Diplomski rad

Tema: Poređenje relacionih i nerelacionih baza podataka

1. Uvod

Podatak – neobrađena, „sirova“ činjenica, pretvorena u formalizovan oblik (broj, reč, slika, itd.).

Čovekov svakodnevni život je realizovan na osnovu raznih podataka. Bilo to njegovi lični podaci, podaci o njegovoj okolnosti, poslovnom ili privatnom svetu pa čak i nepotrebni podaci, naš redovan tok razmišljanja i događaja je vrlo često vezan, donekle i zavisan, od podataka. Razvojom tehnologije kroz vreme, čovek više nije jedini koji zavisi od podataka, već su tu zavisnost i zadobile mašine. Kompjuteri, internet serveri, bilo koji elektronski uređaji organizuju , izvršavaju i koriguju svoj rad na osnovu određenih podataka.

„Data is the new oil“ - Clive Humbly, 2006

Porastom zavisnosti od podataka, velika pažnja je posvećena na razvijanje softverskih sistema čija je glavna funkcija rad sa podacima. To uključuje: skladištenje, pretragu, filtriranje, obradu podataka i razne druge mogućnosti koje zavise od sistema i njegovih korisnika. Baza podataka predstavlja skladišten skup podataka, a pomenuti softverski sistemi se nazivaju sistemi za upravljanje bazama podataka.

Prve softverske realizacije baza podataka su se pojavile 1960-tih godina, kada je Charles Benchman dizajnirao prvi oblik softverske baze podataka, nazvan *Integrated Data Store*. Nakon toga, 1970-tih, kompanije IBM je razvila *Information Management System*. Ova dva primera baza podataka su bile navigacione, jer je pronalazak podataka zahtevao manuelnu navigaciju kroz podatke da bi se pronašao potreban podatak, one takođe nisu imale sposobnost da sobstveno pamte odnose između podataka, takve baze podataka se mogu nazivati nerelacione. U kasnijim 1970-tim godinama, došlo je do prve pojave, takozvane, relacione baze podataka. Relaciona baza podataka omogućuje čuvanje veza između podataka, što uvodi novu dimenziju pri skladištenju podataka. Podaci sada nisu više vezani samo za pojedinačne vrednosti ili elemente, već je veliki značaj podatka postao njegova povezanost sa drugim podacima. Prve relacione baze podataka su bile *INGRES*, čija je ideja osmišljena i realizovana na Berkley univerzitetu u Kaliforniji, i System R od kompanije IBM. System R je bila prva baza podataka koja je koristila poznati programski jezik Structured Query Language (SQL) koji je nastavio da bude korišćen u izradi budućih relacionih baza podataka. Do sadašnjeg vremena, tehnologija je znatno napredovala i to je dovelo do povećanja sposobnosti računara. Baze podataka, relacione i nerelacione, su postale više optimizovane i pojavile su se razne verzije baza podataka i sistema za upravljanje bazama podataka koji su bile dizajnirane sa posebnim namenama u vidu.

U nastavku rada će detaljnije biti razmatrani različiti tipovi baza podataka koji postoje i aktivno se koriste u izradi softverskih sistema, gde će se posebno naglasiti razlike između relacionih i nerelacionih tipova baza podataka, radi utvrđivanja najpogodnijeg tipa baze podataka za određenu namenu i model podataka.

2. Relacione baze podataka

Osnovna ideja relacionih baza podataka je da postoji način pamćenja konekcija između elemenata, međutim vremenom su relacione baze podataka postale standardizovane tako da postoje dodatni uslovi koja svaka relaciona baza podataka treba da ispuni.

2.1 Relacioni model podataka

Jedan od uslova jeste poštovanje **relacionog modela podataka**. Osnovna ideja jeste da su podaci organizovani u vidu tabela. Svaki tip entiteta (podatka) je skladišten u odgovarajuću tabelu koja u sebi skladišti razne podatke o tom tipu entiteta, ta tabela ima redove i kolone, gde kolone predstavljaju svojstva entiteta, tj. obeležja, a redovi predstavljaju pojedinačne elemente (entitete). Svaki entitet koji postoji mora da ima svoj jedinstveni identifikator, taj identifikator se naziva **primarni ključ** i ne mogu da postoje dva elementa u istoj tabeli sa istim primarnim ključem. Sama vrednost obeležja koje predstavlja primarni ključ može biti u raznim oblicima (broj, tekst, itd.).

#	INDEKS	IME	PREZIME	GOD. STUDIRANJA
1	PR 45/2019	MARKO	MARKOVIĆ	4
2	IN 21/2019	ANA	ANIČIĆ	4

Slika 1. Primer tabele Studenata

Na slici 1. može se videti primer jedne tabele u relacionom modelu, u ovoj tabeli će se skladištiti informacije o svim studentima, gde će se pamtiti njihovo ime, prezime, godina studiranja i njihov indeks. Pošto indeks predstavlja nešto jedinstveno za svakog studenta, on će služiti kao primarni ključ u tabeli.

#	ID	NAZIV PREDMETA	BROJ ESPB BODOVA
1	1	ALGEBRA	4
2	2	NEURONSKE MREŽE	8

Slika 2. Primer tabele Predmeta

Na slici 2. je prikazana tabela predmeta, podaci o predmetu su njegov naziv, broj ESPB bodova i predmetov jedinstveni ID koji služi kao primarni ključ.

Veze između entiteta u modelu se mogu realizovati na 2 načina:

1. Pomoću zasebnih tabela

#	INDEKS STUDENTA	ID PREDMETA	NAČIN POLAGANJA
1	PR 47/2019	1	POLAGANJE PREKO KOLOKVIJUMA
2	IN 21/2019	2	SLUŠA PREDMET

Slika 3. Primer tabele Slušanje_Predmeta

Tabela sa slike 3. funkcioniše kao **poveznik** između tabela Student i Predmet i pokazuje koji student sluša koji predmet i na koji način, to omogućujemo pomoću njihovih jedinstvenih identifikatora, tj. primarnih ključeva. Kada se primarni ključ jednog tipa entiteta nalazi u drugom, taj tip obeležja se naziva **strani ključ**. U ovom slučaju postoje dva strana ključa i njihov spoj zapravo predstavlja primarni ključ ove tabele.

2. Pomoću prostiranja primarnog ključa

#	ID	INDEKS STUDENTA	STATUS STUDENTA	RASPOLOŽIVA SREDSTVA
1	1	PR 47/2019	BUDŽET	0.00
2	2	IN 21/2019	SAMOFINANSIRANJE	400.00

Slika 4. Primer tabele Studentski_Profil

Tabela Studentski_Profil sa slike 4. sadrži podatke o studentskim profilima na veb-servisu za studente. Potrebno je znati kojem studentu profil pripada, pošto profil može da pripada samo jednom studentu, studentov primarni ključ se skladišti kao strani ključ u tabeli za studentske profile.

Način modelovanja veze između entiteta zavisi od tipa njihove veze i samih karakteristika modela podataka. Programer, najčešće, ima slobodu da modeluje svoju bazu podataka kako njemu najviše odgovara, u skladu sa modelom podataka i relacionim modelom, funkcionalnostima koje softverski sistem treba da izvršava i drugim faktorima bitnih pri dizajniranju softverskog sistema.

2.2 ACID principi

ACID principi predstavljaju osnovne osobine koje su vremenom dokazane kao potrebne u svim relacionim bazama podataka. Bitni principi su sledeći:

- **Atomicity** (Atomičnost) – pošto se jedna transakcija nad bazom podataka može sadržati iz više pod-zahteva, baza podataka mora da garantuje da će se transakcija izvršiti ispravno pre nego što se desi bilo kakva izmena podataka u bazi, to jest, ukoliko bar jedan deo transakcije bude neuspešan, baza mora da ostane u nepormenjenom stanju.
- **Consistency** (Konzistentnost) – baza podataka mora uvek da bude u važećem stanju nakon svake transakcije, to znači da se mora osigurati da svi podaci koji se unose u bazu moraju da budu validni, u skladu sa ograničenjima tabela i njihvoih obeležja.
- **Isolation** (Izolovanost) – izvršavanje jedne transakcije nad bazom podataka mora da bude izolovano od strane drugih transakcija.
- **Durability** (Istrajnost) – izmene izvršene nad bazom podataka, posle uspešne transakcije, moraju trajno da budu skladištene i zapamćene.

Ispunjavanjem ACID principa poboljšava se integritet podataka koje koristimo u softverskom sistemu, dobijaju se bezbednija ažuriranja podataka i obezbeđuje se pouzdan izvor podataka.

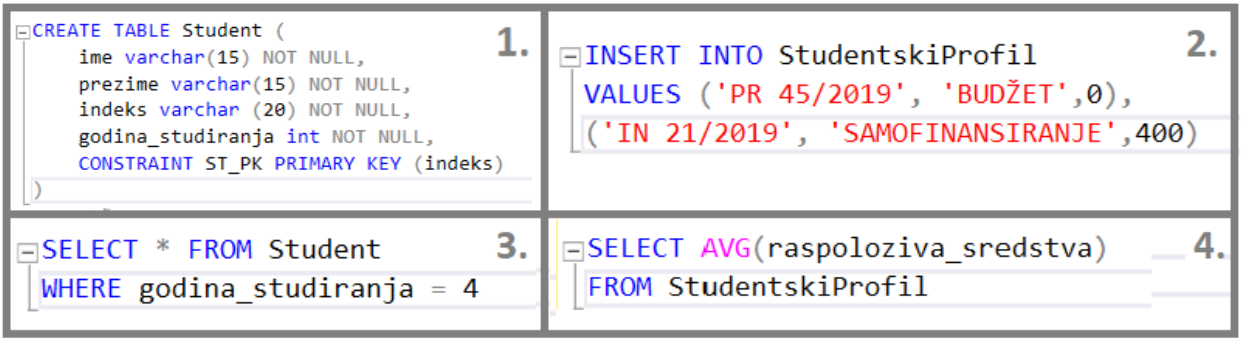
2.3 SQL

SQL (Structured Query Language) je standardizovan programski jezik za rad sa relacionim bazama podataka. Nastao je 1970-tih i podržan je od strane svih aktuelnih sistema za upravljanje relacionim bazama podataka.

„Data is the new oil...“ - Clive Humbly, 2006

„...and SQL is the drill.“ Nepoznato

SQL omogućuje razne operacije nad podacima koje se izvršavaju putem SQL upita.



Slika 5. SQL upiti za (1.) kreiranje tabele, (2.) unos podataka, (3.) pretragu podataka i (4.) obradu podataka.

Razumevanjem sintakse SQL programskog jezika, možemo kreirati kompleksne upite koji uzimaju podatke iz više tabela na osnovu njihovih veza, omogućavajući obradu veće količine podataka u jednom pozivu.

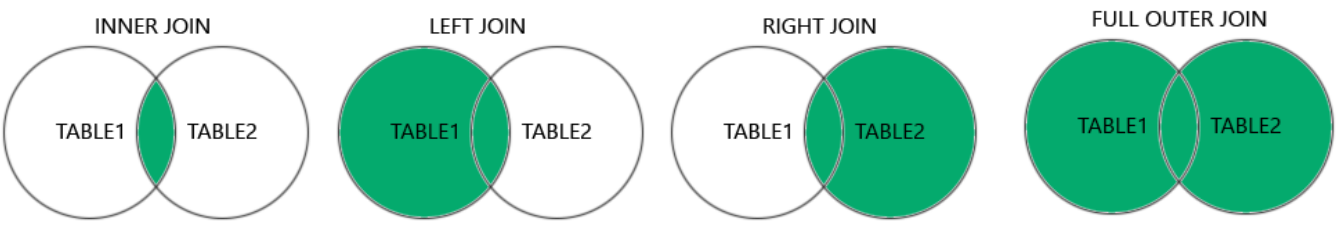
```
SELECT  Student.ime, Student.prezime,
        Predmet.Naziv,
        SlusanjePredmeta.NacinSlusanja,
        Profesor.Ime, Profesor.Prezime,
        Fakultet.Naziv
FROM Student
JOIN StudentskiProfil ON Student.Indeks = StudentskiProfil.indeks_studenta
JOIN SlusanjePredmeta  ON Student.Indeks = SlusanjePredmeta.StudentIndeks
JOIN Predmet ON  SlusanjePredmeta.PredmetId = Predmet.PredmetId
JOIN PredavanjePredmeta ON Predmet.PredmetId = PredavanjePredmeta.PredmetId
JOIN Profesor ON PredavanjePredmeta.ProfesorId = Profesor.ProfesorId
JOIN Fakultet ON Profesor.FakultetId = Fakultet.FakultetId
WHERE StudentskiProfil.status_studenta = 'Budžet'
```

Slika 6. primer dobavljanja podataka iz svih tabela u jednom SQL upitu.

Na slici 6. se može videti primer kompleksnog SQL upita koji objedinjuje podatke iz više tabela radi pridobijanja veće količine podataka. U ovom primeru se dobavljaju podaci o tome koji student sluša koji predmet, na koji način, kod kog profesora i na kojem fakultetu. Takođe se dodaje dodatan uslov da se razmatraju samo studenti na budžetu.

Na slici 6. je dodatano prikazan još jedan bitan deo SQL jezika, a to je spajanje tabela, to jest **JOIN**. Postoji više vrsta spajanja tabela u zavisnosti od postojanja odgovarajuće veze po kojoj se spajaju:

- unutrašnje spajanje (**INNER JOIN**) – pridobijanje podataka koji imaju odgovarajuću vezu između dve tabele, ukoliko neki entitet iz tabele 1. nema vezu sa nekim entitetom iz tabele 2 ili obrnuto. on se neće prikazati.
- Spoljašnje spajanje (**OUTER JOIN**) – spoljašnje spajanje može da bude sa leve strane (**LEFT JOIN**) i sa desne strane (**RIGHT JOIN**). Levo spajanje uzima sve podatke iz leve tabele (od znaka '**=**') i vezane podatke iz tabele sa desne strane, dok je za desno spajanje obrnuto. Razlika u odnosu na unutrašnje spajanje jeste što će se uzeti u obzir entiteti iz jedne tabele koji nemaju odgovarajuću vezu sa drugom tabelom.
- Potpuno spajanje (**FULL JOIN**) – potpuno spajanje predstavlja pridobijanje podataka iz obe tabele na osnovu neke veze bez obzira da li neki entiteti jedne tabele imaju veze sa entitetima druge tabele i obrnuto.



Slika 7. pojašnjenje različitih tipova spajanja tabela.

SQL jezik omogućuje razne dodatne komande koje mogu da pomognu sa izdvajanjem potrebnih podataka, kao što su:

- **GROUP BY** – grupiše podatke na osnovu nekog obeležja, koristi se kada je potrebno grupisanje podataka radi izvršavanja neke operacije na njima. Npr. srednja vrednost (**AVG**), maksimalna (**MAX**) ili minimalna (**MIN**) vrednost itd.
- **HAVING** – koristi se u paru sa **GROUP BY** radi filtriranja podataka, razlika u odnosu na **WHERE** je što **HAVING** filtrira samo grupisane podatke.
- **ORDER BY** – određuje redosled pridobijenih podataka na osnovu nekih obeležja. Podaci mogu biti ispisani uzlazno (**ASC**) i silazno (**DESC**) po nekom obeležju.

SQL je veoma sposoban alat koji nam ne služi samo kao posrednik u dobavljanju i skladištenju podataka, već ima sposobnost da ujedno vrši i kompleksnu obradu nad podacima koja uklanja potrebnu logiku obrade podataka koju bi inače programer morao da implementira, samim time, SQL je defakto programski jezik za rad sa relacionim bazama podataka koji je aktivno ažuriran na svakih par godina i prisutan je u mnoštvu softverskih sistema širom sveta.

2.4. Pristupi za generisanje modela podataka

Bazi podataka se često pristupa putem neke aplikacije, ovaj pristup je čest u višeslojnim softverskim sistemima koji koriste bazu podataka. Ovo ujedno znači da je potrebno definisati model podataka na dva mesta, jednom u bazi podataka i jednom na mestu aplikacije koja pridobija te podatke. Radi uštede vremena i osiguranja da se podaci definišu podjednako na oba mesta, nastale su tehnologije bazirane na generisanju koda koje koriste definisan model na jednom mestu i kreiraju identičan i prilagođen model na drugom mestu. Postoje dva pristupa koja koriste ovakve tehnologije.

- Code-First pristup – generisanje baze podataka na osnovu definisanog modela u aplikaciji.
- Database-First pristup – generisanje koda na osnovu definisane baze podataka.

Alati koji se koriste radi generisanja koda često mogu da funkcionišu na oba načina, odnosno mogu da generišu model podataka na osnovu postojuće baze podataka i obrnuto. Ti alati se zovu ORM (Object Relational Mapping) alati, a neki od tih alata su sledeći:

1. Entity Framework – ORM alat najčešće korišćen u **C#** jeziku.
2. Hibernate – ORM alat koji koristi **Java** klase.
3. Django ORM – ORM alat koji koristi programski jezik **Python**.
4. Sequelize – model definisan pomoću **JavaScript** i **TypeScript** klasa.
5. Laravel – alat koristan u **PHP** jeziku.

3. Nerelacione baze podataka

Baza podataka se smatra nerelacionom ako ne poštuje neka od pravila koja relacione baze moraju da poštuju. Sva pravila koja su osmišljena za relacione baze podataka služe radi uvođenja standardnog principa funkcionisanja i osiguravanja da pri radu sa relacionim bazama podataka, programer zna tačno koja ograničenja postoje i na koji način on može da koristi tu bazu, ovaj pristup nam daje sigurnost, predvidljivost i osnovnu ideju kako da modelujemo našu strukturu podataka. Međutim, relacioni model podataka ne mora uvek da bude idealan način modelovanja nekog modela podataka, zato su vremenom postale prisutnije nerelacione baze podataka. Poenta nerelacionih, to jest NoSQL, baza podataka je fleksibilnost, prilagodivost i sloboda programera da modeluje svoje podatke kako želi u bazi podataka.

Vremenom, prilikom dizajniranja nerelacione baze podataka, fokus je prešao sa kreiranja baze podataka koja se može koristiti za više slučajeva na kreiranje baze podataka koja je prilagođena posebnom načinu korišćenja i okolnostima. Zato su nastali razni tipovi baza podataka koje funkcionišu na veoma različite načine, neke slične relacionim, a neke sa kompletno drugačijim funkcionisanjem. U nastavku će biti razmatrani neki od tipova nerelacionih baza podataka, zajedno sa njihovim karakteristikama, ograničenjima i načinima korišćenja.

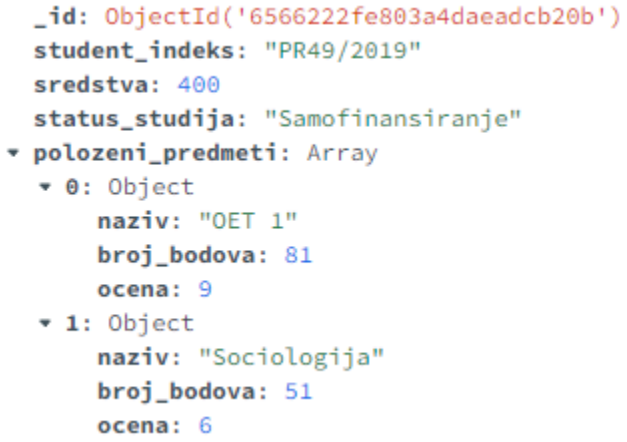
3.1. Dokument baze podataka

Prvi tip nerelacionih baza podataka koji će biti razmatran je Dokument tip baze podataka. Ovaj tip je jendostavan primer koncepta nerelacione baze podataka, jer ne čuva podatke o vezama između entiteta, ne koristi SQL programski jezik i nema predefinisan model podataka kojem je potrebno prilagoditi podatke.

3.1.1 Struktura podataka

Baza podataka tipa Document je struktuirana tako da je podeljena u dokumenta, gde svaki dokument ima više kolekcija podataka. Tipovi podataka mogu biti jednostavni, kao što su brojevi i tekst, a mogu i da budu kompleksni objekti sa više elemenata, slike u binarnom formatu, nizovi podataka i razne druge kompleksne strukture. Jedino ograničenje je da svaki podatak mora da ima unikatan ID da bih se mogao razlikovati od drugih i lakše pronaći.

Fokus ovog tipa baze podataka je na podacima. Njihova struktura nije ograničena sa predefinisanim modelom ili standardom, nego je prepušteno programeru da tu strukturu organizuje kako najviše odgovara njemu i/ili sistemu. Ovaj aspekat je od velikog značaja kada je struktura podataka nekog softverskog sistema od promenljive ili nepredvidljive prirode, sama baza podataka će skladištiti sve podatke koje joj damo (pod uslovom da su validnog oblika, to jest da se mogu definisati JSON notacijom), a njihova obrada i kontrola je prepuštena programeru. Primer prilagođene strukture podataka se može videti na sledećoj slici.



Slika 8. Primer strukture podataka skladištene u Document bazi podataka.

Ukoliko želimo da proširimo već spomenuti tip entiteta Studentski_Profil sa opcijom čuvanja podataka o položenim predmetima, u relacionim bazama podataka bih morali da napravimo dodatnu tabelu koja će čuvati podatke o položenim predmetima, gde je potreno pamtiti vezu sa studentskim profilom i sa predmetom. Međutim, u Dokument bazama podataka možemo čuvati sve ove potrebne podatke u jednoj tabeli, odnosno kolekciji. Na slici 8. se može videti kako su podaci o položenim predmetima čuvani u vidu liste, odnosno niza podataka o položenim predmetima unutar studentskog profila studenta koji ih je položio.

Ovim pristupom su svi potrebni podaci na jednom mestu i nema potrebe za kreiranje dodatke tabele koja će skladištiti podatke o položenim predmetima pa zatim pamtiti vezu između tabela. Na ovakav način možemo čitavu strukturu podataka prilagoditi na razne načine i omogućiti lakši pristup i pregled podataka. Takođe nema potrebe za pristup više tabela u jednom pozivu i nije potrebno vođenje računa o vezama između entiteta u bazi podataka.

Naravno, korišćenjem ovog metoda za čuvanje podataka, gubimo na sigurnosti i integritetu podataka koja je automatski uspostavljena prilikom rada sa relacionim bazama podataka. Kao što je već napomenuto ranije, svi standardi uvedeni za relacione baze podataka su tu da osiguraju integritet i konzistentnost podataka, zanemarujući ova ograničenja, žrtvujemo predvidivost i sigurnost za fleksibilnost i adaptivnost. Dobili smo fleksibilni model, ali sada moramo da vodimo računa o podacima koje ubacujemo, pre nego što ih ubacujemo, jer sama baza podataka neće vršiti nikakvu validaciju podataka za nas, osim provere validne (JSON) strukture podataka. Moguće je postojanje veza između entiteta u bazi podataka, ali je to takođe upotpunosti prepušteno programeru da vodi računa.

3.1.2. Pristup bazi podataka

Baza podataka dokument tipa koja će se koristiti radi primera u ovom radu je MongoDB, ona rukuje sa korisnim alatom zvanim MongoDB Compass putem kojeg možemo da upravljamo sa bazom podataka direktno, slično kao alatka SQL Server Management Studio kod relacionih baza podataka. Putem MongoDB Compass alata, možemo da kreiramo nove dokumente i kolekcije, a pored toga je moguće vršiti unos, brisanje i izmenu postojećih podataka.

Ovaj tip baza podataka takođe ima opciju pretrage podataka putem upita. Slično kao SQL kod relacione baze podataka, koriste se neki jezik upita. Jezik upita zavisi od baze podataka, a kod MongoDB, upiti za pretragu se sastoje iz niza parova OBELEŽJE : VREDNOST, gde se od baze podataka traži da nađe podatke koje imaju navedeno obeležje sa traženom vrednosti, uz vrednost je moguće ubaciti dodatne operatore koji ukazuju da se traži neka vrednost u zavisnosti od zadate, manja vrednost, veća, različita itd. MongoDB daje mogućnost korišćenja mnoštva operatora koja mogu pružati veliku korist prilikom pretragre i obrade podataka. Na slici 9. se može videti primer upita koji se unose u MongoDB Compass, prvi upit pronalazi studentske profile sa određenom sumom sredstava i statusom studija, dok drugi upit pretražuje studentske profile sa sumom sredstava većom od navedene.

```
{sredstva : 400, status_studija : "Samofinansiranje"}
{ sredstva: {$gt : 500} }
```

Slika 9. primer upita za pretragu studentskih profila

3.1.3. Razlog upotrebe

Document baze podataka su dobar primer prednosti i mana nerelacionih baza podataka. Fokusom na čuvanje podataka bez fiksne strukture omogućuje fleksibilnije rukovanje podacima, a pored toga, omogućavaju veoma brza čitanja i pisanja podataka u bazu podataka sa brzim odzivima i transakcijama i kratkotrajnim upitima.

Horizontalna skalabilnost je takođe veliki faktor, jer su document baze podataka veoma sposobne da rade u distribuiranim sistemima oslanjajući se na replikaciju. Pored toga su takođe odličan alat za sisteme čija se struktura podataka menja tokom razvoja, te je efikasno moguća izmena podataka bez velikih operacija nad tabelom. Dok bih kod relacione baze podataka izmena neke karakteristike zahtevalo brisanje i kreiranje tabele iznova, Document baze podataka sa svojom fleksibilnosti omogućavaju ovu izmenu bez velikih i/ili skupih operacija. U nastavku slede primeri softverskih sistema koji danas koriste Document baze podataka.

- BOSCH
- Sanoma
- SEGA
- ZF Technologies
- Forbes

Pored MongoDB postoje razne alternative za nerelacione baze podataka dokument tipa, a neke od njih su:

- Amazon DocumentDB – Amazonova baza podataka koja je kreirana da sarađuje sa AWS (Amazon Web Services) koji nude veliku pomoć u vidu replikacije i backup-a podataka.
- Cosmos DB – podržava različite strukture podataka pored JSON tipa kao što su key-value, graf i columbnar strukture. Laka globalna distribuiranost. Podrška za SQL. Konfigurabilna konzistencija podataka.
- ArangoDB – podržava različite strukture podataka pored JSON tipa kao što su key-value i graf strukture, gde je veliki fokus posvećen na graf strukturu. Koristi AQL (Arango Query Language) koji je sličan SQL-u. Transakcije nad podacima različitih modela.
- Couchbase Server – Multi-Dimensional Scaling koji olakšava distribuciju i replikaciju podataka. Konfigurabilna konzistencija podataka. Koristi N1QL upitni jezik sličan SQL-u. Integrisano keširanje često pristupljenih podataka. Lagano integrisana u druge tehnologije i baze podataka.
- CouchDB – koristi MapReduse upitni jezik baziran na JavaScript-u. ugrađene mogućnosti replikacije i distribuiranosti koje olakšavaju konflikte prilikom replikacije podataka. Moguća sinhronizacija sa mobilnim uređajima.

3.2. Key-Value Store

3.2.1. Struktura podataka

Jedan od najjednostavnijih primera nerelacionih baza podataka su Key-Value Store-ovi, odnosno Key-Value baze podataka. One fukncionišu na jednostavan princip, svi podaci su organizovani u Vrednost-Ključ parove (Key-Value pair). Svaka vrednost je definisana sa svojim jedinstvenim ključem i konačan izgled baze podataka podseća na strukturu rečnika ili HashSet kolekcije.

Često korišćena Key-Value baza podataka je Redis (Remote Dictionary Server), ovaj tip će se koristiti prilikom analize ovog tipa baze podataka.

Cilj Redis baze podataka, i Key-Value baza podataka ujedno, je da obezbedi brz i efikasan način čuvanja podataka jednostavne strukture. Tipovi podataka koji mogu da se čuvaju u Key-Value bazama podataka mogu biti osnovnog tipa, kao broj ili tekst, a mogu biti i malo složeniji objekti sa više svojstva, slike u binarnom obliku i nizovi.

3.2.2. Pristup podacima

Unos i čitanje podataka se obavlja putem ključeva, gde se pri unosu navede za koji ključ se dodeljuje određena vrednost, a pri čitanju sa zatraži vrednost sa određenim ključem. Primer unosa i čitanja podataka se može videti na sledećoj slici.

```
127.0.0.1:6379> set myRedisKey "redisCacheKey123"
OK
127.0.0.1:6379> get myRedisKey
"redisCacheKey123"
127.0.0.1:6379>
```

Slika 9. primer unosa i pregleda neke vrednosti putem Redis baze podataka.

Na slici 9. se može videti jednostavan primer memorisanja neke vrednosti, u ovom slučaju je to neki tekst, ali vrednost može da bude razne prirode u zavisnosti od tipa informacije i načina na koji će se koristiti.

Redis takođe ima podršku da radi sa kolekcijama, odnosno setovima, obezbeđujući razne operacije nad nizovima i podacima u njima. Na slici 10. se mogu videti operacije unosa, pregleda i brisanja elemenata nad kolekcijom položenih predmeta nekog studentskog profila. Može se primetiti da su podaci o položenim predmetima sačuvani sa proizvoljnim oblikom gde su naziv predmeta, ocena i broj bodova razdvojeni sa dve tačke. Prilikom pridobijanja ovih podataka, programer bih morao da parsira ove podatke radi odvajanja pojedinačnih informacija. Ograničenjem kompleksnosti strukture podataka u bazi podataka, poboljšana je brzina i dostupnost podataka, ali izgubljena je mogućnost vršenja kompleksnih obrada podataka od strane baze podataka kao što je moguće u relacionim bazama podataka.

```
127.0.0.1:6379> SADD indeks-PR44_2019-polozeni-predmeti OET:9:81 Sociologija:6:51
(integer) 2
127.0.0.1:6379> SMEMBERS indeks-PR44_2019-polozeni-predmeti
1) "OET:9:81"
2) "Sociologija:6:51"
127.0.0.1:6379> SADD indeks-PR44_2019-polozeni-predmeti Algebra:7:67
(integer) 1
127.0.0.1:6379> SMEMBERS indeks-PR44_2019-polozeni-predmeti
1) "OET:9:81"
2) "Sociologija:6:51"
3) "Algebra:7:67"
127.0.0.1:6379> SREM indeks-PR44_2019-polozeni-predmeti Sociologija:6:51
(integer) 1
127.0.0.1:6379> SMEMBERS indeks-PR44_2019-polozeni-predmeti
1) "OET:9:81"
2) "Algebra:7:67"
127.0.0.1:6379>
```

Slika 10. primer rada sa kolekcijom položenih predmeta nekog studenta

.

3.2.3. Razlog upotrebe

Key-Value baze podataka, sa time što su podaci „lakši“ jer nije potrebno čuvati njihove veze, njihovo skladišetnje i pristupanje rade brzo i efikasno. Ovo omogućuje brz rad sa velikom količinom podataka, pogotovo u distribuiranim sistemima gde je velika učestalost pristupa bazi podataka od strane raznih elemenata sistema. Key-Value baze podataka su projektovane da funkcionišu u ovakvim sistemima i primunjuje se u razne svrhe kao što su:

- keširanje podataka,
- praćenje sesije korisnika,
- eCommerce,
- podaci IoT (Internet of Things) uređaja i senzora,
- geografski podaci,
- podaci za video igre itd.

Sveokupno, Key-Value baze podataka predstavljaju lagano, fleksibilno, skalabilno i adaptivno rešenje za skladištenje podataka jednistavne strukture u distriburianim sistemima, obezbeđujući brz pristup i skladištenje podataka sa visokim performansama i visokom tolerancijom za visoku učestalost pristupa bazi podataka.

Neki od drugih primera aktuelnih Key-Value baza podataka su:

- Riak – visoka pristupnost i skalabilnost sa tolerancijom na ispade. Decentralizovana arhitektura sa efikasnijom perzistencijom podataka naspram Redis baze podataka.
- Amazon DynamoDB – fleksibilnija struktura koja omogućuje skladištenje kompleksnijih podataka sa podrškom AWS tehnologije. Visoka skalabilnost. Čest izbor za Web aplikacije, video igre i IoT tehnologiju.
- Google Cloud Bigtable – podaci kompleksnije strukture, Time-Series podaci, analitika i mašinsko učenje. Podrška za Google usluge i sisteme.
- RocksDB – ugradiva biblioteka koja se često koristi u sklopu drugih aplikacija. Efikasno skladištenje podataka u strukturu nazvanu LSM (Log Structured Merge) drvo.

3.3. Time-Series baze podataka

Prethodno je spomenuto da su nerelacione baze podataka dizajnirane sa specifičnom namerom korišćenja u planu, ovaj tip baza podataka je idealan primer toga. Time Series baze podataka rukuju sa podacima za koje je vreme najznačajniji aspekat, omogućavajući pored toga brzu obradu tih podataka radi kreiranja neke statistike ili razmatranja promene karakteristika podataka.

Time Series kao pojam u bazama podataka je prisutan već neko vreme, gde su prve baze podataka kreirane za razmatranje stanja tržišta, nekretnina i deonica. Međutim, u skorije doba su Time Series baze podataka dobile značajnu pažnju i postale su jedne od najizučavanijih tehnologija. Razlog za povećanje interesa u Time Series baze podataka je povećana zavisnost softverskih sistema od vrlo učestalih merenja, kroz razne aparate i senzore. Bilo je potrebno dizajnirati neki sistem koji će efikasno sklaidštiti te podatke i analizirati ih radi potrebnih analiza u sistemu.

InfluxDB predstavlja jedan aktuelan primer Time-Series baze podataka. Dizajnirana od strane InfluxData grupe, sa svojim viskoim performansama i integracijom sa Cloud sistemima postala je najkorišćenija Time Series baza podataka u današnjici.

3.3.1. Struktura podataka

U InfluxDB bazama podataka ne postoji šema podataka, jedino pravilo koje postoji jeste da svako uneto merenje mora da ima neku vremensku veličinu kao podataka, ukoliko se ta vrednost ne unese onda će baza podataka da pretpostavi da je vremenska vrednost jednaka trenutnom vremenu u nanosekundama. Svakom merenju se mogu dodeliti dodatni parametri koji su bitni za to merenje, ovi parametri mogu da budu tag-ovi ili polja (eng. Field). Tag-ovi su indeksirani i namenjeni da sadrže podatke po kojima će se pretraživati i/ili grupisati podaci i oni su indeksirani radi bolje pretrage, a polja sadrže neke bitne vrednosti koje će se analizirati i nisu indeksirana. U narednom primeru sa vazdušnim senzorima tag-ovi predstavljaju identifikacioni parametar senzora a među poljima su količina ugljen-dioksida, vlažnost vazduha i temperatura vazduha. Broj tag-ova i polja nije ograničen i definiše se prilikom unosa podataka.

InfluxDB baza podataka grupiše kolekcije podataka u “korpe”, odnosno Bucket-e, gde svaka korpa može da ima više različitih tipova podataka, odnosno merenja. Svi podaci u korpi se gledaju kao merenja i imaju vreme kao obavezan parametar. Prilikom kreiranja korpe potrebno je takođe odabrati vremenski period važenja podataka, svako uneto merenje će biti izbrisano iz baze podataka nakon isteka izabranog vremena. Ova pojava brisanja podataka iz baze nakon nekog vremena se zove polisa zadržavanja (eng. Retention Policy).

Potrebno je naznačiti da se ovakvom organizacijom gubi sposobnost za kompleksne upite koji pretražuju različite vrste merenja u isto vreme, stoga InfluxDB i ostale Time-Series baze podataka nisu pogodne za rukovanje sa strukturama koje imaju visoke kardinalitete. Takođe ukoliko kompleksnost

podataka postane prevelika moguće je primetiti visoko opterećenje procesorske jedinice računara i može zahtevati veliku količinu memorijskog prostora.

3.3.2. Pristup podacima

Podaci se mogu uneti pojedinačno, putem raznih fajlova sa podacima ili preko neke URL adrese, ovo omogućuje brz unos velike količine podataka na razne načine. Na sledećoj slici se može videti potrebna komanda za unos podataka putem URL adrese do Excell lista koja će u adekvatnu korpu da unese veliki broj (preko 460 000) merenja osam različitih senzora koji registruju podatke o količini ugljen dioksida, temperaturi i vlažnosti vazduha u određeno vreme.

```
.\influx write --bucket AirSensors --url https://influx-testdata.s3.amazonaws.com/air-sensor-data-annotated.csv
```

Slika 12. Komanda za unos podataka o merenjima parametara vazduha.

Samoj bazi se može pristupiti na više načina, jedan od najčešćih načina je pristup adresi http://localhost:8086 na kojoj se nalazi kontrolna tabla baze podataka koja je pokrenuta na lokalnom računaru. Putem ove kontrolne table moguće je izvršavanje raznih operacija nad bazom podataka, uključujući kreiranje novih korpi, kreiranje konfiguracije za unos podataka u bazu putem mnogih tehnologija i uređaja (C#, C++, Python, Apache Cassandra, Couchbase itd.) i najključnije operacije analizu podataka u bazi podataka. Kontrolna tabla omogućuje analizu raznih merenja na više načina: putem raznih grafova, kružnih dijagrama, tabela i sirove vrednosi podataka. Na sledećoj slici se može videti promena vrednosti količine ugljen-dioksida u vazduhu na dva različita senzora u vremenskom razmaku od 10 minuta vizuelizana uz pomoć grafa.

Slika 13. Promena količine ugljen-dioksida u vazduhu prikaza putem grafa.

3.3.3. Razlog upotrebe

Time-Series podaci su postali veoma prisutni razvojem eCommerce i IoT (Internet of Things) industrije, samim time je i upotreba Time-Series baza podataka takođe postala veoma učestala i prisutna upravo zbog toga što je kreirana sa namerom da radi isključivo sa tim tipom podataka. Obezbeđujući sistem koji adekvatno rukuje sa velikom količinom podataka i omogućuje efikasan unos i obradu tih podataka, kao i njihovo brisanje nakon obrade, Time-Series baze podataka su postale defakto alat u softverskim sistemima koji rade sa Time-Series podacima. Neki od primera softverskih sistema koji koriste Time-Series baze podataka su:

- Amazon – čuvanje podataka IoT senzora u skladištima.
- Uber – geografkso praćenje vozila i korisnika.
- LBBC – praćenje senzora vazdušnih turbina.
- Olympus Controls – praćenje biometričkih senzora na robotskim rukama prilikom fabričke proizvodnje.

Ovaj tip baza podataka predstavlja fantastičan primer idealnog alata za specifičnu upotrebu, što je zapravo i jedan od najvažnijih aspekata nerelacionih baza podataka.

Primeri Time-Series baza podataka pored InfluxDB:

- QuestDB –obrada time-series podataka putem SQL upita. Podrška za izvršavanje upita putem više niti. Veoma pouzdane performanse.
- Amazon Timestream – podrška za AMS. Serverless arhitektura. Podrška za SQL.
- Apache Druid – distribuirana OLAP (Online Analytical Processing) baza podataka sa fokusom na kolone. Podržava ugnježdene podatke. Podrška za SQL upite. Lakša hotizontalna skalabilnost.

3.4. Graf baze podataka

Graf baze podataka imaju ključnu karakteristiku, iako se smatraju nerelacionim jer ne koriste SQL i relacioni model, njihov veliki fokus je na vezama između entiteta. Struktura podataka u graf bazama izgleda kao graf, podeljen na čvorove i veze između njih. Entiteti u bazi podataka predstavljaju čvorove (nodes), dok njihove relacije predstavljaju veze (edges).

Graf baze podataka imaju slične aspekte kao relacione baze podataka po pitanju veza između entiteta, ali Graph baze podataka daju mnogo veći akcenat na njih i mogu da budu mnogo efikasnije u sistemima gde struktura podataka takva da su entiteti često povezani i to sa potencijalno velikim brojem drugih entiteta. U sledećoj tabeli će biti razmatrane razlike između relacione baze podataka i Graph baza podataka.

Kakakteristika	Relaciona baza podataka	Graf baza podataka
Struktura podataka	Tabele sa redovima i kolonama	Čvorovi i veze sa svojstvima
Odnosi između entiteta	Definisani sa stranim ključevima i poveznim tabelama	Definisani sa vezama između čvorova
Fleksibilnost	Krute i ograničene strukture	Fleksibilne strukture
Kompleksne obrade podataka	Zahtevaju kompleksne upite	Brzi upiti i brzi odzivi
Preporučena namena	Sistem baziran na transakcijama sa jednostavnijim vezama između entiteta	Sistemi sa visoko povezanim entitetima

Tabela 1. pregled odnosa nekih karakteristika između relacionih i graf baza podataka.

Graf baza koja će se koristiti kao primer je Neo4j. Projektovana od strane istoimene kompanije, jedna je od najčešće korišćenih Graf baza podataka koja dodatno i poštuje ACID principe. Omogućuje veoma brz početak rada kao i efikasnu vizuelizaciju podataka i njihovih veza, uz pomoć dodatnih biblioteka kao Workplace ili preko pretraživača. Jezik koji koristi Neo4j baza podataka radi izvršavanja upita se zove Cypher (Sajfer), on je kreiran u toku dizajniranja Neo4j baze podataka i prilagođen potrebama baze podataka.

3.4.1. Struktura podataka

Kao što je spomenuto ranije, podaci u Neo4j bazi podataka su organizovani u čvorove i grane, gde svaki element može da ima dodatna polja sa informacijama i mora da ima jedinstveno identifikaciono polje koje je često automatski generisano (primarni ključ). Svaki čvor ili grana mora da pripada nekom tipu, ovaj pristup olakšava pretragu polja u budućem radu i vizuelizaciji, međutim ovo ne mora da znači da svi objekti jednog tipa imaju fiksnu strukturu. Prilikom kreiranja novog objekta nekog tipa, moguće je ubaciti polja koja drugi objekti tog tipa nemaju, ovo omogućuje kreiranje fleksibilnijeg modela bez strukturnih ograničenja. Polja mogu biti raznih jednostavnih tipova kao što su tekst, broj ili boolean vrednost, a mogu i da skladište nizove podataka u vidu liste. Iako je moguće da se u Neo4J bazi podataka skladišti slika, strogo je preporučeno da se ta pojava izbegava jer Neo4J baza podataka nije optimizovana da radi sa podacima u tom formatu.

a.)
```
CREATE (:STUDENT {
        name:'Marko Marković',
        indeks:'PR49_2019'
        })
```

b.)
```
CREATE (:STUDENT {
    name: 'Ana Aničić',
    indeks: 'IN21_2020'
    }) -[:STUDIES_AT {Year:3}]→ (:FACULTY {
        name:'FTN',
        city : 'Novi Sad'
        })
```

c.)
```
MATCH (S:STUDENT) WHERE (S.indeks="PF12_2021")
MATCH (F:FACULTY) WHERE (F.name="FTN")
CREATE (S)-[:STUDIES_AT {Year:1}]→(F)
```

Slika 11. Cypher upiti za kreiranje čvora tipa Student (a.), veze koja povezuje studenta i fakultet (c.) i kombinovanog upita za kreiranja objekta Student, Fakultet i veze između njih,

Na slici 11. se mogu videti primeri upita za kreiranje čvorova i veza uz pomoć Cypher upita. Može se primetiti da je kreiranje veza *(c.)* malo drugačije nego kreiranje samih čvorova *(a.),* jer je potrebno pronaći elemente između kojih se veza kreira, te je potrebno ih pronaći putem MATCH komande ili kreirati ih zajedno sa čvorovima koje spaja *(b.).* Upitni jezik Cypher je projektovan tako da je lagano moguće kreirati više elemenata u jednom upitu koji su povezani nekim vezama, ovo omogućuje unos velike količine ulančanih elemenata u jednom upitu. Neo4J podržava ubacivanje ograničenja na polja čvorova i/ili veza. Na sledećoj slici se može videti upit koji postavlja ograničenje da svako polje indeks čvorova tipa Student mora da bude unikatno.

```
CREATE CONSTRAINT index_unique
FOR (S:STUDENT) REQUIRE (S.indeks) IS UNIQUE
```

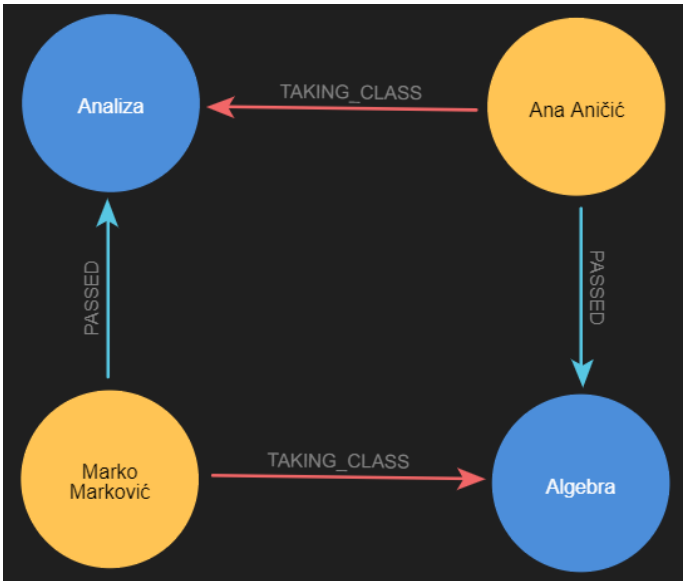
Slika 12. Cypher upit za postavljanje ograničenja unikatnog indeksa na čvorove tipa Student.

3.4.2. Pristup podacima

Graf baze podataka posvećuju veliku pažnju na vizuelizaciju podataka, omogućavajući jednostavan i pregledan prikaz potrebnih elemenata i njihovih veza putem Neo4J alata za prikazivanje podataka kao što su Aura ili Workplace. Na slici 13. se može videti primer Cypher upita kojim se pretražuju svi studenti, predmeti i njihove međusobne veze (može se primetiti da je u trećoj liniji upita dodata ključna reč OPTIONAL, ovo omogućuje prikaz studenata i predmeta koji nemaju međusobne veze), a na slici 14. se nalazi vizuelizacija pridobijenih podataka.

```
1  MATCH (S:Student)
2  MATCH (SUB:Subject)
3  OPTIONAL MATCH (S)-[Relation]→(SUB)
4  RETURN S,SUB,Relation
```

Slika 13. Cypher upit za pregled studenata, predmeta i njihovih veza.



Slika 14. Graf studenata, predmeta i njihovih veza.

Postoje razni drugi Cypher upiti koji mogu učiniti prikaz podataka još detaljnijim u skladu sa potrebama pretrage, takođe je putem Neo4J alata za prikaz podataka moguće prilagođavanje prikaza sa dodatnim faktorima (boje čvorova i veza, pozicije, tekst vidljiv nad elementima itd.). Razumevanjem sposobnosti Neo4J baze podataka i upitnog jezika Cypher moguće je izgenerisati detaljanu sliku elemenata nekog softverskog sistema što može služiti kao velika pomoć prilikom raznih uviđaja sistema.

3.4.3 Razlog upotrebe

Graf baze podataka su same za sebe veoma specifičan slučaj. Spadaju u nerelacione baze podataka ali čitav njihov fokus je upravo na entitetima i njihovim vezama, to jest relacijama. Često su poređene sa relacionim bazama podataka po svojoj sposobnosti rukovanja sa vezama između elemenata, gde su dokazale da zanemarivanjem nekih od ograničenja koja relacione baze podataka postavljaju možemo da dovedemo sposobnost rukovanja sa relacijama na potpuno drugi nivo. Sam upitni jezik Cypher može služiti kao idealan primer toga, kao što se može videti na slici 15. gde se vidi koliko malo koda je potrebno da zameni kompleksan SQL upit.

```
Cypher

MATCH (p:Product)-[:CATEGORY]→(l:ProductCategory)-[:PARENT*0..]
→(:ProductCategory {name:"Dairy Products"})
RETURN p.name

SQL

SELECT p.ProductName
FROM Product AS p
JOIN ProductCategory pc ON (p.CategoryID = pc.CategoryID
AND pc.CategoryName = "Dairy Products")

JOIN ProductCategory pc1 ON (p.CategoryID = pc1.CategoryID)
JOIN ProductCategory pc2 ON (pc1.ParentID = pc2.CategoryID
AND pc2.CategoryName = "Dairy Products")

JOIN ProductCategory pc3 ON (p.CategoryID = pc3.CategoryID)
JOIN ProductCategory pc4 ON (pc3.ParentID = pc4.CategoryID)
JOIN ProductCategory pc5 ON (pc4.ParentID = pc5.CategoryID
AND pc5.CategoryName = "Dairy Products");
```

Slika 15. poređenje kompleksnosti Cypher i SQL upita

Naravno, kao i u drugim slučajevima, ovo dovodi do nekih posledica. Graf baze podataka ne mogu da rade sa toliko kompleksnim podacima kao relacione baze podataka (slike) i podaci ne mogu da imaju toliko visoku sigurnost i integritet koji omogućavaju sva pravila relacione baze podataka, dodatno se smatra da je mnogo kompleksniji proces upoznavanja sa graf bazama podataka naspram relacione baze podataka. Graf baze podatake još uvek spadaju u novije tehnologije koje nisu imale priliku da se rasprostrane po tržištu, te je količina dostupne dokumentacije ograničena.

Sumirano, graf baze podataka omogućuju fleksibilno i pregledno čuvanje podataka koje dodaju potpuno novi vid interakcije sa podacima. Predstavljaju perfektno rešenje za softverske sisteme koji rukuju sa velikim brojem međusobno povezanih elemenata i veoma su sposobne da te veze podignu na novi nivo u poređenju sa drugim bazama podataka a naročito sa relacionim.

Česti slučajevi korišćenja graf baza su:

- biznisi koji prate interesovanja korisnika radi prilagođavanja preporučenog sadržaja,
- sistemi za detekciju prevare kao što koriste JPMorgan Chase i Intuit,
- sistemi za zdravstveno osiguranje (UnitedHealth Group) koji prate odnose klijenata i provajdera osiguranja itd.

Druge česte graf baze podataka su:

- Amazon Neptune – Amazonov alat koji podržava rad sa AWS sistemima. Podržava RDF model. Podržava Gremlin i SPARQL upitne jezike.
- JanusGraph – projektovana po uzoru na Apache Cassandru, Hbase i Google Bigtable. Velike tolerancija na ispade. Razna rešenja za indeksiranje sa podrškom za Gremlin upitni jezik.

3.5. Search Engine baze podataka

Sledeći primer nerelacionih baza podataka su Search Engine baze podataka. Ove nerelacione baze podataka su dizajnirane da olakšaju i ubrzaju pretragu podataka. Prilikom unosa podataka, baza podataka pamti i indeksira određene karakterestike podataka da bi mogla u buduće lakše da izvrši pretragu nad njima.

Prva Search Engine baza podataka je bila Apache Lucene koja je počela kao Open Source biblioteka, to znači da je njen izvorni kod bio dostupan svima da rade sa njim i koriguju za svoje potrebe. Lucene je bila ključan korak za dalji razvoj Search Engine baza podataka, služeći kao temelj za danas često korišćene Elasticsearch i Apache Soir baze podataka.

Search-Engine baza podataka koja će služiti kao primer u ovom radu je Elasticsearch. Kao sama baza podataka, Elasticsearch pripada Open Source kompletu zvanim Elastic Stack koji sadrži par dodatnih alata koji mogu da pomognu Elasticsearch bazi podataka pri radu. Ti alati su Kibana, Beats i Logstash. Kibana služi kao pomoćni alat za rad sa podacima, preko njega se mogu vršiti vizuealizacija i upiti nad bazom podataka, dok Beats i Logstash pomažu pri prikupljanju podataka i vođenja evidencije nad njima.

3.5.1. Struktura podataka

Elasticsearch baza podataka skladišti podatke u dokumentima u JSON formatu, samim time važi da bilo koji podatak može da se skladišti ukoliko može da se zapiše u JSON notaciji. Elasticsearch prihvata podatke u JSON notaciji, dodeljuje im unikatan identfikator, indeksira ih po vrednostima koje sadrže i skladišti te JSON objekte u strukturu koja se zove obrnuti indeks. Ova struktura omogućuje efikasno praćenje unikatnih vrednosti teksta svih dokumenata u bazi podataka i omogućuje veoma brzo pretraživanje svih dokumenata po tekstualnoj vrednosti. Postoje, takođe i druge strukture podataka koje Elasticsearch koristi za skladištenje numeričkih i boolean (true/false) vrednosti. Elasticsearch koristi dinamično mapiranje prilikom kreiranja indeksa nad poljima podataka koji se unose, ovo omogućuje da Elasticsearch sam odluči kako će indeksirati neki podatak ali je takođe moguće definisati posebno mapiranje podataka u skladu sa nekim pravilima koje korisnik želi da poštuje. Ovaj pristup je adekvatan ukoliko želimo da se neki podaci indeksiraju na više određenih načina ili ako radimo sa podacima specifičnog jezika pa se zahteva poseban način indeksiranja.

3.5.2. Pristup Podacima

Komande, to jest upiti nad bazom se vrše u vidu HTTP zahteva. Pošto baza podataka obslužuje korisnika na adresi http://localhost:9200/, nema potrebe da se koristi poseban upitni jezik, jer slanjem HTTP zahteva na tu adresu sa potrebnim podacima moguće je izvršavati sve operacije nad bazom podataka i sva komunikacija može biti odvijena putem HTTP poziva i odgovora. Elasticsearch baza podataka čuva podatke fleksibilnom strukturom u kolekcijama koje se zovu dokumenti, dokumenti se nalaze na čvorovima koji predstavljaju uređaje na kojim je konfigurisana Elasticsearch baza podataka a čvorovi pripadaju jednom Klusteru. Ovakva organizacija omogućuje efikasnu skalabilnost i distribuiranost.

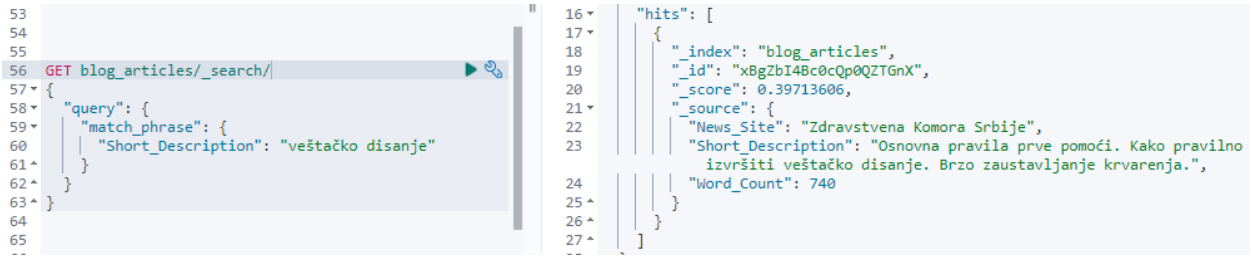
Slika 14. HTTP pozivi za kreiranje dokumenta 1.), pretragu svih podataka 2.), unos podataka 3.) i 4.) i brisanje podataka 5.)

Na slici 14., uz pomoć Kibana alata, se mogu videti upiti za kreiranje kolekcije podataka, odnosno dokumenta, unos konkretnog objekta, pregled podataka, vršenje upita za pretragu podataka i brisanja podataka. Kao što je navedeno ranije, svi ovi zahtevi predstavljaju HTTP pozive gde poziv GET često služi za pridovijanje podataka, DELETE poziv za brisanje a PUT i POST služe za kreiranje podataka i/ili kolekcija. Prilikom generisanja nekog objekta, potrebno mu je dodeliti unikatno identifikaciono polje **_id**, ukoliko je korišćen PUT poziv prilikom unosa objekta potrebno je navesti vrednost identifikacionog polja u sklopu HTTP poziva, dok POST poziv omogućuje da Elasticsearch tom objektu, ukoliko nije navedeno u HTTP pozivu, automatski dodeli unikatnu vrednost identifikacionog polja.

Često korišćen način pretrage podataka je pretraga punog teksta (full text search). Ovaj pristup se zasniva na pretrazi svih polja dokumenata za vrednost nekog teksta koja je zatražena u upitu. Elasticsearch je kreiran da vrši ovoliko detaljne pretrage, oslanjajući se na svoju sposobnost indeksiranja podataka, te process pretrage svih polja svih podataka u svim dokumentima može da se izvrši u veoma kratkom periodu. Naravno, u sklopu samog upita, možemo da specificiramo našu pretragu.

Slika 15. Primer pretrage punog teksta sa komandom match.

Na slici 15. se može videti primer pretrage novinarskog artikla po punom tekstu, gde je kao parametar zadat tekst *veštačko disanje* a izabrana komanda *match.* Elasticsearch će pregledati sve svoje dokumente za podatke koje imaju vrednost tog teksta u svom sadržaju, gde će za svakog dodeliti i faktor važnosti u zavisnosti koliko smatra da je taj podatak traženi. Može se uočiti na slici da je Elasticsearch pronašao jedan artikal koji ima sadržaj o veštačkom disanju, što je traženo, i jedan artikal koji isto ima traženi tekst u sebi, ali sa drugačijim redosledom reči nego što je zapravo traženo i o potpuno drugoj temi. Zbog toga se on nalazi posle prvog artikla koji se više podudara sa traženim tekstom. Ova pojava može da bude veoma problematična u nekim slučajevima i možemo ju izbeći korišćenjem druge komande *match_phrase* koja zahteva da se naš tekst koji smo uneli kao parametar pojavljuje u podatku u istom redosledu kako je zadat i time obezbediti detaljniju i tačniju pretragu. Primer poziva sa *match_phrase* komandom se nalazi na slici 16.



Slika 16. primer preciznije pretrage punog teksta sa komandom match_phrase.

Postoje razne druge komande za još detaljnije pretrage podataka, kao i dodatni parametri koje možemo dodati u komande radi poboljšavanja korisnikovog iskustva prilikom pretrage podataka na nekoj internet stranici ili aplikaciji i učiniti pretragu mnogo efikasnijom.

3.5.3. Razlog korišćenja

Search Engine baze podataka su vrlo sposobne da pretražuju podatke, bilo to struktuirani ili nestruktuirani, na osnovu priloženog teksta, za to se ove baze podataka često koriste radi automatskog dovršavanja reči prilikom pretrage na nekoj veb stranici. Kada korisnik krene da upisuje neke reči u polje za pretragu, Search Engine baze podataka su dovoljno brze da pretraže potrebne podatke sa tim tekstom i pronađu potrebne podatke ili dovrše reči u polju za pretragu sa mogućim potrebnim tekstom. Zbog ovoga korišćenje Search Engine baza podataka znatno može da poboljša korisnikovo iskustvo i operabilnost nekog sistema.

Takođe, ove baze podataka su izuzetno vešte u sortiranju podataka po određenim obeležjima da bi pregled tih podataka bio prilagođen nekim potrebama. Pored toga, odličan su izbor za vršenje evidencije podataka, odnosno logging. Jedan distribuirani sistem, koji može da koristi razne softverske jedinice i platforme, može da prepusti evidentiranje podataka nekoj Search Engine bazi podataka bez brige o performansama.

Pretraga velike količine podataka može da se vrši putem ugrađenih algoritama za pretragu, ali moguće je i implementirati neke prilagođene algoritme i putem njih vršiti pretragu. Ovo je koristan aspekat Search Engine baza podataka jer daje korisniku još veću fleksibilnost prilikom rukovanja podacima. Neke velike firme mogu da ulože veliku pažnju u kreiranje sobstvenog algoritma za pretragu, recimo putem mašinskog učenja, koje postaje sve više zastupljeno vremenom, i onda putem tog algoritma izvršavati pretragu podataka na način prilagođen njihovim potrebama.

Razumevanjem fukncionisanja Elasticsearch baze podatak i strukture HTTP poziva, moguće je vršenje raznih operacija nad Elasticsearch bazi podataka. Elasticsearch, zajedno sa drugim alatima u Elastic Stack-u, predstavlja veoma sposobno rešenje za distribuirane sisteme koji rade sa velikom količinom tekstualnih podataka, gde alati kao Kibana omogućavaju dodatne analitičke sposobnosti i integraciju sa drugim sistemima. Danas je Elasticsearch znatno prisutan u raznim softverskim sistemima, kao što su Github, Amazon i Wikipedia, gde služi kao veliki pomoćnik u pretrazi velike količine podataka za specifični traženi ključan sadržaja.

3.6. Columbnar baze podataka

Poslednji tip nerelacione baze podataka koji će se razmatrati jeste Columbnar tip. Ovaj tip baza podataka ima specifičan način rada sa podacima gde je veći fokus posvećen na kolone u tabelama umesto na redove. Ovim pristupom olakšavamo čuvanje i obradu velike količine podataka, jer čuvanje podataka po koloni zahteva manje memorije i lakše je za grupisanje. Samim time su zahtevi čitanja i pisanja veoma brzi pri korišćenju ovog tipa nerelacionih baza podataka.

Glavni predstavnik ovog tipa nerelacionih baza podataka je **Apache Cassandra** koja je ubrzo nakon nastanka postala jedna od najčešće korišćenih baza podataka u softverskim sistemima širom sveta.

3.6.1. Struktura podataka

Cassandra je slična relacionim bazama podataka po tome što se podaci čuvaju u tabelama koje imaju kolone i redove, ti podaci mogu biti raznih tipova od jednostavnih kao što su brojevi i tekst do kompleksnijih tipova kao što su lista, set, mapa i tuple koji mogu da skladište više elemenata u jednom polju tabele. Upitni jezik koji se koristi za rad sa Cassandrom je CQL (Cassandra Query Language).

Način na koji Cassandra čuva podatke je drugačiji nego kod relacionih baza podataka. Iako i jedna i druga čuvaju podatke u vidu tabela sa redovima i kolonama, relacione baze podataka koriste jedan sistem za upravljanje bazama podataka na kojem su definisane razne baze podataka i unutar njih tabele, dok Cassandra grupiše tabele u ključne prostore (Keyspace), a podaci iz tih tabela su rasprostranjeni na više **čvorova** u sklopu jednog **Klastera** (Cluster). Na ovaj način Cassandra grupiše podatke u svom sistemu na više mesta u zavisnosti od vrednosti ključnih obeležja. Podaci iz tih tabela takođe mogu biti replicirani na više čvorova radi bezbednosti i lakše pristupnosti. Postupak, odnosno strategija, za raspodelu podataka po čvorovima je konfigurabilna sa ciljem da obezbedi programeru da odluči kako mu najbolje odgovara da se podaci raspodele.

```
1 CREATE KEYSPACE faculty
2 WITH replication = {
3     'class' : 'SimpleStrategy',
4     'replication_factor' : 3
5 };
```

Slika 17. CQL upit za kreiranje Keyspace-a Fakultet.

Na slici 17. se može videti upit za kreiranje jednog ključnog prostora, odnosno keyspace-a. Prilikom definisanja jednog ključnog prostora, potrebno je definisati na koji način će se voditi računa o replikaciji podataka. U polju ‘class’ je potrebno navesti strategiju replikacije i u zavisnosti od strategije potrebno je navesti dodatne podatke o procesu replikacije, u ovom primeru se koristi ‘SimpleStrategy’ gde je samo potrebno navesti faktor replikacije koji određuje na koliko čvorova će podaci biti replicirani prilikom unosa podataka. Postoji i ‘NetworkTopologyStrategy’ koja omogućuje dodatnu konfiguraciju koja je korisna ako je sistem kompleksniji i distribuiran, ova strategija može da konfiguriše raspored replika u odnosu na redosled i blizinu čvorova.

U slučaju sa slike 17. će se podaci iz svih tabela replicirati na tri čvora nakon unosa, samim time je pristup podacima lakši jer se opterećenje pristupa može raspodeliti na tri čvora podjednako, ali unos podataka sporiji jer se ti podaci moraju replicirati na više čvorova.

Još jedna sličnost sa relacionim bazama podataka, tabele u Cassandra bazi podataka moraju imati definisan primarni ključ koji obezbeđuje da ne postoje dva elementa u tabeli sa istom vrednosti primarnog ključa, ali taj primarni ključ ima dodatnu korist. Ukoliko tabela ima samo jedno polje definisano kao primarni ključ onda funkcioniše isto kao kod relacionih baza podataka, ali ukoliko postoje više obeležja koja predstavljaju primarni ključ onda se ta obeležja dele na particione ključeve (Partition Key) i ključeve za grupisanje (Clustering Key). Ovi elementi utiču na raspodelu podataka po čvorovima unutar klastera. Svi elementi sa istim particionim ključem će se nalaziti na istom čvoru i biće raspodeljeni (sortirani) na osnovu vrednosti ključeva za grupisanje. Ovo obezbeđuje da su svi podaci koji imaju nešto slično nalaze na istom mestu, što je veoma korisno ukoliko je potrebno pristupiti većem broju podataka sa istom vrednosti particionog ključa.

```
CREATE TABLE IF NOT EXISTS faculty."Student_by_FacultyType" (
    "FacultyType" text,
    "StudentIndex" text,
    "FullName" text,
    "CurrentYear" int,
    PRIMARY KEY ("FacultyType", "StudentIndex")
);
```

Slika 18. CQL upit za kreiranje tabele Student.

Na slici 18. se može videti definicija tabele Student koja ima polja za tip fakulteta na kojem student studira, njegov studentski indeks, godina studiranja i ime i prezime. Prilikom definisanja primarnog ključa, pošto je polje tip fakulteta navedeno prvo ono predstavlja particioni ključ, sva obeležja navedena posle prvog predstavljaju ključeve za grupisanje i to je u ovom slučaju samo indeks studenta. U konkretnom primeru, svi studenti na fakultetu ‘FTN’ će se skladištiti na istim čvorovima i biti sortirani po vrednosti indeksa u rastućem redosledu. Redosled sortiranja je takođe moguće konfigurisati,

3.6.2. Pristup podacima

Potrebno je naglasiti da je strogo preporučeno da pretraga podataka iz određene tabele bude samo po vrednosti particionog ključa. Dodatnim komandama, moguća je pretraga bez navođenja particionog ključa ali može dovesti do veoma sporih upita i nije savetovano jer način raspoređivanja podataka po čvorovima na osnovu particionog ključa znatno otežava proces pretrage tih podataka po drugim obeležjima. Može se primetiti da ovim ograničenjem žrtvujemo kompleksnost upita za brži pristup podacima, samim time strukture tabela se moraju kreirati sa ovim ograničenjem u vidu. Ovakav pristup često dovodi do pojave da svaka tabela odgovara jednom konkretnom upitu koji će biti upućen bazi podataka. Na sledećoj slici se može videti CQL upit za pregled podataka.

```
SELECT * FROM faculty."Student_by_FacultyType"
WHERE "FacultyType" = 'FTN';
```

Slika 19. CQL upit za kreiranje tabele Student.

3.6.3. Razlog upotrebe

Sumirano, Cassandra obezbeđuje jedan kompleksan sistem za upravljanje bazama podataka koji omogućuje detaljno konfigurisanje za potrebe distribuiranih sistema. Davajući programeru slobodu da koriguje fleksibilnost sistema i sigurnost i dostupnost njegovih podataka. Cassandrin način formatiranja tabela može da služi kao odličan primer sistema koji uzima već poznate funkcionalnosti i modifikuje ih kroz razne izmene koje u jednu ruku ograničavaju upotrebu a u drugu ruku znatno ubrzavaju pridobijanja i unosa podataka da bih sveobuhvatno obezbedile idealan sistem za određenu svrhu.

U nastavku su primeri softverskih sistema koji danas aktivno koriste Cassandra bazu podataka:

- Activision – sistem koji vodi računa o obaveštavanju korisnika sa personalizovanim ponudama.
- Apple – više od 10 petabajta podataka na preko 75 000 čvorova.
- Discord
- Ebay – preko 400 miliona čitanja i 100 miliona pisanja u baze podataka na dan.

Neke od drugih često korišćenih baza podataka Columbnar tipa su:

- ScyllaDB – optimizovana za veoma učestale upite i brze odgovore. Fokus na kompleksan okvir čuvanja podataka nazvan Seaster Framework koji je optimizovan za moderan hardware i procesore sa više jezgara. Svako jezgro procesore rukuje sa jednim čvorom baze podataka. Baziran na C++.
- Apache Kudu – fokus nad obradom podataka koji se često menjaju. Podržava čuvanje podataka orijentacijom na redove i kolone. Visoka konzistencija podataka.
- Snowflake – cloud inspirisana baza podataka koja podržava SQL upite. Optimizovana za OLAP sisteme sa fokusom na analizu struktuiranih i slabostruktuiranih podataka. Centralizovana arhitektura skladištenja podataka. Podrška za Amazon i Google servise koji koriste Cloud tehnologiju.
- Google BigQuery – server-less OLAP arhitektura sa podrškom za SQL upite. Koristi Google-ov distribuirani fajl sistem Colossus kao i druge Google-ove usluge. Čuvanje i obrada struktuiranih podataka.

3.7. Sumiranje tipova nerelacionih baza podataka

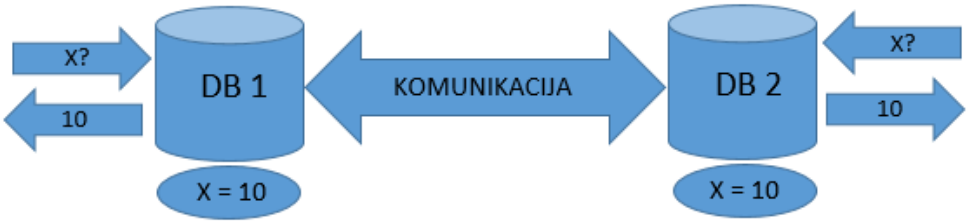
Nerelacione baze podataka su kreirane sa principom da uvode drugačiji način rukovanja sa podacima, oslobađanjem programera od restrikcija relacionih baza podataka dajemo slobodu da se kreiraju raznorazni sistemi koji gledaju na te podatke na unikatne načine. Bilo to fokus na način čuvanja podataka, fleksibilniju strukturu, evidenciju ili prikaz, pa čak i posvećivanje veće pažnje na konfiguraciju sistema koji će da rukuje sa tim podacima, nerelacione baze su fantastičan primer kreiranja specifičnog alata za specifičnu korist radi maksimalnog dobitka. U nastavku je kratak opis svih tipova nerelacionih baza podataka koje su razmatrane u ovom radu i njihove glavne osobine.

- Key-Value Store – brz pristup podacima jednostavne strukture, idealan za keširanje i čuvanje jednostavnih podataka kojima se često pristupa.
- Document – pouzdana baza podataka za sisteme sa vrlo učestalim transakcijama nad podacima sa fleksibilnom strukturom.
- Graf baze podataka – idealna baza podataka za sisteme sa kompleksnijim modelom podataka gde je veza između entiteta u sistemu od podjednake, ako ne i veće značajnosti u odnosu na same podatke entiteta.
- Time-Series – veoma pogodna za veliku količinu podataka čija je ključna vrednost vreme i njihovu analizu.
- Search Engine – baza podataka koja se fokusira na efikasnu i brzu pretragu podataka. Idealna za sisteme u kojima se često pretražuje velika količina podataka po određenom (najčešće tekstualnom) svojstvu.
- Columbnar – visoko konfigurabilna baza podataka sa akcentom na distribuiranost i skalabilnost. Ograničena ali efikasna pretraga. Veoma brzi zahtevi čitanja i pisanja.

4. Klasifikacija spomenutih baza podataka (CAP teorema)

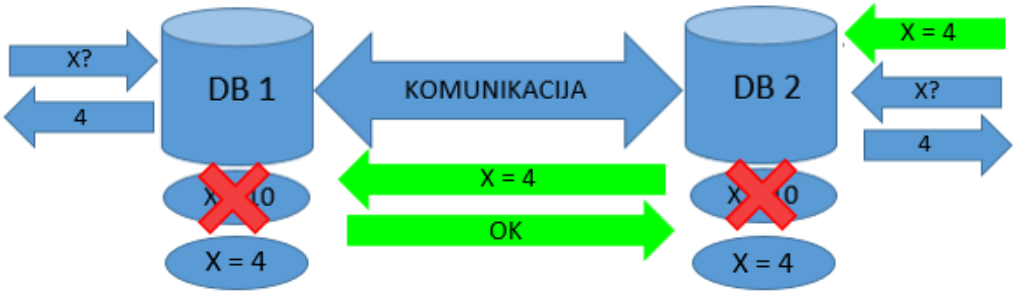
Do sada su bile razmatrane razne baze podataka, zajedno sa njihovim prednostima, manama i namenama. Međutim, u današnjim okolnostima prilikom dizajniranja distribuiranih sistema postoje tri karakteristike koje se smatraju vrlo bitnim i to su: konzistencija (Consistancy), dostupnost (Availability) i tolerancija na particionisanje usled prekida konekcije između odvojenih elemenata sistema (Partition Tolerance). U nastavku će biti opširnije objašnjeno šta svaka od ove tri karakteristike znači.

Konzistencija predstavlja sposobnost sistema da na svim pristupnim tačkama sistema obslužuje korisnike sa istim i tačnim podacima. Ukoliko jedan sistem ili baza podataka ima više čvorova ili elemenata koji rukuju sa istim podacima, ukoliko sistem želi da bude konzistentan, onda korisnici trebaju da dobiju istu informaciju bez obzira kojem čvoru pristupaju.



Slika 20. Konzistentan distribuiran sistem

Na slici 20. je prikazan primer konzistentnog sistema koji predstavljaju dva čvora sa svojim bazama podataka **DB 1** i **DB 2**, ova dva čvora međusobno komuniciraju putem nekog komunikacionog kanala i međusobno prenose informacije o upisima I promenama podataka. Putem ovog prenošenja podataka, odnosno replikacije, omogućujemo da svi korisnici dobijaju iste podatke I samim time sistem je konzistentan. Primer ove komunikacije se nalazi na slici 21. gde je zelenim strelicama predstavljena komunikacija sistema prilikom unosa, odnosno promene nekog podatka.



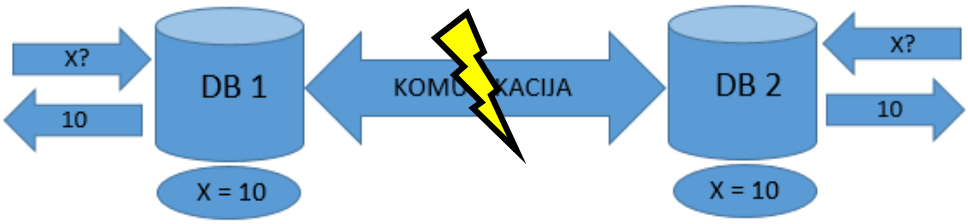
Slika 21. Komunikacija između čvorova sistema prilikom unosa ili promene podataka.

Dostupnost predstavlja sposobnost distribuiranog sistema da u svakom trenutnku bude dostupan svim korisnicima i da se svaki upit nad elementima sistema izvši uspešno bez obzira kojem elementu sistema se pristupa. Dostupnost se često postiže povećanjem broja elemenata kojima korisnici pristupaju I osiguravanjem da svaki od tih elemenata može sam da obavlja svu potrebnu komunikaciju sa korisnicima. Sistemi koji daju akcenat na dostupnost često do neke mere zanemaruju konzistentnost sistema da bih osigurali brze odgovore svakog elementa sistema.

Kao što se može videti na slici 22. prilikom promene vrednosti podatka X, čvor **DB 2** ne inicira signal ka čvoru **DB 1** sa informacijom da se vrednost promenila I u jednom trenutnku ova dva čvora će rukovati sa drugačijim informacijama. Ovo ne znači da sistemi koji daju akcenat na dostupnost nemaju nikakav sistem replikacije podataka, nego ukazuje da je proces replikacije manje bitan te se izvršava u manje učestalom periodu nego kod sistema koji praktikuju konzistenciju nad pristupnosti.

Slika 22. Primer sistema sa akcentom na dostupnosti.

Tolerancija nad particionisanjem je veoma bitna karakteristika kod distribuiranih sistema. Baš zbog same prirode distribuiranih sistema koja zahteva da su elementi sistema odvojeni i da komuniciraju, sistem mora da osigura funkcionisanje ukoliko dođe do prekida u komunikaciji između elemenata sistema i ovo osiguranje se smatra Tolerancijom particionisanja. Pored omogućavanja da elementi sistema funkcionišu iako nemaju međusobnu konekciju, tolerancija particionisanja se može dodatno omogućiti dodavanjem dodatnih veza između elemenata, međutim ove metode znaju da zahtevaju veliku investiciju i zahtevaju znatnu pripremu. Postoje i razne druge metode koje softverski sistemi koriste za ove slučajeve.



Slika 23. Primer sistema koji funkcioniše iako je došlo do ispada prilikom komunikacije.

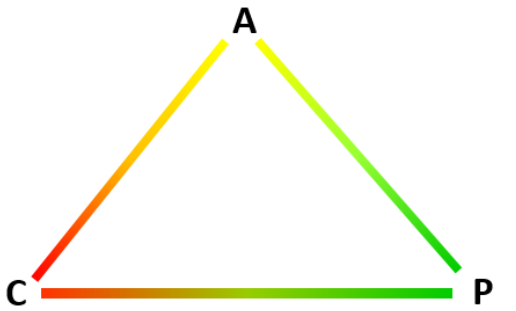
Istraživanjem je došlo do zaključka da su navdene tri karakteristike, Konzistentnist (C), dostupnost (A) i tolerancija particionisanja (P) najbitnije karakteristike kod distribuiranih sistema, međutim ispunjavanje sve tri karakteristike se smatra nemogućim jer ne postoji način da dva elementa imaju konzistentne vrednosti ako nisu povezani, a rad sa manjom količinom elemenata ne ispunjava potrebnu dostupnost sistema. Ovo zapažanje je primetio Erik Brewer koji je došao do zaključka da jedan distribuirani sistem može da ispuni samo dve od tri najbitnije karakteristike distribuiranih sistema i samim time zasnovao Brewerovu teoremu koja se često naziva i **CAP Teorema**. Dodatnim razmatranjem da nije efikasno imati distribuiran sistem koji ne može da funkcioniše prilikom pojave ispada komunikacije, donet je zaključak da je jedino moguće žrtvovati konzistentnost ili dostupnost.

Način na koji se može žrtvovati dostupnost i zadržati konzistentnost pri ispadu jeste da drugi element prestane sa radom dok se komunikacija ne uspostavi ponovo. Garantujemo konzistentnost tako što će sve operacije biti zapamćene i moguće je rukovati samo sa aktuelnim podacima. Postoji i pojam eventualne konzistencije koji znači da će sistem postati konzistentan ubrzo nakon ponovog uspostavjanja veze poja je bila prekinuta između elemenata sistema.

Primer sistema koji zahteva konzistenciju nad dostpunosti jesu transakcioni sistemi, na primer bankarski sistemi. Ukoliko želimo da imamo distribuirani sistem bankomata, neophodno je obezbediti konzistentan sistem koji može dobro da se zaštiti od neadekvatnog rukovanja podacima. Recimo da se desio slučaj da se veza između nekih bankomata i banke prekinula, u ovom slučaju bi bilo katastrofalno omogućiti da bankomati vrše bilo kakvu isplatu novca, te je konzistentnost bitnija nego dostupnost.

Neki sistemi i baze podataka se fokusiraju na dostupnost naspram konzistentnosti. Ovo ukazuje da ukoliko se desi prekid u vezi između elemenata sistema, elementi će nastaviti da rade i obslužuju korisnike, a nakon uspostavljanja prekinute veze će sinhronizovati podatke koji su se promenili. Jasno je da ovim postupkom može doći do neslaganja podataka na dva elementa u sistemu, međutim u nekim sistemima ovo ne mora da predstavlja veliki problem. Primer ovog slučaja jeste internet stranice YouTube, prilikom pregleda informacija o broju pregleda nekog video snimka može doći do pojave da jednom korisniku bude drugačiji broj pregleda nego drugom. Kad su u pitanju veliki brojevi gde je baš tačna vrednost od manje koristi nego okvirna vrednost, korisnije je omogućiti dostpunost naspram konzistentnosti.

Ukratko, što je kompleksnija struktura podataka koja može da sadrži osetljive i bitne podatke, to je veća potreba da sistem bude konzistentan. Dok kod slučaja kada se radi sa velikom količinom podataka gde nema potrebe za velikim detaljisanjem prilikom rada sa tim podacima, daje se mnogo veći akcenat na dostupnost sistema. Odluka da li će se više posvećivati pažnja na konzistentnost ili dostupnost u potpunosti zavisi od uslova u kojima sistem radi i strukture podataka.



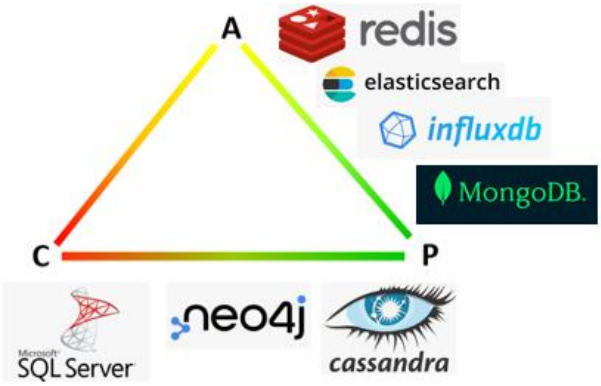
Slika 24. CAP trougao.

Često se postavlja pitanje da li je moguće napraviti distribuiran sistem koji je i konzistentan i dostupan. Iako je moguće delimično izbeći pojavu particionisanja putem omogućavanja većeg broja veza između elemenata sistema, pojava particionisanja je veoma nepredvidljiva i bez pripreme za taj slučaj moguća je pojava velikih problema u distribuiranim sistemima. Naravno, konzistentnost i dostupnost je moguća u ne distribuiranim sistemima bez velikih problema.

CAP teorema predstavlja korisnu klasifikaciju u ranim fazama projektovanja sistema, nakon ustanovljavanja bitnih karakteristika sistema, moguće je uz pomoć CAP teoreme odabrati idealnu bazu podataka za projektovani softverski sistem.

U nastavku će biti sagledane karakteristike svih baza podataka koje su bile razmatrane u radu iz ugla CAP Teoreme, gde će se za svaku bazu podataka navesti da li pripada CP (Konzistentni sa tolerancijom na particionisanje) ili AP (Dostupni sa tolerancijom na particionisanje)

- SQL serveri – **CP** – relacione baze podataka posvećuju veliki nivo pažnje da su svi podaci koji se unose u sistem ispravni i u skladu sa ograničenjima, dodavanjem na to i poštovanje ACID principa, može se primetiti veliki fokus na konzistenciju.
- MongoDB – **AP** – MongoDB omogućuje efektivne replikacije podataka putem svoje distribuirane arhitekture i *Sharding* procesa koji garantuje eventualnu konzistenciju podataka, međutim MongoDB je projekatovan da podrži veliki protok podataka te je mnogo veća pažnja posvećena na dostupnost prilikom rada sa MongoDB bazom podataka.
- Redis – **AP** – Redis je baza podataka koja posvećuje veliku pažnju na efikasno skladištenje velike količine jednostavnih podataka, te je krucijalno omogućiti brzo prihvatanje i rukovanje sa tim podacima.
- InfluxDB – **AP** – česta pojava kod rada sa Time-Series podataka jeste da je njihova količina veoma velika, kao i brzina kojom se ti podaci unose u sistem. Zbog ove prirode podataka, Time-Series baze podataka moraju da omogućavaju veliku dostupnost. InfluxDB postiže eventualnu konzistenciju putem asinhrone repliakcije podataka kroz elemente u sistemu.
- Neo4J – **CP** – Neo4J podržava ACID principe, samim time konzistencija mora biti zagarantovana. Iako je Neo4J podešen da bude tolerantan na particionisanje u sklopu jednog cluster-a, postoje ograničenja prilikom rada u veoma distribuiranim sistemima.
- Elasticsearch – **AP** – Elasticsearch je specijalizovan za rad sa velikom količinom podataka, što uključuje obradu i prihvatanje tih podataka. Elasticsearch koristi metodu sharding-a da replicira podatke na druge čvorove u sistemu što omogućuje visoku dostupnost i eventualnu konzistenciju.
- Cassandra – **CP** – Cassandra je specifičan slučaj iz pogleda CAP teoreme zbog svoje konfigurabilne prirode. Po predviđenim konfiguracijama, Cassandra daje veći akcenat na konzistentnost podataka, pogotovo sa podešavanjem striktne strukture podataka. Međutim moguće je konfiguranje sistema da se daje veći akcenat na dostupnost nego na konzistenciju na razne načine, kao što su smanjenje replikacionog faktora neke tabele. Tako da Cassandra može da adaptira svoje karakteristike u skladu sa potrebama sistema i zahtevane konzistencije ili dostupnosti.



Slika 25. CAP trougao sa dodatim bazama podataka koje su razmatrane u radu.