem slučaju pri obradi ovakvog upita bilo bi skenirano ukupan_broj_čvorova / RF čvorova). Kada se uzmu u obzir slučajevi postojanja ogromnog broja čvorova, ovakvi upiti bi bili poprilično loši sa stanovišta performansi.

U ovakvom slučaju, opcija kreiranja redudantne tabele sa ocenom kao particionim ključem bi značila da su filmovi sa npr. ocenom 10 u istoj jednoj particiji, filmovi sa ocenom 9 u istoj drugoj particiji,... To bi značilo da bi se pri obradi istog upita pretraživao samo jedan čvor, samo onaj kome je dodeljena tražena particija, što bi direktno značilo bolje performanse pri obradi upita ove vrste.

Prilikom obrade upisa u Cassandri, kod slučaja sekundardnih indeksa, flush operacija ispisuje podatke na disk, ali u ovom slučaju i podatke iz indeksirane i osnovne Memtable, sa velikom verovatnoćom da će se podaci ispisati iz obe tabele na disk istovremeno. U procesu kompakcije, indeksna tabela može biti kompaktovana potpuno nezabisno od osnovne tabele. Što se tiče obrade čitanja u slučaju sekundarnih indeksa, Cassandra prvo iz indeksa pribavlja primarni ključ za sve redove koji zadovoljavaju upit, a zatim za svaki od njih pribavlja podatke iz osnovne tabele.

Sekundarni indeksi se ne preporučuju za indeksiranje visoko kardinalnih ili nisko kardinalnih kolona, kao ni za indeksiranje kolona koje se frekventno ažuriraju. Preporučuje se njihovo korišćenje u situacijama kada upit očekuje na desetine do maksimum nekoliko stotina rezultata pretrage i u situacijama kada se indeksiraju kolone srednje kardinalnosti. Sekundarni indeksi predstavljaju dobru alternativu kada je potrebno pretražiti i primarni i sekundarni indeks, jer u tom slučaju na osnovu particionog ključa zna u kom čvoru su smešteni podaci , a na osnovu sekundarnog indeksa zna u kojoj lokalnoj tabeli se konkretno ti podaci i nalaze.

SASI indeksi

SASI (eng. „SSTable-attached secondary index") indeksi ili sekundarni indeksi koji su spregnuti sa SSTable-ovima predstavljaju za sada eksperimentalnu i mnogo efikasniju vrstu sekundarnih indeksa, budući da koriste B+ stabla za indeksiranje podataka i imaju podršku za operatore opsega (čak i „LIKE" operator) prilikom zadavanja upita, kao što je prikazano na narednoj slici.

Cassandra 3.11.12 : MovieCritic : SQL Query

SQL Query

```
1
2
3 select * from "MovieCritic"."Ocene" where ocena >= 8;
4 # nevalidan upit, sekundarni indeksi nemaju podršku za operatore opsega
5
6 CREATE CUSTOM INDEX  fn_ocene ON "MovieCritic"."Ocene" (ocena) USING 'org.apache.cassandra.index.sasi.SASIIndex';
7 # kreiranje SASI indeksa, kako bi se mogli pretraživati filmovi po opsegu ocena
8
9 select * from "MovieCritic"."Ocene" where ocena >= 9;
10 # validan upit
```

sifraFilma	username	password	ocena
1006	tanja	novkovic77	10
1006	vladimir	janjic96	10
1000	tanja	novkovic77	9
1060	tanja	novkovic77	9
1063	tanja	novkovic77	9

Slika 8.1. *Primer SASI indeksa i operatora opsega*

Batch operacije

Batch operacije u Cassandri omogućavaju grupisanje više različitih insert/update/delete upita zajedno, vodeći računa da se ne premaši maksimalan broj dozvoljenih operacija. Cassandra podržava 2 tipa Batch operacija – ***nelogovane i logovane Batch operacije***.

Logovane Batch operacije su atomične, drugim rečima ili će se sve grupisane operacije sigurno kompletirati ili neće ni jedna od njih.

Batch operacije su veoma korisne pri ažuriranju denormalizovanih tabela, kod ažuriranja jedne vrednosti koja je redudantna kroz više različitih tabela, pri čemu se sve operacije ažuriranja vrednosti u ovom slučaju u pozadini tretiraju kao jedna transakcija.

Iako je poboljšanost performansi jedna od prvih asocijacija na Batch operacije, obično je suprotno slučaj. Jedan koordinator čvor je odgovoran za sve upite koji se nalaze između naredba „BEGIN BATCH" I „END BATCH". Koordinator neretko ne poseduje lokalne replike traženih podataka, pa je prinuđen da dopre do više čvorova u klasteru, čime ujedno postaje i usko grlo, budući da upravlja saobraćajem za više različitih upita do više različitih čvorova. Batch operacija može da otkaže ukoliko nema dovoljno aktivnih/raspoloživih čvorova. Izuzetak su Batch operacije gde se sve operacije upisuju u jednu jedinu particiju. U ovom slučaju, samom koordinacijom i operacijama ažuriranja može se efikasno upravljati na jednom čvoru Cassandre.

Logovane Batch operacije su 30% sporije od nelogovanih Batch operacija, ali zato predstavljaju jedno odlično rešenje za sprovođenje konzistentnosti u slučaju ažuriranja denormalizovanih podataka. Na primeru „MovieCritic" baze podataka, ukoliko administrator želi da promeni naziv nekog filma, pri čemu naziv filma predstavlja redudantnu kolonu koja postoji u tabelama „Film", „OdgledaniFilmovi" i „OmiljeniFilmovi", logovana Batch operacija u pozadini bi isto sprovela u delo. U ovom slučaju, potreba za atomičnošću nadmašuje probleme performansi, tako da grupisanje više upita ažuriranja zajedno u okviru jedne logovane Batch operacije predstavlja jednu dobru odluku. Primer Batch operacije i njenih upita prikazan je na narednoj slici.

Slika 9.1. *Batch operacije i njei upiti*

Materialized views u Cassandri

Kod podataka visoke kardinalnosti, gde sekundarni indeksi ne predstavljaju pogodno i preporučivo rešenje, Materialized views (pogledi) pokazali su se kao prikladno rešenje. Ovi pogledi u pozadini predstavljaju standardne tabele, kod kojih svaki pristup prolazi kroz uobičajen process obrade upita za pisanje/čitanje. Cassandra kreira odvojene SSTable-ove za osnovnu tabelu i za sam pogled (view). Podaci u ovim pogledima su raspoređeni serijski na osnovu primarnog ključa pogleda (views).

Restrikcije kod ovakvih pogleda:

- Pogled mora da sadrži sve delove primarnog ključa osnovne tabele kao delove svog primarnog ključa, pri čemu istom može da pridruži maksimalno jednu novu kolonu.
- Pogled izuzima sve one redove koji za novododatu kolonu u primarnom ključu nemaju konkretnu vrednost (poseduju null vrednost).

Evidentno je da kod ovih pogleda Cassandra mora da održava konzistentnost između osnovne tabele i pridruženih pohleda, što ima svoju cenu. Budući da pogledi prestavljaju Cassandrine tabele, pisanje u bilo koju osnovnu tabelu koja poseduje pridružene poglede podrazumeva sledeće:

- Zaključavanje cele particije
- Čitanje trenutnog sadržaja particije
- Preračunavanje izmena koje treba ispratiti u pridruženim pogledima
- Kreiranje batch-a sa grupisanim operacijama za izmene u osnovnoj tabeli i njenim pridruženim pogledima, kako bi se osiguralo da ukoliko izmene nad osnovnom tabelom prođu uspešno, onda se iste isto reflektuju i kod pridruženih pogleda
- Izvršavanje batch operacije

Na ovaj način Cassandra osigurava konzistentno stanje podataka u svim pridruženim pogledima.

Na narednoj slici prikazan je primer kreiranja pogleda „OceneFilmova" pridruženog osnovnoj tabeli „Ocene" kako bi se informacije o ocenama koje su dodeljivane filmovima mogle pretraživati i po koloni „ocena" koja ne predstavlja deo primarnog ključa u tabeli „Ocene".

Cassandra 3.11.12 : MovieCritic : SQL Query

```
1
2
3  CREATE MATERIALIZED VIEW OceneFilmova AS
4      SELECT "sifraFilma", "username", "password", "ocena"
5      FROM "MovieCritic"."Ocene"
6      WHERE "sifraFilma" IS NOT NULL AND "username" IS NOT NULL AND "password" IS NOT NULL AND "ocena" IS NOT NULL
7      PRIMARY KEY (("sifraFilma", "ocena"), "username", "password");
8
9  SELECT * FROM "MovieCritic".OceneFilmova WHERE "sifraFilma" = '1058' AND ocena = 10;
10
```

sifraFilma	ocena	username	password
1058	10	jelena	novkovic81
1058	10	tanja	novkovic77
1058	10	teodora	novkovic97

Slika 10.1. *Primer kreiranje view-a*

Zaključak

Apache Cassandra baza podataka podržava veliki broj opcija za zadavanje upita. Stuktura baze podataka kao i struktura samih upita, utiču na to kako će teći njihova obrada. Postupak obrade upita kreće zadavanjem i slanjem upita iz aplikacije. Proverava se ispravnost upita – sintaksno i semantički – i vraća se greška ukoliko upit nije ispravno napisan.

Kod upita čitanja sam upit se šalje nasumičnom čvoru koji prosleđuje isti svim čvorovima koji poseduju podatak. Radi uspešne obrade upita Cassandra kombinuje rezultate MemTable-a i više potencijalnih SSTable-ova, upoređivajući ih po vremenskoj oznaci kako bi vratila najskoriji rezultat. Pre pretraživanja SSTable-ova Cassandra pretražuje da li je podatak eventualno prisutan u kešu redova, a zatim preko Bloom filtera identifikuje potencijalne SSTable-ove za pretragu. Kako bi proces čitanja bio što efikasniji, Cassandra izbegava sekvencijalno skeniranje celokupnog sadržaja SSTable-ova i najpre proverava keš ključeva za odgovarajući SSTable tražeći indeks particije za odgovarajući particioni ključ. Ukoliko se particioni ključ ne nalazi u kešu, onda se pretražuje prostor na disku koji za sve particione ključeve poseduje indekse njihovih particija. Radi brže pretrage ovog prostora na disku Cassandra koristi Partition Summary koji grupiše svakih N particionih ključeva i mapira lokaciju grupe u odgovarajućem fajlu. Kada se odredi indeks željene particije onda se pomoću kompresione mape ofseta nalazi sama lokacija bloka na disku gde se nalazi podatak. Nakon toga se sekvencijalnim skeniranjem particije odgovarajućeg SSTable-a dolazi do željenog podatka.

Kod upita upisivanja čvor koordinator identifikuje koji čvor je zadužen za upis podataka. Nakon što bude upućen na određeni čvor, zahtev za pisanje prvo stiže do Commit log-a, u kojem se čuvaju keširani upisi i koristi se u slučajevima neuspelih upisa ili slučajevima otkazivanja čvorova. U isto vreme, podaci se upisuju i u Memtable memoriju u kojoj se skladište particije podataka koje Cassandra pretražuje po ključu. Cassandra smatra upis uspešno kompletiranim kada se podatak upiše i u Commit log i u Memtable, iako se realan upis na disk dešava kasnije. Flush operacijom, Cassandra ispušta podatke iz keša na disk u odgovarajuće SSTable-ove, odnosno fizičke fajlove. Cassandra povremeno u pozadini izvodi kompakciju više SSTable-ova u jedan SSTable, radi optimizovanja operacija čitanja. Istovetni proces upisa se obavlja i u čvorovima u kojima se upisuju replike. Nakon što čvor upiše podatke, on obaveštava čvor koordinatora o uspešno završenoj operaciji.

U kom trenutku će Cassandra potvrditi da je upit uspešno kompletiran zavisi od podešene strategije konzistentnosti („any", „one", „quorum", „local quorum", „each quorum", „all").

Literatura

1. Knjiga - Cassandra: The Definitive Guide, Eben Hewitt

2. Knjiga – Beggining Apache Cassandra Development – Discover all aspects of using Apache Cassandra in applications, Vivek Mishra

3. Knjiga – Data Modeling in Apache Cassandra – Five steps to an Awesome Data Model, DataStax

4. Knjiga – CQL for Cassandra – Documentation, DataStax

5. Knjiga - Cassandra: The Definitive Guide: Distributed Data at Web Scale

6. https://docs.datastax.com/en/cql-oss/3.3/index.html

7. https://medium.com/jorgeacetozi/cassandra-architecture-and-write-path-anatomy-51e339bcfeoc
8. https://learning.oreilly.com/library/view/learning-apache-cassandra/9781787127296/e46bfd7a-7b8f-4197-af92-6675a8a9a5d7.xhtml