

UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET



Luka B. Đorović

ANALIZA SLUČAJEVA UPOTREBE
RELACIONIH I KOLONSKI ORIJENTISANIH
NERELACIONIH BAZA PODATAKA

master rad

Beograd, 2024.

Mentor:

dr Saša MALKOV, vandredni profesor
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Ana ANIĆ, vanredni profesor
University of Disneyland, Nedodija

dr Laza LAZIĆ, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: 15. januar 2016.

Ovaj rad posvećujem...

Naslov master rada: Analiza slučajeva upotrebe relacionih i kolonski orijentisanih nerelacionih baza podataka

Rezime:

Ključne reči: analiza, geometrija, algebra, logika, računarstvo, astronomija

Sadržaj

1	Uvod	1
2	Modeli za upravljanje podacima	4
2.1	Relacioni model	4
2.2	Kolonski-orijentisani model	6
2.3	Glavne razlike između relacionog i kolonski-orijentisanog modela . .	12
3	Slučajevi upotrebe	18
3.1	Opis i sadržaj eksperimenta	18
3.2	Primena u online transakcionom procesiranju (OLTP)	24
3.3	Primena u online analitičkom procesiranju (OLAP)	24
3.4	Primena u distribuiranom okruženju	25
4	Zaključak	26
	Bibliografija	27

Glava 1

Uvod

Podaci su najstabilniji deo svakog sistema. Oni su reprezentacija činjenica, koncepata jednog sistema kao i instrukcija u formalizovanom stanju spremnom za dalju interakciju, interpretaciju ili obradu od strane korisnika ili mašine. Iako kroz svoju istoriju računarstvo važi za oblast koja uvodi nove tehnologije i alate neverovatnom brzinom to nije slučaj za svaku njenu granu. Postoje oblasti koje se kroz istoriju nisu menjali, ili su se slabo menjali i proširivali. Primera za to ima puno i oni su uglavnom usko vezani za funkcionalne principe koji se prožimaju kroz računarske mreže, kompilatore, operativne sisteme, sisteme za upravljanje podacima itd.

Kada je reč o istoriji sistema za upravljanje podacima, izdvojio bih tri glavne faze: vreme pre relacionih sistema, vreme neprikosnovene vladavine relacionih sistema i nastanak alternativa relacionim sistemima pod grupnim nazivom NoSQL.

Do 70ih i nastanka relacionih sistema za upravljanje podacima, rukovanje podacima izvodilo se kroz pisanje i čitanje sa fajl sistema operativnog sistema. Rukovanje većim količinama podataka nije bilo standardizovano ni na koji način već su se konvencije uvodile na nivou organizacija. Apolo sletanje na Mesec realizovano je koristeći ovakav vid rada sa podacima, što ovaj poduhvat čini utoliko neverovatnim.

S obzirom da je ovaj vid rada sa podacima imao mnogobrojne mane, među kojima je jedan od glavnih bio težak pristup podacima, javile su se potrebe za unapređenjem. Najuspešniji je bio Edgar F. Codd ¹ koji je 1970. godine objavio rad pod imenom „A Relational Model of Data for Large Shared Data Banks” kao rezultat istraživanja i sopstvenih teorija o organizaciji podataka. Kao dokaz da je

¹Edgar Frank „Ted” Codd (19 Avgust 1923 – 18 April 2003) Američki računarski naučnik

njegov model moguće implementirati pokrenut je System R, čiji je rezultat bio i pojava SQL-a (Structured Query Language) kao standardizovanog jezika za rad sa podacima. Nakon toga pojavili su se Oracle i IBM sa svojim komercijalnim proizvodima za upravljanje relacionih baza podataka. Naredni period obeležio je rad sa podacima koristeći relacioni model.

XXI vek doneo je sa sobom ubrzanu digitalizaciju, povećanu dostupnost interneta, samim tim pojavila se potreba za obradom veće količine podataka. Sve ovo je pokazalo pojedine slabosti dosadašnjih sistema zasnovane na relacionim modelima, koji nisu mogli u svim segmentima da odgovore na zahteve modernog doba. Ovi problemi obično su poznati pod grupnim imenom „problemi velikih podataka” (BigData problems). Došlo je do pojave niza novih modela i principa za čuvanje podataka, a svi pod grupnim imenom „NoSQL” (ili nerelacione) baze podataka. Sistematizovanje ogromne količine fizičkog prostora na disku na kojem se podaci mogu čuvati i kasnije koristiti, kao i fleksibilnost strukture podataka sa kojima se radi, jesu glavni aktuelni problemi tog vremena na koje su se fokusirale tehnologije nastale u NoSQL pokretu. Decenije vladavine relacionih sistema za čuvanje podataka ostavile su dubok trag u praksama rada sa podacima, i sa razlogom predstavljaju defakto standard i dan danas, te je zaočekivati postojanje doze skepse pri korišćenju tehnologija nastalih u ovoj fazi.

Kao važna grupa nerelacionih izdvajaju se kolonski-orijentisane baze podataka. One su uvele tada nekonvencionalne koncepte čuvanja podataka po kolonama. Dakle fizički na disku skladištile su se vrednosti jedne kolone jedna do druge, sa referencom na red kojem pripadaju. To sa sobom vuče razne mogućnosti za optimizaciju ali i novih pristupa modelovanja i organizacije podataka. Ovakav način skladištenja ispitavan je još davnih 70ih godina XX veka, međutim u ranim godinama XXI veka došlo je do obnove interesovanja u akademskim ali i industrijskim krugovima.

Nijedan od navedenih koncepata nije univerzalno rešenje, zato je bitno postojanje materijala koji se bave analizom slučajeva upotrebe tih tehnologija. Pored teorijske analize koja se može pronaći u relevantnim javnim dokumentacijama korisno je imati i konkretne implementacije testova čiji se rezultati mogu iskoristiti kako bi se povukle paralele u skladu sa potrebama realnih sistema.

Cilj ovog rada je analiza i upoređivanje slučajeva upotrebe relacionih i kolonski orijentisanih baza podataka. Rad će se sastojati iz teorijskog opisa navedenih tehnologija kao i opisa konkretnih predstavnika baza podataka koji će biti kori-

šćeni. Na osnovu teorijskih izbora i istraživanja biće analizirani različiti slučajevi upotrebe.

Glava 2

Modeli za upravljanje podacima

2.1 Relacioni model

Opšte karakteristike

Relacioni model je najpopularniji model za rad sa podacima. On podatke kao i veze izmedju njih predstavlja kroz skup relacija. Kao fundamentalna ideja iza relacionog modela stoji tabelarni prikaz podataka, sto uvecava njegovu intuitivnost. Da bi jedna tabela bila validna relacija u relacionom modelu ona mora ispunjavati sledeće uslove:

- Presek kolone i vrste jedinstveno određuje vrednosnu ćeliju.
- Sve vrednosne ćelije jedne kolone pripadaju nekom zajedničkom skupu.
- Svaka kolona ima jedinstveno ime.
- Ne postoje dve identične vrste jedne tabele.

Iako ovakva formalizacija relacije deluje intuitivno (usled istorijskog uticaja koji je relacioni model ostavio na vizualizaciju organizacije podataka) ona je neophodna za definisanje složenijih pojmova.

Koncept ključa relacionog modela

Usled jedinstvenosti svake vrste relacije, jasno je da mora postojati skup kolona za koji važi da nikoja dva reda te relaciju nemaju identične vrednosti za svaku kolonu iz tog skupa. Takav skup se naziva *superključ* relacije. Minimalan superključ

naziva se *ključ kandidat*. Svaka relacija može imati više ključeva kandidata, ali samo jedan od njih je *primarni ključ* koji mora imati definisanu vrednost za svaku njegovu kolonu. *Strani ključ* je kolona ili skup kolona čije vrednosti predstavljaju referencu na određeni red neke druge relacije. Primarni i strani ključ igraju veliku ulogu u očuvanju integriteta baze podataka o čemu će biti reči u nastavku.

Integritet relacionog modela

Integritet relacionog modela predstavlja „istinitost” podataka koji se čuvaju u bazi, što je njena i najveća vrednost. Čuvanje integriteta baze posebno je važno prilikom invazivnih operacija kao što su dodavanje reda, izmena reda ili brisanje reda u tabeli. Postoji više vrsta integriteta u relacionom modelu: *integritet entiteta*, *integritet domena*, *integritet neposojce vrednosti* i *referencijalni integritet*.

Integritet entiteta kaže da svaka vrsta jedne relacije predstavlja jedan entitet i da kao takva ne može u okviru primarnog ključa, koji taj entitet identifikuje, imati nedefinisanu ili nepostojeću vrednost.

Integritet domena nameće shemu po kojoj svaka kolona može uzimati vrednost iz unapred dodeljenih skupova vrednosti.

Integritet nepostojece vrednosti govori o eventualnim kolonama čije vrednosti ne mogu kao vrednost imati nepostojecu vrednost kako se ne bi narušila uspostavljen na biznis logika.

Referencijalni integritet nalaže da se svaki strani ključ jedne tabele, ukoliko je definisan, mora poklapati sa nekim od primarnih ključeva uparene relacije.

PostgreSQL

PostgreSQL je objektno-relacioni sistem za upravljanje bazama podatak nastao kao potomak POSTGRES-a, proizvoda koji je nastao, a kasnije i razvijan na Berkliju, Univerzitet Kalifornija. PostgreSQL je otvorenog koda sa velikom SQL podrškom kao i modernim funkcionalnostima poput: kompleksnih upita, okidača, izmenjivih pogleda, transakcionog integriteta i mnogih drugih. Postgres nudi širok spektar proširenja od strane korisnika poput dodavanja novih tipova podataka, funkcija, operatora, agregatnih funkcija itd.

Kao takav, PostgreSQL je pogodan sistem za čuvanje najkompleksnijih podataka i veza između njih. Mogućnost kreiranja procedura na samoj bazi u integrisanoj SQL sintaksi, daje široke mogućnosti optimizacije aplikacija.

PostgreSQL koristi server-klijent model funkcionisanja. Sastoji se iz serverskog i klijentskog dela procesa. Serverski deo rukuje fajlovima baze podataka, prihvata konekcije, izvršava konkretne operacije nad bazom. Klijentski deo predstavlja aplikaciju kojom korisnik može da komunicira i rukuje podacima na serverskom delu. Klijent i server komunkiraju preko TCP/IP protokola. Serverski deo može raditi sa više konekcija istovremeno tako što svaka klijentska konekcija radi kao zaseban proces.

Postgres iza sebe ima razvijenu društvenu zajednicu, pa samim tim ima dosta izvora i dokumentacije koje mogu olakšati učenje ovog sistema. [4]

2.2 Kolonski-orijentisani model

Opšte karakteristike

[1] Susret sa Big Data problemima dovelo je do potreba za tabelama koje imaju ogroman broj kolona, i ogroman broj redova u okviru tih tabela. Jasno je da nam je za potrebe različitih analitika potreban različit skup kolona. Novonastali zahtevi ukazali su na problem kod postojećih relacionih modela. Svaki upit nad tabelom podrazumevao je dohvatanje svih kolona jednog reda, gde bi se filtriranje nepotrebnih kolona izvršavalo nakon što su se sve kolone učitale u memoriju. Ovo je bila samo jedna od motivacija za implemetanciju sistema zasnovanih na kolonski orijentisanom modelu koji je dizajniran tako da ovakav problem izbegne i uz to donese i druga poboljšanja o kojima će biti reči u nastavku.

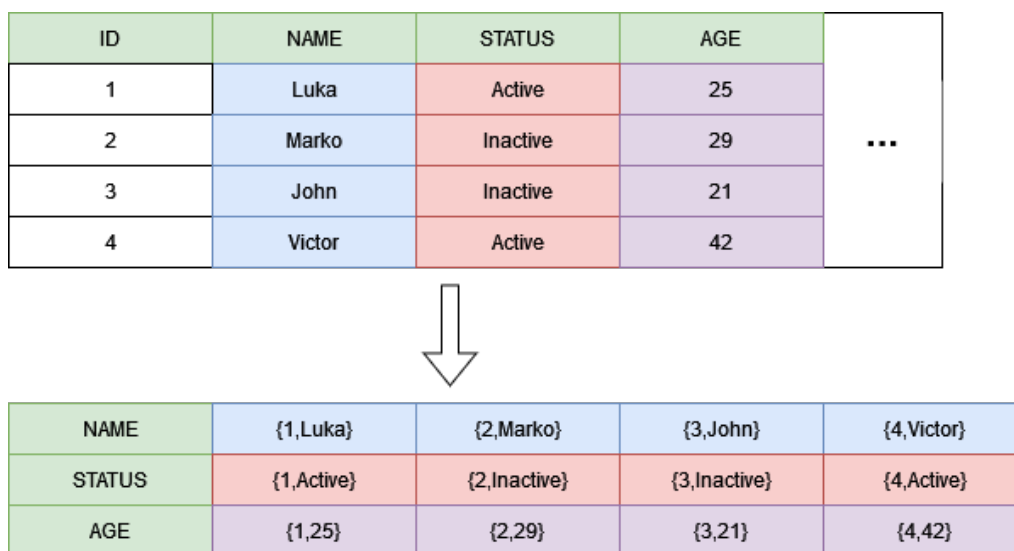
Kao što mu ime govori kolonski orijentisan model podatke na disku skladišti po kolonama a ne po redovima kao što je to slučaj kod relacionh modela. U prevodu, sve vrednosti kolone svih redova skladište se jedna do druge, a na konkretnu vrednosnu ćeliju referiše se pomoću ključa konkretnog reda kao i kolone čiju vrednost želimo da pročitamo. Ovakav dizajn doveo je do toga da za dohvatanje određenog skupa kolona nema potrebe da čitamo sve vrednosti tog sloga, već je dovoljno da znamo konkretan ključ tog reda kao i imena kolona čije vrednosti želimo da pročitamo.

Ovakav vid skladištenja sa sobom nosi veliki potencijal za primenu raznih algoritama za kompresiju podataka. Kompresija nad sličnim podacima koji se nalaze na uzastopnim adresama u memoriji, omogućava izbegavanje čuvanja složenih meta informacija u okviru struktura koje se koriste za tu kompresiju što ovaj model

čini posebno pogodnim za njihovu primenu.

Nedostajanje unapred definisanog skupa kolona, ukida smisao čuvanja nepostojeće vrednosti (null) s obzirom da ukoliko red nema vrednost neke kolone, nema potrebe za upisivanjem bilo koje vrednosti te kolone za taj red.

Kolonski orijentisan model kao i većina ostalih nerelacionih modela, nudi fleksibilnost sheme. To kao posledicu ima da eventualna promena strukture podataka neće bitno uticati na unapred definisanu shemu, kao ni iziskivati dodatnu migraciju podataka, kao što bi to bio slučaj kod relacionog modela. Osim toga fleksibilnost sheme se ogleda i u tome što je broj kolona jednog vektora neograničen, što daje dosta prostora za eksperimentisanje sa dizajnom baze. Primer toga kako ovakvo svojstvo modela može doprineti dostizanju prednosti pri analitičkom sistemu možete videti na slici SLIKA 1.



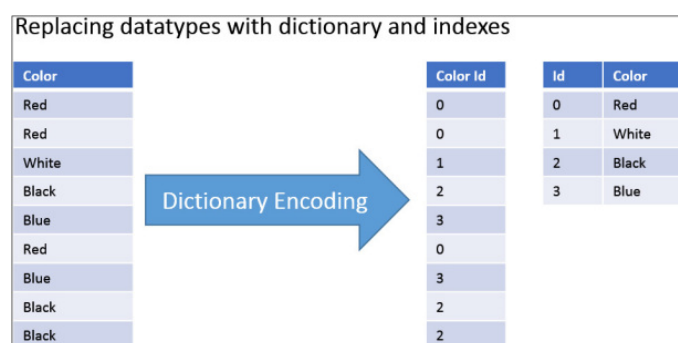
Slika 2.1: Kolonski orijentisan format

Popularni algoritmi kompresije kolonski orijentisanog modela

[2]

Enkodiranje zasnovano na rečniku

Enkodiranje zasnovano na rečniku (Dictionary based encoding) jeste tehnika kompresije podataka koja se može primeniti na vrednosti jedne kolone ili skupa kolona. Najefikasnija je nad kolonama koje imaju mali skup mogućih vrednosti. Rade tako što se u memoriji sačuvaju sve moguće vrednosti te kolone i svakoj od njih se dodeli ključ. Veličina ključa je direktno zavisna od kardinalnosti skupa vrednosti koje se mapiraju. Svaki unos ili izmena vrednosti kolone u konsultaciji sa postojećim rečnikom radi enkodiranje pristigle vrednosti, a svako dohvaćanje vrednosti radi dekodiranje sačuvane vrednosti. Ovim se izbegava ponavljanje velikih podataka tako smanjujući potrebnu memoriju na disku. Uglavnom su pogodne primene nad kolonama sa statičkim i opisnim podacima koji se ponavljaju.

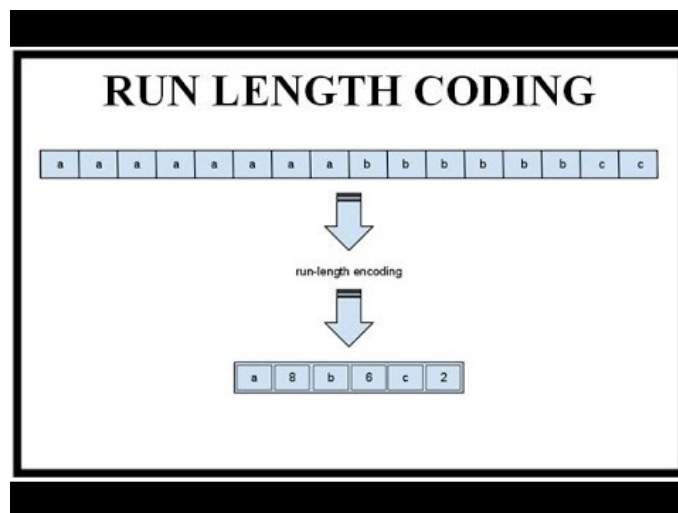


Slika 2.2: Enkodiranje zasnovano na rečniku

Enkodiranje po broju ponavljanja

Enkodiranje po broju ponavljanja (Run Length Encoding) je jednostavan mehanizam za kompresiju podataka pogodan kompresiju kolona kod koje se vrednosti ponavljaju. Funkcioniše tako što kada se nađe na vrednost koja se ponavlja, ne skladišti duplikate već sačuva tu vrednost jednom a dodatno kao meta informaciju prosledi koliko puta se ta vrednost ponavlja. Takav vid optimizacije najkorisnije je očigledno kada su vrednosti sortirane, a s obzirom da su vrednosti kolona u kolon-

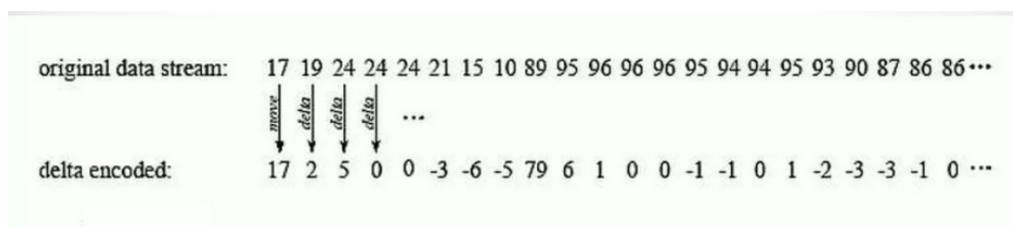
ski orijentisanim bazama jedna do druge to otvara prostor za ovaj vid kompresije podataka kako bi se umanjilo zauzeće prostora.



Slika 2.3: Enkodiranje po broju ponavljanja

Delta enkoding

Delta enkoding je mehanizam za optimizaciju prostora baze podataka koji se zasniva na čuvanju razlike između objekata a ne celih vrednosti. Primera za upotrebu ima dosta a jedan od najčešćih je slučaj datumskih kolona, gde će nam referentna vrednost biti neki konkretan datum, a vrednosti ostalih kolona će biti čuvane kao razlika u odnosu na njega. Kao što je rečeno pogodna je za datumske vrednosti ali postoje i drugi tipovi, kao što su numerički, kod kojih ovaj vid kompresije može doneti unapređenja u vidu slobodnog prostora na disku.



Slika 2.4: Delta enkoding

HBase

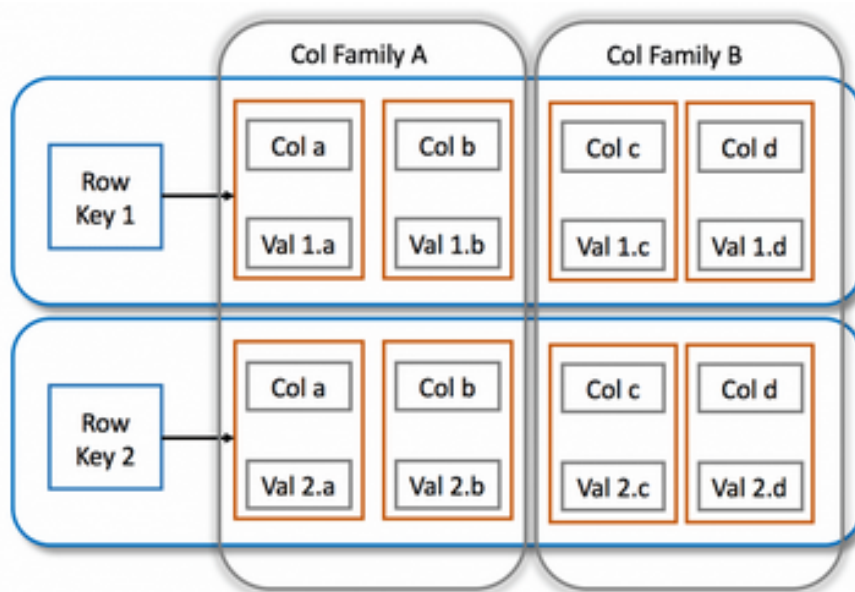
HBase je distribuirana kolonski orijentisana nerelaciona baza podataka pisana u javi. Nastala je 2007 kao prototip BigTable baze koja je modelovana u okviru Google-ovog članka 2006 [3].

Model podataka koji HBase koristi podrazumeva da se kolone svakog reda grupisu u familije tzv familije kolona. Cilj je da kolone koje su po prirodi slične pripadaju istoj familiji kolona, kako bi se nad njima mogli primeniti algoritmi kompresije, s obizrom da će se kolone koje pripadaju jednog familije nalaziti u okviru jednog HFile fajl-a koji se sladišti na disku.

HBase ne podržava postojanje indeksa, međutim podrazumevano ponašanje je da su ključevi svih redova sortirani rastuće, tako da se koristi binarna pretraga za pretragu vredosnti po ključu reda kojem pripada. Ovo daje na važnosti strategiji pro dizajniranju ključa. Poželjno je da svaka pretraga ide direktno preko ključa ili njegovog prefiksa.

Vrednosnu ćeliju u HBase tabeli određuje ključ reda, ime kolone te vrednosne ćelije, kao i familija kojoj kolona pripada.

HBase nije ACID baza podataka, ali garantuje konzistentnost u radu sa jednim redom.



Slika 2.5: Hbase model

Arhitektura HBase klastera sastoji se iz dve glavne komponente: master ser-

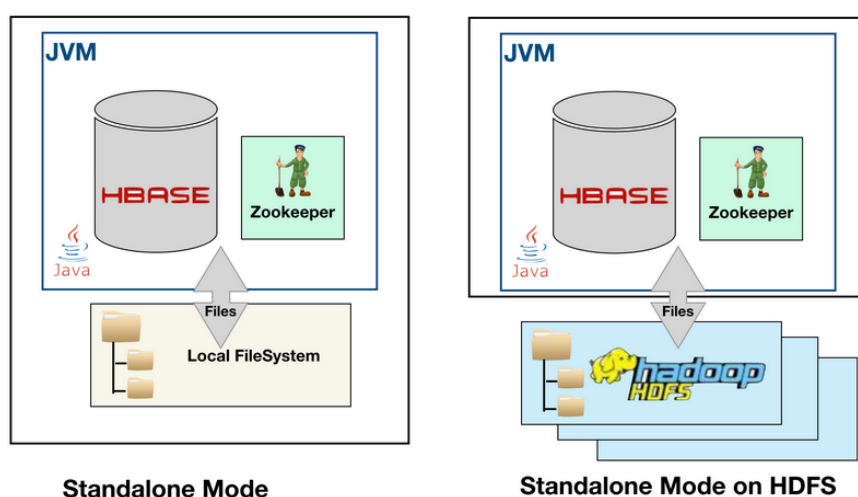
ver i region server. Ove komponente je najbolje opisati kroz interfejsse koje oni implementiraju.

Region server implementira HRegionInterface, odnosno implementira servise koji se bave operacijama nad podacima i održavanjem i upravljanjem regiona. Jedan region čine podaci za redove u nekom rasponu vrednosti ključeva. Region server kreira i dohvata HFile fajlove koji se čuvaju na disku. HBase dozvoljava dva režima: samostalan i distribuirani režim.

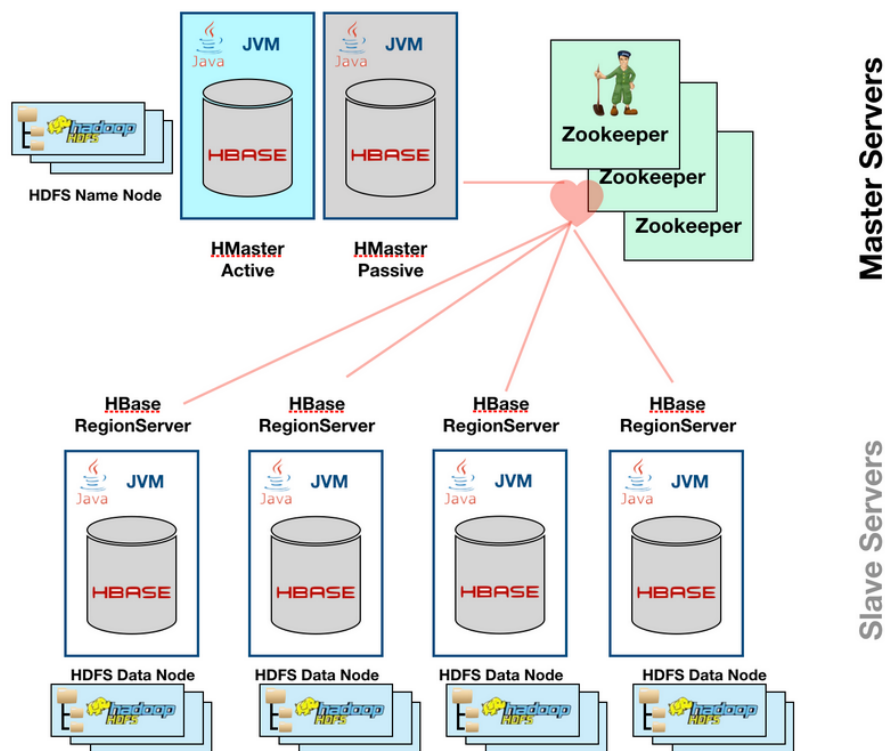
Samostalan režim HFile fajlove može čuvati na lokalnom fajl sistemu i korišćenje HDFS-a je opciono.

U distribuiranom režimu sa druge strane neophodno je korišćenje HDFS-a za skladištenje. Region server možemo zamisliti kao indeks za dohvaćanje HFile-ova sa sa diska. Svaka izmena koja treba da se izvrši na disku prvo se upisuje u WAL (Write ahead log), a nova izmenjena vrednost se upisuje u MemStore fajl. Kada se MemStore fajl upuni tek tada se izmene flush-uju u jedan HFile koji dalje ide na HDFS.

HMaster implementira HMasterInterface koji sadrži servise koji rade sa metainformacijama o tabelama, familijama kolona, kao i regiona. Uloga Mastera je da za svaki konkretan row id može da odredi koji region odnosno region server treba da bude pročitao kako bi se izvršila odgovarajuća operacija. Za postizanje navedene funkcionalnosti HMaster koristi Zookeeper [3] servis. Pored toga master servis ima pozadinske procese koji regulišu rad load balansera i sadržaj hbase:meta tabele.



Slika 2.6: HBase standalone



Slika 2.7: HBase distributed

2.3 Glavne razlike između relacionog i kolonski-orijentisanog modela

Normalizacija i denormalizacija

Da bi se stanje u bazi olakšalo čuvanje podataka konzistentnim u relacionim modelima često se radi na izbegavanju *redudantnosti* u podacima. Redudantni podaci zauzimaju višak prostora na disku i otežavaju kasnije održavanje sistema. Kako bi se izbegla redudantnost postoji jasno definisani postupci koji nam pomažu da organizujemo podatke tako da redudantnost umanjimo. Proces unapređivanja logičkog dizajna baze tako da rešava problem redundantosti podataka, ali ne po cenu očuvanja integriteta, naziva se normalizacija. Teorija o normalizaciji se zasniva nad konceptima normalnih formi iz matematičke logike. U zavisnosti od toga koja pravila zadovoljava određena relacija, dodeljuje joj se normana forma. Trenutno postoji 5 defnisanih normalnih formi.

1. Prva normalna forma (1NF)

- Svaka relacija mora imati primarni ključ koja jedinstveno određuje svaku vrstu.
- Svaka kolona mora sadržati atomičnu (nedeljivu) vrednost.
- Sve vrednosti jedne kolone pripadaju nekom zajedničkom skupu.

2. Druga normalna forma (2NF)

- Relacija mora biti 1NF
- Sve vrednosti kolone koje ne pripadaju superključu relacije, direktno su određene celim primarnim ključem .

3. Treća normalna forma (3NF)

- Relacija mora biti 2NF
- Nijedna kolona van ključa kandidata nije jedinstveno određena drugim kolonama koje ne pripadaju ključu kandidatu.

4. Boyce-Codd normalna forma (BCNF)

- Relacija mora biti 3NF.
- Svaka kolona koja ne pripada ključu mora biti jedinstveno određena vrednostima superključa relacije.

5. Četvrta normalna forma (4NF)

- Relacija mora biti BCNF.
- Regulisanje zavisnosti među kolonama tako da nema ponavljanja podataka.

6. Peta normalna forma (5NF)

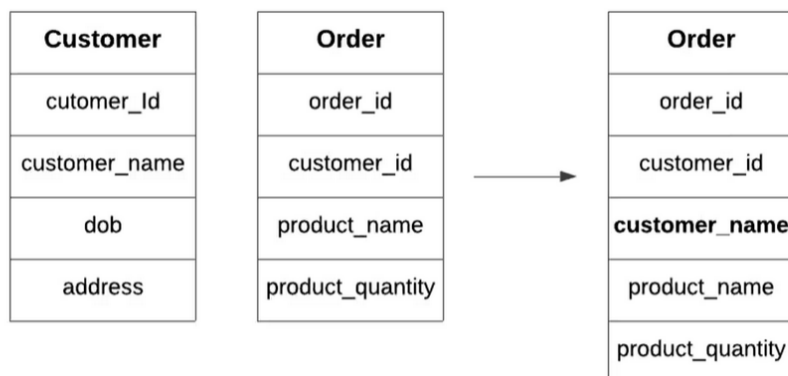
- Relacija mora biti 4NF.
- Regulise zavisnosti između relacija tako da se izbegne potreba za komplikovanim upitima za dohvaćanje podataka.

Modeli koji su bili podloženi normalizaciji obično raspolažu velikim brojem referencijalnih ključeva u tabelama koji predstavljaju referencu ka originalnim podacima iz matične tabele u kojoj se nalaze. Ukoliko želimo da pristupimo celom podatku iz neke tabele koja referiše na njega, podrazumeva se da ćemo pročitati

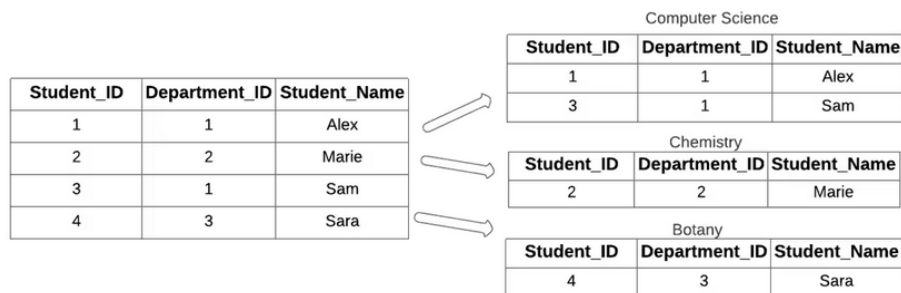
celu tabelu na koju ta referenca pokazuje, što povećava broj operacija čitanja sa diska samim timmože uticati na performanse.

Denormalizacija je strategija kod modela gde je neophodno ubrzati operacije čitanja podataka, odnosno umanjiti broj tabela kojima je neophodno pristupiti kako bi se neki skup podataka pročitao iz baze podataka.

Neke od tehnika denormalizacije su dodavanje redundantne kolone, horizontalna podelela tabele, vertikalna podela tabele, uvođenje izvedene kolone. Pre join kolone je dodavanje kolone tabeli čija se referenca na vrednost te kolone često koristi. Horizontalno deljenje tabele podrazumeva da se na osnovu prirode podataka jedna tabela podeli na više tabela tako da se čitanje svede samo na čitanje grupe redova. Vertikalno deljenje podrazumeva da se tabela podeli grupisanjem kolona koje se često čitaju zajedno. Uvođenje izvedene kolone predstavlja dodavanje kolone koja predstavlja vrednost neke agregatne funkcije, time se izbegava da se pri čitanju ta operacija izvršava, već se radi prosto čitanje.



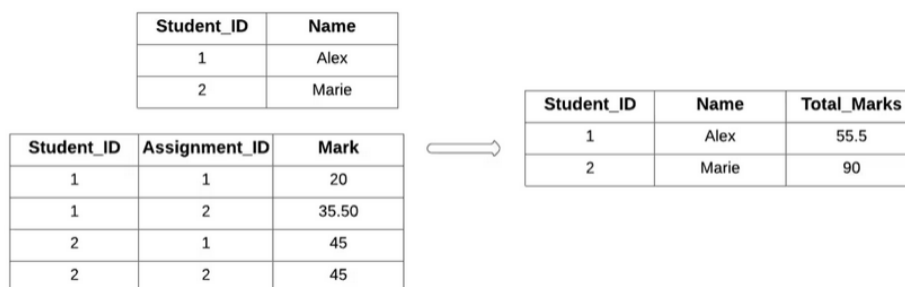
Slika 2.8: Pre join kolone



Slika 2.9: Horizontalna podela tabele



Slika 2.10: Vertikalna podela tabele



Slika 2.11: Uvođenje izvedene kolone

ACID i BASE

ACID (Atomicity, Consistency, Isolation, Durability) svojstva služe kao garancija tačnosti i konzistentnosti podataka prilikom konkurentnom pristupu.

Atomicity se može objasniti pravilom: Jedna transakcija se izvršava u celini ili se ne izvršava nijedan njen deo. U prevodu, dejstvo transakcije je nedeljivo.

Durability garantuje da će kompletirana transakcija u slučaju prekida rada sistema pre nego što su izmene reflektovane na disk, biti upamćena i izvršena nakon restarta sistema. Svaka izmena se upisuje u log fajl pre nego što je reflektovana na disk, kako bi se operacije mogle poništiti u slučaju poništavanja transakcije.

Consistency se čuva od strane korisnika. Bitno je da korisnik koji pokreće transakciju vodi računa o tome da stanje podataka ostane u konzistentom stanju.

Isolation svojstvo nalaže da se transakcije međusobno izolovane tako da izvršavanje jedne transakcije ne može uticati na izvršavanje druge. Ovo se obezbeđuje pomoću scheduler-a od strane samog sistema za upravljanje bazom podataka.

BASE (Basic Availability, Soft state, Eventual consistency) je skup svojstava koje je definisao Eric Brewer , a koja su nastala usled želje da se formalizuju svojstva koja u Big Data svetu garantuju da je baza pogodna za horizontalno skaliranje a ujedno daje vid konzistentnosti koji je neophodan.

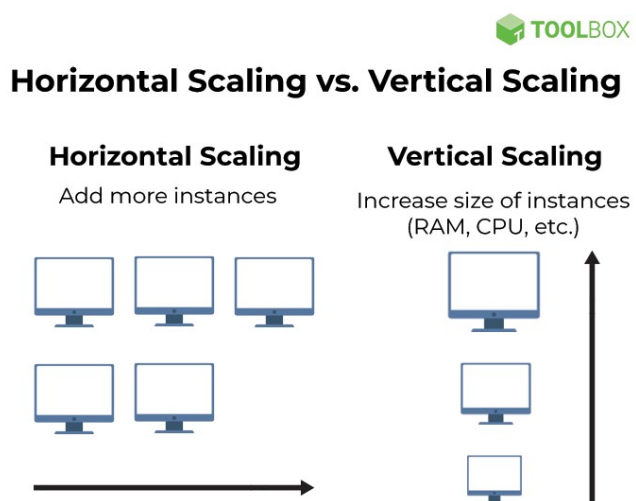
Basically available svojstvo kaze da ukoliko imamo klaster sa vise pojedicanih node-ova baze, problem sa jednim od njih nece spreciti da ostali node-ovi procesiraju zahteve i salju odgovore.

Soft state - svojstvo kaze da se stanje podataka moze menjati, cak i u trenucima kada nema spoljnih komunikacija sa bazom podataka. Ovo je posledica treceg svojstva.

Eventual consistency garantuje da ce baza podataka u periodima kada nema spoljne komunikacije sa klijentima, postati konzistentna kroz neki vremenski period.

Kolonski orijentisan model nacerlo ne mora da ispunjava BASE svojstva, ali to obicno jeste slucaj upravo zbog horizontalnog skaliranja koja ima tendenciju da ponudi.

Skalabilnost



Slika 2.12: Enkodiranje po broju ponavljanja

Glava 3

Slučajevi upotrebe

3.1 Opis i sadržaj eksperimenta

Analiza i upoređivanje slučajeva upotrebe bice realizovani na osnovu teorijskih i praktičnih izvora i istraživanja. Svaki primer će biti pracen eksperimentom koji će se sastojati od izvršavanja različitih vrsta postupaka. Kao platforma za realizaciju eksperimenata koriscen je host sa docker engine-om. Specifikacije Host-a data je na slici. Pokretanje svakog od eksperimenata je identicno. U okviru repozitorijuma nalazi se sav shell i java kod kao i uputstvo za pokretanje svakog od eksperimenta, zajedno sa deployment dijagramom.

Svakom eksperimentu dodeljen je precizno definisan kontekst radi uspostavljanja potpune kontrole okruženja u kojem se eksperiment realizuje. U svrhu definisanja konteksta eksperimenata delom su iskoriscene poznate specifikacije za benchmark bazi podataka. Konkretno za slučaj OLTP okruženja konsultovana je TPC-C specifikacija, za slučaj OLAP okruženja konsultovana je TPC-H specifikacija. Distribuirao okruženje je izuzeto.

Analiza rezultata eksperimenta sprovodi se kroz vise faza. Prva faza je upoređivanje složenosti , sto arhitekturne, sto shematske, realizacije konkretnog slucaja upotrebe kao i da li je konkretan slucaj upotrebe moguće realizovati sa postojecom tehnologijom. Druga faza je upoređivanje efikasnosti, koja podrazumeva upoređivanje vremena izvršavanja programa. Svaka od faza će uključivati tekstualnu diskusiju, slike kao i druge graficke prikaze ukoliko su pogodni.

Kako bi se postigao dovoljan dokaz koncepta (eng. proof of concept), ali i doslednost modernom vremenu, kategorije slučajeva upotrebe koji će biti obuhvaceni su:

1. Onlajn transakciono procesiranje (OLTP)
2. Onlajn analiticko procesiranje (OLAP)
3. Primena u distribuiranom okruženju

Platforma testiranja

Kao okruženje za izvršavanje eksperimenata korišćen je docker. Oba predstavnika bice pokrenuta u okviru nezavisnih kontejnera na jednom host-u sa docker engine-om. Svaki test predstavlja jedan java program koji se izvršava na konkretnom kontejneru, kako bi se izbegao bilo kakav vid spoljne komunikacije i kako bi se stekla što preciznija slika na osnovu rezultata merenja.

Svaki test imaće određene parametrike koji se postavljaju pri pokretanju testa. Parametri su broj iteracija/transakcija koje će test izvršiti kao i broj konekcija(odnosno clijenata) koji će paralelno u zasebnim nitima obavljati svoj deo posla.

Programi za OLAP i OLTP testiranje kreiraju se nezavisno s obzirom na kontekst (drugačiji model i test logika) u kojem se izvršavaju. Za OLTP i OLAP testiranje postoji po jedan maven artifakt kojeg će koristiti i HBase i Postgres jar-ovi koji predstavljaju kostur programa za testiranje. Svrha tog zajedničkog jar-a jeste da garantuje da će oba testa imati isti kostur, te da će se HBase i Postgres testovi razlikovati jedino po implementaciji odgovarajućih utility funkcija.

S obzirom da je u OLAP testu obuhvaćen segment za merenje bulk load-a podataka iz odgovarajućih csv fajlova, taj segment je takođe uključen u kostur testiranja.

Napomena: Kako su za predstavnike izabrani PostgreSQL i HBase, za rezultate merenja u nastavke treba uzeti u obzir da implementacija navedenih koncepata nije opsta za sve Relacione sisteme kao ni za sve kolonski orijentisane baze podataka.

docker-compose.yml

```
1 services:
2   postgres:
3     container_name: postgres
4     ports:
5       - "5433:5432"
6     environment:
```



```

7      - POSTGRES_PASSWORD=postgres
8      - POSTGRES_USER=postgres
9      - POSTGRES_DB=postgresdb
10     build:
11       context: .
12       dockerfile: ./Dockerfile_postgres
13     hbase-master:
14       image: blueskyareahm/hbase-base:2.1.3
15       command: master
16       ports:
17         - 16000:16000
18         - 16010:16010
19       volumes:
20
21     hbase-regionserver:
22       image: blueskyareahm/hbase-base:2.1.3
23       command: regionserver
24       ports:
25         - 16030:16030
26         - 16201:16201
27         - 16301:16301
28
29     zookeeper:
30       image: blueskyareahm/hbase-zookeeper:3.4.13
31       ports:
32         - 2181:2181

```

OLTPBenchmarkExecutor.java

```

1  default void executeOLTPWorkload
2    (BenchmarkUtility util, BenchmarkOLTPUtility oltpUtil,
3     int transNum, int clNum) {
4
5     List<Integer> transPerClientList = new ArrayList<>();
6     int transactionsToAssign = totalTransactions;
7     int transactionsPerClient = transactionsToAssign / clNum;
8
9     for(int i = 0; i<clNum;i++){
10       transPerClientList.add(transactionsPerClient);
11       transactionsToAssign-=transactionsPerClient;
12     }
13     if (transactionsToAssign > 0) {
14       int transactionForLastClient = transPerClientList.get(

```

```

15         clNum - 1);
16         transPerClientList.set(clNum - 1,
17             transactionForLastClient + transactionsToAssign
18             );
19     }
20
21     assert clNum==transPerClientList.size();
22
23     Thread[] threads = new Thread[clNum];
24     CountdownLatch latch = new CountdownLatch(numOfClients);
25     for(int i = 0; i<clNum; i++){
26         threads[i] = new Thread(
27             new BenchmarkSingleClientExecutor(
28                 util, oltpUtil,
29                 i*transPerClientList.get(i),
30                 transPerClientList.get(i),
31                 latch)
32             );
33     }
34
35
36     long startTimestamp = System.currentTimeMillis();
37     for(int i = 0; i<clNum; i++){
38         threads[i].start();
39     }
40     latch.await();
41     long endTimestamp = System.currentTimeMillis();
42     System.err.println("Total benchmark duration: " +
43         (endTimestamp-startTimestamp));
44 }

```

OLAPBenchmarkExecutor.java

```

1  default void executeBulkLoad
2      (BenchmarkUtility util, BenchmarkOLAPUtility olapUtil) {
3      long bulkLoadStart = System.currentTimeMillis();
4      olapUtility.bulkLoad(benchmarkUtility.connect());
5      long bulkLoadEnd = System.currentTimeMillis();
6      System.out.println("Bulk load duration: " +
7          (bulkLoadEnd-bulkLoadStart));
8  }
9
10 default void executeOLAPWorkload

```

```
11      (BenchmarkUtility util , BenchmarkOLAPUtility olapUtil ,
12         int iterNum , int clNum) {
13
14         List<Integer> itsPerClientList = new ArrayList<>();
15         int iterationsToAssign = iterNum;
16         int iterationsPerClient = iterationsToAssign / clNum;
17
18         for(int i = 0; i<clNum;i++){
19             itsPerClientList.add(iterationsPerClient);
20             iterationsToAssign-=iterationsPerClient;
21         }
22         if (iterationsToAssign > 0) {
23             int iterationsForLastClient = itsPerClientList.get(
24                                     clNum - 1);
25             itsPerClientList.set(clNum - 1,
26                                 iterationsForLastClient + iterationsToAssign
27                                 );
28         }
29
30         assert clNum==itsPerClientList.size();
31
32         Thread[] threads = new Thread[clNum];
33         CountDownLatch latch = new CountDownLatch(clNum);
34         for(int i = 0;i<clNum;i++){
35             threads[i] = new Thread(
36                 new BenchmarkSingleClientExecutor(
37                     util , olapUtil ,
38                     i*itsPerClientList.get(i) ,
39                     itsPerClientList.get(i) ,
40                     latch)
41             );
42         }
43
44
45         long startTimestamp = System.currentTimeMillis();
46         for(int i = 0;i<clNum;i++){
47             threads[i].start();
48         }
49         latch.await();
50         long endTimestamp = System.currentTimeMillis();
51         System.err.println("Total benchmark duration: " +
52             (endTimestamp-startTimestamp));
```

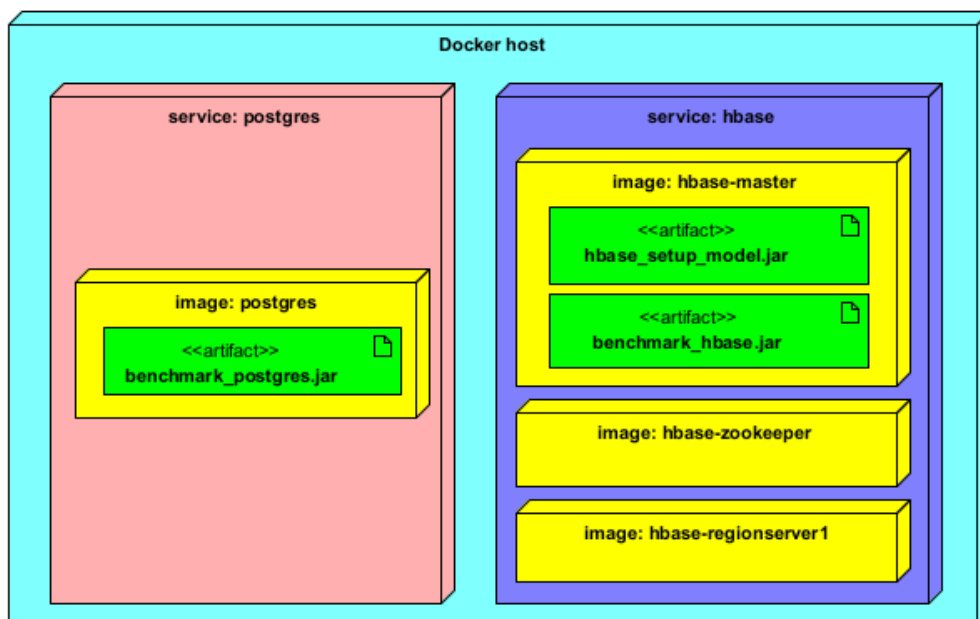
53 }

Priprema okruženja

Priprema okruženja za sprovođenje eksperimenata podrazumeva build-ovanje .jar fajlova za testiranje, pokretanje docker kontejnera , prebacivanje svih neophodnih resursa vezanih za kreiranje modela baze podataka i eventualnih resursa za bulk load vrednosti u tabele na odgovarajuće docker kontejnere.

prepareEnv.sh

```
1 #!/bin/bash
2 echo 'PREPARING ENVIRONMENT...';
3
4 rm -f ./hbase_setup_model.jar
5 rm -f ./benchmark_hbase.jar
6 rm -f ./benchmark_postgres.jar
7
8 export JAVA_HOME="$JAVA_8";
9 mvn -f ./hbase_setup_model clean compile assembly:single;
10 mvn -f ./benchmark_hbase clean compile assembly:single;
11
12 export JAVA_HOME="$JAVA_17";
13 mvn -f ./benchmark_postgres clean compile assembly:single;
14
15
16 docker-compose -f docker-compose.yml up --build -d;
17 docker exec -it hbase-master-1 sh -c "java -jar hbase_setup_model.jar";
```



Slika 3.1: Enkodiranje po broju ponavljanja

3.2 Primena u online transakcionom procesiranju (OLTP)

Online transakciono procesiranje obuhvata kratke, jednostavne, uchestale promene na relativno malom skupu podataka. Primer koji cemo koristiti jeste uopstena transakcija korisnika gde sa jednog racuna treba da se prebaci novac na drugi racun.

Specifikacija PostgreSQL modela:

Specifikacija HBASE modela:

Rezultati:

3.3 Primena u online analitičkom procesiranju (OLAP)

OLAP procesiranje sacinjeno je od skoro isključivo citanja podataka. Upiti koji se koriste obicno imaju parametre, imaju visok nivo kompleksnosti i visok procenat podataka kojima pristupa. Primer koji cemo koristiti jeste uopsten primer održavanja trgovinskog lanca koji ima skup musterija, proizvoda, dobavljača,

narudzbina. Nas OLAP eksperiment ce se sastojati iz dohvatanja izvestaja o ukupnom kvanitetu, ceni nakon odbijanja poreza, prosecnom popustu za dati status stavke narudzbine.

Specifikacija Postgres modela:

Specifikacija HBASE modela:

Rezultati:

3.4 Primena u distribuiranom okruženju

Skalabilnost

Vertikalna skalabilnost

Horizontalna skalabilnost

CAP teorema

Glava 4

Zaključak

Bibliografija

- [1] Designing data-intensive applications. the big ideas behind reliable, scalable and maintainable systems. 2024.
- [2] Optimizacija kod kolonski orijentisanih. 2024. on-line at: <https://chistadata.com/compression-techniques-column-oriented-databases/>.
- [3] asdsadsada. Expected computation time for Hamiltonian path problem. *SIAM Journal on Computing*, 16:486–502, 1987. on-line at: <https://static.googleusercontent.com/media/research.google.com/fr//archive/bigtable-osdi06.pdf>.
- [4] Regina O. Obe and Leo S. Hsu. PostgreSQL: Up and Running.

Biografija autora

Vuk Stefanović Karadžić (*Tršić, 26. oktobar/6. novembar 1787. — Beč, 7. februar 1864.*) bio je srpski filolog, reformator srpskog jezika, sakupljač narodnih umotvorina i pisac prvog rečnika srpskog jezika. Vuk je najznačajnija ličnost srpske književnosti prve polovine XIX veka. Stekao je i nekoliko počasnih doktorata. Učestvovao je u Prvom srpskom ustanku kao pisar i činovnik u Negotinskoj krajini, a nakon sloma ustanka preselio se u Beč, 1813. godine. Tu je upoznao Jerneja Kopitara, cenzora slovenskih knjiga, na čiji je podsticaj krenuo u prikupljanje srpskih narodnih pesama, reformu ćirilice i borbu za uvođenje narodnog jezika u srpsku književnost. Vukovim reformama u srpski jezik je uveden fonetski pravopis, a srpski jezik je potisnuo slavenosrpski jezik koji je u to vreme bio jezik obrazovanih ljudi. Tako se kao najvažnije godine Vukove reforme ističu 1818., 1836., 1839., 1847. i 1852.