NoSQL tehnologije i primjene

Tomić, Nela

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Nela Tomić

NoSQL TEHNOLOGIJE I PRIMJENE

Diplomski rad

Voditelj rada:
Prof. dr. sc. Robert Manger

Zagreb, 2016.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Zahvala

Zahvaljujem svom mentoru, prof. dr. sc. Robertu Mangeru

na ukazanom povjerenju i savjetima tijekom izrade diplomskog rada.

Sadržaj

Predgovor

Svrha ovog rada jest pružiti uvid u relevantnu tematiku informacijskih i računalnih tehnologija današnjice - takozvane NoSQL (ne-relacijske ili preciznije ne-samo-relacijske) tehnologije. Naglasak se stavlja na primjenu ovih tehnologija nad sustavima za upravljanje bazama podataka.

NoSQL tehnologije i njihove primjene nad sustavima za upravljanje bazama podataka obuhvaćaju širok skup koncepata, stoga ih je teško detaljno opisati jednim radom.

U ovom radu iznesene su osnove ovakvih sustava kroz četiri glavna poglavlja, svako od kojih se bavi jednim sustavom za upravljanje bazama podataka po drugačijem modelu pohranjivanja podataka.

Prvim poglavljem daje se uvod u tematiku, to jest motivaciju, povijesni pregled i definiciju NoSQL tehnologija.

U drugom poglavlju govori se o ključ-vrijednost bazama podataka. Naglasak se stavlja na osnovne značajke, upravljanje podacima te oblikovne obrasce koje često koristimo u radu s ovim bazama podataka.

U trećem poglavlju govori se o dokument bazama podataka, gdje se također daje pregled osnovnih značajki te načina upravljanja ovim bazama podataka.

Četvrtim poglavljem opisuju se stupčane baze podataka, njihova arhitektura te pomalo specifična primjena.

Petim poglavljem opisujemo graf baze podataka, njihovu osnovnu podjelu te primjene.

Šesto poglavlje prikazuje implementacije NoSQL baza podataka po svakom od opisanih modela u prijašnjim poglavljima.

1. Općenito o NoSQL tehnologijama

1.1. Aktualni trendovi informacijskih tehnologija

Relacijske baze podataka podupiru većinu poslovnih sustava današnjice. Održavaju dobrostojeću poziciju u mnogim organizacija i to s dobrim razlogom. Funkcionalnost i pouzdanost ovih baza podataka provjerena je nad mnogim sustavima kroz više od trideset godina njihova korištenja. Podržavaju opsežan ekosustav alata, kvalitetno i detaljno su dokumentirane, postoji mnoštvo kvalificirane radne snage za implementaciju i održavanje tih sustava. No ipak organizacije, kako analitičke tako i operativne, sve više uzimaju u obzir drugačija rješenja za svoje poslovne probleme.

2000-tih godina, poznate organizacije bazirane na internetskim uslugama počinju rasti u opsegu. Porastom u broju korisnika internetskih usluga, pojavljuju se i velike količine podataka - počinju se pohranjivati poveznice, podaci društvenih mreža, aktivnosti korisnika te rezultati analize podataka.

Aplikacije koje su jednom bile namijenjene konačnoj grupi potrošača sada se isporučuju kao uvijek dostupni, globalni servisi namijenjeni cijelom rasponu uređaja i korisnika. Sve su češći zahtjevi nad sustavima da stvaraju i rade s velikom količinom često promjenjivih podataka te različitim tipovima podataka - strukturiranim, polu-strukturiranim, nestrukturiranim i polimorfnim. Dvanaest-do-osamnaest mjesečni ciklus razvoja po modelu vodopada postaje sve manje primjenjiv. Naprema njemu, javlja se potreba za manjim timovima razvojnih inženjera koji rade u agilnim sprintovima, brzim iteracijama, generirajući nove funkcionalnosti svaki tjedan ili dva.

Organizacije zahtijevaju arhitekturu koja je u mogućnosti održavati velik porast u količini informacija. Zadovoljiti ovakav porast za računalnim resursima moguće je na dva načina: rastom prema gore ili prema vani. Rast prema gore (*scaling up*) podrazumijeva jača računala, više procesora, diskovnog prostora te memorije. Naravno, rast prema gore realno ima svoja tehnološka ograničenja, kao i puno veću cijenu nabave sa svakim dodatnim povećanjem snage računala. Alternativa je rast prema vani (*scaling out*), što se ostvaruje korištenjem više slabijih računala spojenih u računalnu mrežu – grozd (*cluster*).

Grozd slabijih računala koristi hardver generalne primjene što se pokazalo jeftinijim pothvatom za rast ovakvih raspona. Pokazalo se da je ovakav pristup također povoljniji po pitanju otpornosti sustava na greške. Kvarovi individualnih računala česta su pojava, no njihova brojnost ima prednost da se grozd može podesiti da nastavi s radom u slučaju takvih kvarova – time pružajući visoku dostupnost podataka.

Odlukom velikih internetskih pružatelja usluga da pređu na sustave grozdova, pojavio se novi problem – relacijske baze podataka nisu zamišljene da rade nad velikim brojem računala. Relacijski sustavi pretpostavljaju dobro definiranu strukturu podataka koji su gusti i uniformno raspoređeni. Također pretpostavljaju da se svojstva podataka mogu definirati unaprijed te da su međusobne veze sistematično referencirane i dobro utvrđene. Ovakvi sustavi mogu se nositi s određenom dozom iregularnosti i nedostatkom strukture, no u kontekstu masivno prorijeđenih podataka i slabo definiranih struktura, relacijske baze pokazuju se lošim izborom.

Naravno, postoje relacijski sustavi koji donekle odgovaraju na ovaj problem, kao što su Oracle RAC ili Microsoft SQL Server koji rade na konceptu zajedničkog diskovnog podsustava (*shared disk subsystem*). Koriste datotečni sustav svjestan grozda, koji zapisuje podatke na visoko dostupan diskovni podsustav. No grozd tada ima diskovni podsustav kao zajedničku točku kvara cijelog sustava (*single point of failure*).

Relacijske baze podataka mogu se podesiti i da rade na različitim poslužiteljima, gdje svaki poslužitelj sadrži drugi skup podataka. Ovakav način rada rasterećuje posao s jednog na više poslužitelja, no proces raspodjele podataka mora biti kontroliran sustavom koji prati odgovornost pojedinog poslužitelja za određeni podatak. Time se efektivno gubi mogućnost upita, transakcija, integritet referencijalnih veza (*referential integrity*) i razina konzistentnosti. Rješenjima poput denormalizacije tablica, popuštanjem ograničenja i garancija transakcijskih funkcionalnosti postiže se mogućnost rasta sustava prema van. No u tom slučaju relacijske baze podataka počinju sličiti NoSQL sustavima. Također, uz tehničke probleme, kao velik nedostatak pokazala se i cijena licenci relacijskih baza podataka. S obzirom na to da su takve licence obično definirane po jednom računalu, radeći s cijelim grozdom računala dovodi do frustrirajućih pregovora s odjelima za prodaju relacijskih sustava.

Postalo je dakle jasno da široko prihvaćene relacijske baze podataka nailaze na probleme kada suočene s masivnim količinama podataka - od efikasne obrade podataka, paralelizacije izvođenja, skalabilnosti i cijene.

Došlo se do zaključka da relacijski model podataka nije dobro usklađen s trenutnim potrebama sustava za upravljanje bazama podataka. Ove razmirice relacijskih baza podataka i rada s velikim brojem računala navode velike organizacije da osmisle alternativne načine čuvanja i upravljanja podacima. Dvije kompanije– Google i Amazon – posebno su bile utjecajne. Obje su bile prve na fronti rada na velikim grozdovima računala i zaprimanja velikih količina podataka. Također, obje su uspješne organizacije u rastu, sa snažnom tehničkom podlogom, što im daje motiv, sredstva i mogućnost za osmišljavanje vlastitih rješenja. Istina je da većina organizacija ne dostiže razinu protoka podataka kao Google i Amazon, no problemi koji susreću Google i Amazon itekako su relevantni i puno manjim sustavima, koji se počinju okretati istim rješenjima.

Stoga se postepeno odustaje od korištenja skupih monolitnih poslužiteljskih rješenja i infrastruktura pohrane te se okreće softverskim rješenjima otvorenog izvornog koda (*open source*), većem broju slabijih poslužitelja generalne namjene te pohrani podataka u Oblaku (*Cloud storage*).

1.2. Povijest NoSQL baza podataka

Google je u zadnjih nekoliko godina izgradio masivno skalabilnu infrastrukturu za svoj pretraživač i ostale aplikacije poput Google Maps, Google Earth, Google Mail, Google Finance i Google Apps. Cilj je bio osmisliti sustav sposoban obavljati paralelno procesiranje velike količine podataka. S tim u vidu, napravili su infrastrukturu koja uključuje distribuirani datotečni sustav, bazu podataka zasnovanu na modelu stupaca, distribuirani sustav koordinacije i okolinu za paralelno izvođenje algoritama. 2006. godine Google izdaje seriju dokumenata, opisujući ključne značajke svoje infrastrukture. Najvažniji od tih dokumenata su: „*The Google File System*“, „*MapReduce: Simplified Data Processing on Large Clusters*“, „*Bigtable: A Distributed Storage System for Structured Data*“, „*The Chubby Lock Service for Loosely-Coupled Distributed Systems*”.

U svom Bigtable dokumentu, Google opisuje distribuiranu bazu podataka ovako: „Bigtable je distribuirani sustav pohranjivanja i upravljanja podacima, osmišljen da se može skalirati na visoke razine: radi se o petabajtima podataka nad tisućama računala generalne primjene.“ Google ovim dokumentom opisuje prvu ne-relacijsku bazu podataka po modelu stupca. Izdavanje Google-ove dokumentacije izazvalo je veliki interes među razvojnim inženjerima. Razvojni tim pretraživača Lucene prvi su razvili sustav koji replicira neke od funkcionalnosti Google-ove infrastrukture. Ubrzo se razvojni tim pridružio Yahoo! organizaciji gdje su izgradili čitav sustav koji oponaša Google-ovu distribuiranu infrastrukturu poznat pod imenom Hadoop. Hadoop je postavio temelje ubrzanog razvoja mnoštva sličnih rješenja koja danas nazivamo NoSQL tehnologije.

Godinu dana nakon izdavanja Google-ovih dokumenata, 2007. godine, Amazon je odlučio podijeliti zajednici tehnologije iza arhitekture svog ne-relacijskog visoko skalabilnog distribuiranog sustava za pohranu podataka. Izdali su dokument pod nazivom: „*Dynamo*: *Amazon's Highly Available Key/value Store*“. Amazon je svoj sustav opisao kao: „Dynamo koristimo za upravljanje stanjima servisa koji imaju visoke zahtjeve za pouzdanosti i koji iziskuju pažljivu kontrolu nad balansom dostupnosti, konzistentnosti, performansa i cjenovnoj isplativosti.“ Amazon ovim dokumentom opisuje funkcionalnosti prve ne-relacijske baze podataka po modelu ključ-vrijednost. Do 2009. godine, mnogi proizvodi otvorenog izvornog koda izlaze na tržište: Riak, MongoDB, HBase, Accumulo, Hypertable, Redis, Cassandra, i Neo4j samo su neki od njih.

Mnogi razvojni inženjeri počinju ozbiljno proučavati kako nove tehnologije, metode i proizvode iskoristi za svoje vlastite aplikacije. Ova navala novih tehnologija potaknula je Erica Evansa iz Rackspace-a i Johana Oskarssonsa iz Last.fm-a da organiziraju prvo NoSQL organizirano sastajanje. Htjeli su osmisliti naslov sastanka koji će se brzo proširiti društvenim mrežama, pa su osmislili naslov #NoSQL. To je ujedno i prvo javno korištenje imena NoSQL za opis ovih tehnologija. Opis sastanka glasio je ovako: „Sastajemo se u svrhu razgovora o distribuiranim, ne-relacijskim bazama podataka otvorenog izvornog koda. Naišli ste na ograničenja tradicionalnih relacijskih baza podataka? Nije vam problem zamijeniti upitni jezik (*query language*) za veću skalabilnost sustava? Ili možda samo volite isprobavati sjajne nove stvari? U svakom slučaju, ovaj sastanak je za vas!

Pridružite nam se u otkrivanju zašto su Bigtable i Dynamo klonovi postali tako popularni. Okupili smo predavače s najinteresantnijih projekata današnjice da nam daju uvod u tematiku. Neki od predavača dolaze iz LinkedIn-a, Facebook-a, Powerset-a, StumbleUpon-a, ZVents-a, i couch.io-a te će predstaviti rješenja kao što su Voldemort, Cassandra, Dynamite, HBase, Hypertable i CouchDB.“

Ovo je prvo javno okupljanje gdje su razvojni inženjeri zajedno imali priliku razgovarati o različitim pristupima ne-relacijskim tehnologijama. U sljedećih par godina, različite NoSQL tehnologije i koncepti upravljanja velikim podacima dolaze gotovo iz svih popularnih i velikih organizacija - Facebook, Netflix, Yahoo, EBay, Hulu, IBM su samo neki od njih. Mnogi od njih doprinijeli su daljnjem razvoju tehnologija stavljanjem svojih proizvoda da budu otvorenog izvornog koda.

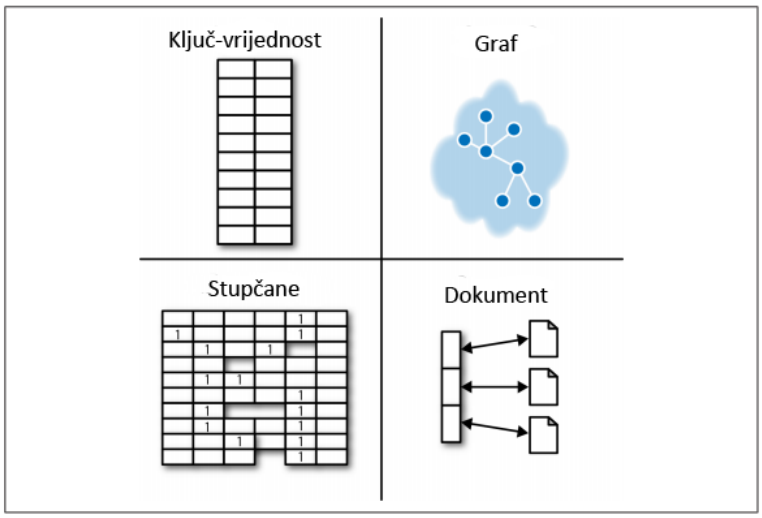
1.3. Definicija, podjela i značajke NoSQL baza podataka

NoSQL baze podataka široko možemo definirati kao skup tehnologija koji omogućavaju brzu i efikasnu obradu podataka s fokusom na agilnosti, pouzdanosti i izvođenju (performansama) sustava. Ti koncepti ostaju vjerni barem sljedećem:

1. Ne zahtijeva se striktna shema za svaki stvoreni zapis.
   Shema relacijskih baza podataka je opis tipova podataka koje pohranjujemo i njihove strukture. NoSQL baze podataka dopuštaju pohranu podataka bez prijašnjeg znanja baze podataka o strukturi podataka koje pohranjuje.
2. Može se distribuirati na hardver generalne namjene
   Relacijske baze podatka često rade bolje ili čak isključivo na specijaliziranom hardveru. To nije slučaj s NoSQL bazama podataka čija je namjena prvenstveno rad na puno slabijih računala. Time se postiže distribuiranost NoSQL baza podatka, to jest zapisi jedne baze mogu se pohraniti na više poslužitelja.
3. Ne koristi matematičku teoriju relacijskih sustava baza podataka.
   U relacijskim bazama podataka, relacije podataka uspostavljaju veze među tablicama. U NoSQL bazama podataka podaci se čuvaju kao agregati. Svi logički povezani podaci spremljeni su unutar jednog zapisa, što često zahtijeva dupliciranje pojedinih podatka.

NoSQL baze podataka možemo podijeliti na četiri tipa, s obzirom na to koji model pohranjivanja podataka koriste:

1. Ključ-vrijednost baze podataka
2. Dokument baze podataka
3. Stupčane baze podataka
4. Graf baze podataka



Slika 1.1 Prikaz različitih modela NoSQL baza podataka

Neke od bitnih značajki pri upravljanju podacima distribuiranih baza podataka su dostupnost podataka, konzistentnost podataka i otpornost particije.

Pojam konzistentnosti (atomarna konzistentnost) odnosi se na ishod jednog slijeda operacije: zahtjev klijenta - odgovor poslužitelja. Preciznije, na svaki zahtjev klijenta mora biti poslan točan odgovor s ažurnim podacima.

Pojam dostupnosti odnosi se na dostupnost servisa poslužitelja. Svaki zahtjev koji je funkcionirajući poslužitelj zaprimio mora rezultirati obradom zahtjeva i odgovorom, ne nužno s ažurnim podacima.

Otpornost particije odnosi se na mogućnost sustava da korektno nastavi s radom u uvjetima kada se dogodila greška u komunikaciji među poslužiteljima.

NoSQL baze podataka omogućuju podešavanje razine ovih svojstava. Obično se odabir vrši između tri modela:

- modela dostupnosti i otpornosti particije
- modela dostupnosti i konzistentnosti
- modela konzistentnosti i otpornosti particije

Na to nas navodi CAP (*Consistency, Availability, Partitioning*) teorem[1] koji kaže da je nemoguće na distribuiranom sustavu zahtijevati sve tri od kvaliteta – dostupnost, konzistentnost, otpornost particije – u isto vrijeme.

U praksi NoSQL baze podataka koriste sve značajke iz CAP teorema ali s manje strogim zahtjevima od onih navedenih u teoremu. Napuštamo stroge zahtjeve nad nekim značajkama da bi ostvarili bolju održivost drugih.

Dva su poznata modela na suprotnim stranama ljestvice konzistentnosti:

1. Potpuna konzistentnost

Potpuna ili ACID (*Atomicity, Consistency, Isolation, Durability*) konzistentnost najčešće je implementirana po modelu konzistentnosti i otpornosti particije – napušta stroge zahtjeve nad dostupnosti podataka u ime potpune konzistentnosti. Svojstva ACID baza podataka su:

Atomarnost: Unutar transakcije, sve operacije će se izvršiti ili se neće izvršiti niti jedna.

Konzistentnost: Izvršenjem transakcije nad bazom, baza prelazi iz jednog validnog stanja u drugo validno stanje. Transakcija u svakom trenutku mora zadovoljiti pravila i protokole sustava.

Izolacija: Istovremeno izvršavanje dviju transakcija rezultira istim stanjem sustava kao da su transakcije izvršene serijski. Svaka transakcija izvršava se u izolaciji od svih ostalih transakcija.

[1] Tvrdnju je predložio Eric Brewer 2000. godine, a formalni dokaz dali su Seth Gilbert i Nancy Lynch 2002. godine. Poznat je i pod imenom Brewerov teorem.

Trajnost: Po završetku transakcije, sustav garantira njenu trajnost čak i u slučaju greške u radu sustava.

2. Naknadna konzistentnost

Naknadna ili BASE (*Basically Available, Soft state, Eventual consistency*) konzistentnost najčešće je implementirana po modelu dostupnosti i otpornosti particije - napušta stroge zahtjeve nad konzistentnosti podataka, u svrhu veće dostupnosti podataka. Svojstva BASE baza podataka su:

Načelno dostupne: Svaki zahtjev nad bazom dobiti će odgovor. Odgovor može biti o uspješnosti operacije ili o nemogućnosti ostvarenja zahtjeva (podaci nisu dostupni, podaci nisu u konzistentnom stanju, došlo je do kvara i slično).

Labavog stanja: Stanje sustava može se mijenjati kroz vrijeme čak i bez klijentskog zahtjeva nad bazom (zbog naknadne konzistentnosti sustav je protočan i stanja konzistentnosti podataka često se mijenjaju).

Naknadno konzistentne: Baza povremeno može biti u nekonzistentnom stanju, no kroz određeno vrijeme, stanje baze podataka vraća se u konzistentno stanje (u konačnici će se novi zapisi propagirati na ostale poslužitelje i sustav će biti konzistentan).

2. Ključ-vrijednost sustavi za upravljanje bazama podataka

2.1. Uvod u ključ-vrijednost baze podataka

Sustav za upravljanje bazama podataka po modelu ključ - vrijednost jest najjednostavniji od modela ne-relacijskih sustava za upravljanje bazama podataka. Skraćeno možemo upotrijebiti i naziv: ključ-vrijednost baze podataka. Kao što ime nagovještava, oblikovanje, upravljanje i pohrana vrijednosti ovog modela temelji se na identifikatorima koje nazivamo ključevi.

Ključ-vrijednost model pohrane zapravo je složenija varijacija pohrane podataka pomoću asocijativnog polja. U računalnoj znanosti, asocijativno polje (rječnik, mapa) jest apstraktni tip podataka sličan tipu polja, no s manje strožim zahtjevima na tipove vrijednosti koje pohranjujemo. Vrijednosti indeksa asocijativnog polja nisu ograničene na cjelobrojni tip, također, dopuštena je pohrana vrijednosti različitih tipova unutar istog asocijativnog polja. Možemo reći da je asocijativno polje kolekcija podataka tipa ključ-vrijednost, gdje se svaki ključ u kolekciji pojavljuje najviše jedanput. Takve strukture podataka popularne su zbog efikasnog O(1) algoritamskog vremena izvođenja pristupa podacima. Upravo radi toga ključ-vrijednost model pohrane podataka poznat je po odličnim performansama i mogućnošću skaliranja.

Ključ-vrijednost baze podataka dijelimo na dva tipa s obzirom na to koji način pohrane podataka koriste:

1. Tip koji pohranjuje podatke samo u memoriji računala (*in-memory cache*). Tipično se koristi u situaciji kada je korisniku potreban brži pristup podacima nego što to dopušta dohvaćanje podataka s trajnih spremišta za pohranu podataka kao što su diskovi. Naravno, gašenjem računala gube se podaci iz memorije.

2. Tip koji čuva podatke dugoročno na trajnim spremištima za pohranu podatka. Ovaj tip nudi prednost brzine pristupa podacima korištenjem memorije, ali i trajno sprema podatke - kao što bi se to očekivalo od baze podatka.

Ključ-vrijednost baze podataka nameću minimalni skup zahtjeva na uređenje i raspored podataka. Glavni zahtjev je da svaka vrijednost ima jedinstveni identifikator, to jest ključ, unutar istog imenika. Imenik je pojam za skup parova ključ-vrijednost koji imaju logičku poveznicu jedni s drugima. Vrijednost koja se pohranjuje može biti bilo koji tip podatka i baza podataka ne mora unaprijed znati tip vrijednosti koji pohranjuje. Odgovornost o tipu koji se pohranjuje leži na programu to jest programeru koji koristi bazu podataka. Operacije nad podacima ovog jednostavnog modela također su jednostavne. Moguće je dohvatiti ili postaviti vrijednost za određeni ključ te izbrisati ključ iz sustava pohrane.

2.2. Osnovne značajke ključ-vrijednost baza podataka

Implementacija ključ-vrijednost baza podataka ima mnoštvo, no sve imaju nekoliko zajedničkih ključnih mogućnosti: jednostavnost, brzinu i skalabilnost.

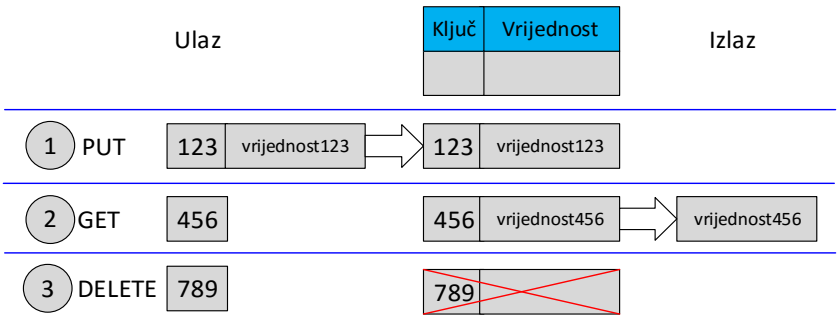
2.2.1. Jednostavnost

Ključ-vrijednost baze podataka rade s jednostavnim podatkovnim modelom te imaju jednostavnu sintaksu za upravljanje podacima. Za rad s podacima nije potrebna prethodna definicija sheme baze niti definicija tipova podataka s kojima radimo. Također, pojavi li se naknadno potreba za dodatnim atributima koje želimo pratiti unutar objekta, baza ne zahtijeva znanje o tome. Dovoljno je ažurirati programski kod koji uključuje nove atribute. Ovakva fleksibilnost korisna je pri radu s atributima koji često mijenjaju tip podatka ili u slučaju kada je isti atribut drugačijeg tipa unutar različitih objekata.

Programsko sučelje ili API (*Application Programming Interface*) za rad s bazom sastoji se od operacija postavljanja vrijednosti (*put*), dohvaćanja vrijednosti (*get*) i brisanja para ključ-vrijednost (*delete*). Koristimo izraze *put, get* i *delete* radi sklada sa standardnim REST (*REpresentational State Transfer*) protokolom. REST protokol jest standardizirano sučelje mrežne arhitekture, dakle skup principa kojima opisujemo pravila komunikacije i prijenosa podataka među komponentama distribuiranog sustava. Sučelja pisana u skladu sa sintaksom REST protokola nazivamo RESTful sučeljima.

Sučelja koja koristimo za rad s podacima možemo opisati na sljedeći način:

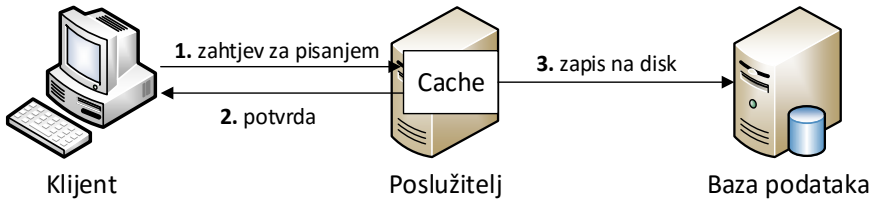
- put ($key as String, $value as item()), čime dodajemo par ključ-vrijednost u bazu ili ažuriramo vrijednost u bazi u slučaju da zadani ključ već postoji.
- get ($key as String) as item(), čime dohvaćamo vrijednost za zadani ključ ili grešku u slučaju da zadani ključ ne postoji
- delete ($key as String), čime brišemo par ključ-vrijednost iz baze ili dobivamo dojavu o grešci u slučaju nepostojećeg zadanog ključa.



Slika 2.1 Primjer sučelja nad ključ-vrijednost bazom podataka

2.2.2. Brzina

Zbog svoje jednostavne ali visoko optimizirane arhitekture temeljene na asocijativnom polju, ključ-vrijednost baza podataka dostavlja veliku propusnost podataka i brzo izvođenje operacija usporedno s ostalim modelima baza podataka. Već napomenuto spremanje podataka u radnu memoriju i čitanje iz iste omogućava velike brzine izvođenja operacija. Naravno radna memorija nije trajno spremište podataka i radi toga ključ vrijednost baze podataka koriste još i čvrste diskove u svrhu trajnog očuvanja podataka. Promjeni li aplikacija vrijednost vezanu uz određeni ključ, baza podataka promjeni prvo zadanu vrijednost u radnoj memoriji i pošalje aplikaciji poruku da je vrijednost sačuvana. Aplikacija je sada slobodna nastaviti dalje s radom, a baza podataka za to vrijeme sprema promjenu na čvrsti disk. U ovom procesu postoji kratki raspon vremena kada je moguće da se dogodi greška u radu poslužitelja te da se podaci nisu sačuvali na disku.



Slika 2.2 Primjer komunikacije klijentskog programa i ključ-vrijednost baze podataka za operaciju pisanja

Slično, operacije čitanja izvode se brzo zbog činjenice da je proces čitanja iz memorije puno brži od dohvaćanja podataka s diska. Brzina također ovisi o načinu spremanja složenijih i međusobno povezanih skupina podataka. Dobra je praksa podatke koje često koristimo jedne s drugima denormalizirati, to jest spremiti ih u isti zapis. Time postižemo da se dohvat takvih podataka vrši samo jednim čitanjem, dakle bolju protočnost upita. S druge strane to znači da se isti podaci moraju pohraniti u više različitih zapisa pa time zauzimamo više diskovnog prostora za spremanje podataka.

S obzirom na to da veličina baze podataka može nadići veličinu radne memorije, ključ-vrijednost sustavi imaju ugrađene algoritme sažimanja podataka čime se postiže povećanje efektivnog kapaciteta memorije. Često čak niti to nije dovoljno te se tada ključ-vrijednost baze podataka ugrađenim algoritmima brinu za racionalno upravljanje podacima koji se spremaju u radnu memoriju. Algoritmima se odlučuje koje će se skupine podataka osloboditi iz memorije da bi se novi podaci mogli zapisati.

Jedan od čestih algoritama jest LRU (*Least Recently Used*) algoritam. Ideja iza algoritma je ta da je veća vjerojatnost da će se podaci kojima se nedavno pristupalo koristiti opet, nego podaci koji se nisu koristili već neko vrijeme.

Neke ključ-vrijednost baze podataka dopuštaju korisniku i mogućnost dinamičkog upravljanja memorijom s obzirom na to kolika je količina upita i ulaznog opterećenja sustava (*ingest load*).

2.2.3. Skalabilnost

Skalabilnost ili sposobnost rasta jest mogućnost dodavanja (ili uklanjanja) poslužitelja u grozd računala – sustav međusobno povezanih poslužitelja obično spojenih u istu lokalnu mrežu koji se ponašaju kao cjelina. Potreba dodavanja novog poslužitelja u sustav

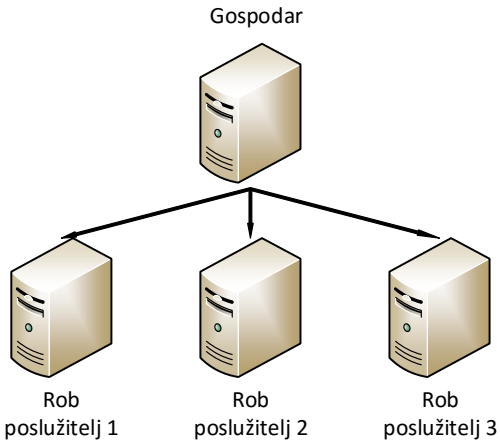
posljedica je povećanja opterećenja na trenutni sustav te se mora obaviti uz minimalno ometanje operacija čitanja i pisanja zapisa koji se izvode.

Ključ-vrijednost baze podataka oslanjaju se na dva pristupa skaliranja s obzirom na to kako se podaci repliciraju:

1. Replikacija poslužitelja po gospodar-rob modelu

Gospodar-rob model (*Master-Slave Replication*) odlično služi situaciju kada sustav dobiva puno više zahtjeva za čitanjem nego za pisanjem. U tom slučaju ima smisla imati više poslužitelja koji odgovaraju na zahtjeve čitanja podataka od onih koje opslužuju zahtjeve pisanja. Gospodarom nazivamo onog poslužitelja koji prihvaća zahtjeve pisanja i čitanja. Odgovoran je za održavanje i ažuriranje zapisa te repliciranje istih na ostale poslužitelje. Rob poslužitelj odgovara samo na upite dohvata to jest čitanja zapisa.
Ovakav model repliciranja tipično koriste ACID-usuglašeni ključ-vrijednost sustavi.

Kako bi se omogućila konzistentnost, gospodar poslužitelj obavlja zapis podataka u bazu te replikaciju istih na ostale poslužitelje, prije nego proglasi transakciju gotovom. Ovakav mehanizam se naziva dvofazno izvršenje transakcije (*two-phase commit*). Dvofazno izvršenje zahtijeva duže vrijeme obrade podataka te stvara dodatan mrežni promet među replikama.

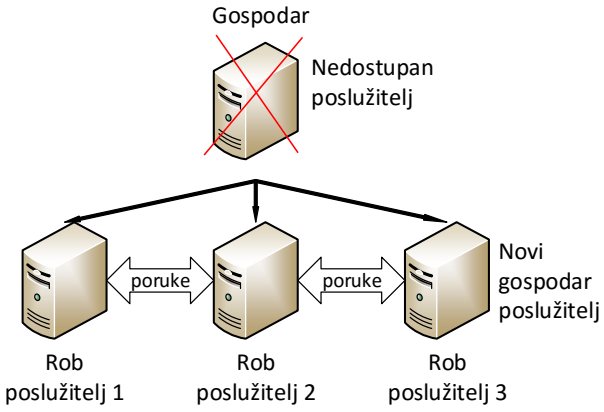


Slika 2.3 Gospodar-rob arhitektura

Prednost ovakve arhitekture jest njezina jednostavnost. Svaki poslužitelj, osim gospodara, komunicira samo još s jednim poslužiteljem.

S obzirom na to da gospodar poslužitelj obavlja sve zapise na bazu, nema potrebe za koordinacijom operacija pisanja niti za rješavanjem sukoba među poslužiteljima koji istovremeno pokušavaju pisati na bazu.

Nedostatak modela jest slučaj ako gospodar poslužitelj prestane raditi. Time cijeli sustav više ne dopušta pisanje na bazu i efektivno gubi funkcionalnost obavljanja operacija. Dakle gospodar poslužitelj jest zajednička točka kvara sustava te utječe na raspoloživost i uporabivost cijelog sustava. U tu svrhu razvijeni su protokoli za distribuirane sustave kojima aktivni poslužitelji mogu otkriti kada neki od poslužitelja u sustavu prestane s radom. Na primjer pojedini poslužitelji robovi mogu slati jednostavne poruke prema gospodaru poslužitelju u svrhu provjere je li još uvijek aktivan. Ne dobiju li odgovor unutar nekog perioda zaključuju da poslužitelj više nije aktivan i započinju protokol odabira novog gospodara poslužitelja sami među sobom. Novi gospodar poslužitelj sada ima za zadatak prihvatiti operacije pisanja i čitanja te sustav nastavlja normalno s radom.



Slika 2.4 U slučaju prestanka rada gospodara poslužitelja, započinje se protokol odabira novog gospodara.

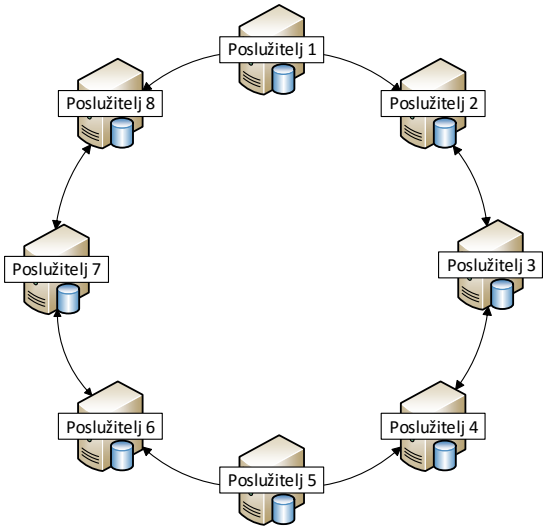
2. Replikacija poslužitelja po modelu ravnopravnih partnera

Gospodar-rob model replikacije, gdje gospodar poslužitelj jedini prihvaća zahtjeve za pisanjem nije optimalan u slučaju da sustav dobiva mnoštvo takvih zahtjeva. Povećanje u broju zahtjeva za pisanjem može dovesti do usporenja sustava a time i ograničiti mogućnost skaliranja. Bolja je opcija u ovom slučaju dopustiti svim poslužiteljima prihvaćanje zahtjeva za pisanjem i čitanjem.

Replikacija po modelu ravnopravnih partnera (*Masterless* ili *Master-Master Replication*) stoga umjesto koncepta primarnog vlasnika particije koristi algoritme otkrivanja najažurnije vrijednosti za pojedini ključ i brisanja starijih vrijednosti. Ovaj se postupak u mnogim ključ-vrijednost modelima obavlja sporo – u trenutku zahtjeva za čitanjem podatka. Zato ovaj model skaliranja nazivamo naknadno konzistentnim. Neki ključ-vrijednost sustavi koriste servise koji provjeravaju konzistentnost tijekom normalnih operacija (ne samo zahtjeva za čitanjem) i time ubrzavaju ovaj proces.

Da bi se omogućilo automatsko rješavanje sukoba potreban je mehanizam koji ukazuje na najnoviju vrijednost podataka. Naknadno konzistentni ključ-vrijednost sustavi rješavaju ovakve sukobe na više načina, na primjer korištenjem vektorskog sata koji rezultira parcijalnim uređajem događaja unutar sustava ili jednostavnije korištenjem vremenske oznake (*timestamp*) koja ukazuje na zastarjele podatke. Kada automatsko rješavanje sukoba nije moguće, sve relevantne kopije podataka šalju se klijentu.

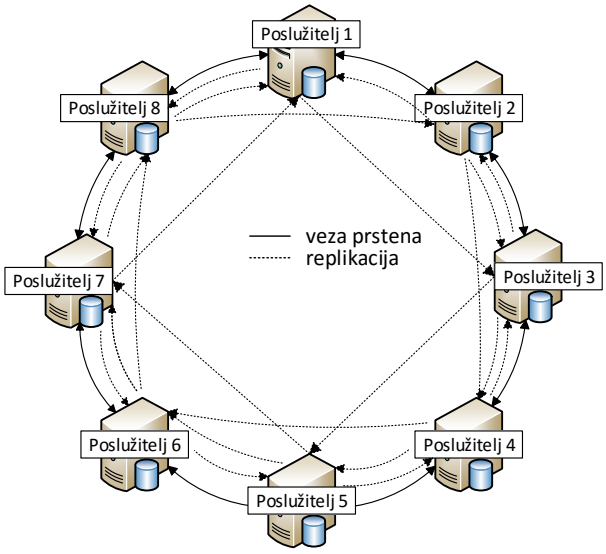
Uzmimo za primjer poslužitelje spojene po modelu ravnopravnih partnera u prstenastu strukturu takvu da je svaki poslužitelj logički povezan sa svojim susjednim poslužiteljem. Prstenasta struktura je korisna apstrakcija za promatranje replikacije u ovakvom modelu. U stvarnoj situaciji – unutar podatkovnog centra, poslužitelji bi bili spojeni na mrežnu sklopku i mogli bi direktno komunicirati jedan s drugim.



Slika 2.5 Grozd računala spojenih u prstenastu strukturu

Administratori baza podataka mogu podesiti ključ-vrijednost bazu podataka da čuva određeni broj replika. Svaki put kada se obavi operacija pisanja na jedan od poslužitelja, on pošalje promjene poslužiteljima koji čuvaju replike.

Uzmimo za primjer sustav koji čuva četiri replike podataka. Poslužitelj replicira svoje podatke svojim susjedima i poslužitelju udaljenom dva čvora ispred.



Slika 2.6 Grozd od osam poslužitelja spojenih u prstenastu konfiguraciju s faktorom repliciranja 4

Čuvanjem višestrukih kopija podataka na više poslužitelja osigurava se visoka dostupnost podataka u slučaju prestanka rada nekog od poslužitelja. Upravljanje poslužiteljima unutar grozda obavlja se automatski pomoću protokola razgovora (*gossip protocol*) među poslužiteljima. Nema potrebe za gospodarom poslužiteljem da koordinira slanje poruka unutar sustava.

Ovakva arhitektura često koristi proces popravka čitanja (*read repair*) koji se sastoji od dva koraka:

1. U trenutku čitanja zapisa određuje se koja od nekoliko dostupnih vrijednosti za ključ je najnovija.
2. Ako se ne može uskladiti dogovor oko najnovije vrijednosti tada se pošiljatelju upita šalju sve vrijednosti i daje mu se na odluku da sam odredi najnoviju vrijednost.

U svrhu ubrzanja procesa spremanja podataka na disk, ono se obavlja u podesivim vremenskim intervalima. Takva postavka ostavlja vremenski prostor unutar kojeg može doći do gubitka podataka, u slučaju da poslužitelj prestane raditi prije nego je obavljeno spremanje podataka na disk.

2.3. Upravljanje ključ-vrijednost bazama podataka

2.3.1. Upravljanje ključevima zapisa

Kao što je već rečeno, ključeve koristimo u svrhu identifikacije te indeksiranja vrijednosti u ključ-vrijednost bazama podataka. Glavna odlika ključa jest da mora biti jedinstven unutar istog imenika.

Konvencija imenovanja ključeva

Brzina dohvata podataka ključ-vrijednost baza uvelike ovisi o načinu imenovanja ključeva. Dobar ključ za određeni upit omogućava jednoznačnu identifikaciju zapisa bez potrebe prolaska kroz vrijednosti tog zapisa. Loše definiran ključ može zahtijevati od programskog koda da utvrdi odgovara li zapis postavljanom upitu, što može dovesti do lošijeg izvođenja programa. Dobro oblikovana konvencija imenovanja ključeva dopušta programerima lako smišljanje imena ključeva za nove entitete, instance i atribute.

Neke generalne smjernice imenovanja ključeva:

1. Korištenje smislenih i jednoznačnih naziva komponenata imena.
2. Korištenje standardnih razdjelnika pri imenovanju naziva komponenata. Jedan od najčešće korištenih razdjelnika je ':'. Formulu imenovanja možemo definirati kao: Ime entiteta + ':' + identifikator entiteta + ':' + atribut entiteta.
3. Poželjno je da su imena ključeva što kraća ali da se pridržavaju navedenih smjernica.
4. U slučaju da je potrebno dohvatiti iz baze podataka raspon vrijednosti, poželjno je da dio imena sadržava oznaku raspona. Raspon mogu biti numeričke vrijednosti ili datumi. Takvim imenovanjem ključeva lagano je napisati funkciju u programu koja dohvaća željeni raspon vrijednosti.

Korištenje ključeva u pretraživanju vrijednosti

Iako je korištenje cijelih brojeva za identifikaciju lokacije ključa praktična ideja, nije dovoljno fleksibilna s obzirom na to da želimo pri imenovanju koristiti brojeve, niz znakova pa čak i listu objekata. Zato se koristi posebna funkcija koja preslikava brojeve, nizove znakova i ostale moguće vrijednosti imena u jedinstvene nizove brojeva ili znakova. Funkciju koja prima proizvoljan niz znakova i vraća jedinstven niz znakova uvijek iste duljine nazivamo funkcijom raspršivanja (*hash function*). Ponekad se dogodi da funkcija raspršivanja za dva različita ulaza vrati isti izlaz, što nazivamo kolizija raspršivanja (*hash collision*).

Vrijednosti koje vraća funkcija raspršivanja međusobno se poprilično razlikuju, čak i ako su ulazne vrijednosti slične – na primjer ključevi s istim prefiksom. Izlaz takvih funkcija izgleda nasumično, no to nikako nije, s obzirom na to da funkcija svaki put za isti ulaz vraća isti izlaz.

Često se koristi funkcija raspršivanja SHA-1 za generiranje raspršnih vrijednosti (*hash values*). Te vrijednosti su brojevi u heksadecimalnom zapisu, što nam daje otprilike $1.4615 * 10^{48}$ različitih mogućnosti.

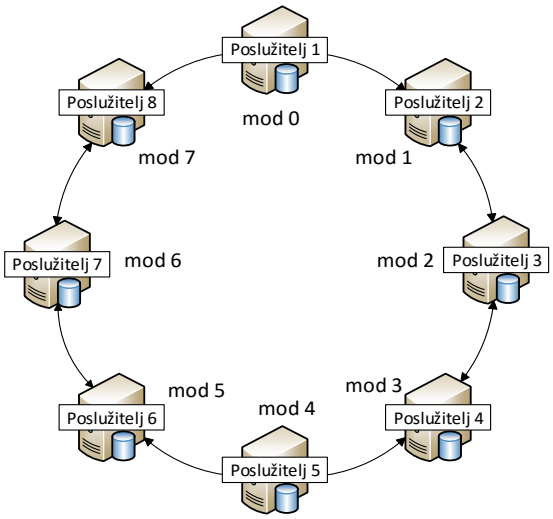
Korištenje ključeva pri particioniranju podataka

Grupiranje ključ-vrijednost parova i pridruživanje tih grupa različitim poslužiteljima unutar grozda nazivamo particioniranje (*partitioning*). Pravilno oblikovanje particija veoma je bitno s obzirom na to da neke implementacije ključ-vrijednost baza podataka ne dopuštaju promjenu broja particija jednom kada se grozd poslužitelja formira, nego samo njihovu distribuciju među poslužiteljima.

Raspršivanje je često korištena metoda u procesu particioniranja jer ravnomjerno raspoređuje ključeve i vrijednosti po svim poslužiteljima. To jest vraćena raspršna vrijednost jednoznačno određuje odgovornost poslužitelja za pojedini ključ.

Uzmimo za primjer grozd od osam poslužitelja povezanih po model ravnopravnih partnera u prstenastu strukturu. Mogli bismo svakom poslužitelju pridružiti osminu ključeva, no dobra praksa je iskoristiti funkciju raspršivanja za taj posao. Jedan način kako to učiniti

jest podijeliti raspršnu vrijednost pojedinog ključa s brojem poslužitelja i dobiveni ostatak pri dijeljenju iskoristimo kao identifikator poslužitelja zaduženog za taj ključ.

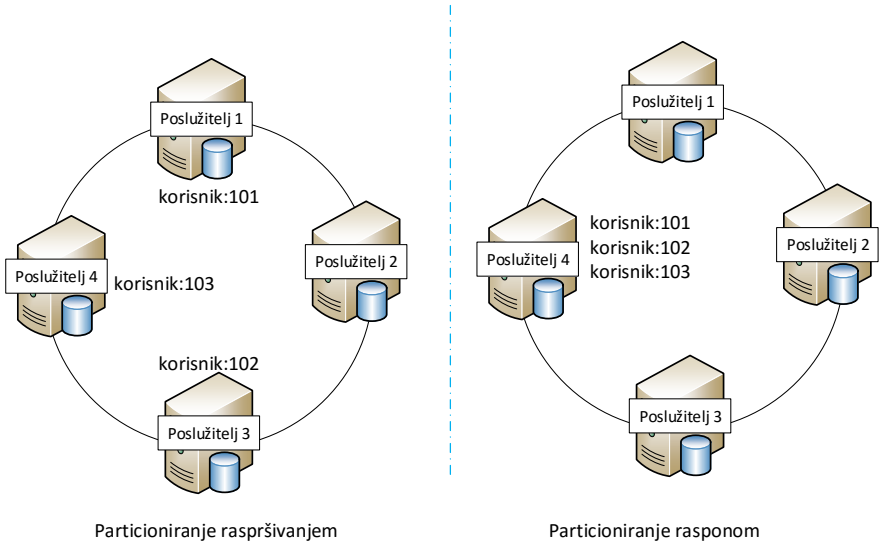


Slika 2.7 Grozd poslužitelja u prstenastoj konfiguraciji s ostacima pridruženim svakom poslužitelju

Korištenjem ove metode razdjeljivanja podataka po grozdu u kombinaciji sa smislenim imenovanjem ključeva implicitno rješavamo i problem kolizije podataka u grozdu bez gospodara poslužitelja. S obzirom na to da je u takvoj konfiguraciji svim poslužiteljima dopušteno pisanje u bazu, nije nezamislivo da dva poslužitelja istovremeno pokušaju obaviti operaciju nad istim ključem. Ovom metodom, ključevi istog imena imat će istu raspršnu vrijednost i biti preusmjereni uvijek na isti poslužitelj.

Još jedan proces razdjeljivanja podataka koji se često koristi jest razdjeljivanje po rasponu. Proces razdjeljivanja po rasponu grupira ključeve unutar predodređenog raspona uvijek na isti poslužitelj. Podrazumijeva se da nad ključevima postoji uređaj. Razdjeljivanje po rasponu zahtijeva postojanje tablice u kojoj su pridružene vrijednosti raspona ključeva određenim poslužiteljima.

Ovakav način razdjeljivanja može dovesti do problema u slučaju da nije uzet u obzir mogući rast broja podataka. Bude li potrebno restrukturiranje sheme razdjeljivanja moguće je da će se pojedine skupine ključeva prebaciti na druge poslužitelje, a time će doći i do migracije podataka među poslužiteljima.



Slika 2.8 Različite sheme raspršivanja dovode do različitog pridruživanja ključeva poslužiteljima

2.3.2. Upravljanje vrijednostima zapisa

NoSQL baze podataka fleksibilne su u načinu spremanju podataka. Ne očekuju definiciju tipova podataka koje spremaju. Vrijednosti mogu biti bilo koji BLOB (*Binary Large OBject*) tip podatka, kao što su slike, internet stranice, dokumenti, video zapisi.

Unatoč fleksibilnosti ključ-vrijednost baza podataka, postoje praktična ograničenja, na primjer na veličinu zapisa. U praksi spremanje prevelikih zapisa dovodi do problema s izvođenjem cjelokupnog sustava te se samim time nameće rekonstrukcija sheme zapisa na manje jedinke.

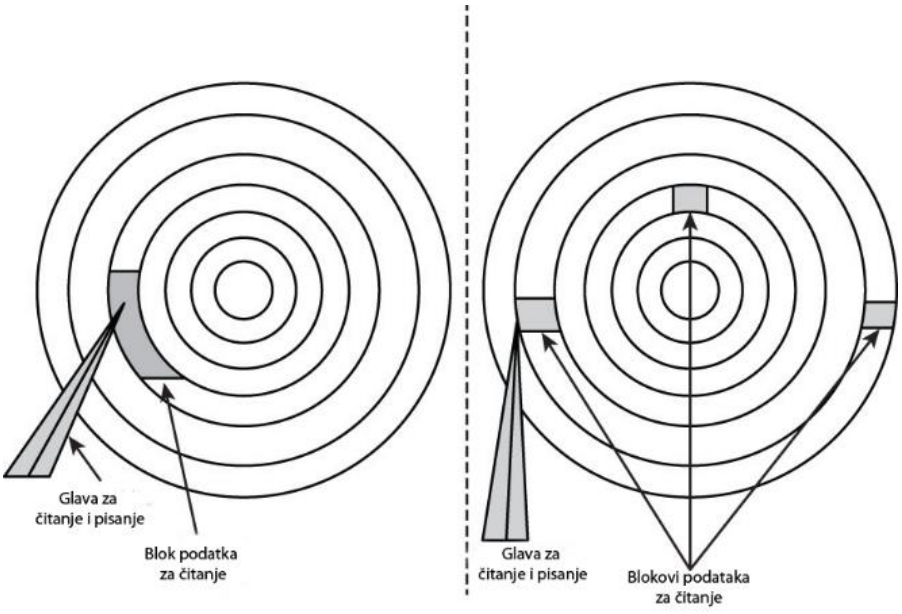
Kako bi olakšalo upravljanje podacima, većina baza podataka pruža pomoćne funkcije za serijalizaciju podatkovnih struktura kao što su liste, skupovi, rječnici i slično. Poneke pružaju i mogućnost sekundarnih indeksa nad strukturiran zapisima poput JSON (*JavaScript Object Notation*) ili XML (*Extensible Markup Language*) formata. To omogućava upite raspona (veće od, manje od) kao i upite jednakosti (jednako kao, nije jednako kao).

Pri dizajnu i izradi aplikacije treba obratiti pozornost na opterećenje poslužitelja. Ako aplikacija često dohvaća određenu skupinu podataka (atributa) zajedno, poželjno je te

podatke grupirati u jedan zapis kako bi se mogli dohvatili u jednom pozivu. Dohvat takvih podataka iz baze može zahtijevati duže vrijeme izvođenja u usporedbi s primitivnim operacijama. Jedan od razloga jest čekanje na čitanje podataka s diska, točnije čekanje da se glava diska postavi na poziciju.

Latencija, ili vrijeme koje je potrebno da se podatak pročita s diska može biti drastično veća nego vrijeme obrade podataka. Jedan od načina da se ubrza taj proces jest spremanje često korištenih podataka u radnu memoriju poslužitelja. Iako princip funkcionira za velik broj slučajeva, ograničen je veličinom memorije poslužitelja.

Drugi način ubrzanja procesa jest spremanje podataka koji se često koriste zajedno na disku. Agregat je kolekcija podataka kojoj pristupamo kao cjelini. Agregati olakšavaju bazi upravljanje podacima, posebno na distribuiranim sustavima gdje se dijelovi podataka mogu nalaziti na udaljenim računalima. Agregat je složenija struktura za korištenje nego što bi bilo korištenje više ključeva za različite dijelove podataka no pokazuju se da ima velike prednosti u određenim situacijama. Već spomenuta prednost jest manji broj čitanja s diska koje treba obaviti da bi se pristupilo svim potrebnim podacima.



Slika 2.9 Čitanje jednog većeg bloka podataka jest brže nego čitanje više manjih blokova referenciranih različitim ključevima

Ključ-vrijednost baze podataka obično spremaju cijele liste podataka zajedno u podatkovni blok te nije potrebno raspršivati više ključeva i naknadno dohvaćati više podatkovnih blokova. Iznimka se javlja u slučaju da je veličina podatka veća nego veličina dostupnog podatkovnog bloka.

Ako često dohvaćamo određenu vrijednost zasebno ali i kao dio podatkovne skupine, dobra praksa je spremiti vrijednost odvojeno, ali i unutar drugog podatkovnog bloka. Time se podaci dupliciraju unutar ključ-vrijednost baze podataka, ali dolazi do poboljšanja izvođenja programa.

Spremanje previše podataka u vrijednost može imati loš utjecaj na izvođenje programa. Mora li se cijela velika struktura spremati u memoriju, a često se pristupa samo pojedinim dijelovima te strukture, onda nepotrebno zauzimamo memoriju.

2.4. Oblikovni obrasci ključ-vrijednost baza podataka

Oblikovni obrasci su iznova iskoristivo generalno rješenje za probleme koje tipično susrećemo pri razvoju softvera. Pri korištenju ključ-vrijednost baza podataka poželjno je razmotriti sljedećih nekoliko oblikovnih obrazaca.

Vrijeme života ključeva

Vrijeme života - TTL (*Time To Live*) ključeva jest obrazac koristan kada u sustavu postoje operacije ili podaci koji se mogu zanemariti nakon određenog perioda neaktivnosti ili nemogućnosti izvršenja. Vrijeme života je izraz često korišten u računalnoj znanosti kako bi se opisao privremeni objekt. Na primjer podatkovni paket poslan s jednog na drugi poslužitelj može imati definiran parametar TTL koji označava koliko puta paket smije biti proslijeđen na putu prema odredištu. Ako je paket poslan više puta od dopuštenog iznosa, paket se odbacuje i neće biti dostavljen. TTL obrazac često je koristan u radu s ključ-vrijednost bazom podataka. Na primjer za spremanje podataka na poslužiteljima s ograničenom memorijom, ili pri spremanju ključeva koji čuvaju podatke samo neko određeno vrijeme.

Oponašanje tablica

Iako većina ključ-vrijednost baza direktno ne podržava strukture kao što tablice u relacijskim baza podataka, ponekad tablica može biti korisna. Da bi oponašali tablicu modeliramo dohvaćanje i postavljanje više atributa vezanih uz jedan objekt. Metoda oponašanja tablica djelomično je već opisana u poglavlju o konvenciji imenovanja ključeva baziranoj na imenu entiteta, identifikatoru entiteta i atributu entiteta. Ovaj obrazac stavlja naglasak na implementaciju osnovnih operacija dohvata i postavljanja vrijednosti (*get, set*), no ne i mogućnosti SQL upita. Obrazac je koristan ako često dohvaćamo i postavljamo skup povezanih atributa i ako se radi o malom broju takvih emuliranih tablica.

Atomarni agregati

Atomarni agregat je cjelina podataka za koje se sve vrijednosti ažuriraju istovremeno ili se ne ažuriraju uopće. Neke ključ-vrijednost baze podataka imaju podršku za transakcije, koje osiguravaju zajedničko ažuriranje podataka. U slučaju korištenja baze koja ne podržava transakcije, obrazac atomarnih agregata jest dobro rješenje. Obrazac koristi samo jednu operaciju za spremanje više vrijednosti. Spremanjem tih vrijednosti kroz više operacija dovodi do rizika da će se neke vrijednosti spremiti dok druge neće. Ovaj obrazac ne zamjenjuje u potpunosti mogućnosti dobivene podrškom transakcija.

Pobrojivi ključevi

Pobrojivi ključevi (*enumerable keys*) su ključevi koji sadrže brojače ili sekvence kao komponentu imena. Pobrojivi ključevi omogućuju pojednostavljenu funkcionalnost pretraživanja po rasponu, omogućujući time i programu da generira nove ključeve i provjerava postojanje ključeva unutar raspona. Ovaj obrazac je koristan pri radu s grupama ključeva, posebno ako želimo dohvaćati ključeve stvorene na određen datum. Generirali bismo seriju ključeva s tim datumom i brojačem u imenu sve dok ne generiramo ključ koji ne postoji, ili ne naiđemo na traženi ključ.

Invertirani indeksi

Standardni indeks sprema identifikator zapisa te popis riječi koje se spominju unutar njega. Pri vršenju upita nad takvim zapisom ovaj pristup nije previše koristan. Puno korisnije bilo bi spremati sve unikatne riječi koje se spominju u zapisima te listu identifikatora zapisa u kojima se spominje pojedina riječ. Takav indeks nazivamo invertirani indeks (*inverted index*). Tako indeksirati možemo i datume (vrijeme kreiranja, ažuriranja …), brojeve (broj stranice, verzija), izraze, čak i geografske podatke (koordinate). Listu zapisa koja sadrži tražene izraze nazivamo listom izraza (*term lists*).

3. Dokument sustavi za upravljanje bazama podataka

3.1. Uvod u dokument baze podataka

Razvojni inženjeri često se okreću ka dokument modelu NoSQL baza podataka kada im je potrebna fleksibilnost ključ-vrijednost modela, ali upravljaju sa složenijim podatkovnim strukturama te im je bitna mogućnost indeksiranja podataka i pretraživanja istih po vrijednosti.

Vidjeli smo da ključ-vrijednost baze podataka za svaki ključ vraćaju vrijednost vezanu za taj ključ. Dokument baze podataka rade po drugačijem principu. Ključ dokument baza podatka jest jedinstven identifikator kao u ključ-vrijednost bazama, no on se ne mora koristi za operacije nad bazom. Dohvaćanje objekta iz baze obavlja se postavljanjem upita na sadržaj unutar dokumenta. Dakle dokument baze podataka možemo smatrati složenijom varijantom ključ-vrijednost baza podataka, gdje se kao vrijednost spremaju dokumenti koji se još mogu i pretraživati.

U nazivu dokument baza podataka, riječ „dokument“ označava općenitu hijerarhijsku strukturu podataka koja može imati podstrukture. To je općenitiji pojam od recimo Microsoft Word dokumenta ili HTML dokumenta, iako su to dva tipa dokumenta koja možemo pohraniti u dokument bazama podataka. Generalno dokument u dokument bazi predstavlja samo-opisujuću stablastu strukturu poput JSON, XML ili BSON (*Binary JSON*) strukture. Te strukture dalje se mogu sastojati od kolekcija elemenata, rječnika, skalarnih vrijednosti i slično. Dokument se sastoji od binarnih podataka ili neformatiranog teksta (*plaintext*), može biti djelomično-strukturiranog formata poput JSON ili XML te može biti visoko-strukturiran ako je usklađen sa shemom poput XSD (*XML Schema Definition*) sheme.

Djelomično-strukturirani format spada u kategoriju strukturiranih formata koji ne prate shemu nekakve formalne strukture. Takvi dokumenti sadržavaju markere da bi razdvojili semantičke elemente i nametnuli hijerarhiju unutar dokumenta. Zato još i kažemo da su samo-opisujuće strukture.

Posljedica korištenja dokument baza podataka jest da se svaki dokument automatski indeksira dodavanjem u bazu. To povlači da se svaki dokument može pretražiti. Znamo li bilo koje svojstvo unutar dokumenta, baza nam lako daje odgovor koji sve dokumenti sadrže zadano svojstvo. Osim povratne informacije koji dokument sadrži određeno svojstvo, baza nam daje i informaciju o točnoj lokaciji tog svojstva unutar dokumenta. Točna lokacija dohvaća se pomoću specijalnog ključa – puta po dokumentu (*document path*) – kojim se obilaze vrijednosti listova po strukturi stabla.

Dokument baze podataka su fleksibilne i ne zahtijevaju shemu podataka. U bazu se mogu spremiti podaci bez da baza ima ikakvo znanje o njihovoj strukturi, tipu ili značenju. To, i činjenica da se struktura podataka može naknadno mijenjati iako je sustav već u produkciji, je odlika koja čini ovu vrstu baza podataka odličnom u agilnom razvojnom procesu. Redizajn sheme za svaku promjenu u strukturi, kao što bi se morao napraviti u relacijskim sustavima, ovdje je nepotreban.

Još jedna razlika s relacijskim sustavima jest ta da se novi atributi mogu naknadno dodavati bez da se mijenja struktura postojećih dokumenata u kolekciji.

Također, u dokument bazi podataka ne postoje prazni atributi. Ako vrijednost za pojedini atribut nije pronađena, pretpostavlja se da nije postavljena ili da nije relevantna za dokument.

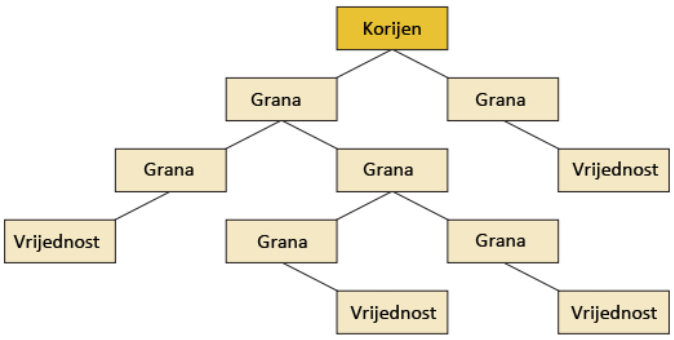
Unatoč tome da shemu nije potrebno definirati prije dodavanja dokumenata u bazu, postoji implicitna organizacija pri dodavanju skupa dokumenata u bazu. Polimorfna shema (*polymorphic schema*) je naziv za organizacijsku shemu dokument baza podataka koja dopušta dodavanje različitih tipova dokumenata u jednu kolekciju.

Entiteti koji pripadaju istoj kolekciji dokumenata mogu imati međusobno drugačije atribute te poredak tih atributa unutar entiteta nije bitan.

3.2. Osnovne značajke dokument baza podataka

3.2.1. Model podataka zasnovan na strukturi stabla

Dokument baze podataka možemo zamisliti kao stablastu strukturu prikazanu na slici 3.1. Stablo dokumenta ima glavni korijenski element. Na idućoj razini ispod korijena nalazi se red grana, pod-grana i vrijednosti. Vrijednosti se obično nalaze na poziciji listova stabla. Za svaku granu (ili vrijednost) postoji put kojim se može doći do te grane krenuvši od korijena.



Slika 3.1 Stablasta struktura dokument baza podataka

Kolekcije dokumenata

Dokumente generalno grupiramo u kolekcije međusobno sličnih dokumenata. Kolekciju možemo zamišljati kao listu dokumenata. Struktura kolekcije podsjeća na strukturu direktorija Windows ili Unix sustava. Dokumenti u kolekcijama ne moraju biti iste strukture, ali je poželjno da strukture budu slične. Kolekcije nam mogu služiti kao način prolaska kroz hijerarhije dokumenata, grupiranja logički sličnih dokumenata ili za postavljanje pravila poput dopuštenja, indeksa ili okidača. Kao što stabla mogu sadržavati pod-stabla, tako i kolekcije mogu sadržavati druge kolekcije. Loše oblikovanje kolekcija može utjecati na izvođenje sustava te usporiti aplikaciju.

Prikaz strukture u sažetom obliku

Dokument baze podataka uobičajeno spremaju dokumente na disk u sažetom obliku. Da bi to bilo moguće, baza podataka mora razumjeti format dokumenta kojeg sprema. Primjerice, spremamo li dokumente u JSON ili XML formatu, baza koristi takvu strukturu da bi bolje upravljala podacima na disku.

Implementacija dokument baze MongoDB pohranjuje dokumente u vlastitoj BSON binarnoj reprezentaciji JSON formata, što je korisno jer JSON može sadržavati puno teksta u imenima atributa. Može se sačuvati dosta diskovnog prostora samom kompresijom tih tekstova atributa kao jednostavnih numeričkih identifikatora, umjesto kao dugačkog niza znakova.

Diskovni prostor još je bitnije sačuvati kod XML formata jer taj format sadrži još otvorene i zatvorene tagove. Implementacija dokument baze MarkLogic sve elemente i atribute tretira kao izraze. Svakom izrazu je dodijeljen jedinstveni brojčani identifikator. Stoga MarkLogic može za takve izraze koristiti svoju binarnu strukturu stabla. Tim postupkom čuva se diskovni prostor puno više nego da smo XML strukturu čuvali kao nizove znakova. MarkLogic također koristi identifikatore izraza u sklopu indeksiranja. Univerzalni indeks indeksira sve strukture: elemente i atribute, veze čvorova roditelja i djece, egzaktne vrijednosti elemenata, tekst i slično. Time se ubrzava izvršavanje upita nad dokumentima.

Djelomično prepravljanje dokumenta

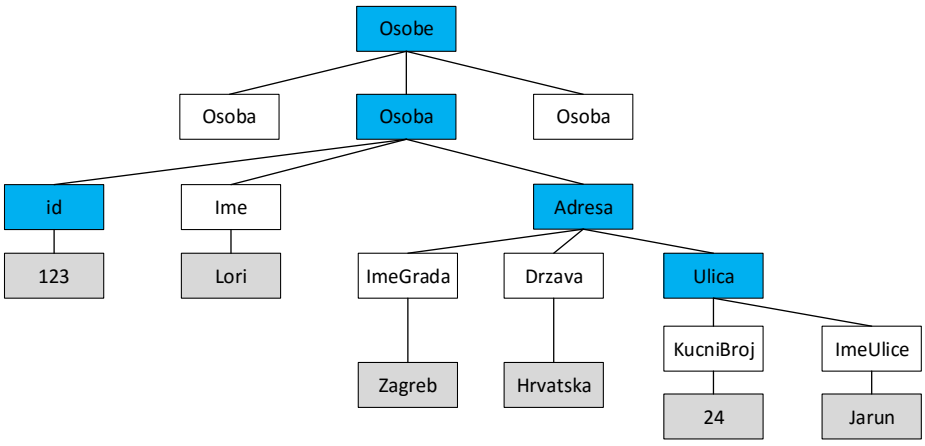
U nekim slučajevima umjesto ažuriranja cijelog dokumenta, potrebno je promijeniti samo manji dio dokumenta ili čak samo jednu vrijednost. Takvu operaciju nazivamo djelomično prepravljanje dokumenata. Ona se odvija unutar transakcije na samoj bazi.

Kod baza koje nemaju podržanu operaciju djelomičnog prepravljanja, program treba odraditi operaciju čitanja, ispravljanja i ažuriranja - RMU (*Read, Modify, Update*) nad cijelim dokumentom. Taj proces može biti skup po pitanju vremena izvođenja te u mnogim NoSQL bazama podataka nije ACID operacija – dakle neki drugi proces može ažurirati dokument između koraka čitanja i ažuriranja prvog procesa.

Operacije djelomičnog prepravljanja također uključuju nadovezivanje elemenata, na primjer na čvor-roditelj unutar strukture stabla. Operacije nadovezivanja omogućuju nizanje podataka na dokumente (*document streaming*). Određuju gdje se u dokumentu dodaju novi elementi te obavljaju samo dodavanje elementa. Time se zaobilazi RMU postupak koji je zahtjevan za memoriju i stvara latenciju sustava. Proces nadovezivanja posebno je koristan aplikacijama koje zahtijevaju analiziranje podataka u stvarnom vremenu (*real-time*) jer su zaobilaženjem RMU postupka podaci odmah dostupni za daljnje upite.

API dokument baza podataka

Svaka dokument baza podataka ima API i upitni jezik koji omogućuje određivanje puta do bilo kojeg čvora ili grupe čvorova. Generalno čvorovi ne moraju imati zasebna imena. Umjesto toga koristi se broj koji određuje njihov položaj u stablu. Kada tražimo određenu vrijednost u dokumentu, možemo navesti put do čvora koji sadrži vrijednost ili možemo koristiti ključnu riječ WHERE, koju nazivamo predikat, da bi suzili izbor među čvorovima.



Slika 3.2 Primjer kako se put do čvora može koristiti kao ključ dohvata određene vrijednosti iz dokumenta. Na primjer put do imena ulice izgleda: Osobe/Osoba[id='123']/Adresa/ulica/ImeUlice/Jarun.

3.2.2. Operacije nad dokument bazama podataka

Osnovne operacije nad dokument bazama podataka su dodavanje, brisanje, ažuriranje zapisa te dohvat podataka. Ne postoji standardni jezik za upravljanje podacima za sve dokument baze podataka, no princip i koncepti su isti iako se implementacije mogu razlikovati.

Naredbom INSERT dodajemo novi dokument u kolekciju dokumenata.

db.collection.insert ($value as item())

Pri dodavanju novog dokumenta u kolekciju dodaje i unikatni identifikator, ako ga korisnik nije sam naveo. Preporuka za unikatne identifikatore je UUID identifikator (*Universally Unique Identifier*). Često je efikasnije dodavati grupe dokumenata u istoj operaciji u bazu, nego svaku pojedinačno.

Metodom REMOVE brišemo sve dokumente iz kolekcije dokumenata ili samo pojedini dokument.

db.collection.remove()

U slučaju da brišemo cijelu kolekciju, ona će nakon brisanja još uvijek postojati u bazi, ali će biti prazna. Da bismo obrisali samo određene dokumente moramo sagraditi upit koji odgovara točno dokumentima koje želimo obrisati.

db.collection.remove ($query as item())

Treba oprezno brisati dokumente koji su možda referencirani u drugim dokumentima.

Jednom kada je dokument dodan u bazu podataka, možemo ga ažurirati pomoću UPDATE metode. Metoda zahtijeva dva parametra: upit te novi skup atributa i njihovih vrijednosti.

Ako zadani atribut ne postoji, metoda će ga sama dodati u dokument i postaviti mu vrijednost. Ako atribut postoji, onda mu se ažurira vrijednost na novododanu.

db.collection.update ($query as item(), $value as item())

Želimo li dohvatiti sve dokumente iz kolekcije ili pak samo specifične, koristimo FIND metodu. Metoda opcionalno prima upit kao parametar kojim specificiramo dokumente koje želimo dohvatiti.

db.collection.find()

db.collection.find ($query as item())

Ovim metodama vraćaju se svi atributi i njihove vrijednosti određenog dokumenta. Želimo li dohvatiti samo pojedine atribute, dodajemo metodi FIND još jedan parametar - listu atributa koje želimo dohvatiti s oznakom "1" uz njih.

db.collection.find ($query as item(), { $key : 1 })

S obzirom na to da se elementi dokumenta automatski indeksiraju moguće je obavljati postupke združivanja nad njima kao i složenije upite kao što su upiti uspoređivanja (manji od, veći od, između dvije vrijednosti i slično). Kompliciranije upite moguće je konstruirati korištenjem Booleovih operatora. Dokument baze podataka omogućuju i traženje podudarnosti s regularnim izrazima ili mogućnost jezičnog pretraživanja teksta (*full-text search*) po zadanim pravilima jezika gdje se kao rezultat vraćaju zapisi koji najbolje odgovaraju zadanom upitu.

Mnoge dokument baze podataka omogućuju stvaranje korisnički definiranih funkcija - UDF (*User Defined Functions*) koje se izvode na poslužitelju. One prihvaćaju skup dokumenata koje zadovoljavaju određeni upit i obavljaju operacije združivanja nad dokumentima. Neke od tih operacija su standardna devijacija, prosjek ili neku drugu operaciju koja kao izlaz daje skalarnu vrijednost. Ove operacije su brze jer se izvode nad indeksima te nije potrebno dohvaćati svaki dokument s diska. Često se nad skupovima rezultata koriste indeksi raspona u svrhu sortiranja i filtriranja dobivenih podataka.

Neke dokument baze dopuštaju upite preko pogleda (*views*). Imamo li puno upita na bazu možda nije optimalno za svaki upit izračunavati vrijednosti. Umjesto toga, dodamo materijaliziran pogled koji unaprijed izračuna vrijednosti te spremi rezultate u bazu. Takvi pogledi se ažuriraju pri upitu u slučaju da je u međuvremenu došlo do promjena.

3.3. Oblikovni obrasci dokument baza podataka

Pravila oblikovanja dokument baza podataka nisu formalna. Samim pogledom na bazu podataka nije moguće reći hoće li baza podataka biti učinkovita u izvođenju operacija. Potrebno je prvo razmisliti kakvi će se generalno upiti postavljati na bazu, koliko veliki podaci će se unositi te koliko često se obavlja ažuriranje. U praksi oblikovanja baza podataka često je korišteno nekoliko oblikovnih obrazaca.

Denormalizacija

NoSQL baze podataka generalno ne omogućavaju spajanje vrijednosti iz više referenciranih dokumenata (*join*), iako neke dopuštaju stvaranje pogleda u vrijeme čitanja dokumenta. Umjesto join operacija koristi se denormalizacija. Denormalizacijom se isti podaci dupliciraju i spremaju u više dokumenta u svrhu bržeg pristupa podacima. Denormalizacija se može izvršavati odmah po unosu novog podatka, i time su novi podaci odmah dostupni u svim dokumentima. Kako bi se to postiglo koriste se okidači u vrijeme spremanja podataka (*pre-commit trigger*).

Ako konzistentnost novih podataka nije toliko bitna koliko brzina unosa, denormalizacija se može izvršavati nakon što je podatak uspješno spremljen u bazu podataka (*post-commit trigger*). Dakle prvo se izvršava operacija unosa podatka, a tek nakon toga se podatak duplicira u ostale dokumente. Time se ostavlja mali vremenski prostor u kojem su podaci u ostalim dokumentima nekonzistentni s novo unesenim ili osvježenim dokumentom.

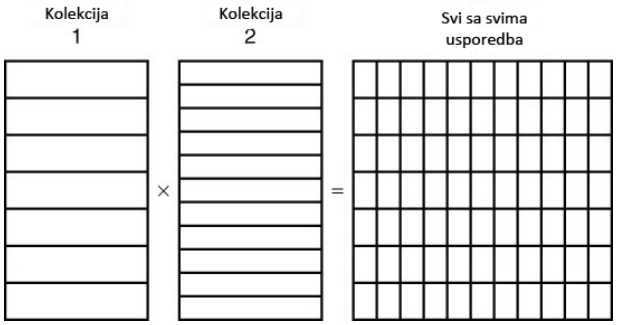
No previše denormalizirani podaci mogu izazvati poteškoće u radu sustava. U takvim podacima mogu se pojaviti anomalije jer je svaku izmjenu denormaliziranog podatka potrebno ažurirati na više lokacija. Također, spremanje velikih dokumenata u memoriju je neučinkovito u slučaju da se dohvaća često samo mali postotak tih dokumenta. Potrebno je naći optimalnu ravnotežu normalizacije i denormalizacije na temelju domenskih znanja o vrsti često postavljanih upita na bazu.

U situaciji kada je potrebno držati bazu normaliziranom, to jest da se povezani podaci spremaju u više kolekcija, join operacije mogu se implementirati u aplikacijskom kodu.

Najgori slučaj korištenja operacija join jest nad dvije velike kolekcije s dvije for petlje na način:

```
for doc1 in collection1:
    for doc2 in collection2:
        <neki posao nad oba dva dokumenta>
```

Radi li se o kolekcijama koje sadrže N i M dokumenata respektivno, tada se naredba izvodi N × M puta. Vrijeme izvođenja ovakvih petlji može jako brzo krenuti rasti. Konkretno sadrži li prva kolekcija 100 000 dokumenata, a druga 500 000 dokumenta, tada se naredba izvodi 50,000,000,000 puta. Ako su kolekcije u bazi velike, potreban je proces indeksiranja, filtriranja i sortiranja podataka u svrhu optimizacije izvođenja join operacija.



Slika 3.3 Join operacije koje uspoređuju sve dokumente u jednoj kolekciji s dokumentima u drugoj kolekciji mogu dovesti do lošeg vremena izvođenja nad velikim kolekcijama dokumenata

Promjenjivi dokumenti

Pri oblikovanju kolekcija dokumenata treba uzeti u obzir i dokumente koji se često ažuriraju te za koje je vjerojatno da će im se veličina mijenjati kroz vrijeme. Takve dokumente nazivamo promjenjivim (*mutable*) dokumentima. Kada stvaramo dokument, sustav upravljanja bazom podataka alocira određenu količinu prostora za taj dokument. Obično uz prostor potreban samom dokumentu sustav zauzme i dodatan dio prostora u slučaju da se veličina dokumenta promjeni. No preraste li dokument količinu prostora koja mu je namijenjena, cijeli dokument je potrebno pročitati, premjestiti na novu lokaciju te osloboditi prostor koji je bio zauzimao.

Jedan način da se ovaj postupak izbjegne jest pri stvaranju dokumenta zauzeti dovoljno prostora uzimajući u obzir budući rast dokumenta. Dakle poželjno je uzeti u obzir moguće velike promjene u veličini dokumenta i planirati za takve situacije.

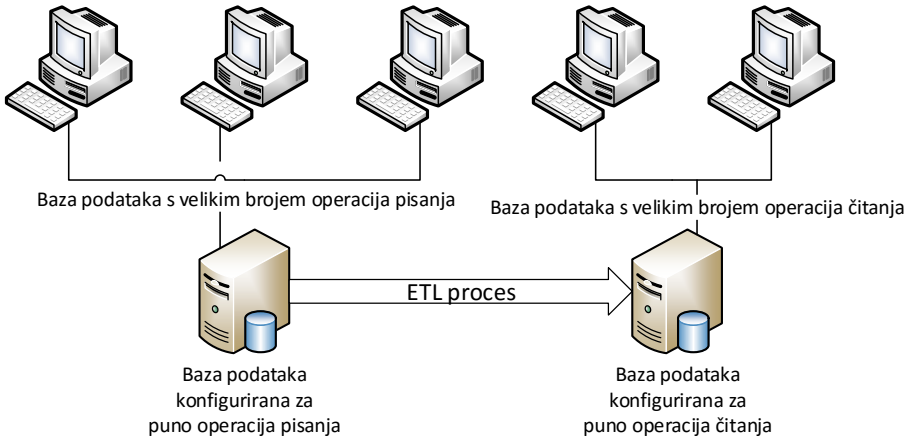
Zlatna zona broja indeksa

Pri oblikovanju dokumenta potrebno je razmisliti o optimalnom broju indeksa nad njima. Premali broj indeksa dovodi do slabijeg izvođenja operacija čitanja, dok prevelik broj indeksa dovodi do slabijeg izvođenja operacija pisanja. Ponekad aplikacije imaju visoki broj zahtjeva za čitanje u odnosu na pisanje nad bazom. Može biti teško procijeniti koja polja indeksirati u svrhu optimalnog filtriranja rezultata, posebno ako uzorak upita nije poznat. Zato ovakve aplikacije obično imaju velik broj indeksa, gotovo na svim poljima koja bi se mogla koristiti pri dohvaćanju rezultata upita.

U slučaju da aplikacije imaju velik broj zahtjeva za pisanje nad bazom, poželjno je minimizirati broj indeksa nad dokumentima. Indeksi su strukture podataka koje zahtijevaju ažuriranje. Njihovo korištenje zauzima procesorske, memorijske i diskovne resurse te povećavaju vrijeme potrebno za dodavanje i ažuriranje dokumenata nad bazom. U takvim slučajevima, obično se ključnim indeksima smatraju oni nad poljima koje čuvaju identifikatore povezanih dokumenata. Ako se može tolerirati sporiji dohvat podataka u svrhu bržeg unosa, minimalni broj indeksa jest dobra praksa.

Međutim, ako je bitno imati brze upite nad bazom podataka koja je zahtjevna za pisanje, dobra je praksa implementirati još jednu bazu podataka koja agregira podatke s obzirom na to koliko su vremenski zahtjevni upiti za čitanjem. Tako funkcioniraju veliki poslovni sustavi. Sustavi za analizu i obradu transakcija oblikovani su za brz unos podataka, i brzo dohvaćanje ciljanih upita za čitanjem.

Podaci se izvlače, transformiraju i spremaju iz produkcijske baze u skladište podataka (*data warehouse*) što se naziva ETL (E*xtraction, Transformation, Load*) proces. Nova baza je obično dobro indeksirana u svrhu bržeg dohvaćanja i analize podataka.



Slika 3.4 Podrška aplikacijama koje su zahtjevne za čitanje i pisanje najbolje je ostvariva korištenjem dviju baza podataka

Oblikovanje veza među dokumentima

Pri oblikovanju baze podataka, često se javlja potreba za nekim od sljedećih tipova veza:

- Jedan-na-mnogo veze (*One-to-many relations*)
- Mnogo-na-mnogo veze (*Many-to-many relations*)
- Hijerarhijske veze

Jedan-na-mnogo veze najjednostavnije su od navedene tri vrste veza. Ova veza nastaje kada jedna instanca entiteta (objekt) sadrži jednu ili više instanci drugog entiteta. Primarni entitet je oblikovan kao dokument koji sadrži polje umetnutih dokumenata, koji predstavljaju povezane entitete.

Mnogo-na-mnogo veza nastaje kada jedna instanca entiteta sadrži jednu ili više instanci drugog entiteta, ali i instanca drugog entiteta sadrži jednu ili više instanci prvog entiteta. Ovaj odnos oblikujemo koristeći dvije kolekcije – jednu za svaki tip entiteta. Svaka kolekcija održava listu identifikatora koji su referenca na povezane entitete. Ovaj obrazac minimizira dupliciranje podataka tako da stavlja referencu na povezan dokument pomoću identifikatora umjesto umetanjem dokumenata. Ovakve veze se moraju pažljivo ažurirati jer dokument baze podataka ne javljaju greške u slučaju krivog referenciranja.

Hijerarhije opisuju instance entiteta koje su u roditelj-dijete vezi. Jednostavna tehnika oblikovanja ovog odnosa jest da se čuvaju reference na roditelja ili dijete pojedinog entiteta. Ovaj obrazac je koristan za slučaj kada je često potrebno prikazati specifičnu instancu, a zatim i generalizirani tip te kategorije, ili kada je potrebno dohvatiti djecu instance.

Umjesto navođenja roditelja u dokumentu-djetetu mogu se čuvati liste svih predaka, što je korisno kada je potrebno znati cijeli put u hijerarhiji od bilo kojeg čvora do korijena. Prednost je da se takav put može dohvatiti jednom operacijom čitanja, dok se korištenjem obrasca roditelj-dijete operacije čitanja povećavaju za svaki dodatni nivo u hijerarhiji. Mana ovog pristupa je da se promjenom unutar hijerarhije mora ažurirati svaki čvor koji je niže u hijerarhiji što može značiti puno operacija pisanja.

3.4. Upravljanje dokument bazama podataka

Poslovni sustavi obično zahtijevaju sigurnost po pitanju pohrane i dostupnosti podataka. Ako baza odgovori da su određeni podaci pohranjeni, želimo da onda zaista i jesu pohranjeni. Također, novi podaci moraju biti dostupni odmah po završetku ažuriranja, kao i indeksi nad tim podacima.

Distribuirane baze podataka često imaju poboljšano vrijeme operacija pisanja, na štetu izvođenja operacija čitanja i upita. To je zato što upiti moraju proći kroz više poslužitelja.

3.4.1. Raspodjela podataka unutar grozda

Ideja iza skaliranja – raspodjele podataka, jest dodavanje novih čvorova u grozd u svrhu da sustav može podnijeti veće opterećenje. Ne želimo migrirati cijelu bazu na veći sustav, niti želimo raditi promjene u aplikaciji da bolje podnosi opterećenje. Promatramo koje mogućnosti pruža sama baza podataka u svrhu boljeg podnošenja opterećenja. Skaliranje se obavlja repliciranjem ili horizontalnim particioniranjem (*sharding*) podataka.

Skaliranje baza podataka koje imaju mnogo zahtjeva za čitanje obavlja se često repliciranjem - dodavanjem više replika za čitanje. Dodavanjem novog čvora u ovakav skup replika, ostali se čvorovi ne moraju ponovno pokrenuti niti aplikacija mora izgubiti vezu sa sustavom replika.

Želimo li skalirati baze koje imaju puno zahtjeva za pisanjem, možemo to učiniti raspodjelom podataka horizontalnim particioniranjem. Ovim postupkom podaci se dinamički premještaju među čvorovima i time osiguravaju ravnotežu opterećenja. Poslužitelji u grozdu imaju istu shemu podataka. Dodavanjem novog čvora u sustav podaci se raspodjeljuju na čvor više. Dok se podaci premještaju, aplikacija ne gubi mogućnost rada nad bazom.

Ako se velik broj novih podataka sprema na zadnje dodani poslužitelj u grozdu, operacije pisanja će se usporiti, no to vrijedi i za operacije čitanja jer će poslužitelj imati problema s praćenjem zahtjeva. Želimo osigurati da se podaci šire ravnomjerno unutar grozda poslužitelja. Postoje različiti pristupi raspodjele podataka unutar distribuiranih dokument baza podataka.

1. Raspodjela pomoću ključa

Pri raspodjeli podataka pomoću ključa, ključem dokumenta (URI, ili neki drugi identifikator) se određuje na kojem će se poslužitelju spremiti određen dokument. Kao i kod ključ-vrijednost baza podataka, moguće je zadati raspon vrijednosti ključeva svakom poslužitelju u grozdu. Temeljem ključa i podatka o rasponu ključeva svakog poslužitelja, klijentski program može odrediti točno s kojim poslužiteljem mora komunicirati da bi dohvatio pojedini dokument.

2. Automatska raspodjela

Automatskom raspodjelom podataka, baza podataka nasumično pridružuje novi dokument nekom od poslužitelja. To znači da programer ne mora više pažljivo odabrati ključeve i pravila raspodjele po poslužiteljima da bi se osigurala dobra propusnost pisanja podataka u bazu. Dođe li do potrebe za dodavanjem novog poslužitelja u sustav, automatskom raspodjelom particije se rebalansiraju među svim poslužiteljima. Takvom raspodjelom, umjesto promjenom raspona ključeva poslužitelja i održavanjem fiksnog broja dokumenata po poslužitelju, dovoljno je preseliti pojedini dokument na novi poslužitelj. Dakle zahtijeva ukupni manji broj premještanja dokumenata nego raspodjela po ključu.

Replike takvih particija, osim što osiguravaju dostupnost podataka, dopuštaju još i obavljanje upita nad njima.

Time se teret operacija čitanja dijeli među particijom i njenim kopijama, to jest postiže se paralelizacija čitanja te se poboljšava izvođenje upita za cijeli grozd.

3.4.2. Repliciranje podataka

Jednom kada su podaci trajni, dakle pohranjeni na disk poslužitelja, prestane li poslužitelj s radom podaci su sigurni ali nedostupni. Zato se često koristi postupak replikacije unutar grozda. Replikaciju je moguće obaviti:

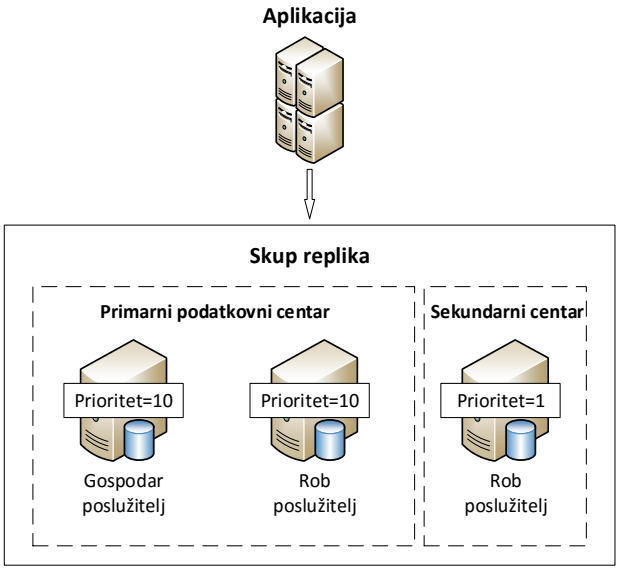
- Unutar transakcije (dvofazno izvršenje): u tom slučaju sve replike imaju ažurni prikaz podataka u bazi
- Nakon što transakcija završi, što znači da će se replike ažurirati u nekom trenutku u budućnosti. Takvu replikaciju nazivano naknadno konzistentnom. Tipično nedosljednost traje nekoliko sekundi.

U procesu replikacije baze podataka, ponekad je korisno podatke (sve, djelomično ili po prioritetu) replicirati u novi podatkovni centar na drugoj fizičkoj lokaciji. Time su podaci dostupni i u slučaju da cijeli podatkovni centar naiđe na probleme. Ovaj proces se obično odvija asinkrono kao kompromis brzine pisanja i konzistentnosti podataka na udaljenom podatkovnom centru. Zbog asinkronog procesa neki podaci na udaljenom centru nisu dostupni odmah po operaciji pisanja u primarni podatkovni centar. Dogodi li se pad primarnog podatkovnog centra, zamjenski centar preuzima operacije pisanja na bazu. Te promjene naknadno se moraju spojiti (ali ne replicirati) s podacima primarnog centra. Takvi postupci mogu stvoriti konflikte među podacima koje je potrebno ručno razriješiti jednom kada se vrati komunikacija među centrima.

Replikacija se obično odvija koristeći gospodar-rob model replikacije. Isti podaci dostupni su na više čvorova tako da je pristup podacima moguć čak i kad gospodar poslužitelj nije dostupan. Svi zahtjevi dolaze prvo na gospodara, a zatim se podaci repliciraju na robove. U slučaju prestanka rada gospodara poslužitelja, replike među sobom izabiru novog gospodara.

Ne moraju svi čvorovi imati isto pravo glasa - moguće im je zadati vrijednost prioriteta.

Neke čvorove se može favorizirati zbog njihove blizine ostalim poslužiteljima, ili zbog veće količine radne memorije. Kada je poslužitelj opet dostupan za rad, pridružuje se poslužiteljima robovima te ažurira svoje podatke bez potrebe da se skup replika mora ugasiti .



Slika 3.5 Konfiguracija skupa replika unutar podatkovnih centara s pridruženim prioritetima pojedinih čvorova

Iako se operacije čitanja i pisanja obavljaju nad primarnim čvorom, aplikacija ne mora znati koji je čvor u skupu replika primarni. To jest aplikacija ne mora znati strukturu gospodar-rob modela. Također u slučaju nedostupnosti primarnog čvora, komunikacija se nastavlja bez potrebe sa znanjem o greškama u komunikaciji čvorova ili o procesu odabira novog primarnog čvora. Skupovi replika obično se koriste za visoku dostupnost podataka, redundantnost podatka, automatski oporavak od greške, skaliranje operacija čitanja te održavanje poslužitelja bez potrebe za prestankom rada sustava.

3.4.3. Upravljanje s konzistentnošću podataka

Postoji cijeli raspon konzistentnosti dokument baza podataka, svaki primjenjiv u ovisnosti o potrebama i zahtjevima konkretne implementacije. Nivo konzistentnosti moguće je drugačije postaviti za svaku različitu operaciju unutar baze podataka.

Naknadnom konzistentnošću, operacija pisanja se smatra uspješnom na poslužitelju koji ju je zaprimio, no poslužitelji-replike nisu ažurirani u trenutku kada se ažurira gospodar poslužitelj. Ažuriraju se naknadno ovisno o postavljenim parametrima replikacije sustava. Ovakav model je prihvatljiv za sustave društvenih mreža jer nije neophodno da je svaka promjena dostupna u stvarnom vremenu. Također, time da se ne mora čekati na ažuriranje replika, to jest ubrzana je brzina pisanja na bazu. Nekonzistentnost obično traje par sekundi dok replike ne ažuriraju nove podatke.

ACID konzistentne baze podataka dopuštaju izvršavanje više operacija unutar iste transakcije. Te operacije se izvršavaju na primarnom poslužitelju i zatim na replikama, i tek tada je transakcija završena. Ako se neka operacija nije izvršila, sve promjene na replikama se vraćaju u svoje prvotno stanje, dakle stanje u kojem su bile prije nego što je transakcija započela (*roll-back*). Time je osigurana konzistentnost replika.

Garanciju da jednom spremljeni podaci u bazu podataka ostaju sigurni i konzistentni nazivamo trajnost podataka. ACID konzistentni sustavi kao i naknadno konzistentni obično zadovoljavaju svojstvo trajnosti podataka. Trajnost se postiže:

- Spremanjem dokumenta na disk prije nego što se vrati odgovor uspješnog spremanja, dakle spremanje podataka unutar transakcije zapisa. Time se usporava ukupno vrijeme pisanja u bazu podataka.
- Spremanjem dokumenta u memoriju, uz zapis dnevnika promjene na disk. Jedan unos u dnevnik promjene daje opis promjene zapisa na bazi podataka. Time postižemo brzo izvođenje operacija pisanja na bazu s osiguranjem trajnosti podataka.

Odaberemo li opciju pisanja novih podataka u memoriju te asinkrono spremanje na disk bez dnevnika promjene, moguće je izgubiti novo unesene podatke u slučaju da poslužitelj prestane s radom prije nego što su se podaci zapisali na disk. Takva opcija ne osigurava trajnost podataka.

4. Stupčani sustavi za upravljanje bazama podataka

4.1. Uvod u stupčane baze podataka

Velike organizacije kao što su Google, Facebook, Amazon i Yahoo! zahtijevaju od sustava za upravljanje bazama podataka da zadovoljavaju njihove potrebe za spremanjem velikih količina podataka. Inženjeri koji su osmislili Google Bigtable očekivali su potrebu za upravljanjem desecima tisuća stupaca te milijardom redaka. Zahtjevi uključuju i veliku brzinu pristupa podacima, visoku dostupnost preko više podatkovnih centara te fleksibilnost upravljanja podacima. Kao rezultat, stupčani sustavi imaju neke sličnosti s ključ-vrijednost, dokument te relacijskim bazama podataka. Stupčane baze podataka mogle bi se koristiti i nad manjim skupom podataka, no za to nisu predviđene.

Izraz Veliki Podaci (*Big Data*) neformalno opisujemo kao skupove podataka prevelikih da bi se efikasno mogli spremiti i analizirali relacijskim bazama podataka.

Formalniju definiciju dala je Gartner istraživačka grupa[2]. Gartner definira Velike Podatke kao podatke visoke brzine, velikog volumena i visoke promjenjivosti. Brzina se odnosi na brzinu kojom se podaci stvaraju ili mijenjaju. Volumen se odnosi na veličinu i količinu podataka s kojima se radi. Promjenjivost se odnosi na raspon tipova podataka i količine informacija koje podaci sadrže.

Stupčane baze podataka imaju neke zajedničke osobine:

- Softverski inženjeri imaju dinamičku kontrolu nad stupcima.
- Podaci se indeksiraju identifikatorom retka, imenom stupca i vremenskom oznakom.
- Softverski inženjeri imaju kontrolu nad lokacijom podataka unutar grozda.
- Čitanja i pisanja retka su atomarne operacije.
- Redci se čuvaju u sortiranom redoslijedu.

[2] *Laney Douglas, „The Importance of 'Big Data': A Definition"*

4.2. Osnovne značajke stupčanih baza podataka

4.2.1. Indeksiranje po retku, imenu stupca i vremenskoj oznaci

Identifikator retka jedinstveno označava pojedini redak, analogno primarnom ključu u relacijskim bazama podataka. Entiteti su modelirani recima. Jedan redak predstavlja jedan entitet. Atributi entiteta predstavljeni su stupcima. Ime stupca jedinstveno određuje stupac. Vrijednost stupca mogu biti samo primitivni tipovi podataka.

Da bi distribucija podataka bila efikasna i da bi upiti na bazu bili dobro raspoređeni, potrebno je dobro odabrati strategiju za ključ retka. Dobro osmišljen ključ – koji opisuje sadržaj zapisa - omogućuje brz pronalazak zapisa u bazi. Aplikacija ne mora pročitati vrijednosti i provjeravati zadovoljene uvjete svakog pojedinačnog zapisa. Ključevi redaka koriste se i za pravilnu distribuciju zapisa među poslužiteljima. Loše osmišljen ključ dovodi do situacije da jedan poslužitelj zaprima većinu zahtjeva, time usporavajući izvođenje cijelog sustava.

Svaka vrijednost unutar stupca može sadržavati više različitih verzija vrijednosti koje su označene drugačijim vremenskim oznakama. Vrijednosti stupca se ne mijenjaju, samo se dodaju nove vrijednosti s različitim vremenskim oznakama. Generalno se koristi model nadovezivanja to jest dodavanja vrijednosti na kraj zapisa, da bi se poboljšala vremena izvođenja operacija pisanja. Novi podaci tako se dodaju u strukturu, a stare vrijednosti dobivaju oznaku za brisanje (*tombstone marker*). Time sustav zna da u nekom trenutku treba pobrisati označene vrijednosti. Kroz vrijeme se provode spajanja podataka i time se stare vrijednosti prebrišu. Taj proces nazivamo sažimanjem (*compaction*).

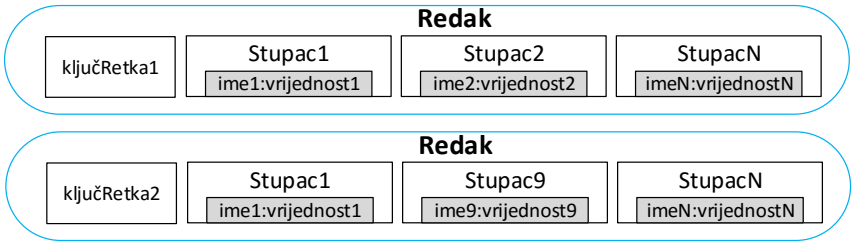
Za baze podataka koje imaju puno zahtjeva za pisanjem, ovakav model vodi do puno spajanja podataka, time i do većeg opterećenja sustava. Stoga stupčane baze dopuštaju spajanja podataka unutar radne memorije čime se smanjuje opterećenje diska i procesora te su podaci brže dohvatljivi. Stupčane baze podataka smanjuju količinu podataka koja se čuva na diskovima tako zvanom blok kompresijom (*block compression*) koja čuva vrijednosti na disku u komprimiranom formatu.

Ovakva kompresija je jednostavna binarna kompresija i obično baze podržavaju barem gzip i LZO (*Lempel, Ziv, Oberhumer*) algoritme kompresije.

4.2.2. Kontrola lokacije podataka

Upitima na bazu često se dohvaćaju blokovi podatka s različitih dijelova diska. Jedan način da se izbjegne takvo dohvaćanje podataka jest čuvanje podataka koje se koriste često u blizini jednih s drugima. Za razliku od relacijskih baza podataka koje spremaju sve vrijednosti pojedinog retka zajedno, stupčane baze podataka spremaju samo dijelove redaka zajedno. Jedan redak može sadržavati više familija stupaca i do ukupno dvije milijarde stupaca. Familije stupaca su logičke grupacije stupaca. Vrijednosti retka i određene familije stupaca pohranjene su zajedno, time omogućujući da se čitanjem jednog bloka podataka može zadovoljiti upit. Familije stupaca vezane uz određen redak mogu ali i ne moraju biti pohranjene zajedno, no stupci unutar iste familije stupaca uvijek su pohranjeni zajedno. Iako se broj stupaca unutar familije može u definiciji tablice mijenjati dinamički – u trenutku izvođenja, familije stupaca se moraju navesti pri definiciji strukture. Tipično se familije stupaca ne mogu mijenjati bez da se poslužitelj privremeno učini nedostupnim.

Mogućnost promjenjivog broja stupaca ima svojih nedostataka. U slučaju da se žele ispisati svi stupci unutar određene familije, potrebno je iterirati po svim recima da bi se dobila kompletna lista. Stoga je potrebno da aplikacija prati model podataka da bi se izbjegle ovakve iteracije.



Slika 4.1 Model podataka unutar familije stupaca

Podaci tablica spremaju se u blokove slijednih redova koje nazivamo tableti (*tablets*), što nam omogućuje distribuirano upravljanje podacima i raspodjelu opterećenja upita po poslužiteljima. Tableti dakle čuvaju grupe zapisa (redaka) za jednu tablicu jedne baze podataka. Tablet je jedinica postojanosti unutar stupčane baze. Tabletom upravlja tablet poslužitelj.

Prestane li on s radom, novi tablet poslužitelj se izabire i dobiva zadatak upravljati podacima tog tableta. Često se tableti raspoređuju po više poslužitelja da se ne bi preopteretio samo jedan poslužitelj. Tablet podataka sadrži dio podataka tablice i kako količina podataka raste, tableti se dodjeljuju novim spremnicima. Svaki spremnik može primiti do 20 GB podataka te svaki tablet poslužitelj može držati do 200 spremnika.

Kako bi riješile problem nadjačavanja podataka (*data override*), stupčane baze podataka podržavaju zaključavanje zapisa (*row locking*). To znači da se polju u retku ne može pristupiti sve dok trenutni proces ne završi s ažuriranjem polja. Bez korištenja procesa zaključavanja, morao bi se koristiti RMU proces koji onemogući cijeli redak za ažuriranje ako neki drugi proces trenutno ažurira polje tog retka.

4.3. Upravljanje podacima stupčanih baza podataka

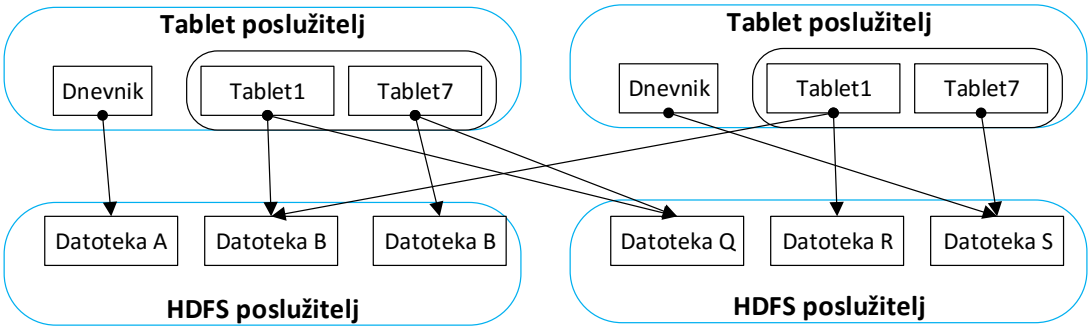
4.3.1. Upravljanje podacima pomoću HDFS sustava

Apache Hadoop je programski paket otvorenog koda korišten za distribuirano spremanje datoteka i distribuiranu obradu podataka.

HDFS (*Hadoop Distributed File System*) je sustav za raspodjelu velikog broja datoteka po grozdu poslužitelja. Uobičajeno čuva tri kopije podataka (primarnu kopiju, repliku unutar grozda, replika unutar drugog grozda), time osiguravajući redundantnost. HDFS sam po sebi ne dopušta promjene nad dijelovima pohranjenih zapisa. Da bi se omogućilo indeksiranje i mogućnost ažuriranje potrebno je imati NoSQL bazu podataka koja radi nad HDFS sustavom.

Slika 4.2 prikazuje poslužitelje stupčane baze podataka koji pohranjuju podatke na Hadoop HDFS particije po Hadoop grozdu.

Umjesto čuvanja puno malih zapisa, stupčana baza podataka čuva manji broj većih zapisa. Stupčana baza daje HDFS sustavu mogućnost indeksiranja i pronalaska manjih zapisa među velikim skupinama podataka.



Slika 4.2 Prikaz rada tablet poslužitelja nad HDFS sustavom

Baza podataka interpretira zapise kao zasebne jedinice, no zapravo su oni podijeljeni po HDFS sustavu. Podaci se stoga brže dohvaćaju jer se operacije odvijaju paralelno na više poslužitelja.

4.3.2. Analiza i obrada podataka pomoću MapReduce sustava

Uz visoko distributivan i paraleliziran sustav, Hadoop daje i mogućnost analize i obrade podataka pomoću Hadoop MapReduce sustava.

Generalno upite možemo podijeliti na one čiji rezultat želimo da nam bude dostupan čim prije te na one gdje treba vremena da se do rezultata dođe. Ponekad se skup rezultata upita mora dalje obraditi i analizirati da bi se došlo do konkretnog zapisa.

Hadoop MapReduce sustav omogućava takvu funkcionalnost. MapReduce posao sastoji se tipično od dvije operacije:

1. Map operacija kojom se prolazi kroz skupove zapisa te se filtrira i vraća relevantne zapise zajednički u traženom formatu (mapiranje)
2. Reduce operacija koja iz dobivenih rezultata vraća odgovor na upit (reduciranje)

Kao još složeniji MapReduce posao, moguće je rezultat jedne operacije staviti kao ulaz druge MapReduce operacije. Time se dopušta nizanje operacija.

4.3.3. Upiti

Upiti korišteni nad bazom daju važne informacija za oblikovanje stupčanih baza podataka. S obzirom na to da stupčane baze drže retke u sortiranom poretku, jednostavno je postaviti upit raspona nad zapisima u bazi.

Neki od osnovnih upita su GET, SET i DEL za dohvaćanje, postavljanje i brisanje zapisa. Postavljanje novih vrijednosti obavlja se SET naredbom, kao i ažuriranje postojećih vrijednosti.

SET ColumnFamilyName[$rowId as String][$columnId as String] = $value as item();

Dohvat podataka obavlja se GET naredbom. Možemo dohvatiti cijelu familiju stupaca, ili samo jedan stupac.

GET ColumnFamilyName [$rowId as String];

GET ColumnFamilyName [$rowId as String][$columnId as String];

Korištenjem DEL naredbe brišemo cijelu familiju stupaca ili samo pojedini stupac.

DEL ColumnFamilyName [$rowId as String];

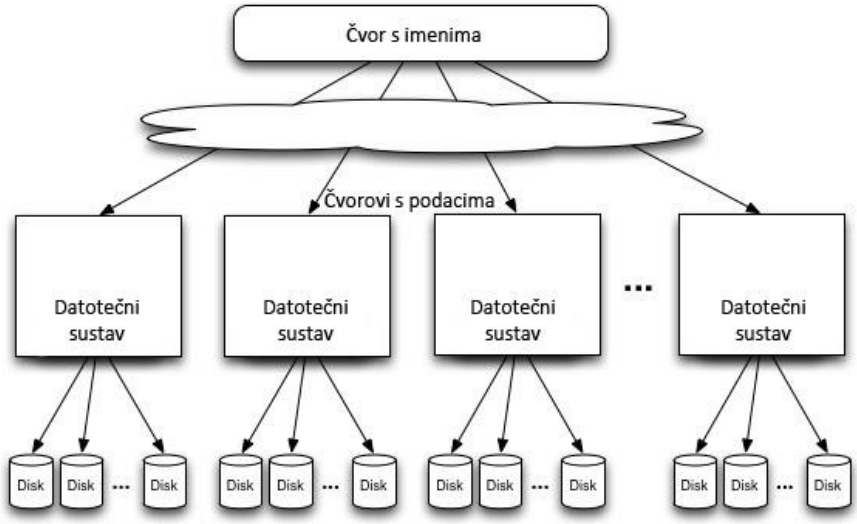
DEL ColumnFamilyName [$rowId as String][$columnId as String];

Stupčane baze podataka pohranjuju zapise upravljanjem skupom ključeva. Ako nas zanima koji ključevi u bazi imaju određenu vrijednost, moramo postaviti upit svakom poslužitelju. Upravlja li svaki poslužitelj milijunima zapisa takvi upiti bi mogli potrajati. Umjesto toga, koristimo Bloom filtere, nazvane po Burtonu Howardu Bloomu koji je predložio tehniku 1970. godine. Bloom filteri mogu se dodati nad imenima stupaca. Umjesto prolaska kroz cijelu bazu podataka, Bloom filter nam daje informaciju da bi određeni redak mogao biti u skupu rezultata, ili da sigurno nije u skupu rezultata. Daljnjom analizom rezultata Bloom filtera uvelike se smanjuje broj ključeva kroz koje treba proći te vrijeme upita i broj operacija nad diskom. Bloom filteri koriste nešto više memorije poslužitelja, ali su također podesivi. Može se podesiti gledana vjerojatnost da je redak zaista dio skupa rezultata.

Vrati li filter pozitivan rezultat za pojedini ključ koji nije dio rezultata, to nazivamo lažna potvrda (*false-positive match*). Promjenom vrijednosti lažne potvrde s 0.01 na 0.1 moglo bi umanjiti utrošak memorije na pola.

4.4. Arhitektura stupčanih baza podataka

Česta su dva tipa arhitekture visoko distribuiranih sustava: arhitektura višestrukih čvorova (*variety of nodes*) te arhitektura ravnopravnih čvorova (*peer-to-peer*). Arhitektura višestrukih čvorova obično je postavljena nad Hadoop sustavom i koristi višestruke Hadoop čvorove – čvorove imena (*name nodes*), čvorove podataka (*data nodes*) te centralizirani čvor koji održava konfiguracijske podatke o grozdu. Arhitektura ravnopravnih čvorova ima samo jednu vrstu čvorova. Bilo koji čvor može preuzeti odgovornost za bilo koji servis ili zadatak koji se mora izvršiti u grozdu.



Slika 4.3 Hadoop HDFS arhitektura

4.4.1. Arhitektura višestrukih čvorova

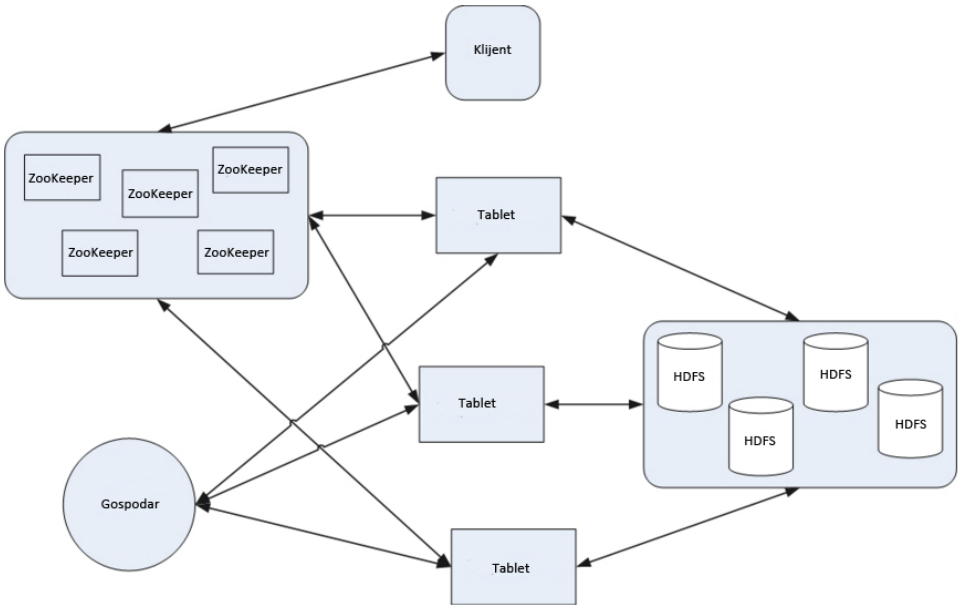
Hadoop HDFS sustav koristi gospodar-rob arhitekturu koja se sastoji od čvorova imena te čvorova podataka. Čvorovi imena upravljaju podatkovnim sustavom i omogućuju centralizirano upravljanje metapodacima. Čvorovi podataka čuvaju podatke te ih repliciraju kako konfiguracija nalaže.

Zookeeper jest centralizirani poslužitelj koji omogućuje koordinaciju među čvorovima unutar Hadoop sustava. On održava zajednički hijerarhijski imenički prostor te je zajednička točka kvara cijelog sustava. Stoga se podaci Zookeeper čvora repliciraju na višestruke čvorove.

Baza podataka uz korištenje Hadoop servisa, unutar svojih procesa upravlja i metapodacima o distribuciji podataka. O različitim tablet poslužiteljima vodi brigu gospodar poslužitelj.

Pri klijentskom upitu za čitanjem ili pisanjem, klijent može prvo pitati Zookeeper poslužitelja za ime poslužitelja koji sadrži informacije o lokaciji tableta unutar grozda. Kako bi se ubrzao daljnji rad s podacima, klijent može privremeno spremiti podatke o tabletu te izbjeći ponovnu komunikaciju sa Zookeeper poslužiteljem. Klijent dalje radi upite prema tom tablet poslužitelju kako bi odredio lokaciju potrebnih podatka (ako se radi o procesu čitanja) ili koji poslužitelj treba primiti podatke (ako se radi o pisanju).

Prednost ovakve arhitekture je da se poslužitelji mogu konfigurirati za specifične zadatke kako bi te zadatke obavljali što efikasnije. Ali zahtijeva održavanje više različitih konfiguracija.



Slika 4.4 Arhitektura stupčane baze podataka nad HDFS sustavom

4.4.2. Arhitektura ravnopravnih čvorova

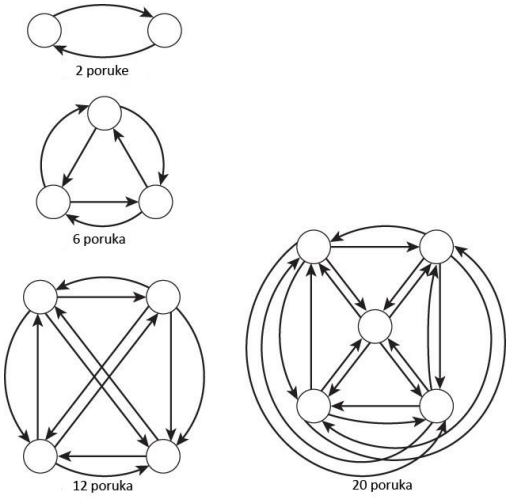
Umjesto korištenja hijerarhijske strukture s fiksiranim funkcionalnostima po poslužitelju, neke implementacije sustava koriste model gdje svaki čvor može preuzeti bilo koju potrebnu ulogu unutar sustava. U modelu ravnopravnih čvorova, svi čvorovi koriste isti softver, no mogu imati različite uloge unutar grozda. Jedna od pogodnosti ovakvog pristupa jest jednostavnost. Niti jedan čvor ne može biti zajednička točka kvara sustava. Skaliranje se obavlja jednostavno - dodavanjem ili micanjem čvorova iz grozda. Kada je čvor maknut iz sustava, ostali čvorovi koju su čuvali njegove replike nastavljaju dalje brigu o podacima maknutog čvora. S obzirom na to da sustav nema gospodara poslužitelja, poslužitelji su odgovorni za brigu o nizu operacija:

- Dijeljenju informacija o stanju poslužitelja u grozdu
- Osiguravanju da čvorovi imaju ažurne podatke
- Osiguravanju da se podaci sačuvaju u slučaju da poslužitelj koji ih je trebao prihvatiti nije dostupan

Postoje protokoli koji implementiraju ove funkcionalnosti.

Protokol razgovora

Dijeljenje informacija o stanju poslužitelja u grozdu zvuči kao trivijalan problem. Jednostavno, svaki poslužitelj može poslati poruku svakom drugom poslužitelju kada ima za podijeliti relevantnu informaciju. Time bi se volumen prometa na mreži povećao svakim dodavanjem novog poslužitelja te bi poslužitelji trošili puno vremena na komuniciranje međusobno. Uzmimo za primjer grozd od 100 čvorova. U tom slučaju potrebno je poslati 9900 poruka za ažuriranje informacija o statusu. To jest imamo li N poslužitelja, potrebno je N×(N–1) poruka da bi svi poslužitelji imali najnoviju informaciju o ostalima.



Slika 4.5 Broj poruka među poslužiteljima brzo raste sa svakim novododanim čvorom

Efikasnija metoda dijeljenja informacija je da svaki poslužitelj ažurira jednog poslužitelja o svojim podacima kao i podacima poslužitelja za koje zna. Taj poslužitelj tada može učiniti istu stvar te podijeliti tu informaciju (o svojim vlastitim podacima) s drugim skupom poslužitelja za koje on zna. Protokol razgovora generalno implementira nasumičan odabir čvorova komunikacije pa postoji doza redundantnosti u dostavljanju informacija. Umjesto nasumičnog odabira postoje i deterministički protokoli odabira. U svakom slučaju podaci se agregiraju u manji broj poruka čime se postupak slanja izvodi efikasnije.

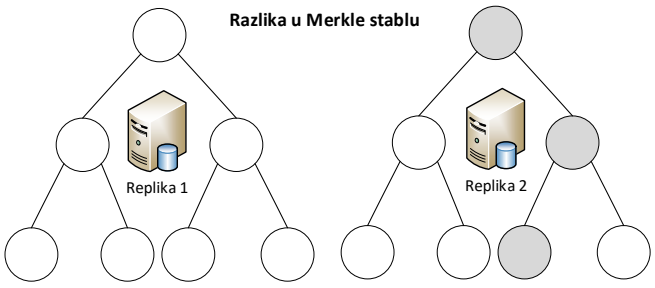
Primjer jednog takvog slanja poruka:

- Čvor u grozdu započinje protokol razgovora s nasumično odabranim čvorom.
- Čvor šalje poruku koja označava početak razgovora odabranom čvoru.
- Odabrani čvor kome je poruka namijenjena šalje poruku da je prihvatio poruku.
- Nakon primitka potvrde odabranog čvora, inicijalni pošiljatelj šalje završnu poruku odabranom čvoru.

Tijekom ove razmjene poruka svaki poslužitelj je dobio ažurirane podatke o svim poslužiteljima za koje poslužitelji koji međusobno razgovaraju znaju. Uz to, šalju se i informacije o verzijama podataka na svakom poslužitelju. Time svaki poslužitelji zna koji od njih dvoje ima ažurne podatke o ostalim poslužiteljima.

Anti-entropija

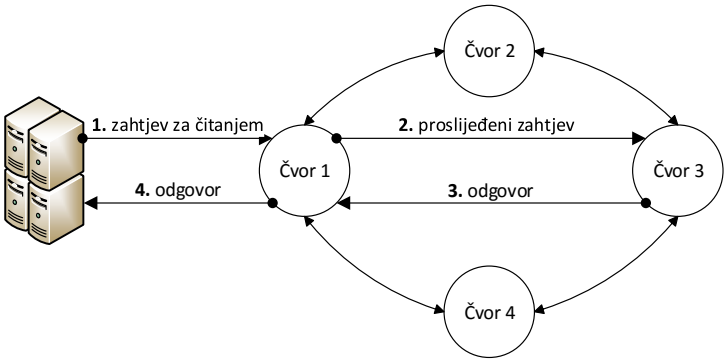
Entropija (neuređenost) podataka raste kada je baza podataka u nekonzistentnom stanju. Postoje algoritmi uređenja (*anti-entropy algorithms*) koji rješavaju ovaj problem - dakle povećavaju uređenost podataka, u svrhu popravka nekonzistentnosti među replikama. Kada poslužitelj započne proces uređenja s drugim poslužiteljem, šalje mu podatkovnu strukturu poznatu pod nazivom Merkle ili raspršno stablo (*hash tree*). Poslužitelj koji je primio strukturu stvara raspršno stablo vlastite kopije podataka. Ako se dva stabla ne poklapaju, poslužitelji utvrđuju koji od njih sadrži ažurne informacije te onaj koji ne sadrži prihvaća ažuriranje svojih podataka.



Slika 4.6 Usporedba podataka u svrhu očuvanja konzistentnosti među replikama

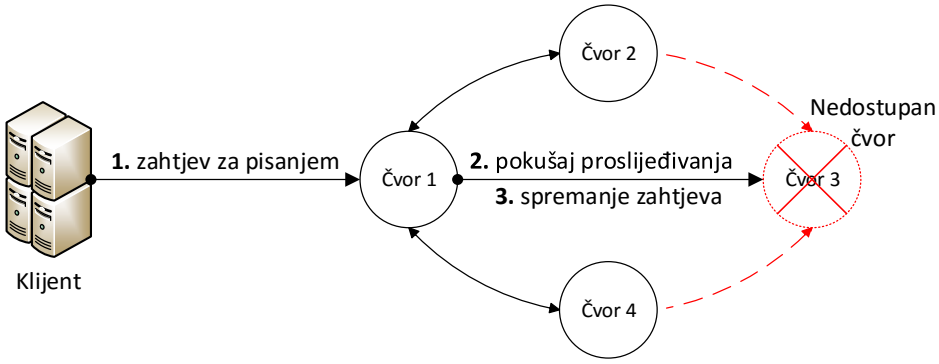
Predaja podataka

Visoka dostupnost podataka između ostalog znači da sustav može prihvatiti zahtjeve za pisanjem čak i kada poslužitelj zadužen za te podatke nije dostupan.



Slika 4.7 Bilo koji čvor u grozdu može prihvatiti klijentski zahtjev te ga proslijediti odgovarajućem poslužitelju

Dogodi li se takva situacija da poslužitelj zadužen za pisanje ne odgovara na zahtjev, umjesto da se gube podaci ili da se trajno promjeni lokacija spremanja podatka za pojedini ključ, inicira se protokol predaje podatka (*hinted handoff*). Proces pohranjuje sve informacije vezane za zapis (lokaciju čvora koji nije dostupan, metapodatke o verziji, same podatke koje treba zapisati) u posebnu tablicu. Jednom kada čvor opet postane dostupan, čvor koji je prihvatio njegove zahtjeve mu predaje podatke o svim promjenama.



Slika 4.8 U slučaju nedostupnosti čvora, ostali čvorovi prihvaćaju zahtjeve nedostupnog čvora i prosljeđuju mu podatke jednom kada postane dostupan

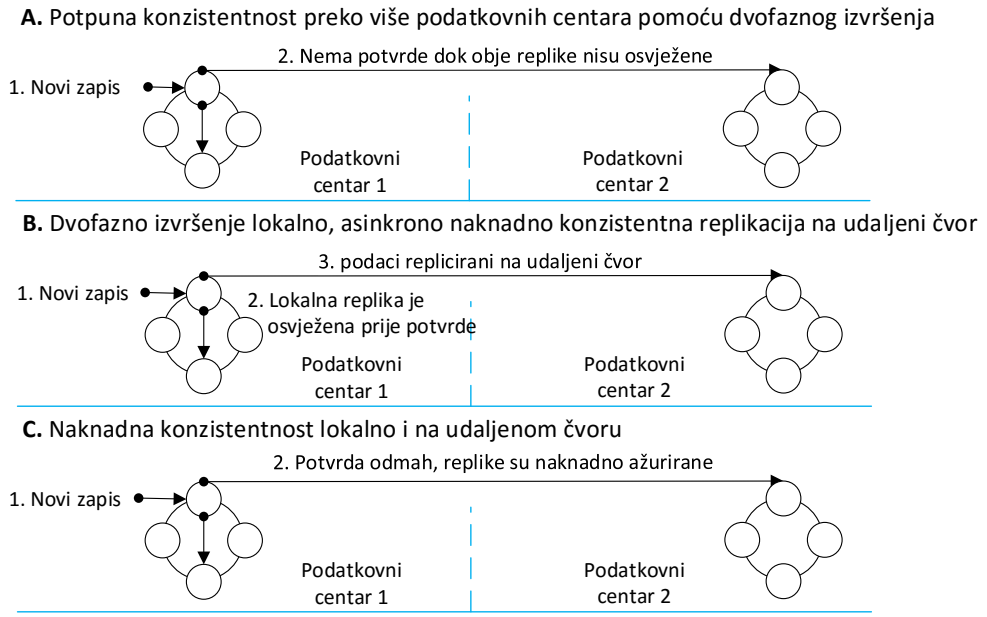
4.5. Upravljanje podatkovnim centrima

U velikim poslovnim sustavima potrebno je imati zaseban podatkovni centar koji služi za čuvanje kopija podataka, što se obično naziva mjesto oporavka od katastrofe (*disaster recovery site*). Praksa je koristiti više slabijih i jeftinijih računala umjesto jednog izrazito jakog. Razlog tome je bolje ukupno izvođenje sustava po novčanoj jedinici. Uz mogućnost lakšeg skaliranja, više poslužitelja čini sustav otporniji na greške – u slučaju da poslužitelj prestane raditi, drugi će preuzeti njegov posao.

4.5.1. Aktivan-aktivan raspodjela zapisa

Aktivan-aktivan raspodjela (*active-active clustering*) zapisa neophodna je za sustav kojem su potrebne brze operacije pisanja u podatkovnim centrima na udaljenim lokacijama u svrhu distribucije zahtjeva najbližem poslužitelju. Tipično se ovakav grozd sastoji od barem dva čvora, gdje svaki pruža iste vrste servisa i sadrži iste podatke. Postupak replikacije se može odvijati na više načina.

- Čvor na kojem se dogodila promjena čeka s potvrdom transakcije dok se promjena ne propagira na ostale replike. Time se usporava ukupno vrijeme promjene. Tim postupkom je osigurana konzistentnost i postojanost podataka na svim čvorovima. (Slika 4.9 A)
- Čvor na kojem se dogodila promjena čeka s potvrdom transakcije dok se promjena ne propagira na lokalne replike, udaljene replike se usklađuju asinkrono. Time potencijalno nije uvijek dostupan najnoviji podatak na udaljenim čvorovima (Slika 4.9 B).
- Čvor odmah nakon primitka podatka šalje potvrdu transakcije. Zatim se podaci repliciraju lokalno te se nakon toga udaljeni čvorovi se asinkrono usklađuju. Time potencijalno nije dostupan ažuriran podatak niti u lokalnom čvoru, niti u udaljenom čvoru. Ako čvor prestane s radom prije dovršetka usklađivanja može doći do gubitka podataka. (Slika 4.9 C)



Slika 4.9 Replikacija nad dva podatkovna centra

4.5.2. Upravljanje vremenom

Mnoge baze podataka identificiraju zadnji zapis na bazi pomoću oznake vremena. Zbog raspodjele podatkovnih centara po svijetu, osmisliti globalno sinkroniziran sat je teško. Većinu zapisa obično ažurira ista aplikacija ili proces i u tom slučaju razlika u lokalnom vremenu neće dovesti do nekonzistentnosti podataka. Tamo gdje to nije slučaj, na primjer globalni financijski servisi gdje se transakcije mogu izvoditi bilo gdje u svijetu, te gdje je redoslijed podataka bitan, sinkronizacija sustava jest problem. Kako bi uvijek bio siguran u konzistentnost podataka koji se protežu kroz brojne podatkovne centre Google je osmislio svoju arhitekturu egzaktnog praćenja vremena - Google TrueTime API. Poslužitelji međusobno sinkroniziraju svoje vrijeme preko NTP (*Network Time Protocol*) mrežnog protokola koji poziva nadređeno računalo s točnijim (atomskim/GPS) satom da mu vrati točno vrijeme. Protokol uzima u obzir vrijeme odaziva poslužitelja, brzinu mreže, fizičku udaljenost i druge parametre. Za podatkovne centre unutar regije to je dovoljno precizno jer svi satovi unutar regije imaju isto odstojanje. *TrueTime* API rješava problem udaljenih podatkovnih centara na dva načina, GPS lociranjem koje može vratiti točno vrijeme i gdje nema razlike u brzini komunikacije te atomskim satom koji dodatno može

sinkronizirati vrijeme. Protokol čak rješava problem prijestupne sekunde (*leap second*). To je pojava koja se događa zbog odstojanja Zemljine putanje i radi toga minuta može imati sekundu više ili manje. Kako se to dogodilo 20 puta od početka mjerenja 1972. godine, Google počinje određeno vrijeme prije promjene u svojem TrueTime API-ju dodavati po par milisekundi svakoj sekundi. Dok dođe do promjene za sekundu više, Google-ovi poslužitelji već su preskočili tu sekundu. Ne može se dodati cijela sekunda odjednom jer bi poslužitelji primijetili razliku u vremenu i sinkronizirali natrag svoje vrijeme na sekundu manje.

4.6. Oblikovni obrasci za stupčane baze podataka

Oblikovanje ključa retka

Stupčane baze podataka distribuiraju podatke na temelju ključeva redaka. Svaki tablet poslužitelj upravlja drugim imeničkim prostorom. To znači da distribucija podataka, a time i izvođenje upita ovisi uvelike o ključevima. Korištenjem vremenske oznake poruka kao ključ retka znači da se svi novi podaci pohranjuju na jedan poslužitelj koji upravlja najvišim vrijednostima ključeva. Time smo usporili protočnost sustava. Umjesto toga, poželjno je odabrati ključ koji ravnomjerno raspodjeljuje podatke po poslužiteljima. Za to se koriste ključevi s nasumičnim vrijednostima kao što je UUID. Programski jezici često imaju klasu koja generira takve identifikatore. Također je sama funkcionalnost njihovog generiranja ugrađena u baze podataka. S obzirom na to da ovim principom imena ključeva više nisu smislena, za pretraživanje podataka, dakle imena stupaca, koriste se same vrijednosti spremljene pod ključem.

Ovakav model imenovanja ključeva ima i svoje nedostatke. Što je više ključ nasumičan, to je manje vjerojatno da će se susjedni reci spremati zajedno. Time smo postigli veću brzinu zapisivanja, ali smanjili brzinu pristupa podacima jer baza mora proći kroz više particija da bi dohvatila povezane podatke. Korištenjem sekundarnih indeksa može se donekle zaobići ovaj problem jer se sekundarni indeksi spremaju odvojeno od primarnih ključeva.

Denormalizacija

Redci pojedine tablice mogu sadržavati više familija stupaca. Svaka familija stupaca može sadržavati mnogo stupaca. Poželjno je u radu sa stupčanim bazama podataka podatke denormalizirati, dakle pohraniti sve povezane podatke u jedan zapis to jest u jedan redak. Time se može izbjeći potreba izvođenja join operacija pri dohvatu podataka, no zahtijeva dupliciranje podataka. Denormalizacijom se također rješava problem da stupčane baze podataka ne podržavaju podupite. Kao vrijednost stupca za određeni ključ ne možemo spremiti listu različitih vrijednosti. Umjesto toga moguće je aplikacijom serijalizirati listu kao niz vrijednosti te ih pohraniti po stupcima. Ili je moguće elemente liste pohraniti kao imena stupaca. Time familija kolumni predstavlja jednu listu.

Izbjegavanje transakcija kroz više redaka

Sve operacije pisanja i čitanja u stupčanim baza podatka su atomarne ako se odvijaju nad jednim retkom, bez obzira na broj stupaca koje se čita ili piše. To znači da ako radimo operaciju nad skupom stupaca, ona će se izvršiti nad čitavim skupom, ili se neće izvršiti uopće. Parcijalni rezultati nisu dopušteni unutar atomarnih operacija.

U slučaju da obavljamo operacije nad više redaka, većina stupčanih baza podataka ne podržava svojstvo atomarnosti. Postoji li potreba za atomarnosti u tom slučaju, najbolje je implementirati transakciju tako da se koristi jedan redak podataka. To moguće zahtijeva promjenu modela podataka i poželjno je takvu situaciju uzeti u obzir pri implementaciji baze podataka.

Izbjegavanje složenih podatkovnih struktura

Složene strukture moguće je pohraniti u stupčanu bazu podataka ali to nije poželjno, osim ako postoji specijalni razlog za čuvanjem tih struktura. Ako se spremljeni objekt tretira kao *String* te će se samo spremiti ili dohvaćati onda bi se trebali i spremiti kao *String*, a ne kao složenu struktura. Ako će se izvršavati operacije nad vrijednostima unutar strukture onda je poželjno rastaviti strukturu na elemente.

Korištenjem različitih stupaca za pojedini atribut olakšava izvođenje operacija nad atributima jer se mogu dodati sekundarni indeksi na te vrijednosti. Također to omogućava grupiranje pojedinih vrijednosti stupaca po familijama stupaca.

Indeksiranje

Indeksiranje dopušta brži dohvat podataka nad bazom. U stupčanim bazama podataka moguće je pretražiti vrijednost stupca da bi se došlo do ključa retka koji sadrži tu vrijednost. Koristeći indeks u mnogo slučajeva ubrzava dohvat podataka. Razlikujemo dvije vrste indeksa, primarne i sekundarne. Primarni indeksi su indeksi nad ključevima redaka tablica. Automatski ih održava sustav upravljanja baza podataka.

Sekundarni indeksi su indeksi koje stvaramo nad vrijednostima jednog ili više stupaca. Dio stupčanih baza podataka ne podržava indeksiranje vrijednosti. To znači da se podaci moraju drugačije oblikovati da bi se postigle određene funkcionalnosti upita. U tom slučaju poželjno je stvoriti vlastite tablice indeksa gdje se kao ključeve redaka stavljaju vrijednosti. Na primjer familije stupaca u imenima i identifikatoru ključa mogu sadržavati opis vrijednosti zapisa kojeg čuvaju. Time se podaci o zapisu mogu prikazati bez daljnje pretrage po vrijednostima podataka. S obzirom na to da se ključevi automatski indeksiraju, svi zapisi s istim ključem retka za familiju stupaca spremaju se lokalno jedni kraj drugih. Time se dohvat podataka ubrzava.

Postoje dvije mogućnosti upravljanja sekundarnim indeksima:

1. Sekundarni indeksi upravljani bazom podataka

Generalno pravilo je da ako baza podataka dopušta stvaranje i upravljanje sekundarnim indeksima, tada bi se takva funkcionalnost trebala koristiti. Prednost sustava koje automatski upravljaju sekundarnim indeksima jest lakša održivost indeksa (nego alternativno korištenje vlastitih tablica indeksa). Baza podataka tada održava sve strukture potrebne za upravljanje indeksima te odlučuje optimalan način njihovog korištenja. Automatsko korištenje sekundarnih indeksa također ima tu prednost da nije potrebno mijenjati aplikacijski kod u svrhu korištenja indeksa.

No postoje slučajevi kada nije poželjno koristiti sekundarne indekse:

- Postoji velik broj istih vrijednosti u stupcu
- Postoji veliki broj jedinstvenih vrijednosti u stupcu
- Veliki broj zapisa nema definiranu vrijednost za stupac

Kada je broj različitih vrijednosti u stupcu malen (kardinalitet stupca) indeksi ne pospješuju izvođenje upita. Na primjer stupci koji samo sadrže vrijednosti da ili ne. Posebno ako ima otprilike jednak broj zapisa svake vrijednosti.

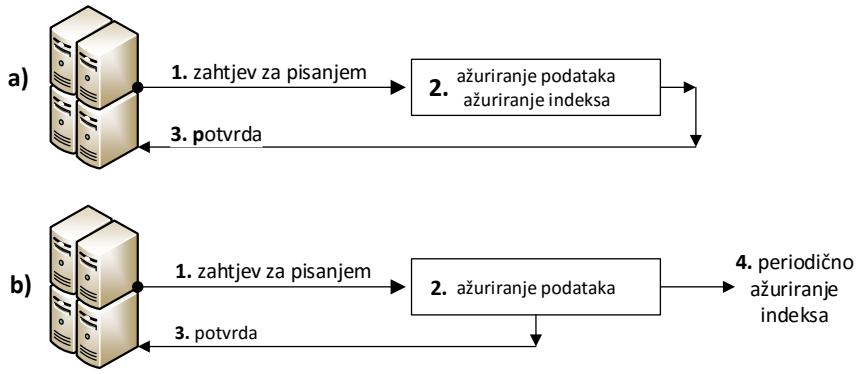
Također, ako stupci sadrže velik broj jedinstvenih vrijednosti kao što su adrese ulica, indeksi nisu od velike koristi. Moguće je da indeks održava toliki broj podataka da proces pretrage po indeksima i dohvaćanja podataka traje duže nego pretraga same tablice podataka.

2. Sekundarni indeksi upravljani korisnički definiranim tablicama

Drugi pristup indeksiranja jest stvaranje tablice indeksa pomoću koje stvaramo i upravljamo indeksima sami. Taj pristup koristimo u slučaju da baza ne podržava automatsko stvaranje indeksa ili u slučaju da želimo indeksirati dosta međusobno različitih vrijednosti. Indeksi koje smo stvorili unutar aplikacije koriste iste tablice, familije stupaca i strukture podataka u kojima pohranjujemo zapise. Eksplicitno moramo stvoriti novu tablicu koja sadrži podatke do kojih želimo doći pomoću indeksa. To naravno koristi i više prostora za pohranjivanje, kao i u slučaju automatskog stvaranja indeksa. Korištenjem tablica indeksa odgovorni smo za upravljanje i održavanje tih indeksa. Postoje generalno dvije opcije za ažuriranje indeksa: možemo ga ažurirati čim se dogodi promjena u podacima spremljenim u bazu, ili možemo u regularnim intervalima zajednički ažurirati sve indekse u tablici indeksa.

Ažuriranjem tablice indeksa u isto vrijeme kao i ažuriranje tablica u bazi, indeksi su konzistentni u svakom trenutku. Ovakav pristup je poželjan ako se korištenje indeksa može odvijati u bilo kojem trenutku. Nedostatak pristupa jest da aplikacija mora raditi dvije operacije pisanja, jednu na bazu a drugu na tablicu indeksa. To može dovesti do dužeg vremena čekanja prilikom operacije pisanja.

Kolektivno ažuriranje indeksa u određenim vremenskim intervalima ima kao prednost da ne tereti dodatno operacije pisanja. Naravno uz nedostatak da postoji period vremena kada podaci u tablici i indeksi nisu konzistentni.



Slika 4.10 a) ažurirane indeksa unutar operacije pisanja
b) kolektivno periodično ažuriranje indeksa

4.7. Alati za rad s Velikim Podacima

Baze podataka oblikovane su da pohranjuju i dohvaćaju velike količine podataka i te operacije rade odlično. Postoje međutim broj popratnih zadataka – osim same pohrane podataka, koje često obavljamo pri radu sa stupčanim bazama podataka. Ti zadaci uključuju:

- Izdvajanje, transformaciju i ubacivanje podataka – ETL
- Analizu podataka
- Nadgledanje izvođenja baze podatka

Izdvajanje, transformaciju i ubacivanje Velikih Podataka – ETL

Pomicanje velikih količina podataka zahtjevno je iz više razloga:

- Nedovoljna protočnost mreže za volumen podataka
- Vrijeme potrebno za kopiranje velikih količina podataka
- Mogućnost da se podaci korumpiraju za vrijeme prebacivanja
- Problemi spremanja ovakvih podataka na više lokacija

Proces skladištenja podataka postao je složeniji pojavom Velikih Podatka. Postoje mnogi ETL alati koji pomažu pri skladištenju takvih podataka kao što su Apache Flume, Apache Sqoop, Apache Pig.

Svaki od ovih alata brine se o određenim potrebama rada s Velikim Podacima i dio su Hadoop ekosistema alata. Apache Flume osmišljen je za premještanje velikih količina podataka. Sustav je distribuiran, pa tako ima prednosti kao što su skalabilnost, pouzdanost, otpornost na greške. Apache Sqoop osmišljen je u svrhu prebacivanja podatka s relacijskih baza podataka na stupčane baza podatka. Sqoop dopušta i masivno paralelne izračune MapReduce poslova. Apache Pig je jezik osmišljen u svrhu bolje protočnosti podataka (*data flow*) među sustavima, postavljanja analitičkih problema te nudi infrastrukturu za izvođenje poslova analize. Sadrži izraze za ubacivanje, filtriranje, agregiranje, join nad podacima i transformaciju podataka. Pig programi se prevode u MapReduce poslove.

Analiza Velikih Podataka

Jedan od razloga zašto velike organizacije pohranjuju velik broj zapisa je taj što ti podaci moguće sadržavaju vrijedne uvide i korisne informacije vezane za poslovni model kompanije. Postoji mnogo načina kako analizirati podatke, na primjer traženjem uzoraka među podacima i izdvajanjem korisnih informacija iz zapisa. Pri ovakvoj analizi, često se koriste znanstvene discipline poput statistike (opis populacije, predviđanje svojstava podataka i slično) te strojnog učenja (preporuke na temelju podataka, pronalazak uzoraka i slično). Nekoliko često korištenih alata pri analizi podataka su MapReduce, Spark, R, Mahout.

MapReduce je distribuirana platforma za paralelnu analizu i obradu podataka na velikim uzorcima podataka, rasprostranjenih preko velikog broja poslužitelja. Spark je distribuirana platforma nastala kao alternativa MapReduce-u uz neke razlike kao sto su više korištenja memorije naprema korištenju diska te generalniji model izračuna. R je statistička platforma koja sadrži mnogo modula za statističke izračune. Biblioteke također sadrže funkcionalnosti za prikupljanje podataka (*data mining*). Mahout je kolekcija alata primjenjivih za strojno učenje. Posebno je pogodan za klasifikacije podataka i funkcionalnost preporuka.

Zaštita podataka

Tradicionalni pristup zaštiti podataka zahtijeva da se aplikacijski kod brine o zaštiti. To uključuje autentikaciju i autorizaciju korisnika te osiguranje da korisnici imaju prava pristupa samo onim podacima nad kojima im je to dopušteno. Akreditacije i certifikati za institucije poput državnih agencija daju jamstvo organizacijama i javnosti da se najbolje prakse koriste vezane uz sigurnost podataka.

Ako je sigurnost implementirana na nivou aplikacije, tada su i aplikacija i baza podataka akreditirane zajedno kao cjelina. Stvaranje sigurnosnog modela zahtijeva prvo stvaranje sigurnosnih dodataka (plug-in) koji omogućavaju autentikaciju korisnika i provjeravaju njihove uloge unutar sustava. Zatim se dodjeljuju prava korisnicima nad određenim zapisima. Ovakav pristup ima svoje nedostatke. Obzirom da aplikacija ima puni pristup bazi podataka, želimo li omogućiti vanjskoj organizaciji rad nad aplikacijom, moramo im omogućiti pristup cijeloj bazi.

Zato je ponekad korisno stvoriti aplikaciju nad sigurnim sustavom za upravljanje bazama podataka i time imati dodatno jamstvo sigurnosti nad aplikacijom.

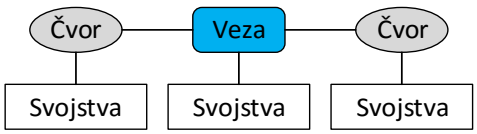
Dolazi li sustav s ugrađenim sustavom zaštite, onda je poželjno koristiti taj pristup umjesto pisanja vlastitog koda. Postoje prednosti pri korištenju sustava koji dolazi s ugrađenim sustavom autentikacije, autorizacije i podrškom za pristup podacima na temelju uloga – RBAC (*Role Based Access Control*). RBAC dopušta dodjeljivanje privilegija ulogama za akcije nad pojedinim zapisima. Ulogama je lakše upravljati nego se baviti individualnim korisničkim pristupom. Za promjenu prava korisniku potrebno je promijeniti prava grupi te će promjena biti vidljiva svim korisnicima unutar grupe.

Dodavanje prava korisnicima olakšano je u sustavima koji u svrhu zaštite nad podacima imaju dodatnu ćeliju kao dio ključa – takozvani domet ili vidljivost ključa (*key visibility*). Ovaj Booleov izraz možemo koristiti da limitiramo vidljivost podataka određenim ulogama unutar sustava. Tako se onemogućuje korisnicima mogućnost čitanja, a time i mogućnost prepravljanja i brisanja redaka u tablici. Želimo li onemogućiti brisanje cijelih tablica, tada postavke tablica treba definirati također s limitiranom vidljivosti.

5. Graf sustavi za upravljanje bazama podataka

5.1. Uvod u graf baze podataka

Graf možemo opisati kao matematički objekt sastavljen od vrhova i bridova. Vrhove nazivamo još i čvorovima, subjektima ili objektima te predstavljaju entitete. Veze među entitetima, još zvane predikati, predstavljaju bridovi koji mogu biti usmjereni. Smjer veze koristi se za detaljniji opis semantike veze među podacima. Za razliku od ostalih NoSQL baza podataka koje modeliraju podatke dvama bitnim strukturama - ključ i vrijednost, graf baze podataka bazirane su na trima glavnim strukturama: čvorovi, veze, svojstva.



Slika 5.1 Graf baza podataka sa čvor-veza-čvor strukturom. Čvorovi i veze mogu imati zasebna svojstva.

Čvorovima se dodjeljuju identifikatori te baza razumije da su dva čvora s istim identifikatorom zapravo isti čvor. Svojstva čvorova predstavljaju atribute entiteta te metapodatke entiteta kao što su vremenske oznake ili oznake verzija. Veze imaju svoj tip, početni čvor, krajnji čvor te svojstva. Dodavanje svojstava vezama korisno je za čuvanje metapodatka veza, kao i informacija poput udaljenosti čvorova ili zajedničkih svojstava čvorova.

Fleksibilna priroda vrhova i bridova čini ove baze odličnim za modeliranje konkretnih i apstraktnih veza među objektima. Idealne su za slučaj kada imamo puno objekata međusobno povezanih na složen način vezama koje imaju različita svojstva. Organizacija graf baza podataka dopušta nam da podatke jednom pohranimo i zatim ih tumačimo na različite načine ovisno o vezama među podacima. Dodavanje novih tipova veza među čvorovima je lako, no mijenjanje postojećih čvorova i veza među njima slično je migraciji podataka jer se sve promjene moraju obaviti nad svim čvorovima i svim vezama među njima. S obzirom na to da nema granice broja i vrsta veza među čvorovima, svi zajedno mogu biti prikazani u istoj bazi podataka.

Graf baze dopuštaju upite nad čvorovima, vezama i svojstvima. Lako gradimo upite poput pronalaska susjednog čvora, ali i upite koji zahtijevaju prolazak po čvorovima i traženje obrazaca. Upit nad grafom još zovemo prolazak ili put po grafu (*traversing the graph*). Možemo promijeniti uvjete prolaska grafa bez mijenjanja čvorova ili veza. Upiti nad vezama brzo su izvršivi. Povezanost čvorova ne računa se u trenutku upita, nego je trajno ugrađena kao veza (*persisted as a relationship*). Prolazak grafa takvim vezama brže je nego računati ih za svaki upit. Join operacije graf baza su brze i jeftine po pitanju resursa. Ova brzina proizlazi iz male veličine čvorova te iz mogućnosti da se podaci u grafu čuvaju u radnoj memoriji. Jednom kada je graf pohranjen u memoriju, dohvaćanje podataka ne zahtijeva operacije nad diskom.

Za razliku od ostalih NoSQL baza podataka, graf bazu teško je skalirati nad više poslužitelja zbog bliske međusobne povezanosti čvorova u grafu. Podaci se mogu replicirati na višestruke poslužitelje što ubrzava operacije čitanja. No operacije pisanja nad više poslužitelja i upiti nad čvorovima na različitim lokacijama mogu biti teške za implementaciju.

Graf baze podataka optimizirane su za upravljanje transakcijama, s naglaskom na transakcijski integritet i visoku dostupnost podataka. Stoga se često koriste za upravljanje OLTP (*Online Transaction Processing Systems*) sustavima. Većina tehnologije iza ovih baza podataka proizlazi iz Google-ove Pregel dokumentacije koja opisuje tehnologije koje Google koristi za rangiranje internetskih stranica po modelu grafa.

5.2. Vrste graf baza podataka

5.2.1. Graf baze trojki

Graf baze trojki smatramo podvrstom graf baza podataka. Drugačije su arhitekture kao i primjene od generalnih graf baza podataka.

Model podataka sastoji se od mnogo jednostavnih subjekt-predikat-objekt trojki. Jedna trojka mogla bi opisivati tip subjekta, druga brojčano svojstvo koje pripada subjektu, a treća vezu s drugim subjektom.

Individualno su trojke semantički slabe, no brojnošću pružaju bogat skup informacija i mogućnosti izvođenja zaključaka. Baze trojki pohranjuju trojke kao zasebne objekte, što im dopušta mogućnost horizontalnog skaliranja, ali ujedno onemogućava brz prolazak po grafu. Upitom nad bazom, baza trojki mora stvoriti povezane strukture iz individualnih objekata što dodatno povećava vrijeme odaziva baze.

Tipično odgovaraju na semantičke upite za činjenicama nad samim podacima, a ne za stanjem veza među čvorovima, veličinom grafa, udaljenosti među čvorovima i slično. Za upite nad bazama trojki generalno se koristi SPARQL semantički upitni jezik.

Svaku veza i tip opisujemo zasebnim rječnikom kojeg nazivamo ontologija. Ontologija opisuje skup tipova, moguće s nasljedstvima i vezama među ostalim tipovima drugih ontologija. Ontologije opisujemo koristeći trojke modela podataka sastavljenih od RDF (*Resource Description Framework*) izraza.

Postoje situacije kada je subjekt-predikat-objekt prejednostavan za opisati tvrdnje koje imaju smisla samo u određenom kontekstu. Postoji način kako oblikovati kontekst grafa unutar baza trojki pomoću imenovanog grafa. Tvrdnje se dodaju u imenovani dio grafa, i time je moguće ograničiti upite na određeni podskup informacija. Model imenovanog grafa sadrži čvorove i veze. Čvorovi mogu imati svojstva te više oznaka. Oznake koristimo za grupiranje čvorova. Veze također mogu imati svojstva i jednu oznaku, usmjerene su te uvijek imaju početni i završni čvor. Smjer i oznaka veze daju semantičko pojašnjenje strukture čvorova.

5.2.2. Generalne graf baze

Generalne graf baze podataka omogućuju efikasan pronalazak informacija o vezama ili mrežama veza. Mogu odgovarati na upite o samim podacima, ali općenito ih koristimo za matematičke informacija o vezama. Sljedeće operacije generalno ne nalazimo u bazama trojki:

- Najkraći put: Pronalazi minimalni broj skokova za put među dva čvora i pronalazi optimalnu rutu obilaska preko tih čvorova.
- Svi putevi: Pronalazi sve moguće puteve između dva čvora.

- Bliskost: Za skup čvorova unutar grafa vraća koliko su međusobno slični.
- Udaljenost: Za skup čvorova unutar grafa vraća koliko su međusobno udaljeni.
- Podgraf: Vraća odgovor sadrži li graf određeni podgraf, ili pronalazi podgraf grafa koji zadovoljava dane uvjete.

Ovi algoritmi su matematički složeni i čine ovakve upite težim nego upitima za skupom činjenica. Razlog tome jest da pojedini algoritam prolaska grafa može otići nepredvidljivo duboko u graf, u potrazi za odgovorom upita. Upiti nad bazom trojki s druge strane ograničeni su dubinom unutar upita i prolaze kroz unaprijed poznat skup čvorova navedenih u samim upitima.

Jedan od modela grafa koji se može oblikovati generalnim graf bazama podataka jest hipergraf. Model hipergrafa dopušta da veza oblikuje bilo koji broj čvorova. Dok recimo imenovani graf dopušta za svaku vezu samo jedan početni i završni čvor. Ovakvi grafovi korisni su za oblikovanje mnogo-na-mnogo veza. Modeli hipergrafa i imenovanog grafa su izomorfni. Dakle uvijek je moguće informacije oblikovane jednim modelom grafa oblikovati pomoću modela drugog tipa grafa.

5.2.3. Razlike u arhitekturi

Razlike u samom modelu podataka generalne graf baze i graf baze trojki vodi do razlika u arhitekturi. Zbog upita nad vezama, kao što su broj skokova među čvorovima, generalne graf baze obično zahtijevaju da svi podaci budu pohranjeni na jednom poslužitelju u svrhu bržeg izvođenja samih upita. Graf baze trojki s druge strane dopuštaju distribuciju podataka sličnim principima kao i ostale NoSQL baze podataka. Imaju specijalizirane indekse nad trojkama koji omogućavaju upite nad više poslužitelja u grozdu. Kao što smo već spomenuli, to je zato što im je glavna funkcija spremanje i dohvaćanje podataka, a ne vraćanje složenih metrika ili statistika o povezanosti samih subjekata.

Baze trojki građene su nad skupom RDF standarda i SPARQL (*SPARQL Protocol and RDF Query Language*) upitnim jezikom. Generalne graf baze imaju svaka svoj zasebni, ponešto različiti, skup terminologija, modela podataka i operacija nad upitima (korištenjem Cypher upitnog jezika na primjer).

S obzirom na to koju vrstu upita koje koristimo, odabiremo graf bazu ili bazu trojki s drugačijim arhitekturama:

- Jedan poslužitelj za cijelu bazu: Visoka cijena poslužitelja. Ostali poslužitelji služe kao replike podataka za slučaj kvara glavnog poslužitelja ili mogu opsluživati operacije čitanja s određenim zakašnjenjem. Algoritmi analize grafova u ovom slučaju rade jako brzo.
- Više poslužitelja – particionirana baza: Manje cijene poslužitelja. Svaki poslužitelj je odgovoran za dijelove podataka i također, svaki je replika nekog drugog poslužitelja u grozdu. Ovakva opcija usporava algoritme nad grafovima, ali SPARQL upiti još su uvijek brzi nad podacima spremljenim na samo jednom poslužitelju.

5.3. Osnovne značajke graf baza podataka

Performanse i fleksibilnost

Oblikovanje složenih veza među objektima graf bazama podataka pokazuje veliko poboljšanje brzine izvođenja sustava naprema oblikovanju sličnih modela u relacijskim i ostalim NoSQL bazama. U kontrastu s relacijskim bazama, kojima performanse izvođenja join operacija povećanjem broja podataka drastično opadaju, performanse graf baza podataka ostaju relativno konstantne. To je zato što su upiti lokalizirani nad dijelom grafa, pa je i brzina upita proporcionalna samo dijelu grafa koji se treba proći, a ne cjelokupnom grafu.

U današnje vrijeme želimo moći oblikovati podatke u koraku s ostatkom aplikacije, koristeći tehnologije usuglašene s današnjim inkrementalnim i iterativnim modelom razvoja. Priroda graf baza podataka koja ne zahtijeva shemu, u kombinaciji s programskim sučeljem prilagođenim testiranju te upitnim jezikom, pruža graf bazama podataka upravo tu prednost.

Graf baze prirodno su aditivne, što znači je dodavanje novih čvorova i veza ili podgrafova na postojeću strukturu moguće bez smetnji na trenutne upite i funkcionalnost aplikacije.

Ne moramo oblikovati detaljno domenu modela prijevremeno, nego možemo dodavati informacije kako dobivamo uvide u prostor problema.

Konzistentnost

S obzirom na to da graf baze podataka rade nad skupom visoko povezanih čvorova, većina ih ne preporučuje distribuciju čvorova po više poslužitelja. Generalno se zapisi u bazu odvijaju samo nad gospodarom poslužiteljem. Replikacija poslužitelja odvija se asinkrono. Ovakva funkcionalnost jest normalna kod ostalih NoSQL baza podataka, no najčešće za situaciju replikacije među grozdovima različitih podatkovnih centara, a ne među dva poslužitelja u istom grozdu. Zbog toga graf baze ne garantiraju konzistentnost unutar grozda, već samo unutar gospodara poslužitelja. Zbog asinkrone, naknadno konzistentne prirode replika, moguće je da prestankom rada sustava neki od spremljenih podataka nisu dostupni na replikama.

Graf baze osiguravaju ACID- konzistentnost koristeći transakcije. Pisanje na bazu događa se unutar konteksta transakcije. Čvorovi i veze uključene u transakciju zaključavaju se za ostale procese za vrijeme trajanja transakcije. Dogodi li se da je transakcija neuspješna, sve promjene se vraćaju u prvotno stanje. Zapisi unutar jedne transakcije nisu vidljivi ostalim transakcijama. Pokuša li više transakcija napraviti promjene nad istim elementom, baza će serijalizirati transakcije. Jednom kada je transakcija izvršena sustav garantira da su promjene sačuvane u bazu.

Dostupnost

Dostupnost graf baza podataka generalno je ostvarena standardnim gospodar-poslužitelj modelom repliciranja. Repliciranjem podataka na poslužitelje robove odvija se u čestim vremenskim intervalima. Gospodar poslužitelj odgovoran je za pisanje, dok su poslužitelji robovi odgovorni za čitanje.

Pisanje na poslužitelje robove jest omogućeno dodatnom konfiguracijom te se unutar transakcije sinkronizira na gospodara poslužitelja. Ostali poslužitelji robovi čekaju zatim da im gospodar poslužitelj pošalje nove podatke.

Apache Zookeeper može pratiti transakcije nad čvorovima i stanja poslužitelja. Njemu poslužitelji šalju upit o tome tko je gospodar poslužitelj. Prestane li gospodar poslužitelj s radom, izabire se novi. Ovime postižemo i veću trajnost podataka (na dvije instance baze umjesto jedne) no pod cijenu veće latencije. Naime, dodatan promet na mreži i protokol koordinacije poslužitelja robova može biti znatno sporiji nego pisanje na gospodara. Ovim postupkom ne implicira se da su operacije upita distribuirane na poslužiteljima jer svi zapisi moraju proći kroz gospodara.

Skaliranje

Najčešća tehnika skaliranja u NoSQL bazama podataka jest horizontalno particioniranje, gdje se podatke dijeli i distribuira nad više poslužitelja. S graf bazama podataka to je teško postići. S obzirom na to da svaki čvor može biti u vezi s bilo kojim drugim čvorom, bolja je opcija čuvati sve povezane čvorove na istom poslužitelju u svrhu efikasnijeg obilazaka grafa. No graf baze svejedno možemo skalirati koristeći druge tehnike. Generalno govoreći postoje četiri načina kako skalirati graf bazu podataka.

1. Skaliranje za operacije čitanja ostvarujemo dodavanjem poslužitelja robova koji odgovaraju na zahtjeve za čitanjem, dok gospodar poslužitelj odgovara na zahtjeve za pisanjem.
2. S obzirom na to da današnja računala mogu imati veliku radnu memoriju, moguće je cijeli skup čvorova i veza pohraniti u radnu memoriju svih poslužitelja. Ova tehnika je korisna samo u slučaju da skup podataka s kojim radimo stane cijeli u radnu memoriju.
3. U slučaju da je cijeli graf prevelik za pohranu u memoriji, možemo u memoriju poslužitelja replika pohraniti samo zasebne dijelove grafa. U tom slučaju preusmjeravamo svaki upit u grozdu u onu instancu baze gdje se u memoriji nalazi dio grafa potreban za izvršenje upita. Ako je većina upita lokalizirana nad nekim početnim čvorom i okružujućim podgrafovima, tada je ovo dobra strategija.
4. Još jedan način jest da podatke aplikacijski podijelimo na više poslužitelja koristeći znanja o domeni upita – naravno ako je domena takva da se podaci mogu logički razdijeliti. To podrazumijeva da se pojedini upit odvija isključivo nad podacima jednog poslužitelja. Primjer toga su domenski specifični problemi - poslužitelji na fizički

različitim lokacijama odgovaraju na upite koji dolaze s lokacije unutar njihove domene.

Upiti

Graf baze podatka podržavaju više upitnih jezika, kao što su Gremlin ili Cypher. Upitni jezik omogućava slaganje složenih upita nad elementima te oznakama čvorova i veza. Cypher je ekspresivan upitni jezik. Oblikovan je da bude lako čitljiv i razumljiv programerima i oblikovnim inženjerima. Laka čitljivost proizlazi iz činjenice da njime oblikujemo podatke na sličan način kako intuitivno oblikujemo grafove dijagramima.

Koristeći ASCII znakove za prikaz čvorova i veza možemo oblikovati podatke i njihove relacije. Modeliramo čvorove pomoću zagrada (i), a veze među oznakama -> i <- , gdje < i > označavaju smjer veze. Između uglatih zagrada [i] navodimo ime veze, s prefiksom :. Slično vrijedi za oznaku čvora. Svojstva veza i čvorova navodimo među vitičastim zagradama { i }. Model ASCII prikaza vidimo na sljedećem primjeru.

($key1 as String)-[:$relation as String { $attribtues as Object() }]-> ($key2 as String)

Kao i mnogi upitni jezici, Cypher koristi kompoziciju izraza u izgradnji upita.

Koristeći MATCH ključnu riječ uspoređuju se obrasci među vezama. WHERE izrazom filtriraju se vrijednosti čvora ili veza među rezultatima. RETURN specificira što se vraća kao rezultat upita, čvor, veza ili vrijednosti. CREATE izrazom stvaramo čvorove i veze. DELETE izrazom brišemo čvorove, veze i svojstva. SET izrazom postavljamo svojstva.

Čvorovima možemo pridružiti jednu ili više oznaka sličnih RDF formatima. Svaki čvor kome je dana oznaka imat će indeksirane elemente oznake. Iako se može postavljati upite nad elementima bez dediciranih indeksa, uvelike se povećava brzina upita dodavanjem indeksa. Indekse je moguće podesiti za svaki element, ali ih stavljamo samo nad elementima koje koristimo u upitima. Ako više oznaka čvorova koristimo za traženje elementa potrebno je podesiti višestruke indekse.

Indeksi pomažu optimizirati pronalaženje specifičnih čvorova. Ponekad situacija zahtijeva da želimo dohvatiti te čvorove direktno, a ne procesom otkrivanja čvorova putem po grafu. Na primjer, u situaciji da želimo identificirati početni čvor nekog puta, moramo naći

specifičan čvor s određenom kombinacijom oznaka i svojstava. Da bi se takva funkcionalnost postigla, Cypher dopušta stvaranje indeksa nad oznakama i svojstvima. Indeksiranje čvorova obavljamo pri dodavanju čvorova u bazu ili kasnije postupkom iteracije nad čvorovima. Prvo stvaramo tablicu indeksa imena čvorova. Kada stvorimo čvor dodajemo ga u tablicu indeksa. To se obavlja unutar konteksta transakcije. Podržano je i korištenje zasebnog servisa Lucene kao servisa indeksiranja koji omogućuje i puno pretraživanje po tekstu.

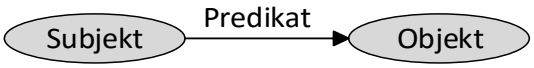
5.4. Osnovni pojmovi vezani uz graf baze trojki

5.4.1. Vezani standardi

RDF standard

Jezik Semantičkog Weba stvoren je da omogući predstavljanje bogatog i kompleksnog znanja o entitetima (stvarima, pojmovima), grupama entiteta i odnosima među entitetima.

W3C RDF standard jest skup W3C specifikacija i protokola osmišljenih u svrhu oblikovanja i razmjene podataka na webu. Omogućuje integriranje skupova podataka različitih organizacija u jednu cjelinu. Konceptualno, moguće je korištenjem RDF standarda pohraniti dva skupa podataka različitih sustava u jednu graf bazu te izvoditi upite nad spojenom bazom. RDF model koristi usmjerene grafove, gdje svaka veza ima početni čvor i odredišni čvor.



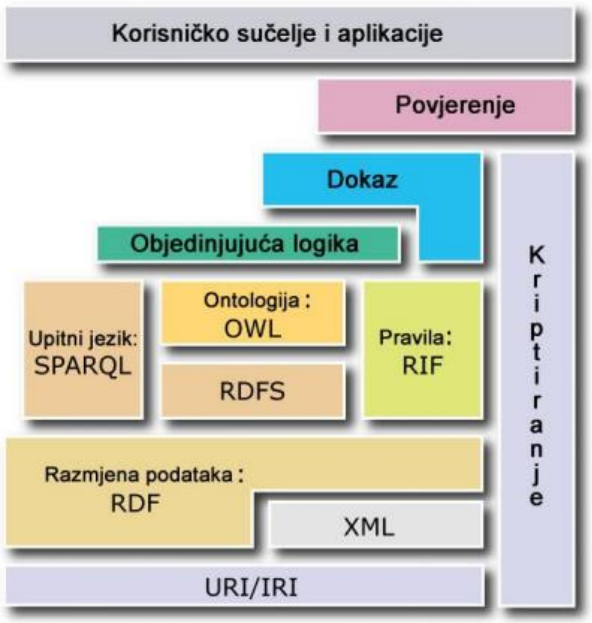
Slika 5.2 RDF model čvor-veza-čvor modela

Početni čvor nazivamo subjekt, a odredišni objekt. Veza među njima zove se predikat. Cijelu strukturu nazivamo tvrdnjom. Mogućnost putovanja grafom proizlazi iz činjenice da dva čvora iz različitih grupa mogu pokazivati na isti objekt. Jednom kada se zaključi da su čvorovi zapravo isti, radi se spajanje grafova.

Ovaj proces je koristan u područjima poput logike te pronalasku uzoraka. Podatke o samoj vezi – metapodatke veze, moguće je spremiti u čvorove trojke što omogućuje lakše upravljanje trojkama.

Glavni koncepti RDF standarda:
URI: Jedinstven identifikator subjekta ili predikata.
Imenik: Zasebni prostor u kojem čuvamo identifikatore grafa. Imenici dopuštaju zajedničko korištenje RDF konstrukcija s neovisnim ontologijama.
RDF tip: Tip kojeg koristimo da bi opisali da je subjekt instanca određenog tipa. RDF dopušta nasljeđivanje RDF tipova među ontologijama.
Subjekt: Entitet kojeg opisujemo.
Predikat: Veza subjekta i objekta.
Objekt: Još jedan subjekt kada opisujemo vezu među subjektima, ili svojstvo subjekta.

RDF podržava više formata – jezika, za izgradnju izraza. Neki od čestih jezika su N-Triples, Turtle, i RDF/XML. Korišteni format ovisi o alatu s kojim radimo.
U ekosustavu graf baza postoje i drugi korisni standardi:
SPARQL: Semantički upitni jezik.
RDF shema (RDFS): Opisuje validne izraze dopuštene za određenu shemu.
OWL nekad nazivan i RDFS+ : Podskup OWL jezika često se koristi kao dodatka RDFS definicijama.
PROV-O (*PROV Ontology*) : Služi da dokumentiranje promjena podataka.



Slika 5.3 Tehnologije semantičkog web-a

SPARQL upiti

SPARQL je rekurzivni akronim za *SPARQL Protocol and RDF Query Language*.

To je protokol koji definira uporabu SPARQL upitnog jezika po uzoru na SQL. SPARQL uvažava i koristi specifičnosti strukture i načina pohrane podataka u RDF-u te prijenosa podataka http-om.

SPARQL URL se sastoji od tri dijela:
1. URL SPARQL endpoint-a.
2. Imena grafova nad kojima se postavlja upit (Opcionalno, kao dio niza koji čini upit).
3. Sam SPARQL upit (Kao dio niza koji čini upit).

Između ostalog omogućava:

- Dohvat podataka u obliku URI-ja, literala, praznih čvorova, podgrafa
- Projekciju, podupite, agregatne funkcije, negaciju i slično
- Pretraživanje grafa
- Konstrukciju novog grafa na temelju dohvaćenih podataka
- Transformacija RDF podataka iz jednog rječnika u drugi.

Postoje implementacije u više programskih jezika koje prema W3C standardu omogućuju sljedeće operacije:

SELECT - iz izvora podataka izdvaja one elemente koji se podudaraju sa zadanim uzorcima
ASK - vraća samo logičku vrijednost istine ili laži (true ili false) u ovisnosti postoje li elementi koji se podudaraju po zadanim uzorcima u upitu
CONSTRUCT - stvara RDF graf prema predlošku zadanom u upitu te koristi WHERE dio upita da bi zamijenio varijable u predlošku s konkretnim vrijednostima
DESCRIBE – vraća jedan RDF graf s podacima o URI-u. URI može biti konstanta ili varijabla čija se vrijednost dobije iz WHERE dijela upita

Opis ontologije

Ontologija ili RDF rječnik je semantički model kojim opisujemo skupove tipova, svojstava i veza određene domene RDF sustava. Ontologija se sastoji od pojmova koji opisuju neku domenu, nazive bitnih koncepata iz domene, znanje i ograničenja na pojmove iz domene. Jedan sustav može biti opisan pomoću više rječnika. Također, moguće je koristiti više rječnika za opis različitih aspekta istog subjekta.

Jezici ontologija:

- RDF – iskazivanje tvrdnji
- RDFS – opisivanje podataka
- OWL – bogatije opisivanje podataka i veza

Jedan jezik za oblikovanje podataka i strukture unutar RDF formata nazivamo RDFS (*RDF Schema Language*). RDFS-om se ne predstavljaju sami podaci, nego znanje o podacima. OWL (*The Web Ontology Language*), pruža dopune na RDFS koje omogućuju modeliranje kompleksnijih scenarija. Na primjer predikat inverseOf može se koristiti za opis da su veze u suprotnosti jedna s drugom. Dakle postojanjem jedne veze može se zaključiti da postoji i druga veza u suprotnom smjeru. Ovakvo zaključivanje može biti korisno za skraćivanje puteva unutar upita, što može uvelike pojednostavniti buduće upite.

OWL proširuje mogućnosti RDFS-a:

- karakteristike svojstava (TransitiveProperty, SymmetricProperty, InverseOf, …)
- ograničenja svojstava (minCardinality, maxCardinality, …)
- preslikavanja (equivalentClass, equivalentProperty, sameAs, …)
- složeni razredi (intersectionOf, unionOf, …)

Opis porijekla podataka

Dokumentacija promjena nad sadržajem podataka kroz vrijeme naziva se provenijencija podataka. RDF standard koji opisuje ovakve promjene kroz vrijeme jest W3C PROV-O. Pruža način opisa dokumenata, verzija dokumenata, promjena, ljudi odgovornih za promjene, softver ili mehanizam korišten za te promjene.

PROV- O opisuje glavne klase:

✓ prov:Entity: subjekt kojeg se stvara, mijenja ili koristi kao ulaz.

✓ prov:Activity: proces koji mijenja entitet.

✓ prov:Agent: osoba ili proces koji provodi neku aktivnost nad entitetom.

Ove glavne klase mogu biti korištene pri opisu raspona aktivnosti i promjena nad sadržajem.

PROV-O uključuje svojstva i veze, neke od kojih su:

✓ wasGenertedBy: ukazuje na to koji agent je stvorio entitet.

✓ wasDerivedFrom: pokazuje lanac verzioniranja ili iz čega su se početno entiteti udružili.

✓ startedAtTime, endedAtTime: daje informacije o tome kako se aktivnost odvijala.

✓ actedOnBehalfOf: dopušta agentu procesu da zabilježi koji agent ga je zadužio za posao.

PROV- O grupa standarda uključuje i validator servisa koji se može stvoriti i pustiti da radi nad bazom trojki koja koristi PROV-O podatke. Koristan je u slučaju da se u bazi podataka žele čuvati podaci o promjenama zapisa unutar baze.

5.4.2. Integracija funkcionalnosti dokumenata i baza trojki

Sve je češći slučaj integracije funkcionalnosti dokument baza podataka i funkcionalnosti baza trojki. To možemo ostvariti na više načina:

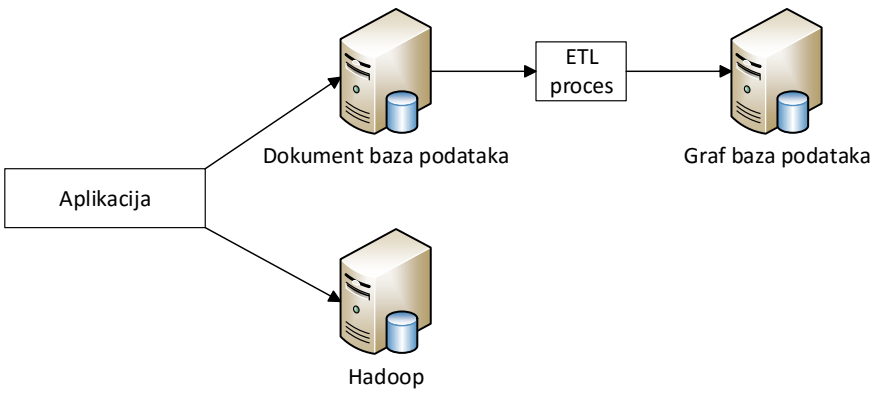
1. Dokument baze podataka mogu se proširiti RDF standardom i mogućnosti izvršavanja SPARQL upita nad kolekcija unutar dokumenta. To je posebno korisno zato što dokument baze podataka ne podržavaju oblikovanje veza među dokumentima (osim jednostavnih referenca) te isto tako ne mogu dohvatiti više vezanih dokumenata u jednom upitu. Specijalizirani indeksi se koriste kako bi se osiguralo da se upiti specifični bazama trojki brzo izvršavaju nad dokumentima. Slično tome, unutar baza trojki možemo prikazati dokument (na primjer u JSON formatu) kao subjekt i koristiti trojke za opis metapodatka dokumenta, porijekla dokumenta i veza. Kao rezultat možemo zatražiti spojeni dokument stvoren u vrijeme upita iz više povezanih dokumenata.

2. Integracijom baza trojki i dokument baza u jedan sustav. Integracije različitih baza podataka unutar istog sustava nazivamo višejezična dosljednost (*Polyglot Persistence*). Ovom metodom dobivamo najbolje iz obje baze podataka: dokument baza podataka brine o zapisima i pretraživanju, a graf baza podataka brine o vezama među zapisima i njihovim svojstvima. Najveći izazov u toj implementaciji predstavlja sinkronizacija podataka između više modela i izvora podataka te je potrebno napraviti specifičnu implementaciju ovisnu o domenskom problemu i ograničenjima. Opterećenje sustava može se, u ovisnosti o upitima koji se izvršavaju, preusmjeriti na implementaciju koja je optimirana za rješavanje takvih upita. Zajedničkim pristupom dokumentima i semantičkim izrazima dolazimo do potrebe stvaranja kombinatornih upita. Dakle stvaramo upit nad dokument bazom i nad bazom trojki da bi odgovorili pitanje vezano za zapis u bazi. Kombinatorni upit bi mogao biti upit kojim želimo dohvatiti sve dokumente kolekcije s određenom vrijednosti i određenom vezom prema drugom dokumentu.

Uzmimo za primjer da pohranjeni dokument u dokument bazi podataka sadrži tekst, koji je semantički izdvojen, obogaćen i pohranjen u graf bazu podataka. To znači da je tekst u dokumentu analiziran i iz teksta su izdvojeni entiteti i veze među njima. Ako se takav dokument promjeni, semantički podaci također se mijenjaju. U tom slučaju želimo moći promijeniti i skup podataka izdvojen iz dokumenta i pohranjen u bazu trojki. To možemo učiniti na dva načina:

1. Spremanjem trojki u dokument iz kojeg su izdvojene. Možemo na primjer spremiti JSON reprezentaciju trojki kao element unutar dokumenta. Time smo postigli povezivanje svih indeksa unutar jednog dokumenta. S obzirom na to da dokument baze podatka imaju mogućnost pretraživanja po riječima, rasponu, kao i mogućnost semantičkih upita (SPARQL) to znači da možemo jednim API upitom dohvatiti sve potrebne informacije. Ovaj postupak funkcionira neovisno o ontologiji ili tipu upita pretrage nad dokumentom. Radi se samo o različitim tipovima indeksa nad istim dokumentom.

2. Korištenjem imenovanog grafa. Kao ime grafa koristimo identifikator dokumenta te spremamo sve izdvojene trojke dokumenta u taj graf. Time smo olakšali ažuriranje izdvojenih metapodataka. Nedostatak pristupa jest da je potrebno ručno napisati kod na strani poslužitelja da izvrši upit nad bazom trojki te drugi za upit nad dokument bazom da bi se razriješio upit.



Slika 5.4 Višejezična dosljednost: integracija graf i dokument baze u jedan sustav

6. Implementacije NoSQL baza podataka

Ovim poglavljem dajemo pregled osnovnih funkcionalnosti popularnih implementacija NoSQL baza podataka: Redis, MongoDB, Cassandra i Neo4j. Sve implementacije su otvorenog izvornog koda.

Implementacije baza podataka rađene su na DigitalOcean Cloud servisu koji pruža infrastrukturu za korištenje virtualnih poslužitelja i razne druge usluge vezane za njih.

Servis ima opciju stvaranja virtualnih poslužitelja koji unaprijed imaju instalirane i konfigurirane određene aplikacije i baze podataka. Korištenjem takvih predefiniranih poslužitelja korisnik može odmah po stvaranju virtualnog poslužitelja početi koristiti odabranu bazu podataka.

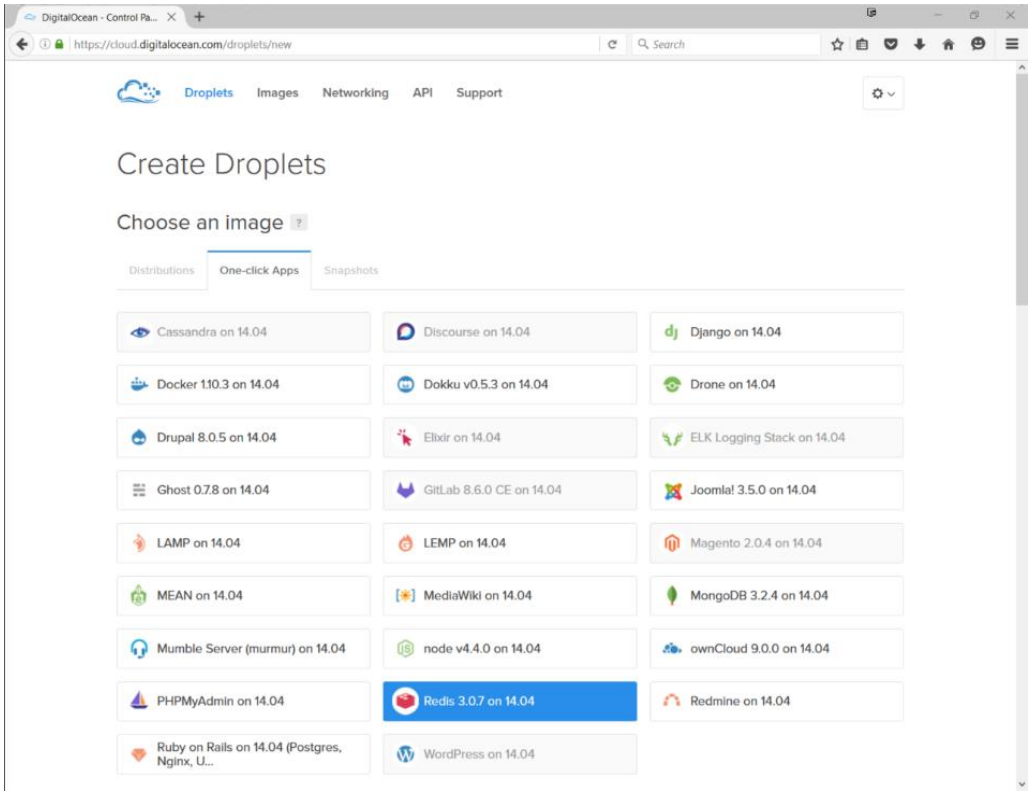
Konkretno, u ovom slučaju stvorena su četiri Linux virtualna poslužitelja, na svaki od kojih je instalirana po jedna od NoSQL baza podataka.

Cijeli postupak je jednostavan. Potrebno se prvo registrirati na https://www.digitalocean.com. Zatim odabiremo stvaranje novog virtualnog poslužitelja. Sada možemo odabrati operativni sustav poslužitelja - u slučaju da ne želimo koristiti unaprijed konfigurirane aplikacije na poslužitelju.

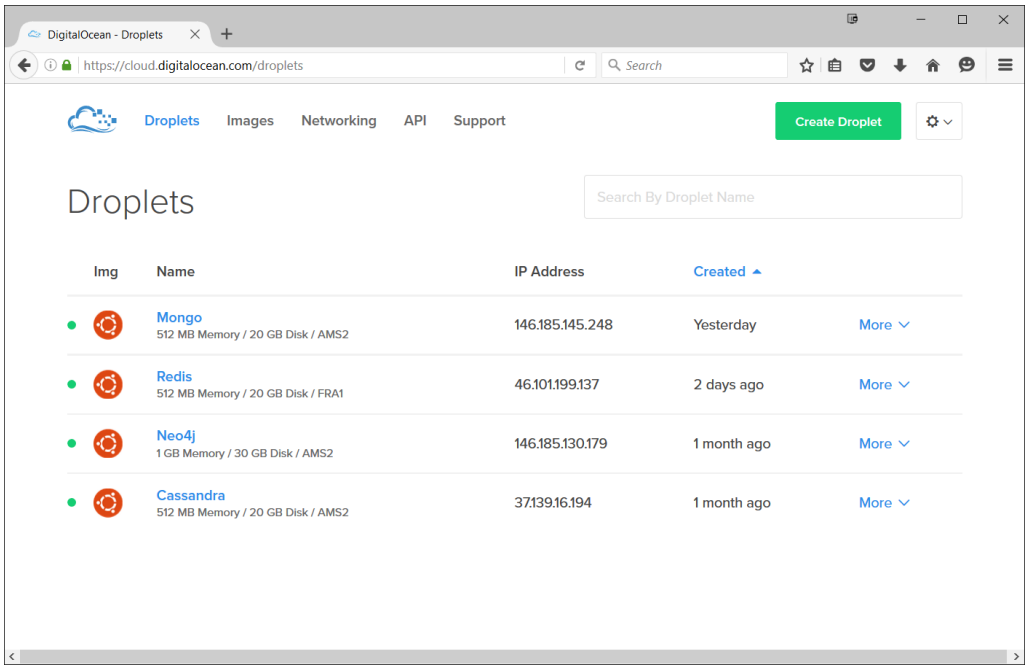
Ili, kao u našem slučaju, odabrati aplikaciju koju želimo imati instaliranu te je time odmah odabran i operativni sustav za koji je ta aplikacija konfigurirana.

Nakon toga, odabiremo koliko jak poslužitelj nam treba, regiju u kojoj želimo da se nalazi podatkovni centar te koliko želimo virtualnih poslužitelja pokrenuti.

Želimo li naknadno dodati novi virtualni poslužitelj to činimo jednostavno klikom na „Create droplet“ opciju unutar sučelja.



Slika 6.1 Odabir virtualnog poslužitelja koji ima već postavljenu Redis bazu podataka

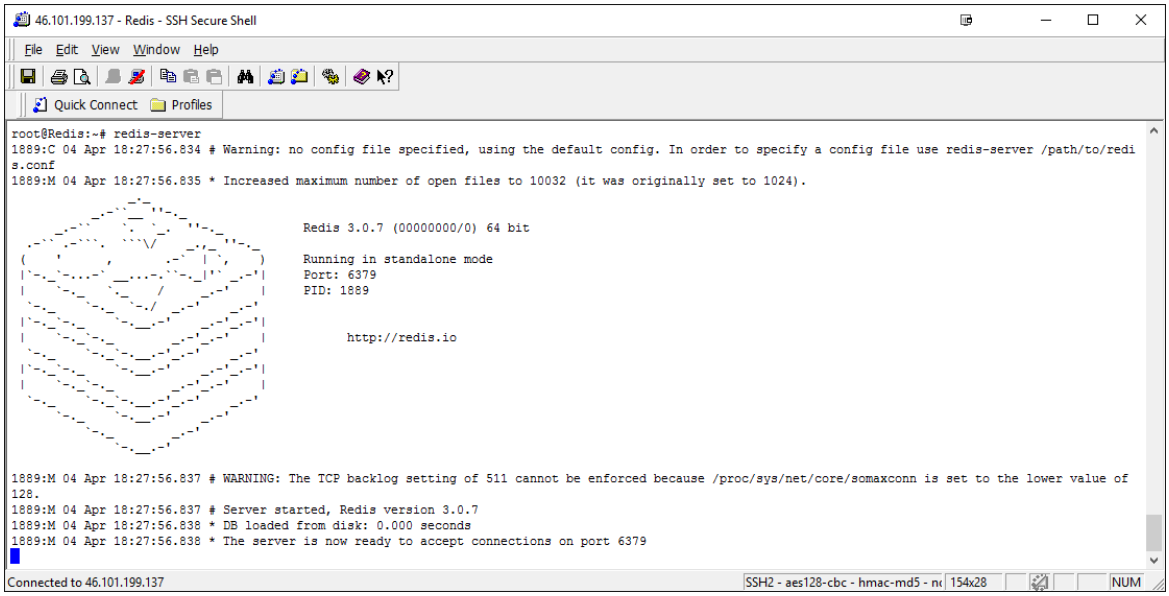


Slika 6.2 Prikaz virtualnih poslužitelja

6.1. Redis baza podataka

Redis je jednostavna i izrazito brza ključ-vrijednost baza podataka. Originalno je pohranjivala podatke samo u memoriji te s vremenom dodala mogućnost pohrane podataka na disk. U slučaju jedne instalacije Redis-a na poslužitelju, ostvarivanje veze se obavlja se koristeći naredbu u shell-u:

$ redis-server



Slika 6.3 Prikaz pokretanja Redis-a u shell sučelju

Osnovne funkcionalnosti Redis baze podataka možemo pokazati na primjeru baze podataka koja opslužuje aplikaciju za skraćivanje URL-a. Baza podataka sprema aplikacijski generirani ključ, koji sadrži originalni URL sa svim parametrima. Otvaranjem URL-a s generiranim ključem, aplikacija će pretražiti bazu podataka za ključ te ako ga nađe, preusmjerit će korisnika na prvobitni URL. Ako pretpostavimo da je aplikacija na adresi http://pmfurl.hr/, a originalni URL-a poput http://www.math.pmf.unizg.hr/?.... URL se može skratiti na http://pmfurl.hr/mpmf.

Za dodavanje novog zapisa u bazu podataka Redis koristi se naredba SET koja prima dva parametra, ključ i vrijednost. Dohvaćanje se obavlja naredbom GET samo s parametrom ključa. Brisanje se obavlja naredbom DEL koja također prima samo parametar ključa.

```
127.0.0.1:6379> set pmfm https://www.math.pmf.unizg.hr/
OK
127.0.0.1:6379> get pmfm
"https://www.math.pmf.unizg.hr/"
127.0.0.1:6379> del pmfm
(integer) 1
```

Naredbom RENAME mijenja se ime ključa, a naredbom TYPE dohvaća tip vrijednosti postavljen za zadani ključ.

Da bi se smanjio mrežni promet, mogu se postavljati višestruke vrijednosti odjednom, pomoću MSET naredbe. Slično, naredbi MGET se može predati više ključeva te dohvatiti njihove vrijednosti poredane u listu.

```
127.0.0.1:6379> MSET googleg http://www.google.com/ yahoo http://www.yahoo.com/
OK
127.0.0.1:6379> MGET googleg yahoo
1) "http://www.google.com/"
2) "http://www.yahoo.com/"
```

Redis prepoznaje tip Integer kao cijeli broj (iako baza radi s tipom String) i dopušta jednostavne operacije nad takvim brojevima. U slučaju da se želi pratiti koliko ima kratkih ključeva u bazi, može se stvoriti brojač (ključ „count“) te ga povećavati s INCR naredbom.

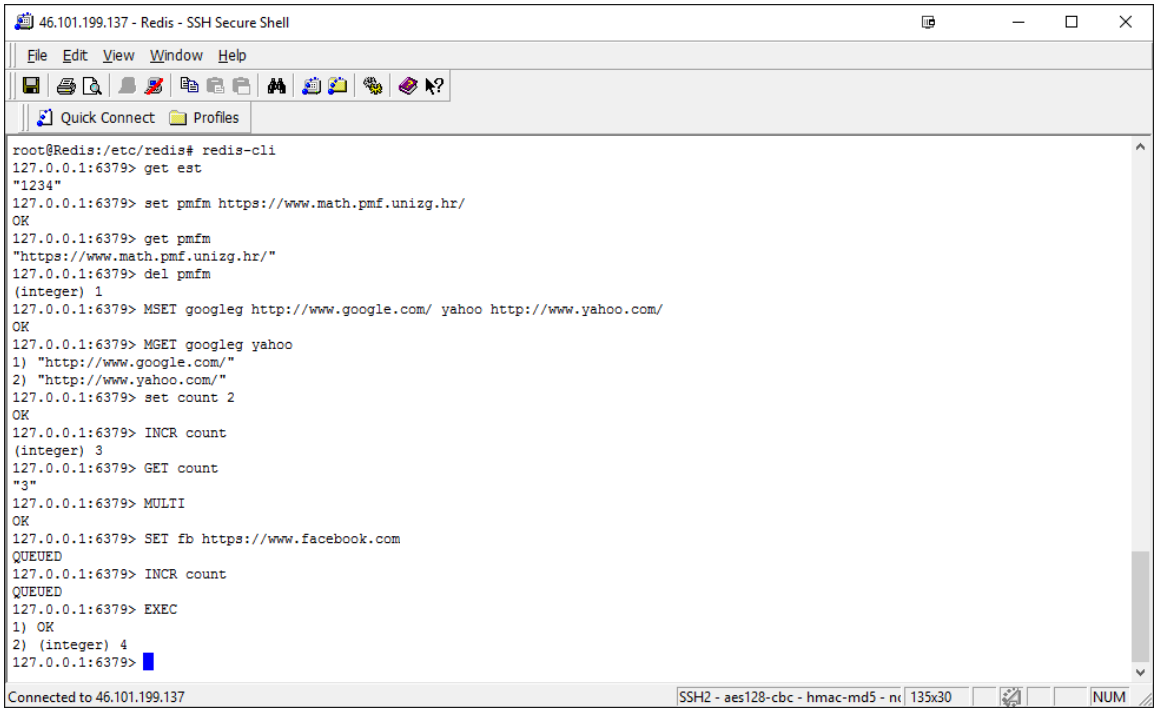
```
127.0.0.1:6379> SET count 2
OK
127.0.0.1:6379> INCR count
(integer) 3
127.0.0.1:6379> GET count
"3"
```

Povećati broj spremljen za neki ključ može se i za proizvoljnu veličinu naredbom INCRBY. Operacije smanjenja se mogu izvršavati naredbama DECR za jedan i DECRBY za proizvoljan broj.

Redis-ova MULTI atomarna blok-naredba ima sličan koncept kao transakcije u relacijskim bazama podataka. Omatajući operacije u MULTI blok znači da će sve naredbe biti izvršene, ili niti jedna.

Unutar jedne transakcije može se spremiti novi zapis za URL i također povećati brojač (count) zapisa. U tom slučaju transakcija započinje s MULTI naredbom, a izvršava se s EXEC naredbom. Ako je za vrijeme izvršenja transakcije potrebno odbaciti promjene te transakcije, to je moguće s DISCART naredbom, što je analogno naredbi ROLLBACK u SQL-u.

```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> SET fb https://www.facebook.com
QUEUED
127.0.0.1:6379> INCR count
QUEUED
127.0.0.1:6379> EXEC
1) OK
2) (integer) 2
```



Slika 6.4 Primjeri dodavanja i brisanja vrijednosti pomoću shell sučelja

Redis ima ugrađene funkcionalnosti za pohranjivanje lista, skupova, sortiranih skupova, rječnika i sličnih struktura.

Rječnici

Rječnici su u Redis-u ugniježđeni objekti koji mogu primiti bilo koji broj ključ-vrijednost parova. U našem primjeru, možemo ih koristiti za praćenje korisnika koji su se prijavili za servis skraćivanja URL-a.

Umjesto zasebnih ključ-vrijednosti parova, može se stvoriti rječnik koji će čuvati logički grupirane ključ-vrijednost parove. Ova metodologija često se koristi kada se želi simulirati ponašanje relacijske baze podataka koja u svakom retku ima definirane stupce. Za pohranu podataka u rječnik koristi se naredba HMSET te za dohvat naredba HVALS.

```
127.0.0.1:6379> HMSET user:nela name "Nela Tomic" password s3cret
OK
127.0.0.1:6379> HVALS user:nela
1) "Nela Tomic"
2) "s3cret"
```

Za dohvat jedne vrijednosti iz rječnika potrebno je proslijediti ključ rječnika i ključ tražene vrijednosti. U navedenom primjeru se dohvaća samo lozinka korisnika.

```
127.0.0.1:6379> HGET user:nela password
"s3cret"
```

Izbrisati vrijednosti iz rječnika obavlja se naredbom HDEL, a dohvaćanje broja polja u rječniku pomoću naredbe HLEN.

Liste

Liste sadrže višestruke sortirane vrijednosti i mogu se koristiti kao redovi (FIFO) ili stogovi (LIFO). Dopuštaju i operacije umetanja vrijednosti na bilo koju poziciju liste, ograničenja veličine liste, pomicanje elemenata među listama i slično.
Aplikacija iz primjera dopušta korisnicima da postave listu URL-ova koje žele posjetiti. Radi lakšeg pretraživanja i upravljanja podacima aplikacija koristi ključ formata korisničkoIme:listaZelja, gdje korisnickoIme je ovisno o vlasniku liste.

```
127.0.0.1:6379> RPUSH nela:listaZelja math google facebook
(integer) 3
```

Koristeći naredbu LRANGE može se dohvatiti bilo koji dio liste navodeći prvi i zadnji element.

Indeks liste počinje brojem 0. Negativan indeks liste označava indeks koji se počeo brojati od kraja liste.

```
127.0.0.1:6379> LRANGE nela:listaZelja 0 -1
1) "math"
2) "google"
3) "facebook"
```

Naredba LREM miče sa zadanog ključa navedene vrijednosti. Zahtijeva i broj koji označava koliko pronađenih vrijednosti želimo maknuti. Postavi li se taj broj na 0 označava da želimo maknuti sve pronađene vrijednosti.

```
127.0.0.1:6379> LREM nela:listaZelja 0 google
127.0.0.1:6379> LRANGE nela:listaZelja 0 -1
1) "math"
2) "facebook"
```

Postavljanjem broja na negativan uklanja se naveden broj vrijednosti ali krenuvši s kraja liste.

Skupovi

Skupovi su neuređene kolekcije elemenata koje ne sadrže duplicirane elemente. Korisni su za operacije nad podacima kao što su unija naredbom SUNION ili presjek naredbom SINTER .

Želi li se kategorizirati skup URL-ova sa zajedničkim ključem, to se može učiniti naredbom SADD. Dohvaćanje vrijednosti obavlja se naredbom SMEMBERS i parametrom ključa. SREM uklanja navedenu vrijednost za zadani ključ.

```
127.0.0.1:6379> SADD novosti nytimes.com jutarnji.hr
(integer) 2
127.0.0.1:6379> SMEMBERS novosti
1) "jutarnji.hr "
2) "nytimes.com"
127.0.0.1:6379> SREM novosti nytimes.com
```

Vrijeme isteka važnosti

Jedan od najčešćih slučajeva korištenja Redis baze podatka je za brz pristup podacima iz memorije. Ti podaci su često generirani i kratkog vijeka.

Postavljanjem parametra isteka važnosti nad ključevima i brisanjem istih kada im prođe rok, omogućava se da taj broj ključeva ne preraste zadanu granicu.

EXPIRE naredbom i brojem sekundi postavljamo koliko je dugo ključu dopušteno da živi to jest postavljamo ključu TTL parametar.

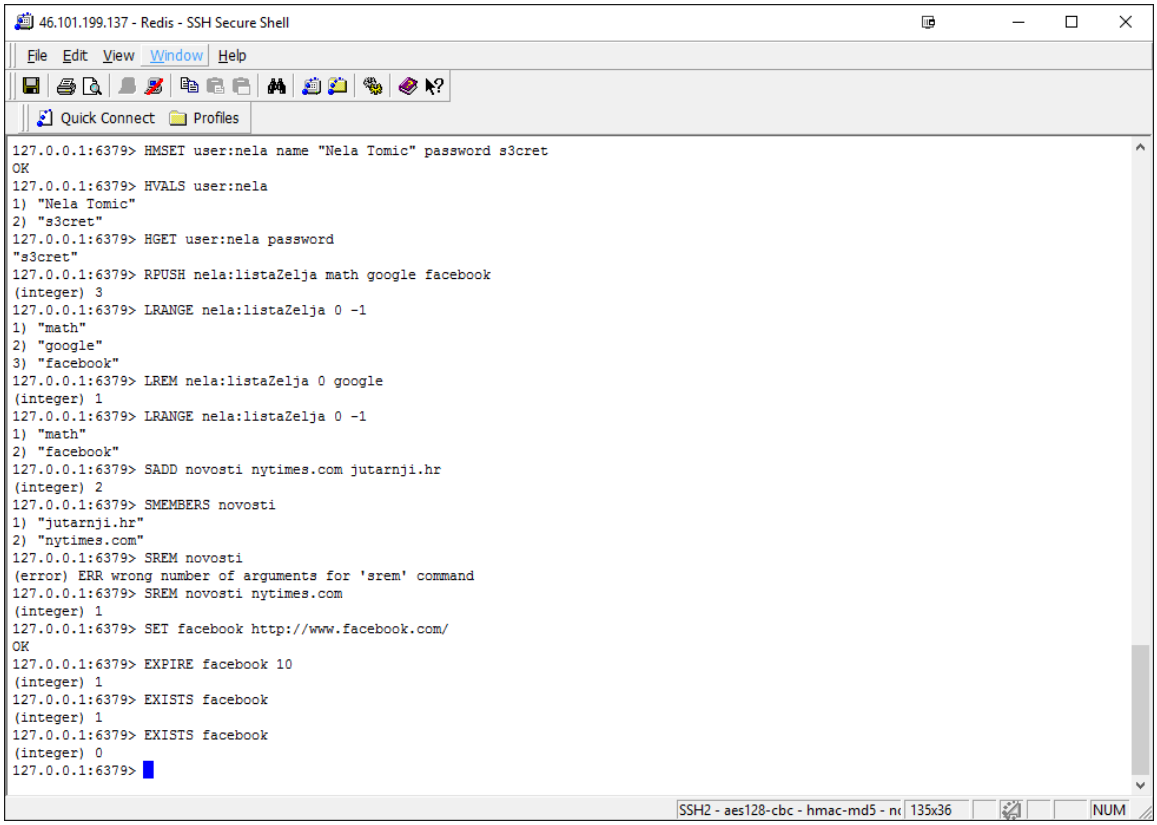
U primjeru postavljamo vrijeme života ključa na 10 sekundi te naredbom EXISTS provjeravamo postoji li ključ još uvijek (prije i nakon što je isteklo 10 sekundi).

```
127.0.0.1:6379> SET facebook http://www.facebook.com/
OK
127.0.0.1:6379> EXPIRE facebook 10
(integer) 1
127.0.0.1:6379> EXISTS facebook
(integer) 1
127.0.0.1:6379> EXISTS facebook
(integer) 0
```

Ove naredbe su toliko puno korištene u Redis-u, da postoji i skraćena naredba SETEX kojom postavljamo ključ TTL parametar i vrijednost.

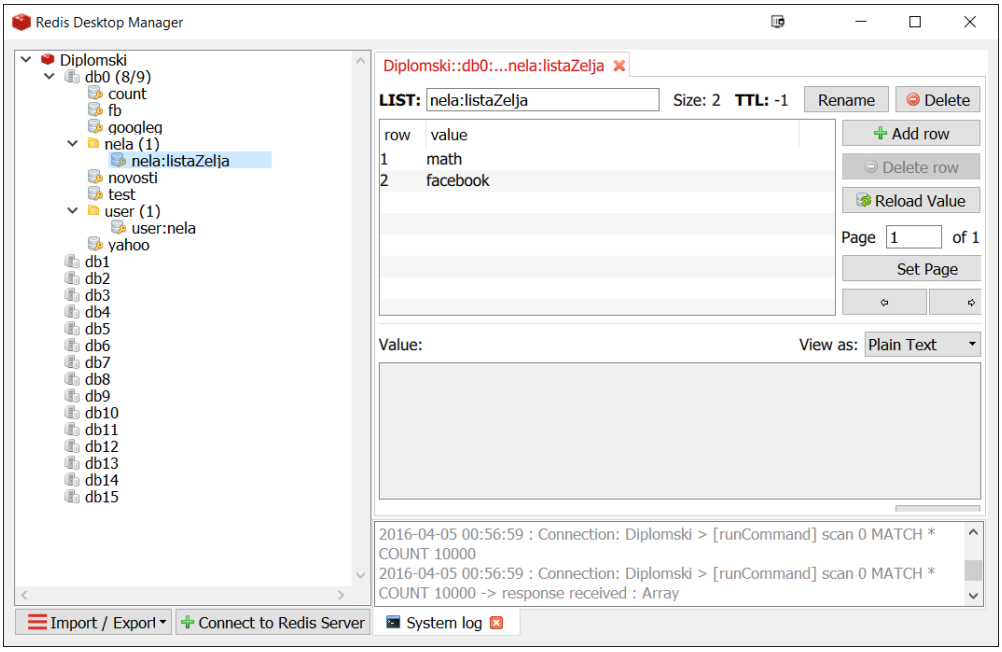
```
127.0.0.1:6379> SETEX facebook 10 www.facebook.com
```

Prije nego što istekne TTL vrijeme ključa, moguće ga je maknuti s ključa naredbom PERSIST.



Slika 6.5 Primjeri upravljanja kolekcijama

Preuzimanjem jednog od mnogih besplatnih Redis grafičkih sučelja dobivamo malo pregledniji uvid u stanje baze podataka nego što to imamo korištenjem shell-a.



Slika 6.6 Primjer jednog Redis grafičkog sučelja

6.2. MongoDB baza podataka

MongoDB danas je daleko najpopularnija dokument baza podataka. Odličan je izbor za projekte u kojima se često mijenja model podataka i kojima ubrzano raste količina podataka.

Osnovne funkcionalnosti MongoDB baze podataka možemo pokazati na primjeru baze podataka koja opslužuje aplikaciju za knjižnicu. Baza podataka sprema podatke o korisnicima knjižnice te njihovim preferencijama žanrova i listom želja.

Prilikom korištenja MongoDB potrebno je prvo kreirati bazu podataka za knjižnicu, što je moguće naredbom:

```
$mongo knjiznica
```

MongoDB sprema dokumente u kolekcije. Kolekcija je grupa povezanih dokumenata koji dijele skup indeksa. Slične su tablicama u relacijskim bazama.

Stvaranje nove kolekcije izvršava se automatski dodavanjem prvog dokumenta u kolekciju. Pri dodavanju novih dokumenata nema potrebe za definiranjem sheme.

Dodavanje novog dokumenta se obavlja naredbom INSERT kojoj se predaje dokument sa svim atributima. Objekt se predaje u standardnom JSON formatu gdje vitičaste zagrade označavaju sadržaj dokumenta. Dokumenti se mogu ugnježđivati u proizvoljnu dubinu.

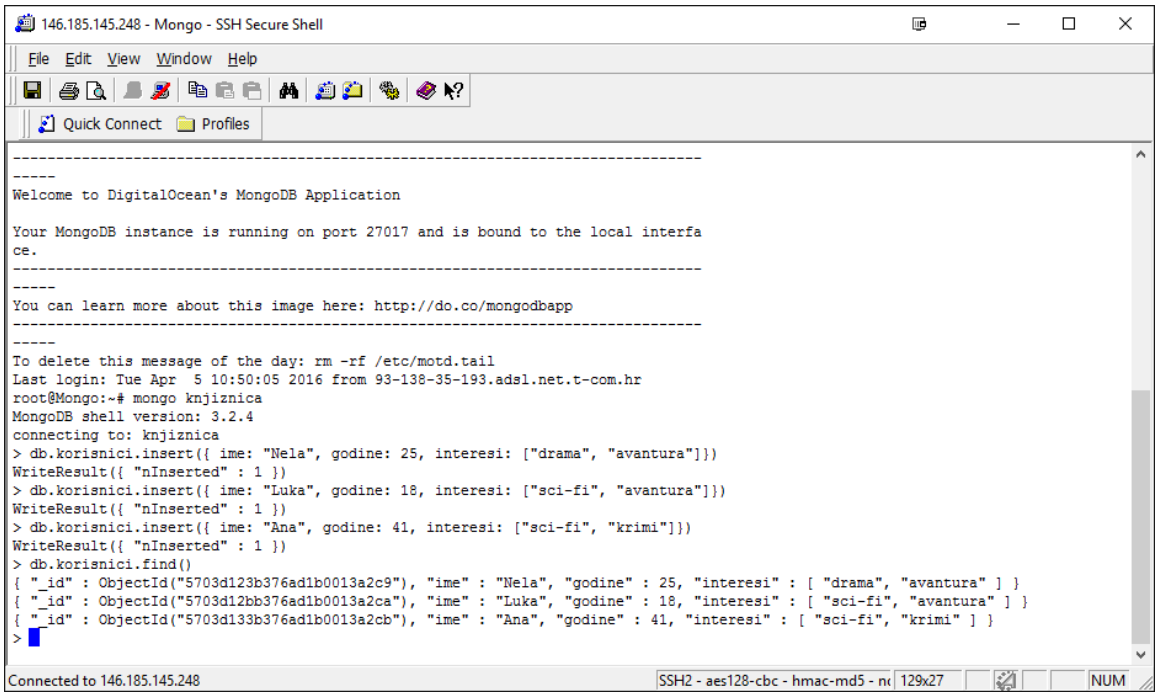
```
> db.korisnici.insert({ ime: "Nela", godine: 25, interesi: ["drama", "avantura"]})
> db.korisnici.insert({ ime: "Luka", godine: 18, interesi: ["sci-fi", "avantura"]})
> db.korisnici.insert({ ime: "Ana", godine: 41, interesi: ["sci-fi", "krimi"]})
```

Naredbom FIND se, ovisno o zadanim parametrima, pretražuju i vraćaju zapisi iz baze podatka. U slučaju kad se ne zadaju parametri pretraživanja, naredba vraća sve dokumente u zadanoj kolekciji.

```
db.korisnici.find(                  <- kolekcija
    { godine: { $gt: 25 } }         <- izraz upita
    { ime: 1}                       <- projekcija
).limit(5)                          <- modifikator kursora
```

```
> db.korisnici.find()
{ "_id" : ObjectId("5703d123b376ad1b0013a2c9"), "ime" : "Nela", "godine" : 25, "interesi" : [ "drama", "avantura" ] }
{ "_id" : ObjectId("5703d12bb376ad1b0013a2ca"), "ime" : "Luka", "godine" : 18, "interesi" : [ "sci-fi", "avantura" ]}
{ "_id" : ObjectId("5703d133b376ad1b0013a2cb"), "ime" : "Ana", "godine" : 41, "interesi" : [ "sci-fi", "krimi" ] }
```

Identifikator objekta (_id) automatski je generiran od strane MongoDB-a i time uvijek jedinstven. U slučaju da postoji potreba može se i ručno postaviti pri unosu dokumenta.



Slika 6.7 Primjer korištenja INSERT i FIND naredbi

Upiti

Za dohvat točno određenog korisnika potrebno je pretražiti kolekciju pomoću identifikatora. Ako je identifikator automatski generiran od strane baze, tip podatka mu je ObjectId te ga je u upitu potrebno postaviti u odgovarajući tip. Pretvorba objekta se može obaviti pomoću ugrađene funkcije ObjectId($kljuc as String).

```
> db.korisnici.find({"_id": ObjectId("5703d123b376ad1b0013a2c9")})
{ "_id" : ObjectId("5703d123b376ad1b0013a2c9"), "ime" : "Nela", "godine" : 25, "interesi" : [ "drama", "avantura" ] }
```

Funkcija find() prihvaća i opcionalne parametre, mogu se navesti imena atributa koje se želi dohvatiti. Ako se atribut postavi na vrijednost 1 (true), dohvaćaju se samo zadani parametri, a ako se postavi na 0 dohvaćaju se svi parametri osim zadanog.

```
> db.korisnici.find({"_id": ObjectId("5703d123b376ad1b0013a2c9")}, {ime: 1})
{ "_id" : ObjectId("5703d123b376ad1b0013a2c9"), "ime" : "Nela" }

> db.korisnici.find({"_id": ObjectId("5703d123b376ad1b0013a2c9")}, {ime: 0})
{ "_id" : ObjectId("5703d123b376ad1b0013a2c9"), "godine" : 25, "interesi" : [ "drama", "avantura" ] }
```

Upit je moguće graditi kombinacijom kriterija, kao što su atributi, raspon i slično. Žele li se dohvatiti svi korisnike koji imaju manje godina od 26, to se može učiniti na sljedeći način:

```
> db.korisnici.find( {godine : {$lt : 26}}, { ime: 1, godine: 1})
{ "_id" : ObjectId("5703d123b376ad1b0013a2c9"), "ime" : "Nela", "godine" : 25 }
{ "_id" : ObjectId("5703d12bb376ad1b0013a2ca"), "ime" : "Luka", "godine" : 18 }
```

Naredba	Opis
$regex	Regularni izraz
$ne	Nije isto kao
$lt	Manje od
$lte	Manje od ili jednako
$gt	Veće od
$gte	Veće od ili jednako

Slika 6.8 Neke od naredbi za kriterije upita

Analogno tome, upiti se mogu postavljati i nad ugniježđenim strukturama. Na primjer atribut „interesi“ je polje koje pretražujemo za vrijednost „sci-fi“.

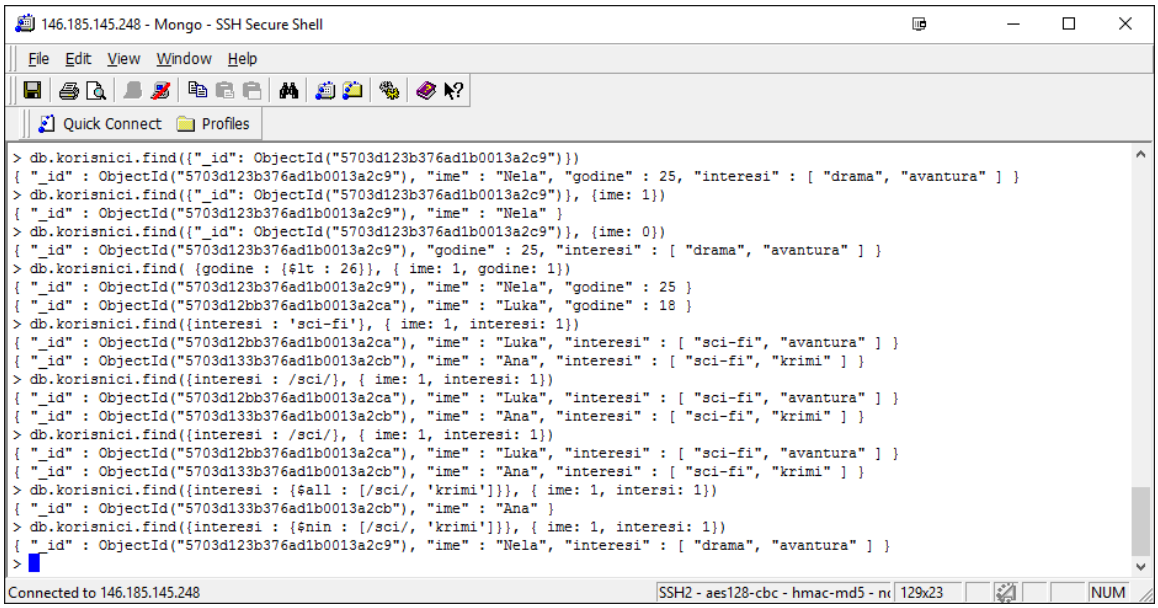
```
> db.korisnici.find({interesi : 'sci-fi'}, { ime: 1, interesi: 1})
{ "_id" : ObjectId("5703d12bb376ad1b0013a2ca"), "ime" : "Luka", "interesi" : [ "sci-fi", "avantura" ] }
{ "_id" : ObjectId("5703d133b376ad1b0013a2cb"), "ime" : "Ana", "interesi" : [ "sci-fi", "krimi" ] }
```

Ili nad parcijalnim stringovima:

```
> db.korisnici.find({interesi : /sci/}, { ime: 1, interesi: 1})
{ "_id" : ObjectId("5703d12bb376ad1b0013a2ca"), "ime" : "Luka", "interesi" : [ "sci-fi", "avantura" ] }
{ "_id" : ObjectId("5703d133b376ad1b0013a2cb"), "ime" : "Ana", "interesi" : [ "sci-fi", "krimi" ] }
```

MongoDB omogućuje i upite kojima zahtijevamo da svi uvjeti moraju biti zadovoljen ili niti jedan:

```
> db.korisnici.find({interesi : {$all : [/sci/, 'krimi']}}, { ime: 1, intersi: 1})
{ "_id" : ObjectId("5703d133b376ad1b0013a2cb"), "ime" : "Ana" }>
> db.korisnici.find({interesi : {$nin : [/sci/, 'krimi']}}, { ime: 1, interesi: 1})
{ "_id" : ObjectId("5703d123b376ad1b0013a2c9"), "ime" : "Nela", "interesi" : [ "drama", "avantura" ] }
```



Slika 6.9 Primjeri upita nad bazom

Ažuriranje

MongoDB podržava parcijalno ažuriranje skupa atributa u dokumentu. UPDATE naredba poziva se s dva parametra: kriterij i operacija. Prvi je kriterij po kojem se ažurira, a drugi može biti objekt koji će zamijeniti postojeće vrijednosti, ili modifikator $set koji

zamjenjuje samo određene atribute. Na primjeru želimo proširiti objekt korisnika da sadrži listu želja.

```
> db.korisnici.update({ime: 'Nela'}, {$set: {zelje : { knjige: ['Clash of Titans',
'Squall'] } } })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })


> db.korisnici.find({"_id": ObjectId("5703d123b376ad1b0013a2c9")})
{ "_id" : ObjectId("5703d123b376ad1b0013a2c9"), "ime" : "Nela", "godine" : 25, "interesi" :
[ "drama", "avantura" ], "zelje" : { "knjige" : [ "Clash of Titans", "Squall" ] } }
```

Definiranjem gornje operacije bez $set modifikatora, postojeći dokument bi se zamijenio novim.

Naredba	Opis
$set	Postavlja polje sa zadanom vrijednosti
$unset	Briše zadano polje
$inc	Uvećava broj u zadanom polju
$pop	Briše zadnji (ili prvi) element u polju
$push	Dodaje element u polje
$pushAll	Dodaje sve vrijednosti u elemente polja
$addToSet	Slično kao $push, ali ne dodaje duple vrijednosti
$pull	Briše vrijednost iz polja koje zadovoljava uvjet
$pullAll	Briše sve vrijednosti iz polja koje zadovoljavaju uvjet

Slika 6.10 Modifikatori funkcije ažuriranja

Reference

Zbog svoje distributivne prirode, MongoDB nije osmišljen za rad s join operacijama. Ponekad je ipak korisno čuvati reference dokumenata. U tom slučaju se koristi konstrukcija poput:

{ $ref : "collection_name", $id : "reference_id" }.

Aplikacija knjižnice se može proširiti s gradovima u kojima žive korisnici te ih pridijeliti korisniku kao referencu.

```
> db.korisnici.update({_id: ObjectId("5703d123b376ad1b0013a2c9")}, {$set: { grad: {$ref:
'gradovi', $id: "zgb"} }})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.korisnici.find({"_id": ObjectId("5703d123b376ad1b0013a2c9")})
{ "_id" : ObjectId("5703d123b376ad1b0013a2c9"), "ime" : "Nela", "godine" : 25, "interesi" :
[ "drama", "avantura" ], "zelje" : { "knjige" : [ "Clash of Titans", "Squall" ] }, "grad" :
DBRef("gradovi", "zgb") }
```

Navedeni primjer prikazuje da nakon što je korisniku referenciran grad, postavljanjem upita nad korisnikom, baza podataka vraća i referencu na grad.

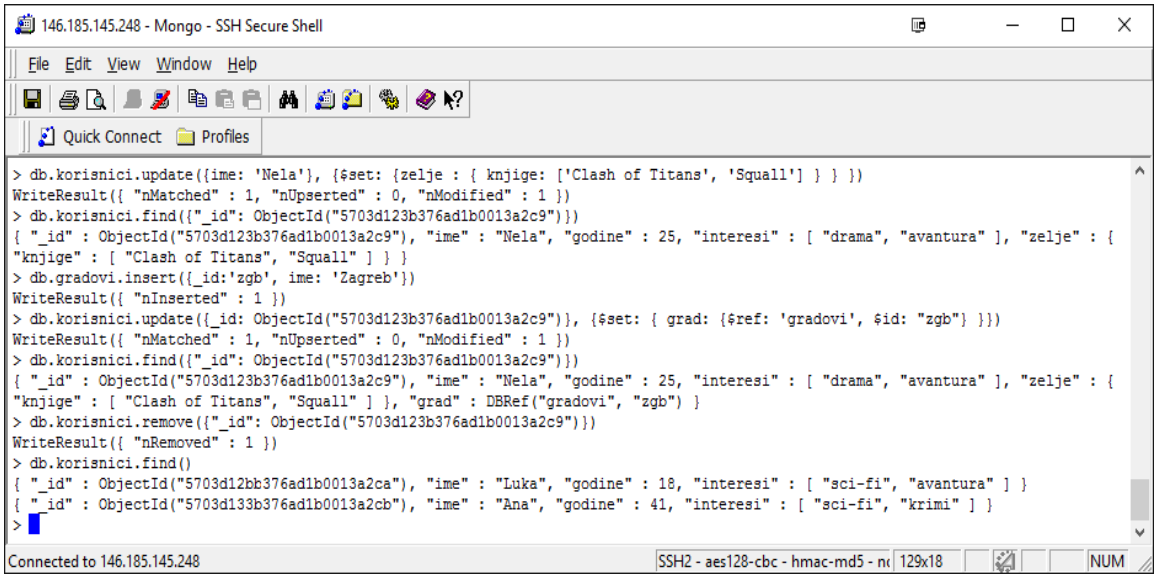
Brisanje

Brisanje dokumenata iz kolekcije odvija se na sličan način kao korištenje find() funkcije. Zamijeni li se ključna riječ find() s remove(), svi dokumenti koji odgovaraju kriteriju biti će pobrisani.

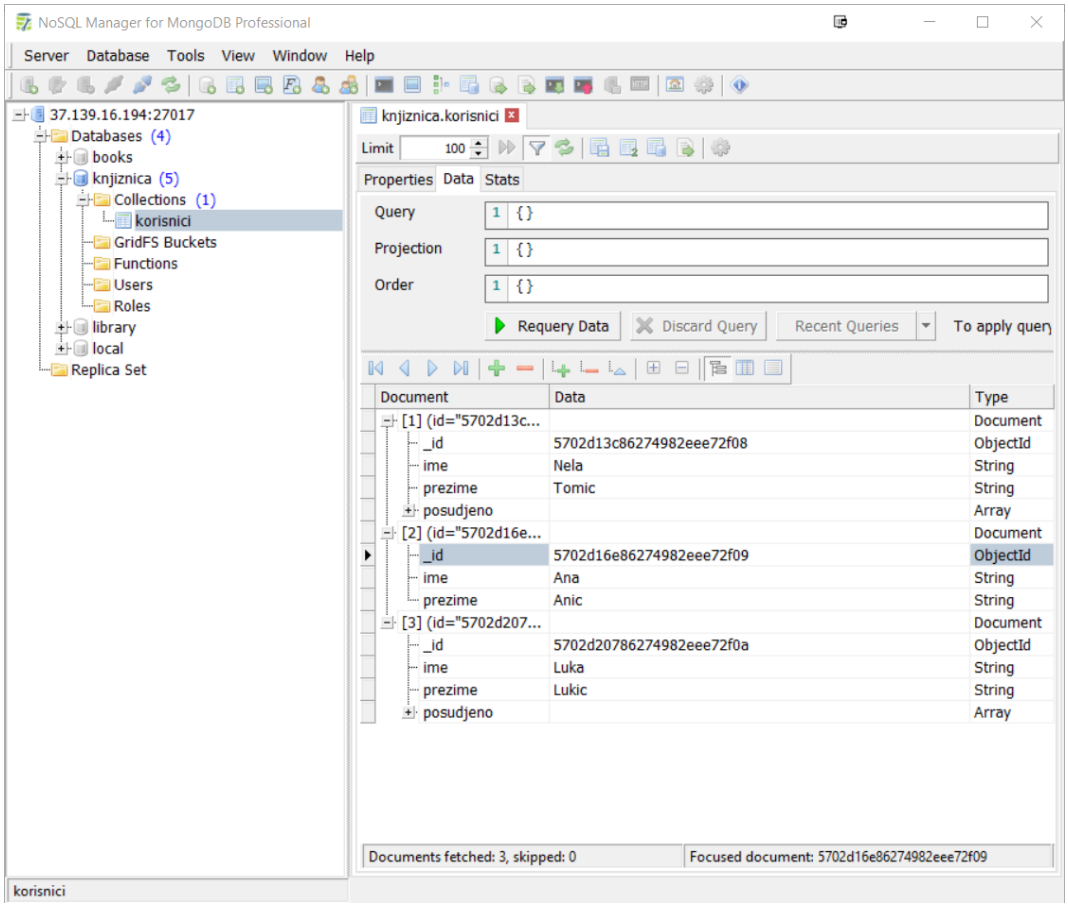
Zato je preporučeno prvo koristiti find() da bi se provjerili kriteriji, a tek onda remove().

```
> db.korisnici.remove({"_id": ObjectId("5703d123b376ad1b0013a2c9")})
WriteResult({ "nRemoved" : 1 })

> db.korisnici.find()
{ "_id" : ObjectId("5703d12bb376ad1b0013a2ca"), "ime" : "Luka", "godine" : 18, "interesi" :
[ "sci-fi", "avantura" ] }
{ "_id" : ObjectId("5703d133b376ad1b0013a2cb"), "ime" : "Ana", "godine" : 41, "interesi" :
[ "sci-fi", "krimi" ] }
```



Slika 6.11 Primjer ažuriranja i brisanja



Slika 6.12Prikaz jednog grafičkog sučelja MongoDB baze

6.3. Cassandra baza podataka

Apache Cassandra jest popularna implementacija stupčanog modela za bazu podataka. Sustav je inicijalno razvijen u Facebook-u za funkcionalnosti pohranjivanja poruka i pretraživanja poštanskog sandučića. Osmišljen je za rad s velikom količinom podataka te pruža visoku dostupnost i skalabilnost istih. Model podataka Cassandra baze podataka može se zamisliti kao puno redaka od kojih svaki sadrži listu. Iz tih lista mogu se dodavati i uklanjati vrijednosti. Elementi liste koji ne sadrže nikakvu vrijednost jednostavno se mogu ostaviti praznima. Cassandra koristi upitni jezik CQL (*Cassandra Query Language*). Interakcija s bazom se obavljati koristeći CQL shell, cqlsh.

Osnovne funkcionalnosti Cassandra baze podataka možemo pokazati na primjeru baze podataka koja opslužuje aplikaciju Wikipedia. Baza podataka sadrži stranice sa sadržajem i verzijama ažuriranja stranica.

Funkcionalnost baze podataka Cassandra možemo prikazati implementacijom baze za aplikaciju Wikipedia koja sadrži stranice sa sadržajem i verzijama ažuriranja stranica. Kao primarni ključ koristimo naslov stranice koji je jedinstven. U pokaznom primjeru koristi se faktor replikacije 1 koji označava da se baza podataka ne replicira, ali u stvarnim sustavima se može definirati kompleksna replikacija.

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

cqlsh> USE Wikipedia;
```

U novostvorenom imeniku potrebno je kreirati tablicu koja čuva podatke.

```
cqlsh:wikipedia> CREATE TABLE wikipages (naslov text, tekst text, verzije map<timestamp,
text>, PRIMARY KEY (naslov));
```

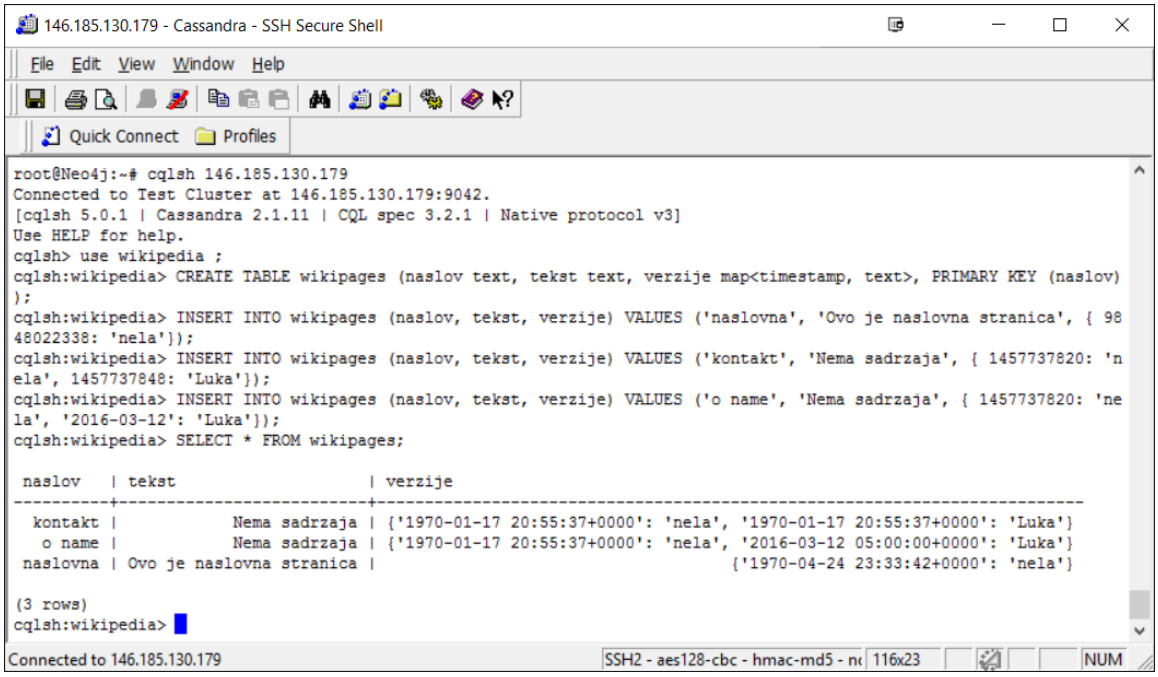
Stupci, osim osnovnih tipova podataka, mogu sadržavati kolekcije, koje mogu biti liste, skupovi i slično. Dodavanje novih zapisa u tablicu se obavlja naredbom INSERT kojoj je potrebno definirati stupce koji se žele popuniti te njihove vrijednosti.

```
cqlsh:wikipedia>  INSERT INTO wikipages (naslov, tekst, verzije) VALUES ('naslovna', 'Ovo
je naslovna stranica', { 9848022338: 'nela'});

cqlsh:wikipedia>  INSERT INTO wikipages (naslov, tekst, verzije) VALUES ('kontakt', 'Nema
sadrzaja', { 1457737820: 'nela', 1457737848: 'Luka'});

cqlsh:wikipedia>  INSERT INTO wikipages (naslov, tekst, verzije) VALUES ('o name', 'Nema
sadrzaja', { 1457737820: 'nela', '2016-03-12': 'Luka'});

cqlsh:wikipedia> SELECT * FROM wikipages;
```



Slika 6.13 Primjer jednostavnih operacija nad bazom

Upiti

Naredbom SELECT se u ovisnosti o parametrima pretražuju i dohvaćaju podaci iz tablice. U slučaju da nismo zadali parametre pretraživanja, baza podataka vraća sve zapise u tablici. Parametri pretraživanja se zadaju pomoću WHERE naredbe nakon koje se navode uvjeti. Za dohvat točno određene stranice najbolje je definirati primarni ključ u uvjetu pretraživanja.

```
cqlsh:wikipedia> SELECT * FROM wikipages WHERE naslov = 'naslovna' ;
```

Naredba SELECT može označiti koji stupci se dohvaćaju. Ako želimo dohvatiti sve stupce šaljemo oznaku * (zvjezdica). Ako želimo smanjiti broj stupaca potrebno je navesti imena stupaca u SELECT naredbi odvojenih zarezom.

```
cqlsh:wikipedia> SELECT naslov, tekst FROM wikipages WHERE naslov = 'naslovna';
```

Analogno uvjetima u relacijskoj bazi podataka, cqlsh podržava široku funkcionalnost uvjeta za filtriranje podataka koji su prikazani u tablici.

Uvjet	Opis
=	Jednako
>	Veće od
>=	Veće jednako
<	Manje
<=	Manje jednako
IN	Unutar (jedno od)
CONTAINS	Sadrži (za kolekcije)

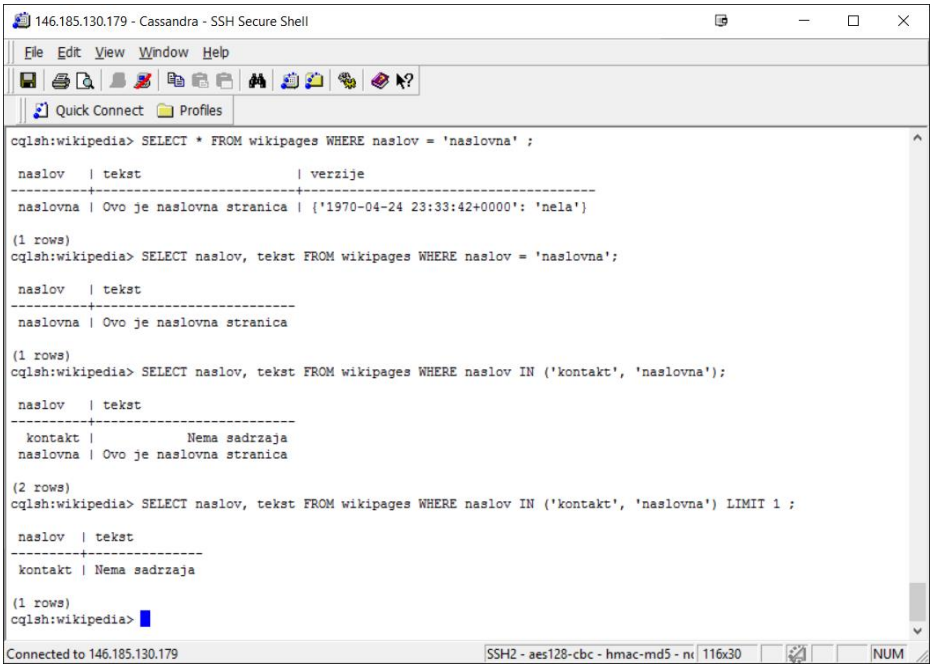
Slika 6.14 Neki od podržanih uvjeti filtriranja

Želimo li pronaći sve stranici u tablici kojima je naslov: kontakt ili naslovna možemo upotrijebiti uvjet IN koji prima listu String-ova kao parametar.

```
qlsh:wikipedia> SELECT naslov, tekst FROM wikipages WHERE naslov IN ('kontakt', naslovna');
```

Kako stupčane baze podataka služe za pohranu velikih količina podataka, često je potrebno ograničiti količinu vraćenih podataka koje upit vraća. Ograničavanje se može ostvariti korištenjem naredbe LIMIT koja kao parametar prima broj rezultata na koliko je ograničeno.

```
cqlsh:wikipedia> SELECT naslov, tekst FROM wikipages WHERE naslov IN ('kontakt', 'naslovna')
LIMIT 1 ;
```



Slika 6.15 Primjer korištenja SELECT naredbe

Ažuriranje

Cassandra dopušta parcijalno ažuriranje vrijednosti stupaca i kolekcija. Ažuriranje se obavlja naredbom UPDATE koja definira u kojoj tablici se nalazi zapis koji se ažurira. Uz to je potrebno navesti naredbu SET kojom se definira koji stupac se mijenja te koja je nova vrijednost. Navodimo i uvjet filtriranja podataka koji definira nad kojim podacima se izvršava ažuriranje. U slučaju da se uvjet filtriranja ne navede ažuriraju se svi zapisi u tablici.

```
cqlsh:wikipedia> UPDATE wikipages SET tekst = 'Ovo je stranica o nama' WHERE naslov = 'o
nama';

cqlsh:wikipedia> UPDATE wikipages SET verzije = {'2016-03-12': 'Ivan'}  WHERE naslov = 'o
nama';
cqlsh:wikipedia> select * from wikipages where naslov = 'o nama' ;
```

Vrijednosti kolekcija ažuriraju se analogno ažuriranju stupaca, osim što je potrebno i navesti koja vrijednost unutar kolekcije se ažurira.

```
cqlsh:wikipedia> UPDATE wikipages SET verzije['2016-03-11'] = 'Nela'  WHERE naslov = 'o
nama';
cqlsh:wikipedia> select * from wikipages where naslov = 'o nama' ;
```

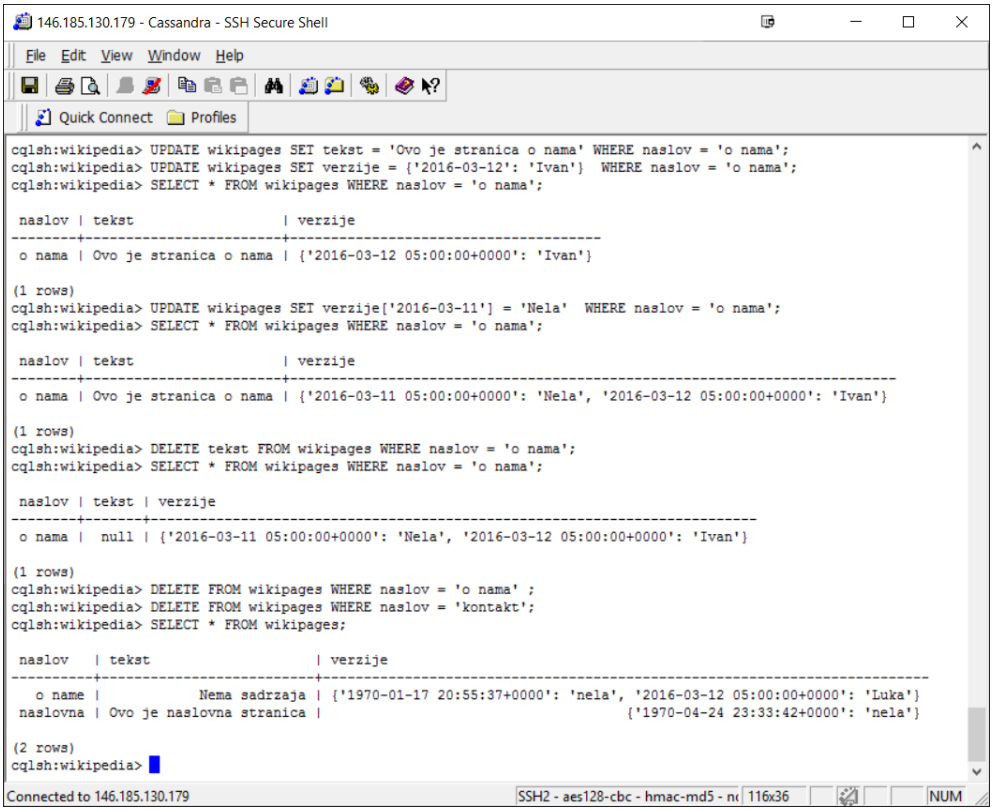
Brisanje

Cassandra prepoznaje više razina brisanja podataka, brisanje iz stupaca i brisanja cijelih zapisa. Brisanje se obavlja naredbom DELETE i sama funkcionalnost je određena zadanim parametrima. Ako naredbi predamo naziv jednog ili više stupaca obrisat će se samo vrijednost u stupcima.

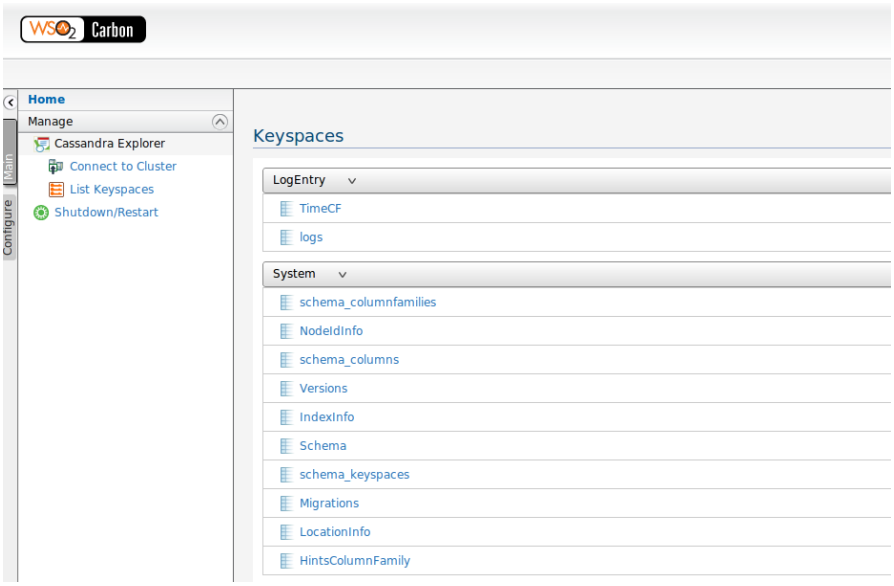
```
cqlsh:wikipedia> DELETE tekst FROM wikipages WHERE naslov = 'o nama';
cqlsh:wikipedia> select * from wikipages where naslov = 'o nama' ;
```

Ako imena stupaca nisu definirana, DELETE briše cijeli redak po uvjetima filtriranja podataka.

```
cqlsh:wikipedia> DELETE FROM wikipages WHERE naslov = 'o nama' ;
cqlsh:wikipedia> DELETE FROM wikipages WHERE naslov = 'kontakt';
cqlsh:wikipedia> select * from wikipages;
```



Slika 6.16 Primjer operacija ažuriranja i brisanja



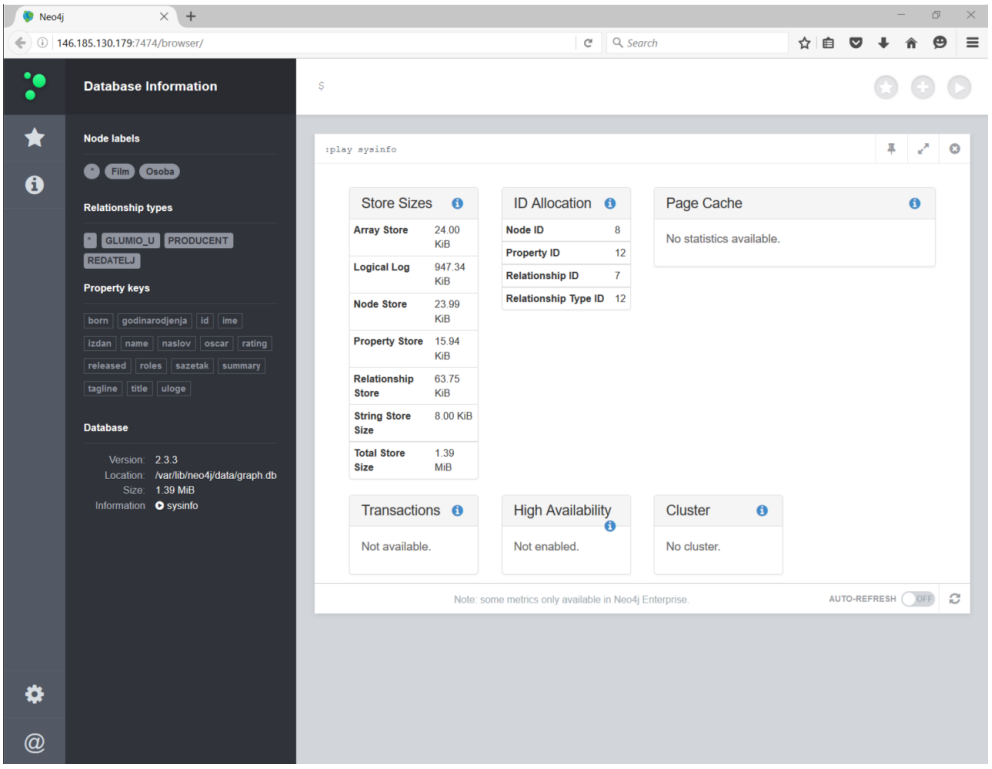
Slika 6.17 Primjer jednog grafičkog sučelja Cassandra baze

6.4. Neo4j baza podataka

Neo4j jest graf baza podataka koja se koristi pretežito u svrhu pohranjivanja veza među objektima. Dopušta stvaranje varijabilnih podataka na jednostavan i prirodan način. Hvali se time da može pohraniti desetke milijardi objekata i jednako toliko veza među njima.

Osnove funkcionalnosti Neo4j baze podataka mogu se prikazati korištenjem Cypher jezika na primjeru baze podataka koja opslužuje aplikaciju videoteka. Baza podataka čuva podatke o glumcima te filmovima u kojima su glumili.

Neo4j u svom skupu alata uključuje web-baziran shell preglednik koji se koristi za postavljanje upita i prikazivanje rezultata.



Slika 6.18 Neo4j web-baziran shell preglednik

Neo4j koristi JSON strukturu za postavljanje upita. Sama izjava nalazi se u atributu „statement“.

Standardni upit ima sljedeću formu:

```
{
   "statements":[
      {
         "statement":"neki izraz",
         "resultDataContents":[
            "row",
            "graph"
         ],
         "includeStats":true
      }
   ]
}
```

Za korištenje Neo4j baze podataka nije potrebno definirati sheme podataka koji će se spremati.

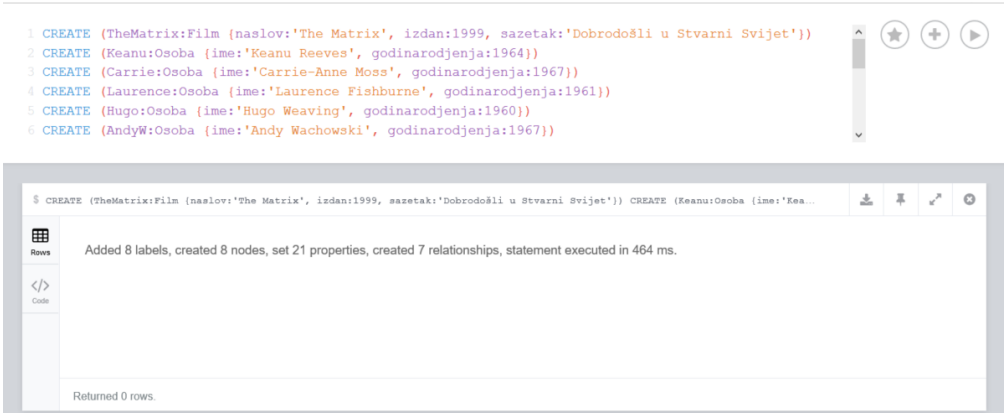
Podaci se dodaju korištenjem naredbe CREATE. Koja je u svom osnovnim obliku sastavljena od imena novog čvora (npr. „TheMatrix“) te tipa čvora („Film“). Podaci vezani za čvor se predaju u JSON formatu koji su sadržani unutar vitičastih zagrada.

Dodavanje podataka u bazu činimo pomoću CREATE naredbe.

```
CREATE (TheMatrix:Film {naslov:'The Matrix', izdan:1999, sazetak:'Dobrodošli u Stvarni
Svijet'})
CREATE (Keanu:Osoba {ime:'Keanu Reeves', godinarodjenja:1964})
CREATE (Carrie:Osoba {ime:'Carrie-Anne Moss', godinarodjenja:1967})
CREATE (Laurence:Osoba {ime:'Laurence Fishburne', godinarodjenja:1961})
CREATE (Hugo:Osoba {ime:'Hugo Weaving', godinarodjenja:1960})
CREATE (AndyW:Osoba {ime:'Andy Wachowski', godinarodjenja:1967})
CREATE (LanaW:Osoba {ime:'Lana Wachowski', godinarodjenja:1965})
CREATE (JoelS:Osoba {ime:'Joel Silver', godinarodjenja:1952})
```

Analogno dodavanju novih čvorova dodaju se i veze između čvorova.

```
CREATE
  (Keanu)-[:GLUMIO_U {uloge:['Neo']}]->(TheMatrix),
  (Carrie)-[:GLUMIO_U {uloge:['Trinity']}]->(TheMatrix),
  (Laurence)-[:GLUMIO_U {uloge:['Morpheus']}]->(TheMatrix),
  (Hugo)-[:GLUMIO_U {uloge:['Agent Smith']}]->(TheMatrix),
  (AndyW)-[:REDATELJ]->(TheMatrix),
  (LanaW)-[:REDATELJ]->(TheMatrix),
  (JoelS)-[:PRODUCENT]->(TheMatrix)
```

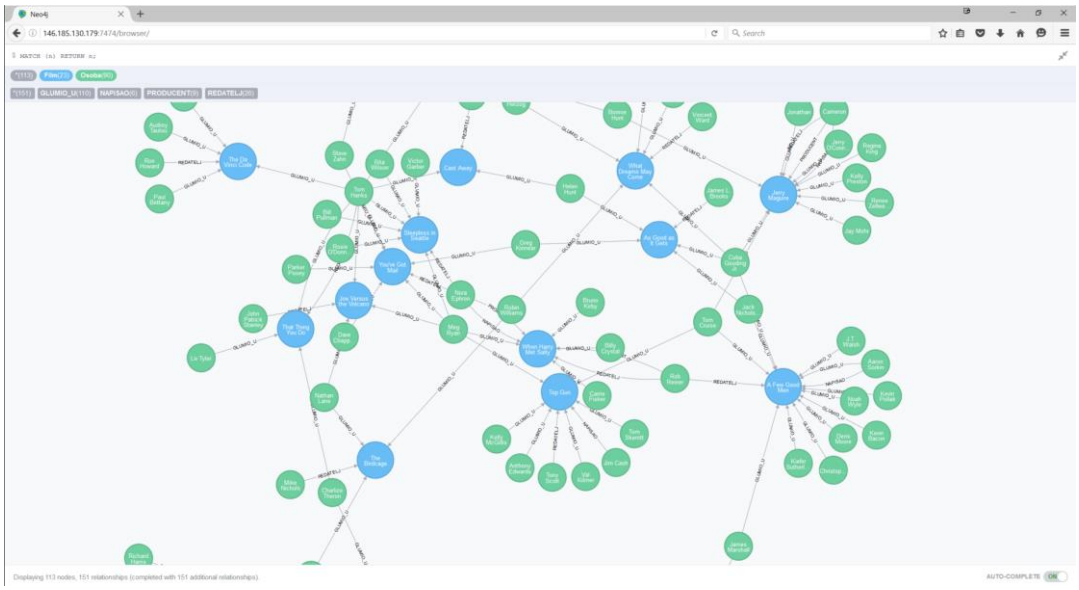


Slika 6.19 Dodavanje novih čvorova i veza u bazu podataka

U slučaju dodavanja veza između čvorova, CREATE naredba prihvaća ključ početnog čvora, naziv relacije koja se stvara s opcionalnim atributima te naziv odredišnog čvora. Neo4j podržava samo jednosmjerne veze. Baza podataka dana u ovom primjeru dodano je nadopunjena s većom količinom filmova, glumaca i njihovih veza na isti način kao što je prikazano. Kako bi se ispisali svi podaci koje baza sadrži koristi se naredba:

```
MATCH (n) RETURN n;
```

Ovakvim upitom dohvaćaju se svi rezultati u JSON formatu, no Neo4j preglednik pruža vizualnu reprezentaciju rezultata.

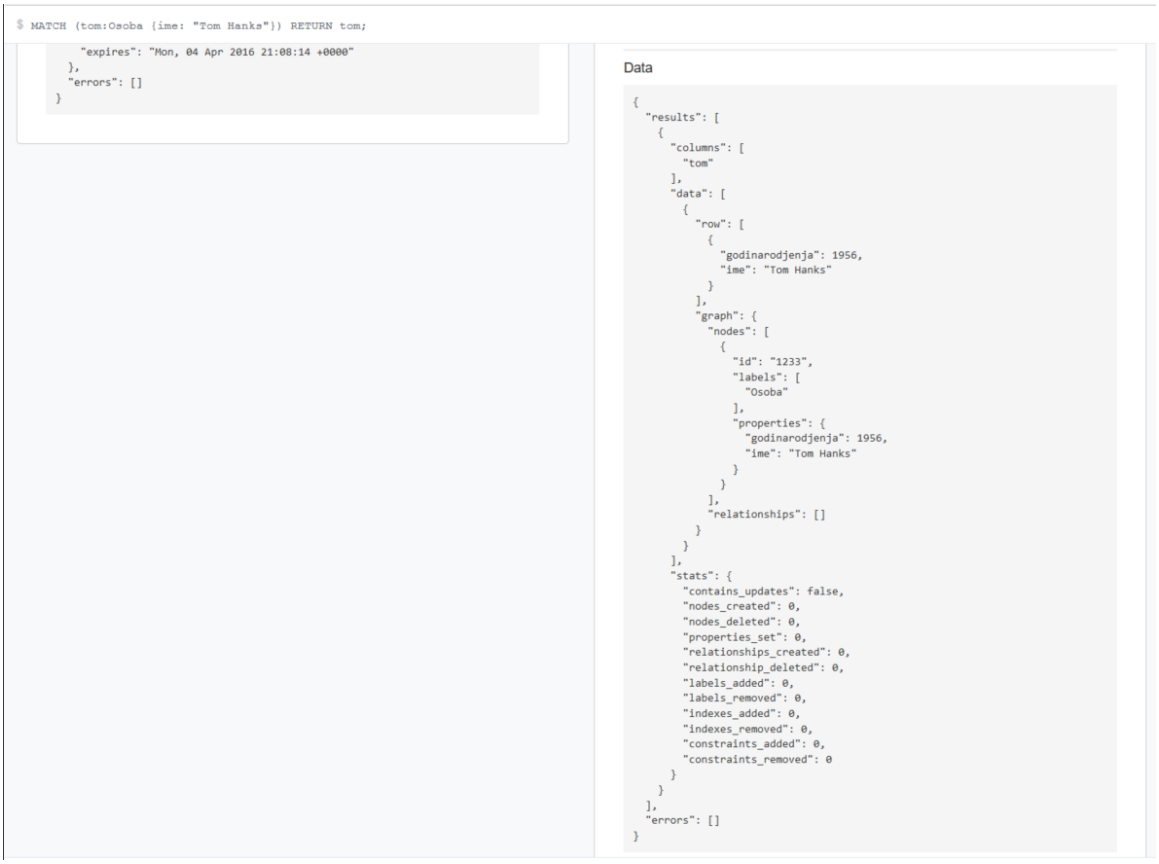


Slika 6.20 Vizualni prikaz svih čvorova i veza u bazi podataka

Upiti

U slučaju kada se naredbi MATCH ne zadaju parametri i uvjeti filtriranja, naredba vraća sve podatke u bazi. Uvjeti filtriranja mogu se zadati na dva načina; unutar naredbe MATCH ili dodatnim korištenjem naredbe WHERE. Za jednostavne upite dovoljno je koristiti atribute za pretraživanje.

```
MATCH (tom:Osoba {ime: "Tom Hanks"}) RETURN tom;
```



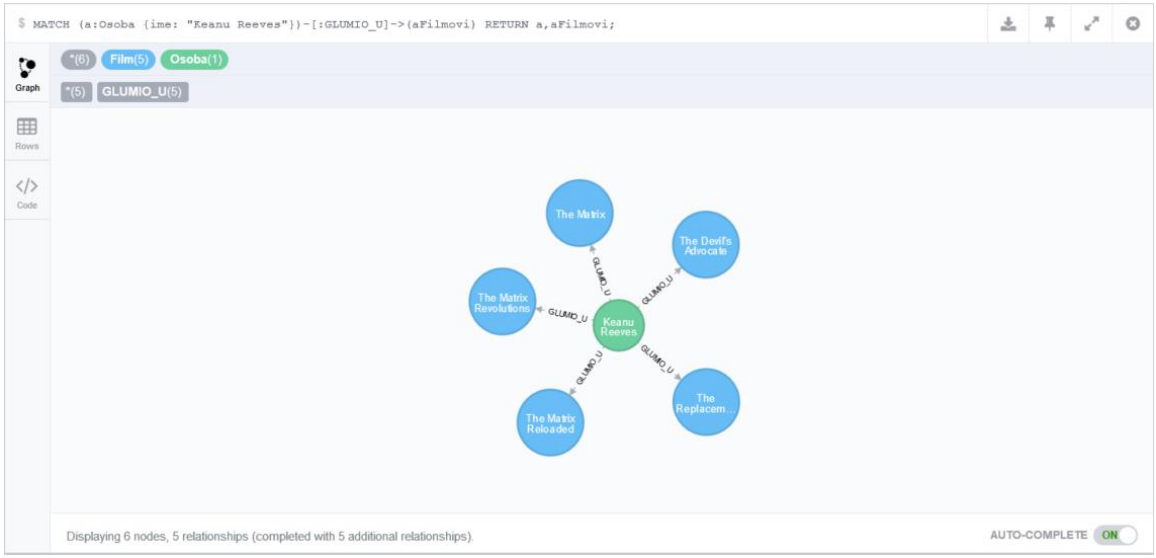
Slika 6.21 Primjer jednostavnog pretraživanja grafa

Naprednije filtriranje rezultata može se postići korištenjem naredbe WHERE koja podržava više uvjeta pretraživanja poput: veći od, manji od i slično. Isti rezultat kao u gornjem primjeru se može postići korištenjem WHERE naredbe na sljedeći način:

```
MATCH (o:Osoba) WHERE o.ime = "Keanu Reeves" RETURN o
```

Neo4j omogućuje i naprednije upite poput dohvata čvora i svih povezanih čvorova.

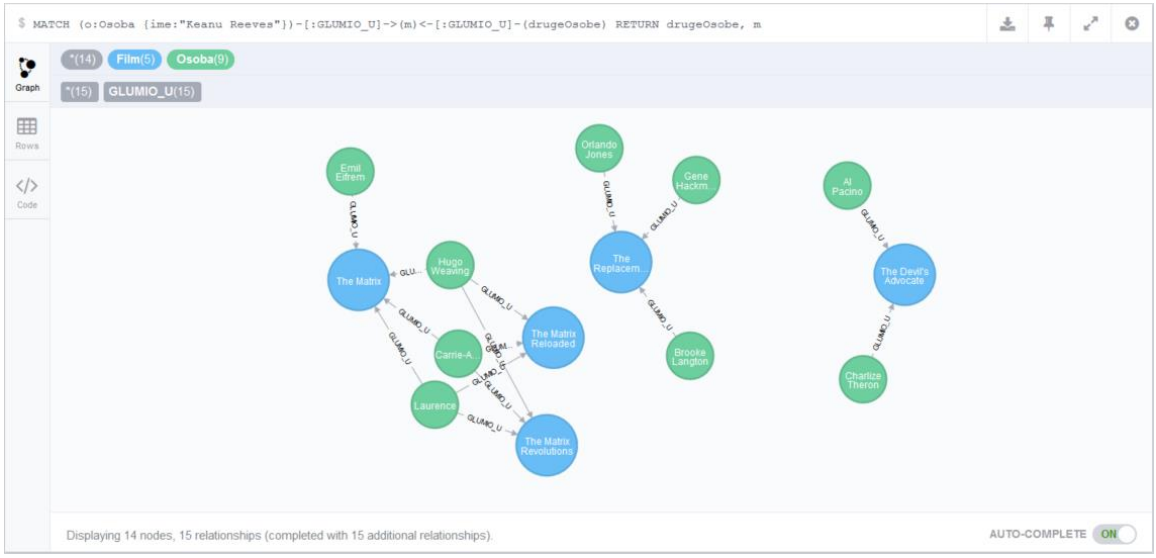
```
MATCH (a:Osoba {ime: "Keanu Reeves"})-[:GLUMIO_U]->(aFilmovi) RETURN a,aFilmovi;
```



Slika 6.22 Primjer dohvata povezanih čvorova zadanog čvora

S obzirom na to da se graf baze temelje na prolasku po grafu, jednostavno je dohvatiti i sve glumce koji su glumili s Keanu Reevesom u ostalim filmovima.

```
MATCH (o:Osoba {ime:"Keanu Reeves"})-[:GLUMIO_U]->(m)<-[:GLUMIO_U]-(drugeOsobe) RETURN
drugeOsobe, m
```

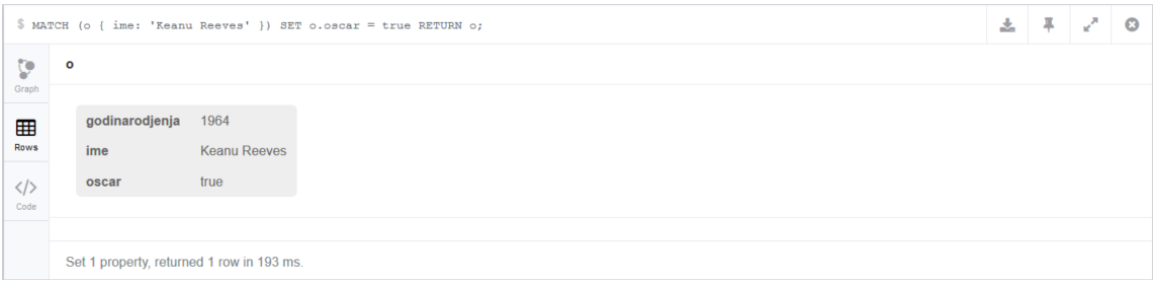


Slika 6.23 Složenija operacija upita nad bazom

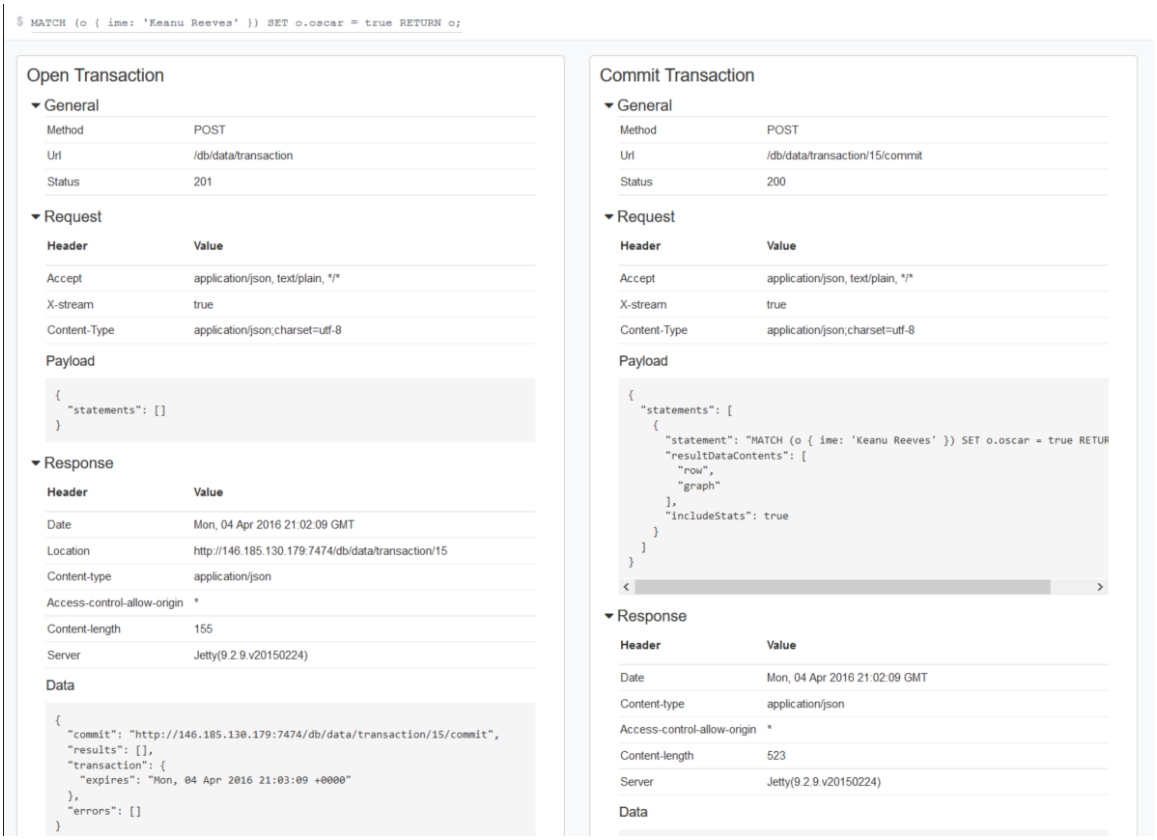
Ažuriranje

Neo4j pohranjuje dodatne atribute za čvorove i veze. Atributi se ažuriraju SET naredbom koja prima nazive atributa koje mijenjamo i nove vrijednosti. SET naredba, osim što može ažurirati postojeće atribute, može određeni atribut i dodati u slučaju da ne postoji. Primjer pokazuje dodavanje atributa „Oscar“ glumcu Keanu Reeves.

```
MATCH (o { ime: 'Keanu Reeves' }) SET o.oscar = true RETURN o;
```



Slika 6.24 Primjer ažuriranja



Slika 6.25 Detaljan prikaz komunikacije s bazom podataka za izvršenje operacije

Brisanje

Brisanje zapisa može biti problematično u graf bazama podataka s obzirom na to da čvorovi mogu imati proizvoljan broj ulaznih i izlaznih veza prema drugim čvorovima. Pri brisanju potrebno je naglasiti da baza podataka pobriše sve veze s čvorom koji se briše.

```
MATCH (o:Osoba { ime: “Carrie-Anne Moss” }) OPTIONAL MATCH (o)-[r1]-() DELETE o,r1;
```

Slika 6.26 Brisanje čvora i njegovih veza

Slika 6.27 Brisanje svih čvorova i njihovih veza

Bibliografija

[1] P. J. Sadalage, M. Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, Addison-Wesley, 2012.

[2] E. Redmond, Jim R. Wilson, Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement, Pragmatic Bookshelf, 2012.

[3] S. Tiwari, Professional NoSQL, John Wiley & Sons Inc, 2011.

[4] G. Vaish, Getting Started with NoSQL, Packt Publishing Ltd, 2013.

[5] D. G. McCreary, A. M. Kelly. Making Sense of NoSQL: A guide for managers and the rest of us, Manning Publications, 2013.

[6] D. Sullivan, NoSQL for Mere Mortals, Addison-Wesley, 2015.

[7] A. Fowler, NoSQL For Dummies, John Wiley & Sons, 2015.

[8] I. Robinson, J. Webber, E. Eifrem, Graph Databases, O'Reilly Media, 2015.

[9] https://www.fer.unizg.hr/predmet/nmbp, Napredni modeli i baze podataka 2015./2016.

Sažetak

Relacijske baze podataka podupiru većinu poslovnih sustava današnjice. Funkcionalnost i pouzdanost ovih baza podataka provjerena je mnogim godinama njihovog korištenja. No ipak organizacije sve više uzimaju u obzir drugačija rješenja za svoje poslovne probleme. Motivacije za to su razne. Od tehničkih – potreba za upravljanjem novim tipovima podataka različitih struktura, potreba za razmještanjem podataka van ograničenja mogućnosti trenutnih sustava. Preko ekonomskih – želja za pronalaskom alternativnih rješenja skupih namjenskih sustava poznatih proizvođača, kako hardverskih tako i softverskih. Sve do praktičnih - nove metodologije razvojnih inženjera baziraju se na agilnosti i brzini razvoja, u svrhu efikasnije prilagodbe trenutnim zahtjevima tržišta.

Skup rješenja, implementacija i tehnologija koja nisu dio relacijskog ekosustava te udovoljavaju aktualnim tehnološkim trendovima današnjice, počela su se okupljati pod zajedničkom kategorijom ne-relacijskih sustava za upravljanjem bazama podataka, popularno zvanih NoSQL baze podataka.

U usporedbi s relacijskim bazama podataka, NoSQL sustavi dijele nekoliko ključnih karakteristika: fleksibilniji model podataka, veća skalabilnost, superiornije performanse. Istovremeno, NoSQL rješenja, kao kompromis za bolje izvođenje, odbacuju temelje koje čine relacijske baze podataka korisnima generacijama aplikacija, kao što su ekspresivan upitni jezik, visoka konzistentnost podataka te sekundarni indeksi.

NoSQL baze podataka možemo podijeliti na četiri tipa s obzirom na to koji model pohranjivanja podataka koristimo:

Ključ-vrijednost baze podataka

Ključ-vrijednost baze podataka jednostavne su i fleksibilne za korištenje. Osnovne odlike su im brzina pristupa podacima te visoka mogućnost skaliranja. Imaju osnovnu arhitekturu asocijativnog polja koja dopušta pohranu raznih tipova podataka za vrijednost indeksa polja, kojeg zovemo ključ. Ključevi moraju biti jedinstveni unutar imenika i koristimo ih za dohvat i spremanje podataka različitih struktura i veličina.

Neke nedostatke, poput nedostatka jezika upita, mogu se ublažiti korištenjem oblikovnih obrazaca ili zasebnih alata.

Dokument baze podataka

Dokumenti u dokument bazama podataka predstavljaju hijerarhijske fleksibilne strukture koje ne zahtijevaju predefiniranu shemu. Organizirani su u logički povezane skupove koje nazivamo kolekcijama. Kolekcije sadrže dokumente koji imaju slične tipove entiteta, iako im se ključevi i vrijednosti međusobno mogu razlikovati. Denormalizacija, indeksi i način pretrage dokumenata igraju ključnu ulogu u dobroj izvedbi ovih baza podataka. Preporučljivo je koristiti oblikovne obrasce pri oblikovanju hijerarhijskih veza, mnogo-na-mnogo veza, kao i jedna-na-mnogo veza. Ponekad se dokumenti umeću jedni u druge, a ponekad se međusobno referenciraju, ovisno o tipu veze.

Stupčane baze podataka

Stupčane baze podataka najviše su skalabilne od svih NoSQL baza podataka. Stoga podržavaju visoku dostupnost podataka, čak i preko više podatkovnih centara. Osmišljene su za rad nad velikim skupinama podataka. Poželjno je pri radu s takvim podacima koristi zasebne i ugrađene alate osmišljene za brz prijenos, analizu i upravljanje Velikim Podacima. Osnovne logičke komponente ovih baza podataka su imenici, familije stupaca, stupci i ključevi redaka. Fleksibilne su po pitanju tipa i strukture podataka koji se pohranjuju te omogućavaju dinamičko mijenjanje stupaca unutar familije stupaca.

Graf baze podataka

Teorija grafova pruža solidnu podlogu metoda i algoritama za oblikovanje podataka po modelu grafa i analizom veza među njima. Grafovi se sastoje od dvije osnovne komponente, čvorova kojima modeliramo entitete te bridova kojima modeliramo veze među entitetima. S obzirom na to koje vrste veza koje prikazujemo, bridovi mogu biti usmjereni ili neusmjereni. Graf model, unatoč svojoj jednostavnosti, može imati cijeli spektar svojstava. Time omogućava široku primjenu modela pri oblikovanju kako apstraktnih tako i stvarnih međusobno povezanih pojava i objekata.

Summary

Relational databases have a long-standing position in most organizations, and for good reason. Their functionality and reliability has clearly been proven over time. But organizations are increasingly considering alternatives to legacy relational infrastructure. In some cases the motivation is technical — such as a need to handle new, multi-structured data types or scale beyond the capacity constraints of existing systems — while in other cases the motivation is driven by the desire to identify viable alternatives to expensive proprietary database software and hardware. A third motivation is agility or speed of development, as companies look to adapt to the market more quickly and embrace agile development methodologies.

The term “NoSQL” started to make an umbrella category for all non-relational databases that satisfy to some extent these principles. When compared to relational databases, many NoSQL systems share several key characteristics including a more flexible data model, higher scalability, and superior performance.

But most of these NoSQL databases also discard the very foundation that has made relational databases so useful for generations of applications - expressive query language, secondary indexes and strong consistency.

We can group NoSQL databases into four distinct models:

Key-Value databases

Key-value databases are simple and flexible but they satisfy the need of fast retrieval services and high scalability options. They are based on the associative array, which is a more generalized data structure than arrays. Keys may be integers, strings, lists of values, or other types. An important constraint on keys is that they must be unique within a namespace. Keys are used to look up values and those values can vary by type. Some of the limitations of key-value databases, such as lack of query language, are mitigated with additional features such as search tools.

Document databases

Documents are flexible data structures that do not require predefined schemas. Documents are organized into related sets called collections. Collections should contain similar types of entities, although the keys and values may differ across documents. Denormalization, query processors and indexes play crucial roles in the overall performance of document databases. It helps to use design patterns when modeling common relations such as one-to-many, many-to-many, and hierarchies. Sometimes embedded documents are called for, whereas in other cases, references to other document identifiers are a better option when modeling these relations.

Column databases

Column family databases are some of the most scalable databases available. They also support high availability, even cross-data center availability. Column family databases are designed for large volumes of data. It helps to use tools designed specifically for moving, processing, and managing Big Data and Big Data systems. They are flexible in regard to the type of data stored and the structure of schemas. They provide developers with the flexibility to change the columns of a column family. The basic, logical components of a column family database are namespaces, column families, columns, and row keys.

Graph databases

Graph theory provides a solid foundation of methods and algorithms for building graph databases and analyzing relations between entities. Graphs are composed of two simple components: vertices that represent entities and edges that represent relationships between entities. Depending on the types of relations, edges may be directed or undirected. This simplicity quickly gives way to a broad range of graph properties and features that are useful for modeling a number of phenomena.