# NoSQL baze podataka kao potpora web aplikacijama

**Škender, Domagoj**

**Sveučilište u Zagrebu**

**Ekonomski fakultet**

**Diplomski sveučilišni studij Menadžerska informatika**

# NOSQL BAZE PODATAKA KAO POTPORA WEB APLIKACIJAMA

**Diplomski rad**

**Domagoj Škender**

**Zagreb, rujan, 2021.**

**Sveučilište u Zagrebu**

**Ekonomski fakultet**

**Diplomski sveučilišni studij Menadžerska informatika**

# NoSQL BAZE PODATAKA KAO POTPORA WEB APLIKACIJAMA

# NoSQL DATABASES SUPPORTING WEB APPLICATIONS

**Diplomski rad**

**Domagoj Škender, 0067482477**

**Mentor: prof. dr. sc. Katarina Ćurko**

**Zagreb, rujan, 2021.**

_____

(ime i prezime studenta/ice)

# IZJAVA O AKADEMSKOJ ČESTITOSTI

Izjavljujem i svojim potpisom potvrđujem da je _____ (vrsta rada) isključivo rezultat mog vlastitog rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu, a što pokazuju korištene bilješke i bibliografija. Izjavljujem da nijedan dio rada nije napisan na nedozvoljen način, odnosno da je prepisan iz necitiranog rada, te da nijedan dio rada ne krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

Student/ica:

U Zagrebu, _____ _____

ZAHVALA

Zahvaljujem se svojoj mentorici prof. dr. sc. Katarini Ćurko, koja mi je kroz kolegij Upravljanje podacima i kroz veliku pristupačnost pobudila interes koji će dvije godine kasnije rezultirati potpunom karijernom promjenom i nalaženjem istinskog profesionalnog zadovoljstva i ostvarenosti. Osim toga, vjerovala je u mene i gurnula me u smjeru uzimanja izazovne i nimalo trivijalne teme, kroz koju sam naučio nevjerojatno mnogo, a opet mi davši potpunu slobodu oko detalja.

Posebno mjesto u ovim zahvala mora imati i supruga Justina, koja mi je bila nevjerojatna podrška kroz pisanje ovog rada i preuzela na sebe disproporcionalnu brigu za našu kćerkicu Ritu tijekom dugih noći istraživanja, pisanja i kodiranja.

Netradicionalno za ovakve zahvale, moram imenovati i prijatelja Maria Novaka, čovjeka koji osim što me je naučio razmišljati kao softverski arhitekt, je i upalio tu iskru u meni i gurao me kada mi se činilo da u ovim godinama, s jednom nogom u potpuno drugačijoj karijeri, više ne mogu učiniti takvu tranziciju i da IT znanja ima jednostavno previše za apsolvirati u ovoj fazi života.

Na kraju želim se zahvaliti svim kolegama i profesorima na EFZG-u koji su mi pomogli učiniti iskustvo studiranja, koliko god klišeizirano zvučalo, najljepšim razdobljem u dosadašnjem životu.

SAŽETAK I KLJUČNE RIJEČI

Cilj ovog diplomskog rada je pružiti organiziran i sustavan pregled trenutno dostupnih tehnologija i paradigmi baza podataka te čitatelju približiti funkcionalnosti, tehnologije i načine modeliranja podataka NoSQL baza. Tradicionalne relacijske SQL baze podataka biti će u upotrebi još dugi niz godina, posebice kao "kralježnice" poslovnih informacijskih sustava. Ipak, paralelno (posebice od početka milenija) se razvijaju nove tehnologije baza podataka različitih primarnih namjena i implementacija, najčešće okrenute potpori web aplikacijama u klijentsko – poslužiteljskim arhitekturama.

Rad je baziran na istraživanju što aktualnije dostupne literature i službenih dokumentacija obrađenih tehnologija, a teži ostvariti balans poslovne i tehničke tematike, pa se te tehnologije promatraju iz poslovnog kuta – analitike podataka, te iz tehničkog kuta – kako baze pružaju podatkovnu potporu aplikacijskom softveru. Nakon pojašnjenja nekih osnovnih (poslovnih i tehničkih) pojmova nužnih za dublje razumijevanje tematike, poput upravljanja podacima, baza podataka, poslovne inteligencije i Velikih podataka, kreće se u sustavni pregled osnovnih tipova baza podataka. Konsenzus programerske zajednice i sve dostupne literature jest da se osim tradicionalnih, relacijskih, SQL baza podataka koje vuku korijene još s kraja 70-ih, danas iskristaliziralo još četiri osnovna tipa tzv. NoSQL baza: ključ-vrijednost sustavi, graf-baze, stupčane baze te dokumentne baze.

Srž rada je u približavanju načina funkcioniranja, modela podataka te prednosti i mana osnovnih tehnologija baza podataka, s primjerima praktičnog korištenja. Istraživanje se temeljilo na razvoju vlastite prototipne aplikacije (lista zadataka, tzv. "To-do app") koja kao svoju podatkovnu potporu ima NoSQL rješenje. Istraživanje daje uvid u fleksibilnost i mogućnosti nerelacijskih baza podataka, a općenitije u prednosti aplikacija korisnica takvih baza u klijentsko – poslužiteljskoj arhitekturi (web aplikacija).

*Ključne riječi: Baze podataka, SQL baze, NoSQL baze, Web aplikacije, Web razvoj*

SUMMARY AND KEYWORDS

The aim of this thesis is to provide an organized and systematic view of currently available database technologies and paradigms. Special focus is given to helping readers understand the functionalities, technologies and data modelling techniques of NoSQL databases. Traditional relational SQL databases will be in use, especially as the backbone of business IT system for many years to come. That being said, in the 21st century we're seeing an advent of new database technologies with differing implementations and purposes, primarily aimed towards supporting web applications in client – server architectures.

This paper is primarily based on researching the most current literature and the official documentation of technologies covered, and aims to achieve a balance between a business – oriented and a technical approach so the technologies are viewed through a business lens – data analytics and from a technical standpoint – how databases support applications. Some clarification of basic (business and technical) terms is necessary: data management, databases, business intelligence and big data, which is followed by a systematic overview of database types. Both the available literature and the developer community seem to reach a consensus about there being four basic types of NoSQL databases (in addition to traditional, relational SQL databases that trace their roots all the way to the late 1970s). These four types are: document, key – value, wide – column and graph databases.

The essence of this thesis is in creating an understanding of operating principles, data models and the pros and cons of the basic types of database technologies, with examples of practical usage. The research part of the paper is based on developing a prototype "To – do" application which uses a NoSQL solution as its data support. The research gives insight into the flexibility and possibilities of non – relational databases supporting web applications, and in a more general view, of the advantages of applications in a client – server architecture where the end – user accesses the application from a web browser.

*Keywords: Databases, SQL databases, NoSQL databases, Web applications, Web development*

# Sadržaj

# 1. Uvod

## 1.1. Predmet i cilj rada

Predmet rada je jezgrovito i koncizno obraditi temu baza podataka kao potpore web aplikacijama u klijentsko poslužiteljskoj arhitekturi, s daleko najvećim fokusom, kao što je vidljivo iz naslova, na novija rješenja u vidu ne-relacijskih, tzv. NoSQL baza. Kako bi se tematika mogla razumjeti, obradit će se nužni osnovni pojmovi vezani uz upravljanje podacima, baze podataka, analizu podataka i skladišta podataka, te će se približiti teme poslovne inteligencije i Velikih podataka koje su u ovakvom kontekstu neizbježne. Nakon kratkog osvrta na tehnologiju relacijskih baza (koja je i uz postojanje ovih novijih rješenja i dalje uvelike prisutna u poslovnim informacijskim sustavima te će još dugo biti), detaljnije će se obrađivati same NoSQL baze, specifičnosti, prednosti i mane svake od osnovne četiri vrste, te njihove poslovne i tehničke primjene. Nužan je i kratak pregled internetskih tehnologija (tehnologija esencijalnih za izgradnju web aplikacija) radi boljeg razumijevanja aplikativnog softvera kojemu će te baze služiti kao podatkovna potpora.

Cilj rada leži u pružanju jedinstvenog i sistematiziranog pregleda tehnologija koje se upotrebljavaju za izgradnju baza podataka, s konzistentnom strukturom radi što lakšeg praćenja. Sam sadržaj rada i naslovi poglavlja odaju kako ovaj rad teži ostvarivanju što veće ravnoteže između tehničkog i poslovnog pogleda na obrađivanu tematiku, tj. između "kako" i "zašto" (se grade baze podataka i razvijaju aplikacije).

## 1.2. Izvori podataka i metode prikupljanja

U ovom diplomskom radu korišteno je nekoliko metoda istraživanja. Analizirana je literatura u vidu znanstvenih i stručnih knjiga i članaka te je zbog dinamike promjena u promatranom području pridodana posebna pažnja kako bi literatura bila što aktualnija. Također, zbog prirode obrađene tematike pridodan je nešto veći fokus internetskim izvorima te službenim dokumentacijama promatranih tehnologija. Na temelju prikupljenih činjenica stvorena je sistematizirana teorijska podloga metodama deskripcije i kompilacije. Za stvaranje dubljeg uvida u područje istraživanja korištena je metoda razvoja softvera (prototipiranje) te je stvoren funkcionalni prototip aplikacije čija su svojstva (s korisničke i s podatkovne strane) analizirana.

Naposljetku je korištena metoda sinestezije kojom je iz prikupljenih podataka čitavoga rada deriviran jedinstveni zaključak.

### 1.3. Sadržaj i struktura rada

Rad je, nakon ovog uvoda, podijeljen na tri velike teoretske cjeline te jednu praktičnu cjelinu, tj. istraživanje, nakon kojih slijedi autorov zaključak kao zasebno poglavlje te prilozi.

Prva cjelina sadrži 6 poglavlja i daje pojmovno određenje najvažnijih koncepata vezanih uz podatke: *Podaci i informacije, upravljanje podacima i informacijama, baza podataka, podatkovna analitika, poslovna inteligencija i skladište podataka*, *Veliki podaci – primjena i problemi* te *modeliranje i skaliranje baza podataka.*

Druga cjelina, tzv. "meso" rada dublje ulazi u tematiku baza podataka, obrađuje ukratko temu klasičnih relacijskih baza te, nakon uvoda u koncept ne-relacijskih (NoSQL) baza, daje sustavan pregled svakog od njihovih 4 osnovnih tipova: *Ključ-vrijednost sustavi*, *graf-baze*, *stupčane baze, dokumentne baze*.

Treća cjelina, najkraća, sastoji se od osnovnog, jezgrovitog pregleda tehnologija potrebnih za izgradnju web aplikacija.

Četvrta cjelina je praktični dio rada, empirijsko istraživanje u kojem se putem razvoja softvera istražuje ponašanje izabranog NoSQL modela podataka kao potpore prototipnoj web aplikaciji.

Posljednje, izdvojeno, poglavlje rada čini autorov zaključak na temelju provedenih teorijskih i praktičnih analiza i istraživanja.

## 2. Najvažniji pojmovi vezani uz podatke u poslovanju i informacijskim tehnologijama

### 2.1. Podaci i informacije

"Podaci su nova nafta!" - sveprisutna je mantra u poslovnom svijetu koja se najčešće pripisuje britanskom matematičaru Cliveu Humbyju (2006), inače autoru programa lojalnosti velikog britanskog maloprodajnog lanca Tesco. Iako je Humby daleko od izumitelja programa

lojalnosti u maloprodaji, bio je ključan u izgradnji takvog programa baziranog na skladištu podataka, tehnologiji koja je Tescu osim očitih psiholoških prednosti (percepcije potrošača) donijela niz u to vrijeme revolucionarnih pogodnosti poput ogromne količine podataka o svojim kupcima i njihovim navikama, mogućnost prilagođavanja marketinške strategije svakom pojedinačnom kupcu (što je u dugom roku mnogo jeftinije od masovnih strategija s niskim stopama izazivanja željenog efekta) te moć praćenja prodajnih trendova u praktički stvarnom vremenu, na licu mjesta (dućanu)[1].

Bez (kvalitetne) podatkovne potpore, nemoguće je poslovati, ali gledano šire od same komercijalne djelatnosti, nemoguće je provoditi ikakav složeniji pothvat, od prijateljske badminton lige do administracije kompleksne institucije poput Sveučilišta u Zagrebu i svega što to obuhvaća (praćenje upisa studenata, plaćanje participacija, upisi ocjena, vođenje prisustva na nastavi, plaćanje osoblja, nabava itd.). Coronel i Morris (2019) postavljaju zanimljiv misaoni eksperiment čitateljima, pozivajući ih da pokušaju zamisliti kako bi bilo voditi neki oblik poslovanja bez da znaju tko su mu kupci, koje proizvode prodaju, tko su im zaposlenici, tko im duguje novac te kome oni duguju novac. Nastavljaju naglašavajući važnost držanja svih ovih podataka i mnogih drugih, te činjenja istih dostupnima donositeljima odluka. Zaključuju kako je primarna funkcija poslovno-informacijskih sustava pomoći poduzećima koristiti informacije kao organizacijski resurs[2].

Zbog navedenih razloga, važno je, prije ulaženja u tehničke i detaljnije poslovne aspekte baza podataka, reći ponešto o samim konceptima podataka i informacija.

Varga (2014) definira podatak kao "skup prepoznatljivih znakova, odnosa simbola na nekom mediju, npr. papiru, filmu, magnetskom ili kojem drugom mediju". U nastavku piše kako "pomoću podataka zapisujemo činjenice, pritom ne razmatramo ni njihovu interpretaciju ni njihov kontekst"[3]. Broj 5 je primjer podatka, koji, ovisno o kontekstu, može označavati ocjenu diplomskog rada, broj golova u sezoni nekog stopera u engleskoj Premier Ligi i još štošta drugo. Riječ "Zagreb" može označavati mjesto rođenja neke osobe, sjedište tvrtke ili, jasno, prikazivati obilježje glavnoga grada za Republiku Hrvatsku. Koristeći neki od programa za obradu teksta poput Windows Notepada, moguće je otvoriti datoteku sa više stotina tisuća pojedinačnih podataka – cijelih brojeva između (uključujući) 0 i 255, koji se van konteksta čine

[1] Humby, C., Hunt, T. (2004) Scoring Points - How Tesco Is Winning Customer Loyalty, Kogan Page Ltd, London, str. 11-12
[2] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 3
[3] Varga M. (2014) Upravljanje podacima, Element, Zagreb, str. 2

besmislenima, no ustvari se radi o slici u popularnom JPEG formatu i ti podaci predstavljaju boje pojedinačnih piksela na ekranu.

Upravo u davanju konteksta i interpretaciji podataka leži objašnjenje koncepta informacije, koju Varga (2014) definira kao "činjenicu s određenim značenjem" čija je "temeljna svrha da pomogne pri donošenju odluke"[4]. Pri donošenju odluka potrebne su kvalitetne informacije, čija su nužna obilježja **točnost** (ispravno opisivanje stvarnoga stanja), **potpunost** (potpuno opisivanje stvarnoga stanja bez sakrivanja nepovoljnih činjenica), **primjerenost** (odgovara kontekstu i potrebama osobe koja je prima), **pravovremenost** (dobivena na vrijeme)[5].

Kada se informacijama i podacima pridodaju ekspertne procjene, vještine i iskustvo, dobije se koncept znanja, koje je ključ poslovnog i drugog uspjeha. Ono može biti kodificirano i zapisano u raznim priručnicima, udžbenicima i bazama znanja, a može biti i na većoj razini apstrakcije, teško za definirati i nalaziti se isključivo u glavama eksperata.

Izuzetno je važno u ovom trenutku naglasiti podjelu digitalnih podataka prema razini strukturiranosti. U sklopu poglavlja o bazama podataka ova tematika će biti obrađena detaljnije, te je za sada dovoljno uvesti sljedeću podjelu:

- *Strukturirani podaci* su podaci prikazani u strogom formatu, kao što se pojavljuju u relacijskoj bazi podataka. Primjerice, svaki zapis u tablici "Zaposlenici" neke školske/testne baze podataka "Kompanija" slijedi identičan format kao svi ostali zapisi: Ime, srednji inicijal, prezime, OIB, datum rođenja, adresa, spol, plaća, odjel[6].
- *Slabo strukturirani* ili *polu-strukturirani podaci* su podaci koji mogu imati nekakvu strukturu, neki od atributa mogu se pojavljivati kod više entiteta, dok drugi atributi postoje u nekoliko ili čak jednom entitetu. Dodatni atributi mogu se uvoditi prema potrebi, bez strogog pridržavanja nekoj predefiniraj shemi. Primjer korištenja polu-strukturiranih podataka je kolekcija bibliografskih referenci za akademski članak, gdje će neki od zapisa opisivati reference knjiga (s podacima o izdavaču, godini izdanja itd.), dok će internetski izvori sadržavati drugi set podataka (internetsku poveznicu, ime web stranice), a biti će i zajedničkih atributa poput naslova i godine[7].
- *Nestrukturirani podaci* su "objekti kao zvučne sekvence, nepokretne slike (fotografije), pokretne slike ili video sekvence, pa i sam tekst. Podaci se jednog objekta (slike,

[4] Varga M. (2014) Upravljanje podacima, Element, Zagreb, str. 3
[5] Varga M. (2014) Upravljanje podacima, Element, Zagreb, str. 5
[6] Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow, str. 162, 426
[7] Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow, str. 426-427

sekvence) češće pohranjuju u jednoj samostalnoj datoteci, a rjeđe kao jedan zapis datoteke ili baze podataka" (Varga, 2014)[8]. Elmasri i Navathe (2016) u ovu kategoriju dodaju i web stranice kodirane u HTML jeziku[9].

## 2.2. Upravljanje podacima i informacijama

U prethodnom poglavlju definirani su koncepti podataka i informacija te su oni označeni kao vitalni resursi za funkcioniranje svake ljudske institucije, pa je shodno tome njima, kao i svakim drugim resursom (poput sirovine, potrošnog materijala, zgrada, vozila, novca i drugih financijskih instrumenata…) potrebno efektivno i efikasno upravljati. O toj temi napisane su nebrojene tisuće stranica, no za zamišljeni obuhvat ovoga rada, adekvatno je definirati koncept upravljanja podacima, njegove konkretne ciljeve te osnovna područja aktivnosti koja upravljanje podacima obuhvaća. Za tu svrhu, možda se najbolje poslužiti knjigom Data Management Body of Knowledge organizacije DAMA International. DAMA International je neprofitna i nezavisna međunarodna udruga tehničkih i poslovnih profesionalaca koja se bavi razvojem i propagacijom koncepata i najboljih praksi upravljanja podacima i informacijama[10].

DAMA (2017) tako upravljanje podacima definira kao "razvoj, izvršenje i nadzor planova, politika, programa i praksi koje stvaraju, kontroliraju, štite i povećavaju vrijednost podatkovne i informacijske imovine kroz čitav njen životni ciklus"[11]. Upravljanje podacima je multidisciplinarna aktivnost, ravnomjerni spoj tehničkog i poslovnog, u kojemu sudjeluje širok spektar osoblja, od programera, administratora baza podataka i mrežnih inženjera do poslovnih skrbnika podataka (*Data Steward*) i glavnih informacijskih menadžera (*CIO – Chief Information Officer*)[12].

Unutar neke organizacije, osnovni ciljevi upravljanja podacima uključuju:

- Razumijevanje i podrška informacijskim potrebama organizacije i njenih ključnih sudionika, uključujući kupce, zaposlenike i poslovne partnere
- Stvaranje, spremanje, zaštita i osiguravanje integriteta podatkovne imovine
- Osiguravanje kvalitete podataka i informacija

[8]Varga M. (2014) Upravljanje podacima, Element, Zagreb, str. 12
[9] Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow, str. 428
[10] DAMA Mission, Vision, Purpose and Goals. Dostupno na: https://www.dama.org/cpages/mission-vision-purpose-and-goals [12.11.2020.]
[11] DAMA International (2018) Data Management Body of Knowledge, Technics Publications, Basking Ridge, str. 17
[12] Varga M. (2014) Upravljanje podacima, Element, Zagreb, str. 172

- Osiguravanje privatnosti i povjerljivosti podataka sudionika
- Prevencija neautoriziranog i neprimjerenog pristupa, manipulacije ili korištenja podataka i informacija
- Osiguravanje da se podaci mogu efektivno koristiti kako bi dodali vrijednost organizaciji[13]

Kao što je navedeno, upravljanje podacima obuhvaća golem broj poslovnih i tehničkih aktivnosti (između kojih granice mogu biti mutne te koje se presijecaju s drugim organizacijskim funkcijama – budući da upravljanje podacima nije izolirani organizacijski otok u nekom poduzeću, već ostvaruje konstantnu dvostranu komunikaciju s raznim aspektima poslovanja). DAMA (2017) navodi 11 skupina, funkcija gdje pripadaju sve aktivnosti upravljanja podacima[14]: Vladanje podacima, upravljanje arhitekturom podataka, modeliranje i dizajn podataka, skladištenje podataka i operacije, upravljanje sigurnošću podataka, integracija i interoperabilnost podataka, upravljanje dokumentima i sadržajima, upravljanje referentnim i matičnim podacima, skladištenje podataka i poslovna inteligencija, upravljanje metapodacima te upravljanje kvalitetom podataka.

## 2.3. Baza podataka

"*Baza podataka je kolekcija povezanih podataka – zabilježenih činjenica s implicitnim značenjem*" (Elmasri, Navathe, 2016)[15].

U širem smislu, baza podataka je svaki organizirani alat koji sprema podatke i informacije kojima se može pristupiti na efektivan i efikasan način kada za time nastane potreba. Isprintani telefonski imenik koji sadrži imena, telefonske brojeve i adrese rezidenata nekog područja primitivni je oblik baze podataka[16]. Ručni sustavi bazirani na papirima organiziranim po registratorima te dalje po ormarima služili su veći dio ljudske povijesti kao (jedino) rješenje za repozitorije podataka. Ipak, kako je rasla kompleksnost poslovanja, rasla je i količina podataka

[13] DAMA International (2018) Data Management Body of Knowledge, Technics Publications, Basking Ridge, str. 18
[14] DAMA International (2018) Data Management Body of Knowledge, Technics Publications, Basking Ridge, str. 45-46
[15] Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow, str. 4
[16] Alvaro, F. (2018) SQL: Easy SQL Programming & Database Management for Beginners, Your Step-By-Step Guide To Learning The SQL Database, Amazon, e-izdanje, str. 4

te su se mijenjale potrebe za izvještavanjem. Organizacije su se stoga okrenule računalnoj tehnologiji za pomoć (Coronel, Morris, 2019)[17].

Efikasno upravljanje podacima zahtjeva upotrebu računalne baze podataka. Baza podataka je dijeljenja (između korisnika i/ili aplikativnih softvera) računalna struktura koja u sebi sprema sljedeće:

- Podatke krajnjih korisnika – sirove činjenice od koristi krajnjim korisnicima
- *Metapodatke* – "podatke o podacima", pomoću kojih se upravlja podacima krajnjih korisnika

*Metapodaci* opisuju karakteristike i setove veza između podataka unutar baze podataka. Konkretnije, komponenta metapodataka unutar baze sprema informacije poput imena svakog od podatkovnih elemenata, vrste vrijednosti u kojima se ti elementi nalaze (cijeli ili decimalni brojevi, tekst, datumi…) i smije li neko podatkovno polje biti ostavljeno praznim. Metapodaci dakle pomažu dati kompletniju sliku podataka unutar baze. Zbog koncepta metapodataka, često se može čuti da je baza podataka "kolekcija samo-opisujućih podataka"[18].

### *2.3.1. Sustav za upravljanje bazom podataka (DBMS)*

Sustav za upravljanje bazom podataka (*Database Management System* – u nastavku *DBMS*) set je programa koji upravljaju strukturom baze podataka i kontroliraju pristup podacima spremljenim u bazi. DBMS služi kao posrednik između korisnika i baze. Struktura baze podataka spremljena je u trajnoj memoriji računala kao set datoteka kojima je moguće pristupiti samo pomoću DBMS-a. DBMS prima sve zahtjeve (za prikaze, dodavanje, brisanje i ažuriranje) podataka od aplikacija koje koriste baze i prevodi ih u kompleksne operacije potrebne za ispunjavanje tih zahtjeva – na taj način "spašavajući" krajnjeg korisnika i aplikativni softver od dobrog dijela interne kompleksnosti same baze[19]. Aplikativni softver, na primjer aplikacija za obračun plaća, ili aplikacija za rezerviranja sjedala u nekom kino multipleksu, pisan je u nekom od programskih jezika kao što su Java, C, Python ili JavaScript. Softver se s bazom koja mu služi podatkovna podrška spaja pomoću standarda poznatog kao

[17] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 15
[18] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 6-7
[19] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 7

Open Database Connectivity (ODBC) koji pruža API[20] koji omogućava programima da pozivaju bazu (na kojoj DBMS tada stvara upite)[21].

Neki od popularnih DBMS-ova su MySQL, Oracle, PostgreSQL, Microsoft Access, MariaDB i IBM DB2.

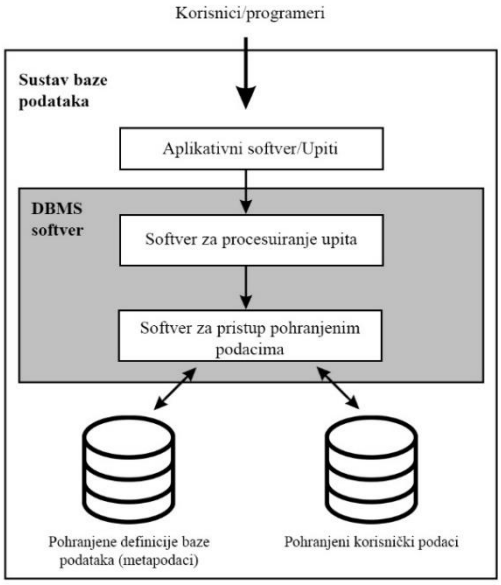
Prema Coronelu i Morrisu (2019)[22], DBMS obavlja veći broj funkcija ključnih za integritet i konzistentnost podataka u bazi, kao što su upravljanje rječnikom podataka (metapodacima), upravljanje performansama kroz raspoređivanje podataka po datotekama (iako korisnik percipira bazu kao jedinstvenu cjelinu), transformacija i prezentacija podataka (npr. prilagodba formata datuma ovisno o korisnikovoj regiji), sigurnost i ograničavanje pristupa ovisno o razini prava korisnika (sam pristup te pristup određenim akcijama poput modificiranja ili brisanja), stvaranje sigurnosnih kopija, gore spomenuto omogućavanje komunikacije s korisničkim softverom, te pružanje pristupa bazi pomoću jezika za upite (*query language*). Jezik za upite je ne proceduralni programski jezik koji omogućuje korisnicima manipulaciju nad podacima (stvaranje, pregledavanje, ažuriranje i brisanje – CRUD – *create, read, update, delete*) pomoću DBMS-a. Najpopularniji takav jezik je SQL (Structured Query Language), razvijen još 1970-ih od strane IBM-a. Svaki od proizvođača DBMS-ova ima vlastiti "dijalekt" SQL-a, no njihove sintakse se razlikuju tek u naprednim i specifičnim funkcijama, dok je osnovna sintaksa industrijski standard.

Budući da se ovaj rad u nastavku intenzivno bavi *upitima* nad bazama te različitim jezicima za upite, poželjno je detaljnije definirati koncept *upita*. Iako u konvencionalnom jeziku izraz ima konotaciju "pitanja", u području baza podataka najčešće se koristi za sve interakcije s bazom – dohvat određenih podataka, izmjene podataka i generiranje izvještaja na temelju podataka[23].

Potrebno je još naglasiti kako većina dodatnih funkcionalnosti koje DBMS pruža glede validacije unosa točnog tipa podataka (npr. određeno polje je zadano da prima samo datum ili samo broj bez decimalnih znakova, ili jednostavno ne može biti bez vrijednosti), ograničenog pristupa određenim korisnicima, sprječavanja brisanja vezanih podataka i sl., može biti implementirano i na razini programskog koda aplikacije koja će bazu koristiti. U praksi taj

[20] API - Application Programming Interface – Aplikacijsko programsko sučelje – računalno sučelje koje omogućava interakciju između različitih programa kroz unaprijed definirane moguće pozive, konvencije, vrste podataka itd. Najjednostavnije rečeno – sučelje koje omogućava jednom računalnom programu pristup/kontrolu na funkcijama drugoga – u ovom primjeru kontrolu sustava za rezervaciju sjedala nad DBMS-om.
[21] Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow, str. 49
22 Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 24-27
[23] Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow, str. 6

izbor ovisi o potrebama projekta, korporativnim praksama te naposljetku o osobnom stilu i preferencijama programera.




*Slika 1: Pojednostavljeni prikaz okoline sustava baze podataka. Izvor: Vlastita izrada autora, prema Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow*

### *2.3.2. Klasifikacija baza podataka*

DBMS može se upotrijebiti za izgradnju različitih tipova baza podataka ovisno o potrebama korisnika, što dovodi i do različitih metoda klasifikacija baza. Primjerice, mogu se klasificirati prema broju podržanih korisnika, fizičkoj lokaciji podataka, vrsti podataka koji se pohranjuju, uporabnoj namjeni podataka i razini strukturiranosti podataka. U jednostavnom softveru za jedno računalo će se pronaći baza podataka koja podržava samo jednog korisnika (tj. aplikaciju) odjednom, što znači da dok korisnik A koristi bazu, upiti od korisnika B i C moraju čekati. Češće se nailazi na tzv. "*multiuser*" baze podataka koje podržavaju veći broj korisnika odjednom. One se granaju pak na relativno ograničene "*workgroup*" baze podataka koje podržavaju do 50 korisnika odjednom te "*enterprise*" baze namijenjene podatkovnoj podršci velikim sustavima (obično po više stotina korisnika odjednom). Geo lokacijski gledano, baza podataka može biti cijela fizički smještena na jednome mjestu pa se tada radi o *centraliziranoj* bazi, dok je baza smještena na više odvojenih lokacija (i samim time i više fizičkih tvrdih diskova) poznata kao *distribuirana* baza. Zbog sve jačeg trenda pružanju usluga skladištenja podataka pojedincima i institucijama, važno je spomenuti i "*cloud*" baze, tj. "baze u oblaku". Podatkovne usluge "u oblaku", poput onih koje pružaju Microsoft Azure i Amazon Web Services omogućuju korisnicima *outsourcing* (izdvajanje) infrastrukture, pa se treća strana, za naknadu, brine o postavljanju i održavanju baze, njenim performansama i kapacitetima. Krajnji

korisnik najčešće u tim slučajevima niti ne zna pozadinsku softversku i hardversku infrastrukturu (kakav tip fizičkih medija se koristi, koji DBMS je izabran i sl.). To je vrlo agilan pristup koji omogućava brzi dokup daljnjih kapaciteta te dostupnost softverske infrastrukture koja bi se nekada postavljala tjednima, u nekoliko minuta. Još jedna podjela jest na baze opće namjene i visokospecijalizirane baze, gledano s aspekta koji će tip podataka biti spreman – širok spektar ili usko specifičan za neku disciplinu poput medicine ili geologije.

Postoje još dvije podjele uže vezane uz područje ovoga rada: prema *razini strukturiranosti podataka* te prema *načinu korištenja* tj. vremenskoj osjetljivosti dohvata informacija.

Podjela prema razini strukturiranosti je među glavnim razlozima za postojanje NoSQL baza i ta tematika detaljno je pokrivena u 3. cjelini. Za sada je dovoljno ponoviti osnovnu podjelu iz poglavlja 2.1. na strukturirane, polu-strukturirane i nestrukturirane podatke.

Najpopularnija klasifikacija baza podataka jest prema načinu korištenja – svakodnevne transakcije ili analitika i donošenje odluka.

1. *Operativna ili transakcijska baza podataka* baza je dizajnirana kao podrška svakodnevnim operacijama organizacije – prodaja proizvoda i usluga, plaćanja, nabava i sl. Takve transakcije dolaze često s velikog broja mjesta i moraju biti obavljene precizno i trenutačno.
2. *Analitička baza* fokusirana je na spremanje povijesnih podataka i poslovne metrike, a namjena joj je pružati podršku za donošenje strateških i taktičkih odluka. Takva baza zove se još i *skladište podataka* (*Data Warehouse*)[24]

## 2.4. Podatkovna analitika, poslovna inteligencija i skladište podataka

U poglavljima 2.1. i 2.2. objašnjeni su koncepti podataka i informacija te njihova važnost za funkcioniranje svake ljudske organizacije. Za informacijske potrebe na najnižim, operativnim razinama poslovanja (koje robe je potrebno naručiti u poslovnicu, koje pristigle račune je potrebno platiti i sl.) nisu potrebne nekakve posebne analize i koriste se podaci iz tekućeg poslovanja koje pružaju transakcijske baze. Menadžeri na srednjim (taktičkim) i najvišim (strateškim) razinama u kompleksnom poslovnom okruženju trebaju sumirane informacije. Zbog sve većih pritisaka modernog poslovanja: globalizacije, širenja tržišta, spajanja i

[24] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 9-11

preuzimanja, razvoja tehnologije i pojačane regulacije, stvorila se potreba za općenitijim i integriranim okvirom potpore odlučivanju nego što bi nudile izolirane poslovne aplikacije poput onih za financije, upravljanje kupcima, ljudske resurse itd. Taj okvir postao je poznat kao poslovna inteligencija (*business intelligence – BI*)[25].

DAMA (2017) poslovnu inteligenciju definira dvojako. Prvo, izraz se odnosi na "vrstu analize podataka usmjerenu ka razumijevanju organizacijskih aktivnosti i prilika, a čiji se rezultati koriste za poboljšanje poslovnog uspjeha. Ako organizacija ispravno ispita vlastite podatke, može dobiti uvid u svoje proizvode, usluge i partnere koji omogućuje donošenje boljih odluka radi ispunjenja strateških ciljeva". Prema DAMA-i, "poslovna inteligencija" odnosi se i na "set tehnologija koje podupiru takvu vrstu analize podataka. Kao evolucija prethodne generacije alata za potporu odlučivanju, BI alati omogućuju upite nad podacima, rudarenje podataka, statističku analizu, izvještavanje, modeliranje scenarija, vizualizaciju podataka te stvaranje digitalnih nadzornih ploča (*dashboarding*)[26]".

Coronel i Morris (2019) naglašavaju kako poslovna inteligencija nije sama po sebi proizvod, već okvir koncepata, praksi, alata i tehnologija koji pomažu poduzećima stvoriti konkurentsku prednost kroz bolje razumijevanje vlastitih kompetencija, stvaranje serija slika stanja kroz vrijeme i identifikaciju prilika. BI pruža okvir za:

- Prikupljanje i spremanje operativnih podataka
- Agregiranje operativnih podataka u podatke namijene potpori odlučivanju
- Analizu podataka za odlučivanje kako bi se generirale informacije
- Prezentiranje tih informacija krajnjim korisnicima kao potporu odlučivanju
- Donošenje poslovnih odluka koje pak generiraju daljnje podatke koji se skupljaju, spremaju itd. (čime se proces ponavlja)
- Praćenje rezultata poslovnih odluka
- Predviđanje budućih ponašanja i ishoda[27]

Podaci kojima se BI služi - podaci za potporu odlučivanju, razlikuju se od operativnih podataka u namjeni, pa samim time i u formatima i strukturi. Operativni podaci obično se nalaze u relacijskim bazama podataka (o kojima će biti više u nastavku) i sadrže zapis za svaku

[25] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 590
[26] DAMA International (2018) Data Management Body of Knowledge, Technics Publications, Basking Ridge, str. 384
[27] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 591-592

transakciju koja se dogodi u danu. Zbog svoje strukture bazirane na mnogim izdvojenim tablicama, takve baze nisu pogodne za kompleksne upite. Podaci za potporu odlučivanju razlikuju se od operativnih podataka u 3 osnovna područja: **vremenskim rasponima** (podaci za potporu odlučivanju pokrivaju duža vremenska razdoblja), **granuliranosti** (menadžeri generalno promatraju različite agregate podataka nasuprot pojedinačnih zapisa) te **dimenzionalnosti** (mogućnost promatranja podataka po različitim dimenzijama poput prodaje po regiji, državi, dućanu i kupcu)[28].

Neki od popularnih softverskih BI rješenja su Dundas BI, Oracle BI, Sisense, Zoho Analytics te Microsoft Power BI.

Praktično značenje ovakvih sustava leži u tome što menadžment nekog velikog poduzeća može dobiti objedinjeni prikaz podataka iz potpuno heterogenih, raznovrsnih izvora, te ih onda prikazati, vizualizirati i gledati kroz razne dimenzije kroz jedinstveno sučelje prilagođeno korisnicima čija je razina tehničkog znanja orijentirana ka standardnim uredskim softverskim alatima (što je zbog radne specijalizacije najčešće i slučaj kod menadžmenta). Microsoftov Power BI može pristupiti podacima iz stotina različitih izvora, kao što su Dynamics 365, Salesforce, Azure SQL DB, Excel, i SharePoint, a ti se podaci uživo, inkrementalno osvježavaju i automatski uklapaju u zadanu shemu[29]. Ilustracije radi, regionalni menadžer nekog velikog trgovačkog lanca za Zapadnu Europu pomoću ovakvog alata ima lako dostupne, vizualizirane, najsvježije podatke koje može promatrati iz raznih dimenzija i razina agregacije, za različita vremenska razdoblja iz čitavog podatkovnog sustava poduzeća. Taj veliki heterogeni sustav može sadržavati podatke o prodaji iz više tisuća poslovnica koje koriste Oracle baze podataka, podatke vezane za digitalni marketing s Facebooka, različite Excel tablice koje na tjednoj bazi stvara niži menadžment, podatke o zaposlenicima sa Salesforce platforme itd. Kao što je na kraju potpoglavlja 2.3.2. spomenuto, baza koje će služiti kao podatkovna potpora BI softveru nešto je drugačije dizajnirana od one koja opslužuje transakcijske sustave (aplikacije za isplatu plaća, fakturiranje, rezervacije u hotelima, prodaju i sl.).

Zbog velikog broja razlika u strukturi i potrebama koje podaci za potporu odlučivanju imaju u odnosu operativne podatke, oni se spremaju u posebnu vrstu baze podataka – skladište

[28] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 601-602

[29] Go from data to insight to action with Power BI Desktop. Dostupno na: https://powerbi.microsoft.com/en-us/desktop/ [20.2.2021.]

podataka (*data warehouse*). Coronel i Morris (2019) vide skladište podataka kao temelj infrastrukture poslovne inteligencije[30].

Skladište podataka jest dakle posebna vrsta baze podataka koja se periodički puni podacima iz transakcijskih baza, ali i drugih izvora podataka. Podaci prolazi kroz vrstu softverskog filtra, ETL procedure kako bi se prilagodili shemi (gledano iz softverske perspektive), tj. potrebama odlučivanja (gledano iz poslovne perspektive). Budući da podatkovno opslužuje softver namijenjen potpori odlučivanju, skladište će imati mnogo više čitanja podataka (često vrlo kompleksnih upita), nego li novih zapisa. Specifičnost skladišta je u tome što se radi o kolekciji integriranih, subjektno-orijentiranih, vremenski određenih i ne-volatilnih podataka.

**Integrirani podaci** – skladište podataka je centralizirana, konsolidirana baza koja integrira podatke čitave organizacije iz različitih izvora, što najčešće povlači i različite formate. Integracija podataka podrazumijeva da su svi poslovni entiteti, podatkovni elementi, karakteristike podataka i poslovne metrike opisivani na jednoličan način kroz čitav sustav. U praksi, računala (osim možda nekakvih generaliziranih AI alata) nemaju ljudsku logiku prepoznavanja uzoraka i sličnosti, već za zadatke agregiranja podataka trebaju jedinstveni način prikaza istih. U praksi, jedna poslovnica nekog poduzeća može za statuse narudžbi koristiti zapise "otvorena", "primljena", "otkazana" i "zatvorena", dok ih duga poslovnica vodi kao "1", "2", "3" i "4". Računovodstvo nekog fakulteta može studentima dodjeljivati atribute "preddiplomski", "diplomski" i "integrirani", dok IT služba iste podatke vodi kao "PD", "D" i "INT". Kako bi se skladište moglo sagraditi, potrebno je "raspetljati" postojeće razlike u formatima, a proces izgradnje jedinstvenog zajedničkog formata podataka može biti dug. Ipak, napor je isplativ zbog poboljšanih mogućnosti donošenja odluka te razumijevanja vlastitog poslovanja.

**Subjektno orijentirani podaci** – skladište podataka ima drastično različitu organizaciju podataka od baze transakcijskih aplikacija. Podaci su organizirani i agregirani po temama poput prodaje, marketinga, financija i distribucije. Za svaku od tema, skladište sadrži specifične subjekte od interesa – proizvodi, kupci, odjeli, regije, promocije i sl. U klasičnoj transakcijskoj bazi, podaci se spremaju kao pojedinačni zapisi (npr. fakture sa svojim datumom, iznosom, vrstom proizvoda i kupcem kojem su izdane), dok je u skladištu fokus na podatke, a ne na procese koji te podatke modificiraju. Stoga će umjesto spremanja pojedinačnih faktura,

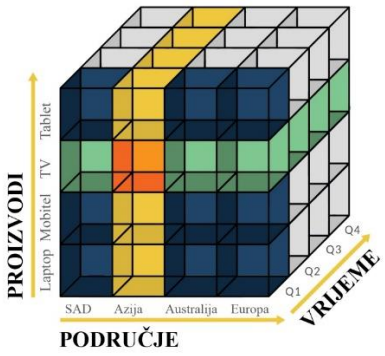
[30] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 594

skladište podataka spremati agregirane komponente kao što su "prodaja po vrsti proizvoda" i "prodaja po kupcu".

**Vremenska određenost** – za razliku od operativnih podataka koji su fokusirani na trenutne transakcije, skladište predstavlja tok podataka kroz vrijeme. Svako periodično učitavanje podataka u skladište izaziva ponovno izračunavanje svih vremenski ovisnih agregata. Tako će npr. novo učitavanje prodajnih podataka za prethodni tjedan u skladište automatski ažurirati tjedne, mjesečne, godišnje i sve druge vremenski ovisne agregate za proizvode, kupce, poslovnice i druge varijable. Vremenska komponenta je u skladištu krucijalna, budući da je ono zapravo serija snimaka stanja različitih varijabli kroz vrijeme (*snapshots*). Svi podaci sadrže vremenski ID koji se koristi za agregirati sumarne preglede i agregate po tjednu, mjesecu, kvartalu, godini itd. Nakon što podaci ETL procedurom uđu u skladište, nije moguće više mijenjati dodijeljeni vremenski ID.

**Ne-volatilnost** – podaci koji jednom uđu u skladište više se ne uklanjaju. Budući da podaci u skladištu predstavljaju poslovnu povijest kompanije, operativni podaci iz tekućeg poslovanja im se konstantno dodaju. Stoga skladište podataka konstantno raste, a DBMS mora biti sposoban upravljati bazama od više terabajta[31].

Skladišta podataka spremaju podatke u strukturu poznatu kao dimenzijski model (detaljnije o modeliranju baza podataka u izdvojenom poglavlju), što omogućava BI aplikacijama analizu podataka po dimenzijama trodimenzionalnih kocaka nasuprot jednostavnim tablicama poput onih koje pruža Microsoft Excel. Takve analize im omogućuju OLAP tehnologije (*Online Analytical Processing* – Online analitičko procesuiranje)[32].




*Slika 2: OLAP kocka. Izvor: izradio autor prema: https://olap.com/olap-definition/*

[31] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 606-607
[32] What is the definition of OLAP? Dostupno na: https://olap.com/olap-definition/ [21.2.2021.]

Neke najvažnije metode OLAP analize, ujedno i programirane funkcionalnosti BI alata su:

- *Drill up* – sumiranje podataka s povećanom generalizacijom (tjedni – kvartalni – godišnji podaci o prodaji)
- *Drill down* – suprotnost od drill up, povećava se razina detalja
- *Pivot* – Zakretanje dimenzija OLAP kocke
- *Slice and dice* – filtriranje po nekoj od dimenzija OLAP kocke
- *Sortiranje*
- *Selekcija* – filtriranje po nekoj vrijednosti ili rasponu vrijednosti
- *Izračunavanje deriviranih atributa iz postojećih vrijednosti*[33].

## 2.5. Veliki podaci (Big data) – primjena i problemi

Određena razina baratanja informatičkim pojmovima se de facto podrazumijeva u današnje vrijeme – razumijevanje jedinica mjere digitalnih informacija – megabajta i gigabajta te koliko je istih potrebno za zapisivanje teksta, slika i audiovizualnih sadržaja određene kvalitete i duljine. Već i površni korisnik osobnog računala sposoban je otprilike procijeniti veličinu u megabajtima za tekstualnu PDF datoteku od pet strana, fotografiju osrednje kvalitete slikanu mobitelom ili video duljine tridesetak sekundi – s jednakom intuitivnošću kao da se radi o kratkim distancama u metrima ili nekoj drugoj prirodnoj pojavi. I laiku je jasno kako jedan YouTube sadrži milijarde videa, koje gledaju milijuni korisnika ostavljajući nebrojene komentare i "*like-ove*", a tim korisnicima se pak preporučaju sadržaji na temelju podataka o njima samima. Dakako, svi ti podaci negdje se moraju spremati te nekako biti učinjeni dostupnima kada su zatraženi od strane korisnika preko aplikacije ili od same aplikacije radi izvršavanja neke pozadinske funkcije. Tako se dolazi do pojma Veliki podaci – Big data.

Prasad (2016) daje sljedeću definiciju Velikih podataka:

*"Big data je svaka količina strukturiranih, polu-strukturiranih i nestrukturiranih podataka velikog opsega, koja ima potencijal rudarenja za informacije, a gdje pojedinačni zapisi više nisu bitni, već samo agregati. Podaci postaju Veliki podaci u trenutku kada ih postane teško obrađivati tradicionalnim tehnikama."*[34]

[33] Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow, str. 1114
[34] Prasad, Y.L. (2016) Big Data Analytics Made Easy, Notion Press, Chennai, str. 10

Batinić i Dobrinić (2019) ističu kako se elementi Velikih podataka temelje na opažanjima, eksperimentima, evidentiranju raznovrsnih aktivnosti, a posebno naglašavaju njihovu važnost u područjima digitalnog marketinga te umjetne inteligencije[35].

Najčešće se granicu Velikih podataka određuje prema kriteriju 5V/6V. Različiti autori daju različite poglede na ovu kraticu, obično bez nekih većih razlika u srži (spajanje pojmova, biranje sinonima koji također počinju slovom "V"). Prasad (2016) tako ističe:

- Volumen (*Volume*) – Big data implicira ogroman volumena podataka generiranih od strane senzora i strojeva, od strane eksplozije interneta, društvenih mreža, e-trgovine, GPS uređaja itd.
- Brzina (*Velocity*) – stope kojima se podaci stvaraju, npr. Twitter korisnici generiraju oko 450 milijuna novi "Tweet-ova" dnevno.
- Raznolikost (*Variety*) – miješanje strukturiranih (relacijske baze podataka), polu-strukturiranih (e-mailovi, "Tweet-ovi", sistemski zapisnici, korisničke ocjene) i nestrukturiranih (slike, audio i video zapisi) podataka.
- Istinitost (*Veracity*) – odnosi se na subjektivnosti, "buku" i abnormalnosti u podacima. Radi dobivanja značajnijih uvida iz podataka, potrebno ih je inicijalno pročistiti.
- Validnost (*Validity*) – odnosi se na prikladnost podataka za odluke koje se donose.
- Volatilnost (*Volatility*) – odnosi se na koliko dugo će podaci biti validni, budući da s obzirom na konstantan priljev novih podataka, ono što validno u danom trenutku, možda neće biti za nekoliko dana ili čak za nekoliko minuta[36].

Radi slikovitijeg definiranja pojma primjerima iz stvarnog života, Coronel i Morris (2019) ističu kako telekomunikacijske kompanije poput američkog AT&T-a upravljaju sustavima koji sadrže podatke o trilijunima telefonskih poziva, s priljevom novih podataka od 70.000 poziva u sekundi. Ne samo da te kompanije moraju spremati i upravljati tim ogromnim kolekcijama podataka, već moraju biti sposobni brzo pristupiti bilo kojoj činjenici u toj kolekciji. Slično tako, najpoznatija web tražilica Google prema procjenama obavlja 91 milijun pretraga dnevno, na kolekciji podataka od nekoliko terabajta (a rezultati pretraga su dostupni gotovo trenutačno)[37]. Kako bi ovakvi sustavi povezanih podataka uvijek bili dostupan (unutar razumnog vremena) aplikaciji preko koje im korisnik pristupa, vrlo je bitno kako su ti podaci

[35] Batinić, P., Dobrinić, D.(2019) Implementacija velikih vrsta podataka u CRM. Hrčak [online]. Dostupno na: https://hrcak.srce.hr/234546 [28. 6. 2021]
[36] [36] Prasad, Y.L. (2016) Big Data Analytics Made Easy, Notion Press, Chennai, str. 10
[37] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 4

organizirani, od izbora tehnologije (DBMS), preko njihovih softverskih obilježja i međupovezanosti pa do fizičkog smještaja po tvrdim diskovima. Kao uvod u SQL i NoSQL tehnologije, koje se ne pojavljuju u vakuumu, u izolaciji, u idućem poglavlju biti će dan kratak uvod u koncepte modeliranja i skaliranja baza podataka.

## 2.6. Modeliranje i skaliranje baza podataka

Različiti tipovi baza podataka ("različiti" gledajući i s uporabne i s tehničke strane) imaju raznovrsne oblike organizacije podataka, što igra ključnu ulogu u njihovim performansama pri različitim vrstama upita i operacija (pretraživanjima, izmjenama, umetanjima, brisanjima). Dodatna dimenzija koju se mora uzeti u obzir pri izboru baze jest njeno ponašanje i mogućnosti kod distribuiranih sustava – najjednostavnije – situacija gdje je zbog vrste i opsega poslovanja (i/ili potrebe za snažnijom pouzdanošću) bazu nužno "razbiti" na više fizički zasebnih računala poslužitelja.

### *2.6.1. Distribuirani sustavi i skaliranje baza podataka*

Za tvornički ugrađeni softver u razne elektroničke naprave te za aplikacije dizajnirane za osobna računala i mobilne uređaje, ne postoji potreba dizajniranja distribuirane softverske arhitekture. Ipak, većina velikih sustava su distribuirani sustavi, što znači da je sistemski softver distribuiran preko mnogo različitih računala. Izbor arhitekture i strukture je ključna odluka o kojoj ovise performanse i pouzdanost sustava. Izbor arhitekturnog stila trebao bi ovisiti o ne-funkcijskim zahtjevima sustava glede performansi, sigurnosti (i s aspekta hakerskih napada i s aspekta okolnosti poput padova mreže ili napajanja el. energijom), dostupnosti korisnicima i lakoće održavanja[38].

Distribuirani sustav je prema Sommervilleu (2016) svaki sustav koji uključuje više računala, umjesto jedne aplikacije koja se pokreće na jednom stroju. Čak i naoko monolitne aplikacije na osobnom računalu ili laptopu, poput softvera za obradu slika, mogu zapravo biti distribuirani sustavi. Izvršavaju temeljne zadatke na jednom računalu, ali često ovise o eksternom sistemu u oblaku za ažuriranja, spremanje podataka i druge usluge[39].

Glavne prednosti razvoja distribuiranih sustava su sljedeće:

[38] Sommerville I. (2016) Software Engineering, Pearson, Harlow, str. 170-171
[39] Sommerville I. (2016) Software Engineering, Pearson, Harlow, str. 491

1. *Dijeljenje resursa* – distribuirani sustavi omogućuju dijeljenje hardverskih i softverskih resursa – poput diskova, printera, datoteka i kompajlera, koji su povezani s računalima u mreži.
2. *Otvorenost* – distribuirani sustavi su obično dizajnirani oko standardnih internetskih protokola tako da oprema i softver različitih proizvođača postaju kompatibilni bez problema.
3. *Istovremenost* – više procesa može se odvijati istovremeno na zasebnim računalima u mreži. Ti procesi mogu (ali ne moraju) komunicirati jedan s drugim u sklopu svoje normalne operacije.
4. *Skalabilnost* – barem u principu, distribuirani sustavi su skalabilni, što znači da se mogućnosti sustava mogu povećati dodajući nove resurse kako bi se nosilo s pojačanim zahtjevima.
5. *Tolerancija za kvarove* – dostupnost više računala i potencijal za replikaciju informacija među njima znače da distribuirani sustavi mogu podnijeti neku količinu hardverskih i softverskih kvarova. U većini distribuiranih sustava, degradirana razine usluge može biti i dalje pružana u slučaju kvarova. Jedino gubitak mreže će dovesti do potpunog gubitka usluge[40].

Skalabilnost sustava označava njegovu moć da pruža uslugu visoke kvaliteta kako zahtjevi na sustav rastu. Fokus ovog rada je na bazama podataka (shodno tome na njihovom skaliranju), no baze je nemoguće promatrati u potpunosti u izolaciji. Stoga će u poglavlju 4.1. biti više riječi o klijentsko-poslužiteljskoj arhitekturi i ostalim slojevima na koje je potrebno obratiti pozornost pri skaliranju sustava. Sustavi (samim time i baze podataka) mogu skalirati prema gore (vertikalno) ili prema van (horizontalno). Vertikalno skaliranje je nadogradnja sistema jačim resursima, npr. dodavanje još diskovnog prostora u poslužiteljsko računalo (ili zamjena procesora jačim modelom ili dodavanje još radne memorije). Horizontalno skalirati znači dodati još resursa u sustav – npr. osposobiti još jedno poslužiteljsko računalo da radi uz postojeće. Horizontalno skaliranje je često financijski efikasnije zbog računalstva u oblaku koje omogućuje brzo i jednostavno dodavanje i uklanjanje servera iz sustava[41]. Potrebno je naglasiti i jedno zdravorazumsko ograničenje – potencijal za horizontalno skaliranje je (hipotetski, u slučaju kvalitetne arhitekture sustava) neograničen, dok je za vertikalno ograničen – matična

[40] Sommerville I. (2016) Software Engineering, Pearson, Harlow, str. 491
[41] Sommerville I. (2016) Software Engineering, Pearson, Harlow, str. 494

ploča računala poslužitelja prima ograničen broj HDD-ova ili SSD-ova, ograničen broj pločica RAM-a i jedan procesor, koliko god snažan bio i koliko god jezgara imao.

Jasno je kako distribuirani sustavi podrazumijevaju i distribuirane baze podataka, a minimalni uvjeti koji moraju biti zadovoljeni kako bi se baza podataka smatrala distribuiranom su sljedeći:

- **Povezanost čvorova baze preko računalne mreže**. Postoji više računala, koja se obično nazivaju **čvorovi** (*nodes*). Ti čvorovi moraju biti povezani mrežom kako bi se podaci i upute (programski kod) mogli prenositi među njima.
- **Logička interrelacija povezanih baza podataka**. Esencijalno je da informacije u različitim čvorovima koji sadrže baze budu logički povezani.
- **Moguća neprisutnost homogenosti povezanih čvorova**. Nije nužno da svi čvorovi budu identični u vidu podataka koje sadrže, hardvera i softvera.

Pritom, čvorovi se svi mogu nalaziti u relativnoj blizini (ista zgrada ili grupa zgrada) i biti povezani lokalnom mrežom, ili mogu biti široko geografski distribuirani i povezani internetom[42].

Nakon određivanja pojma distribuiranih baza i približavanja načina raspoređivanja fizičkih medija po računalnim sistemima, potrebno je dotaknuti se i pojmova *fragmentacije* i *replikacije* podataka, što, u osnovi, odgovara na pitanje rasporeda samih podataka po fizičkim medijima.

U distribuiranoj bazi podataka, potrebno je donijeti odluku koji čvorovi će spremati koji fragment sveukupne baze podataka. Potrebno je prethodno bazu podijeliti na logičke jedinice te prema nekom ključu iste spremati po bazama u čvorovima. Taj proces se naziva *fragmentacija*. Pritom, potrebni su metapodaci u vidu fragmentacijske i alokacijske sheme, koji omogućuju pristupanje tim distribuiranim podacima kada je sustavu i/ili krajnjem korisniku to potrebno, kao da se radi o jedinstvenoj bazi podataka na jednom čvoru (poslužitelju)[43].

Osim čistog "razbijanja" baze na fragmente i rasporedu po čvorovima, postoji mogućnost i njihove *replikacije* (kopiranja). Replikacija je koristan alat za poboljšanje dostupnosti podataka. Dva ekstrema u dizajnu distribuiranog sustava su:

- Potpuna replikacija čitave baze na svakom čvoru
- Izbjegavanje ikakve replikacije – svaki fragment je spremljen na točno jednom čvoru (računalu poslužitelju).

[42] Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow, str. 842-843
[43] Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow, str. 847

Potpuna replikacija bi omogućavala funkcioniranje sustava dokle god je barem jedan čvor operabilan, što drastično povećava dostupnost. Također, poboljšane su performanse dohvata podataka budući da bilo koji upit mora dohvaćati podatke sa samo jednog čvora, umjesto da se za jedan upit mora ostvariti komunikacija s većim brojem računala, potencijalno na različitim kontinentima. Jasno, takav sustav bi bio izuzetno spor za operacije ažuriranja (budući da se podaci moraju ažurirati na svakom od čvorova). U praksi se teži pronaći balans između ta dva ekstrema, ovisno o zahtjevima sustava – *parcijalna replikacija*. Tako će neki od fragmenata baze biti replicirani jednom, neki više puta, neki na svim čvorovima, a neki niti jednom. Ponovno, potrebni su metapodaci radi praćenja tih kopija podataka po čvorovima – *replikacijska shema*[44].

Prije kretanja u pojam modeliranja baza podataka, potreban je kratak osvrt na CAP teorem i ACID svojstva.

### *2.6.2. ACID i CAP*

U poglavlju 2.3.1. istaknuta je važnost DBMS-a kao softvera koji obavlja i korisnicima apstrahira niz funkcija ključnih za integritet i konzistentnost podataka u bazi. Transakcije koje se izvršavaju u bazi podataka trebale bi, prema Elmasriju i Navatheu (2016)[45] posjedovati nekolicinu svojstava koje se često obuhvaćaju pamtljivim akronimom **ACID**:

- **Atomičnost (*atomicity*)**. Svaka transakcija je atomična jedinica procesuiranja – ili se izvršava u potpunosti, ili se uopće ne izvršava.
- **Očuvanje konzistentnosti (*consistency preservation*)**. Transakcija se mora izvršavati od početka do kraja bez smetnje od drugih transakcija – trebala bi voditi bazu podataka iz jednog konzistentnog stanja u sljedeće.
- **Izolacija (*isolation*).** Transakcija bi se trebala pojavljivati kao da se izvršava u izolaciji od ostalih transakcija, iako se velik broj transakcija izvršava istovremeno. Dakle, izvršenje transakcije ne smije biti ometano od strane bilo koje druge transakcije koja se izvršava istovremeno.
- **Trajnost (*durability*).** Promjene nastale na bazi podataka izvršenjem neke transakcije moraju opstati u bazi te ne smiju biti izgubljene uslijed kvarova.

U praksi, na primjerima ova svojstva bi značila sljedeće:

[44] Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow, str. 847-850
[45] Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow, str. 757-758

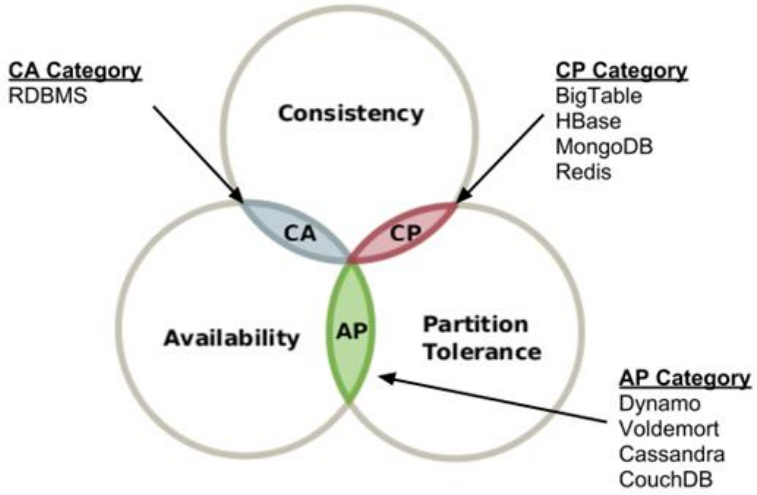
- Atomičnost – Korisnik banke (A) prebacuje preko bankovne aplikacije svojem prijatelju (B) 50 kuna za platiti termin badmintona. U bankovnom sustavu ta transakcija sastoji se od umanjivanja iznosa na računu A (vrijednosti u bazi podataka) za 50, te uvećavanje računa B za isti iznos. Ako bi došlo do pada mreže usred te transakcije (nakon što je izvršeno umanjenje A, no ne i uvećanje B), cijela transakcija će se odbaciti, te na bazi neće nastati nikakva promjena, umjesto da nastane polovična promjena (posljedice koje su logične i nije ih potrebno posebno objašnjavati).
- Očuvanje konzistentnosti – Nadovezujući se na prethodni primjer, korisnik A prije transakcije na računu ima 2000 kuna, a korisnik B 2500. Nakon transakcije korisnik A ima 1950 kuna, a korisnik B 2550 kuna. Zbroj prije i poslije transakcije je jednak, te je svojstvo konzistentnosti baze ostvareno.
- Izolacija – Ako bi se u primjer uvela i druga transakcija s računa A, npr. rata kredita za svih 2000 kuna, koja se slučajno odvija istovremeno kao prva, DBMS bi osigurao da se jedna od njih izvrši prije druge. U praksi, to bi spriječilo izvršenje one koja je kasnije na redu za izvršavanje zbog nedovoljno sredstava na računu (jednostavnosti radi, u primjeru nema dozvoljenih minusa na računu).
- Trajnost – Transakcije iz bankovnog primjera su izvršene, no došlo je do kratkotrajnog pada električnog napajanja kod poslužitelja. Budući da su transakcije izvršene u potpunosti, zapisane su u trajnu memoriju, te će promjene na bazi nastale tim transakcijama opstati nakon ponovnog podizanja sustava.

Kontrola ACID svojstava postaje kompleksnija u distribuiranom sustavu s replikacijom podataka (budući da može postojati više kopija svakog podatka). Ako se jedna kopija nekog podatka ažurira, ažuriranje je potrebno primijeniti konzistentno na svaku od preostalih kopija istog podatka. Postoji i mogućnost da se jedna kopija objekta X ažurira transakcijom T1, a istovremeno se druga kopija (na drugom čvoru) ažurira transakcijom T2, što stvara dvije nekonzistentne kopije istog objekta na dva različita čvora u distribuiranom sustavu. Ako dvije druge transakcije, T3 i T4, pokušavaju pročitati vrijednost X, svaka od njih bi mogla dobiti različitu kopiju objekta X. CAP teorem je uveden kako bi se objasnilo neke od međusobno sukobljenih zahtjeva u distribuiranom sustavu s replikacijom podataka. CAP je akronim za tri poželjna (i međusobno sukobljena) svojstva u takvim sustavima:

- **Konzistentnost (Consistency)** – čvorovi imaju identične kopije repliciranog podatkovnog objekta kada im različite transakcije pristupaju.

- **Dostupnost (Availability)** – svaki upit za čitanjem ili ažuriranjem podatkovnog objekta će ili biti uspješno procesuiran ili će javiti grešku kako operacija ne može biti izvršena.
- **Tolerancija na particioniranje (Partition Tolerance)** – sustav može nastaviti s pružanjem usluge unatoč kvarovima u komunikaciji među pojedinačnim čvorovima.

**CAP teorem** glasi kako nije moguće garantirati sva tri poželjna svojstva (konzistentnost, dostupnost i toleranciju na particioniranje) istovremeno u distribuiranom sustavu s replikacijom podataka. Dizajner sustava stoga mora birati koja dva od tri svojstva će sustav garantirati[46].




*Slika 3: CAP po različitim tipovima BP. Izvor: The Total Newbie's Guide to Cassandra. Dostupno na: https://blog.insightdatascience.com/the-total-newbies-guide-to-cassandra-e63bce0316a4*

### *2.6.3. Podatkovni model*

Podatkovni model je prvi korak u dizajniranju baze podataka i služi kao most između stvarnih pojava i računalne baze podataka. Model je relativno jednostavan prikaz, najčešće grafički kompleksnih podatkovnih struktura iz stvarnog problemskog područja, apstrakcija. U okruženju baza podataka, model prikazuje podatkovne strukture i njihove karakteristike, međusobne odnose, ograničenja, transformacije i druge konstrukcije bitne za specifičnu problemsku domenu[47].

[46] Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow, str. 888-889
[47] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 34-35

Prema Coronelu i Morrisu (2019)[48], osnovni elementi podatkovnog modela su entiteti, atributi, relacije i ograničenja (*constraints*).

**Entitet** je mjesto, osoba, stvar ili događaj o kojemu će prikupljati i spremati podaci. Entitet predstavlja određeni tip objekta u stvarnom svijetu, a unutar sebe ima pojedinačne instance. Npr. entitet KUPAC može imati velik broj pojedinačnih instanci, npr. "Art Vandelay" ili "George Costanza". Entiteti mogu biti fizički objekti kao kupci ili proizvodi, ali i apstrakcije poput ruta leta ili glazbenih koncerata.

**Atribut** opisuje karakteristiku nekog entiteta. Npr. entitet KUPAC može biti opisan atributima kao što su ime, prezime, broj telefona, adresa, broj kreditne kartice itd.

**Relacija** opisuje povezanost, odnose između entiteta. Npr. može postojati relacija između kupaca i agenata gdje jedan agent služi više kupaca, dok svaki kupac ima samo jednog agenta. Svaki zaposlenik može imati mnogo vještina, dok svaka od vještina može biti prisutna kod više zaposlenika. Svaka poslovnica ima samo jednog poslovođu, a jedan poslovođa može voditi samo jednu poslovnicu. Podatkovni modeli koriste tri vrste relacija: jedan-prema-jedan (1:1), jedan-prema-više (1:M ili 1..*) i više-prema-više (M:N ili *..*).

**Ograničenja** su restrikcije uvedene na podatke koje DBMS osigurava radi osiguravanja integriteta baze podataka. Studentov prosjek ocjena se može kretati između 2.0 i 5.0. Ime studenta može sadržavati maksimalno 40 znakova. Svaki kolegij može imati isključivo jednog profesora.

Dizajner baze podataka se pri određivanju prethodno opisanih elemenata vodi **poslovnim pravilima** kako bi razumio vrste podataka koji se stvaraju u organizaciji. Poslovno pravilo je kratak, precizan i nedvosmislen opis politike, procedure ili principa unutar specifične organizacije.

U idućem poglavlju slijedi detaljno objašnjenje modeliranja podataka za izabrane tipove baza podataka, dok je za sada dovoljno samo podijeliti modele podataka prema Elmasriju i Navatheu (2016) na modele visoke razine apstrakcije (konceptualne modele - pružaju koncepte koji su najbliži načinu na koji većina korisnika percipira podatke, na temelju entiteta, atributa i relacija) modele, modele niske razine apstrakcije (fizičke modele namijenjene IT stručnjacima - pružaju koncepte koji opisuju detalje kako se podaci spremaju na fizičkim računalnim skladišnim medijima) te na reprezentacijske modele (nalaze se između dva prethodna ekstrema,

[48] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 36-38

a pružaju koncepte koji su razumljivi krajnjim korisnicima, a opet nisu daleko od načina na koji se podaci organiziraju na računalnom mediju).

U podatkovnom modelu, važno je učiniti distinkciju između opisa baze podataka i same baze. Opis se naziva *shema baze podataka*, specificira se u fazi dizajna baze i ne očekuju se česte promjene, iako može evoluirati kroz razvoj povezane aplikacije (npr. potreba da se entitetu KUPAC pridoda novi atribut "e-mail adresa", koji nije bio dio izvorne sheme). Većina podatkovnih modela posjeduje određene konvencije za prikaz shema u obliku dijagrama. Sami podaci u bazi, za razliku od sheme, mogu se mijenjati jako često. Svi podaci u bazi u određenom trenutku nazivaju se *stanje BP* ili snimak (*snapshot*). Kod kreiranja nove baze, u DBMS-u se specificira shema, a nova baza je u *praznom stanju*. Prvo populiranje (učitavanje) podataka bazu dovodi u *inicijalno stanje*. Od tog trenutka, svaka operacija ažuriranja stvara novo stanje baze, a DBMS pomoću svojih metapodataka osigurava kako je svako novo stanje – validno stanje, tj. zadovoljava strukturu zadanu shemom i poštuje sva zadana ograničenja[49].

# 3. Tipovi suvremenih baza podataka

## 3.1. Relacijske (SQL) baze

### *3.1.1. Osnovni koncept i modeliranje*

U prethodnim poglavljima objašnjeni su osnovni i općeniti koncepti nužni za razumijevanje funkcioniranja baza podataka. U nastavku slijedi sužavanje tih koncepata u obliku pregleda konkretnih implementacija kroz različite tehnologije. Iako su upravo njihove alternative primarni fokus ovoga rada, bilo bi nemoguće istinski shvatiti te alternative bez barem osnovnog upoznavanja/podsjetnika na najstariju te i dalje najrašireniju vrstu baza – relacijske (SQL) baze, koje vuku korijene još iz 70-ih godina prošlog stoljeća.

Relacijska baza podataka je entitet koji se sastoji od logičkih jedinica – *tablica*. Te tablice su međusobno povezane na unaprijed definirani način – *relacije*. Na taj način su podaci pojednostavljeni u manje, logičnije i lakše upravljive jedinice koje optimiziraju performanse baze. Tablice se sastoje od redova i stupaca gdje se podaci spremaju. U relacijskom modelu, te

[49] Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow, str. 33-35

tablice su međusobno povezane što ubrzava dohvat podataka kod korisničkih upita. Redovi u tablicama nazivaju se još i *zapisi*, te će npr. u tablici KUPCI svaki red sadržavati podatke za jednog kupca. S druge strane, svaki stupac sadržava samo jedan *atribut* za kupca (ime, prezime, ime kompanije, adresa, broj telefona). Stupci su konzistentni, sadržavaju isti tip podatka u svakom redu. Redoslijed stupaca i redova nije naročito bitan za funkcioniranje baze[50].

Tablica se dakle sastoje od redova i stupaca gdje se nalaze podaci. U relacijskoj bazi, te tablice su međusobno povezane što poboljšava proces dohvata podataka kada korisnik pokrene upit[51]. Radi ilustracije, u nastavku slijede dvije tipične povezane tablice u relacijskoj bazi, KUPCI i NARUDŽBE.

| ID KUPCA | IME | POZICIJA | KOMPANIJA | STATE | CONTACT NO |
|---|---|---|---|---|---|
| 1 | Kathy Ale | President | Tile Industrial | TX | 3461234567 |
| 2 | Kevin Lord | VP | Best Tooling | NY | 5181234567 |
| 3 | Kim Ash | Director | Car World | CA | 5101234567 |
| 4 | Abby Karr | Manager | West Mart | NV | 7751234567 |

*Tablica 1: Tablica KUPCI. Izvor: vlastita izrada prema Alvaro, F. (2018) SQL: Easy SQL Programming & Database Management for Beginners, Your Step-By-Step Guide to Learning the SQL Database, Amazon, e-izdanje*

| ID NARUDŽBE | DATUM | ID KUPCA | ID PROIZVODA | KOLIČINA |
|---|---|---|---|---|
| 1 | 23-05-16 | 1 | 4 | 300 |
| 2 | 09-09-16 | 1 | 5 | 100 |
| 3 | 17-02-16 | 3 | 2 | 150 |
| 4 | 12-05-16 | 2 | 2 | 500 |

*Tablica 2: Tablica NARUDŽBE. Izvor: vlastita izrada prema Alvaro, F. (2018) SQL: Easy SQL Programming & Database Management for Beginners, Your Step-By-Step Guide to Learning the SQL Database, Amazon, e-izdanje*

Ovakve tablice su osnova relacijskih baza podataka. Važno za zamijetiti u ovim primjerima je postojanje jednog zajedničkog polja u obje – ID KUPCA – tzv. zajednički ključ koji međusobno povezuje tablice u relacijskoj bazi. Postojanje zajedničkih ključeva čini mogućim spajanje podataka iz više tablica, formirajući veći set podatkovnih entiteta[52].

Prvi stupci u obje tablice nazivaju se *primarnim ključevima* (*primary keys*). Primarni ključ je specijalno polje ili kombinacija polja koji svaki zapis u tablici čini jedinstvenim, što garantira integritet podataka. Polja definirana kao primarni ključevi ne mogu sadržavati *null* vrijednosti (poseban oblik vrijednosti u SQL-u i drugim programskim jezicima koji služi kao indikator da

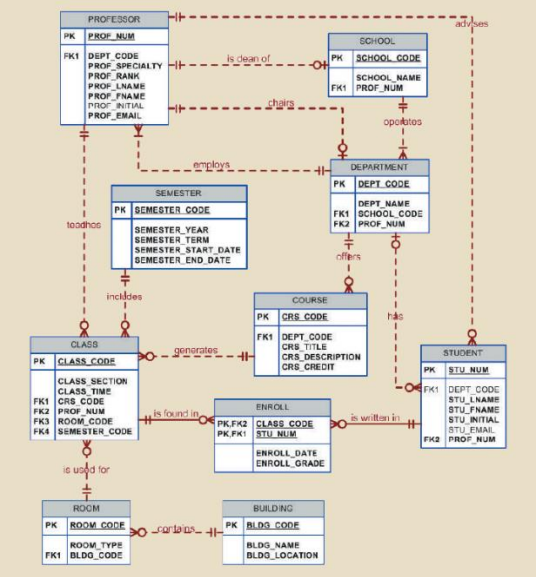
[50] Alvaro, F. (2018) SQL: Easy SQL Programming & Database Management for Beginners, Your Step-By-Step Guide to Learning the SQL Database, Amazon, e-izdanje, str. 10-12
[51] Alvaro, F. (2018) SQL: Easy SQL Programming & Database Management for Beginners, Your Step-By-Step Guide to Learning the SQL Database, Amazon, e-izdanje, str. 11
[52] Alvaro, F. (2018) SQL: Easy SQL Programming & Database Management for Beginners, Your Step-By-Step Guide to Learning the SQL Database, Amazon, e-izdanje, str. 12

vrijednost atributa nije poznata ili nedostaje – *null* **nije** prazno polje[53]). Definiranje primarnih ključeva odvija se tijekom kreacije tablice, a njihovu dodjelu u od strane korisnika određenom obliku (broj znamenki, numerički ili alfanumerički) obavlja DBMS pri stvaranju svakog novog zapisa. Obično se tablice s primarnim ključevima nazivaju i tablicama roditeljima (*parent tables*), što znači da pružaju informacije o drugim, ovisnim tablicama – tablicama djecom (*child tables*)[54].

Tablice djeca u sebi imaju *strane ključeve* (*foreign keys*) – atribute čije vrijednosti moraju biti jednake ili primarnom ključu u nekoj drugoj tablici ili *null*. Strani ključevi osiguravaju referencijski integritet, stanje gdje je svaka referenca na neku instancu entiteta od strane drugog entiteta validna, što osigurava DBMS[55]. Konkretno, u gornjem primjeru ID KUPCA je primarni ključ u tablici KUPCI, a strani ključ u tablici NARUDŽBE. Zapisi iz tablice NARUDŽBE mogu samostalno stajati, budući da imaju vlastite, jedinstvene primarne ključeve, pa se ta tablica, kao i tablica KUPCI nazivaj jakim entitetom. Nasuprot tome, mogu postojati tablice gdje su primarni ključevi ili jednaki primarnim ključevima iz neke tablice roditelja, ili su kompozitni, tj. sadrže primarne ključeve svojih tablica roditelja i neku drugu alfanumeričku vrijednost. Takvi entiteti čije postojanje ovisi o drugim entitetima s kojima imaju relaciju nazivaju se slabim entitetima[56].

Prije kreiranje baze kroz sučelje DBMS-a, relacijske baze modeliraju se najčešće grafički koristeći ER (*entity-relation*) dijagrame, u kojima su definirani prethodno identificirani entiteti (tablice), njihovi atributi, primarni i strani ključevi te relacije među entitetima s kardinalnostima (opisi vrsta veza – KUPCI *zadaju* NARUDŽBE, NARUDŽBE *pripadaju* KUPCIMA; te kardinalnosti među njima kako su opisane u 2.6.3. – 1:1, 1:M i M:N).

[53] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 74
[54] Alvaro, F. (2018) SQL: Easy SQL Programming & Database Management for Beginners, Your Step-By-Step Guide to Learning the SQL Database, Amazon, e-izdanje, str. 91
[55] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 75
[56] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 125




*Slika 4: Primjer ER dijagrama za bazu podataka sveučilišta. Izvor: Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston*

Prethodno su objašnjeni svi glavni koncepti gornje slike, potrebno je samo dodati kako ovaj dijagram koristi tzv. "*crow's feet*" notaciju kako bi se odredila kardinalnost relacija među entitetima (oznake na krajevima crta).

Prije prelaska na temu ne relacijskih baza, nužno je spomenuti još tri važna koncepta u SQL bazama koji će biti spominjani i uspoređivani u nastavku: normalizacija, indeksi te sam SQL, jezik za upite po kojemu relacijske baze dobivaju svoje drugo, kolokvijalno ime[57].

**Normalizacija** je proces gdje dizajniranja i redizajniranja baze reduciranjem velikih tablica u manje tablice, tj. grupiranja istovjetnih tipova podataka u zasebne tablice izdvajanjem te referenciranjem stranim ključevima. Na taj način izbjegavaju se duplikatni podaci u tablicama, štedi na diskovnom prostoru te poboljšavaju performanse. Ovisno o poslovnim potrebama, upitima koji se češće ponavljaju, dizajnerskim praksama itd., normalizacija (izdvajanje podataka u zasebne tablice) može se raditi do određenih dubina, tzv. "*normalnih formi*" (NF), kojih u najboljim praksama postoji tri. Krajnji cilj su dobro organizirani, lako upravljivi i uvijek točni podaci bez nepotrebnih duplikata[58].

Kada se baza podataka počne usporavati tijekom upita, u svakom DBMS-u moguće je implementirati **indekse** radi boljih performansi. Indeksi su važni objekti koji služe za brzi dohvat gdje se nalaze (u kojim zapisima) određene vrijednosti atributa, a funkcioniraju u osnovi

[57] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 113
[58] Alvaro, F. (2018) SQL: Easy SQL Programming & Database Management for Beginners, Your Step-By-Step Guide to Learning the SQL Database, Amazon, e-izdanje, str. 97

identično indeksima na krajevima knjiga, izdvajajući vrijednosti atributa te stvarajući referencu na kojim primarnim ključevima u tablicama su prisutne pojedine vrijednosti. Intuitivno razumljiv primjer indeksa bio bi kada bi se često spominjanje riječi u ovom radu (DBMS, SQL, NoSQL, Atributi itd.) izdvojile u listu-stupac negdje na početku rada, sa nizom stranica na kojima se te riječi pojavljuju s njihove desne strane. Na ovaj način DBMS ne mora više kod svakog upita s uvjetom prisutnosti neke vrijednosti atributa (npr. traže se svi zaposlenici koji rade u organizacijskoj jedini OJ3) pretraživati redak po redak i uspoređivati je li ta vrijednost atributa prisutna, već će provjeravati direktno iz indeksa. Indeksi se ne nalaze direktno u tablici nego u metapodacima (nisu vidljivi krajnjem korisniku pri normalnom dohvatu tablica), ali i sami zauzimaju prostor na disku. Pametno postavljeni indeksi uvelike ubrzavaju dohvat podataka kod operacija čitanja, no mogu usporiti operacije brisanja, ažuriranja i umetanje (budući da kod svakog ažuriranja mora biti ažuriran i odgovarajući indeks)[59].

Najkorištenija relacijska baza podataka na svijetu, sa preko 100 milijuna preuzetih primjeraka je MySQL[60]. Druge iznimno popularne SQL baze su PostgresSQL, Microsoft SQL Server, SQLite i Oracle SQL.

### *3.1.2. Upiti*

Upiti na relacijske baze izvode se pomoću specijaliziranog programskog jezika poznatog kao SQL (*Structured Query Language* – Strukturirani jezika za upite). SQL je sačinjen od naredbi koje korisnicima omogućuje stvaranje baza podataka i tabličnih struktura, provođenje različitih manipulacija i administraciju podataka i stvaranje upite radi izvlačenja korisnih informacija. Svi relacijski DBMS softveri podržavaju SQL, a mnogi proizvođači razvili su svoje ekstenzije osnovnog seta naredbi. SQL je moćan i koristan, ali nije namijenjen kao samostojeći jezik za razvoj aplikacija. Unos (većih količina) podataka je moguć kroz SQL, ali nepraktičan, te jezik ne služi za razvoj specijalnih izvještaja, izbornika, modalnih ekrana i drugih funkcionalnosti koje krajnji korisnici očekuju od aplikacija. SQL se fokusira na definiranje podataka (*data definition*) – stvaranje tablica i indeksa te manipulaciju podataka (*data manipulation*) – dodavanje, ažuriranje, brisanje i dohvat[61].

[59] Alvaro, F. (2018) SQL: Easy SQL Programming & Database Management for Beginners, Your Step-By-Step Guide to Learning the SQL Database, Amazon, e-izdanje, str. 92 - 93
[60] Havaš L., Lesar M. (2012) Primjena SQL-a u programima otvorenog koda. Hrčak [online], str. 168. Dostupno na: https://hrcak.srce.hr/94801 [28. 7. 2021]
[61] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 244

SQL je relativno jednostavan za naučiti. Njegov osnovni vokabular naredbi sadrži manje od 100 ključnih riječi (koje su pritom lako pamtljive i intuitivne). Također, SQL je neproceduralni programski jezik – u njemu se naređuje samo *što* treba biti učinjeno, bez brige za *kako*. Kompleksne aktivnosti poput fizičkog spremanja podataka u memoriji su krajnjem korisniku ili programeru apstrahirane od strane DBMS-a koji ih obavlja u pozadini[62].

Bilo bi van opsega ovoga rada ulaziti u sintaksu SQL jezika kroz prolaženje konkretnih naredbi i njihovih funkcija (SELECT, FROM, WHERE, ORDER BY, INSERT INTO, UPDATE, DELETE…), no uz prethodno navođenje osnovnih operacija koje on omogućuje, potrebno je posebnu pozornost obratiti na JOIN naredbu koja zapravo i omogućuje način rada relacijskih baza. JOIN naredba omogućuje inteligentno kombiniranje informacija iz dvaju ili više tablica. Primarni i strani ključevi te relacijska shema omogućuju relacije, tj. povezanost tablica, a JOIN naredbe spajaju te tablice povezane zajedničkim atributima[63].

SQL se s programskim kodom u samih aplikacija koje koriste baze može pisati ili direktno nakon što se aplikacije preko API-ja spoje na DBMS, ili se može apstrahirati pomoću okvira za ORM (*Object-Relational Mapping* – Objektno-relacijsko mapiranje). Ti okviri dolaze kao apstrakcijski sloj, alat koji programerima olakšava rad, budući da bi u suprotnom rezultate SQL upita morali ručno transformirati u podatkovne strukture kakve se koriste u programskim jezicima za razvoj aplikativnog softvera[64]. Nekoliko najčešće korištenih takvih struktura objašnjeno je u poglavlju 4.4. o JavaScript jeziku, za sad je dovoljno reći kako tablični sadržaj nije direktno upotrebljiv u programskom kodu.

### *3.1.3. Prednosti, mane i područje primjene*

**Prednosti**

- Fleksibilni upiti – DBMS-ovi pružaju podršku za raznolika opterećenja. Apstrahiraju interne implementacije i omogućuju mehanizme koji optimiziraju brzinu upita prema fizičkom smještaju podataka na mediju.
- Smanjeni otisak kod skladištenja podataka – zbog normalizacije i drugih prilika za optimizaciju, smanjena količina podataka (zbog nedostatka duplikatnih vrijednosti) maksimizira performanse baze i smanjuje količinu korištenja računalnih resursa.

[62] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 245
[63] Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston, str. 83
[64] AxiomQ: Comparing ORM vs SQL: What to Know. Dostupno na: https://axiomq.com/blog/comparing-orm-vs-sql-what-to-know/ [30.7.2021.]

- Snažna i kroz godine razvijena semantika koja jamči integritet podataka – ACID svojstva su prisutna i jamče validne transakcije.

**Mane**

- Rigidni podatkovni modeli – potreban pažljiv dizajn unaprijed kako bi se osigurale adekvatne performanse i otpornost pri evoluciji. Predefinirana shema pri mijenjanju može stvarati razdoblja nedostupnosti baze.
- Limitirana horizontalna skalabilnost – ovisno o proizvođaču, ili je u potpunosti ne podržana, podržana na *ad-hoc* načine ili podržana kroz upotrebu relativno nezrelih tehnologija.
- Jedinstvena točka pucanja (*Single point of failure*) – otežano fragmentiranje može stvoriti jednu točku – u slučaju nedostatka replikacije (sigurnosnih kopija) – na kojoj cijeli sustav može puknuti u slučaju kvara poslužitelja koji sadrži bazu koji dijeli više softverskih servisa. Čak i ako postoji replikacija baza, zbog kvara prije nego li se unesene promjene propagiraju na ostale kopije, može doći do gubljenja nekih transakcija iz baze distribuiranog sustava[65].

**Područje primjene**

SQL baze su najbolje rješenje za podatke relacijske prirode, tj. podatke koji će se prirodno uklapati u međusobno povezane tablice. Najbolje rješenje je za kada je u integritet podataka prema CAP-u ključan. Relacijske baze su najbolje i kada će vertikalno skaliranje poslužitelja biti dovoljno za poslovne potrebe i rast podataka s vremenom. Budući da nisu građene s konceptima kao što su eventualna konzistentnost (objašnjeno u sljedećem poglavlju) i fragmentacija podataka na umu, razvoj aplikacija s relacijskom bazom kao potporom je nešto jednostavniji. Konzistentnost ažuriranja i pisanja je zajamčena SQL DBMS-ovima. Tehnologija je poprilično svestrana te nudi relativnu lakoću kod kompleksnih upita[66].

Poslovni sustavi u različitim korporacijama, podatkovni sustavi banaka i drugih institucija s financijskim transakcijama gdje je ACID ključan te bilo koje aplikacije gdje je većina podataka strukturirano su područja gdje SQL baze i dalje dominiraju.

[65] Anderson B., Nicholson B. (2021) SQL vs. NoSQL Databases: What's the Difference? IBM [online]. Dostupno na: https://www.ibm.com/cloud/blog/sql-vs-nosql [28. 7. 2021.]
[66] Anderson B., Nicholson B. (2021) SQL vs. NoSQL Databases: What's the Difference? IBM [online]. Dostupno na: https://www.ibm.com/cloud/blog/sql-vs-nosql [28. 7. 2021.]

## 3.2. Uvod u NoSQL baze

Nakon približavanja/ponavljanja osnova relacijske tehnologije, rad u nastavku skreće fokus na klasu sustava razvijenih radi upravljanja ogromnim količinama podataka u organizacijama kao što su Google, Amazon, Facebook i Twitter, za primjene poput društvenih mreža, korisničkih profila, marketinga i prodaje, online objava i tweetova, karta i drugih prostornih podataka te e-maila. Izraz **NoSQL** se najčešće interpretira kao "ne samo SQL" ("*Not Only SQL*"), dakle nije "Ne SQL". Mnoge aplikacije trebaju nove sustave kao komplement tradicionalnim SQL sustavima radi poboljšanja sposobnosti upravljanja podacima. Većina NoSQL sustava su distribuirane baze podataka s fokusom na spremanje polu-strukturiranih podataka, visoke performanse, dostupnost, replikaciju podataka i skalabilnost, nasuprot instantnoj konzistentnosti, snažnim jezicima za upite i skladištenju strukturiranih podataka. NoSQL sustavi generalno prilaze problemu **konzistentnosti** među više kopija (replika) podataka koristeći paradigmu poznatu kao *eventualna konzistentnost* (*eventual consistency*). Unutar CAP teorema, NoSQL sustavi stavljaju najveći fokus na **dostupnost**.

Četiri osnovne kategorije NoSQL baza podataka su ključ-vrijednost sustavi, graf-baze, stupčane baze te dokumentne baze, s time da neki sustavi ne pripadaju čisto u jednu od kategorije već posjeduju karakteristike dvije ili više[67].

NoSQL je prema Babić, Jakšić i Pošić (2019)[68] gotovo pokret, a nastao je zbog povećanja prometa na web stranicama i potrebe da se sprema sve više i više podataka. Među inicijatorima pokreta bili su Google sa svojom Bigtable bazom i Amazon s Dynamom, a postoje četiri osnovna atributa koji se smatraju razlozima NoSQL pokreta:

- Volumen – količina podataka koje je potrebno procesuirati i spremati u bazu
- Brzina – potreba da se podaci procesuiraju što prije moguće, nezavisno o njihovim količinama
- Varijabilnost – rigidne sheme i nedostatak dinamičnosti u relacijskim bazama stvaraju problem kod uvođenja promjena koje stoga postaju skupe (vremenski i financijski)
- Agilnost – potreba za jednostavnošću kod unosa i dohvata podataka iz baze (potreba za jednostavnijim upitima).

[67] Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow, str. 883
[68] Babić A.., Jakšić D., Poščić P. (2019) Querying Data in NoSQL Databases. Hrčak [online], str. 257 - 258. Dostupno na: https://hrcak.srce.hr/219991 [28. 7. 2021]

U sljedeća 4 poglavlja slijedi detaljnije objašnjenje osnovnih kategorija uz pregled nekih od implementacija konkretnih proizvođača, svaki sa svojim specifičnostima, no svi NoSQL sustavi imaju i neke zajedničke karakteristike, i to gledano s aspekta distribuiranih sustava (i baza) te s aspekta podatkovnih modela i jezika upita (*query languages*).

**NoSQL karakteristike vezane uz distribuirane sustave i baze podataka.** NoSQL sustavi naglašavaju visoku dostupnost, pa je replikacija podataka inherentna mnogima od ovih sustava. Skalabilnost je također bitna karakteristika, budući da mnoge aplikacije koje koriste NoSQL sisteme imaju podatke koji stalno rastu u volumenu. Visoke performanse su isto potrebna karakteristika, dok stalna konzistentnost često i nije važna kod NoSQL aplikacija.

1. Skalabilnost. Kao što je već objašnjeno u potpoglavlju 2.6.1., postoje dvije vrste skalabilnosti u distribuiranim sustavima: horizontalna i vertikalna. U NoSQL sustavima se generalno koristi horizontalna skalabilnost, gdje se distribuirani sustav širi dodavanjem još čvorova za skladištenje i za procesuiranje podataka kako raste volumen podataka. U NoSQL sustavima horizontalna skalabilnost se primjenjuje dok je sustav u operaciji, pa su nužne tehnike za distribuciju postojećih podataka među novim čvorovima bez prekidanja operacije ukupnog sistema.
2. Dostupnost, replikacija i eventualna konzistentnost. Mnoge aplikacije koje koriste NoSQL sustave traže kontinuiranu dostupnost sustava. Kako bi se to postiglo, podaci se repliciraju kroz dva ili više čvora na transparentan način, pa ako jedan čvor doživi kvar, podaci su i dalje dostupni na ostalima. Replikacija poboljšava dostupnost podataka, a osim toga može i poboljšati performanse čitanja, budući da zahtjevi za čitanjem (*read requests*) mogu biti ispunjeni s bilo kojeg od repliciranih čvorova. S druge strane performanse kod pisanja (*write*) postaju nezgrapne budući da ažuriranje mora biti obavljeno na svakoj od kopija podataka po čvorovima. Mnoge NoSQL aplikacije ne zahtijevaju istovremenu konzistentnost po svim čvorovima, već njeguju opušteniju paradigmu poznatu kao *eventualna konzistentnost* (*eventual consistency*) – podaci na svim kopijama po čvorovima će postati identični s nekim vremenskim odmakom.
3. Replikacijski modeli. Dva najvažnija replikacijska modela u SQL sustavima su *master-slave* (gospodar-sluga, nadređeni-podređeni, u nastavku će se koristiti izvorni engleski izraz) i *master-master* (gospodar-gospodar). Master-slave replikacija traži da jedna kopija bude glavna kopija, sve operacije pisanja (*write*) moraju biti primijenjene na *master* kopiji te se dalje propagirati na *slave* kopije, obično koristeći eventualnu

konzistentnost (*slave* kopije će *eventualno* postati identične *master* kopiji). Ako je dozvoljeno čitanje sa slave kopija, postoji mogućnost dohvata podataka koji nisu u tom trenutku najnoviji. Master-master replikacija, s druge strane, dozvoljava i čitanje i zapisivanje na svaku od replika, što ponovno ne jamči da će svako čitanje na različitim čvorovima s različitim kopijama vratiti identične podatke. Različiti korisnici mogu istovremeno izvršavati nove zapise na isti podatkovni objekt na različitim čvorovima sustava, pa će vrijednosti biti privremeno nekonzistentne. Kao dio master-master replikacijske sheme, važno je osmisliti metodu kako "pomiriti" konfliktne operacije pisanja na isti objekt u različitim čvorovima.

4. Fragmentacija datoteka (*sharding*). U mnogim NoSQL aplikacijama datoteke (ili kolekcije podatkovnih objekata) mogu imati milijune zapisa (ili dokumenata ili objekata), kojima istovremeno mogu pristupati tisuće korisnika. Stoga, nije praktično spremati čitavu datoteku na jedan čvor. Horizontalno particionairanje ili *sharding* često se koristi u NoSQL sustavima kako bi se teret pristupa datotekama podijelio između više čvorova. *Sharding* u kombinaciji s replikacijom fragmenata poboljšava balansiranje opterećenja sustav i dostupnost podataka.
5. Pristup podacima visokih performansi. U mnogim NoSQL aplikacijama, potrebno je pronaći individualne zapise ili objekte među milijunima. Kako bi se to postiglo, većina sustava koristi jednu od dvije tehnike: *hashing funkcije* i *particioniranje prema rasponima ključeva*. Većina pojedinačnih pristupanja objektima u NoSQL bazi biti će na temelju poznate vrijednosti ključa, umjesto prema uvjetima kompleksnih upita. Ključ objekta je jedinstveni indikator objekta dodijeljen od strane DBMS-a, ID. Kod *hashinga*, tzv. "*hash funkcija*" primjenjuje se na ključ, te izračunava gdje će objekt biti smješten (na kojem čvoru). Kod particioniranja prema rasponu ključeva, lokacija je određena prema rasponu vrijednosti ključeva objekata, tj. ako npr. ključevi imaju konvenciju imenovanja K001, K002, …, K100, i postoje četiri particije, prvih 25 ključeva biti će u prvoj particiji, drugih 25 u drugoj itd. Drugi indeksi mogu biti dodani radi pronalaženja objekata prema atributima različitima od ključeva[69].

**NoSQL karakteristike vezane uz podatkovne modele i jezike upita.** NoSQL sustavi daju prednost performansama i fleksibilnosti nad snažnim modeliranjem i kompleksnim upitima.

[69] Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow, str. 885 - 887

1. Nedostatak potrebe za shemom. Fleksibilnost koju pruža nezahtjevanje sheme u mnogim NoSQL sustavima ostvarena je kroz dozvoljavanje polu-strukturiranih, samo-opisujućih podataka. Korisnici mogu specificirati djelomičnu shemu u nekima od sustava radi poboljšanja efikasnosti spremanja, no nije nužno imati shemu u većini NoSQL sustava. Budući da nema sheme koja bi provodila ograničenja (*constraints*), bilokakva željena ograničenja moraju se programirati na razini aplikativnog softvera koji koristi podatke. Postoje različiti jezici za opisivanje polu-strukturiranih podataka, kao što su JSON (JavaScript Object Notation) i XML (Extensible Markup Language). JSON se koristi u više NoSQL sustava, a i u istraživačkom dijelu ovog rada.
2. Manje moćni jezici za upite. Mnoge aplikacije koje koriste NoSQL sustave nemaju potrebe za snažnim jezikom za upite kao što je SQL, budući da pretrage (*read* operacije) u tim sustavima često pronalaze pojedinačne objekte u određenoj datoteci, prema njihovim ključevima. NoSQL sustavi obično pružaju set funkcija i operacija u obliku API-ja kojemu se pristupa direktno iz programskog koda. Stoga se CRUD operacije (*create*, *read*, *update*, *delete* – stvaranje, čitanje, ažuriranje, brisanje) izvršavaju pozivanjem funkcija u kodu unutar programskog jezika poput JavaScripta, nasuprot pisanju SQL upita ili korištenju ORM-a. Neki NoSQL sustavi pružaju i svoj jezik za upite, obično (ali ne uvijek) s manje mogućnosti od SQL-a. Velika većina ih ne nudi nikakav oblik *join* operacija, pa one moraju biti implementirane na razini aplikacijskog koda.
3. Verzioniranje. Neki NoSQL sustavi nude spremanje više verzija podataka u svojim bazama, s vremenskim oznakama kada su stvoreni. To pruža mogućnost vraćanja na starija stanja baze u slučaju kvarova i grešaka[70].

## 3.3. Ključ-vrijednost sustavi

### *3.3.1. Osnovni koncept i modeliranje*

Ključ-vrijednost sustavi fokusiraju se na visoke performanse, dostupnost i skalabilnost, spremajući podatke u distribuirani sustav. Podatkovni model korišten u ključ-vrijednost bazama je relativno jednostavan, a u mnogima od tih sustava ne postoji ikakav jezik za upite, već set operacije koje aplikacijski programeri mogu koristiti. **Ključ** (*key*) je jedinstveni identifikator povezan sa stavkom podataka i on se koristi za rapidno lociranje te podatkovne

[70] Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow, str. 887

stavke. **Vrijednost** (*value*) je sama stavka podataka, a može biti u vrlo različitim formatima kod različitih sustava ovog tipa. U nekim slučajevima, vrijednost je vrste – *string* (niz znakova), pa programski kod same aplikacije mora interpretirati strukturu vrijednosti za stavke podataka. U drugim slučajevima, dopušten je neki od standardnih formata podataka, npr. redovi slični onima u relacijskoj bazi, JSON objekti ili neki drugi oblik samo-opisujućih podataka. Različiti ključ-vrijednost sustavi tako mogu spremati nestrukturirane, polu-strukturirane ili strukturirane podatke (što uključuje i slike, zvukove, internetske stranice, dokumente…). **Glavna karakteristika** ključ-vrijednost sustava je činjenica da svaka vrijednost (stavka podataka) mora biti asocirana s jedinstvenim ključem i da dohvat tih vrijednosti prema ključu mora biti ekstremno brz[71].

Najpoznatiji primjer ovakvog sustava je najvjerojatnije Redis (skraćenica za *Remote Dictionary Server*), brzi ključ-vrijednost podatkovni sustav otvorenog koda koji se koristi kao baza podataka, predmemorija (*cache*), broker za poruke te za redove čekanja. Ovaj sustav izabran je kao primjer za objašnjavanje čitave svoje kategorije NoSQL baza u ovom radu. Redis pruža odgovore ispod milisekunde, omogućujući obradu milijuna zahtjeva u sekundi za aplikacije koje se odvijaju u stvarnom vremenu (videoigre, financijske usluge, IoT tehnologije itd.). Specifičnost Redisa je što su podaci smješteni u radnu memoriju poslužitelja – RAM, što je, uz jednostavni podatkovni model, i razlog nevjerojatno brzih performansi kod prosječnih operacija čitanja ili pisanja. Redis koristi master-slave arhitekturu i podržava asinkronu (neistovremenu) replikaciju gdje se podaci mogu kopirati na više servera replika. To omogućava poboljšane performanse kod čitanja (budući da se zahtjevi mogu podijeliti između više poslužitelja) i brži oporavaka kada glavni server doživi kvar[72]. Podaci se dakle drže u memoriji radi održavanja brzina po kojima je Redis poznat, a ovisno kako se konfigurira, spremaju se (zapisuju) na disk nakon određenog broja sekundi. Moguće je specificirati npr. spremanje podataka u trajnu memoriju (disk) nakon svakih 100 promjena i najmanje 5 sekundi. Ako se sustav pritom sruši, posljednjih nekoliko promjena može biti izgubljeno[73].

Primjeri implementacija za ključ-vrijednost sustave su Redis, Voldemort, Riak i Amazonov Dynamo[74].

[71] Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow, str. 896
[72] Amazon Web Services: Redis. Dostupno na: https://aws.amazon.com/redis/ [1.8.2021.]
[73] Stojanović A. (2016) Osvrt na NoSQL baze podataka – četiri osnovne tehnologije. Hrčak [online], str. 46. Dostupno na: https://hrcak.srce.hr/192140 [28. 7. 2021]
[74] Database Zone: NoSQL Database Types. Dostupno na: https://dzone.com/articles/nosql-database-types-1 [29.7.2021.]

### *3.3.2. Upiti*

Prema Stojanoviću (2016)[75] gore opisani model podataka je po svojoj strukturi sličan rječniku (zapravo cijela baza funkcionira na način na koji su opisani indeksi u poglavlju o SQL bazama). Kao što je prethodno navedeno, ove baze nemaju jezik za upite, već omogućuju samo pronalaženje, dodavanje i uklanjanje parova na osnovi ključa. Općenito su podržane tri operacije:

- GET (ključ) – vraća vrijednost pridruženu zadanom ključu
- PUT (ključ, vrijednost) – Dodaje novi par ključ-vrijednost ili ako željeni ključ već postoji, pridružuje mu novu vrijednost, pritom "pregazeći" staru
- DELETE (ključ) – Uklanja par (ključ, vrijednost), a ako željeni ključ ne postoji, javlja grešku.

Stojanović dalje ističe dva temeljna pravila za ovakve baze:

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

### *3.3.3. Prednosti, mane i područje primjene*

**Prednosti**

- Fleksibilne podatkovne strukture
- Velike brzine kod pisanja i čitanja
- Pogodnost za korištenje kao *cache* (predmemorija)
- Jednostavnost korištenja (nepostojanje jezika za upite, osnovne operacije)
- API s velikim brojem podržanih programskih jezika koji mogu pozivati ugrađene operacije. Konkretno: Java, Python, PHP, C, C++, C#, JavaScript, Node.js, Ruby, R, Go i mnogi drugi[76].
- Građen oko replikacije (preko više poslužitelja) i perzistentnosti podataka – lako skaliranje, brzi oporavak izgubljenih podataka kod padova poslužitelja

**Mane**

[75] Stojanović A. (2016) Osvrt na NoSQL baze podataka – četiri osnovne tehnologije. Hrčak [online], str. 45 - 46. Dostupno na: https://hrcak.srce.hr/192140 [28. 7. 2021]
[76] Amazon Web Services: Redis. Dostupno na: https://aws.amazon.com/redis/ [1.8.2021.]

- Nedostatak jezika za upite, samim time i podrške za kompleksnije upite – moguć samo dohvat vrijednosti stavki prema ključevima
- Nepraktičnost za rad s kompleksnim modelima podataka
- Podaci u memoriji koji još nisu zapisani na disk mogu biti izgubljeni

**Područje primjene**

Amazon, koji nudi Redis u sklopu svojih cloud usluga, ističe mnoga područja za Redis (samim time i za ključ-vrijednost sustave općenito)[77].

- Caching (predmemorija). Redis je odličan izbor za implementaciju visoko dostupne predmemorija koja povećava brzinu dohvata (smanjuje latenciju), povećava propusnost i smanjuje opterećenje glavne BP (bila ona relacijska ili NoSQL). U praksi to znači da će se rezultati čestih upita na bazu i često korišteni objekti (poput slika, datoteka ili metapodataka) spremati u predmemoriju u Redisu, što znatno ubrzava njihov dohvat.
- Ljestvice najboljih igrača u videoigrama koje se ažuriraju u stvarnom vremenu.
- Spremanje sesija (npr. za web dućane).
- Analitika u stvarnom vremenu – procesuiranje i analiza podataka u stvarnom vremenu s latencijama ispod milisekunde za društvene mreže, targetirane oglase te rezultate senzora za IoT (*Internet of Things*) uređaje.

## 3.4. Graf-baze podataka

### *3.4.1. Osnovni koncept i modeliranje*

Grafne baze podataka su baze dizajnirane s ciljem davanja jednake važnosti vezama među podacima kao i samim podacima. Namijenjene su spremanju podataka bez ograničavanja unaprijed određenim modelom. Umjesto toga, podaci se spremaju kako se i skiciraju, pokazujući kako se svaki pojedinačni entitet povezuje s drugima. Dok relacijske baze računaju relacije u vrijeme upita kroz procesorski skupe JOIN operacije, grafna baza sprema te veze zajedno s podacima u modelu. Ova paradigma omogućava putanje kroz milijune konekcija po sekundi po procesorskoj jezgri[78].

[77] Amazon Web Services: Redis. Dostupno na: https://aws.amazon.com/redis/ [1.8.2021.]
[78] Neo4j dokumentacija. Dostupno na: https://neo4j.com/developer/ [1.8.2021.]

Graf je prema Stojanoviću (2016)[79] "*struktura podataka koja se sastoji od čvorova (nodes) i lukova (relationships) koji povezuju te čvorove. Čvorovi u takvim sustavim predstavljaju podatke, a lukovi odnose među njima.*".

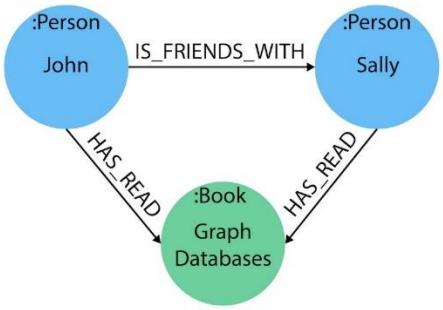
Upravo to su glavne komponente u najpopularnijoj implementaciji, Neo4j. **Čvorovi** su entiteti u grafu. Svaki od njih može sadržavati atribute (ključ-vrijednost parove) koji se nazivaju svojstva (*properties*). Čvorove je moguće dodatno označiti **naljepnicama**/oznakama/labelama (*labels*), koje predstavljaju njihove različite role u domeni. Te oznake mogu služiti i radi vezanja metapodataka (poput indeksa ili informacija o ograničenjima) određenim čvorovima.

**Lukovi** (*relationships*) pružaju direktno, imenovane, semantički relevantne veze između dva čvora-entiteta (npr. Zaposlenik RADI_ZA Kompaniju). Veza uvijek ima smjer, vrstu, početni čvor i krajnji čvor. Poput čvorova, i veze također mogu imati svojstva (*properties*). U većini slučajeva, veze imaju kvantitativna svojstva, poput težine, cijene, distance, ocjena, vremenskih intervala i sl. Zbog efikasnog načina spremanja veza, dva čvora mogu dijeliti bilo koji broj ili tip lukova bez žrtvovanja performansi[80].

**Naljepnice** (*labels*) su imenovani konstrukti u grafu koji se koriste za grupirati ili kategorizirati čvorove u skupove. Svi čvorovi označeni istom naljepnicom pripadati će istom skupu. Mnogi upiti mogu raditi s ovim skupovima umjesto s čitavim grafom, što olakšava pisanje tih upita i čini ih efikasnijima. Nema ograničenja broja naljepnica koje se mogu pridružiti čvoru, no nije obvezno dodati nijednu[81].

Slika ispod preuzeta je direktno iz dokumentacije Neo4j sustava, a predstavlja jednostavni model na kojemu su vidljivi svi gore opisani koncepti: čvorovi (pojedinačne osobe ili knjige s atributima te naljepnicama kojima su označeni kao osobe ili knjige) i lukovi koji predstavljaju odnose (prijateljstvo, pročitana knjiga) koji su dodatno objašnjeni atributima (npr. ocjene za knjige, datumi za zasnivanje prijateljstva ili dovršenje knjige).

[79] Stojanović A. (2016) Osvrt na NoSQL baze podataka – četiri osnovne tehnologije. Hrčak [online], str. 48. Dostupno na: https://hrcak.srce.hr/192140 [28. 7. 2021]
[80] Neo4j dokumentacija: What is a Graph Database? Dostupno na: https://neo4j.com/developer/guide-data-modeling/ [1.8.2021.]
[81] Neo4j dokumentacija: Graph Modelling Guidelines. Dostupno na: https://neo4j.com/developer/graph-database/ [1.8.2021.]




*Slika 5: Jednostavni Neo4j model. Izvor: Neo4j dokumentacija: Graph Modelling Guidelines? Dostupno na: https://neo4j.com/developer/graph-database/ [1.8.2021.]*

Neo4j pruža pune karakteristike baze podataka, uključujući usklađenost s principima ACID transakcija, podršku za klastere te za *runtime failover* – mogućnost detektiranja kvarova u sustavu te prebacivanje na i dalje aktivnu repliku bez gubitka funkcionalnosti[82].

Kao i druge NoSQL baze, Neo4j obećaje mogućnost neograničenog horizontalnog skaliranja, kroz metodu *shardinga* (horizontalnog particioniranja), pružajući jedinstveni pregled i upite preko više baza, kroz tehnologiju *Fabric* virtualnu *master* bazu koja upravlja ostalim geografski distribuiranim fragmentima na kojima je smještena ukupna grafna baza[83].

Primjeri graf-baza su Neo4j kao najpopularnija implementacija ove kategorije, Titan, JanusGraph i DGraph[84].

### *3.4.2. Upiti*

Ne može se govoriti o standardiziranom jeziku upita poput SQL-a, već svaki proizvođač nudi vlastito rješenje[85]. Neo4j koristi *Cypher* deklarativni jezik sličan SQL-u, ali optimiziran za grafove, koji su počele koristiti i druge baze kao što je SAP HANA Graph i Redis Graph, pa je moguće da je razvoj standarda u povojima[86]. Upiti u Cypheru su obično prezentirani kao putovanja kroz graf do specifičnih informacija. Babić, Jakšić i Poščić (2019)[87] daju sljedeći primjer upita za pretragu osoba koje žive u određenoj zemlji:

[82] Neo4j dokumentacija: What is a Graph Database? Dostupno na: https://neo4j.com/developer/guide-data-modeling/ [1.8.2021.]
[83] Neo4j dokumentacija: Database Scalability. Dostupno na: https://neo4j.com/product/neo4j-graph-database/scalability/ [1.8.2021.]
[84] Database Zone: NoSQL Database Types. Dostupno na: https://dzone.com/articles/nosql-database-types-1 [29.7.2021.]
[85] PhoenixNAP: What is a Graph Database. Dostupno na: https://phoenixnap.com/kb/graph-database [1.8.2021.]
[86] Neo4j dokumentacija: What is a Graph Database? Dostupno na: https://neo4j.com/developer/guide-data-modeling/ [1.8.2021.]
[87] Babić A.., Jakšić D., Poščić P. (2019) Querying Data in NoSQL Databases. Hrčak [online], str. 262. Dostupno na: https://hrcak.srce.hr/219991 [28. 7. 2021]

(:Person)-[:LIVES_IN]->(:City)-[:IS_LOCATED_IN]->(:Country),

U ovom primjeru se prolazi grafom kroz tri čvora (u običnim zagradama) i dva luka (u uglatim zagradama). Dvotočka i ime predstavljaju tip čvora (naljepnice), što čini upit efikasnijim jer je moguće ignorirati čvorove bez tih naljepnica. Dvotočke i imena u uglatim zagradama definiraju vrste odnosa. Znakom "->" definirano je da se radi o usmjerenim odnosima, a da odnosi u upitu nisu usmjereni koristilo bi se "-".

Babić, Jakšić i Poščić daju primjer[88] gdje Neo4j baze istinski mogu zasjati u odnosu na relacijske, a to je pretraživanje prijatelja od prijatelja u setu podataka od 1 000 000 ljudi. Relacijske baze snažno zaostaju u brzini upita što više u dubinu ide, kako je prikazano u sljedećoj tablici:

| Dubina | Relacijska baza | Neo4j | Broj vraćenih rezultata |
|---|---|---|---|
| 2 | 0,016s | 0,01s | ~ 2500 |
| 3 | 30,267s | 0,168s | ~ 110 000 |
| 4 | 1545,505s | 1,359s | ~ 600 000 |
| 5 | Nedovršeno | 2,132s | ~ 800 000 |

*Tablica 3: Usporedba RDBMS-a i Neo4j-a u pretraživanju prijatelja od prijatelja na velikom setu podataka. Izvor: Babić A.., Jakšić D., Poščić P. (2019) Querying Data in NoSQL Databases. Hrčak [online], str. 262. Dostupno na: https://hrcak.srce.hr/219991 [28. 7. 2021.]*

### *3.4.3. Prednosti, mane i područje primjene*

**Prednosti**

- Jednostavno i intuitivno modeliranje s lakom translacijom iz relacijskog modela u slučaju migracije iz SQL rješenja u grafno
- Fleksibilnost i jednostavnost dodavanja i uklanjanja čvorova, svojstava, lukova i naljepnica
- Ogroman broj mogućih područja primjene – jedno od najmanje nišama ograničenih NoSQL rješenja
- Sustav naljepnica, instance entiteta mogu pripadati različitim grupama, bez višestruke prisutnosti u različitim tablicama, već se samo istom entitetu, npr. nekoj osobi dodaju nove naljepnice kako se pridružuje novim skupinama u nekoj instituciji koja koristi grafnu bazu

[88] Babić A.., Jakšić D., Poščić P. (2019) Querying Data in NoSQL Databases. Hrčak [online], str. 263. Dostupno na: https://hrcak.srce.hr/219991 [28. 7. 2021]

- Posjedovanje svih ACID svojstava

**Mane**

- Nepostojanje standardnog jezika upita - jezik ovisi o korištenoj platformi
- Relativno mala baza korisnika, što može dovesti do problema s nalaženjem podrške u slučaju tehničkih problema[89]

**Područje primjene**

Nema nekog posebnog ograničenja područja primjena ovog tipa BP, a nekih od primjera koje ističe sam proizvođač Neo4j-a su:

- Preporuke u stvarnom vremenu – građenje sustava preporuka na temelju podataka o kupcima u e-commerceu ili streaming servisima poput Netflixa
- Vođenje internih organizacijskih shema u velikim korporacijama te građenje baza znanja, koje se ujedno mogu koristiti i za sustave korisničke podrške (osoblje koristeći aplikacije bazirane na grafnim bazama podataka lakše pronalazi potrebne informacije i brže rješava korisničke probleme)
- Otkrivanje prevara u stvarnom vremenu – banke, osiguravajuće kuće i mikrotransakcije u *gaming* industriji
- Grafne pretrage – ovakve BP dalju bolje rezultate korisničkih pretraga, budući da osim traženja samog unesenog pojma, korisnici često žele vidjeti i povezane, slične pojmove, a grafne baze su snažno usmjerene upravo ka vezama među entitetima[90]

## 3.5. Stupčane baze podataka

### *3.5.1. Osnovni koncept i modeliranje*

Od analiziranih tipova NoSQL rješenja, stupčane (*Column-oriented*) baze su najsličnije relacijskim bazama. Babić, Jakšić i Poščić (2019)[91] pišu kako se radi o bazama podataka gdje se koriste tablice – vrijednosti se spremaju u redove i stupce. Ove baze koriste identifikatore redova i stupaca kao ključeva koji se koriste za upite. Slične su na taj način proračunskim tablicama poput Excelovih (misli se na koordinate ćelija tipične za programe poput Excela – abecedna koordinata za stupce i numerička za redove koje zajedno tvore kompozitni ključ

[89] PhoenixNAP: What is a Graph Database. Dostupno na: https://phoenixnap.com/kb/graph-database [1.8.2021.]
[90] Neo4j dokumentacija: What is a Graph Database? Dostupno na: https://neo4j.com/developer/guide-data-modeling/ [1.8.2021.]
[91] Babić A.., Jakšić D., Poščić P. (2019) Querying Data in NoSQL Databases. Hrčak [online], str. 260 - 261

pojedinačnih ćelija). Ćelije u ovakvim bazama mogu sadržavati bilokakvu vrijednost, baš poput stavki u ključ-vrijednost sustavima. Ćelije mogu sadržavati i tzv. BLOB objekte a opet unutar tablice (*binary large objects* – najjednostavnije rečeno: slike, zvukove i druga multimedija u obliku koda), tako da ne ovise o eksternom sustavu datoteka za spremanje multimedije. U relacijskim bazama se, usporedbe radi, generalno u ćelije spremaju reference, adrese multimedijskih objekata spremljenih negdje u sustavu datoteka, a ne direktno u bazu. Stupci koji sadrže povezane podatke se zajedno grupiraju u obitelji stupaca (*column family*). To pruža odlične performanse pri određenim vrstama pretraga. S druge strane, mana je što se takvi veliki objekti ne mogu pretraživati upitima, pa je pažljiv dizajn redova i stupaca ključan. Prema Stojanoviću (2016)[92], za razliku od relacijskih baza, stupčani sustavi mogu pristupati pojedinačnim stupcima nezavisno od drugih stupaca, što je jedna od glavnih prednosti ovih baza s aspekta performansi. Vrijednosti koje pripadaju jednom stupcu na disku su smještene na jednom cjelovitom prostoru (*keyspace*), tako da je učitavanje jednog stupca u memoriji efikasno. Po potrebi se iz tih stupaca mogu sastaviti redovi, formirajući tablice kao u relacijskim bazama. Ovakvi sustavi efikasni su kada se upiti odnose samo na jedan manji broj stupaca stoga što ne moraju učitavati čitavu tablicu. Stojanović daje primjer koji približava korisnost ove implementacije sa zamišljenom tablicom koja zauzima 25GB diskovnog prostora. Neka postoje četiri stupca - Stupac A: 8GB, stupac B: 5GB, stupac C: 11GB i stupac D: 1GB. Ako upit koristi samo stupac D, tada je dovoljan samo 1GB prostora za taj upit, umjesto da DBMS mora prolaziti kroz čitavu tablicu od 25GB.

Stupčani sustavi generalno smještaju podatke za svaki stupac u zasebnu datoteku. Vrijednosti za svaki red tog stupca smještene su jedna do druge. S obzirom da I/O operacija učitavanja može učitati cijeli blok podataka odjednom, učitavanje stupaca je efikasno, posebno ako su potrebni svi podaci iz nekog stupca. Budući da su svi podaci iz jednog stupca istog tipa (*string, float, integer, date, BLOB, JSON*…), oni se mogu komprimirati, pa je slučaju efikasnost upita još veća jer se reduciraju I/O operacije. Zbog ovih tehničkih karakteristika Stojanović ističe kako su stupčani sustavi pogodni za obradu vrlo velikih količina podataka. Općenito je učitavanje i analiziranje svih vrijednosti jednog stupca prilično efikasno, kao i dodavanje i brisanje stupaca. Sve ovo čini stupčane baze idealnima za *batch* obrade – neinteraktive zadatke procesuiranja bez korisničkog sučelja, repetitivne poslove obrade velikog volumena prikupljenih podataka kada za to dođe vrijeme. Dobar primjer bili bi podaci o transakcijama na

[92] Stojanović A. (2016) Osvrt na NoSQL baze podataka – četiri osnovne tehnologije. Hrčak [online], str. 50 - 51. Dostupno na: https://hrcak.srce.hr/192140 [28. 7. 2021]

kreditnim karticama neke banke ili mjesečna izdaja prikupljenih faktura od strane neke kompanije koja se bavi veleprodajom[93]. S druge strane, učitavanje svih stupaca koji bi tvroili jedan red, ili dodavanje ili modifikacija na razini reda, nije efikasno, pa se takve operacije u ovim sustavima izbjegavaju koliko god je moguće[94] – ako poslovna pravila traže mnogo takvih operacija, bolje je jednostavno koristiti relacijsku bazu.

Popularan primjer stupčane baze podataka je Apache Cassandra, koji koristi podatkovni model građen oko i optimiziran prema upitima. Cassandra ne podržava relacijski podatkovni model namijenjen za relacijske baze, iako bi laiku zbog tabličnih prikaza dvije paradigme mogle biti slične (tj. preciznije kod stupčane baze se iz stupaca *mogu dobiti* redovi). U relacijskim bazama, podaci su smješteni u normalizirane tablice sa stranim ključevima kojima se referenciraju povezani podaci u drugim tablicama. Upiti koje aplikacije koriste su pogonjeni strukturom tablica, a povezani podaci se prikazuju JOIN operacijama među tablicama.

Cassandra automatizira mnoge procese bitne za performanse i skalabilnost. Automatski pomoću ugrađenog *hashing* algoritma odlučuje gdje (na kojima čvorovima) spremati podatke, i izbjegava potrebu za centraliziranim *master* čvorom koji donosu odluke o spremanju (već se one donose na razini pojedinačnih čvorova). Sustav je visoko skalabilan i moguće mu je lagano poboljšati performanse dodavanjem još strojeva, baš zbog spomenutog nedostatka *master* čvora koji bi morao biti dovoljno jak da orkestrira upravljanje podacima. To znači da svi čvorovi mogu biti jeftiniji strojevi poslužitelji, tzv. *commodity serveri*. Još jedan razlog za skalabilnost je smanjeni naglasak na konzistentnost podataka, koji bi također zahtijevao *master* čvor koji će tu konzistentnost pratiti i provoditi na temelju pravila. Umjesto toga, čvorovi međusobno automatski komuniciraju pomoću tehnologije poznate kao "*gossip protocol*" i međusobno si šalju metapodatke, što čini dodavanje novih čvorova trivijalnim[95].

U Cassandri, modeliranje podataka je pogonjeno upitima. Uzorci pristupa podacima i upita iz aplikacija određuju strukturu i organizaciju podataka kod dizajna baze. Podaci su dakle modelirani kako bi se najbolje uklapali u neki upit ili set upita. JOIN operacije nisu podržane u Cassandri, pa će sva potrebna polja (stupci) biti grupirane u jedinstvenu tablicu. Podaci se

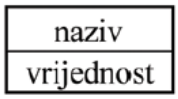
[93] Talend: Beginner's Guide to Batch Processing. Dostupno na: https://www.talend.com/resources/batch-processing/ [1.8.2021.]
[94] Stojanović A.(2016) Osvrt na NoSQL baze podataka – četiri osnovne tehnologije. Hrčak [online], str. 51. Dostupno na: https://hrcak.srce.hr/192140 [28. 7. 2021]
[95] Ubuntu Blog: What is Cassandra and Why are Big Tech Companies using it? Dostupno na: https://ubuntu.com/blog/apache-cassandra-top-benefits [1.8.2021.]

stoga ponavljaju kroz više tablica – proces denormalizacije. Duplikacija podataka i visoka propusnost pisanja se koriste za postizanje jakih performansi čitanja[96].

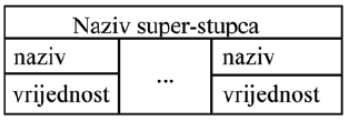
Cassandrin model podataka ne poznaje koncepte stranih ključeva niti relacijskog integriteta, baziran je oko upita, denormaliziran te se u slučajevima kada niti denormalizirani duplicirani podaci ne mogu u potpunosti integrirati kompleksne odnose među entitetima za neki upit, oslanjaju na kod s aplikacijske strane[97].

Stojanović (2016) definira Cassandrin podatkovni model kroz četiri osnovne komponente na sljedeći način[98]:

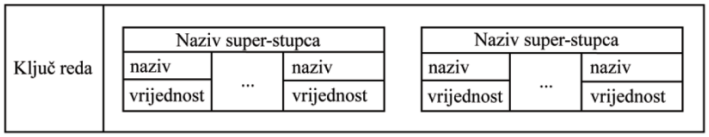
"*Najjednostavnija podatkovna struktura Cassandrinog modela je stupac, sastavljen od naziva i vrijednosti. Ti se stupci mogu kombinirati u veće strukture – super-stupce, koji se sastoje od sortiranog niza stupaca. Stupci se nalaze u redovima, a red koji se sastoji samo od stupaca naziva se obitelj stupaca. Kada se u redu nalaze super-stupci, onda se to naziva obitelj super-stupaca. Općenito, obitelj stupaca koristi se za podatke koji su na neki način povezani, kao što su osobni detalji i vrsta posla za neku osobu.*"




*Slika 6: Cassandra stupac. Izvor: Stojanović A.(2016) Osvrt na NoSQL baze podataka – četiri osnovne tehnologije. Hrčak [online]. Dostupno na: https://hrcak.srce.hr/192140 [28. 7. 2021]*

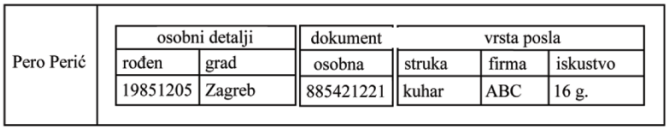



*Slika 7: Cassandra super-stupac. Izvor: Stojanović A.(2016) Osvrt na NoSQL baze podataka – četiri osnovne tehnologije. Hrčak [online]. Dostupno na: https://hrcak.srce.hr/192140 [28. 7. 2021]*




*Slika 8: Cassandra obitelj (super) stupaca. Izvor: Stojanović A.(2016) Osvrt na NoSQL baze podataka – četiri osnovne tehnologije. Hrčak [online]. Dostupno na: https://hrcak.srce.hr/192140 [28. 7. 2021]*

[96] Apache Cassandra dokumentacija. Dostupno na: https://cassandra.apache.org/doc/latest/cassandra/data_modeling/intro.html [1.8.2021.]
[97] Apache Cassandra dokumentacija. Dostupno na: https://cassandra.apache.org/doc/latest/cassandra/data_modeling/intro.html [1.8.2021.]
[98] Stojanović A.(2016) Osvrt na NoSQL baze podataka – četiri osnovne tehnologije. Hrčak [online], str. 51 – 52. Dostupno na: https://hrcak.srce.hr/192140 [28. 7. 2021]




*Slika 9: Primjer podataka o osobi u Cassandri. Izvor: Stojanović A.(2016) Osvrt na NoSQL baze podataka – četiri osnovne tehnologije. Hrčak [online]. Dostupno na: https://hrcak.srce.hr/192140 [28. 7. 2021]*

Najvažnije baze ovog tipa na tržištu su HBase, Apache Cassandra, Hypertable i "pradjed" svih BP ove kategorije, Google BigTable[99].

### *3.5.2. Upiti*

Cassandra za upite koristi *Cassandra Query Language*[100] – CQL, jezik sličan SQL-u, koji služi za stvaranje ažuriranje sheme i pristup podacima. CQL omogućuje korisnicima organizaciju podataka unutar clustera Cassandra čvorova koristeći:

- **Keyspace** (ekvivalent pojedinačnim SQL bazama[101]) – definira kako se (koliko puta i po kojima) set podataka replicira po čvorovima. Keyspace-ovi sadrže tablice.
- **Obitelji stupaca (*column family*)** (ekvivalent SQL tablicama[102]) – definira shemu za kolekciju particija. Tablice sadrže particije, koje sadrže redove, koji sadrže stupce. Cassandrine tablice mogu fleksibilno dodavati nove stupce s ništa vremena nedostupnosti.
- **Particije** – definiraju obvezni dio primarnog ključa koji svi redovi u Cassandri moraju imati kako bi se identificirao čvor gdje je neki red spremljen. Svi upiti moraju sadržavati ključ particije (*partition key*). Particijski ključ se koristi za dijeljenje podataka po čvorovima, koristeći *hashing* funkcije.
- **Redove** - sadrže kolekciju stupaca identificiranu preko primarnog ključa kojega čini ključ particije i opcionalni ključ klastera.
- **Stupce** - podatak sa zadanim tipom koji pripada redu.

[99] Database Zone: NoSQL Database Types. Dostupno na: https://dzone.com/articles/nosql-database-types-1 [29.7.2021.]
[100] Apache Cassandra dokumentacija. Dostupno na: https://cassandra.apache.org/doc/latest/cassandra/architecture/overview.html [1.8.2021.]
[101] Insight Data Science: The Total Newbie's Guide to Cassandra. Dostupno na: https://blog.insightdatascience.com/the-total-newbies-guide-to-cassandra-e63bce0316a4 [1.8.2021.]
[102] Insight Data Science: The Total Newbie's Guide to Cassandra. Dostupno na: https://blog.insightdatascience.com/the-total-newbies-guide-to-cassandra-e63bce0316a4 [1.8.2021.]

### *3.5.3. Prednosti, mane i područje primjene*

**Prednosti**

- Pogodnost za obradu vrlo velikih količina podataka
- Brzina pri *batch* obradama
- Brzina kod dohvata podataka jednog stupca
- Snažna dostupnost prema CAP teoremu
- Lakoća i niski troškovi skaliranja

**Mane**

- Nedostatak stranih ključeva i referencijskog integriteta
- Nemogućnost JOIN operacija među keyspace-ovima, sporost kod učitavanja svih stupaca istog keyspace-a koji bi tvorili jedan red
- Relativno spora eventualna konzistentnost – mogućnost kontradiktornih podataka po čvorovima

**Područje primjene**

- Podaci vremenskih serija – Cassandra je odlična u spremanju podataka vremenskih serija, gdje stare podatke ne treba nikad ažurirati. Primjer su dnevnici (*logs*) cloud infrastrukture i aplikacija, te podaci raznih industrijskih senzora i IoT tehnologija
- Globalno distribuirani podaci: lokalni Cassandra cluster može spremati podatke, a sustav postići konzistentnost kasnije. Ne postoji *master* čvor, lagano je dodavati nove čvorove, pa je jeftina geografska ekspanzija baze podataka moguća[103]

## 3.6. Dokumentne baze podataka

### *3.6.1. Osnovni koncept i modeliranje*

Dokumentne baze podataka podatke spremaju u JSON, BSON ili XML dokumente. U dokumentnoj baze, dokumenti mogu biti "ugniježđeni" jedni u drugima (*nested*). Korisnik može odrediti elemente za indeksiranje radi bržih upita.

[103]Ubuntu Blog: What is Cassandra and Why are Big Tech Companies using it? Dostupno na: https://ubuntu.com/blog/apache-cassandra-top-benefits [1.8.2021.]

Dokumenti se spremaju i dohvaćaju u obliku koji je mnogo bliži podatkovnim objektima kakvi se koriste u izvornom kodu aplikacija (detaljnije objašnjeno u poglavlju 4.4. o JavaScriptu), što znači da je potrebno manje prevođenja kako bi se podaci koristili u aplikaciji. SQL podaci moraju biti često sastavljati i rastavljani kada se kreću naprijed-nazad između aplikacije i baze.

Dokumentne baze su popularne među programerima zbog svoje fleksibilnost koja omogućuje dorađivanje struktura dokumenata prema evoluirajućim zahtjevima aplikacije. Ta fleksibilnost ubrzava razvoj jer podaci efektivno postaju kao kod i pod kontrolom su programera. U relacijskim bazama, može biti potrebna intervencija administratora baza kako bi se promijenila njihova struktura.

Sve najviše korištene dokumentne baze imaju arhitekturu usmjerenu ka skalabilnost glede volumena podataka i prometa.

Područja primjene uključuju e-commerce platforme, platforme za trgovanje i razvoj mobilnih aplikacija u različitim industrijama[104].

Vodeća dokumentna baza je MongoDB. Građena je na distribuiranoj, horizontalno skalirajućoj arhitekturi i nudi cloud platformu za upravljanje podacima i pružanje podatkovne potpore aplikacijama. MongoDB upravlja transakcijskim, operacijskim i analitičkim radnim opterećenjima na svim razinama skaliranja. Nudi potpunu izolaciju svojih dokumenata kod ažuriranja, dakle u potpunosti posjeduje ACID svojstva – bilokakve pogreške će izazvati otkazivanje operacije ažuriranja i konzistentan pogled na dokument za korisnika. Nudeći transakcije preko više dokumenata, jedna je od rijetkih baza koje kombiniraju jamstvo ACID-a tradicionalnih relacijskih baza s brzinom, fleksibilnošću i snagom NoSQL modela.

MogoDB je baziran na distribuiranoj arhitekturi koja omogućava skaliranja preko velikog broja čvorova, koji mogu biti manji i financijski prihvatljivi strojevi. Koristi vlastitu tehnologiju horizontalnog particioniranja (*sharding*) preko instanci u clusteru, gdje su pritom dokumenti najmanje nezavisne jedinice koje se ne razbijaju preko više poslužitelja. Dokumente je moguće u DBMS-u označiti kako bi uvijek bili fizički spremljeni u određenim državama ili regijama, bliže klijentima koji će upućivati zahtjeve na te poslužitelje. To smanjuje latenciju dohvata podataka i u nekim situacijama osigurava pridržavanje zakonskih ograničenja glede gdje se ti podaci legalno mogu skladištiti. Svaka MongoDB particija (*shard*) se ponaša kao niz replika: sinkronizirani cluster tri ili više individualnih poslužitelja koji kontinuirano kopiraju podatke

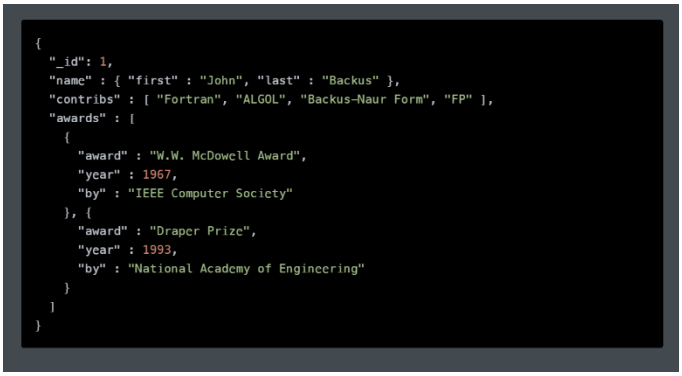
[104] MongoDB dokumentacija: Understanding the Different Types of NoSQL Databases. Dostupno na: https://www.mongodb.com/scale/types-of-nosql-databases [1.8.2021.]

među sobom, nudeći redundanciju i osiguravajući distribuirani sustav od nedostupnosti zbog kvarova ili održavanja.

MongoDB ima vlastiti set kontrola i integracija za cyber sigurnost i za verziju upravljanu direktno od klijenata i za cloud verziju[105].

Temelj ove baze su JSON objekti (*JavaScript Object Notation*), podatkovne strukture u JavaScript programskom jeziku, jednostavni asocijativni kontejneri gdje je ključ tipa *string* mapiran na vrijednost (koja može biti broj, *string*, funkcija ili čak drugi objekt).

Kako je JavaScript postao standardni jezik klijentske strane web razvoja, JSON se kao podatkovna struktura koja je i ljudski i strojno čitljiva proširio u sve pore softvera. Danas se koristi za API-je, konfiguracijske datoteke, aplikacijske dnevnike (*logs*) i baze podataka. MongoDB koristi posebno razvijenu verziju JSON-a poznatu kao BSON (*Binary JSON*), krajnjem korisniku identičnu JSON-u no optimiziranu za brži rad u bazama podataka[106].




*Slika 10: Primjer JSON objekta. Izvor: MongoDB dokumentacija: Comparing MongoDB vs PostgreSQL. Dostupno na: https://www.mongodb.com/compare/mongodb-postgresql [1.8.2021.]*

Shema kod baze utemeljene ja JSON-u/BSON-u je dinamička i fleksibilna u usporedbi s rigidnim tabličnim podatkovnim modelom relacijskih baza. JSON dokumenti su polimorfni – polja mogu varirati između dokumenata unutar jedne kolekcije (što nikako nije moguće s redovima u tablici). Nema nikakve potrebe za unaprijed deklariranom strukturom dokumenata kod kreacije baze, dokumenti su samo-opisujući te je dovoljno samo krenuti s pisanjem koda. Ako dođe potreba za dodavanjem novog polja u neki dokument, moguće je to napraviti bez utjecaja na ostale dokumente u kolekciji, bez ažuriranja centralnog kataloga DBMS-a i bez

[105] MongoDB dokumentacija: JSON and BSON. Dostupno na: https://www.mongodb.com/json-and-bson [1.8.2021.]
[106] MongoDB dokumentacija: Comparing MongoDB vs PostgreSQL. Dostupno na: https://www.mongodb.com/compare/mongodb-postgresql [1.8.2021.]

razdoblja nedostupnosti za održavanje baze – nema "skupe" ALTER TABLE operacije ili, još gore, potpunog redizajniranja sheme[107].

Struktura u ovom tipu baze je vrlo jednostavna: baza podataka može sadržavati jednu ili više **kolekcija** (MongoDB ekvivalent tablica u relacijskim bazama). Kolekcije ne zahtijevaju da njihovi dokumenti imaju istu shemu, tj. dokumenti u jednoj kolekciji ne moraju imati isti set polja i tipova podataka za polja. Ipak, ako za time postoji potreba, moguće je primijeniti validacijska pravila za dokumente unutar kolekcije kroz operacije ažuriranja i umetanja. Baze i kolekcije se ne moraju kao u relacijskim bazama eksplicitno unaprijed definirati, automatski se kreiraju prvim korištenjem, tj. umetanjem prvih kolekcija/dokumenata. Kolekcijama se dodjeljuje nepromjenjivi ID, koji ostaje isti kroz sve članove seta replika u particioniranom clusteru[108].

MongoDB podržava normalizirane podatkovne modele kroz relacije koristeći reference između dokumenata. U ovoj bazi se češće ide na direktno "ugnježđivanje" potrebnih podataka u neki dokument, no postoje situacije gdje takvi duplirani podaci ne bi nudili adekvatne performanse tijekom čitanja i kod kompleksnih M:N relacija. Nude se dvije ugrađene funkcije za izvođenje JOIN operacija - *$lookup* i *$graphLookup*[109].

API ove baze službeno podržava jako velik broj programskih jezika – C, C++, C#, Go, Javu, Node.js, PHP, Python, Ruby, Rust, Scalu, Swift, s *community* rješenjima otvorenog koda za druge[110].

Najpopularnije dokumentne baze podataka su MongoDB i CouchDB[111].

### *3.6.2. Upiti*

MongoDB koristi jednostavni i moćni jezik upita zvan MQL (*MongoDB Query Language*) koji podržava sve funkcionalnosti SQL-a, uključujući JOIN operacije. Radi se zapravo o setu naredbi koje se preko API-ja pišu direktno u aplikativnom kodu, a sintaksom podsjećaju na JavaScript i slične jezike.

---

[107] MongoDB dokumentacija: JSON and BSON. Dostupno na: https://www.mongodb.com/json-and-bson [1.8.2021.]
[108] MongoDB dokumentacija: Databases and Collections. Dostupno na: https://docs.mongodb.com/manual/core/databases-and-collections/ [1.8.2021.]
[109]MongoDB dokumentacija: Data Model Design. Dostupno na: https://docs.mongodb.com/manual/core/data-model-design/ [1.8.2021.]
[110] MongoDB dokumentacija: Start Developing with MongoDB. Dostupno na: https://docs.mongodb.com/drivers/ [1.8.2021.]
[111] Database Zone: NoSQL Database Types. Dostupno na: https://dzone.com/articles/nosql-database-types-1 [29.7.2021.]


*Slika 11: Umetanje dokumenta u kolekciju u MQL-u. Izvor: MongoDB dokumentacija: MongoDB CRUD Operations. Dostupno na: https://docs.mongodb.com/manual/crud/ [1.8.2021.]*


*Slika 12: Operacija pretraživanja po koleciji u MQL-u. Izvor: MongoDB dokumentacija: MongoDB CRUD Operations. Dostupno na: https://docs.mongodb.com/manual/crud/ [1.8.2021.]*

### *3.6.3. Prednosti, mane i područje primjene*

**Prednosti**

- Spremanje i dohvat podataka u obliku koji je mnogo bliži objektima kakvi se koriste u izvornom kodu aplikacija
- Brzina i fleksibilnost tipična za NoSQL baze kombinirana sa zadržanim ACID svojstvima
- Neograničeno horizontalno skaliranje
- Dinamična i fleksibilna shema
- Podržava referenciranje i normalizaciju ako za time postoji potreba, te JOIN operacije preko više kolekcija

**Mane**

- Sporije operacija pisanja i ažuriranja kod distribuiranih sustava – veći broj instanci koje treba ažurirati

**Područje primjene**

Ne postoji nikakvo stvarno ograničenje područja korištenja za MongoDB i druge dokumentne baze, službena MongoDB dokumentacija ističe sljedeće kao neke od primjera: IoT, mobilne aplikacije, katalozi proizvoda, analitika u stvarnom vremenu, gaming i platni promet[112].

# 4. Ključne tehnologije za izgradnju web aplikacije

U nastavku su ukratko opisane tehnologije korištene za razvoj aplikacije JALA. Aplikacija je bazirana na standardnom setu web tehnologija – HTML, CSS i JavaScript te na bazičnoj klijentsko – poslužiteljskoj arhitekturi. To omogućava univerzalnu pristupačnost i funkcionalnost preko različitih platformi (računala s bilo kojim od glavnih operativnih sistema te mobilnih telefona i tableta), budući da se koristi preko web preglednika, bez potrebe za lokalnom instalacijom. U ovoj prototipnoj iteraciji aplikacije, funkciju baze podataka i osiguravanja perzistentnosti podataka kroz sesije (paljenja i gašenje web preglednika ili samog uređaja) omogućava lokalna memorija web preglednika, tzv. LocalStorage, koja je zbog svog podatkovnog modela i jednostavnog API-ja idealna razvojna NoSQL baza za ovakav projekt.

## 4.1. Klijentsko – poslužiteljska arhitektura i web aplikacije

Distribuirani sustavi kojima se pristupa preko interneta su organizirani kao klijentsko-poslužiteljski (*client-server*) sustavi. U takvom sustavu, korisnik komunicira s programom koji se pokreće na njegovom lokalnom uređaju, npr. u web pregledniku na računalu ili u aplikaciji na mobilnom uređaju. Taj program pak komunicira s programom na udaljenom računalu, web poslužitelju (serveru). Udaljeno računalo pruža servise, poput pristupa web stranicama koje su dostupne eksternim klijentima. Klijentsko-poslužiteljski model je generalni arhitekturalni model za aplikacije i nije ograničen samo na aplikacije distribuirane preko više strojeva. Moguće ga je koristiti i kao model za logiku interakcije gdje su i klijent i poslužitelj na istom računalu (što i je česta praksa u razvoju softvera – virtualni poslužitelj za razvoj, kopiranje koda na testno pa produkcijsko okruženje u različitim fazama dovršenosti).

[112] MongoDB dokumentacija: MongoDB Use Cases. Dostupno na: https://www.mongodb.com/use-cases [1.8.2021.]

Klijentsko – poslužiteljska arhitektura bazirana je na jasnoj razdvojenosti između prezentacije informacija i izračuna koji stvaraju i obrađuju te informacije. Obično se takva arhitektura dijeli na četiri međusobno komunicirajuća logička sloja:

1. *Prezentacijski sloj* koji se bavi prezentiranjem informacija korisniku te upravljanjem svim korisničkim interakcijama (inputima).
2. *Sloj za upravljanje podacima* koji osigurava da se podaci šalju od i prema klijentu. Ovaj sloj može implementirati i razne validacije na podacima, generirati web stranice itd.
3. *Sloj za aplikacijsko procesuiranje* koji se bavi implementacijom logike aplikacije i osigurava potrebnu funkcionalnost krajnjim korisnicima.
4. *Sloj baze podataka* koji sprema podatke, upravlja transakcijama i upitima[113].

U developerskoj praksi često se koriste i pojmovi *front-end* (FE) i *back-end* (BE). Front-end odnosi se na prvu točku gornjeg popisa, tj. korisničko sučelje, a back-end na preostale tri, poslužitelja, aplikaciju i bazu podataka koji rade iza scena kako bi dostavili podatke korisniku[114].

## 4.2. HTML (Hypertext Markup Language)

Jedna od ključnih tehnologija za izgradnju web stranica je označni jezik imena HTML (Hypertext Markup Language). HTML opisuje strukturu web stranica, tj. pruža autorima mogućnost:

- Izdavanja online dokumenata s naslovima različitih hijerarhijskih pozicija, poglavljima teksta, tablicama, listama, slikama itd.
- Dohvata online informacija preko *hypertext* poveznica, klikom na gumb.
- Dizajniranje formi za izvršavanja transakcija sa udaljenim servisima, za pretraživanje informacija, stvaranje rezervacija, narudžbu proizvoda itd.
- Uključivanja proračunskih tablica, videa, zvukova i druge multimedije direktno u svoje dokumente.

S HTML-om, autori opisuju strukturu stranica koristeći oznake (*markup*). *Elementi* koje jezik koristi označavaju dijelove sadržaja kao što su "odjeljak" (*paragraph*, <p>), "lista" (*list*, <ul>), "tablica" (*table*, <table>) i slično[115].

[113] Sommerville I. (2016) Software Engineering, Pearson, Harlow, str. 499 - 500
[114] Concepta: What is the Difference between Front-End and Back-End Development? Dostupno na: https://www.conceptatech.com/blog/difference-front-end-back-end-development [31.7.2021.]
[115] HTML & CSS. Dostupno na: https://www.w3.org/standards/webdesign/htmlcss [15.7.2021.]


*Slika 13: Primjer HTML koda. Izvor: Vlastita izrada autora.*

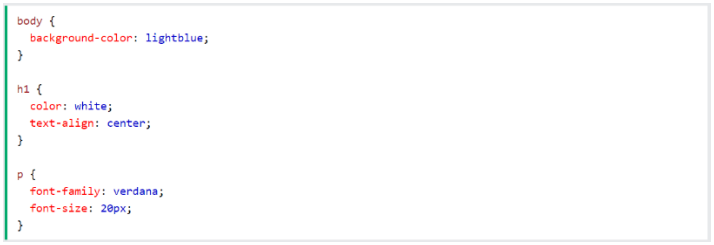
HTML kod je osnova svih web stranica i aplikacija. U ogromnoj većini slučajeva se s poslužitelja klijentu isporučuju datoteke sa sufiksom .html, koje svi moderni web preglednici mogu otvarati. Iako postoji veći broj proizvođača web preglednika s različitim programskim implementacijama, HTML je standard te će stvarati identičan prikaz nezavisno radi li se o Google Chromeu, Operi, Safariju ili nekom drugom web pregledniku. Za održavanje tog važnog standarda koji omogućava moderni Internet, brinu se dvije koordinirane organizacije: Web Hypertext Application Technology Working Group (WHATWG) i World Wide Web Consortium (W3C). HTML je živući, evoluirajući standard, te se danas nalazi u svojoj petoj iteraciji (HTML5)[116].

## 4.3. CSS (Cascading Style Sheets)

Druga ključna tehnologija za izradu web stranica i aplikacija je CSS – Cascading Style Sheets. CSS je jezik za opisivanje prezentacije web stranica, uključujući boje, raspored i razmake među elementima, fontove, veličine teksta, veličine i oblike elemenata itd. Omogućava prilagodbu prezentacije različitim vrstama uređaja, ovisno o veličinama ekrana. CSS je nezavisan od HTML-a (obično se nalazi u zasebnim datotekama sa sufiksom .css, iako se može pisati i unutar HTML datoteka) što se naziva odvajanjem strukture (sadržaja) od prezentacije. To razdvajanje čini održavanje stranica lakšim, daje mogućnost dijeljenja .css datoteka među više stranica te prilagođavanje stranica različitim okolinama[117].

[116] Library of Congress: HyperText Markup Language (HTML) 5. Dostupno na: https://www.loc.gov/preservation/digital/formats/fdd/fdd000481.shtml [28.7.2021.]
[117] HTML & CSS. Dostupno na: https://www.w3.org/standards/webdesign/htmlcss [15.7.2021.]

CSS nije programski jezik u klasičnom smislu budući da ima vrlo specifičnu namjenu baziranu isključivo na definiranju i opisivanju, s gotovo ništa mogućnosti kalkulacija, rada s podatkovnim strukturama i sl.




*Slika 14: Primjer CSS koda. Izvor: W3 Schools - CSS Tutorial. Dostupno na: https://www.w3schools.com/css/ [16.7.2021.]*

Riječ "kaskadno" (*cascading*) u imenu jezika odnosi se na temeljnu paradigmu jezika – stil koji je primijenjen na HTML element "roditelj" će se propagirati niz lanac na svu njegovu "djecu", osim ako se za "djecu" ne specificira drugi stil. Ako se za *body* element odredi plava boja teksta i veličina fonta od 12 piksela, svi naslovi, paragrafi i drugi elementi unutar *body* elementa će također posjedovati ta svojstva, osim ako im se specifično ne definiraju drugačije vrijednosti[118].

Kao konstantno evoluirajuća tehnologija, CSS je trenutno u svojoj trećoj verziji (CSS3).

## 4.4. JavaScript - Jezik interneta koji omogućuje web aplikacije

JavaScript je memorijski neintenzivni, interpretirani (kompajlira se tijekom, a ne prije izvođenja – nema potrebe za stvaranjem izvršnih datoteka prije pokretanja aplikacija) programski jezik. Najpoznatiji je po svojoj izvornoj svrsi, kao skriptni jezik za web stranice – omogućava dinamičko mijenjanje HTML i CSS koda (strukture i izgleda) bez učitavanja novih datoteka s poslužitelja. Kroz godine (stvoren 1995.) je uvelike evoluirao, pa se danas uz tehnologije kao što su Node.js i Electron.js može koristiti i kao serverski jezik te za izradu desktop aplikacija. Najčešće (između ostaloga i u projektnom dijelu ovog rada) JavaScript se izvodi na klijentskoj strani weba, dakle u korisnikovom web pregledniku, radi kontroliranja ponašanja web stranice/aplikacije u pri raznim događajima (*events*). To se događa pomoću JavaScript mehanizma (*engine*) koji svi moderni web preglednici imaju, kojemu se pristupa pomoću javno dostupnog API-ja. Unatoč kroz godine razvijenoj multifunkcionalnosti, web

[118] W3 Schools – HTML Styles - CSS. Dostupno na: https://www.w3schools.com/html/html_css.asp [16.7.2021.]

preglednici ostaju najčešća okolina (*runtime*) za izvođenje JavaScript koda, s namjenom manipulacije DOM-a (*Document-Object Model*)[119].

Skriptni kod izvršava se u trenutku preuzimanja web stranica s poslužitelja, ili se pokreće kao reakcija na neke događaje (korisnik pritisne gumb, korisnik drži strelicu miša iznad nekog HTML elementa, prolaz nekog određenog vremenskog intervala). Skriptiranje čini stranice više dinamičkima, kroz dodavanje i uklanjanje HTML elemenata ili mijenjanje njihovog izgleda kroz manipulaciju CSS-a, ili čak slanjem i primanjem podataka s poslužitelja bez ponovnog učitavanja stranice (pomoću AJAX-a – *Asynchronous JavaScript and XML*). Ova dodatna interaktivnost čini ponašanje web stranica sličnijom tradicionalnim softverskim aplikacijama. Takve web stranice se često nazivaju **web aplikacijama**[120].

Poput svih drugih programskih jezika, JavaScript u sebi ima ugrađene razne tipove podatkovnih struktura. Prema McMillanu (2014)[121], svaki računalni program se u osnovi sastoji od nekog oblika podatkovnih struktura za upravljanje podacima za čije manipuliranje je pisan, te od algoritama koji te strukture pretvaraju iz njihovog *input* oblika u željeni *output* oblik.

Podatkovne strukture relevantne za opseg ovog rada su objekti (*objects*) i nizovi (*array*s).

Nizovi (*arrays*) su linearne kolekcije elemenata kojima se pristupa pomoću indeksa, cijelih brojeva koji indiciraju udaljenost elementa od prvoga elementa u nizu[122]. Primjer *array*-a u JavaScriptu:

```
životinje = ['pas', 'mačka', 'kokoš', 'miš', 'plavetni kit']
```

Objekti su podatkovne strukture koje spremaju podatke u obliku ključ-vrijednost parova, kao što npr. telefonski imenik sprema svoje podatke kao ime i broj telefona. Kada se traži neki telefonski broj, prvo se pretražuje ime, a kada se ime pronađe, broj telefona će biti odmah kraj njega. Ključ je dakle element koji se koristi za izvršavanje pretrage, a vrijednost je rezultat te pretrage. Za razliku od nizova, redoslijed parova kod objekata nije bitan niti postoje indeksi. Valja napomenuti da ista ili vrlo slična podatkovna struktura postoji u mnogim drugim programskim jezicima, te se u njima često naziva *dictionary*[123]. Primjer objekta u JavaScriptu:

---

[119] MDN Web Docs: About JavaScript. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript [28.7.2021.]
[120] JavaScript Web APIs. Dostupno na: https://www.w3.org/standards/webdesign/script.html [28.7.2021.]
[121] McMillan M. (2014) Data Structures & Algorithms with JavaScript, O'Reilly, Sebastopol, str. ix
[122] McMillan M. (2014) Data Structures & Algorithms with JavaScript, O'Reilly, Sebastopol, str. 13
[123] McMillan M. (2014) Data Structures & Algorithms with JavaScript, O'Reilly, Sebastopol, str. 89

*osoba = {ime: "Ivan, prezime: "Horvat", dob: 25, pušač: false}*

Objekti i nizovi se ovisno o potrebama mogu kombinirati, gnijezditi jedni u druge bez nekog ograničenja, pa je moguće imati objekt koji uz neki ključ kao vrijednost ima niz, ili niz koji sadrži objekte, neki od kojih ponovno sadrže nizove itd.

## 4.5. Web okviri (*framework*)

Moderne web aplikacije zbog funkcionalnosti svojih korisničkih sučelja imaju složene programske strukture. Ručno pisanje programskog koda, zbog kompleksnosti aplikacije, može rezultirati nejednakom kvalitetom i sadržajem individualnih dijelova aplikacije. Održavanje već razvijenih aplikacija je iz istih razloga otežano. Stoga se web aplikacije često razvijaju upotrebom različitih okvira (*frameworkova*), koji omogućavaju jednostavniju i više uniformnu strukturu skripti te lakše održavanje[124].

Neki od popularnih okvira za razvoj korisničkih sučelja u web aplikacijama su Vue.js, Angular.js i React.js. Za ovaj rad je odabran posljedni, React.js. React je JavaScript *framework* za izgradnju korisničkih sučelja dizajniran s postepenom implementacijom u vidu, dakle može ga se upotrijebiti u onoj količini koliko je koristan i potreban, od dodavanja malo interaktivnosti bazičnoj HTML stranici do građenja kompleksne aplikacije[125].

U projektu je uporabljen zbog lakše organizacije koda vizualnih elemenata (lakše praćenje kako količina JavaScript koda i HTML anotacije raste) i ponajviše zbog lakoće dinamičkog stvaranja vizualnih elemenata od strane korisnika, što je srž aplikacije, a ujedno Reactova specijalnost.

## 4.6. Lokalna memorija web preglednika (Local Storage)

Svi moderni web preglednici pružaju API poznat kao *Web Storage API* koji pregledniku (a samim time i programeru) daje mehanizma sigurnog spremanje ključ-vrijednost parova. *Storage* objekti su jednostavni ključ-vrijednost parovi slični objektima u JavaScriptu i drugim programskim/skriptnim jezicima, uz razliku što perzistiraju među učitavanjima i osvježavanjima stranice. I ključevi i vrijednosti su obvezno tipa *string* (običan tekst).

[124] Kaluža M., Troskot K., Vukelić B.: Comparison of Front-End Frameworks for Web Applications Development Zbornik Veleučilišta u Rijeci, Vol. 6 (2018), str. 261. Hrčak [online]. Dostupno na: https://hrcak.srce.hr/199922 [8. 7. 2021]
[125] React - Getting Started. Dostupno na: https://reactjs.org/docs/getting-started.html [15.7.2021.]

Web Storage API ima 2 glavna mehanizma:

- *sessionStorage* – održava zasebno područje za spremanje za svaku domenu (stranicu) koje je dostupno za upotrebu tijekom čitave sesije na toj stranici (dokle god je web preglednik otvoren, što uključuje i osvježavanja, tj. ponovna učitavanja stranice).
- *localStorage* – radi identično, osim što podaci perzistiraju i nakon zatvaranja i ponovnog otvaranja web preglednika.

Ugrađene JavaScript funkcije kojima se pristupa Web Storage API-ju omogućavaju dohvat, brisanje, dodavanje i ažuriranje podataka, baš kao na bazi podataka[126].

Budući da je *Storage* objekt temeljen na jednostavnim ključ-vrijednost parovima, isprva bi se moglo učiniti kako bi mogao nastati problem sa spremanjem dubljih podatkovnih struktura (ugniježđenih, "*nested*" objekata). No, u praksi se taj problem vrlo lagano zaobilazi – JavaScript objekti se u kodu pretvaraju u *string* (običan tekst) te spremaju kao vrijednost za određeni ključ, koji će ovdje predstavljati ekvivalent kolekciji iz dokumentne baze podataka poput MongoDB-a. Kada ih je potrebno dohvaćati radi upita nad njihovim dijelovima, ponovno se u kodu pretvaraju u objekte. Za obje akcije JavaScript ima ugrađene funkcije – *JSON.parse* i *JSON.stringify*, pa je proces vrlo jednostavan.

U poglavlju 5.3. detaljnije je objašnjeno modeliranje, dohvat i korištenje podataka iz localStorage objekta kroz rad u JALA aplikaciji.

# 5. Istraživanje upotrebe NoSQL baza kroz razvoj aplikacije s listom zadataka

## 5.1. Predmet i cilj istraživanja

Predmet istraživanja je razvoj prototipne aplikacije korištenjem najraširenijih front-end tehnologija te Web Storage API-ja za podatkovni sloj, efektivno stvarajući NoSQL bazu podataka sa klijentske strane. Cilj istraživanja je na relativno ograničenom setu aplikacijskih funkcionalnosti, a samim time i podatkovnih entiteta spoznati prednosti i izazove vezane uz

[126] MDN Web Docs: Using the Web Storage API. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API [28.7.2021.]

modeliranje podataka koristeći ovakvo rješenje kao podatkovnu potporu aplikacije u razvoju. Osim ispunjavanja glavnog cilja – istražiti "ponašanje" ovog rješenja tijekom samog razvoja aplikacije, bila bi šteta ne i "gledati u budućnost" - na temelju akumuliranih teoretskih znanja iz ovoga rada promišljati o ponašanju podatkovnog sloja tijekom (hipotetskih) daljnjih iteracija aplikacije, koje bi uključivale nove funkcionalnosti i skaliranje u distribuirani sustav.

## 5.2. Opis aplikacije

Aplikacija "JALA" (skraćenica za "Just Another List App") razvijena od strane autora za potrebe ovog diplomskog rada je web aplikacija dvostruke svrhe:

- Istraživanje ponašanja ne-relacijskih baza podataka kao potpore web aplikacijama,
- Pružanje funkcionalnog, modernog, usko specijaliziranog alata za produktivnost svim zainteresiranim stranama (samome autoru rada, profesorima, drugim studentima te široj javnosti).

Osnovna zamisao je izdvajanje funkcije vođenja liste zadataka na razini tjedna, budući da drugi softverski alati za produktivnost tu funkcionalnost posjeduju kao jedan od svojih modula (Trello, Notion itd.), no do njih je često nekoliko klikova/pritisaka na dodirni ekran. S druge strane, ugrađena rješenja mobilnih operativnih sustava za vođenje bilješki nisu građena oko liste zadataka po danima, pa je potrebno npr. datume pisati u naslove bilješki (što pak dovodi do toga da se liste zadataka miješaju s korisnikovim generalnim bilješkama), implementacija "checkboxova" je ili nepostojeća ili zahtjeva korisničku akciju dodavanja. Poanta ove aplikacije leži, dakle, u tome što je u svojoj srži građena kao lista zadataka. U tome slijedi globalne trendove izdvajanja funkcionalnosti u zasebne aplikacije umjesto velikih, jedinstvenih i modularnih. Na primjer, kod društvenih mreža, gdje je nekada dominirao Facebook, tržište je izvrsno prihvatilo izdvajanje njegove funkcionalnosti objave statusa u Twitter, fotografija u Instagram te kratkih videozapisa u TikTok.

Glavna prednost aplikacije leži u brzini pristupanja listi zadataka te dodavanju novih, brisanju, mijenjanju redoslijeda te označavanju istih dovršenima ili nedovršenima. Na taj način sprječava se gubitak vremena te smanjuje mogućnost gubljenja toka misli dok korisnik pristupi sučelju za dodavanje novih zadataka. Zadaci su tematski podijeljeni na 2 glavne vrste entiteta:

- Dani
- Projekti.

*Dani* su zamišljeni za vođenje lista zadataka vezanih uz pojedinačne dane (vođenje dnevnih zadataka poput "kupi mlijeko", "idi u teretanu" ili "plati račun za vodu"), dok su *Projekti* namijenjeni za organizaciju zadataka vezanih uz kompleksnije i dugotrajnije poslovne i osobne projekte (fakultet, posao, uređenje stana). Među ovim entitetima nema direktne povezanosti u ovoj verziji aplikacije. Drugi podatkovni entiteti koje aplikacija uključuje su Korisnički podaci i Servisni podaci (podaci koju krajnji korisnik neće direktno vidjeti, već služe programskom kodu za aktivnosti poput dodjeljivanja identifikatora zapisima).

Aplikacija je pisana s idejom jednake pristupačnosti na računalu i na mobilnom telefonu ili tabletu, što je ostvareno kroz responzivni dizajn, tj. pisanje CSS koda na način da se korisničko sučelje prilagođava ovisno o detektiranoj veličini ekrana. Teško se može smatrati progresivnom web aplikacijom u punom smislu pojma, no pokazuje neke vezane funkcionalnosti. Prema Staničiću i Kraljević (2019)[127], "*progresivna web aplikacija (PWA) je noviji termin koji označava skup određenih funkcionalnosti koje možemo dodati modernim web aplikacijama kako bi ih poboljšali. Moderni web preglednici omogućavaju web aplikacijama velik broj novih mogućnosti koje prije nisu bile moguće, a pomoću kojih web aplikacije mogu sve više sličiti nativnim mobilnim aplikacijama, kako u izgledu tako i u ponašanju. Epitet progresivno se odnosi na progresivno dodavanje funkcionalnosti kako bi web aplikacija izgledala kao nativna mobilna aplikacija, te kako bi korisnik imao korisničko iskustvo kao da koristi nativnu mobilnu aplikaciju. Naravno ako korisnik koristi stariji web preglednik, web aplikacija će i dalje funkcionirati sa osnovnim skupom funkcionalnosti kao i do sada. Tri glavne karakteristike progresivnih web aplikacija su pouzdanost (momentalno učitavanje, rad čak u izvanmrežnom načinu rada), brzina (brzi odgovori na korisničku interakciju s glatkim animacijama i prijelazima) i privlačnost (osjećaj nativne aplikacije s unaprijeđenim korisničkim iskustvom)"*. Aplikacija JALA uistinu simulira iskustvo nativne mobilne aplikacije, te poput njih koristi i za vlastitu upotrebu rekombinira ugrađene elemente mobilnog softvera (spremanje u lokalnu memoriju mobilnog web preglednika, nativni kalendar, nativni alat za izbor boja, nativne notifikacije o uspješnim i neuspješnim radnjama). Također, nakon inicijalnog učitavanja vizualnih elemenata (dobivanja HTML, CSS i JavaScript koda sa poslužitelja), izvanmrežni način rada u slučaju (namjernog ili slučajnog) prekida internetske veze je moguć, budući da se podaci spremaju u lokalnu memoriju klijenta.

[127] Staničić K., Kraljević S. (2019) Analiza značajki i performansi progresivnih web aplikacija. Hrčak [online]. Dostupno na: https://hrcak.srce.hr/236123 [29. 6. 2021]

Ova tema aplikacije (lista zadataka) izabrana je za istraživanje budući da se radi o poslovnom slučaju koji stvara optimalan broj entiteta (naspram prototipiranja nekog kompleksnijeg poslovnog sustava) i zbog svoje univerzalne funkcionalnosti vraća fokus istraživanja na promatrano tehničko područje (štedeći vrijeme koje bi u slučaju prototipiranja poslovnog sustava bilo utrošeno na proučavanje poslovnog slučaja, npr. osiguranja, obračuna plaća itd.).

S arhitekturalnog aspekta, aplikacija JALA u trenutnoj iteraciji predstavlja maksimalno pojednostavljenu verziju klijentsko-poslužiteljske arhitekture. Aplikacija se u cijelosti dostavlja klijentu s besplatnog *hosting* (smještaj na internetskom poslužitelju) servisa GitHub Pages (što je zapravo proširena funkcionalnost GitHuba, repozitorija koda i sustava za kontrolu verzija koji je korišten u njenom razvoju), no na njemu se zapravo ne odvija ništa osim posluživanja koda i vizualnih resursa u cijelosti klijentu (korisničkom računalu), koji ovdje služi i kao prezentacijski sloj, i sloj za implementaciju logike aplikacije i čak baza podataka. To je omogućeno snagom moderne iteracije JavaScripta (ES6+) te Web Storage API-jem koji omogućuje razvoj jednostavne NoSQL baze na klijentskoj strani.

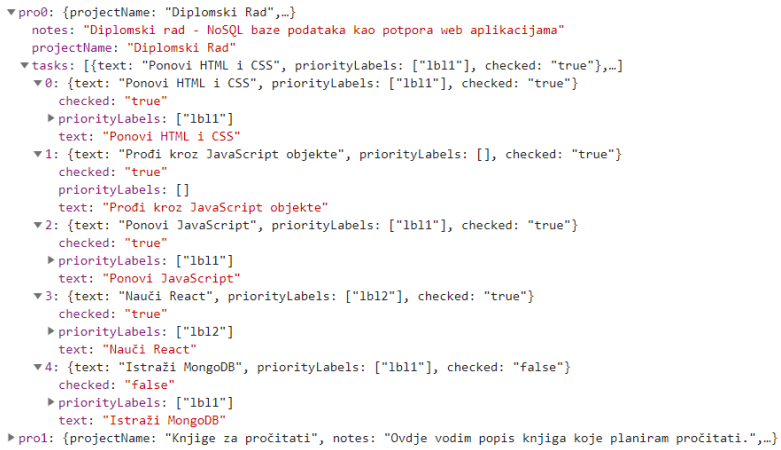
## 5.3. Modeliranje, dohvat i korištenje podataka

Podaci aplikacije podijeljeni su u četiri ključ-vrijednost para s ugniježđenim objektima kao vrijednostima. Ti parovi ekvivalent su kolekcija u dokumentnim bazama podataka. Objekti unutar njih ekvivalent su pak dokumenata unutar kolekcija. Ključevi/kolekcije su:

- *projects*
- *days*
- *userData*
- *service*

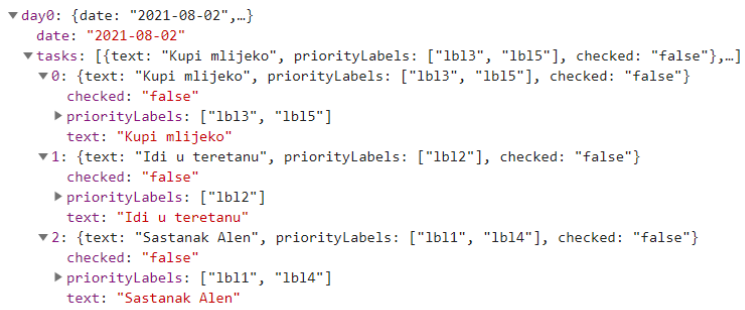
U *projects* kolekciji nalaze se objekti/dokumenti koji ponovno sadrže ključ-vrijednost parove. Neki od ključeva kod pojedinačnih dokumenata imaju vrijednost vrste *string* (*notes* – bilješke, *projectName* – ime projekta, dok je na ključ *tasks* – zadaci vezana vrijednost u oblik JavaScript *array*-a – niza, koji pak sadrži objekte – pojedinačne zadatke. Budući da je dodavanje labela prioritea/vrste zadatka – *priorityLabels* neobavezno, ali ih može biti i više, njihova vrijednost je potencijalno prazni niz, i to niz referenci ključeva na labele u *userData* kolekciji. Ovdje se vidi mogućnost implementacije relacijske logike u NoSQL sustavima gdje je to potrebno. Labele su dio korisničkih postavki, tj. korisnik ih dodaje sam prema svojim željama, i nema ih potrebe "gnijezditi" (*nesting*) u svaki od zadataka, pa je iskorištena mogućnost referenciranja,

koja je provedena na razini aplikativnog koda. Još jedan tip vrijednosti koji se nalazi u svakome od zadataka je *boolean*, tj. true ili false, na ključu *checked* – označeno (ovisno je li korisnik zadatak označio dovršenim ili nije).



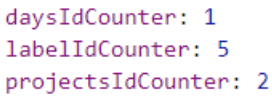

*Slika 15: Podatkovna struktura projects kolekcije. Izvor: Vlastita izrada autora.*

*days* kolekcija funkcionira gotovo identično *projects* kolekciji, uz razliku što se umjesto ključa *projectName* na svakome od dana nalazi ključ *date* – datum, te nema ključ-vrijednost parova *notes* (bilo bi besmisleno budući da nema potrebe za detaljnijim objašnjavanjem pojedinačnog dana kao što ima kod projekata). *date* vrijednost je isključivo tipa *date* – datum, te se validacija trenutno provodi na razini aplikativnog koda (preko HTML input formi).



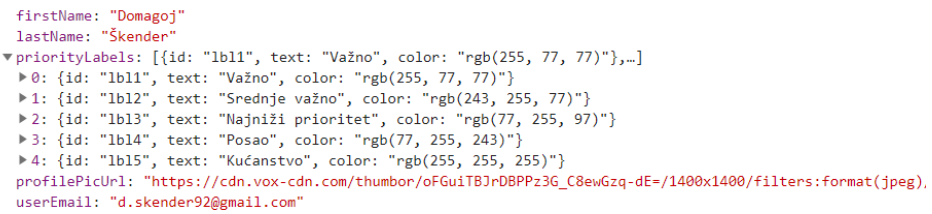

*Slika 16: Podatkovna struktura days kolekcije. Izvor: Vlastita izrada autora.*

Strukturno najjednostavnija kolekcija je *service*. Tamo se nalaze brojači za ID-jeve dana, projekata i labela, spremanjem u bazu osigurava se njihovo perzistiranje i onemogućuju potencijalno dupliranje ID-jeva, koje bi narušilo funkcioniranje čitave aplikacije. Radi se o običnim parovima ključeva i vrijednosti – brojeva.




*Slika 17: Podatkovna struktura service kolekcije. Izvor: Vlastita izrada autora.*

Posljednja kolekcija zove se *userData* i sadrži korisničke podatke. Ime, prezime i e-mail adresa trivijalni su ključ-vrijednost parovi, gdje je vrijednost obični *string*. Zanimljivije vrijednosti nalaze se uz *priorityLabels* – labelama prioriteta/vrste zadatka te uz *profilePicUrl* – URL korisnikove odabrane profilne slike. *priorityLabels* se sastoji od niza objekata s dodanim labelama, s dodijeljenim ID-jevima, upisanim tekstom te bojom u RGB obliku, koju će front-end kod iskoristiti za obojati u korisnikove odabrane boje u modalima za dodavanje/ažuriranje projekata i dana, budući da se ID-jevi referenciraju u dokumentima u kolekcijama *days* i *projects*. *profilePicUrl* sprema vrijednost URL-a korisnikove odabrane profilne slike s Interneta te će prema njoj HTML kod istu prikazati u za to određenom mjestu u gornjem desnom kutu ekrana. Budući da je LocalStorage u svojoj osnovi mala NoSQL baza, može podnijeti i prethodno u teorijskom dijelu spomenute BLOB objekte. Naime, koristeći neki od mnogobrojnih dostupnih online pretvarača, moguće je bilo koju sliku u PNG ili JPEG formatima transformirati u čisti kod. Ako bi se taj kod unio kroz formu kao vrijednost u ovo polje, moguće ju je direktno spremiti u bazu. Jedino ograničenje je što slika ne smije biti prevelika, budući da većina web preglednika alocira samo 5MB za LocalStorage po domeni (stranici).




*Slika 18: Podatkovna struktura userData kolekcije. Izvor: Vlastita izrada autora.*

Podaci se trenutno dohvaćaju API pozivima u JavaScript kodu na klijentu, što bi se u budućoj iteraciji aplikacije s MongoDB bazom pretvorilo u vrlo slične MQL naredbe, pisane u backend kodu. Korišteni API pozivi su:

- localStorage.getItem(*ključ*) – dohvat željene kolekcije za daljnju obradu
- localStorage.setItem (*ključ, vrijednost*) – spremanje u slučaju nepostojeće vrijednosti ključa, ažuriranje u slučaju već postojeće vrijednosti ključa

- localStorage.clear –brisanje svih podataka iz za aplikaciju alocirane memorije preglednika – koristi se samo kod funkcije vezane na gumb za brisanje svih podataka

Kod dohvata iz LocalStoragea, API pozivi se zamataju u funkcije JSON.parse i JSON.stringify, što je trenutno potrebno za konverziju vrijednosti iz ključ-vrijednost parova u lokalnoj memoriji, budući da te vrijednosti mogu biti samo tipa *string*, tj. obični tekstovni niz. Stoga se kod spremanja pretvaraju u *string*, a kod dohvata u kod radi obrade nazad u JavaScript (JSON) objekte.

U trenutnoj iteraciji aplikacije, ID-evi dokumenata se generiraju pomoću ručno pisanih funkcija u samom aplikacijskom kodu. U MongoDB-u bi taj dio bio automatiziran i programeru apstrahiran. Kolekcije trenutno ne trebaju ID-jeve budući da se aplikacija sprema lokalno s klijentske strane i postoji samo jedna instanca svake kolekcije. U praksi slijede sljedeću konvenciju imenovanja:

- projects – pro{dodijeljeni broj projekta} – npr. pro0, pro1, … pro12, …, proXX
- days – day{dodijeljeni broj dana} – npr. day0, day1, … day12, …, dayXX
- priorityLabels.id – lbl{dodijeljeni broj labele prioriteta/vrste zadatka} – lbl0, lbl1, … lbl12, … lbl XX

## 5.4. Zaključna razmatranja i potencijalne buduće iteracije aplikacije

Aplikacija JALA je školska prototipna aplikacija razvijena za potrebe ovog rada i primarno služi kao demonstrator alternativnog modela podataka u odnosu na relacijski model. Ipak, osnovne funkcionalnosti su u potpunosti zaokružene te je pažnja posvećena korisničkom iskustvu i općenitom vizualnom dizajnu. Kao takva u potpunosti je upotrebljiva za svoju deklariranu namjeru – organizaciju osobnih zadataka po pojedinačnim danima te prema dužim projektima. Tehnički i u ovom prototipnom obliku prikazuje klijentsko – poslužiteljsku arhitekturu, budući da se nalazi na (besplatnom) poslužitelju – GitHub Pages.

Prvi sljedeći korak u daljnjem razvoju bio bi, jasno, implementacija NoSQL baze podataka na stvarnom, dediciranom poslužitelju, te bi to sigurno bila dokumentna baza, najvjerojatnije MongoDB, primarno iz razloga istovjetnosti podatkovnih struktura – JSON objekti bez ikakvih izmjena translatiraju u MongoDB dokumente, a trenutni nad-objekti koji sadrže pojedinačne objekte bi postali MongoDB kolekcije i dokumenti. S upotrebnog gledišta, prvo unaprjeđenje logike aplikacije na redu za implementaciju bilo bi stvaranje opcionalne povezanosti

pojedinačnih zadataka kroz dane i projekte. Konkretno, isti zadatak bi tada mogao biti prisutan i u nekom od dana, ali i u projektu, a njegovo označavanje dovršenim bi se automatski propagiralo na oba mjesta u korisničkom sučelju (budući da se radi o istom entitetu u bazi). Aplikacija se trenutno čitava odvija na klijentskoj strani kroz JavaScript, te bi uz implementaciju baze bilo potrebno dio funkcionalnosti prebaciti na poslužiteljsku stranu kroz implementaciju nekog *backend* koda koji će se spajati na API DBMS-a i omogućavati dvostranu komunikaciju između poslužitelja i klijenta (slanje upita na poslužitelja prema akcijama na korisničkom sučelju; vraćanje podataka iz baze klijentu). Za te potrebe koristio bi se neki od *backend* jezika kao što su PHP, JavaScript (kroz Node.js) ili Python.

Velik dio ručno pisanog koda će u tom slučaju biti zamijenjen automatiziranim upitima dostupnima u MongoDB DBMS-u. Čitava funkcionalnost dodjele ID-jeva podatkovnim entitetima biti će za programera u potpunosti automatizirana. Baza podataka na poslužitelju omogućuje skaliranje u slučaju komercijalnog uspjeha projekta te postojanja krajnjih korisnika van fakultetskog i prijateljskog okruženja. Takva baza također olakšava stvaranje korisničkih profila, te laganu implementaciju pristupa svojemu profilu i podacima s više uređaja (npr. korisnik na radnom mjestu na računalu upiše neke zadatke ili čitave projekte, a kasnije ih izmjenjuje/dodaje/označuje završenim s mobilnog uređaja). Zbog zadržavanja obilježja progresivne web aplikacije i mogućnosti vanmrežnog načina rada, upotreba LocalStorage-a bila bi zadržana i u svim budućim iteracijama aplikacije, nasuprot upitima na bazu za svaku izmjenu. Ovisno o daljnjem razvoju funkcionalnosti, potrebno bi bilo donijeti odluku o točnom modalitetu osvježavanja baze (koliko često će se osvježavati, te hoće li to biti određeno nekim događajima u aplikaciji ili će se raditi u unaprijed definiranim vremenskim intervalima, jasno samo kada je internetska veza prisutna). Još jedna manje važna prednost DBMS jest i dodatan sloj validacije tipova podataka iz formi.

Glavna prednost izbora upravo NoSQL baze i modela baziranog na JSON objektima leži u fleksibilnosti i nedostatku rigidne podatkovne sheme. Ovakve aplikacije su podložne čestim promjenama, novim i nepredvidivim idejama, a nedostatak/neobveznost međusobnih relacija unutar podatkovnih struktura te mogućnost dodavanja novih atributa na određenim vrstama entiteta bez potrebe da se stare instance istih struktura ažuriraju tim novim atributima, pruža veliku kreativnu slobodu za daljnji razvoj. Također, velika prednost je i nativna podrška za *nizove* (*arrays*) u ovakvim bazama, budući da je zbog svoje prirode aplikacija snažno bazirana na atributima s više vrijednosti (broj zadataka u danu, broj zadataka u projektu, korisnikove dodane labele za prioritete i vrstu zadataka), a oni u ovom načinu rada umjesto dodavanja u

zasebne povezane tablice, čine jednostavan niz u kolekciji za korisničke postavke. Nema potrebe za pažnjom oko podatkovnog integriteta, kaskadnog brisanja (npr. kod već uporabljenih labela), nego se te vrijednosti lagano brišu, a u slučaju nepronalaženja, tj. nevalidnih referenci, jednostavno se neće pojaviti na korisničkom sučelju. Konkretno, ako se neka labela prioriteta koja je upotrjebljena na više zadataka kroz administrativno sučelje izbriše iz baze, jednostavno se više neće pojavljivati na onim zadacima gdje je prethodno dodana, bez ikakvog rizika za integritet podataka.

U hipotetskom slučaju komercijalne održivosti aplikacije, sustava s velikim brojem čvorova, trenutna organizacija podataka je idealna. Naime, dokumenti s korisničkim podacima (osobni korisnički podaci, dani, projekti) mogli bi se rasporediti po čvorovima po regionalnim ili nekim drugim ključevima, te lagano dohvaćati bilo korištenjem hash funkcija bilo particioniranjem prema rasponu ključeva, a sama poslovna logika aplikacije nalaže da bi neki kompleksniji upiti kroz više dokumenata (*joinovi*) bili rijetkost. Podatkovni sloj aplikacije dakle idealan je za horizontalno particioniranje (*sharding*) i minimalnu količinu replikacije (potrebno je samo dovoljno kopija da jamči dostupnost, tj. minimizira rizik nedostupnosti u slučaju kvarova, a podaci su vezani uz pojedinačne korisnike – istim setovima podataka ne pristupa veliki broj ljudi).

Jedina planirana buduća funkcionalnost u kojoj bi NoSQL rješenje moglo predstavljati manu u odnosu na relacijsku bazu jest implementacija nekog oblika pregleda projekata ili dana prema labelama prioriteta, budući da se radi o uvjetnoj pretrazi, što bi predstavljalo WHERE naredbu u SQL-u i JOIN operacije, a koja je ipak sporija kod dokumentne baze, budući da se mora pretraživati svaki pojedinačni dokument (unutar neke kolekcije). Ipak, niti to nije prevelik problem, budući da se takvi upiti mogu predvidjeti i postaviti indeksi na bazu, a i opseg podataka koji se stvara na razini pojedinačnog korisnika je dovoljno malen da bi se to usporavanje mjerilo u milisekundama, što korisnik ne bi niti osjetio u praksi.

# 6. Zaključak

Prije svega potrebno je naglasiti da iako se radi o tehnologiji koja potječe još iz 70-ih godina prošlog stoljeća, tradicionalne relacijske baze još dugo (a potencijalno nikad) neće biti u potpunosti zamijenjene. Uvijek će imati svoje mjesto u poslovnim informacijskim sustavima,

relativno su jednostavno za korištenje, njihovi DBMS-ovi se brinu za integritet podataka u područjima upotrebe gdje je to esencijalno te osiguravaju konzistentnost u CAP smislu te ACID svojstva. Svatko s ponešto poslovnog iskustva, bez velike IT ekspertize, zna da su neki podaci jednostavno prirodno pogodni za uklapanje upravo u međusobno povezane tablice. Osim toga, ne stvara svaki poslovni sustav količinu podataka koji se ne bi mogli adekvatno spremati i procesuirati na jednom poslužitelju. Uvijek je tu i mogućnost vertikalnog skaliranja (unaprjeđenje tog poslužitelja s jačim procesorom, više radne memorije, još tvrdih diskova i SSD-ova) u kombinaciji sa skladištem podataka kad transakcijski podaci više nisu bitni već samo njihovi agregati. Arhitektura takvih podataka i SQL jezik čine relacijske DBMS-ove i dalje najbržim i najboljim rješenjem za dohvat podataka, barem u slučaju kompleksnih izvještaja koji uključuju spajanje preko velikog broja tablica. Naravno, tu je i živo tržište velikih korporacija koje ulažu u daljnja unaprjeđenja svojih relacijskih sustava kako bi pratili korak s vremenom, pa se razvijaju nove mogućnosti, često i hibridne, uzimajući inspiraciju upravo od novije nastalih NoSQL sustava.

Prava potreba za NoSQL rješenjima nastaje s globalno dostupnim digitalnim uslugama, društvenim mrežama, stranicama za *streaming* video sadržaja i slično, distribuiranim sustava koje stvaraju ogromne količine podataka, tzv. *Big data*, što stvara neizbježno povlači horizontalno skaliranjee njihovih baza. Pritom, većina takvih globalnih servisa mora biti dostupno 24 sata na dan, iz jednostavnih razloga. Neki servis za računovodstvo ili revizijska kuća mogu si priuštiti nedostupnost svoje jedine transakcijske baze podataka radi održavanja između 1 i 3 ujutro, dok za YouTube, Amazon ili Netflix to baš i nije mogućnost. Njihovi ogromni, 24 sata dostupni, globalno fragmentirani (po tisućama poslužitelja) podatkovni sustavi trpe doduše bez većih problema eventualnu konzistentnost, tj. činjenicu da ažuriranja možda neće biti instantno vidljiva krajnjem korisniku – Netflix neće imati nikakvih problema zato što je novi film učitan na poslužitelj u Sacramentu korisniku koji šalje HTTP zahtjev poslužitelju u Stockholmu dostupan 10 minuta kasnije, nakon što se proces replikacije obavi. Radi se o različitim potrebama i prioritetima prema CAP teoremu.

NoSQL baze su, kao što je u radu opisano, podosta raznolike, od više-manje univerzalnih kojima gotovo da bi se i moglo zamijeniti SQL baze poput dokumentnog MongoDB-a, pa do vrlo niši prilagođenih sustava koji obično rade u paru s nekom tradicionalnijom bazom, kao što je slučaj kod Redisa. Iako čak niti to ne vrijedi za sve njih, zajednička točka većine NoSQL rješenja je baziranost na nekom obliku ključ-vrijednost parova, što automatski dovodi do samoopisujućih podataka i slabijih ili nepostojećih jezika upita te prebacivanja zadataka

rekonfiguracije podataka na kod aplikativnog softvera. Paradigma ponajviše zajednička NoSQL rješenjima je generalni nedostatak unaprijed zadanih rigoroznih podatkovnih shema, čime bi se mogao i zaključiti ovaj diplomski rad: SQL za strukturu i konzistentnost, NoSQL za fleksibilnost i neograničeno skaliranje.

# 7. Literatura


1) Alvaro, F. (2018) SQL: Easy SQL Programming & Database Management For Beginners, Your Step-By-Step Guide To Learning The SQL Database, Amazon, e-izdanje
2) Amazon Web Services: Redis. Dostupno na: https://aws.amazon.com/redis/ [1.8.2021.]
3) Anderson B., Nicholson B. (2021) SQL vs. NoSQL Databases: What's the Difference? IBM [online]. Dostupno na: https://www.ibm.com/cloud/blog/sql-vs-nosql [28. 7. 2021.]
4) Apache Cassandra dokumentacija. Dostupno na: https://cassandra.apache.org/doc/latest/ [1.8.2021.]
5) AxiomQ: Comparing ORM vs SQL: What to Know. Dostupno na: https://axiomq.com/blog/comparing-orm-vs-sql-what-to-know/ [30.7.2021.]
6) Babić A.., Jakšić D., Poščić P.(2019) Querying Data in NoSQL Databases. Hrčak [online]. Dostupno na: https://hrcak.srce.hr/219991 [28. 7. 2021]
7) Batinić P., Dobrinić D. (2019) Implementacija velikih vrsta podataka u CRM. Hrčak [online]. Dostupno na: https://hrcak.srce.hr/234546 [28. 6. 2021]
8) Concepta: What is the Difference between Front-End and Back-End Development? Dostupno na: https://www.conceptatech.com/blog/difference-front-end-back-end-development [31.7.2021.]
9) Coronel C., Morris S. (2019) Database Systems: Design, Implementation and Management, Cengage, Boston
10) DAMA International (2018) Data Management Body of Knowledge, Technics Publications, Basking Ridge

11) DAMA Mission, Vision, Purpose and Goals. Dostupno na: https://www.dama.org/cpages/mission-vision-purpose-and-goals [12.11.2020.]
12) Database Zone: NoSQL Database Types. Dostupno na: https://dzone.com/articles/nosql-database-types-1 [29.7.2021.]
13) Elmasri R., Navathe S.B. (2016) Fundamentals of Database Systems, Pearson, Harlow
14) Go from data to insight to action with Power BI Desktop. Dostupno na: https://powerbi.microsoft.com/en-us/desktop/ [20.2.2021.]
15) Havaš L., Lesar M. (2012) Primjena SQL-a u programima otvorenog koda. Hrčak [online]. Dostupno na: https://hrcak.srce.hr/94801 [28. 7. 2021.]
16) HTML & CSS. Dostupno na: https://www.w3.org/standards/webdesign/htmlcss [15.7.2021.]
17) Humby, C., Hunt, T. (2004) Scoring Points - How Tesco Is Winning Customer Loyalty, Kogan Page Ltd, London
18) Insight Data Science: The Total Newbie's Guide to Cassandra. Dostupno na: https://blog.insightdatascience.com/the-total-newbies-guide-to-cassandra-e63bce0316a4 [1.8.2021.]
19) JavaScript Web APIs. Dostupno na: https://www.w3.org/standards/webdesign/script.html [28.7.2021.]
20) Kaluža M., Troskot K., Vukelić B.: Comparison of Front-End Frameworks for Web Applications Development, Zbornik Veleučilišta u Rijeci, Vol. 6 (2018). Hrčak [online]. Dostupno na: https://hrcak.srce.hr/199922 [8. 7. 2021]
21) Library of Congress: HyperText Markup Language (HTML) 5. Dostupno na: https://www.loc.gov/preservation/digital/formats/fdd/fdd000481.shtml [28.7.2021.]
22) McMillan M. (2014) Data Structures & Algorithms with JavaScript, O'Reilly, Sebastopol
23) MDN Web Docs: About JavaScript. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript [28.7.2021.]
24) MDN Web Docs: Using the Web Storage API. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API [28.7.2021.]
25) MongoDB dokumentacija. Dostupno na: https://www.mongodb.com/ [1.8.2021.]
26) Neo4j dokumentacija. Dostupno na: https://neo4j.com/developer/ [1.8.2021.]
27) PhoenixNAP: What is a Graph Database. Dostupno na: https://phoenixnap.com/kb/graph-database [1.8.2021.]

28) Prasad, Y.L. (2016) Big Data Analytics Made Easy, Notion Press, Chennai
29) React Documentation - Getting Started. Dostupno na: https://reactjs.org/docs/getting-started.html [15.7.2021.]
30) Sommerville I. (2016) Software Engineering, Pearson, Harlow
31) Staničić K., Kraljević S. (2019) Analiza značajki i performansi progresivnih web aplikacija. Hrčak [online]. Dostupno na: https://hrcak.srce.hr/236123 [29. 6. 2021]
32) Stojanović A. (2016) Osvrt na NoSQL baze podataka – četiri osnovne tehnologije. Hrčak [online]. Dostupno na: https://hrcak.srce.hr/192140 [28. 7. 2021]
33) Talend: Beginner's Guide to Batch Processing. Dostupno na: https://www.talend.com/resources/batch-processing/ [1.8.2021.]
34) Ubuntu Blog: What is Cassandra and Why are Big Tech Companies using it? Dostupno na: https://ubuntu.com/blog/apache-cassandra-top-benefits [1.8.2021.]
35) Varga M. (2014) Upravljanje podacima, Element, Zagreb
36) W3 Schools - CSS Tutorial. Dostupno na: https://www.w3schools.com/css/ [16.7.2021.]
37) W3 Schools – HTML Styles - CSS. Dostupno na: https://www.w3schools.com/html/html_css.asp [16.7.2021.]
38) What is the definition of OLAP? Dostupno na: https://olap.com/olap-definition/ [21.2.2021.]

## 8. Popis slika