Diplomski rad

Tema: Poređenje relacionih i nerelacionih baza podataka

# 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.

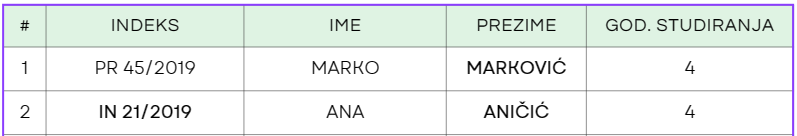
Prve softverske realizacije baza podataka su se pojavile 1960-tih godina, kada je Charles Benchman dizajnirao prvi oblik softverske baze podataka, nazvan Integrated Data Store. Nakon toga, 1970-tih, kompanije IBM je razvila Information Management System. Ova dva tipa baza podataka su bile navigacione, 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 njegova povezanost sa drugim podacima. Prve relacione baze podataka su bile INGRES, čija je ideja osmišljena i realizovana na Berklez 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.

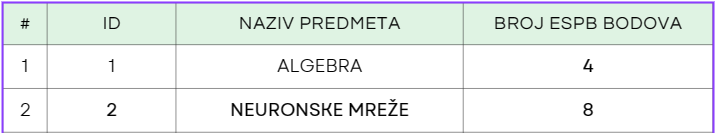
# Relacione baze podataka

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.

# Relacioni model podataka

Jedan od uslova jeste poštovanje **relacionog modela podataka**. Osnovna ideja relacionog modela podataka 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.).

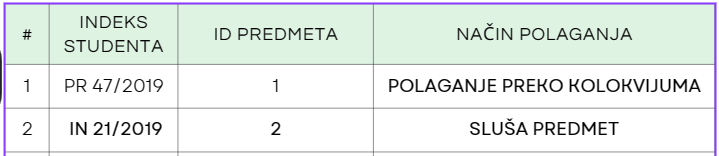
*Slika 1. Primer tabele Studenata*

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.

*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.

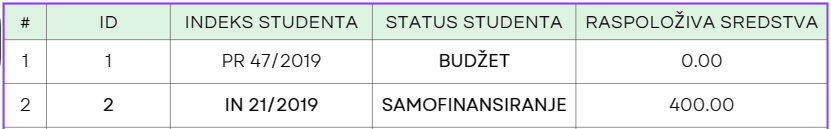
Veze između entiteta u modelu se mogu realizovati na 2 načina:

1. Pomoću zasebnih tabela

*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**.

1. Pomoću prostiranja primarnog ključa



*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, funkcionalnostima koje softverski sistem treba da izvršava i drugim faktorima bitnih pri dizajniranju softverskog sistema.

# 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.

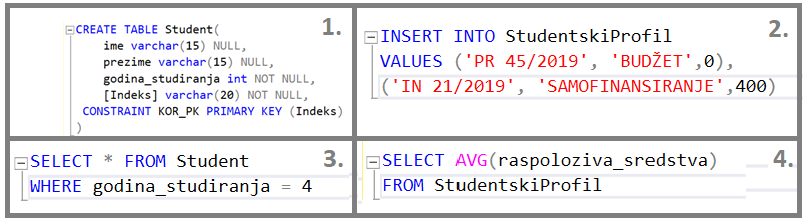
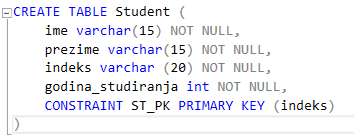
Ispunjavanjem ACID principa poboljšava se integritet podataka koje koristimo u softverskom sistemu, dobijaju se bezbednija ažuriranja podataka i obezbeđuje se pouzdan izvor podataka.

# SQL

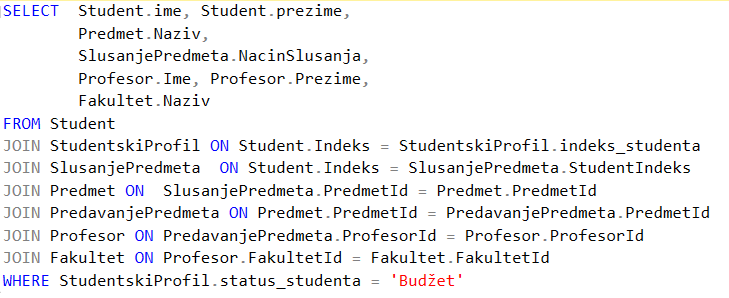
SQL (Structured Query Language) je standardizovan programski jezik za rad sa relacionim bazama podataka. Nastao je 1970-tih i podržan je od strane svih aktuelnih sistema za upravljanje relacionim bazama podataka.

*„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.

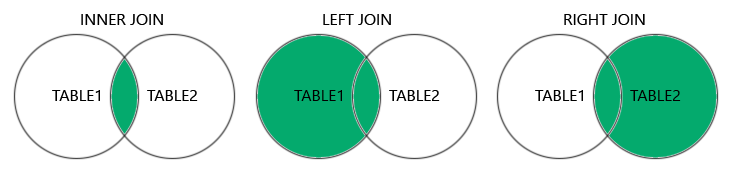
*Slika 5. SQL upiti za (1.) kreiranje tabele, (2.) unos podataka, (3.) pretragu podataka i (4.) obradu podataka.*

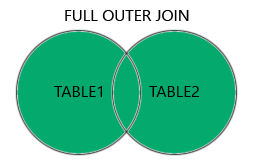
Razumevanjem sintakse SQL programskog jezika, možemo kreirati kompleksne upite koji uzimaju podatke iz više tabela na osnovu njihovih veza, omogućavajući obradu veće količine podataka u jednom pozivu.

*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.**

# Nerelacione baze podataka

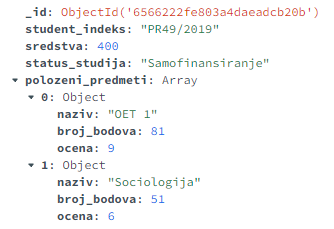
’ Baza podataka se smatra nerelacionom ako ne poštuje pravila koja relacione baze moraju da poštuju. Sva pravila koja su osmišljena za relacione baze podataka služe radi uvođenja standardnog principa funkcionisanja i osiguravanja da pri radu sa relacionim bazama podataka, programer zna tačno koja ograničenja 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 nastale 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.

# Dokument baze podataka

Prvi tip nerelacionih baza podataka koji će biti razmatran je Dokument tip baye 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.

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 ogranišenje 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), 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 Key-Value 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 strukture podataka. Moguće je postojanje veza između entiteta u bazi podataka, ali je to takođe upotpunosti prepušteno programeru da vodi računa.

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 unosi u MongoDB Compass, ovaj upit pronalazi studentske profile sa određenom sumom sredstava i statusom studija, dok drugi upit pretražuje studentske profile sa sumom sredstava većom od navedene.

*Slika 9. primer upita za pretragu studentskih profila*

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

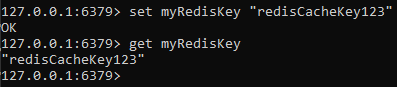
* Amazon DocumentDB
* Cosmos DB
* ArangoDB
* Couchbase Server
* CouchDB

# Key-Value Store

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 i koristiće se u aplikativnom delu rada koji će biti predstavljen u nastavku.

Cilj Redis baze pozadataka, 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.

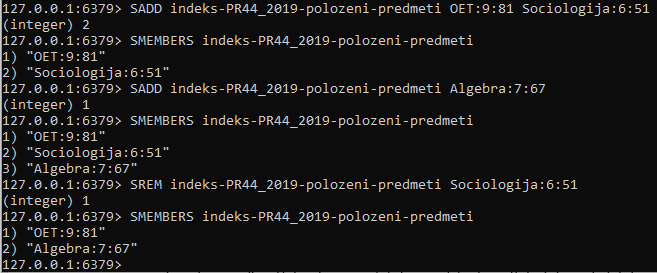
Unos i čitanje podataka se obavlja putem ključeva, gde se pri unosu navede za koji ključ se dodeljuje određena vrednost, a pri čitanju sa zatraži vrednost sa određenim ključem. Primer unosa i čitanja podataka se može videti na sledećoj slici.

*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.

Kao što je napomenuto ranije, nerelacione baze podataka su često projektovane sa konkretnom namenom u planu, Key-Value baze podataka služe kao dobar primer tog koncepta, jer su namenjene da služe primarno za lakše skladištenje velike količine podataka bez fokusa na obradu i kompleksnost podataka.

*Slika 10. primer rada sa kolekcijom položenih predmeta nekog studenta.*

Korisno je napomenuti da 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 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 dizajnirane da funkcionišu u ovakvim sistemima i primunjuje se u razne svrhe kao što su:

* keširanje podataka,
* praćenje sesije korisnika,
* analitika u realnom vremenu,
* 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 raznih struktura podataka u distriburianim sistemima, obezbeđujući brz pristup i skladištenje podataka sa visokim performansama i visokom tolerancijom za visoku učestalost pristupa bazi podataka.

Neki od primera aktuelnih Key-Value baza podataka su:

* Redis
* Riak
* Amazon DynamoDB
* Google Cloud Bigtable
* Azure Cosmos DB
* RocksDB itd.

# 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.

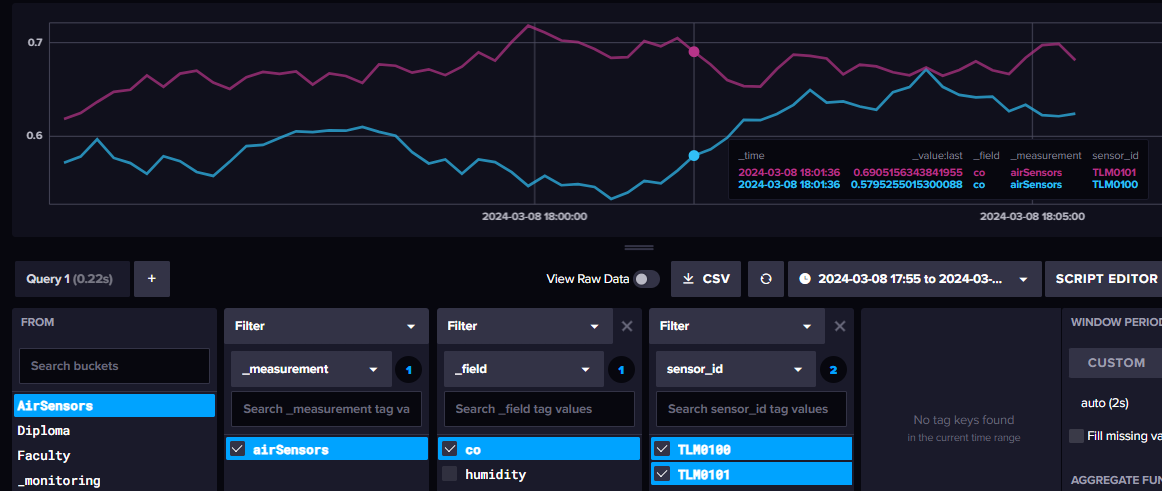
Još jedan ključan aspekat, tokom rada sa Time Series podacima, jeste rukovanje sa obrađenim podacima. Sa obzirom da u nekom sistemu mogu da postoje veliki broj raznih senzora i da njihova učestalost vršenja merenja može biti veoma velika, baza podataka mora da rukuje sa vrlo velikom količinom podataka. Nakon obrade tih podataka, sama njihova vrednost može da bude manje bitna u kasnijem periodu, te je potrebno te podatke izbrisati iz baze podataka. Time Series, za razliku od relacionih i drugih tipova baza podataka, imaju ovu opciju ugrađenu i omogućenu u startu.

InfluxDB baza podataka grupiše kolekcije podataka u “korpe”, odnosno Bucket-e, gde svaka korpa može da ima više različitih tipova podataka. 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).

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.

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

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 namenjeni da sadrže podatke po kojima će se pretraživati podaci i oni su indeksirani radi bolje pretrage a polja sadrže neke bitne vrednosti koje će se analizirati. U primeru sa vazdušnim senzorima tag-ovi predstavljaju identifikacioni parametar senzora a među poljima su količina ugljen-dioksida, vlažnost vazduha i temperatura.

Samoj bazi se može pristupiti na više načina, jedan od najčešćih načina je pristup adresi <http://localhost:8086> na kojoj se nalazi kontrolna tabela 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.*

InfuxDB, kao i ostale Time-Series baze podataka su idealne za rad sa velikom količinom podataka i njihovu analizu ali fokusom na ove parametre gubi sposobnost za kompleksne upite koji pretražuju različite vrste merenja u isto vreme, stoga nisu pogodne za rukovanje sa strukturama koje imaju visoke kardinalitete. Takođe ukoliko kompleksnost podataka postane prevelika moguće je primetiti visoko operećenje procesorkse jedinice računara i može zahtevati veliku količinu memorijskog prostora.

Primeri Time-Series baza podataka pored InfluxDB:

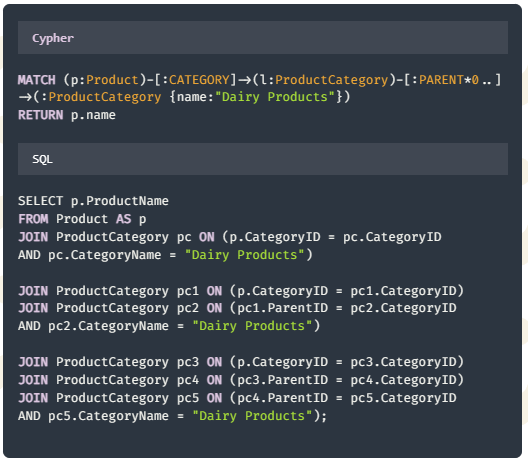
* QuestDB
* Amazon Timestream
* Apache Druid
* Prometheus

# Graf baze podataka

Graf baze podataka imaju ključnu karakteristiku, iako se smatraju nerelacionim jer ne koriste SQL i relacioni model, njihov 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). Vezama je posvećena velika pažnja, jer pored označavanja da dva entiteta imaju neku konekciju, veze u sebi mogu i da nose neke podatke o toj konekciji.

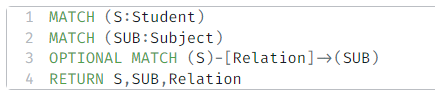
Graf baze podataka imaju slične aspekte kao relacione baze podataka po pitanju veza između entiteta, ali Graph baze podataka daju mnogo veći akcenat na njih i mogu da budu mnogo efikasnije u sistemima gde struktura podataka takva da su entiteti često povezani i to sa potencijalno velikim brojem drugih 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 | Brze upiti i brzi odzivi |
| Preporučena namena | Sistem baziran na transakcijama sa jednostavnijim vezama između entiteta | Sistemi sa veoma povezanim entitetima |

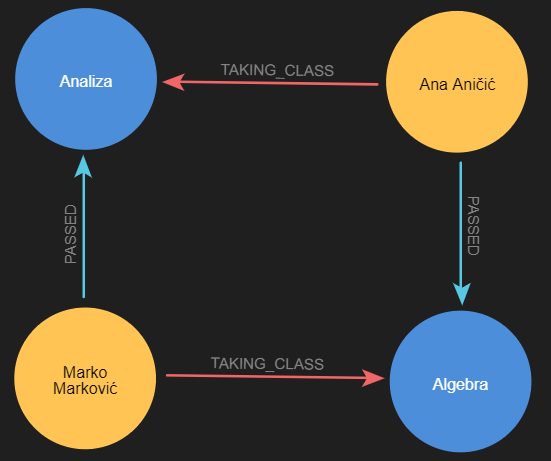
*Tabela 1. pregled odnosa nekih karakteristika između relacionih i graf baza podataka.*

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. Na slici pored se može videti razlika u količini koda potrebnoj da se izvrši ista operacija u SQL-u i u Cypher jeziku.

*Slika 11. Primer Cypher jezika.*

Graf baze podataka posvećuju veliku pažnju na vizuelizaciju podataka, omogućavajući jednostavan i pregledan prikaz potrebnih elemenata i njihovih veza. Prilikom izvršavanja bilo kakvog upita koji pretražuje i dobavlja podatke, svi nađeni podaci se prikazuju u vidu grafa u sklopu aplikacije putem koje se upravlja sa bazom podataka. Na slici 12. 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 13. se nalazi vizuelizacija pridobijenih podataka.

*Slika 12. Cypher upit za pregled studenata, predmeta i njihovih veza.*



*Slika 13. Graf studenata, predmeta i njihovih veza.*

Od svih tipova baza podataka koje će se razmatrati, graf baze podataka su jedine koje pored mogućnosti čuvanja i obrade podataka imaju mogućnost vizualizacije podataka i njihovih veza. Zbog toga predstavljaju jedan veoma koristan alat za programere i korisnike sistema kojima je potrebno da pregledaju stanje podataka u sistemu.

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.

# 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 bih mogla lakše da izvrši pretragu nad njima.

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, pored pretrage, 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 pretragu podataka na način prilagođen njihovim potrebama.

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 komande i upiti nad bazom podataka, dok Beats i Logstash pomažu pri prikupljanju podataka i vođenja evidencije nad njima.

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.), unos objekta 2.) i 3.), brisanje objekta sa određenim identifikacionim poljem, pretraga objekata bez parametara 5.) i sa parametrima 6.).*

Na slici 13., 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 se koristi 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 POST poziv prilikom unosa objekta potrebno je navesti vrednost identifikacionog polja u sklopu HTTP poziva, dok PUT poziv omogućuje da Elasticsearch tom objektu, ukoliko nije navedeno u HTTP pozivu, automatski dodeli unikatnu vrednost identifikacionog polja. Pretraga podataka po tekstu može biti po celom tekstu koji mora da se pojavi u nekom polju objekta ili po delimičnom tekstu, gde znak zvezdice (\*) označava da nešto može biti pre ili posle traženog teksta.

Razumevanjem fukncionisanja Elasticsearch baze podatak i strukture HTTP poziva, moguće je vršenje raznih operacija nad Elasticsearch bazi podataka, a upoznavanjem sa drugim alatima u Elastic Stack-u moguća je efikasna vizuelizacija podataka i integracija Elasticsearch baze podataka u razne druge softverske sisteme. 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.

# Columbnar baze podataka

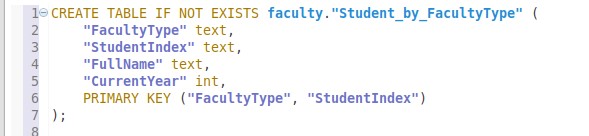
Poslednji tip nerelacione baze podataka koji će se razmatrati jeste Columbnar tip. Ovaj tip baza podataka ima specifičan način rada sa podacima gde je veći fokus posvećen na kolone u tabelama umesto na redove. Ovim pristupom olakšavamo čuvanje i obradu velike količine podataka.

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. 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. Relacione baze podataka koriste jedan sistem za upravljanje bazama podataka na kojem su definisane razne baze podataka i unutar njih tabele. 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.

Slika 11. CQL upit za kreiranje Keyspace-a Fakultet.

Na slici 11. 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. U ovom slučaju 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.

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.

Slika 12. CQL upit za kreiranje tabele Student.

Na slici 12. 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 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.

Potrebno je naglasiti da je strogo preporučeno da pretraga podataka iz određene tabele bude samo po vrednosti particionog ključa i ključa za grupisanje te tabele. 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.

Slika 12. CQL upit za kreiranje tabele Student.

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. Cassandra 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 oslobađaju korisnika od ograničenja relacionih baza podataka da bih sveobuhvatno obezbedile idealan sistem za određenu svrhu.

Neke od drugih često korišćenih baza podataka Columbnar tipa su:

* ScyllaDB
* Apache Kudu
* Snowflake
* Google BigQuery