Usporedba relacijskih i nerelacijskih baza podataka na primjeru informatičkog sustava ambulante

Bačelić, Patricija

SVEUČILIŠTE U SPLITU

PRIRODOSLOVNO-MATEMATIČKI FAKULTET

ZAVRŠNI RAD

USPOREDBA RELACIJSKIH I NERELACIJSKIH BAZA PODATAKA NA PRIMJERU INFORMATIČKOG SUSTAVA AMBULANTE

Patricija Bačelić

Split, rujan 2023

SVEUČILIŠTE U SPLITU

PRIRODOSLOVNO-MATEMATIČKI FAKULTET

ZAVRŠNI RAD

USPOREDBA RELACIJSKIH I NERELACIJSKIH BAZA PODATAKA NA PRIMJERU INFORMATIČKOG SUSTAVA AMBULANTE

Patricija Bačelić

Split, rujan 2023

Temeljna dokumentacijska kartica

Završni rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

Usporedba relacijskih i nerelacijskih baza podataka na primjeru informatičkog sustava ambulante

Patricija Bačelić

SAŽETAK

Cilj ovog završnog rada je opisati osnovne karakteristike relacijskih i nerelacijskih baza podataka i glavne razlike među njima. Osim samog opisa razlika, rad se sastoji i od praktičnog dijela u kojem se uspoređuju performanse relacijskih i nerelacijskih baza podataka na primjeru informacijskog sustava ambulante. Za predstavnika relacijskih baza podataka je izabran MySQL, dok je za nerelacijske baze podataka izabran MongoDB. U radu se prikazuju rezultati vremena izvršavanja jednostavnijih i složenijih CRUD upita. Naposljetku, sam rad služi kao orijentacija kod odabira baze podataka, budući da jedna vrsta baza podataka može biti pogodnija za korištenje od druge, ovisno o zahtjevima projekta na kojem se radi.

Ključne riječi: Baza podataka, MySQL, MongoDB, relacijske baze podataka, nerelacijske baze podataka.

Rad sadrži: 37 stranica, 22 slike, 1 tablicu i 6 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: **Doc. dr. sc. Monika Mladenović**, *docent Prirodoslovno-matematičkog fakulteta, Sveučilište u Splitu*

Neposredni voditelj: **Dino Nejašmić**, **mag. educ. math. et inf.**, *predavač Prirodoslovno-matematičkog fakulteta, Sveučilište u Splitu*

Ocjenjivači: **Doc. dr. sc. Monika Mladenović**, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Dino Nejašmić, **mag. educ. math. et inf.**, *predavač Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Antonela Prnjak, **mag. educ. inf.**, *asistent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Rad prihvaćen: rujan 2023.

Basic documentation card

Thesis

University of Split
Faculty of Science
Department of Informatics
Ruđera Boškovića 33, 21000 Split, Croatia

Comparison of relational and non-relational databases on the example of the clinic's IT system

Patricija Bačelić

ABSTRACT

The aim of this thesis is to describe the basic characteristics of relational and non-relational databases and the main differences between them. In addition to the description of the differences, the work also consists of a practical part in which the performance of relational and non-relational databases is compared using the example of the information system of the clinic. MySQL was chosen as the representative of relational databases, while MongoDB was chosen for non-relational databases. The paper presents the runtime results of simpler and more complex CRUD queries. Ultimately, the work itself serves as an orientation for database selection, as one type of database may be more suitable for use than another, depending on the requirements of the project being worked on.

Key words: Database, MySQL, MongoDB, relational databases, non-relational databases.

Thesis consist of: 37 pages, 22 pictures, 1 table and 6 references. The original is in Croatian language.

Menthor: **Doc. dr. sc. Monika Mladenović**, *assistant Professor of faculty of Science, University of Split*

Supervisor: **Dino Nejašmić**, **mag. educ. math. et inf.**, *lecturer at the Faculty of Science, University of Split*

Ocjenjivači: **Doc. dr. sc. Monika Mladenović**, *assistant Professor at the faculty of Science, University of Split*

Dino Nejašmić, **mag. educ. math. et inf.**, *lecturer at the Faculty of Science, University of Split*

Antonela Prnjak, **mag. educ. inf.**, *assistant at the Faculty of Science, University of Split*

Thesis accepted: September, 2023.

IZJAVA

Kojom izjavljujem s punom materijalnom i moralnom odgovornošću da sam završni s naslovom **Usporedba relacijskih i nerelacijskih baza podataka na primjeru informatičkog sustava ambulante** izradila samostalno pod voditeljstvom Dina Nejašmića, mag. educ. math. et inf., pred. U radu sam primijenila metodologiju znanstvenoistraživačkog rada i koristila literaturu koja je navedena na kraju završnog rada. Tuđe spoznaje, stavove, zaključke, teorije i zakonitosti koje sam izravno ili parafrazirajući navela u završnom radu na uobičajen, standardan način citirala sam i povezala fusnotama s korištenim bibliografskim jedinicama. Rad je pisan u duhu hrvatskog jezika.

STUDENTICA:

Patricija Bačelić

*Zahvaljujem se profesoru **Dinu Nejašmiću** na pruženoj prilici za mentorstvo, pomoći te savjetima tijekom izrade završnog rada.*

Zahvaljujem se svojoj obitelji na najboljoj mogućoj podršci tijekom studiranja.

Zahvaljujem se Luciji Popov, Domagoju Franiću, Ivi Ćatipović i Tomislavu Križanu što su mi bili oslonac tijekom studiranja.

Sadržaj:

Uvod

U današnje vrijeme, skladištenje, upravljanje i analiza podataka igraju ključnu ulogu u svim aspektima poslovanja i tehnologije. S razvojem tehnologije, razvile su se i različite tehnike i modeli za organizaciju podataka, a posebno su se istaknule relacijske i nerelacijske baze podataka. U kontekstu raznolikih potreba, relacijske i nerelacijske baze podataka već duže vremena zauzimaju ključno mjesto kao glavni alati za upravljanje podacima i za njihovu pohranu.

Relacijske baze podataka, koje se temelje na konceptu tablica i relacija među njima, su bile osnova za mnoge aplikacije pružajući strukturiranu, tabličnu organizaciju podataka i osiguravajući cjelovitost i dosljednost podataka.

No, s pojavom velikih količina nestrukturiranih i polustrukturiranih podataka, kao i potrebom za skalabilnošću i fleksibilnošću u modeliranju podataka, nerelacijske baze su postale snažna alternativa te su postale sve primjenjenije. Nerelacijske baze podataka nude različite modele pohrane te se ponajviše koriste zbog sposobnosti rješavanja problema visokog opterećenja i potrebe za horizontalnim skaliranjem.

Ovaj završni rad posvećen je usporedbi performansi relacijskih i nerelacijskih baza podataka, istražujući njihove ključne karakteristike, prednosti i nedostatke. Cilj je pružiti dublji uvid u karakteristike ovih dviju baza podataka te pokazati situacije u kojima bi se jedan model mogao pokazati boljim za korištenje od drugog.

1 Baze podataka

Baza podataka je kolekcija podataka, ograničenja i operacija koja reprezentira neke aspekte realnog svijeta. Tehnološkim napretkom, napredovale su i baze podataka. Prvo su bile hijerarhijske, s podacima pohranjenima u zapise, datoteke i polja, gdje su općenito dominirali brojčani i tekstualni sadržaji.(Maleković, Rabuzin: Uvod u baze podataka, 2016) Međutim, danas se u bazama pohranjuje široki spektar slikovnog, zvukovnog te ostalog sličnog sadržaja.

1.1 Entiteti i atributi

Entiteti predstavljaju osnovne komponente stvarnog svijeta koje želimo modelirati. Svaki entitet ima svoje karakteristike ili atribute koji opisuju svoja svojstva. Na primjer, u sustavu za upravljanje ambulantom, entiteti su pacijenti, osoblje, pregledi, nalazi i slično. Dok su atributi za entitet pacijenti ime, prezime, datum rođenja, OIB, adresa stanovanja i slično.

1.2 Veze

Veze prikazuju odnose između različitih entiteta. One definiraju kako entiteti međusobno komuniciraju i kako su povezani. Na primjer, u sustavu za upravljanje ambulantom, veza pripadanje označuje da entitet bolest pripada nekoj podgrupi bolesti, kao i što entitet podgrupa bolesti pripada nekoj grupi bolesti.

1.3 Modeli baza podataka

1.3.1 Hijerarhijski Model

Hijerarhijski model baze podataka organizira podatke u strukturu sličnu stablu. Svaki entitet ima nadređeni i podređeni entitet. Ovaj model je efikasan za situacije gdje su odnosi često hijerarhijski, ali može biti ograničavajući kada su odnosi složeniji ili kad se suočavamo s kompleksnim oblicima podataka.(LucidChart, 2023)

1.3.2 Mrežni Model

Mrežni model također koristi strukturu sličnu stablu, ali dopušta više nadređenih entiteta za svaki podređeni entitet. Ovaj model je fleksibilniji od hijerarhijskog, ali i dalje ima neka ograničenja u modeliranju kompleksnih odnosa.(LucidChart, 2023)

1.3.3 Relacijski Model

Relacijski model baze podataka, koji je najčešće korišten model, temelji se na matematičkoj teoriji relacija. Podaci se organiziraju u tablice (relacije), gdje svaki red predstavlja zapis, a svaki stupac predstavlja atribut. Ovaj model omogućuje složene upite i fleksibilno modeliranje, ali zahtijeva razumijevanje normalizacije podataka. U poglavlju „Relacijske baze podataka“ je napisano više o prednostima, kao i o nedostacima takvih baza podataka.(LucidChart, 2023)

1.3.4 Nerelacijski Model (NoSQL)

Nerelacijski modeli baza podataka nude alternativne pristupe pohrani podataka, često koriste formate kao što su dokumenti, stupci, ključevi i vrijednosti te grafovi. Ovi modeli su fleksibilniji za nestrukturirane i polustrukturirane podatke te omogućuju skalabilnost i brzinu. Više o nerelacijskim bazama podataka se nalazi u poglavlju „Nerelacijske baze podataka“.(LucidChart, 2023)

Važno je napomenuti da, osim navedenih osnovnih modela, postoje i drugi modeli baza podataka koji su razvijeni kako bi zadovoljili različite potrebe korisnika. Modeli poput objektno-orijentiranog, temporalnog, prostornog, itd., nude različite druge prednosti. U ovom radu sam se usredotočila na relacijske i nerelacijske baze podataka, dok ostali modeli nisu detaljno obrađeni. Međutim, važno je napomenuti da se ti modeli mogu bolje prilagoditi specifičnim zahtjevima pojedinog projekta ili aplikacije.

2 Relacijske baze podataka

Relacijske baze podataka predstavljaju jedan od najrasprostranjenijih modela za pohranu i organizaciju strukturnih podataka. U relacijskom modelu podataka podaci se prikazuju u obliku relacija tj. tablica. Ovaj model se gradi na temelju matematičke teorije relacija, gdje se podaci predstavljaju kroz tablice. Svaka tablica sastoji se od redaka i stupaca, pri čemu svaki redak sadrži određeni podatak, dok svaki stupac označava atribut. (Maleković, Rabuzin: Uvod u baze podataka, 2016)

Svaka tablica predstavlja jedan tip entiteta. Na primjer, u bazi podataka o sustavu za ambulantu, postoje tablice za osoblje, pacijente, preglede i sl. Atributi svakog entiteta predstavljaju stupce u tablici, te svaki stupac sadrži određeni tip podataka. Redci predstavljaju pojedinačne zapise o entitetima te sadrže određene vrijednosti atributa.

Relacijska baza podataka osigurava:

- Ažurnost pohranjenih podataka- omogućuje brzo i precizno ažuriranje podataka kako bi se osigurala njihova aktualnost i točnost
- Izgradnja sigurnosti i nadzora pristupa podacima-omogućuje postavljanje pravila i ograničenja pristupa podacima
- Mogućnost postavljanja više upita u svrhu izrade analiza i sinteza podataka- omogućuju izvođenje različitih upita za analizu i kombiniranje podataka
- Točnost pohranjenih podataka-baza podataka čuva podatke na precizan način kako bi se osigurala točnost i dosljednost informacija
- Trajno očuvanje integriteta pohranjenih podataka-omogućava da se integritet podataka održava tijekom njihove pohrane i manipulacije, čime se sprječavaju greške
- Najmanja redundancija- minimizira se duplikacija podataka, čime se smanjuje prostor za greške i olakšava održavanje
- Mogućnost kontrole i administriranja s jednog mjesta- omogućuje se centralno upravljanje bazom podataka i administracija sa sigurnog i praktičnog aspekta (Wikipedia: Relacijska baza podataka, 2023)

2.1 Prednosti i nedostaci

Prednost relacijskih baza podataka je što su podaci organizirani na jasan i strukturiran način. Osim toga, podaci su normalizirani radi smanjenja redundancije te postizanja konzistentnosti. Naposljetku, korištenje relacijske baze podataka omogućuje korištenje kompliciranih upita kako bi se podacima manipuliralo.

Nedostatak relacijskih baza je što one zahtijevaju ispravnu definiciju prije samog unosa podataka, što kod velikih baza može biti problem. Samim time, pri velikim količinama podataka, izvođenje kompleksnih upita može biti sporo.

Unatoč svemu, relacijske baze podataka, poput MySQL-a su i dalje ključna komponentna mnogih aplikacija, jer pružaju strukturiran način za pohranu i upravljanje podacima.

2.2 Vrste relacijskih baza podataka

- **Klasične Relacijske Baze Podataka-** tradicionalne relacijske baze, poput MySQL-a, PostgreSQL-a i Microsoft SQL Server-a, koje koriste SQL za upite i upravljanje podacima. Ove baze podataka koriste tablice s redcima i stupcima, definirane sheme i ključevima za održavanje integriteta podataka.

- **Objektno-Relacijske Baze Podataka-** kombiniraju značajke objektno-orijentiranih modela s relacijskim modelom. To omogućuje pohranu i upravljanje složenijim strukturama podataka poput objekata, nasljeđivanja i metoda. Primjer ovog tipa je Oracle Database.

- **In-Memory Relacijske Baze Podataka-** baze podataka koje čuvaju podatke u radnoj memoriji umjesto na diskovima, što omogućava izuzetno brze upite. Primjeri uključuju SAP HANA i Oracle TimesTen.

- **Distribuirane Relacijske Baze Podataka-** omogućuju pohranu podataka na više fizičkih lokacija. To je korisno za postizanje skalabilnosti i dostupnosti. Primjer je Apache Cassandra.

Svaka od ovih vrsta relacijskih baza podataka ima svoje specifične karakteristike te svaka od njih može odgovarati potrebama sustava ili aplikacije.

2.3 MySQL

MySQL je popularni relacijski sustav za upravljanje bazama podataka koji se temelji na SQL jeziku. MySQL koristi SQL za upite i manipulaciju podacima. SQL je standardni jezik za relacijske baze podataka koji omogućava izvođenje raznih operacija kao što su SELECT (upit), INSERT (umetanje), UPDATE (ažuriranje) i DELETE (brisanje).

U MySQL-u se entiteti modeliraju kao tablice, a u svakoj tablici se definiraju nazivi i tipovi stupaca, kao i ograničenja i ključevi ukoliko su potrebni. Ovaj pristup organizira podatke u hijerarhijsku i povezanu strukturu, što olakšava izvođenje složenih upita. MySQL pruža dosta mogućnosti za upravljanje podacima, pogodan je za male web stranice ili za velike poslovne sustave.(MySQL, 2023)

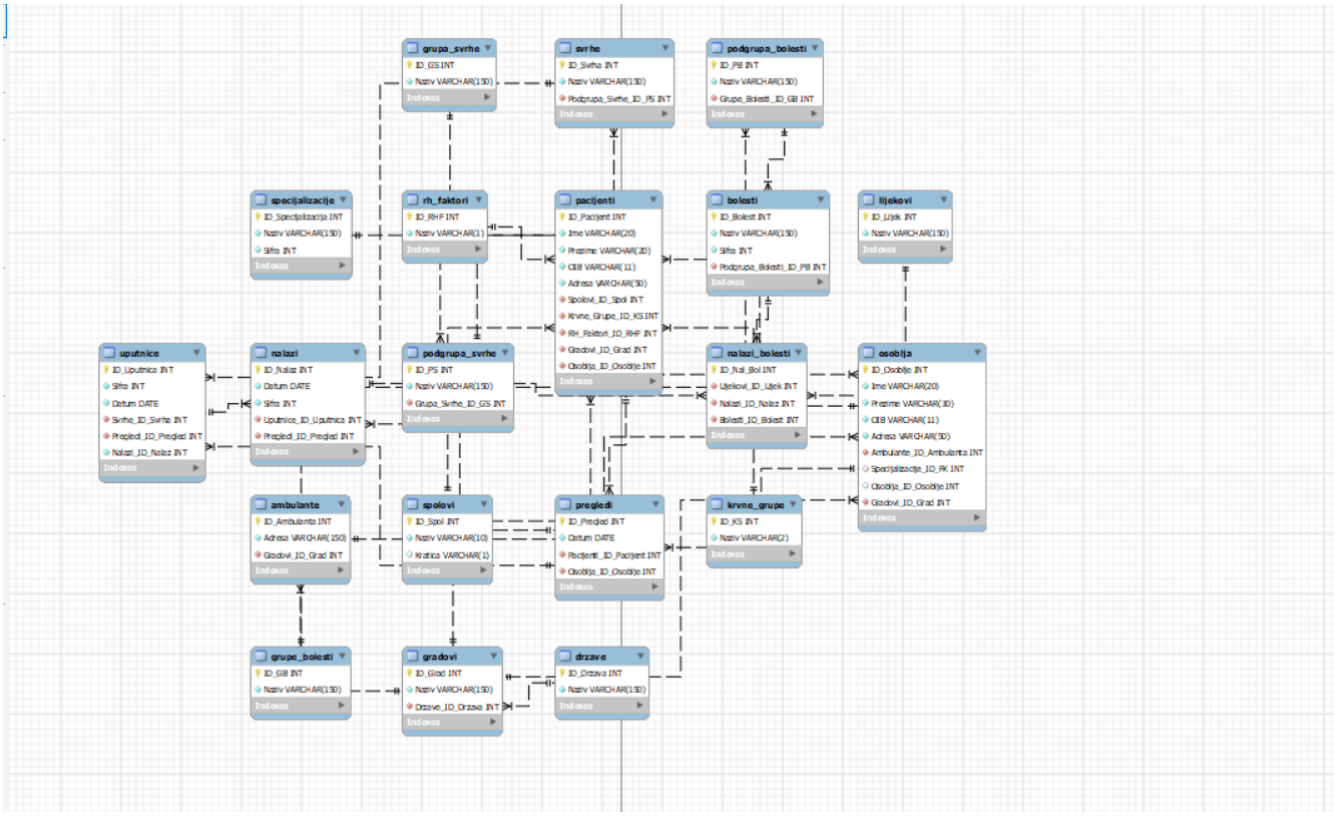
Iako postoji mnogo drugih relacijskih sustava za upravljanje bazama podataka, kao što su PostgreSQL, Microsoft SQL Server, Oracle Database, i drugi, MySQL sam odabrala ponajprije zbog svoje trenutne popularnosti i dostupnosti. Danas je on široko rasprostranjen te postoji velika zajednica korisnika, kao i mnoštvo dostupnih resursa za učenje i podršku. Nadalje, MySQL je open-source softver, te se može koristiti bez troškova licenciranja. Osim toga, MySQL se jednostavno može integrirati s mnogim programskim jezicima i aplikacijskim okruženjima.

```
CREATE TABLE Drzave
(
    ID_Drzava INT NOT NULL AUTO_INCREMENT,
    Naziv VARCHAR(100) NOT NULL,
    PRIMARY KEY (ID_Drzava)
);
```

Slika 1 *Kreiranje tablice "Države"*

ID_Drzava	Naziv
1	Germany
2	United Kingdom
3	Malawi
4	Comoros
5	Antigua and Barbuda
6	Jamaica
7	Namibia
8	Monaco
9	Lesotho
10	Saudi Arabia
11	United Kingdom
12	Palau
13	Slovenia
14	Uruguay

Slika 2 *Prikaz podataka iz tablice "Države"*



Slika 3 *Prikaz modela u MySQL bazi podataka*

3 Nerelacijske baze podataka

Nerelacijske baze podataka, poznatije kao NoSQL baze, razvijene su kako bi se nosile s izazovima brzog rasta i raznolikosti podataka u današnjem svijetu. Razvile su se kako bi odgovorile na potrebe za brzinom, skalabilnošću i fleksibilnošću u manipulaciji s velikim i raznolikim količinama podataka.

Za razliku od relacijskih baza koje koriste tablice, nerelacijske baze koriste različite modele za organizaciju podataka, ovisno o specifičnoj primjeni. Nerelacijske baze omogućuju pohranu nestrukturiranih, polustrukturiranih i strukturiranih podataka.(Wikipedia, NoSQL, 2023)

3.1 Prednosti i nedostaci

Jedna od najvećih prednosti nerelacijskih baza podataka je skalabilnost. Nerelacijske baze podataka često omogućuju horizontalnu skalabilnost dodavanjem novih čvorova, što je osobito važno za aplikacije s brzim rastom prometa i podataka. Nadalje, fleksibilnost je također jedna od bitnih stavki ovakve vrste baze podataka, jer ne zahtjeva strogo definiranu shemu kao kod relacijskih baza podataka. To je korisno kod rada s aplikacijama gdje se shema često mijenja. Konačno, relacijske baze podataka mogu pružiti brže upite i odgovore, jer podaci nisu normalizirani kao u relacijskim bazama podataka.

S druge strane, nerelacijske baze podataka mogu zahtijevati više vremena za obradu složenih upita iz nekoliko razloga. Prvo, nerelacijske baze podataka često čuvaju podatke u obliku koji nije normaliziran kao kod relacijskih baza podataka. To znači da podaci mogu biti pohranjeni u više denormaliziranim formatima ili u obliku ključ-vrijednost, što može otežati brzo dohvaćanje i povezivanje podataka. Drugo, modeli podataka u nerelacijskim bazama često su kompleksniji za rad jer se mogu koristiti različite tehnike za pohranu podataka ovisno o potrebama aplikacije. Ovo povećava složenost upita jer programeri moraju razumjeti kako su podaci strukturirani i kako se mogu izvući iz baze. Konačno, nerelacijske baze podataka često nemaju dovoljno razvijene jezike za upite i standardizirane načine za manipulaciju podacima u usporedbi s relacijskim bazama podataka. Iz tog slijedi da razvojni timovi moraju uložiti dodatan napor u razvoj i optimizaciju upita, što može dovesti do sporijih performansi u usporedbi s relacijskim bazama. Osim toga, kompleksniji upiti u nerelacijskim bazama često zahtijevaju dublje razumijevanje strukture podataka i modela baze podataka, što može rezultirati manje intuitivnim upitima za programere.

3.2 Vrste nerelacijskih baza podataka

- **Dokumentne Baze -**baze podataka koje pohranjuju podatke u dokumentima, obično u JSON ili BSON formatu. One omogućuju fleksibilno modeliranje podataka jer dokumenti unutar iste kolekcije ne moraju imati istu strukturu. Pogodne su za situacije gdje se podaci često mijenjaju i gdje je potrebna dinamička shema. Najpoznatiji primjer ovakve baze podataka je MongoDB.(Wikipedia: NoSQL, 2023)
- **Stupčane Baze** -baze podataka koje pohranjuju podatke po stupcima umjesto po redcima kao u relacijskim bazama. One omogućuju brze upite za određene atribute i olakšavaju rad s velikim količinama podataka. Često se koriste za velike sustave, kako su skalabilne i otporne na kvarove. Najpoznatiji primjer ovakve baze podataka je Cassandra.(Wikipedia: NoSQL, 2023)
- **Ključ-Vrijednost Baze** - baze podataka koje pohranjuju podatke kao parove ključ-vrijednost. Koriste se ponajviše zbog svoje brzine, međutim nisu pogodne za složenije upite. Najpoznatiji primjer ovakve baze podataka su Redis i Amazon DynamoDB.(Wikipedia: NoSQL, 2023)
- **Grafovske Baze** -pohranjuju podatke koristeći koncept grafova, s čvorovima i vezama između njih. One su idealne su za modeliranje i upite koji se temelje na relacijama među entitetima. Najpoznatiji primjer je Neo4j.(Wikipedia: NoSQL, 2023)

3.3 MongoDB

MongoDB je popularni dokumentni NoSQL sustav za upravljanje bazama podataka koji umjesto tradicionalnih tablica, koristi fleksibilni model dokumenta, gdje se podaci pohranjuju u JSON-sličnim zapisima poznatim kao BSON.

MongoDB je posebno koristan za aplikacije koje rade s nestrukturiranim i polustrukturiranim podacima kao što su tekstualni dokumenti, JSON zapisi ili kompleksni objekti. Također, nudi mogućnost skalabilnosti dodavanjem novih čvorova kako bi se nosili s rastućim prometom i količinama podataka, što je posebno korisno u projektima gdje se zahtjevi mijenjaju često.

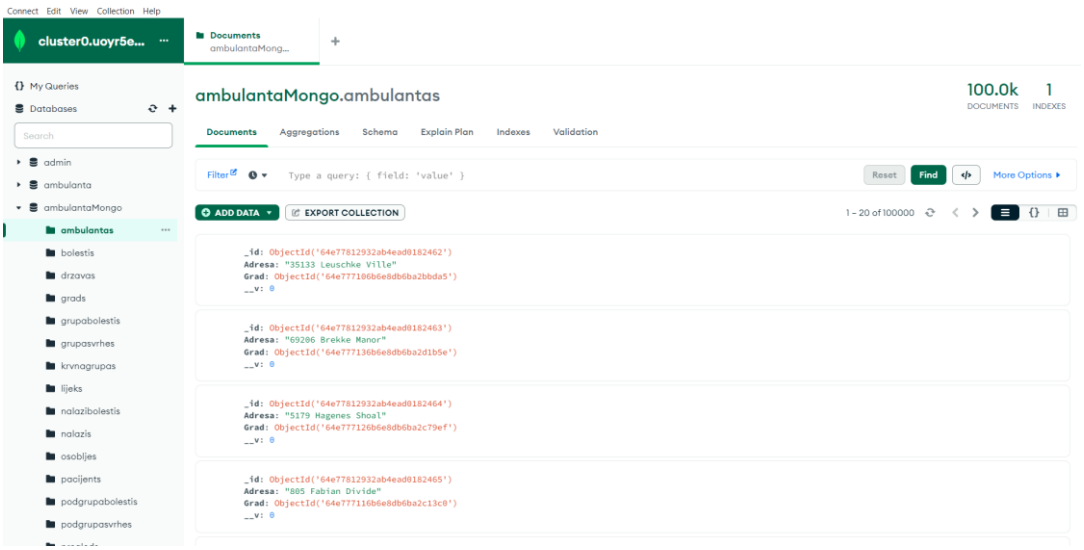
U ovakvoj vrsti baze podataka, podaci se grupiraju u kolekcije, a svaka kolekcija sadrži dokumente različitih struktura. Unutar iste kolekcije se mogu nalaziti dokumenti koji imaju različite atribute i strukture, što omogućuje fleksibilan pristup za pohranu raznolikih podataka.(MongoDB, 2023)

```
const mongoose = require('mongoose');
const drzavaSchema = new mongoose.Schema({
  Naziv: String,
});
const Drzava = mongoose.model('Drzava', drzavaSchema);
module.exports = Drzava;
```

Kod 1 *Stvaranje dokumenta "Drzava"*

```
_id: ObjectId('64e65ffc638630ee3212eb01')
Naziv: "Sint Maarten"
__v: 0

_id: ObjectId('64e65ffc638630ee3212eb04')
Naziv: "Tajikistan"
__v: 0

_id: ObjectId('64e65ffc638630ee3212eaf7')
Naziv: "Aland Islands"
__v: 0

_id: ObjectId('64e65ffc638630ee3212eb00')
Naziv: "Seychelles"
__v: 0

_id: ObjectId('64e65ffc638630ee3212eafa')
Naziv: "Uruguay"
__v: 0
```

Slika 4 *Prikaz podataka iz dokumenta "Drzava"*



Slika 5 *Prikaz modela u MongoDB bazi podataka*

4 Usporedba performansi relacijskih i nerelacijskih baza podataka

Svaka baza podataka ima svoje prednosti i mane, što čini mjerenje performansi ključnim korakom pri odabiru odgovarajuće baze za specifičan projekt. U ovom radu uspoređujemo relacijsku bazu podataka **MySQL** i nerelacijsku bazu podataka **MongoDB**. Postoji nekoliko vrsta testiranja baza podataka koje se često provode kako bismo ocijenili njihove performanse:

- Performanse skaliranja: Ovakvo testiranje provjerava kako se baza ponaša pod povećanim opterećenjem, tj. kad se sustav skalira prema gore, fokusirajući se na trenutne performanse.
- Testiranje sigurnosti: Provjerava kako baza podataka štiti podatke od neovlaštenog pristupa.
- Skalabilnost: Ova vrsta testiranja istražuje kako se baza ponaša kada se sustav skalira prema gore ili prema dolje, fokusirajući se na to kako bi sustav mogao poboljšati performanse tijekom vremena

Osim ovih testova, postoji i mnogo drugih koji se bave pitanjem oporavka baze podataka u slučaju gubitka ili oštećenja podataka, te različitih testiranja koja provjeravaju integritet podataka.

U našem istraživanju fokusiramo se na mjerenje vremena izvršavanja **CRUD upita** (Create, Read, Update, Delete) jer želimo prikazati performanse baza podataka u tipičnim scenarijima upotrebe. Ovo omogućuje direktnu usporedbu između MySQL-a i MongoDB-a, što je ključno za odabir odgovarajuće baze za projekt.

Prije samog testiranja, trebalo je stvoriti i popuniti baze podataka s dovoljno podataka kako bismo mogli istaknuti razlike između njih. Svaka od ovih baza ima 20 tablica odnosno dokumenata. Dok su relacijske baze podataka međusobno povezane putem veza i omogućuju manipulaciju putem stranih ključeva, nerelacijske baze nemaju takvu vrstu povezanosti, što je njihova suština. Ipak, kako bi se upiti koji se izvode nad relacijskim bazama mogli primijeniti na MongoDB, kreirane su reference na druge dokumente.

Nakon stvaranja baza podataka, generirali smo 100,000 podataka. To se smatra neizrečenim standardom kako bismo istaknuli razlike između baza. Generiranje podataka obavljeno je korištenjem paketa **"faker"** koji sadržava ugrađene funkcije prilagođene potrebama baza podataka. Umjesto podataka koji nisu odgovarali, koristili su se generirani tekstovi uz pomoć funkcije "lorem.words()".

Baze podataka smo testirali izravno u razvojnom okruženju **Visual Studio Code**, prateći vrijeme izvođenja uz pomoć funkcije **"performance.now()"**. Upite smo podijelili na jednostavnije i složenije za svaku od operacija u CRUD skupu.

Za izvršavanje koda bilo je potrebno instalirati Node.js, te dodatne pakete **"mysql2"** i **"mongoose"** kako bi omogućili izvođenje i izvršavanje upita.

Posebno važno je napomenuti da se MongoDB ubrzao nakon što smo počeli koristiti **"bulk insert".** "Bulk insert" je tehnika koja omogućava brži unos velikog broja podataka odjednom, što je doprinijelo ubrzanju MongoDB-a.

4.1 Select upiti

U ovom poglavlju se analiziraju performanse izvođenja SELECT upita u MySQL i MongoDB bazama podataka. SELECT upiti se koriste za dohvaćanje podataka iz baza, a ja sam u narednim poglavljima analizirala jednostavne i složene SELECT upite te usporedila njihove brzine izvođenja u oba sustava.

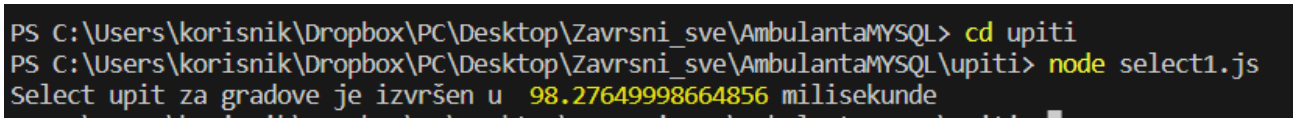
4.1.1 Jednostavni select upiti

U ovom poglavlju je provedena usporedbu između izvođenja jednostavnih SELECT upita u MySQL i MongoDB bazama podataka. Cilj je istražiti brzinu izvršavanja osnovnih SELECT upita za jednostavno dohvaćanje podataka.

Prvi SELECT upit koji sam izabrala bio je upit za pronalaženje svih gradova. Razlog za odabir ovog upita je njegova jednostavnost i učestalost korištenja u stvarnim aplikacijama.

```
const mysql = require('mysql2');
const { performance } = require('perf_hooks');
const connection = mysql.createConnection({
host: 'localhost',
user: 'root',
password: 'Petra123!',
database: 'ambulanta'
});
const select1 = 'SELECT * FROM gradovi';
const start = performance.now();
connection.query(select1, (error, results) => {
if (error) {
console.error('Pogreška pri izvođenju upita:', error);
return;
}
const kraj = performance.now();
const vrijeme = kraj - start;
console.log('Select upit za gradove je izvršen u ', vrijeme, 'milisekunde');
});
connection.end();
```

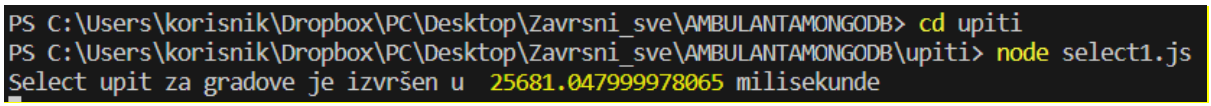
Kod 2 *Upit za pronalaženje svih gradova(MySQL)*



Slika 6 *Vrijeme izvršavanja upita za pronalaženje svih gradova(MySQL)*

```
const mongoose = require('mongoose');
const { performance } = require('perf_hooks');
const Grad = require('../modeli/gradovi');

const opcije = {
useNewUrlParser: true,
useUnifiedTopology: true,
wtimeoutMS: 30000,
};
mongoose.connect('mongodb+srv://pbacelic:patricijamongo@cluster0.
uoyr5ek.mongodb.net/ambulantaMongo', opcije);
const select1 = {};
const start = performance.now();
Grad.find(select1)
.then((results) => {
const kraj = performance.now();
const vrijeme = kraj - start;
console.log('Select upit za gradove je izvršen u ', vrijeme,
'milisekunde');
//console.log(results);
})
.catch((error) => {
console.error('Pogreška pri izvođenju upita:', error);
});
```

Kod 3 *Upit za pronalaženje svih gradova(MongoDB)*



Slika 7 *Vrijeme izvršavanja upita za pronalaženje svih gradova(MongoDB)*

Nakon izvršenja upita, MySQL baza podataka je prikazala sve gradove za 98.28 milisekunda, dok je mongoDB baza podataka izvršila isti takav upit za 25681.05 milisekunda. MySQL baza podataka je brža za 25582.77 milisekunde.

Rezultati mjerenja pokazuju da MySQL baza podataka značajno nadmašuje MongoDB u brzini izvođenja jednostavnih SELECT upita. Razlog za ovu razliku je zbog različitih načina obrade podataka i internim strukturama ovih dviju baza podataka. MySQL je relacijska baza podataka koja optimizira upite s relacijskim modelom, dok MongoDB, kao nerelacijska baza, često zahtijeva skeniranje cijele kolekcije dok traži podatke.

4.1.2 Složeni select upiti

U ovom poglavlju je provedena usporedba između izvođenja složenih SELECT upita u MySQL i MongoDB bazama podataka. Fokus je na zahtjevnijim upitima koji uključuju više tablica ili filtriranje podataka.

Drugi SELECT upit koji sam odabrala je pronalaženje svih pacijenata koji žive u određenom gradu. Razlog za odabir ovog upita je njegova složenost i potreba za spajanjem dviju tablica, što često predstavlja realan scenarij u aplikacijama. Osim toga, želim usporediti kako MySQL i MongoDB obrađuju složene upite s JOIN operacijom.

```
const mysql = require('mysql2');
const { performance } = require('perf_hooks');
const connection = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: 'Petra123!',
    database: 'ambulanta'
  });
  const select2 = `
  SELECT Pacijenti.Ime, Pacijenti.Prezime, Gradovi.Naziv
  FROM Pacijenti
  INNER JOIN Gradovi ON Pacijenti.Gradovi_ID_Grad = Gradovi.ID_Grad
  WHERE Gradovi.Naziv = 'Washington';`;
  const start = performance.now();
  connection.query(select2, (error, results) => {

    if (error) {
      console.error('Pogreška pri izvođenju upita:', error);
      return;
    }
    const kraj = performance.now();
    const vrijeme = kraj - start;
    console.log('Select upit za pacijente koji žive u Washingtonu je
 izvršen u ', vrijeme, 'milisekunde');
  });
  connection.end();
```

Kod 4 *Upit za pronalaženje svih pacijenata koji žive u Washingtonu(MySQL)*

```
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AmbulantaMYSQL> cd upiti
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AmbulantaMYSQL\upiti> node select2.js
Select upit za pacijente koji žive u Washingtonu je izvršen u  41.56780004501343 milisekunde
```

Slika 8 *Vrijeme izvršavanja upita za pronalaženje svih pacijenata koji žive Washingtonu(MySQL)*

```
const mongoose = require('mongoose');
const { performance } = require('perf_hooks');
const Pacijent = require('../modeli/pacijenti');
const opcije = {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  wtimeoutMS: 30000,
};
mongoose.connect('mongodb+srv://pbacelic:patricijamongo@cluster0.uoy
r5ek.mongodb.net/ambulantaMongo', opcije);
const gradID = new
mongoose.Types.ObjectId('64e777106b6e8db6ba2bb248');
const start = performance.now();
Pacijent.find({ Gradovi_ID_Grad: gradID })
  .then((pacijenti) => {
    const kraj = performance.now();
    const vrijeme = kraj - start;
    console.log('Select upit za gradove je izvršen u ', vrijeme,
'milisekunde');
    console.log('Pronađeni su pacijenti',pacijenti);
  })
  .catch((error) => {
    console.error('Pogreška pri dohvaćanju pacijenata:', error);
  });
```

Kod 5 *Upit za pronalaženje svih pacijenata koji žive u Splitu(MongoDB)*

```
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AMBULANTAMONGODB> cd upiti
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AMBULANTAMONGODB\upiti> node select2.js
Select upit za gradove je izvršen u  2660.1150000095367 milisekunde
Pronađeni su pacijenti [
  {
    _id: new ObjectId("64e7c0866006cf7313452875"),
    Ime: 'Lola',
    Prezime: 'Christiansen',
    OIB: '65554957522',
    Adresa: '1905 Koby Canyon',
    Spolovi_ID_Spol: new ObjectId("64e7bb9880e8fb412879f1d1"),
    Krvne_Grupe_ID_KS: new ObjectId("64e7bbfd3ed7c191a62a3560"),
    RH_Faktori_ID_RHF: new ObjectId("64e7bbce78a4ca449d7d9270"),
    Gradovi_ID_Grad: new ObjectId("64e777106b6e8db6ba2bb248"),
    Osoblja_ID_Osoblje: new ObjectId("64e7be14782b85970b053a36"),
    __v: 0
  },
  {
    _id: new ObjectId("64e7c0866006cf731345392b"),
    Ime: 'Tristin',
    Prezime: 'Boyer',
    OIB: '94780960967',
    Adresa: '6357 Littel Coves',
    Spolovi_ID_Spol: new ObjectId("64e7bb9880e8fb412879f1cf"),
    Krvne_Grupe_ID_KS: new ObjectId("64e7bbfd3ed7c191a62a3560"),
    RH_Faktori_ID_RHF: new ObjectId("64e7bbce78a4ca449d7d9270"),
    Gradovi_ID_Grad: new ObjectId("64e777106b6e8db6ba2bb248"),
    Osoblja_ID_Osoblje: new ObjectId("64e7be16782b85970b05f61a"),
    __v: 0
  }
]
```

Slika 9 *Vrijeme izvršavanja upita za pronalaženje svih pacijenata koji žive u Splitu(MongoDB)*

Nakon izvršenja select upita, pronađeno je 6 gradova u MySQL bazi podataka, dok su u MongoDB pronađena 3 rezultata, MySQL baza podataka je prikazala sve pacijente koji žive u gradu „Washingtonu“ za 41.57 milisekunda, dok je MongoDB baza podataka izvršila isti takav upit za grad „Split“ u 2660.11 milisekunda. MySQL baza podataka je brža za 2618.54 milisekunde.

Rezultati pokazuju da MySQL i dalje brže izvršava složene SELECT upite. Razlika proizlazi iz posljedice načina na koji obje baze podataka optimiziraju JOIN operacije i upite s više uvjeta. MySQL, kao relacijska baza podataka, može bolje optimizirati takve upite.

4.2 Insert upiti

U ovom poglavlju se analiziraju performanse INSERT upita u MySQL i MongoDB bazama podataka. INSERT upiti se koriste za dodavanje novih podataka u baze, a jednostavnim i složenim INSERT upitima se uspoređuje njihova brzina izvođenja u oba sustava.

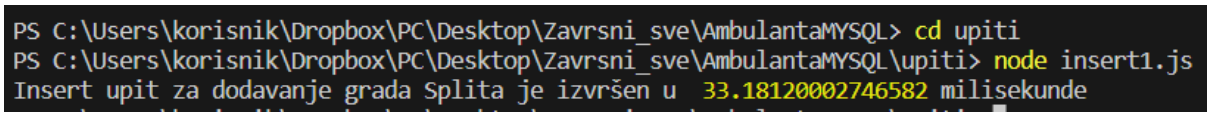
4.2.1 Jednostavni insert upiti

U ovom poglavlju se uspoređuje izvođenje jednostavnih INSERT upita za unos novih podataka u MySQL i MongoDB baze podataka.

Prvi INSERT upit koji sam obradila je dodavanje grada "Split" u baze podataka. Odabir ovog upita temelji se na njegovoj jednostavnosti i čestoj primjeni. Cilj je vidjeti kako MySQL i MongoDB rukuju ovom uobičajenom operacijom te usporediti brzinu njihovih izvođenja.

```
const mysql = require('mysql2');
const { performance } = require('perf_hooks');
const connection = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: 'Petra123!',
    database: 'ambulanta'});
  const insert1 = 'INSERT INTO Gradovi (Naziv, Drzave_ID_Drzava)
VALUES ("Split", 191)';
  const start = performance.now();
  connection.query(insert1, (error, results) => {
    if (error) {
      console.error('Pogreška pri izvođenju upita:', error);
      return;}
    const kraj = performance.now();
    const vrijeme = kraj - start;
    console.log('Insert upit za dodavanje grada Splita je izvršen
u ', vrijeme, 'milisekunde');});
  connection.end();
```

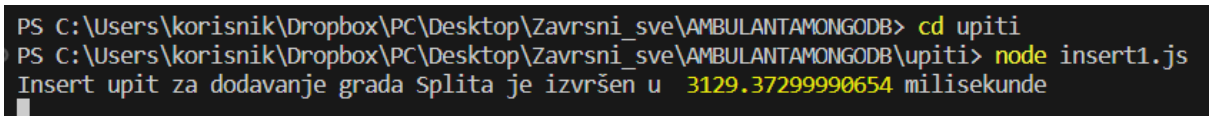
Kod 6 *Upit za dodavanje grada "Split"(MySQL)*



Slika 10 *Vrijeme izvršavanja upita za dodavanje grada „Split“(MySQL)*

```
const { performance } = require('perf_hooks');

const Grad = require('../modeli/gradovi');
const opcije = {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  wtimeoutMS: 30000,
};
mongoose.connect('mongodb+srv://pbacelic:patricijamongo@cluster0.uoy
r5ek.mongodb.net/ambulantaMongo', opcije);
const insert1 = {
  Naziv: 'Split',
  Drzava: new mongoose.Types.ObjectId('64e65ffc638630ee3212ebab'),
};
const start = performance.now();
Grad.create(insert1)
  .then(() => {
    const kraj = performance.now();
    const vrijeme = kraj - start;
    console.log('Insert upit za dodavanje grada Splita je izvršen u
', vrijeme, 'milisekunde');
  })
  .catch((error) => {
    console.error('Pogreška pri unosu podataka:', error);
  });
```

Kod 7 *Upit za dodavanje grada "Split" (MongoDB)*



Slika 11 *Vrijeme izvršavanja upita za dodavanje grada „Split“(MongoDB)*

Nakon izvršenja INSERT upita, MySQL baza podataka brže je dodala grad "Split" za 33.18 milisekunde, dok je MongoDB baza podataka izvršila isti upit za 3129.38 milisekunda. MySQL baza podataka brža je za 3096.2 milisekunde.

Rezultati pokazuju da MySQL brže izvršava jednostavne INSERT upite. To posljedica razlika u internim mehanizmima za unos podataka.

4.2.2 Složeni insert upiti

U ovom poglavlju se uspoređuje izvođenje složenih INSERT upita u MySQL i MongoDB bazama podataka.

Drugi INSERT upit odnosi se na dodavanje člana osoblja u baze podataka. Razlog za odabir ovog upita je njegova složenost i potreba za unosom podataka u više tablica, što često predstavlja realan scenarij u aplikacijama. Osim toga, želim usporediti kako MySQL i MongoDB obrađuju složene INSERT upite.

```
const mysql = require('mysql2');
const { performance } = require('perf_hooks');
const connection = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: 'Petra123!',
    database: 'ambulanta'
  });
  const insert2 = `INSERT INTO Osoblja (Ime, Prezime, OIB, Adresa,
Ambulante_ID_Ambulanta, Gradovi_ID_Grad)
  VALUES ('Patricija', 'Bačelić', 54574117082, 'Donji Banovci 198
Grebastica', 1, 2);`
  const start = performance.now();
  connection.query(insert2, (error, results) => {
    if (error) {
      console.error('Pogreška pri izvođenju upita:', error);
      return;
    }
    const kraj = performance.now();
    const vrijeme = kraj - start;
    console.log('Insert upit dodavanje osoblja je izvršen u ',
vrijeme, 'milisekunde');
  });
  connection.end();
```

Kod 8 *Upit za dodavanje člana osoblja(MySQL)*

```
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AmbulantaMYSQL> cd upiti
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AmbulantaMYSQL\upiti> node insert2.js
Insert upit dodavanje osoblja je izvršen u  24.541899919509888 milisekunde
```

Slika 12 *Vrijeme izvršavanja upita za dodavanje člana osoblja(MySQL)*

```
const mongoose = require('mongoose');
const { performance } = require('perf_hooks');
const Osoblja = require('../modeli/osoblja');
const opcije = {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  wtimeoutMS: 30000,
};
mongoose.connect('mongodb+srv://pbacelic:patricijamongo@cluster0.uoy
r5ek.mongodb.net/ambulantaMongo', opcije);
const insert2 = {
  Ime: 'Patricija',
  Prezime: 'Bačelić',
  OIB: 54574117082,
  Adresa: 'Donji Banovci 198 Grebastica',
  Ambulante_ID_Ambulanta: new
mongoose.Types.ObjectId('64e77814932ab4ead018eb32') ,
  Gradovi_ID_Grad: new
mongoose.Types.ObjectId('64e777106b6e8db6ba2bb1cd'),
};
const start = performance.now();
Osoblja.create(insert2)
  .then(() => {
    const kraj = performance.now();
    const vrijeme = kraj - start;
    console.log('Insert upit za dodavanje osoblja je izvršen u ',
vrijeme, 'milisekunde');
  })
  .catch((error) => {
    console.error('Pogreška pri unosu podataka:', error);
  });
```

Kod 9 *Upit za dodavanje člana osoblja(MongoDB)*

```
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AMBULANTAMONGODB> cd upiti
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AMBULANTAMONGODB\upiti> node insert2.js
Insert upit za dodavanje osoblja je izvršen u  2791.272700071335 milisekunde
```

Slika 13 *Vrijeme izvršavanja upita za dodavanje člana osoblja(MongoDB)*

Nakon izvršenja upita, MySQL baza podataka je dodala atribut u tablicu osoblje za 24.54 milisekunde, dok je MongoDB baza podataka izvršila isti takav upit za 2791.27 milisekunda. MySQL baza podataka je brža za 2766.73 milisekunde.

Ovaj primjer pokazuje da MySQL može biti brži za obradu složenih INSERT upita, ali isto tako treba imati na umu da MongoDB često koristi različite pristupe za unos podataka, u ovom slučaju su korišteni JSON dokumenti za unos, što znatno usporava unos podataka.

4.3 Update upiti

U ovom poglavlju se analiziraju performanse izvođenja UPDATE upita u MySQL i MongoDB bazama podataka. UPDATE upiti koriste se za izmjenu postojećih podataka u bazama, a usporedit ćemo brzinu izvođenja jednostavnih i složenih UPDATE upita u oba sustava.

4.3.1 Jednostavni update upiti

Prvi UPDATE upit odnosi se na promjenu naziva grada u bazama podataka. Ovaj primjer odabran je zbog česte potrebe za izmjenom osobnih podataka u aplikacijama.

```
const mysql = require('mysql2');
const { performance } = require('perf_hooks');
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'Petra123!',
  database: 'ambulanta',
});
const update1 = `
  UPDATE Gradovi
  SET Naziv = 'Split'
  WHERE ID_Grad = 1;
`;
const start = performance.now();
connection.query(update1, (error, results) => {
  if (error) {
    console.error('Pogreška pri izvođenju upita:', error);
    return;
  }
  const kraj = performance.now();
  const vrijeme = kraj - start;
  console.log('Update upit za grad je izvršen u ', vrijeme,
'milisekunde');
});
connection.end();
```

Kod 10 *Upit za ažuriranje naziva grada(MySQL)*

```
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AmbulantaMYSQL> cd upiti
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AmbulantaMYSQL\upiti> node update1.js
Update upit za grad je izvršen u  16.229699969291687 milisekunde
```

Slika 14 *Vrijeme izvršavanja upita za ažuriranje naziva grada(MySQL)*

```
const mongoose = require('mongoose');
const { performance } = require('perf_hooks');
const Grad = require('../modeli/gradovi');
const opcije = {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  wtimeoutMS: 30000,
};
mongoose.connect('mongodb+srv://pbacelic:patricijamongo@cluster0.uoy
r5ek.mongodb.net/ambulantaMongo', opcije);
const staroImeGrada = 'Aydenmouth';
const novoImeGrada = 'Split';
const start = performance.now();
Grad.updateOne({ Naziv: staroImeGrada }, { $set: { Naziv:
novoImeGrada } })
  .then((result) => {
    const kraj = performance.now();
    const vrijeme = kraj - start;
    console.log(`Update upit za grad  je izvršen u ${vrijeme}
milisekundi`);
  })
  .catch((error) => {
    console.error('Pogreška pri izvođenju upita:', error);
  });
```

Kod 11 *Upit za ažuriranje naziva grada(MongoDB)*

```
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AMBULANTAMONGODB> cd upiti
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AMBULANTAMONGODB\upiti> node update1.js
Update upit za grad  je izvršen u 2603.0629999637604 milisekundi
```

Slika 15 Vrijeme izvršavanja upita za ažuriranje naziva grada(MongoDB)

Nakon izvršenja UPDATE upita, MySQL baza podataka brže je promijenila adresu pacijenta za 8.62 milisekunde, dok je MongoDB baza podataka izvršila isti upit za 1557.21 milisekunda. MySQL baza podataka brža je za 1548.59 milisekunde.

Rezultati pokazuju da MySQL brže izvršava jednostavne UPDATE upite, razlog tome je što je ovakva baza podataka pogodnija za implementaciju u MySQL više nego u MongoDB, te su zapravo reference na druge dokumente u ovom slučaju te koje usporavaju update upit.

4.3.2 Složeni update upiti

Drugi UPDATE upit odnosi se na promjenu mjesta stanovanja liječnika u bazama podataka. Razlog za odabir ovog upita je složenost i potreba za izmjenom podataka u više tablica.

```
const mysql = require('mysql2');
const { performance } = require('perf_hooks');
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'Petra123!',
  database: 'ambulanta',
});
const update2 = `
  UPDATE Osoblja
  SET Gradovi_ID_Grad = 3
  WHERE Gradovi_ID_Grad = 2;
`;
const start = performance.now();
connection.query(update2, (error, results) => {
  if (error) {
    console.error('Pogreška pri izvođenju upita:', error);
    return;
  }
  const kraj = performance.now();
  const vrijeme = kraj - start;
  console.log('Update upit za osoblje izvršen u ', vrijeme,
          'milisekunde');
});
connection.end();
```

Kod 12 *Upit za ažuriranje naziva grada(MySQL)*

```
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AmbulantaMYSQL> cd upiti
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AmbulantaMYSQL\upiti> node update2.js
Update upit za osoblje izvršen u  18.92550003528595 milisekunde
```

Slika 16 *Vrijeme izvršavanja upita za ažuriranje naziva grada(MySQL)*

```
const mongoose = require('mongoose');
const { performance } = require('perf_hooks');
const Osoblja = require('../modeli/osoblja');
const opcije = {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  wtimeoutMS: 30000,
};
mongoose.connect('mongodb+srv://pbacelic:patricijamongo@cluster0.uoy
r5ek.mongodb.net/ambulantaMongo', opcije);
const filter = { Gradovi_ID_Grad: '64e777106b6e8db6ba2bb1cd' };
const update = { $set: { Gradovi_ID_Grad: '64e777116b6e8db6ba2beaf9'
} };
const start = performance.now();
Osoblja.updateMany(filter, update)
  .then((result) => {
    const kraj = performance.now();
    const vrijeme = kraj - start;
    console.log('Update upit za osoblje izvršen u ', vrijeme,
'milisekunde');
  })
  .catch((error) => {
    console.error('Pogreška pri ažuriranju podataka:', error);
  });
```

Slika 17 *Upit za ažuriranje naziva grada(MongoDB)*

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS  SQL CONSOLE

PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AMBULANTAMONGODB> cd upiti
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AMBULANTAMONGODB\upiti> node update2.js
Update upit za osoblje izvršen u  2805.528900027275 milisekunde
```

Slika 18 *Vrijeme izvršavanja upita za ažuriranje naziva grada(MongoDB)*

Nakon izvršenja upita za ažuriranje naziva grada, MySQL baza je upit izvršila za 18.92 milisekunde, dok je MongoDB upit izvršio za 2805.53 milisekunde. Time je MySQL baza upit izvršila brže, za 2876.61 milisekundu.

Rezultati pokazuju da MySQL brže izvršava složene UPDATE upite.

4.4 Delete upiti

U ovom poglavlju se analiziraju performanse izvođenja DELETE upita u MySQL i MongoDB bazama podataka. DELETE upiti koriste se za brisanje podataka iz baza, a usporedit će se brzina izvođenja jednostavnih i složenih DELETE upita u oba sustava.

4.4.1 Jednostavni delete upiti

U ovom poglavlju uspoređuje se izvođenje jednostavnih DELETE upita za brisanje podataka u MySQL i MongoDB bazama podataka.

Prvi DELETE upit odnosi se na brisanje grada iz baza podataka. Ovaj primjer odabran je zbog česte potrebe za brisanjem u sustavu.

```
const mysql = require('mysql2');
const { performance } = require('perf_hooks');
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'Petra123!',
  database: 'ambulanta',
});
const delete1 = `
  DELETE FROM Gradovi
  WHERE ID_Grad = 100002;
`;
const start = performance.now();
connection.query(delete1, (error, results) => {
  if (error) {
    console.error('Pogreška pri izvođenju upita:', error);
    return;
  }
  const kraj = performance.now();
  const vrijeme = kraj - start;
  console.log('Delete upit za grad izvršen u ', vrijeme,
'milisekunde');
});
connection.end();
```

Kod 13 *Upit za brisanje grada(MySQL)*

```
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AmbulantaMYSQL> cd upiti
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AmbulantaMYSQL\upiti> node delete1.js
Delete upit za grad izvršen u  27.953400015830994 milisekunde
```

Slika 19 *Vrijeme izvršavanja upita za brisanje grada(MySQL)*

```
const mongoose = require('mongoose');
const { performance } = require('perf_hooks');
const Grad = require('../modeli/gradovi');
const opcije = {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  wtimeoutMS: 30000,
};
mongoose.connect('mongodb+srv://pbacelic:patricijamongo@cluster0.uoy
r5ek.mongodb.net/ambulantaMongo', opcije);
const filter = { _id: '64e777106b6e8db6ba2b9f0e' }; //briše se Split
const start = performance.now();
Grad.deleteOne(filter)
  .then((result) => {
    const kraj = performance.now();
    const vrijeme = kraj - start;
    console.log('Delete upit za grad izvršen u ', vrijeme,
'milisekunde');
  })
  .catch((error) => {
    console.error('Pogreška pri brisanju podataka:', error);
  });
```

Kod 14 *Upit za brisanje grada(MongoDB)*

```
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AMBULANTAMONGODB> cd upiti
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AMBULANTAMONGODB\upiti> node delete1.js
Delete upit za grad izvršen u  2855.1161999702454 milisekunde
```

Slika 20 *Vrijeme izvršavanja upita za brisanje grada(MongoDB)*

Upit za brisanje grada se također izvršio brže u MySQL bazi podataka. Izvršavanje je trajalo 27.95 milisekunda, dok je kod MongoDB baze podataka izvršavanje trajalo 2855.12 milisekunda. Time je MySQL baza podataka brža za 2827.17 milisekunda.

Rezultati pokazuju da MySQL brže izvršava jednostavne DELETE upite, razlog tome je što se modeliranje podataka u slučaju ove baze temelji na dokumentima, tada je pretpostavka da bi MongoDB bio brži.

4.4.2 Složeni delete upiti

Drugi DELETE upit odnosi se na brisanje osoblja prema gradu u kojem žive iz baza podataka. Ovaj primjer odabran je zbog složenosti i potrebe za brisanjem podataka iz više tablica

```
const mysql = require('mysql2');
const { performance } = require('perf_hooks');
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'Petra123!',
  database: 'ambulanta',
});
const deleteOsobljeQuery = `
  DELETE FROM Osoblja
  WHERE Gradovi_ID_Grad = 2;
`;
const start = performance.now();
connection.query(deleteOsobljeQuery, (error, results) => {
  if (error) {
    console.error('Pogreška pri izvođenju upita:', error);
    return;
  }
  const kraj = performance.now();
  const vrijeme = kraj - start;
  console.log('Delete upit za osoblje izvršen u ', vrijeme,
'milisekunde');
});
connection.end();
```

Kod 15 *Upit za brisanje osoblja s obzirom na grad u kojem žive(MySQL)*

```
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AmbulantaMYSQL> cd upiti
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AmbulantaMYSQL\upiti> node delete2.js
Delete upit za osoblje izvršen u  23.985899925231934 milisekunde
```

Slika 21 *Vrijeme izvršavanja upita za brisanje osoblja s obzirom na grad u kojem žive(MySQL)*

```
const mongoose = require('mongoose');
const { performance } = require('perf_hooks');
const Osoblja = require('../modeli/osoblja');
const opcije = {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  wtimeoutMS: 30000,
};
mongoose.connect('mongodb+srv://pbacelic:patricijamongo@cluster0.uoy
r5ek.mongodb.net/ambulantaMongo', opcije);
const filter = { Gradovi_ID_Grad: '64e777106b6e8db6ba2bb1cd' };
const start = performance.now();
Osoblja.deleteMany(filter)
  .then((result) => {
    const kraj = performance.now();
    const vrijeme = kraj - start;
    console.log('Delete upit za osoblje izvršen u ', vrijeme,
'milisekunde');
  })
  .catch((error) => {
    console.error('Pogreška pri brisanju podataka:', error);
  });
```

Kod 16 *Upit za brisanje osoblja s obzirom na grad u kojem žive(MongoDB)*

```
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AMBULANTAMONGODB> cd upiti
PS C:\Users\korisnik\Dropbox\PC\Desktop\Zavrsni_sve\AMBULANTAMONGODB\upiti> node delete2.js
Delete upit za osoblje izvršen u  3332.465399980545 milisekunde
```

Slika 22 *Vrijeme izvršavanja upita za brisanje osoblja s obzirom na grad u kojem žive(MongoDB)*

Upit za brisanje člana osoblja se također izvršio brže u MySQL bazi podataka. Izvršavanje je trajalo 23.98 milisekunda, dok je kod MongoDB baze podataka izvršavanje trajalo 3332.47 milisekunda. Time je MySQL baza podataka brža za 3308.49 milisekunda.

Tablica 1 prikazuje vrijeme izvršavanja svih upita u MySQL-u i MongoDB-u.

	MySQL(ms)	MongoDB(ms)
Select upit za pronalaženje svih gradova	98.28	25681.05
Select upit za pronalaženje svih pacijenata koji žive u nekom gradu	41.57	2618.54
Insert upit za dodavanje grada	33.18	3129.38
Insert upit za dodavanje člana osoblja	24.54	2791.27
Update upit za ažuriranje naziva grada	16.22	2603.06
Update upit za ažuriranje člana osoblja s obzirom na grad u kojem žive	18.92	2805.53
Delete upit za brisanje grada	27.95	2855.12
Delete upit za brisanje člana osoblja s obzirom na grad u kojem žive	23.98	3332.47

Tablica 1 *Usporedba izvršavanja upita između MySQL i MongoDB*

5 Zaključak

Kao što je u uvodu rečeno, glavni cilj rada je bio usporediti performanse između relacijskih i nerelacijskih baza podataka kako bi se donijela informacija, odnosno odluka o tome koju bazu podataka je bolje koristiti za određene potrebe.

Rezultati testiranja ukazuju na to da je MySQL baza podataka izvrsna opcija za implementaciju u scenarijima gdje je shema podataka dobro definirana i aplikacija zahtijeva složene upite i strogu konzistenciju podataka. Ova relacijska baza podataka je pokazala izvrsnu učinkovitost i brzinu obrade podataka, čineći je pouzdanim izborom za takve situacije.

Baza podataka na kojoj su se radile usporedbe performansi je inicijalno kreirana u relacijskoj bazi podataka, te je samim time rezultat performansi bolji u MySQL-u nego u MongoDB-u. MongoDB je dosta usporeniji, nego što bi u teoriji trebao biti, zbog referenca na druge dokumente.

Međutim, važno je napomenuti da postoji niz situacija u kojima bi MongoDB mogao biti preferirani izbor. Prvenstveno, MongoDB se ističe u slučajevima gdje je potrebna fleksibilna shema podataka i kada se radi s nestrukturiranim ili promjenjivim podacima. Također, MongoDB se posebno ističe u okruženjima koja zahtijevaju brze operacije čitanja i pisanja, kao i skalabilnost na razini više poslužitelja.

Drugi scenarij u kojem MongoDB može sjajno odgovarati je razvoj MVP-a (Minimum Viable Product) ili prototipa aplikacije, gdje brzina implementacije i iteracije može biti ključna. MongoDB omogućuje brzo postavljanje i prilagodbu sheme podataka tijekom razvoja.

U zaključku, odabir između MySQL i MongoDB ili bilo koje druge baze podataka ovisi o specifičnim potrebama projekta. Svaka od ovih baza podataka ima svoje prednosti i nedostatke, i važno je pažljivo razmotriti zahtjeve aplikacije kako bi donijeli odluku o tome koju bazi podataka koristiti.

Literatura

[1] Mirko Maleković, Kornelije Rabuzin: Uvod u baze podataka

[2] LucidChart: https://www.lucidchart.com (01.09.2023)

[3] MySQL: https://www.mysql.com (01.09.2023)

[4] MongoDB: https://www.mongodb.com (01.09.2023)

[5] Wikipedia, Relacijska baza podataka: https://hr.wikipedia.org/wiki/Relacijska_baza_podataka (01.09.2023)

[6] Wikipedia, NoSQL: https://en.wikipedia.org/wiki/NoSQL (01.09.2023)