Usporedna analiza performansi relacijskih i NoSQL baza podataka

Šimić, Bartul

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

USPOREDNA ANALIZA PERFORMANSI RELACIJSKIH I NOSQL BAZA PODATAKA

Bartul Šimić

Split, rujan 2024.

Temeljna dokumentacijska kartica

Završni rad

Sveučilište u Splitu

Prirodoslovno-matematički fakultet

Odjel za informatiku

Ruđera Boškovića 33, 21000 Split, Hrvatska

COMPARATIVE ANALYSIS OF THE PERFORMANCE OF RELATIONAL AND NOSQL DATABASES

Bartul Šimić

SAŽETAK

Ovaj završni rad proučava i uspoređuje performanse relacijskih i NoSQL baza podataka, ukazuje na ključne razlike i potencijalne prednosti korištenja svakog tipa u specifičnim scenarijima. Testiraju se CRUD operacija je napravljano na četiri baze podataka: MySQL, Microsoft SQL Server, MongoDB, i CouchDB. Rezultati pokazuju kako MongoDB ima bolje performanse u većini testova, a prate ga relacijski sustavi MySQL i SQL Server. CouchDB se u ovim testovima pokazao najsporiji. Rad naglašava važnost odabira odgovarajuće tehnologije baze podataka na temelju specifičnih tehničkih i operativnih zahtjeva projekta, ističući kako nema jedinstvenog rješenja koje je optimalno za sve vrste aplikacija.

Ključne riječi: baza podataka, SQL, NoSQL, MySQL, Microsoft SQL Server, MongoDB, CouchDB

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 30 stranica, 1 grafički prikaz, 3 tablice i 8 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: **doc. dr. sc. Divna Krpan,** viši predavač Prirodoslovno-matematičkog fakulteta, Sveučilišta u Split

Ocjenjivači: **doc. dr. sc. Divna Krpan,** viši predavač Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

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

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

Rad prihvaćen: rujan, 2024.

Basic documentation card

Thesis

University of Split

Faculty of Science

Department of Computer Science

Ruđera Boškovića 33, 21000 Split, Croatia

USPOREDNA ANALIZA PERFORMANSI RELACIJSKIH I NOSQL BAZA PODATAKA

Bartul Šimić

ABSTRACT

This thesis studies and compares the performance of relational and NoSQL databases, highlighting key differences and potential advantages of using each type in specific scenarios. CRUD operations were tested on four databases: MySQL, Microsoft SQL Server, MongoDB, and CouchDB. The results show that MongoDB performs better in most tests, followed by relational systems MySQL and SQL Server. CouchDB was the slowest in these tests. The paper emphasizes the importance of selecting the appropriate database technology based on specific technical and operational requirements of the project, noting that there is no one-size-fits-all solution optimal for all types of applications

Key words: database, SQL, NoSQL, MySQL, Microsoft SQL Server, MongoDB, CouchDB

Thesis deposited in library of Faculty of science, University of Split

Thesis consists of: 30 pages, 1 figure, 3 tables and 8 references. Original language: Croatian

Mentor: **Divna Krpan, Ph.D.** Senior Lecturer of Faculty of Science, University of Split

Reviewers: **Divna Krpan, Ph.D.** Senior Lecturer of Faculty of Science, University of Split

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

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

Thesis accepted: September 2024

Sadržaj

Uvod

U današnje digitalno doba podaci su iznimno vrijedan resurs, stoga treba znati ispravno odabrati način skladištenja podataka s obzirom na potrebe i ograničenja projekta na kojem radimo. Tradicionalno su se za skladištenje podataka najčešće koristile relacijske baze podataka.

Međutim, pojava velikih podataka, računarstva u oblaku i potreba za visoko skalabilnim i distribuiranim sustavima dovela je do pojave NoSQL baza podataka. Ove baze podataka pružaju fleksibilniji pristup pohrani podataka, omogućujući upravljanje nestrukturiranim i polustrukturiranim podacima u distribuiranim okruženjima. Za razliku od relacijskih baza podataka, koje zahtijevaju čvrstu shemu, NoSQL baze podataka nude alternativu i omogućuju agilnije prakse razvoja te sposobnost rukovanja velikim količinama podataka s različitim strukturama.

Cilj ovog rada je istražiti razlike između relacijskih i NoSQL baza podataka s obzirom na njihovu arhitekturu, prednosti, ograničenja i performanse. Provođenjem niza testova performansi, rad ima za cilj istražiti mogućnosti oba pristupa te saznati kako se ove baze podataka ponašaju u različitim uvjetima, i ponuditi praktične smjernice za odabir odgovarajuće tehnologije baza podataka.

1. Sustav za pohranu podataka

Odabir sustava za upravljanje bazama podataka (engl. *Database Management System,* skraćeno DBMS) jedna je od najkritičnijih odluka u razvoju softvera. Izbor između relacijske i NoSQL baze može značajno utjecati na performanse, skalabilnost i sami uspjeh aplikacije. S obzirom na širok raspon dostupnih sustava za upravljanje bazama podataka, bitno je razumjeti njihove prednosti i nedostatke.

1.1. Ograničenja postojećih sustava

Relacijske baze podataka su već desetljećima najčešći odabir kod upravljanja podacima, no usprkos tome one imaju svoje mane. Jedan od najvećih izazova pri njihovom korištenju je skalabilnost. Baze podataka su ograničene fizičkim hardverom, količinom memorije i radom procesora. Korištenjem neke aplikacije, njena baza podataka se širi, a povećanjem broja korisnika raste i broj istovremenih upita na bazu. Memorija poslužitelja ograničava količinu podataka koje možemo pohraniti u bazu, a rad procesora ograničava koliko brzo možemo izvršavati operacije i upite nad tim podacima. Relacijske baze podataka dizajnirane su za vertikalno skaliranje, to znači da se poboljšanje performansi postiže povećavanjem resursa poslužitelja na kojemu se baza podataka nalazi. Međutim, ovaj način skaliranja je sam po sebi ograničen, jer nijedan poslužitelj nema neograničene resurse.

Drugo ograničenje relacijskih baza podataka je njihova čvrsta shema. Kod relacijskih baza, shema mora biti definirana prije nego što se podaci mogu zapisati. Ova zahtjev može predstavljati problem u sustavima čiji se razvoj odvija paralelno s rastom aplikacije i baze korisnika. U takvim sustavima često dolazi do promjena strukture podataka čija implementacija ponekad zahtjeva privremeno zaustavljanje rada aplikacije i izvršavanje kompleksnih migracija. Cijeli ovaj proces sam po sebi uvodi rizike i usporava vrijeme razvoja.

NoSQL baze podataka nude rješenje za ova ograničenja. Dizajnirane su za horizontalno skaliranje, za razliku od vertikalnog skaliranja u kojemu se povećavaju performanse jednoga poslužitelja, kod horizontalnog skaliranja, se povećava broj poslužitelja, što

omogućava gotovo neograničenu skalabilnost. NoSQL baze podataka ne koriste čvrste sheme, što omogućava veću fleksibilnost u obradi nestrukturiranih podataka. Međutim, ova fleksibilnost dolazi po cijenu konzistentnosti i mogućnosti složenijih upita, koji su temelj relacijskih baza podataka.

Fleksibilnost ovakvih baza najviše dolazi do izražaja kada se koriste uz *auto-scaling*. Sve vodeće cloud platforme pružaju ovu opciju a ona omogućava da se broj poslužitelja na kojima je pokrenuta baza podataka automatski povećava u ovisnosti o potražnji u stvarnom vremenu. Broj korisnika svake aplikacije varira kroz različito doba dana, tjedna ili godine, na ovaj način će u doba najveće potražnje biti pokrenuto najviše poslužitelja te korisnici neće osjetiti nikakve poteškoće zbog povećanog prometa na aplikaciji. U periodima kada se smanji broj korisnika, nepotrebni poslužitelji se ponovno gase da bi se uštedjelo na troškovima rada poslužitelja.

1.2. Očekivani rezultati

Očekivani rezultati ove studije uključuju detaljno razumijevanje karakteristika performansi relacijskih i NoSQL baza podataka. Usporedbom performansi MySQL, Microsoft SQL Server, MongoDB i CouchDB, rad ima za cilj identificirati scenarije u kojima jedan tip baze bolji od drugog. Ovaj rad pružit će vrijedne uvide za programere i administratore baza podataka, pomažući im odabrati najprikladniju tehnologiju baza podataka na temelju specifičnih zahtjeva projekta, karakteristika podataka i potreba za skalabilnošću.

2. Relacijske baze podataka

Relacijske baze podataka temelje se na relacijskom modelu, po tom modelu podaci su organizirani u tablice (relacije) sastavljene od redaka i stupaca. Svaka tablica predstavlja jedan entitet, redci tablice predstavljaju zapise a stupci atribute. [1]

Relacijski model prvi je predstavio Edgar F. Codd 1970. godine. Neki od najpoznatijih sustava za upravljanje podacima, kao što su MySQL, PostgreSQL, Oracle i Microsoft SQL Server, se temelje na njemu. Ključne prednosti relacijskih baza podataka su očuvanje integriteta pohranjenih podataka, mala redundancija, mogućnost upita s kompleksnim uvjetima. Te značajke čine relacijske baze podataka posebno pogodnima za aplikacije kao što su financijski sustavi, sustavi za upravljanje odnosima s klijentima (engl. *Customer Relationship Management*, skraćeno CRM) te sustavi za upravljanje inventarom koji je prikazan u ovom radu. [2]

Relacijski model koristi relacijsku algebru, skup operacija koji se može koristiti za manipulaciju i dohvat podataka pohranjenih u relacijskim tablicama. Osnovne operacije relacijske algebre su: selekcija, projekcija, unija, presjek i spoj. Strukturirani upitni jezik (engl. *Structured Query Language*, skraćeno SQL) je standardni jezik koji se koristi za interakciju s relacijskim bazama podataka.

SQL pruža razne mogućnosti te omogućuje korisnicima dohvat podataka iz različitih tablica, filtriranje rezultata na temelju uvjeta i izvođenje složenih izračuna. SQL također podržava operacije za manipulaciju podacima, uključujući umetanje, ažuriranje i brisanje.

2.1. Prednosti i nedostaci

Jedna od ključnih prednosti relacijskih baza podataka je integritet podataka, koji se postiže primjenom ograničenja kao što su primarni i strani ključevi te jedinstvene vrijednosti. Ova ograničenja osiguravaju da podaci ostaju točni, konzistentni i pouzdani, čime se osigurava njihova kvaliteta. Također, relacijske baze podržavaju složene upite, omogućujući korisnicima spajanje podataka iz različitih tablica, filtriranje rezultata i agregiranje podataka. Ove mogućnosti složene analize čini relacijske baze idealnima za aplikacije koje

zahtijevaju izvještavanje i analizu podataka. Relacijske baze podataka se pridržavaju ACID svojstava (atomarnost, konzistentnost, izolacija, trajnost), čime se osigurava pouzdana i dosljedna obrada transakcija čak i u slučaju sustavnih grešaka. Ova vrsta baze se koriste već desetljećima i njihova primjena je iznimno široka, rezultat toga je iznimno velik broj programera koji ih koristi, odnosno razvijanje ekosustava u kojemu je lako pronaći odgovore na pitanja, savjete i podršku.

Međutim, relacijske baze podataka imaju i svoje nedostatke. Primarno, dizajnirane su za vertikalno skaliranje, što znači da performanse ovise o povećanju resursa na poslužitelju. Ovakav pristup otežava rad s velikim količinama podataka ili istovremenim operacijama, što može predstavljati problem u okruženjima s velikim prometom. Osim toga, relacijske baze zahtijevaju unaprijed definiranu shemu podataka. Promjene u strukturi podataka mogu biti izazovne, spore i skupe, što ih čini manje pogodnima za aplikacije koje često mijenjaju strukturu podataka.

2.2. MySQL

MySQL je jedan od najrasprostranjenijih sustava za upravljanje relacijskim bazama podataka. Razvila ga je švedska tvrtka MySQL AB, a trenutno je u vlasništvu tvrtke Oracle Corporation. Ovaj sustav naširoko se koristi u web aplikacijama i sustavima za upravljanje sadržajem, prvenstveno zbog svoje fleksibilnosti i pouzdanosti. Popularan je zbog jednostavnosti korištenja, visokih performansama i zbog toga što je program otvorenog kôda, a to zajednici omogućava stalno unaprjeđivanje i prilagodbu različitim potrebama.

MySQL koristi arhitekturu klijent-poslužitelj, pri čemu poslužitelj baze podataka upravlja podacima i obrađuje upite, dok se klijenti povezuju na poslužitelj kako bi komunicirali s bazom podataka. MySQL podržava razne module za pristup podacima, uključujući InnoDB, MyISAM i Memory, od kojih je svaki optimiziran za različitu primjenu. InnoDB je zadani mehanizam pohrane, pruža ACID svojstva transakcija, podršku za strane ključeve i zaključavanje redaka, što ga čini prikladnim za aplikacije koje zahtijevaju očuvanja integritet podataka i kontrolu istovremenog pristupa. [3]

MySQL također podržava replikaciju podataka, što omogućuje ravnomjernije opterećenje među poslužiteljima, kao i sigurnosno kopiranje i oporavak u slučaju pogrešaka. Funkcija particioniranja omogućuje razdvajanje tablica na manje, upravljivije dijelove, čime se poboljšava učinkovitost pri radu s velikim skupovima podataka. Pored toga, sustav nudi

mogućnost full-text pretraživanja, što ubrzava rad u aplikacijama koje zahtijevaju pretragu velikih količina tekstualnih podataka.

MySQL se često koristi u web aplikacijama. Također se koristi u sustavima za upravljanje sadržajem kao što su WordPress i Drupal, te na platformama za e-trgovinu, online forumima i skladištima podataka.

2.3. Microsoft SQL Server Management Studio 18

Microsoft SQL Server Management Studio (SSMS) je alat namijenjen za upravljanje relacijskim bazama podataka na Microsoft SQL Serveru. Ovaj alat koristi se u raznim poslovnim okruženjima, od manjih projekata do velikih sustava, a omogućuje administratorima i razvojnim inženjerima kreiranje, konfiguriranje i nadziranje baza podataka te pisanje i izvršavanje SQL skripti. [4]

SSMS nudi integracijske usluge, koje omogućuju ekstrakciju, transformaciju i učitavanje podataka (ETL) iz različitih izvora, kao i analitičke usluge koje podržavaju izgradnju rješenja za online analitičku obradu podataka (OLAP) i rudarenje podataka. Usluge izvještavanja omogućuju izradu, upravljanje i distribuciju izvještaja u raznim formatima, uključujući tablice, grafove i matrice.

SQL Server široko se često koristi u granama kao što su bankarstvo i zdravstvo. Često se koristi za skladištenje podataka, poslovnu inteligenciju, upravljanje odnosima s klijentima (engl. *Customer Relationship Management*, skraćeno CRM) i sustave za planiranje resursa tvrtke (engl. *Enterprise resource planning*, skraćeno ERP).

3. NoSQL baze podataka

3.1. Pregled

NoSQL baze podataka predstavljaju alternativu tradicionalnim relacijskim bazama, osmišljene kako bi riješile probleme skalabilnosti i fleksibilnosti. Za razliku od relacijskih baza podataka koje koriste tablice za pohranu podataka, NoSQL baze podataka koriste različite modele podataka, neki od njih su: skladišta dokumenata (engl. *Document Store*), skladišta ključ-vrijednosti (engl. *Key-Value Store*), stupčani sustavi (engl. *Column-Family Store*) i grafičke baze podataka.

Naziv "NoSQL" znači "Ne samo SQL", taj naziv je odabran da bi se naglasila činjenica da se ovakve baze razlikuju od tradicionalnog SQL jezika za upite. Umjesto toga, NoSQL baze podataka često koriste prilagođene jezike za upite ili API-je prilagođene njihovim specifičnim modelima podataka. Ova fleksibilnost čini NoSQL baze pogodnima za aplikacije koje zahtijevaju rad s velikim količinama podataka, visok promet i potrebu za visokom dostupnošću, kao što su moderne web aplikacije, analitika velikih podataka i obrada u stvarnom vremenu.

Primjer NoSQL baze podataka su skladišta dokumenata poput MongoDB-a i CouchDB-a, koja pohranjuju podatke u dokumentima nalik JSON-u (Javascript Object Notation). Ovakvi dokumenti mogu sadržavati složene strukture, uključujući ugniježđene nizove i parove ključ-vrijednost, što ih čini idealnim za aplikacije koje obrađuju složene podatke. Također, postoje skladišta ključ-vrijednost poput Redisa i DynamoDB-a, koja pohranjuju podatke u obliku jedinstvenih ključeva i pripadajućih vrijednosti, omogućujući brze operacije čitanja i pisanja. Ovaj model posebno je koristan za real-time analizu podataka i predmemoriju (engl. cache).

Stupčani sustavi, kao što su Apache Cassandra i HBase, organiziraju podatke u stupce i obitelji stupaca, što omogućava visok kapacitet pisanja podataka, osobito u distribuiranim okruženjima. Grafičke baze podataka, poput Neo4j-a i Amazon Neptunea, pohranjuju podatke u obliku čvorova i rubova, što je posebno korisno za aplikacije koje prate složene odnose, kao što su društvene mreže ili sustavi za otkrivanje prijevara.

3.2. Prednosti i nedostaci

NoSQL baze podataka pružaju veću skalabilnost u usporedbi s relacijskim bazama jer su dizajnirane za horizontalno skaliranje. Tako podaci mogu biti distribuirani na više poslužitelja, što omogućuje gotovo neograničenu skalabilnost. Ovakav pristup čini ih pogodnima za aplikacije s velikim prometom i velikim količinama podataka. Još jedna važna prednost je fleksibilnost NoSQL baza, koje ne zahtijevaju unaprijed definiranu shemu podataka, čime su idealne za aplikacije koje su još u razvoju i čija se struktura podataka često mijenja.

Međutim, NoSQL baze imaju i svoje nedostatke. Jedan od glavnih izazova je kompromis u pogledu konzistentnosti podataka. Dok relacijske baze jamče trenutačnu konzistentnost, NoSQL baze često koriste princip "eventualne konzistentnosti", gdje podaci postaju konzistentni s vremenom, što može biti problematično u određenim aplikacijama. Također, NoSQL baze nude ograničene mogućnosti izvođenja složenih upita u usporedbi s relacijskim bazama, iako se to može nadoknaditi prilagođenim logikama ili korištenjem specifičnih jezika za upite. Konačno, programerima naviknutim na rad s relacijskim bazama može biti potrebna dodatna obuka kako bi se prilagodili radu s NoSQL bazama, jer zahtijevaju drugačiji pristup modeliranju podataka i radu s upitima. [7]

3.3. MongoDB

MongoDB je jedna od najpoznatijih NoSQL baza podataka, orijentirana na dokumente. Podaci se pohranjuju u formatu BSON (engl. Binary JSON), što omogućuje pohranu složenih i hijerarhijskih struktura podataka. Za upite se koristi MongoDB Query Language (MQL), koji je specifičan za ovaj sustav. MongoDB je najčešće u upotrebi kod web aplikacija, sustava za upravljanje sadržajem, e-trgovinskih platformi te za analizu velikih količina podataka. [5]

Jedna od ključnih značajki MongoDB-a je njegova distribuirana arhitektura, koja omogućava pohranu podataka u kolekcijama dokumenata. Svaki dokument je BSON objekt koji može sadržavati složene strukture poput ugniježđenih polja, nizova i parova ključ-vrijednost. MongoDB podržava sharding, tehniku za raspodjelu podataka na više poslužitelja, što omogućava horizontalno skaliranje i osigurava visoku dostupnost sustava.

MongoDB nudi razne mogućnosti indeksiranja, poput indeksa pojedinačnih polja, kombinacije više polja, te geoprostornih i tekstualnih indeksa. Ove mogućnosti ubrzavaju upite omogućujući lakše pronalaženje podataka prema zadanim indeksima. No, indeksi treba koristiti umjereno jer prekomjerno indeksiranje može nepotrebno opterećivati memoriju.

Još jedna značajka MongoDB-a su njegovi agregacijski okviri, koji pružaju napredne alate za obradu podataka poput filtriranja, grupiranja i transformacije.

MongoDB podržava replikaciju i sharding. Replikacija omogućava postavljanje više replika podataka, što osigurava redundanciju i visoku dostupnost sustava. Sharding omogućuje distribuciju podataka na više poslužitelja, što značajno poboljšava skalabilnost i performanse.

3.4. CouchDB

CouchDB je još jedna NoSQL baza podataka orijentirana na dokumente. Za razliku od MongoDB-a, CouchDB pohranjuje podatke u jednostavnom JSON formatu. Glavna razlika između ove dvije baze je način pristupa podacima – CouchDB koristi HTTP zahtjeve za interakciju s bazom.

CouchDB također koristi distribuiranu arhitekturu s ugrađenom podrškom za replikaciju i sinkronizaciju podataka, što ga čini pouzdanim rješenjem za distribuirana okruženja. [6]

MapReduce je model za obradu podataka koji omogućuje programerima kreiranje prilagođenih funkcija za mapiranje podataka u parove ključ-vrijednost, a zatim daljnju redukciju i agregaciju tih podataka s pomoću reduce funkcije. Ovaj model integriran je u jezike poput JavaScripta, Pythona i Rubyja, što ga čini fleksibilnim za različite razvojne platforme.

Za pristupanje podacima CouchDB koristi REST API, što znači da korisnici komuniciraju s bazom podataka koristeći HTTP metode (GET, POST, PUT, DELETE) za izvođenje osnovnih operacija poput kreiranja, čitanja, ažuriranja i brisanja podataka (CRUD). Ovaj pristup olakšava integraciju s postojećim web aplikacijama.

Osim toga, CouchDB podržava ACID svojstva na razini pojedinačnih dokumenata, osiguravajući da se podaci ažuriraju dosljedno i pouzdano čak i u distribuiranim okruženjima, što ga čini jedinstvenim među mnogim NoSQL bazama.

CouchDB se često koristi u aplikacijama koje zahtijevaju offline pristup i distribuiranu pohranu podataka. Njegova jednostavnost korištenja i REST API čine ga popularnim izborom za web i mobilne aplikacije, posebno one koje trebaju raditi u okruženjima s povremenom povezanošću s internetom.

4. Testiranje

Nakon uvodnog istraživanja na red je došao i praktični dio ovog rada. Ideja istraživanja je bila osmišljavanje različitih testova koji će omogućiti usporedbu različitih sustava za upravljanje bazama podataka i tako postaviti kvalitetne temelje za analizu dobivenih rezultata. Promatrani sustavi imaju drugačije arhitekture te različite načine izvršavanja, također razlikuju se i po jezicima koje koriste za pisanje upita. Stoga je potrebno odabrati niz testova koje je moguće replicirati u svim sustavima, iz tog razloga su odabrane CRUD operacije, također te operacije su osnova na kojoj se zasnivaju svi složeniji upiti stoga ih je korisno usporediti.

4.1. Metodologija

Da bi se osiguralo jednako okruženje za sve sustave, svi testovi su napravljeni u lokalnom okruženju. Ova metoda ima neke nedostatke, ali ipak dovoljno je dobra za procjenu performansi različitih sustava.

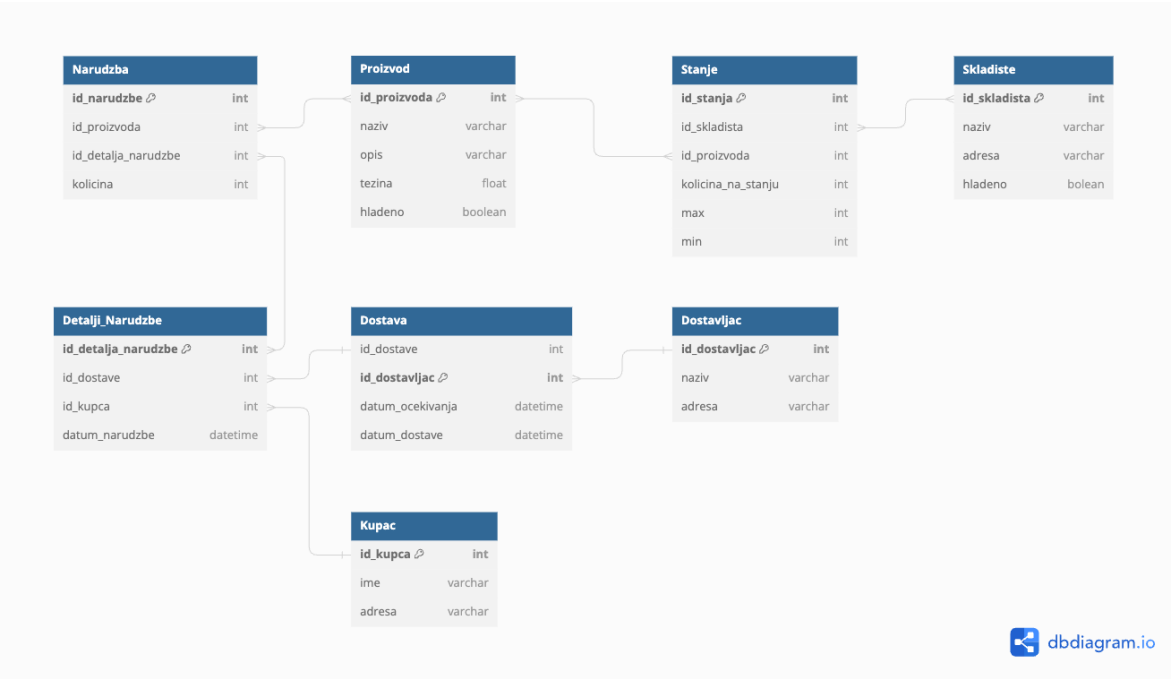
Prvi očiti nedostatak je taj što lokalno okruženje nije realna situacija u kojoj ovakvi sustavi svakodnevno funkcioniraju. Simuliranje pravog okruženja je jako nepraktično radi velikog broja varijabli koje utječu na performanse baze podataka unutar aplikacije koju koriste korisnici. Obično su baza podataka i aplikacija poslužene na nekom od poznatih cloud poslužitelja i to na više različitih čvorova na mreži, takav način omogućava redundantnost u slučaju kvara pojedinih čvorova kao i također smanjenje latencije koja nastaje radi prijenosa podataka putem interneta radi fizičke udaljenosti korisnika i čvora na kojemu se aplikacija nalazi.

U stvarnim okolnostima je također moguće da više različitih korisnika u isto vrijeme zahtjeva manipulira iste podatke unutar baze podataka što dovodi do dodatnog kašnjenja i opterećenja na sustav, i takav slučaj također nije pokriven ovakvim testom.

Radi raznovrsnijih podataka, testiranje se radi na različitim skupovima podataka, odnosno na različitoj količini podataka, a to su skupovi od 1.000, 10.000 i 100.000 redaka.

4.2. Model baze podataka

Za potrebe ovog rada, osmišljena je baza podataka koju koristi fiktivna trgovina koja se bavi skladištenjem i dostavljanjem proizvoda. Na Slika 4.1 vidimo ERD dijagram baze (engl. *Entity Relationship Diagram*).



Slika 4.1 - ERD Dijagram

Najvažnija tablica u modelu je `Narudzba`, ona služi kao poveznica različitih proizvoda i same narudžbe, ako se u narudžbi nalazi više proizvoda, svaki od njih će biti naveden kao zasebni redak u tablici `Narudzba`. U tablici `Detalji_Narudzbe` se zapisuje vrijeme narudžbe te podaci o dostavi i kupcu. `Dostava` također ima svoje polja koja je pobliže opisuju a to su vrijeme početka dostave i očekivano vrijeme pristizanja proizvoda kao i podaci o kupcu i o dostavljaču.

Baza također služi za pregledavanje trenutnog stanja proizvoda na različitim skladištima. Svaki redak u tablici `Stanje` ima podatak o količini određenoga proizvoda na određenome skladištu, kao i dodatne podatke o minimalnoj i maksimalnoj dopuštenoj

količini. `Skladište` je također krajnja tablica kao `Dostavljac` i `Kupac`, u njoj su zapisani podaci o imenu i adresi skladišta te jeli ono hlađeno ili nije.

Za korištenje NoSQL baza korišten je isti model kao i kod SQL baza, svaka tablica je prevedena u zasebnu kolekciju u MongoDB-u, odnosno bazu podataka u CouchDB-u. struktura podataka unutar kolekcija je ostala ista. Dodatna pogodnost kod NoSQL baza je ta što su podaci već generirani u JSON formatu, pa s toga nije potrebno raditi nikakve dodatne obrade podataka prije njihovog umetanja u kolekcije.

4.3. Skripte

Da bi testiranje u različitim sustavima bilo ravnopravno, svi testovi su izvršeni kroz skripte u programskome jeziku Python, on je odabran za ovu ulogu zbog svoje rasprostranjenosti, dostupne dokumentacije i podrške za rukovanje sa svim sustavima koji su uključeni u testiranje. Na ovaj način je olakšano automatsko ponavljanje testova, a potrebne izmjene je lako napraviti i ponovno testirati.

Skripte su u projektu podijeljene na nekoliko različitih vrsta:

- Skripte za generiranje podataka: njihova svrha je da se omogući korištenje identičnih podataka u različitim bazama. Pokreću se prve i spremaju podatke u JSON format.
- Skripte za spajanje na bazu i postavljanje upita: postoje 4 ovakve skripte, za svaki sustav koji se testira po jedna. One koriste već postojeće biblioteke za rukovanje bazama, prvo se osigurava spoj na bazu a zatim za svaku CRUD operaciju postoji posebna funkcija.
- Pomoćne skripte: ovo skripte olakšavaju pisanje ostatka kôda i sprječavaju bespotrebno ponavljanje istog kôda, koriste se u ostalim skriptama. Primjer ovakve funkcije je funkcija za mjerenje vremena.

4.3.1. Mjerenje vremena

Usprkos tome što je mjerenje vremena jedan od najbitnijih dijelova kôda, mjerenje vremena nije toliko komplicirano kao što vidimo u Isječak iz koda 1. Funkcija za mjerenje vremena prima više ulaznih parametara. Najbitniji je parametar `func`, preko njega se prima funkcija kojoj se mjeri vrijeme izvršavanja, a njeni parametri se prosljeđuju preko

parametara `*args` i `**kwargs`. U kôdu se ova funkcija koristi na mnogo mjesta, tako da se dodatno prosljeđuje parametar `func_name` koji služi isključivo kao dodatni komentar prilikom ispisa izmjerenoga vremena, da bi se naznačilo koja se točno funkcija mjerila.

Samo mjerenje je zapravo dosta jednostavno. Prije i poslije izvršavanja željene funkcije uzima se trenutno vrijeme, izračuna se njihova razlika i dobije se točno vrijeme izvršavanja funkcije koje se zatim ispisuje u terminal.

```
import time

def time_function(func_name, func, *args, **kwargs):
    start_time = time.time()
    result = func(*args, **kwargs)
    duration = time.time() - start_time
    print(f"{duration:.4f} - {func_name}")
    return result
```

Isječak iz koda 1 - Funkcija za mjerenje vremena

4.3.2. Generiranje podataka

Ručno kreiranje ovoliko velikog broja podataka ne dolazi u obzir, na sreću postoje biblioteke koje rješavaju ovaj problem. Za Python je dostupna biblioteka Faker, ona se koristi za generiranje jednostavnih tekstualnih ili brojčanih podataka. Nastala je po uzoru na istoimene biblioteke za PHP, Pearl i Ruby programske jezike. Osnovna verzija biblioteke je pogodna za generiranje korisničkih podataka kao što su ime, prezime i adresa. Prilikom generiranja nasumičnih brojeva ili datuma se određuju minimalna i maksimalna željena vrijednost kao i najmanji razmak između dvije susjedne vrijednosti. Različite zemlje i jezici koriste različite pisma i na drugačiji način pišu datume. U Europi se koriste datumi u formatu dan, mjesec godina ili godina, mjesec, dan, dok se u Sjedinjenim Američkim Državama koristi najčešće koristi format mjesec, dan, godina. Faker podržava oba načina pisanja kao što podržava različita pisma. Tako bi postavljanjem lokaliteta na ruski, dobili tekst na ćirilici.

Ako ove funkcionalnosti nisu dovoljno raznolike za nečiju potrebu, moguće je napraviti vlastiti generator koji se može koristiti unutar Fakera, pa tako postoji mnogo različitih generatora koji su orijentirani na specifična područja rada kao što su npr. sport, filmovi ili glazba. Valja napomenuti da verzije Fakera za različite programske jezike nemaju iste funkcionalnosti jer nemaju iste autore. [8]

Isječak iz koda 2 prikazuje primjenu biblioteke za generiranje podataka o proizvodu.

```
def generiraj_proizvod():
    return {
        "naziv": fake.word(),
        "opis": fake.sentence(),
        "tezina": round(random.uniform(0.1, 100.0), 2),
        "hladeno": random.choice([True, False])
    }
```

Isječak iz koda 2 - Funkcija za generiranje proizvoda

`Ime` proizvoda je riječ, `opis` rečenica, `tezina` može biti broj između 0.1 i 100, zaokružen na 2 decimale, a `hladeno` je boolean, može biti ili istinit ili lažan.

4.3.3. Spajanje na bazu

Prije upita izvršavanja upita, potrebno se spojiti na bazu a to se radi na sličan u svim sustavima, prilikom kreiranja baze odaberu se parametri za spajanje kao što su host, port, korisničko ime i šifra.

```
self.connection = mysql.connector.connect(
    host=db_config['host'],
    database=db_config['database'],
    user=db_config['user'],
    password=db_config['password']
)
```

Isječak iz koda 3 - MySQL - Spajanje na bazu podataka

```
self.client = MongoClient(config['host'], config['port'])
self.db = self.client[config['database']]
```

```
self.collection = self.db[config['collection']]
```

Isječak iz koda 4 - MongoDB - Spajanje na bazu podataka

U oba primjera se vidi korištenje posebne biblioteke za spajanje na bazu, one se također koriste i za izradu upita prema bazi. U cilju postizanja jednakih uvjeta za svaku bazu je korištena odgovarajuća biblioteka.

4.3.4. Umetanje

Kod relacijskih baza postoji dodatni korak izrade tablica prije nego li u njih možemo umetati podatke.

```
create_table_query = '''
CREATE TABLE IF NOT EXISTS proizvodi (
    id INT AUTO_INCREMENT PRIMARY KEY,
    ime VARCHAR(255),
    opis VARCHAR(255),
    tezina FLOAT,
    hladeno BOOLEAN
);
'''
```

Isječak iz koda 5 - MySQL - Kreiranje tablice

Razlika između fiksne sheme podataka koju koristi MySQL i nestrukturiranih dokumenata koje koristi MongoDB jasno se vidi iz kôda. Za umetanje je potrebno napraviti tablicu s unaprijed određenim tipovima podataka, i zatim je potrebno točno mapirati vrijednosti iz objekta proizvoda u ispravne stupce tablice proizvodi.

```
def insert_logic():
    insert_query = '''
    INSERT INTO proizvodi (ime,opis,tezina,hladeno)
    VALUES (%s, %s, %s, %s);
    '''
    data_to_insert = [(p['naziv'], p['opis'], p['tezina'],
p['hladeno']) for p in products]

    self.cursor.executemany(insert_query, data_to_insert)
```

```
        self.connection.commit()

        return self.cursor.rowcount
```

Isječak iz koda 6 - MySQL - Umetanje u tablicu

Isječak iz koda 7 prikazuje korištenje funkcije za mjerenje vremena. Valja napomenuti da se učitavanje podataka u memoriju aplikacije nalazi van funkcije kojoj se mjeri vrijeme da taj dio procesa, koji nije bitan za mjerenje, ne promijeni rezultate testiranja.

```
def insert_data(self, file_path):
    with open(file_path, 'r') as json_file:
        products = json.load(json_file)

    return time_function('MYSQL - insert', insert_logic)
```

Isječak iz koda 7 - Mjerenje vremena umetanja

```
def insert_logic():
    result = self.collection.insert_many(products)

    return result
```

Isječak iz koda 8 - MongoDB - Umetanje u tablicu

4.3.5. Čitanje

U oba slučaja su naredbe su dosta kratke i jednostavne.

```
def read_logic():
    select_query = "SELECT * FROM proizvodi"

    self.cursor.execute(select_query)
    records = self.cursor.fetchall()

    return records
```

Isječak iz koda 9 - MySQL - Čitanje iz tablice

```
def read_logic():
    records = self.collection.find({})

    return records
```

Isječak iz koda 10 - MongoDB - Čitanje iz tablice

4.3.6. Ažuriranje

Proizvoljnim odabirom je odlučeno da će se u usporedbi svim proizvodima ažurirati svojstvo `hladeno`.

```
def update_logic():
    update_query = ''' UPDATE proizvodi SET hladeno = false
'''
    self.cursor.execute(update_query)
    self.connection.commit()

    return self.cursor.rowcount
```

Isječak iz koda 11 - MySQL - Ažuriranje tablice

```
def update_logic():
    result = self.collection.update_many({}, {'$set':
{'hladeno':'false'}})
    return result
```

Isječak iz koda 12 - MongoDB - Ažuriranje tablice

4.3.7. Brisanje

Kao i kod naredbi za čitanje, brisanje je također dosta slično i jednostavno u svim sustavima.

```
def delete_logic():
    delete_query = "DELETE FROM proizvodi"
    self.cursor.execute(delete_query)
    self.connection.commit()

    return self.cursor.rowcount
```

Isječak iz koda 13 - MySQL - Brisanje iz tablice

```
def delete_logic():
    result = self.collection.delete_many({})
    return result

return time_function('MONGO - delete', delete_logic)
```

Isječak iz koda 14 - MongoDB - Brisanje iz tablice

Kod korištenja CouchDB-a brisanje cijele kolekcije se radi tako da se briše element po element, ne postoji opcija brisanja svih elemenata istovremeno kao kod drugih sustava.

```
def delete_logic():
    for doc_id in list(self.db):
        self.db.delete(self.db[doc_id])
    return len(self.db)
```

Isječak iz koda 15 - CouchDB - Brisanje iz tablice

4.3.8. CouchDB

Prilikom testiranja CouchDB je konstantno pokazivao najlošije rezultate među ostalim bazama. Razlog tome je korištenje HTTP zahtjeva prilikom svakoga upita i manjkavost metoda za obradu veće količine podataka od jednom. Uzmimo brisanje za primjer, CouchDB ne podržava naredbu koja bi bila ekvivalentna MySQL naredbi `DELETE FROM [table]` koja bez previše muke briše sve podatke iz željene tablice. Da bi se postigao isti rezultat u CouchDB-u potrebno je prvo dohvatiti identifikatore svih polja koja želimo obrisati, a zatim sastaviti novi niz podataka u kojem ćemo uz svaki identifikator postaviti *flag* `_deleted = true` te zatim pozvati zahtjev za ažuriranje podataka. Iz ovih razloga je odlučeno da se pri testiranju koristi metoda navedena u prethodnom poglavlju

Slična je bila situacija s umetanjem podataka, no u ovome slučaju je ipak pronađeno rješenje za poboljšanje rezultata. Prvotno se koristila metoda upisivanja dokumenta po dokument, no prethodno opisana metoda s ažuriranjem podataka je u ovom slučaju moguća jer nije potreban prethodni korak dohvaćanja svih identifikatora, već se oni pri umetanju generiraju automatski.

```
def insert_logic():
            self.db.update(products)
            return len(products)
```

Isječak iz koda 16 - Umetanje skupa podataka u CouchDB-u

Korištenjem ove metode rezultati umetanja su se približili rezultatima umetanja u ostale baze.

4.4. Rezultati testiranja

Nakon izvršenih testova s različitim količinama podataka dobiveni su sljedeći rezultati.

1.000 Redaka	**Brisanje**	**Umetanje**	**Čitanje**	**Ažuriranje**
MongoDB	0.0048	0.0106	0.0000	0.0103
MySQL	0.0040	0.0189	0.0132	0.0055
SSMS	0.0064	0.0362	0.0221	0.0084
CouchDB	35.189	0.0397	0.0356	31.460

Tablica 4.1 - Rezultati testiranja za 1.000 redaka

10.000 redaka	**Brisanje**	**Umetanje**	**Čitanje**	**Ažuriranje**
Mongo	0.0348	0.0858	0.0000	0.0841
MySQL	0.0356	0.1595	0.0208	0.0301
SSMS	0.0452	0.2152	0.0245	0.0392
CouchDB	383.985	1.1037	0.5982	301.182

Tablica 4.2 - Rezultati testiranja za 10.000 redaka

100.000 redaka	Brisanje	Umetanje	Čitanje	Ažuriranje
Mongo	0.3116	0.7055	0.0000	0.6817
MySQL	0.3750	10.429	0.1864	0.2590
SSMS	0.4023	12.532	0.1924	0.2932
CouchDB	3.791.503	123.642	45.058	3.641.592

Tablica 4.3 - Rezultati testiranja za 100.000 redaka

Iz rezultata se može vidjeti da MongoDB prednjači u gotovo svim kategorijama bez obzira na količinu podatka s kojima se rukuje. Testiranje osnovnih CRUD operacija nad velikim skupom podataka odgovara arhitekturi MongoDB-a zato što se ne koriste nikakvi uvjeti za pretraživanje podataka, tako da se obrada radi nad cijelim skupom podataka. Jedina kategorija u kojoj MongoDB kaska za MySQL bazom podataka je Ažuriranje. Razlog tome je strukturiranost relacijske baze podataka, redci se brže pronalaze i u njima se ažurira samo jedan podatak, odnosno stupac.

U usporedbi između dvije relacijske baze podataka vidimo da je Microsoft Server nešto sporiji od MySQL-a, što je također bilo očekivano zato što je on prigodniji za korištenje kao skladište podataka, gdje se radi s puno većim količinama podataka i dosta složenijim upitima i funkcionalnostima. Jednostavne CRUD operacije nisu jača strana ovakve baze podataka.

CouchDB uvjerljivo ima najlošije u ovom testiranju. Kombinacija modela MapReduce i izvršavanja upita preko HTTP zahtjeva nije najbolji odabir za ovaj način testiranja. U drugačijemu okruženju i testovima fokusiranima na manju količinu podataka ali velik broj istovremenih upita bi se ova baza bi zasigurno imala bolje rezultate, jer je dizajnirana za takve primjene.

5. Zaključak

Prema rezultatima provedenog testiranja može se zaključiti da MongoDB i MySQL s razlogom imaju toliko velik broj korisnika. To su provjereni sustavi koji će zadovoljiti potrebe za većinu aplikacija. No bilo bi nepravedno na temelju isključivo ovog testiranja potpuno isključiti primjenu CouchDB-a, cilj ovoga testa nije bio dati prednost nekoj od navedenih tehnologija, te je sigurno da bi CouchDB zablistao u drugačijim okolnostima.

Testirane tehnologije „ispod haube“ funkcioniraju na različite načine te je svaka od njih najbolji odabir kad se njen potencijal maksimalno iskoristi. Na posljetku baza podataka je samo jedan od alata u rukama programera, a različiti zadaci zahtijevaju različite alate.

Pri razvoju aplikacije nipošto se ne smije zanemariti faza istraživanja i odabira arhitekture koja prethodi razvojnoj fazi. Programer mora poznavati sve zahtjeve aplikacije na kojoj radi, te prema njima odabrati arhitekturu koja će se koristiti pri razvoju. Također je bitno imati plan kako će se ta arhitektura prilagođavati rastu aplikacije (predviđenom ili ne predviđenom).

Potrebno je konstantno ulagati vrijeme i trud u svoj osobni razvoj, pratiti nove tehnologije, testirati različite scenarije, pokušavati doći do limita određene tehnologije i naučiti kako premostiti probleme koji nastaju u tim okolnostima. Bitno je – igrati se. Kroz igru učimo, rastemo i postajemo bolji!

Literatura

[1] R. Manger, Osnove projektiranja baza podataka, Zagreb: Srce, 2010.

[2] S. B. N. Ramez Elmasri, Fundamentals of database systems, New York: Pearson, 2016.

[3] Oracle, »MySQL Dokumentacija,« Oracle, 2024. [Mrežno]. Available: https://dev.mysql.com/doc/. [Pokušaj pristupa 25 08 2024].

[4] Microsoft, »Microsoft SQL Dokumentacija,« 2024. [Mrežno]. Available: https://learn.microsoft.com/en-us/sql/?view=sql-server-ver16. [Pokušaj pristupa 25 08 2024].

[5] I. MongoDB, »MongoDB Dokumentacija,« 2024. [Mrežno]. Available: https://www.mongodb.com/docs. [Pokušaj pristupa 25 08 2024].

[6] A. S. Foundation, »CouchDB Dokumentacija,« 2024. [Mrežno]. Available: https://docs.couchdb.org/en/stable. [Pokušaj pristupa 25 08 2024].

[7] D. Sullivan, NoSQL for Mere Mortals, Michigan: Pearson, 2015.

[8] joke2k, »Faker Dokumentacija,« 2024. [Mrežno]. Available: https://pypi.org/project/Faker/. [Pokušaj pristupa 25 08 2024].

Skraćenice

ATM	*Asynchronous Transfer Mode*	asinkroni način prijenosa
SQL	Structured Query Language	jezik strukturiranih upita
NoSQL	Not Just SQL	ne samo SQL
MQL	MongoDB Query Language	MongoDB jezik upita
DBMS	Database Management System	sustav za upravljanje bazom podataka
CRM	Customer Relationship Management	sustav za upravljanje odnosa s klijentima
ACID	Atomicity, Consistency, Isolation, and Durability	atomarnost, konzistentnost, izolacija, trajnost
SSIS	SQL Server Integration Services	integracijske usluge SQL servera
SSAS	SQL Server Analitical Services	analitičke usluge SQL servera
SSRS	SQL Server Reporting Services	usluge izvještaja SQL servera
ETL	Extraction, Transformation, Loading	ekstrakcija, transformacija, učitavanje
CRM	Customer Relationship Management	upravljanje odnosima s klijentima
ERP	Enterprise Resource Planing	planiranje resursa tvrtke
JSON	JavaScript Object Notation	JavaScript objektna notacija
BSON	Binary JSON	Binarni JSON
CRUD	Create, Read, Update, Delete	Kreiraj, Čitaj, Ažuriraj, Obriši