DOI: 10.19279/TVZ.PD.2016-4-1-06

OSVRT NA NOSQL BAZE PODATAKA – ČETIRI OSNOVNE TEHNOLOGIJE

REVIEW OF NOSQL DATABASES – FOUR BASIC TECHNOLOGIES

Aleksandar Stojanović

Tehničko veleučilište u Zagrebu

Sažetak

Iako su relacijske baze podataka i dalje najpopularnija vrsta baza podataka, posljednjih se godina na tržištu pojavilo nekoliko "alternativnih", takozvanih NoSQL baza podataka. To su općenito baze podataka koje nisu bazirane na relacijskom modelu podataka, pa stoga upitni jezik SQL (koji je standardni upitni jezik za relacijske baze podataka) uglavnom nije u takvim sustavima zastupljen. NoSQL sustavi nastali su iz novih zahtjeva za većom fleksibilnošću i boljim performansama u pohrani i obradi velike količine podataka, uglavnom zbog popularnosti interneta i internetskih tehnologija i sve veće količine podataka. U ovom radu analizirane su četiri osnovne vrste NoSQL sustava za rad s bazama podataka: ključ-vrijednost, graf, dokument i stupčani sustavi. Analizirane su specifične karakteristike svakog od tih sustava i opisane su neke primjene za koje su ti sustavi napravljeni. U zaključku su ukratko opisani neki općeniti nedostaci ovakvih sustava, zajedno s pregledom budućeg razvoja NoSQL tehnologije.

Ključne riječi: *NoSQL, graf, dokument, ključ/vrijednost, stupac..*

Abstract

Although relational databases are still the most popular type of database management system, recent years have seen the emergence of several "alternative" databases, the so-called NoSQL database systems. These database systems are not based on relational data model, so the query language SQL mostly is not used with such systems. NoSQL databases emerged from new needs for greater flexibility and better performance in storage and processing of large quantities of data, mostly because of popularity of the Internet and related technologies and ever larger quantities of data.
This paper analyses four basic types of NoSQL database systems: Key/value, graph, document, and columnar. Also, specific characteristics of each of these systems are analyzed and some applications of the systems are described. Conclusion gives a brief description of some drawbacks of such systems in general along with the review of the future development of the NoSQL technology.

Keywords: *databases, NoSQL, graph, document, key/value, column.*

1. Uvod

1. *Introduction*

Relacijske baze podataka i SQL u upotrebi su već gotovo pola stoljeća. One su praktički postale standardni sustavi za rad s podacima. Kada se danas kaže baza podataka, obično se misli na relacijsku bazu podataka, to jest sustav za rad s podacima koji je zasnovan na relacijskom modelu podataka. Popularizacijom Interneta javili su se i novi tehnološki izazovi koji obuhvaćaju razna područja, kao što su operacijski sustavi, programski jezici, arhitektura softverskih sustava, ali i baza podataka. Jedan od najvećih izazova je količina podataka koji se očituje u poteškoćama sustava (baze podataka) da prihvati i obradi veliku količinu podataka u što kraćem vremenu. Iz tih izazova nastale su nove vrste baza podataka kod kojih se podaci ne organiziraju prema principima relacijskog modela nego, prema puno jednostavnijem i "slobodnijem" modelu podataka, zavisno od vrste takve jedne baze podataka. Jedna od najuočljivijih karakteristika većine ovih baza

podataka je to da one ne zahtijevaju definiciju sheme podataka, kao što je to slučaj s relacijskim bazama podataka. NoSQL baze podataka su zbog toga dinamičke, u smislu da je tip i broj atributa nekog entiteta "otvoren", to jest može ga se po potrebi mijenjati bez da to utječe na druge podatke. Prednost ovog pristupa je u tome što je jednostavnije preslikati strukturu podataka u nekom programskom jeziku na sličnu sturkturu podataka neke takve baze podataka, čime se pojednostavljuje interakcija između aplikacije i baze podataka. Još jedna važna karakteristika NoSQL baza podataka je to da su one napravljene za distribuiran način upotrebe – dok su relacijske baze podataka napravljene za rad na jednom računalu, NoSQL sustavi napravljeni su tako da se više njih raspoređuje na veći broj računala gdje svaki radi sa svojom skupinom podataka. Time se značajno ubrzava prihvat i obrada podataka.
Iako se trenutno puno piše o NoSQL tehnologijama i neki čak tvrde da će one zamijeniti relacijske baze podataka, autor ovog članka polazi od pretpostavke da će relacijske baze podataka još puno decenija biti zastupljene i da će NoSQL tehnologije jednostavno popuniti dio tržišta koji se susreće s prethodno opisanim problemima i za koje će NoSQL baze podataka biti samo jedan dodatak u kolekciji postojećih tehnologija koje su kod takvih korisnika zastupljene. To je slično kao i kod programskih jezika, gdje ne postoji jedan jezik koji je najbolji za sve, pa mnoge organizacije upotrebljavaju dva ili više programskih jezika. S obzirom na to da se relacijske baze podataka odlično uklapaju u većinu potreba mnogih organizacija, NoSQL će najvjerojatnije biti riješenje za one vrste problema za koje relacijske baze podataka nisu niti napravljene.
Iako postoji nekoliko vrsta NoSQL baza podataka, u ovom radu opisane su četiri takve tehnologije: ključ/vrijednost, graf, dokument i stupčane baze podataka. Ključ-vrijednost sustavi spadaju u one jednostavnije. Takvi sustavi podatke smještaju u obliku parova, gdje je prvi član ključ, a drugi vrijednost koja je pridružena tom ključu. Ovakve baze podataka imaju odlične performanse, ali nisu praktične za kompleksne modele podataka. Graf-sustavi bazirani su na modelu grafa i manje se koriste, ali su izuzetno korisne u radu s podacima koji su jako povezani. Ovakve se baze podataka sastoje od čvorova i veza među njima, te omogućuju efikasno prolaženje kroz čvorove slijedeći njihove veze. Dokument-sustavi rade s podacima koji se sastoje od jedinstvenog identifikatora i vrijednosti, što se skupa naziva dokumentom. Vrijednost, međutim, može sadržavati i drugi dokument, tako da ovakvi sustavi podržavaju ugnježđene vrijednosti, što omogućava veliku fleksibilnost strukturiranja podataka. Sustavi bazirani na stupcima (stupčane baze podataka) podatke koji se nalaze u istom stupcu (ako ih promatramo kroz dvodimenzionalnu tablicu) fizički smještaju zajedno, što omogućava dobre performanse za određene vrste pretraživanja.

2. Ključ/vrijednost sustavi

2. *Key/value systems*

Ključ/vrijednost baze podataka zasnovane su na jednostavnom modelu podataka koji se sastoji od parova oblika (*ključ, vrijednost*), gdje je jednom tekstualnom ključu pridružena jedna vrijednost bilo kojeg tipa. Takav je model podataka po strukturi sličan riječniku. Ovakve baze podataka nemaju upitni jezik (kao što je to SQL u relacijskim bazama podataka) - one samo omogućavaju pronalaženje, dodavanje i uklanjanje parova na osnovu ključa. Općenito, one podržavaju tri operacije [5]:

- GET(ključ) – Vraća vrijednost pridruženu zadanom ključu
- PUT(ključ, vrijednost) – Dodaje novi par (*ključ, vrijednost*) ili pridružuje novu vrijednost postojećem ključu
- DELETE(ključ) – Uklanja par (*ključ, vrijednost*); ako ključ ne postoji može dojaviti grešku

Ključ može biti u raznim formatima, kao što je specifikacija direktorija za neku datoteku, niz znakova koji predstavlja nekakav jednistven kôd, poziv REST mrežnog servisa, internetska adresa i sl. Vrijednosti pridružene ključu mogu također biti raznolike, kao što su slike, zvuk, internetske stranice, dokumenti i sl.
Pored gore navedenih operacija, za ovakve baze podataka općenito vrijede dva pravila:

- *Različiti ključevi* – Svaki ključ u tablici mora biti jedinstven.
- *Nema upita na osnovu vrijednosti* – Upit može biti baziran samo na ključu, a ne i na vrijednosti. Drugim riječima, u ovakvoj bazi podataka ne mogu se postavljati upiti kojima se traži određena vrijednost, nego samo upit kojim se traži određeni ključ.

2.1 Prednosti korištenja "key-value" baza podataka

2.1 *Advantages of using* "*key/value*" *databases*

Osnovna tehnička prednost ovih baza podataka je jednostavnost koja se očituje u performansama i skalabilnosti. Baze podataka s jednostavnim operacijama, kao što su ključ/vrijednost baze podataka, mogu imati vrlo veliku skalabilnost – s obzirom da su operacije jednostavne podaci se mogu modelirati tako da se maksimalno iskoristi efikasnost ovakvog sustava. Na primjer, moguće je promatrati koliko vremena treba za izvršavanje operacija PUT ili GET na 100.000 elemenata i podesiti sustav i model podataka za maksimalnu efikasnost. To je znatno teže napraviti sa standardnim SQL bazama podataka jer tamo upiti mogu biti puno kompleksniji, uzimajući u obzir da mnogi SQL upiti sadrže "join"-operacije koje se povezuju s drugim tablicama. Nadalje, SQL-upiti mogu sadržavati druge, ugnježđene SQL-upite, što dodatno komplicira optimizaciju.

2.2 Dva primjera ključ/vrijednost baza podataka

2.2 *Two examples of key/value databases*

Postoji nekoliko industrijskih baza podataka baziranih na ovom konceptu. U ovom dijelu ukratko su opisana dva takva sustava otvorenog koda: Riak i Redis [9].

2.2.1 Riak

2.2.1 *Riak*

Riak je distribuirana ključ/vrijednost baza podataka gdje vrijednosti mogu biti bilo što – XML, JSON, slike, zvuk i sl. S obzirom da je Riak napravljen da radi u internetskom okruženju, komande se šalju putem HTTP protokola. Prednosti ove baze podataka su u podršci za sustave gdje je neophodna stalna raspoloživost, kao što su kupovine preko Interneta (na primjer, Amazon). Riak također podržava nekoliko programskih jezika, a s obzirom da je napisan u Erlangu može ga se u tom jeziku proširivati po potrebi. Za potrebe kompleksnijih struktura podataka ili ako distribuiranost sustava nije neophodna Riak vjerojatno nije najbolji izbor.

2.2.2 Redis

2.2.2 *Redis*

Redis je vrlo jednostavna ključ/vrijednost baza podataka koja cijeli skup podataka drži u memoriji i povremeno ga asinkrono upisuje na disk. Redis se može konfigurirati za spremanje podataka nakon određenog broja sekundi. Na primjer, može se specificirati da sprema podatke nakon 100 promjena i najmanje 5 sekundi. S obzirom da se podaci spremaju asinkrono, ako se sustav sruši posljednjih nekoliko promjena može biti izgubljeno. Zbog toga Redis podržava *master /slave* replikaciju od prve verzije. S obzirom da Redis drži podatke u memoriji, to je jedna od najbržih baza podataka ovog tipa i pogodan je za aplikacije gdje povremeni manji gubitak podataka nije veliki problem.

3. Graf-baze podataka

3. *Graph databases*

Graf-baze podataka korisne su u aplikacijama u kojima su važni odnosi među podacima. Za razliku od relacijskog modela podataka u takvim sustavima odnosi među podacima specificirani su eksplicitno. Iako relacijske baze podataka mogu prikazati odnose među podacima, ti su odnosi implicitni (realizirani pomoću ključa) i nisu fleksibilni. To se može vidjeti u tablicama 1, 2, 3 i 4.
Ovo je tipičan primjer relacijskog modela podataka u kojem su definirana četiri entiteta: korisnik, narudžba, artikl i proizvod. Odnose među ovim entitetima moguće je dobiti upotrebom operacije pridruživanja (*join*). Na primjer, da bi se dobila informacija o tome što je naručio neki kupac treba povezati ove četiri

Tablica 1 *Relacijska tablica korisnik.*
Table 1 *Relational table korisinik.*

ID	Korisnik	Adresa	Telefon	Email
1	Pero	Savska 11	012 333 4444	pero@email.com
2	Ivo	Ilica 87	987 654 3211	ivo@email.com
...	...	...	...	...
55	Marko	Selska 25	663 012 9876	marko@email.com

Tablica 2 *Relacijska tablica narudžba.*
Table 2 *Relational table narudžba.*

ID	Korisnik
773	1
162	1
...	...
319	55

Tablica 3 *Relacijska tablica artikl.*
Table 3 *Relational table artikl.*

Narudžba	Proizvod	Količina
773	1882	5
773	3091	2
...	...	...
162	9091	2

Tablica 4 *Relacijska tablica proizvod.*
Table 4 *Relational table proizvod.*

ID	Opis	Cijena
1882	...	1200
3091	...	180
9091	...	715

tablice, što relacijski sustavi mogu izvršiti prilično efikasno. Međutim, ako treba dobiti informaciju o tome koji je kupac naručio proizvod 3091 relacijska baza podataka to također može naći, ali ovaj put takva operacija je puno manje efikasna. Nadalje, ako se ovakav upit proširi na nešto kao "koji kupci su kupili proizvod X koji su također kupili proizvod Y", tada bi kod veće količine podataka došlo do ozbiljnih problema s performansama. Problem je u tome što se u relacijskom modelu podataka odnosi među podacima otkrivaju pretraživanjem zato jer su veze implicitne – one su realizirane pomoću stranih ključeva.

Još jedan tipičan primjer nedostataka relacijskog modela je društvena mreža. U sljedećem primjeru pokazani su podaci i osobi i njenim prijateljima (tablice 5 i 6).

Tablica 5 *Relacijska tablica osoba.*
Table 5 *Relational table osoba.*

ID	Osoba
17	Ivo
25	Pero
...	...
41	Marija

Tablica 6 *Relacijska tablica OsobaPrijatelj.*
Table 6 *Relational table OsobaPrijatelj.*

IDOsoba	IDPrijatelj
17	25
25	17
25	41
...	...
41	17

Ako nas zanima tko je Perin prijatelj relacijski sustav može lako doći do odgovora prateći identifikator IDPrijatelj za osobu 25 (Perin identifikator). Odgovor su osobe s identifikatorom 17 i 41, to jest Ivo i Marija. Ako, međutim, pokušamo postaviti upit koji ide malo dublje u ovakvu "mrežu", kao na primjer, tko je prijatelj od prijatelja od prijatelja od Pere, onda bi to zahtijevalo veći broj pretraživanja iste tablice (rekurzivno), pa bi zbog toga i performanse bile slabije. Kao što je prethodno rečeno, problem je u tome što odnosi među podacima nisu direktni, nego ih sustav treba pronaći. U knjizi *Neo4j in Action* [8] autori Partner i Vukotic prikazali su rezultate eksperimenta u kojem su usporedili relacijsku bazu s graf-bazom Neo4j u traženju podataka tipa prijatelj-od-prijatelja do dubine 5 sa skupom od 1 000 000 osoba. Ti su rezultati prikazani u tablici 7.
Vidi se da svaki dodatni nivo zahtijeva od relacijske baze podataka znatno više vremena, dok se ono samo neznatno povećava kod graf-baze podataka.

Tablica 7 *Usporedba performansi relacijske baze podataka i Neo4j na pretraživanju povezanih podataka.*

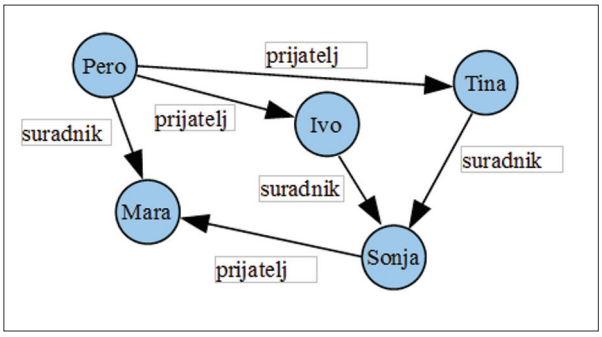
Table 7 *Performance comparison of relational database with Neo4j in speed of search through related data.*

Dubina	Relacijska baza podataka	Neo4j	Broj vraćenih slogova
2	0.016s	0.01s	~2500
3	30.267s	0.168s	~110.000
4	1543.505s	1.359s	~600.000
5	Nezavršeno	2.132s	~800.000

3.1 Graf kao model podataka

3.1 *Graph as a data model*

Graf-baze podataka umjesto relacijskog koriste koncept grafa kao model podataka. Graf je struktura podataka koja se sastoji od čvorova i lukova koji povezuju te čvorove. Čvorovi u takvim sustavima predstavljaju podatke, a lukovi odnose među njima. Graf-baze podataka pored čvorova i lukova sadrže i *svojstva* koja su skup ključ/vrijednost parova i koja sadrže podatke smještene u dotični čvor.
Na primjer, na Slici 1 prikazan je graf koji predstavlja grupu ljudi među kojima postoje odnosi *prijatelj* i *suradnik*.



Slika 1 *Graf koji predstavlja grupu ljudi među kojima postoje odnosi prijatelj i suradnik.*

Figure 1 *A graph representing a group of people in friend and coworker relationship*

Baze podataka bazirane na ovakvom modelu eksplicitno čuvaju veze među čvorovima. Izvršavanje upita kod ovakvih sustava sastoji se od praćenja *putanja*, odnosno veza među čvorovima. Na primjer, ako nas zanima tko je prijatelj Perinog prijatelja, možemo jednostavno pratiti vezu od Pere do Tine, pa od Tine do Sonje, pa od Sonje do Mare. Graf-baze podataka su za ovakvo prolaženje kroz graf vrlo efikasne jer rade s direktnim vezama među čvorovima, to jest oni te veze ne moraju tražiti.

3.2 Upiti

3.2 *Queries*

Za graf-baze podataka ne postoji standardni upitni jezik, kao što je to slučaj sa SQLom kod relacijskih baza podataka. Jedan primjer takvih upitnih jezika je Cypher [3] koji se koristi za graf-bazu podataka Neo4j. Taj je upitni jezik baziran na konceptu podudaranja uzoraka (eng. *pattern matching*). Ti se uzorci upotrebljavaju za pronalaženje struktura unutar grafa, nakon čega se te strukture mogu dalje analizirati. Na primjer, sljedeći izraz podudarat će se s dijelom grafa u kojem `Osoba ŽIVI_U državi`:

```
(:Osoba)-[:ZIVI_U]->(:Grad)-[:NALAZI_SE]-
>(:Drzava)
```

Gornji izraz je u stvari uzorak (eng. *pattern*) koji traži poklapanje s tri čvora, tako da je tip veze između prva dva `ZIVI_U`, a između posljednja dva `NALAZI_SE`. Dio u okruglim zagradama predstavlja čvor. Naziv s dvotočkom ispred označava tip čvora, što omogućava efikasnije pretraživanje jer se sustav može fokusirati samo na određene tipove čvorova. Dio u uglatim zagradama označava odnos i nalazi se između „`-`„ i „`->`" (za usmjerene odnose) ili „`-`„ i „`-`„ (za neusmjerene odnose). U gornjem primjeru, `ZIVI_U i NALAZI_SE` su tipovi odnosa, slično tipovima kod čvorova.
Da bi se dodao novi čvor u graf upotrebljava se naredba `CREATE`. U sljedećem primjeru dodaje se čvor tipa Knjiga, s naslovom i godinom izdanja:

```
CREATE (:Knjiga { naziv: „Diskretna
matematika", objavljeno: 2005 })
```

U gornjem primjeru, *naziv* i *objavljeno* su svojstva koja predstavljaju ključ i vrijednost. Upiti u Cypheru obično se sastoje od više *klauzula*, slično kao u SQLu. Na primjer, sljedeći upit vraća čvor s nazivom "Poruka" i podatkom u tom čvoru koji za ključ "poslao" ima pridruženu vrijednost "Marko":

```
match (x:Poruka {poslao: 'Marko'})
return x
```

Ovdje se vidi da Neo4j baza podataka sprema podatke u obliku ključa i vrijednosti. U stvari, struktura podataka koju možemo spremiti u čvor je slična onoj kod dokument-baza podataka.

4. Dokument-baze podataka

4. *Document databases*

Dokument-baze podataka trenutno su najpopularnija vrsta NoSQL baza podataka. Jedan od razloga je taj da je njihov model podataka intuitivan. Osnovni element ovog modela podataka je *dokument*: uređeni skup ključeva s pridruženim vrijednostima. Dokumenti se smještaju u *kolekcije*, što bi bilo nešto slično tablicama kod relacijskih baza podataka, ako jedan dokument odgovara redku tablice. Za razliku od relacijskih baza podataka, dokument-sustavi imaju dinamičke sheme, što znači da svaki dokument u nekoj kolekciji može imati drugačiju strukturu. U sljedećem primjeru dokumenta upotrebljava se JavaScript notacija (kao u sustavu MongoDB):

```
{ime: "Pero", prezime: "Perić"}
```

U gornjem primjeru, *ime* i *prezime* su ključevi kojima su pridružene vrijednosti *Pero* i *Perić*. Dokumenti, u stvari, mogu imati hijerarhijsku strukturu zato jer jedan dokument može biti dio drugog, što korisnicima omogućava fleksibilnost i nudi intuitivan model podataka, posebno ako dolaze iz objektno-orijentiranog razvoja.
U sljedećem primjeru pokazan je dokument koji ima kompleksniju strukturu.

```
{
      autor: "Pero Perić",
      knjige: [{ naziv: "Knjiga 1",
                  godina: 2012 },
               { naziv: "Knjiga 2",
                  godina: 2013 }],
      ostalo: { izdavači: ["Addison-
                  Wesley", "Prentice
                  Hall", "Wiley"],
      područja:  ["Računarstvo",
                  "Matematika",
                  "Statistika"]
            }
}
```

Mnogi dokument-sustavi spremaju podatke u JSON formatu. Jednostavnost ovog formata omogućava prikazivanje bilo kakvih podataka i jednostavnu konverziju objekata u raznim programskim jezicima u JSON, bez upotrebe ORMa (object-relational mapper).
Da bi korisnik znao što predstavljaju podaci u ovakvom sustavu, upotrebljavaju se prethodno spomenute kolekcije. Na primjer, jedna kolekcija može sadržavati knjige, druga popis studenata, treća predmete, itd. Aplikacija vodi evidenciju o tome koji podatak ide u koju kolekciju.
Dokument-sustavi, kao i mnogi drugi NoSQL sustavi, nemaju shemu: Svaki dokument može imati svoju strukturu. Na primjer, jedan dokument o autoru ne mora sadržavati iste vrijednosti kao drugi. Ta se fleksibilnost može iskoristiti za poboljšavanje performansi. Jedan primjer bi bio taj da ne moramo spremati `NULL` vrijednosti. Na primjer,

```
{
      ime: "Pero",
      prezime: "Perić",
      kućni_tel: 9998887777,
      mobitel: 1112223333
}
{
      ime: "Ivo",
      prezime: "Ivić",
      mobitel: 5556667777
}
```

U ovom primjeru, drugi dokument nema kućni broj telefona, pa ga niti ne moramo smjestiti u dokument.
Većina dokument-sustava napravljena je s ciljem skalabilnosti u odnosu na količinu podataka. Model podataka zasnovan na dokumentima omogućava to da se podaci mogu rasporediti na više servera. Većina ovakvih sustava automatski vodi računa o raspodjeli podataka i povezivanju s drugim serverima. Nadalje, oni podržavaju i niz drugih mogućnosti:

- Indeksiranje – za bržu obradu upita ovakvi sustavi podržavaju sekundarne indekse, odnosno indekse bazirane na podacima koji nisu glavni ključ dokumenta
- Replikacija – čuvanje kopija podataka na više servera

- Particioniranje – raspodjela podataka na više servera radi veće efikasnosti u radu s velikom količinom podataka
- Agregacija – za kombiniranje i transformaciju dokumenata (pomoću filtriranja, grupiranja, sortiranja, …)
- Specijalni tipovi kolekcija – omogućavaju spremanje dokumenata kojima istekne vrijeme
- Spremište za datoteke – omogućava spremanje velikih datoteka

Za razliku od relacijskih baza podataka, dokument sustavi ne podržavaju operacije pridruživanja (join); razlog je omogućavanje veće skalabilnosti za distribuirane sustave.

4.1 Upiti

4.1 *Queries*

Dokument-sustavi obično podržavaju API za postavljanje upita i modifikaciju podataka. Na primjer, MongoDB sustav [1, 2] upotrebljava JavaScript kao jezik za rad s bazom podataka. Kao jedan primjer, da bi dobili sve korisnike koji se prezivaju Perić možemo pisati ovako:

```
db.korisnici.find({ prezime: "Perić",
adresa: "Ilica 112" })
```

U gornjem primjeru u kolekciji *korisnici* traži se dokument u kojem je polju *prezime* pridružena vrijednost *Perić*, a polju *adresa* vrijednost *Ilica 112*. Ako je potrebno dobiti sve korisnike koji su stariji od 30 godina mogu se definirati uvjeti:

```
db.korisnici.find({starost: {$gt: 30}})
```

U gornjem primjeru, `{$gt: 30}` je kriterij koji kaže da starost mora biti veća od 30; *gt* je veće-od operator.
Dodavanje novih dokumenata je slično:

```
db.korisnici.insert({ ime: "Pero",
prezime: "Perić" })
```

Gornji dokument će u ovom slučaju automatski dobiti ključ `_id`, ako on nije eksplicitno specificiran.
Da bi se promijenilo neko polje u dokumentu upotrebljava se `$set` operator. U sljedećem primjeru mijenja se vrijednost polja *starost* na 30 dokumenta koji sadrži vrijednost *Perić* za polje *prezime*:

```
db.korisnici.update({prezime: „Perić"}
{$set {starost: 30}})
```

Na sličan način može se obrisati neki dokument. U sljedećem primjeru briše se dokument koji ima vrijednost Perić za polje prezime:

```
db.korisnici.remove({ prezime: "Perić" })
```

Ako je potrebno obrisati sve dokumente iz kolekcije, jednostavno se piše

```
db.korisnici.remove()
```

API koji ovakvi sustavi podržavaju dovoljno je fleksibilan da se mogu specificirati bilo kakve vrste upita.

5. Stupčane baze podataka

5. *Columnar databases*

Kao što i sam naziv kaže, kod stupčanih sustava naglasak je na stupcima, koji se mogu promatrati slično kao stupci tablice. Za razliku od relacijskih baza podataka, stupčani sustavi mogu pristupati pojedinačnim stupcima neovisno od drugih stupaca, što je jedna od osnovnih prednosti ovakvih sustava sa stajališta performansi [6]. Nadalje, vrijednosti koje pripadaju jednom stupcu na disku su smještene na jednom cjelovitom prostoru, tako da je učitavanje jednog stupca u memoriju efikasno. Po potrebi, iz tih stupaca mogu se sastaviti redovi, kao kod "klasičnih" tablica. Ovakvi sustavi su efikasni kada se upiti odnose na samo jedan manji broj stupaca zato jer ne moraju učitavati cijelu tablicu. Na primjer, neka tablica zauzima 25 GB prostora, s tim da je svaki stupac sljedeće veličine [4]:

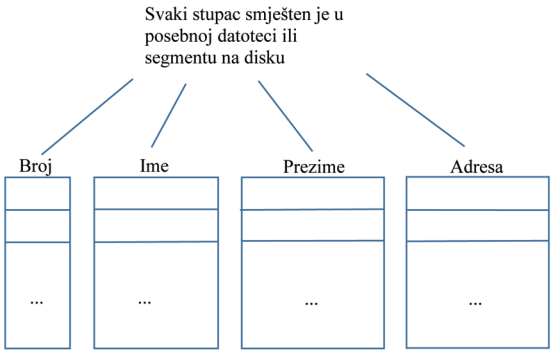
- stupac A: 8 GB
- stupac B: 5 GB
- stupac C: 11 GB
- stupac D: 1 GB

Ako upit koristi samo stupac D, tada je dovoljan samo 1 GB prostora za taj upit, umjesto 25 GB za cijelu tablicu. Za sustav koji radi na razini redova, bilo bi potrebno svih 25 GB prostora.
Stupčani sustavi u najjednostavnijem obliku smještaju podatke za svaki stupac u posebnoj datoteci. Vrijednosti za svaki red tog stupca smještene su jedna do druge. S obzirom da I/O operacija učitavanja može učitati cijeli blok podataka odjedanput, učitavanje stupaca je

efikasno, posebno ako su potrebni svi podaci iz nekog stupca. Nadalje, s obzirom da su svi podaci u stupcu istoga tipa, oni se mogu komprimirati, pa je u tom slučaju efikasnost upita još veća jer se reduciraju I/O operacije (koje su inače spore), iako se tada CPU mora nešto više opteretiti (zbog dekomprimiranja).
Zbog svojih tehničkih karakteristika, osnovna prednost ovakvih NoSQL sustava je ta da su pogodni za obradu vrlo velikih količina podataka. Općenito, kod ovakvih sustava učitavanje i analiziranje svih vrijednosti jednog stupca je prilično efikasno (što je također slučaj i s dodavanjem ili brisanjem stupaca). Zbog toga su ovi sustavi idealni za batch-obradu. S druge strane, učitavanje svih stupaca koji bi tvorili jedan red, ili dodavanje ili modifikacija reda, nije efikasno, pa se takve operacije kod ovakvih sustava (ako je moguće) izbjegavaju (one su u stvari manje efikasne nego kod relacijskih baza podataka).

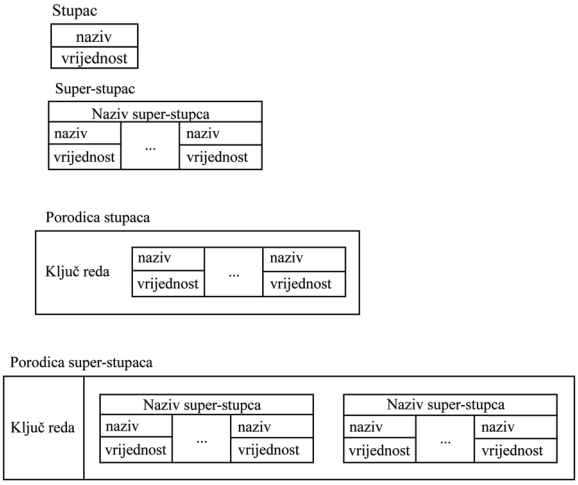
5.1 Model podataka

Kod tradicionalnih relacijskih baza podataka, tablice se sastoje od redova i stupaca, a svaki redak ima istu strukturu (što zavisi od broja i tipa stupaca). Takvi sustavi su projektirani i optimizirani za rad s redovima. U takvim sustavima redovi su obično smješteni jedan iza drugog da bi se optimiziralo njihovo učitavanje u memoriju. Kod stupčanih sustava vrijednosti su grupirane po stupcima, a ne redovima (slika 2).
Iako su ove dvije vrste baza podataka organizirane na drugačiji način, stupčane baze podataka često podržavaju relacijski model zato jer je model podataka dovoljno sličan – iz stupaca uvijek možemo dobiti redove, iako takvi sustavi nisu za to optimizirani.
Kod nekih hibridnih baza podataka (između dokument i stupčanih), kao što je Cassandra [11], također se upotrebljava ključ za pristup podacima, ali se taj ključ sastoji od reda i stupca u kojem se nalazi podatak. Model podataka kod ovakvih sustava temelji se na djelomično popunjenoj tablici. Četiri osnovne komponente ovog modela podataka prikazane su na slici 3.



Slika 2 *Organizacija stupaca kod stupčanih baza podataka.*

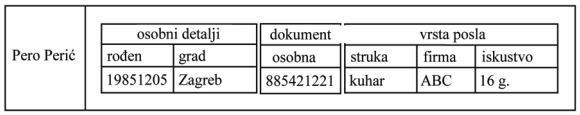
Figure 2 *Organization of columns in a columnar database.*



Slika 3 *Komponente modela dijelomično popunjene tablice.*

Figure 3 *Components of a model of a partially populated table.*

Najjednostavnija struktura podataka ovog modela je stupac, koji se sastoji od naziva i vrijednosti.
Ti se stupci mogu kombinirati u veće strukture, *super-stupce*, koji se sastoje od sortiranog niza stupaca. Stupci se nalaze u redcima, a redak koji se sastoji samo od stupaca naziva se *porodicom stupaca*. Kada se u retku nalaze super-stupci onda se to naziva *porodicom super-stupaca*.
Ovaj model podataka je u stvari riječnik koji se sastoji od drugih riječnika. Na slici 4 prikazan je primjer podataka o osobi.
Ovdje su pokazana dva super-stupca, *osobni detalji* i *vrsta posla*, i jedan stupac, *dokument*. Općenito, porodica stupaca koristi se za podatke koji su na neki način povezani, kao što



Slika 4 *Primjer podataka o osobi.*

Figure 4 *Example of person data.*

su osobni detalji i vrsta posla za neku osobu. Iako je ovaj model podataka sličan dokument-sustavima, razlika je u načinu pristupa podacima: Kod dokument-sustava upotrebljava se ključ kojim se dolazi do dokumenta, dok je kod stupčanih sustava ključ kompleksniji, odnosno sastoji se od retka i stupca. Drugim riječima, struktura podataka kod dokument-sustava je hijerarhijska i ne postoje pojmovi retka i stupca.

6. Zaključak

6. *Conclusion*

Iako NoSQL tehnologije imaju prednosti sa stajališta performansi, to je još mlada i nezrela tehnologija. Jedan od općenitih problema ovih tehnologija je taj da one nemaju čvrstu teoretsku osnovu. Za razliku od relacijskog modela podataka koji je zasnovan na teoriji relacijske algebre i relacijskog računa iz kojeg je nastao i SQL, standradni upitni jezik relacijskih baza podataka, te mnogi drugi principi (kao što je normalizacija), za NoSQL sustave još nisu utvrđeni temeljni principi korištenja koji bi mogli poslužiti kao osnova za razvoj njihove teoretske pozadine. Kod ovih sustava ne postoji čvrsti princip modeliranja podataka, postavljanja upita, osiguravanja i analiziranja podataka – svaki sustav ima svoj način strukturiranja podataka, svoj upitni jezik, svoj način osiguravanja podataka, svoje preporuke i alate za analizu [7, 10]. Međutim, bez obzira na ove nedostatke, NoSQL će prema analizama mnogih stručnjaka ući u široku primjenu, ali uglavnom kao dio jednog šireg ekosustava baza podataka.

7. Reference

7. *References*

[1] K. Banker, MongoDB in Action, Manning, 2012.

[2] K. Chodorow, MongoDB – The Definitive Guide, O'Reilly, 2013.

[3] I. Robinson, J. Webber, E. Eifrem, Graph Databases, O'Reilly, 2013.

[4] J. Steemann, Column-oriented databases, triAGENS, http://www.slideshare.net/arangodb/introduction-to-column-oriented-databases

[5] D. McCreary, A. Kelly, Making Sense of NoSQL, Manning, 2014.

[6] D.J. Abadi, P.A. Boncz, S. Harizopoulos, "Column-oriented database systems", VLDB Endowment, 2009.

[7] S. Tiwari, Professional NoSQL, Wiley, 2011.

[8] A. Vukotic, N. Watt, Neo4j in Action, Manning, 2015.

[9] E. Redmond, J.R. Wilson, Seven Databases in Seven Weeks, The Pragmatic Programmers, 2012.

[10] J. Vaughan, IT pros talk top enterprise NoSQL architecture challenges, http://searchdatamanagement.techtarget.com/feature/IT-pros-talk-top-enterprise-NoSQL-architecture-challenges

[11] E. Hewitt, Cassandra – The definitive Guide, O'Reilly, 2011.

AUTORI · *AUTHORS*

Aleksandar Stojanović

Aleksandar Stojanović je asistent na Tehničkom veleučilištu u Zagrebu i održava nastavu na predmetima iz područja programskog inženjerstva, objektno-orijentiranog programiranja i naprednog programiranja. Godine 1996. diplomirao je informacijske i komunikacijske znanosti na Filozofskom fakultetu sveučilišta u Zagrebu, a 1999. godine magistrirao je računarstvo u SAD-u na sveučilištu Midwestern State University, te je radio kao programski inženjer na programskim sustavima iz područja telekomunikacija, financija i energetike. Njegova područja interesa uključuju programske jezike, prevodioce i principe programiranja. Autor je knjige “Elementi računalnih programa s primjerima u Pythonu i Scali”.