NOSQL baze podataka

Stipić, Robert

Undergraduate thesis / Završni rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:630673

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-08-07**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

Robert Stipić

NoSQL baze podataka

Završni rad

Pula, rujan, 2019.

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Robert Stipić

NoSQL baze podataka

Završni rad

JMBAG: 0303054488, redoviti student

Studijski smjer: Sveučilišni preddiplomski studij Informatika

Predmet: Baze podataka I

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: doc.dr.sc. Tihomir Orehovački

Pula, rujan, 2019.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Robert Stipić, kandidat za prvostupnika informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, 18.09.2019. godine

IZJAVA

o korištenju autorskog djela

Ja, Robert Stipić dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom NoSQL baze podataka koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 18.09.2019.

Potpis

_____

NoSQL baze podataka

Robert Stipić

Sažetak: U završnom radu pojašnjene su karakteristike SQL i NoSQL pristupa radu s podatcima. Objašnjeno je zašto je tradicionalni SQL pristup napušten u rješavanju problema koje su donijele sve veće količine podataka, što je dovelo do razvoja NoSQL rješenja i njihov konstanti rast u radu s podatcima. Način rada s NoSQL bazama podataka objašnjen je na primjerima dokument baze MongoDB.

Ključne riječi: Baza podataka, SQL, NoSQL, MongoDB

Abstract: In this bachelor's thesis has been explained characteristics of SQL and NoSQL approach working with data. It describes why traditional SQL approach has been abandoned with solving problems which big data brought, which led to development of NoSQL solutions and their constant growth when working with data. Working with NoSQL databases has been explained on examples in document database MongoDB.

Keywords: Database, SQL, NoSQL, MongoDB

Sadržaj

1. Uvod 1
2. Baze podataka 3
   2.1 Modeli podataka 7
   2.2 Objektni model podataka 7
   2.3 Konceptualna shema relacijskog modela podataka 8
   2.4 Relacijski model 12
   2.5 SQL 13
3. NoSQL pokret 16
   3.1 Povijest NoSQL-a 16
   3.2 Što je NoSQL ? 17
      3.2.1 Kako su velike kompanije utjecale na razvoj NoSQL-a 21
      3.2.2 Google Bigtable paper 21
      3.2.3 Amazon Dynamo paper 22
   3.3 Vrste NoSQL baza podataka 23
      3.3.1 Ključ-vrijednost baze podataka 24
      3.3.2 Graf baze podataka 27
      3.3.3 Stupčane baze podataka 30
      3.3.4 Dokument baze podataka 31
4 MongoDB 33
   4.1 Instalacija mongo poslužitelja na lokalno računalo 34
   4.2 JSON 36
   4.3 Mongo ljuska 36
      4.3.1 CRUD operacije 38
      4.3.2 Kolekcije 51
      4.3.3 Indeksiranje podataka 55
      4.3.4 Rad s geoprostornim podatcima 59

5 Zaključak 64
6 Literatura 66

Popis Slika

Slika 1. Arhitektura Amazon platforme 23
Slika 2. Primjer ključ-vrijednost baze podataka 25
Slika 3. Primjer modela graf baze podataka 29
Slika 4. Primjer obitelji stupaca 31
Slika 5. Preuzimanje *mongo Community Server-a* 35
Slika 6. Dodavanje puta do mongo poslužitelja 35
Slika 7. Pokretanje MongoDB servisa 36
Slika 8. Primjena osnovnih operacija u mongo ljusci 37
Slika 9. Primjena funkcije *insertMany()* 38
Slika 10. Dodavanje ugniježđenog dokumenta 39
Slika 11. Dodavanje reda vrijednosti 40
Slika 12. Operacija *find()* 40
Slika 13. Operacija *findOne()* s filterom 41
Slika 14. Primjena operatora usporedbe *$lt* 42
Slika 15. Primjena operatora usporedbe *$in* 42
Slika 16. Primjena logičkog operatora *$or* 43
Slika 17. Primjena logičkog operatora *$and* i operacije *count()* 44
Slika 18. Primjena logičkog operatora *$not* 44
Slika 19. Primjena elementarnog operatora *$exists* 45
Slika 20. Primjena elementarnog operatora *$type* 45
Slika 21. Primjena *$regex* operatora 45
Slika 22. Primjena *$expr* operatora 46
Slika 23. Projektiranje podataka 46
Slika 24. Primjena operacije *updateOne()* i operatora *$set* 47
Slika 25. Primjena operacije *updateMany()* i operatora *$inc* 48
Slika 26. Primjena operatora *$rename*. 49
Slika 27. Primjena operatora *$unset*. 49
Slika 28. Izmjenjivanje reda vrijednosti. 49
Slika 29. Primjena operatora *$push*. 50
Slika 30. Primjena operatora *$pop*. 50
Slika 31. Primjena operacije *deleteOne()*. 50
Slika 32. Primjena veze jedan na jedan korištenjem reference. 53
Slika 33. Primjena veze jedan naprema više korištenjem reference. 54
Slika 34. Primjena veze više naprema više korištenjem referenci. 54
Slika 35. Primjer studenta sa svim vrstama referenci. 55
Slika 36. Izrada jedinstvenog indeksa. 56
Slika 37. Izrada kombiniranog indeksa. 57
Slika 38. Izrada tekstualnog indeksa. 58
Slika 39. Pretraživanje tekstualnog indeksa. 58
Slika 40. Dodavanje geoprostornih podataka u kolekciju. 59

Slika 41. Izrada geoprostornog indeksa 60
Slika 42. Primjena operatora *$near* i *$geometry*. 60
Slika 43. Spremanje geografskih točaka u konstante. 61
Slika 44. Primjena operatora *$geoWithin* 61
Slika 45. Dodavanje poligona u kolekciju. 62
Slika 46. Primjena operatora *$geoIntersects* 62
Slika 47. Primjena operatora *$centerSphere*. 63

Popis Tablica

Tablica 1. Vrste funkcionalnosti za vezu između tipova entiteta E1 i E2 11
Tablica 2. Aplikacije za rad s NoSQL bazama podataka 24

1. Uvod

Davno su ljudi počeli razmjenjivati i koristiti informacije, odnosno podatke čija se količina povećava konstantno kako svijet napreduje tako da je sve više podataka koje ljudi koriste što se odrazilo na načine spremanje i upotrebe istih. Podatci se mogu spremati na razne načine putem umjetnosti, zapisivanja na papir, zapisivanja u digitalnom obliku i slično. Neki od ovih načina su efikasniji jer nude bolju organizaciju i efikasniju upotrebu od drugih, ipak jedan način rada s podatcima se istaknuo više od drugih spremanje i organiziranje podataka u digitalnom obliku.

Tehnologija stalno napreduje što rezultira sve većim količinama podataka potrebnim za njen napredak, tako se stalno mijenjaju i poboljšavaju rješenja koja se koriste za efikasniji rad s podatcima. Prvo su se pojavile tradicionalne SQL baze podataka koje svoja rješenja za rad s podatcima razvijaju još od 1970-ih godina do danas, tako da imaju jako dobru teoretsku osnovu i pružaju puno mogućnosti u radu s podatcima. SQL baze podataka prate određene norme i pravila koje ih iz toga razloga čine SQL bazama podataka. Ovaj pristup dugo se smatrao optimalan za rad s podatcima, te rješenja koja su odstupala od normi i pravila koje koristi SQL pristup teško su se razvijala i nisu nailazila na veliki interes korisnika. Ipak kako se količina podataka koja se koristi u digitalnom obliku povećavala i razvoj novih tehnologija je zahtijevao fleksibilnija rješenja u radu s podatcima od tradicionalnih koje su nailazile na probleme poput sporog rada s velikim količinama podataka i spremanja velikih količina podataka na jednog poslužitelja bez mogućnosti raspodjele istih na više poslužitelja razvio se NoSQL pokret.

Pojam NoSQL prvi put upotrijebljen je 1998. godine, čak 30 godina poslije prvih SQL rješenja za rad s podatcima tako da se može smatrati relativno novim pokretom u radu s podatcima. NoSQL nema veliku teoretsku osnovnu niti prati norme i pravila kao SQL. Ovo ipak ne znači da se NoSQL mora puno razlikovati od SQL-a, NoSQL se može shvatiti kao inačica SQL-a koja nedostatke rada s podatcima na tradicionalni način rješava odbacujući neka pravila SQL-a kako bi se pronašli fleksibilniji i efikasniji načini rada s podatcima gdje SQL ne može ponuditi optimalno rješenje.

U ovom radu bit će ukratko objašnjene prednosti i nedostaci spremanja podataka u digitalni oblik, što su to tradicionalne baze podataka odnosno SQL baze podataka, koje pravila i norme prate takve baze, ali isto tako zašto se javila potreba za novim rješenjima u radu s podatcima, te kako su ta rješenja prenesena na razne NoSQL baze podataka koje su podijeljene u četiri glavne vrste: ključ-vrijednost baze podataka, dokument baze podataka, stupčane baze podataka i graf baze podataka. Rad s dokument bazom podataka MongoDB bit će detaljno objašnjen kako bi se stvorila bolja slika kako ove baze funkcioniraju i koja rješenja mogu ponuditi u radu s podatcima.

2. Baze podataka

Baze podataka predstavljaju višu razinu rada s podacima u odnosu na klasične programske jezike. Riječ je o tehnologiji koja je nastala s namjerom da se uklone slabosti tradicionalne "automatske obrade podataka" iz 60-tih i 70-tih godina 20. stoljeća. Ta tehnologija osigurala je veću produktivnost, kvalitetu i pouzdanost u razvoju aplikacija koje se svode na pohranjivanje i pretraživanje podataka u računalu (Manger, 2012).

Baza podataka je skup međusobno povezanih podataka, pohranjenih u vanjskoj memoriji računala. Podaci su istovremeno dostupni raznim korisnicima i aplikacijskim programima. Ubacivanje, promjena, brisanje i čitanje podataka obavlja se posredstvom zajedničkog softvera. Korisnici i aplikacije pritom ne moraju poznavati detalje fizičkog prikaza podataka, već se referenciraju na logičku strukturu baze (Manger, 2012).

Oblikovanje fizičkog prikaza baze podataka izvršava se pomoću sustava za upravljanje bazom podataka ili DBMS[1], fizički prikaz baze podataka zasniva se na logičkoj strukturi baze podataka. Logička struktura baze izrađuje se na osnovnu skupa pravila koja se još nazivaju model podataka. Ovisno o odabranom modelu za izradu baze podataka ista logička struktura baze podataka se može razlikovati. U današnje vrijeme DMBS najčešće podržavaju jedan od sljedećih modela: relacijski model, mrežni model, hijerarhijski model i objektni model, te su podatci u bazi podataka logički organizirani u skladu s odabranim modelom.

Da bi baza podataka predstavljala višu razinu rada s podatcima mora imati logičku strukturu podataka koju je moguće fizički prikazati pomoću DBMS-a, osim mogućnosti za pohranjivanje, pretraživanje, brisanje i dodavanje podataka koji se jednom riječju nazivaju CRUD[2]. Osim logičke strukture, fizičkog prikaza i CRUD-a baza podataka da bi se smatrala višom razinom rada s podatcima mora nastojati ostvariti sljedeće ciljeve (Manger, 2012):

[1] DBMS(engl. Data Base Management System) je poslužitelj baze podataka

[2] CRUD operacije za dodavanje, dohvaćanje, izmjenu i brisanje podataka, dolazi od engleskih riječi Create, Read, Insert i Update

Fizička nezavisnost podataka. Razdvaja se logička definicija baze od njene stvarne fizičke grade. Znači, ako se fizička grada promijeni (na primjer, podaci se prepišu u druge datoteke na drugim diskovima), to neće zahtijevati promjene u postojećim aplikacijama.

Logička nezavisnost podataka. Razdvaja se globalna logička definicija cijele baze podataka od lokalne logičke definicije za jednu aplikaciju. Znači, ako se logička definicija promijeni (na primjer uvede se novi zapis ili veza), to neće zahtijevati promjene u postojećim aplikacijama. Lokalna logička definicija obično se svodi na izdvajanje samo nekih elemenata iz globalne definicije, uz neke jednostavne transformacije tih elemenata.

Fleksibilnost pristupa podacima. U starijim mrežnim i hijerarhijskim bazama, staze pristupanja podacima bile su unaprijed definirane, dakle korisnik je mogao pretraživati podatke jedino onim redoslijedom koji je bio predviđen u vrijeme projektiranja i implementiranja baze. Danas se zahtijeva da korisnik može slobodno prebirati po podacima, te po svom nahođenju uspostavljati veze medu podacima. Ovom zahtjevu zaista zadovoljavaju jedino relacijske baze.

Istovremeni pristup do podataka. Baza mora omogućiti da veći broj korisnika istovremeno koristi iste podatke. Pritom ti korisnici ne smiju ometati jedan drugoga, te svaki od njih treba imati dojam da sam radi s bazom.

Čuvanje integriteta. Nastoji se automatski sačuvati korektnost i konzistencija podataka, i to u situaciji kad postoje greške u aplikacijama, te konfliktne istovremene aktivnosti korisnika.

Mogućnost oporavka nakon kvara. Mora postojati pouzdana zaštita baze u slučaju kvara hardvera ili grešaka u radu sistemskog softvera.

Zaštita od neovlaštenog korištenja. Mora postojati mogućnost da se korisnicima ograniče prava korištenja baze, dakle da se svakom korisniku reguliraju ovlaštenja što on smije, a što ne smije raditi s podatcima.

Zadovoljavajuća brzina pristupa. Operacije s podacima moraju se odvijati dovoljno brzo, u skladu s potrebama određene aplikacije. Na brzinu pristupa može se utjecati odabirom pogodnih fizičkih struktura podataka, te izborom pogodnih algoritama za pretraživanje.

Mogućnost podešavanja i kontrole. Velika baza zahtijeva stalnu brigu: praćenje performansi, mijenjanje parametara u fizičkoj gradi, rutinsko pohranjivanje rezervnih kopija podataka, reguliranje ovlaštenja korisnika. Također, svrha baze se vremenom mijenja, pa povremeno treba podesiti i logičku strukturu. Ovakvi poslovi moraju se obavljati centralizirano. Odgovorna osoba zove se administrator baze podataka. Administratoru trebaju stajati na raspolaganju razni alati i pomagala.

Navedene ciljeve je potrebno nastojati ostvariti, ali i ostvariti u velikoj mjeri da bi baza podataka funkcionirala na normalan način te ispunila sve zahtjeve korisnika i smanjila moguće greške i zastoje na minimum.

Izrada baze podataka naziva se projekt. Projekt izrade baze podataka možemo podijeliti u pet faza: analiza potreba, modeliranje podataka, implementacija, testiranje i održavanje. Neke od ovih faza završavaju sa završnom verzijom projekta, dok se neke poput održavanja baze podataka koriste tijekom cijelog životnog vijeka baze podataka sa svrhom unaprjeđenja iste. Faze izrade baze podataka detaljnije će biti objašnjene na primjeru uvođenja baze podataka u poduzeće ili ustanovu.

Analizi potreba. Proučavaju se tokovi informacija u poduzeću. Uočavaju se podaci koje treba pohranjivati i veze medu njima. U velikom poduzećima, gdje postoje razne grupe korisnika, pojavit će se razni “pogledi” na podatke. Te poglede treba uskladiti tako da se eliminira redundancija i nekonzistentnost. Na primjer, treba u raznim pogledima prepoznati sinonime i homonime, te uskladiti terminologiju.

Analiza potreba također treba obuhvatiti analizu transakcija (operacija) koje će se obavljati nad bazom podataka, budući da to može isto imati utjecaja na sadržaj i konačni oblik baze. Važno je procijeniti frekvenciju i opseg pojedinih transakcija, te zahtjeve na performanse. Rezultat analize je dokument (pisan neformalno u prirodnom jeziku) koji se zove specifikacija potreba.

Modeliranje podataka. Različiti pogledi na podatke, otkriveni u fazi analize, sintetiziraju se u jednu cjelinu - globalnu shemu. Precizno se utvrđuju tipovi podataka. Shema se dalje dotjeruje (“normalizira”) tako da zadovolji neke zahtjeve kvalitete. Također, shema se prilagođava ograničenjima koje postavlja zadani model podataka, te se dodatno modificira da bi bolje mogla udovoljiti zahtjevima na performanse. Na kraju se iz sheme izvode pogledi (pod-sheme) za pojedine aplikacije (grupe korisnika).

Implementacija. Na osnovu sheme i pod-shema, te uz pomoć dostupnog DBMS-a, fizički se realizira baza podataka na računalu. U DBMS-u obično postoje parametri kojima se može utjecati na fizičku organizaciju baze. Parametri se podešavaju tako da se osigura efikasan rad najvažnijih transakcija. Razvija se skup programa koji realiziraju pojedine transakcije te pokrivaju potrebe raznih aplikacija. Baza se inicijalno puni podacima.

Testiranje. Korisnici pokusno rade s bazom i provjeravaju da li ona zadovoljava svim zahtjevima. Nastoje se otkriti greške koje su se mogle potkrasti u svakoj od faza razvoja: dakle u analizi potreba, modeliranju podataka, implementaciji. Greške u ranijim fazama imaju teže posljedice. Na primjer, greška u analizi potreba uzrokuje da transakcije možda korektno rade, no ne ono ˇsto korisnicima treba već nešto drugo. Dobro bi bilo kad bi takve propuste otkrili prije implementacije. Zato se u novije vrijeme, prije prave implementacije, razvijaju i približni prototipovi baze podataka, te se oni pokazuju korisnicima. Jeftinu izradu prototipova omogućuju jezici 4. generacije i objektno-orijentirani jezici.

Održavanje. Odvija se u vrijeme kad je baza već ušla u redovnu upotrebu. Sastoji se od sljedećeg: popravak grešaka koje nisu bile otkrivene u fazi testiranja; uvođenje promjena zbog novih zahtjeva korisnika; podešavanje parametara u DBMS u svrhu poboljšavanja performansi. Održavanje zahtijeva da se stalno prati rad s bazom, i to tako da to praćenje ne ometa korisnike. Administratoru baze podataka trebaju stajati na raspolaganju odgovarajući alati.

2.1 Modeli podataka

DMBS najčešće podržava neki od četiri navedena modela: relacijski model, objektni model, relacijski model i mrežni model, pomoću ovih modela izrađuje se logički model baze podataka koji se kasnije uz pomoć DMBS-a normalizira. Modeli podataka predstavljaju osnovu za faze modeliranja podataka i implementacije prilikom izrade baze podataka.

Hijerarhijski model razvijen je prvi od navedenih modela još u 60-im godinama 20. stoljeća, razvila ga je tvrtka IBM. Ovaj model je korišten za prvi IBM-ov DBMS naziva Information Management System koji koristi programski jezik DL 1.

Hijerarhijski model. Specijalni slučaj mrežnog. Baza je predočena jednim stablom (hijerarhijom) ili skupom stabala. Svako stablo sastoji se od čvorova i veza „nadređeni-podređeni“ između čvorova. Čvorovi su tipovi zapisa, a odnos „nadređeni-podređeni" izražava hijerarhijske veze među tipovima zapisa (Manger, 2012).

Mrežni model. Baza je predočena mrežom koja se sastoji od čvorova i usmjerenih lukova. Čvorovi predstavljaju tipove zapisa(slogova podataka), a lukovi definiraju veze među tipovima zapisa (Manger, 2012).

Mrežni model može se smatrati naprednijom verzijom hijerarhijskog. Razvijena je na idejama Charlesa Bachman-a. Nastao je kako bi riješio neke od problema koji su nastajali korištenjem hijerarhijskog modela ponajviše nedostatak fleksibilnosti. Glavna razlika između ova dva modela je odnos „roditelj-dijete", hijerarhijski model dopušta da jedan roditelj ima više djece, ali dijete može imati samo jednog roditelja. U mrežnom modelu roditelji mogu imati više djece, ali i djeca mogu imati više roditelja što ga čini složenijim od hijerarhijskog modela.

2.2 Objektni model podataka

Objektni model. Inspiriran je objektno-orijentiranim programskim jezicima. Baza je skup trajno pohranjenih objekata koji se sastoje od svojih internih podataka i “metoda” (operacija) za rukovanje s tim podacima. Svaki objekt pripada nekoj klasi. Između klasa

se uspostavljaju veze nasljeđivanja, agregacije, odnosno međusobnog korištenja operacija (Manger, 2012).

Osim što se pomoću objektnog modela mogu jasno definirati objekti i veze među njima. Objektni model podržava četiri bitna svojstva enkapsulaciju, skrivanje podataka, nasljeđivanje i polimorfizam.

Objedivanje podataka i operacija naziva se enkapsulacija (engl.encapsulation) – objekt se ne sastoji isključivo od podataka već sadrži i metode neophodne za operacije nad tim podatcima. Pritom su podaci privatni za svaki objekt te nisu dostupni ostalim dijelovima programa. Na taj način podaci zaštićeni od neovlaštene promjene izvan objekta koja bi mogla narušiti njegovu cjelovitost (Šribar i Motik, 2014).

Mehanizam zaštite podataka naziva se skrivanje podataka(engl. data hiding). Svaki objekt svojoj okolini pruža isključivo podatke i operacije koji su neophodni da bi okolina objekt mogla koristiti. Ti javno dostupni podaci zajedno s operacijama koje ih prihvaćaju ili vračaju čine sučelje (engl. interface) objekta (Šribar i Motik, 2014).

Mogućnost da objekti naslijede neke metode ili podatke od objekata više razine s kojima su povezani naziva se nasljeđivanje. Ovo svojstvo omogućuje povezanim objektima da koriste neke dijelove drugog objekta ako im je potrebno ali da ti dijelovi nisu definirani u oba objekta.

Svojstvo da različiti objekti istu operaciju obavljaju na različiti način zove se polimorfizam (engl. polimorphism). Ovo svojstvo se još naziva svojstvom promjenjivosti. Polimorfizam omogućuje da koristimo podatke i metode nadređenog objekta baš kao i nadređeni objekt bez grešaka zbog mogućih različitih tipova podataka ili metoda (Šribar i Motik, 2014).

2.3 Konceptualna shema relacijskog modela podataka

Relacijski model bio je teoretski zasnovan još krajem 60-tih godina 20. stoljeća, u radovima E.F. Codd-a. Model se dugo pojavljivao samo u akademskim raspravama i knjigama. Prve realizacije na računalu bile su suviše spore i neefikasne. Zahvaljujući

intenzivnom istraživanju, te napretku samih računala, efikasnost relacijskih baza postepeno se poboljšavala. Sredinom 80-tih godina 20. stoljeća relacijski model je postao prevladavajući. I danas većina DBMS koristi taj model (Manger, 2012).

Prilikom izrade relacijskog modela potrebno je prvo napraviti konceptualnu shemu koja se može prikazati pomoću E-R(engl. entity-relationship) modela odnosno model entiteta i veza. Ovaj model sastoji se od entiteta, atributa i veza. Notacija nam pomaže utvrditi pravila kojih se moramo držati prilikom izrade E-R modela, a rezultati će biti dijagram. Tri dijagrama koja se mogu koristiti za izradu E-R modela su: izvorni Chenov dijagram, reducirani Chenov dijagram i UML-ov class dijagram. Ovo nisu jedini dijagrami pomoću kojih se može izraditi E-R model, ali su najpoznatiji dijagrami koji se koriste za ovu svrhu.

Izvorni Chenov dijagram. Kao grafički elementi pojavljuju se pravokutnici, rombovi, „mjehurići“ i spojnice među njima. Pritom pravokutnici označavaju entitete, rombovi veze, a mjehurići atribute. U sam dijagram su kao jedini tekstovni elementi ubačena imena entiteta, veza i atributa te oznake takozvanih kardinalnosti veza. Prednost takvog načina prikazivanja je da je sva informacija prikazana na dijagramu. Nedostatak je u tome što dijagram može postati nepregledan i prenatrpan mjehurićima ako on sadrži mnogo atributa (Manger, 2012).

Reducirani Chenov dijagram. Riječ je o pojednostavnjenoj vrsti izvornog Chenova dijagrama, gdje su zbog bolje preglednosti nacrtani samo pravokutnici (entiteti), rombovi (veze) i spojnice među njima, a izbačeni su mjehurići (atributi). I dalje su na dijagramu prisutna imena entiteta i veza te oznake kardinalnosti veza. Nedostatak informacije o atributima na dijagramu nadomješta se tekstom uz dijagram (Manger, 2012).

UML-ov class-dijagram. UML je standardizirani i danas vrlo popularan grafički jezik koji se rabi u objektno-orijentiranim metodama za razvoj softvera. Class-dijagram je jedan od standardnih UML-dijagrama i on originalno sluţi za prikaz klasa objekata i veza između tih klasa. Taj dijagram možemo uporabiti za prikaz konceptualne sheme baze tako da entitet interpretiramo kao posebnu vrstu klase koja ima atribute, ali nema operacije. Entitet se tada crta kao pravokutnik s upisanim imenom entiteta na vrhu i

upisanim imenima svih atributa u sredini. Veza (ili asocijacija po UML-ovoj terminologiji) crta se kao spojnica između pravokutnika s upisanim imenom na sredini i upisanim oznakama kardinalnosti (ili multipliciteta po UML-ovoj terminologiji) na krajevima. Dijagram sadrži svu potrebnu informaciju pa nema potrebe za tekstovnim nadopunama (Manger, 2012).

E-R dijagram sastoji se od entiteta, atributa i veza. U najširem smislu entiteti predstavljaju stvari, bića, pojave ili događaje, veze predstavljaju odnose među entitetima, a atributi opisuju entitete ili veze.

Kandidat za ključ je atribut ili skup atributa čije vrijednosti jednoznačno određuju primjerak entiteta zadanog tipa. Dakle, ne mogu postojati dva različita primjerka entiteta istog tipa s istim vrijednostima kandidata za ključ (Manger, 2012).

Entitet može imati jedan ili više primarnih i stranih ključeva. Primarni ključ je jedan od atributa koji je jedinstven za svaki entitet istog tipa, tako da prilikom odabira primarnog ključa moramo odabrati atribut za koji znamo da neće moći biti ponovljen za neki entitet istog tipa. Stranih ključeva unutar jednog entiteta može biti više, a strani ključevi su primarni ključevi drugih entiteta s kojima je određeni entitet povezan nekom vrstom veze. Ključevi služe za identifikaciju pojedinog entiteta i povezivanje različitih entiteta.

Uspostavljanjem veze između dvaju ili više entiteta izražavamo činjenicu da se ti entiteti nalaze u nekom odnosu. Veza se uvijek definira na razini tipova entiteta, no realizira se povezivanjem pojedinih primjeraka entiteta dotičnih tipova. Za sada ćemo se ograničiti na takozvane binarne veze, koje su najjednostavnije i najčešće u primjenama. Binarna veza uspostavlja se između točno dva tipa entiteta. Stanje binarne veze opisuje se kao skup uređenih parova primjeraka entiteta koji su trenutačno povezani (Manger, 2012).

Veze među entitetima mogu biti različite, više je načina na koji entiteti mogu biti povezani. Strani ključevi su zapravo ranije definiranje veze entiteta prikazane E-R modelom baze. Veze mogu biti entitet i sadržavati svoje atribute. Veze među entitetima

izražavaju se pomoću svojstva funkcionalnosti i obveznosti članstva i kardinalnosti. Ova svojstva omogućuju da se E-R model na pravi način prebaci u bazu podataka.

OZNAKA	NAZIV	OPIS
1:1	Jedan-naprema-jedan	Jedan primjerak od E1 može biti povezan najviše s jednim primjerkom od E2. Također, jedan primjerak od E2 može biti povezan najviše s jednim primjerkom od E1.
1:M	Jedan-naprema-mnogo	Jedan primjerak od E1 može biti povezan s više primjeraka od E2. Istovremeno, jedan primjerak od E2 može biti povezan najviše s jednim primjerkom od E1.
M:1	Mnogo-naprema-jedan	Jedan primjerak od E1 može biti povezan najviše s jednim primjerkom od E2. Istovremeno, jedan primjerak od E2 može biti povezan s više primjeraka od E1
M:M	Mnogo-naprema-mnogo	Jedan primjerak od E1 može biti povezan s više primjeraka od E2. Također, jedan primjerak od E2 može biti povezan s više primjeraka od E1.

Tablica 1. Vrste funkcionalnosti za vezu između tipova entiteta E1 i E2 (Manger, 2012).

Promatramo vezu između tipova entiteta E1 i E2. Funkcionalnost te veze je svojstvo koje kaže je li za odabrani primjerak entiteta jednog tipa moguće jednoznačno odrediti

povezani primjerak entiteta drugog tipa. Drugim riječima, funkcionalnost je svojstvo koje kaže može li se veza interpretirati kao preslikavanje (funkcija) iz skupa primjeraka entiteta jednog tipa u skup primjeraka entiteta drugog tipa. S obzirom da se ista veza može promatrati u dva smjera, od E1 do E2 i obratno, postoje četiri vrste funkcionalnosti koje su opisane u Tablici 1 (Manger, 2012).

2.4 Relacijski model

Relacijski model zahtijeva da se baza podataka sastoji od skupa pravokutnih tabela - tzv. relacija. Svaka relacija ima svoje ime po kojem je razlikujemo od ostalih u istoj bazi. Jedan stupac relacije obično sadrži vrijednost jednog atributa (za entitet ili vezu) - zato stupac poistovjećujemo s atributom i obratno. Atribut ima svoje ime po kojem ga razlikujemo od ostalih u istoj relaciji. Vrijednosti jednog atributa su podaci istog tipa. Dakle, definiran je skup dozvoljenih vrijednosti za atribut, koji se zove domena atributa. Vrijednost atributa mora biti jednostruka i jednostavna (ne da se rastaviti na dijelove). Pod nekim uvjetima toleriramo situaciju da vrijednost atributa nedostaje (nije upisana). Jedan redak relacije obično predstavlja jedan primjerak entiteta, ili bilježi vezu između dva ili više primjeraka. Redak nazivamo n-torka. U jednoj relaciji ne smiju postojati dvije jednake n-torke. Broj atributa je stupanj relacije, a broj n-torki je kardinalnost relacije (Manger, 2012).

Relacijska shema manje je razumljiva korisnicima od konceptualne, jer su u njoj i entiteti i veze među entitetima pretvoreni u relacije pa je teško razlikovati jedno od drugog. Ipak, važno svojstvo relacijske sheme je da se ona može više-manje izravno implementirati pomoću današnjih DBMS-a. Zahvaljujući današnjem softveru od relacijske sheme do njezine konačne implementacije vrlo je kratak put (Manger, 2012).

Odabir ključeva predstavlja bitan korak u izradi relacijske sheme osim odabira primarnih ključeva koji mora biti atribut pojedinog entiteta koji jednoznačno određuje entitet te ne mogu postojati dva entiteta istog tipa s istim primarnim ključem, osim primarnog ključa moraju se odrediti i strani ključevi između povezanih entiteta. Strani ključ zapravo predstavlja primarni ključ drugog entiteta.

Kandidat za primarni ključ mora zadovoljavati neka svojstva da bi se mogao smatrati primarnim ključem.

Ključ K relacije R je podskup skupa atributa od R s ovim svojstvima: 1. Vrijednosti atributa iz K jednoznačno određuju n-torku u R. Dakle u R ne mogu postojati dvije n-torke s istim vrijednostima atributa iz K. 2. Ako iz K izbacimo bilo koji atribut, tada se narušava svojstvo 1. Ta su svojstva „vremenski neovisna“, u smislu da vrijede u svakom trenutku bez obzira na povremene unose, promjene i brisanja n-torki (Manger, 2012).

Građu relacije kratko opisujemo takozvanom shemom relacije: to je redak koji se sastoji od imena relacije te od popisa imena atributa odvojenih zarezima i zatvorenih u zagrade. Primarni atributi su podvučeni (Manger, 2012).

2.5 SQL

SQL (engl. Structured Query Language) je najraširenija jezik za rad s relacijskom bazom podataka. Jezik je nastao u 70-im godinama 20. stoljeća u sklopu razvojno-istraživačkog projekta System R unutar kompanije IBM. Voditelj tog projekta i glavni dizajner SQL-a bio je Donald Chamberlin. Jezik se postupno usavršavao, a njegova dotjerana varijanta pojavljuje se u današnjem IBM-ovu relacijskom DBMS-u zvanom DB2.

U širenju SQL-a tijekom 80-ih godina važnau ulogu odigrala je softverska kuća Oracle Corporation: ona je ugradila SQL u svoj DMBS, te ga je time učinila dostupnim i popularnim na svim važnijim računalnim platformama. Drugi tadašnji proizvođači DMBS-a, naprimjer Ingres Corporation, Digital Equipment Corporation, Informix inc, Sybase inc, pod pritiskom tržišta bili su prisiljeni prihvatiti SQL i odustati od mogućih vlastitih jezika. Uporabu SQL-a u razvoju web-aplikacija tijekom 90-ih godina potakla je kompanija MySQL AB svojim besplatnim DBMS-om. Zbog pojave raznih „dijalekata", već 1986. godine bio je donesen prvi ISO/ANSI standard za SQL – njegova zadnja verzija objavljena je 2008. godine (Manger, 2012).

Relacijski model završava logičko oblikovanje baze podataka i omogućava prelazak na fizičko oblikovanje baze podataka. Za fizičko oblikovanje relacijskog modela koristi se

SQL jezik koji optimiziran za rad s relacijskim modelom podataka i jedan je od najpoznatijih jezika za rad s DBMS-om. Sintaksa SQL jezika slična je izvornom engleskom jeziku što znatno olakšava njegovu upotrebu. Na osnovu relacijskog modela pomoću DMBS-a kreiraju se tablice jedna tablica predstavlja jedan entitet ili relaciju, stupac unutar tablice sprema podatke za jedan atribut, dok red tablice predstavlja sve atribute za jednog člana entiteta. Primarni ključ jednoznačno određuje jedan red tablice odnosno u DBMS tablici ne smiju postojati dva retka iste tablice s istim primarnim ključem. Osim primarnog ključa tablica može sadržavati strani ključ, strani ključ je rezultat veza relacijskog modela opisanih u tablicama 1. i 2.. Strani ključ zapravo je dodatni atribut u SQL tablici preuzet iz druge tablice u kojoj ima ulogu primarnog ključa, na ovaj način veza iz relacijskog modela prenesena je u DBMS. Dodavanjem stranog ključa dodaje se dodatni stupac u tablici, pomoću stranog ključa jednom retku tablice se dodaje jedan redak povezane tablice bez potrebe da se dodaju svi atributi povezane tablice, nego se njima pristupa preko stranog ključa koji ima ulogu primarnog ključa u povezanoj tablici, a da pritom ova veza zauzme najmanje moguće prostora. Ukratko strani ključ služi za povezivanje tablica unutar DMBS-a, jedna tablica može sadržavati više stranih ključeva, ali i primarnih ključeva, više primarnih ključeva u jednoj tablici predstavljaju složeni ključ. Potreba za složenim ključem stvara se kada jedan primarni ključ ne može jednoznačno odrediti jedan redak tablice.

SQL je jezik koji se koristi za interakciju s relacijskim DMBS-ovima, SQL omogućuje efikasan način za rad s podatcima iz relacijskog modela. Neke od glavnih uloga SQL-a su:

- Operacije kreiranja, pretraživanja, brisanja i mijenjanja podataka.
- Kreiranje i održavanje baza podataka.
- Projektiranje i kreiranje tablica baze podataka.
- Izvršavanje administrativnih zadataka poput dodjeljivanja prava korisnicima, kreiranje sigurnosnih kopija, povećanje sigurnosti baze podataka i slično.
- Izvršavanje upita nad podatcima.

SQL nije zaseban jezik za rad s podatcima, nego predstavlja kombinaciju četiri jezika za rad s podatcima povezana u jedan. Jezici koji čine SQL su jezik za opis podataka, jezik za manipuliranje podatcima, jezik za postavljanje upita i jezik za kontrolu podataka.

Jezik za opis podataka (engleski Data Description Language – DDL). Služi projektantu baze ili administratoru za zapisivanje sheme ili pogleda. Tim se jezikom definiraju podaci i veze među podacima na logičkoj razini. Naredbe DDL obično podsjećaju na naredbe za definiranje složenih tipova podataka u jezicima poput COBOL-a ili C-a (Manger, 2012).

Jezik za manipuliranje podacima (engleski Data Manipulation Language – DML). Služi programeru za uspostavljanje veze između aplikacijskog programa i baze. Naredbe DML omogućuju „manevriranje“ po bazi i jednostavne operacije kao što su upis, promjena, brisanje ili čitanje zapisa. U nekim softverskim paketima DML je zapravo biblioteka potprograma: „naredba“ u DML-u svodi se na poziv potprograma. U drugim paketima zaista se radi o posebnom jeziku: programer tada piše program u kojem su izmiješane naredbe dvaju jezik, pa takav program treba prevoditi pomoću dvaju prevoditelja (DML-precompiler, obični compiler) (Manger, 2012).

Jezik za postavljanje upita (engleski Query Language – QL). Služi neposrednom korisniku za interaktivno pretraživanje baze. To je jezik koji podsjeća na govorni (engleski) jezik. Naredbe su neproceduralne, dakle takve da samo specificiraju rezultat koji želimo dobiti, a ne i postupak za dobivanje rezultata (Manger, 2012).

Jezik za kontrolu podataka (engleski Data Control Languange – DCL). Ovaj jezik služi za dodjelu prava korisnicima i zaštitu podataka. Svaki korisnik ima dodijeljenu memoriju gdje se spremaju podatcima, tim podatcima može pristupiti samo taj korisnik ili nadređeni korisnik. Pomoću jezika za kontrolu podataka korisnik može omogućiti drugom korisniku da koristi njegove podatke, ali isto tako pomoću ovog jezika može se definirati na koji način drugi korisnik može koristiti te podatke (Manger, 2012).

3. NoSQL pokret

Relacijske baze podataka koje podržavaju SQL jezik u širokoj su upotrebi u današnje vrijeme, razvijaju se desetljećima što je rezultiralo da su imaju jako dobru teoretsku osnovu i pružaju puno mogućnosti u radu s podatcima. Ipak to nije slučaj za NoSQL (engl. Not Only SQL) sustave koji su relativno mladi sustavi ako ih usporedimo s relacijskim bazama podataka, kao što im ime govori NoSQL baze podataka imaju sličnost s SQL bazama podataka, ali ne pridržavaju se svih aspekata SQL-a kao npr. NoSQL ne koristi tablice, niti se podatci spremaju u redove. Potreba za NoSQL-om stvorila se razvojem računala i interneta, ponajviše zbog sve veće upotrebe web pretraživača, društvenih mreža i sl., što je rezultiralo sve većim količinama i vrstama podataka koje se koriste. NoSQL predstavlja pokret koji se javio zbog problema koji nastaju prilikom rada s velikim količinama podataka u relacijskim bazama podataka.

3.1 Povijest NoSQL-a

Termin NoSQL prvi je upotrijebio Carlo Strozzi 1998. godine tijekom posjete San Francisku termin se odnosio na njegovu relacijsku bazu podataka koja nije podržavala standardni SQL.

NoSQL nije tehnologija koju su izmislili nekoliko ljudi u garaži ili matematička teorija o strukturama podataka. Koncepti koje koristi NoSQL razvijali su se polako tijekom nekoliko godina. Nezavisne skupine su tada iskoristile ove ideje i primijenili ih na vlastite probleme u radu s podacima, stvarajući tako različite NoSQL baze podataka koje postoje danas (Fowler, 2015).

Mnoge NoSQL baze podataka otvorenog koda (engl. open source) pojavile su se do 2009. godine. Riak, MongoDB, HBase, Accumulo, Hypertable, Redis, Cassandra i Neo4j su stvorene između 2007 i 2009. Ovo su samo neke NoSQL baze stvorene u to vrijeme, kao što se može vidjeti, puno sustava proizvedeno je u kratkom vremenskom razdoblju. Međutim, čak i sada, inovacija se događaju velikom brzinom (Fowler, 2015).

Iako je pojam NoSQL upotrijebljen prije 20 godina, ipak NoSQL u širu upotrebu ulazi nakon konferencije 2009. godine koju su organizirali Jon Oskarsson i Eric Evans, te su

pozvali mnoge informatičke divove poput Facebook-a, Linkendl-a, Powerset-a, itd. tema je bila sve veći broj ne relacijski baza podataka poput MongoDB, Hbase, Riak, koje su se počele koristiti za rad s velikim količinama podataka koje su se počele pojavljivati ponajviše na internetu. Zaključeno je da bi se sve ne relacijske baze podataka trebali zvati NoSQL, te nakon konferencije termin NoSQL ponovo ulazi u upotrebu.

3.2 Što je NoSQL ?

NoSQL predstavlja pokret koji se bavi razvijanjem ne relacijskih sustava za rad s podatcima čiji je izvorni kod javno objavljen. Kao što je ranije rečeno NoSQL dolazi od pojma ne samo SQL (engl. Not Only SQL) jer je moguće koristiti SQL u nekim ne relacijskim bazama podataka, ali ovo svojstvo nije u praksi često. NoSQL je još samo pokret pa nema jasno definiranu teoretsku osnovu kao SQL za kojeg postoje već definirani standardi, norme i pravila. Pošto NoSQL predstavlja sve baze podataka koje ne koriste relacijski model, razumljivo je da postoji više vrsta NoSQL baza podataka koje se mogu bitno razlikovati po implementaciji, načinu korištenja, svrsi i sl., zajedničko svojstvo im je da ne koriste relacijski model podatka. Najpoznatije vrste ne relacijski baza podataka su ključ-vrijednost, graf, dokument i stupčane baze podataka. NoSQL u užem smislu teško je definirati jer se ne relacijske baze bitno razlikuju od implementacije do primjene, ipak u širem smislu NoSQL se može definirati kao skup različitih tehnologija koje ne prate relacijski model, ali neka svojstva istog koriste.

NoSQL sustavi su jedinstveni jer je pokret izvornog koda pokrenuo njihov razvoj, a ne kompanije koje razvijaju sustave za komercijalnu upotrebu. Stoga se može pronaći mnoštvo NoSQL sustava izvornog koda koji odgovaraju svakoj potrebi. Kada softverski inženjeri nisu u mogućnosti pronaći NoSQL bazu koja odgovara njihovim potrebama, napravili bi ju i podijelili kao aplikaciju izvornog koda (Fowler, 2015).

NoSQL baze podataka kategorizirane su na temelju načina pohranjivanja podataka. Zbog potrebe za dohvaćanjem određenih podataka iz velikog skupa, uglavnom u stvarnom vremenu, NoSQL prati horizontalnu skalabilnost. Oni su optimizirani za

umetanje i pronalaženje operacija na velikom skupu podataka s ugrađenim mogućnostima za repliciranje i klasteriranje[3] (Vaish, 2013).

Horizontalna skalabilnost je jedna od glavnih funkcionalnosti NoSQL-a, za razliku od SQL-a koji prati vertikalnu skalabilnost. Horizontalna skalabilnost omogućava da se operacije čitanja i pisanja podataka provode na više servera ili čvorova u isto vrijeme velikom brzinom, ovo zapravo znači NoSQL baza podataka nije spremljena na jednom serveru kao što je slučaj s SQL-om, nego se sprema na više servera koji rade paralelno što znatno ubrzava rad s podatcima. Isto tako moguće je dodati novi server ako je potrebno bez da to utječe na rad sustava. Ova funkcionalnost pruža idealno rješenje za internetske stranice i mobilne aplikacije gdje je količina korisnika i podataka u stalnom rastu.

Za razliku od SQL-a koji koristi ACID osobine transakcija, NoSQL ih pokušava zaobići, ponajviše zato što ACID osigurava veliku razinu pouzdanosti u radu s podatcima, no prilikom rada s većim količinama podataka javljaju se mnogi problemi uzrokovani ACID osobinama transakcija, ipak postoji mali broj NoSQL sustava koji koriste ACID osobine transakcija. NoSQL radi na principu BASE[4] konzistentnosti. BASE model pokazao se jako korisnim za internetske stranice koje imaju veliki broj korisnika i podataka, a najviše dolazi do izražaja prilikom pretraživanja i dodavanja tih podataka. Tako da društvene mreže i Internet pretraživači u velikoj mjeri koriste NoSQL.

Da bi horizontalno skaliranje bilo moguće baza podataka mora biti distribuirana, distribuirana baza podataka korisnicima DBMS-a izgleda kao jedna baza podataka, ali ona je zapravo spremljena na više računala u dijelovima, računala su povezana pomoću mreže i surađuju zajedno kako bi održali konzistentnost baze podataka.

Horizontalno skaliranje je odlično, ali CAP teorem, kojeg je napisao Eric Brewer 2000. godine, dokazuje da u distribuiranom DBMS-u, mogu se postići samo dva od tri navedena svojstva u isto vrijeme (Hills, 2016):

[3] Klasteriranje – raspodjela podataka na više poslužitelja.

[4] BASE – (engl. Basically Available, Soft state, Eventual consistency) koristi se za opisivanje svojstava NoSQL baza podataka.

- **Konzistentnost**: ovo je posebna vrsta konzistencije ACID modela. Ovo se ne odnosi na provedbi ograničenja konzistencije. Umjesto toga, zahtjeva se da u DBMS-u koji ima više kopija istih podataka, kopije se međusobno moraju podudarati. U horizontalno skaliranoj bazi može se jamčiti da se sve kopije podudaraju ako se žrtvuje dostupnost ili particijalna tolerancija.

- **Dostupnost**: s puno računala koja koriste puno prostora za pohranu preko velikog broja mrežnih veza, u nekim segmentima mogu se javiti pogreške. Takav hardver mora s vremena na vrijeme imati određene greške. Motivacija za korištenje puno računala i kopija istih podataka je da za bilo koji zahtjev za podatke, postoji velika vjerojatnost da neko radno računalo može pristupiti nekoj kopiji podataka i zadovoljiti zahtjev čak ako su neka od računala ili dijelovi podataka nedostupni. 100% dostupnost može se ostvariti samo ako se žrtvuje konzistentnost ili particijalna tolerancija.

- **Particijalna tolerancija**: Particioniranje ili pregrađivanje distribuiranog sustava je slučaj kada mrežni kvar uzrokuje da jedan podskup računala u distribuiranom sustavu ne može komunicirati s ostalim računalima. Rezultat pregrađivanja je da kopije baze podataka održavane od strane različitih računala, mogu se međusobno uskladiti. Nadograđivanje završeno u jednom skupu računala neće biti završeno u drugi skup. DBMS može biti programiran tako da podnosi pregrađivanje, ali samo uz žrtvu konzistentnosti ili dostupnost.

Cap teorem dokazuje da se u distribuiranim sustavima koji koriste horizontalno skaliranje koje koristi većina NoSQL sustava ne može ostvariti ACID osobine transakcija, tako da se u takvim sustavima ostvaruje BASE konzistentnost:

- **Osnovna dostupnost**: Većina podataka bi trebala biti dostupna većinu vremena. Ukoliko podatcima nije moguće pristupiti baza će vratiti odgovor o grešci. Tako da i prilikom uspješnog ili ne uspješnog zahtjeva nad podatcima dobiti će se odgovor. Ukratko ako je neki dio podataka je nedostupan to neće utjecati na rad ostatka baze podataka.

- **Meko stanje**: Sustav ili dio njega povremeno može biti ponekad ne dostupan bez ikakvog zahtjeva na sustav. Promjene na strani klijenta neće se učitati u isto vrijeme kao na serveru, nego je potrebno osvježiti softver na strani klijenta da promjene budu učitane. Ovo označuje da klijent i server nisu savršeno optimizirani ako se mijenjaju neki podatci na strani servera.
- **Naknadna konzistentnost**: Sustav može poprimiti nekonzistentno stanje, ali ovo stanje konzistentnosti označava da će sustav postati konzistentan kroz neko vrijeme. Ipak za vrijeme ne konzistentnosti sustav neće moći primati zahtjeve.

BASE transakcije zvuče pomalo zastrašujuće, ali ponekad ova svojstva su baš što je potrebno. Facebook koriste stotine milijuna korisnika, vjerojatno ne bi mogao izvršavati sve zahtjeve dosljedno s ACID osobinama transakcija. Tako da Facebook koristi meko stanje za objave njegovih korisnika. Postoje situacije u kojima objave se izgube, točnije ne pronađu put do servera. Ipak nedostatak je minimalan: nekolicina korisnika ponekad mora ponoviti svoje ažuriranje ili zahtjev. Dok god se ovo ne događa prečesto Facebook može uspješno omogućiti milijunima korisnika da podjele njihove objave ili misli u stvarnom vremenu. Ovo ne bi bilo moguće s ACID osobinama transakcija (Hills, 2016).

Korištenje BASE-a rezultirat će time da baza ima promjenjiva stanja konzistentnosti i dostupnosti podataka. Ali operacije nad takvom bazom izvršavat će se puno većom brzinom nego u ACID bazama. Horizontalna skalabilnost omogućuje da se na neke servere spreme kopije podataka koje služe samo za čitanje, te se iz tog razloga čitanje podataka izvršava brže nego kada se koristi vertikalna skalabilnost koja sve podatke sprema na jedan server koji služi za čitanje i pisanje podataka.

Kopije podataka koje služe za čitanje u nekim situacijama mogu biti ne ažurirane, iako su se promjene dogodile na bazi, nisu još ažurirane na serverima za čitanje. U tom slučaju neko vrijeme podatci će biti zastarjeli, tako da transakcije za koje je potrebno izvršiti promjene u stvarnom vremenu kao prebacivanje novca putem računa banke mogu dovesti do određenih gubitaka.

3.2.1 Kako su velike kompanije utjecale na razvoj NoSQL-a

Internet zadnjih desetljeća doživljava ogroman rast korisnika, što je rezultiralo povećanjem podataka koji se koriste na njemu. U današnje vrijeme jako je teško pronaći osobu koja nikada nije čula za Internet ili ga koristila, što samo govori koliko je Internet popularan i koliki mu je broj korisnika. Podatci na internetu s vremenom su počeli zauzimati sve više prostora, a tradicionalni relacijski model u tim situacijama počeo je nailaziti na poteškoće u performansama. Tako da su velike kompanije poput Google-a, Amazon-a itd. počele tražiti rješenja kako raditi s velikim količinama podataka na što brži način.

Istraživanja na ovu temu rezultirali su radovima čija se rješenja temelje na odbacivanju nekih svojstava ACID transakcija kako bi se omogućila bolja efikasnost i brzina u radu s velikim količinama podataka. Dva najpoznatija rada su Google-ov „Bigtable paper“ i Amazon-ov „Dynamo paper“ koji opisuju rad s velikim količinama podataka na internetu koristeći metode koje danas predstavljaju temelj NoSQL pokreta.

3.2.2 Google Bigtable paper

Bigtable je distribuirana baza podataka napravljena od strane Google-a 2006. godine za upravljanje velikim količinama strukturiranih podataka. Bigtable je petabajte podataka podijelio na tisuće različitih servera koji surađuju kako bi korisnici efikasno mogli koristiti Google-ove web aplikacije. Neki od Google-ovih projekata koji koriste Bigtable su Google Earth, Google Analitycs i Google Maps, čak preko 60 Google aplikacija koristi Bigtable.

Na prvi pogled slično kao u relacijskom modelu, Bigtable sprema redove s jednim ključem i sprema podatke u redove unutar srodnih stupaca. Dohvaćanje svih srodnih podataka je lagano kao i dohvaćanje određenog podatka korištenjem ključa, nego pomoću složenih upita kao u relacijskim bazama podataka (Fowler, 2015).

Ovaj model isto označava da je distribucija podataka jednostavnija nego s relacijskim bazama podataka. Korištenjem jednostavnih ključeva povezani podatci kao što su sve stranice na jednoj web lokaciji mogu se grupirati zajedno, što povećava

brzinu analize. Bigtable može se smatrati alternativom za više tablica s relacijama. To znači da Bigtable stupci dopuštaju srodni podatcima da budu spremljeni u jedan zapis (Fowler, 2015).

Bigtable je dizajniran da bude distribuiran na velikom broju servera, osnovno svojstvo za sve NoSQL baze podataka napravljene nakon ekspanzije podataka koja je rezultat razvoja interneta. Kombinacija više manjih servera (engl. commodity servers) olakšava korištenje ovakve baze podataka. Na primjer, Dell-ov ili HP-ov server s možda 2 procesora, s 8 do 16 jezgri i 32 do 96GB radne memorije. Ne predstavlja ništa posebno, ali više ovakvih server su jeftiniji nego kupovina jednog velikog servera (Fowler, 2015).

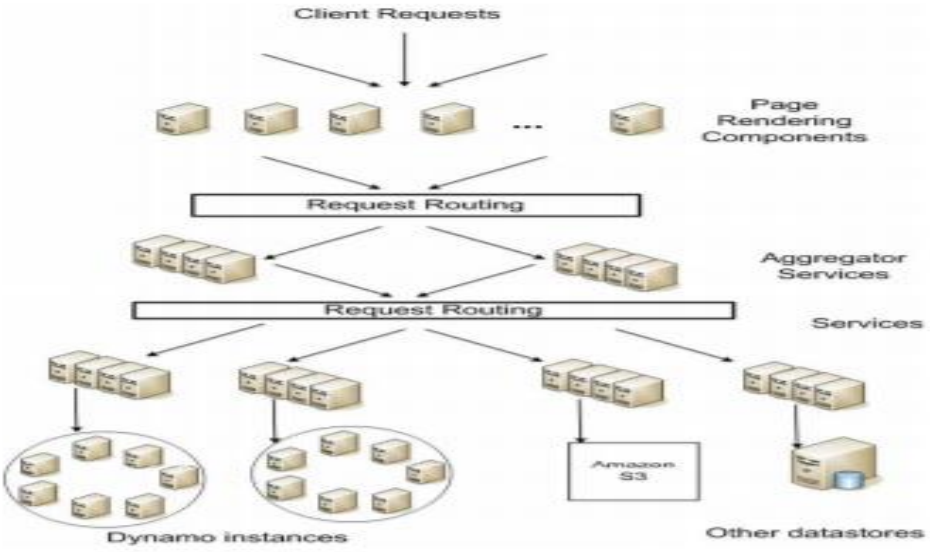
Bigtable se pokazao kao odlično rješenje za skalabilnost, učinkovitost i dostupnost u radu s velikim količinama podataka, te je postigao široku primjenu u velikom broju Google-ovih projekata.

3.2.3 Amazon Dynamo paper

Amazon je 2007. godine objavio svoj vlastiti rad koji opisuje njihovu aplikaciju za spremanje podataka Dynamo. Amazonovim riječima „Dynamo se koristi za upravljanje uslugama s vrlo visokim zahtjevima pouzdanosti i zahtjeva čvrstu kontrolu transakcija između dostupnosti, konzistencije, isplativosti i performansa (Fowler, 2015).“

U radu se opisuje kako se velik dio Amazon-ovih podataka pohranjuje pomoću primarnog ključa, kako se dosljedno heširanje koristi za particioniranje i distribuciju podataka, te kako se dijeljenje objekta koristi za održavanje dosljednosti u podatkovnim centrima (Fowler, 2015).

Dynamo rad osnovno opisuje prvi globalno distribuiranu ključ-vrijednost bazu podataka u Amazonu. Ključevi ovdje su logični identifikator, a vrijednosti mogu biti bilo koja binarna vrijednost od interesa programeru. Jako jednostavan model, zapravo (Fowler, 2015).



Slika 1. Arhitektura Amazon platforme (Hastorun, D. Jampani, M. i dr, 2007).

Navedeni radovi koji pokušavaju pronaći način obrade velikih količina podataka na web lokacijama odbacujući tradicionalni relacijski model oblikovanja rezultirali su konferencijom 2009. godine iz poglavlja 4.1 na kojoj su se svi ovakvi sustavi koji koriste ne relacijski model podataka objedini u NoSQL pokret. Sustavi koje opisuju Amazon-ov Dynamo paper i Google-ov Bigtable paper mogu se smatrati jednima od začetnika NoSQL baza podataka kakve danas poznajemo, arhitektura Amazon platforme prikazana je na slici 1.

3.3 Vrste NoSQL baza podataka

NoSQL baze podataka podijeljene su u četiri kategorije: ključ-vrijednost baze, dokument baze, graf baze i stupčane baze, kategorije se razlikuju po načinu i obliku zapisivanja podataka na bazu. Operacije za rad s podatcima mogu se razlikovati ovisno o strukturi i konceptu NoSQL baze podataka. Ovisno o slučaju upotrebe (engl. Use Case) ove baze mogu ponuditi optimalno rješenje za određeni problem ovisno o potrebama projekta. Projekt može sadržavati više vrsta NoSQL baza podataka jer svaka može biti optimalna za neki slučaj upotrebe. Sve navedene kategorije podržavaju osnovne CRUD operacije, ali ostale operacije za rad s podatcima se mogu razlikovati ovisno o namjeni određene baze.

Relacijski model u NoSQL bazama nije moguće definirati, ipak ove baze ovisno o potrebama projekta mogu djelomično pratiti neku shemu koja definira neke relacije među podatcima ukoliko je ista potrebna za uspješan rad projekta. NoSQL u teoriji ne zahtijeva veze među podatcima samim time relacijski model nije potreban za izradu NoSQL baze, što nije slučaj SQL koji je zasnovan na relacijskom modelu. Ova karakteristika daje dodatnu fleksibilnost prilikom rada s NoSQL bazama.

Ključ-vrijednost baze	**Dokument baze**	**Graf baze**	**Stupčane baze**
Dynamo	MongoDB	Neo4J	Apache Cassandra
Riak	Cosmos DB	ArangoDB	Hypertable
Apache Ignite	Caché	RedisGraph	MonetDB

Tablica 2. Aplikacije za rad s NoSQL bazama podataka

Tablica 2. prikazuje jedne od najpoznatijih aplikacijskih rješenja koje se koriste za rad s navedenim NoSQL bazama. Većina ovih baza je otvorenog koda, tako da je moguće na osnovu toga dodatno prilagođavati NoSQL baze svojim potrebama i potrebama određenog projekta.

3.3.1 Ključ-vrijednost baze podataka

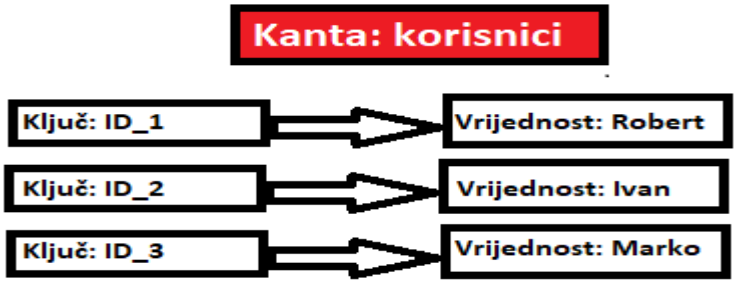
Struktura ključ-vrijednost baza sastoji se od ključa i vrijednosti, ključ predstavlja jedinstvenu vrijednost pomoću koje se razlikuje od drugih ključeva unutar baze, te mu se dodjeljuje vrijednost tako se povezuju ova dva atributa koja sačinjavaju ključ-vrijednost baze podataka. Domena vrijednosti nije definirana tako da ona može biti zapis, slika, zvuk, dokument, Web-adresa i sl. ovo svojstvo omogućava veliku fleksibilnost prilikom spremanja različitih vrsta podataka. Ova vrsta NoSQL baza smatra se najjednostavnijom jer sadrži samo parove ključeva i vrijednosti koji zajedno sačinjavaju jedan zapis u bazi.

Ključ-vrijednost parovi obično se organiziraju u „kante“ (engl. bucket). Kanta se ugrubo može smatrati ekvivalentom tablice u ključ-vrijednost bazi podataka. Kanta je

logičko grupiranje ključeva. Vrijednosti ključeva moraju biti jedinstvene unutar kante, ali mogu biti duplicirani u drugim kantama. Sve operacije nad podatcima bazirane su na kantama i ključevima. Drugim riječima, nije moguće raditi operacije na bilo čemu što se nalazi u vrijednosnoj komponenti ključ-vrijednost para. Sve operacije nad podatcima se izvršavaju s određivanje ključa i kante (Coronel i Morris, 2019).

Kante se koriste kako bi se logički podatci držali fizički zajedno. Logički podatci su kombinacija ključ-vrijednost parova, dok kanta predstavlja zajedničko obilježje ključ-vrijednost parova unutar kante, ali ime kante mora biti jedinstveno odnosno ne mogu postojati dvije kante s istim imenom. Za razliku od SQL-a jedinstveni ključ mora biti jedinstven samo u kanti u kojoj se nalazi, te se može opet koristiti u drugoj kanti tako da unutar baze može biti više istih jedinstvenih ključeva, dok god se nalaze u različitim kantama, što nije slučaj s SQL-om gdje se jedinstveni ključ gleda na razini cijele baze. Ovaj slučaj možemo zamisliti tako ukoliko bi na bazi imali kantu dobavljači kojoj bi ključ predstavljao zapis DOB1, a vrijednost informacije o dobavljaču, ključ DOB1 mogao bi ponovo se koristiti u kanti dobavljači iz inozemstva.

Operacije nad ključ-vrijednost bazama prilično su jednostavne – samo operacije dohvaćanja, spremanja i brisanja se koriste. Dohvaćanje koristi se za dobivanje vrijednosne komponente para. Spremanje se koristi za dodjeljivanje vrijednosti ključu. Ako kombinacija kante i ključa ne postoji, onda se dodaje kao novi ključ-vrijednost par. Ako već postoji kombinacija kante i ključa, onda se postojeća vrijednost komponente zamjenjuje novom vrijednošću. Brisanje se koristi za brisanje ključ-vrijednost para (Coronel i Morris, 2019).



Slika 2. Primjer ključ-vrijednost baze podataka

Slika 2 prikazuje odnos podataka u ključ-vrijednost bazi. Iako postoji mnogo ključ-vrijednost baza Riak ključ-vrijednost baza je jedna od najpoznatijih za rad s ključ-vrijednost bazama. Riak je osnovan 2008. godine, te je bila jedna od prvih kompanija koja se počela baviti razvojem distribuiranih sustava koristeći horizontalno skaliranje za brže izvođenje osnovnih CRUD operacija. Trenutno je jedna od većih kompanija posvećenih rješavanju kompleksnih problema distribuiranih sustava.

Riak je distribuirana baza podataka dizajnirana za pružanje maksimalne dostupnosti podataka podjelom podataka na više poslužitelja. Sve dok vaš Riak klijent može doći do jednog Riak poslužitelja, trebao bi moći pisati podatke. Riak se koristi kao sustav s eventualnom konzistencijom u ovome pristupu podaci koje želite pročitati trebaju ostati dostupni u većini slučajeva kada se dogodi neka greška, iako možda nisu najnovija verzija tih podataka (12).

Ciljevi Riak-a su (12):

- **Dostupnost.** Riak operacije pisanja i čitanja izvršava preko više poslužitelja tako da nudi dostupnost podatka čak i kada hardver ili sama mreža nekog poslužitelja nije dostupna.
- **Operativna jednostavnost.** Lagano dodavanje novih uređaja na Riak klaster (engl. Cluster) bez izlaganja većim operativnim opterećenjima.
- **Skalabilnost.** Riak automatski distribuira podatka preko klastera i postiže skoro linearno povećanje performansi kako dodajete kapacitet.
- **Bez ovladavanja.** Zahtjevi se ne drže kao taoci određenog poslužitelja u klasteru koji može ili ne mora biti dostupan.

Ako se podatci koji se koriste ne bi trebali čuvati na jednom poslužitelju i zahtijevaju distribuiranu arhitekturu baze podataka. Izrada distribuiranih baza podataka ispravno je teška i Riak je napravljen za rješavanje problema dostupnosti podataka sa što manje mogućih nedostupnosti podataka (12).

3.3.2 Graf baze podataka

Graf baza podataka je NoSQL baza podataka zasnovana na teoriji grafova za pohranjivanje podataka o okruženjima koja su bogata relacijskim vezama. Teorija grafova je polje matematičke i informatičke znanosti koja modelira veze između čvorova. Modeliranje i pohranjivanje podataka o vezama je fokus graf baza podataka. Teorija grafova je dobro utvrđeno polje proučavanja koje seže stotinama godina. Kao rezultat toga, stvaranje modela baze podataka temeljenog na teoriji grafova odmah omogućuje bogat izvor algoritama i aplikacija koji su pomogli da baze grafova brzo dobiju sofisticiranost (Coronel i Morris, 2019).

Zanimanje za graf baze podataka javilo se u području društvenih mreža. Društvene mreže uključuju širok spektar aplikacija van tipičnih poput Facebook-a, Twitter-a i Instagram-a koje odmah padnu na pamet. Stranice za upoznavanje, upravljanje znanjem, logistika i usmjeravanje, upravljanje matičnim podatcima, identitetom i pristupom sve su područja koja se uveliko oslanjaju na praćenje složenih veza među objektima (Coronel i Morris, 2019).

Ovakva vrsta baza sastoji se od entiteta koji se još nazivaju čvorovima, čvorovi su međusobno povezani vezama, tako da ovaj oblik NoSQL baza ipak prati neku vrstu relacija, ali ona ne mora biti napravljena po ranije definiranom modelu. Veze u graf bazama označavaju odnose među entitetima, veze se zapisuju pomoću lukova. Razlika između relacijskih modela koji se koriste u SQL-u i graf baza u tome je što veze u graf bazama imaju svoje atribute. Veze prate neki smjer koji je bitan za efikasnije dohvaćanje podataka iz graf baza. Kao što je rečeno ranije graf baze nemaju definirani relacijski model, ovo svojstvo omogućava da se entiteti i veze među njima mogu naknadno dodavati, mijenjati ili brisati.

U današnjem svijetu stalnog tehnološkog napretka i promjena mogućnost izmjena entiteta i njihovih veza daje veliku razinu fleksibilnosti graf bazama podataka. Iz ovog razloga nije potrebno raditi cijelu rekonstrukciju modela baze kao što je slučaj u relacijskom, mrežnom ili hijerarhijskom modelu koji se koriste u SQL bazama podataka,

taj postupak može biti vrlo zahtjevan, ali isto tako može dovesti do velikih gubitaka dok rekonstrukcija ne bude gotova.

Bez obzira na činjenicu da se gotovo sve može modelirati kao grafikon, živimo u pragmatičnom svijetu proračuna, vremenskih linija projekta, korporativnih standarda i komoditiziranih skupova vještina. Graf baza podataka pruža snažnu, ali neobičnu tehniku modeliranja podataka, sama po sebi ne pruža dovoljno opravdanje za zamjenu dobro uspostavljene, dobro razumljive podatkovne platforme. Graf baze podataka nude izuzetno fleksibilan model podataka i način isporuke usklađen s današnjim agilnim postupcima isporuke softvera (Robinson, Webber i Eifrem, 2013).

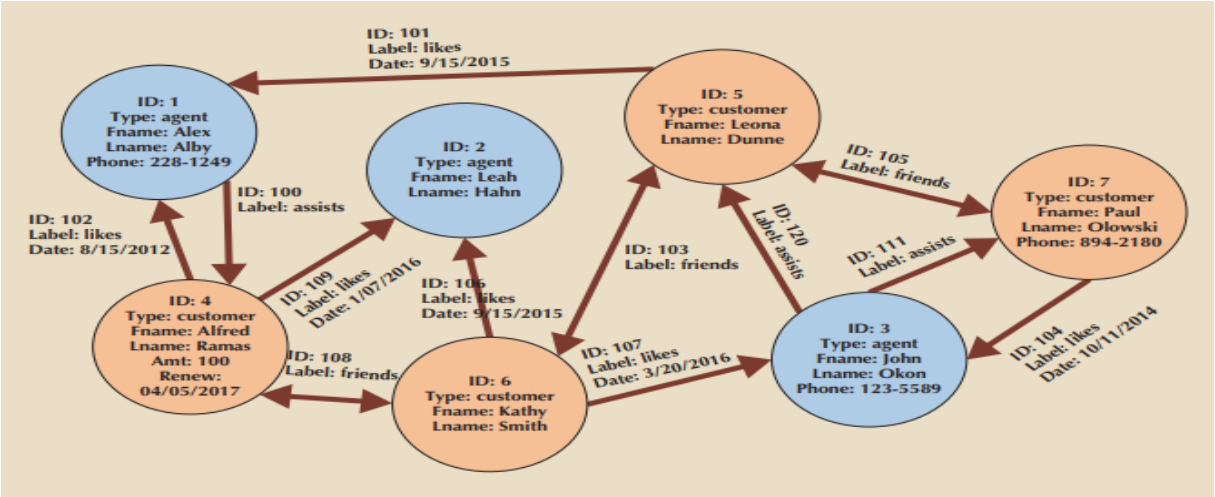
Značajke graf baza podataka (Robinson, Webber i Eifrem, 2013):

- **Performanse.** Stoga je jedan od uvjerljivih razloga za odabir baze podataka grafikona čisto povećanje performansi kada se radi o međusobno povezanim podacima naspram relacijskim bazama podataka i NoSQL skladištima. Za razliku od relacijskih baza podataka, gdje se učinkovitost upita za pridruživanje (engl. joins) smanjuje kako se skup podataka povećava, performanse graf baza obično ostaju relativno konstantne, čak i kako skup podataka raste. To je zato što su upiti lokalizirani u dijelu grafa. Kao rezultat, vrijeme izvršavanja svakog upita proporcionalno je samo veličini dijela grafikona koji je prošao da bi udovoljio tom upitu, a ne veličini ukupnog grafa.

- **Fleksibilnost.** Kao razvijač i arhitekti podataka želimo povezati podatke kako zahtjeva domena, omogućujući strukturi i shemi da se pojave u skupu s našim rastućim razumijevanjem problematičnog prostora, a ne da se nametnu unaprijed, kad znamo najmanje o stvarnom obliku i razradi podataka. Grafovi su prirodno izmjenjivi, što znači da postojećoj strukturi možemo dodati nove vrste odnosa, nove čvorove i nove grafove bez ometanja postojećih upita i funkcionalnosti aplikacija. Ove stvari uglavnom imaju pozitivne posljedice na produktivnost programera i rizik projekta.

- **Agilnost.** Želimo biti u mogućnosti razvijati naš model podataka u skladu s ostatkom naše aplikacije, koristeći tehnologiju usklađenu s današnjim postupnim

postupcima isporuke softvera. Moderne graf baze omogućuju nam lagan razvoj i lagano održavanje sustava. Konkretno, priroda ovog modela podataka bez sheme omogućuje da razvijamo aplikaciju na kontroliran način.



Slika 3. Primjer modela graf baze podataka (Coronel i Morris, 2019)

Model graf baze podataka prikazan je na slici 3 model se sastoji od čvorova koji predstavljaju entitete, svaki entitet sadrži osnovne podatke o korisniku ID označava jedinstveni identifikator svakog čvora, Type je atribut koji označava vrstu korisnika može biti agent ili kupac ovisno o tome je izražena boja čvora agenti imaju plavu boju, dok čvorovi kupaca su označeni smeđom bojom. Ostali atributi čvorova koji su zajednički svima su ime i prezime, ipak čvorovi ne moraju imati istu strukturu što se može vidjeti na slici 3, tako da neki imaju dodatne atribute poput broja telefona i sl., veze između čvorova su označene lukovima. Lukovi isto tako mogu imati svoje atribute koji opisuju odnose između čvorova, zajednički atribut svim lukovima je ID njihov jedinstveni identifikator, dok ostali atributi lukova su rezultat međusobne interakcije korisnika unutar aplikacije. Tako da agenti imaju svoje kupce kojima pomažu, kupcu se agent može sviđati i u tom slučaju zapisuje se datum kada je kupac napravio tu akciju. Kupci međusobno mogu biti prijatelji i svaka vrsta veze sadržava svoj ID.

3.3.3 Stupčane baze podataka

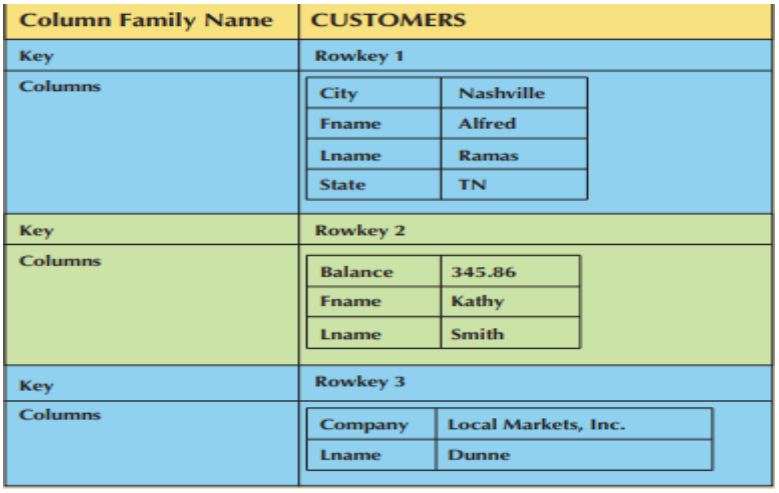
Pojam „stupčano orijentirane baze podataka“ može se odnositi na dvije različite vrste tehnologija koje se često miješaju jedna s drugom. U jednom smislu, stupčano orijentirana baza podataka može se odnositi na tradicionalne, relacijske tehnologije baza podataka koje koriste stupčano spremanje umjesto pohrane usmjerene na retke. Relacijske baze podataka prikazuju podatke u logičkim tablicama; međutim, podaci se zapravo pohranjuju u blokove podataka koji sadrže redove podataka. Svi podaci za određeni red pohranjuju se zajedno u nizu s nizom redova u istom bloku podataka. Stupčano orijentirane baze podataka pohranjuje podatke u blokove po stupcima umjesto u retke. Ova vrsta pohrane podataka djeluje vrlo dobro za baze podataka koje se prvenstveno koriste za pokretanje upita preko nekoliko stupaca, ali i mnogih redaka, kao što se događa u mnogim sustavima izvještavanja i skladištima podataka (Coronel i Morris, 2019).

Druga upotreba izraza baze podataka orijentiranih na stupce, koja se također naziva i baza podataka obitelji stupaca, to opisuje vrstu NoSQL baza podataka koje koriste koncept skladištenja podataka u stupce preko granica relacijskog modela. Kao ostale NoSQL baze podataka, ovaj pristup ne zahtijeva da se podaci podudaraju s unaprijed definiranim strukturama niti podržavaju SQL za upite (Coronel i Morris, 2019).

Model ove vrste NoSQL baza podataka nastao je zasnovan na Google-ovom BigTable radu. Kasnije su nastale mnoge poznate baze poput HBase, Hypertable i Cassandra. Cassandra je nastala kao projekt Facebook-a za lakše pretraživanje, ali Facebook-u ju je kasnije objavio kao bazu otvorenog koda, te je počelo njeno razvijanje u jednu od najpopularnijih stupčanih baza podataka, koju u današnje vrijeme koriste mnoge velike kompanije poput Netflix-a, Apple-a, GitHub-a i sl. glavna karakteristika ove baze podataka je njena fleksibilnost i brze obrade velikih količina podataka.

Podatci u stupčanim bazama kao što samo ime govori spremaju se u stupce, takvi podatci mogu se zamisliti kao obrnute tablice, gdje redak predstavlja kolekciju koja povezuje stupce s istim ključem. Stupci se spremaju u ključ-vrijednost parove, ključ označava ime stupca, dok vrijednost označava komponentu u koju se spremaju podatci.

Stupci ne moraju imati istu strukturu, ali trebali bi predstavljati slične zapise koji su definirani nazivom retka tablice. Skup stupaca koji pripadaju istom retku tablice nazivaju se obitelj stupaca.



Slika 4. Primjer obitelji stupaca (Coronel i Morris, 2019).

Obitelj stupaca sa slike 4 prikazuje kombinaciju ključ-vrijednost parova, u kojoj ključ predstavlja jedinstveni identifikator za svaki par, dok vrijednosti predstavljaju podatke o kupcima, poput imena, prezimena, grada, stanja i sl. ove vrijednosti ne moraju imati istu strukturu što se može vidjeti na slici 4.

3.3.4 Dokument baze podataka

Dokument baze podataka konceptualno su slične ključ-vrijednost bazama i mogu se smatrati podvrstom ključ-vrijednost baza. Dokument baza podataka je NoSQL baza podataka koja podatke pohranjuje u označene dokumente ključ-vrijednost parova. Za razliku od ključ-vrijednost baza gdje komponenta vrijednosti može sadržavati bilo koju vrstu podatka, dokument baze uvijek pohranjuju dokument u komponentu vrijednosti. Druga važna razlika je da ključ-vrijednost baze podataka ne pokušavaju razumjeti sadržaj vrijednosne komponente, dok dokument baze to čine (Coronel i Morris, 2019).

Ključ-vrijednost baze podatke spremaju u kante, podatci u dokument bazama se spremaju u kolekcije (engl. collections). Kolekcije predstavljaju skup podataka sačinjenih od ključa i vrijednosti iste ili slične strukture. Ključ mora biti jedinstveni identifikator koji

se može eksplicitno navesti, ako to nije učinjeno baza će ga sama dodijeliti dodanoj vrijednosti. Pretraživanje podataka najčešće se radi na vrijednosti, a ne pomoću ključa kao u ključ-vrijednost bazama podataka, iako je moguće pretraživanje po ključu u dokument bazama to nije česti slučaj. Pretraživanje po vrijednosti dokument bazama daje dodatnu sofisticiranost za dohvaćanje podataka. Velika prednost dokument baza je u tome što ne zahtijevaju ranije definiranu shemu, ovo svojstvo znači da svaki dokument unutar kolekcije može imati svoju shemu koja se može razlikovati od sheme drugih dokumenata. Dokument baze također podržavaju horizontalno skaliranje što rezultira bržim radon s podatcima jer su podatci podijeljeni na više servera, ako su podatci indeksirani što je još jedna mogućnost dokument baza podataka koja ubrzava pretraživanje, izmjenjivanje i brisanje podataka.

4 MongoDB

MongoDB u nastavku rada samo mongo prvi put objavljen je 2009. godine, relativno nova baza u usporedbi sa suvrenemenim divovima poput Oracle-a čija prva izdanja je objavljena davne 1970. godine. Kao dokument baza podataka grupirana je u NoSQL kategoriju, ističe se među distribuiranim ključ-vrijednost bazama podataka, reimplementacijama Amazon-ovog Dynamo-a i Google-ovog Big Table-a. mongo na mnogo načina je bliži MySQL-u nego stupčanim bazama podataka poput HBase-a (O'Higgins, 2011).

Mongo je snažna, fleksibilna i skalabilna baza podataka opće namjene. Kombinira sposobnost horizontalnog skaliranja s značajkama kao što su sekundarni indeksi, upiti raspona, sortiranje, sakupljanje (engl. aggregation) i geoprostorni indeksi (engl. geospatial indexies) (Chodorow, 2013).

Mongo je dizajniran da podržava horizontalno skaliranje. Njegov model podataka koji je orijentiran na dokumente olakšava podjelu podataka na više poslužitelja. Mongo se automatski brine za ravnomjernu raspodjelu i dohvaćanje podataka preko klastera, automatsku raspodjelu dokumenata i usmjeravanje korisničkih zahtjeva na isprave uređaje. To omogućava razvijačima da se usredotoče na razvijanje aplikacije, a ne na skaliranje iste. Kad klaster treba veći kapacitet, mogu se dodati novi uređaji i mongo će shvatiti kako se postojeći podaci trebaju raspodijeliti na njima.

Kao jedno od najefikasnijih rješenja za rad s dokument bazama podataka ,mongo osim osnovnih operacija dodavanja, izmjenjivanja, brisanja i dohvaćanja nude se i neke jedinstvene operacije koje većina drugih baza podataka nema ili nisu na razini koju mongo pruža.

Sekundarni indeksi omogućuju brže izmjenjivanje, dohvaćanje i brisanje podataka, indeksi koje mongo podržava su jedinstveni, kombinirani, tekstualni i geoprostorni indeksi.

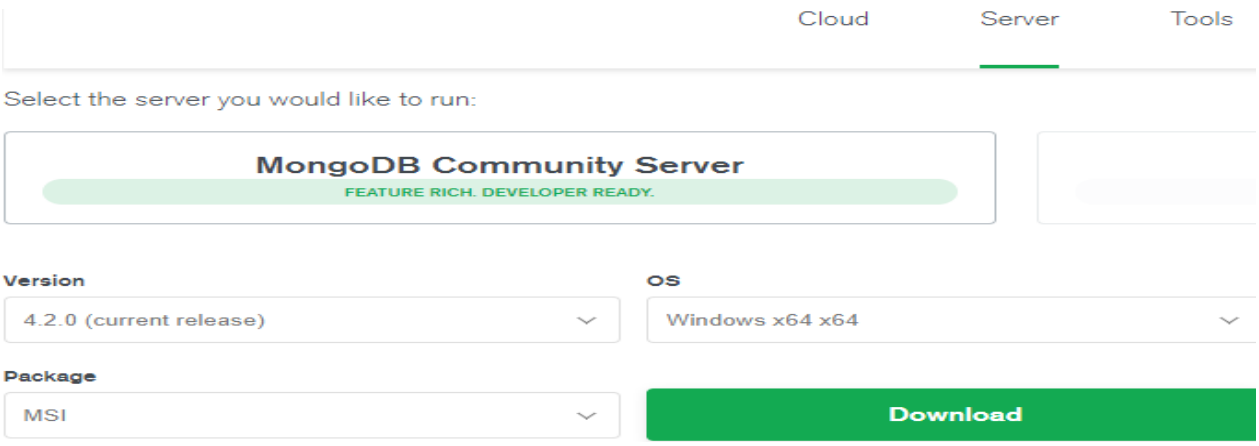
Sakupljanje je još jedna jedinstvena operacija mongo-a, ovo svojstvo omogućava složena sakupljanja podataka i njihovo oblikovanje pomoću cjevovoda sakupljanja (engl. aggregation pipeline) iz jednostavnih skupova podataka u kompleksnije skupove.

Specijalne kolekcije su još jedno svojstvo mongo-a, specijalne kolekcije nazivaju se ograničene kolekcije (engl. capped collections). Ove kolekcije su kolekcije ranije određenih veličina, kreiraju se eksplicitno i služe za brisanje starijih podataka automatski kada kolekcija pređe neku ranije definiranu veličinu u fizičkoj memoriji računala ili kada se pređe maksimalan broj dokumenata po kolekciji koji isto mora biti ranije definiran.

Mongo sam po sebi teži ostvarivanju mnogih mogućnosti SQL baza podataka na efikasniji način. Horizontalno skaliranje osim što donosi mnogo prednosti na razini performansi i bržeg rada baze, ipak ne može podržati neke osnovne funkcije SQL baza podataka, te su one odbačene da bi se moglo ostvariti horizontalno skaliranje. Tako da pridruživanja i komplekse transakcije nije moguće u mongo-u bar na način na koji se koriste u SQL-u.

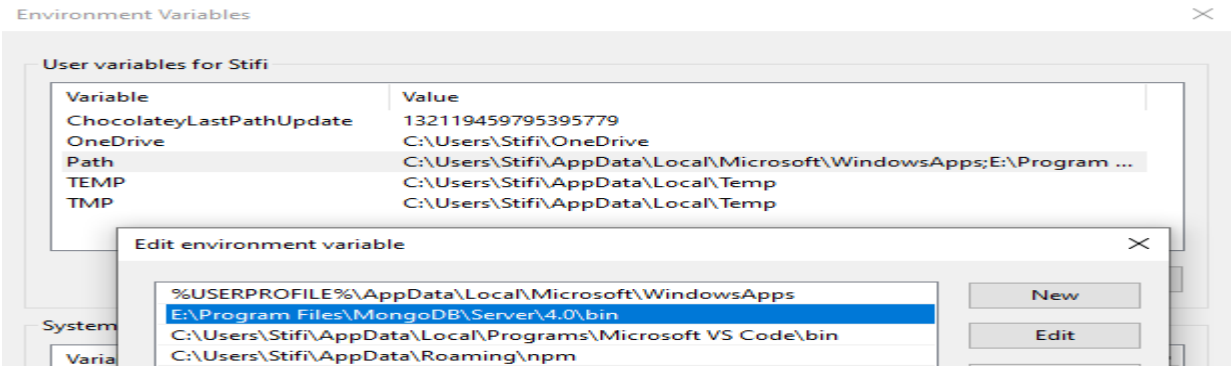
4.1 Instalacija mongo poslužitelja na lokalno računalo

Za instalaciju mongo poslužitelja na lokalno računalo u *Windows* operativnom sustavu potrebno je sa službene mongo stranice preuzeti najnoviju verziju *mongo Community Server-a* koja je zapravo besplatna inačica mongo poslužitelja koja podržava većinu operacija koje mongo pruža.



Slika 5. Preuzimanje *mongo Community Server-a* (13)

Pritiskom na dugme *Download* prikazano na slici 5 preuzeti će se najnovija verzija 4.2.0 mongo poslužitelja, koja je dovoljna za rad s mongo-om na lokalnom računalu. Upute za instalaciju mogu se pronaći u službenim dokumentima mongo-a na njihovoj stranici.



Slika 6. Dodavanje puta do mongo poslužitelja.

Nakon uspješne instalacije ako u *Environment Variables* ne postoji put do *bin* datoteke koja se nalazi na destinaciji gdje je instaliran mongo poslužitelj, potrebno ga je dodati označavanjem *path* varijable i pritiskom na dugme *edit*, otvorit će se novi prozor nakon toga potrebno je pritisnuti dugme *new* i dodati put do mongo bin datoteke koja se nalazi na lokaciji gdje je mongo poslužitelj instaliran što je prikazano na slici 6.

```
C:\WINDOWS\system32>net start MongoDB
The MongoDB Server service is starting...
The MongoDB Server service was started successfully.
```

Slika 7. Pokretanje MongoDB servisa.

Slika 7 prikazuje pokretanje mongo servisa koji mora biti pokrenut da bi rad s mongo poslužiteljem bio moguć izvršava se preko prozora naredbenog bloka (engl. command prompt). Potrebno je otvoriti naredbeni blok kao administrator, te upisati *net start MongoDB* ova naredba će pokrenuti servis mongo poslužitelja i omogućiti rad s istim. Za prekid rada servisa potrebno je ponovno upisati u naredbeni blok koji je pokrenut kao administrator *net stop MongoDB*.

4.2 JSON

JSON je oblik razmjene podataka za koji su se mnogi sustavi složili da koriste za komuniciranje podataka. JSON se često naziva „format razmjene podataka“ ili jednostavno „format podataka“ (Bassett, 2015).

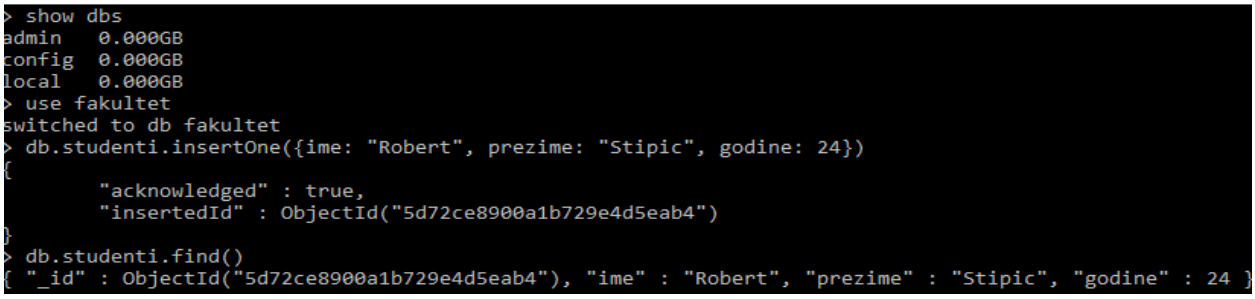
JSON označava JavaScript[5] objektnu notaciju (engl. JavaScript Object Notation). Naziv ovog formata razmjene podataka može zavarati ljude da misle kako će trebati naučiti JavaScript da bi razumjeli i koristili JSON. Postoji neka korist u učenju JavaScript-a prije JSON-a jer je nastao kao podskupina JavaScript-a, ali za razumijevanje JSON-a učenje JavaScript-a nije nužno (Bassett, 2015).

4.3 Mongo ljuska

Okruženje za rad s mongo poslužiteljem naziva se mongo ljuska[6] (engl. shell), pristup ljuski se odvija se preko naredbenog bloka potrebno je imati ranije pokrenut mongo servis inače ljuska se neće moći koristiti. Za pokretanje ljuske koristi se naredba *mongo* unutar naredbenog bloka i ako je mongo servis pokrenut upalit će se Mongo ljuska.

[5] JavaScript- skriptni programski jezik koji se koristi za razvijanje web aplikacija

[6] Mongo ljuska – sučelje za razvijanje MongoDB baza podataka



Slika 8. Primjena osnovnih operacija u mongo ljusci.

Osnovne operacije poput pravljenja nove baze, kolekcije, dodavanje i dohvaćanje podataka i JSON sintaksa koju koristi mongo prikazuje slika 8 prva operacija *show dbs* prikazuje postojeće baze na mongo poslužitelju može se vidjeti da su u mongo-u prilikom instalacije već napravljene tri baze *admin, config i local* koje služe za spremanje metapodataka (engl. metadata), upisivanjem komande *use fakultet* napravljena je nova baza podataka fakultet kada se prvi podatak doda u nju, ako baza već postoji na ovaj način ćemo pristupiti istoj. Naredba *db.studenti.insertOne()* db označava referencu na bazu koju trenutno koristimo u ovom slučaju fakultet, te će napraviti novu kolekciju studenti u ako ne postoji ili će dodati novi zapis ako kolekcija postoji. Naredbi *insertOne()* predaje se dokument koji se nalazi unutar vitičastih zagrada u ovom slučaju *{ime: "Robert", prezime: "Stipic", godine: 24}* ovaj zapis je u JSON obliku gdje ime, prezime i godine predstavljaju atribute ili polja koji se odvajaju zarezom, dok nakon dvotočke se tim istim atributima dodaje vrijednost, vrijednosti pod navodnicima zapisuju se kao tekstualni zapis, bez navodnika zapisuju se brojevi, mongo to prepoznaje i zapisuje ih kao decimalan zapis ukoliko nije definirana neka od drugih vrsta brojeva koje mongo podržava. Jedinstveni identifikator ukoliko nije eksplicitno naveden dodijeljen je od strane mongo-a kao *ObjectID* sva polja i vrijednosti zapisane u JSON obliku na bazi se pretvaraju u BSON[7] oblik koji predstavlja binarni JSON razumljiv računalu zapisan u kombinaciji nula i jedinica kako bi se ubrzale operacije za rad s podatcima i sačuvao prostor. Zadnja naredba *db.studenti.find()* koristi se za dohvaćanje podataka koji su zapisani na bazi fakultet u kolekciji studenti u BSON obliku, ali ih prilikom vračanja u ljusku pretvara u JSON oblik za lakše čitanje od strane korisnika, naredbi *find()* može se

[7] BSON – (engl. Binary JSON) Zapisuje JSON notaciju na računalo u obliku 0 i 1

dati argument da vrati samo određene podatke, ako to nije učinjeno vratit će sve dokumente unutar kolekcije o čemu će se govoriti u nastavku rada.

4.3.1 CRUD operacije

Osnovne operacije za rad s podatcima su CRUD operacije, one služe za dodavanje, izmjenjivanje, dohvaćanje i brisanje podataka. CRUD operacije izvršavaju se na razini kolekcije unutar baze podataka, sintaksa CRUD operacija je slična i razlikuje se u argumentima koji se predaju operaciji ovisno o njenoj namjeni. CRUD naredbe u mongo ljusci izgledaju ovako *db.MojaKolekcija.naredba({})*, *db* označava bazu podataka koja se trenutno koristi, *MojaKolekcija* je kolekcija na kojoj se izvršavaju naredbe, *naredba({})* naredba može biti dodavanje, dohvaćanje, izmjenjivanje i brisanje unutar uglatih zagrada se upisuju sve operacije koje se žele izvršiti pomoću određene naredbe, one se razlikuju ovisno od naredbe koja se koristi, dok se unutar vitičastih zagrada predaje dokument ili više njih u JSON obliku koji označava što bi naredba točno trebala raditi.

4.3.1.1 Dodavanje podataka

Podatci se dodaju s dvije naredbe *insertOne({})* koristi se za dodavanje jednog dokumenta u bazu kao što je prikazano na Slici 8, druga funkcija *insertMany([{}, {}])* sadrži listu dokumenata koji se zapisuju između uglatih zagrada, ova funkcija koristi se za dodavanje dva ili više dokumenata od jednom. Svaki dokument nalazi se unutar vitičastih zagrada, zasebni dokumenti odvajaju se zarezom.

```
> db.studenti.insertMany([{ime: "Marko", prezime: "Markovic", godine: 21, redovan: true}, {ime: "Josip", prezime: "Josipovic"
, redovan: false, godine: 22}, {"_id": "1", ime: "Ivan", prezime: "Ivanovic", redovan: false, godine: 30, spol: "M"}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5d73d1f5106427ed90d1efe9"),
                ObjectId("5d73d1f5106427ed90d1efea"),
                "1"
        ]
}
```

Slika 9. Primjena funkcije *insertMany()*.

U bazu *fakultet* sa slike 8, dodana su tri dokumenta u kolekciju *studenti*, sva tri dokumenta nalaze se između uglatih zagrada koje označavaju dodavanje više dokumenata od jednom u bazu. Slika 9 prikazuje tri dokumenta različitih struktura, dok

prva dva sadrže polja *ime, prezime, godine* koja su ranije objašnjen, ali i jedan novi tip podataka zapisan u polju *redovan* koji označava Boolean-ov operator (engl. Boolean) ovaj operator može biti točan (engl. true) ili netočan (engl. false), te na ovaj način određuje dali je student redovan ili izvanredan. Treći dokument je još zanimljiviji jer ima dodatno polje *spol* ovo dokazuje da struktura dokumenata u istoj kolekciji ne mora biti ista, tako da dokumenti ne moraju imati ista polja ako se nalaze u istoj kolekciji. *"_id": "1"* prikazuje eksplicitno dodavanje jedinstvenog identifikatora nekom dokumentu, dok je za prva dva dokumenta jedinstveni identifikator automatski generiran od strane mongo-a i razlikuju se samo u zadnjoj znamenki jer mongo generira jedinstvene identifikatore po vremenu i redoslijedu dodavanja dokumenata, ali garantira da dva dokumenta ne mogu imati isti automatsko generirani jedinstveni identifikator.

4.3.1.1.1 Dodavanje ugniježđenih dokumenata

Zanimljivo svojstvo koje se može ostvariti putem mongo-a je dokument unutar dokumenta, ovo svojstvo naziva se ugnježđivanje dokumenata. Ugniježđeni dokumenti mogu podražavati do 100 razina ugnježđivanja tako da jedan dokument može imati do 100 pod dokumenata.

```
> db.studenti.insertOne({ime: "Iva", prezime: "Ivic", spol: "Z", godine: 19, fakultet:{naziv: "Fakultet ekonomije i turizma",
 smjer: "ekonomija", vrsta:"preddiplomski", godina: 2}})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5d73ea92106427ed90d1efeb")
}
```

Slika 10. Dodavanje ugniježđenog dokumenta.

Pomoću funkcije *insertOne()* prikazanoj na slici 10 dodan je još jedan dokument u kolekciju *studenti*, ovaj dokument sadrži osnovna polja kao i kod ranije dodanih dokumenata u kolekciju, ali sadržava ugniježđeni dokument koji je određen poljem *fakultet*, te označava još jedan dokument unutar osnovnog dokumenta koji sadržava podatke o studentu, dok dokument *fakultet* sadržava podatke o fakultetu kojeg student pohađa.

4.3.1.1.2 Dodavanje redova vrijednosti

Red (engl. Array) vrijednosti također se može dodati u mongo ljusci, red predstavlja listu vrijednosti za određeno polje. Red opisuje jedno polje kojemu se dodjeljuje više vrijednosti sličnih značajki, a može biti bilo koja vrsta podatka koju mongo podržava.

```
> db.studenti.insertOne({ime: "Mislav", prezime: "Misic", godine: 22, spol: "M", hobi: ["nogomet", "biciklizam"]})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5d73fb62106427ed90d1efec")
}
```

Slika 11. Dodavanje reda vrijednosti.

Osnovne informacije o studentu poput imena, prezimena, godina i spola dodaju se na ranije utvrđen način, ipak polje *hobi* se razlikuje unutar uglatih zagrada pomoću kojih se dodaju nizovi vrijednosti dodane su dvije vrijednosti *nogomet* i *biciklizam* u ovom slučaju tekstualni zapisi, ali ove vrijednosti su mogle biti bilo koja vrsta podataka koju mongo podržava uključujući čak i dokumente koji isto mogu biti zasebne vrijednosti u redovima prikazuje slika 11.

4.3.1.2 Dohvaćanje podataka

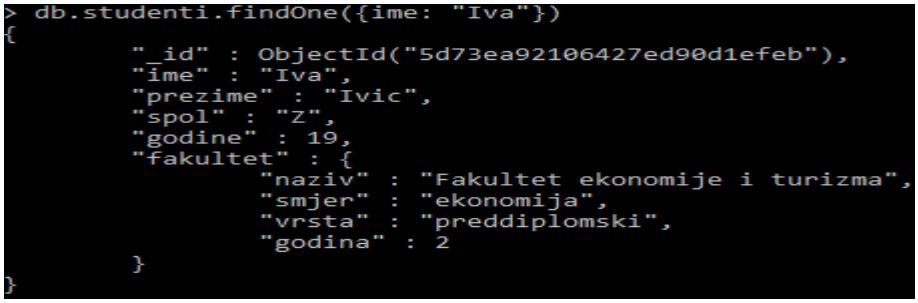
Operacije, filteri i operatori koriste se za dohvaćanje podataka u mongo ljusci. Operacije predstavljaju funkcije koje se koriste za dohvaćanje podataka, operacije *insertOne()* i *insertMany()* ranije su objašnjene za dodavanje podataka u bazu, operacije za dohvaćanje podataka su *find()* i *findOne()* ovim operacijama dodaju se filteri i operatori. Filteri označavaju po kojem polju ili poljima želimo pretraživati podatke, dok operatori služe za određivanje parametara za vrijednosti koje želimo da dohvaćeni podatci imaju.

```
> db.studenti.find()
{ "_id" : ObjectId("5d72ce8900a1b729e4d5eab4"), "ime" : "Robert", "prezime" : "Stipic", "godine" : 24 }
{ "_id" : ObjectId("5d73d1f5106427ed90d1efe9"), "ime" : "Marko", "prezime" : "Markovic", "godine" : 21, "redovan" : true }
{ "_id" : ObjectId("5d73d1f5106427ed90d1efea"), "ime" : "Josip", "prezime" : "Josipovic", "redovan" : false, "godine" : 22 }
{ "_id" : "1", "ime" : "Ivan", "prezime" : "Ivanovic", "redovan" : false, "godine" : 30, "spol" : "M" }
{ "_id" : ObjectId("5d73ea92106427ed90d1efeb"), "ime" : "Iva", "prezime" : "Ivic", "spol" : "Z", "godine" : 19, "fakultet" :
{ "naziv" : "Fakultet ekonomije i turizma", "smjer" : "ekonomija", "vrsta" : "preddiplomski", "godina" : 2 } }
{ "_id" : ObjectId("5d73fb62106427ed90d1efec"), "ime" : "Mislav", "prezime" : "Misic", "godine" : 22, "spol" : "M", "hobi" :
[ "nogomet", "biciklizam" ] }
```

Slika 12. Operacija *find()*.

Operacija *find()* bez ikakvih filtera i operatora vratit će sve podatke unutar kolekcije kao na slici 12, ako broj dokumenata veći od 20 koje će operacija *find()* vratiti, prikazuje se prvih 20 dokumenata u obliku pokazivača, te je nakon toga potrebno upisati funkciju *it*

za prikaz idućih 20 i tako sve dok se ne dođe do zadnjeg dokumenta u svrhu štednje memorije i lakšeg prikaza podataka.



Slika 13. Operacija *findOne()* s filterom.

Operacija *findOne()* vratiti će jedan dokument iz kolekcije, čak ako postoji više dokumenata koji odgovaraju kriterijima pretraživanja. Unutar vitičastih zagrada predaje se filter *find* operacijama u slučaju sa slike 13 polje koje se koristi za filtriranje je *ime* studenta i pretraživati će studente čije ime je jednakost imenu Iva, operacije može sadržavati i više filtera ovisno o potrebi. Vidljivo je da operacija *findOne()* vratila dokument u drukčijem obliku nego operacija *find()* sa slike 12, razlog ovoga je samo da podatci budu lakši za čitanje korisniku ljuske, ovo svojstvo može se ostvariti i s operacijom *find()* tako da se na kraju doda operacija .*pretty()* koja služi samo za lakši pregled dohvaćenih podataka, tako da bi operacija *find()* izgledala ovako *db.studenti.find().pretty()*, te bi se svi podatci ispisali u obliku kao u funkciji *findOne()*.

4.3.1.2.1 Operatori dohvaćanja podataka

Jasnije je koje se operacije koriste za dohvaćanje podataka i kako se dodaje filter u operaciju za dohvaćanje željenih dokumenata, zadnji parametar koji se koristi za dohvaćanje dokumenata su operatori. Operatori predstavljaju moćno sredstvo za traženje dokumenata u kolekciji, operatori mogu biti logički, usporedbe (engl. comparison), procjene (engl. evaluation), elementarni, geoprostorni, projekcijski (engl. projection) i drugi, najbitniji će biti objašnjeni u radu.

4.3.1.2.1.1 Operatori usporedbe

Operatori usporedbe koriste se za uspoređivanje vrijednosti koje se nalaze unutar dokumenata, ova vrsta operatora najčešće se koristi za usporedbu numeričkih podataka. Operatori se zapisuju u obliku *$operator* kako bi mongo mogao prepoznati da se radi o operatoru. Operatori usporedbe su:

- $eg – pronalazi vrijednosti koje su jednake navedenoj vrijednosti
- $gt – pronalazi vrijednosti koje su veće od navedene vrijednosti
- $gte - pronalazi vrijednosti koje su veće ili jednake navedenoj vrijednosti
- $in – prima više vrijednosti u niz i pronalazi sve koje su jednake navedenim vrijednostima
- $lt - pronalazi vrijednosti koje su manje od navedene vrijednosti
- $lte – pronalazi vrijednosti koje su manje ili jednake navedenoj vrijednosti
- $ne – pronalazi vrijednosti koje nisu jednake navedenoj vrijednosti
- $nin – prima više vrijednosti u niz i pronalazi sve koje nisu jednake navedenim vrijednostima

```
> db.studenti.find({godine: {$lt: 22}})
{ "_id" : ObjectId("5d73d1f5106427ed90d1efe9"), "ime" : "Marko", "prezime" : "Markovic", "godine" : 21, "redovan" : true }
{ "_id" : ObjectId("5d73ea92106427ed90d1efeb"), "ime" : "Iva", "prezime" : "Ivic", "spol" : "Z", "godine" : 19, "fakultet" :
{ "naziv" : "Fakultet ekonomije i turizma", "smjer" : "ekonomija", "vrsta" : "preddiplomski", "godina" : 2 } }
```

Slika 14. Primjena operatora usporedbe *$lt*.

Većina operatora usporedbe koristi se na isti način, osim operatora *$nin* i *$in* koji primaju niz vrijednosti, slika 14 prikazuje operator *$lt* koji pronalazi sve dokumente u kolekciji *studenti* čije godine su manje od 22, operator se dodaje filteru u ovom slučaju *godine* tako da se stavi u vitičaste zagrade, te mu se doda vrijednost za koju želimo dohvatiti dokumente.

```
> db.studenti.find({godine: {$in: [24, 30]}})
{ "_id" : ObjectId("5d72ce8900a1b729e4d5eab4"), "ime" : "Robert", "prezime" : "Stipic", "godine" : 24 }
{ "_id" : "1", "ime" : "Ivan", "prezime" : "Ivanovic", "redovan" : false, "godine" : 30, "spol" : "M" }
```

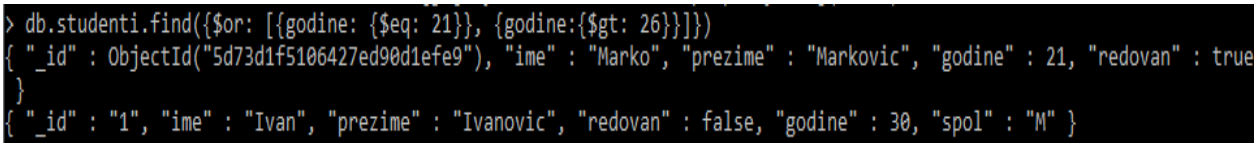
Slika 15. Primjena operatora usporedbe *$in*.

Operatori *$in* i I *$nin* koriste malo drugačiju sintaksu od ostalih operatora usporedbe, operator se kao i ostali stavlja u vitičaste zagrade, dok se dodatno vrijednosti koje se traže stavljaju u uglate zagrade kao na slici 15. Operator *$in* dohvaća sve dokumente čije vrijednosti su jednake navedenima unutar uglatih zagrada za filtrirano polje, a operator *$nin* dohvatio bi sve dokumente čija vrijednost nije jednaka navedenima.

4.3.1.2.1.2 Logički operatori

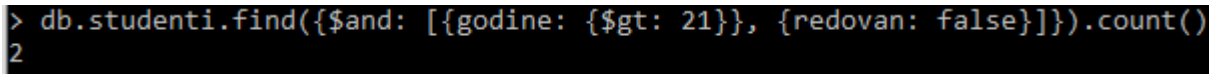
Postoje četiri logička operatora u mongo-u, svrha ovih operatora je usporediti jeli neka vrijednost dodana filteru točna ili netočna, te ovisno o prirodi operatora vratiti rezultate. Logički operatori su:

- $and – ovaj operator predstavlja logičko i, ako su obije tvrdnje točne za neki dokument funkcija će ga vratiti.
- $not – vraća dokumente koji ne odgovaraju izrazu *find()* operacije.
- $or – ovaj operator predstavlja logičko ili, vraća dokumente za koje je bar jedna tvrdnja točna.
- $nor – vraća sve dokumente za koje niti jedna tvrdnja nije točna.



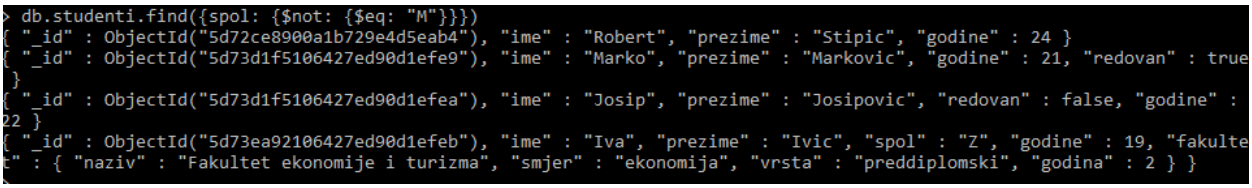
Slika 16. Primjena logičkog operatora *$or*.

Dohvaćanje podataka pomoću logičkog operatora ili prikazano je na slici 16, unutar operacije prvo se navodi operator *$or* koji prima niz vrijednosti unutar uglatih zagrada, operacija sa slike dohvatit će sve studente kojima su godine jednake 21 ili veće od 26 pomoću ranije objašnjenih operatora usporedbe. Drugi logički operator *$nor* predstavlja suprotnost *$or* operatoru točnije taj operator bi dohvatio sve dokumente studenata čije godine nisu jednake 21 ili nisu veće od 26.



Slika 17. Primjena logičkog operatora *$and* i operacije *count()*.

Logičko i koje se dobiva operatorom *$and* koristi se kada se žele dohvatiti podatci za koje su sve navedene tvrdnje točne, primjer sa slike 17 dohvatit će sve dokumente kojima su godine veće od 21, te nisu redovni studenti. Dodatna operacija koja se može koristiti prilikom dohvaćanja podataka je *count()*, pomoću ove operacije ne vraćaju se vrijednosti dokumenata nego broj dokumenata koji odgovara kriterijima *find()* operacije.



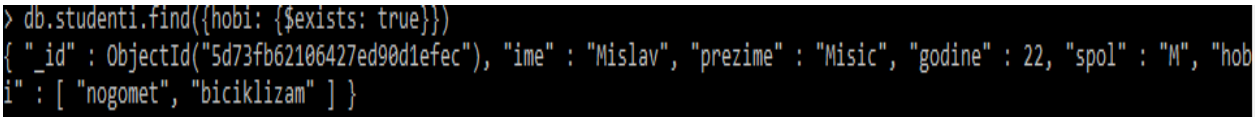
Slika 18. Primjena logičkog operatora *$not*.

Zadnji logički operator *$not* koristi se kada želimo dohvatiti dokumente za koje izraz operacije nije točan, ali isto tako vratit će sve dokumente koji ne sadrže filtrirano polje jer ni u tom slučaju izraz operacije nije točan jer dokument ne sadrži to polje što prikazuje slika 18. Za razliku od ostalih logičkih operatora logično ne navodi se poslije filtera i vratit će sve studente čiji spol nije *M*.

4.3.1.2.1.3 Elementarni operatori

Za razliku od ranije objašnjenih operatora elementarni operatori ne koriste vrijednosti polja za dohvaćanje nego kriteriji dohvaćanja su sama polja, te postoje dva elementarna operatora:

- $exists – dohvaća dokumente u kojima postoji navedeno polje.
- $type – dohvaća dokumente u kojima polje odgovara određenom tipu podatka.



Slika 19. Primjena elementarnog operatora *$exists*.

Ovaj operator koristi se kada želimo dohvatiti dokumente u kojima određeno polje postoji ili ne postoji, prima dvije vrijednosti *true* ili *false*, *true* će dohvatiti sve dokumente gdje polje postoji, dok *false* dohvaća dokumente u kojima navedeno polje ne postoji. Unutar operacije ovaj operator navodi se nakon filtera odnosno polja koje želimo provjeriti kao na slici 19.

```
> db.studenti.find({hobi: {$type: "array"}})
{ "_id" : ObjectId("5d73fb62106427ed90d1efec"), "ime" : "Mislav", "prezime" : "Misic", "godine" : 22, "spol" : "M", "hob
i" : [ "nogomet", "biciklizam" ] }
```

Slika 20. Primjena elementarnog operatora *$type*.

Ovaj operator koristi se kada želimo dohvatiti dokumente u kojima navedeno polje odgovara tipu podatka koji je naveden unutar *find()* operacije, u primjeru sa slike 20 dohvatit će sve dokumente čije polje *hobi* je tipa niz, dok će se o ostalim tipovima podataka govoriti u nastavku rada.

4.3.1.2.1.4 Ostali operatori

Mongo podržava veliki broj operatora, tako da svi neće biti objašnjeni u ovom radu, ali moguće ih je pronaći na službenim stranicama mongo-a. Neki od operatora će biti objašnjeni u ovom poglavlju, a neki u idućim poglavljima kada u slučajevima upotrebe kada se koriste ti operatori poput operatora za indeksiranje i geoprostornih operatora.

U ovom poglavlju bit će objašnjena dva operatora procjene:

- $regex – operator koji dohvaća dokumente koji sadrže izraz naveden u operaciji
- $expr – operator koji omogućava dohvaćanje dokumenta, korištenjem izraza koji uključuje dva ili više polja.

```
> db.studenti.find({"fakultet.naziv": {$regex: /ekonomije/}})
{ "_id" : ObjectId("5d73ea92106427ed90d1efeb"), "ime" : "Iva", "prezime" : "Ivic", "spol" : "Z", "godine" : 19, "fakultet"
: { "naziv" : "Fakultet ekonomije i turizma", "smjer" : "ekonomija", "vrsta" : "preddiplomski", "godina" : 2 } }
```

Slika 21. Primjena *$regex* operatora.

Dohvaćanje dokumenata na osnovu nekog izraza, točnije sadrži li odabrano polje neki izraz ili vrijednost moguće je pomoću operatora *$regex* prikazano je na slici 21. Ovaj operator najčešće se koristi za pretraživanje tekstualnih zapisa, iako je bilo moguće pretražiti polje *naziv* u ugniježđenom dokumentu *fakultet* na ovaj način *db.studenti.find({"fakultet.naziv": "ekonomija"})*, ali niti jedan dokument ne bi bio dohvaćen jer na ovaj način mongo traži punu jednakost u polju *naziv* koje se mora staviti unutar navodnih znakova jer je dio ugniježđenog dokumenta, te je potrebno staviti naziv dokumenta i polja pod navodne znakove kako bi mongo znao gdje treba pretraživati podatke. Korištenjem operatora *$regex* mongo će vratiti sve dokumente u kojima je vrijednost koja se traži dio ili cijela vrijednost polja. Jedina razlika s obzirom na druge operatore je što se vrijednost koja se dohvaća stavlja unutar kosih zagrada.

```
> db.studenti.find({$expr:  {$gt: ["$godine", "$fakultet.godina"]}})
```

Slika 22. Primjena *$expr* operatora.

Kada se dohvaća dokument za koji je potrebno pretražiti kolekciju na osnovu dva polja koristi se *$expr* ovaj operator uzima dva polja koja se stavljaju unutar uglatih zagrada i odvajaju se zarezom, polja se stavljaju pod navodne znakove i ispred ide znak *$*. Slika 22 prikazuje dohvaćanje dokumenata u kojima je vrijednost polja *godine* veća od vrijednosti polja *godina* u ugniježđenom dokumentu *fakultet*.

4.3.1.2.2 Projektiranje podataka

Pronalaženje i dohvaćanje podataka u obliku spremljenom na bazu pomoću operacija, filtera i operatora mnogo je jasnije, ipak mongo može oblikovati podatke koje dohvaća i vratiti ih u željenom obliku koji nije zapisan na taj način u bazi. Za ovo koristi se projektiranje ili oblikovanje podataka koje se definira kao drugi filter unutar operacija za dohvaćanje podataka.

```
> db.studenti.find({godine: {$gte: 24}}, {ime: 1, prezime: 1, _id:0})
{ "ime" : "Robert", "prezime" : "Stipic" }
{ "ime" : "Ivan", "prezime" : "Ivanovic" }
```

Slika 23. Projektiranje podataka.

Oblikovanje podataka u potrebno obliku je moguće u mongo-u, ovo svojstvo ostvaruje se preko drugog filtera koji se predaje operaciji dohvaćanja podataka. Slika 23 prikazuje slučaj upotrebe kada se unutar kolekcije *studenti* dohvaćaju svi studenti koji imaju 24 ili više godina što je definirano prvim filterom, dok se unutar drugog filtera odabiru polja koja se žele dohvatiti zato je potrebno navesti polje koje želimo dohvatiti i označiti ga s vrijednošću 1, dok sva ostala polja koja nisu navedena osim *_id* polja ne trebaju se eksplicitno navoditi da ne želimo ih prikazati, jedino polje *_id* će uvijek biti unutar oblikovanih podataka ukoliko se eksplicitno ne doda vrijednost 0.

4.3.1.3 Izmjenjivanje podataka

Pronalaženje željenih podataka objašnjeno je u prethodnom poglavlju, kada je jasnije kako pronaći željene podatke moguće ih je izmijeniti bez pamćenja svih podataka koji se nalaze na bazi kojih može biti mnogo. Podatci se mogu izmjenjivati na razini cijelog dokumenta ili samo određenih polja koja se žele mijenjati, isto vrijedi za nizove podataka i ugniježđene dokumente. Osim što možemo izmijeniti postojeće podatke pomoću operacija izmijene podataka mogu se dodavati nova polja, nizovi podataka ili ugniježđeni dokumenti. Operacije koje se koriste za ovu svrhu su *updateOne()* i *updateMany()*.

```
> db.studenti.updateOne({_id: ObjectId("5d72ce8900a1b729e4d5eab4")}, {$set: {godine: 25, redovan: true, spol: "M"}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Slika 24. Primjena operacije *updateOne()* i operatora *$set.*

Operacija *updateOne()* koristi se za izmjenu jednog dokumenta unutar kolekcije kojeg prvog pronađe, iako može biti više dokumenata koji odgovaraju filteru ova operacija izmijenit će samo prvi, za izmjenu više dokumenata koristi se operacija *updateMany()*. Slika 24 prikazuje izmjenu dokumenta koji se pronalazi pomoću filtera koji može biti bilo koja izvedenica filtera iz operacije dohvaćanja podataka, u ovom slučaju polje *_id* koje je zapravo jedinstveni identifikator, te predstavlja najsigurniji način pronalaženja određenog dokumenta jer ne mogu postojati dva dokumenta s istim *_id* poljem. Drugi dio operacije koji se stavlja unutar vitičastih zagrada i odvaja zarezom od

filtera koji je zapravo definira način na koji želimo izmijeniti postojeća polja ili dodati nova. Navedeni *_id* pripada studentu s imenom *Robert* i prezimenom *Stipic* izgled ovoga dokumenta prikazan je na slici 8, te se navedenom dokumentu izmjenjuje vrijednost polja *godine*, dok polja *redovan* i *spol* se dodaju jer dokument s pripadajućim jedinstvenim identifikatorom nije imao polja *redovan* i *spol* uz pomoć operatora *$set* koji se koristi za izmjenjivanje vrijednosti polja ili novog polja pripadajuće vrijednosti navedenoj unutar operacije *updateOne()* ili *updateMany()*.

```
> db.studenti.updateMany({spol: "M"}, {$inc: {godine: 2}}, {$set: {redovan: true}})
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
```

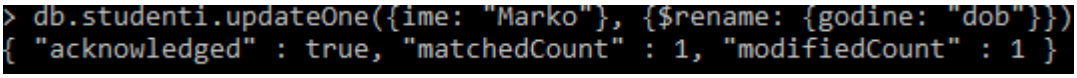
Slika 25. Primjena operacije *updateMany()* i operatora *$inc.*

Povećanje vrijednosti polja moguće je pomoću operatora izmijene *$inc*, ovaj operator uzima vrijednost polja i povećava za određenu vrijednost navedenoj u operaciji, slika 25 prikazuje povećanje polja *godine*, svim studentima čiji *spol* je *M*, korištenjem operacije *updateMany()* neće se izmijeniti samo prvi podataka koji je pronađen nego svi koji odgovaraju filteru. Operator *$inc* može se kombinirati s drugim operatorima poput operatora *$set*, tako da će osim povećanja godina za 2 svim studentima koji odgovaraju filteru vrijednost polja *redovan* biti postavljena na *true* ili ako polje ne postoji kod studenta bit će napravljeno s vrijednošću *true*.

Osim operatora *$inc* koji služi za povećavanje numeričkih vrijednosti, numeričke vrijednosti mogu koristiti druge operatore na isti način poput *$min*, *$max* i *$mul*. Operator *$min* promijenit će vrijednost polja na navedenu samo ako je trenutna vrijednost polja veća od navedene. Operator *$max* označava suprotnost operatoru *$min*, te se koristi kada je trenutna vrijednost polja manja od navedene u tom slučaju vrijednost polja bit će promijenjena. Operator *$mul* koristi se kada vrijednost nekoga polja želimo pomnožiti s navedenom vrijednošću unutar operacije.

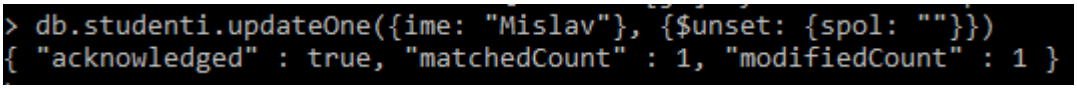
4.3.1.3.1 Izmjenjivanje polja

Osim što je moguće izmjenjivati ili dodavati polja s pripadajućim vrijednostima unutar operacija *updateOne()* i *updateMany()*. Izmjena imena polja ili brisanje polja isto je moguće u mongo-u. Operatori koji se koriste za ovo su *$unset* i *$rename*.



Slika 26. Primjena operatora *$rename*.

Operator *$rename* koristi se kada se želi izmijeniti naziv postojećeg polja, pomoću filtera pronađen je student *Marko*, te naziv polja *godine* izmijenjen je u *dob* kao na slici 26.

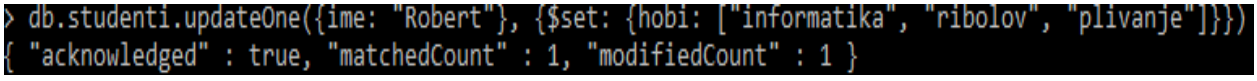


Slika 27. Primjena operatora *$unset*.

Operator *$unset* koristi se kada se želi obrisati polje dokumenta kao na slici 27, u ovom slučaju studentu *Mislav* obrisano je polje *spol*, ali da bi ovaj operator potrebno je dodijeliti neku vrijednost inače neće uspjeti operacija, tako da je moguće samo dodati prazne navodne znakove da bi se određeno polje obrisalo.

4.3.1.3.2 Izmjenjivanje redova vrijednosti

Redovi vrijednosti također se mogu izmjenjivati, redovi vrijednosti ne koriste istu strukturu kao obična polja kojima se dodaje vrijednost, polje reda vrijednosti kao što samo ime govori može sadržavati više vrijednosti, a ne samo jednu iz tog razloga postoje posebni operatori koji se koriste za izmjenjivanje nizova vrijednosti. Dodatni operatori koji se koriste za izmjenjivanje nizova vrijednosti su *$push*, *$pull*, *$pop* i *$addToSet*.



Slika 28. Izmjenjivanje reda vrijednosti.

Izmjenjivanje reda vrijednosti može se ostvariti pomoću operatora *$set*. Jedina razlika prilikom izmjenjivanja redova vrijednosti je što se vrijednosti koje se žele dodati stavljaju u uglate zagrade kao na slici 28. Ukoliko red vrijednosti ne postoji napraviti će se s pripadajućim vrijednostima unutar operacije, ali ako postoji postojeće vrijednosti biti će izmijenjene s vrijednostima navedenim u operaciji izmijene.

```
> db.studenti.updateOne({ime: "Robert"}, {$push: {hobi: "rukomet"}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Slika 29. Primjena operatora *$push*.

Na slici 29 prikazano je dodavanje novih vrijednosti u polje reda vrijednosti, ali da pri tome ostanu stare vrijednosti ostvaruje se preko operatora *$push*, za razliku od operatora *$set* koji će obrisati postojeće vrijednosti i dodati nove. Na isti način s istom sintaksom koristi se operator *$pull* koji služi za brisanje određene vrijednosti iz polja, isto pravilo vrijedi i za operator *$addToSet* samo što će ovaj operator dodati vrijednost ukoliko navedena vrijednost ne postoji, dok operator *$push* će dodati vrijednosti koje već postoje što može dovesti do ponavljajućih vrijednosti.

```
> db.studenti.updateOne({ime: "Robert"}, {$pop: {hobi: 1}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Slika 30. Primjena operatora *$pop*.

Operator *$pop* koristi se kada je potrebno izbaciti prvu ili zadnju vrijednost iz reda vrijednosti. Polju reda vrijednosti dodjeljujemo vrijednost 1 kada želimo izbrisati zadnju vrijednost kao na slici 30, a kada želimo izbrisati prvu vrijednost dodjeljuje se vrijednost -1.

4.3.1.4 Brisanje podataka

Brisanje podataka je zadnja CRUD operacija, u pravilu najlakša ako je poznato kako dohvaćati podatke, operacija brisanja koristi samo filter koji se koristi prilikom pronalaženja dokumenata kako bi se pronašao dokument koji se želi obrisati. Operacije za brisanje podataka su *deleteOne()* i *deleteMany()*.

```
> db.studenti.deleteOne({godine: {$gt: 31}})
{ "acknowledged" : true, "deletedCount" : 1 }
```

Slika 31. Primjena operacije *deleteOne()*.

Operacije *deleteOne()* i *deleteMany()* rade na istom principu kao operacije *findOne()* i *findMany()* samo što neće dohvatiti cijeli dokument, nego ga obrisati. U primjeru sa slike 31 obrisat će se prvi student kojeg operacija pronađe, a da mu je vrijednost polja

godine veća od 31, dok bi operacija *deleteMany()* obrisala sve dokumente čija je vrijednost polja *godine* veća od 31.

4.3.2 Kolekcije

Podatci se pomoću pripadajućih operacija spremaju u dokumente, dokumenti se spremaju u kolekcije, kolekcije se spremaju u bazu podataka, nakon što je jasnije kako napraviti kolekciju i raditi s CRUD operacijama unutar kolekcije. U ovom poglavlju bit će objašnjeno neke operacije koje se koriste za rad s kolekcijama, koje vrste podataka možemo spremati u dokumente i kako povezivati dokumente iz različitih kolekcija ako su pripadajuće veze potrebne i samim time stvoriti neku vrstu sheme između dokumenata.

4.3.2.1 Operacije za rad s kolekcijama

Osim što je moguće napraviti kolekcije, moguće je izbrisati istu u cijelosti. Za brisanje cijele kolekcije koriste se dvije operacije *db.imekolekcije.deleteMany({})* i *db.imekolekcije.drop()* prva operacija *deleteMany({})* bez ikakvog filtera obrisat će sve dokumente unutar kolekcije, samim time i kolekciju jer u mongo-u kolekcija postoji samo ako postoje podatci unutar kolekcije. Druga operacija napravit će istu stvar pomoću operacije *drop()* koja ne prima nikakve argumente unutar sebe.

Za prikaz kolekcija unutar baze koristi se operacija *show collections* koja ispisuje sve kolekcije unutar trenutne baze podataka. Osim normalnih kolekcija koje su bile korištene u ovom radu, moguće je napraviti ograničene kolekcije koje se rade eksplicitno. Ograničene kolekcije su specijalna vrsta kolekcije za koje se odredi količina podataka koja se može spremiti i koliko dokumenata maksimalno kolekcija može sadržavati. Ova vrsta kolekcije korisna je za podatke koji nakon određenog vremena nisu potrebni, te se automatski brišu nakon što kolekcija pređe ograničenja navedena prilikom njenog pravljenja. Operacija za pravljenje ograničenih kolekcija je *db.createCollection("imeKolekcije", {capped: true, size: 15000, max: 10})*, *imeKolekcije* označava naziv kolekcije u bazi, *capped: true* označava da je kolekcija ograničena, *size: 15000* označava maksimalnu veličinu podataka spremljenih u kolekciju u ovom slučaju 15000 bajtova, *max: 10* je optimalna opcija koja označava koliko je dokumenata

maksimalno moguće u kolekciji, kada se ovo ograničenje pređe najstariji dokument se briše iz kolekcije.

4.3.2.2 Tipovi podataka unutar dokumenata

Polja unutar dokumenata mogu podržavati različite tipove podataka. Neki tipovi podataka su bili ranije korišteni u radu poput: jedinstvenog identifikatora, tekstualnih zapisa, redova vrijednosti, ugniježđenih dokumenata, Boolean-ovih operatora i običnih decimalnih brojeva u 64 bitnom zapisu koje mongo automatski koristi ako eksplicitno nije naveden drugi tip numeričkog zapisa. Ostali tipovi podataka su:

- NumberInt() – koristi se za zapisivanje cijelih brojeva veličine do 32 bita, raspon 32 bitnog cijelog broja je +-2,147,483,647.
- NumberLong() – koristi se za zapisivanje cijelih brojeva veličine do 64 bita, raspon 64 bitnog cijelog broja je +-9,223,372,036,854,775,807.
- NumberDecimal() – koristi se za zapisivanje decimalnih brojeva do 124 bita, ova vrsta brojeva garantira maksimalnu preciznost do 34 decimalne znamenke, koristi se ukoliko je potrebna velika preciznost rada s brojevima.
- Null – koristi se kada neko polje nema vrijednost, ali ipak polje postoji unutar dokumenta.
- Date() – koristi se za zapisivanje vremena u 64 bitnom broju, jedinica zapisivanja su milisekunde.
- Timestamp() – označava zapisivanje vremena u 64 bitnom broju, ali mongo garantira jedinstvenost ovog zapisa, tako da osim vremena prati se redoslijed dodavanja dokumenata kako bi svaki Timestamp() bio jedinstven.

4.3.2.3 Referenciranje dokumenata iz različitih kolekcija

Izradu sheme između dokumenata može se ostvariti putem referenciranja, iako mongo ne zahtijeva nikakvu povezanost dokumenata samim time izradu nekakvog

oblika sheme. Shemu je ipak moguće izraditi ukoliko je potrebna na bazi tako da se polje jednog dokumenta referencira na jedinstveni identifikator drugog dokumenta.

4.3.2.3.1 Referenciranje veze jedan naprema jedan

Ova veza koristi se kada jednom dokumentu pripada samo jedan dokument iz druge kolekcije s kojim je povezan. Ovo vezu na primjeru kolekcije *studenti* iz prethodnih poglavlja može se zamisliti tako da jedan student može imati samo jedan indeks, ali isto tako jedan indeks pripada samo jednom studentu u svrhu toga koristi se veza jedan naprema jedan.

```
> db.indeksi.insertOne({jmbag: "3030-A", datum_izdavanja: new Date("2014-07-28"), dodano: new Timestamp(), godina_izdavanja: NumberInt(2014)})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5d7b8531918c003c03fe851b")
}
> db.studenti.updateOne({ime: "Robert"}, {$set: {indeks: ObjectId("5d7b8531918c003c03fe851b")}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Slika 32. Primjena veze jedan na jedan korištenjem reference.

U novu kolekciju indeksi na slici 32 dodaje se novi dokument čiji se jedinstveni identifikator koristi za stvaranje veze jedan na jedan između dva dokumenta različitih kolekcija. Novi dokument u kolekciji *indeksi* ima polje *jmbag* koje je tekstualni zapis, dok ostala tri polja koriste tipove podataka koji nisu korišteni u radu. Polje *datum_izdavanja* ima vrijednost datuma spremljenog u obliku *"godina-mjesec-datum"*, polje *dodano* ima vrijednost trenutnog vremena dodavanja na bazu, polje *godina_izdavanja* ima vrijednost 32 bitnog cijelog broja. Pomoću operacije *updateOne()* studentu imena *Robert* dodaje se novo polje *indeks* koje ima vrijednost jedinstvenog identifikatora dokumenta spremljenog u kolekciju *indeksi* koji je zapravo referenca na taj dokument. Slučaj veze jedan naprema jedan bolje je implementirati pomoću ugniježđenih dokumenata, ipak ovo svojstvo moguće je ostvariti pomoću referenci.

4.3.2.3.2 Referenciranje veze jedan naprema mnogo

Veza jedan naprema mnogo koristi se kada jednom dokumentu pripada više dokumenata iz druge kolekcije. Ovu vezu na primjeru kolekcije *studenti* može se zamisliti tako da jedan student može imati samo jedno prebivalište, ali grad prebivališta može imati više studenata koji prebivaju u njemu.

```
> db.gradovi.insertOne({ime: "Zagreb", zupanija: "Grad Zagreb", postanski_broj: 10000})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5d7aa0e36be20488a1983e2b")
}
> db.studenti.updateMany({spol: "M"}, {$set: {prebivaliste: ObjectId("5d7aa0e36be20488a1983e2b")}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Slika 33. Primjena veze jedan naprema više korištenjem reference.

U novu kolekciju gradovi dodaje se dokument s poljima *ime*, *zupanija* i *postanski_broj*. Pomoću operacije *updateMany()* dodaje se jedinstveni identifikator grada svim studentima čiji je spol *M* u novo polje *prebivaliste* prikazuje slika 33. Više studenata može imati grad *Zagreb* kao prebivalište, ali jedan student može imati samo jedno prebivalište.

4.3.2.3.3 Referenciranje veze mnogo naprema mnogo

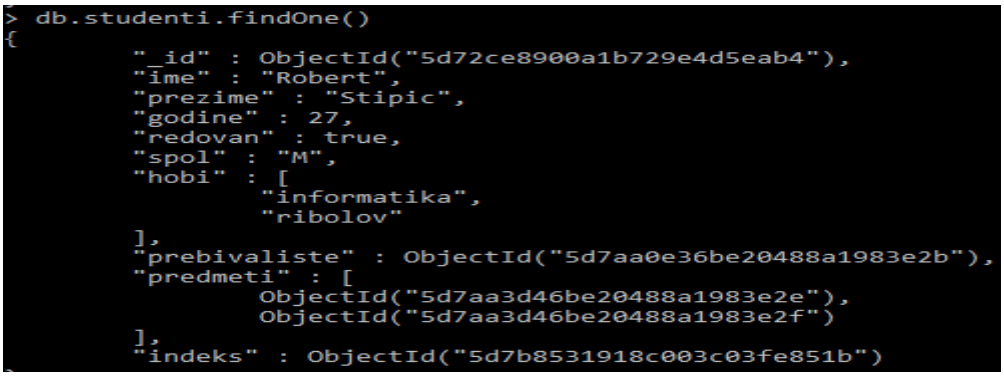
Veza mnogo naprema mnogo koristi se kada više dokumenata imaju pripadajuće veze s više dokumenata iz druge kolekcije. Redovi vrijednosti koriste se za ostvarivanje ovakvih veza. Ovu vezu na primjeru kolekcije *studenti* može se zamisliti tako da da više studenata može pohađati više predmeta, ali predmet može pohađati više studenata.

```
> db.predmeti.insertMany([{naziv: "ekonomija", ects_bodovi: 6},{naziv: "informatika", ects_bodovi: 5}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5d7aa3d46be20488a1983e2e"),
                ObjectId("5d7aa3d46be20488a1983e2f")
        ]
}
> db.studenti.updateMany({godine: {$gte: 22}},{$set: {predmeti:[ObjectId("5d7aa3d46be20488a1983e2e"), ObjectId("5d7aa3d4
6be20488a1983e2f")]}})
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
```

Slika 34. Primjena veze više naprema više korištenjem referenci.

Korištenjem operacije *insertMany()* zapisuju se dva predmeta u novu kolekciju *predmeti*, te su automatski generirana dva jedinstvena identifikatora što prikazuje slika 34. Pomoću operacije *updateMany()* svim studentima koji imaju veću ili jednaku

vrijednost polja *godine* od 22, dodaje se red vrijednosti u polje *predmeti* u koje se dodaju jedinstveni identifikatori kolekcije *predmeti*. Više studenata može pohađati više predmeta, ali predmet može pohađati više studenata.



Slika 35. Primjer studenta sa svim vrstama referenci.

Slika 35 prikazuje studenta sa svim vrstama objašnjenih veza, veze se referenciraju pomoću jedinstvenih identifikatora koje pripadaju dokumentima iz drugih kolekcija. Tako da se polja *prebivaliste* i *indeks* referenciraju na jedan dokument iz pripadajućih kolekcija, dok se polje *predmeti* referencira na više dokumenata iz kolekcije *predmeti*.

4.3.3 Indeksiranje podataka

Indeksiranje podataka koristi se za poboljšanje performansi operacija dohvaćanja, izmjenjivanja i brisanja podataka unutar kolekcije. Točnije sve operacije koje pretražuju kolekcije, izradom indeksa na određena polja unutar kolekcije. Indeksi nisu zamjena za kolekciju nego dodatak za brže izvršavanje navedenih operacija koje je zapravo sortirana lista vrijednosti pomoću koje mongo može brže pretraživati podatke za indeksirano polje, tako da svako indeksirano polje sadrži pokazivač na cijeli dokument unutar kolekcije. Dodavanje indeksa na neko polje ima cijenu, prilikom dodavanja novih dokumenata indeks za to polje mora se napraviti i staviti na odgovarajuće mjesto u indeksiranom polju, tako da indeksiranje svih polja unutar kolekcije nije najbolje rješenje jer će znatno usporiti dodavanje novih dokumenata, iz tog razloga indekse je najbolje koristiti na poljima koja se često koriste za pretraživanje podataka. Jedino polje koje je uvijek indeksirano je *_id*.

Pretraživanje podataka može se izvršavati na dva načina pretraživanje kolekcije (engl. collection scan) i pretraživanje indeksa (engl. index scan). Pretraživanje kolekcije radi na principu da se pretražuje cijela kolekcija dok se ne pronađu željeni dokumenti za kolekcije s velikim brojem dokumenata pretraživanje kolekcije može potrajati, pretraživanje indeksa izvršava se na sortiranim vrijednostima tako da mongo može preskakati određene vrijednosti jer su sortirane, samim time smanjiti vrijeme potrebno da se pronađu željeni dokumenti.

4.3.3.1 Jedinstveni indeksi

Indeksi koji se izrađuju na jednom polju u kolekciji nazivaju se jedinstvenim indeksima. Ova vrsta indeksa koristi vrijednosti jednog polja kako bi se izradila sortirana lista vrijednosti polja u padajućem (engl. descending order) ili rastućem obliku (engl. ascending order). Padajući oblik označava da najveća vrijednost ide prva, a ostale vrijednosti idu prema najmanjoj koja je zadnja. Rastući oblik označava da najmanja vrijednost ide prva, a ostale vrijednosti idu prema najvećoj prateći redoslijed.

```
> db.studenti.createIndex({godine: 1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
```

Slika 36. Izrada jedinstvenog indeksa.

Sintaksa operacije za izradu jedinstvenog indeksa je *createIndex()*, indeks se izrađuje na jedno polje u primjeru sa slike 36, indeksirano polje bit će *godine* u kolekciji *studenti*, vrijednost koja se dodaje polju može biti 1 za sortiranje u rastućem obliku, dok vrijednost -1 se koristi za sortiranje u padajućem obliku. Za brisanje indeksa koristi se ista sintaksa sa slike, jedina razlika je da umjesto operacije *createIndex()* koristi se operacija *dropIndex()*.

4.3.3.2 Kombinirani indeksi

Kombinirani indeks omogućava izradu indeksa na dva ili više polja na način da se vrijednosti polja koje se koriste za izradu kombiniranog indeksa grupiraju u jedan indeks, koji je kombinacija vrijednosti oba polja. Tako da operacija pretraživanja koja ima oba polja kao filter koristit će indeks pretraživanje. Ova vrsta indeksa koristi se za poboljšanje performansi pretraživanja koja koriste dva ili više filtera.

```
> db.studenti.createIndex({ime: 1, spol: 1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "ok" : 1
}
```

Slika 37. Izrada kombiniranog indeksa.

Kombinirani indeks izrađuje se na isti način kao jedinstveni indeks, jedina razlika je što kombinirani indeks kao argument operacije *createIndex()* prima dva ili više polja, koja naprave jedan kombinirani indeks koji je zapravo spoj vrijednosti tih polja kao na slici 37. Zanimljivo svojstvo kombiniranog indeksa zasniva se na tome se može koristiti s lijeva na desno, tako da prvo polje koje definirano prilikom izrade može se koristi kao zaseban indeks ako je jedini filter po kojemu se vrijednosti pretražuju što označava da se prilikom pretraživanja podataka koristilo samo polje *ime* opet bi se izvršilo indeks pretraživanje. Slučaj s tri indeksa bi omogućio iako je indeks izrađen na tri polja, ako se za filter koriste samo prva dva polja opet će biti moguće indeks pretraživanje, dok god filter prati izradu indeksa s lijeva na desno.

4.3.3.3 Tekstualni indeksi

Tekstualni indeksi su posebna vrsta indeksa koju mongo podržava, koristi se za spremanje indeksa u poljima koja sadrže neki tekst u sebi koji je duži od jedne riječi, ova vrsta indeksa sprema svaku riječ zasebno u red vrijednosti za jedan dokument. Na engleskom jeziku mongo može prepoznati bitne riječi u tom slučaju riječi koje se stalno

ponavljaju poput veznika i prefiksa neće spremiti u indeks. Tako da se ova vrsta indeksa može zamisliti kao indeks reda vrijednosti najbitnijih riječi unutar nekog teksta i moguće je imati samo jedan tekstualni indeks po kolekciji.

```
> db.studenti.createIndex({"fakultet.naziv": "text"})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 3,
        "numIndexesAfter" : 4,
        "ok" : 1
}
```

Slika 38. Izrada tekstualnog indeksa.

Slika 38 prikazuje izradu tekstualnog indeksa koji se izrađuje na isti način kao jedinstveni ili kombinirani, osim što za vrijednost indeksiranom polju koje je u ovom slučaju pod navodnim znacima jer se radi o polju u ugniježđenom dokumentu na koje se isto može izraditi indeks daje vrijednost *"text"* kako bi se izradio tekstualni indeks.

```
> db.studenti.findOne({$text: {$search: "ekonomije"}})
{
        "_id" : ObjectId("5d73ea92106427ed90d1efeb"),
        "ime" : "Iva",
        "prezime" : "Ivic",
        "spol" : "Z",
        "godine" : 19,
        "fakultet" : {
                "naziv" : "Fakultet ekonomije i turizma",
                "smjer" : "ekonomija",
                "vrsta" : "preddiplomski",
                "godina" : 2
```

Slika 39. Pretraživanje tekstualnog indeksa.

Tekstualni indeks pretražuje se pomoću specijalnog operatora *$text* koji označava da se pretražuje tekstualni zapis, pomoću operatora *$search* definira se vrijednost koja se pretražuje kao na slici 39. Polje koje se pretražuje nije potrebno definirati jer po kolekciji može postojati maksimalno jedan tekstualni indeks, te ne treba paziti na mala i velika slova jer sve riječi se spremaju malim slovima u tekstualni indeks. Operacija koja se koristi za pregled postojećih indeksa u kolekciji je *db.imekolekcije.getIndexes()*.

4.3.4 Rad s geoprostornim podatcima

Mongo podržava rad s geoprostornim podatcima, točnije s lokacijama zapisanima u obliku koordinata s karte. Geoprostorni podatci mogu se spremati na bazu, dohvaćati najbliže zapise u bazi nekoj određenoj lokaciji, dohvaćati sve dokumente unutar nekog radijusa, dohvaćati sve dokumente unutar nekog područja i provjeriti nalazi li se neka točka unutar nekog područja. Za rad s geoprostornim podatcima koristi se geoJSON[8], notacija koja koristi JSON sintaksu za rad s geoprostornim podatcima. Neke od operacija geoJSON-a su implementirane u mongo-u i mogu se koristiti za rad s geoprostornim podatcima unutar kolekcija.

4.3.4.1 Spremanje geoprostornih podataka

Geoprostorni podatci spremaju se u kolekcije zajedno s ostalim podatcima unutar dokumenta, ipak da bi rad s geoprostornim podatcima bio ostvariv i da mongo može prepoznati da se radi o geoprostornim podatcima koristi se posebna sintaksa zapisivanja geoprostornih podataka.

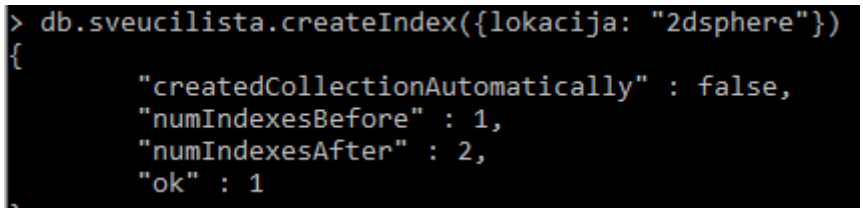
```
> db.sveucilista.insertMany([{naziv: "Sveuciliste Jurja Dobrile u Puli", mjesto: "Pula", lokacija: {type: "Point", coordinates:[13.854182, 44.86738
6]}}, {naziv: "Sveuciliste Josipa Jurja Strossmayera Osijek", mjesto: "Osijek", lokacija: {type: "Point", coordinates:[18.6823586, 45.5588705]}}, {
naziv: "Sveuciliste u Rijeci", mjesto: "Rijeka", lokacija: {type: "Point", coordinates:[14.407021, 45.3348518]}}, {naziv: "Sveuciliste u Zagrebu",
mjesto: "Zagreb", lokacija: {type: "Point", coordinates:[15.9698015, 45.8105703]}}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5d7be955918c003c03fe851c"),
                ObjectId("5d7be955918c003c03fe851d"),
                ObjectId("5d7be955918c003c03fe851e"),
                ObjectId("5d7be955918c003c03fe851f")
        ]
}
```

Slika 40. Dodavanje geoprostornih podataka u kolekciju.

Korištenje operacije *insertMany()* dodaju se četiri sveučilišta u kolekciju *sveucilista*, dokumenti se sastoje od dva polja *naziv* i *mjesto* koja opisuju sveučilište koje se dodaje u kolekciju, *lokacija* je ugniježđeni dokument koji služi za spremanje geoprostornih podataka. Geoprostorni podatci moraju se spremiti u ugniježđeni dokument s imenima

[8] geoJSON – JSON notacija prilagođena za rad s geoprostornim podatcima

polja kao na slici 40, polje *type* definira vrstu geoprostornog podatka koji se dodaje *"Point"* označava da se radi o točki na karti koja ima svoju geografsku dužinu i širinu. Polje *coordinates* prima koordinate točke u red vrijednosti, prva vrijednost uvijek mora biti geografska dužina, dok druga vrijednost uvijek mora biti geografska širina. Na ovaj način spremljena su četiri sveučilišta s odgovarajućim geografskim koordinatama u kolekciju *sveucilista*.

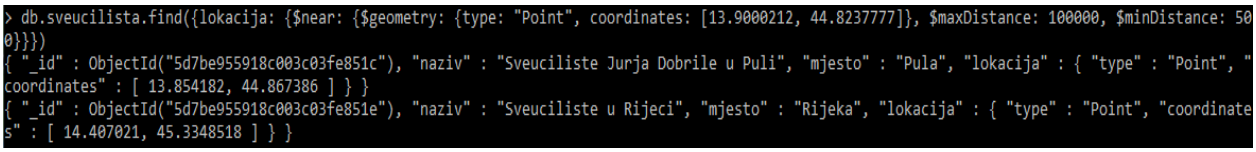


Slika 41. Izrada geoprostornog indeksa.

Za pretraživanje geoprostornih podataka i upotrebu geoprostornih operatora potrebno je napraviti geoprostorni indeks, ovakav indeks izrađuje se tako da se polju ugniježđenog dokumenta kao vrijednost *createIndex()* operacije definira *"2dsphere"* kao na slici 41.

4.3.4.2 Geoprostorni operator $near

Operator *$near* koristi se za dohvaćanje dokumenata koji se nalaze blizu geografske točke koja je navedena kao filter za pretraživanje kolekcije, uz određene parametre koliko maksimalno dokumenti mogu biti udaljeni od točke, ali koliko minimalno dokumenti moraju biti udaljeni od točke definirane u filteru.



Slika 42. Primjena operatora *$near* i *$geometry*.

Za dohvaćanje dokumenata pomoću *$near* operatora potrebno je prvo kao filter navesti polje *lokacija* u kojemu su spremljeni geoprostorni podatci. Zatim se navodi operator *$near* koji služi za dohvaćanje lokacija koje su unutar parametara od maksimalno 100000 metara i minimalno 500 metara, pomoću operatora *$maxDistance* i

$minDistance čije vrijednosti su izražene u metrima kao na slici 42. Operatoru *$near* kao vrijednost daje se operator *$geometry* koji govori mongo-u da podatci koje će primiti su geoJSON objekt. geoJSON objekt sastoji se od točke i koordinata u redu vrijednosti, gdje prva koordinata predstavlja geografsku dužinu, a druga geografsku širinu. Operacija *find()* vratila je dva dokumenta koja se nalaze unutar parametara definiranih operatorima *$maxDistance* i *$minDistance* od točke navedene u filteru.

4.3.4.3 Geoprostorni operator $geoWithin

Operator *$geoWithin* koristi se za pronalaženje dokumenata koji se nalaze unutar određenog oblika odnosno područja koje je spoj više geografskih točaka međusobno povezanih da bi stvorili neko geografsko područje. Neko područje mora imati minimalno tri geografske točke koje se mogu spojiti i samim time napraviti granice određenog područja.

```
> const l1 = [15.50818, 45.91157]
> const l2 = [18.98163, 45.88864]
> const l3 = [19.06952, 45.2329]
> const l4 = [15.4248, 45.21548]
```

Slika 43. Spremanje geografskih točaka u konstante.

U privremene konstante spremljene su geografska dužina i širina kao red vrijednosti svaka konstanta predstavlja jednu geografsku točku, ovo nije potrebno napraviti da bi operator *$geoWithin* radio, geografske koordinate spremljene su u privremene konstante kako bi sintaksa *$geoWithin* operatora bila lakša za čitanje i pisanje ovo je prikazano na slici 43.

```
> db.sveucilista.find({lokacija: {$geoWithin: {$geometry: {type: "Polygon", coordinates: [[l1, l2, l3, l4, l1]]}}}})
{ "_id" : ObjectId("5d7be955918c003c03fe851d"), "naziv" : "Sveuciliste Josipa Jurja Strossmayera Osijek", "mjesto" : "Osijek", "lokacija" : { "type
" : "Point", "coordinates" : [ 18.6823586, 45.5588705 ] } }
{ "_id" : ObjectId("5d7be955918c003c03fe851f"), "naziv" : "Sveuciliste u Zagrebu", "mjesto" : "Zagreb", "lokacija" : { "type" : "Point", "coordinat
es" : [ 15.9698015, 45.8105703 ] } }
```

Slika 44. Primjena operatora *$geoWithin*.

Operator *$geoWithin* izvršava se na polju *lokacija*, ali za razliku od operatora *$near* polje *type:"Polygon"* napravit će poligon koji će spojiti geografske koordinate spremljene

u konstante *l1*, *l2*, *l3* i *l4* u neki oblik područja. Polje *coordinates* uzima skup točaka koje se međusobno povezuju u neki geografski oblik čije su veze granice područja, s tim da prva točka mora se navesti kao prva i zadnja točka poligona kako bi se područje zatvorilo. Bez konstanti koordinate bi se unutar polja *coordinates* zapisivale kao na slici 43, nakon što je određen geografski oblik za koji se dohvaćaju dokumenti koji su unutar istog, funkcija *find()* vraća dva sveučilišta što prikazuje slika 44.

4.3.3.4 Geoprostorni operator $geoIntersects

Poligon kao vrstu geoprostornog podatka moguće je spremiti na bazu, tako da mongo omogućava da se izvršavaju operacija nad poligonima poput *$geoIntersects* operatora koji provjerava nalazi li se geografska točka unutar nekog geografskog područja.

```
> db.podrucje.insertOne({naziv: "Podrucje 1", lokacija: {type: "Polygon", coordinates: [[l1, l2, l3, l4, l1]]}})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5d7bf0fd918c003c03fe8520")
}
```

Slika 45. Dodavanje poligona u kolekciju.

Dodavanje poligona u kolekciju prati istu sintaksu kao dodavanje geografskih točaka, samo polju *type* kao vrijednost predaje se *Polygon*, a polje *coordinates* kao vrijednost prima više točaka koje sačinjavaju određeno područje prikazuje slika 45. Točke koje sačinjavaju *Podrucje 1* su varijable sa slike 43.

```
> db.podrucje.find({lokacija: {$geoIntersects: {$geometry: {type: "Point", coordinates: [16.37787, 45.62995]}}}})
{ "_id" : ObjectId("5d7bf0fd918c003c03fe8520"), "naziv" : "Podrucje 1", "lokacija" : { "type" : "Polygon", "coordinates" : [ [ [ 15.50818, 45.91157
], [ 18.98163, 45.88864 ], [ 19.06952, 45.2329 ], [ 15.4248, 45.21548 ], [ 15.50818, 45.91157 ] ] ] } }
```

Slika 46. Primjena operatora *$geoIntersects*.

Operator *$geoIntersects* provjerava nalazi li se točka koja se predaje operatoru *$geometry* unutar nekog područja u bazi pomoću *find()* operacije. Slika 46 prikazuje da je točka koja se provjerava unutar područja sa slike 43, te je operacija dohvatila to područje jer odgovara parametrima filtera.

4.3.3.5 Geoprostorni operator $centerSphere

Dohvaćanje dokumenata koji se nalaze unutar određenog radijusa, moguće je pomoću operatora *$centerSphere* koji se koristi u kombinaciji s operatorom *$geoWithin*. Operator *$centerSphere* kao prvi parametar prima točku koja označava središte radijusa, dok drugi parametar predstavlja veličinu radijusa za kojeg se dohvaćaju sve točke koje su unutar radijusa. Drugi parametar radi na principu broj kilometara/6378.1 kako bi se napravio radijus željenog broja kilometara od središnje točke radijusa.

```
> db.sveucilista.find({lokacija: {$geoWithin: {$centerSphere: [[15.4248, 45.21548], 200/ 6378.1]}}})
{ "_id" : ObjectId("5d7be955918c003c03fe851e"), "naziv" : "Sveuciliste u Rijeci", "mjesto" : "Rijeka", "lokacija" : { "type" : "Point", "coordinate
s" : [ 14.407021, 45.3348518 ] } }
{ "_id" : ObjectId("5d7be955918c003c03fe851f"), "naziv" : "Sveuciliste u Zagrebu", "mjesto" : "Zagreb", "lokacija" : { "type" : "Point", "coordinat
es" : [ 15.9698015, 45.8105703 ] } }
{ "_id" : ObjectId("5d7be955918c003c03fe851c"), "naziv" : "Sveuciliste Jurja Dobrile u Puli", "mjesto" : "Pula", "lokacija" : { "type" : "Point", "
coordinates" : [ 13.854182, 44.867386 ] } }
```

Slika 47. Primjena operatora *$centerSphere*.

Za dohvaćanje podataka unutar određenog radijusa koristi se kolekcija *sveucilista*, pomoću operatora *$geoWithin* dohvaćaju se sveučilišta koja su unutar radijusa definiranog operatorom *$centerSphere* prikazano je na slici 47. Prvi parametar unutar uglatih zagrada predstavlja geografsku dužinu i širinu točke koja se koristi kao središte radijusa, te iz tog razloga nije potrebno koristiti operator *$geometry* jer središte radijusa može biti samo točka što nije potrebno prethodno definirati. Drugi parametar *200/6378.1* označava za koju udaljenost središte radijusa ima do rubova točnije 200 kilometara. Operacija *find()* dohvaća tri dokumenta u kolekciji *sveucilista* koja se nalaze unutar radijusa definiranog operatorom *$centerSphere*.

5 Zaključak

Podatci se koriste već dugo vremena, može se reći da su podatci stari koliko i čovječanstvo koje ih koristi. U današnje vrijeme rad s podatcima doveden je na visoku razinu, ponajviše zahvaljujući pojavi računala i svim mogućnostima koja računala omogućuju u radu s podatcima. Prve tradicionalne baze podataka datiraju još u 70-te godine prošlog stoljeća njihov razvoj traje još od tada, ipak pojavom interneta i stalnog tehnološkog napretka javili su se novi izazovi u radu s podatcima koji su rezultirali novim rješenjima na tom području odbacujući tradicionalne načine rada s podatcima sa svrhom pronalaženja optimalnijih i efikasnijih rješenja koja prate stalni tehnološki napredak.

SQL dugi niz godina bio je jedino rješenje koje se prihvaćalo kao idealan pristup za rad i spremanje podataka u vanjsku memoriju računala. U ovom radu objašnjeni su modeli podataka koje SQL koristi, kako su strukturirani podatci i pripadajuće veze u modelima podataka koje SQL podržava. Što je zapravo SQL koje su njegove karakteristike i značajke, ali isto tako kako je nastao. Primjena SQL-a objašnjena je na primjerima kako bi se primijetile razlike između SQL i NoSQL pristupa, te kada je bolje koristiti SQL prije NoSQL-a, ali i obrnuto.

Termin NoSQL prvi put upotrijebljen je 1998. godine, ipak razvoj NoSQL nije počeo odmah nego je dugo bio tema rasprava i znanstvenih radova, ali bez konkretnih sustava koji su koristili NoSQL tehnologiju. SQL baze podataka koje su se koristile na internetu za društvene mreže, web tražilice, online trgovine počele su nailaziti na velike probleme u rad s podatcima koristeći SQL pristup za rad s podatcima. Velike kompanije počele su tražiti rješenja za ove probleme i pronašle su ih odbacujući neka svojstva SQL pristupa i razvijati NoSQL sustave koji su se pokazala učinkovitijima i efikasnijima u situacijama gdje su SQL baze podataka nailazile na probleme. Iako je termin NoSQL prvi put upotrjebljen 1998. godine, radovi velikih kompanija Google-ov Bigtable i Amazon-ov Dynamo izdani 2006 i 2007 započeli su sve veći interes za distribuirane baze podataka, samim time i NoSQL pokret. Ovi radovi rezultirali su razvojem mnogih najpoznatijih NoSQL baza podataka koje su u velikom broju razvijene između 2007. i 2009. godine,

od tada njihova upotreba i razvoj konstantno rastu, iako i dalje nemaju teoretsku osnovu nude mnoga efikasna rješenja za današnje potrebe u radu s podatcima.

U ovom radu objašnjene su četiri najraširenije baze podataka ključ-vrijednost, graf, dokument i stupčane baze podataka. Stvari koje su karakteristične za svaku od ovih baza, po čemu se razlikuju jedna od druge, ali i koje su im sličnosti. Navedena su najpoznatija aplikacijska rješenja za svaki tip NoSQL baze podataka kojih ima mnogo u današnje vrijeme. Iako je logično pomisliti da NoSQL baze istog tipa rade na istom principu to ne mora biti točno, zajedničko im je strukturiranje i kategorizacija podataka, ali operacije, programski jezici, slučajevi upotrebe, načini upotrebe mogu se bitno razlikovati. Neka osnovna svojstva koja dijele ove baze su mogućnost različitog strukturiranja istih podataka, horizontalno skaliranje, brzina u radu s velikim količinama podataka, repliciranje podataka i prilagođavanje promjenama.

Upotreba NoSQL baze na primjerima detaljno je objašnjena pomoću dokument baze MongoDB koja nudi visok spektar operacija za rad s podatcima, rad s posebnim vrstama podataka, ali i jako dobro implementiranu JSON notaciju za rad s bazom. Pomoću ovih primjera moguće je shvatiti kako pomoću indeksa napraviti optimalniju i bržu bazu podataka u MongoDB-u, te jedno zanimljivo svojstvo MongoDB-a koji omogućava rad s geoprostornim podatcima na relativno jednostavan način na samoj bazi podataka.

6 Literatura

1. Bassett, L. (2015) *Introduction to JavaScript Object Notation*. 1. Edited by Meg Foley. Sebastopol: O'Reilly Media, Inc. [20.08.2019]
2. Chodorow, K. (2013) *MongoDB: The Definitive Guide*. 2. Sebastopol: O'Reilly Media, Inc.[19.08.2019]
3. Coronel, C. and Morris, S. (2019) *Database Systems: Design, Implementation, and Management*. 13. [18.08.2019]
4. Fowler, A. (2015) *NoSQL For Dummies*. 2. New Jersey: John Wiley & Sons, Inc. [18.08.2019]
5. Hills, T. (2016) *NoSQL and SQL Data Modeling Bringing Together Data, Semantics, and Software*. 1. Technics Publicationss. [18.08.2019]
6. Manger, R. (2012) *Baze podataka*. 1. Edited by S. Gračan. Zagreb: Element d.o.o. [08.08.2018]
7. O'Higgins, N. (2011) *MongoDB and Python*. 1. Sebastopol: O'Reilly Media, Inc. [21.08.2019]
8. Robinson, I., Webber, J. and Eifrem, E. (2013) *Graph Databases*. 1. Edited by L. Mike and J. Nathan. O'Reilly Media, Inc.[20.08.2019]
9. Šribar, J. and Motik, B. (2014) *DEMISTIFICIRANI C++*. Zagreb: 4. Element d.o.o. [12.08.2018]
10. Vaish, G. (2013) *Getting Started with NoSQL*. 1. Packt Publishing Ltd. [18.08.2019]
11. Hastorun, D. Jampani, M. i dr. (2007) *Dynamo: Amazon's Highly Available Key-value Store*. 1. Washington: Amazon [18.07.2019]
12. https://docs.riak.com/riak/kv/2.2.3/learn/why-riak-kv/index.html [20.08.2019]
13. https://www.mongodb.com/download-center/community [10.07.2019]