SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

USPOREDBA RELACIJSKIH I NOSQL BAZA PODATAKA

Završni rad

Dominik Vidović

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 05.06.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Dominik Vidović
Studij, smjer:	Računalno inženjerstvo
Mat. br. Pristupnika, godina upisa:	R4584, 27.07.2020.
OIB Pristupnika:	30056904953
Mentor:	izv. prof. dr. sc. Ivica Lukić
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Usporedba relacijskih i NoSQL baza podataka
Znanstvena grana rada:	**Obradba informacija (zn. polje računarstvo)**
Zadatak završnog rad:	U završnom radu treba usporediti strukturu relacijske baze podataka s različtim NoSQL bazama podataka. Objasniti razlike te usporediti prednosti i nedostatke. Usporediti skalabilnost SQL-a i NoSQL-a (vertikalna i horizontalna), što to znači, te navesti na primjerima, gdje se oni koriste i zašto se koriste, koje su prednosti i nedostatci za pojedinačne primjere. Tema je rezervirana za: Dominik Vidović
Prijedlog ocjene završnog rada:	Vrlo dobar (4)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 2 razina
Datum prijedloga ocjene od strane mentora:	05.06.2023.
Datum potvrde ocjene od strane Odbora:	12.07.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 12.07.2023.

Ime i prezime studenta:	Dominik Vidović
Studij:	Računalno inženjerstvo
Mat. br. studenta, godina upisa:	R4584, 27.07.2020.
Turnitin podudaranje [%]:	12

Ovom izjavom izjavljujem da je rad pod nazivom**: Usporedba relacijskih i NoSQL baza podataka**

izrađen pod vodstvom mentora izv. prof. dr. sc. Ivica Lukić

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

U današnjem informacijskom dobu, baze podataka igraju ključnu ulogu u poslovanju i upravljanju informacijama. Relacijske baze podataka su standardni oblik baze podataka koji se koristi u većini aplikacija, no s pojavom novih tehnologija pojavile su se i alternative, poput NoSQL (engl. *Not only Structured Query Language*) baza podataka.

U ovom završnom radu usporedit ćemo strukturu relacijskih baza podataka s različitim NoSQL bazama podataka, objasniti njihove razlike te usporediti njihove prednosti i nedostatke. Posebno ćemo se fokusirati na skalabilnost SQL-a (engl. *Structured Query Language*) i NoSQL-a (vertikalnu i horizontalnu) te ćemo navesti primjere gdje se ove baze podataka koriste i zašto se koriste, istaknuvši prednosti i nedostatke za pojedinačne primjere.

Ovaj rad je namijenjen svima koji se bave upravljanjem bazama podataka ili su zainteresirani za temu baza podataka. Cilj ove teme je pružiti ključne razlike i prednosti između relacijskih i NoSQL baza podataka, te istaknuti primjere primjene skalabilnost SQL-a i NoSQL-a u praksi.

1.1. Zadatak završnog rada

Cilj ovog završnog rada je usporediti strukturu relacijske baze podataka s različitim NoSQL bazama podataka, objasniti njihove razlike i usporediti prednosti i nedostatke. Također, bit će uspoređena skalabilnost SQL-a i NoSQL-a, odnosno vertikalna i horizontalna skalabilnost te će se navesti primjeri gdje se oni koriste i zašto se koriste. Kroz primjere baza podataka biti će pokazana njihova razlika u strukturi i načinu i brzini obrade upita. Sve navedeno će doprinijeti boljem razumijevanju i odabiru baza podataka za određenu primjenu.

2. PREGLED PODRUČJA RADA

U današnjem digitalnom dobu, količina podataka koju generiramo i obrađujemo eksponencijalno raste. Ovaj porast podataka izaziva velike izazove u njihovom skladištenju i upravljanju. Usporedo s tim izazovima, razvijaju se različite vrste baza podataka kako bi se adekvatno nosile s raznovrsnim zahtjevima modernih aplikacija.

Jedna od ključnih dilema s kojom se suočavaju razvijatelji softvera i arhitekti baza podataka je izbor između tradicionalnih relacijskih baza podataka i novijih NoSQL baza podataka. Relacijske baze podataka su dominantne već desetljećima, dok su NoSQL baze podataka relativno novi pristup koji se pojavio kao odgovor na potrebu za skalabilnošću, fleksibilnošću i visokom dostupnošću podataka. U narednim poglavljima detaljnije će se pojasniti usporedba relacijskih i NoSQL baza podataka u smislu skalabilnosti i strukture pojedinih baza podataka. Za primjer usporedbe koristiti će se relacijska baza podataka i dokument NoSQL baza podataka te tip graf, tip dokument i tip široki stupac. Područje usporedbe svih tipova baza podataka je opširno tako da će se u ovom radu usporediti općenite stvari svake baze te će se za primjer NoSQL baze podataka koristiti MongoDB, Neo4j i Apache Cassandra baze podataka koji je jedan od mnogih tipova NoSQL baza podataka. Ovaj rad pokriva područje skalabilnost baze podataka kao što su horizontalna i vertikalna skalabilnost te pokriva prednosti i nedostatke svake baze podataka u smislu njihove strukture i u smislu njihove skalabilnost. Ujedno će se navesti primjena svakih od baza podataka bile to relacijske ili neki tip NoSQL baze podataka te koji ih poslodavci koriste. Detaljnije usporedbe relacijske i NoSQL baze podataka se mogu pregledati na [1]. Navedeni rad pokriva područje općenite usporedbe relacijskih i NoSQL baza podataka te detaljniju usporedbu relacijske baze podataka i graf tip NoSQL baze podataka. Bazira se na detaljnom opisu vrsta podataka kao što su strukturirani, polustrukturirani i nestrukturirani podaci, a u ovom radu će se samo navesti i ukratko opisati. Navedeni rad ujedno pokriva detaljnu usporedbu i analizu graf i relacijske baze podataka u smislu brzine izvođenja upita.

Za izradu ovog rada korišten je internet pretraživač koji sa određenim upitima daje potrebne internetske stranice preko kojih su izvučene bitne informacije za izradu ovog rada. Informacije su izdvojene sa vjerodostojnih stranica koji prenose točne informacije. Ujedno za izradu ovog rada, kao inspiracija, uzeti su završni, diplomski te znanstveni radovi kao što su [1][2] kako bi se pobliže shvatila i razumjela tema rada. Za one koji žele znati više ili žele bolje razumjeti druge tipove NoSQL baza podataka kao što je baza širokih-stupaca (engl. *wide-*

column) mogu proučiti rad [2]. Navedeni rad objašnjava u detalje način korištenja Apache Cassandra[1] te način korištenja i koje su joj osobine, objašnjava arhitekturu i dijelove tog tipa NoSQL baze podataka te ih uspoređuje i opisuje sa relacijskom bazom podataka.

Kao što je i prijašnje objašnjeno u sljedećim poglavljima opisujemo relacijske i NoSQL baze podataka, njihove primjene, strukture, prednosti i nedostatke, opisujemo načine primjene te vrste i mogućnosti skalabilnosti pojedinih baza podataka te na kraju uz primjere baze podataka uspoređujemo njihovu strukturu i brzinu obrađivanja zahtjeva.

[1]Apache Cassandra – besplatan, distribuiran, otvorenog koda, NoSQL sustav za upravljanje bazom podataka s pohranom širokih stupaca

3. RELACIJSKE BAZE PODATAKA

U narednom poglavlju opisat će se relacijska baza podataka, struktura relacijske baze podataka te objasniti prednosti i nedostatci koje baza podataka donosi. Također navest će se primjeri upotrebe relacijskih baza podataka i nabrojati te opisati koje velike kompanije trenutno koriste relacijske baze podataka i za što ih koriste.

3.1. Definicija relacijske baze podataka

Relacijska baza podataka je jedna od vrsta baza podataka koja pohranjuje i omogućuje pristup točkama podataka koje su međusobno povezane. U relacijskoj bazi podataka podaci su pohranjeni u jednoj ili više tablica (relacija) i više stupaca i redaka, što olakšava vidjeti i razumjeti kako su različite strukture podataka međusobno povezane [3]. Svaka tablica, koja se naziva i relacija, sadrži jednu ili više kategorija podataka u stupcima ili atributima. Svaki redak, koji se također naziva zapis, sadrži jedinstvenu instancu podataka ili ključ za kategorije definirane stupcima [4]. Model relacijske baze podataka razvio je E.F. Codd [2]iz IBM-a [3]1970-ih. Omogućuje da se bilo koja tablica poveže s drugom tablicom koristeći zajednički atribut. Umjesto korištenja hijerarhijskih struktura za organiziranje podataka, Codd je predložio prijelaz na korištenje podatkovnog modela gdje se podaci pohranjuju, pristupa im se i povezuju u tablicama bez reorganizacije tablica koje ih sadrže [3]. Stupci predstavljaju određene vrste podataka, kao što su tekst, brojevi ili datumi, a svaki red sadrži određeni skup vrijednosti za svaki atribut. Odnosi između tablica mogu se uspostaviti definiranjem primarnih i stranih ključeva, koji omogućuju povezivanje podataka između različitih tablica na temelju zajedničkih vrijednosti što će se detaljnije objasniti u poglavlju 3.2.. To omogućuje učinkovito postavljanje upita i dohvaćanje podataka.

3.2. Opis strukture relacijske baze podataka

Relacijska baza podataka je pojašnjena kao zbirka povezanih podatkovnih tablica koje organiziraju podatkovne točke s definiranim odnosima radi lakšeg pristupa [5]. Podatkovne strukture, uključujući podatkovne tablice, indekse i preglede, ostaju odvojene od fizičkih struktura za pohranu, omogučujući administratorima baze podataka da osiguraju da je relacijska baza podataka točna i dostupna. Struktura modela relacijske baze podataka temelji se na relacijskom modelu, koji je intuitivan, jednostavan način predstavljanja podataka u tablicama.

[2]E.F. Codd – Edgar Frank Codd, Engleski informatičar
[3]IBM – Međunarodna poslovna strojna korporacija (eng. *International Business Machines Corporation)*

U relacijskom modelu podaci su organizirani u tablice koje sadrže retke i stupce. Svaki stupac opisuje određene podatke u tablici, a svaki red pohranjuje podatke za jedan zapis. Kako bi se osiguralo da je relacijska baza podataka točna i dostupna, tablice moraju biti povezane s pristupnim podacima [5]. Relacijske baze podataka rade za strukturirane podatke s definiranim odnosima koji se mogu organizirati u tabličnom formatu [4]. Dijelovi strukture relacijske baze podataka su:

- **Tablice** – tablica je zbirka povezanih podataka organiziranih u retke i stupce. Svaka tablica ima jedinstveni naziv i sastoji se od jednog ili više stupaca, također poznatih kao polja ili atributi, i jednog ili više redaka, također poznatih kao zapisi. Svaki stupac u tablici ima tip podataka koji definira vrstu podataka koji se mogu pohraniti u tom stupcu.
- **Primarni ključevi** – primarni ključ je stupac ili kombinacija stupaca u tablici koji jednistveno identificira svaki red u toj tablici. Vrijednosti u stupcu primarnog ključa moraju biti jedinstvene i ne mogu biti ništavne (engl. *Null*) vrijednosti. Primarni ključevi koriste se za provođenje integriteta podataka i uspostavljanje odnosa između tablica.
- **Strani ključevi** – strani ključ je stupac ili kombinacija stupaca u jednoj tablici koji se odnosi na primarni ključ u drugoj tablici. Stani ključevi koriste se za uspostavljanje odnosa između tablica i za provedbu referentnog integriteta, koji osigurava da podaci u povezanim tablicama ostanu dosljedni.
- **Odnosi** – odnosi između tablica uspostavljaju se preko primarnih i stranih ključeva. Postoje tri vrste odnosa: jedan naprema jedan, jedan naprema više i više naprema više. U odnosu jedan naprema jedan, svaki redak u jednoj tablici povezan je s točno jednim retkom u drugoj tablici. U odnosu jedan naprema više, svaki redak u jednoj tablici povezan je s jednim ili više redaka u drugoj tablici. U odnosu više naprema više, svaki redak u jednoj tablici može biti povezan s jednim ili više redaka u drugoj tablici i obrnuto. Odnosi više naprema više zahtijevaju treću tablicu, poznatu kao spojna ili povezujuća tablica, za uspostavljanje odnosa.
- **Indeksi** – indeksi se koriste za poboljšanje izvedbe upita baze podataka. Indeks je struktura podataka koja omogućuje brži pristup podacima nego skeniranje cijele tablice. Indeksi se mogu stvoriti na jednom ili više stupaca u tablici i mogu se koristiti za provođenje jedinstvenosti ili za ubrzavanje upita.
- **Prikazi ili pogled** – pogled je virtualna tablica koja se temelji na rezultatu upita baze podataka. Pogledi se mogu koristiti za pojednostavljenje složenih upita i za pružanje pojednostavljenog prikaza podataka korisnicima. Prikazi se također mogu koristiti za

ograničavanje pristupa osjetljivim podacima pružanjem podskupa podataka određenim korisnicima.

Poznavajući sve ove komponente strukture relacijske baze podataka moguće je izraditi bilo koju bazu podataka koja se temelji na podacima koji sadrže ili ih povezuje odnos između njih.

3.2.1. Prednosti i nedostatci

Relacijske baze podataka imaju nekoliko prednosti i nedostataka. Jedna od glavnih prednosti korištenja relacijske baze podataka jest to što se u njoj može jednostavno postavljati upite, što omogućuje korištenje pohranjenih procedura za manipuliranje podacima i osigurava dosljedan dizajn baze podatak. Relacijske baze podataka rade sa strukturiranim podacima, podržavaju *ACID* (engl. *Atomicity, Consistency, Isolation, and Durability*) transakcijsku dosljednost i pružaju fleksibilan način strukturiranja podataka koji nije moguć s drugim tehnologijama baza podataka [6]. Još jedna prednost korištenja relacijske baze podataka a to je veća vjerojatnost da će proizvesti točne i međusobno povezane tablice, budući da koristi primarne i strane ključeve za uspostavljanje relacije [7]. Relacijske baze podataka mogu se koristiti za praćenje inventara, obradu transakcija e-trgovine, upravljanje velikim količinama ključnih informacija o kupcima i više. Po ovome možemo svrstati prednosti kao što su:

- **Strukturiranje podataka** – pružaju strukturirani način pohranjivanja podataka
- **Skalabilnost** – relacijske baze podataka je moguće proširiti
- **Sigurnost** – koriste korisničke uloge i dopuštenja pri čitanju podataka
- **Integritet podataka** – osigurava se točnost podataka u tablicama uz pomoć ključeva
- **Lagano kreiranje** – SQL je standardiziran i jednostavan jezik za naučiti

Međutim, relacijske baze podataka također imaju ograničenja kada je riječ o velikim količinama transakcija ili velikim količinama pohrane podataka, a može se pojaviti i problem brzine [6]. Osim toga, „relacijski model izumljen je godinu dana nako slijetanja na Mjesec“, a nekim programerima softvera možda se neće svidjeti ova vrsta baze podataka jer je to stara tehnologija [8]. Ukratko, relacijske baze podataka su učinkovit sustav za pohranu informacija o odnosima između različitih entiteta. Imaju nekoliko prednosti, uključujući jednostavno postavljanje tablice. Međutim, oni također imaju ograničenja, uključujući probleme s velikom količinom transakcija ili velikom količinom pohrane podataka, te temeljne troškove uključene u njihovu upotrebu. Uzimajući ove stvari u obzir možemo reći da su nedostatci relacijske baze podataka:

- **Performanse** – sporo ako se radi sa velikim količinama podataka
- **Složenost** – složeno postavljanje i održavanje
- **Redudancija** – može doći do redudantnosti podataka jer se podaci često dupliciraju u više tablica što može dovesti do gubitka prostora za pohranu i sporijih upita
- **Ograničena fleksibilnost** – imaju krutu strukturu što može otežati dodavanje novih tipova podataka ili izmjene *sheme* baze podataka
- **Trošak** – mogu biti skupe za licenciranje i održavanje što može biti problem kod malih tvrtki ili organizacija s ograničenim proračunom.

3.3. Primjeri upotrebe relacijske baze podataka

Relacijske baze podataka naširoko se koriste u raznim industrijama i aplikacijama. Na primjer, popis kontakata za ljudske resurse uobičajeni je primjer relacijske baze podataka u kojoj su informacije pohranjene u tablici sa skupom stupaca i redaka. Relacijske baze podataka mogu se koristiti za praćenje inventara, obradu transakcija e-trgovine, upravljanje velikim količinama ključnih informacija o kupcima i više [8]. Koriste se u bankarstvu, zdravstu, obrazovanju, vladi i mnogim drugim industrijama. U bankarstvu se relacijske baze podataka koriste za pohranu podataka o klijentima, povijesti transakcija i stanja računa. U zdravstvu se koriste za pohranu podataka o pacijentima, medicinske dokumentacije i rezultata pretrega. U obrazovanju se koriste za pohranjivanje podataka o studentima, ocjenama i evidenciji prisutnosti [8]. Relacijske baze podataka također se koriste u aplikacijama e-trgovine za pohranu informacija o proizvodu, narudžbi kupaca i pojedinosti o otpremi. Na primjer, Amazon koristi relacijsku bazu podataka za pohranu informacija o proizvodu, narudžbi kupaca i pojedinosti o otpremi. Osim toga, platforme društvenih medija kao što su Facebook i Twitter koriste relacijske baze podataka za pohranu korisničkih informacija, postova i komentara [8]. Relacijske baze podataka također se koriste u upravljanju opskrbnim lancem za praćenje inventara, narudžbi i pošiljaka. Ukratko, relacijske baze podataka koriste se u širokom rasponu aplikacija i industrija za pohranu podataka i upravljanje njima na strukturiran i organiziran način.

Relacijske baze podataka kao i NoSQL baze podataka se koriste u mnogo različitih aplikacija i industrija., a kao i što je prije navedeno te industrije su:

- **Bankarstvo** – banke koriste relacijske baze podataka za pohranjivanje podataka o računima klijenata, povijesti transakcija i drugih financijskih podataka. Relacijske baze

podataka pomažu osigurati sigurnost i točnost financijskih podataka, a bankama omogućuju brzu i učinkovitu obradu transakcija.

- **E-trgovina** – trgovci koriste relacijske baze podataka za pohranjivanje informacija o proizvodima, narudžbi kupaca i pojedinosti o otpremi. Relacijske baze podataka pomažu online trgovcima u upravljanju podacima.
- **Zdravstvo** – zdravstvene organizacije koriste relacijske baze podataka za pohranu podataka o pacijentima, medicinske dokumentacije i drugih zdravstvenih informacija. Relacijske baze podataka pomažu u osiguravanju točnosti i sigurnosti podataka o pacijentima, a zdravstvenim radnicima omogućuju jednostavan pristup i dijeljenje podataka o pacijentima.
- **Proizvodnja** – proizvodne tvrtke koriste relacijske baze podataka za upravljanje zalihama, rasporedima proizvodnje i podacima o opskrbnom lancu. Relacijske baze podataka pomažu proizvođačima optimizirati svoje poslovanje, smanjiti otpad i poboljšati kvalitetu proizvoda
- **Obrazovanje** – obrazovne ustanove koriste relacijske baze podataka za pohranjivanje podataka o studentima, ocjenama i drugim akademskim podacima. Relacijske baze podataka pomažu nastavnicima da prate napredak učenika, analiziraju akademsku izvedbu i pružaju personalizirana iskustva učenja za učenike.

Kao što se vidi po primjerima SQL ili relacijske baze podataka su dosta zastupljene u različitim industrijama svijeta. Trenutno neke od većih kompanija u svijetu koriste SQL tipove baza podataka kako bi pružili svojim korisnicima, a i svojoj tvrtki sve što im je potrebno, a neki od tih kompanija su:

- **Amazon** – koristi relacijske baze podataka za upravljanje informacijama o kupcima, poviješću narudžbi i informacijama o proizvodu. To pomaže Amazon-u pružiti personalizirane preporuke proizvoda, učinkovito obraditi narudžbe i poboljšati cjelokupno korisničko iskustvo.
- **Google** – Koristi relacijske baze podataka za pohranu i upravljanje korisničkim podacima, poviješću pretraživanja i informacijama o oglašavanju. Relacijske baze podataka pomažu Google-u da isporuči relevantne rezultate pretraživanja, ciljane oglase i prilagođeni sadržaj korisnicima.
- **Facebook** – koristi relacijske baze podataka za pohranjivanje i upravljanje korisničkim podacima, uključujući informacije o profilu, prijateljske veze i sadržaj koji generiraju

korisnici. Relacijske baze podataka pomažu Facebook-u pružiti personalizirano korisničko iskustvo, olakšati društvene interakcije i zaštititi korisničke podatke.

- **Microsoft** – koristi relacijske baze podataka u raznim proizvodima i uslugama, uključujući SQL poslužitelj, Azure SQL bazu podataka i Dynamics 365. Ove relacijske baze podataka pomažu Microsoft-u upravljati i analizirati podatke u raličitim poslovnim funkcijama, uključujući prodaju, marketing, korisničku službu i financije.
- **Oracle** – Oracle je vodeći pružatelj sustava za upravljanje relacijskim bazama podataka, uključujući Oracle Database i MySQL.
- **Airbnb** – koristi relacijske baze podataka za upravljanje oglasima, rezervacijama i informacijama o korisnicima. Relacijske baze podataka pomažu spojiti goste s domaćinima i obraditi rezervacije.

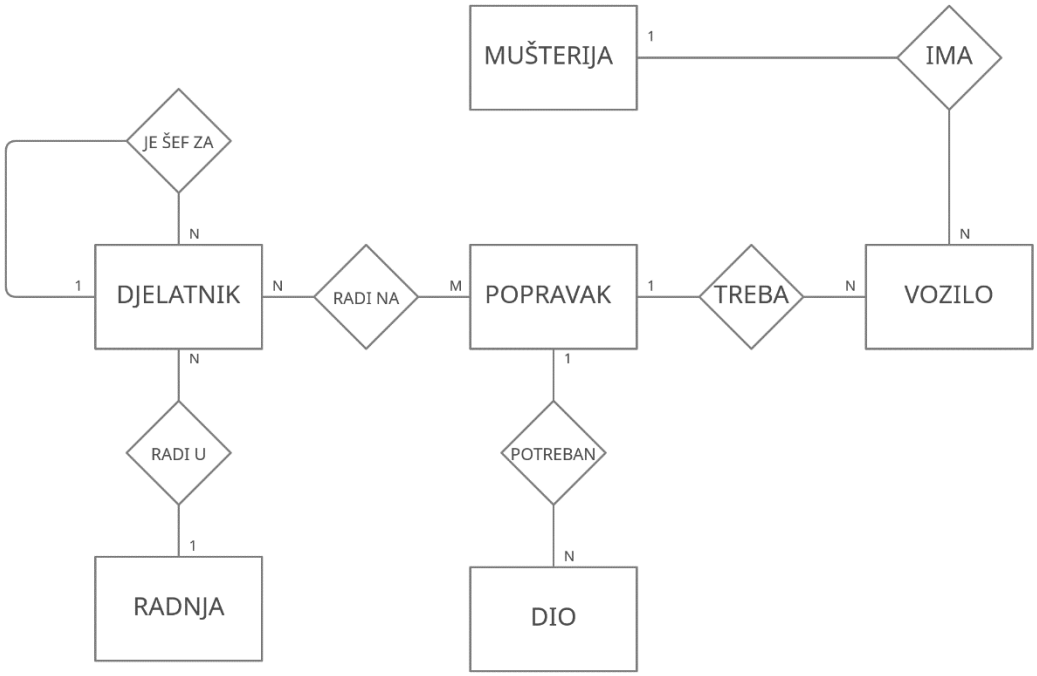
3.4. Izrađeni primjer relacijske baze podataka

Kao primjer relacijske baze podataka izrađena je jednostavna baza podataka vulkanizerske radnje koja će se kasnije ujedno koristiti kao primjer usporedbe relacijske i NoSQL baze podataka. U nastavku će biti prikazan E/R dijagram[4] (engl. *Entity-Relationship dijagram*) dijagram koji će pobliže objasniti izgled baze podataka i relacija unutar njega te će se ujedno prikazati koje tipove podataka svaki entitet unutar baze podataka sadrži te kako se sve to može isprogramirati u funkcionalnu bazu podataka.

3.4.1. Konceptualno oblikovanje

Na osnovu analize specifikacije dobiven je E/R dijagram te su prikazani entiteti, veze između entiteta i funkcionalnosti veza na slici 3.1. Svaki entitet unutar baze podataka ima svoje atribute te neki od tih atributa mogu biti kandidati za ključ entiteta.

[4] E/R dijagram - dijagram koji se koristi u području baza podataka kako bi se prikazale veze između entiteta, atributa i veza



Slika 3.1. *E/R dijagram za bazu podataka vulkanizerske radnje*

3.4.2. Logičko oblikovanje

Prevođenjem E/R dijagrama (Slika 3.1) u relacijski model dobivena je sljedeća relacijska shema baze podataka vulkanizerske radnje, odnosno skup relacija prikazanih na slici 3.2. Nazivi kao što su *MUŠTERIJA* ili *DJELATNIK* su entiteti, a sve unutar zagrada su atributi. Podcrtani atributi su primarni ključevi tog entiteta, a kao što se može i vidjeti na slici 3.2. neki entiteti sadrže i strane ključeve ili primarne ključeve drugih entiteta. Atributi kao što su *OIB* ili *IME* su unutar baze podataka prikazani kao nizovi znakova dok je *DATUM ROĐENJA* prikazano kao datum.

MUŠTERIJA (OIB, IME, PREZIME, DATUM ROĐENJA, POŠTANSKI BROJ, IME GRADA, ADRESA)

DJELATNIK (ID DJELATNIKA, ID RADNJE, ID ŠEFA, IME, PREZIME, DATUM ROĐENJA, PLAĆA)

POPRAVAK (ID POPRAVKA, ID DIJELA, DATUM POČETKA, CIJENA, DATUM ZAVRŠETKA)

RADI NA (ID POPRAVKA, ID DJELATNIKA)

RADNJA (ID RADNJE, NAZIV, BROJ DJELATNIKA)

DIO (ID DIJELA, NAZIV, CIJENA)

VOZILO (REGISTARSKA OZNAKA, ID MUŠTERIJE, NAZIV PROIZVOĐAČA, BROJ MJESTA, MARKA VOZILA)

Slika 3.2. *Relacijska shema baze podataka vulkanizerske radnje*

3.4.3. Fizičko oblikovanje

Fizička *shema* (SQL naredbe za implementaciju baze podataka) baze podataka Vulkanizerske radnje je prikazana na primjerima koda 3.1 i 3.2., te je dobivena na osnovu relacijske sheme baze podataka. SQL naredbe su pisane unutar LINQPad 5[5].

[5] LINQPad 5 - razvojno okruženje za .NET programiranje koje omogućava programerima da brzo i jednostavno testiraju i provjere različite upite na podacima, LINQ izraze.

```
CREATE TABLE Mušterija
(
   OIB char(11) PRIMARY KEY,
   Ime varchar(30),
   Prezime varchar(30),
   Datum_rođenja datetime,
   Poštanski_broj int,
   Ime_grada varchar(50),
   Adresa varchar(100)
);

CREATE TABLE Vozilo
(
   Registarska_oznaka varchar(10) PRIMARY KEY,
   ID_mušterije char(11),
   Naziv_proizvođača varchar(30),
   Marka_vozila varchar(30),
   Broj_mjesta int NOT NULL,

   CONSTRAINT chk_vozilo CHECK(Broj_mjesta > 1),
   CONSTRAINT vozilo_fk FOREIGN KEY (ID_mušterije) REFERENCES
Mušterija(OIB) ON UPDATE CASCADE,
);


CREATE TABLE Djelatnik
(
   ID char(5) PRIMARY KEY,
   ID_radnje char(5),
   ID_šefa char(5),
   Ime varchar(30),
   Prezime varchar(30),
   Datum_rođenja datetime,
   Plaća decimal (7, 2),

   CONSTRAINT chk_placa CHECK(Plaća > 3000),
   CONSTRAINT djelatnik_fk1 FOREIGN KEY (ID_radnje) REFERENCES Radnja (ID)
ON UPDATE CASCADE,
   CONSTRAINT djelatnik_fk2 FOREIGN KEY (ID_šefa) REFERENCES Djelatnik (ID)
);
```

Primjer koda 3.1. *Entiteti Mušterija, Vozilo i Djelatnik*

Kao što se može vidjeti naredba *CREATE TABLE* se koristi za pravljenje tablice naziva *MUŠTERIJA* što predstavlja entitet. Taj entitet ima svoje atribute koje su napisane unutar te naredbe te su im određeni tipovi veličina (*char*, *varchar*, *int* itd.) i primarni ključ je označen kao *PRIMARY KEY* dok su strani ključevi označeni kao *FOREIGN KEY* koji se mora pozvati na entitet od kojeg je on primarni ključ.

```
CREATE TABLE Radnja
(
   ID char(5) PRIMARY KEY,
   Naziv varchar(30),

   CONSTRAINT chk_Broj_djelatnika CHECK (Broj_djelatnika > 0)
);

CREATE TABLE Dio
(
   ID char(5) PRIMARY KEY,
ID_Popravka CHAR(5),
   Naziv char(30),
   Cijena int,

CONSTRAINT popravak_fk FOREIGN KEY(ID_Popravka) REFERENCES Popravak(ID),
   CONSTRAINT chk_cijena CHECK (Cijena > 0)
);
CREATE TABLE Popravak
(
   ID char(5) PRIMARY KEY,
   ID_dijela char(5),
   ID_vozila varchar(10),
   Datum_pocetka datetime,
   Datum_zavrsetka datetime,
   Cijena int,

   CONSTRAINT chk_cijenapopravak CHECK(cijena > 0),
   CONSTRAINT popravak_fk_vozilo FOREIGN KEY(ID_vozila) REFERENCES
Vozilo(Registarska_oznaka) ON UPDATE CASCADE,
);
CREATE TABLE RADI_NA
(
   ID_Popravka char(5),
   ID_Djelatnika char(5),

   CONSTRAINT radina_pk PRIMARY KEY (ID_Popravka, ID_Djelatnika),
   CONSTRAINT radina_fk_popravak FOREIGN KEY(ID_Popravka) REFERENCES
Popravak(ID) ON UPDATE CASCADE,
   CONSTRAINT radina_fk_djelatnik FOREIGN KEY(ID_Djelatnika) REFERENCES
Djelatnik(ID) ON UPDATE CASCADE
);
```

Primjer koda 3.2. *Entiteti Radnja, Dio, Popravak i Radi_na*

Kod kreiranja baze podataka uz pomoć naredbi, entiteta i atributa moguće je ujedno i napraviti funkcije koje mogu služiti za vraćanje ukupne cijene popravka, procedure za ispis detalja o popravkama zadane mušterije ili se mogu kreirati pogledi na sve djelatnike i šefa djelatnika. Funkcije, procedure i pogledi se mogu kreirati po želji i potrebama korisnika koji

koristi bazu podataka. Kako bi jedna funkcija izgledala za primjer je prikazano na primjeru koda 3.3.

```
CREATE FUNCTION UKUPNA_CIJENA_POPRAVKA(@SIFRA_POPRAVKA Char(5)) RETURNS INT
AS
BEGIN
DECLARE @SUM_CIJENA INT
SELECT @SUM_CIJENA = SUM(D.Cijena)
FROM Dio D, Popravak P
WHERE P.ID = D.ID_Popravka AND
   P.ID = @SIFRA_POPRAVKA
RETURN @SUM_CIJENA
END
```

Primjer koda 3.3. *Funkcija za vraćanje ukupne cijene popravka*

Uz pomoć naredbe *CREATE FUNCTION* kreiramo funkciju koja vraća ukupnu cijenu popravka (cijeli broj). Kao što se može vidjeti u funkciji mi iz entiteta *DIO* i entiteta *POPRAVAK* uzimamo potrebne atribute i uz pomoć njih izračunavamo ukupnu cijenu popravka koja je jednaka sumi cijene svako dijela potrebnog za popravak. Funkciji se predaje Sifra popravka kako bi znali za koji popravak računamo ukupnu cijenu jer unutar baze podataka možemo imati više popravaka koji se odrađuju.

3.4.4. Unos podataka

Kako bi unijeli podatke u bazu podataka moramo imati podatke za sve entitete, moramo imati podatke za svaki atribut unutar nekog entiteta i tako za svaki entitet. Za primjer su uzeti podaci o mušterijama. Podatke možemo zapisati u tekstualnu datoteku (Slika 3.3) i uz pomoć izrađene funkcije ili samo nekoliko naredbi ih možemo povući iz datoteke i ubaciti u bazu podataka.

```
0000000000,Luka,Vukic,2001-06-13 00:00:00,31000,Osijek,Kajzestrase 69
0000000001,Petar,Peric,2002-04-10 00:00:00,32015,Tenja,Svete Ane 3A
0000000002,Ivica,Ivcic,2003-11-01 00:00:00,31256,Retfala,Svete Ane 4a
0000000003,Matej,Matijevic,2004-02-12 00:00:00,69692,Vinkovci,Petrova 30
0000000004,Marko,Markovic,2002-01-22 00:00:00,12345,Kresinci,Majnina 10B
0000000005,Petar,Zrinski,2000-12-21 00:00:00,10000,Zagreb,Sestrina 69
0000000006,Ante,Pavelic,1988-12-15 00:00:00,96545,Nustar,Bratova 21
0000000007,Leonard,Kraso,1980-09-20 00:00:00,77777,Cezar,Mamina 22
0000000008,Josip,Crnjic,2002-08-02 00:00:00,63254,Rim,Tatina 6A
0000000009,Darko,Bijelic,2000-06-17 00:00:00,13265,Egipt,Kajzestrase 20
```

Slika 3.3. *Podaci o mušterijama*

Nakon što izradimo *shemu* baze podataka sa svim entitetima, atributima, privatnim, stranim ključevima, funkcijama i procedurama uz pomoć alata za izradu baze podataka poput

LINQPad-a ili nekog drugog i nakon što sa uspješnošću ubacimo podatke u bazu podataka i potvrdimo da je sve funkcionalno možemo reći da imamo funkcionalnu relacijsku bazu podataka.

4. NOSQL BAZE PODATAKA

U narednom poglavlju opisat će se NoSQL baza podataka, njihova struktura te će se objasniti prednosti i nedostatci baze podataka. Također navest će se primjeri upotrebe NoSQL baze podataka i nabrojati te opisati koje velike kompanije trenutno koriste NoSQL baze podataka i za što ih koriste.

4.1. Definicija NoSQL baze podataka

NoSQL baze podataka su nerelacijske baze podataka koje pohranjuju i dohvaćaju podatke u netabularnom formatu u usporedbi s relacijskim tablicama. Oni pružaju mehanizam za pohranjivanje i dohvaćanje podataka koji je modeliran na drugačiji način od tabličnih odnosa koji se koriste u relacijskim bazama podataka [9]. NoSQL baze podataka dolaze u različitim vrstama na temelju njihovog modela podataka, uključujući dokument, ključ-vrijednost, široki stupac i grafikon što će više biti pojašnjeno u poglavlju 4.2.. Baze podataka dokumenata pohranjuju polustrukturirane podatke i opise tih podataka u formatu dokumenta, dok baze podataka ključ-vrijednost pohranjuju podatke kao zbirku parova ključ-vrijednost. Baze podataka sa širokim stupcima pohranjuju podatke u tablicama s redovima i stupcima, ali svaki redak može imati drugačiji skup stupaca, a baze podataka s grafikonima pohranjuju podatke u čvorovima i rubovima. NoSQL baze podataka koriste se za pohranu i dohvaćanje ogromnih količina podataka koji zahtijevaju ogromne potrebe za pohranom [9]. Ponekad im se daje prednost u odnosu na relacijske ili SQL baze podataka zbog njihove sposobnosti pohranjivanja strukturiranih, nestrukturiranih, polustrukturiranih i polimofrnih podataka kao što su objave na društvenim mrežama, podaci senzora ili datoteke dnevnika koji se ne uklapaju dobro u krutu strukturu tradicionalne relacijske baze podataka. NoSQL baze podataka pružaju niz pogodnosti uključujući fleksibilne modele podataka, horizontalno skaliranje, jako brze upite i jednostavnost korištenja za pogramere [9]. Sve se više koriste u velikim podacima i web aplikacijama u stvarnom vremenu. Međutim, nedostatak standardiziranog jezika za upite može otežati izvođenje složenih upita ili spajanje podataka iz različitih zbirki ili tablica. Izraz NoSQL izvorno se mogao uzeti za riječ – to jest, SQL se nije koristio kao API[6] za pristup podacima. Međutim, sveprisutnost i koristnost SQL-a uzrokovala je da mnoge NoSQL baze podataka dodaju podršku za SQL. Danas je općenito prihvaćeno da NoSQL označava *ne samo SQL*.

[6] API – Programsko sučelje aplikacije (engl. *Application Programming Interface*)

4.2. Opis strukture NoSQL baze podataka

NoSQL baze podataka su netabularne baze podataka koje pohranjuju podatke drugačije od relacijskih tablica. Dolaze u različitim vrstama na temelju njihovog podatkovnog modela, uključujući dokument, ključ-vrijednost, široki stupac i grafikon. NoSQL baze podataka su nerelacijske, distribuirane, fleksibilne i skalabilne i ne zahtijevaju fiksnu *shemu* [10]. NoSQL baze podataka dizajnirane su za specifične slučajeve upotrebe i jednostavnije su za korištenje od relacijskih baza podataka za zahtjeve oblaka, mobilnih uređaja, društvenih medija i velikih podataka [10]. Struktura NoSQL baza podataka razlikuje se ovisno o vrsti baze podataka. Baze podataka dokumenata pohranjuju polustrukturirane podatke i opise tih podataka u formatu dokumenata, dok baze podataka ključ-vrijednost pohranjuju podatke kao zbirku parova ključ-vrijednost. Baze podataka sa širokim stupcima pohranjuju podatke u tablice s recima i stupcima, ali stupci se mogu razlikovati od retka do retka, a baze podataka s grafikonima pohranjuju podatke u čvorovima i rubovima, koji predstavljaju odnose između podatkovnih točaka. NoSQL baze podataka pružaju niz prednosti, uključujući fleksibilne modele podataka, horizontalno skaliranje, brze upite i jednostavnost korištenja. Uglavnom se koriste za rad s velikim podacima koji nisu nužno strukturirani ili povezani, a dizajnirani su za rukovanje velikim količinama podataka i velikim korisničkim opterećenjem. NoSQL baze podataka nadaleko su poznate po svojoj jednostavnosti razvoja, funkcionalnosti i performansi.

Ove baze podataka razvijene su kao odgovor na ograničenja tradicionalnih relacijskih baza podataka kada se radi o upravljanju velikim, nestrukturiranim ili polustrukturiranim podacima. Struktura NoSQL baza podataka razlikuje se od strukture relacijskih baza podtaka na sljedeće načine:

- **Podatkovni model** – NoSQL baze podatka podržavaju različite vrste podatkovnih modela, od kojih svaki drugačije pohranjuje podatke. Pohrane ključ-vrijednost pohranjuju podatke kao jednostavan par ključ-vrijednost, dok pohrane orijentirane na dokument pohranjuju podatke u *JSON* ili *XML* dokumente. Baze podataka grafova pogranjuju podatke kao čvorove i rubove, a pohrane obitelji stupaca pohranjuju podatke u tablice sa stupcima koji se mogu dinamički dodavati ili uklanjati.
- ***Shema*** - za razliku od relacijskih baza podataka, NoSQL baze podataka nemaju fiksnu shemu. To znači da se struktura podataka može dinamički mijenjati kako se dodaju novi podaci, bez potrebe za mijenjanjem cijele *sheme* baze podataka.

- **Skalabilnost** – NoSQL baze podataka dizajnirane su za horizontalno skaliranje, što znači da se novi poslužitelji mogu dodati sustavu kako bi se povećala njegova procesorska snaga i kapacitet pohrane. To je u suporotnosti s relacijskim bazama podataka, koje se obično skaliraju vertikalno, zahtijevajući veći i snažniji *hardver* za rukovanje povećanim radnim opterećenjem.
- **Dostupnost** – NoSQL baze podataka dizajnirane su da budu visoko dostupne i tolerantne na greške. Podaci se često repliciraju na više poslužitelja u skupini, osiguravajući da baza podataka ostane dostupna čak i ako jedan ili više poslužitelja prekine sa radom.
- **Dosljednost** – NoSQL baze podatak daju prednost dostupnosti i toleranciji particije u odnosu na dosljednost. To znači da u slučaju mrežne particije ili drugog kvara, baza podataka klijentima može vratiti nedosljedne ili zastarjele podatke. To je u suprotnosti s relacijskim bazama podataka, koje daju prednost dosljenosti nad dostupnošću i tolerancijom particije..

4.2.1. Prednosti i nedostatci

NoSQL baze podataka imaju nekoliko prednosti i nedostataka. Jedna od glavnih prednosti NoSQL baza podataka je njihov fleksibilni model podataka, koji im omogućuje pohranjivanje i kombiniranje bilo koje vrste podataka, strukturiranih i nestrukturiranih. Također mogu dimanički ažurirati *shemu* kako bi se razvijala s promjenjivim zahtjevima bez ikakvih prekida ili zastoja aplikacije [11]. NoSQL baze podataka dizajnirane su za rukovanje velikim količinama nestrukturiranih ili polustrukturiranih podataka i bolje su prilagođene zahtjevima za oblak, mobilne uređaje, društvene medije i velike podatke [12]. Također su skalabilnije i mogu obraditi više podataka od tradicionalnih relacijskih baza podataka [13]. NoSQL baze podataka također su jednostavnije za korištenje i zahtijevaju manje održavanja od relacijskih baza podataka. Međutim, NoSQL baze podataka imaju neke nedostatke. Nisu učinkoviti u održavanju atomičnosti ili točnosti, čime se smanjuje puzdanost informacija. NoSQL baze podataka također su manje *zrele* od relacijskih baza podataka i imaju manju podršku [13]. Osim toga, NoSQL bazama podataka može biti teže postavljati upite i može zahtijevati složenije modeliranje podataka. Može se reći da NoSQL baze podataka imaju mnogo prednosti, ali i mnogo nedostataka koji im omogućuju da u jednu ruku budu dobre za neke stvari, a loše za druge. U nastavku se može pronaći grupirane prednosti i nedostatke NoSQL baza podataka:

- **Skalabilnost** – NoSQL baze podataka vrlo su skalabilne, što znači da mogu obraditi ogromne količine podataka i prometa. Mogu se horizontalno skalirati dodavanjem više

poslužitelja u skupini, omogučujući besprijekorno širenje bez potrebe za značajnim promjenama u arhitekturi baze podataka.

- **Fleksibilnost** – NoSQL baze podataka mogu pohranjivati različite vrste podataka, uključujući strukturirane, polustrukturirane i nestrukturirane podatke. Ova fleksibilnost razvojnim pogramerima omogućuje rad s modelima podataka koji bolje odgovaraju njihovim potrebama, bez potrebe da se prilagođavaju krutoj *shemi*.
- **Dostupnost** – NoSQL baze podataka dizajnirane su da budu visoko dostupne s podacima koji se repliciraju na više poslužitelja u skupinu. Ovo osigurava da baza podataka ostaje dostupna čak i u slučaju kvara poslužitelja ili prekida mreže.
- **Performanse** – NoSQL baze podataka dizajnirane su za pružanje brzih operacija čitanja i pisanja, što ih čini idealnim za aplikacije koje zahtijevaju nisku *latenciju* i visoku propusnost. Mogu učinkovito obrađivati složene upite i transakcije, omogučujući obradu podataka u stvarnom vremenu.
- **Nedostatak usklađenosti s *ACID-om*** – NoSQL baze podataka često žrtvuju stroga jamstva dosljednosti koje nude relacijske baze podataka u korist skalabilnosti i dostupnosti. To znači da možda nisu prikladni za aplikacije koje zahtijevaju jaku dosljednost podataka, kao što su financijski sustavi ili aplikacije.
- **Ograničene mogućnosti upita** – NoSQL baze podataka nude ograničene mogućnosti upita u usporedbi s relacijskim bazama podataka. Zbog toga može biti izazovno izvršavanje složenih upita ili spajanje podataka i više izvora.
- **Krivulja učenja** – NoSQL baze podataka, zahtijevaju drugačiji pristup modeliranju podataka i postavljanju upita, što može zahtijevati značajna ulaganja u učenje i obuku progamera
- **Sigurnost** – NoSQL baze podataka mogu biti ranjive na sigurnosne prijetnje kao što su napadi ubrizgavanjem, budući da im često nedostaju *robusne* sigurnosne značajke koje nude relacijske baze podataka.
- **Nedostatak standardizacije** – NoSQL baze podataka nisu standardizirane, što znači da različite baze podataka mogu implementirati različite modele podataka i jezike upita. Zbog toga može biti izazovna migracija podataka između različitih NoSQL baza podataka ili njihova integracija s drugim sustavima.

4.3. Primjeri upotrebe NoSQL baze podataka

NoSQL baze podataka koriste se u raznim industrijama i aplikacijama. Uglavnom se kroiste za rad s velikim podacima koji nisu nužno strukturirani ili povezani, što ih čini idealnim za aplikacije velikih podataka. NoSQL baze podataka prikladnije su za pohranjivanje i modeliranje strukturiranih, polustrukturiranih i nestrukturiranih podataka u jendoj bazi podaka i često pohranjuju ogromne količine podataka. Neke od popularnih NoSQL baza podataka i slučajevi njihove upotrebe uključuju MongoDB, koji se obično koristi u okruženjima gdje je potrebna fleksibilnost s velikim nestrukturiranim podacima sa *shemama* koje se stalno mijenjaju [14]. Cassandra se koristi za obradu podataka u stvarnom vremenu, dok se Neo4J koristi za graf baze podataka, a Redis se koristi za predmemoriju i čekanje poruka [15]. NoSQL baze podataka također se koriste u sustavima za upravljanje sadržajem, upravljanju podacima o proizvodu, upravljanju odnosima s kupcima itd. NoSQL baze podataka bolji su izbor od relacijskih baza podataka kada je potrebno pohraniti velike količine nestrukturiranih podataka s promjenjivim *shemama*, a obično imaju svojstva horizontalnog skaliranja koja im omogućuje pohranu i obranu velikih količina podataka [16].

U današnje vrijeme NoSQL tipovi baza podataka se sve više koriste u industriji i različitim aplikacijama.

- **Društveni mediji** – platforme društvenih medija kao što su Facebook i Twitter koriste NoSQL baze podataka za pohranu korisničkih profila, postova i drugih društvenih podataka. NoSQL baze podataka omogućuju ovim platformama rukovanje korisnika i velikom količinom podataka uz pružanje brzih operacija čitanja i pisanja.
- **Zdravstvo** – zdravstvene aplikacije koriste NoSQL baze podataka za pohranjivanje zapisa pacijenata, medicinskih slika i podataka senzora. NoSQL baze podataka omogućuju pružateljima zdravstvenih usluga pohranjivanje i obradu velikih količina podataka, istovremeno osiguravajući privatnost i sigurnost podataka.
- **Igre** – platforme za online igranje koriste NoSQL baze podataka za pohranjivanje podataka o igračima, statistike igre i rangiranja na ljestvici s najboljim rezultatima. NoSQL baze podataka nude brze operacije čitanja i pisanja, visoku dostupnost i skalabilnost.
- **E-trgovine** – NoSQL baze podataka koriste se u aplikacijama e-trgovine za rukovanje velikim količinama transakcija kupaca, informacijama o proizvodima i korisničkim

profilima. Ove baze podataka pružaju visoku dostupnost i skalabilnost kako bi podržale zahtjeve platformi za online kupnju.

- **Analitika** – NoSQL baze podataka koriste se u aplikacijama za analizu podataka za pohranu i obradu velikih količina nestrukturiranih podataka kao što su datoteke dnevnika, podaci senzora i tokovi klikova. Ove baze podataka nude visoku skalabilnost i fleksibilnost za rukovanje nepredvidivom prirodom velikih podataka.
- **Internet stvari** (engl. *Internet of Things*) – *IoT* generiraju golemu količinu podataka koje je potrebno pohraniti i obraditi u stvarnom vremenu. NoSQL baze podataka idealne su za *IoT* aplikacije jer mogu rukovati velikim nestrukturiranim podacima i pružiti visoku skalabilnost i dostupnost.

Kao što se može vidjeti u primjerima mnogo različitih industrija koristi NoSQL tipove baza podataka. U današnje vrijeme mnogo velikih kompanija je bazirano na NoSQL bazama podataka kao što su:

- **Amazon** – koristi NoSQL baze podataka kao što je DynamoDB za pohranjivanje podataka kataloga proizvoda, profila kupaca i povijesti kupovine. Ove baze pdoataka pružaju visoku skalabilnost i dostupnost za rukovanje golemim prometom i količinom podataka kao e-trgovine.
- **Uber** – Uber koristi NoSQL baze podataka kao što je MongoDB za pohranu podataka o vožnji i vozaču, korisničkih profila i povijesti transakcija. Ove baze podataka nude visoku skalabilnost i dostupnost kako bi podržale Uber-ove globalne operacije i obradile milijune zahtjeva dnevno.
- **Netflix** – koristi NoSQL baze podataka kao što je Cassandra za pohranu korisničkih profila, video metapodataka i povijesti gledanja. Ove baze podataka omogućuju Netflix-u pružanje personaliziranih preporuka koristnicima i rješavanje velike potražnje za uslugama.
- **Airbnb** – koristi NoSQL baze podataka za pohranu podataka domaćina i gostiju, informacija o rezervaciji i upita za pretraživanje. Ove baze podataka pružaju visoku skalabilnost i dostupnost za podršku Airbnb-ovim globalnim operacijama i rješavanju nepredvidive prirode potražnje za putovanjima.
- **LinkedIn** – koristi NoSQL baze podataka kao što je Apache Kafka i Apache Cassandra za obradu poruka i analitiku podataka. Ove baze podataka pružaju visoku propusnost,

nisku *latenciju* i mogućnosti obrade podataka u stvarnom vremenu, što LinkedIn-u omogućuje besprijekorno korisničko iskustvo i dragocjene uvide korisnicima.

Ukratko, današnje kompanije koriste NoSQL baze podataka zbog njihove skalabilnost, fleksibilnosti i brzine. NoSQL tipovi baza podataka mogu rukovati sa velikim nestrukturiranim količinama podataka te pružaju brze operacije zapisa i čitanja podataka u stvarnom vremenu.

4.1. Izrađeni primjer NoSQL baze podataka

Za primjer NoSQL baze podataka izrađena je baza podataka bazirana na dokument tipu NoSQL baze podataka. Dokument tip NoSQL baze podataka je izrađen na sliku ili primjer relacijske baze podataka iz odjeljka 3.4. Baza podataka se bazira na automehaničarskoj radnji i izrađena je u C#[7] programskom jeziku pisan unutar *Microsoft Visual Studio-a.*[8]

4.1.1. Fizičko oblikovanje

Na sljedećim primjerima koda se može vidjeti izrađena NoSQL baza podataka dokument tipa uz pomoć programskog jezika C# i uz pomoć imenskog prostora (engl. *Namespace*) MongoDB.Bson za bilješku atributa kojeg je potrebno dodatno instalirati unutar Visual Studio-a. Entiteti su prikazani kao klase a atributi svakog entiteta ili klase su prikazani kao atributi unutar klase.

[7] C# - naziv za objektno orijentirani programski jezik korišten za izradu brojnih različitih programa i aplikacija: mobilne aplikacije, aplikacije za stolna računala, usluge temeljene na oblaku, web stranice, poslovni softver i igre

[8] Microsoft Visual Studio - integrirano razvojno okruženje tvrtke Microsoft. Koristi se za razvoj računalnih programa uključujući web stranice, web aplikacije, web usluge i mobilne aplikacije.

```
using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;

[BsonIgnoreExtraElements]
public class Mušterija
{
    [BsonId]
    public string OIB { get; set; }
    [BsonElement("Ime")]
    public string Ime { get; set; }
    [BsonElement("Prezime")]
    public string Prezime { get; set; }
    [BsonElement("Datum_rođenja")]
    public DateTime Datum_rođenja { get; set; }
    [BsonElement("Poštanski_broj")]
    public int Poštanski_broj { get; set; }
    [BsonElement("Ime_grada")]
    public string Ime_grada { get; set; }
    [BsonElement("Adresa")]
    public string Adresa { get; set; }
}
[BsonIgnoreExtraElements]
public class Vozilo
{
    [BsonId]
    public string Registarska_oznaka { get; set; }
    [BsonElement("ID_mušterije")]
    public string ID_mušterije { get; set; }
    [BsonElement("Naziv_proizvođača")]
    public string Naziv_proizvođača { get; set; }
    [BsonElement("Marka_vozila")]
    public string Marka_vozila { get; set; }
    [BsonElement("Broj_mjesta")]
    public int Broj_mjesta { get; set; }
    [BsonIgnoreIfNull]
    [BsonElement("chk_vozilo")]
    public int? chk_vozilo { get; set; }
    [BsonIgnoreIfNull]
    [BsonElement("vozilo_fk")]
    public Mušterija vozilo_fk { get; set; }
}
```

Primjer koda 4.1. *Klasa Mušterija i Vozilo*

```
[BsonIgnoreExtraElements]
public class Djelatnik
{
    [BsonId]
    public string ID { get; set; }
    [BsonElement("ID_radnje")]
    public string ID_radnje { get; set; }
    [BsonElement("ID_šefa")]
    public string ID_šefa { get; set; }
    [BsonElement("Ime")]
    public string Ime { get; set; }
    [BsonElement("Prezime")]
    public string Prezime { get; set; }
    [BsonElement("Datum_rođenja")]
    public DateTime Datum_rođenja { get; set; }
    [BsonElement("Plaća")]
    public decimal Plaća { get; set; }
    [BsonIgnoreIfNull]
    [BsonElement("chk_placa")]
    public decimal? chk_placa { get; set; }
    [BsonIgnoreIfNull]
    [BsonElement("djelatnik_fk1")]
    public Radnja djelatnik_fk1 { get; set; }
    [BsonIgnoreIfNull]
    [BsonElement("djelatnik_fk2")]
    public Djelatnik djelatnik_fk2 { get; set; }
}
[BsonIgnoreExtraElements]
public class Radnja
{
    [BsonId]
    public string ID { get; set; }
    [BsonElement("Naziv")]
    public string Naziv { get; set; }
}

[BsonIgnoreExtraElements]
public class Dio
{
    [BsonId]
    public string ID { get; set; }
    [BsonElement("ID_Popravka")]
    public string ID_Popravka { get; set; }
    [BsonElement("Naziv")]
    public string Naziv { get; set; }
    [BsonElement("Cijena")]
    public int Cijena { get; set; }
    [BsonIgnoreIfNull]
    [BsonElement("popravak_fk")]
    public Popravak popravak_fk { get; set; }
    [BsonIgnoreIfNull]
    [BsonElement("chk_cijena")]
    public int? chk_cijena { get; set; }
}
```

Primjer koda 4.2. *Klasa Djelatnik, Radnja i Dio*

```
[BsonIgnoreExtraElements]
public class RADI_NA
{
    [BsonElement("ID_Popravka")]
    public string ID_Popravka { get; set; }
    [BsonElement("ID_Djelatnika")]
    public string ID_Djelatnika { get; set; }
    [BsonIgnoreIfNull]
    [BsonElement("radina_pk")]
    public RADI_NA radina_pk { get; set; }
    [BsonIgnoreIfNull]
    [BsonElement("radina_fk_popravak")]
    public Popravak radina_fk_popravak { get; set; }
    [BsonIgnoreIfNull]
    [BsonElement("radina_fk_djelatnik")]
    public Djelatnik radina_fk_djelatnik { get; set; }
}
```

Primjer koda 4.3. *Klasa RADI_NA*

Kao što se i može prepoznati, svaka klasa predstavlja entitet iz relacijske baze podataka, a svaki atribut svake klase predstavlja atribute entiteta iz relacijske baze podataka. Ono što je potrebno uočiti je da unutar NoSQL baza podataka kao što su recimo MongoDB koncept stranih i privatnih ključeva je drugačiji nego kod relacijskih baza podataka. U danim C# klasama odnosi stranog ključa predstavljeni su korištenjem referenci umjesto eksplicitnog definiranja stranih ključeva ili ograničenja, odnosi između entiteta ili klasa su uspostavljeni putem referenci na objekte klasa. Na primjer unutar klase Vozilo, atribut ID_mušterije se koristi kao referenca na objekt klase Mušterija.

5. SKALABILNOST RELACIJSKIH I NOSQL BAZA PODATAKA

Skalabilnost baza podataka je ključna za osiguranje učinkovitog i pouzdanog rada aplikacija koje koriste baze podataka. Relacijske i NoSQL baze podataka su dvije glavne vrste baza podataka koje se koriste u različitim scenarijima. Ovaj dio teme usredotočuje se na skalabilnost ovih dviju vrsta baza podataka, s posebnim naglaskom na vertikalnu i horizontalnu skalabilnost. Također, bit će prikazani primjeri skalabilnosti relacijskih i NoSQL baza podataka, te će se analizirati prednosti i nedostatci upotrebe oba tipa baza podataka. Razumijevanje razlika između relacijskih i NoSQL baza podataka, te njihove skalabilnosti, može biti od velike koristi pri odabiru baze podataka koja najbolje odgovara potrebama aplikacije.

5.1. Vrste skalabilnosti baze podataka

Kada govorimo o bazama podataka i vrstama njihove skalabilnosti razlikujemo tri osnovne podjele skalabilnosti, a to su vertikalna skalabilnost, horizontalna skalabilnost i distribuirana skalabilnost. U nastavku se detaljnije pojašnjava razlika vertikalne i horizontalne skalabilnost.

5.1.1. Vertikalna skalabilnost

Vertikalna skalabilnost u bazama podataka odnosi se na mogućnost povećanja kapaciteta jednog stroja dodavanjem više resursa istom logičkom poslužitelju [15]. To uključuje nadogradnju hardvera poslužitelja, kao što je dodavanje više RAM-a (engl. *Random Access Memory*), CPU-a (engl. *Central Processing Unit*) ili pohrane, kako bi se povećala njegova procesorska snaga i kapacitet pohrane. Vertikalno skaliranje relativno je jednostavno za administraciju i može biti jeftinije od horizontalnog skaliranja, koje uključuje dodavanje više poslužitelja za rad na jednom radnom opterećenju [18]. Međutim, okomito skaliranje ima svoja ograničenja, budući da postoji ograničenje koliko se jedan stroj može nadograditi i možda neće moći podnijeti ekstremno velika radna opterećenja [17]. Neki od prednosti i nedostataka vertikalne skalabilnosti su:

- **Pojednostavljena arhitetktura** – svi se podaci pohranjuju na jednom poslužitelju što olakšava upravljanje
- **Isplativost** – nadogradnja hardvera postojećeg poslužitelja može biti jeftinija nego dodavanje više poslužitelja

- **Poboljšana izvedba** – dodavanje više resursa može poboljšati izvedbu baze podataka
- **Ograničena skalabilnost** – maksimalni kapacitet poslužitelja ograničen je fizičkim resursima hardvera
- **Jedna točka kvara** – ako poslužitelj prestane sa radom, cijela baza postaje nedostupna
- **Zastoj tijekom nadogradnje** – nadogradnjom poslužitelja zahtijeva zastoj i zaustavlja rad aplikacije

5.1.2. Horizontalna skalabilnost

Horizontalna skalabilnost, također poznata kao (engl. *scale-out*), vrsta je skalabilnosti baze podataka koja uključuje dodavanje više strojeva u sustav baze podataka kako bi se povećao kapacitet [19]. To znači da je radno opterećenje raspoređeno na više poslužitelja, što omogućuje gotovo neograničeni rast. Horizontalna skalabilnost često se koristi za velike baze podataka koje zahtijevaju visoku dostupnost, toleranciju na greške i skalabilnost. Temelji se na ideji dodavanja više strojeva u skup resursa, što može biti isplativije od nadogradnje jednog stroja. Horizontalna skalabilnost također može iskoristiti replikaciju podataka, pri čemu jedan stroj drži primarnu kopiju cijele baze podataka dok se višestruke kopije i/ili predmemorija koriste za dijeljenje opterećenja samo za čitanje. To omogućuje poboljšano vrijeme odziva, bolje performanse i veću dostupnost [19]. Horizontalna skalabilnost ima nekoliko ključnih prednosti u odnosu na vertikalni pristup, uključujući mogućnost skaliranja na gotovo bilo koju veličinu i potpuno iskorištavanje prednosti elastičnog računarstva u oblaku [19]. Horizontalna skalabilnost je dobar izbor za aplikacije koje zahtijevaju visoku dostupnost, toleranciju na greške i skalabilnost. Neki od prednosti i nedostataka horizontalne skalabilnost su:

- **Povećana skalabilnost** – omogućuje neograničenu skalabilnost budući da se u skupinu može dodati više poslužitelja
- **Visoka dostupnost** – podaci raspoređeni na više poslužitelja, ako jedan od njih zakaže podaci su i dalje dostupni na drugim poslužiteljima u skupini
- **Poboljšana izvedba** – raspodjelom radnog opterećenja na više poslužitelja može poboljšati izvedbu baze podataka
- **Složena arhitektura** – više poslužitelja u skupini znači veća složenost baze podataka
- **Trošak** – dodavanje više poslužitelja može biti skupo, zahtijevajući softverske i mrežne licence i dodatni hardver
- **Dosljednost podataka** – distribucija podataka na više poslužitelja može otežati održavanje dosljednosti podataka i osigurati da su svi podaci ažurni.

5.2. Prednosti i nedostatci skalabilnosti relacijskih i NoSQL baza podataka

Skalabilnost je važan faktor koji treba uzeti u obzir pri odabiru između relacijskih i NoSQL baza podataka. NoSQL baze podataka dizajnirane su za horizontalno skaliranje, što znači da se mogu skalirati na jeftinom *hardveru*, što omogućuje gotovo neograničeni rast [20]. Fleksibilniji su i lakši za skaliranje od relacijskih baza podataka, što ih čini idealnim za velike podatkovne aplikacije. NoSQL baze podataka mogu rukovati velikim količinama nestrukturiranih ili polustrukturiranih podataka, što ih čini idealnim za velike podatkove aplikacije. Prednosti i nedostatci skalabilnosti NoSQL baza podataka su:

- **Horizontalna skalabilnost** – NoSQL baze podataka dizajnirane su da budu horizontalno skalabilne, što olakšava dodavanje više poslužitelja za rukovanje povećanim prometom i količinom podataka
- **Fleksibilnost *sheme*** – NoSQL baze podataka su fleksibilne u pogledu *sheme*, što znači da možete jednostavno dodati ili ukloniti polja bez potrebe za izmjenom cijele *sheme*
- **Eventualna dosljednost** – NoSQL baze podataka često koriste eventualnu dosljednost, što znači da podaci možda neće biti odmah dosljedni u svim čvorovima u skupini
- **Vertikalna skalabilnost** – NoSQL baze podataka mogu se vertikalno skalirati dodavanjem više resursa poslužitelju isto kao što vrijedi i za relacijske baze podataka

S druge strane, relacijske baze podataka dizajnirane su za vertikalno skaliranje, što znači da za povećanje skale morate nadograditi stroj/poslužitelj koji poslužuje (engl. *Hosting*, usluga putem koje se pojedincu osiguravaju pohranjivanje i računalni resursi) bazu podataka, što može biti skupo [20]. Relacijske baze podataka dizajnirane su za rukovanje strukturiranim podacima i imaju unaprijed definiranu *shemu*, što znači da je struktura podataka fiksna i mora se definirati prije nego što se podaci mogu dodati u bazu. Vertikalna skalabilnost često se koristi za male do srednje baze podataka koje zahijetvaju visoku izvedbu i nisku *latenciju*. Sveukupno, NoSQL baze podataka su fleksibilnije i skalabilnije od relacijskih baza podataka, što ih čini idealnim za velike podatkovne aplikacije i *agilne* prakse razvoja *softvera*.

Svaki tip baze podataka ima svoje prednosti i nedostatke što se tiče njihovih skalabilnosti, jesu li više horizontalno skalabilne ili su više vertikalno skalabilne. Navedene prednosti i nedostatci su za relacijske baze podataka:

- **Vertikalna skalabilnost** – relacijske baze podataka mogu se vertikalno skalirati dodavanjem više resursa poslužitelju. Ovaj pristup je jednostavan i može biti jeftiniji od horizontalnog skaliranja
- **Utvrđena tehnologija** – relacijske baze se koriste već duže vrijeme i uspostavile su najbolje prakse i tehnike optimizacije za skaliranje
- **Ograničena horizontalna skalabilnost** – relacijske baze podataka mogu se horizontalno skalirati, ali mogu biti složene i teške za upravljanje jer su podaci raspoređeni na više poslužitelja
- **Promjene *sheme*** – kako aplikacija raste, *shema* baze podataka može se morati modificirati, što može biti teško i dugotrajno

6. USPOREDBA STRUKTURE RELACIJSKIH I NOSQL BAZA PODATAKA

Kao što se moglo pročitati u prijašnjim poglavljima relacijske i NoSQL baze podataka su u nekim pogledima slični, ali su u mnogima različiti. Ovo poglavlje se fokusira na usporedbu strukture relacijskih i različitih tipova NoSQL baza podataka kao što su dokument tip, graf tip i baza podataka širokih stupaca. Točni tip baze podataka koji će se uspoređivati su relacijske baze podataka sa MongoDB bazom podataka, relacijske baze sa Neo4j i relacijska baza sa Apache Cassandra bazom podataka (dokument, graf, široki stupac).

6.1. Usporedba relacijske i MongoDB

Relacijske baze podataka se primarno sastoje od entiteta i njihovih atributa. Svaki entitet ima svoj primarni ključ, a neki od entiteta ima i strani ključ koji je samo privatni ključ nekog drugog entiteta. Entiteti mogu imati odnos jedan prema jedan, jedan prema više i više prema više. Što se tiče MongoDB koji je dokument tip baze podataka on se sastoji od dokumenata. Podaci se spremaju unutar kolekcija te su dosta fleksibilni u svojoj reprezentaciji. Relacije ili odnosi između podataka se ostvaruju koristeći reference ili ugnježđene dokumente. Neke od glavnih razlika i objašnjena tih razlika su prikazane u tablici 6.1. [21].

Tablica 6.1. *Usporedba relacijske i MongoDB baze podataka*

Relacijska baza podataka	MongoDB baza podataka
Model relacijske baze podataka u kojem su podaci pohranjeni u više tablica.	Baza podataka otvorenog koda orijentirana na dokumente koja nema koncept tablica, shema, redaka ili SQL-a.
Zapisi se pohranjuju kao redovi u tablicama, pri čemu su tablice organizirane u stupce pri čemu je svaki stupac pripisan jednoj vrsti podataka.	Koristi različite formate za pohranu podataka kao što su pohrane dokumenata.
Slijedi tipičan dizajn sheme koji se sastoji od nekoliko tablica i odnosa među njima.	Temelji se na prikazu podataka bez sheme bez obzira na koncept odnosa.
Vertikalno skalabilni, što znači da kada se opterećenja baze podataka povećaju, bazu podataka skalirate povećanjem kapaciteta postojećeg hardvera.	Je baza podataka koja odgovara svima i smatra se skalabilnijom od tradicionalnih modela relacijskih baza podataka.

6.1.1. Strukturiranje relacijske i MongoDB baze podataka

Struktura relacijske baze podataka i MongoDB baze podataka se po prijašnjoj tablici mnogo razlikuje, ali sama struktura, to jest sama usporedba strukture nam ne daje nikakve točne rezultate koja je baza podataka bolja već nam daje informacije za šta se točno svaka baza podataka može koristiti. Što se tiče brzine obrade podataka, to jest obrađivanja zahtjeva unutar baze podataka, možemo pogledati istraživanje koje se bazira na usporedbi brzine obrade podataka ovisno o količini podataka i ovisno o zahtjevu koji je potrebno obraditi [22]. Za inspiraciju kod testiranja baze podataka uzeto je to isto istraživanje. Kreirane su dvije baze podataka unutar Visual Studio-a i C# programskog jezika. Za primjer relacijske baze podataka korištena je baza podataka Vulkanizerske radnje koja je navedena i detaljnije objašnjena unutar odjeljka 3.4. Primjer NoSQL baze podataka dokument tipa je isto Vulkanizerska radnja, to jest svi entiteti i atributi koji se nalaze unutar relacijske baze podataka su prebačeni i postavljeni kao jedan dokument unutar NoSQL baze podataka.

Za detaljnije objašnjenje i izgled NoSQL baze podataka dokument tipa može se pogledati odjeljak 4.1. U ovom eksperimentu će se provoditi testiranje brzine umetanja, brisanja i promjene podataka unutar baza podataka te će se rezultati usporediti i prikazati na grafu. Nakon tog testiranja testirati će se određeni upiti koji će biti prikazani i objašnjeni na grafu.

Kao što je prijašnje objašnjeno za testiranje i izradu baza podataka korišten je Visual Studio i C# programski jezik. Pošto će se sve izrađivati i provoditi unutar Visual Studio potrebno je prebaciti relacijsku bazu podataka u C#. Dijelove koda relacijske i NoSQL baze podataka se nalaze na primjerima koda 6.1 i 6.2.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;
using MySql.Data.MySqlClient;
using System.Diagnostics;
namespace NoSQLDB
{
        public class DatabaseManager
    {
        private const string ConnectionString =
"server=localhost;uid=root;pwd=MySQLserver123;database=test";

        public void CreateTables()
        {
            using (MySqlConnection connection = new
MySqlConnection(ConnectionString))
            {
                connection.Open();
                using (MySqlCommand command = connection.CreateCommand())
                {
                    command.CommandText = @"
                    CREATE TABLE Radnja
                    (
                        ID CHAR(5) PRIMARY KEY,
                        Naziv VARCHAR(30)
                    );

                    CREATE TABLE Musterija
                    (
                        OIB CHAR(11) PRIMARY KEY,
                        Ime VARCHAR(30),
                        Prezime VARCHAR(30),
                        Datum_rodenja DATETIME,
                        Postanski_broj INT,
                        Ime_grada VARCHAR(50),
                        Adresa VARCHAR(100)
                    );
```

Primjer koda 6.1. *Dio koda za relacijsku bazu podataka*

Na primjeru koda 6.1. je prikazan dio koda za relacijsku bazu podataka napisanu unutar Visual studio-a sa C# programskim jezikom. Kako bi se osposobila konekcija sa SQL serverom korišten je MySQL Server koji se može preuzeti sa interneta. Kako bi se Visual studio uspješno povezao na MySQL server potrebno je instalirati *MySQL data client*. Na primjeru koda 6.1. se može vidjeti kako nam je potreban *ConnectionString* gdje je potrebno postaviti *Connection String* koji se dobije od strane *MySQL servera*. Ujedno se unutar primjera kod 6.1. može vidjeti jedna tablica *MUŠTERIJA* sa svojim atributima.

```
using System;
using System.Collections.Generic;
using System.Text;
using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;
using MongoDB.Driver;
using System.Diagnostics;

namespace NoSQLDB
{
    [BsonIgnoreExtraElements]
    public class Musterija
    {
        [BsonId]
        public string OIB { get; set; }
        public string Ime { get; set; }
        public string Prezime { get; set; }
        public DateTime Datum_rođenja { get; set; }
        public int Poštanski_broj { get; set; }
        public string Ime_grada { get; set; }
        public string Adresa { get; set; }
    }
```

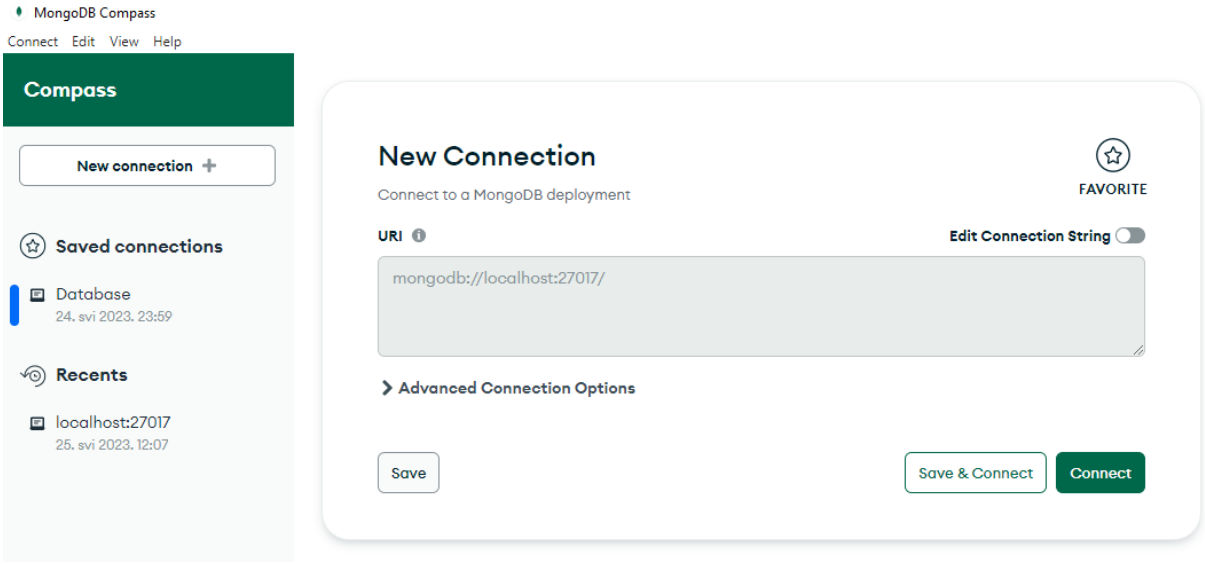
Primjer koda 6.2. *Dio koda za NoSQL bazu podataka*

Na primjeru koda 6.2. se može pobliže vidjeti dio koda za NoSQL bazu podataka. Isto kao i kod relacijske baze podataka za NoSQL bazu podataka potrebno je povezati se na server. Za server korišten je MongoDB server koji je dokument tip NoSQL baze podataka. Kako bi se postigla uspješna konekcija i pravilno postavljanje baze podataka potrebno je instalirati *MongoDB.Driver*, a za lakše upravljanje bazom podataka korišten je *MongoDB Compass*. Ujedno se na slici može vidjeti i primjer jednog od dokumenta pod nazivom *MUSTERIJA* isto kao i kod relacijske baze podataka te ima iste atribute kao i entitet u relacijskoj bazi podataka. Na primjeru koda 6.3. se može vidjeti primjer povezivanja na MongoDB server.

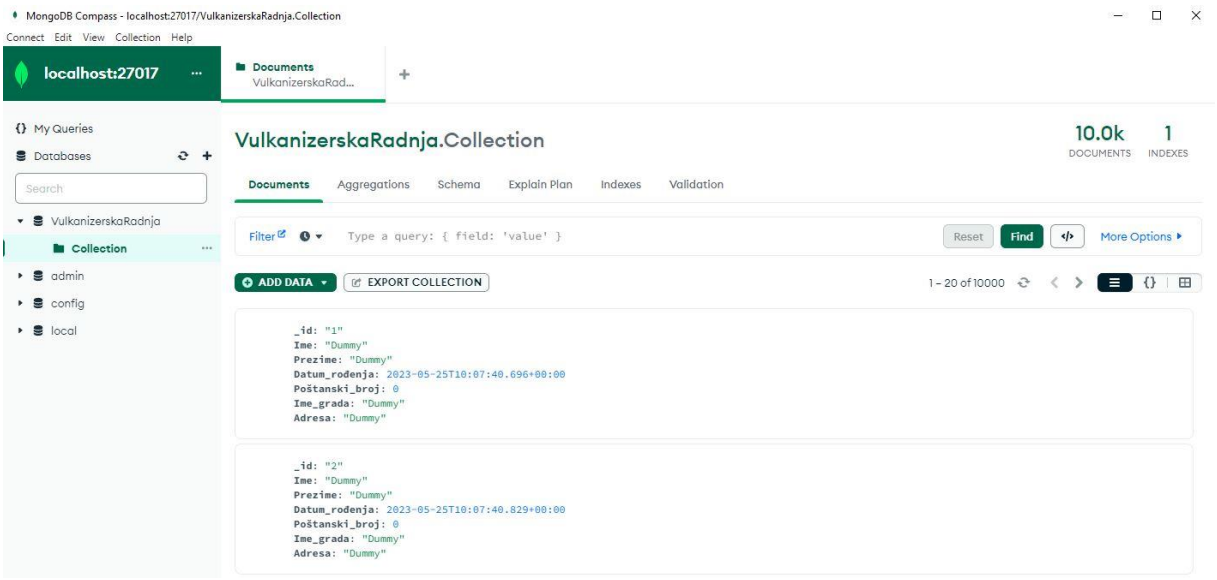
```
var client = new MongoClient("mongodb://localhost:27017");
var database = client.GetDatabase("VulkanizerskaRadnja");
var collection = database.GetCollection<Musterija>("Collection");
```

Primjer koda 6.3. *Dio koda za povezivanje na MongoDB bazu podataka*

Kao što se može i vidjeti na primjeru koda 6.3. za uspješno povezivanje na MongoDB klijenta potreban nam je *Connection String* koji nam dodjeljuje MongoDB. Pošto se unutar MongoDB-a može izraditi više baza podataka potrebno je navesti koju točno bazu podataka želimo koristiti, a u ovom slučaju to je Vulkanizerska radnja. Unutar svake baze podataka možemo imati više kolekcija, ali za ovaj eksperiment koristiti će se samo jedna kolekcija pod nazivom *Collection*. Kako sve to detaljnije izgleda unutar *MongoDB Compass-a* može se pogledati na slici 6.1. i 6.2.

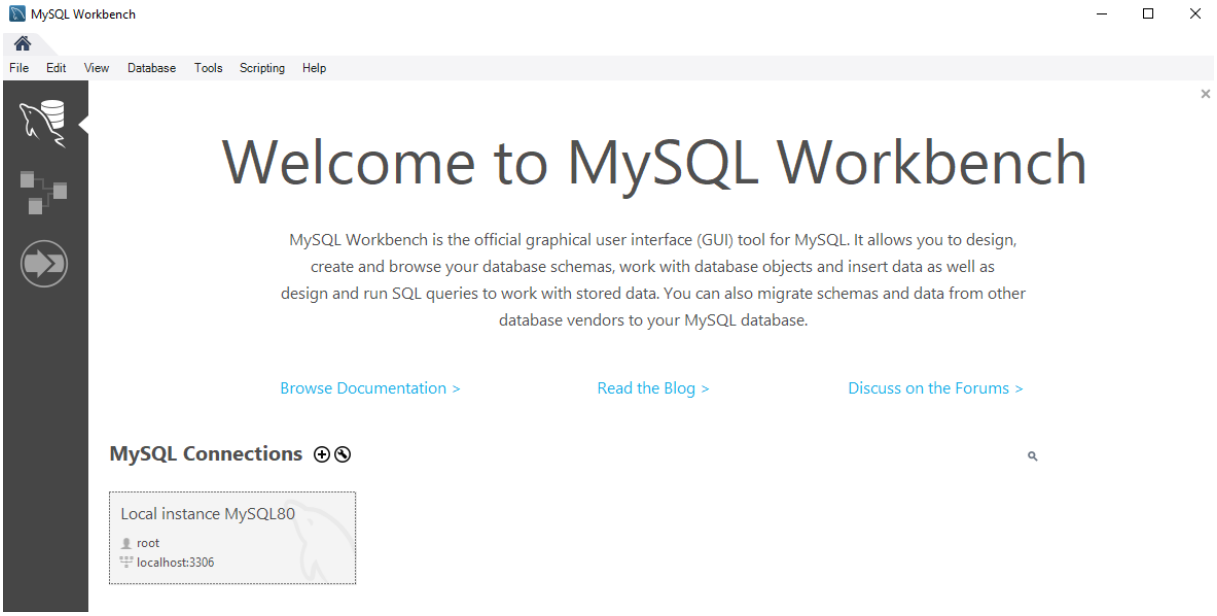


Slika 6.1. *MongoDB Compass konekcija*

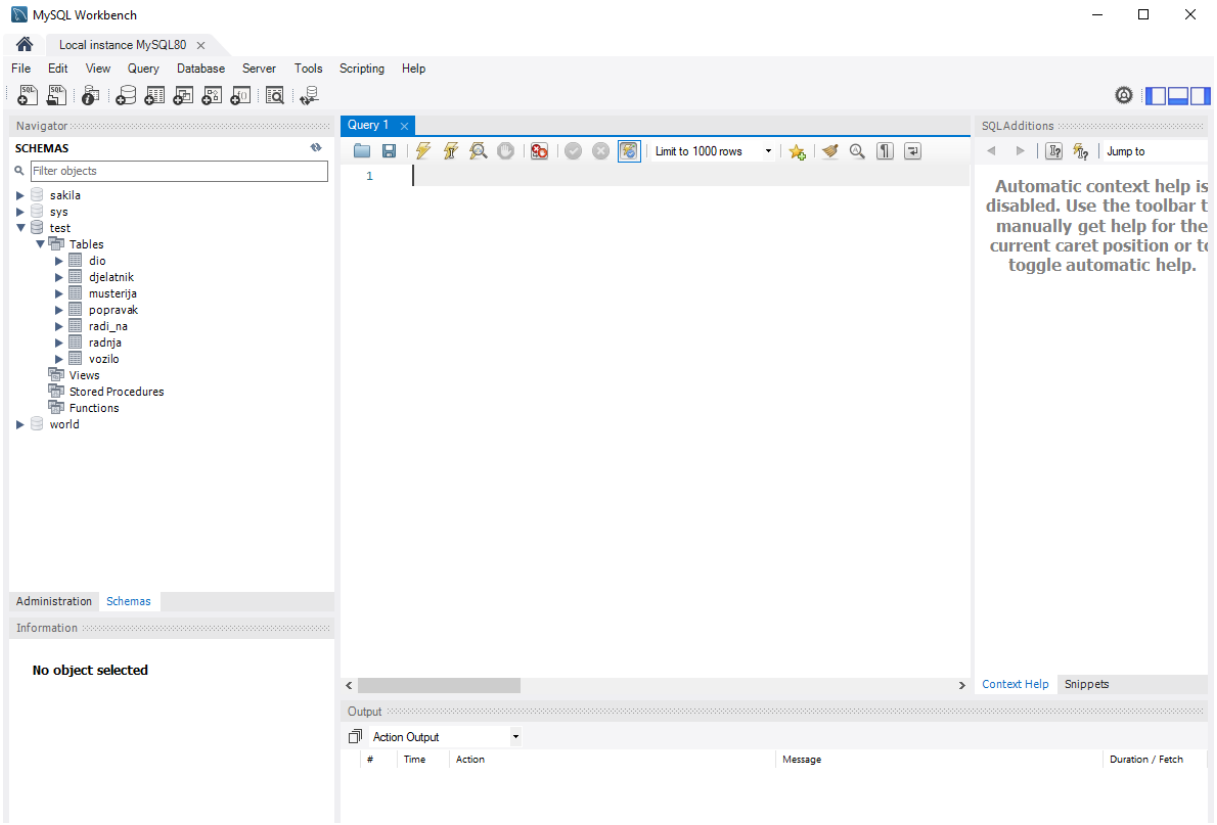


Slika 6.2. *MongoDB Compass prikaz baze podataka*

Što se tiče povezivanja relacijske baze podataka na MySQL server ono se može pobliže pogledati na slikama 6.3. i 6.4.



Slika 6.3. *MySQL Workbench*



Slika 6.4. *MySQL Baza podataka*

Na slici 6.4. se može vidjeti baza podataka pod nazivom *test* i njezini entiteti kao što su *DIO, DJELATNIK, RADNJA* i drugi. Svaka tablica se kreirala uz pomoć klase *DatabaseManager* koja u sebi sadrži funkciju za kreiranje entiteta i atributa te njihovo povezivanje sa privatnim i stranim ključevima (Primjer koda 6.1.).

6.1.2. Testiranje unosa, brisanja i promjene podataka

U ovom dijelu se odrađuje testiranje na bazama podataka. Primjeri testiranja su unos, brisanje i promjena podataka za obje baze podataka. Na primjerima koda 6.4. i 6.5. se može vidjeti na koji način se unose podaci u obje baze podataka kada je osigurana uspješna konekcija sa serverom. Bitno je napomenuti kako se za unos podataka u relacijsku bazu podataka koristili određeni *Stringovi* pošto se ti *Stringovi* zapravo predaju MySQL serveru koji ih prevede u određene upite dok unutar Visual Studio-a za MongoDB bazu podataka imamo implementirane funkcije koje direktno komuniciraju sa MongoDB bazom podataka.

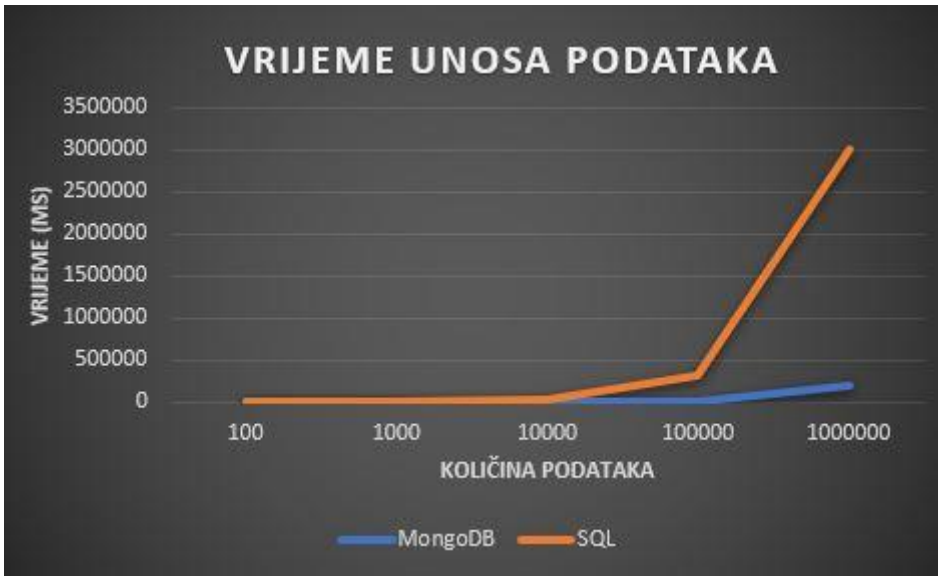
```
public void InsertDummyValues(int dataAmount)
        {
            using (MySqlConnection connection = new
MySqlConnection(ConnectionString))
            {
                connection.Open();

                Stopwatch stopwatch = new Stopwatch();
                stopwatch.Start();
                int dummy_oib = 0;
                for (int i = 0; i < dataAmount; i++)
                {
                    dummy_oib++;
                    using (MySqlCommand command =
connection.CreateCommand())
                    {
                        command.CommandText = "INSERT INTO Musterija (OIB,
Ime, Prezime, Datum_rodenja, Postanski_broj, Ime_grada, Adresa) VALUES
(@OIB, @Ime, @Prezime, @Datum_rodenja, @Postanski_broj, @Ime_grada,
@Adresa)";


                        command.Parameters.AddWithValue("@OIB",
dummy_oib.ToString());
                        command.Parameters.AddWithValue("@Ime",
"dummy_ime");
                        command.Parameters.AddWithValue("@Prezime",
"dummy_prezime");
                        command.Parameters.AddWithValue("@Datum_rodenja",
DateTime.Now);
                        command.Parameters.AddWithValue("@Postanski_broj",
12345);
                        command.Parameters.AddWithValue("@Ime_grada",
"dummy_ime_grada");
                        command.Parameters.AddWithValue("@Adresa",
"dummy_adresa");

                        command.ExecuteNonQuery();
                    }
                }


                stopwatch.Stop();
                Console.WriteLine($"Time taken to insert {dataAmount}
records: {stopwatch.ElapsedMilliseconds} ms");
            }
```

Primjer koda 6.4. *Unos Dummy podataka u SQL bazu podataka i mjerenje unosa podataka*

```
public static long CalculateInsertTime(IMongoCollection<Musterija>
collection, int numInserts)
        {
            int id = 0;
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();

            for (int i = 0; i < numInserts; i++)
            {
                id++;
                collection.InsertOne(new Musterija
                {
                    OIB = id.ToString(),
                    Ime = "Dummy",
                    Prezime = "Dummy",
                    Datum_rođenja = DateTime.Now,
                    Poštanski_broj = 0,
                    Ime_grada = "Dummy",
                    Adresa = "Dummy"
                }); ;
            }
            stopwatch.Stop();

            return stopwatch.ElapsedMilliseconds;
        }
```

Primjer koda 6.5. *Unos Dummy podataka u NoSQL bazu podataka i mjerenje unosa podataka*

Za unos podataka koristilo se 100, 1000, 10000, 100000 i 1000000 *dummy* podataka kako bi se dobili što opširniji rezultati. Nakon odrađenog testiranja brzine unosa podataka u svaku bazu podataka dobije se rezultat prikazan na slici 6.5.



Slika 6.5. *Rezultat testiranja brzine unosa podataka*

Na slici 6.5. možemo vidjeti koliko je svakoj bazi podataka potrebno vremena da unese određeni broj podataka u milisekundama. Razlika se primjećuje nakon unosa više od 10000 podataka gdje brzina unosa relacijske baze podataka znatno opada (potrebno je više vremena da se podaci unesu u bazu podataka). Kod unosa 1000000 podataka vidi se znatna razlika u vremenu unosa podataka gdje je relacijskoj bazi podataka potrebno čak sat vremena da unese toliku količinu podataka dok je NoSQL bazi podataka dokument tipa potrebno samo nekoliko minuta.

Što se tiče ažuriranja i brisanja podataka iz bazi podataka koristilo se 100, 1000 i 10000 *dummy* podataka kako bi se skratilo vrijeme eksperimenta jer se pokazalo kako relacijska baza podataka treba znatno više vremena da odradi ažuriranje i brisanje podataka kada je riječ o velikim količinama podataka. Na primjerima koda 6.6. i 6.7. se mogu vidjeti funkcije koje su se koristile za ažuriranje i brisanje podataka iz obje baze podataka.

```
public static long CalculateDeleteTime(IMongoCollection<Musterija>
collection, int numDeletes)
        {
            int id = 0;
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();

            for(int i = 0;i<numDeletes;i++)
            {
                id++;
                var filter = Builders<Musterija>.Filter.Eq("OIB",id);
                collection.DeleteOne(filter);
            }

            stopwatch.Stop();

            return stopwatch.ElapsedMilliseconds;
        }

        public static long CalculateUpdateTime(IMongoCollection<Musterija>
collection, int numUpdates)
        {
            int id = 0;
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();

            for(int i = 0;i<numUpdates;i++)
            {
                id++;
                var filter = Builders<Musterija>.Filter.Eq("OIB", id);
                var update = Builders<Musterija>.Update.Set("Ime",
"Dominik");
                collection.UpdateOne(filter, update);

            }
            stopwatch.Stop();
            return stopwatch.ElapsedMilliseconds;
        }
```

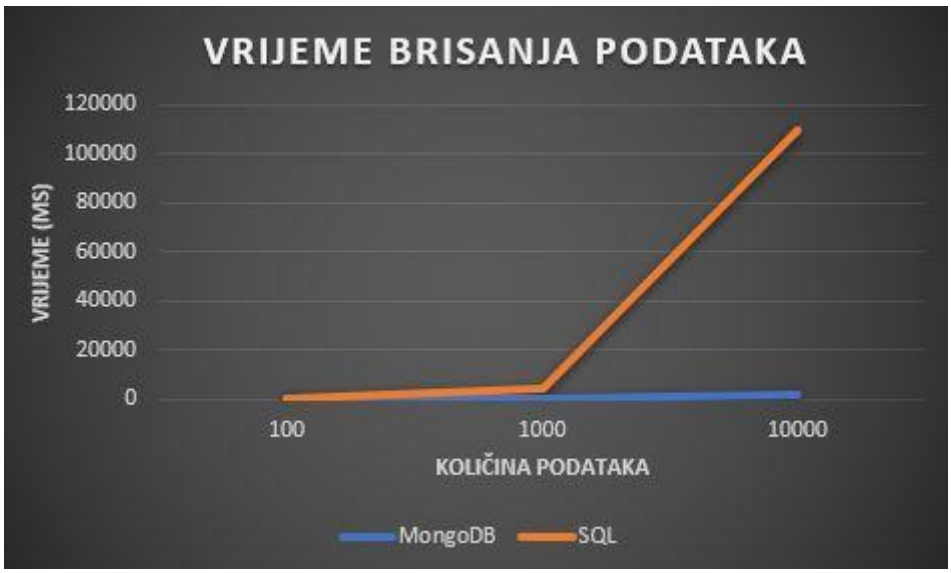
Primjer koda 6.6. *Brisanje i ažuriranje podataka NoSQL baze podataka*

```
public void DeleteDummyValues(int dataAmount)
        {
            using (MySqlConnection connection = new
MySqlConnection(ConnectionString))
            {
                connection.Open();
                Stopwatch stopwatch = new Stopwatch();
                stopwatch.Start();
                int dummy_oib = 0;
                for (int i = 0; i < dataAmount; i++)
                {
                    dummy_oib++;
                    using (MySqlCommand command =
connection.CreateCommand())
                    {
                        command.CommandText = "DELETE FROM Musterija WHERE
OIB = @OIB";
                        command.Parameters.AddWithValue("@OIB", dummy_oib);
                        command.ExecuteNonQuery();
                    }
                }
                stopwatch.Stop();
                Console.WriteLine($"Time taken to Delete {dataAmount}
records: {stopwatch.ElapsedMilliseconds} ms");
            }
        }
        public void UpdateDummyValues(int dataAmount)
        {
            using (MySqlConnection connection = new
MySqlConnection(ConnectionString))
            {
                connection.Open();
                Stopwatch stopwatch = new Stopwatch();
                stopwatch.Start();
                int dummy_oib = 0;
                for (int i = 0; i < dataAmount; i++)
                {
                    dummy_oib++;
                    using (MySqlCommand command =
connection.CreateCommand())
                    {
                        command.CommandText = "UPDATE Musterija SET Ime =
@newName WHERE OIB = @OIB";
                        command.Parameters.AddWithValue("@newName",
"Dominik");
                        command.Parameters.AddWithValue("@OIB", dummy_oib);
                        command.ExecuteNonQuery();
                    }
                }
                stopwatch.Stop();
                Console.WriteLine($"Time taken to Update {dataAmount}
records: {stopwatch.ElapsedMilliseconds} ms");
            }
        }
```

Primjer koda 6.7. *Brisanje i ažuriranje podataka SQL baze podataka*

Uz pomoć funkcija prikazanih na primjerima koda 6.6. i 6.7. se izračunava koliko vremenski je potrebno svakoj bazi podataka da ažurira i izbriše podatke. Ažuriranje i brisanje na svakoj bazi podataka se radilo nad entitetom ili dokumentom *MUSTERIJA*, a kod ažuriranja se ažurirao atribut *IME* za svaki dokument i entitet unutar baze podataka. Rezultati izvršenih mjerenja su prikazani na slikama 6.6. i 6.7.

Slika 6.6. *Rezultat ažuriranja podataka*



Slika 6.7. *Rezultat brisanja podataka*

Kao što se može vidjeti na rezultatima mjerenja relacijskoj bazi podataka je potrebno znatno više vremena da ažurira i obriše određene podatke. Nakon unosa 1000 podataka može se vidjeti znatan skok u potrebnom vremenu za odrađivanje brisanja i ažuriranja.

Po ovim svim rezultatima može se zaključiti da MongoDB baza podataka ili dokument tip NoSQL baze podataka znatno brže unosi, ažurira i briše podatke od relacijske baze podataka. MongoDB i relacijska baza podataka imaju dosta slična vremena unosa, brisanja i ažuriranja ako se to odrađuje nad malim brojem podataka, čim se broj podataka poveća MongoDB baza podataka prevladava u brzini obrade podataka.

6.1.3. Testiranje nad određenim upitima

U ovom poglavlju testiranje će se više bazirati na određene upite kao što su dohvaćanje svih mušterija čije ime je Dominik ili dohvaćanje svih mušterija čije ime je Dominik ili se prezivaju Vidović. Ovo testiranje će se provesti nad već napravljenim bazama podataka te će se svaki upit preoblikovati tako da odgovara određenoj bazi podataka. Za testiranje baza koristiti će se tri upita, a ti upiti su:

- **Q1** - dohvaćanje svih mušterija čije je ime Dominik
- **Q2** - brisanje svih mušterija čije je ime Dominik ili se prezivaju Vidović
- **Q3** - ažuriranje adrese svih mušterija čije je ime Dominik ili se prezivaju Vidović ili dolaze iz Osijeka

Na sljedećim primjerima koda se mogu vidjeti funkcije korištene za testiranje baza podataka. Potrebno je napomenuti da se testiranje svih upita obrađuje nad 5000 podataka prepravljenih da ne budu *dummy* podaci već da budu smisleni podaci (barem za ime, prezime i ime grada mušterije).

```
public static void InsertMongoDBData(IMongoCollection<Musterija>
collection)
        {
            int id = 0;
            Random random = new Random();
            List<string> randomImeValues = new List<string> { "Dominik",
"Random1", "Random2", "Random3", "Random4", "Random5", "Random6",
"Random7", "Random8", "Random9" };
            List<string> randomPrezimeValues = new List<string> {
"Vidović", "Random1", "Random2", "Random3", "Random4", "Random5",
"Random6", "Random7", "Random8", "Random9" };
            List<string> randomImeGradaValues = new List<string> {
"Osijek", "Random1", "Random2", "Random3", "Random4", "Random5", "Random6",
"Random7", "Random8", "Random9" };
            for (int i = 0; i < 5000; i++)
            {
                id++;
                collection.InsertOne(new Musterija
                {
                    OIB = id.ToString(),
                    Ime =
randomImeValues[random.Next(randomImeValues.Count)],
                    Prezime =
randomPrezimeValues[random.Next(randomPrezimeValues.Count)],
                    Datum_rođenja = DateTime.Now,
                    Poštanski_broj = 0,
                    Ime_grada =
randomImeGradaValues[random.Next(randomImeGradaValues.Count)],
                    Adresa = "Dummy"
                }); ;
            }
        }
```

Primjer koda 6.8. *Unos 5000 podataka u MongoDB bazu podataka*

```
public void InsertMySQLData()
        {
            using (MySqlConnection connection = new
MySqlConnection(ConnectionString))
            {
                connection.Open();
                int dummy_oib = 0;

                Random random = new Random();
                List<string> randomImeValues = new List<string> {
"Dominik", "Random1", "Random2", "Random3", "Random4", "Random5",
"Random6", "Random7", "Random8", "Random9" };
                List<string> randomPrezimeValues = new List<string> {
"Vidović", "Random1", "Random2", "Random3", "Random4", "Random5",
"Random6", "Random7", "Random8", "Random9" };
                List<string> randomImeGradaValues = new List<string> {
"Osijek", "Random1", "Random2", "Random3", "Random4", "Random5", "Random6",
"Random7", "Random8", "Random9" };

                for (int i = 0; i < 5000; i++)
                {
                    dummy_oib++;
                    using (MySqlCommand command =
connection.CreateCommand())
                    {
                        command.CommandText = "INSERT INTO Musterija (OIB,
Ime, Prezime, Datum_rodenja, Postanski_broj, Ime_grada, Adresa) VALUES
(@OIB, @Ime, @Prezime, @Datum_rodenja, @Postanski_broj, @Ime_grada,
@Adresa)";

                        command.Parameters.AddWithValue("@OIB",
dummy_oib.ToString());
                        command.Parameters.AddWithValue("@Ime",
randomImeValues[random.Next(randomImeValues.Count)]);
                        command.Parameters.AddWithValue("@Prezime",
randomPrezimeValues[random.Next(randomPrezimeValues.Count)]);
                        command.Parameters.AddWithValue("@Datum_rodenja",
DateTime.Now);
                        command.Parameters.AddWithValue("@Postanski_broj",
12345);
                        command.Parameters.AddWithValue("@Ime_grada",
randomImeGradaValues[random.Next(randomImeGradaValues.Count)]);
                        command.Parameters.AddWithValue("@Adresa",
"dummy_adresa");

                        command.ExecuteNonQuery();
                    }
                }

            }
        }
```

Primjer koda 6.9. *Unos 5000 podataka u SQL bazu podataka*

Na prijašnjim primjerima koda možemo vidjeti unos 5000 podataka u MongoDB i SQL bazu podataka. Funkcije prikazane na primjerima koda unose 5000 podataka tipa *MUSTERIJA* te na nasumičan način odabire *IME*, *PREZIME* i *IME_GRAD* za entitet ili dokument (ovisno o bazi podataka), a svi ostali podaci o mušteriji su *dummy* podatci. Na sljedećim primjerima koda mogu se vidjeti funkcije korištene za upite za svaku bazu podataka.

```
public static List<Musterija> GetMusterijeByIme(IMongoCollection<Musterija>
collection)
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            var filter = Builders<Musterija>.Filter.Eq("Ime", "Dominik");
            var result = collection.Find(filter).ToList();

            stopwatch.Stop();
            Console.WriteLine("Vrijeme potrebno za prvi upit (MongoDB): " +
stopwatch.ElapsedMilliseconds);
            return result;
        }

        public static void
DeleteMusterijeByImeAndPrezime(IMongoCollection<Musterija> collection)
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            var filter = Builders<Musterija>.Filter.Eq("Ime", "Dominik") |
Builders<Musterija>.Filter.Eq("Prezime", "Vidović");
            var result = collection.DeleteMany(filter);

            stopwatch.Stop();
            Console.WriteLine("Vrijeme potrebno za Drugi upit (MongoDB): "
+ stopwatch.ElapsedMilliseconds);
        }

        public static void
UpdateMusterijeByImePrezimeAndGrad(IMongoCollection<Musterija> collection)
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            var filter = Builders<Musterija>.Filter.Eq("Ime", "Dominik") |
Builders<Musterija>.Filter.Eq("Prezime", "Vidović") |
Builders<Musterija>.Filter.Eq("Ime_grada", "Osijek");
            var update = Builders<Musterija>.Update.Set("Adresa", "Nova
adresa");
            var result = collection.UpdateMany(filter, update);
            stopwatch.Stop();
            Console.WriteLine("Vrijeme potrebno za Treci upit (MongoDB): "
+ stopwatch.ElapsedMilliseconds);
        }
```

Primjer koda 6.10. *Q1, Q2, Q3 upiti za NoSQL bazu podataka*

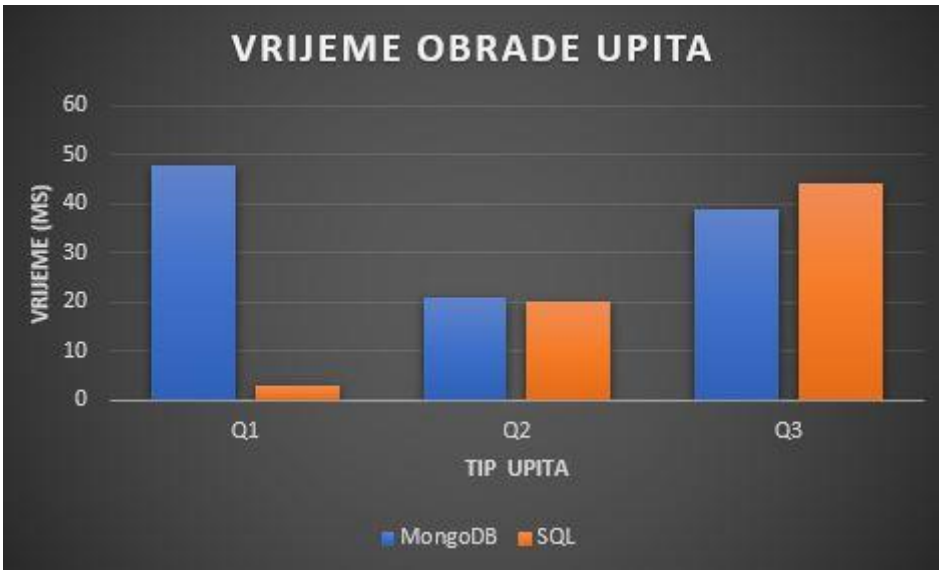
```
public void SelectMusterijeByIme()
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            using (MySqlConnection connection = new
MySqlConnection(ConnectionString))
            {
                connection.Open();

                string query = "SELECT * FROM Musterija WHERE Ime = @Ime";
                using (MySqlCommand command = new MySqlCommand(query,
connection))
                {
                    command.Parameters.AddWithValue("@Ime", "Dominik");
                    command.ExecuteNonQuery();
                }
            }
            stopwatch.Stop();
            Console.WriteLine("Vrijeme potrebno za dohvaćanje (MySQL): " +
stopwatch.ElapsedMilliseconds);
        }

        public void DeleteMusterijeByImePrezime()
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            using (MySqlConnection connection = new
MySqlConnection(ConnectionString))
            {
                connection.Open();

                string query = "DELETE FROM Musterija WHERE Ime = @Ime OR
Prezime = @Prezime";
                using (MySqlCommand command = new MySqlCommand(query,
connection))
                {
                    command.Parameters.AddWithValue("@Ime", "Dominik");
                    command.Parameters.AddWithValue("@Prezime", "Vidović");

                    command.ExecuteNonQuery();
                }
            }
            stopwatch.Stop();
            Console.WriteLine("Vrijeme potrebno za brisanje (MySQL): " +
stopwatch.ElapsedMilliseconds);
```

Primjer koda 6.11. *Q1 i Q2 upit za SQL bazu podataka*

```
        public void UpdateMusterijeByImePrezimeImegrada()
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            using (MySqlConnection connection = new
MySqlConnection(ConnectionString))
            {
                connection.Open();

                string query = "UPDATE Musterija SET Adresa = @NewAdresa
WHERE Ime = @Ime OR Prezime = @Prezime OR Ime_grada = @ImeGrada";
                using (MySqlCommand command = new MySqlCommand(query,
connection))
                {
                    command.Parameters.AddWithValue("@NewAdresa", "nova
adresa");
                    command.Parameters.AddWithValue("@Ime", "Dominik");
                    command.Parameters.AddWithValue("@Prezime", "Vidović");
                    command.Parameters.AddWithValue("@ImeGrada", "Osijek");

                    command.ExecuteNonQuery();
                }
            }
            stopwatch.Stop();
            Console.WriteLine("Vrijeme potrebno za ažuriranje (MySQL): " +
stopwatch.ElapsedMilliseconds);
        }
```

Primjer koda 6.11. *Q3 upit za SQL bazu podataka*

Na prijašnjim primjerima koda se mogu vidjeti upiti za obje baze podataka. Upit za NoSQL bazu podataka je isti kao i upit za SQL bazu podataka razlika je samo u načinu pisanja koda za MongoDB i MySQL. Nakon pokretanja funkcija za kreiranje 5000 podataka te pokretanja funkcija za upite nad tim podacima u obje baze podataka dobijemo sljedeće rezutlate prikazane na slici 6.8.



Slika 6.8. *Rezultat obrade upita*

Prema slici 6.8. se može vidjeti kako SQL baza podataka je znatno brža što se tiče dohvaćanja podataka od MongoDB baze podataka. Što se tiče brisanja podataka i ažuriranja podataka uz određene upite SQL baza podataka je slična MongoDB bazi podataka.

Ovdje možemo zaključiti kako je SQL baza podataka bolja od MongoDB baze podataka što se tiče upita koji rezultiraju u dohvaćanju podataka. Upiti koji rezultiraju u brisanju ili ažuriranju podaka MongoDB i SQL baza podataka su slične.

6.2. Usporedba relacijske i Neo4j

U ovom poglavlju će se usporediti relacijska i graf tip baze podataka. Za usporedbu je uzet MySQL server za predstavnika relacijske baze podataka i za NoSQL već spomenuti Neo4j. Usporedba ovih baza podataka dana je tablicom 6.2. [23].

Tablica 6.2. *Usporedba relacijske i Neo4j baze podataka*

Parametar usporedbe	Relacijska baza podataka	Neo4j graf baza podataka
Pohrana podataka	Podaci su pohranjeni u fiksne, unaprijed definirane tablice koje se sastoje od stupaca i redaka s povezanim podacima koji se često odvajaju između tablica, što smanjuje učinkovitost upita.	Struktura graf baza podataka sa svojstvom slobodnog indeksa rezultira bržim transakcijama i obradom podataka.
Modeliranje podataka	Model baze podataka se mora razviti modeliranjem i potrebno ga je prevesti iz konceptualnog u logički pa tek onda u fizički model. Sve vrste podataka i izvori podataka moraju bit unaprijed poznati, pa sve promjene zahtijevaju stanku u radu za implementaciju promjena.	Fleksibilan proces modeliranja i model podataka koji rezultira nepostojanjem neusklađenosti između logičkog i fizičkog modela. Vrste podataka i izvori podataka mogu se dodati ili mijenjati u bilo kojem trenutku, što dovodi do dramatično kraćih vremena razvoja i prave agilne iteracije.
Performanse upita	Performanse obrade podataka padaju sa brojem i dubinom *JOIN* – ova.	Procesiranje grafa osigurava nimalo latentnosti i izvođenje u realnom vremenu, bez obzira na broj ili dubinu odnosa.
Obrada pri skaliranju	Skaliranje kroz replikaciju i povećanje arhitekture je moguće ali je skupo. Složene veze podataka nisu pogodne za skaliranje.	Model grafa prirodno (inherentno) skalira upite temeljene na uzorku. Skaliranje arhitekture omogućuje integritet podataka putem replikacije

Učinkovitost podatkovnih centara	Konsolidacija poslužitelja je moguća, ali je skupo za arhitekturu. Skaliranje arhitekture je skupo u smislu kupnje, korištenja energije i vremena upravljanja.	Podaci i veze se pohranjuju prirodno zajedno sa poboljšanjem performansi kako složenost i skaliranje raste. To dovodi do konsolidacije poslužitelja i nevjerojatno učinkovitog korištenja sklopovlja.
Upitni jezik	SQL: Upitni jezik kojim se povećava složenost s porastom broja JOIN operacija potrebnih za povezivanje podataka.	Cypher: Nativni upitni jezik za graf baze podataka koji pruža najučinkovitiji i najekspresivniji način za opis povezanosti upitima.

Kao što se može vidjeti prema usporedbi, relacijske baze podataka se razlikuju dosta po strukturi od NoSQL baza podataka i po jezicima, načinima pohrane podataka, modeliranju podataka, učinkovitosti i primjeni. Što se tiče brzini obrade upita relacijske i NoSQL baza podataka provedeno je istraživanje da se odredi brzina svake od baza podataka nad nekim podacima [24]. Unutar tog istraživanja provela se usporedba relacijske baze podataka sa NoSQL bazom podataka graf tipa. Kao inspiracija za usporedbu baza podataka uzeto je to istraživanje. Sa već kreiranom relacijskom bazom podataka koja je definirana unutar poglavlja 3.4. i već korištena u poglavlju 6.1. i novom bazom podataka kreiranom uz pomoć Neo4j servera raditi će se usporedba unosa, brisanja i ažuriranja podataka te će se prikazati rezultati usporedbe. Ujedno obaviti će se eksperiment koji će dati rezultate brzine obrade određenih upita.

6.2.1. Strukturiranje Neo4j baze podataka

Struktura i primjeri koda za relacijsku bazu podataka mogu se pronaći unutar poglavlja 3.4. i unutar poglavlja 6.1. Struktura Neo4j graf tipa baze podataka se izrađuje na sliku relacijske baze podataka. Pošto se graf tip baze podataka sastoji od čvorova, svaki čvor unutar Neo4j baze podataka će biti definiran kao svaki entitet relacijske baze podataka. Potrebno je napomenuti kako graf tipovi bazi podataka nemaju definiran izgled ili strukturu kao što ostali tipovi baza podataka imaju. U prethodnom poglavlju dokument tip baze podataka

imao je definiranu vrstu dokumenta te atribute unutar tog dokumenta isto kao što je relacijska baza imala entitete i atribute koji su definirali taj entitet. Kod Neo4j baze podatka svaki čvor može biti definiran na svoj način. Ako uzmemo entitet *MUSTERIJA* iz relacijske baze podataka sa njegovim atributima, možemo napraviti Neo4j bazu podataka koja će imati jedan čvor naziva *MUSTERIJA*, ali taj čvor neće imati definirane atribute jer se kod bazi podataka graf tipa atributi mijenjaju, to jest svaki čvor unutar grafa, bilo kakvog da je on tipa , može imati drugačije atribute. Iako svaki čvor može imati drugačije atribute, kod graf tipova baza podataka nastoji se poštovati konzistentnost između čvorova istog tipa. Kako bi uspješno mogli povezati Visual Studio sa Neo4j servisom potrebno je instalirati Neo4j *driver* unutar Visual Studio-a uz pomoć *NuGet-a* te je potrebno instalirati Neo4j servis na računalu. Nakon uspješne instalacije Neo4j-a, isto kao i kod MongoDB-a, dobijemo *connection string* od strane Neo4j servisa sa kojim se možemo povezati na njega. Kod Neo4j-a nije potrebno instalirati dodatne programe za lakše korištenje baze podataka već se sve može odraditi uz pomoć internetskog preglednika. Na slici 6.9. se može vidjeti izgled Neo4j-a na pregledniku. Potrebni podaci za povezivanje na bazu podataka su *Connection string*, *username* koji se dobije od strane Neo4j-a i *password* koji postavlja korisnik.

Slika 6.9. *Izgled sučelja Neo4j na pregledniku*

Kada se uspješno osposobi konekcija između Visual Studio-a i Neo4j baze podataka potrebno je kreirati određen *template* i funkcije kako bi mogli testirati brzinu obrade podataka.

Na primjeru koda 6.12. je prikazan *template* za čvor *MUSTERIJA* te njegovi atributi. Potrebno je napomenuti da iako svaki čvor može imati određen broj atributa te ne mora svaki čvor istog tipa imati iste atribute, u ovom testiranju će svaki čvor imati sve atribute jer se testira brzina obrade podataka i nije bitno koliko atributa svaki čvor ima.

```
public class MusterijaNeo
    {
        public string OIB { get; set; }
        public string Ime { get; set; }
        public string Prezime { get; set; }
        public DateTime Datum_rođenja { get; set; }
        public int Poštanski_broj { get; set; }
        public string Ime_grada { get; set; }
        public string Adresa { get; set; }
    }
```

Primjer koda 6.12. *Čvor Mušterija*

Nakon kreiranih klasa unutar Visual Studio-a za svaki čvor potrebno je svaki *template*, svaki *constraint* i svaki *assert* postaviti za bazu podataka. Na sljedećem primjeru koda može se vidjeti postavljanje *constraint-ova* i *assert-ova* za neke od čvorova. Ujedno se može vidjeti i *template* za čvor *RADI_NA* koji služi za povezivanje čvorova *DJELATNIK* i *POPRAVAK*.

```
public void CreateTemplates()
        {
            using (var driver = GraphDatabase.Driver(Neo4jUri,
AuthTokens.Basic(Neo4jUser, Neo4jPassword)))
            using (var session = driver.AsyncSession())
            {
                var musterijaTemplateQuery = "CREATE CONSTRAINT ON
(m:Musterija) ASSERT m.OIB IS UNIQUE";
                session.RunAsync(musterijaTemplateQuery);


                var radiNaTemplateQuery1 = "CREATE CONSTRAINT FOR ()-
[rn:RADI_NA]-() REQUIRE EXISTS(rn.ID_Popravka)";
                var radiNaTemplateQuery2 = "CREATE CONSTRAINT FOR ()-
[rn:RADI_NA]-() REQUIRE EXISTS(rn.ID_Djelatnika)";

                var radiNaTemplateQuery3 = @"CREATE INDEX FOR ()-
[rn:RADI_NA]-() ON (rn.relationshipId)";

                session.RunAsync(radiNaTemplateQuery1);
                session.RunAsync(radiNaTemplateQuery2);
                session.RunAsync(radiNaTemplateQuery3);
            }
        }
```

Primjer koda 6.13. *Template za Musterija i Radi_na čvorove*

6.2.2. Testiranje unosa, brisanja i promjene podataka

Testiranje Neo4j baze podataka bazira se na brzini unosa, brisanja i ažuriranja podataka. Razmatrana količina *dummy* podataka je 100, 1000 i 10000. Na sljedećim primjerima koda se mogu vidjeti tri funkcije koje se koriste za unos, brisanje i ažuriranje podataka te još jedna funkcija koja se koristi za kreiranje objekta tipa *MusterijaNeo* što je zapravo izrađeni *template* na primjeru koda 6.12.

```
public void InsertData(int count)
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            using (var driver = GraphDatabase.Driver(Neo4jUri,
AuthTokens.Basic(Neo4jUser, Neo4jPassword)))
            using (var session = driver.AsyncSession())
            {
                for (int i = 1; i <= count; i++)
                {
                    var musterija = GenerateMusterija(i);
                    var query = $"CREATE (:Musterija {{OIB:
'{musterija.OIB}', Ime: '{musterija.Ime}', Prezime: '{musterija.Prezime}',
Datum_rođenja: date('{musterija.Datum_rođenja:yyyy-MM-dd}'),
Poštanski_broj: {musterija.Poštanski_broj}, Ime_grada:
'{musterija.Ime_grada}', Adresa: '{musterija.Adresa}'}})";
                    session.RunAsync(query);
                }
            }
            stopwatch.Stop();
            Console.WriteLine("INSERT: " + stopwatch.ElapsedMilliseconds);
        }

public MusterijaNeo GenerateMusterija(int oib)
        {
            var musterija = new MusterijaNeo
            {
                OIB = oib.ToString(),
                Ime = "Dummy",
                Prezime = "Dummy",
                Datum_rođenja = new DateTime(1990, 1, 1),
                Poštanski_broj = 12345,
                Ime_grada = "Dummy",
                Adresa = "Dummy"
            };
            return musterija;
        }
```

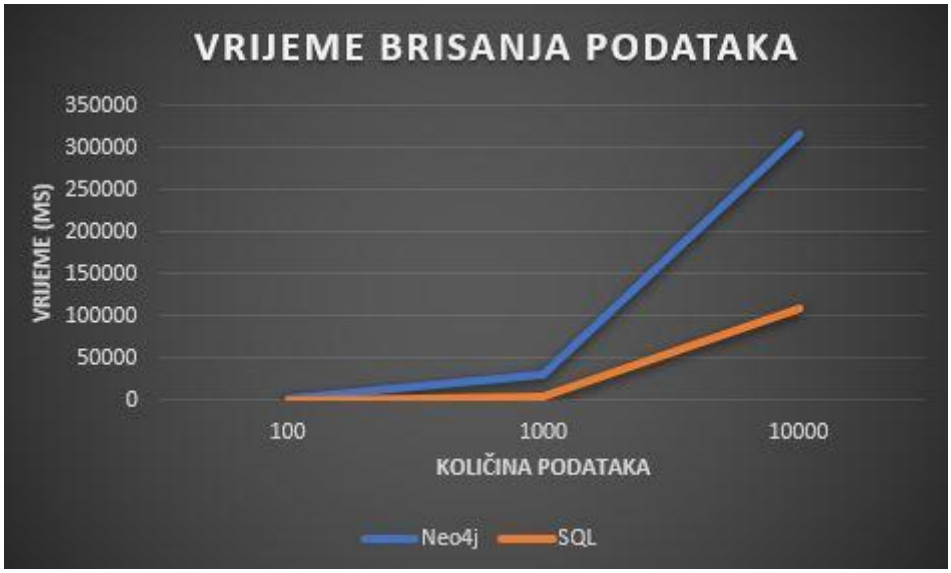
Primjer koda 6.14. *Funkcija za unos podataka i funkcija za kreiranje objekta*

```
public void UpdateData(int count)
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            using (var driver = GraphDatabase.Driver(Neo4jUri,
AuthTokens.Basic(Neo4jUser, Neo4jPassword)))
            using (var session = driver.AsyncSession())
            {
                for (int i = 1; i <= count; i++)
                {
                    var query = $@"
                    MATCH (m:Musterija)
                    WHERE m.OIB = '{i}'
                    SET m.Ime = 'Dominik'";
                    session.RunAsync(query);
                }
            }
            stopwatch.Stop();
            Console.WriteLine("UPDATE: " + stopwatch.ElapsedMilliseconds);
        }
        public void DeleteData(int count)
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            using (var driver = GraphDatabase.Driver(Neo4jUri,
AuthTokens.Basic(Neo4jUser, Neo4jPassword)))
            using (var session = driver.AsyncSession())
            {
                for (int i = 1; i <= count; i++)
                {
                    var deleteQuery = $"MATCH (n) WHERE ID(n) = {i} DELETE
n";
                    session.RunAsync(deleteQuery);
                }
            }
            stopwatch.Stop();
            Console.WriteLine("DELETE: " + stopwatch.ElapsedMilliseconds);
        }
```

Primjer koda 6.15. *Funkcija za ažuriranje i funkcija za brisanje podataka*

Funkcija za unos i brisanje jednostavno unose i brišu mušterije ovisno o oib-u koji se postavlja pri unosu podataka od jedan pa nadalje. Kod ažuriranja podataka, podatak koji se ažurira je ime mušterije koje se postavlja na Dominik kod svake mušterije. Koristeći ove funkcije te uz pomoć *Stopwatch* objekta su izračunati rezultati koji su dobiveni na sljedećim slikama.

Slika 6.10 Rezultat mjerenja unosa podataka



Slika 6.11. *Rezultat mjerenja brisanja podataka*

Slika 6.12. *Rezultat mjerenja ažuriranja podataka*

Prema prijašnjim slikama može se vidjeti kako je SQL baza podataka znatno brža što se tiče unosa, brisanja i ažuriranja podataka od Neo4j baze podataka. Najveća razlika kod ovog testiranja je ta što Neo4j baza podataka je vremenski za unos, brisanje i ažuriranje ista, to jest vrijeme potrebno za unos, brisanje i ažuriranje bilo koje količine podataka je isto dok za SQL bazu podataka vremena su drugačija.

6.2.3. Testiranje nad određenim upitima

U ovom dijelu testiranja testirati će se određeni upiti za Neo4j i SQL bazu podataka. Upiti su isti kao i u testiranju iz poglavlja 6.1. Testiranje će se raditi nad tri upita te na 5000 podataka. Sve vrijednosti potrebne za testiranje kao što su ime, prezime i ime grada će biti postavljeni nasumično dok će ostali podatci biti postavljeni kao *dummy* podatci. Upiti koji se koriste u testiranju su:

- **Q1** - dohvaćanje svih mušterija čije je ime Dominik
- **Q2** - brisanje svih mušterija čije je ime Dominik ili se prezivaju Vidović
- **Q3** - ažuriranje adrese svih mušterija čije je ime Dominik ili se prezivaju Vidović ili dolaze iz Osijeka

Kako bi uspješno izvršili upite potrebno je prvo postaviti podatke u bazu podataka, a za to će se koristiti funkcija koja je prikazana u primjeru koda 6.16.

```
public void QueryInsert()
        {
            using (var driver = GraphDatabase.Driver(Neo4jUri,
AuthTokens.Basic(Neo4jUser, Neo4jPassword)))
            using (var session = driver.AsyncSession())
            {
                for (int i = 1; i <= 5000; i++)
                {
                    var musterija = GenerateMusterijaQuery(i);
                    var query = $"CREATE (:Musterija {{OIB:
'{musterija.OIB}', Ime: '{musterija.Ime}', Prezime: '{musterija.Prezime}',
Datum_rođenja: date('{musterija.Datum_rođenja:yyyy-MM-dd}'),
Poštanski_broj: {musterija.Poštanski_broj}, Ime_grada:
'{musterija.Ime_grada}', Adresa: '{musterija.Adresa}'}})";
                    session.RunAsync(query);
                }
            }

        }

        public MusterijaNeo GenerateMusterijaQuery(int oib)
        {
            Random random = new Random();
            List<string> randomImeValues = new List<string> { "Dominik",
"Random1", "Random2", "Random3", "Random4", "Random5", "Random6",
"Random7", "Random8", "Random9" };
            List<string> randomPrezimeValues = new List<string> {
"Vidović", "Random1", "Random2", "Random3", "Random4", "Random5",
"Random6", "Random7", "Random8", "Random9" };
            List<string> randomImeGradaValues = new List<string> {
"Osijek", "Random1", "Random2", "Random3", "Random4", "Random5", "Random6",
"Random7", "Random8", "Random9" };
            var musterija = new MusterijaNeo
            {
                OIB = oib.ToString(),
                Ime = randomImeValues[random.Next(randomImeValues.Count)],
                Prezime =
randomPrezimeValues[random.Next(randomPrezimeValues.Count)],
                Datum_rođenja = new DateTime(1990, 1, 1),
                Poštanski_broj = 12345,
                Ime_grada =
randomImeGradaValues[random.Next(randomImeGradaValues.Count)],
                Adresa = "Dummy"
            };
            return musterija;
        }
```

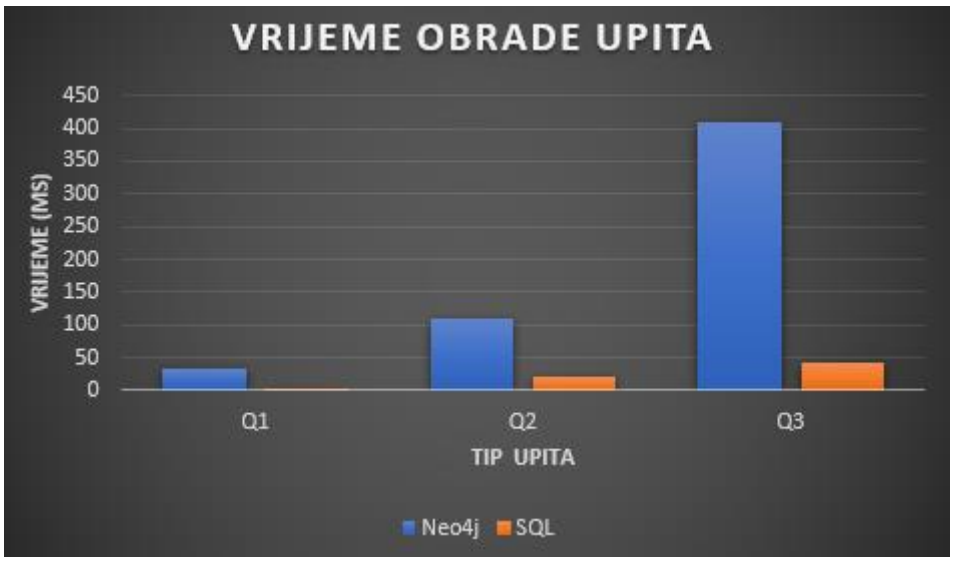
Primjer koda 6.16. *Funkcija za unos 5000 podataka*

Nakon unosa 5000 podataka tipa *MUSTERIJA* te nasumično postavljanje podataka za ime, prezime i ime grada se kreiraju upiti pod **Q1**, **Q2** i **Q3**. Za kreiranje upita i mjerenje vremena koriste se funkcije prikazane na sljedećem primjeru koda.

```
public void FirstQuery()
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            var musterijaData = new List<MusterijaNeo>();

            using (var driver = GraphDatabase.Driver(Neo4jUri,
AuthTokens.Basic(Neo4jUser, Neo4jPassword)))
            using (var session = driver.AsyncSession())
            {
                var query = $"MATCH (m:Musterija {{Ime: 'Dominik'}}) RETURN
m";
                var result = session.RunAsync(query);
            }
            stopwatch.Stop();
            Console.WriteLine("Q1: " + stopwatch.ElapsedMilliseconds);
        }

        public void SecondQuery()
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            using (var driver = GraphDatabase.Driver(Neo4jUri,
AuthTokens.Basic(Neo4jUser, Neo4jPassword)))
            using (var session = driver.AsyncSession())
            {
                var query = $"MATCH (m:Musterija {{Ime: 'Dominik'}}) WHERE
m.Prezime = 'Vidović' DELETE m";
                session.RunAsync(query);
            }
            stopwatch.Stop();
            Console.WriteLine("Q2: " + stopwatch.ElapsedMilliseconds);
        }
public void ThirdQuery()
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            using (var driver = GraphDatabase.Driver(Neo4jUri,
AuthTokens.Basic(Neo4jUser, Neo4jPassword)))
            using (var session = driver.AsyncSession())
            {
                var query = $"MATCH (m:Musterija) WHERE m.Ime = 'Dominik'
OR m.Prezime = 'Vidović' OR m.Ime_grada = 'Osijek' SET m.Adresa = 'nova
adresa'";
                session.RunAsync(query);
            }
            stopwatch.Stop();
            Console.WriteLine("Q3: " + stopwatch.ElapsedMilliseconds);
        }
```

Primjer koda 6.17. *Funkcije za upite*

Uz pomoć funkcija prikazanih na primjeru koda 6.17. možemo testirati bazu podataka u smislu potrebnog vremena za obradu upita. Rezultati usporedbe SQL i Neo4j baze podataka prikazani su na slici 6.13.



Slika 6.13. *Rezultat obrade upita*

Prema slici 6.13. može se zaključiti kako je relacijska baza podataka znatno brža u obradi upita (barem što se tiče ovih jednostavnih upita). Vrijeme potrebno za obradu upita nad 5000 podataka je skoro dva puta veće za Neo4j kod svakih od upita dok je kod trećeg upita vrijeme potrebno za obradu čak i pet puta veće.

6.3. Usporedba relacijske i Apache Cassandra

U ovom poglavlju će se usporediti relacijska baza podataka i Apache Cassandra baza podataka koja se bazira na tipu širokih stupaca. Temeljna usporedba kao i objašnjena razlike svake od baza podataka dana su tablicom 6.3.

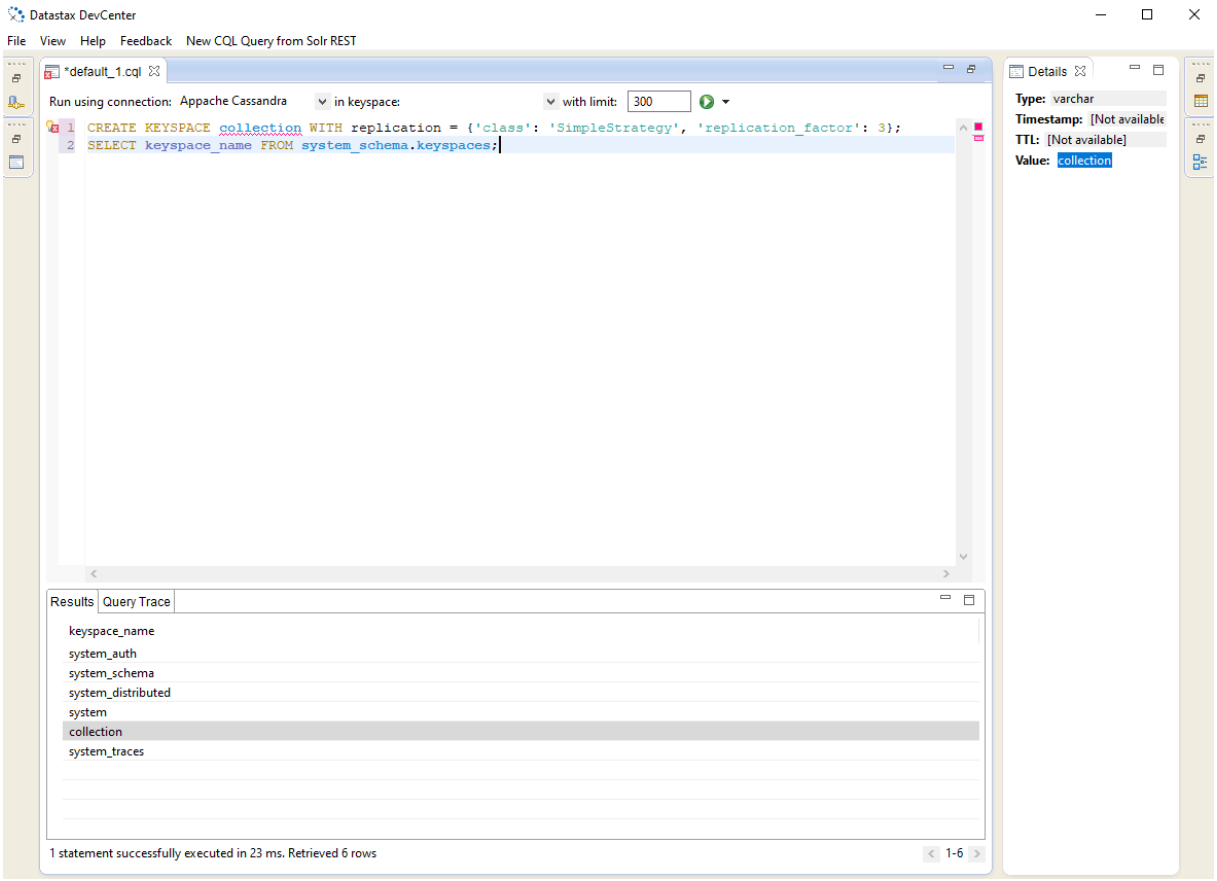
Tablica 6.3. *Usporedba relacijske i Appache Cassandra baze podataka*

Apache Cassandra	**Relacijske baze podataka**
Podržava jednostavan upitni jezik.	Podržavaju moćan upitni jezik.
U konačnici je dosljedna. Ima BASE svojstva (engl. ***B****asically****A****vailable*, ***S****oft-state* i ***E****ventualconsistency*).	Imaju ACID svojstva (nedjeljivost (engl. *Atomicity*), dosljednost (engl. *Consistency*), izolacija (engl. *Isolation*) i izdržljivost (engl. *Durability*)).
Podržava transakcije, no ne zadržava ACID svojstva baze podataka.	Podržavaju transakcije i zadrţavaju ACID svojstva baze podataka.
Tablica je lista parova ključ-vrijednost (RED x KLJUČ STUPCA x VRIJEDNOST STUPCA).	Tablica se sastoji od polja koje sadrţi druga polja. (RED x STUPAC).
Tablice ili porodice stupaca su entiteti (engl. *keyspace).*	Tablice su entiteti baze podataka.
Red je jedinica replikacije.	Red je individualan zapis.
Stupac je spremnik podataka.	Stupac predstavlja atribut relacije
Svaki red ne mora imati popunjene sve definirane stupce.	Pri unosu podataka svaki red mora biti popunjen (barem s ništavnim vrijednostima).
Osim predefinirani tipova podataka, korisnik moţe stvarati svoje tipove podataka.	Mogu koristiti samo predefinirane tipove podataka.

Za provođenje testiranja brzine unosa, brisanja i ažuriranja podataka potrebno je prvo preuzeti Apache Cassandra sa interneta te instalirati na računalo. Nakon što se Apache Cassandra uspješno instalira na računalo potrebno je preuzeti program pod nazivom DevCenter koji služi kao pojednostavljeno sučelje za rad nad bazom podataka. Kako bi se napisani kod mogao koristiti i implementirati za ovu bazu podataka potrebno je unutar Visual Studio-a instalirati *Cassandra.Driver* te ga ubaciti kao biblioteku unutar koda.

6.3.1. Strukturiranje Apache Cassandra baze podataka

Kod kreiranja NoSQL baze podataka širokih stupaca potrebno je napraviti tablice unutar kojih će se podaci spremiti. Struktura Apache Cassandra se sastoji od kolekcija i tablica unutar tablica. Tablice se sastoje od stupaca i svaki stupac predstavlja jedan podataka. Svaki podataka može a i ne mora imati iste atribute kao ostali stupci. Kako bi se kreirala baza podataka unutar Visual Studio-a potrebno je prvo osposobiti konekciju između Visual Studio-a i baze podataka. Prije nego što se osposobi konekcija potrebno je odrediti kolekciju u koju će se spremati podaci. Kolekcija se može kreirati uz pomoć naredbenog retka (engl. *Command prompt*) ili PowerShell-a. Pošto već postoji program sa pojednostavljenim sučeljem DevCenter koristiti će se taj program. Izgled DevCenter-a se može vidjeti na sljedećoj slici.



Slika 6.14. *DevCenter*

Na slici 6.14. je prikazano sučelje za korištenje baze podataka Apache Cassandra. Prvo je potrebno uključiti bazu podataka (pošto Apache Cassandra se ne tretira kao servis na računalu koji je stalno uključen i stalno sluša) uz pomoć naredbenog retka te nakon toga je potrebno povezati se uz pomoć sljedećih varijabli u DevCenter-u:

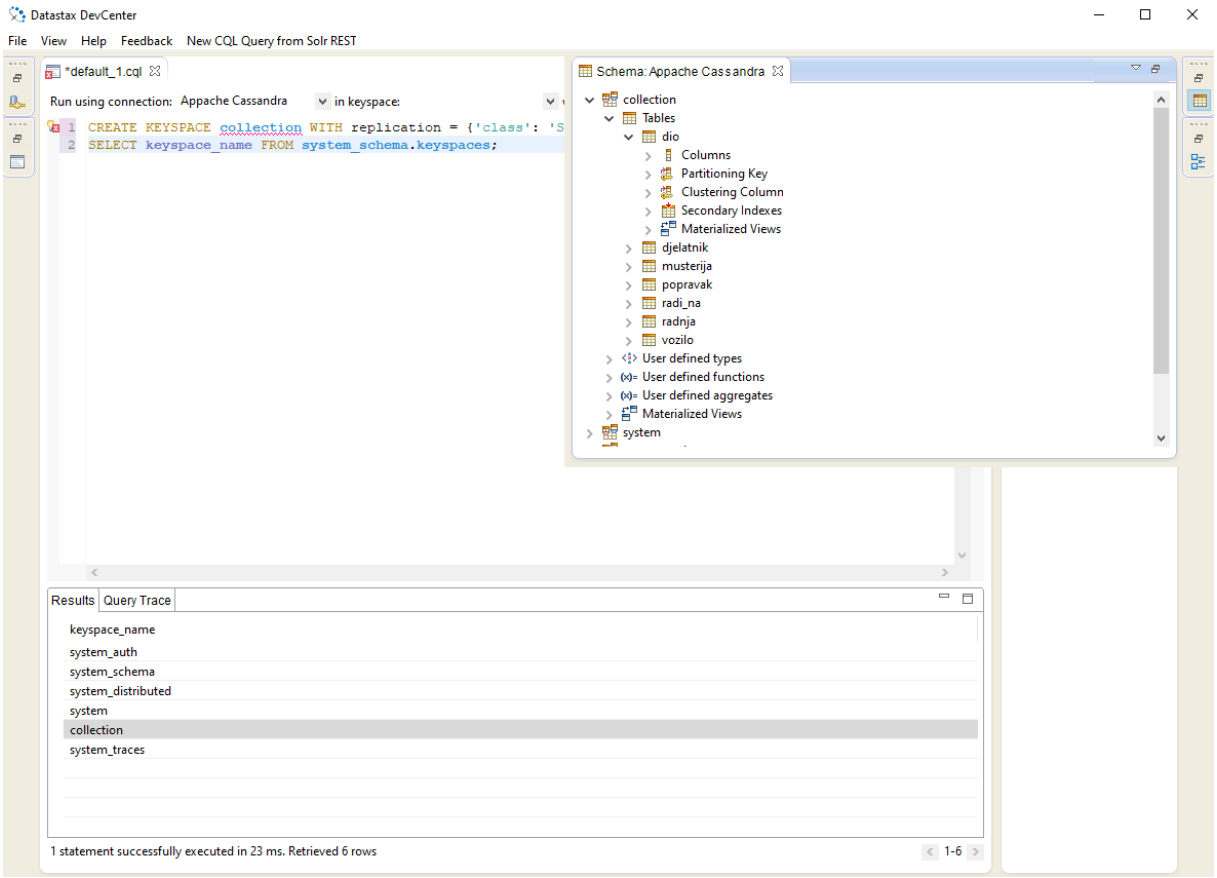
- ***Contact hosts*** **–** predstavlja IP adresu kreiranog čvora kod instalacije baze podataka, najčešće je 127.0.0.1
- ***Native protocol port*** – port na kojem baza podataka sluša, najčešće je to port 9042

Nakon što se DevCenter uspješno poveže na instaliranu bazu potrebno je kreirati kolekciju. Na slici 6.14. se mogu vidjeti naredbe korištene za kreiranje kolekcije te ujedno naredba za prikaz svih kolekcija unutar baze podataka. U sljedećem primjeru koda može se vidjeti funkcija za kreiranje potrebnih tablica unutar kolekcije te povezivanje na bazu podataka.

```
class Cassandra
    {
        private Cluster cluster;
        private ISession session;

        public void Connect()
        {
            cluster =
Cluster.Builder().AddContactPoints("127.0.0.1").Build();
            session = cluster.Connect("collection");
        }
        public void Close()
        {
            session?.Dispose();
            cluster?.Dispose();
        }


        public void CreateTables()
        {
            session.Execute("CREATE TABLE Musterija (" +
            "OIB text PRIMARY KEY," +
            "Ime text," +
            "Prezime text," +
            "Datum_rodenja timestamp," +
            "Postanski_broj int," +
            "Ime_grada text," +
            "Adresa text" +
        ");");
            session.Execute("CREATE TABLE Vozilo (" +
            "Registarska_oznaka text PRIMARY KEY," +
            "ID_musterije text," +
            "Naziv_proizvodaca text," +
            "Marka_vozila text," +
            "Broj_mjesta int" +
        ");");
```

Primjer koda 6.18. *Funkcija za povezivanje baze podataka i dio funkcije za kreiranje tablica*

U primjeru koda 6.18. se može vidjeti funkcija za povezivanje na bazu podataka koja zahtjeva već prije navedenu IP adresu 127.0.0.1 te naziv kolekcije koja je kreirana unutar DevCenter-a (slika 6.14.). Funkcija *CreateTables()* koristi upite za kreiranje tablica *MUSTERIJA, VOZILO* i druge. Tablice su kreirane na sliku relacijske baze podataka. Nakon pokretanja ovih funkcija kreiraju se tablice unutar kolekcije gdje možemo spremati podatke. Na slici 6.15. se može vidjeti kreirana kolekcija i tablice unutar te kolekcije.



Slika 6.15. *Prikaz kolekcije i tablice unutar DevCenter-a*

Nakon uspješnog instaliranja, povezivanja i kreiranja Apache Cassandra NoSQL baze podataka širokih stupaca može se testirati brzina unosa, brisanja i ažuriranja podataka.

6.3.2. Testiranje unosa, brisanja i promjene podataka

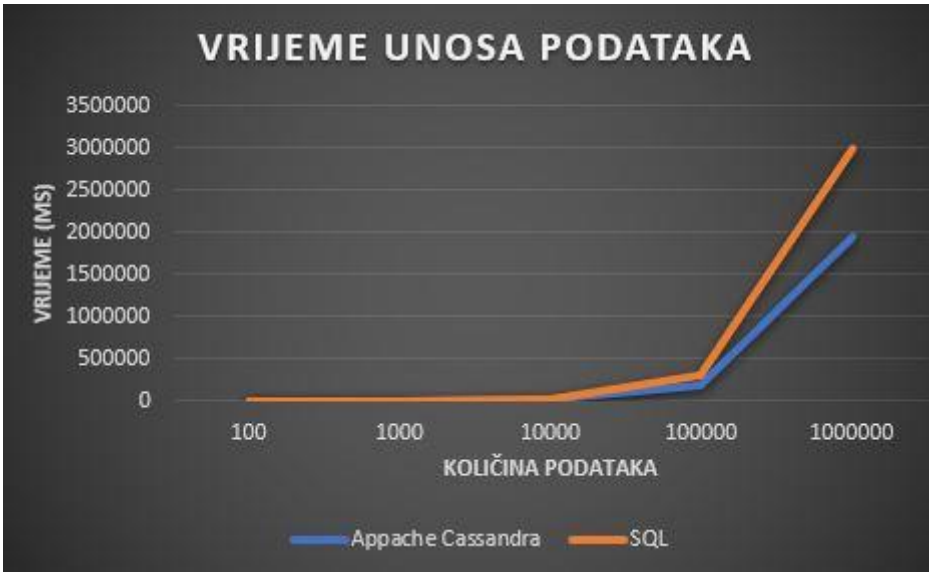
U ovom odjeljku će se testirati brzina ili potrebno vrijeme za unos, brisanje i ažuriranje podataka. Količina podataka nad kojom će se obaviti testiranje sastoji se od 100, 1000, 10000, 100000 i 1000000 *dummy* podataka. Testiranje će se, kao i u prijašnjim testiranjima, provoditi sa relacijskom bazom podataka. Za detaljniji izgled ili strukturu

relacijske baze podataka pogledati odjeljak 3.4. ili 6.1. Kako bi se testirao unos, brisanje i ažuriranje podataka unutar Visual Studio-a potrebno je kreirati funkcije koje će unositi, brisati i ažurirati podatke te pritom mjeriti vrijeme od početka unosa do završetka unosa podataka. Na sljedećem primjeru koda se mogu vidjeti funkcije korištene za odrađivanje testiranja.

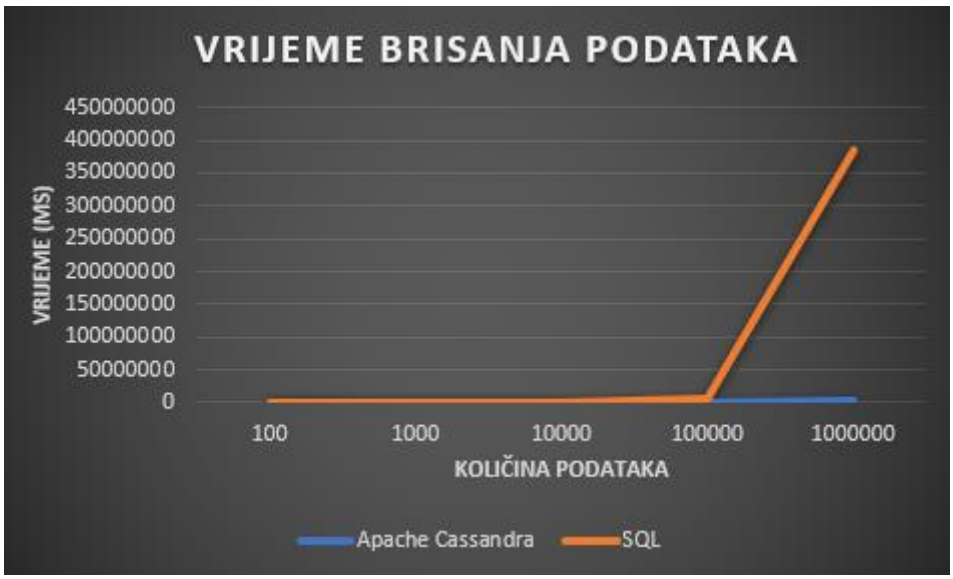
```
public void InsertData(int amount)
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            for(int i = 1;i<=amount;i++)
            {
                string oib = $"OIB{i}";
                string ime = "Dummy";
                string prezime = "Dummy";
                DateTime datumRodjenja = DateTime.Now;
                int postanskiBroj = 10000 + i;
                string imeGrada = "Dummy";
                string adresa = "Dummy";

                string insertQuery = $"INSERT INTO Musterija (OIB, Ime,
Prezime, Datum_rodenja, Postanski_broj, Ime_grada, Adresa) " +
                $"VALUES ('{oib}', '{ime}', '{prezime}',
'{datumRodjenja:yyyy-MM-dd HH:mm:ss}', {postanskiBroj}, '{imeGrada}',
'{adresa}');";
                session.Execute(insertQuery);
            }
            stopwatch.Stop();
            Console.WriteLine("Insert time: " +
stopwatch.ElapsedMilliseconds);
        }

        public void DeleteData(int amount)
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            for (int i = 1; i <= amount; i++)
            {
                string oib = $"OIB{i}";
                string deleteQuery = $"DELETE FROM Musterija WHERE OIB =
'{oib}';";
                session.Execute(deleteQuery);
            }
            stopwatch.Stop();
            Console.WriteLine("Delete time: " +
stopwatch.ElapsedMilliseconds);
        }
public void UpdateData(int amount)
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            for(int i = 1; i<=amount;i++)
            {
                string oib = $"OIB{i}";
                string updateQuery = $"UPDATE Musterija SET Ime = 'Dominik'
WHERE OIB = '{oib}';";
                session.Execute(updateQuery);
            }
            stopwatch.Stop();
            Console.WriteLine("Update time: " +
stopwatch.ElapsedMilliseconds);
        }
```

Primjer koda 6.19. *Funkcije za unos, brisanje i ažuriranje podataka*

Svaka funkcija se koristi za unos, brisanje i ažuriranje određenog broja podataka. Svaka funkcija koristi objekt *Stopwatch* koji se koristio i u prijašnjim testiranjima kako bi se odredilo potrebno vrijeme za izvršenje funkcije. Funkcija *InsertData* unosi određeni broj podataka u bazu podataka uz pomoć *insertQuery-a*, *DeleteData* briše određenu količinu podataka uz pomoć *deleteQuery-a* dok *UpdateData* ažurira određenu količinu podataka uz pomoć *updateQuery-a* koji postavlja ime na Dominik svakog tipa *MUSTERIJA*. Rezultati usporedbe relacijske baze podataka sa Apache Cassandra može se vidjeti na sljedećim slikama.



Slika 6.16. *Rezultat brzine unosa podataka*



Slika 6.17. *Rezultat brzine brisanja podataka*

Slika 6.18. *Rezultat brzine ažuriranja podataka*

Prema prošlim slikama može se vidjeti krajnji rezultat testiranja brzine unosa, brisanja i ažuriranja podataka. Što se tiče unosa podataka može se vidjeti kako je relacijskoj bazi podataka potrebno više vremena kod unosa podataka nego što je potrebno Apache Cassandra. Velika razlika u vremenu se može vidjeti kod unosa maksimalno postavljene količine podataka pri čemu se može zaključiti kako relacijska baza podataka, pri većoj količini podataka, zahtijeva više vremena za obradu. Kod brisanja podataka može se vidjeti kako je relacijskoj bazi podataka potrebno znatno više vremena nego što je potrebno Apache Cassandra bazi podataka. Relacijskoj bazi podataka je potrebno čak sat vremena za unos milijun podataka dok je NoSQL bazi podataka potrebno nekoliko minuta za brisanje podataka. Kod ažuriranja podataka, određene su količine podataka od 100, 1000 i 10000 jer se pri testiranju moglo primjetiti kako je relacijska baza podataka zahtijevala jako puno vremena za obradu 10000 podataka dok bi za obradu više podataka od 10000 bilo potrebno znatno više vremena. Pri ažuriranju podataka relacijska baza podataka se opet pokazala sporija od NoSQL baze podataka širokih stupaca.

Kako bi zaključili ova testiranja nad relacijskim i NoSQL bazama podataka potrebno je još odraditi određene upite nad Apache Cassandra bazom podataka kako bi mogli vidjeti koliko je vremena potrebno toj bazi podataka kako bi obradila određene upite i na kraju dala određene rezultate.

6.3.3. Testiranje nad određenim upitima

U ovom odjeljku će se odraditi testiranje nad određenim upitama. Određeni upiti su rađeni u svakoj usporedbi baza podataka kao što su oni u odjeljku 6.1.3. i 6.2.3. Upiti će se raditi nad 5000 podataka kao i u prošlim testiranjima. Upiti koji su korišteni su isti upiti kao i u ostalim istraživanjima a to su:

- **Q1** - dohvaćanje svih mušterija čije je ime Dominik
- **Q2** - brisanje svih mušterija čije je ime Dominik ili se prezivaju Vidović
- **Q3** - ažuriranje adrese svih mušterija čije je ime Dominik ili se prezivaju Vidović ili dolaze iz Osijeka

Kako bi uspješno mogli odraditi testiranje potreban je unos od 5000 podataka u bazu podataka. Za unos koristiti će se posebna funkcija koja se može vidjeti na primjeru koda 6.19.

```
public void InsertRealData()
        {
            Random random = new Random();
            List<string> randomImeValues = new List<string> { "Dominik",
"Random1", "Random2", "Random3", "Random4", "Random5", "Random6",
"Random7", "Random8", "Random9" };
            List<string> randomPrezimeValues = new List<string> {
"Vidović", "Random1", "Random2", "Random3", "Random4", "Random5",
"Random6", "Random7", "Random8", "Random9" };
            List<string> randomImeGradaValues = new List<string> {
"Osijek", "Random1", "Random2", "Random3", "Random4", "Random5", "Random6",
"Random7", "Random8", "Random9" };
            for (int i = 1;i<=5000;i++)
            {
                string oib = $"OIB{i}";
                string ime =
$"{randomImeValues[random.Next(randomImeValues.Count)]}";
                string prezime =
$"{randomPrezimeValues[random.Next(randomImeValues.Count)]}";
                DateTime datumRodjenja = DateTime.Now;
                int postanskiBroj = 10000 + i;
                string imeGrada =
$"{randomImeGradaValues[random.Next(randomImeValues.Count)]}";
                string adresa = "Dummy";

                string insertQuery = $"INSERT INTO Musterija (OIB, Ime,
Prezime, Datum_rodenja, Postanski_broj, Ime_grada, Adresa) " +
                $"VALUES ('{oib}', '{ime}', '{prezime}',
'{datumRodjenja:yyyy-MM-dd HH:mm:ss}', {postanskiBroj}, '{imeGrada}',
'{adresa}');";
                session.Execute(insertQuery);
            }
        }
```

Primjer koda 6.20. *Funkcija za unos 5000 podataka*

Na primjeru koda 6.20. se može vidjeti funkcija za unos 5000 podataka. Funkcija je slična kao funkcija korištena u prijašnjim testiranja nad MongoDB i Neo4j bazom podataka. Funkcija koristi tri liste veličine deset. Svaka lista ima jednu stvarnu vrijednost i devet *dummy* vrijednosti. Prva lista prikazuje listu imena mušterija, lista sadrži jedno ime Dominik, a ostala imena su *dummy* imena. Druga lista prikazuje vrijednosti prezimena gdje je jedna vrijednost Vidović, a treća lista prikazuje vrijednosti imena gradova gdje je jedna vrijednost Osijek. Unutar petlje vrijednosti se nasumično dodaju u tablicu.

Nakon unosa ovih podataka potrebno je kreirate funkcije za upite za bazu podataka. Na sljedećim primjerima koda mogu se vidjeti kreiranje funkcije za upite u bazu podataka.

```
public void FirstQuery()
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            string selectQuery = "SELECT * FROM Musterija WHERE Ime =
'Dominik' ALLOW FILTERING;";
            session.Execute(selectQuery);
            stopwatch.Stop();
            Console.WriteLine("First query: " +
stopwatch.ElapsedMilliseconds);
        }
        public void SecondQuery()
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            string selectQuery = "SELECT OIB FROM Musterija WHERE Ime =
'Dominik' ALLOW FILTERING;";
            RowSet result = session.Execute(selectQuery);
            var oibList = result.Select(row =>
row.GetValue<string>("oib")).ToList();

            foreach (var oib in oibList)
            {
                string deleteQuery = $"DELETE FROM Musterija WHERE OIB =
'{oib}';";
                session.Execute(deleteQuery);
            }

            selectQuery = "SELECT OIB FROM Musterija WHERE Prezime =
'Vidović' ALLOW FILTERING;";
            result = session.Execute(selectQuery);
            oibList = result.Select(row =>
row.GetValue<string>("oib")).ToList();

            foreach (var oib in oibList)
            {
                string deleteQuery = $"DELETE FROM Musterija WHERE OIB =
'{oib}';";
                session.Execute(deleteQuery);
            }
            stopwatch.Stop();
            Console.WriteLine("Second query: " +
stopwatch.ElapsedMilliseconds);
        }
```

Primjer koda 6.21. *Funkcije za upite Q1, Q2 u bazu podataka*

Na primjeru koda 6.21. se nalaze dvije funkcije, prva funkcija se koristi za prvi upit u bazu podataka **Q1** dok se druga funkcija koristi za drugi upit u bazu podataka ili **Q2**. Upiti nisu jednostavni kao u ostalim bazama podataka jer Apache Cassandra zahtijeva da se upiti pišu red po red, to jest ne mogu se u jednom redu izabrati sve mušterije čije je ime Dominik ili prezime Vidović te je ujedno potrebno prvo dohvatiti OIB ili primarni ključ svake mušterije te onda raditi upit. Na sljedećem primjeru koda se nalazi i funkcija za dohvaćanje OIB-a.

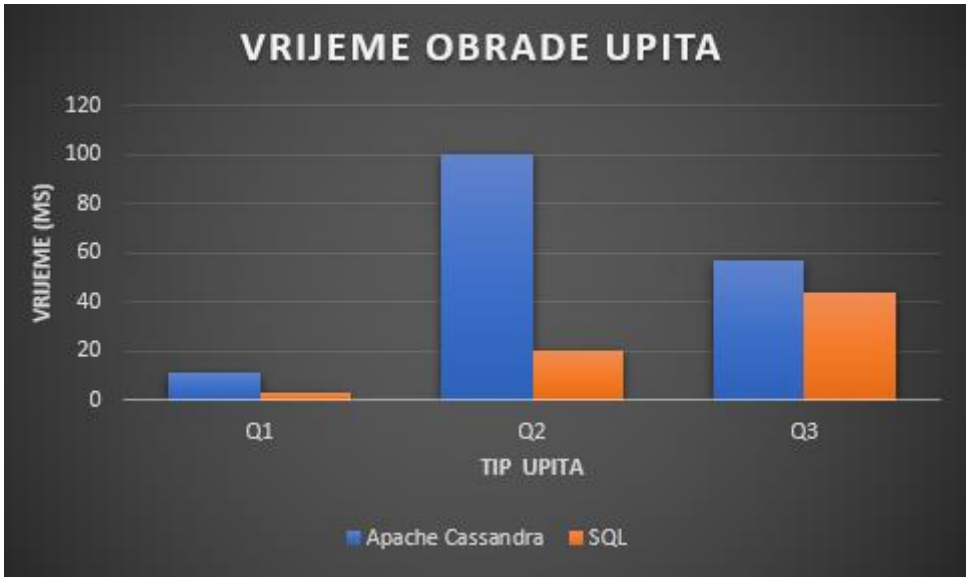
```
public void ThirdQuery()
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            string selectQuery = "SELECT OIB FROM Musterija WHERE Ime =
'Dominik' ALLOW FILTERING;";
            RowSet result = session.Execute(selectQuery);
            var oibList = result.Select(row =>
row.GetValue<string>("oib")).ToList();

            string updateQuery = $"UPDATE Musterija SET Adresa = 'nova
adresa' WHERE OIB IN ({GetOibListString(oibList)});";
            session.Execute(updateQuery);

            selectQuery = "SELECT OIB FROM Musterija WHERE Prezime =
'Vidović' ALLOW FILTERING;";
            result = session.Execute(selectQuery);
            oibList = result.Select(row =>
row.GetValue<string>("oib")).ToList();

            updateQuery = $"UPDATE Musterija SET Adresa = 'nova adresa'
WHERE OIB IN ({GetOibListString(oibList)});";
            session.Execute(updateQuery);

            selectQuery = "SELECT OIB FROM Musterija WHERE Ime_grada =
'Osijek' ALLOW FILTERING;";
            result = session.Execute(selectQuery);
            oibList = result.Select(row =>
row.GetValue<string>("oib")).ToList();

            updateQuery = $"UPDATE Musterija SET Adresa = 'nova adresa'
WHERE OIB IN ({GetOibListString(oibList)});";
            session.Execute(updateQuery);
            stopwatch.Stop();
            Console.WriteLine("Third query: " +
stopwatch.ElapsedMilliseconds);
        }
        private string GetOibListString(List<string> oibList)
        {
            string oibValues = string.Join(",", oibList.Select(oib =>
$"'{oib}'"));
            return oibValues;
        }
```

Primjer koda 6.22. *Funkcija za upit Q3 i funkcija za dohvaćanje OIB-a*

Na primjeru koda 6.22. se može vidjeti funkcija za treći upit u bazu podataka ili **Q3**, ujedno se unutar primjera koda nalazi i funkcija za dohvaćanje OIB-a. Funkcija prvo dohvaća OIB-e svih mušterija čije je ime Dominik ili prezime Vidović ili dolaze onih koji dolaze iz Osijeka. Nakon što se dohvati svaki OIB tek onda se kreira upit i ažurira se adresa mušterije

čije je ime Dominik, te upit za ažuriranje adrese gdje je prezime Vidović i zadnji upit za mušterije koje dolaze iz Osijeka.

Rezultate usporedbe relacijske baze podataka i baze podataka Apache Cassandra sa određenim upitima može se vidjeti na slici 6.19.



Slika 6.19. *Rezultat obrade određenih upita*

Kako se može vidjeti prema slici 6.19. Apache Cassandra je sporija od relacijske baze podataka što se tiče obrade određenih upita (barem ovih jednostavnih upita). Kod upita za dohvaćanje svih mušterija čije je ime Dominik relacijska baza podataka je dvostruko brža od Apache Cassandra. Što se tiče drugog upita gdje se brišu mušterije čije je ime Dominik ili prezime Vidović relacijska baza podataka je znatno brža od Apache Cassandra. Za treći upit može se reći da je vrijeme obrade upita slično kao i kod prvog upita samo što je potrebno vrijeme obrade malo više nego kod prvog upita što je logično pošto je upit složeniji od prvog

7. ZAKLJUČAK

U ovom završnom radu smo usporedili relacijske baze podataka s Apache Cassandrom, MongoDB i Neo4j kako bismo istražili njihove karakteristike, prednosti i primjenu u različitim scenarijima. Relacijske baze podataka su temelj moderne aplikacijske razvojne paradigme. One koriste tabličnu strukturu s čvrsto definiranim shemama, što omogućuje dosljednost podataka, strogo provjeravanje integriteta i napredne mogućnosti upita pomoću SQL-a. Relacijske baze su najprikladnije za složene transakcijske sustave poput financijskih aplikacija i e-trgovine. S druge strane, Apache Cassandra, MongoDB i Neo4j su popularne NoSQL baze podataka koje nude alternativne pristupe skladištenju podataka. Apache Cassandra je široki stupac (engl. *wide-column)* NoSQL baza podataka koja je iznimno skalabilna i otporna na kvarove. Cassandra koristi fleksibilni model podataka bez *sheme* i raspodjeljuje podatke horizontalno na više čvorova u klasteru. To je idealno za velike količine podataka i visoku dostupnost, često se koristi u aplikacijama za analitiku u stvarnom vremenu, *IoT* uređajima i sustavima za praćenje vremenskih serija. MongoDB je dokumentna NoSQL baza podataka koja koristi JSON-slične dokumente za pohranu podataka. Omogućuje fleksibilnost u strukturi podataka i nema stroge sheme. MongoDB je popularan izbor za aplikacije s promjenjivom strukturom podataka, brzim prototipiranjem i agilnim razvojem. Također nudi bogate mogućnosti upita i indeksiranja. Neo4j je grafička NoSQL baza podataka koja je posebno dizajnirana za pohranu i upravljanje podacima u obliku grafova. Grafovi su pogodni za modeliranje složenih odnosa i povezanosti među podacima. Neo4j pruža moćan jezik upita za rad s grafovima i koristi optimizirane algoritme za brzu obradu upita.

Svaka od ovih baza podataka ima svoje prednosti i specifične primjene. U odabiru baze podataka, važno je uzeti u obzir karakteristike aplikacije, zahtjeve za skalabilnošću, konzistentnošću podataka, složenost upita i fleksibilnost strukture podataka. U zaključku, relacijske baze podataka ostaju snažan izbor za aplikacije koje zahtijevaju dosljednost i cjelovitost podataka, dok Apache Cassandra, MongoDB i Neo4j pružaju alternativne pristupe skladištenju podataka s naglaskom na skalabilnost, fleksibilnost ili graf strukturu. Odabir prave baze podataka ovisi o specifičnim zahtjevima aplikacije i prioritetima u pogledu performansi, skalabilnosti i fleksibilnosti podataka.

LITERATURA

[1] Marko Brica, USPOREDBA GRAF I RELACIJSKIH BAZA PODATAKA, Repozitorij Elektrotehničkog fakulteta Osijek, Osijek, 2018., dostupno na: https://repozitorij.etfos.hr/islandora/object/etfos%3A1744/datastream/PDF/view [pristupljeno 01.05.2023.]

[2] Gabrijela Kramar, USPOREDBA APACHE CASSANDRA I RELACIJSKE BAZE PODATAKA NA PRIMJERU SREDNJE ŠKOLE, Repozitorij Elektrotehničkog fakulteta Osijek, Osijek, 2016., dostupno na: https://repozitorij.etfos.hr/islandora/object/etfos%3A916/datastream/PDF/view [pristupljeno 01.05.2023.]

[3] Google Cloud, What is a relational database?, Google LLC, Mountain View, California, USA, dostupno na: https://cloud.google.com/learn/what-is-a-relational-database [pristupljeno 01.05.2023.]

[4] TechTarget, Relational database , TechTarget, Newton, Massachusetts, USA, dostupno na: https://www.techtarget.com/searchdatamanagement/definition/relational-database [pristupljeno 01.05.2023.]

[5] Database Design, Introduction to Relational Database Structure, Database Design, dostupno na: https://www.relationaldbdesign.com/database-design/module2/intro-relational-database-structure.php [pristupljeno 01.05.2023.]

[6] DatabaseTown, Relational Database: Benefits and Limitations, DatabaseTown, dostupno na: https://databasetown.com/relational-database-benefits-and-limitations/ [pristupljeno 01.05.2023.]

[7] The Crazy Programmer, Advantages and Disadvantages of Relational Database, The Crazy Programmer, 2021, dostupno na: https://www.thecrazyprogrammer.com/2021/09/advantages-and-disadvantages-of-relational-database.html [pristupljeno 01.05.2023.]

[8] Dsstream, What is a Relational Database?, DSstream, dostupno na: https://dsstream.com/what-is-a-relational-database/ [pristupljeno 01.05.2023.]

[9] SolarWinds, NoSQL Database, SolarWinds, Austin, Texas, USA, dostupno na: https://www.solarwinds.com/resources/it-glossary/nosql-database [pristupljeno 01.05.2023.]
[10] TechTarget, NoSQL (Not Only SQL), TechTarget, Newton, dostupno na: https://www.techtarget.com/searchdatamanagement/definition/NoSQL-Not-Only-SQL [pristupljeno 01.05.2023.]

[11] MongoDB, NoSQL Databases: Pros and Cons, MongoDB, New York, 2021, dostupno na: https://www.mongodb.com/scale/nosql-databases-pros-and-cons [pristupljeno 01.05.2023.]

[12] DataStax, SQL vs. NoSQL: Pros, Cons, and Differences, DataStax, Santa Clara, dostupno na: https://www.datastax.com/blog/sql-vs-nosql-pros-cons [pristupljeno 01.05.2023.]

[13] Green Garage, 7 Pros and Cons of NoSQL: Is It Right for You?, Green Garage, dostupno na: https://greengarageblog.org/7-pros-and-cons-of-nosql [pristupljeno 01.05.2023.]

[14] Ubuntu, NoSQL Databases: What is MongoDB and Its Use Cases?, Canonical, UK, dostupno na: https://ubuntu.com/blog/nosql-databases-what-is-mongodb-and-its-use-cases [pristupljeno 01.05.2023.]

[15] KDnuggets, NoSQL Databases in Action: Use Cases and Insights, KDnuggets, 2023, dostupno na: https://www.kdnuggets.com/2023/03/nosql-databases-cases.html [pristupljeno 01.05.2023.]

[16] MongoDB, When to Use NoSQL, MongoDB, USA, dostupno na: https://www.mongodb.com/nosql-explained/when-to-use-nosql [pristupljeno 01.05.2023.]

[17] freeCodeCamp, Horizontal vs Vertical Scaling in Database, Quincy Larson, USA, dostpuno na: https://www.freecodecamp.org/news/horizontal-vs-vertical-scaling-in-database/ [pristupljeno 01.05.2023.]

[18] Wikipedia, Database Scalability, Wikimedia Foundation, USA, dostpuno na: https://en.wikipedia.org/wiki/Database_scalability [pristupljeno 01.05.2023.]

[19] dbWatch, What Does Horizontal & Vertical Database Scalability Mean?, dbWatch, Norway, dostpuno na: https://blog.dbwatch.com/what-does-horizontal-vertical-database-scalabilty-mean [pristupljeno 01.05.2023.]

[20] DB Services, NoSQL vs. Relational Database, DB Services, USA, dostupno na: https://dbservices.com/blog/nosql-vs-relational-database [pristupljeno 01.05.2023.]

[21] Difference Between RDBMS and MongoDB, DifferenceBetween.net, dostupno na: http://www.differencebetween.net/technology/difference-between-rdbms-and-mongodb/ [pristupljeno 01.05.2023.]

[22] N. Ghadiri, M. Moradi, Comparing the performance of updating the records at difference scales, ResearchGate, dostupno na: https://www.researchgate.net/figure/Comparing-the-performance-of-updating-the-records-at-different-scales-times-are-in_fig6_281629941 [pristupljeno 01.05.2023.]

[23] Neo4j vs. RDBMS and other NoSQL, Neo4j, Inc., dostupno na: https://neo4j.com/product/#comparison [01.05.2023.]

[24] S. Batra, C. Tyagi, Comparative Analysis of Relational And Graph Databases, International Journal od Soft Computing and Engineering, Volume 2, str. 509 – 512, svibanj 2012.

SAŽETAK

Okosnica završnog rada je usporedba relacijskih i NoSQL baza podataka. Kroz objašnjena strukture te prednosti i nedostataka pojedine baze podataka i njihove skalabilnost te izradom i objašnjenjem izrade oba tipa baza podataka oni su uspoređeni po njihovom načinju skaliranja te po njihovoj strukturi. U cilju dubljeg razumijevanja razlike između oba tipa baza podataka odrađena je usporedba nad izrađenim primjerima gdje se odrađivala usporedba nad unosom podataka, brisanju podataka te ažuriranju podataka kao i usporedba nad obradom određenih upita za svaku od baza podataka.

Rezultat je sveobuhvatan pregled razlika i prednosti korištenja različitih vrsta baza podataka, pružajući vrijedan izvor za istraživače i stručnjake na tom području.

Ključne riječi: baze podataka, horizontalna skalabilnost, NoSQL baze podataka, prednosti i nedostatci, relacijske baze podataka, struktura baze podataka,usporedba,vertikalna skalabilnost

ABSTRACT

COMPARISON OF RELATIONAL AND NOSQL DATABASES

The backbone of the final paper is a comparison of relational and NoSQL databases. Through the explained structures and the advantages and disadvantages of individual databases and their scalability, and by creating and explaining the creation of both types of databases, they were compared according to their scaling method and their structure. In order to gain a deeper understanding of the difference between both types of databases, a comparison was made of the created examples where a comparison was made of data entry and the speed of retrieving the same data.

The result is a comprehensive overview of the differences and advantages of using different types of databases, providing a valuable resource for researchers and practitioners in the field.

Keywords: advantages and disadvantages, comparison, database structure, databases, horizontal scalability, NoSQL databases, relational databases, vertical scalability

ŽIVOTOPIS

Dominik Vidović rođen je u Kninu, Republika Hrvatska, 18. ožujka 2002. godine kao najmlađi od dvoje djece roditelja Davora i Mihaele. Osnovnu školu završava u Tenji, a srednju školu završava u Osijeku.

Godine 2020. upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer računarstvo - računalni inženjer.

Za vrijeme školovanja zbog svojih postignuća i izvrsnosti kroz cijelo vrijeme školovanja nagrađen je izravnim upisom od strane Fakulteta elektrotehnike, računarstva i informacijskih tehnologija.

Vlastoručni potpis

_____

Dominik Vidović