Modeliranje ne relacijskih baza podataka

Tivanovac, Matija

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni preddiplomski studij računarstva

MODELIRANJE NERELACIJSKIH BAZA PODATAKA

Završni rad

Matija Tivanovac

Osijek, 2016.

SADRŽAJ

1. UVOD

Tema ovog završnog rada je usporediti NoSQL i SQL, tj. relacijske i nerelacijske baze podataka. Osnovni cilj i namjera je predstaviti podatke i informacije na jednom mjestu, osvijestiti o već poprilično razvijenim i rastućim alternativnim tehnologijama upravljanja bazom podataka i drugačijem načinu pristupanja modeliranju istih. Predstavljenu teoriju podupire primjer baze izrađen u relacijskom i nerelacijskom načinu modeliranja baze podataka.

Spomenuti primjer korišten je u svrhu usporedbe baza podataka te kako bi pokazao njihove razlike, prednosti i nedostatke. Prikazane informacije ne potiču na potpuno ostavljanje relacijskih baza, točnije SQL-a i isključivo korištenje NoSQL rješenja, već objašnjavaju situacije i vremena kada SQL jednostavno nije dovoljan, pa je potrebna druga tehnologija. Razvojem interneta takve situacije i projekti su sve češći i uključuju popularne i velike online trgovine, široko korištene društvene webstranice kao i neophodne webtražilice. Svi oni imaju potrebu za velikim bazama podataka, sa ogromnim brojem podatka, složenim atributima i vezama između podatka kao i ostale zahtijeve koje bolje, brže i učinkovitije obrađuju nerelacijske baze podataka stvorene upravo zbog takvih potreba. Za razliku od dominacije SQL-a, kod nerelacijskih baza ne postoji jedna koja bi zadovoljila sve potrebe različitih korisnika, već izraz „NoSQL“ obuhvaća jednu lepezu različitih tipova baza koje rješavaju specifične probleme za koje su projektirane.

Zbog spomenutog i nemogućnosti obrade svake nerelacijske baze podataka, u ovom radu dana je općenita teorijska osnova za usporedbu relacijsih i nerelacijskih baza, opće informacije o modeliranju takvih baza te pozadina vezana uz nastanak i potrebu za NoSQL rješenjima. Zatim slijedi konkretan primjer baze u obje verzije koji će biti prikazan postupno, od modeliranja i stvaranja baze, do unosa podataka i obrade istih.

1.1. Zadatak završnog rada

Zadatak ovog završnog rada je modeliranje ne relacijske baze podataka (tzv. NoSQL) i njezina usporedba s klasičnim SQL bazama.

2. TEORIJSKE OSNOVE

2.1. Nerelacijske (NoSQL) baze podataka

2.1.1. Povijest

U današnjem tehnološki naprednom svijetu sve se vrti oko podataka i njihove obrade. Oni su osnova infromacijskog svijeta, čak ih je moguće usporediti s naftom koliko su zapravo važni, tj. koliko im se pažnje pruža proporcionalno s napretkom tehnologije. Da bi imali koristi od podataka, potrebno ih je organizirati i obraditi, za što se koriste, naravno, baze podataka. Kada ljudi govore o bazama podataka najčešće misle na relacijske baze i relacijske sustave upravljanja bazom podataka (SUBP), kao i na SQL kao njihovog dominantnog predstavnika. Takav stav nije upitan ako se uzme u obzir da se gotovo sve baze jos od 1970-ih oslanjaju na relacijski model i SQL jezik koji im je omogućavao obavljanje opsežnih upita kada su im bili potrebni. Postojali su i drugačiji pristupi i razmišljanja, poput objektno orijentiranih baza podataka ili XML skladišta podataka, ali oni nisu nikada istinski zaživjeli jer su ih relacijski SUBP-i upili ili su ostali primjenjivi samo za specifične slučajeve.

Razvojem interneta, poticanjem korisnički stvaranih sadržaja nastala je potreba za preispitivanjem takvog razmišljanja da jedna veličina odgovara svima što je uzrokovalo nastanak raznolikih alternativnih baza podataka. Internet je sa sobom donio velika opterećenja, tj. promet na web stranice, posebice na one bitnije i popularnije poput Google-a, Ebay-a, Amazon-a i sličnih. Zbog toga su te kompanije morale pronaći bolja rješenja za pohranu podataka, pa se javljaju prve baze podataka koje nisu bile relacijske poput Bigtable (Google) i Dynamo (Amazon). Taj preokret i novonastale baze počele su se uobičajeno grupirati sa izrazom NoSQL koji se odnosi na uporabu nerelacijskih baza podataka, najčešće na području web aplikacija i projekata.

Za ime „NoSQL“ odgovoran je Carlo Strozzi koji je tako nazvao svoju relacijsku bazu otvorenog koda. Njegova baza je pohranjivala tablice kao ASCII datoteke i nije koristila SQL, već se bazom upravljalo pomoću skripti ljuske. Ali pravo prihvaćanje izraza NoSQL dogodilo se nakon sastanka i konferencije 2009. godine. Tada su predstavljene neke od popularnih nerelacijskih baza podataka zainteresirnim posjetiteljima. Organizator je htio da ime tog sastanka bude nešto kratko i pamtljivo kako bi se moglo podijeliti na društvenim mrežama, i taj izraz je ostao zapamćen od tada.

2.1.2. Opća obilježja

NoSQL baze podataka nemaju strogu definiciju. NoSQL baza podataka je dizajnirana za pohranu, distribuciju i pristup podatcima što čini pomoću metoda koje se razlikuju od relacijskih baza. Postoje zajednička obilježja koja se odnose na sve NoSQL baze podataka:

- Nisu relacijske – ne koriste relacijski model niti SQL jezik
- Većina je otvorenog koda (eng. Open-source)
- Vrlo dobro rade kao distribuirane baze – rad na više čvorova (računala) sa više baza
- Baze 21. stoljeća – „NoSQL“ se ne odnosi na baze koje su postojale prije SQL-a niti na objektno orijentirane baze podataka
- Nemaju shemu (eng. Schema-less)

Kada se kaže da ovakve baze podataka nemaju shemu, ta izjava nije u potpunosti točna. Sama baza možda nema strogo određenu shemu, ali to ne znači da ju aplikacija vezana uz bazu također nema. Dakle, postoji tzv. implicitna shema. S jedne strane možemo mijenjati bazu, dodavati i micati podatke po želji bez definiranja strukturnih promijena, ali opet moramo pratiti podatke i način na koji se njima upravlja i pronaći implicitnu shemu da bi mogli učinkovito manipulirati podatcima.

2.1.3. Model podataka

Model podataka nam pruža način na koji prikazujemo podatke i upravljamo njima. Kod baze podataka, on opisuje interakciju sa podatcima u bazi, odnosno organizaciju podataka u istoj. NoSQL baze možemo podijeliti, klasificirati upravo po tom podatkovnom modelu na četiri različite kategorije od kojih svaka ima svoja obilježja i zato svaka odgovara specifičnim zahtjevima i tipu projekta za koji se koriste:

- Model ključ-vrijednost (eng. Key-value) – jednostavan sustav koji pristupa vrijednostima preko ključa. Vrijednost je skupina koju baza pohranjuje bez ikakvog znanja o tim podatcima. Aplikacija je ta koja zna što je pohranjeno. Primjeri baza: Redis, Riak, Memcached, Project Voldemort, DynamoDB....
- Dokument baze (eng. Document) – pohrana hijerarhijskih struktura direktno u bazu. Dokumenti su kao vrijednost kod ključ-vrijednost modela. Zbog strukturiranih vrijednosti podataka, moguće je provesti upite nad dokumentom. Neke od baza: MongoDB, CouchDB, OrientDB, Terrastore....

- Model stupaca (eng. Column family) – strukturirani sustav koji koristi red i stupac kao ključeve. Strukture ima dvije razine: prva je red koji se sastoji od više detaljnijih vrijednosti. Te vrijednosti druge razine su stupci. Baze uključuju: Cassandra, Hbase, BigTable...
- Graf baze (eng. Graph) – kod grafova pohranjujemo entitete (čvorove) i odnose, veze između entiteta, tj. rubove. Čvorovi i rubovi imaju svoja svojstva. Rubovi imaju smjer, a čvorovi su podijeljeni prema vezama. Primjeri ovih baza: Neo4J, Infinite Graph, FlockDB...

NoSQL baze se mogu razvrstati na još jedan, općenitiji način. Ključ-vrijednost, dokument i baze s modelom stupca su agregatno orijentirane baze podataka. Agregat je skupina srodnih podataka kojom upravljamo kao cjelinom. Ove baze prilikom pohranjivanja ili čitanja podataka, upravljaju i svim podatcima vezanim uz te, jer su dio jednog agragata. Ova osobina omogućava spomenutim bazama horizontalnu skalabilnost, tj. povećanje resursa sustava povećanjem broja čvorova, računala. Budući da je cijeli agregat pohranjen na jednom čvoru, distribucija je uvelike olakšana, jer ako želimo prikupiti podatke iz tablice, dovoljno je jednim pozivom dohvatiti cijeli agregat koji sadrži sve povezane podatke. Ovdje je jako bitno znati kako će podatke koristiti aplikacija i prema tome formirati agregate. S druge strane graf baze ne koriste agregate, umjesto toga one imaju jako male čvorove koji često sadrže samo jedan podatak, ali ti čvorovi su međusobno povezani sa složenom mrežom rubova koji predstavljaju odnose između čvorova. Graf baze posjeduju upitni jezik koji im omogućuje navigaciju kroz graf. Ovakve baze bolje funkcioniraju na jednom serveru nego na distribuiranom sustavu.

2.1.4. Konzistentnost

Jedna od temeljnjih obilježja baze podataka je njena konzistentnost, tj. baza podataka mora biti dosljedna, podatci u cijeloj bazi moraju biti pravilni i točni što se odnosi i na svaku kopiju te baze kako bi svi korisnici vidjeli iste, ažurirane podatke. Ovo je posebno važno za distribuirane baze podataka, jer se mora osigurati dosljednost u svim čvorovima. Kad se radi o distribuiranim NoSQL bazama važni pojmovi su CAP teorem i „eventualna dosljednost“. NoSQL baze imaju transakcije, štoviše, graf baze koriste ACID transakcije, dok se agregatno orijentirane baze ponašaju drugačije. One imaju transakcijske granice, ograničenja. Atomizirana ažuriranja postoje, ali samo unutar istog agregata. Nije moguće manipulirati sa dva agregata unutar jedne transakcije. Promijena koja utječe na dva ili više agregata donosi dio vremena kada podatci nisu konzistentni.

Taj „prozor nedosljednosti“ će s vremenom nestati nakon što se ažuriraju svi čvorovi – eventualna dosljednost. Kada se spominju transakcije, često se kaže da NoSQL baze podataka imaju BASE svojstva transakcija. Takav model određuje da sustav može biti privremeno nedosljedan kako bi se upravljalo transakcijama (eng. Basically Available), da se stanje sustava stalno mijenja (eng. Soft state) i da će u konačnci, kada se obave promjene, sustav poprimiti konzistentno stanje (Eventual consistency). ACID i BASE koncepte moramo promatrati ne kao međusobno isključujuće opcije već kao spektar. BASE koncept se često spominje uz još jedan važniji, pojam, a to je CAP teorem. On može pojasniti zasto je nekad potrebno žrtvovati konzistentnost radi ostalih svojstva. CAP teorem podrazumijeva tri svojstva:

- Dosljednost (eng. Consistency) – svi čvorovi imaju identične kopije podataka dostupnih za transakcije
- Dostupnost (eng. Availability) – svaki zahtjev za podatke će se obraditi uspješno ili će postojati poruka da se zahtjev ne može obaviti
- Tolerancija particija (eng. Partition tolerance) – sustav će nastaviti raditi i u slučaju da se prekine veza između čvorova što bi stvorilo particije u kojima čvorovi mogu komunicirati međusobno samo unutar vlastite particije

CAP teorem govori da od prethodno navedenih svojstava možemo imati samo 2, jedan se mora žrtvovati. Sustav sa jednim čvorom je očit primjer CA sustava. On ima svojstva konzistentnosti i dostupnosti, ali ne tolerira particije. Budući da jedno računalo ne može podijeliti podatke, ne mora se niti brinuti za nošenje sa particijama. Kada se radi o distribuiranom sustavu mora postojati tolerancija particija. To je pravi značaj ovog teorema. Iako se radi o izboru o dva svojstva, praktično se misli da ako postoji sustav kojem se mogu dogoditi particije (distribuirani sustavi), onda se mora birati između dostupnosti i dosljednosti. Ovo također nije isključiva odluka, jer uvijek se može odabrati određeni stupanj dostupnosti i određeni stupanj dosljednosti, tako da sustav nije niti u jednoj krajnosti zbog čega djelomično ima oba svojstva. Bilo bi još bolje na to gledati kao omjer konzistentnosti i kašnjenja. Konzistentnost možemo smanjiti povećanjem čvorova koji sudjeluju u interakciji, ali svaki dodatni čvor povećava vrijeme odziva te interakcije. Dostupnost može biti oznaka za maksimalno vrijeme odziva koje ćemo tolerirati, pa ako ono postane preveliko, smatramo da podatci nisu dostupni.

2.2. Relacijske (SQL) baze podataka

2.2.1. Općenito i model podataka

Relacijske baze podataka sadrže i upravljaju relacijski strukturiranim podatcima, te posjeduju sustav za manipulaciju istih. Podatke pohranjuju u dvodimenzionalne tablice koje se sastoje od redova i stupaca. Interakcija sa sustavom za upravljanje relacijskim bazama se najčešće ostvaruje korištenjem upitnog jezika SQL. Tip podataka koji se mogu pohraniti određuje sustav, a podatci su organizirani prema eksplicitno određenoj shemi koja je tipično bazirana na ER modelu. Relacijski model baze i SQL kao standardni upitni jezik su široko prihvaćeni i dugo ih se smatralo kao jedinom alternativom pohrane i organiziranja podataka kojoj može pristupiti više korisnika na konzistentan način.

Najvažnija obilježja relacijskih baza podataka:

- Perzistencija (eng. Persistence) – trajna pohrana padataka. Baza pohranjuje podatke na tvrdi disk ili slični medij koji ne gubi podatke nakon što mu se prekine napajanje
- Integracija (eng. Integration) – integracija između aplikacija preko dijeljene baze podataka
- Standardni jezik i tip baze – nakon dugogodišnjeg postojanja, relacijski model kao i SQL jezik koji se koristi za interakciju s bazom, su postali svima poznati i svojevrsni standard u tom području
- Višekorisnički rad (eng. Concurrency) – velik broj korisnika pristupa istim podatcima u bazi istovremeno, zbog čega baze podataka podržavaju transakcije za održavanje konzistentnosti
- Prikaz podataka, izvještaj (eng. Reporting) – jednostavan relacijski model i standardni SQL osnova su alata za izvještavanje

Budući da postoji shema i struktura podataka, potrebno je istu i poštovati. To znači razvrstati podatke u tablice po redovima (n-torkama) i stupcima (atributima). Svaki presjek stupca i retka ima samo jednu vrijednost i svaka varijabla ima svoj tip podatka od kojih svaki ima ograničenu domenu dopuštenih vrijednosti. Primarni ključevi se postavljaju kako bi se mogao identificirati svaki redak u tablici. Da bi pristupili podatcima, trebalo bi biti dovoljno imati samo tri informacije: ime tablice, ime stupca i primarni ključ retka. Strani ključevi se koriste kako bi međusobno povezali više tablica, a moguće je i obavljati matematičke operacije nad relacijama. Takva struktura omogućuje bazi podataka da zna kakvi podatci su pohranjeni. Tako strukturirani podatci omogućavaju kompleksne upite nad njima, kao i upite između različitih baza zbog stranih

ključeva. Također zbog ovakve strukture, relacijske baze podataka su vertikalno skalabilne. To znači da najbolje funkcioniraju na jednom čvoru, računalu. Ako je potrebno rasti zbog količine podataka, onda je najbolja opcija povećanje resursa tog čvora.

2.2.2. Integritet i sigurnost

Definirana shema omogućava bazi provjeravanje interakcije sa podatcima i kontrolu integriteta baze. Koriste se razna integritetska ograničenja koja se provjeravaju pri svakoj operaciji. Ograničenja se mogu stvoriti po potrebi, a postoje i već definirana osnovna ograničenja koja omogućuju normalno funkcioniranje baze. Osim integriteta bitna je i sigurnost baze. Baza podataka omogućuje autentifikaciju korisnika preko korisničkog imena i lozinke, a ovlasti dodijeljene svakom korisniku ga ograničavaju na njemu dopuštene akcije. Sigurnost je omogućena i postavljanjem pogleda, koji prikazuju korisniku samo one dijelove baze za koje ima pristup, kao i ograničeno upravljanje tim dijelovima baze podataka.

2.2.3. Konzistentnost

Konzistentnost podataka u bazi i kontrola višekorisničkog rada postiže se korištenjem transakcija. Transakcije osiguravaju da svi korisnici vide točne i ažurirane podatke. Transakcije u relacijskim bazama podataka imaju ACID osobine:

- Atomiziranost (eng. Atomicity) – znači da se transakcije moraju obaviti u cjelosti ili se ne smiju obaviti uopće. Ako je transakcije djelomično obavila posao i zaustavila se, onda se sve promijene poništavaju
- Konzistentnost (eng. Consistency) – transakcije su primarno sredstvo za održavanja dosljednosti baze podataka
- Izoliranost (eng. Isolation) – transakcije nisu ovisne jedna o drugoj, ne znaju što druge transakcije rade i postoje li uopće, čak i kada se izvršavaju istovremeno. Njihov rezultat uvijek izgleda kao da su se izvršavale slijedno
- Trajnost (eng. Durability) – odnosi se na činjenicu da su sve promjene trajne nakon što su svi dijelovi transakcije završili.

ACID transakcije omogućuju relacijskoj bazi podataka dosljednost i dostupnost, budući da najbolje rade na jednom čvoru, pa se tolerancija particija zanemaruje.

2.3. Redis

2.3.1. Općenito

Redis (eng. **RE**mote **DI**ctionary **S**ervice) je baza podatkovnih struktura (eng. data structure store), otvorenog koda (BSD licenciran) koja se u potpunosti nalazi u radnoj memoriji zbog čega ima visoke performance, podržava replikaciju i pohranu podataka na disk, te pruža jedinstveni model podataka za rješavanje problema. Osim što služi kao baza, redis je i red (stog) sa mogućnošću blokiranja, sustav za objavljivanje i pretplatu, sadrži opcije za postavljanje vremena isticanja podataka te različite razine postojanosti uz velike brzine: čak i preko 100k pisanja po sekundi i preko 80k čitanja podataka po sekundi.

Projekt Redis je započeo Salvatore Sanfilipo kako bi poboljšao svoj proizvod za web analizu. Nakon što je redis postao stabilan, porasla mu je popularnost, a Salvatorea je zaposlila tvrtka VMWare kako bi radio na redisu.

Redis organizira podatke po modelu ključa i vrijednosti koji se pohranjuju u RAM. To mu omogućava spomenutu brzinu, ali ga zato ograničava na veličinu radne memorije, zbog čega se ne koristi često kao glavna baza podataka, već kako bi ubrzao postojeći sustav baze. Za razliku od drugih baza, redis podržava složenije podatkovne strukture koje više odgovaraju načinu na koji aplikacije koriste podatke, što olakšava predodžbu i korištenje podataka u višim slojevima aplikacija. Redis nema upitni jezik, sva čitanja podataka se obavljaju direktno preko naredbi, zbog čega je jako važno dobro dizajnirati cijeli model baze i predvidjeti sva pristupanja podatcima prije samog stvaranja baze podataka.

2.3.2. Arhitektura i model podataka

Glavni procesi od kojih se sastoji Redis su njegov klijent i server koji mogu biti na istom ili različitim računalima. Server pohranjuje podatke, a preko klijenta se upisuju naredbe. Klijent može biti standardna konzola ili bilo koji drugi Redis API ostalih programskih jezika.

Redis je u svojoj osnovi ipak ključ-vrijednost baza, što znači da se i sve njegove strukture sastoje od dvije stvari: vrijednosti koja je nestrukturirana skupina podataka i ključa preko koje pristupamo vrijednostima. Ključevi su binarno sigurni, što znači da se bilo koji binarni niz podataka može koristiti kao ključ. Pametno bi bilo odabrati nekakav uzorak za postavljanje ključeva, primjerice: tip_objekta:id.

Tipovi podataka u Redisu:

- String – najjednostavniji tip podataka u Redisu. Mogu se pohraniti svi tipovi stringova, a ograničenje duljine je 512 MB
- Lista – niz stringova organiziranih kao povezani popis. To znači da se elementi u listu mogu dodavati na kraj i početak brzo ili sporije preko indeksa. Koriste se najčešće kao redovi ili stogovi
- Hash tablica – sastoje se od nekoliko polja sa pripadajućim vrijednostima i sve je spremljeno pod jednim ključem. Najčešće se koriste za pohranu složenih objekata gdje polja predstavljaju atribute
- Set – set je jako sličan listi. Razlika je u tome što su svi članovi seta jedinstveni i nisu ogranizirani kao povezani popis. Često se koriste za pohranu ključeva za stvaranje indeksa
- Poredani set – set jedinstvenih članova, ali svaki član ima pridodanu brojčanu vrijednost koja može biti i decimalna i po kojoj su članovi poredani

2.3.3. Ostala obilježja

Pohrana podataka – iako je redis baza „u memoriji“, postoje tri načina za pohranu podataka na disk. Asinkrono spremanje „slike“ baze u obliku RDB datoteke. Spremannje obavlja baza automatski tijekom određenih vremenskih intervala ili nakon određenog broja naredbi. AOF (eng. append only file) pohranjuje sve naredbe koje mijenjaju podatke u bazi. Svaka promjena se bilježi u međuspremniku i u intervalima sinkronizira na disk. Također je moguće ručno pohraniti podatke u RDB datoteku pomoću naredbe SAVE.

Replikacija – Redis podržava replikaciju koristeći master-slave servere. Slave ili sporedni serveri su kopije glavnih, master servera. Sinkronizacija podataka između servera odvija se asinkrono u pozadini. Replikacija se koristi zbog proširenja mogućnosti ili bolje organizacije, kao primjerice, sa serverima koji obavljaju samo čitanje podataka.

Podijela baze (eng. sharding) – pruža mogućnost horizontalne skalabilnosti time da se baza razlomi na više dijelova koji se prošire na druge servere čime se oslobađa memorija i povećavaju performanse baze podataka. Sustav koji to omogućava zove se Redis Cluster i podržava proširenje na 1000 čvorova.

2.4. MongoDB

2.4.1. Općenito

MongoDB (od riječi eng. hu**mongou**s) je baza podataka otvorenog koda koja koristi model dokumenta kako bi organizirala podatke. Pruža visoke performance, dostupnost i automatsku skalabilnost. Mongo nema točno definiranu shemu i napisan je u C++. Prvi put objavljen 2009. mongo je brzo postao popularan zbog svoje dokument strukture i mogućnosti upita. Dokument je zamijenio red u tablici, zbog čega je moguće prikazati složene hijerarhijske odnose sa jednim dokumentom. Vrijednosti i ključevi doumenata nisu ograničeni veličinom, a budući da nema sheme, dodavanje i brisanje podataka je uvelike olakšano

2.4.2. Model podataka

Dokument je osnovni tip podatka i sličan je redu kod relacijskih baza podataka. To je uređeni set ključeva sa pridruženim vrijednostima. Svaki dokument ima poseban ključ _id koje je jedinstven unutar kolekcije. Dokument je pohranjen kao binarni prikaz BSON (Binary JSON). Svaki dokument se sastoji od polja, a polje sadrži vrijednost određenog tipa, što uključuje polja, binarne podatke i pod-dokumente. Dokumenti su organizirani u kolekcije koje nemaju shemu, što znači da svaki dokument u kolekciji može imati drugačiju strukturu. Iako nema shemu, ipak bi trebalo paziti kako se podatci organiziraju. Objekti bi se trebali staviti u jedan dokument ako će se koristiti zajedno, podatci se mogu duplicirati, jer se baza pohranjuje na disk i preporučljivo je koristiti agregacije kako bi se podatci bolje grupirali.

2.4.3. Ostala obilježja

Ad hoc upiti – odnose se na upite koji se mogu provesti nad bazom bez prijašnjeg razmišljanja kakve upite će baza prihvatiti. Mogu se obaviti svi upiti sa dobro postavljenim uvjetima. MongoDB upiti pretpostavljaju da su oznake podataka pohranjene unutar svakog dokumenta. Upiti mogu vratiti podskup dokumenata ili cijelu kolekciju

Sekundarni indeksi – ako provedemo upit bez postojećih indeksa u bazi, zapravo pretražujemo cijeli skup podataka. Sekundarne indekse postavljamo mi, dok _id polja koja služe kao jedinstveni indeksi postavlja sama baza automatski. Indeksi su implementirani kao B stable i mogu se postaviti do 64 indeksa po kolekciji

Replikacija – sustav za replikaciju MongoDB baza zove se replica set. On distribuira podatke na više računala radi boljeg upravljanja nepotrebnim viškom podataka ili zbog bolje sigurnosti u slučaju prestanka rada servera. Set kopija se sastoji od jednog glavnog čvora i više sekundarnih čvorova. Posebnost kod MongoDB baze je u tome što će u slučaju prestanka rada servera, baza automatski odabrati sekundarni čvor i unaprijediti ga u primarni.

Skalabilnost – MongoDB je napravljan kako bi olakšao horizontalnu skalabilnost. Koristi sustav auto-sharding koji automatski upravlja distribucijom podataka po čvorovima. Automatski dodaje čvorove na koje možemo podijeliti bazu i osigurava nastavak rada u slučaju kvara servera bez pogreške. Takav sustav olakšava rad aplikacije, jer sam upravlja distribucijom i osigurava ju, što znači da aplikacija to ne mora raditi.

3. PRIMJER BAZE PODATAKA

3.1. Redis model Rent a car baze

Baza podataka se sastoji od tri skupine podataka: automobila, klijenata i ugovora. Budući da se koristi Redis baza podataka koja ne podržava sustav upita potrebno je dobro isplanirati model baze kako bi ju mogli efikasno pretraživati. Automobile možemo prikazati kao objekte sa atributima. Isti princip se primjenjuje na klijente kao i ugovore. Za prikaz objekata koristi se Redis hash tablica. U polja tablice se unose atributi i pridružuje im se pripadna vrijednost. Cijela tablica se sprema pod ključem koji je string i predstavlja ime proizvođača i tip automobila, npr. „renault_clio“. Kod klijenata ključ predstavlja tip objekta i njegov identifikacijski broj, npr. „klijent:135628“. Kod ugovora naziv ključa predstavlja tip objekta i prezime klijenta s kojim je ugovor sklopljen radi lakšeg pretraživanja baze i veze sa klijentima, kao npr. „ugovor:markovich“.

Pretraživanje podataka vezanih uz automobile će se provesti na principu povratnih indeksa (eng. reverse index search). Ako pretražujemo bez ovakvog sustava imali bi O(n) operaciju koja bi prošla po cijeloj bazi kako bi pronašla odgovarajući podatak. Takav postupak nije učinkovit, ali što je još bitnije, nije siguran. Ključeve bi pretraživali sa KEYS pattern naredbom. Gdje pod argument „pattern“ možemo staviti bilo koju riječ ili dio riječi ili slovo i pretraživati po takvom uzorku. Problem te naredbe je u tome što je nju opasno izvoditi na veliki bazama podataka, posebice ako imamo distribuiranu bazu. Takvo pretraživanje može prilično usporiti bazu ili ju čak blokirati na neko vrijeme dok traži ključeve, prilikom čega baza ne bi bila dostupna korisnicima. Povratnim pretraživanjem indeksa rješavamo taj problem. Redis ima tip podataka pogodan za stvaranje indeksa – setove. Set je niz elemenata koji se ne ponavljaju. U ovom slučaju zapravo je korišten sorted set. Sorted set nam, za razliku od seta, omogućuje da uz elemente seta unesemo i brojčane vrijednosti vezane uz njih. Dakle, u sorted set unijet ćemo jedinstvene ključeve svakog automobila i svakom ključu ćemo pridružiti njegovu cijenu automobila. Naravno, kada bi unosili svaki automobil u jedan set, opet bi imali previše ključeva za pretražiti, zbog čega je stvoreno više poredanih setova od kojih svaki predstavlja tip automobila. Znači postoje „coupe“, „SUV“, „karavan“ i slični sorted setovi. Takvim postupkom dobivamo uvid u automobile grupirane po tipu i poredane po cijeni i možemo svakom pristupiti pretraživanjem po ključu što je O(1) operacija.

Kako bi pretraživanje automobila bilo još jednostavnije i detaljnije napravljeni su dodatni setovi. Set koji predstavlja popis svi mogućih vrsta automobila u rent a car tvrtci -

„auto_vrste“. Svaki element u tom setu je identifikacijski ključ seta za svaki tip automobila. Također su stvoreni setovi koji predstavljaju popis automobila grupiranih po državama iz kojih potječu proizvođači. To su „italija:auto“, „njemacka:auto“ i slični. U tim setovima elementi su ključevi koji predstavljaju modele sutomobila. Pohranjen je i set koji sadrži ključeve svih tih setova. To je set „drzave“.

Za pretraživanje klijenata također je stvoren set sa indeksima, tj ključevima koji predstavljaju pojedinog klijenta i njegove podatke. Najbitnije svojstvo po kojem bi pretraživali klijente je činjenica jesu li platili posudbu ili ne. Zbog toga postoje dva seta „platio:da“ i „platio:ne“ koji sadrže ključeve svakog klijenta za kojeg odgovaraju te tvrdnje.

Podatci vezani za automobile su šifra koja je zapravo ime modela automobila radi bolje preglednosti podataka prilikom pretraživanja, zatim imamo registarsku oznaku kao i datum registracije. Registracijska oznaka je potrebna zbog lakšeg traženja automobila i razlikovanja od sličnih vozila, dok datum registracije govori o tome kad je automobil registriran i koliko vremena ima do sljedeće registracije koja je uvjetovana zakonom. Sljedeća svojstva su opisna, a to su boja automobila, broj sjedala, snaga motora, vrsta motora, godina proizvodnje i vrsta koja je bitna za pretraživanje.

Hash tablica za klijente sadrži ime i prezime klijenta, spol, trenutnu adresu, kontaktni broj telefona te podatak o plaćanju.

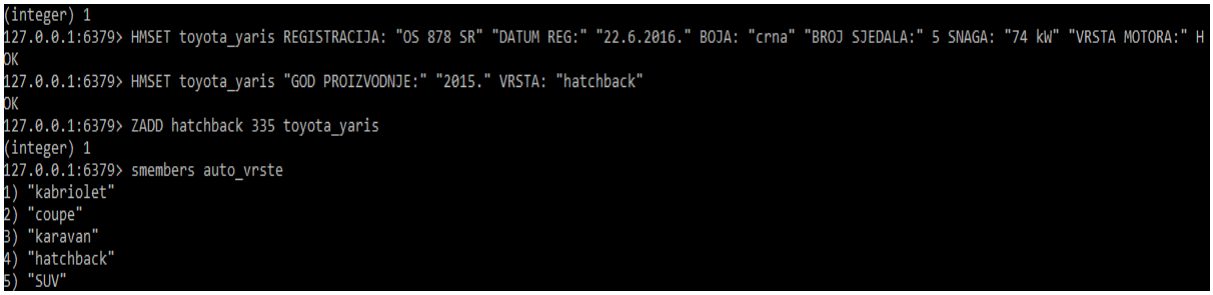
Hash tablica za ugovore sadrži bitne atribute za ugovor. To su datumi iznajmljivanja i vraćanja, ukupnu cijenu ovisno o trajanju posudbe, identifikaciju klijenta sa kojim je sklopljen ugovor kao i informacija o tome koji automobil je posuđen.

Redis baza podataka se pokreće tako da se prvo preko komandnog sučelja pokrene Redis-server.exe, a zatim redis-cli. Redis automatski stvori bazu i možemo odmah početi unositi podatke. Za navedene postupke u daljnjem tekstu sintaksu je moguće vidjeti u priloženom kodu (Prilog [1]). Prvo kreiramo hash tablice za svaki automobil kako bi pohranili njihove podatke. To radimo naredbom „HMSET key field value“. Ovom naredbom moguće je unijeti više vrijednosti u jednu hash tablicu odjednom. Nakon svakog pohranjenog automobila, odmah ga unesemo i u poredani set naredbom „ZADD key score member“. Gdje je „key“ tip automobila, „score“ cijena, a „member“ ključ hash tablice automobila.

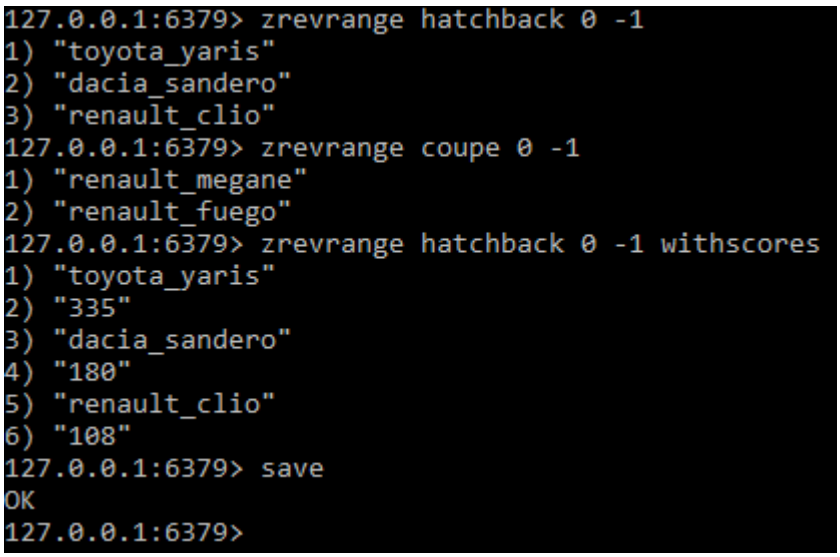
Set svih vrsta automobila stvaramo naredbom „SADD key member“, gdje je „key“ ime seta, „auto_vrste“, a „member“ je vrsta automobila. Zatim na isti način napravimo setove vezane uz pojedine države kao i set koji sadrži popis svih država u bazi.

Podatke o klijentima i ugovorima unosimo u hash tablicu na isti način kao i podatke o automobilima. Zatim stvaramo dva seta koji sadrže ključeve klijenata koji su platili, odnosno, nisu platili iznajmljivanje automobila.

Ispis članova seta vrši se naredbom „SMEMBERS key“. Primjerice, ako upišemo „auto_vrste“ kao ključ, dobit ćemo popis svih vrsta automobila(Sl.3.1.). Zatim možemo pretražiti jedan od sortiranih setova naredbom „ZREVRANGE key start stop WITHSCORES“ upisivanjem jedne od vrsta automobila koju nam je dao prije navedeni set kao ključ. Da bi dobili cijeli popis za početak i kraj navedemo 0 i -1 kao na slici 3.2. koji predstavljaju indekse prvog i krajnjeg elementa u sortiranom setu.



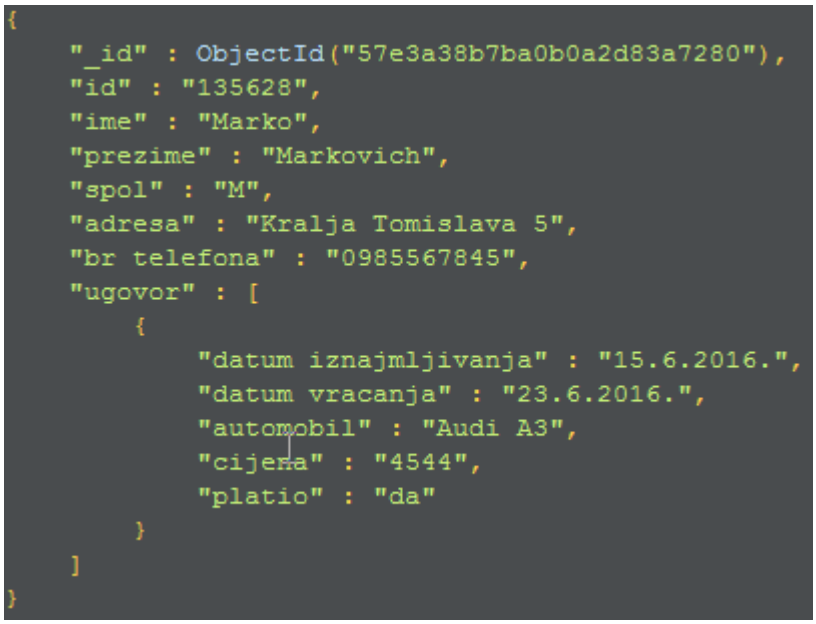
Sl. 3.1. Unos podataka za automobil, stavljanje istog u set i ispis vrsta



Sl. 3.2. Ispis automobila grupiranih po vrsti i poredanih po cijeni

3.2. MongoDB model Rent a car baze

Da bi projektirali istu bazu u MongoDB moramo ju malo drugačije organizirati. Baza ima dvije kolekcije: auto i klijenti. Zbog strukture dokumenta i tipova podataka koje podržava, cijeli objekt „ugovor“ moguće je umetnuti u dokumente kolekcije klijenti. Podatci o ugovorima su sada umetnuti dokument u polju kod ključa „ugovor“. Baza sada izgleda preglednije, jer kod podataka o klijentima imamo odmah i uvid u ugovore koji su s njima sklopljeni(Sl.3.3.).



Sl. 3.3. Dokument sa podatcima o klijentu i ugovoru

Modeliranje baze u MongoDB ne zahtijeva toliko planiranja kao kod Redis baza. Mongo podržava ad hoc upite, što znači da je dovoljno stvoriti dokumente i kolekcije, a zatim ih je moguće pretraživati upitima bez potrebe za stvaranjem dodatnih struktura. Kod modeliranja je ipak poželjno paziti na strukturu i organiziranje dokumenata, što znači, spojiti one objekte koji su slični i povezani jedni s drugima, kao na slici 3.3. ili ih barem staviti u istu kolekciju. Mongo ne podržava JOIN operacije, pa se kasnije ne mogu spojiti dokumenti bez prepravljanja cijelog sadržaja. Također treba paziti na tip podataka koji se unosi u dokumente, npr. string, broj, datum ovisno o željenim upitima i agregacijama kasnije.

Kolekcije auto i klijenti sadrže iste atribute kao i prilikom pohranjivanja u Redis bazu, jedino automobili imaju malo izmijenjen „id“, koji je sada samo model automobila. No to nije ni važno kada mongo automatski dodaje svoje jedinstveno „_id“ polje.

MongoDB pokrećemo slično kao i Redis, iz komandnog sučelja, pokretanjem servera i sučelja za upis naredbi. Budući da Mongo ima kvalitetnih grafičkih sučelja, korišteno je Robomongo

sučelje za interakciju s bazom. Sintaksa sljedećih naredbi može se detaljno vidjeti u prlogu (Prilog [2]) Bazu možemo stvoriti naredbom „use database_name“ koja će provjeriti postoji li baza sa navedenim imenom i ako ne postoji, stvoriti će bazu. Također, naredbom „db.collection_name.insert()“ moguće je stvoriti kolekciju ako već ne postoji. Ako postoji, onda će se unijeti dokument u tu kolekciju. Unutar zagrade „insert“ naredbe unosimo podatke u dokument.

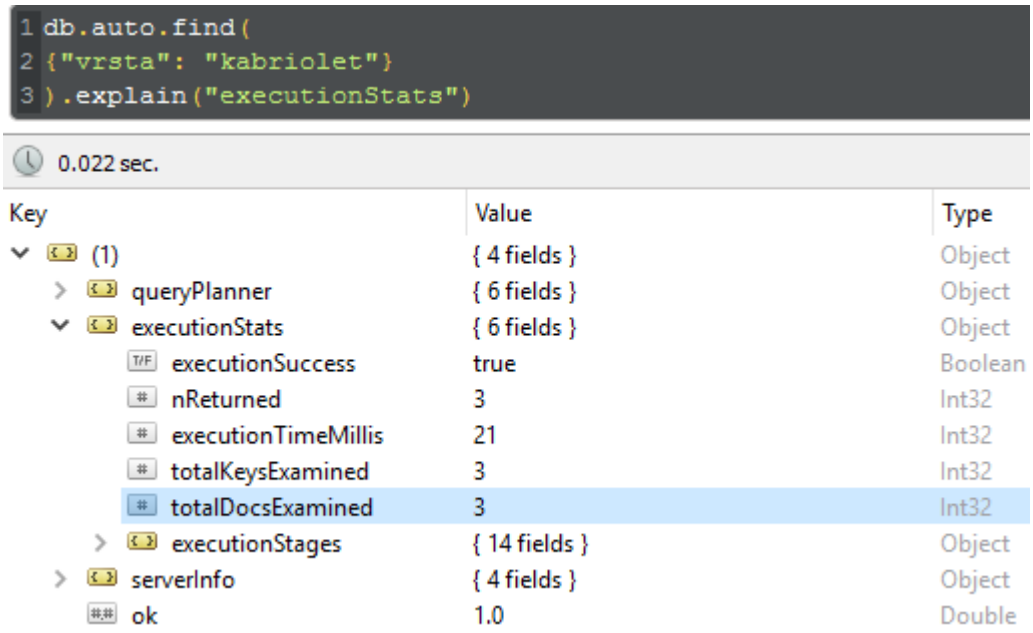
Upitni jezik kod MongoDB radi tako da mu sa naredbom „db.collection_name.find()“ predamo argumente koje treba tražiti i on onda pretražuje cijelu bazu kako bi našao odgovarajuće dokumente, te vraća natrag cijele dokumente, osim ako nije određeno drugačije. Iako je Mongo jako brz, nije poželjno da se nepotrebno koriste resursi i vrijeme da bi se pronašlo nešto. Zato je dobro koristiti indekse. Indeksi se stvaraju za one atribute koji se često pretražuju. Primjerice, stvoren je indeks za atribut „vrsta automobila“. Pretraživanjem baze po vrsti automobila prije stvaranja indeksa, Mongo mora proći kroz cijelu kolekciju(Sl.3.4.). Nakon stvaranja indeksa i ponovnog pretraživanja, vidljivo je da je traženje skraćeno(Sl.3.5.).

```
1 db.auto.find(
2 {"vrsta": "kabriolet"}
3 ).explain("executionStats")
```

0.016 sec.

Key	Value	Type
(1)	{ 4 fields }	Object
queryPlanner	{ 6 fields }	Object
executionStats	{ 6 fields }	Object
executionSuccess	true	Boolean
nReturned	3	Int32
executionTimeMillis	10	Int32
totalKeysExamined	0	Int32
totalDocsExamined	15	Int32
executionStages	{ 14 fields }	Object
serverInfo	{ 4 fields }	Object
ok	1.0	Double

Sl. 3.4. Pretraživanje prije indeksiranja



Sl.3.5. Pretraživanje poslije indeksiranja

Ako želimo grupirati dokumente i vršiti operacije nad njima poput zbrajanja, minimalne i maksimalne vrijednosti, te prosječne vrijednosti, tada to možemo napraviti pomoću agregatnih funkcija. Stvorena je agragatna funkcija nad kolekcijom „auto“ koja grupira automobile po vrsti i vraća najskuplji automobil od svake vrste(Sl.3.6.)

```
db.auto.aggregate(
{
    $group: {
        _id: "$vrsta",
        "najskuplji": {$max: "$cijena"}
    }
}
)
```

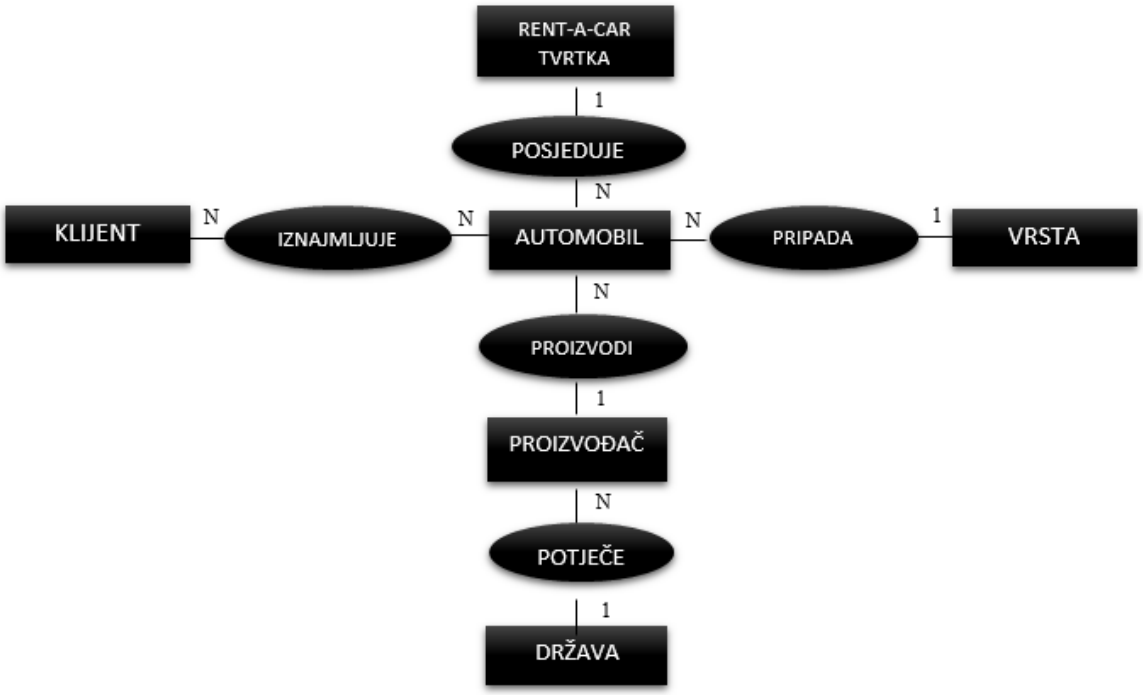
0 sec.

```
/* 1 */
{
    "_id" : "hatchback",
    "najskuplji" : 335.0
}

/* 2 */
{
    "_id" : "karavan",
    "najskuplji" : 210
}

/* 3 */
{
    "_id" : "SUV",
    "najskuplji" : 825
}

/* 4 */
{
    "_id" : "coupe",
    "najskuplji" : 145.0
}

/* 5 */
{
    "_id" : "kabriolet",
    "najskuplji" : 650.0
```

Sl. 3.6. Najskuplji automobil po vrsti automobila

3.3. SQL model Rent a car baze

Da bi napravili relacijski model baze podataka i napisali ju u SQL-u, potrebno je prvo projektirati eng. Entity-Relationship (ER) dijagram te baze(Sl.3.7.).



Sl. 3.7. ER model Rent a car baze podataka

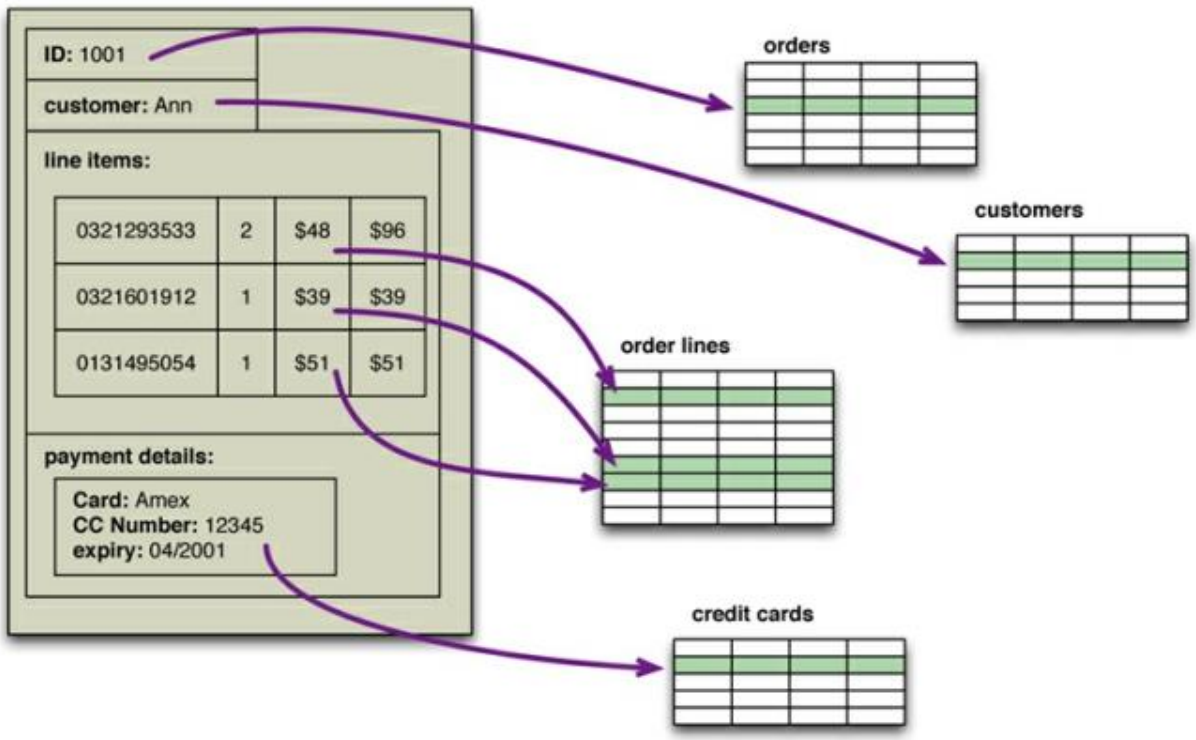
Tab. 3.1. Atributi entiteta i veza za dijagram sa slike 3.7.

Entitet/Veza	Atributi
Klijent	šifra_klijenta, ime, prezime, spol, broj telefona, platio, adresa
Iznajmljivanje	šifra ugovora, datum iznajmljivanja, datum vraćanja, cijena
Automobil	šifra_automobila, registracijska_oznaka, snaga motora, vrsta_motora, boja, broj_sjedala, datum_registracije, godina_proizvodnje
Vrsta	šifra_vrste, tip
Proizvođač	šifra_proizvođača, naziv
Rent-a-car tvrtka	šifra_tvrtke, naziv tvrtke, adresa
Država	šifra_države, naziv

Iz prikazanog dijagrama sa slike 3.7. i prema podatcima iz tablice 3.1. očito je da je model ovakve relacijske baze prilično drugačiji od modela u Redis i Mongo bazama. Vidljive su veze koje imaju vlastite atribute kao i nedostatak objekta/entiteta ugovora, te činjenica da su proizvođač i vrsta automobila odvojeni u posebne tablice. Ovakav model gore prikazane baze je moguć i potreban u relacijskim bazama zato što one moraju biti normalizirane. Znači da ne smiju postojati duplicirani podatci zbog čega su neki entiteti morali biti podijeljeni u zasebne tablice. Korištenjem stranih ključeva, moguće je povezati entitete preko veza/relacija što direktno utječe na mogućnosti provođenja složenih upita nad tom bazom. Sintaksa stvaranja SQL baze vidljiva je u prilogu. (Prilog [3])

4. USPOREDBA MODELA NERELACIJSKIH (NOSQL) I RELACIJSKE BAZE PODATAKA

Za usporedbu modela koriste se baze podataka Rent a car tvrtke stvorene u Redis, MongoDB i SQL bazama podataka. Model baze podatak je prvi i odmah uočljiv pokazatelj koliko su relacijske i nerelacijske baze podataka različite. Slika 3.7. i tablica 3.1. pokazuju standardni način modeliranja relacijske baze podataka. Relacijske baze zahtijevaju da se svaka skupina podataka razdvoji u više tablica ili relacija. Tada se podatci mogu normalizirati, povezati stranim ključevima i pripremiti za detaljne upite i analize podataka. Problem relacijskog modela je u tome što se on gotovo uvijek razlikuje on načina na koji želimo prikazati podatke u aplikaciji ili memoriji. U memoriji i aplikacijama najčešće prikazujemo složenije strukture, primjerice hijerarhijski strukturirane podatke. Relacijske modele je zato uvijek potrebno mapirati u pomoću same aplikacije da bi ih prikazali na način na koji želimo i koji je prikladniji. Ovakav problem kada se model baze razlikuje od prikaza u memoriji ili aplikaciji zove se neusklađenost impedancija (impedancija baze i aplikacije), eng. impedance mismatch i prikazan je na slici 4.1.



Sl. 4.1. Neusklađenost impedancija [1]

Neusklađenost prikaza podataka u memoriji i u bazi se ne može ukloniti zato što relacijske baze podataka imaju čvrsto određenu shemu po kojoj se modeliraju. Izgled same sheme mora se definirati prije stvaranja baze i moramo ga se pridržavati tijekom cjelokupnog modeliranja baze podataka. To znači da nije lako mijenjati izgled baze jednom kad ju popunimo podatcima i integriramo s aplikacijom.

NoSQL baze podatka nemaju strogo definiranu shemu. Na primjeru Redis i MongoDB baza vidjeljivo je da je to svojstvo točno. U obje baze podataka možemo bez problema dodavati podatke, ali ih i brisati bez da vodimo računa o dodavanju stupaca, brisanju redaka, povezivanju baza itd. Redis je baza po modelu ključ-vrijednost, a MongoDB ima gotovo sličan princip, samo što su njegovi ključevi i pripadajuće vrijednosti organizirani u dokumente. Ključevi su jedinstveni, Mongo čak i sam automatski stvara identifikacijske ključeve u dokumentima, a vrijednosti mogu biti bilo kakav tip podataka, posebice kod redisa. U Redis bazi najbitniji je ključ, bazu ne zanima niti zna što je pohranjeno kao vrijednost, što znači da je tu vrijednost uvijek moguće mijenjati. Sličnost u modelu podataka između ove dvije nerelacijske baze ju u agregatnoj orijentiranosti. Relacijske baze manipuliraju povezanim podatcima pohranjenim u relacije, dok Redis i Mongo vrše interakciju sa agregatima. Agregat je skup srodnih podataka koji se tretiraju kao cjelina. Agregat je prirodnija struktura podataka, koja se lakše može predočiti u memoriji ili prikazati na sučelju aplikacije. Unatoč tome što nemaju shemu i koriste agregate, NoSQL baze podataka nisu toliko dinamične. Nerelacijske baze podataka imaju tzv. implicitnu shemu. Nemoguće je raditi potpuno bez sheme, mora postojati neki sustav prema kojem se manipulira podatcima. Jer ako baza nema shemu, to ne znači da ju nema i naša aplikacija. To je jedan od nedostataka ovih baza, jer je ipak potrebno pogledati podatke i pokušati shvatiti uzorak po kojem se vrši interakcija s njima kako bi baza i aplikacija radile učinkovito.

Agregatna orijentacija olakšava svojstvo skalabilnosti. Relacijske baze su dizajnirane za vertikalnu skalabilnost, tj. kako bi povećali performanse i nosili se sa povećanim prometom, dodaju se resursi jednom računalu, čvoru. Problem kod takvog sustava je u tome što je skup i ograničen, ne mogu se resursi dodavati beskonačno. Nerelacijske baze podataka su horizontalno skalabilne, što znači da se resursi baze povećavaju tako da se dodaju računala, čvorovi sa svojim instancama baze čime je olakšana distribucija baze podataka. Agregat je opet ovdje bolja struktura za manipulaciju. Umjesto dijeljenja podataka u više baza, moguće je pohraniti cijeli agragat na jednom čvoru što znači da ne moramo radit puno različitih upita kako bi okupili podatke i pristupili im. Kod relacijskih baza je to problem, jer je teško prikupiti sve podatke jednim upitom, najčešće ih je potrebno nekoliko.

Zbog različitih načina distribucije i pristupu modelu podataka, održavanje konzistentnosti je još jedno svojstvo u kojem se relacijske i nerelacijske baze razlikuju. SQL baza održava konzistentnost ACID transakcijama. One joj osiguravaju dostupnost i dosljednost podataka u bazi. NoSQL baze podataka konzistnetnost ne mogu riješiti samo svojim BASE svojstvima. Bez obzira na BASE svojstva, one mogu imati atomizirane transakcije. Ali te transakcije postoje samo unutar jednog agregata. Kada je potrebno sa jednom transakcijom djelovati na dvs ili više agragata, javljaju se problemi. Zato distribuirane baze podataka treba promatrati ne toliko u BASE smislu, već u svojstvima CAP teorema. Kada promatramo oba tipa baza podatak pomoću CAP teorema, vidljivo je da za distribuirane baze nema toliko razlike bile one relacijske ili ne. Ako je baza distribuirana, uvijek postoji mogućnosti da će doći do stvaranja particije i gubitka veze sa nekim grupama čvorova. U takvim situacijama moramo birati između preostala dva svojstva, a to su dostupnost i dosljednost. Potrebno je odrediti koje svojstvo je važnije i zatim da bi ga povećali, moramo do određenog stupnja smanjiti drugo svojstvo.

Redis i MongoDB međusobno imaju sličnosti. Osnova obje baze su pohrana podataka po ključevima sa pripadnim vrijednostima. Obje baze su zbog toga agregatno orijentirane. Nemaju shemu i podržavaju velik broj različitih tipova podataka. Razlikuju se u svojoj primjeni. Redis je specijalizirana baza podataka. Radi u memoriji, zbog čega je jako brz. Podržava specifične strukture podataka koji odgovaraju specifičnim problemima i primjenama. Primjena Redis baze je ipak i dalje velika jer podržava pohranu podataka na disk i nedostakak upitnog jezika nadomješta svojim strukturama kojima se mogu kreirati indeksi na ključeve. Također je bitna organizacija podataka i model baze ako se u Redisu planira napraviti složenija baza podataka. MongoDB je ipak prikladniji za širok spektar projekata. Nije toliko različit od relacijskih baza, jer pruža neke slične funkcije poput složenih upita i indeksiranja, te grupiranja podataka pomoću agregatnih funkcija. Zbog svoje široke primjene zaostaje za nekim aspektima za koje je redis više specijaliziran, dok s druge strane pruža dovoljnu fleksibilnost i velik broj mogućnosti zbog čega sigurno opravdava svoju titulu kao jedna od najpopularnijih i najkorištenijih baza podataka.

Pojava nerelacijskih nije ugrozila postojanje relacijskih baza podataka, jer i dalje postoje stvari u kojima su one bolje i učinkovitije. Bitno je shvatiti da NoSQL ne znači „ne SQL“ već „ne samo SQL“ čime nas upućuje u budućnost projekata koji za svaki dio svoje aplikacije koriste različite baze podataka čime bi se baze međusobno kao grupa upotpunjavale i pokrile pojedinačne nedostatke zbog čega bi cijeli sustav bio vrlo učinkovit

5. ZAKLJUČAK

Glavni zadatak ovog rada je bio usporediti relacijske i nerelacijske baze podataka, kako teorijski tako i na konkretnom primjeru. Odabrane su baze MongoDB i Redis, dan je njihov teorijski osvrt i projektirane su baze podataka za Rent a car tvrtku kao što je učinjeno i za relacijsku bazu u SQL-u.

Model nije dovoljno složen kako bi pokazao praktične, detaljnije razlike u performansama koje također definiraju ove baze podataka, ali ipak je dostatan da pokaže osnovne teorijske razlike koje se odnose na model podataka, osnovne funkcije i svojstva pojedine baze, način razmišljanja i pristupanja radu s ovim alatima. Predstavljene su i neke od tehničkih razlika poput uspostave i korištenja obrađenih baza na računalu i sintakse za svaku bazu. Pružene informacije korisne su u planiranju cilja kojem bi neki projekti sa ovakvim bazama mogli težiti ili kao predodžba potrebnih ulaganja i eventualnih dobiti ako bi odlučili upotrijebiti ove sustave.

LITERATURA

[1] P. J. Sadalage, M. Fowler - NoSQL Distilled - Addison-Wesley - 2012. – NoSQL općenito, 21.7.2016.

[2] myNoSQL, NoSQL Databases and Polyglot Persistance: A Curated Guide - http://nosql.mypopescu.com/ - NoSQL općenito - 21.7.2016.

[3] J. L. Harrington - Relational database design and implementation - Morgan Kaufmann – Burlington - 2009. – Relacijske baze podataka, 20.8.2016.

[4] I. Lukić – Upravljanje transakcijama – Baze podataka predavanje 9 – Loomen stranica kolegija Baze podataka – 20.8.2016.

[5] Redis, fom the Ground UP - http://blog.mjrusso.com/2010/10/17/redis-from-the-ground-up.html#heading_toc_j_11 – 29.8.2106. - Redis općenito

[6] T. Macedo, F. Oliveira – Redis Cookbook – O'Reilly – Sebastopol – 2011. – Redis općenito i upute – 27.8.2106.

[7] J. L. Carlson - Redis in action – Manning - Shelter Island - 2013. – Redis upute 25.8.2016. – 5.9.2016.

[8] Youtube, Redis Tutorials: Zero to Hero with NoSQL Redis - https://www.youtube.com/watch?v=A4gRg-9jNF4 - 25.8.2016. – Redis upute i instalacija

[9] K. Banker- MongoDB in action – Manning - Shelter Island - 2012. – MongoDB , 1.9.2016.

[10] Youtube, Mongodb Tutorial for Beginners - https://www.youtube.com/watch?v=W-WihPoEbR4 – 2.9.2016. – MongoDB upute i instalacija

[11] D. Hows, P. Membrey, E. Plugge, T. Hawkins – The Definitive Guide to MongoDB – Apress – New York – 2015. – MongoDB upute – 3.9.2016.

[12] K. Chodorow – MongoDB, The Definitive Guide – O'Reilly – Sebastopol – 2013. – MongoDB općenito i upute – 1.9.2016. – 10.9.2016

[13] K. Seguin - The Little MongoDB Book – MongoDB upute, 1.9.2016. – 5.9.2016.

SAŽETAK

Ovaj rad sadrži teorijsku osnovu NoSQL baza podataka, SQL baza podataka i primjer obje baze korišten kako bi se objasnila oba koncepta. Opisane su Redis i MongoDB baze i obje su korištene kako bi se napravila baza podataka Rent a car tvrtke. U prilogu se nalazi obje NoSQL baze podataka kao i SQL baza zajedno sa uputama kako ih pokrenuti i prilagoditi.

Ključne riječi: MongoDB, NoSQL, Redis, relacijske baze, SQL

ABSTRACT

Modelling a non-relational database

This paper contains a theoretical background for NoSQL databases, SQL databases and an example of both, used to compare their differences. Redis and MongoDB databases are explained and both of them were used to make a database for a Rent a car company. The appendix contains both NoSQL databases, a SQL database and instructions on how to start them up and how to configure them.

Keywords: MongoDB, NoSQL, Redis, relational databases, SQL

ŽIVOTOPIS

Matija Tivanovac rođen je 15.12.1994. godine u Osijeku. Živi u Belišću gdje stječe osnovnoškolsko obrazovanje u Osnovnoj školi Ivana Kukuljevića Belišće. Godine 2009. upisuje opću gimnaziju u Valpovu. Godine 2013. završava srednju školu sa odličnim uspjehom i položenom državnom maturom, te upisuje preddiplomski sveučilišni studij računarstva na Elektrotehničkom fakultetu u Osijeku. Od ostalih znanja i vještina posjeduje odlično znanje engleskog i vrlo dobro znanje njemačkog jezika, vješt je u radu na računalu i ima vozačku dozvolu B kategorije.

PRILOZI

Na optičkom disku uz rad u .doc i .pdf formatu nalaze se i ovi prilozi:

[1] redis_kod.txt – kod NoSQL baze za rent a car tvrtku u .txt formatu

[2] mongo_kod.txt – kod NoSQL baze za rent a car tvrtku u .txt formatu

[3] sql_kod.txt – kod relacijske baze podataka u .txt formatu

[4] dump.rdb – Redis RDB datoteka