MongoDB

Pavlović, Maja

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

Maja Pavlović

MongoDB

Završni rad

PULA, 2018.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

Maja Pavlović

MongoDB

Završni rad

JMBAG: 0303061351, redoviti student
Studijski smjer: Preddiplomski sveučilišni studij Informatika

Predmet: Baze podataka
Mentor: doc. dr. sc. Tihomir Orehovački

Pula, listopad 2018. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani _____, kandidat za prvostupnika _____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, _____, _____ godine

IZJAVA
o korištenju autorskog djela

Ja, _____ dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom

_____

_____

_____ 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, _____ (datum)

Potpis

_____

SADRŽAJ

1. Uvod

Prenošenje i pohranjivanje informacija vrlo je bitno za čovječanstvo. Još u vrijeme kada ljudi nisu znali čitati niti pisati, svoje su znanje, iskustvo i poruke prenosili usmenom predajom. Iako je usmena predaja učinkovita te se koristi i danas, nije dovoljna da sačuva cjelokupno ljudsko znanje. Taj problem riješio se izumom pisma, koja su se razvila u Egiptu, i to s ciljem zapisivanja i pohranjivanja ekonomskih, književnih, religijskih i političkih informacija. Time se može reći da je započela povijest baza podataka iako ne onakvu kakvu danas poznajemo.

S vremenom, potreba za pohranjivanjem podataka i obradom istih se povećavala. Ručno zapisivanje i bilježenje poslovnih događaja, bilo je jasno, postaje preteško i ne udovoljava brzini poslovanja. Pretraživanje pojedinih podataka je bilo odviše sporo, a ukoliko bi postojao specifični upit po kojem bi trebalo pronaći određenu skupinu podataka još i više. Razvoj računala, a ubrzo i veliki broj popratnih aplikacija, zahvatilo je tržište. Prva su računala bila namijenjena samo za računanje i razvijanje algoritama, no njegovim razvitkom ponajviše brzine i fleksibilnosti, ubrzo su zahvatila i poslovni svijet. Kako su podaci velikim organizacijama izuzetno važni, izumila se zbirka informacija koja se može organizirati – baza podataka.

70-tih godina prošlog stoljeća, Edgar Codd napisao je niz radova kojim opisuje načine za izgradnju baze podataka. Njegove su se ideje razvile u rad pod nazivom Relacijski model podataka za velike zajedničke banke podataka (*A Relational Model of Data for Large Shared Data Banks).*

Relacijski model podataka postao je standard skupa sa SQL jezikom. Ovakve baze podataka koriste se već dugi niz godina, pa čak još i danas, ali sve teže održavaju korak s novim tehnologijama. Tu se konkretno misli na NoSQL baze podataka. NoSQL baze podataka nastale su iz razloga da nadomjeste nedostatke i ograničenja relacijskih baza podataka. Postoji nekoliko vrsta modela koja sadrže vlastite implementacije sustava za upravljanje bazama podataka. Jedan od njih je i softver

otvorenog koda MongoDB koji je poznat po svojim karakteristikama kao što su jednostavnost korištenja, fleksibilnost i dostupnost.

U ovome radu objasnit će se relacijski model podataka, koje su njegove prepoznatljivosti i što je dovelo do toga da stručnjaci pronađu novi način pohranjivanja podataka. Također, osvrnut će se na SQL osnove. U trećem poglavlju govorit će se o sve popularnijem modelu baze podataka – NoSQL-u. Objasnit će se njegove karakteristike i vrste modela koje postoje. U četvrtom poglavlju prikazat će se jedna od NoSQL tehnologija - MongoDB baza podataka.

U radu će biti napravljena MongoDB baza podataka za iznajmljivanje brodova koja prikazuje način izrade, upita i održavanja baze podataka u MongoDBu. Na kraju će se usporediti prednosti i mane MongoDBa s relacijskom bazom podataka te će se izvesti zaključak.

2. Relacijske Baze Podataka

Relacijski model baza podataka iznio je 70-tih godina, u svojim radovima, Edgar F. Codd. Ovaj model temelji se na relacijskoj algebri koju je Codd, također, iznio u svojim radovima. Relacijska algebra podrazumijeva definiranje tablica nad operacijama. Rezultat odabranih operacija je relacija. Neke od najpoznatijih operacija su: Unija, Razlika, Kartezijev Produkt, Presjek, Selekcija, Projekcija,... Na osnovu relacijske algebre nastao je jezik za upite (*query language*) u relacijskim bazama podataka.

Sljedeća potpoglavlja osvrnuti će se na pojedine teorijske osnove o relacijskim bazama podataka, Coddova pravila, Normalizaciju, SQL jezik te prednostima i manama ovog modela.

2.2. Pojmovi relacijskog modela

Temelj relacijskog modela je, kao što mu i ime navodi, relacija. Relacija se može promatrati kao tablica u kojoj se prikupljaju i pohranjuju informacije o nekom entitetu. Svaka relacija/entitet ima svoje jedinstveno ime po kojem je prepoznatljiva i razlikuje se od ostalih relacija u istoj bazi podataka. Entitet može biti bilo što, ali njegove se pojave moraju međusobno razlikovati. Primjerice, Knjige jesu entitet jer svaku knjigu možemo razlikovati njenim naslovom, brojem stranica, žanrom, itd., dok Oblak ne možemo svrstati u entitet. Nadalje, svaki entitet sadrži atribute koji opisuju njegova svojstva. Atributi kvalificiraju, identificiraju i izražavaju stanje pojedinog entiteta. On mora imati svoje jedinstveno ime kojim ga razlikujemo od ostalih u relaciji. Atributi sadrže vrijednosti iste vrste, tj. dodijeljen mu je skup dopuštenih vrijednosti, koji se naziva domena atributa. Svaki atribut može poprimiti jednu vrijednost [10].
U relaciji obično postoji jedan atribut, koji jedinstveno opisuje entitet u kojem se nalazi, te se on naziva ključem. Ako postoji više kandidata za ključ, odabire se jedan, te ga se proglašava primarnim ključem. On služi za identifikaciju entiteta i povezivanje s drugim entitetima tj. tablicama. Vrijednost primarnog ključa ne smije niti u jednoj n-torki ostati neupisana.

Nakon što su se odredili entiteti i atributi te pronašli primarni ključevi za svaku tablicu potrebno je pronaći, dakako, ako postoje, njihove međusobne veze. Postoje više vrsta veza, a neke od najčešćih su sljedeće:

- Veza jedan – prema – jedan (1:1) – jedan entitet može biti u vezi s najviše jednim entitetom drugog tipa i obratno.
- Veza jedan – prema – više (1: M) – jedan entitet može biti u vezi s 0,1 ili više entiteta drugog tipa, ali entitet drugog tipa može biti u vezi s najviše jednim entitetom prvog tipa.
- Veza više – prema – više (M:N) – jedan entitet prvog tipa može biti u vezi s 0, 1 ili više entiteta drugog tipa i obratno.

Veza jedan – prema – jedan i jedan – prema – više najčešće se ostvaraju tako da se doda atribut koji je primarni ključ povezanom entitetu. Tako nadodan primarni ključ naziva se stranim ključem. U slučajevima kada postoji veza više – prema – više, stvara se nova relacija koja u sebi, najčešće, sadrži samo primarne ključeve.

2.3. Coddova pravila

Coddova pravila podrazumijevaju skup od 13 pravila za sustave koji upravljaju bazom podataka, koju je razvio, već prije spomenuti, E. F. Codd. Codd je definirao ova pravila kako bi izbjegao nesuglasice oko toga koji sustav za upravljanje bazom podataka je, a koji nije relacijski. Pravila predstavljaju idealnu relacijsku bazu podataka te bi se trebala koristiti kao osnova za definiciju relacijskog sustava za upravljanje bazom podataka. Iako u početku pravila nisu bila popularna u komercijalnom korištenju, kasniji DBMS-ovi temelje se na njegovim pravilima.
Od 13 definiranih pravila 6 ih mora biti ispunjeno da bi se DBMS zvao relacijskim [8],[4].

0. Nulto pravilo: Temeljno pravilo (eng. *Foundation rule*) – bilo koji DBMS koji se smatra relacijskim, mora upravljati bazom podataka na potpuno relacijski način i relacijskom metodom.
1. Predstavljanje informacija (eng. *The Information rule*) – sve informacije u bazi podataka moraju biti predstavljene isključivo vrijednostima u tablicama (relacijama).

2. Obavezna dostupnost (eng. *Guaranteed access rule*) – svi podaci trebaju biti logički dostupni kroz kombinaciju naziva tablice (relacije), vrijednostima primarnog ključa i naziva stupca (atributa).
3. Sustavno tretirana *null* vrijednost (eng. *Systematic treatment of null values*) – DBMS mora podržavati nulte vrijednosti kako bi se mogle predstaviti informacije koje nedostaju ili su neprimjenjive neovisno o tipu podataka.
4. Dinamički online katalog (eng. *Dynamic online catalog based on the relational model*) – struktura baze podataka opisana je u bazi podataka kao i obični podaci. Autorizirani korisnici mogu koristiti jezik za upite nad tim podacima.
5. Sveobuhvatni jezik za manipulaciju podacima (eng. *Comprehensive data sublanguage rule*) – baza podataka mora podržavati barem jedan jezik čiji se izrazi pomoću definirane sintakse mogu prepoznati kao nizovi znakova i koji će podržavati: *Database Description, Data Definition Language, Data Manipulation Language, Data Query Language*, Ograničenje integriteta, Autorizaciju, Transakcijske zahtjeve.
6. Ažuriranje pogleda (eng. *View updating rule*) – prezentacija podataka može se obaviti pomoću različitih logičkih kombinacija koji se nazivaju *Views* (pogledi). Svi pogledi koji se po relacijskoj teoriji mogu ažurirati, također se moraju moći ažurirati i u implementiranom modelu.
7. Visoka razina unosa, izmjene, brisanja (eng. *High-level insert, update, delete*) – sustav mora podržavati istodobni unos, izmjenu i brisanje.
8. Fizička nezavisnost podataka (eng. *Physical data independence*) – aktivnosti ili aplikacije koje korisnik koristi prema bazi moraju biti neovisne o metodi, strukturi i načinu spremanja.
9. Logička neovisnost podataka (eng. *Logical data independence*) – promjene na logičkoj razini ne smiju utjecati niti zahtijevati promjenu na aplikacijskom programu.
10. Neovisnost integriteta (eng. *Integrity independence*) – ograničenja integriteta moraju biti definirana i odvojena od aplikacijskog programa. Mijenjanje ograničenja treba biti dopušteno bez utjecaja na aplikacije.
11. Neovisnost distribucije (eng. *Distribution independence*) – bez obzira na to, podržava li sustav distribuciju, jezik sustava mora biti takav da distribuciju podržava bez utjecaja na aplikacijske programe.

12. Pravilo o nesubverzivnosti (eng. *Nonsubversion rule*) – ako sustav podržava jezik niske razine, taj jezik ne smije biti korišten za zaobilaženje pravila o integritetu.

2.4. Normalizacija

Normalizacija baze podataka je tehnika kojom organiziramo podatke unutar baze. Primarna relacijska shema može sadržavati nepravilnosti kao npr. redundancije (ponavljajući podaci) ili anomalije kod *Inserta*, *Updatea* i *Deletea* koje je potrebno otkloniti prije nego što se krene fizički oblikovati baza podataka.
Normalizacija smanjuje veličinu tablica na manje i čitljivije te ih povezuje odgovarajućim vezama. Postoji šest normalnih formi, a najčešće se koriste prve tri. Svaka normalna forma sadrži određene zahtjeve koje je potrebno ispuniti da bi se moglo prijeći u sljedeću normalnu formu.

U nastavku će biti opisane normalne forme s popratnim primjerima. Prema [15], [19].

2.4.1. Prva Normalna forma

Relacija se nalazi u Prvoj normalnoj formi, ako relacija sadrži vrijednosti atributa koje su jednostruke i jednostavne. Svaka relacija mora imati ključ koji ga jedinstveno identificira.

Promatrajući tablicu 1 - Zaposlenik, njene atribute i vrijednosti može se primijetiti ako se ona ne nalazi u Prvoj normalnoj Formi. Naime, sve vrijednosti nisu jednostruke i jednostavne. Da bi relacija bila u Prvoj normalnoj formi potrebno je vrijednosti atributa svesti na atomske. Tako će se ponoviti informacije o Zaposleniku koji ima dva Broja_telefona (tablica 2).

Tablica 1 - Zaposlenik

ID_zaposlenika	Ime_zaposlenika	Mjesto stanovanja	Broj telefona
111	Marin	Pula	098574123
432	Romanno	Pazin	091456351
154	Drago	Šegotići	099635874
698	Jasmin	Labin	095741589 099423212

Tablica 2 - Zaposlenik

ID_zaposlenika	Ime_zaposlenika	Mjesto stanovanja	Broj telefona
111	Marin	Pula	098574123
432	Romanno	Pazin	091456351
154	Drago	Šegotići	099635874
698	Jasmin	Labin	095741589
698	Jasmin	Labin	099423212

2.4.2. Druga Normalna forma

Relacija se nalazi u Drugoj normalnoj formi ako se nalazi u Prvoj normalnoj formi i ako je svaki njezin neključni atribut potpuno ovisan o primarnom ključu tj. ne sadrži parcijalne ovisnosti atributa o primarnom ključu.

U tablici 3. Pregled_predmeta, vidljiva je redundancija u atributu Naziv_predmeta koji se pojavljuje uz studenta koji je taj predmet položio.

Tablica 3-Pregled_predmeta

JMBAG	Id_predmeta	Naziv_predmeta	Ocijena
4587412	A1	Baze podataka	4
9614731	A2	Računalne mreže	3
4587412	A2	Računalne mreže	4

Atributi JMBAG, *Id*_predmeta funkcionalno su zavisni o atributu Ocjena, dok je atribut Ocjena potpuno funkcionalno zavisan o atributima JMBAG i *Id*_predmeta. Time se dolazi do zaključka da je Naziv_predmeta nezavisan o atributu Ocjena i JMBAG jer je funkcionalno zavisan jedino od *Id*_predmeta. Druga normalna forma izvodi se tako da se uz primarnu relaciju dodaje onoliko novih relacija koliko ima atributa koji sudjeluju u parcijalnoj ovisnosti.
Ti se atributi prebacuju u nove relacije dokle god postoji parcijalna ovisnost.

Tako je tablica 3. Pregled_predmeta, „razlomljena“ na još jednu tablicu te je dobivena relacija Predmet. Time su svi atributi funkcionalno zavisni o svom primarnom ključu.

Tablica 4-Prelged_predmeta

JMBAG	Id_predmeta	Ocjena
4587412	A1	4
9614731	A2	3
4587412	A2	4

Tablica 5-Predmet

Id_predmeta	Naziv predmeta
A1	Baze podataka
A2	Računalne mreže

2.4.3. Treća Normalna forma

Relacija je u Trećoj normalnoj formi, ako je u Prvoj normalnoj formi i Drugoj normalnoj formi i ako ne postoje tranzitivne ovisnosti.

Tablica 6-Zaposlenik

Id_zaposlenika	Ime_zaposlenika	Mjesto stanovanja	Poštanski broj	Godina_rođenja
111	Marin	Pula	52100	1980
432	Romanno	Pazin	52000	1966
154	Drago	Krnica	52208	1960
698	Jasmin	Pula	52100	1977
956	Goran	Pula	52100	1991

U tablici 6. Zaposlenik, mogu se vidjeti ponavljanja podataka atributa Mjesto stanovanja i Poštanski broj. Mjesto stanovanja tranzitivno je ovisan o atributu *Id*_zaposlenika. Da bi relacija bila u Trećoj normalnoj formi, potrebno je polaznu relaciju rastaviti na dvije ili više relacija. U prvoj relaciji ostavlja se primarni ključ i atributi koji nisu tranzitivno zavisni. U ovom slučaju to su atributi: *Id*_zaposlenika, Ime_zaposlenika, Poštanski broj, Godina_rođenja. U drugu se relaciju prebacuju tranzitivno ovisni atributi (tablica 7., tablica 8.).

Tablica 7-Zaposlenici

Id_zaposlenika	Ime_zaposlenika	Godina_rođenja	Poštanski broj
111	Marin	1980	52100
432	Romanno	1966	52000
154	Drago	1960	52208
698	Jasmin	1977	52100
956	Goran	1991	52100

Tablica 8-Poštanski broj

Poštanski broj	Mjesto stanovanja
52100	Pula
52000	Pazin
52208	Krnica
52220	Labin

2.4.4. Boyce-Coddova Normalna forma

Relacija je u Boyce-Coddovoj normalnoj formi (BCNF), ako se nalazi u Drugoj normalnoj formi i Trećoj normalnoj formi te ako su njezine determinante kandidati za ključ.

Tablica 9 - Prijava_pogon

Id_zaposlenika	Id_pogona	Naziv_pogona	Prijavljen
111	12	Mala Presa	05:50
212	12	Mala Presa	06:00
541	31	Teška Presa	05:54
871	95	Dorada	06:20

U tablici 9, Prijava_pogon, imamo sljedeće ovisnosti: *Id*_zaposlenika funkcionalno je ovisan o *Id*_pogona i Prijavljen. I*d*_pogona funkcionalno je ovisan o Naziv_pogona. Iz toga proizlaze dvije determinante koje su ujedno i kandidati za ključ: *Id*_zaposlenika i *Id*_pogona. Prelazak relacije u Boyce-Coddovu normalnu formu radi se rastavljanjem polazne relacije na dvije ili više relacija, pri čemu se iz polazne

relacije premještaju atributi s determinantama koji nisu kandidati za ključ. Prevedena relacija koja se nalazi u Boyce-Coddovoj normalnoj formi vidljiva je u tablici 10. i tablici 11.

Tablica 10- Pogon

Id_pogona	Naziv_pogona
12	Mala Presa
31	Teška Presa
95	Dorada

Tablica 11-Prijava_pogon

Id_zaposlenika	Id_pogona	Prijavljen
111	12	05:50
212	12	06:00
541	31	05:54
871	95	06:20

Potrebno je naglasiti da postoje relacije koje se nalaze u 3NF ali nisu u BCNF.

2.4.5. Četvrta Normalna forma

Postoji mogućnost da relacija sadrži višeznačnu ovisnost koja uzrokuje redundanciju. Kako redundancija nije uzrokovana funkcionalnim ovisnostima onda možemo reći da je polazna relacija u Boyce-Coddovoj normalnoj formi.

Polazna relacija uvijek ima tri atributa a postupak prelaska u Četvrtu normalnu formu izvodi se tako da se polazna relacija pretvara u dvije manje relacije tako da se u njima više ne nalaze višeznačne ovisnosti.

2.4.6. Peta Normalna forma

Relacija se nalazi u Petoj normalnoj formi ako ona ne sadrži zavisnost spoja, tj. ako se relacija više ne može pouzdano dekomponirati.

2.5. SQL jezik

SQL (*Structured Query Language*) strukturirani je upitni jezik koji se koristi za upravljanje i manipulacijom relacijskih baza podataka. SQL programski jezik za relacijske baze podataka nastao je sredinom 70-tih godina prošlog stoljeća. Ovaj

jezik se, kao i drugi programski jezici, godinama usavršavao te je 1986. godine usvojen službeni SQL standard ISO-ANSI. Kako ovaj standard nije konačan, od njegova osnutka pa do danas, usavršen je nekoliko puta s dodatnim funkcijama. Zadnja nadogradnja bila je 2011. godine te je nazvan inačica SQL:2011.

SQL naredbe podijeljene su u nekoliko vrsta, među kojima su: **DML** (*Data Manipulation Data*) tj. jezik na manipulaciju podacima**, DDL** *(Data Definition Language*) jezik za definiciju podataka**, TCL** (*Transaction Control Language*) jezik za upravljanje tijekom transakcija, **DCL** (*Data Control Language*) jezik za upravljanje podacima.

DDL, kao što mu i samo ime govori, služi za stvaranje i definiranje tablica, pogleda (*viewova*), shema itd. DDL se sastoji od tri osnovne naredbe:

- *CREATE* – kreiranje strukture tablica i atributa i veza između relacija
- *ALTER* – mijenjanje strukture
- *DROP* – brisanje strukture
- *TRUNCATE* – brisanje svih redaka iz tablice
- *RENAME* – preimenovanje postojećeg objekta u bazi podataka

DML služi za manipulaciju podacima. Koristi se za ažuriranje, brisanje, dodavanje podataka u tablicu. Osnovne naredbe su:

- *SELECT* – dohvaćanje podataka iz baze podataka
- *INSERT* – dodavanje novog retka u bazu podataka
- *UPDATE* – ažuriranje podataka
- *DELETE* – brisanje podataka iz baze podataka

TCL je jezik za upravljanje podacima tijekom transakcija. Njegove osnovne naredbe su:

- *COMMIT* – završava transakciju
- *ROLLBACK* – poništava transakciju
- *SAVEPOINT* – postavlja sigurnosnu točku u transakciji
- *SET TRANSACTION* – specificira detalje/karakteristike transakcije

DCL jezik služi za manipuliranje pristupima i ovlastima. Postoje dvije naredbe:

- *GRANT* – daje korisniku privilegiju pristupa za točno određene dijelove baze
- *REVOKE* – oduzima korisniku dane privilegije

Na slici 1, pomoću naredbe *INSERT INTO* upisani su podaci u relaciju zaposlenik. Podaci koji će biti pohranjeni u tablicu upisuju se nakon naredbe *VALUES*. Kod promijene prezimena na slici 2, mijenjaju se atributi tako da se prvo koristi ključna riječ *Update* nakon koje ide ime relacije te ključna riječ *Set* u kojoj se određuje atribut s novim vrijednostima. Ako postoji više atributa međusobno se odvajaju zarezima.

Upiši novog zaposlenika Mirka Mirića čiji je OIB 87896544123 i koji je rođen 1966. godine.

```
INSERT INTO zaposlenik (ime.prezime. oib. godina_rođ)
VALUES ( 'Mirko'. 'Mirić'. 87896544123. 1966):
```

Slika 1. Umetanje novog zapisa u relaciju u SQL-u , izvor: vlastiti izvor

Promijeni prezime zaposlenika Jadranke Jović u Jandroković čiji je OIB 741854123.

```
UPDATE zaposlenik SET prezime='Jandroković'
WHERE oib= 741854123:
```

Slika 2. Ažuriranje relacije u SQL-u, izvor: vlastiti izvor

Na slici 3. se koristila naredba *SELECT* koja je u ovom slučaju napravila spajanje dviju tablica te je iz dobivene tablice izdvojila tražene podatke. Nakon naredbe *SELECT* mora se specificirati naziv tablice ispred imena atributa (student.jmbag). Ako se to ne napravi sustav bi javio grešku jer ne može identificirati atribut. Greška bi bila javljena iz razloga jer postoji mogućnost da više tablica u bazi podataka sadrži iste nazive atributa. Naredba *FROM* služi da bazi podataka prikaže iz kojih tablica će povlačiti podatke. Može se koristiti naredba *WHERE* koja ograničava prikaz rezultata, kako je i učinjeno na primjeru koji prikazuje slika 2., gdje je ograničeno da se

prikazuju rezultati samo onih studenata koji su upisali traženi kolegij. Također, mogu se koristiti i daljnje naredbe kao što su: *ORDER BY* i *LIMIT* koji ograničavaju broj rezultata koji će biti izlistan u tablici.

Ispiši imena, prezimena i JMBAG studenata koji su upisali predmet čiji je id=4512.

```
SELECT student.jmbag , student.ime, student.prezime
FROM student, upisao
WHERE student.jmbag=upisao.jmbag AND upisao.id_predmeta=4512
ORDER BY student.prezime;
```

Slika 3. Pretraživanje relacija u SQL-u, izvor: vlastiti izvor

2.6. Prednosti i mane relacijskog modela

Jedna od mnogih prednosti, koje relacijske baze podataka nude, definitivno je jednostavnost. Relacijski model strukturira podatke na jednostavan način izbjegavajući složenost. Struktura tablica intuitivna je i lako prihvatljiva većini korisnika. Podaci su organizirani tako da pojednostavljuju razvoj i korištenje baze podataka. Ako se želi pristupiti podacima u relacijskoj bazi podataka to se čini putem posebnih funkcija i naredbi. Nije potrebno prelaziti kroz stablo ili hijerarhiju kao kod pojedinih baza podataka.

SQL je jezik za upite četvrte generacije te dopušta korisnicima da navedu što žele bez navođenja kako to mora biti učinjeno. Jednostavnost pronalaženja podataka vidljiva je i u filtriranju podataka na temelju sadržaja, broju stupaca ili samo jednog traženog podatka. Time je korisnicima omogućeno da se kroz veliki broj podataka izlista i prikažu samo relevantni podaci. Ono što je svakako bitno napomenuti jest da relacijska baza podataka sadrži brojne provjere valjanosti pri upisivanju podataka čime se omogućuje da svi podaci budu prisutni i da se nalaze unutar zadanog raspona. Integritet podataka pomaže pri osiguravanju točnosti i dosljednosti podataka pohranjenih u bazi. Model relacijske baze podataka je skalabilan i promjenjiv čime pruža fleksibilnost kod zahtjeva za promjenom i sve većom količinom podataka. Analitičar baze podataka može jednostavno dodavati, mijenjati i uklanjati

tablice i stupce bez da to utječe na bazu. U teoriji nema ograničenja u broju redaka, stupaca i tablica. Praksa je ipak nešto drugo. Rast i promjena baze podataka ograničena je DBMS-om i Hardverom. Još jedna od prednosti relacijskih baza podataka je metodologija koja osigurava da podaci nemaju anomalije koje mogu utjecati na integritet podataka, a to osigurava normalizacija baze podataka [20].

Iako relacijska baza podataka ima dosta pozitivnih strana, ona također ima i neke nedostatke. Jedna od takvih nedostataka je skupoća postavljanja, dizajniranja i održavanja baze podataka. Da bi se sastavila baza podataka potrebno je kupiti poseban softver što može biti izuzetno skupo ako se radi o velikoj bazi podataka. Unos podataka, kreiranje korisničkih uloga, dodjela prava korisnicima iziskuje mnogo vremena i kvalitetnog dizajnera baze podataka. Iako je navedeno da relacijska baza podataka sadrži jednostavan dizajn i jednostavna je za upotrebu može se slobodno reći da je to „dvosjekli mač“. Ta jednostavnost dizajna i upotrebe može dovesti do razvoja i implementacije vrlo slabo dizajniranog sustava za upravljanja bazom podataka i velikih problema u budućnosti.

Sljedeći nedostatak je složenost informacija. U današnje vrijeme informacije su sve veće i različitije a relacijske baze podataka izrađuju se prema nekim zajedničkim karakteristikama. Složene slike, brojevi, multimedijski proizvodi prkose jednostavnoj kategorizaciji koja vodi u neku novu vrstu baze podataka [11].
Nadalje, složeni upiti zahtijevaju sofisticiraniju obradnu moć. Iako većina stolnih računala može upravljati bazama podataka veličine i složenosti nekog malog poduzeća, baza podataka s vanjskim izvorima podataka, složenim strukturama podataka koje odgovaraju većim organizacijama zahtijeva snažnije poslužitelje i profesionalniji softver da vrati rezultate u prihvatljivom vremenu [6].

3. NoSQL

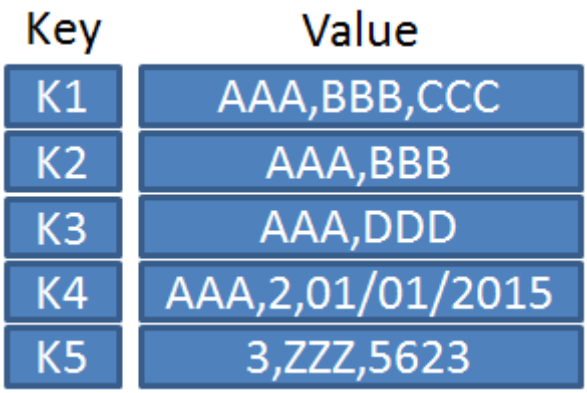
Napredovanjem *web* tehnologija, količina podataka o korisnicima, proizvodima, događajima itd. drastično je porasla. U središtu skoro svakog poslovanja su *web* aplikacije i mobilne aplikacije koje zahtijevaju da se podrži veliki broj istodobnih korisnika, da su uvijek dostupne, da rukuju polustrukturiranim i nestrukturiranim podacima te brzu prilagodbu za novim zahtjevima, ažuriranjima i novim značajkama. Da bi se ove suvremene aplikacije razvile i konkurirale na i ovako već zahtjevnom tržištu, od programera se očekuje da svoje aplikacije i usluge isporuče što brže. Ovakav način razvoja naziva se agilnim. Ovakve zahtjeve relacijski model baze podataka ne može ispuniti.

U prethodnom poglavlju ustanovljeno je, da je njen model podataka fiksan i definiran shemom koja se ne mijenja. Ako bi netko i mijenjao shemu to bi značilo usporavanje ili čak i zaustavljanje razvoja, ne samo iz razloga jer je to dugotrajan, ručan i skup proces već i jer utječe na druge aplikacije. Iz tih su se razloga velike organizacije poput Googlea i Amazona okrenule alternativnom rješenju koje će zadovoljiti potrebu za fleksibilnošću i agilnošću – NoSQL bazi podataka [26].

Kada kažemo NoSQL onda mislimo na nerelacijsku bazu podataka i bez upotrebljavanja SQL jezika. Ovakva baza podataka omogućuje efikasnu i brzu obradu podataka. Stoga, postoji i više vrsta NoSQL tehnologija, a u ovom poglavlju spomenuti ćemo njih četiri:

- **Ključ/Vrijednost baza podataka** – ova baza podataka zasniva se na jednostavnom modelu koji uključuje, kako mu i naziv kaže, ključ s pridruženom vrijednošću. Baza podataka ključne vrijednosti pohranjuje kao zbirku parova ključ: vrijednost, u kojima ključ funkcionira kao jedinstveni identifikator. Ključ i vrijednost mogu poprimiti razne formate poput internetskih adresa, jedinstven kod, slike, zvuk, itd. Ovakav model je po strukturi sličan rječniku i hash tablici. Ključ/Vrijednost model nema jezik za upite kao RDBMS. Koriste se jednostavne operacije kao što su dodavanje (*PUT*), brisanje (*DELETE*), dohvaćanje (*GET*). Upiti za pretraživanja rade se ručno na razini aplikacije.

Dva bitna pravila koja se odnose na ovaj model jest da svaki ključ mora biti jedinstven i da upit može biti baziran samo na ključu [22]. Postoji nekoliko baza podataka koja se temelje na ključ/vrijednost modelu. Neki od najpoznatijih su: Riak, Redis, Aerospike, Berkeley DB.



Slika 4. Ključ/Vrijednost baza podataka, izvor: https://en.wikipedia.org/wiki/Keyvalue_database#/media/File:KeyValue.PNG

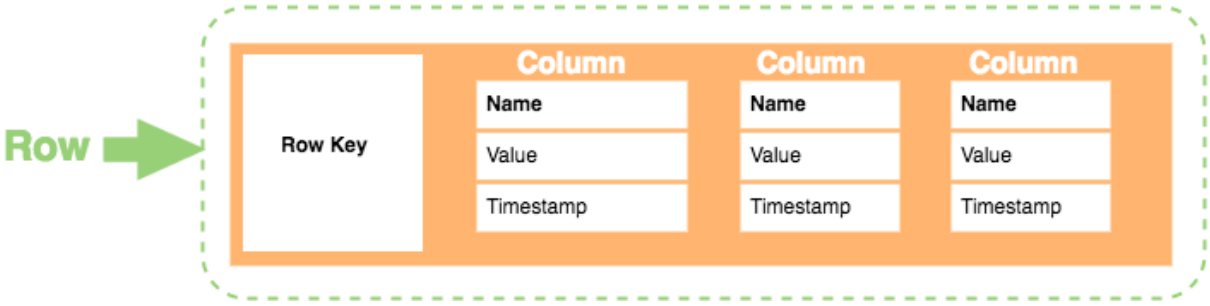
- **Graf baze podataka** – ovaj model NoSQL baze podataka pohranjuje podatke kao čvorove koji bi u relacijskoj bazi podataka predstavljali redak u tablici, dok lukovi povezuju čvorove i predstavljaju odnose među njima. Za razliku od relacijskog modela koji ima statičnu shemu, graf model se može razvijati tijekom vremena. Ovaj model se koristi u sustavima koji trebaju mapirati odnose kao npr. Sustavi za rezerviranje ili upravljanja odnosa s kupcima[19]. Također, ne postoji standardni upitni jezik nad graf modelom ali oni postoje. Jedan takav primjer je upitni jezik Cypher koji se koristi za graf bazu Neo4j. Na slici 5 može se vidjeti primjer upita u jeziku Cypher. Sljedeći upit vraća *Usera* (korisnika) koji ima *id*= 112233. Osim prije spomenute baze, postoji još nekoliko primjera : AllegroGraph, IBM Graph, Titan..

```
MATCH (korisnik)
WHERE korisnik.id = 112233
RETURN korisnik
```

Slika 5. Upit u Cypheru,izvor: vlastiti izvor

- **Stupčaste baze podataka** – ovaj model podataka organizira podatke u stupce umjesto u redove. Stupčani model se može pronaći i u SQL i u NoSQL bazi podataka. Postoji nekoliko vrsta implementacija koje koriste stupčaste baze podataka i međusobno se razlikuju npr: Google Big Table, Hbase i Casssandra.

  Sljedeći redovi objasnit će karakteristike Cassandre. Ovaj sustav sadrži četiri osnovne komponente: stupac, super-stupac, obitelj stupaca i obitelj super stupaca. Svaki stupac sadrži naziv, vrijednost i vremensku oznaku te su oni osnovne jedinice pohrane. Stupci koji su ugniježđeni nazivaju se super stupci. Stupci koji se zajedno često upotrebljavaju nazivaju se obitelj stupaca (slika 6.). Kod stvaranja obitelji stupaca potrebno je definirati koje stupci će biti unutra te je također potrebno zadati tip podataka za svaki od njih.



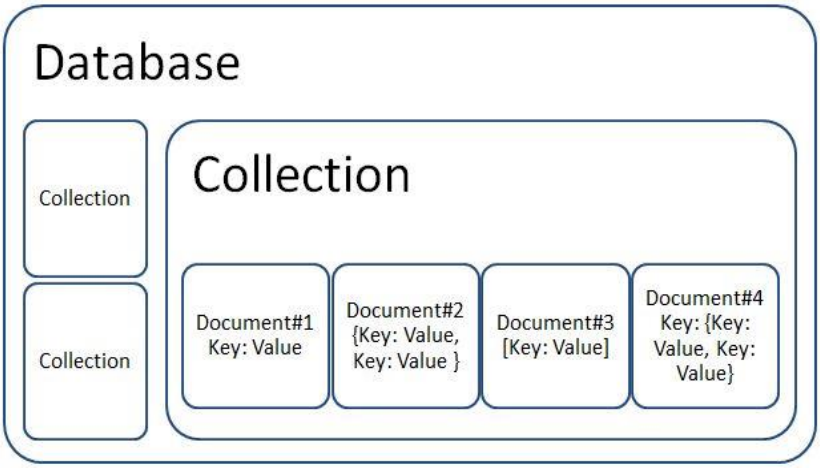
Slika 6. Prikaz Obitelji-stupaca, izvor: https://database.guide/what-is-a-column-store-database/

Svaka obitelj mora biti definirana unutar *keyspacea* (ključa) koji se sastoji od retka i stupca te grupira obitelji prema aplikacijama koje ih koriste. Prednost stupčaste baze podataka jest u tome da može obraditi veliku količinu podataka pa se u većini slučajeva koristi za kataloge, otkrivanje prijevara i *batch* obradu.

- **Dokument baza podataka** – također jedna od vrsta nerelacijskih baza podataka koja je dizajnirana za pohranjivanje polustrukturiranih podataka kao dokumente. Dokument baza podataka jedna su od najpopularnijih vrsta NoSQL-a radi svoje intuitivnosti. Ona omogućava programerima kreiranje i ažuriranje programa bez potrebe za referenciranjem glavne sheme. Veću popularnost korištenja dokument baza podataka postigao se korištenjem

JavaScripta i Java Notation Objecta (JSON), pogotovo među programerima web aplikacija.

Osim prethodno spomenutih može se koristiti i XML. Osnovni element ovog modela je, kao što mu i ime kaže, dokument: uređeni skup ključeva s pridruženom vrijednošću. Svaki dokument može imati istu ili različitu strukturu podataka, a svaki dokument je sam za sebe i ne mora nužno ovisiti o bilo kojem drugom dokumentu. Drugim riječima, dokument baze podataka imaju dinamičku shemu za razliku od relacijske čime se postiže fleksibilnost. Dokumenti se smještaju u kolekcije koje se mogu poistovjetiti tablicama u relacijskom modelu podataka. Za svaku kolekciju definira se tip dokumenta koji „čuva“. Tako npr. jedna kolekcija sadrži proizvode a druga kupce.



Slika 7. Primjer modela dokument baze podataka, izvor: https://techietweak.wordpress.com/2016/01/09/no-sql/

Aplikacija sama vodi brigu o tome koji podatak se sprema u koju kolekciju. Dokument baze podataka se u većini slučajeva koriste za upravljanje sadržajem, Internet trgovini te izradi *web* i mobilnih aplikacija iz razloga jer su one podvrgnute stalnom mijenjanju te je također brzina implementacije vrlo važna.

Primjeri implementacija koje koriste ovaj model su: CouchDB, Couchbase Server, DocumentDB, MarkLogic i MongoDB.

4. MongoDB

MongoDB je sustav za upravljanje baze podataka otvorenog koda koji koristi model baze podataka usmjeren na dokument. U prethodnom se poglavlju spominjalo da MongoDB spada u jednu od brojnih NoSQL tehnologija koje služe za upravljanje velikom količinom podataka te da se ne uklapaju u tradicionalni relacijski model.

U sljedećih nekoliko potpoglavlja detaljnije će se objasniti MongoDB, njegova implementacija te će se na kraju usporediti sa relacijskom bazom podataka.

4.1. Povijest

MongoDB stvoren je od strane Dwighta Merrimana i Eliota Horwitza kada su zapeli u problemu s razvojem i skalabilnosti s tradicionalnim relacijskim modelom pri izgradnji *web* aplikacije na DoubleClicku, online oglašavačkoj tvrtki koja je trenutno u vlasništvu Googlea. Ime baze potječe od riječi *humongous (*hr. ogroman*)*, a predstavlja ideju za obradu velike količine podataka. Ova dva programera formirali su tvrtku 2007. godine kako bi mogli komercijalizirati svoju ideju. Godine 2009. ovaj sustav za upravljanje bazom podataka objavljen je kao softver otvorenog koda i dostupan je pod uvjetima iz verzije 3.0 GNU Affero General Public License. Tvrtka je preimenovana 2013. godine u MongoDB Inc.

4.2. Karakteristike MongoDB-a

MongoDB koristi model podataka baziran na dokumentu jer je bolji i prirodniji način za opisivanje podataka. Dokumenti predstavljaju jednu zasebnu strukturu s pripadajućim podacima ugrađenim kao pod – dokumenti i nizovi. Time je dokumente moguće usporediti s objektima u objektno-orijentiranom programiranju. Stoga i ne čudi da je programerima jednostavnije i brže programirati kako će se podaci u aplikaciji pohranjivati u bazi podataka.

MongoDB pohranjuje podatke kao JSON (JavaScript Object Notation) dokumente u binarnom zapisu pod nazivom BSON (Binary JSON). Rad u BSON-u olakšava

aplikacijama obradu, sortiranje i usporedbu podataka. BSON dokumenti sadrže jedno ili više polja, a svako polje sadrži vrijednost određene vrste podataka koje uključuju polja, binarne podatke i pod-dokumente. MongoDB pruža upravljačke programe za sve popularne programske jezike i okvire za što lakši i prirodniji razvoj. Programski jezici koji su podržani od MongoDB-a su: Java, JavaScript, C# /.NET, Python, Perl, PHP, Scala, itd. MongoDB driveri su dizajnirani da budu specifični za dani jezik [13].

Postoji još nekoliko bitnih karakteristika MongoDB baze podataka [25]:

- Agregacija – operacije koje se vrše nad podacima i vraćaju izračunate rezultate. Postoje tri vrste izvršavanja agregacije podataka: Agregirajući cjevovod, Map-reduce i Agregirajuće funkcije.
- Indeksiranje – indeksi omogućuju visoke performanse izvršenja operacija kod najčešćih upita. Oni predstavljaju jednu vrstu strukture podataka koja omogućava brzo pronalaženje dokumenata na osnovu određenih vrijednosti koje se nalaze u poljima dokumenata.
- Replikacija – sinkronizacija podataka između većeg broja MongoDB instanci.
- Ad-hoc upiti – upiti koji se mogu obaviti nad bazom bez da je unaprijed definirana. Ad – hoc upit ne ostaje dugo u sustavu i dinamički se stvara na zahtjev korisnika.
- Particioniranje – raspodjela podataka na više servera/instanci radi bolje efikasnosti i povećanja kapaciteta sustava.
- Skalabilnost – MongoDB omogućava horizontalnu skalabilnost.

MongoDB je baza podataka koja ima različitu upotrebu. Postoji puno projekata i velikih organizacija koje koriste ovu bazu podataka. Njena dinamička shema i objektno orijentirana struktura čine je idealnim izborom za analitiku u realnom vremenu, mobilne aplikacije, e-trgovine, mobilne i društvene mreže, održavanje podataka na temelju lokacije tj. geoprostorni podaci i slično.

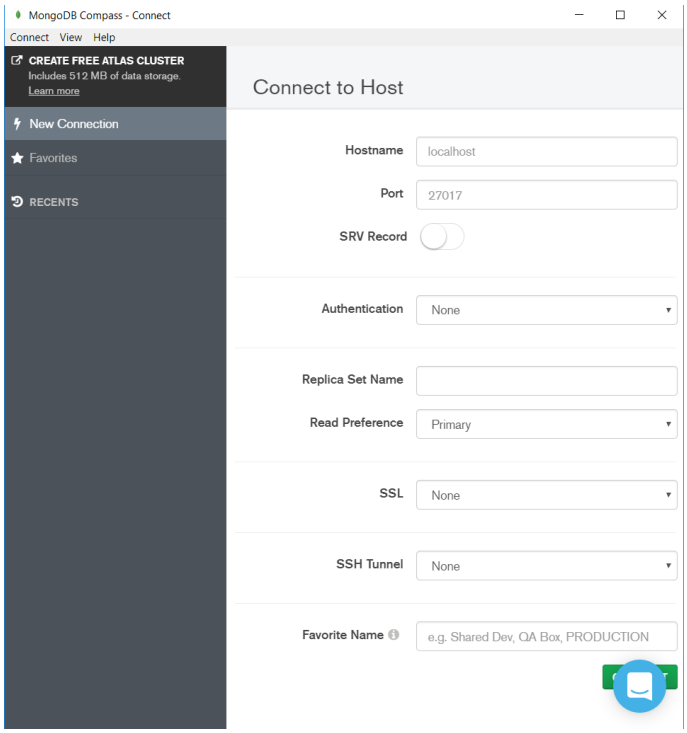
4.3. Implementacija

U ovom potpoglavlju bit će izrađena RentMeMedulin baza podataka koristeći MongoDB. Također, bit će korištena i baza podataka Northwind koja je preuzeta sa

GitHub-a.[1] U ovome radu korištena je verzija MongoDB 4.0. Neki dijelovi baze rađeni su u Mongo Shellu, ali je također, radi lakšeg manipuliranja i prikaza podataka instaliran i MongoDB Compass koji predstavlja korisničko grafičko sučelje.

4.3.1. Izrada Baze podataka

Da bi se napravila baza podataka RentMeMedulin, prvo je potrebno spojiti se na *poslužitelj*.
Slika 8. prikazuje MongoDB Compass i spajanje na *poslužitelj* s portom 27017.

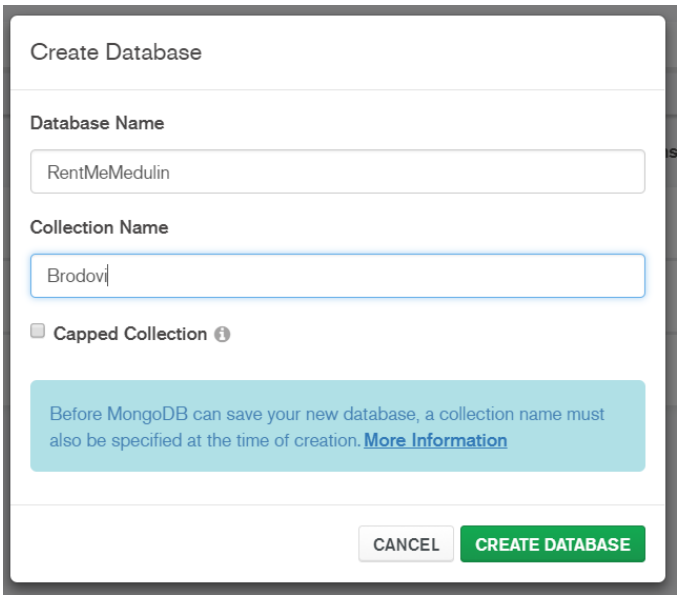


Slika 8. Spajanje na poslužitelj, izvor: vlastiti izvor

Nakon spajanja na *poslužitelj*, sljedeće potrebno je napraviti željenu bazu podataka. U Compassu je to vrlo jednostavno. Potrebno je samo kliknuti na *kreiraj bazu podataka* (*Create Database*). Otvorit će se zasebni prozor u kojem se traži da se navede ime baze i ime jedne kolekcije. U ovom slučaju, za ime je izabrano RentMeMedulin a za kolekciju Brodovi, jer će se za početak moći iznajmljivati samo brodovi. Također, kao što je vidljivo na slici, korisničko sučelje nudi i postavljanja

[1] GitHub: https://github.com/tmcnab/northwind-mongo

Capped Collectiona. To je kružna kolekcija s fiksnom veličinom koje slijede redoslijed umetanja radi postizanja visokih performansi kod operacija umetanja, ažuriranja i brisanja. Pod kružnim se misli, da kada se dostigne zadana veličina, počet će se brisati najstariji dokumenti u kolekciji bez ikakvih eksplicitnih naredbi. *Capped Collections* su najbolje za pohranjivanje *log* podacima, *cache* podacima ili drugim velikim podacima [15].
Nakon što su ispunjena sva potrebna polja potrebno je kliknuti na dugme *Create Database* i baza je napravljena (slika 9.).



Slika 9. Kreiranje baze u Compassu, izvor: vlastiti izvor

Ako se baza podataka radi u Mongo Shellu postupak ide tako da se napiše naredba ***use Database_name,*** odnosno u ovom slučaju ***use RentMeMedulin.*** Ova naredba će kreirati bazu podataka ako ona ne postoji, u suprotnom se spaja s imenovanom bazom. Nakon toga je potrebno napraviti kolekciju koja se kreira naredbom ***db.createCollection(),*** odnosno ***db.createCollection(Brodovi).***

Nakon što se napravila kolekcija Brodovi, potrebno je kreirati i dokumente koji se nalaze unutar te kolekcije. U Compassu je potrebno samo kliknuti na dugme *create document* i prikazat će se prozor za kreiranje dokumenta. Može se primijetiti kako je automatski dodan *_id*, odnosno jedinstveni identifikator. Ispod njega mogu se dodavati polja tako da im se odabere naziv, napiše vrijednost i odabere koja je

njegova struktura (*int32, double, date, string, object,arrray*, itd.). Nakon unesenog, potrebno je pritisnuti ***insert.***

Slika 10. Kreiranje dokumenta u MongoDB Compass,izvor: vlastiti izvor

U Mongo Shell-u potrebno je napisati naredbu ***db.collection.insert(naziv_dokumenta).*** Ova naredba također nudi dvije mogućnosti: ***db.collection.insertOne() i db.collection.insertMany()*** .

U prvom slučaju se dodaje jedan dokument u kolekciju, dok se u drugom može dodavati više dokumenata od jednom u istu kolekciju.

Na slici 11. možemo vidjeti dodavanje jednog dokumenta u kolekciju Vrsta_broda u Mongo Shellu. Može se primijetiti kako je *_id* napravljen proizvoljno. Ako ne bi bio napisan, MongoDB bi ga napravio automatski. Polja je potrebno dodavati unutar vitičastih zagrada, odvojene zarezom.

```
> db.Vrsta_broda.insertOne({_id: "kat", naziv: "Katamaran"})
{ "acknowledged" : true, "insertedId" : "kat" }
```

Slika 11. Kreiranje jednog dokumenta,izvor: vlastiti izvor

Ako postoje podaci koje je potrebno staviti u listu, onda ih se nakon imena polja stavlja unutar uglatih zagrada. Polja koja se nalaze unutar liste, moraju biti unutar posebnih vitičastih zagrada odvojena zarezom.

Na slici 12., vidljiv je primjer dodavanja više dokumenata u jednu kolekciju. U tom se primjeru može primijetiti jedna od karakteristika MongoDB-a, a to je dinamička

shema. Dokument čiji je *id*=pdp i nosi naziv Poludeplasman jedini sadrži polje opis. U relacijskim bazama podataka to nije moguće i baza bi javljala grešku. Za druge dokumente opis nije bio potreban jer njihov naziv govori sam za sebe, čime je omogućena štednja memorije i niti jedno prazno polje.

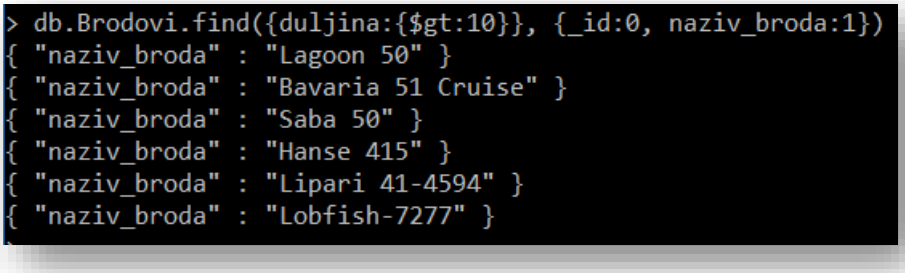
```
db.Vrsta_broda.insertMany([
{_id:"jed", naziv:"Jedrilica"},
{_id:"pdp", naziv:"Poludeplasman", opis:"Gliser s kabinom"},
{_id:"spdb", naziv:"Gliser"}
])
```

Slika 12. Kreiranje više dokumenata u jednu kolekciju, izvor: vlastiti izvor

4.3.2. Upiti

U Mongo Shellu za upite je potrebno napisati naredbu *find().* Tako bi naredba izgledala ***db.collection.find(query)***. Unutar zagrada pišemo polja koja želimo da budu pronađena. Ako ništa nije definirano, ispisat će se svi dokumenti u kolekciji. Nadalje, osim parametra unutar zagrada mogu se staviti određeni kriteriji poput: usporedbi (veće od, manje od), sortiranja (uzlazno, silazno), ograničavanja rezultata (određuje koliko će dokumenata biti prikazano), itd.

Slika 13. prikazuje primjer upita gdje se pretražuje kolekcija Brodovi po parametru duljina. Prikazat će se samo oni rezultati čija je duljina broda veća od ($gt = *greater than*) 10 metara. Sljedeći parametar _*id*:0, naziv_broda: 1 govori da će se u rezultatima prikazati samo naziv broda i niti jedno drugo polje.

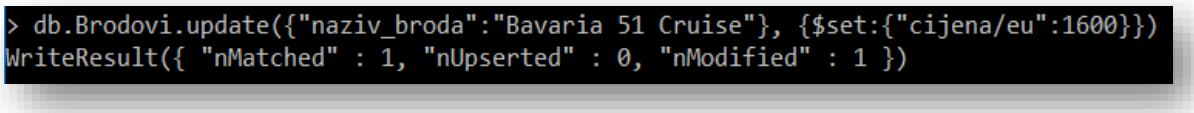


Slika 13. Upit u Mongo Shellu, izvor: vlastiti izvor

4.3.3. Ažuriranje i brisanje

Ažuriranje u MongoDB radi tako da se napiše naredba *update*, tj. ***db.collection.update().*** Ova naredba ažurirat će postojeći dokument u kolekciji s vrijednostima koje su postavljene unutar zagrada.

Na slici 14. nalazi se primjer ažuriranja kolekcije brodovi čije je ime broda Bavaria 51 Cruise. Naredbom *$set* promijenila se vrijednost parametra cijena/eu u 1600.



Slika 14. Ažuriranje dokumenta, izvor: vlastiti izvor

Kao i kod *inserta* tako i ovdje postoji mogućnost ažuriranja samo jednog dokumenta pomoću naredbe ***db.collection.updateOne()*** ili ažuriranja više dokumenata odjednom s naredbom ***db.collection.updateMany()***. U Compassu je dovoljno kliknuti na ikonu koja predstavlja uređivanje dokumenata, izmijeniti željena polja te na kraju kliknuti na *update*.

Brisanje se u MongoDB-u radi tako da se upiše naredba *remove().* Cijela bi naredba tako izgledala ***db.collection.remove()***. Za brisanje je potrebno, unutar zagrada, specificirati koji se dokument briše tako da se unese bilo koji naziv polja. Ako to nije učinjeno i zagrade ostanu prazne, obrisat će se svi dokumenti unutar kolekcije. Na slici 15. vidljiv je primjer brisanja jednog dokumenta. Briše se dokument iz kolekcije

Brodovi s nazivom broda Pasara Adria.

```
> db.Brodovi.remove({naziv_broda:"Pasara Adria"})
WriteResult({ "nRemoved" : 1 })
```

Slika 15. Brisanje dokumenta, izvor: vlastiti izvor

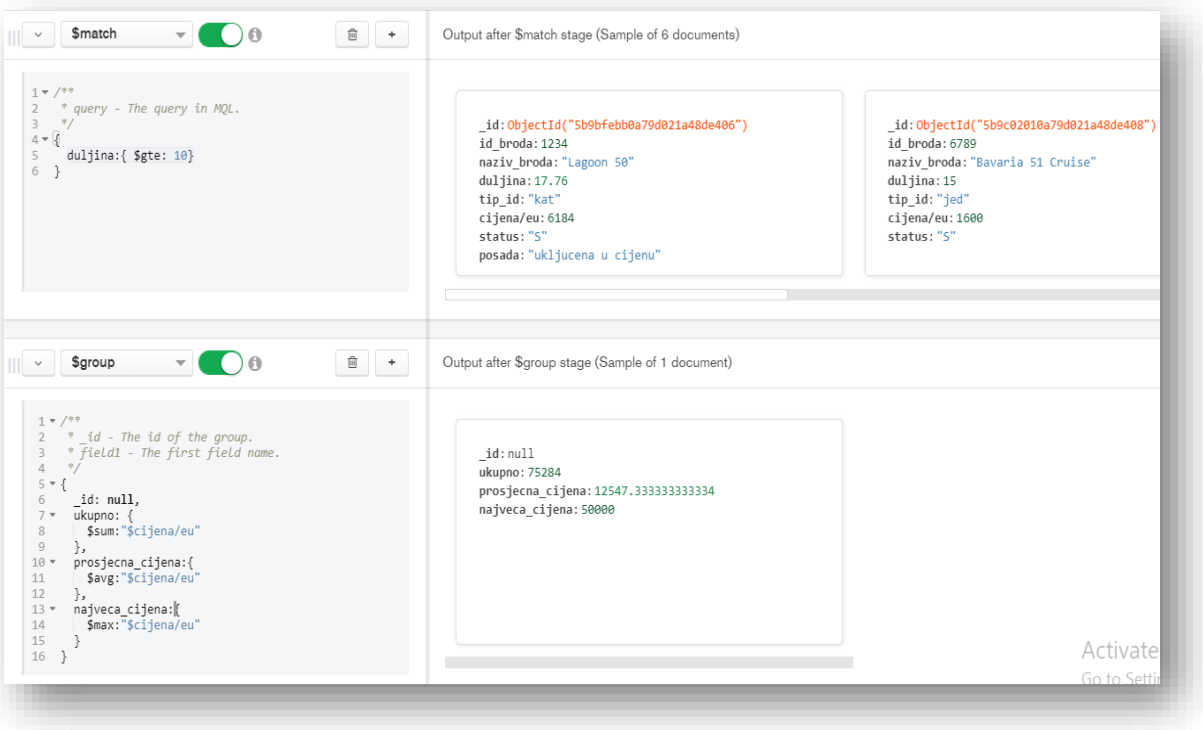
Iako je na ovom primjeru tražen brod s imenom Pasara Adria, potrebno je biti vrlo pažljiv jer postoji mogućnost da postoji još neki brod koji ima isti naziv, stoga je preporučljivo brisati po jedinstvenom identifikatoru *(id-u*). Nadalje, postoji mogućnost da se postave uvjeti po kojima će dokumenti biti izbrisan (slika 16.).

```
> db.Brodovi.remove({duljina:{lt:12}})
```

Slika 16. Brisanje dokumenata s uvjetom,izvor: vlastiti izvor

4.3.4. Agregacija i Indeksi

Agregacija sadrži operacije koje obrađuju dokumente i vraćaju rezultate.
Na sljedećem primjeru vidljiv je primjer agregacije koja je napravljena u Compassu. Prvo što se da primijetiti jest operacija *$match* koja vraća dokumente/rezultate koji odgovaraju uvjetu. Ovaj primjer pretražuje brodove čija je duljina veća od 10 metara. Rezultati su vidljivi odmah pored napisanog uvjeta.



Slika 17. Agregacija $match i $group u Compassu, izvor: vlastiti izvor

Ispod operacije *$match* nalazi se još jedna operacija koja se nadovezuje na prethodnu operaciju, a to je *$group*. Operacija *$group* grupira dokumente pomoću određenih izraza. Svaki *$group* mora sadržavati svoj *_id* koji ga opisuje.
Na primjeru je vidljivo da je *_id* ostao prazan. Ispod njega su napravljena tri polja: ukupno, prosjecna_cijena i najveca_cijena. Svaki od ovih polja sadrži izraz koji izračunava ukupni zbroj cijene brodova u eurima, prosječnu cijenu brodova u eurima i na kraju najveću cijenu brodova u eurima. U Mongo Shellu, agregacija se radi tako da se upiše naredba ***db.collection.aggregate()***. Unutar zagrada se zapisuju uvjeti koje operacija treba vratiti.

Sljedeći primjer prikazuje agregaciju pomoću funkcija *$match* i *$project*. Ova agregacija nam vraća rezultate iz baze podataka Northwind, točnije: *ProductID, ProductName* i *UnitPrice* gdje je *SupplierID*=1. *$project* možemo usporediti sa naredbom *SELECT* u SQL-u koja nam vraća željena polja. Ako se želi prikazati određeno polje uz njegovo ime stavlja se 1, u suprotnom se stavlja 0 ili se ostavlja prazno. Potrebno je napomenuti da ako je *_id* prazan, MongoDB će ga svejedno prikazati u rezultatima, stoga ga je potrebno posebno postaviti kao *_id*=0

```
> db.products.aggregate([
... {$match:{SupplierID:"1"}},
... {$project:{_id:0, ProductID:1, ProductName:1, UnitPrice:1}}
... ])
{ "ProductID" : "1", "ProductName" : "Chai", "UnitPrice" : "18.00" }
{ "ProductID" : "2", "ProductName" : "Chang", "UnitPrice" : "19.00" }
{ "ProductID" : "3", "ProductName" : "Aniseed Syrup", "UnitPrice" : "10.00" }
```

Slika 18. Agregacija $match i $project u Shellu, izvor: vlastiti izvor

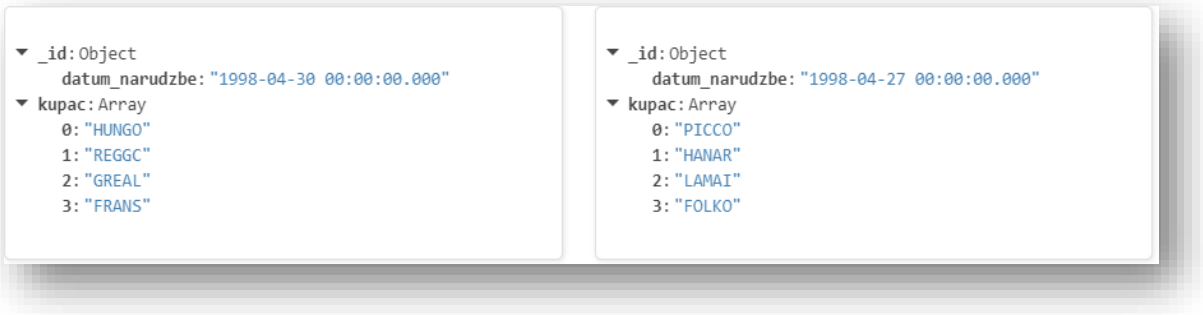
Na sljedećem primjeru koji je prikazan na slici 19. nalazi se agregacija s funkcijom *$group* i izrazom *$addToSet*. *$addToSet* sličan je naredbi *$push,* razlika je u tome što naredba *$addToSet* dodaje jedinstvene elemente u skup dok *$push* dodaje elemente u polje s duplikatima.

Ovaj primjer grupira dokumente u kolekciji orders u bazi podataka Northwind prema datumu narudžbe (*OrderDate*).

```
1  $group
2  {
3    _id: {datum_narudzbe:'$OrderDate'},
4    kupac: {
5      $addToSet: '$CustomerID'
6    }
7  }
```

Slika 19. Primjer agregacije $grup i $addToSet u Compassu, izvor: vlastiti izvor

$addToSet izrađuje popis kupaca (*CustomerID*) koji su obavili narudžbu na određeni datum.



Slika 20. Rezultat agregacije $addToSet, izvor: vlastiti izvor

Sljedeći primjer prikazat će agregatnu funkciju *$lookup* koja se može poistovjetiti sa JOIN-om u SQL-u. *$lookup* izvodi *LEFT OUTER JOIN* i spaja dvije kolekcije u istoj bazi podataka. Da bi se izvelo pridruživanje potrebno je koristiti sljedeću sintaksu:

```
{
   $lookup:
     {
       from: <collection to join>,
       localField: <field from the input documents>,
       foreignField: <field from the documents of the "from" collection>,
       as: <output array field>
     }
}
```

Slika 21. Sintaksa $lookup, izvor:
https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/#definition

U polje *from* se specificira ime kolekcije s kojom se želi pridružiti. *LocalField* podrazumijeva polje iz kolekcije u kojem se operacija *$lookup* izvodi. U *foreignField* se stavlja polje kolekcije koja se pridružuje, dok se u polje *as* specificira ime novog polja u koju će se dodavati rezultati.

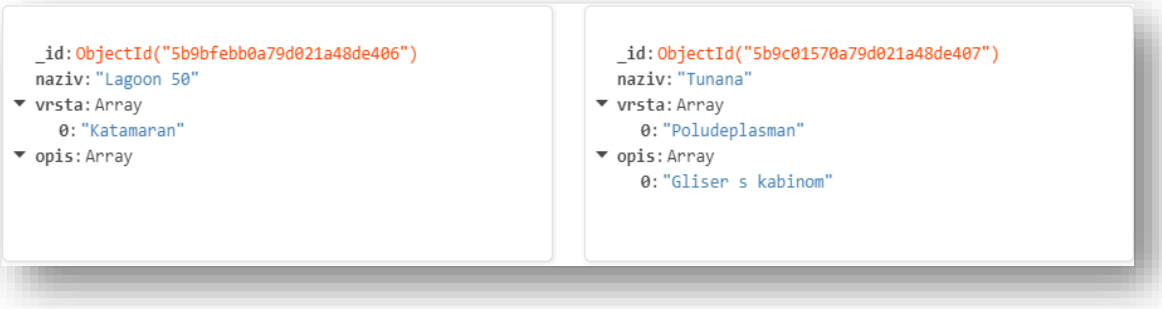
Primjer *$lookupa* s naredbom *$project* vidljiv je na slici 22. Agregacija $lookup rađena je u kolekciji Brodovi koja se pridružuje s kolekcijom Vrsta_broda. *LocalField* se nalazi u kolekciji Brodovi pod nazivom tip_*id*, *foreignField* u kolekciji Vrsta_broda koji je jedinstveni identifikator. Oni su povezani preko ručnih referenci (*Manual*

references) o kojima će biti više rečeno u sljedećem potpoglavlju.

```
[{
    $lookup: {
        from: 'Vrsta_broda',
        localField: 'tip_id',
        foreignField: '_id',
        as: 'brod'
    }
}, {
    $project: {
        naziv: '$naziv_broda',
        vrsta: '$brod.naziv',
        opis: '$brod.opis'
    }
}]
```

Slika 22. Agregacija $lookup i $project u Compassu, izvor: vlastiti izvor

Rezultati koji će biti vraćeni pomoću ove agregacije su: naziv broda, naziv vrste broda i opis vrste broda.



Slika 23. Rezultat agregacije $lookup i $project u Compassu, izvor: vlastiti izvor

Indeksi, kao što se već spominjalo, pohranjuju vrijednosti određenog polja ili skupine polja poredanih po vrijednostima. Poredak unosa indeksa pruža učinkovitije upite. Osim toga MongoDB može vratiti rezultate sortirane pomoću poretka u indeksima [14]. Obavezan indeks je *_id* , koji onemogućava korisniku da upiše dva *_id*-a s istim vrijednostima. On je dodijeljen i stalan što znači da ga nije moguće izmijeniti niti brisati.

U Compassu se indeks radi tako da se odabere opcija *create index*. Otvorit će se zasebni prozor koji traži upisivanje naziva indeksa (koji je opcionalan). Zatim se

odabire polje koje se uvrštava kao indeks te se odabire tip indeksa. Osim jednog polja, moguće je odabrati i više njih. Opcije nude mogućnosti da indeks bude kreiran u pozadini, da bude jedinstven, stvaranja TTL indeksa (posebni indeksi koji automatski brišu dokumente iz kolekcije nakon određenog vremena) te na kraju stvaranja djelomičnih indeksa.

4.3.5. Reference

U većini slučajeva podaci u MongoDB su denormalizirani tj. podaci su uglavnom pohranjeni unutar jednog dokumenta. Međutim, postoje slučajevi kada je potrebno pohraniti podatke u zasebne dokumente, kolekcije ili čak i drugu bazu podataka. Tada je potrebno koristiti reference.

U Mongu postoje dvije vrste referenca: Ručne reference (*Manual References*) i DBRefs. Ručne reference se izrađuju tako da se *_id* polje jednog dokumenta sprema u drugi dokument kao referenca. Ovakve reference su jednostavne i koriste se u većini slučajeva [14].

Baza RentMeMedulin osim kolekcije Brodovi, sadrži i kolekciju Vrsta_broda. Poznato je da postoji više vrsta brodova pa tako imamo: katamarane, jedrilice, barke, glisere, itd. Slika 24. prikazuje kolekciju Vrsta_broda i sve njegove dokumente.

Vrsta_broda

	_id String	naziv String	opis String
1	"kat"	"Katamaran"	No field
2	"jed"	"Jedrilica"	No field
3	"pdp"	"Poludeplasman"	"Gliser s kabinom"
4	"spdb"	"Gliser"	No field
5	"barka"	"barka"	"plovilo s manjom snagom"
6	"fisher"	"fisherman"	No field

Slika 24. Kolekcija Vrsta_broda, izvor:vlastiti izvor

Kolekcija Brodovi u svojim dokumentima sadrži tipove kojima pripadaju određeni brodovi. Za većinu brodova korištena je Ručna referenca, koja se napravila tako da se u kolekciju Brodovi pod poljem tip_*id* postavio *_id* od Vrste_broda (slika 25.).

```
_id: ObjectId("5b9cc7781899e73038386f12")
id_broda: 8541
naziv_broda: "Hanse 415"
duljina: 13
tip_id: "jed"
cijena/eu: 3500
posada: "ukljucena u cijenu"
```

Slika 25. Ručno referenciranje, izvor: vlastiti izvor

Druga vrsta, DBRefs povezuje jedan dokument s drugim pomoću *_id* vrijednosti prvog dokumenta, naziva kolekcije ili baze podataka. Uključivanjem ovih imena DBRef-ovi omogućuju lakše povezivanje dokumenata smještenih u više kolekcija s dokumentima iz jedne kolekcije. Primjer izrade DBRef-a može se vidjeti na slici 26.

Slika 26. Primjer DBRef-a, izvor:vlastiti izvor

U polju tip_*id* vidljiva su dva DBRef-a jer ovaj brod možemo svrstati i u katamaran i jedrilicu. DBRef se napravio tako da se prvo definira *$ref*, koje sadrži naziv kolekcije, u ovom slučaju Vrsta_broda. Zatim se dodaje polje *$id*, koje sadrži vrijednost *_id* polja referenciranog dokumenta te se na kraju dodaje *$db* koje sadrži ime baze podataka. Ovo polje je opcionalno, što znači da ga nije potrebno definirati.

Nije preporučljivo korištenje DBRef-a za samo jednu referencu.

Za manipuliranje DBRef-ovima aplikacija mora izvršiti dodatne upite da bi vratila referencirani dokument. Postoje upravljački programi koji sadrže pomoćne metode koje stvaraju automatske upite za DBRef-ove.
Primjeri upravljačkih programa koji podržavaju DBRef: C#, Java, JavaScript, Node.js, Perl, Python, Ruby [14].

4.3.6. MapReduce

MapReduce služi za obradu velike količine podataka u korisne agregirane podatke. MapReduce naredba prima dva primarna ulaza od koje je jedna *map* funkcija, a druga *reduce.* Funkcija *map* čita svaki dokument u kolekciji koji odgovara uvjetima upita. Za one ključeve koji imaju više vrijednosti, MongoDB koristi funkciju reduciranja koja prikuplja i kondenzira podatke dobivene u *map* funkciji. Na kraju sustav sprema rezultate kao dokument ili kao kolekciju.

Slika 27. prikazuje prije navedene funkcije. Varijabla *map* sadrži funkciju mapiranja koja pretražuje dokument s nazivom polja *Gender*. Varijabla *red* reducira zapise, dodaje vrijednosti i vraća njihov zbroj.

```
> var map= function (){ emit(this.Gender, 1)};
> map
function (){ emit(this.Gender, 1)}
> var red= function(gender, count){
... return Array.sum(count);};
```

Slika 27. Primjer funkcija MapReduce u Mongo Shellu, izvor: vlastiti izvor

Rezultat prikazuje (slika 28.) da je pretraženo jedanaest dokumenata, da je funkcija *map* emitirala 11 dokumenata s parovima ključ/vrijednost, te da je funkcija *reduce* grupirala mapirane dokumente koji imaju iste ključeve u 2 dokumenta. Ako se rezultati mapReducea žele pogledati to se čini pomoću uobičajene naredbe *find().* U ovom slučaju to bi izgledalo: *db.primjer.find()*

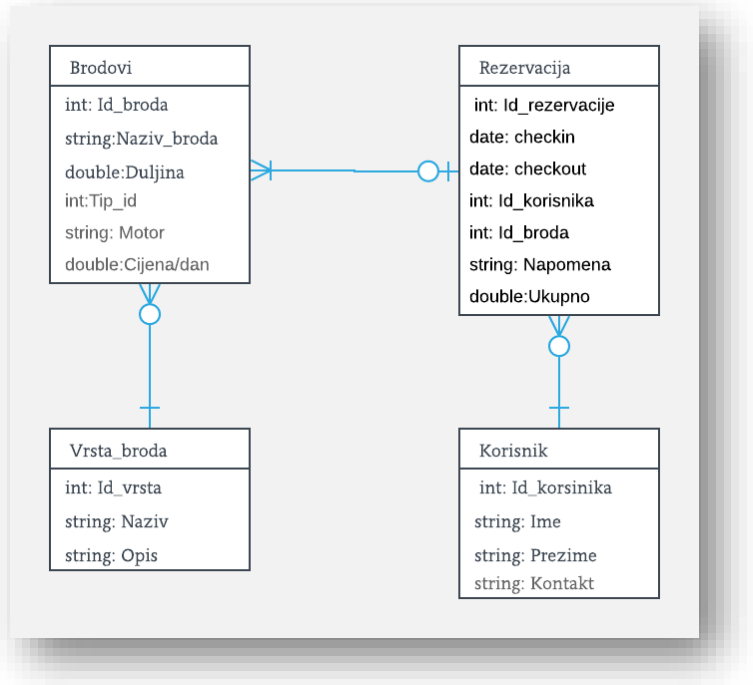
```
> db.employees.mapReduce( map, red, {out: "primjer"})
{
        "result" : "primjer",
        "timeMillis" : 869,
        "counts" : {
                "input" : 11,
                "emit" : 11,
                "reduce" : 2,
                "output" : 2
        },
        "ok" : 1
}
```

Slika 28. Rezultat mapReduce funkcije u Mongo Shellu, izvor: vlastiti izvor

5. Usporedba MongoDB-a i Relacijske baze podataka

Da bi se napravila relacijska baza podataka potrebno je proći nekoliko koraka. Prvi korak je izgradnja konceptualnog modela u kojem se opisuje način na koji baza podataka izgleda, tj. prikazuje se njen sadržaj i međusobne veze. Konceptualni

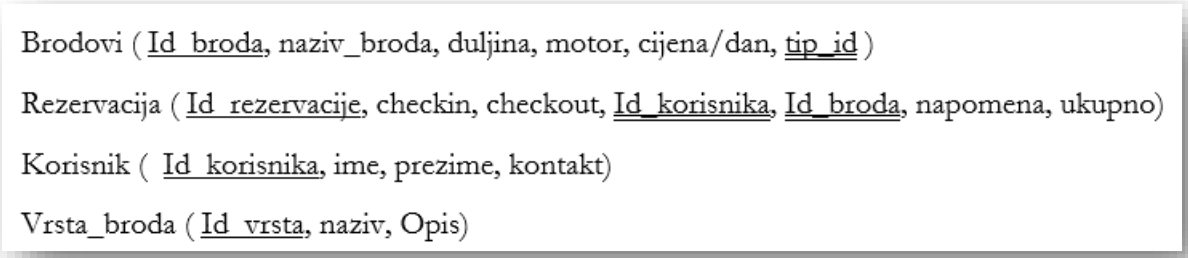
model baze podataka za iznajmljivanje brodova možemo vidjeti na slici 29.



Slika 29. Konceptualni model, izvor:vlastiti izvor

Sljedeći važan korak kod modeliranja relacijske baze podataka jest pretvaranje konceptualnog modela u logički model. Rezultat logičkog modela je logička shema koja je sastavljena od relacija odnosno tablica. U ovom se koraku radi i normalizacija koja je objašnjena u prvom poglavlju i jedna je od glavnih karakteristika Relacijskog modela.

Sljedeći korak je fizička shema i implementacija koju karakteriziraju SQL naredbe kao npr: *create database, create table* kojim se stvara oblikovana baza i relacije na računalu (vidljivo na slici 30.). Nakon toga, baza podataka se puni podacima u odgovarajuće tablice.



Slika 30. Relacijska shema, izvor: vlastiti izvor

Kod izrade MongoDB baze podataka za iznajmljivanje brodova nije bilo potrebno raditi konceptualni model niti ga pretvarati u logičku shemu. Razlog tome je što NoSQL modeli baza podataka nemaju unaprijed predodređenu shemu za razliku od relacijskog modela.

MongoDB sadrži dinamičku shemu, koja omogućuje brzo i jednostavno mijenjanje podataka i sheme čime se olakšava upravljanje, održavanje i prilagođavanje novim poslovnim zahtjevima. Nepromjenjiva shema jedna je od većih mana relacijskog modela, jer postoje situacije koje od baze podataka traže različit broj redaka, vrijednosti i veličine polja, što za MongoDB bazu podataka ne predstavlja problem. To je vidljivo na primjeru gdje Korisnik može imati više vrsta kontakata, npr. *e-mail* i telefon koji su različitog tipa a nalaze se pod istim atributom.

Sljedeća bitna razlika je u tome što u relacijskoj bazi podataka programer baze mora eksplicitno dodavati i brinuti se za primarne i strane ključeve. MongoDB baza podataka automatski dodaje primarni ključ odnosno jedinstveni identifikator (*ObjectId (<hexadecimal>)*), a strani ključevi ne postoje, ali, mogu se dodavati ugniježđeni dokumenti ili reference koje će povezati pojedine dokumente. Međutim, kako MongoDB pohranjuje ključ za svaki dokument, ne postoji normalizacija i ne koriste se „standardne veze“ kao kod relacijskog modela, može rezultirati redundancijom i nepotrebnim povećanjem memorije [5].

Za razliku od relacijskih baza podataka koje se zasnivaju na vertikalnom skaliranju - koja dodaje resurse jednome računalu točnije, čvoru (npr. memorija, procesor, itd.), MongoDB se bazira na horizontalnoj skalabilnosti, koja dodaje čvorove u sustav te

raspoređuje opterećenje na više poslužitelja. Time se riješio problem sa sve većom količinom podataka te je olakšana distribucija podataka.[10]

Relacijske baze podataka koriste SQL kao jezik za upite. Spomenuto je da je SQL moćan jezik koji je dizajniran za manipuliranjem podacima u relacijskoj bazi. Ovaj jezik uključuje: umetanje podataka u relaciju, ažuriranje, brisanje, upite, kreiranje i izmjena sheme te kontrolu pristupa podacima.

MongoDB koristi deklarativni jezik JSON, koji se u većini slučajeva ne razlikuje puno od SQL što se tiče mogućnosti, ali za pojedince je ovaj jezik više intuitivniji i jednostavniji za razliku od SQL-a. JSON više odgovara web programerima jer im je jezik intuitivan pa prototipiranje aplikacija ide puno brže, čime se skraćuje vrijeme izrade i izdavanja aplikacije.
Sljedeća tablica prikazuje usporedbu dvaju modela.

Tablica 12. Usporedba SQL-a i MongoDB-a,

	SQL	MongoDB
Kreiranje tablice	CREATE TABLE Brodovi (id_broda INT NOT NULL,Naziv_broda VARCHAR(30), Duljina Double, Cijena DOUBLE, tip_id char(5), PRIMARY KEY (Id_broda),CONSTRAINT FK_tip_broda FOREIGN KEY (tip_id) REFERENCES Vrsta_broda(tip_id)	db.createCollection(“Brodovi“)
Unos zapisa	INSERT INTO Brodovi VALUES (111, 'Salona 37', '11.30', 1200, 'jed')	db.Brodovi.insertOne({ id_broda: “111“, naziv_broda: “Salona 37“, duljina:“11.30“, cijena:1200,tip_id:“jed“)}
Ažuriranje	ALTER TABLE Brodovi ADD Godina_izrade DATETIME	db.Brodovi.updateMany({ }, { $set: {Godina_izrade: new Date() }}
Ažuriranje zapisa	UPDATE Brodovi SET cijena= 2000 WHERE duljina>10	db.Brodovi.updateMany ({duljina:{$gt:10}},

		{$set:{cijena:“2000“}})
Dohvaćanje zapisa	SELECT * from Brodovi WHERE tip_id=“jed“ ORDER BY naziv_broda ASC	db.Brodovi.find({tip_id:“jed“}). sort{naziv_broda:1}
Agregatne funkcije	Select id_korisnika, SUM(ukupno), FROM Rezervacija, GROUP BY id_korisnika	db.Rezervacija.aggregate([{ $group: { _id: “id_korisnika“, total:{$sum:“ukupno“}} }])
Brisanje zapisa	DELETE FROM Brodovi where duljina>10	db.Brodovi.deleteMany({duljina:{$gt:10}})

izvor: vlastiti izvor

Iz tablice je vidljivo da se manipuliranje podacima ne razlikuje previše osim u samoj sintaksi. Međutim, bitno je primijetiti kako se u SQL-u kod kreiranja tablice moraju odmah definirati atributi s tipom podataka, dok to u MongoDB-u nije potrebno već se to radi prilikom unošenja podataka u kolekciju.

Postoje situacije kada više korisnika pokušava ili želi pristupiti podacima u istoj tablici u istom trenutku. To može dovesti do nekonzistentnosti ili netočnih podataka. Da bi se to spriječilo koriste se transakcije čija su svojstva ACID (***A**tomicity, **C**onsistency, **I**solation, **D**urability*). SQL podržava transakcije.

U MongoDB-u operacija nad jednim dokumentom je atomska. Budući da se mogu upotrebljavati ugniježđeni dokumenti i polja u jednoj strukturi dokumenta da bi se napravile „veze“ i ne koristi se normalizacija, atomska cjelina s jednim dokumentom uklanja potrebu za transakcije s više dokumenata. Međutim, za situacije koje zahtijevaju atomizaciju za ažuriranje više dokumenata MongoDB (tek od verzije 4.0) pruža mogućnost obavljanja transakcija s više dokumenata. Transakcije s više dokumenata mogu se koristiti na više operacija, kolekcija, baza podataka i dokumenata. Kod operacija kao što je ažuriranje, zajamčena je potpuna izolacija svojstvima ACID-a. Takve se transakcije baziraju na „sve-ili-ništa“, odnosno, operacija se zaustavlja ako ima ili dođe do bilo kakvih pogrešaka, a sve izmjene koje su se dogodile u transakciji će biti odbijene bez da uopće budu vidljive. Ako se

transakcija izvrši, bit će spremljene sve izmjene podataka u transakciji. [16][2]

Kako relacijske baze podataka iza sebe imaju dugu povijest koja datira još od 1980 godine ne čudi što imaju veliku podršku, brojne nadogradnje i testiranja. MongoDB je još uvijek „mlad“ na tržištu, pa ni ne čudi što nije toliko opsežno dokumentiran ili testiran. Osim toga nedostaje veća dostupnost podrške i stručnjaka [6].

[2] Documentation- Transactions

6. Zaključak

Relacijske baze podataka već su dugi niz godina u upotrebi. Većina organizacija svoje je podatke pohranjivala i još uvijek pohranjuje prema Coddovom modelu. Prepoznatljiva je po tablicama, atributima, SQL upitnom jeziku, striktnoj shemi koju nije moguće mijenjati i normalizaciji. Idealna za manje organizacije s manjom količinom podataka, ali nedovoljno dobra za brzi napredak *weba*, *web* tehnologija i velikih organizacija.

Iako su još uvijek nepoznate širokoj javnosti, NoSQL tehnologije pokazale su se izvrsnima u problemima kao što su velika količina podataka, stalne izmjene, brzom razvoju i sličnom, radi vrlo visoke fleksibilnosti i skalabilnosti.

Postoji nekoliko vrsta NoSQL baza podataka, a u ovom modelu detaljnije je obrađena dokument baza podataka koja je predstavljena preko MongoDB implementacije. Iako ograničen nedostupnošću operacija pridruživanja i mogućnošću redundancije koja zauzima veliku količinu memorije, uočljivo je koliko je ovaj model intuitivan i fleksibilan i jednostavan za korištenje. To mu omogućuju brojne karakteristike kao npr. dinamička shema, agregacija, indeksi i popularan jezik među programerima: JSON koji omogućava još brži i jednostavniji razvoj aplikacija.

MongoDB implementacija i njegove karakteristike detaljnije su obrađene preko baze podataka RentMeMedulin i Northwind. Iako su mogućnosti MongoDB-a velike, ove dvije baze podataka nisu dovoljno složene da bi se prikazale njene stvarne mogućnosti pogotovo kada su u pitanju performanse. Unatoč tome, predstavljene su teorijske osnove koje uključuju izradu baze podataka, spajanje na *poslužitelj*, ažuriranje, kreiranje i brisanje podataka, a popratni primjeri daju predodžbu kako se radi manipulacija podacima u grafičkom korisničkom sučelju i u standardnom Mongo Shellu.

7. Literatura

Internet izvori

1. **Anić, V.,** (2016), NoSQL baza podataka računalnih komponenti, Sveučilište Josipa Jurja Strossmayera u Osijeku: Elektrotehnički Fakultet, Osijek, Završni Rad,
   URL:
   https://repozitorij.etfos.hr/islandora/object/etfos%3A850/datastream/PDF/view
2. **Baze podataka**, Postupak normalizacije i normalne forme
   [pristupljeno: 25.08.2018]
3. **Codd'sRules**,Techopedia,
   URL: https://www.techopedia.com/definition/1170/codds-rules
   [pristupljeno: 26.08.2018]
4. **Dadić, T.** (2012) Baze podataka, Fakultet prirodoslono-matematičkih znanosti Sveučilišta u Splitu], URL:
   http://mapmf.pmfst.unist.hr/~tdadic/Dadic_BazePodataka.pdf
   [pristupljeno:25.08.2018]
5. **Data Flair** (2018)**,** Advantages of MongoDB|Disvantages of MongoDB,
   URL: https://data-flair.training/blogs/advantages-of-mongodb/
   [pristupljeno: 20.09.2018]
6. **Easylearning** (2016), What is MongoDB? The advantages & Disvantages of MongoDB,
   URL: http://blog.easylearning.guru/what-is-mongodb-the-advantages-disadvantages-of-mongodb/ [pristupljeno: 20.09.2018]
7. **Gačić, J. (**2017), NoSQL baze podataka, Sveučilište u Zagrebu: Prirodoslovni - Matematički Fakultet, Matematički Odsjek, Zagreb,Diplomski Rad, URL:
   https://repozitorij.pmf.unizg.hr/islandora/object/pmf%3A3234/datastream/PDF/view
8. **Gilikin, J.,** What are the limitations of relational databases in business applications?,
   URL: https://smallbusiness.chron.com/limitations-relational-databases-business-applications-24159.html

9. **Grašic, Ž**. (2016), SQL i standardi za pristup bazama podataka, Grafički Fakultet, Studij: Grafička Tehnologija, Seminar, URL: http://app.evasms.com/claroline/claroline/backends/download.php?url=L3NlbWluYXJza2lfcmFkb3ZpLzEyX1NRTF9pX3N0YW5kYXJkaV96YV9wcmlzdHVwX2JhemFtYV9wb2RhdGFrYV9HcmFzaWNaZWxpbWlyLnBkZg%3D%3D&cidReset=true&cidReq=MP1516 [pristupljeno: 29.08.2018]
10. **Janjić, V**. (2018), NoSQL baza podataka, Sveučilište Josipa Jurja Strossmayera u Osijeku: Fakultet Elektrotehnike, Računarstva i Informacijskih tehnologija, Diplomski Rad URL: https://repozitorij.etfos.hr/islandora/object/etfos%3A1734/datastream/PDF/view
11. **Kaluža, M.** (2008), Sustavi baza podataka, Rijeka Veleučilište, URL: https://www.veleri.hr/files/datotekep/nastavni_materijali/k_informatika_1/Predavanja_skripta.pdf [pristupljeno 25.08.2018]
12. **Kratki uvod u SQL kroz primjere,** URL:http://grdelin.phy.hr/~ivo/Nastava/Baze_podataka/predavanja-2002/06/ [pristupljeno : 31.08.2018]
13. **Manger, R**., Osnove projektiranja baza podataka Tečajevi Srca,Sveučilište u Zagrebu, URL: https://www.srce.unizg.hr/files/srce/docs/edu/osnovnitecajevi/d310_polaznik.pdf [pristupljeno 25.08.2018]
14. **Martin, A**., Disadvantages of a Relational Database, URL: https://www.techwalla.com/articles/disadvantages-of-a-relational-database [pristupljeno 31.08.2018]
15. **MongoDB Architecture Guide**, MongoDB 4.0, URL: https://www.mongodb.com/collateral/mongodb-architecture-guide [pristupljeno 08.09.2018]
16. **MongoDB**, Documentation, URL: https://docs.mongodb.com/ [pristupljeno 08.09.2018]
17. **MongoDB-Capped Collections**, TutorialsPoint, URL: https://www.tutorialspoint.com/mongodb/mongodb_capped_collections.html [pristupljeno 12.09.2018]

18. **Normalizacija baza podataka,** Razno Sveznadar Informatika, URL: razno.sveznadar.info/10-doc-PDF/Normalizacija.pdf [pristupljeno: 27.09.2018]
19. **O'Connor, J.,** (2017), Aggregations in MongoDB by example, URL: https://www.compose.com/articles/aggregations-in-mongodb-by-example/ [pristupljeno 12.08.2018]
20. **Peroković, M**., (2014), Primjena NoSQL baza podataka na sustav za upravljanje dokumentima, Sveučilište u Zagrebu: Fakultet Organizacije i Informatike,Varaždin, Diplomski Rad, URL: https://bib.irb.hr/datoteka/714321.1-maperokov-primjenaNoSQLnaDMS.pdf
21. **Rouse, M**., NoSQL (Not Only SQL database), URL: https://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL
22. **Singh, C**., Normalization in DBMS:1NF,2NF, 3NF and BCNF in database, URL: https://beginnersbook.com/2015/05/normalization-in-dbms/
23. **Soltezs Lee, D**., What are the Advantages of a Relational Database Model?,Techwalla, URL: https://www.techwalla.com/articles/what-are-the-advantages-of-a-relational-database-model [pristupljeno 31.08.2018]
24. **Stojanović, A**.,(2016), Osvrt na NoSQL baze podataka - četiri osnovne tehnologije, URL: https://repozitorij.etfos.hr/islandora/object/etfos%3A850/datastream/PDF/view
25. **Tivanovac, M**., (2016), Modeliranje Nerelacijskih Baza Podataka, Sveučilište Josipa Jurja Strossmayera u Osijeku: Fakultet Elektrotehnike, Računarstva i Informacijskih tehnologija, Završni rad, URL: https://repozitorij.etfos.hr/islandora/object/etfos:970/preview
26. **Tomić, N**. (2016), NoSQL tehnlogije i primjene, Sveučilište u Zagrebu: Prirodoslovni-matematički fakultet, Matematički Odsjek, Zagreb, Diplomski rad, URL: http://digre.pmf.unizg.hr/4847/1/NelaTomic_NoSQL_2016.pdf
27. **Veljković, N**.(2013), Nerelacione baze podataka NoSQL, Univerzitet u Beogradu, Matematicki fakultet [pristupljeno 08.09.2018]

28. **Why NoSQL Database,** Couchbase
URL: https://www.couchbase.com/resources/why-nosql
29. **/dev/Kico** (2013), Things they don't tell you about MongoDB,
URL: https://www.itexto.com.br/devkico/en/?p=44&utm_content=bufferac855&utm_source=buffer&utm_medium=twitter&utm_campaign=Buffer [pristupljeno: 20.09.2018]

Popis slika:

Popis tablica

8. Temeljna dokumentacijska kartica

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli
Preddiplomski studij Informatika

MongoDB

Maja Pavlović

SAŽETAK

Razvojem weba i web tehnologija, potreba za spremanjem velike količine podataka postala je prioritetna. Relacijske baze podataka ne mogu se nositi s teretom stalnog izmjenjivanja, nadograđivanja i novim poslovnim zahtjevima. NoSQL tehnologije sadrže novi pristup u manipuliranju podacima. Jedna od takvih tehnologija opisana je u ovome radu – MongoDB. Jednostavnost, skalabilnost, dostupnost i dinamička shema samo su neke od karakteristika koja ova baza podataka pruža.
Ovaj rad se osvrnuo na relacijski model baza podataka, njegove karakteristike i mane i NoSQL modele koji su detaljnije opisani. Na kraju je predstavljena i napravljena MongoDB baza podataka koja na primjerima prikazuje njegove najbitnije karakteristike.

Ključne riječi: NoSQL, MongoDB, RDBMS
Mentor: doc. dr. sc. Tihomir Orehovački

9. Basic documentation card

Jurja Dobrila University of Pula
Faculty of Informatics
Undergraduate Study Programme - Informatics

MongoDB

Maja Pavlović

ABSTRACT

With the development of the web and web technologies, the need to store great amounts of information and data has become a priority. Relational databases are not able to function with the constant need to edit, update and the needs of new business requests. NoSQL technologies have a new way of manipulating the data. One of those technologies is mentioned in this paper- MongoDB. Simplicity, scalability, availability and dynamic scheme are just a few of the many great characteristics that this kind of database offers. This paper explains the relational database model, it's characteristics and disadvantages. Also, the NoSQL models are explained more elaborately. In the end the MongoDB database has been presented and created for exemplary purpose which shows it's most noticeable and important characteristics.

Key words: MongoDB, NoSQL, RDBMS
Supervisor: doc.dr.sc. Tihomir Orehovački