Usporedba SQL server i MongoDB baza podataka na primjeru web aplikacije

Aglić Čuvić, Marina

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

DIPLOMSKI RAD

USPOREDBA SQL SERVER I MONGODB BAZA PODATAKA NA PRIMJERU WEB APLIKACIJE

Marina Aglić Čuvić

Split, rujan 2023.

Želim izraziti iskrenu zahvalnost doc. dr. sc. Moniki Mladenović na vrijednoj pomoći i korisnim savjetima tijekom cijelog procesa.

Također, želim se srdačno zahvaliti svojim prijateljima i obitelji koji su bili uz mene i bez kojih ovaj period ne bi prošao tako brzo i lako.

Temeljna dokumentacijska kartica

Diplomski rad

Sveučilište u Splitu

Prirodoslovno-matematički fakultet

Odjel za informatiku

Ruđera Boškovića 33, 21000 Split, Hrvatska

USPOREDBA SQL SERVER I MONGODB BAZA PODATAKA NA PRIMJERU WEB APLIKACIJE

Marina Aglić Čuvić

SAŽETAK

Cilj rada je istraživanje i usporedba performansi nad relacijskim i nerelacijskim bazama podataka. Za relacijsku bazu podataka smo koristili Microsoft SQL Server, a za nerelacijsku bazu podataka MongoDB. Analizirat ćemo i testirati njihove performance u operacijama dohvaćanja podataka, ažuriranja podataka i brisanja podataka koristeći različite skupove podataka počevši od 100 pa sve do 1000000. Rezultati ukazuju na to da je MS SQL baza podataka brže dohvaćala veće skupove podataka, te je brže ažurirala podatake, dok je MongoDB bio brži u dohvaćanju određenih informacija iz različitih skupova podataka i brisanju.

Ključne riječi: relacijska baza podataka; nerelacijska baza podataka; SQL Server; MongoDB

Rad sadrži: [49] stranica, [41] grafički prikaz i [42] literaturna navoda. Izvornik je na hrvatskom jeziku

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

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

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

Doc. dr. sc. Divna Krpan, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Doc. dr. sc. Goran Zaharija, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Rad prihvaćen: rujan 2023.

Basic documentation card

Thesis

University of Split

Faculty of Science

Department of computer science

Ruđera Boškovića 33, 21000 Split, Croatia

COMPARISON OF SQL SERVER AND MONGODB DATABASE ON THE EXAMPLE OF A WEB APPLICATION

Marina Aglić Čuvić

ABSTRACT

The goal of the thesis is to research and compare the performance of relational and non-relational databases. We used Microsoft SQL Server for the relational database, and MongoDB for the non-relational database. We will analyze and test their performance in data acceptance, data update and data deletion operations using different data sets starting from 100 up to 1000000. The results indicate that the MS SQL database accepted larger data sets faster, and updated data faster, while MongoDB was faster at retrieving specific information from different datasets and deleting it.

Key words: relational database; non-relational database; SQL Server; MongoDB

Thesis consists of: [49] pages, [41] figures and [42] references

Original language: Croatian

Mentor: **Monika Mladenović, Ph.D.,** *Assistant professor of Faculty of Science, University of Split*

Supervisor: **Monika Mladenović, Ph.D.,** *Assistant professor of Faculty of Science, University of Split*

Reviewers: **Monika Mladenović, Ph.D.,** *Assistant Professor of Faculty of Science, University of Split*

Divna Krpan, Ph.D., *Assistant Professor of Faculty of Science, University of Split*

Goran Zaharija, Ph.D., *Assistant Professor of Faculty of Science, University of Split*

Thesis accepted: September 2023.

Sadržaj

Uvod

Baze podataka ključan su dio informacijskog sustava za pohranjivanje, organiziranje i analizu podataka u modernom digitalnom dobu. Zbog velikog raspona zahtjeva koje korisnici postavljaju pred baze podataka, razvijene su mnoge vrste sustava.

Relacijske baze podataka, često koriste tablice i odnose među njima za pohranjivanje strukturiranih podataka. Umjesto upotrebe tablica, nerelacijske baze podataka promiču fleksibilniji pristup pohrani podataka čestim korištenjem JSON dokumenta ili drugih oblika. Svaka od ovih baza ima različite kvalitete i prednosti.

Ovaj diplomski rad ima za cilj istražiti i usporediti performanse relacijskih i nerelacijskih baza podataka, fokusirajući se na različite aspekte obrade podataka. U svrhu istraživanja, koristi ćemo Miscrosoft SQL Server kao predstavnika relacijske baze podataka i MongoDB kao predstavnika nerelacijske baze podataka. Analizirat ćemo njihove performanse u kontekstu dohvaćanja, ažuriranja i brisanja podataka koristeći skupove podataka različitih veličina, od manjih sa 100 unosa do velikih s milijunima unosa. Ove operacije ključne su u svakodnevnom radu s bazama podataka te su od presudnog značenja za pravilno funkcioniranje informacijskih sustava.

Iako i relacijski i nerelacijski sustavi imaju prednosti i nedostatke, ovaj rad će se usredotočiti na to koliko dobro funkcioniraju u stvarnim scenarijima upotrebe kako bi se steklo dublje razumijevanje njihovih prednosti i nedostataka. Kako bi se zajamčila usporedivost rezultata, analiza će se provesti pod pretpostavkom da obje baze imaju istu strukturu podataka i podatke.

Kroz sljedeće poglavlje, analizirat ćemo teorijski dio relacijskih i nerelacijskih baza podataka, nakon čega ćemo dublje opisati naš istraživački proces i zaključke do kojih smo iz njega dobili.

Ovim istraživanjem težimo pružiti dublje razumijevanje prednosti i nedostataka oba pristupa u praksi, čime ćemo pomoći organizacijama i stručnjacima u donošenju odluka o odabiru baze podataka koje najbolje odgovaraju njihovim specifičnim potrebama.

1. Baze podataka

Baza podataka je organizirana zbirka podataka elektronički pohranjenih u računalni sustav. [1] Osigurava da se podaci, koji su organizirani, lako i brzo pronađu. Istu bazu podataka mogu koristiti istovremeno i korisnici i aplikacije. Podaci u bazi podataka su u obliku koji je neovisan o aplikacijama koje ih koriste. [2] Također, korisnik kada koristi bazu podataka ne mora biti upoznat sa njenom tehničkom strukturom zato što koristi njenu logičku strukturu. Temeljni princip iza baze podataka je, umjesto da svaka aplikacija stvara svoje datoteke za pohrani, sve one dijele zajedničku zbirku podataka. Aplikacije nemaju izravan pristup podacima koji su pohranjeni na disku već koriste programski sustav za upravljanje tim podacima.

Sustav za upravljanje bazom podataka (eng. *Database Managment System, DBMS*) je programski sustav koji upravlja svim aktivnostima baze podataka. Stvaranje strukture, brisanje, ažuriranje i dohvaćanje podataka primjeri su operacija nad bazama podataka. Neke od aktivnosti proizlaze iz samih korisnika koji, unatoč tome što ništa ne znaju o bazi podataka, na neki je način kontroliraju. Sustav za upravljanje bazom podataka zadužen je za pohranu podataka, administraciju sustava i oporavak podataka u slučaju pada baze podataka. [2]

Arhitektura DBMS-a omogućuje da se sustav baze podataka podijeli na diskretne komponente koje se mogu individualno prilagođavati, mijenjati i nadomjestiti. Također pomaže u razumijevanju komponenti baze podataka. Baza podataka sadrži važne podatke i omogućuje korisnicima siguran pristup. Kao rezultat toga, odabir prave arhitekture DBMS-a olakšava jednostavno i učinkovito upravljanje podacima. [3]

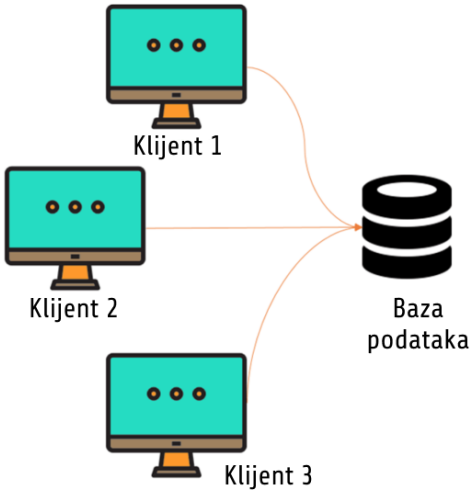
Postoje uglavnom tri vrste DBMS arhitekture [3]:

1. Jednoslojna arhitektura
2. Dvoslojna arhitektura
3. Troslojna arhitektura

Najjednostavnija arhitektura baze podataka, poznata kao jednoslojna arhitektura, stavlja klijenta, poslužitelja i bazu podataka na isti sustav. Instaliranje baze podataka u Vaš sustav i pristup njoj jednostavan je primjer jednoslojne arhitekture. Međutim takva se arhitektura rijetko primjenjuje u proizvodnji. [3]

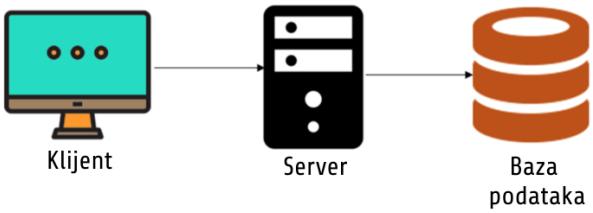
Slika 1. Jednoslojna arhitektura DBMS-a

Dvoslojna arhitektura je arhitektura u kojoj prezentacijski sloj radi na klijentu (računalo, mobilni uređaj, tablet, itd.), a podaci se pohranjuju na poslužitelju. DBMS je sigurniji s dvoslojnom arhitekturom jer nije odmah izložen krajnjem korisniku. Također, omogućuje izravniju i bržu komunikaciju. [3]



Slika 2. Dvoslojna arhitektura DBMS-a

Troslojna arhitektura je najčešća arhitektura u DBMS-u. Ona odvaja razvoj i održavanje funkcionalnih operacija, logike, pristupa podacima, pohrane podataka i korisničkog sučelja u diskretne module. Prezentacijski sloj, aplikacijski sloj i poslužitelj čine troslojnu arhitekturu. [3]

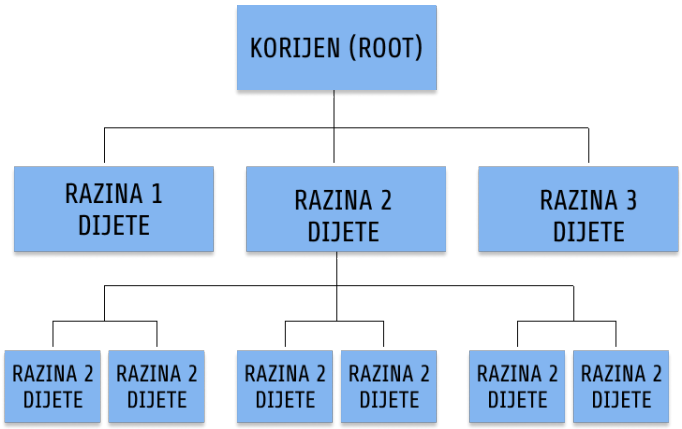


Slika 3. Troslojna arhitektura DBMS-a

Napretkom računalne tehnologije počeli su se otklanjati nedostatci baza podataka tog razdoblja koji nisu bili učinkoviti zbog navigacije kroz tonu papira. U 1960-im, prvi modeli

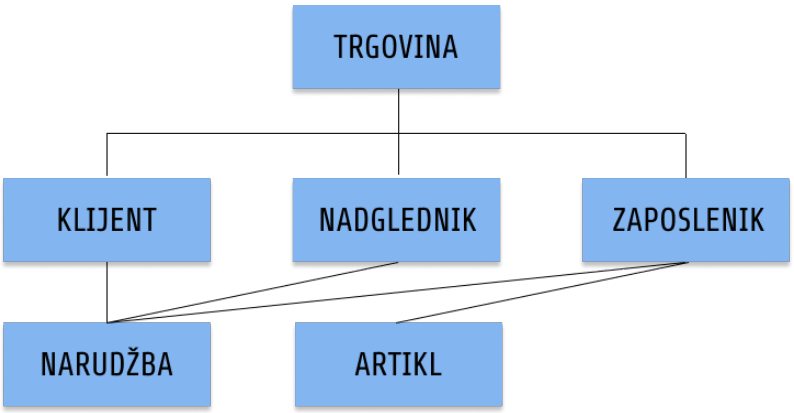
baza podataka razvijeni su s uvođenjem prvih računala. To su hijerarhijski model (eng. *hierarchical model*) i mrežni model (eng. *network model*).

Hijerarhijski model (eng. *hierarchical model*) baze podataka zasniva se na hijerarhijskim strukturama koje je razvio IBM (*International Business Machine*). Podaci u ovom modelu su raspoređeni u strukturu nalik stablu (eng. *tree like structure*). Pohranjuju se u obliku zapisa koji su zbirka polja (eng. *field*) te su povezani putem veza, odnosno vezom „jedan-na-više“ (1:N). U vezi „jedan-na-više“, element na jednoj strani relacije se naziva roditelj i povezan je s elementima nižeg nivoa, tj. djecom. Korijen (eng. *root*) u hijerarhijskoj bazi podataka treba biti samo roditelj, a ostali nekorijenski elementi su djeca. Pretraživanje se vrši tako što se prolazi kroz svaki element stabla. Hijerarhijske baze podataka pristupaju podacima koristeći hijerarhijski jezik za manipulaciju podacima (eng. *hierarchical data manipulation language*). [4]



Slika 4. Hijerarhijski model baze podataka

Mrežni model (eng. *network model*) je poboljšanje u odnosu na hijerarhijski model. Namijenjen je rješavanju pitanja hijerarhijskog modela, specifičnosti i fleksibilnosti. Omogućuje prikaz odnosa „više-na-više” (M:N), uz odnos „jedan-na-više“ (1:N). U odnosu „više-na-više“ svako dijete može imati više od jednog roditelja. Mrežni model se sastoji od skupa zapisa i skupa veza, pri čemu svaki zapis sadrži podatke iz jedne instance entiteta i polja koja odgovaraju karakteristikama. U svakom polju postoji jedna vrijednost atributa. Pogodnost prikazivanja i upravljanja podacima je prednost ove arhitekture. Međutim, u mrežnom modelu predstavljanje M:N („više-na-više“) veze nije optimalan odgovor, pa su te veze prikazuju pomoću na 1:M i 1:N veza. Za pristup podacima, programski jezik računala koristi jezik za upravljanje podacima (eng. *data manipulation language,* DML). Prednosti ove arhitekture uključuju brz pristup podacima, dobro upravljanje integritetom i neovisnost podataka, dok nedostaci uključuju složenost sustava i tešku implementaciju. [4]



Slika 5. Mrežni model baze podataka

2. Relacijska baza podataka

Edgar Frank Codd, engleski matematičar, stvorio je relacijskih model baze podataka. Opisao ga je svom članku „*A Relational Model for Data for Large Shared Data Banks*“, objavljenom 1970. godine. U to je vrijeme Edgar bio zaposlen u IBM-u (*International Business Machine*). Oni su trebali biti prvi koji će implementirati relacijski model pohrane podataka, ali se nije dogodilo zbog IBM-ove tvrdoglavosti u držanju zastarjele hijerarhijske paradigme u IMS (*Integrated Managment System*) sustavu. *Oracle*, koji je i danas sinonim za baze podataka, bio je prva korporacija koja je primijenila relacijski model baze podataka. [1]

Relacijske baze podataka danas se široko koriste zbog svoje lakoće razumijevanja i praktične upotrebe. Relacijski model baza podataka se sastoji od više tablica. Svaka tablica se sastoji od više stupaca te svaki od tih stupaca ima jedinstveno ime. Osnovni ciljevi relacijskog modela baze podataka su [5]:

1. omogućiti nezavisnost podataka,
2. dati teorijski okvir za dosljednu obradu semantičkih podataka i bavljenje redundantnošću podataka,
3. omogućiti razvoj skupno orijentiranih jezika za obradu podataka,
4. dati bogat model podataka za opis i obradu jednostavnih i kompleksnih podataka

Koristeći zajedničko polje, relacijski model omogućuje povezivanje bilo koje datoteke s bilo kojom drugom. Složenost modela je pala jer su se modifikacije mogle napraviti bez narušavanja sposobnosti sustava za pristup podacima. Za lakše razumijevanje relacijske baze podataka, proći ćemo neke od osnovnih pojmova.

2.1. Modeliranje podataka

Entiteti i atributi su najčešći pojmovi DBMS-a.

Entitet (eng. *entity*) je bilo koji objekt iz stvarnog svijeta. Entitet može predstavljati neku stvar, osobu ili mjesto. Zapravo, to je ono o čemu želimo prikupljati te će ti podaci biti pohranjeni u bazi podataka.

Atribut (eng. *attribute*) predstavlja svojstvo nekog entiteta. U bazama podataka, atributi pobliže opisuju entitete.

U relacijskoj bazi podataka prikupljamo podatke u obliku tablice. Dakle, redovi tablice predstavljaju entitete istog tipa, a stupci tablice se smatraju atributima entiteta prisutnih u toj tablici.

Jedna od najznačajnijih prednosti relacijske baze podataka je mogućnost povezivanja podataka smještenih u više tablica. U relacijskoj bazi podataka, odnosi između entiteta su predstavljeni putem veza ili relacija (eng. *relation*). U bilo kojem trenutku, tablice su uključene u samo jedan oblik relacije. Kada su dvije tablice povezane, podaci iz tih povezanih tablica se mogu učinkovito kombinirati u svrhu stvaranja upita, obrazaca i izvješća. Najčešće su korištene binarne veze (eng. *binary relationships*) zato što imaju najviše funkcionalnosti [6]:

1. 1:1 (jedan naprema jedan) – Jedan zapis u prvoj tablici može biti povezan samo s jednim zapisom u drugoj tablici, također jedan zapis u drugoj tablici može biti povezan samo s jednim zapisom u prvoj tablici.
2. 1:N (jedan naprema više) – Jedan zapis u prvoj tablici može biti povezan s 0, 1 ili više zapisa u drugoj tablici, ali jedan zapis u drugoj tablici može biti povezan samo s jednim zapisom u prvoj tablici.
3. M:N (više naprema više) – Jedan zapis u prvoj tablici može biti povezan s 0, 1 ili više zapisa u drugoj tablici, također jedan zapis u drugoj tablici može biti povezan s 0,1 ili više zapisa u prvoj tablici. Međutim, ovakve veze nisu poželjne u bazi podataka te se taj problem rješava uvođenjem novog entiteta čiji će se primarni ključ sastojati od dva strana ključa dvije povezane tablice.

Svaki redak u tablici treba imati jedinstveni identifikator. Tome nam služi primarni ključ (eng. *primary key*) koji jedinstveno određuje pojedini entitet. Naziv tablice, naziv stupca i primarni ključ retka potrebni su kako bi pronašli važne podatke. Zbog toga je nužno da primarni ključ bude jedinstven jer nam to osigurava da baza podataka daje podatke koje smo tražili.

Kompozitni primarni ključ (eng. *composite primary key*) je vrsta primarnog ključa koja se kreira korištenjem dva ili više atributa, odnosno s više stupaca tablice. Atributi koji čine kompozitni primarni ključ mogu biti različitih tipova podataka što je bitno za naglasiti.

Strani ključ (eng. *foreign key*) opisuje atribut unutar tablice koji se koristi kao identifikacijski ključ u barem jednoj drugog tablici. Identifikacijski ključevi mogu se ponovno koristiti u drugim tablicama za stvaranje željenih veza među tablicama .

2.2. Upitni jezik relacijskih baza podataka

Relacijskim bazama podataka se upravlja korištenjem standardiziranog programskog jezika poznatog kao SQL (eng. *Structured Query Language*) koji se također koristi za upravljanje i manipulaciju podacima koji su pohranjeni u tim bazama podataka. SQL se koristi za sljedeće [7]:

1. Modificiranje strukture tablica i indeksa baze podataka
2. Dodavanje, ažuriranje i brisanje redaka podataka
3. Dohvaćanje podskupova informacija unutar sustava upravljanja relacijskim bazama podataka (RDBMS) – te se informacije mogu koristiti za obradu transakcija, analitičke aplikacije i druge aplikacije koje zahtijevaju komunikaciju s relacijskom bazom podataka

SQL upiti i druge operacije izraženi su kao izjave koje se kompiliraju u programe koji korisnicima omogućuju dodavanje, promjenu i dohvaćanje podataka iz tablice baze podataka. SQL kao jezik ima naredbe i sintaksu za izdavanje tih naredbi [7]:

1. Jezik za definiranje podataka (eng. *Data Definition Language*, DDL) – programski jezik koji se koristi za definiranje tablice podataka.
2. Jezik za manipuliranje podacima (eng. *Data Manipulation Language*, DML) – strukturirani računalni jezik koji se koristi u bazama podataka za manipuliranje podacima u tablici dodavanjem, promjenom ili uklanjanjem podataka.
3. Upitni jezik (eng. *Data Query Language*) – jezik upita koji se sastoji od samo jedne naredbe, *SELECT*, koja služi za dobivanje određenih podataka iz tablice.
4. Jezik kontrole podataka (eng. *Data Control Language*) – sadrži naredbe koje se koriste za dodjelu ili opoziv privilegija korisničkog pristupa u bazi podataka.
5. Jezik kontrole transakcija (eng. *Data Transaction Language*) – sadrži naredbe koje se koriste za promjenu stanja nekih podataka u bazi podataka.

2.3. Sustavi za upravljanje relacijskim bazama podataka

Sustav upravljanja relacijskim bazama podataka izgrađen je na relacijskom modelu (eng. *Relational Database Managment System*, *RDBMS*). U osnovi, RDBMS prenosi podatke u bazu podataka, tamo ih čuva i dohvaća kako bi ih aplikacije mogle mijenjati. RDBMS razlikuje sljedeće vrste operacija [8]:

1. Logičke operacije – u ovom slučaju aplikacija određuje koji je sadržaj potreban.
2. Fizičke operacije – u ovom slučaju RDBMS određuje kako se stvari trebaju učiniti i izvodi operaciju.

Prema web-stranici *DB-Engines*, koja rangira sustave za upravljanje bazama podataka, najkorišteniji sustavi su: Oracle, MySQL i Microsoft SQL Server.

Rank			DBMS	Database Model	Score		
Sep 2023	Aug 2023	Sep 2022			Sep 2023	Aug 2023	Sep 2022
1.	1.	1.	Oracle	Relational, Multi-model	1240.88	-1.22	+2.62
2.	2.	2.	MySQL	Relational, Multi-model	1111.49	-18.97	-100.98
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	902.22	-18.60	-24.08
4.	4.	4.	PostgreSQL	Relational, Multi-model	620.75	+0.37	+0.29
5.	5.	5.	MongoDB	Document, Multi-model	439.42	+4.93	-50.21
6.	6.	6.	Redis	Key-value, Multi-model	163.68	+0.72	-17.79
7.	7.	7.	Elasticsearch	Search engine, Multi-model	138.98	-0.94	-12.46
8.	8.	8.	IBM Db2	Relational, Multi-model	136.72	-2.52	-14.67
9.	↑ 10.	↑ 10.	SQLite	Relational	129.20	-0.72	-9.62
10.	↓ 9.	↓ 9.	Microsoft Access	Relational	128.56	-1.78	-11.47

Slika 6. DB-Engines rang sustava za upravljanje bazama podataka

Larry Ellison, Bob Miner i Ed Oates pokrenuli su 1977. godine tvrtku *Software Development Laboratories* koji je kasnije postao *Relational Software Inc* (RSI). RSI je preimenovan 1983. godine u *Oracle System Corporation*, a zatim u *Oracle Corporation*. [8]

Prijelomni trenutak u povijesti relacijskih baza podataka bio je uvođenje Oracle V2 (verzija 2) kao prvog komercijalno dostupnog RDBMS-a baziranog na SQL-u od strane RSI-a 1979. godine. [8]

Oracle V3 (verzija 3) bila je prva relacijska baza podataka koja je radila na glavnim računalima, miniračunalima i osobnim računalima, a objavljena je 1983. godine. Kako je baza podataka bila razvijena u C-u, mogla se prenijeti na razne platforme. [8]

Viševerzijska konzistentnost čitanja implementirana je u V4 (verzija 4). Računanje klijent/poslužitelj i sustavi distribuirane baze podataka podržani su u V5 (verzija 5), koja je uvedena 1985. godine. Disk I/O, zaključavanje redaka, skalabilnost i sigurnosno kopiranje te oporavak poboljšani su u V6 (verzija 6). Također, V6 doživjela je debi PL/SQL jezika, vlasničkog proceduralnog proširenja za SQL. [8]

Mrežno računalstvo prvi put je implementirano u Oracle Database 10g 2003 godine. Izgradnjom mrežne arhitekture temeljene na jeftinim robnim poslužiteljima, tvrtke su mogle vizualizirati mrežne resurse. Glavni cilj bio je učiniti bazu podataka samoupravljačkom i samopodešavajućom. Virtualizacijom i pojednostavljenjem upravljanja pohrane baze

podataka, *Oracle Automatic Storage Managment* (Oracle ASM) pomogao je postizanju ovog cilja. [8]

Oracle Database 11g koji je objavljen 2007. godine, dodao je niz novih mogućnosti koje administratorima i programerima omogućuju da lako reagiraju na promjenjive poslovne potrebe. Ključ za prilagodbu je racionalizacija informacijske infrastrukture kombiniranjem podataka i automatiziranjem procesa gdje je to moguće. [8]

Oracle Database pomaže da brzo i sigurno pohranimo i dohvatimo podatke te da radi na raznim hardverskim i operativnim sustavima uključujući Windows Server, Unix i nekoliko GNU/Linux verzija. Ima vlastiti mrežni stog koji omogućuje aplikacijama s različitih platformi da komuniciraju s Oracle bazom podataka koja se temelji na Unix-u. [9]

Slika 7. Oracle logo

MySQL je sustav za upravljanje relacijskim bazama podataka bazama podataka koji je besplatan i otvorenog koda (eng. *open-source*). MySQL, kao i druge relacijske baze podataka, organizira podatke u tablice s recima i stupcima. SQL ili strukturirani jezik upita, je programski jezik koji korisnicima omogućuje definiranje, manipulaciju, kontrolu i upit nad podacima. [10]

MySQL je razvio MySQL AB, švedska tvrtka koju su osnovali David Axmark, Allan Larsson i Michael „Monty“ Widenius iz Finske. Widenius i Axmark su prvi razvili MySQL 1994. godine, a prvi put je objavljen 23. svibnja 1995. Izvorno je napisan u mSQL-u i baziran na jeziku niske razine ISAM (*Indexed Sequential Access Method*), za koji su dizajni otkrili da je prespor i krut za osobnu upotrebu. Dizajnirali su novo SQL sučelje uz održavanje mSQL API-ja. [11]

Baziran je na modelu klijent-poslužitelj. Temeljna komponenta MySQL-a je MySQL poslužitelj koji upravlja svim naredbama baze podataka. MySQL poslužitelj dostupan je kao samostalni program i kao biblioteka (eng. *library*) koja se može ugraditi u druge programe. Kreiran je s ciljem brze obrade velikih baza podataka. Unatoč činjenici da je MySQL inače instaliran na jednom sustavu, on može prenijeti bazu podataka na nekoliko mjesta jer joj korisnici mogu pristupiti putem različitih MySQL klijentskih sučelja. Ova sučelja

komuniciraju s poslužiteljem pomoću SQL naredbi i zatim pokazuju rezultate. MySQL je napisan u C i C++ programskim jezicima i dostupan na preko 20 platformi. [12]

On je najpopularniji sustav otvorenog koda zbog svoje fleksibilnosti i snage. Koristi se za pohranu i dohvaćanje podataka u širokom rasponu popularnih aplikacija, web-mjesta i usluga kao dio široko korištene tehnologije LAMP (*Loop-mediated Isothermal Amplification*), koji se sastoji od operacijskog sustava temeljenog na Linuxu, web poslužitelja Apache, baze podataka MySQL i PHP-a za obradu. [10]

Slika 8. MySQL logo

Microsoft je razvio sustav za upravljanje relacijskim bazama podataka poznat kao *Microsoft SQL Server*. To je softverski proizvod poznat kao poslužitelj baze podataka, a njegov glavni zadatak je pohranjivanje i dohvaćanje podataka kada drugi softverski programi to zahtijevaju. Ti drugi softverski programi mogu raditi na istom računalu ili na drugom računalu preko mreže. Najčešći način za dobiti podatke u bazi podataka SQL Servera je preko upita (eng. *query*), a sam upit se izražava pomoću varijante SQL-a nazvane T-SQL (*Transact-SQL*). T-SQL je proširenje proceduralnog jezika SQL Servera koje je razvio Microsoft. Za upute za manipulaciju podacima (DML) i definiranje podacima (DDL) nudi *REPL* (*Read-Eval-Print-Loop*) instrukcije koje proširuju uobičajeni skup instrukcija SQL-a tako da uključuju postavke specifične za SQL Server, sigurnost i upravljanje statistikom baze podataka. [13]

Zbog svog vizualnog sučelja te opcija i alata, SQL Server je prikladan za stvaranje baza podataka, pohranjivanje svih potrebnih informacija u relacijske baze podataka kao i za upravljanje tim podacima bez problema. To je posebno važno npr. za web stranice koje sadrže registraciju kako bi se korisnici prijavili. [14]

SQL Server ima niz ključnih značajki koje ga izdvajaju, uključujući široki raspon alata za upravljanje podacima i analizu, kao i alate poslovne inteligencije koji se mogu koristiti za dobivanje uvida o tvrtki i klijentima uz pomoć strojnog učenja. [14]

SQL poslužitelji pružaju visoku dostupnost kako bi omogućili brže operacije prebacivanja. Njegove značajke u memoriji omogućuju besprijekornu vezu s obitelji poslužitelja Microsoft Servera, omogućujući poboljšanu fleksibilnost i jednostavnost korištenja. [14]

Slika 9. Microsoft SQL Server logo

2.4. ACID svojstva

Zbirka karakteristika za operacije baza podataka poznata kao ACID (eng. *atomicy, consistency, isolation, durability*) u informatici ima za cilj osigurati valjanost podataka unatoč pogreškama, nestanku struje i drugim nepogodama. [15]

Atomarnost (eng. *atomicy*), zapravo atomska transakcija osigurava da svaki *commit* koji napravimo ispravno dovršava cijelu proceduru. Ili, ako se veza izgubi usred akcije, baza podataka se vraća na stanje u kojem je bila prije nego što je *commit* pokrenut. Ovo je ključno za izbjegavanje situacija u kojima je transakcija djelomično dovršena, što dovodi do nejasnog ukupnog stanja kao rezultat pada ili prekida rada. Koristeći atomičnost, osiguravamo da je cijela transakcija bila uspješna -ili da nijedna nije -ili da je na neki način bila neuspješna. [16]

Dosljednost (eng. *consistency*) se odnosi na održavanje integriteta podataka. Ograničenja integriteta koja su podacima nametnuta pravilima baze podataka neće biti prekinuta dosljednom transakcijom. Provedba dosljednosti jamči da ako baza podataka prijeđe u nezakonito stanje, postupak će biti zaustavljen i izmjene će se poništiti, vraćajući bazu u izvorno stanje. Provođenje deklarativnih ograničenja nametnutih bazi podataka još je jedna tehnika koja osigurava dosljednost unutar nje tijekom svake transakcije. [16]

Izolacija (eng. *isolation*), odnosno izolirane transakcije, smatraju se „serijskim“ što znači da se svaka transakcija odvija u zasebnom nizu bez ikakvih transakcija koje se odvijaju istovremeno. Čitanja ili pisanja napravljena kao dio različitih transakcija koje se odvijaju u istoj bazi podataka neće utjecati na bilo koje čitanje ili pisanje u bazu podataka. Kako bi se osiguralo da se svaka transakcija završi prije nego započne druga, formira se globalni poredak gdje svaka transakcija stoji u redu. Važno je za napomenuti da se više transakcija može odvijati sve dok nijedna od njih ne može utjecati na druge transakcije koje se odvijaju istovremeno. To bi moglo usporiti transakcije budući da će nekoliko postupaka možda morati pričekati prije nego što započnu, međutim izolacije daje veću zaštitu podataka, stoga

se isplati. Izolacija se može postići upotrebom klizne ljestvice permisivnosti koja ide između onoga što se naziva optimističnim transakcijama i pesimističnim transakcijama:

1. Optimistična transakcijska shema pretpostavlja da sljedeće transakcije neće čitati ili pisati istu lokaciju više od jednom. Ako transakcija pogodi dvaput isto mjesto, ova shema će prekinuti obje transakcije i pokušati ih ponovno.
2. Pesimistična transakcijska shema pruža manje slobode i zaključat će resurse pod pretpostavkom da će transakcije utjecati na druge. To rezultira manjih brojem prekida i ponovnih pokušaja, ali također znači da su transakcije prisiljene čekati u redu. [16]

Izdržljivost (eng. *durability*) osigurava da će promjene na bazi podataka koje su uspješno izvršene trajati neograničeno dugo, čak i u slučaju kvarova sustava. Ovo jamči da podaci baze podataka neće biti neovlašteno mijenjani od strane prekida usluge, padova i drugih slučajnih kvarova. Izdržljivost se postiže upotrebom zapisa promjena (eng. *changelogs*) koji se pozivaju kada se baze podataka ponovno pokrenu. [16]

3. Nerelacijske baze podataka

3.1. Big data i BigTable

Velika količina podataka (eng. *big data*) je skup informacija koji ima ogroman volumen i uvijek se brzo širi. Nijedan tipičan sustav za upravljanje podacima ne može učinkovito pohraniti ili analizirati te podatke zbog njihove veličine i složenosti. To je vrsta podataka koja je, kao što i samo ime kaže, iznimno velika. Najbolji primjer velike količine podataka bi bile društvene mreže, *Facebook*, *Instagram*, itd. Statistika pokazuje da se 500+ terabajta novih podataka svaki dan unese u baze podataka društvene mreže *Facebook*. Ti podaci se uglavnom generiraju u smislu prijenosa fotografija, videozapisa, razmjena poruka, objavljivanja komentara, itd. Velika količina podataka može biti strukturirana, nestrukturirana i polustrukturirana. [17]

Vlasnički sustav baze podataka pod nazivom *Google BigTable* kreirala je američka tvrtka Google za iznimno velike količine podataka. Distribuirani sustavi klastera koje koristi visoko skalabilni NoSQL sustav baze podataka pružaju iznimne performanse. Google koristi BigTable za vlastite svrhe, uključujući Gmail, Google Analytics, Google Maps i Google Search. Google BigTable dobar je izbor i za druge Big Data aplikacije zbog niske latencije i velike propusnosti podataka. U bazi podataka namjerno se koristi podatkovni model koji se temelji na retku i stupcu. Podaci se također pohranjuju metodama kompresije. BigTable nastao je 2004. godine i ima značajan utjecaj na način na koji su baze podataka dizajnirane za velike podatkovne aplikacije. Druge tvrtke i timovi otvorenog koda uspjeli su stvoriti vlastite sustave baza podataka s usporedivim mogućnostima i strukturom na temelju Googleovih objavljenih specifikacija za BigTable. [18]

Tablica 1. Broj 1000-bajtnih vrijednosti pročitanih/napisanih po sekundi [19]

Experiment	**# of Tablet Servers**			
	1	**50**	**250**	**500**
random reads	1212	593	479	241
random reads (mem)	10811	8511	8000	6250
random writes	8850	3745	3425	2000
sequential reads	4425	2463	2625	2469
sequential writes	8547	3623	2451	1905
scans	15385	10526	9524	7843

U Tablici 1 možemo vidjeti broj operacija u sekundi po poslužitelju tableta. Test nasumičnog čitanja usporediv je s mjerilom nasumičnog čitanja, međutim, budući da je grupa lokaliteta koja sadrži referentne podatke označena kao da se nalazi u memoriji, čitanja se izvršavaju iz memorije poslužitelja tableta, a ne iz Google datotečnog sustava. [19]

Dizajn relacijskih baza podataka temeljio se na ideji da će trebati samo jedan stroj za njihovo izvršavanje. Osim toga, dizajnirane su prije nego je Internet postao iznimno raširen. Milijuni i milijarde umreženih ljudi i uređaja imaju potencijal generirati količinu podataka koja je znatno veća od one koju može obraditi jedan poslužitelj. [20]

Nadmoć relacijskog modela trenutno je dovedena u pitanje korištenjem NoSQL-a u 2010-ima. Naziv *NoSQL* je nesretan jer se zapravo ne odnosi ni na jednu posebnu tehnologiju, izvorno je zamišljeno jednostavno kao privlačan *hashtag* (#) na Twitteru za susret o distribuiranim, nerelacijskim bazama podataka otvorenog koda 2009. godine. Međutim, izraz je bio pun pogodak i ubrzo je stekao popularnost među digitalnim *startup* tvrtkama i šire. [21]

Nerelacijska baza podataka (eng. *non-relational database*) je ona koja ne koristi standardnu tabličnu shemu redaka i stupaca koja se vidi kod relacijske baze podataka. Nerelacijske baze podataka koriste model pohrane koji je prilagođen individualnim potrebama podataka koji se pohranjuju. Podaci se mogu pohraniti na različite načine, a svi načini pohrane podataka imaju jednu zajedničku stvar, a to je da ne koriste relacijske paradigme. Skladišta podataka kojima nije potreban SQL za upite nazivaju se NoSQL, a NoSQL je kratica za *non-relational database*, odnosno nerelacijske baze podataka. Pohrana podataka koristi različite računalne jezike i tehnike za upit nad podacima. [22]

Podatke je sve lakše dobiti i zabilježiti u današnjem svijetu zbog platformi kao što su *Facebook*, *Google+* i druge. Osobni korisnički podaci, društveni grafikoni, geolokacijski podaci i korisnički generirani sadržaji su samo neke od vrsta podataka koje rastu eksponencijalno. Za pravilo pružanje navedene usluge potrebno je obraditi veliku količinu podataka i za to je namijenjena NoSQL baza podataka. [23]

Carlo Strozzi je 1998. godine smislio frazu NoSQL kako bi opisao svoju bazu podataka otvorenog koda kojoj je nedostajalo sučelje. Idući koji je upotrijebio termin NoSQL bio je Eric Evans početkom 2009. godine. Upotrijebio je termin NoSQL koji označava baze podataka koje su nerelacijske, distribuirane i ne pridržavaju se tradicionalnih sustava relacijskih baza podataka. [23]

Postoje četiri kategorije NoSQL baza podataka [23]:

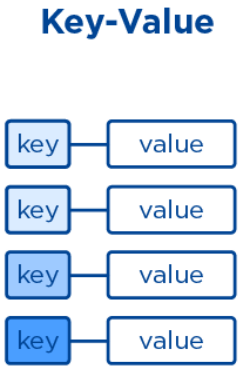
1. Ključ-vrijednost baze podataka (eng. *Key-value databases*)
2. Grafovske baze podataka (eng. *Graph databases*)
3. Dokumentne baze podataka (eng. *Document databases*)
4. Stupčaste baze podataka (eng. *Columnar databases*)

3.2. Vrste nerelacijskih baza podataka

Ključ-vrijednost baza podataka je NoSQL baza podataka koja dohvaća povezane vrijednosti pomoću pojedinačnih ili kombinacija ključeva i one nemaju specifičnu strukturu. Vrijednosti se pridružuju svojim ključevima koji su jedinstveni identifikatori. Također, ključ može biti bilo što, ali budući da je jedino sredstvo za dohvaćanje vrijednosti, treba ga pažljivo imenovati. Vrijednosti mogu biti bilo koji oblik objekta kao što je broj ili tekst, ili čak neka druga kombinacija ključ-vrijednost. [24]

Ove baze podataka rade tako da čuvaju strukturu podataka u memoriji koja je mapirana na podatke pohranjene na disku. Sposobnost stvaranja, uređivanja, dohvaćanja i uklanjanja podataka pomoću ključeva jedna je od najosnovnijih karakteristika koju svaka ključ-vrijednost baza podataka mora imati. [25]

Neki smatraju da je ovaj tip NoSQL baze podataka najjednostavniji. Prikladan je za skalabilne strukture podataka i druge poslovne aplikacije koje zahtijevaju fleksibilnost. Baze podataka ključ-vrijednost pružaju mehanizam za pohranu podataka u obliku koji omogućuje fleksibilnije dohvaćanje i također prihvaća izmjene sheme tijekom razvoja aplikacije.

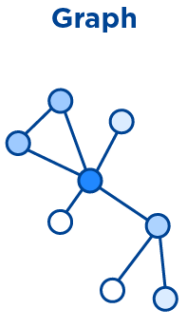


Slika 10. Ključ-vrijednost baza podataka

Baza podataka grafa je oblik NoSQL baze podataka koja koristi topografsku mrežnu strukturu za pohranu podataka. Koncept dolazi iz teorije grafova u kojoj se čvorovi, relacije i svojstva koriste za predstavljanje skupova podataka u grafovima. [26]

Čvorovi (eng. *nodes*) predstavljaju entitete u grafu. Oni se mogu označiti oznakama koje će predstavljati različite uloge. Također, mogu sadržavati bilo koji broj parova ključ-vrijednost ili svojstva (eng. *properties*). Relacije (eng. *relationship*) tvore vezu između dva entiteta čvora i uvijek imaju smjer, tip te početni i krajnji čvor. [27]

Prednost ovih baza podataka u odnosu na relacijske je to što se ne trebaju koristiti *JOIN* upit za dobiti željene podatke. U bazama podataka grafa te veze se pohranjuju zajedno s podacima. Sve u sustavu je dizajnirano da brzo prolazi kroz podatke te svaka jezgra može podnijeti milijune veza u sekundi. [27]



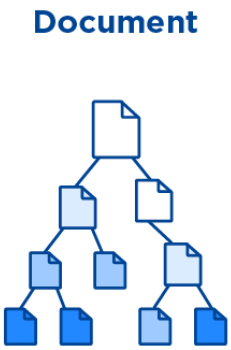
Slika 11. Baza podataka grafa

Dokumentna baza podataka je oblik NoSQL baze podataka koja umjesto stupaca i redaka pohranjuje podatke kao JSON (*JavaScript Object Notation*) dokumente. JSON je jezik za pohranu i upite podataka. Sustavi baza podataka mogu se stvoriti grupiranjem ovih dokumenata u zbirke. [28]

Nema potrebe specificirati shemu prije umetanja podataka u ove baze podataka jer funkcioniraju bez sheme što osigurava ogromnu fleksibilnost u pohranjivanju velikog raspona podataka. Ove baze podataka su nekakva vrsta ključ-vrijednost baze podataka. Za svaki ključ, zapis se može spremiti kao vrijednost, a dokumenti su naziv za te zapise. U kontekstu pohrane dokumenata, dokument je datoteka koja sadrži strukturirane podatke. [29]

Dokumentne baze podataka nude brza pretraživanja, strukturu koja je prikladna za obradu velikih količina podataka, fleksibilno indeksiranje i lako održavanje same baze podataka.

Dobro funkcionira za *online* aplikacije i u potpunosti su ga integrirale ogromne IT tvrtke kao što je Amazon. [28]

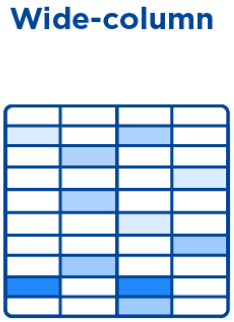


Slika 12. Dokumentna baza podataka

Stupčasta baza podataka je oblik nerelacijske baze podataka koja pohranjuje podatke u stupce. Namijenjene su bržem čitanju podataka i bržem odgovaraju na upite. Izgled baze podataka temeljen na stupcima također smanjuje troškove pohrane podataka dok istovremeno povećava performanse upita. [30]

Obitelji stupaca (eng. *Column Families*) su kategorije u koje se odvajaju stupci u stupčastoj bazi podataka. Svaka obitelj stupaca ima skupinu konceptualno poveznih stupaca koji se često mijenjaju ili dohvaćaju u istu vrijeme. Druge informacije koje su dostupne neovisno mogu se čuvati u nekoliko obitelji stupaca. Novi stupci mogu se dinamički dodavati u obitelj stupaca, a redak ne mora uključivati vrijednosti za sve stupce. [30]

Stupčaste baze podataka prevladavaju ograničenja postojećih relacijskih baza podataka čineći ih bazom podataka poslovne inteligencije (eng. *business intelligence*) budućnosti. Čak i ako baza sadrži milijune zapisa, ova baza podataka daje pristup najrelevantnijim elementima pri tom ubrzavajući upite. Također, olakšava ukupnu analizu podataka. [31]



Slika 13. Stupčasta baza podataka

3.3. BASE svojstva

Nerelacijske baze podataka koriste paradigmu nazvanu BASE model (eng. *Basically Available*, *Soft state*, *Eventual consistency*). Ova arhitektura može rukovati i upravljati nestrukturiranim podacima uz fleksibilnost koju pružaju NoSQL i srodne tehnike. [32]

Tri koncepta čine BASE:

1. Osnovna dostupnost (eng. *basic availability*). Pristup nerelacijskoj bazi podataka fokusiran je na dostupnost podataka i u slučaju višestrukih kvarova. NoSQL baze podataka distribuiraju podatke kroz nekoliko sustava za pohranu s visokim stupnjem replikacije, umjesto da drže jednu veliku pohranu podataka i koncentriraju se na toleranciju grešaka te pohrane. Ako, u neobičnom slučaju, problem spriječi korisnike u pristupu određenom dijelu podataka, to ne znači nužno da će cijela baza podataka pasti. [32]
2. Meko stanje (eng. *soft state*). Ograničenja dosljednosti ACID paradigme uvelike su napuštena u BASE bazama podataka. Konzistentnost podataka odgovornost je programera i njome ne bi trebala upravljati baza podataka, prema jednoj od temeljnih ideja BASE-a. [32]
3. Eventualna dosljednost (eng. *eventual consistency*). Nerelacijske baze podataka samo zahtijevaju da podaci konvergiraju u dosljedno stanje u nekom trenutku u budućnosti kao jedini kriterij dosljednosti. Međutim, nema obećanja kada će se to dogoditi. To je potpuno odstupanje od ACID-ovog zahtjeva trenutne dosljednosti,

koji zabranjuje izvršenje transakcije dok prethodna ne završi i baza podataka ne postigne dosljedno stanje. [32]

3.4. CAP teorem

CAP teorem navodi da se u bilo kojem masovno distribuiranom sustavu upravljanja podacima, mogu biti zajamčena samo dva od tri svojstva – dosljednost, dostupnost i tolerancija particije. [29]

1. Dosljednost (eng. *Consistency*, *C*) – bez obzira na koji čvor se klijent povezuje, dosljednost znači da svi klijenti gledaju istovremeno iste podatke te kada se podaci zapisuju na čvor, moraju se odmah proslijediti ili replicirati na svaki drugi čvor u sustavu prije nego što upisivanje bude „uspješno“. [32]
2. Dostupnost (eng. *Availability*, *A*) – kada klijent traži podatke, dostupnost osigurava da će zahtjev biti ispunjen čak i ako jedan ili više čvorova nije dostupno, odnosno da svaki radni čvor vraća valjani odgovor na odgovarajući zahtjev. [32]
3. Tolerancija particije (eng. *Partition tolerance*, *P*) – kod distribuiranih sustava, particija je komunikacijski prekid uzrokovan izgubljenom ili trenutno odgođenom vezom između dva čvora. Tolerancija particije se odnosi na zahtjev da klaster funkcionira bez obzira na broj neuspješnih komunikacijskih veza između čvorova. [32]

Eric Brewer, sa sveučilišta u Kaliforniji, iznio je hipotezu 2000. godine da tri svojstva dosljednost, dostupnost i tolerancija particija ne mogu istovremeno postojati u masivnom distribuiranom sustavu. Istraživači sa MIT-a (Massachusetts Institute of Technology) kasnije su dokazali ovu hipotezu. [29]

Veliki distribuirani sustavi mogu samo kombinirati dosljednost i dostupnost (*CA*), dosljednost i toleranciju particije (*CP*) ili dostupnost i toleranciju particije (*AP*). Sva tri sustava ne mogu biti istovremeno prisutna.

Slika 14. Tri moguće kombinacije prema CAP teoremu

CA (*Consistency* and *Availability*) baza podataka nudi dostupnost i dosljednost na svim čvorovima. Međutim, to ne može učiniti ako postoji particija između bilo koja dva čvora sustava pa stoga nije u mogućnosti pružiti toleranciju grešaka. [32]

CP (*Consistency* and *Partition tolerance*) baza podataka odriče se dostupnosti kako bi osigurala dosljednost i toleranciju particija. Sustav mora učiniti čvor nedostupnim kada se particija razvije između bilo koja dva čvora dok se particija ne popravi. [32]

AP (*Availability* and *Partition tolerance)* baza podataka odbacuje dosljednost kako bi osigurala dostupnost i toleranciju particija. Svi čvorovi su još uvijek dostupni kada se particija dogodi, no oni na pogrešnom kraju particije mogu pružiti starije podatke od drugih. Ova baza podataka obično ponovno sinkronizira čvorove kako bi popravili sva odstupanja u sustavu kada se particija riješi. [32]

3.5. Sustavi za upravljanjem nerelacijskim baza podataka

MongoDB je NoSQL sustav za upravljanje bazom podataka koji je besplatan i otvorenog koda. NoSQL baze podataka zamjenjuju tradicionalne relacijske baze podataka. MongoDB je sustav koji može raditi s velikim skupovima podataka, pohraniti i dohvatiti podatke orijentirane na dokumente. Također, može i rukovati širokim rasponom tipova podataka. [33]

Svaka baza podataka u MongoDB sustavu se sastoji od zbirki, koje se sastoje od dokumenta. Svaki dokument u zbirci je jedinstven, s različitom količinom polja. Veličina i sadržaj svakog dokumenta mogu se razlikovati. Struktura tih dokumenata sličnija je načinu na koji

programeri generiraju klase i objekte u svojim programima. Isto tako, dokumenti ne moraju imati unaprijed određenu shemu, nego se polja mogu definirati u hodu. [33]

Shema MongoDB baze podataka olakšava predstavljanje hijerarhijskih veza, pohranjivanje nizova i pohranjivanje drugih kompliciranih struktura. [33]

Što se tiče skalabilnosti, klastere su definirale tvrtke diljem svijeta, a neki pokreću 100 ili više čvorova i milijune dokumenata u bazi podataka, što ukazuje na to da su MongoDB okruženja nevjerojatno skalabilna. [33]

Slika 15. MongoDB logo

Apache Cassandra je distribuirani sustav upravljanja nerelacijskim bazama podataka dizajniran tako da rukuje ogromnim količinama podataka. [34]

Jedna od glavnih prednosti Cassandra sustava je ta što pruža visokodostupnu uslugu bez jedne točke kvara. To je ključno za tvrtke koje si ne mogu priuštiti gubitak podataka ili da im se sustav pokvari. Budući da ne postoji jedinstvena točka kvara, pruža potpuno dosljedan pristup i dostupnost. [34]

Još jedna važna značajka Cassandre je njezina sposobnost upravljanja velikim količinama podataka na nekoliko poslužitelja. Također, može i brzo pisati velike količine podataka bez utjecaja na brzinu ili ispravnost. Za velike količine podataka jednako je brz i precizan kao i za manje količine. [34]

Cassandrina skalabilnost je još jedan od razloga zašto je toliko tvrtki koristi. Njen dizajn korisnicima omogućuje da odgovore na nagle skokove potražnje dopuštajući im da jednostavno dodaju još jedan hardver kako bi primili više klijenata i podataka. To omogućuje jednostavno skaliranje bez gašenja ili bez velikih izmjena. [34]

Slika 16. Apache Cassandra logo

Hbase je nerelacijska distribuirana baza podataka otvorenog koda temeljena na *Javi*. Omogućuje pohranjivanje golemih količina rijetkih podataka na način otporan na greške. Također, prikladna je za operacije čitanja i pisanja velike propusnosti na ogromnim skupovima podataka s malim ulazno/izlaznim kašnjenjem. [35]

Format podataka Hbase-a je sličan Google-ovoj velikoj tablici koja korisnicima omogućuje pristup velikim količinama podataka nasumično. Koristi prednost tolerancije grešaka Hadoop datotečnog sustava. [36]

Hbase sustav izgrađen je za linearno skaliranje. Sličan je običnoj bazi podataka po tome što se sastoji od skupa standardnih tablica s recima i stupcima. Primarni ključ mora biti povezan na svaku tablicu, isto tako za pristup nekoj od tablica moramo koristiti taj primarni ključ. [37]

Svaki redak može biti zapis, a stupac može biti vremenska oznaka kada je zapis napisan ili naziv poslužitelja s kojeg je izvješće nastalo. Mnogi se atributi mogu kombinirati zajedno kao obitelj stupaca, pri čemu se elementi obitelji stupaca pohranjuju zajedno. Shema tablice i obitelji stupaca moraju biti unaprijed definirani. Novi stupci mogu se dodati obiteljima u bilo kojem trenutku što čini shemu prilagodljivom promjenama zahtjeva aplikacije. [37]

Slika 17. Apache Hbase logo

4. Razvoj aplikacije

Aplikacija pomoću koje smo testirali vrijeme izvršavanja za relacijsku i nerelacijsku bazu podataka izrađena je pomoću *Node.js* okruženja, biblioteke *express* koja proširuje funkcionalnost *Node.js*-a i ejs (eng. *Embedded JavaScript*) programskog jezika za prikaz podataka. Sama aplikacija služi za evidenciju kolegija koje drže pojedini profesori. Sastoji se od prijave pomoću korisničkog imena i lozinke te prikaza statusa odabranog kolegija za tog prijavljenog profesora. Važno je za napomenuti da su obje baze testiranje na istoj aplikaciji.

Za izvršavanje web aplikacija izvan preglednika klijenta, Node.js je besplatna, višeplatformska *runtime* okolina. Koristi se za programiranje na strani poslužitelja i općenito se koristi za neblokirajuće poslužitelje vođene događajima kao što su web-stranice i pozadinske API usluge. [38]

Ključno je za shvatiti da je Node.js okruženje za izvršavanje JavaScripta, a ne okvir ili biblioteka.

API (eng. *Application Programming Interface*) je sučelje za programiranje aplikacije gdje dvije softverske aplikacije mogu komunicirati jedna s drugom. API određuje kako razvojni programer treba tražiti usluge od operativnog sustava ili drugog programa i kako pružiti podatke u različitim situacijama. Pomoću API-ja mogu se razmjenjivati bilo koji podaci. [39]

Dvije povezane komponente čine API:

1. Standard koji opisuje kako se podaci šalju između programa kao zahtjevi za obradu i vraćanje potrebnih informacija
2. Softversko sučelje izrađeno u skladu s tom specifikacijom i javno dostupno za korištenje [39]

Express je pozadinski okvir Node.js-a koji je brz i jednostavan. Nudi moćne mogućnosti i alate za stvaranje skalabilnih pozadinskih aplikacija. Za razvoj i implementaciju masovnih aplikacija spremnih za poduzeće, okvir nudi paket alata za web aplikacije. Možemo stvarati aplikacije, API krajnje točke, sustave usmjeravanja i okvire. [40]

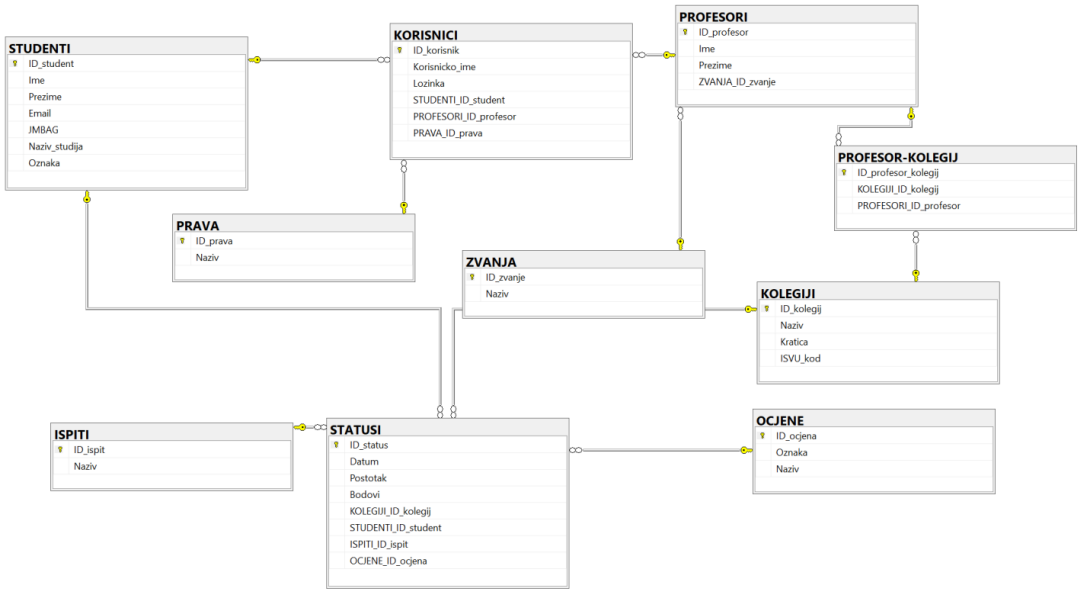
Dakle, pomoću *express*-a ćemo spojiti bazu podataka s našom aplikacijom.

4.1. Baze podataka

Za relacijsku bazu podataka koristimo bazu podataka koju je stvorio Microsoft poznatu kao SQL Server. Kao što smo već rekli, to je softverski proizvod poznat kao i poslužitelj baze podataka, a njegov glavni posao je pohranjivanje i dohvaćanje podataka za druge softverske programe koji mogu raditi na istom računalu ili drugom računalu preko mreže. Transact SQL (T-SQL) je glavni jezik za upite relacijskih baza podataka, što znači da osim jednostavnih SQL upita podržava i kompliciranije zadatke.

Relacijska baza podataka se sastoji od sljedećih entiteta: Student, Kolegij, Status, Prava, Ispit, Ocjena, Korisnik, Profesor i Zvanje.

Na slici 22. vidljiv je dijagram baze podataka u SQL Serveru s entitetima, njihovim atributima te vezama među njima.



Slika 18. Dijagram baze podataka u SQL Serveru

U SQL Serveru možemo pisati pohraniti procedure koje su nam potrebne za aplikaciju. Pohranjena procedura je SQL upit spreman za izvršavanje koji je unaprijed pripremljen i pohranjen u bazi podataka. Pomoću tih pohranjenih procedura ostvarili smo funkcionalnost aplikacije, a one koje su nam pomogle testirati brzinu izvršavanja su sljedeće:

1. *getStatusiByKolegij* – na odabir kolegija iz *select* elementa prikazuje status tog odabranog kolegija

```
ALTER PROCEDURE [dbo].[getStatusiByKolegij]
    @id_kolegij int
AS
BEGIN
    SET NOCOUNT ON;
SELECT std.ID_student, std.Ime, std.Prezime, s.ID_status,
CONVERT(varchar, s.Datum, 104) AS Datum, s.Postotak, s.Bodovi,
o.ID_ocjena, o.Oznaka, i.ID_ispit, i.Naziv
FROM ISPITI i
INNER JOIN STATUSI s ON i.ID_ispit = s.ISPITI_ID_ispit
INNER JOIN OCJENE o ON s.OCJENE_ID_ocjena = o.ID_ocjena
INNER JOIN STUDENTI std ON s.STUDENTI_ID_student = std.ID_student
WHERE KOLEGIJI_ID_kolegij=@id_kolegij
FOR JSON PATH, ROOT('statusKolegija')
END
```

Slika 19. Procedura getStatusiByKolegij

Također, u ovoj proceduri koristimo *FOR JSON* izraz. Kada koristimo *FOR JSON* izraz, možemo eksplicitno definirati izlaznu strukturu JSON-a ili dopustiti da struktura izjave *select* odluči o izlazu. Da bi zadržali potpunu kontrolu nad izlaznim formatom, koristimo *FOR JSON PATH*. [41]

U ovoj proceduri smo JSON rezultatu dodijelili alias *statusKolegija* kojemu ćemo u aplikaciji pristupiti za prikazati željene podatke.

Sljedeća procedura koju smo napravili je *getStatusiByKolegij* koja na odabir kolegija iz *select* elementa prikazuje status tog odabranog kolegija.

2. *getInfoStudenta* – prikaz osnovnih informacija o studentu

```
ALTER PROCEDURE [dbo].[getInfoStudenta]
    @idStudent int
AS
BEGIN
    SET NOCOUNT ON;
   SELECT  STUDENTI.ID_student, STUDENTI.Ime,
   STUDENTI.Prezime, STUDENTI.Email, STUDENTI.JMBAG,
   STUDENTI.Naziv_studija
    FROM STUDENTI
    WHERE STUDENTI.ID_student=@idStudent
    FOR JSON PATH, ROOT('infoStudenta')
END
```

Slika 20. Procedura getInfoStudenta

3. *updateOcjena* – ažurira ocjenu studenta u prikazanoj tablici

```
ALTER PROCEDURE [dbo].[updateOcjena]

    @id_status int,
    @ocjena int
AS
BEGIN

    SET NOCOUNT ON;

    UPDATE STATUSI
    SET  OCJENE_ID_ocjena= @ocjena
    WHERE ID_status = @id_status
END
```

Slika 21. Procedura updateOcjena

4. *deleteStatus* – briše redak iz tablice Statusi

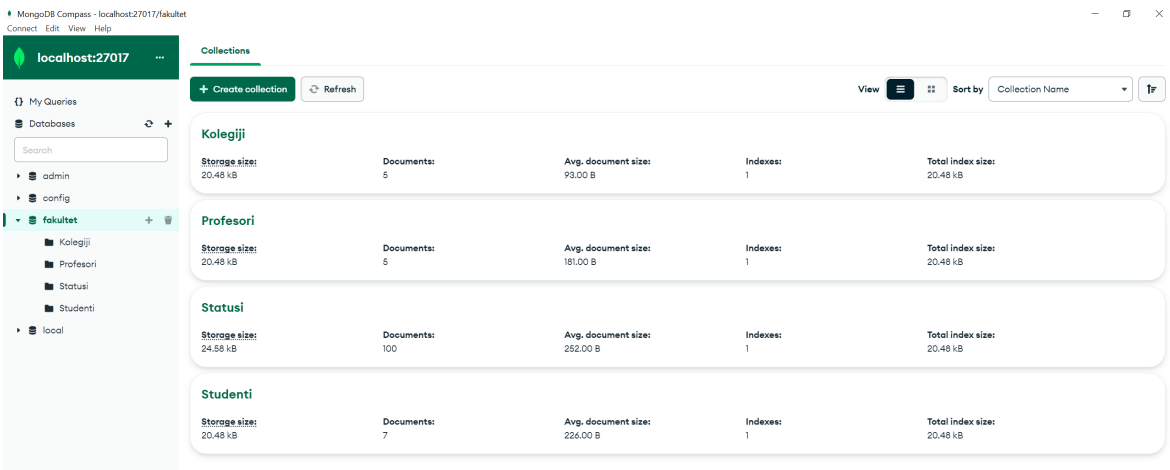
```
ALTER PROCEDURE [dbo].[deleteStatus]
    @id_status int
AS
BEGIN
    SET NOCOUNT ON;
    DELETE from STATUSI
    WHERE ID_status = @id_status
END
```

Slika 22. Procedura deleteStatus

Za nerelacijsku bazu podataka koristimo MongoDB koji je dizajniran za spremanje, upravljanje i brzu obradu velike količine nestrukturiranih ili polustrukturiranih podataka. Već smo spomenuli da je njegova ključna značajka fleksibilnosti i sposobnost rada s podacima koji se ne uklapaju u tradicionalne relacijske baze podataka.

Nerelacijska baza podataka sastoji se od sljedećih kolekcija: Kolegiji, Profesori, Statusi i Studenti.

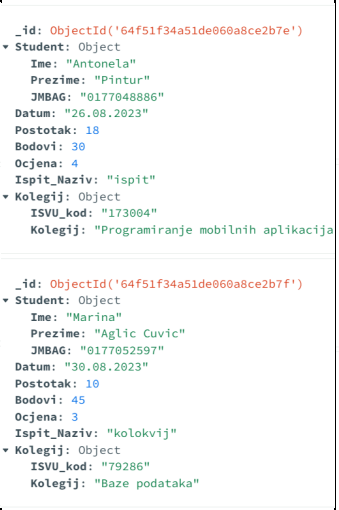
Na slici 27. možemo vidjeti kako izgledaju te kolekcije u MongoDB.



Slika 23. Nerelacijska baza podataka

U tablici 2 možemo vidjeti kako izgleda svaka od kolekcija.

Tablica 2. Kolekcije

Kolekcija Kolegiji	Kolekcija Profesori	Kolekcija statusi	Kolekcija Studenti
			

4.2. Aplikacijski sloj

Pohranjene procedure koje smo definirali u MS SQL bazi podataka pozivamo uz pomoć *express* aplikacije te različitih rute i metoda *app.get*, *app.post* i *app.delete* koje ona pruža.

Za prikaz podataka na temelju odabranog kolegija koristimo *post* metodu i pohranjenu proceduru *getStatusiByKolegij*. Rezultat koji daje procedura spremamo u niz status i šaljemo ga za prikaz u *data.ejs*.

```
app.post('/status', function(req, res){
    const idKolegij = req.body.cmb;
    if(!isNaN(idKolegij)){
        const startTime = performance.now();
        const request = new sql.Request();
        request.input("id_kolegij", sql.Int(), idKolegij)
        .execute("getStatusiByKolegij").then(function(result){
            const endTime = performance.now();
            const executionTime = endTime - startTime;
            console.log("Stored procedure execution time (getStatusi): " + executionTime + " ms");
            if (result.recordset.length > 0) {
                const data = JSON.parse(result.recordset[0][JSON_COLUMN_KEY]);
                status = data.statusKolegija;
                res.redirect("/data");
            }
        })
    }else{
        res.redirect("/data");
    }
});
```

Slika 24. Pozivanje procedure getStatusiByKolegij

Varijabla JSON_COLUMN_KEY je *environment* varijabla u koju je spremljen naziv stupca generiranog unutar MS SQL-a. Taj stupac sadrži JSON podatke te uz pomoć vrijednosti dohvaćamo *statusKolegija* koji nama treba.

Za dobiti osnovne informacije o pojedinom studentu koristimo metodu *get* i pohranjenu proceduru *getInfoStudenta* koju pozivamo iz baze podataka.

```
app.get('/info', function (req, res) {
    const idStudent = req.query.id;
    const startTime = performance.now();
    const request = new sql.Request();
    request.input("idStudent", sql.Int(), idStudent)
        .execute("getInfoStudenta")
        .then(function (result) {
            const endTime = performance.now();
            const executionTime = endTime - startTime;
            console.log("Stored procedure execution time (getInfo): " + executionTime + " ms");
            if (result.recordset.length > 0) {
                const data = JSON.parse(result.recordset[0][JSON_COLUMN_KEY]);
                infoStudenta = data.infoStudenta;
                res.send(infoStudenta);
            } else {
                res.send({ message: "Student not found." });
            }
        })
        .catch(error => {
            console.error(error);
            res.send({ error: "An error occurred" });
        });
});
```

Slika 25. Poziv procedure getInfoStudenta

Sljedeća metoda koju koristimo je opet *post* metoda pomoću koje ćemo ažurirati ocjenu.

```
app.post('/updateOcjena/:idStatus', function (req, res) {
    const idStatus = req.params.idStatus;
    const ocjena = req.body.ocjena;
    const startTime = performance.now();
    const request = new sql.Request();
    request.input("id_status", sql.Int(), idStatus)
    request.input("ocjena", sql.Int(), ocjena)
        .execute("updateOcjena")
        .then(function (result) {
            const endTime = performance.now();
            const executionTime = endTime - startTime;
            console.log("Stored procedure execution time (updateOcjena): " + executionTime + " ms");
            res.sendStatus(200);
        })
        .catch(error => {
            console.log(error);
            res.sendStatus(500);
        })
});
```

Slika 26. Pozivanje procedure updateOcjena

Zadnja metoda koju koristimo je *delete* metoda s kojom ćemo postići mogućnost brisanja retka tablice.

```
app.delete('/deleteStatus/:idStatus', function (req, res) {
    const idStatus = req.params.idStatus;
    const startTime = performance.now();
    const request = new sql.Request();
    request.input("id_status", sql.Int(), idStatus)
        .execute("deleteStatus")
        .then(function (result) {
            const endTime = performance.now();
            const executionTime = endTime - startTime;
            console.log("Stored procedure execution time (deleteStatus): " + executionTime + " ms");
            res.sendStatus(200);
        })
        .catch(error => {
            console.log(error);
            res.sendStatus(500);
        })
});
```

Slika 27. Pozivanje procedure deleteStatus

U nerelacijskoj bazi podataka, da bi mogli dohvaćati, uređivati i brisati podatke, trebamo definirati njihove modele u našoj aplikaciji. Tako smo definirali dva modela da bi mogli testirati brzinu baze: statusiModel i studentiModel.

Definiranje tog modela omogućuje strukturu podataka koja će se dohvaćati, uređivati ili spremati u MongoDB kolekciji, a izgleda ovako:

```
const mongoose = require('mongoose');

const kolegijSchema = new mongoose.Schema({
  ISVU_kod: String,
  Kolegij: String
});

const studentSchema = new mongoose.Schema({
  Ime: String,
  Prezime: String,
  JMBAG: String
});

const statusSchema = new mongoose.Schema({
  Student: studentSchema,
  Datum: String,
  Postotak: Number,
  Bodovi: Number,
  Ocjena: Number,
  Ispit_Naziv: String,
  Kolegij: kolegijSchema
});

const Status = mongoose.model('Status', statusSchema, 'Statusi');

module.exports = Status;
```

Slika 28. Model status kolekcije

```
const mongoose = require('mongoose');

const studentSchema = new mongoose.Schema({
  Ime: String,
  Prezime: String,
  Email: String,
  JMBAG: String,
  Korisnicko_ime: String,
  Lozinka: String,
  Naziv_studija: String,
  Oznaka: String,
  Prava: String
});

const Student = mongoose.model('Student', studentSchema, 'Studenti');

module.exports = Student;
```

Slika 29. Model student kolekcije

Poslužit ćemo se ugrađenim *express* metodama *get*, *put* i *delete* za rad nad podacima.

Prvo koristimo *get* metodu da bi dohvatili kolegij za koji želimo prikazati status. Zatim s ugrađenom funkcijom *find()* izrađujemo upit kako bi pronašli taj dokument koji zadovoljava te kriterije. Tražimo dokumente u kolekciji Status gdje se vrijednost kolegija podudara.

```
app.get("/data", async (req, res) => {
  if (req.session.user) {
    try {
      const selectedKolegij = req.query.cmb;
      const startTime = performance.now();
      const statusData = await Status.find({"Kolegij.Kolegij": selectedKolegij});
      const endTime = performance.now();
      const executionTime = endTime - startTime;
      console.log("Stored procedure execution time (getStatusi): " + executionTime + " ms");
      res.render("data.ejs", {
        username: req.session.user.username,
        kolegijiNazivi: req.session.user.kolegijiNazivi,
        statusData: statusData,
        selectedKolegij: selectedKolegij
      });
    } catch (err) {
      console.error(err);
      res.status(500).json({ error: "Internal server error" });
    }
  } else {
    res.redirect("/login");
  }
});
```

Slika 30. Upit za dohvat statusa

Idući upit koji radimo je za dohvati osnovnih informacija o pojedinom studentu i to također radimo uz pomoć *get* metode. Informacije dohvaćamo preko JMBAG-a i funkcije *findOne()* kako bi pronašli prvi dokument koji zadovoljava kriterije.

```
app.get("/getStudentInfo", async (req, res) => {
  const jmbag = req.query.jmbag;
  try {
    const startTime = performance.now();
    const studentInfo = await Student.findOne({ JMBAG: jmbag });
    const endTime = performance.now();
    const executionTime = endTime - startTime;
    console.log("Stored procedure execution time (getInfo): " + executionTime + " ms");
    res.json(studentInfo);
  } catch (err) {
    console.error('Error fetching student info:', err);
    res.status(500).json({ error: "Internal server error" });
  }
});
```

Slika 31. Upit za dohvat informacija o pojedinom studentu

S obzirom da i u ovoj bazi želimo mogućnost ažuriranja podataka, to ćemo postići s metodom *put*. Taj upit pišemo pomoću ugrađene metode *findById()* i *save()* gdje prvo pronađemo dokument u kolekciji s određenim id-om i nakon toga pridružujemo mu novu vrijednost atributa ocjena te to spremamo s metodom *save()*.

```
app.put('/updateOcjena/:id', async (req, res) => {
  const statusId = req.params.id;
  const { ocjena } = req.body;

  try {
    const startTime = performance.now();
    const status = await Status.findById(statusId);

    if (!status) {
      return res.json({ success: false, error: 'Status not found.' });
    }
    status.Ocjena = ocjena;
    await status.save();
    const endTime = performance.now();
    const executionTime = endTime - startTime;
    console.log("Stored procedure execution time (updateOcjena): " + executionTime + " ms");
    res.json({ success: true, updatedStatus: status });

  } catch (error) {
    console.error('Error while updating:', error);
    res.json({ success: false, error: 'Error while updating.' });
  }
});
```

Slika 32. Upit za ažuriranje ocjene

Zadnji upit koji radimo je za brisanje dokumenta u kolekciji s metodom *delete*. Pronalazimo dokument koji želimo izbrisati pomoću id-a i pozivamo metodu *findOneAndRemove()* za brisanje.

```
app.delete("/deleteStatus/:id", async (req, res) => {
  const statusId = req.params.id;
  try {
    const startTime = performance.now();
    await Status.findOneAndRemove(statusId);
    const endTime = performance.now();
    const executionTime = endTime - startTime;
    console.log("Stored procedure execution time (deleteStatus): " + executionTime + " ms");
    res.json({success: true})
  } catch(err) {
    console.log("Error deleting row: ", err);
    res.status(500).json({error: "Internal server error"});
  }
});
```

Slika 33. Upit za brisanje

5. Usporedba relacijske i nerelacijske baze podataka

Svi testovi performansi provedeni su na osobnom računalu sa specifikacijama koje uključuju:

- Naziv uređaja: DESKTOP-63FIKMC
- Procesor: Intel(R) Core(TM) i5-5287U CPU @ 2.90GHz 2.90 GHz
- RAM: 8,00 GB
- ID uređaja: 46482072-166D-42C7-923C-44DD929D032E
- ID proizvoda: 00330-80130-83823-AA310
- Tip sustava: 64-bit operating system, x64-based processor
- Windows 10 Pro

Test koristi sljedeće baze podataka:

1. Microsoft SQL Server
2. MongoDB

Svaka baza podataka testirana je na četiri skupa podataka: 100, 1000, 100 000 i 1 000 000.

Testiramo sljedeće procedure:

1. Dohvaćanje podataka
2. Uređivanje podataka
3. Brisanje podataka

5.1. Usporedba performansi

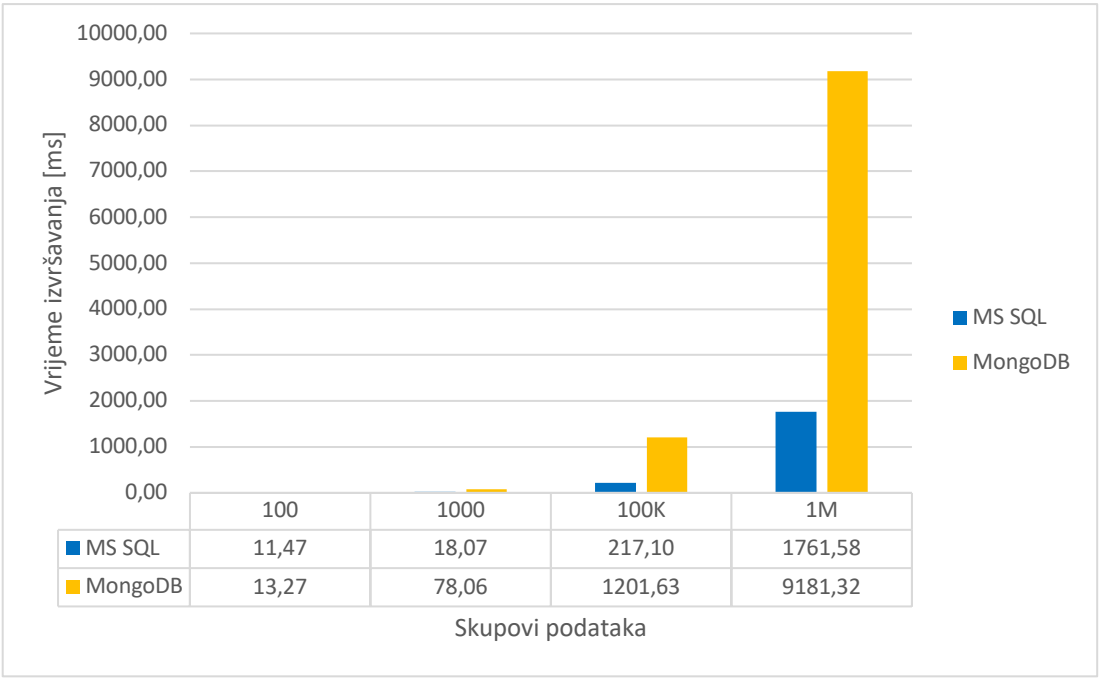
Prvo smo testirali vrijeme izvršavanja MS SQL baze podataka i MongoDB baze podataka na proceduri koja dohvaća podatke iz tablice, odnosno kolekcije Statusi. U tablici 3 su prikazani su upiti za dohvaćanje podataka za svaku od baza, a rezultati su prikazani na slici 34.

Tablica 3. Upiti za dohvaćanje podataka

MS SQL	MongoDB

`ALTER PROCEDURE [dbo].[getStatusiByKolegij]`
`  @id_kolegij int`
`AS`
`BEGIN`
`  SET NOCOUNT ON;`
`SELECT std.ID_student, std.Ime, std.Prezime, s.ID_status,`
`CONVERT(varchar, s.Datum, 104) AS Datum, s.Postotak, s.Bodovi,`
`o.ID_ocjena, o.Oznaka, i.ID_ispit, i.Naziv`
`FROM ISPITI i`
`INNER JOIN STATUSI s ON i.ID_ispit = s.ISPITI_ID_ispit`
`INNER JOIN OCJENE o ON s.OCJENE_ID_ocjena = o.ID_ocjena`
`INNER JOIN STUDENTI std ON s.STUDENTI_ID_student = std.ID_student`
`WHERE KOLEGIJI_ID_kolegij=@id_kolegij`
`FOR JSON PATH, ROOT('statusKolegija')`
`END`
	`const selectedKolegij = req.query.cmb;`
	`const statusData = await Status.find({"Kolegij.Kolegij": selectedKolegij});`

`app.post('/status', function (req, res) {`
`  const idKolegij = req.body.cmb;`
`  if (!isNaN(idKolegij)) {`
`  const request = new sql.Request();`
`  request.input("id_kolegij", sql.Int(), idKolegij)`
`  .execute("getStatusiByKolegij").then(function (result) {`
`  if (result.recordset.length > 0) {`
`  const data = JSON.parse(result.recordset[0][JSON_COLUMN_KEY]);`
`  status = data.statusKolegija;`
`  res.redirect("/data");`
`  }`
`  })`
`  } else {`
`  res.redirect("/data");`
`  }`
`});`
	`app.get("/data", async (req, res) => {`
	`  if (req.session.user) {`
	`  try {`
	`  const selectedKolegij = req.query.cmb;`
	`  const statusData = await Status.find({"Kolegij.Kolegij": selectedKolegij});`
	`  res.render("data.ejs", {`
	`  username: req.session.user.username,`
	`  kolegijiNazivi: req.session.user.kolegijiNazivi,`
	`  statusData: statusData,`
	`  selectedKolegij: selectedKolegij`
	`  });`
	`  } catch (err) {`
	`  console.error(err);`
	`  res.status(500).json({ error: "Internal server error" });`
	`  }`
	`  } else {`
	`  res.redirect("/login");`
	`  }`
	`});`



Slika 34. Rezultati dohvaćanja podataka

Što se tiče vremena izvršavanja za skup podataka od 100 možemo vidjeti su vremena jako slična, međutim već kod skupa podataka od 1000 primjećujemo da je dohvaćanje podataka

kod MS SQL baze podataka brže nego kod MongoDB. Što više povećavamo skup podataka, MS SQL baza i dalje puno brže dohvaća podatke nego MongoDB.

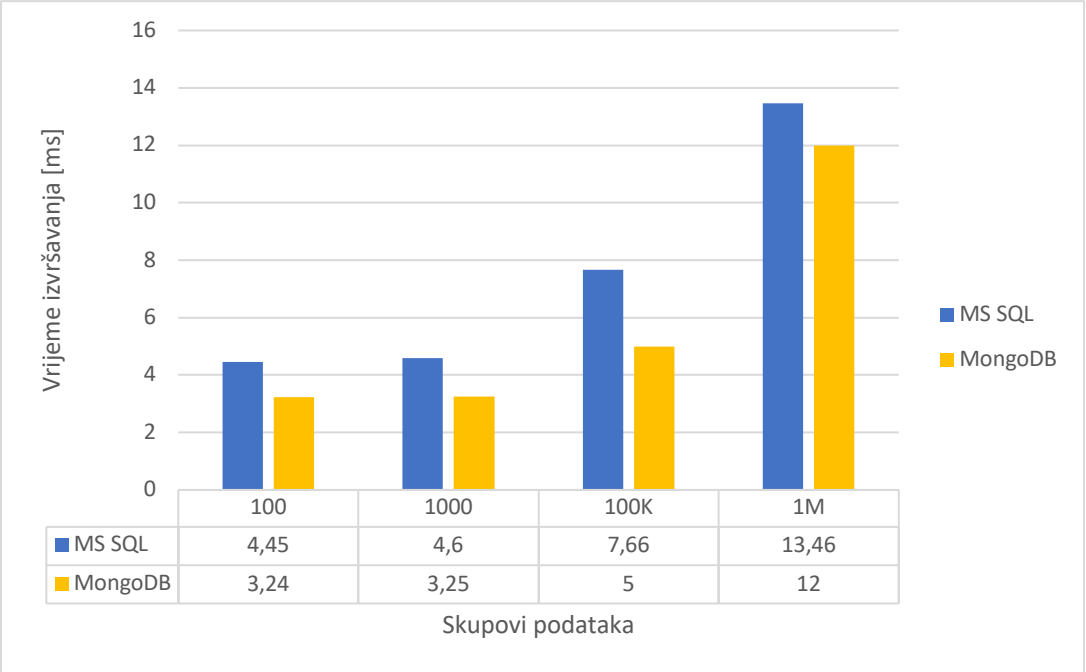
Sljedeća procedura za koju smo testirali vrijeme izvršavanja je dohvaćanje informacija studenata iz baze podataka. (Slika 35.). Upiti su prikazani u tablici 4.

Tablica 4. Upiti za dohvaćanje pojedinog studenta

MS SQL	MongoDB

`ALTER PROCEDURE [dbo].[getInfoStudenta]`
`  @idStudent int`
`AS`
`BEGIN`
`  SET NOCOUNT ON;`
`  SELECT  STUDENTI.ID_student, STUDENTI.Ime,`
`  STUDENTI.Prezime, STUDENTI.Email, STUDENTI.JMBAG,`
`  STUDENTI.Naziv_studija`
`  FROM STUDENTI`
`  WHERE STUDENTI.ID_student=@idStudent`
`  FOR JSON PATH, ROOT('infoStudenta')`
`END`
	`const jmbag = req.query.jmbag;`
	`const studentInfo = await Student.findOne({ JMBAG: jmbag });`

`app.get('/info', function (req, res) {`
`  const idStudent = req.query.id;`
`  const request = new sql.Request();`
`  request.input("idStudent", sql.Int(), idStudent)`
`  .execute("getInfoStudenta")`
`  .then(function (result) {`
`  if (result.recordset.length > 0) {`
`  const data = JSON.parse(result.recordset[0][JSON_COLUMN_KEY]);`
`  infoStudenta = data.infoStudenta;`
`  res.send(infoStudenta);`
`  } else {`
`  res.send({ message: "Student not found." });`
`  }`
`  })`
`  .catch(error => {`
`  console.error(error);`
`  res.send({ error: "An error occurred" });`
`  });`
`});`
	`app.get("/getStudentInfo", async (req, res) => {`
	`  const jmbag = req.query.jmbag;`
	`  try {`
	`  const studentInfo = await Student.findOne({ JMBAG: jmbag });`
	`  res.json(studentInfo);`
	`  } catch (err) {`
	`  console.error('Error fetching student info:', err);`
	`  res.status(500).json({ error: "Internal server error" });`
	`  }`
	`});`



Slika 35. Rezultat dohvaćanja informacija studenta

Iz rezultata možemo zaključiti da je MongoDB baza malo brža u dohvaćanju informacija za pojedinog studenta u odnosu na MS SQL bazu. Vrijeme izvršavanja se povećava s obzirom na skup podataka, što vrijedi za obje baze.

Iduća procedura na kojoj smo testirali obje baze podataka je *update*. Unutar tablice i kolekcije Statusi smo dodali mogućnost ažuriranja ocjene, a te upite možemo vidjeti u tablici 5. Rezultati izvršavanja su prikazani na slici 36.

Tablica 5. Upiti za ažuriranje podataka

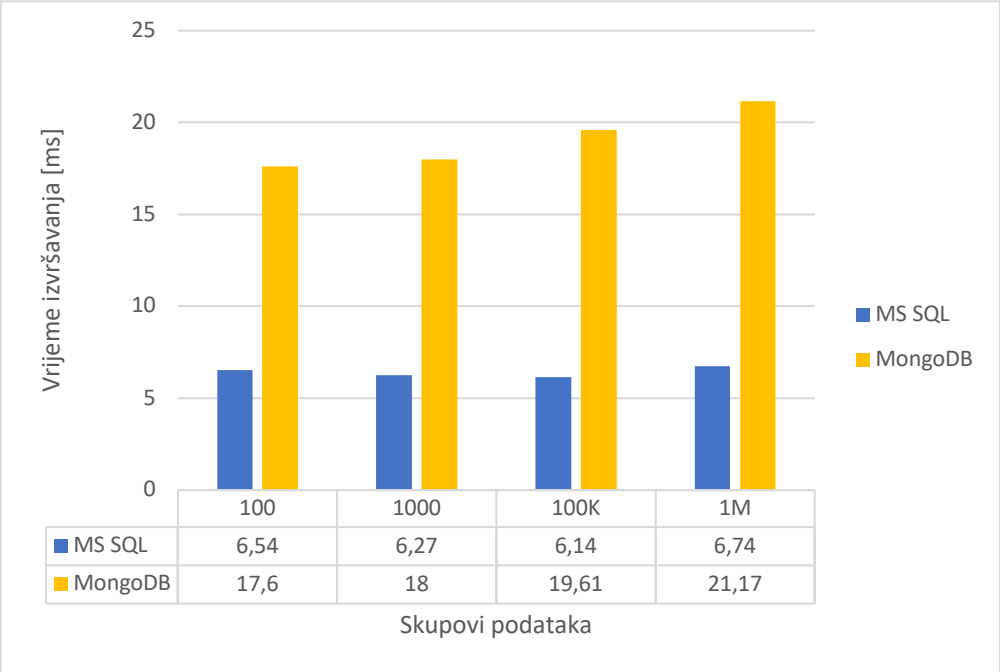
MS SQL	MongoDB

`ALTER PROCEDURE [dbo].[updateOcjena]`
`  @id_status int,`
`  @ocjena int`
`AS`
`BEGIN`
`  SET NOCOUNT ON;`
`  UPDATE STATUSI`
`  SET  OCJENE_ID_ocjena= @ocjena`
`  WHERE ID_status = @id_status`
`END`
	`const statusId = req.params.id;`
	`const { ocjena } = req.body;`
	`const status = await Status.findById(statusId);`
	`status.Ocjena = ocjena;`
	`await status.save();`

```
app.post('/updateOcjena/:idStatus', function (req, res) {
    var idStatus = req.params.idStatus;
    var ocjena = req.body.ocjena;
    var request = new sql.Request();
    request.input("id_status", sql.Int(), idStatus)
    request.input("ocjena", sql.Int(), ocjena)
        .execute("updateOcjena")
        .then(function (result) {
            res.sendStatus(200);
        })
        .catch(error => {
            console.log(error);
            res.sendStatus(500);
        })
})
```

```
app.put('/updateOcjena/:id', async (req, res) => {
  const statusId = req.params.id;
  const { ocjena } = req.body;
  try {
    const status = await Status.findById(statusId);
    if (!status) {
      return res.json({ success: false, error: 'Status not found.' });
    }
    status.Ocjena = ocjena;
    await status.save();
    res.json({ success: true, updatedStatus: status });

  } catch (error) {
    console.error('Error while updating:', error);
    res.json({ success: false, error: 'Error while updating.' });
  }
});
```



Slika 36. Rezultati ažuriranja podatka

Možemo vidjeti da je MS SQL baza podataka brža u ažuriranju podataka bez obzira na skup podataka, vrijeme ostaje jako slično. S druge strane, u MongoDB bazi podataka, vrijeme izvršavanja povećava se s obzirom na skup podataka.

Zadnja procedura na kojoj smo testirali vrijeme izvršavanja baza podataka je brisanje. Brisali smo jedan redak u tablici kod MS SQL baze, i jedan dokument u MongoDB bazi (Slika 37.). U tablici 6 su prikazani upiti za obje baze.

Tablica 6. Upiti za brisanje podataka

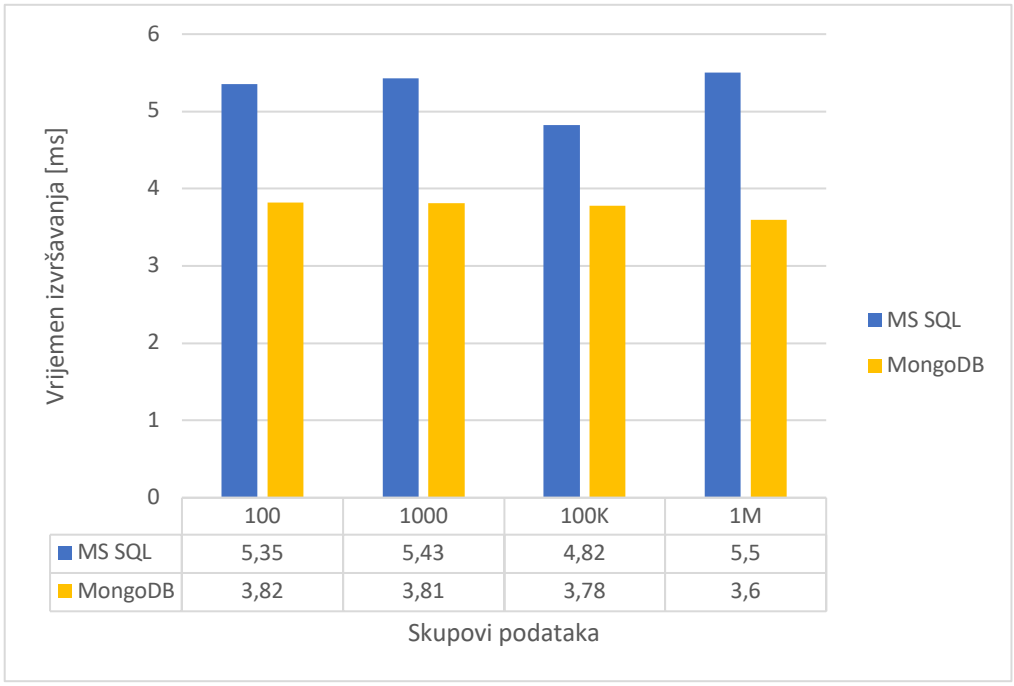
MS SQL	MongoDB

```
ALTER PROCEDURE [dbo].[deleteStatus]
    @id_status int
AS
BEGIN
    SET NOCOUNT ON;
    DELETE from STATUSI
    WHERE ID_status = @id_status
END
```

```
const statusId = req.params.id;
await Status.findOneAndRemove(statusId);
```

```
app.delete('/deleteStatus/:idStatus', function (req, res) {
    var idStatus = req.params.idStatus;
    var request = new sql.Request();
    request.input("id_status", sql.Int(), idStatus)
        .execute("deleteStatus")
        .then(function (result) {
            res.sendStatus(200);
        })
        .catch(error => {
            console.log(error);
            res.sendStatus(500);
        })
});
```

```
app.delete("/deleteStatus/:id", async (req, res) => {
  const statusId = req.params.id;
  try {
    await Status.findOneAndRemove(statusId);
    res.json({success: true})
  } catch(err) {
    console.log("Error deleting row: ", err);
    res.status(500).json({error: "Internal server error"});
  }
});
```



Slika 37. Rezultati izvršavanja brisanja

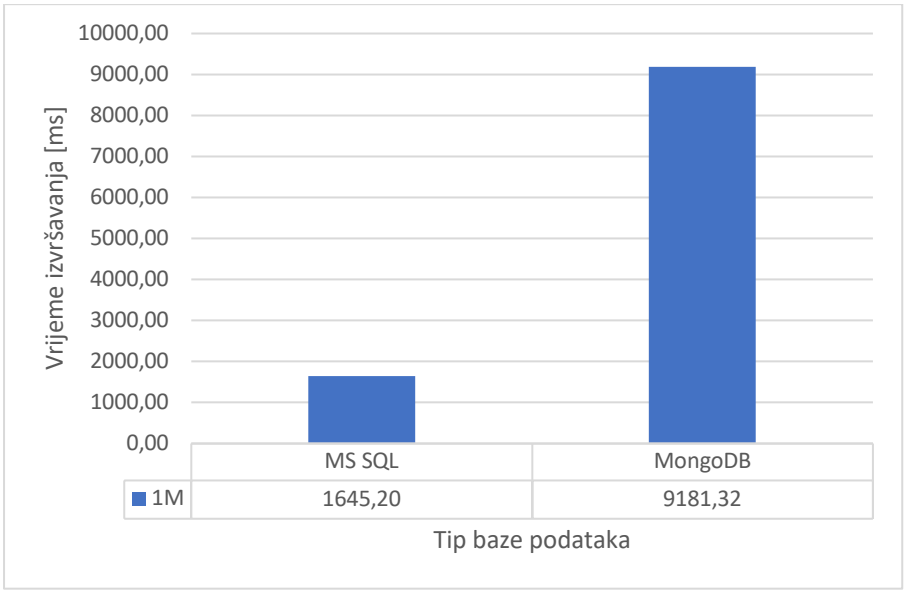
Iz rezultata vidimo da su obje baze brze kada je u pitanju brisanje podataka bez obzira na skup podataka, s tim da je MongoDB baza malo brža.

Nakon analize rezultata izvršavanja, odlučili smo provesti dodatne testove jer smo primijetili da je relacijska baza podataka pružila bolje performanse kod dohvaćanja velikog skupa podataka.

Izvršili smo test u kojem smo umjesto korištenja pohranjene procedure getStatusiByKolegij, pisali upit izravno u aplikaciji da vidimo kakve će nam to rezultate donijeti. Ovaj put smo testirali samo na milijun podataka. Slika 38. prikazuje te rezultate.

```
app.post('/status', function(req, res){
    const idKolegij = req.body.cmb;
    if(!isNaN(idKolegij)){
        const startTime = performance.now();
        const request = new sql.Request();
        request.input("id_kolegij", sql.Int(), idKolegij)
        .query("SELECT std.ID_student, std.Ime, std.Prezime, s.ID_status" +
               "CONVERT(varchar, s.Datum, 104) AS Datum, s.Postotak, s.Bodovi" +
               "o.ID_ocjena, o.Oznaka, i.ID_ispit, i.Naziv " +
               "FROM ISPITI i " +
               "INNER JOIN STATUSI s ON i.ID_ispit = s.ISPITI_ID_ispit " +
               "INNER JOIN OCJENE o ON s.OCJENE_ID_ocjena = o.ID_ocjena " +
               "INNER JOIN STUDENTI std ON s.STUDENTI_ID_student = std.ID_student " +
               "WHERE KOLEGIJI_ID_kolegij=@id_kolegij FOR JSON PATH, ROOT('statusKolegija')")
        .then(function(result){
            const endTime = performance.now();
            const executionTime = endTime - startTime;
            console.log("SQL query execution time: " + executionTime + " ms");
            if (result.recordset.length > 0) {
                const data = JSON.parse(result.recordset[0]['JSON_COLUMN_KEY']);
                status = data.statusKolegija;
                res.redirect("/data");
            }
        })
    } else {
        res.redirect("/data");
    }
});
```

Slika 38. SQL upit unutar aplikacije

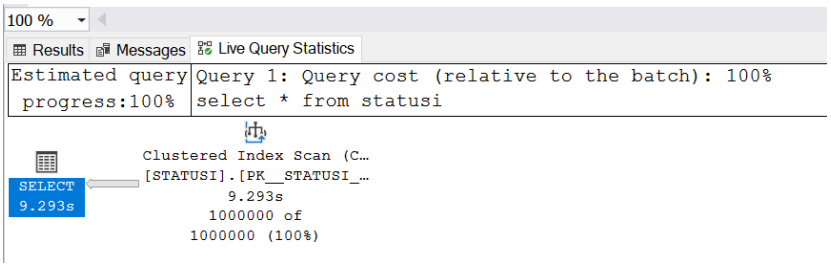


Slika 39. Rezultati izvršavanja pozivanjem upita unutar aplikacije

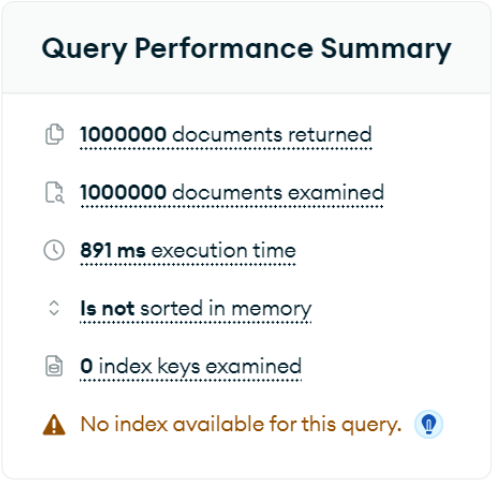
U rezultatima testiranja možemo primijetiti značajnu razliku u izvedbi između dvije baze. SQL baza je pokazala impresivno brže izvršavanje s vremenom izvođenja od 1645,20ms,

dok je MongoDB baza podataka zahtijevala značajno više vremena. Ovi rezultati ukazuju na prednost SQL baze u pogledu brzine izvođenja u ovom specifičnom testiranju.

Dodatno, za testiranje performansi, izvršili smo dohvat svih milijun podataka iz baze unutar samih aplikacija. Ovo testiranje izvedeno je u okruženju Microsoft SQL Servera i MongoDB baze podataka (Slika 40. i 41.).



Slika 40. Dohvat podataka unutar MS SQL



Slika 41. Dohvat podataka unutar MongoDB

Zanimljivo je za primijetiti da su se rezultati značajno promijenili kada smo izvršili testiranje izravno unutar samih aplikacija svake od baza. U prijašnjem testiranju, gdje smo testirali baze unutar same aplikacije, SQL je pokazao bolje performanse u odnosu na MongoDB. Međutim, u ovakvom testiranju MongoDB je ostvario značajno bolje rezultate.

Moguće je da su razlike u izvedbi posljedica različitih uvjeta izvođenja. Također, različiti pristupi modeliranju podataka i indeksiranju, kao i način na koji su optimizirani upiti i indeksi, mogu značajno utjecati na izvedbu baze podataka.

Zaključak

U ovom diplomskom radu smo detaljno istražili koncepte baza podataka i njihovu primjenu. Prvo smo razmotrili osnovne definicije i karakteristike baza podataka istražujući njihovu ključnu ulogu u organizaciji i pohrani podataka.

Dalje smo se fokusirali na relacijske i nerelacijske baze podataka tako što smo analizirali njihove prednosti i nedostatke.

Kroz istraživanje, usporedili smo performanse relacijske i nerelacijske baze podataka u kontekstu različitih operacija nad različitim skupovima podataka. Za relacijsku bazu podataka smo uzeli Microsoft SQL Server, a za nerelacijsku bazu MongoDB. Za početak smo testirali obje baze u dohvaćanju filtriranih podataka gdje su za mali skup podataka imale slične rezultate izvršavanja, međutim kako se skup podataka povećavao, tako je i relacijska baza podataka imala bolje vrijeme izvršavanja. Zatim smo testirali dohvaćanje manjeg broja podataka, odnosno dohvaćanje određenih informacija. Ovdje smo primijetili da je MongoDB brži za manje skupove podataka, dok su oba vremena izvršavanja ostala stabilna s povećanjem skupa podataka. Sljedeća operacija nad kojom smo radili usporedbu bila je operacija ažuriranja gdje smo mogli vidjeti da je MS SQL baza podataka brža kada je u pitanju ažuriranje podataka. Čak je i s povećanjem broja podataka vrijeme izvršavanja ostalo skoro pa isto, dok se za nerelacijsku bazu podataka vrijeme izvršavanja povećavalo kako se povećavao skup podataka. Zadnji dio našeg istraživanja bavio se operacijom brisanja podataka. Rezultati su pokazali da su obje baze podataka ponudile vrlo slične performanse bez obzira na veličinu podataka. Zbog zanimljivih rezultata, dalje smo testirali vrijeme izvođenja kod MS SQL baze podataka. S obzirom da smo koristili pohranjene procedure, testirali smo i kakvo će biti vrijeme izvršavanja za dohvaćanje milijun podataka kroz upit napisan unutar aplikacije. Međutim, dobili smo slične rezultate.

Na kraju smo usporedili performanse unutar samih okruženja za dohvat milijun podataka gdje se MongoDB pokazao znatno bržim od MS SQL-a.

Naše istraživanje pokazuje da se odluka između MS SQL i MongoDB baze podataka oslanja na posebne zahtjeve aplikacija. Dok bi MS SQL mogao biti bolja opcija za kompliciranije upite, MongoDB je bolji u brzom upravljanju velikim količinama podataka. Prilikom odabira baze podataka, ključno je uzeti u obzir zahtjeve aplikacije i količinu podataka.

Popis literature

[1] T. Carić i M. Buntić, »Uvod u relacijske baze podataka,« 2015. [Mrežno]. Available: http://files.fpz.hr/Djelatnici/tcaric/Tonci-Caric-Baze-podataka.pdf.

[2] Oracle, »OCI,« [Mrežno]. Available: https://www.oracle.com/database/what-is-database/.

[3] R. Peterson, »Database Architecture in DBMS: 1-Tier, 2-Tier and 3-Tier,« 2022. [Mrežno]. Available: https://www.guru99.com/dbms-architecture.html.

[4] N. H. Bercich, »The Evolution of the Computerized Database,« 2002. [Mrežno]. Available: https://concept.journals.villanova.edu/article/view/311/274.

[5] T. Šimić, »Objektno orijentirane baze podataka,« 2015. [Mrežno]. Available: https://zir.nsk.hr/islandora/object/unipu:233/preview.

[6] »The 3 Types of Relationships in Database Design,« 2016. [Mrežno]. Available: https://database.guide/the-3-types-of-relationships-in-database-design/.

[7] P. Loshin, »Structured Query Language (SQL),« 2022. [Mrežno]. Available: https://www.techtarget.com/searchdatamanagement/definition/SQL.

[8] »Introduction to Oracle Database,« 2015. [Mrežno]. Available: https://docs.oracle.com/cd/E11882_01/server.112/e40540/intro.htm#CNCPT001.

[9] »What Is Oracle Database,« 2022. [Mrežno]. Available: https://www.oracletutorial.com/getting-started/what-is-oracle-database/.

[10] M. Drake, »What is MySQL?,« 2020. [Mrežno]. Available: https://www.digitalocean.com/community/tutorials/what-is-mysql.

[11] »MySQL,« 2022. [Mrežno]. Available: https://en.wikipedia.org/wiki/MySQL#History.

[12] L. Moore, »MySQL,« 2018. [Mrežno]. Available: https://www.techtarget.com/searchoracle/definition/MySQL.

[13] »Microsoft SQL Server,« 2022. [Mrežno]. Available: https://en.wikipedia.org/wiki/Microsoft_SQL_Server.

[14] S. D. Perez, »What is Microsoft SQL Server and what is it for?,« 2021. [Mrežno]. Available: https://intelequia.com/en/blog/post/2948/what-is-microsoft-sql-server-and-what-is-it-for.

[15] »ACID,« 2022. [Mrežno]. Available: https://en.wikipedia.org/wiki/ACID.

[16] »ACID Explained: Atomic, Consistent, Isolated & Durable,« 2020. [Mrežno]. Available: https://www.bmc.com/blogs/acid-atomic-consistent-isolated-durable/.

[17] »What is Big Data? Introduction, Types, Characteristics, Examples,« 2022. [Mrežno]. Available: https://www.guru99.com/what-is-big-data.html.

[18] »What is Big Data and BigTable,« [Mrežno]. Available: https://codingcompiler.com/what-is-big-data-and-bigtable/.

[19] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes i R. E. Gruber, »Bigtable: A Distributed Storage System for Structured Data,« 2008.

[20] D. Kelly, »A Brief History of Databases: From Relational, to NoSQL, to Distributed SQL,« 2022. [Mrežno]. Available: https://www.cockroachlabs.com/blog/history-of-databases-distributed-sql/#the-arrival-of-the-nosql-database.

[21] M. Kleppmann, Designing Data-Intensive Applications, 2017.

[22] »Non-relational data and NoSQL,« 2022. [Mrežno]. Available: https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data.

[23] »NoSQL,« 2020. [Mrežno]. Available: https://www.w3resource.com/mongodb/nosql.php.

[24] A. Williams, »NoSQL database types explained: Key-value store,« 2021. [Mrežno]. Available: https://www.techtarget.com/searchdatamanagement/tip/NoSQL-database-types-explained-Key-value-store.

[25] »A Guide to Key-Value Databases,« 2022. [Mrežno]. Available: https://www.influxdata.com/key-value-database/.

[26] »What Is a Graph Database?,« [Mrežno]. Available: https://phoenixnap.com/kb/graph-database.

[27] »What is a Graph Database?,« 2022. [Mrežno]. Available: https://neo4j.com/developer/graph-database/.

[28] »What is a Document Database?,« 2021. [Mrežno]. Available: https://phoenixnap.com/kb/document-database.

[29] A. Meier i M. Kaufmann, SQL & NoSQL Databases, 2019.

[30] »What is Columnar Database?,« 2021. [Mrežno]. Available: https://hevodata.com/learn/columnar-databases/.

[31] »Columnar Database,« 2022. [Mrežno]. Available: https://www.heavy.ai/technical-glossary/columnar-database.

[32] M. Chapple, »Lifewire,« 2023.. [Mrežno]. Available: https://www.lifewire.com/abandoning-acid-in-favor-of-base-1019674.

[33] »CAP Theorem,« 2019. [Mrežno]. Available: https://www.ibm.com/cloud/learn/cap-theorem.

[34] »MongoDB,« 2020. [Mrežno]. Available: https://www.techtarget.com/searchdatamanagement/definition/MongoDB.

[35] »Cassandra Introduction: What is Apache Cassandra?,« [Mrežno]. Available: bmc.com/blogs/apache-cassandra-introduction/.

[36] »Apache HBase,« 2022. [Mrežno]. Available: https://en.wikipedia.org/wiki/Apache_HBase.

[37] »HBase - Overview,« [Mrežno]. Available: https://www.tutorialspoint.com/hbase/hbase_overview.htm.

[38] » Apache HBase,« [Mrežno]. Available: https://www.ibm.com/topics/hbase.

[39] »Netguru,« 2023.. [Mrežno]. Available: https://www.netguru.com/glossary/node-js.

[40] »TechTarget,« [Mrežno]. Available: https://www.techtarget.com/searchapparchitecture/definition/application-program-interface-API.

[41] »kinsta,« 2022.. [Mrežno]. Available: https://kinsta.com/knowledgebase/what-is-express-js/#what-is-expressjs.

[42] »Mirosoft,« 2023.. [Mrežno]. Available: https://learn.microsoft.com/en-us/sql/relational-databases/json/format-query-results-as-json-with-for-json-sql-server?view=sql-server-ver16.

Popis slika

Popis tablica