UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Nataša Knez

Primerjava relacijske in NoSQL podatkovne baze in opredelitev kriterijev za pomoč pri izbiri najprimernejše podatkovne baze

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Marko Bajec

Ljubljana, 2012

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01773/2011
Datum: 01.09.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **NATAŠA KNEZ**

Naslov: **PRIMERJAVA RELACIJSKE IN NOSQL PODATKOVNE BAZE IN OPREDELITEV KRITERIJEV ZA POMOČ PRI IZBIRI NAJPRIMERNEJŠE PODATKOVNE BAZE**

COMPARISON OF RELATIONAL AND NOSQL DBMS AND THE DETERMINATION OF CRITERIA FOR THE SELECTION OF MOST APPROPRIATE DBMS

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Relacijske podatkovne baze niso več edina rešitev, ki se uporablja za hranjenje in obdelavo velikih količin (poslovnih) podatkov. V zadnjih letih so zelo popularne postale tudi t.i. noSQL rešitve, ki so v marsičem precej drugačne kot klasične relacijske podatkovne baze. V diplomskem delu opišite noSQL podatkovne baze, jih primerjate s klasičnimi relacijskimi ter opredelite kriterije, ki so lahko v pomoč pri izbiri sistema za upravljanje s podatkovnimi bazami, ki je v določenih okoliščinah najbolj primeren.

Mentor:
prof. dr. Marko Bajec

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisana **Nataša Knez**,

z vpisno številko **63050146**,

sem avtorica diplomskega dela z naslovom:

Primerjava relacijske in NoSQL podatkovne baze in opredelitev kriterijev za pomoč pri izbiri najprimernejše podatkovne baze

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelala samostojno pod mentorstvom

  prof. dr. Marka Bajca

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 14.3.2012 Podpis avtorja:

Zahvala

Iskreno se zahvaljujem mentorju, prof. dr. Marku Bajcu, za koristne nasvete in napotke pri izdelavi diplomskega dela ter strokoven pregled. Hvala za ustrežljivost in prijaznost.

Posebna zahvala je namenjena mojim staršem, mami Jani in očetu Ivanu, za spodbudo in podporo, razumevanje in potrpežljivost skozi celoten študij.
Starim staršem za neprestano skrb in stiskanje pesti.

Največjo zahvalo pa dolgujem Matjažu, ki mi vedno stoji ob strani, za spodbudne besede, za vso pozitivno energijo in ogromno mero ljubezni, katere niti NoSQL podatkovna baza ne zmore obdelati in shraniti.

Staršem

Kazalo

Seznam uporabljenih kratic

ACID	Atomicity, Consistency, Isolation, Durability (atomarnost, konsistentnost, izolacija, trajnost - lastnosti transakcij v relacijski podatkovni bazi)
API	Application Programming Interface (aplikacijski programski vmesnik)
BASE	Basically Available, Soft-state, Eventual Consistency (večinoma razpoložljiv, mehko stanje, eventuelna konsistentnost – lastnosti modela BASE, ki ga uporablja NoSQL)
BLOB	Binary Large Object (veliki binarni objekt)
CAP	Consistency, Availability, Partition Tolerance (konsistentnost, razpoložljivost, particijska toleranca)
CPE	Centralna procesna enota, procesor
DDL	Data Definition Language (jezik za definiranje podatkovnih struktur in tipov)
DML	Data Manipulation Language (jezik za manipulacijo s podatki)
EB	Exabyte (eksabajt)
EDI	Electronic Data Interchange (računalniška izmenjava podatkov)
HDFS	Hadoop Distributed File System (porazdeljen datotečni sistem za ogrodje Hadoop)
HQL	Hypertable Query Language (poizvedovalni jezik za Hypertable)
JDBC	Java Database Connectivity (Java API za dostop do podatkovne baze)
JSON	JavaScript Object Notation (format podatkov za izmenjavo strukturiranih dokumentov v spletu)

MVCC	Multiversion Concurrency Control (mehanizem za nadzor nad sočasnim izvajanjem transakcij nad večimi verzijami podatkov)
NoSQL	Not Only SQL (oznaka za novo generacijo podatkovnih baz)
OLE-DB	Object Linking and Embedding, Database (API za dostop do podatkovne baze)
OLTP	Online Transaction Processing (sprotna obdelava transakcij)
PB	Podatkovna baza
REST	Representational State Transfer (množica arhitekturnih principov za razvoj spletnih storitev)
SOAP	Simple Object Access Protocol (standard za razvoj spletnih storitev, ki temelji na XML)
SQL	Structured Query Language (poizvedovalni jezik v relacijskih podatkovnih bazah)
SUPB	Sistem za upravljanje podatkovne baze
THRIFT	Ogrodje za definicijo in kreiranje storitev za različne programske jezike
UnQL	Unstructured Data Query Language (enotni poizvedovalni jezik za NoSQL)
URL	Uniform Resource Locator (enoličen naslov spletne strani)
XML	Extensible Markup Language (format podatkov za izmenjavo strukturiranih dokumentov v spletu)

Povzetek

Diplomska naloga obravnava trenutno priljubljeno temo na področju podatkovnih baz – NoSQL podatkovne baze, ki se od tradicionalnih relacijskih podatkovnih baz razlikujejo v večih pogledih. Njihova ključna značilnost so horizontalna skalabilnost in shranjevanje ter obdelava ogromnih količin podatkov. V uvodnih poglavjih so predstavljene relacijske in NoSQL podatkovne baze. Opisane so glavne značilnosti, ključni dejavniki razvoja NoSQL podatkovnih baz ter osnovni koncepti in tehnike. Glavni del naloge najprej obravnava podrobno primerjavo med relacijskimi in NoSQL podatkovnimi bazami, prednosti in pomanjkljivosti ene in druge, klasifikacijo NoSQL v posamezne razrede, osnovne značilnosti vsakega od razredov in njihovo medsebojno primerjavo. Sledi opredelitev in opis kriterijev za lažjo izbiro med relacijskimi in NoSQL podatkovni bazami ter med posameznimi razredi znotraj NoSQL. Poleg opisa izbranih kriterijev so predstavljeni tudi primeri uporabe za posamezen razred. Zaključno poglavje izpostavi nekatere izzive pri izdelavi diplomske naloge in poda naše videnje prihodnosti podatkovnih baz NoSQL.

Ključne besede:

NoSQL, relacijska podatkovna baza, SQL, podatkovna baza, CAP

Abstract

The thesis discusses new developments in the area of next generation databases – NoSQL databases, which differ from traditional relational databases in many views. Their key features are horizontal scalability, storing and processing huge amounts of data. In the introductory chapter the relational and NoSQL databases are defined and the main characteristics, key factors of development of NoSQL databases and basic concepts and techniques are described in detail. The main part of the thesis deals with the comparison between relational and NoSQL databases, advantages and disadvantages of both, classification of NoSQL in individual classes, basic features of each class and the comparison among them. The discussion is followed by the definition and the description of criteria which enable potentional users to choose between relational and NoSQL databases as well as among individual classes of NoSQL. The description of the chosen criteria is followed by the presentation of several use cases of implementing the individual class. The final chapter highlights some of the challenges that the author deals with during writing the thesis and provides an insight into the future development of NoSQL databases.

Key words:

NoSQL, relational database, SQL, database, CAP

Uvod

Relacijske podatkovne baze so v zadnjih štiridesetih letih postale sinonim za shranjevanje podatkov. Izbira med podatkovnimi bazami je bila tako omejena na preučevanje razlik med razpoložljivimi komercialnimi in odprtokodnimi relacijskimi podatkovnimi bazami. Dolgo časa se relacijskim podatkovnim bazam ni uspela približati nobena druga podatkovna baza, a ravno, ko se je trg podatkovnih baz zdel že popolnoma zrel, je prišlo do velikih premikov.

Spletne aplikacije, ki so v lasti podjetij kot so Google, Amazon, Facebook, Twitter, itd., namreč danes ustvarjajo in obdelajo ogromne količine podatkov v obsegu več petabajtov. Takšna količina presega zmogljivosti tradicionalnih relacijskih podatkovnih baz. Poleg tega gre razvoj v smeri računalništva v oblaku, ki zahteva drugačne, bolj zmogljive in razpoložljive podatkovne baze, ki bodo omogočale nemoteno delovanje spletne aplikacije, ki lahko streže več milijonom uporabnikov. Vse to je botrovalo pojavu nove vrste podatkovnih baz, znanih pod imenom NoSQL.

Čeprav relacijske podatkovne baze ponujajo najboljšo kombinacijo preprostosti, robustnosti, prilagodljivosti, zmogljivosti in skalabilnosti, za določene primere uporabe niso najbolj primerne. Tu nastopijo podatkovne baze NoSQL, katerih glavne značilnosti so visoka skalabilnost in razpoložljivost, prilagodljiv podatkovni model, odsotnost relacij in stikov, preprost vmesnik in eventuelna konsistentnost. Vendar pa je za razliko od relacijskih podatkovnih baz, ki se med seboj bistveno ne razlikujejo, izbira med podatkovnimi bazami NoSQL precej zahtevnejši problem. Vsaka rešitev NoSQL je namreč namenjena določenemu primeru uporabe, oviro pa predstavlja tudi odsotnost standardov, formalnih opredelitev in pomanjkanje verodostojnih, strokovnih virov na tem področju.

Cilj diplomskega dela je opredeliti kriterije, ki bodo na prvem nivoju služili za pomoč pri izbiri med relacijsko in NoSQL podatkovno bazo, na drugem nivoju pa med posameznimi razredi NoSQL podatkovnih baz. Na tovrsten način bodo imeli uporabniki lažje delo pri morebitni odločitvi za prehod na podatkovno bazo NoSQL in izbiro rešitve znotraj posameznega razreda NoSQL.

Diplomska naloga je razdeljena na pet poglavij. V prvem poglavju *Relacijska podatkovna baza* bomo predstavili bistvene značilnosti relacijskega podatkovnega modela.

Drugo poglavje *Podatkovne baze NoSQL* obsega kratko predstavitev zgodovine in ključnih dejavnikov za razvoj NoSQL podatkovnih baz. Zaradi boljšega razumevanja pomena novih naborov tehnologij podatkovne baze bomo pojasnili pomen izraza NoSQL in podrobneje ter sistematično pogledali temeljne koncepte in tehnike. Razložili bomo pomen modela BASE, primerjavo z njegovo alternativo ACID ter Brewerjev CAP teorem in njegovo pomembno vlogo v porazdeljenih podatkovnih bazah. Analizirana je konsistentnost in njene različne oblike. Poleg tega je pojasnjen programski model MapReduce, ki poenostavlja porazdeljeno računalništvo in se uporablja pri nekaterih NoSQL podatkovnih bazah za zapletene agregacije.

Prednosti in pomanjkljivosti obeh vrst podatkovnih baz bomo globlje spoznali skozi analizo in primerjavo njunih značilnosti v okviru tretjega poglavja *Primerjava med relacijskimi in NoSQL podatkovnimi bazami*.

Četrto poglavje *Klasifikacija podatkovnih baz NoSQL* bo osredotočeno na različne razrede NoSQL podatkovnih baz. Opisali bomo značilnosti njihovih podatkovnih modelov, modelov poizvedb in tipične predstavnike ter medsebojno primerjavo.

Peto poglavje *Opredelitev kriterijev za pomoč pri izbiri najprimernejše podatkovne baze* je namenjeno predstavitvi kriterijev za izbiro med relacijsko ali NoSQL podatkovno bazo ter med posameznimi razredi NoSQL podatkovnih baz. Podanih je nekaj primerov uporabe za posamezno izbiro in opis uporabe opredeljenih kriterijev na hipotetičnem primeru.

V zaključnem poglavju bomo povzeli ugotovitve in sklepne misli ter podali napoved za prihodnost NoSQL podatkovnih baz.

Poglavje 1

Relacijska podatkovna baza

1.1 Podatkovna baza

Podatkovna baza je mehanizirana, večuporabniška, formalno definirana in centralno nadzorovana zbirka podatkov. Poleg samih operativnih podatkov vsebuje tudi opise podatkov, znanih kot sistemski katalog ali podatkovni slovar [11,38].

Vsaka zbirka podatkov pa ni podatkovna baza. Podatki morajo ustrezati določeni meri kakovosti, ki se meri glede na natančnost, razpoložljivost, uporabnost in prilagodljivost, kar dosežemo s sistemi za upravljanje s podatkovno bazo - SUPB-ji. SUPB je skupek programske opreme, ki omogoča kreiranje, vzdrževanje in nadzor nad dostopom do podatkov v podatkovni bazi [11]. Izraz podatkovna baza se uporablja za podatke in podatkovne strukture in ne SUPB.

Model, s katerim opišemo, kaj bi želeli hraniti ter kakšne povezave obstajajo med elementi, ki jih želimo hraniti, se imenuje podatkovni model. Je način, kako na visoki ravni abstrakcije opišemo podatke, ki jih želimo hraniti ter skrijemo nepomembne podrobnosti [11]. Izraža torej uporabnikovo predstavo, kako naj bodo podatki shranjeni.

Poznamo relacijske, hierarhične, mrežne in objektno-relacijske podatkovne modele. Najbolj znan in široko uporabljen med njimi je relacijski podatkovni model.

1.2 Relacijska podatkovna baza

1.2.1 Značilnosti relacijskega podatkovnega modela

Teorijo relacijskega podatkovnega modela je leta 1970 postavil Dr. Edgar F. Codd v svojem dokumentu »A Relational Model of Data for Large Shared Data Banks«. Na temelju tega dela so se razvile relacijske podatkovne baze, ki se še danes množično uporabljajo za shranjevanje podatkov. Relacijski podatkovni model je zelo enostaven za razumevanje, na voljo pa je tudi enostaven, a kljub temu močan jezik za poizvedovanje po vsebini podatkovne baze.

Primeri najbolj poznanih relacijskih podatkovnih baz:

- komercialne: IBM DB2, Oracle, Microsoft SQL Server, Sybase, itd.
- odprtokodne (s komercialnimi možnostmi): MySQL, Ingres, VoltDB, itd.

Osnovne prednosti relacijskega podatkovnega modela:

- definiran je formularno in osnovan na matematičnih strukturah ali relacijah;
- ne vsebuje elementov fizičnega shranjevanja podatkov, s čimer je zagotovljena podatkovna neodvisnost;
- relacije so predstavljene s tabelami, ki so človeku dobro razumljive.

Podatkovna baza, ki temelji na relacijskem podatkovnem modelu, je predstavljena z množico relacij, kjer je vsaka relacija tabela z enoličnim imenom. Tabele predstavljajo le logični pogled na podatke. Sestavljene so iz stolpcev ali atributov in vrstic ali zapisov (*Tuples*) ter imajo definirana medsebojna razmerja (*Relationships*).

Vsak atribut ima pripadajočo domeno, ki določa podatkovni tip vrednosti, ki jih atribut lahko zavzame, in integritetne omejitve za zagotavljanje celovitosti podatkov (npr. vrednost atributa ne sme biti prazna, vrednost atributa je lahko samo pozitivna številka, itd.).

Vsaka vrstica v tabeli je enolično določena s primarnim ključem, s čimer se preprečuje podvajanje podatkov. Primarni ključ je hkrati tudi indeks. Indeks je struktura, ki omogoča hiter dostop do vrstic tabele na podlagi vrednosti enega ali več stolpcev, odvisno od programskih zahtev.

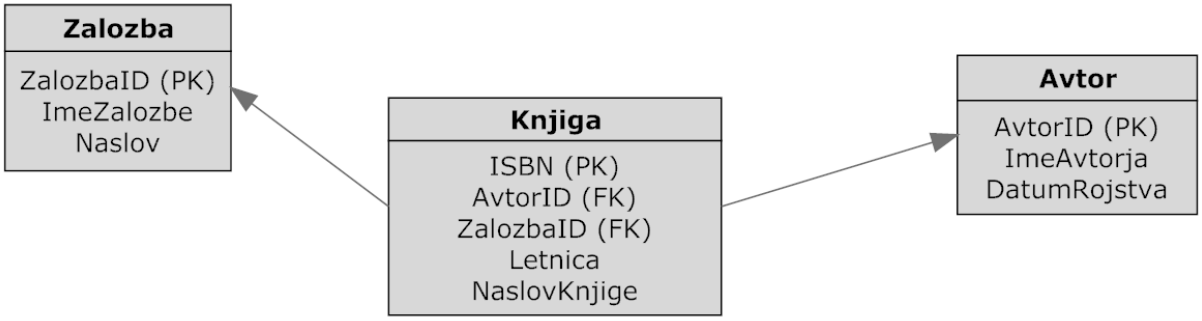
Tabele so lahko med seboj povezane z uporabo tujih ključev. Primarni ključ nadrejene tabele nastopa v podrejeni tabeli kot tuji ključ. Za zagotavljanje celovitosti podatkov v povezanih tabelah se uporablja referenčna integriteta. Ta zahteva, da tuji ključ v podrejeni tabeli lahko zavzame samo tiste vrednosti, ki jih zavzema tudi primarni ključ v nadrejeni tabeli.

Celovitost podatkov lahko zagotovimo tudi v sami aplikaciji, vendar ker te niso vedno dosledno pisane, bi lahko prišlo do podvajanja. Zaradi tega razloga to prepustimo podatkovni

bazi. To je dobrodošla rešitev v primeru, ko do istih podatkov dostopamo preko različnih aplikacij, saj se celovitost podatkov lahko zagotavlja na eni centralni lokaciji.

Podvojitve podatkov odstranimo s procesom normalizacije, tako da premestimo stolpce v ločene tabele in uporabimo tuje ključe. Normalizacija privede do takšne strukture podatkovnega modela relacijske podatkovne baze, ki zagotavlja skladnost podatkov in preprečuje podvojenost podatkov. Obratni proces je denormalizacija, ki je v določenih primerih lahko potrebna.

Slika 1.1 je primer tipičnega relacijskega podatkovnega modela, ki prikazuje podatke o knjigah, njihovih avtorjih in založbah.



ISBN	AvtorID	ZalozbaID	Letnica	NaslovKnjige
1-12345-678-1	143-12345	03-3672983	2011	NoSQL
1-16948-574-2	398-98675	04-2945762	1995	Podatkovne baze
2-38943-947-2	364-28564	03-1058376	2010	CouchDB
1-34697-743-4	648-23964	03-3456096	2010	Cassandra

Slika 1.1. Primer tipičnega relacijskega podatkovnega modela in tabele

1.2.2 Lastnosti transakcij ACID

Ena zelo pomembnih značilnosti relacijske podatkovne baze je njihova sposobnost, da za transakcije zagotovijo lastnosti ACID. ACID je kratica za atomarnost (*Atomicity*), konsistentnost (*Consistency*), izolacijo (*Isolation*) in trajnost (*Durability*). Predstavlja standard za opredelitev najvišje ravni celovitosti transakcij v sistemih za upravljanje s podatkovnimi bazami. Gre za kazalnike, ki jih lahko uporabimo za oceno, ali je bila transakcija pravilno in uspešno izvedena.

Transakcija v podatkovni bazi je ena ali več logičnih operacij, ki dostopajo ali spreminjajo vsebino podatkovne baze in jih izvede uporabnik ali aplikacija [11]. Primer transakcije je vstavljanje naslova knjige ali posodobitev letnice knjige.

Formalna opredelitev transakcije po ACID je sledeča:

- **Atomarnost** (*Atomicity*): transakcija predstavlja atomaren, to je nedeljiv sklop operacij, kar pomeni, da se morajo vse spremembe podatkov znotraj transakcije izvesti v celoti ali pa sploh ne. Delno izvedene transakcije niso možne. Atomarnost mora zagotavljati SUPB. Če pride do napake, se transakcija razveljavi in povrne se prvotno stanje (*Rollback*).

- **Konsistentnost** (*Consistency*): Pri izvajanju transakcij podatkovna baza preide iz enega konsistentnega stanja v drugo. Če transakcije kršijo pravilo konsistentnosti, potem morajo biti vse spremembe v transakciji razveljavljene. Zagotavljanje konsistentnosti je naloga SUPB in programerjev (preprečuje vsebinsko neskladnost).

- **Izolacija** (*Isolation*): izolacija pomeni, da rezultati spremembe med transakcijo niso vidni, dokler transakcija ni potrjena. Transakcije se izvajajo neodvisno ena od druge, delni rezultati transakcije ne smejo biti vidni drugim transakcijam. Za izolacijo skrbi SUPB.

- **Trajnost** (*Durability*): zahteva trajnosti je, da so spremembe zapisane na disk preden podatkovna baza potrdi transakcijo. S tem se potrjeni podatki v primeru odpovedi sistema ne izgubijo.

1.2.3 Poizvedovalni jezik

Obstaja veliko razlogov, da je relacijska podatkovna baza postala tako zelo priljubljena v zadnjih štirih desetletjih. Eden izmed njih je uporaba poizvedovalnega jezika SQL (*Structured Query Language*), ki je najbolj široko podprt standardni jezik za podatkovne baze. Je funkcijsko bogat in uporablja preprosto, deklarativno sintakso.

SQL lahko nastopa kot:

- **Jezik za definiranje podatkovnih struktur in tipov** (*Data Definition Language, DDL*): Zagotavlja ukaze za definiranje relacijske sheme, oblikovanje indeksov in definiranje pogledov (CREATE), brisanje relacij (DROP), spreminjanje relacijske sheme (ALTER) in dodeljevanje pravic dostopa (GRANT/REVOKE).

- **Jezik za manipulacijo s podatki** (*Data Manipulation Language, DML*): Vključuje ukaze za vnos (INSERT), brisanje (DELETE) in posodabljanje (UPDATE) n-teric.

- **Poizvedovalni jezik** (*Query Language*): omogoča poizvedovanje nad eno ali več tabelami. Tabele, do katerih dostopa ena sama poizvedba, morajo biti združene s stikom. Lahko izvajamo bogat nabor operacij z uporabo funkcij na osnovi relacijske algebre, da bi na primer našli največjo ali najmanjšo vrednost v naboru, ali pa filtriramo in uredimo rezultate.

Primer poizvedbe z SQL v podatkovni bazi DB2, ki vrne število knjig za posameznega avtorja na podlagi zgornjega podatkovnega modela:

```
SELECT count(k.ISBN) AS stevilo_knjig, a.ImeAvtorja AS Avtor
FROM Knjiga k INNER JOIN Avtor a ON k.AvtorID = a.AvtorID
GROUP BY a.ImeAvtorja;
```

SQL je enostaven za uporabo. Osnovno sintakso se je mogoče hitro naučiti. Obstaja tudi veliko robustnih orodij, ki vključujejo intuitivne grafične vmesnike za vpogled in delo s podatkovno bazo. Zaradi tega ga lahko stroka, ki ni vešča programiranja, uporablja, da relativno hitro dobi željen rezultat.

Deloma zato, ker je standardni jezik, SQL omogoča enostavno integracijo SUPB s široko izbiro podatkovnih baz. Vse kar potrebujemo je gonilnik za naš programski jezik.

Poglavje 2

Podatkovne baze NoSQL

V preteklosti so se relacijske podatkovne baze uporabljale za skoraj vse. Razlog je njihov bogat nabor funkcij, možnost opravljanja zmogljivih poizvedb in upravljanje transakcij. Njihove številne funkcionalnosti pa so hkrati pomanjkljivost, saj je izgradnja porazdeljenih SUPB-jev zelo zapletena. Še posebej težko je v njih realizirati transakcije in operacije stikov.

Iz tega razloga so se pojavile nerelacijske podatkovne baze, ki sicer omogočajo omejen nabor funkcij in nepopolno podporo ACID, vendar so primernejše za uporabo v porazdeljenemu okolju. Te podatkovne baze se trenutno imenujejo NoSQL, kar je okrajšava za »Not only SQL«. Izraz se je prvič pojavil leta 1998, šele v letu 2009 pa so podatkovne baze NoSQL postale bolj opazne v strokovni javnosti.

2.1 Definicija

Podatkovne baze NoSQL so naslednja generacija podatkovnih baz. Izraz NoSQL ponazarja, da te podatkovne baze ne podpirajo poizvedovalnega jezika SQL in niso relacijske. Hkrati pa pomeni tudi »Not Only SQL«, kar nakazuje na dejstvo, da so za različne primere uporabe potrebne različne podatkovne baze.

Večina NoSQL podatkovnih baz je razvitih z namenom, da tečejo na gručah računalnikov, zato morajo biti porazdeljene in odporne proti napakam. Za ta namen je potrebno sprejeti kompromise glede ACID lastnosti, upravljanja transakcij in zmogljivosti poizvedb. Običajno so namenjene spletnim aplikacijam, so skalabilne, ne zahtevajo oziroma omogočajo prilagodljivejšo podatkovno shemo, so odprtokodne in prinašajo lastne poizvedovalne jezike [12]. Dodatna prednost je tudi ta, da ne potrebujejo administratorja podatkovne baze.

NoSQL podatkovne baze imajo običajno pet ključnih značilnosti, ki jih bomo podrobneje spoznali v nadaljevanju in pri primerjavi z relacijsko podatkovno bazo [9]:

- **Sposobnost horizontalne skalabilnosti**
- **Sposobnost repliciranja in particioniranja podatkov čez več strežnikov**
- **Dostop s preprostim vmesnikom API**
- **Šibkejši model nadzora nad sočasnim izvajanjem transakcij**
- **Prilagodljiva shema**

2.2 Zgodovina

Carlo Strozzi je prvi uporabil izraz »NoSQL« leta 1998 za poimenovanje svoje odprtokodne relacijske podatkovne baze, ki ni izpostavljala vmesnika SQL. Strozzi je predlagal, da ker se NoSQL gibanje oddaljuje od relacijskega modela v celoti, bi se ta moral primernejše imenovati NoREL [31,37].

Izraz NoSQL je bil ponovno uporabljen leta 2009 na konferenci zagovornikov nerelacijskih podatkovnih baz. Z izrazom so nameravali označiti pojav vedno več nerelacijskih, porazdeljenih podatkovnih shramb, ki pogosto niso poskušale zagotoviti lastnosti ACID, ki so ključne za tradicionalne relacijske podatkovne baze [37].

Spodnja slika 2.1 [15] prikazuje zanimanje za NoSQL od leta 2009 do danes na podlagi iskanj po spletu. Opazno je strmo naraščanje priljubljenosti NoSQL.

Slika 2.1. Naraščanje zanimanja za NoSQL

Zaposlenec podjetja Rackspace, Eric Evans, je kot cilj gibanja NoSQL izpostavil iskanje alternativnih rešitev za situacije, v katerih uporaba relacijske podatkovne baze ni primerna.

Ponudniki rešitev NoSQL se strinjajo, da izraz »NoSQL« ni popoln, je pa privlačen za javnost. Beseda »No« je pravzaprav okrajšava za »Not Only«, kar pomeni, da cilj podatkovnih baz NoSQL ni v celoti zavrniti SQL, ampak bolj premostiti tehnične omejitve relacijskih podatkovnih baz [8]. Pravzaprav je NoSQL bolj zavrnitev določene arhitekture programske in strojne opreme za podatkovne baze kot pa katerekoli tehnologije, jezika ali produkta. Relacijske podatkovne baze so se razvile v drugačnem obdobju z različnimi tehnološkimi omejitvami, ki so vodile v za tisti čas optimalno arhitekturno obliko. Za današnje razmere pa prej uspešna arhitektura predstavlja ovire.

Google je v zadnjih nekaj letih zgradil masivno skalabilno infrastrukturo za svoj iskalnik in ostale aplikacije, kot sta Gmail in Google Maps, z namenom vzporednega obdelovanja ogromnih količin podatkov [34]. Med drugim je razvil tudi stolpično usmerjeno podatkovno shrambo BigTable, ki jo je opisal v dokumentih, namenjenih širši javnosti. To je pritegnilo veliko zanimanja med odprtokodnimi razvijalci. Leto kasneje je tudi Amazon predstavil svojo porazdeljeno in visoko razpoložljivo podatkovno shrambo Dynamo.

S sprejetjem tovrstnih podatkovnih baz s strani dveh tako velikih podjetij se je v tem prostoru pojavilo še več novih produktov in tako je v nekaj letih NoSQL postal rešitev za upravljanje velikih podatkov v nekaj zelo dobro poznanih podjetjih kot so Facebook, Yahoo!, Ebay, Twitter, itd.

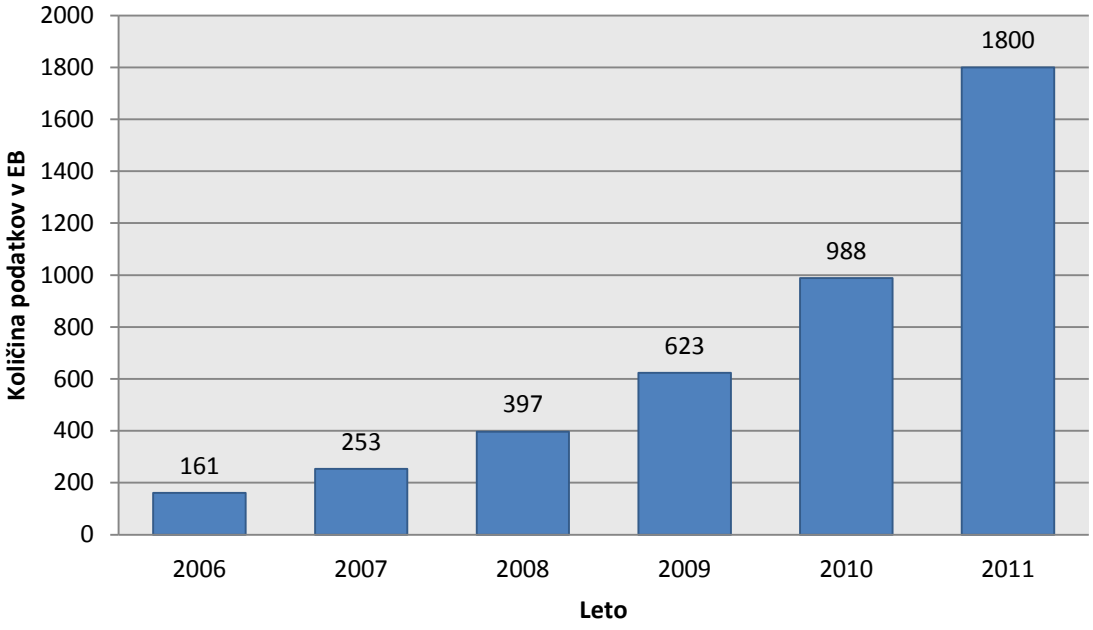
2.3 Dejavniki

Obstaja več ključnih dejavnikov za razvoj podatkovnih baz NoSQL [31], in sicer:

- **Nenehna rast podatkov**
- **Polstrukturirani in nestrukturirani podatki ter statične poizvedbe**
- **Izogibanje preslikavi iz objektov v relacije**
- **Zahteve računalništva v oblaku**
- **Stroškovno učinkovita skalabilnost**
- **Sprejemanje kompromisov glede ACID lastnosti**

Nenehna rast podatkov
Kot prikazuje graf na sliki 2.2 [22], je bila leta 2006 količina digitalnih podatkov približno 161 eksabajtov (EB). Eksabajt je 1.1 milijona terabajtov. V letu 2010 se je ta številka povzpela na skoraj 1000 eksabajtov. Za primerjavo, 161 EB je približno 3-milijonkrat več kot

je količine informacij v vseh knjigah, ki so bile kdajkoli napisane. V letu 2011 je količina ustvarjenih in repliciranih informacij presegla 1.8 zetabajtov (1800 EB) in naj bi naslednjih pet let naraščala s faktorjem 9 [20].



Slika 2.2. Naraščanje količine shranjenih podatkov

Še nekaj ponazoritev:

- Google obdela 24 petabajtov podatkov na dan [34].
- YouTube vsak dan streže 100 milijonov video posnetkov [19].
- Facebook je za leto 2009 objavil, da se na njihovem socialnem omrežju nahaja 60 milijard slik, ki skupno zavzemajo 1.5 petabajtov prostora [34].
- Borza v New Yorku vsak dan ustvari približno en terabajt novih podatkov [21].

Kot lahko vidimo, obstaja veliko različnih vrst podatkov, ki jih je potrebno shranjevati, obdelati in nad njimi poizvedovati, prav tako obstajajo tudi različna podjetja, ki uporabljajo tovrstne podatke.

S tem ko število virov in količina podatkov naraščata, se pojavljajo naslednji pomembni izzivi:

- **Učinkovito shranjevanje in dostopanje:** učinkovito shranjevanje in dostopanje do velike količine podatkov je težavno. Dodaten zaplet prinašajo še zahteve po toleranci v primeru okvar.

- **Učinkovita obdelava podatkov:** vse večja potreba po obdelavi večjih količin podatkov v čim krajšem času.
- **Obnavljanje podatkov:** delo z velikimi nabori podatkov vključuje tudi ogromno število vzporednih procesov, zato je obnavljanje v primeru napak in zagotavljanje rezultatov v razumno kratkem času lahko zelo zapleteno.
- **Upravljanje podatkovne strukture:** velik izziv predstavlja tudi upravljanje stalno spreminjajoče se podatkovne sheme in metapodatkov za polstrukturirane in nestrukturirane podatke, ki prihajajo iz različnih virov.

Polstrukturirani in nestrukturirani podatki ter statične poizvedbe
Potrebe glede shranjevanja podatkov so se s časom bistveno spremenile. Relacijske podatkovne baze so namenjene shranjevanju podatkov, ki so strogo strukturirani v relacije, in ki ustrezajo vnaprej določeni shemi.

S prihodom Web 2.0 se je pojavljajo vedno več polstrukturiranih (npr. dokument XML, e-pošta) in nestrukturiranih podatkov. Slednji nimajo nobene posebne notranje strukture. Lahko gre za golo besedilo (dejanski dokument), slike, video, itd.

Danes niso potrebne niti dinamične poizvedbe, še posebej na področju spleta, saj večina aplikacij že uporablja vnaprej pripravljene izjave ali shranjene procedure.

Izogibanje preslikavi iz objektov v relacije
Večina NoSQL podatkovnih baz je namenjenih shranjevanju podatkovnih struktur, ki so za razliko od relacijskih podatkovnih struktur bodisi preproste bodisi bolj podobne tistim iz objektno-usmerjenih programskih jezikov. Tako ni potrebe po dragi preslikavi iz objektov v relacije. Relacijske podatkovne baze namreč ponujajo tako veliko funkcionalnosti, da je potrebno objekte oziroma podatke spremeniti tako, da se prilegajo podatkovni bazi. To je zlasti pomembno za aplikacije z enostavnimi podatkovnimi strukturami, ki imajo bolj malo koristi od značilnosti relacijskih podatkovnih baz.

Zahteve računalništva v oblaku
Računalništvo v oblaku zahteva od podatkovnih shramb dve glavni značilnosti:

- Visoka skalabilnost in zmogljivost
- Nizki stroški upravljanja

Stroškovno učinkovita skalabilnost
Ker s spletom svet postaja vse bolj povezan, so lahko spletna mesta podvržena velikim razlikam v številu zahtev uporabnikov. Nekatere od teh (npr. Twitter) so povezane s predvidljivimi dogodki kot so svetovna prvenstva, božič, volitve, itd., drugi pa so nepredvidljivi in globalni (npr. 11. september 2001), ki predstavljajo velik izziv za portale

novic. Posledično se mora infrastruktura spletnih mest v ozkem časovnem okviru hitro prilagoditi številu zahtev. Tradicionalni pristop k skalabilnosti predstavlja dodajanje spletnih strežnikov. Vendar ta rešitev deluje le, dokler promet nad podatkovno bazo ne predstavlja ozkega grla. Tako se je pojavila potreba po skalabilnih podatkovnih bazah.
Za razliko od relacijskih podatkovnih baz je večina NoSQL podatkovnih baz načrtovana za dobro skalabilnost v horizontalni smeri in se zato ne zanašajo na visoko razpoložljivo strojno opremo. Strežnike je mogoče dodajati ali odvzemati brez kompleksnih operacij in velikih stroškov kot je to pri relacijskih podatkovnih bazah. Slednje so sicer lahko skalabilne, vendar so pri tem stroškovno neučinkovite.

Sprejemanje kompromisov glede ACID lastnosti
Bogat nabor funkcij in lastnosti ACID, ki jih implementira SUPB, je lahko nepotreben za določene aplikacije in primere uporabe. Sprejetje možne izgube podatkov ali možne začasne nekonsistentnosti, kar je največja ovira za skalabilnost relacijskih podatkovnih baz, lahko prinese boljšo prilagodljivost. To velja za večino družabnih strani, saj npr. za pisanje statusov na Facebooku ne potrebujemo ACID.

S sklepanjem kompromisov pri določenih strogih omejitvah relacijskih podatkovnih baz in s porazdelitvijo podatkov, podatkovne baze NoSQL omogočajo boljšo zmogljivost za sledeče primere uporabe:

- Ogromne količine podatkov
- Polstrukturirani podatki
- Veliko število pisalnih operacij
- Nizka latenca bralnih operacij
- Potreba po skalabilnosti brez ozkih grl
- Ni potrebe po ad hoc poizvedbah

Če povzamemo, so pionirji gibanja NoSQL predvsem velika spletna podjetja in podjetja, ki nudijo visoko skalabilne spletne strani, kot so Facebook, Google, Amazon. Ostali so sprejeli njihove ideje in jih prilagodili tako, da ustrezajo njihovim zahtevam in potrebam.

2.4 Kritike

NoSQL podatkovne baze so s svojim pojavom povzročile veliko razburjenja med pristaši relacijskih podatkovnih baz, ki NoSQL zavračajo na podlagi spodnjih kritik [31].

Skeptičnost s poslovne strani
Ker je večina NoSQL podatkovnih baz odprtokodnih, so zelo dobro sprejete s strani razvijalcev, ki jim ni potrebno skrbeti glede licenc in problemov oglaševalske podpore. Vendar pa lahko to prestraši poslovne uporabnike, še posebej v primeru napak, za katere noben ne prevzame odgovornosti.

NoSQL kot nič novega
Pogoste pripombe kritikov NoSQL so, da NoSQL podatkovne baze niso nič novega, saj so drugi poskusi kot so objektne podatkovne baze prisotni že desetletja. Primer je Lotus Notes, ki ga je mogoče razumeti kot zgodnjo dokumentno shrambo, ki podpira porazdelitev in repliciranje.

NoSQL je pomenil v celoti »No to SQL«
Sprva je veliko zagovornikov NoSQL, predvsem blogerji, razumelo izraz in gibanje kot zanikanje relacijskih podatkovnih baz v celoti in razglasili so celo njihovo »smrt«. Eric Evans, ki se ga pogosto povezuje z izrazom »NoSQL«, je v blogu leta 2009 predlagal, da bi moral izraz pomeniti »Not only SQL«, torej »Ne samo SQL«, namesto »No to SQL«. Ta izraz je bil sprejet s strani številnih blogerjev, saj poudarja, da na področju podatkovnih bazah ne obstajajo samo relacijske podatkovne baze, ampak tudi alternative.
Nekateri blogerji poudarjajo, da je izraz nenatančen in da bodo potrebne trdne tehnične definicije, kaj točno za neko rešitev pomeni, da je NoSQL podatkovna baza, poleg dejstva, da ni relacijska podatkovna baza. Drugi trdijo, da nima nič opraviti z SQL in bi se moral imenovati nekaj v smeri »NoACID«. Ostali pa kritizirajo, da izraz »NoSQL« opredeljuje bolj to kar počne samo gibanje in ne za kar stoji. Problem tega imena je torej, da opredeljuje vse tisto, kar ni. Zaradi tega je v protislovju s samim seboj in ni natančen glede tega, kaj vključuje ali izključuje.

Vroče razprave o izrazu in njegovem prvem pomenu kot celotno zanemarjanje relacijskih podatkovnih baz so povzročile veliko izzivalnih izjav pri zagovornikih NoSQL in številne razprave, ki niso obrodile sadov.

Ne obnašajo se vsi relacijski SUPB-ji kot MySQL
Velikokrat se kot bistvena prednost NoSQL pred relacijskimi bazami omenja dobra skalabilnost. To je pogosto pojasnjeno na primeru MySQL, vendar je potrebno poudariti, da vse relacijske podatkovne baze niso enake MySQL.

Res je, da nekateri posamezni relacijski SUPB-ji niso dobro oziroma so težje skalabilni, vendar to ne pomeni, da tega ne zmore vsak SUPB. Včasih priljubljena spletna stran razglasi, da določen SUPB ne ustreza njihovim potrebam in se zato selijo na določeno NoSQL podatkovno bazo. Mnenje nekaterih v svetu SUPB je, da veliko teh premikov ni zato, ker bi bila podatkovna baza, ki so jo uporabljali, pomanjkljiva, pač pa zato, ker je bila uporabljena na način, za katerega ni bila načrtovana. Relacijska podatkovna baza je, kadar je zasnovana pravilno, lahko bolj zmogljiva v primerjavi s slabo zasnovano podatkovno bazo.

2.5 Osnovni koncepti in tehnike

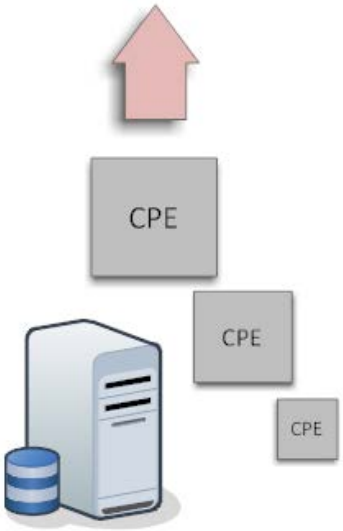
2.5.1 Skalabilnost

Današnje spletne aplikacije se morajo soočiti z izzivom strežbe zahtev več milijonov uporabnikov iz celega sveta, ki pričakujejo, da bo aplikacija vedno razpoložljiva in njeno delovanje zanesljivo. Uspešne spletne aplikacije nimajo le velikih baz uporabnikov, ampak tudi rastejo hitreje kot pa se povečuje zmogljivost računalniške strojne opreme. Spletna aplikacija, ki postaja velika, mora biti skalabilna.

Skalabilnost spletne aplikacije pomeni, da se lahko kakršnakoli količina podatkov obdela v končnem času, če aplikacija teče na zadostnem številu strežnikov ali t.i. vozlišč.

Obstajata dve vrsti skalabilnosti: vertikalna skalabilnost in horizontalna skalabilnost.

Vertikalna skalabilnost (*Scale-up*)
Vertikalna skalabilnost (slika 2.3 [3]) je dosežena tako, da sistem za upravljanje s podatkovno bazo uporablja več procesorskih jeder, ki si delijo pomnilnik in diske.



Slika 2.3. Vertikalna skalabilnost

Značilnosti:

- Dodajanje virov enemu vozlišču v sistemu
    - dodajanje več CPE ali pomnilnika
- Premik sistema na večji računalnik

Prednosti:

- Hitro in preprosto

Slabosti:

- Drago
- Problem priklenitve na ponudnika (*Vendor lock-in*)

Horizontalna skalabilnost (*Scale-out*)
Horizontalna skalabilnost (slika 2.4 [3]) je porazdelitev tako podatkov kot bremena čez več strežnikov, brez da bi si strežniki delili skupen pomnilnik.

Slika 2.4. Horizontalna skalabilnost

Značilnosti:

- Sistemu se dodajajo vozlišča

Prednosti

- Večja prilagodljivost

Pomanjkljivosti

- Večja kompleksnost

Nekatere NoSQL podatkovne baze ponujajo oboje, vertikalno in horizontalno skalabilnost, vendar bo naš glavni poudarek na horizontalni skalabilnosti. Razlog je ta, da je število jeder, ki si lahko delijo pomnilnik, omejeno, horizontalna skalabilnost pa se v splošnem izkaže kot cenejša, ker lahko izkorišča poceni strežnike.

2.5.2 Pristopi za omogočanje horizontalne skalabilnosti

Podatkovna baza je lahko horizontalno skalabilna na tri različne načine [27]:

- s količino bralnih operacij,
- količino pisalnih operacij in

- velikostjo podatkovne baze.

Trenutno obstajata dva pristopa, ki se uporabljata za doseganje horizontalne skalabilnosti, to sta: replikacija (*Replication*) in delitev (*Sharding*).

2.5.2.1 Replikacija

Replikacija v primeru porazdeljenih podatkovnih baz pomeni, da je podatek shranjen v več kot enem vozlišču. To je zelo koristno za povečanje zmogljivosti podatkovne baze, saj omogoča porazdeljenost bralnih operacij čez več vozlišč. Prednost replikacije je tudi ta, da naredi gručo odporno na napake posameznih vozlišč. Če eno vozlišče odpove, potem ga lahko nadomesti drugo vozlišče, ki vsebuje enake podatke [27].

Včasih je koristno replicirati podatke tudi po različnih podatkovnih centrih, kar naredi podatkovno bazo odporno v primeru naravnih katastrof. Prednost je tudi, da so podatki lahko geografsko bližje uporabnikom, kar zmanjša čas dostopa.

Negativna stran replikacije podatkov so pisalne operacije. Te morajo isti podatek zapisati na več vozlišč. Podatkovna baza ima v bistvu dve možnosti za to: bodisi mora biti pisalna operacija potrjena (*Commit*) na vseh replikacijskih vozliščih preden podatkovna baza vrne potrditev, bodisi se pisalna operacija najprej izvede na enem ali omejenem številu vozlišč in nato sčasoma še na vseh ostalih. Izbira ene od možnosti odloča o tem, kako razpoložljiva in kako konsistentna bo podatkovna baza. Več o tem bomo spoznali v okviru razlage Brewer-jevega CAP teorema.

Sinhrona in asinhrona replikacija
Replikacija je torej lahko sinhrona, kar pomeni, da lahko zagotovi, da so replike podatkov vedno sinhronizirane. Slabost je počasnost, ker mora sistem ob vsaki posodobitvi podatka počakati, da so posodobljene tudi vse replike. Druga možnost pa je asinhrona replikacija, kjer se replikacije posodobijo asinhrono v ozadju. Asinhrona replikacija je hitrejša, zlasti v primeru oddaljenih replik, vendar pa se nekatere posodobitve lahko izgubijo pri izpadih. Nekateri sistemi posodabljajo lokalne replike sinhrono, oddaljene replike pa asinhrono.

2.5.2.2 Delitev

Gre za delitev podatkov na več kosov, ki so lahko porazdeljeni čez več vozlišč [27]. Prednost delitve je, da se vozlišča lahko dodajajo gruči z namenom povečanje zmogljivosti bralnih in pisalnih operacij, brez da bi bilo potrebno spremeniti aplikacijo. Mogoče je celo zmanjšati velikost gruče deljene podatkovne baze, ko se količina zahtev zmanjša.

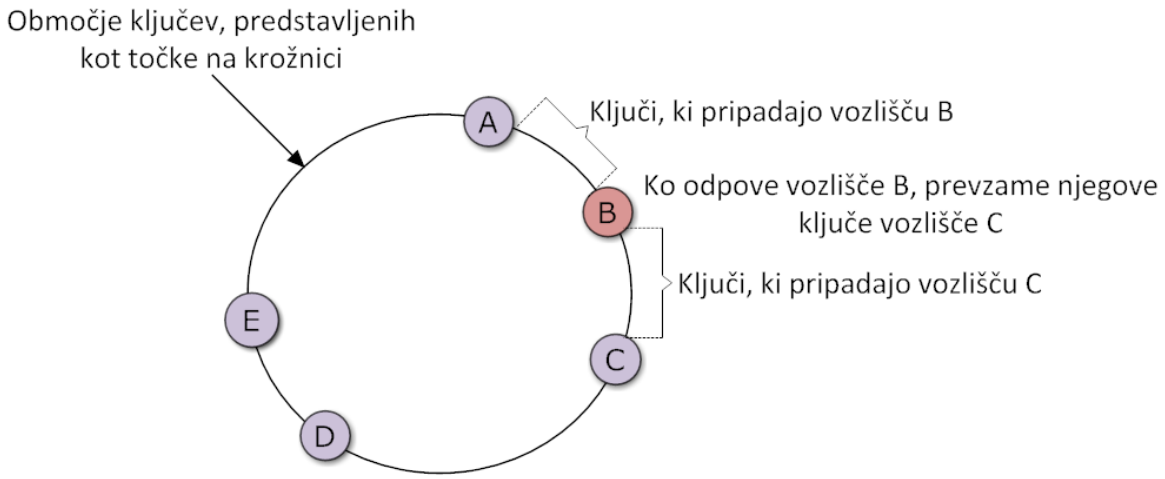
Negativna stran delitve je, da tipične operacije podatkovne baze postanejo zelo kompleksne in neučinkovite. Ena od najpomembnejših operacij v relacijski podatkovni bazi je stična operacija (*Join*). Stik se izvaja na dveh tabelah podatkov, ki sta povezani s parom atributov.

Implementacija porazdeljenega stika v deljeni podatkovni bazi bi pomenila iskanje vseh elementov ene table, ki so povezani z vsakim elementom druge tabele. To bi zahtevalo veliko zahtev za vsa vozlišča, ki shranjujejo podatke v eno izmed tabel, kar bi povzročilo povečanje omrežnega prometa. Zaradi tega večina deljenih podatkovnih baz ne podpira stičnih operacij.
Več ko je uporabljenih vozlišč znotraj gruče deljene podatkovne baze, večja je verjetnost, da eno od vozlišč ali ena od mrežnih povezav odpove. Zato deljenje pogosto nastopa v kombinaciji z replikacijo, zaradi česar je gruča odpornejša na okvare strojne opreme.

Konsistentno zgoščevanje
Ker so kosi podatkov porazdeljeni po večih vozliščih, potrebujemo način, da preslikamo vsak ključ v pripadajoče vozlišče. Eden je ta, da uporabimo zgoščevalno funkcijo (*Hash function*), ki uporablja primarni ključ podatkov za določitev povezanih kosov podatkov. Primer zgoščevalne funkcije bi bil sledeč:

Particija = ključ mod (število vseh vozlišč)

Pomanjkljivost tega načina je, da sprememba števila vozlišč zahteva ponovno porazdelitev vseh ključev. Da bi zmanjšali število ključev, potrebnih za ponovno porazdelitev, večina shramb NoSQL uporablja tehniko konsistentnega zgoščevanja (*Consistent hashing*) prikazanega na sliki 2.5 [31].



Slika 2.5. Konsistentno zgoščevanje

V tem primeru se ključi za vsako vozlišče nahajajo na obodu kroga, ki se razteza od predhodnega vozlišča do vozlišča, ki je lastnik ključev. Če pride do okvare lastnika, vse njegove pripadajoče ključe posvoji sosednje vozlišče v smeri urinega kazalca. Na ta način se ponovno porazdelijo samo ključi okvarjenega vozlišča, ne pa vsi.
V primeru dodajanja vozlišča v gručo pa ta prevzame ključe, ki se nahajajo med njim in njegovim sosedom v nasprotni smeri urinega kazalca. Torej spet ni potrebno, da bi se vsi ključi porazdelili na nov nabor vozlišč.

2.5.3 Model BASE

Za vedno večje število aplikacij sta razpoložljivost in particijska toleranca veliko bolj pomembni kot stroga konsistentnost. Te lastnosti je težko doseči z lastnostmi ACID, zato se uporablja pristop kot je BASE.

BASE je kratica za *Basically Available, Soft-state, Eventual Consistency*, ki je jo skoval Dan Pritchett v članku revije *ACM Queue magazine* kot nasprotje kratice ACID. Opisuje torej podatkovne baze, ki ne implementirajo v celoti modela ACID. Glavna razlika je, da so te podatkovne baze eventuelno konsistentne. Ideja je, da če se odpovemo konsistentnosti, lahko pridobimo več pri razpoložljivosti in močno izboljšamo skalabilnost podatkovne baze.

Lastnosti modela BASE so sledeče [31]:

- **Večinoma razpoložljiv** (*Basically Available*): sistem zagotavlja razpoložljivost glede na CAP teorem.
- **Mehko stanje** (*Soft-state*): stanje sistema se zaradi asinhronih posodobitev čez čas spremeni tudi brez sprememb uporabnika. Posledica tega je, da so kopije podatkov v nekem času lahko nekonsistentne.
- **Eventuelna konsistentnost** (*Eventual Consistency*): nekatera vozlišča imajo točne podatke, druga vozlišča imajo zastarele podatke. Kopije podatkov sčasoma postanejo konsistentne, ko se nad njimi več ne izvajajo posodobitve.

Lastnosti BASE bi lahko povzeli na naslednji način: aplikacija deluje v bistvu ves čas (večinoma razpoložljiv), ni nujno, da je ves čas tudi konsistentna (mehko stanje), vendar sčasoma bo (eventuelna konsistentnost).

2.5.4 CAP teorem

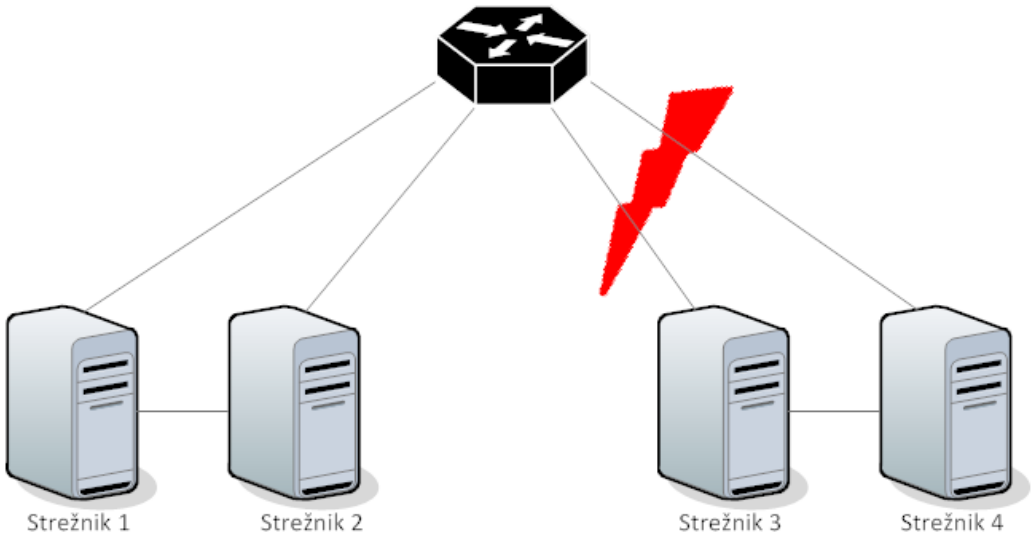
Eric Brewer je leta 2000 predstavil CAP teorem, ki je danes široko sprejet v skupnosti NoSQL. Kratica CAP je okrajšava za tri lastnosti: konsistentnost (*Consistency*), razpoložljivost (*Availability*) in particijsko toleranco (*Partition Tolerance*). Teorem zagovarja stališče, da porazdeljene podatkovne baze lahko zagotavljajo le dve izmed treh lastnosti.

Potrebno je opozoriti, da **konsistentnost** v CAP ni ista kot konsistentnost v ACID. Pri ACID pomeni, da se podatki, ki ne ustrezajo določenim omejitvam, ne shranijo. V primeru CAP pa se konsistentnost nanaša na atomarnost in izolacijo branj in pisanj. Konsistentne operacije branja in pisanja pomenijo, da sočasne operacije vidijo iste podatke, ki so v tistem času veljavni in konsistentni [34].

CAP dodatno vpeljuje tudi pojem šibke ali eventuelne konsistentnosti, ki pomeni, da ko so podatki zapisani, bodo nekateri uporabniki še vedno dostopali do starejše verzije podatkov.

Razpoložljivost, še posebej pa visoka razpoložljivost, pomeni, da je sistem zasnovan in implementiran na takšen način, da omogoča neprekinjeno izvajanje operacij (branja in pisanja), tudi če pride do izpada vozlišč v gruči, ali pa v času vzdrževanja in nadgrajevanja strojne in programske opreme.

Particijska toleranca v porazdeljeni podatkovni bazi pomeni, da lahko iz podatkovne baze še vedno beremo in vanjo pišemo, čeprav so njeni deli popolnoma nedostopni (slika 2.6). Vzroki za nastanek t.i. particij vozlišč je lahko prekinjena omrežna povezava med velikimi števili vozlišč podatkovne baze ali pa okvara strojne opreme. Za dosego particijske tolerance potrebujemo veliko replik. Več replik imamo, boljše možnosti obstajajo, da bo podatek razpoložljiv, tudi če so nekatera vozlišča nedelujoča ali nedostopna.



Slika 2.6. Nastanek particij vozlišč

Particijsko toleranco je mogoče doseči z mehanizmom, ki omogoča, da se pisanja, namenjena nedostopnim vozliščem, pošljejo na dostopna vozlišča. Ko so odpovedana vozlišča ponovno vzpostavljena, prejmejo pisanja, ki so jih zamudila. Podatkovna baza z močno particijsko toleranco se lahko razteza čez več podatkovnih centrov, medtem ko je tista s šibko particijsko toleranco vezana na en sam podatkovni center.

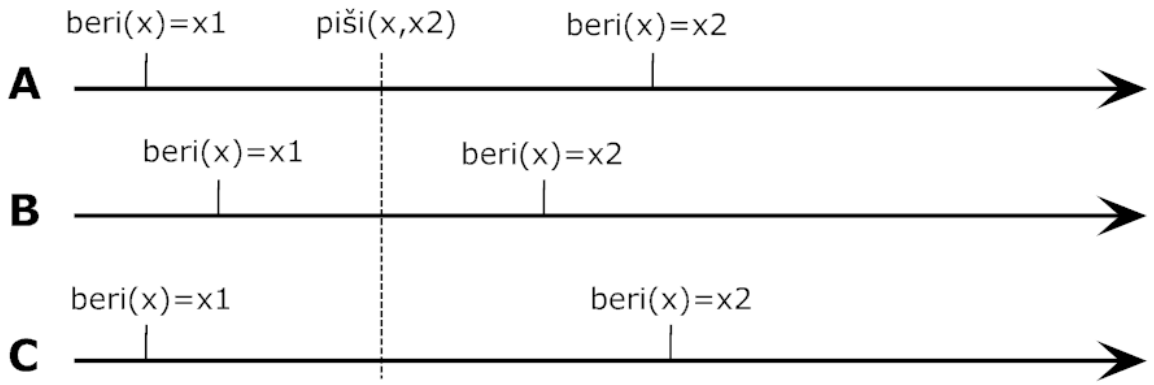
2.5.4.1 Eventuelna konsistentnost

Konsistentnost, ki jo opredeljuje ACID, je veliko ozko grlo za skalabilnost, zato je prišlo do pojava eventuelne konsistentnosti. Za pomoč pri pojasnitvi močne in eventuelne konsistentnosti bo uporabljena naslednja terminologija [27]:

A, B, C	Neodvisni procesi, ki nameravajo brati ali spremeniti podatkovno bazo
x	Podatkovni element x
x1, x2, x3	Različne vrednosti elementa x v podatkovni bazi
piši(x, Vrednost)	Pisalna operacija določenega procesa nad podatkovno bazo

beri(x) = Vrednost Bralna operacija določenega procesa nad podatkovno bazo

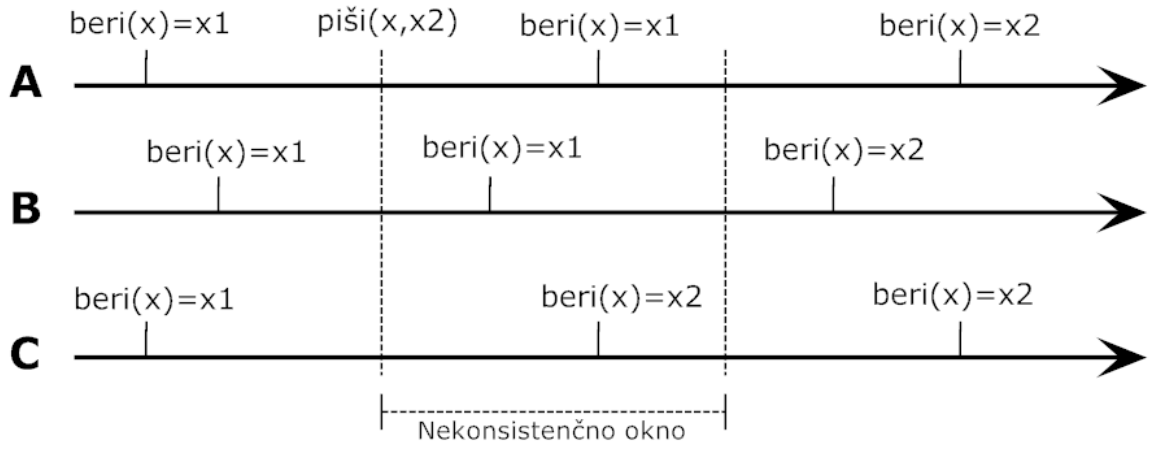
Močna konsistentnost pomeni, da bodo vsi procesi povezani s podatkovno bazo vedno videli isto različico vrednosti. Potrjena vrednost se v tem primeru takoj odraža pri bralni operaciji, dokler ni spremenjena s pisalno operacijo (slika 2.7 [27]).



Slika 2.7. Močna konsistentnost

Eventuelna konsistentnost je šibkejša in ne zagotavlja, da vsak proces vidi enako različico podatkovnega elementa. Tudi proces, ki zapisuje vrednost, lahko dobi staro različico, ko se nahaja v t.i. »nekonsistenčnem oknu« (*Inconsistency window*). Eventuelna konsistentnost je običajno posledica replikacij podatkov preko različnih vozlišč. Čeprav je lahko podatek v nekem času zastarel, bodo sčasoma vse replike konsistentne.

Slika 2.8 [27] prikazuje eventuelno konsistentnost. Procesi A, B in C lahko vidijo različne verzije podatkovnega elementa med odprtim nekonsistenčnim oknom, ki ga povzroči asinhrona replikacija.



Slika 2.8. Eventuelna konsistentnost

Ko proces A piše x2, traja nekaj časa, dokler nova vrednost ni replicirana na vsako vozlišče, ki je odgovorno za ta podatkovni element. Če bi podatkovna baza čakala na rezultat pisalne operacije, dokler nova vrednost ne bi bila v celoti replicirana, potem bi blokirala čakajoči proces B in bi lahko povzročila zakasnitev za odjemalca. Poleg tega bi se lahko zgodilo, da vsa vozlišča ne bi bila dostopna zaradi težav s povezavo ali okvare strojne opreme, kar bi lahko blokiralo celotno aplikacijo.

Eventuelna konsistentnost je uporabna, če so sočasne posodobitve istih particij podatkov malo verjetne, in če odjemalci niso takoj odvisni od branja posodobitev narejenih s strani drugih odjemalcev. Poleg tega izbira konsistentnega modela za sistem (ali del sistema) nakazuje, kako so zahteve odjemalcev odposlane replikam kot tudi, kako so replike razmnožijo in uveljavijo spremembe.

Eventuelno konsistenten sistem lahko odjemalcem zagotavlja tudi več različnih dodatnih zagotovil glede konsistentnosti.

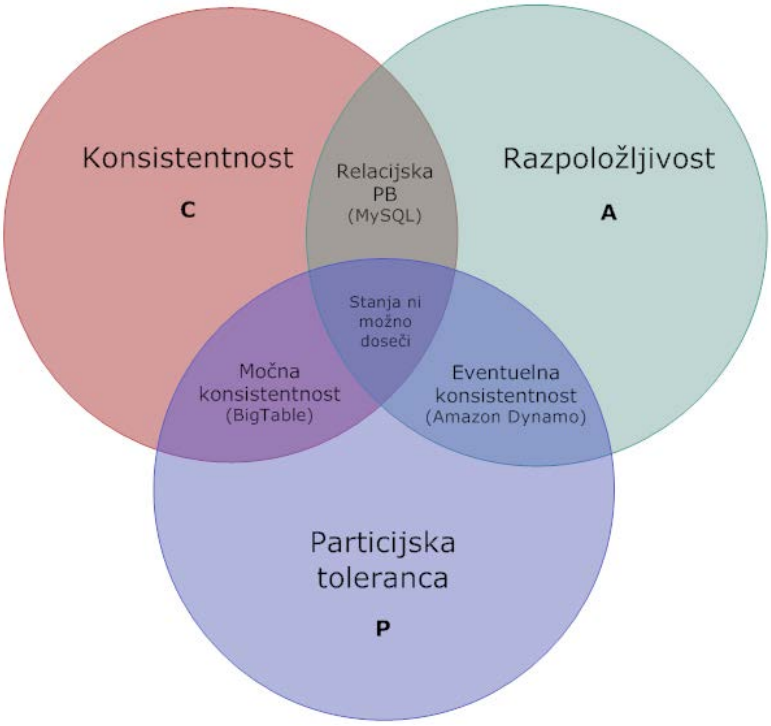
Nekatere porazdeljene podatkovne baze lahko zagotovijo t.i. **konsistentnost beri-svoja-lastna-pisanja** (*Read-your-own-writes*). Ta vrsta konsistentnosti omogoča, da odjemalec vidi svoje posodobitve takoj, ko so bile izdane in zaključene, ne glede na to, če je pisal na en strežnik in je nato bral iz različnih strežnikov.

Konsistentnost seje (*Session Consistency*) je konsistentnost tipa beri-svoja-lastna-pisanja, ki je omejena na posamezno sejo (običajno je vezana na en strežnik), tako da odjemalec vidi svoje posodobitve takoj, samo če bere znotraj iste seje, v kateri jih je zapisal. Če proces začne novo sejo, bi lahko v času nekonsistenčnega okna videl starejšo verzijo podatka, ki ga je prej zapisal.

Tretja različica eventuelne konsistentnosti je **enakomerna konsistentnost branj** (*Monotonic Read Consistency*), ki zagotavlja, da ko je na novo zapisana vrednost prebrana prvič, vsa nadaljnja branja tega podatkovnega vrnejo zadnjo verzijo. Ta vrsta konsistentnosti omogoča, da podatkovna baza replicira novejše zapisane podatke, preden dovoli uporabnikom, da vidijo novo različico.

2.5.4.2 Sprejemanje kompromisov pri CAP

Izberemo lahko samo med dvema lastnostma, pri tretji je potrebno sprejeti kompromis. Ne obstaja takšno stanje, kjer bi dobili vse tri lastnosti. Zato je potrebno prepoznati, katero od CAP lastnosti naša aplikacija potrebuje. Slika 2.9 [4] prikazuje prekrivanje CAP lastnosti.

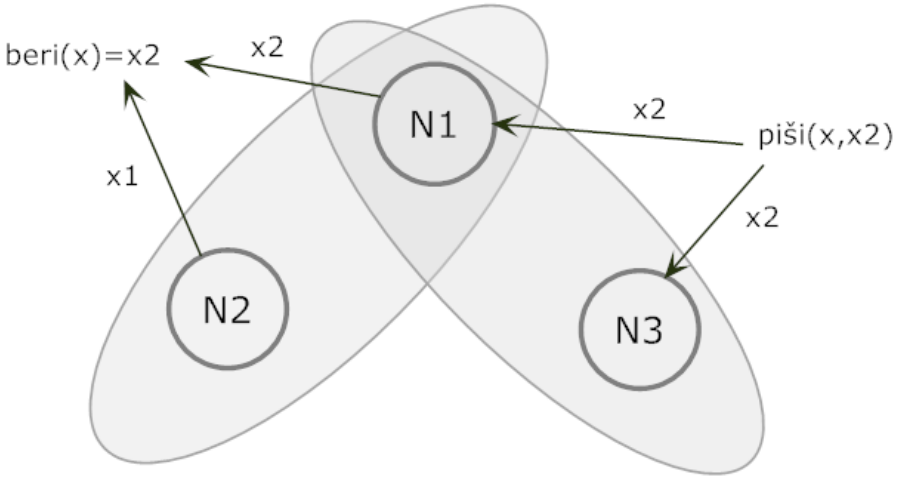


Slika 2.9. Sprejemanje kompromisov pri CAP

Za bolj podroben opis bomo uporabili naslednjo terminologijo:

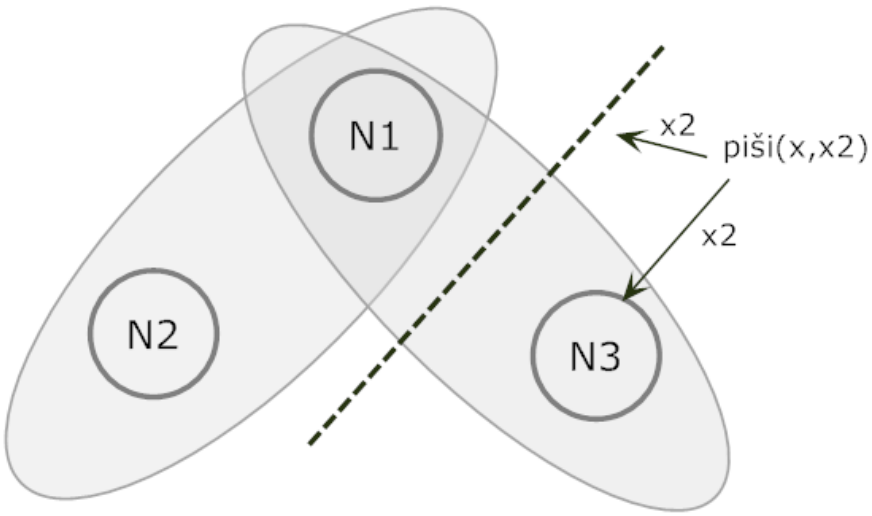
N število vozlišč, na katera je repliciran podatkovni element
R število vozlišč, iz katerih je potrebno prebrati vrednost, da je ta pravilna
W število vozlišč, na katera mora biti zapisana nova vrednost predno se pisalna operacija zaključi

Za uveljavitev močne konsistentnosti je potrebna zadostitev pogoja R+W>N. Tako lahko zagotovimo, da bralna operacija vedno bere iz vsaj enega vozlišča z zadnjo vrednostjo. Slika 2.10 [27] prikazuje primer konfiguracije z N=3, R=2 in W=2. Nova vrednost x2 je zapisana na dve vozlišči in prebrana iz dveh vozlišč. Posledica tega je prekrivanje bralnih in pisalnih operacij na enem vozlišču, tako da ima bralna operacija vedno konsistentno vrednost.



Slika 2.10. Pogoj za doseganje močne konsistentnosti

V primeru težav s povezavo, so vozlišča razdeljena na dve particiji kot je prikazano na sliki 2.11 [27]. V tem primeru ima podatkovna baza dve možnosti za rešitev problema: ali zavrne vse pisalne ali bralne operacije, ki bi lahko povzročile nekonsistentnost, ali pa daje prednost razpoložljivosti pred konsistentnostjo. V drugem primeru mora podatkovna baza najti način za rešitev nekonsistentnosti, ko vozlišča niso več ločena. Mehanizem, ki to omogoča se imenuje MVCC (*Multiversion Concurrency Control*) in bo razložen v nadaljevanju.



Slika 2.11. Problem pisanja in branja pri nastanku particij

Sistemi z eventuelno konsistentnostjo izpolnjujejo samo pogoj R+W≤N, kar pomeni, da so particijsko tolerantni in razpoložljivi. Pri tem pa imajo lahko nekateri sistemi različne vrednosti za R in W, da podatkovno bazo prilagodijo za različne scenarije uporabe [34]:

- Velik R in majhen W sta dobra za tiste aplikacije, ki več pišejo kot berejo. Ko zadošča pisanje na eno vozlišče, je možnost podatkovne nekonsistentnosti precej visoka. Vendar je pri scenariju, ko je R=N, branje možno samo, ko so vse replike v gruči razpoložljive.

- Majhen R in velik W sta dobra za aplikacije z več bralnimi operacijami. Ko ima sistem več branj kot pisanj, je smiselno uravnovesiti obremenitev branj na vse replike v gruči. V primeru, da je R=1, je vsako vozlišče neodvisno od kateregakoli drugega vozlišča, kar zadeva bralne operacije. Ko je W=N, se posodobitve zapišejo na vse replike.

- Majhna R in W tako uveljavljata razpoložljivost, vendar zmanjšujeta konsistentnost.

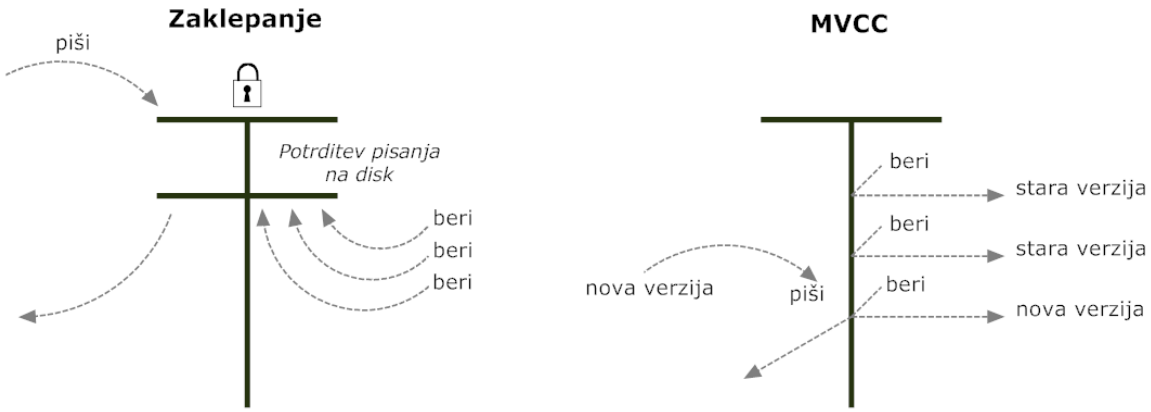
2.5.5 MVCC

Podatkovna struktura relacijskih podatkovnih baz je tabela. Če želimo posodobiti vrstico tabele, mora sistem podatkovne baze zagotoviti, da nihče drug ne poskuša posodobiti iste vrstice, hkrati pa nihče ne sme brati iz te vrstice, medtem ko se posodablja.

Pogost način za reševanje tega problema je t.i. zaklepanje. Če želi več odjemalcev hkrati dostopati do tabele, jo zaklene le prvi odjemalec, vsi ostali morajo čakati. Ko je zahteva prvega odjemalca obdelana, dostop dobi naslednji odjemalec, medtem ko vsi ostali čakajo, itd. To serijsko izvajanje zahteva veliko procesorske moči strežnika, če zahteve odjemalcev prispejo vzporedno. V primeru visoke obremenitve lahko relacijska podatkovna baza preveč časa ugotavlja, komu je kaj dovoljeno in v kakšnem zaporedju, kot pa da bi opravljala dejansko delo.

Namesto zaklepanja nekatere NoSQL baze (npr. CouchDB, Riak, Voldemort) uporabljajo **MVCC** (*Multi-version Concurrency Control*). MVCC je mehanizem za nadzor nad sočasnim izvajanjem transakcij nad večimi verzijami podatkov [27]. Omogoča večim procesom vzporeden dostop do istih podatkov, ne da bi prišlo do mrtvih zank in pokvarjenih podatkov. Ker se zahteve lahko izvajajo vzporedno, to omogoča tudi boljšo izrabo procesorske moči strežnika.

MVCC se uporablja tako v nekaterih relacijskih, kot tudi v večini nerelacijskih podatkovnih bazah. Slika 2.12 [4] prikazuje razliko med MVCC in tradicionalnim mehanizmom zaklepanja.

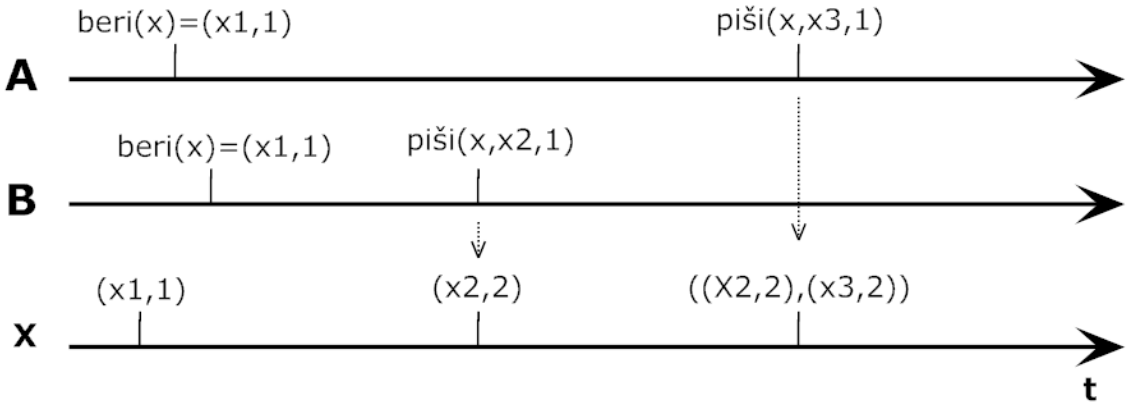


Slika 2.12. Primerjava med zaklepanjem in MVCC

Namesto da bi dovolil vsakemu procesu ekskluzivni dostop do podatkov za določen čas, MVCC omogoča procesom vzporedno branje podatkov, tudi če proces podatek posodablja.

Na primer, imamo skupino zahtev, ki čakajo na dostop do podatka. Prva zahteva prebere podatek, med njenim branjem pa druga zahteva spremeni isti podatek. Ker druga zahteva vključuje popolnoma novo verzijo podatka, se jo preprosto zapiše v podatkovno bazo brez čakanja, da se prva zahteva izvede do konca. Ko pride tretja zahteva, prebere novo verzijo podatka, medtem pa lahko prva zahteva še vedno bere staro verzijo podatka.
Bralna zahteva bo tako ob prihodu vedno videla najnovejši podatek v podatkovni bazi.

Nastanek konfliktov pri MVCC
Da se ohranja konsistentnost, ima vsak podatek neke vrste časovni žig ali številko spremembe (*Revision*). Če proces prebere podatek, ne dobi samo njegove vrednosti, ampak tudi pripadajočo številko spremembe. Torej, če ta proces poskuša posodobiti ta podatek, potem zapiše novo vrednost s prejšnjo prebrano številko spremembe v podatkovno bazo. Če je dejanska sprememba v shrambi enaka, potem se zapiše nova vrednost in številka spremembe podatka se poveča. Če pa sprememba v shrambi ni enaka spremembi, ki jo je prebral pisalni proces, potem je v tem času moral nek drug proces podatek posodobiti, zato pride do konflikta. V relacijski podatkovni bazi bi bil pisalni proces v tem primeru preklican ali ponovno izveden. Slika 2.13 [27] prikazuje primer nastanka konflikta v primeru uporabe MVCC.



Slika 2.13. Nastanek konflikta pri MVCC

Procesa A in B poskušata pisati novo vrednost podatkovnega elementa x. Oba najprej prebereta element x skupaj z njegovo trenutno številko spremembe. Proces B prvi zapiše novo verzijo v podatkovno bazo. Proces A poskuša pisati novo verzijo x, ki temelji na starejši verziji od trenutne, in tako ustvari konflikt.

Načini za reševanje konfliktov

V porazdeljenih podatkovnih bazah obstajata vsaj dva primera za takšen konflikt. Prvi je, da dva procesa poskušata pisati isti podatek v isto vozlišče. V tem primeru lahko podatkovna baza zazna konflikt med pisalno operacijo in jo prekine. Tako bi moral odjemalec ponovno prebrati podatek in znova poskusiti izvesti želeno posodobitev, tako kot se to izvede v relacijskih podatkovnih bazah.

Drug primer je, da več odjemalcev posodobi isti podatek na različnih vozliščih. Če porazdeljena podatkovna baza uporablja asinhrono replikacijo, potem tega konflikta ni mogoče zaznati med pisalnimi operacijami. Vozlišča se morajo naprej sinhronizirati preden lahko obravnavajo konflikt. Konflikt se lahko razreši med replikacijo ali med prvo bralno operacijo podatka v konfliktu.

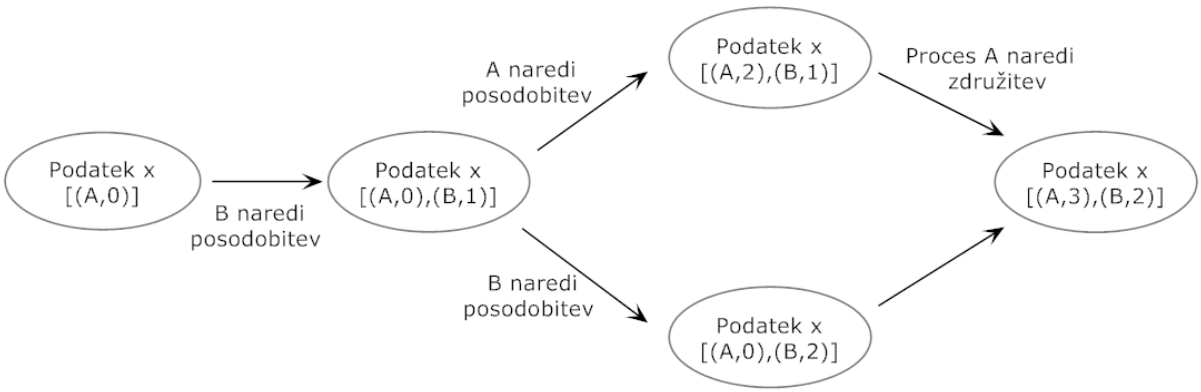
Nekatere podatkovne baze shranjujejo vse konfliktne spremembe podatkov in omogočijo odjemalcem odločitev, kako naj se konflikt obravnava. V takšnih sistemih bralne zahteve vrnejo vse konfliktne verzije vrednosti, med katerimi odjemalec izbere eno ali pa združi različne verzije in zapiše popravljeno številko spremembe nazaj v podatkovno bazo.

2.5.6 Vektorske ure

Pri eventuelni konsistentnosti je potrebno določiti ali je bilo več verzij istega podatka ustvarjenih v vzporednem ali zaporednem vrstnem redu. Poleg tega mora podatkovna baza vedeti, katera verzija je najnovejša. Preprosta rešitev za to bi bila uporaba časovnih žigov, ampak v tem primeru bi bilo potrebno, da so vsa vozlišča časovno sinhronizirana in lahko bi se zgodilo, da bi dve vozlišči pisali isti podatek točno ob istem času. Da bi se izognili temu problemu, uporabljajo nekatere porazdeljene podatkovne baze mehanizme za sledenje verzijam podatkov. Ena od njih so vektorske ure [27].

Vektorske ure (*Vector Clocks*) so uporabne za določitev, ali so podatek spremenili sočasni procesi. Vsak podatek vsebuje vektorsko uro, ki je sestavljena iz n-teric (*Tuples*) z ločeno uro za vsak proces, ki je spremenil podatkovni element. Vsaka ura se začne z nič, poveča pa jo njen lastni proces med vsako pisalno operacijo. Za povečanje svoje lastne ure uporablja pisalni proces največjo vrednost ure v vektorju, ki jo poveča za ena. Ko se dve verziji elementa združita, se lahko vektorske ure uporabi za detektiranje konfliktov s primerjavo ur za vsak proces. Če se razlikuje več kot ena ura, potem je zagotovo prišlo do konflikta. Če ni konflikta, potem se trenutna verzija podatka določi s primerjavo največjih vrednosti ur konkurenčnih verzij.

Slika 2.14 [27] prikazuje primer vektorskih ur vključno s podatkom x in dvema procesoma A in B.



Slika 2.14. Detekcija konflikta z vektorskimi urami

Na začetku ima podatek x vektorsko uro ([A;0]), ker ga je sprva zapisal proces A. Proces B posodobi podatek in zažene svojo uro za ta podatek. Ker je bil zadnji maksimum verzije x 0, proces nastavi svojo uro za x na 1. Ker sta A in B pisala v podatek eden za drugim, konflikt ni bil zaznan. V naslednjem koraku oba procesa pišeta vzporedno v x. Vzporedno ne pomeni vedno, da procesa pišeta v x ob točno istem času. Možno je, da pišeta v različni repliki, ki sta lahko kasneje sinhronizirani. Oba procesa povečata vektorsko uro na opisan način. Ob naslednji sinhronizaciji se ti dve vektorski uri razlikujeta na dveh položajih, zato je detektiran konflikt, ki ga mora potem obravnavati odjemalec.

Pomanjkljivost vektorskih ur je, da postanejo vedno večje z vsakim procesom, ki piše v podatek.

2.5.7 MapReduce

MapReduce je ogrodje za vzporedno programiranje, ki omogoča porazdeljeno obdelavo velikih naborov podatkov na gruči računalnikov [34]. Patentiral ga je Google, vendar so njegovi koncepti prosto dostopni in prisotni v številnih odprtokodnih implementacijah kot je npr. zelo priljubljen Hadoop.

Programer z uporabo ogrodja MapReduce napiše dve funkciji – funkcijo map in funkcijo reduce – od katerih vsaka določa preslikavo iz enega nabora parov tipa ključ/vrednost v drugega.

map(key,value) → list (ikey, ivalue)
reduce(ikey, list(ivalue)) → list(fvalue)

Funkcija *map* se kliče s parom tipa ključ/vrednost za vsak podatkovni zapis in vrne vmesne pare tipa ključ/vrednost, ki jih nato funkcija *reduce* uporabi za združitev vseh vmesnih vrednosti z enakim vmesnim ključem. Torej vsak klic funkcije *reduce* dobi vmesni ključ in seznam vrednosti, ki pripadajo ključu, in izvede agregacijo čez vse pare ter vrne manjši nabor podatkov ali eno vrednost kot končni rezultat.

Recimo, da imamo sledečo zbirko parov tipa ključ/vrednost:

[{"1000" : "Janez"}], {"1000" : "Jana"}, {"2000" : "Matjaž"}, {"3000" : "Ivan"}]

To je zbirka parov tipa ključ/vrednost, kjer ključ predstavlja poštno številko in vrednost ime osebe, ki prebiva na tej poštni številki. Preprosta funkcija map na tej zbirki bi lahko pridobila vsa imena tistih, ki prebivajo na določeni poštni številki. Izhod takšne funkcije map je sledeč:

[{"1000": ["Janez", "Jana"]}, {"2000" : ["Matjaž"]}, {"3000" : ["Ivan"]}]

Funkcija reduce bi lahko pri tem izhodu preprosto preštela število oseb, ki pripadajo določeni poštni številki. Končni izhod bi izgledal takole:

[{"1000" : 2}, {"2000" : 1}, {"3000" : 1}]

Funkciji map in reduce sta neodvisni od velikosti podatkov ali gruče, na kateri se izvajata, zato se jih lahko uporablja nespremenjene za nabore podatkov poljubnih velikosti.
Če se velikost vhodnih podatkov podvoji, bo delo sicer potekalo dvakrat počasneje, vendar če se podvoji tudi velikost gruče, bo delo potekalo tako hitro kot prej. To v splošnem ne velja za poizvedbe na osnovi jezika SQL [35].

Aplikacije, ki so napisane z uporabo ogrodja MapReduce, so lahko samodejno porazdeljene čez več računalnikov brez potrebe, da razvijalec napiše posebno kodo za sinhronizacijo in vzporedno izvajanje. Lahko se uporablja za opravljanje nalog na velikih naborih podatkov, ki bi bili preveliki za obdelavo na enem stroju.

Obdelave MapReduce se lahko neodvisno izvajajo in ponovno zaženejo. Obstaja tudi možnost špekulativnega izvajanja (npr. pri implementaciji Hadoop), ki poskrbi, da se obdelava namesto na enem počasnem vozlišču izvede na enem ali več drugih, ki so hitrejši [36].

Poglavje 3

Primerjava med relacijskimi in NoSQL podatkovnimi bazami

Čeprav NoSQL podatkovne baze prinašajo veliko prednosti, to ne pomeni, da bi morali relacijske podatkovne baze zavreči. Relacijske in NoSQL podatkovne baze se med seboj bistveno razlikujejo in so izdelane za različne potrebe. Relacije in ACID transakcije so v določenih primerih uporabe, kot so bančni sistemi in borza, še vedno potrebne in nujne. Podatki morajo biti namreč tu vedno pravilni in razpoložljivi, saj pri upravljanju s finančnimi podatki ne sme priti do napak. Na drugi strani pa imamo na primer aplikacije za družabna omrežja, kjer sta bolj kot konsistentnost pomembni skalabilnost in visoka razpoložljivost. Če se naša zadnja objava pojavi na strani šele čez nekaj minut, to ni kritičnega pomena.

Samo podrobna primerjava obeh tipov podatkovnih baz nas pripelje do popolnega razumevanja njunih prednosti in pomanjkljivosti.

3.1 Podatkovni model

Pri relacijski podatkovni bazi moramo pred delom s podatki najprej določiti shemo, kar vključuje:

- shemo vsake tabele, njenih atributov (stolpcev) in razmerij med entitetami,
- določitev indeksov za potrebe poizvedovanja.

Relacijski model ne uveljavlja posebnega načina za delo s podatki – zgrajen je s poudarkom na celovitosti in normalizaciji podatkov, preprostosti in abstrakciji, ki so vsi izjemno pomembni za velike kompleksne aplikacije.

NoSQL podatkovne baze ne zahtevajo enotne podatkovne sheme in omogočajo spreminjanje le-te brez izpada sistema. Atribute lahko poljubno dodajamo, podatki pa lahko zavzamejo poljubne vrednosti. Prav tako ni relacij med podatki. Podatki, ki so povezani, so običajno pogrupirani in shranjeni kot enota (npr. dokument, stolpec). Podatkovne baze NoSQL tako omogočajo veliko večjo prilagodljivost podatkovnega modela kot relacijske podatkovne baze, je pa skrb za podatke brez sheme prepuščena aplikaciji, ki mora vedeti, kaj ti podatki predstavljajo ter kje in kako so shranjeni.

Podatkovne baze NoSQL imajo podatkovne strukture, ki so različne od tistih v relacijskih podatkovnih bazah, kjer je enotna podatkovna struktura tabela. Razlog je ta, da so v relacijskih podatkovnih bazah stolpci definirani na enem centralnem mestu. To posledično otežuje horizontalno skalabilnost, saj bi bilo potrebno v primeru spremembe sheme ustaviti vsa vozlišča v gruči.

Podatkovne baze NoSQL omogočajo shranjevanje podatkov, ki so v razmerju, znotraj enega zapisa. Vendar morajo biti za ta namen podatki denormalizirani. V primeru hierarhičnih podatkov je taka denormalizacija sprejemljiva, ni pa primerna, če med podatki obstajajo kompleksna razmerja. Razlog je, da pri NoSQL celovitost podatkov ne more biti zagotovljena na nivoju podatkovne baze, ker se podatki particionirajo po vozliščih. Ni zagotovil za unikatnost podatkov, referenčne celovitosti in omejitev celovitosti.

Primerjavo med obema podatkovnima modeloma povzema spodnja preglednica 3.1.

Preglednica 3.1. Primerjava podatkovnih modelov relacijskih in NoSQL podatkovnih baz

	Podatkovni model	
	Relacijski	**NoSQL**
Podatkovna struktura	Uniformna - tabela (stolpci in vrstice). Vrstice znotraj neke tabele imajo isto shemo.	Različna (par ključ/vrednost, dokument, razširljiv zapis,...).
Podatkovna shema	Potrebna.	Ni potrebna.
Struktura podatkov	Podatki so strukturirani. Struktura je vezana na vsebino podatkov.	Podatki so polstrukturirani in nestrukturirani. Struktura je vezana na funkcionalnost aplikacije.
Razmerja	So temelj relacijskega podatkovnega modela.	Nobeno razmerje ni izrecno definirano med podatki.
Celovitost podatkov	Zagotavlja podatkovna baza.	Mora zagotoviti programer v aplikaciji.

Spremembe podatkovnega modela je pri NoSQL mogoče doseči precej enostavneje kot pri relacijskih podatkovnih bazah. Vozlišče lahko enostavno vzamemo iz gruče in ga dodamo

nazaj v gručo kot nov strežnik, ki ima vlogo gospodarja. Mehanizem replikacije bo poskrbel za sinhronizacijo podatkov in razmnožitev spremenjenega podatkovnega modela na ostala vozlišča v gruči.

Še ena razlika med relacijskimi in NoSQL podatkovnimi bazami je strukturiranost naborov podatkov. Za relacijske podatkovne baze so značilni strukturirani podatki, v NoSQL pa so lahko podatki tudi polstrukturirani in nestrukturirani.

3.1.1 Stiki

Medtem ko so stiki ena izmed osnovnih operacij v relacijskih podatkovnih bazah, se jim podatkovne baze NoSQL izogibajo. Razlog je, da stične operacije ni možno omogočiti na skalabilen način, če imamo ogromno podatkov, ki so porazdeljeni čez več 10 vozlišč. Podatkovni strežnik mora namreč opraviti zapleten nabor operacij na velikih količinah porazdeljenih podatkov.

3.1.2 Podatkovno modeliranje

V nasprotju z NoSQL, relacijska podatkovna baza zahteva proces podatkovnega modeliranja. To je proces, ki zagotavlja, da relacije (tabele) ne bodo vsebovale redundantnih ali dvoumnih podatkov, ki ne bodo predmet nepravilnosti pri vnosu, brisanju in popravljanju le-teh. V primeru, da je ta proces dobro opravljen, imamo v podatkovni bazi logično strukturo, ki bolj odraža strukturo podatkov, ki jih bo vsebovala, ne pa toliko strukture aplikacije. Podatki so tako neodvisni od aplikacije, kar pomeni, da lahko isti nabor podatkov uporabljajo tudi druge aplikacije, prav tako pa lahko spremenimo logiko aplikacije, brez da bi to vplivalo na podatkovni model. Na voljo imamo tudi več CASE (*Computer-aided software engineering*) orodij za izdelavo podatkovnega modela.

3.2 Dostop do podatkov

Značilnosti načinov dostopa do podatkov v relacijskih podatkovnih bazah in podatkovnih bazah NoSQL so predstavljene v spodnji preglednici 3.2 [5].

Preglednica 3.2. Primerjava dostopa do podatkov pri relacijskih in NoSQL podatkovnih bazah

Dostop do podatkov	
Relacijska PB	**NoSQL PB**
Operacije shranjevanja, posodabljanja, brisanja in poizvedovanja z uporabo SQL.	Operacije shranjevanja, posodabljanja, brisanja in poizvedovanja izpostavlja API vmesnik.
SQL poizvedbe lahko dostopajo do podatkov iz ene ali več tabel s pomočjo stikov.	Nekatere implementacije zagotavljajo za opredeljevanje kriterijev za filtriranje preprost poizvedovalni jezik podoben SQL.
SQL poizvedbe vključujejo funkcije za agregacijo in kompleksno filtriranje.	Običajno omogoča samo osnovno filtriranje (>,<,=).
Podpirajo vgraditev logike blizu shranjenih podatkov (prožilci, shranjene procedure, SQL predprocesirani paketi in funkcije).	Vsa aplikacijska logika in logika za celovitost podatkov sta vsebovani v kodi aplikacije.

3.2.1 Aplikacijski vmesnik

Razlike v aplikacijskih vmesnikih, ki jih uporabljata obe vrsti podatkovnih baz, so prikazane v spodnji preglednici 3.3 [5].

Preglednica 3.3. Primerjava aplikacijskega vmesnika relacijskih in NoSQL podatkovnih baz

Aplikacijski vmesnik	
Relacijska PB	**NoSQL PB**
Ponavadi imajo svoj specifičen API ali izkoristijo generičen API kot npr. OLE-DB ali JDBC.	Za dostop do podatkov ponavadi zagotavljajo API-je, ki temeljijo na standardih kot so SOAP, REST, THRIFT.
Potrebna je preslikava iz relacijske strukture podatkov v strukturo, ki je primerna za uporabo v aplikaciji.	Struktura podatkov v podatkovni bazi je analogna strukturi, ki jo uporablja programski jezik oziroma aplikacija.

3.2.2 Poizvedovalni jezik

Relacijske podatkovne baze uporabljajo zelo razširjen in priljubljen poizvedovalni jezik SQL, ki omogoča skoraj neomejeno indeksiranje ter enostavne in ad hoc poizvedbe kot tudi dinamične in kompleksne poizvedbe preko več tabel. Osnovne operacije, ki jih SQL podpira so:

- Stiki (JOIN)
- Filtri (WHERE)
- Funkcije

- Agregacije (GROUP BY, AVG, COUNT, MAX, MIN…)
- Skalarne (ABS, DAY, DATE, TIME, POWER, SUBSTR…)

Podatkovne baze NoSQL imajo sicer lahko omejeno podporo dinamičnim poizvedbam in indeksiranju (npr. MongoDB), vendar za razliko od SQL ne podpirajo stikov, bolj kompleksnih filtrov in agregacij, zato morajo biti te operacije implementirane v aplikacijski kodi. V splošnem uporabljajo preprost vmesnik, ki podpira tri osnovne operacije:

- get(key) – vrne vrednost, ki pripada danemu ključu
- put(key, value) – shrani ali posodobi vrednost skupaj s pripadajočim ključem
- delete(key) – izbris vrednosti in pripadajočega ključa

Nekatere podatkovne baze NoSQL imajo na voljo tudi svoj poizvedovalni jezik, ki je podoben SQL (npr. MongoDB, Hypertable, Cassandra).

Kljub temu je ena od močno pogrešanih funkcij NoSQL ravno SQL, in če želijo biti podatkovne baze NoSQL tako široko uporabljene kot relacijske, bodo verjetno morale poiskati neko podobno alternativo. Ponudniki rešitev NoSQL vedno bolj stremijo k poizvedovalnim jezikom, ki so podobni SQL, kajti zavedajo se, da če ne bodo našli nekega skupnega nabora operacij nad podatki, potem obstaja verjetnost, da bo ena ali druga implementacija postala bolj priljubljena in se bodo uporabniki ali preselili k produktu, ki rešuje njihov problem, ali pa bodo morali vsi ponudniki implementirati svoj nabor operacij, ki bo konkurenčen.

Primer poizkusa razvoja skupnega poizvedovalnega jezika za podatkovne baze NoSQL predstavlja jezik UnQL (*Unstructured Data Query Language*) [23].

UnQL je produkt dveh podjetij, CouchDB in SQLite. Razčleni lahko vse izjave jezika SQL in podpira številne nove izjave, operatorje in tudi izraze, ki se jih lahko razširi za bolj kompleksne dokumente. Tako kot SQL je bil osnovan na temelju relacijske algebre. Če bi ga sprejeli ostali ponudniki NoSQL rešitev, bi UnQL lahko predstavljal enotni vmesnik za podatkovne baze NoSQL. Ena izmed podatkovnih baz, ki UnQL že uporablja, je Couchbase 2.0 [17].

Za hitro obdelavo ogromnih količin podatkov ima veliko podatkovnih baz NoSQL na voljo lastno implementacijo ogrodja MapReduce.

MapReduce je primeren za probleme, ki potrebujejo paketno analiziranje celotnega nabora podatkov, zlasti za ad hoc analize. Relacijski SUPB je dober za poizvedbe in posodobitve, kjer je nabor podatkov indeksiran za hitro vračanje in posodabljanje relativno majhne količine podatkov. MapReduce ustreza aplikacijam, kjer so podatki zapisani enkrat, in velikokrat

prebrani, medtem ko so relacijske podatkovne baze dobre za nabor podatkov, ki se stalno posodabljajo [35].

Iste funkcionalnosti lahko realiziramo tako z SQL kot tudi z MapReduce, s to razliko, da je v MapReduce to bolj kompleksno. Kljub temu ima pristop MapReduce nekatere večje prednosti pred SQL [28]:

- Programerji, ki ne poznajo SQL ali relacijske teorije, jim je MapReduce lažje razumljiv in ga lahko začnejo hitro uporabljati.
- Funkciji map in reduce sta lahko visoko paralelizirani na poceni strojni opremi.

3.2.3 Sekundarni indeksi

Podatkovne baze NoSQL v splošnem ne podpirajo sekundarnih indeksov (to je indeksov po atributih, ki niso primarni ključ), vendar jih razvijalci lahko implementirajo na svoj način. Pogost pristop je kreiranje ločene tabele, ki vsebuje vse vrednosti stolpca, nad katerim bi navadno ustvarili sekundarni indeks, in pripadajoče ključe vrstic iz glavne tabele.

3.3 Prenosljivost

Podatki v relacijskih podatkovnih bazah so aplikacijsko neodvisni, zato jih lahko uporabljajo tudi druge aplikacije. Prav tako lahko podatke enostavno prenesemo na drugo relacijsko podatkovno bazo, saj vse relacijske podatkovne baze izpostavljajo skupni vmesnik SQL.

Prenosljivost v primeru NoSQL podatkovnih baz pa je problematična, ker vsaka izpostavlja svoj lasten nabor API vmesnikov, knjižnic in jezikov za interakcijo s podatki, ki jih vsebujejo. Z izbiro podatkovne baze NoSQL smo tako omejeni na enega ali več programskih jezikov in načinov dostopa. Organizacije ne morejo preprosto zamenjati podatkovne baze, ne da bi bile potrebne korenite spremembe kode v aplikaciji. Pojavi se problem t.i. priklenitve na ponudnika.

3.4 Konsistentnost

Relacijske podatkovne baze v okviru modela ACID zagotavljajo močno konsistentnost. Podatkovne baze NoSQL pa opuščajo zagotovila ACID, saj jim je pomembnejša skalabilnost, visoka razpoložljivost in zmogljivost, zato v splošnem podpirajo šibko oziroma eventuelno konsistentnost ali pa zagotavljajo konsistentnost samo znotraj objekta (to je zapisa ali dokumenta).

Relacijske podatkovne baze temeljijo na modelu ACID, medtem ko je osnova NoSQL podatkovnih baz model BASE. Razlike med modeloma ACID in BASE so prikazane v spodnji preglednici 3.4 [6,31].

Preglednica 3.4. Primerjava ACID in BASE

ACID	BASE
Konsistentnost je najpomembnejša	Razpoložljivost je najpomembnejša
Močna konsistentnost	Eventuelna konsistentnost
Počasnejše	Hitrejše
Manjša prilagodljivost	Enostavnejši razvoj
Manjša skalabilnost	Večja skalabilnost
Zahtevana podatkovna shema	Podatkovna shema ni zahtevana

NoSQL podatkovne baze lahko podpirajo atomarnost in izolacijo v primeru, ko vsaka transakcija dostopa samo do podatkov znotraj enega objekta (npr. na nivoju ključa: dve operaciji na istem ključu bosta zaporedni, da ne pokvarita parov ključ/vrednost). Transakcije v relacijskih podatkovnih bazah zagotavljajo, da bosta na primer naročilo in postavka naročila posodobljena skupaj, pri NoSQL pa sta lahko del istega objekta in sta prav tako hkrati posodobljena. To sicer odpravlja potrebo po dragih porazdeljenih protokolih potrjevanja, vendar mora biti vsaka logična transakcija, ki se razteza čez več kot en objekt, v aplikaciji razdeljena na ločene transakcije. Sistem zato ne zagotavlja niti atomarnosti niti izolacije. Razvijalec mora tako v sami aplikaciji implementirati željene funkcionalnosti ACID in podatkovno celovitost [2].

3.5 Razpoložljivost

Za razliko od relacijskih podatkovnih baz, dajejo podatkovne baze NoSQL prednost visoki razpoložljivosti pred konsistentnostjo. Poleg replikacije na več vozlišč in sposobnosti obnavljanja v primeru odpovedi, je visoka razpoložljivost mogoča tudi zaradi dejstva, da podatkovne baze NoSQL ne zahtevajo sheme. Pri relacijskih bi bilo potrebno za vsako spremembo sheme zaustaviti vsa vozlišča in s tem celotno podatkovno bazo.
Visoka razpoložljivost je med drugim potrebna tudi za doseganje visoke skalabilnosti.

3.6 Skalabilnost

Če imamo eno samo vozlišče, podatkovne baze NoSQL ne prinašajo nobenih prednosti v primerjavi z relacijskimi podatkovnimi bazami. Njihova ključna lastnost in hkrati prednost je visoka in stroškovno učinkovita skalabilnost čez 10 ali več 100 vozlišč, saj naraščajoča količina podatkov, zahteva visoko skalabilnost pri najnižjih možnih stroških.

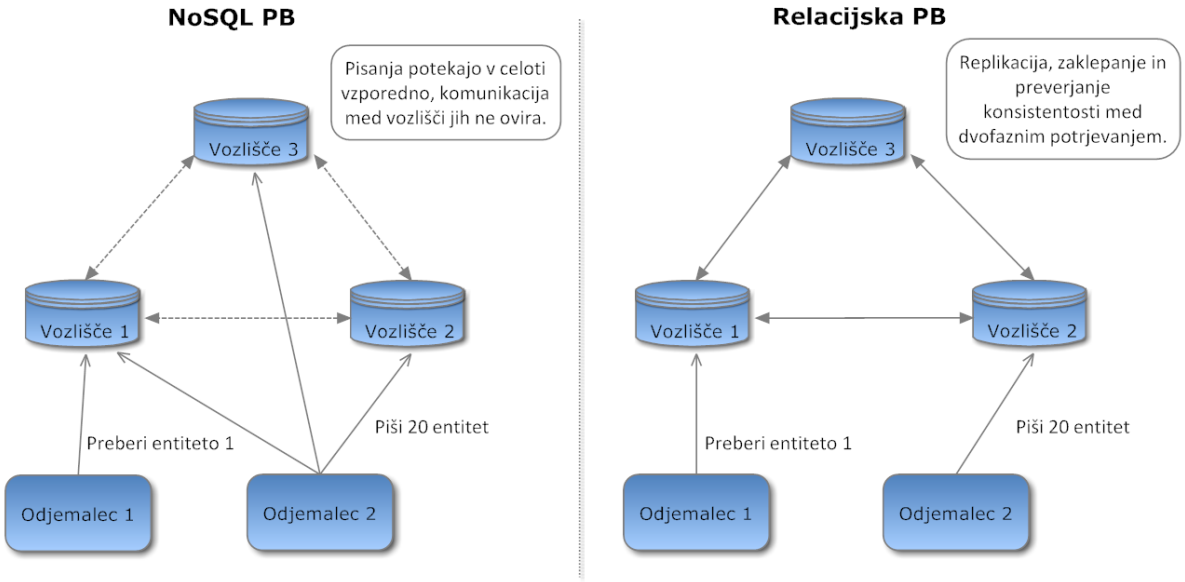
Čeprav so relacijske podatkovne baze prav tako lahko skalabilne z nakupom večjih in hitrejših strežnikov in shrambe ali z zaposlovanjem specialistov, ki poskrbijo za dodatne nastavitve, je to drago in ne tako enostavno, kar bomo pojasnili v nadaljevanju. Kompromis glede lastnosti CAP teorema, to je konsistentnosti, razpoložljivosti in particijske tolerance, pa lahko v primeru NoSQL podatkovnih baz precej zmanjša stroške pri doseganju visoke skalabilnosti.

3.6.1 Problem skalabilnosti pri relacijskih podatkovnih bazah

Horizontalna porazdelitev obremenitve in podatkov oziroma horizontalna skalabilnost je velik problem relacijskih podatkovnih baz. Bralne operacije so sicer lahko skalabilne z replikacijo podatkov na več vozlišč in z uravnoteženjem obremenitev bralnih zahtev. Vendar to ne pride v poštev pri pisalnih operacijah, ker je potrebno vzdrževati podatkovno konsistentnost. Pisalne operacije so lahko skalabilne samo s particioniranjem podatkov. To pa seveda spet vpliva na bralne operacije, saj so porazdeljeni stiki lahko počasni in težki za implementacijo. Poleg tega pa morajo relacijske podatkovne baze za ohranjanje lastnosti ACID zaklepati podatke, kar vpliva na razpoložljivost in zmogljivost.

Dejstvo je, da relacijske podatkovne baze ne morejo doseči samodejne delitve podatkov na enostaven način. To bi zahtevalo posebne podatkovne entitete, ki so lahko neodvisno porazdeljene in obdelane, kar pa tabele ne morejo biti.

Podatkovne baze NoSQL pa logičnih entitet ne porazdeljujejo preko več tabel, ampak so te vedno shranjene na enem mestu. Logična entiteta je lahko karkoli, od preproste vrednosti do kompleksnega objekta ali dokumenta JSON. Podatkovne baze NoSQL ne vsiljujejo referenčne celovitosti med temi logičnimi entitetami. Uveljavljajo samo konsistentnost znotraj ene entitete, včasih pa tudi tega ne.



Slika 3.1. Razlika med skalabilnostjo pri NoSQL in relacijskih podatkovnih bazah

To omogoča samodejno porazdelitev podatkov čez večje število vozlišč in neodvisno pisanje vanje. Če moramo zapisati 20 entitet v gručo podatkovne baze NoSQL s tremi vozlišči, lahko enakomerno razporedimo pisanja na vsa vozlišča (slika 3.1 [24]). Podatkovni bazi za to ni potrebno sinhronizirati vozlišč, prav tako ni potrebe po dvofaznem potrjevanju, kar pomeni, da lahko *odjemalec 1* vidi spremembe na *vozlišču 1* preden *odjemalec 2* zapiše vseh 20 entitet.

Porazdeljena relacijska podatkovna baza na drugi strani pa mora zahtevati konsistentnost ACID na vseh treh vozliščih. To pomeni, da *odjemalec 1* ne bo videl nobenih sprememb, dokler ne bodo vsa tri vozlišča vrnila potrditev ali pa bo blokiran, dokler se dvofazna potrditev ne bo izvedla. Poleg te sinhronizacije mora relacijska podatkovna baza prebrati podatke tudi iz drugih vozlišč, da bi zagotovila referenčno celovitost. Podatkovne baze NoSQL teh operacij ne izvajajo.
Dejstvo, da je takšna rešitev lahko horizontalno skalabilna pomeni tudi, da lahko izkorišča porazdeljenost za omogočanje visoke razpoložljivosti. To je zelo pomembno v oblaku, kjer bi vsako vozlišče lahko izpadlo v vsakem trenutku.

3.7 Količina podatkov

Podatkovne baze NoSQL so bolj učinkovite pri upravljanju z ogromnimi količinami podatkov kot pa relacijske podatkovne baze. Razlog je povezan z že omenjeno večjo skalabilnostjo in opuščanjem zahteve po močni konsistentnosti in zagotovilih ACID.

3.8 Administracija podatkovne baze

Kljub številnim izboljšavam na področju upravljanja relacijskih podatkovnih baz, so še vedno potrebni dragi, visoko usposobljeni administratorji. Ti so tesno vključeni v proces načrtovanja, postavitve in stalnega nastavljanja zmogljivih SUPB-jev.

Podatkovne baze NoSQL pa so že od vsega začetka zasnovane tako, da zahtevajo manj upravljanja. Samodejno popravljanje, porazdelitev podatkov in enostavnejši podatkovni modeli vodijo do manjših potreb po upravljanju. Za izvajanje in razpoložljivost vseh kritičnih delov podatkovnih shramb so odgovorni razvijalci aplikacij.

Poleg hitre skalabilnosti so NoSQL podatkovne baze načrtovane z mislijo na redundanco podatkov, zato razvijalcem ni potrebno poskrbeti za lastne mehanizme za zagotavljanje redundance.

3.9 Cena

Podatkovne baze NoSQL običajno uporabljajo gruče poceni strežnikov, medtem ko se relacijske zanašajo na drage lastniške strežnike in sisteme za shranjevanje. Posledično so stroški na gigabajt ali transakcij na sekundo pri NoSQL lahko velikokrat manjši kot pri relacijskih podatkovnih bazah.
Visoka skalabilnost, nizka zakasnitev in enostavnejši podatkovni model ter model programiranja so vse lastnosti NoSQL, ki jih relacijske podatkovne baze na stroškovno učinkovit način ne zagotavljajo.

3.10 Raven zrelosti

NoSQL podatkovne baze so dokaj nove tehnologije in zaenkrat precej nezrelo področje, kjer se stvari še razvijajo. To pomeni, da mnoge od njih niso tako funkcionalno popolne kot bi morale biti, imajo mnogo hroščev in ne zagotavljajo podpore in varnosti v tolikšni meri kot relacijske podatkovne baze. Pričakuje se, da bodo tovrstne težave odkrite in rešene v naslednjih nekaj letih, ko bodo NoSQL podatkovne baze pridobile priljubljenost in pritegnile pozornost raziskovalcev varnosti. Prav gotovo je v pomoč, da so odprtokodne, saj se koda lahko prenese in analizira.

Relacijske podatkovne baze pa na drugi strani predstavljajo zrelo tehnologijo s širokim naborom funkcionalnosti, ki je bila in še vedno je množično uporabljena za shranjevanje podatkov, in ki vključuje veliko dobrih, zrelih orodij in dobro dokumentacijo.

3.11 Varnost

Za varno shranjevanje informacij mora podatkovna baza zagotavljati [10]:

- zaupnost,
- celovitost in
- razpoložljivost.

Podatki morajo biti dostopni, ko je to potrebno (razpoložljivost), vendar le pooblaščenim posameznikom ali sistemom (zaupnost), ki jih edini lahko spreminjajo (celovitost).

Varnostni mehanizmi pri relacijskih podatkovnih bazah vključujejo varnost na podlagi uporabniških vlog, šifrirane komunikacije, nadzor nad dostopom do vrstic in stolpcev, nadzor dostopa do shranjenih procedur. Relacijski SUPB-ji kot so Oracle, Microsoft SQL Server, IBM DB2 imajo tudi lastnosti ACID, ki zagotavljajo zanesljivo izvrševanje transakcij. Replikacija podatkov in dnevniški zapisi zagotavljajo trajnost in celovitost podatkov. Vendar so ti varnostni mehanizmi dragi v smislu plačevanja licenc pa tudi posledično manjše hitrosti dostopa do podatkov.

Podatkovne baze NoSQL z varnostnega vidika niso tako robustne kot relacijske podatkovne baze. Kot prvo v splošnem zagotavljajo eventuelno konsistentnost podatkov, kar pomeni, da uporabniki lahko dostopajo do zastarelih podatkov. Slabost je tudi pomanjkanje zaupnosti in celovitosti. Ker nimajo sheme, uporabnikom ne moremo določiti dovoljenj za dostop, kot lahko to naredimo pri relacijskih podatkovnih bazah na nivoju tabel in celo stolpcev.

S tem ko same podatkovne baze NoSQL ne zagotavljajo dovolj dobre varnostne podpore, morata biti zaupnost in celovitost v celoti zagotovljeni v okviru aplikacije, ki uporablja podatke.

Horizontalna skalabilnost odpira tudi novo varnostno žarišče, to je možnost napada s krajo identitete vozlišča [29]. Večina NoSQL rešitev ne zagotavlja robustnega mehanizma za obrambo v primeru takih napadov.

Težko je tudi klasificirati varnostne ranljivosti po posameznih kategorijah NoSQL podatkovnih baz, ker tudi znotraj vsake kategorije obstajajo podatkovne baze, ki različno naslavljajo problem varnosti.

Na področju podatkovnih baz NoSQL ni še sprejetih standardov za avtentikacijo, avtorizacijo in šifriranje. Trenutno najboljše prakse na tem področju vključujejo implementiranje varnostnih mehanizmov na nivoju vmesne plasti (*Middleware*) in ignoriranje varnosti na ravni gruče.

3.12 Vloga razvijalca

Relacijska podatkovna baza močno poenostavi razvoj mehanizmov za sočasni dostop. Lastnosti ACID razvijalca razbremenijo dela povezanega z zaklepanjem, zagotavljanjem konsistentnosti, reševanjem konfliktov in posodobitvami.

V primeru NoSQL podatkovnih baz mora razvijalec v sami aplikaciji implementirati morebitne potrebne zahteve ACID.

Čeprav mora razvijalec v aplikaciji ročno poskrbeti tudi za implementacijo indeksov in stičnih operacij, je prednost ta, da so te operacije lahko bolj učinkovite kot v primeru SQL pri relacijskih podatkovnih bazah, saj ima aplikacija več znanja o podatkih, ki se uporabljajo.

Zaradi slabo podprte varnosti v podatkovnih bazah NoSQL ima razvijalec tudi večjo odgovornost, da sam poskrbi za varno in robustno aplikacijo.

Poglavje 4

Klasifikacija podatkovnih baz NoSQL

V zadnjih nekaj letih so se glede na posebne zahteve po učinkovitosti in skalabilnosti ter specifičnem naboru funkcij razvile različne NoSQL podatkovne baze. Posledično obstajajo tudi različni pristopi za njihovo klasificiranje v posamezne razrede. Naslednja podpoglavja zajemajo preučitev različnih razredov nerelacijskih podatkovnih baz.

4.1 Terminologija podatkovnega modela

Za razliko od relacijskih podatkovnih baz je terminologija pri NoSQL podatkovnih bazah pogosto nekonsistentna. Težko jih je opisati, saj so se razvile ad hoc na podlagi potreb v praksi in imajo glede na različne potrebe tudi različne podatkovne modele in funkcionalnosti.

NoSQL podatkovne baze omogočajo shranjevanje skalarnih vrednosti, kot so številke in nizi, ter shranjevanje velikih binarnih objektov (*Binary large objects - BLOBs*). Nekatere od njih lahko shranjujejo tudi bolj kompleksne vrednosti. Vse NoSQL podatkovne baze shranjujejo podatke kot nabore parov tipa ključ/vrednost, vendar uporabljajo različne podatkovne strukture, in sicer [9]:

- **Objekt:** je analogen objektu v programskih jezikih, ampak brez postopkovnih metod. Vrednosti so lahko reference na objekte ali ugnezdeni objekti.
- **Dokument**: omogoča shranjevanje vrednosti v obliki ugnezdenih dokumentov ali seznamov kot tudi skalarnih vrednosti. Dokument se razlikuje od n-terice v relacijski tabeli v tem, da atributi niso opredeljeni v globalni shemi, kar dovoljuje večji razpon različnih podatkovnih tipov za vrednosti.
- **Razširljiv zapis** (*Extensible record*): je hibrid med n-terico in dokumentom, kjer so v preprosti shemi sicer opredeljene družine atributov, vendar se lahko vsakemu zapisu doda nove atribute (znotraj družine atributov).

4.2 Klasifikacija

V diplomski nalogi se osredotočamo na klasifikacijo Ricka Cattela [9], ki različne NoSQL podatkovne baze glede na njihov podatkovni model klasificira v tri različne kategorije:

- **Shrambe tipa ključ/vrednost** (*Key-value stores*): shranjujejo vrednosti in pripadajoče ključe, na podlagi katerih poizvedujemo po vrednostih.
- **Dokumentne shrambe** (*Document stores*): shranjujejo vrednosti v obliki dokumentov.
- **Shrambe tipa razširljiv zapis** (*Extensible record stores*): shranjujejo razširljive zapise, ki so lahko vertikalno in horizontalno particionirani po vozliščih.

Klasifikacijo na podlagi podatkovnega modela prikazuje preglednica 4.1.

Preglednica 4.1. Klasifikacija na podlagi podatkovnega modela

Kategorija	**Nekaj predstavnikov NoSQL podatkovnih baz**
Shrambe tipa ključ/vrednost	Redis Scalaris Voldemort Riak ...
Dokumentne shrambe	SimpleDB CouchDB MongoDB Terrastore ...
Shrambe tipa razširljiv zapis	Bigtable HBase Hypertable Cassandra ...

4.2.1 Podatkovne baze izven našega obsega

Formalne klasifikacije NoSQL podatkovnih baz ni. Nekateri avtorji npr. uporabljajo široko opredelitev NoSQL, ki vključuje tudi ostale podatkovne baze, ki niso relacijske, in sicer gre za:

- **Podatkovne baze tipa graf** (*Graph database systems*)**:** strukturirane so kot graf vozlišč in omogočajo učinkovito porazdeljeno shranjevanje in poizvedovanje podatkov ter reference med njimi (npr. Neo4j in OrientDB).
- **Objektne podatkovne baze** (*Object-oriented database systems*)**:** podobne so podatkovnim bazam tipa graf, le da so vozlišča objekti, ki so analogni objektom programskega jezika.
- **Porazdeljene objektne podatkovne baze** (*Distributed object-oriented stores*)**:** podobne so objektnim podatkovnim bazam, le da so grafi objektov porazdeljeni v pomnilniku večih strežnikov (npr. GemFire).

Te podatkovne baze so dobra izbira za aplikacije, kjer imamo podatke lahko shranjene v pomnilniku in ko je med njimi veliko povezav. Mnoge od njih zagotavljajo tudi horizontalno skalabilnost in porazdeljene poizvedbe, za razliko od NoSQL pa na splošno zagotavljajo ACID transakcije.

Ker naj bi se izraz »NoSQL« nanašal predvsem na nove podatkovne baze, ki so pritegnile zanimanje leta 2009, smo zgoraj omenjene vrste podatkovnih baz izpustili iz področja obravnave in se osredotočili na novejše.

4.3 Opisi kategorij podatkovnih baz NoSQL

Večina priljubljenih NoSQL podatkovnih shramb je prevzela ideje bodisi Google BigTable bodisi Amazon Dynamo. BigTable je pokazal, da je lahko podatkovna baza skalabilna do tisoč vozlišč. Dynamo pa je vpeljal idejo o eventuelni konsistentnosti kot način za doseganje visoke razpoložljivosti in skalabilnosti. NoSQL shrambe po navdihu Bigtable so shrambe tipa razširljiv zapis, medtem ko je Dynamo večinoma vplival na shrambe tipa ključ/vrednost.

4.3.1 Shrambe tipa ključ/vrednost

Shrambe tipa ključ/vrednost veljajo pravzaprav za temelj, na katerem so osnovane vse podatkovne baze NoSQL. Večina podatkovnih baz NoSQL namreč za shranjevanje podatkov uporablja mehanizem ključ/vrednost, pa čeprav morda to na prvi pogled ni razvidno.

Podatkovni model

Podatkovni model je enostaven in temelji na množici parov ključ/vrednost. Vsak ključ je unikaten, povečini gre za niz, in se preslika v pripadajočo vrednost. Pogosto je dolžina ključev, ki jih je potrebno shraniti, omejena na določeno število bajtov, medtem ko je glede vrednosti manj omejitev. Te so lahko poljubnega podatkovnega tipa, saj jih podatkovna baza namreč vedno shranjuje kot objekt BLOB.

Shranjevanje podatkov v obliki ključ/vrednost je pravzaprav mehanizem, ki ga uporabljajo predpomnilniki. Predpomnilnik je hitri pomnilnik, ki vsebuje največkrat uporabljene podatke aplikacije, z namenom da razbremeni počasnejši disk. Shrambe tipa ključ/vrednost so podobne predpomnilnikom, saj omogočajo hiter dostop do podatkov, ki so običajno majhne in enostavne zbirke ključev in vrednosti.

Shrambe tipa ključ/vrednost ne podpirajo sekundarnih indeksov.

Slika 4.1 [7] prikazuje primer podatkovnega modela za shrambo tipa ključ/vrednost. Za lažjo ponazoritev se uporablja nomenklatura tabel in vrstic, dejansko pa gre preprosto za zbirko parov tipa ključ/vrednost.

Slika 4.1. Primer podatkovnega modela za shrambo tipa ključ/vrednost

Shramba se tako ne zaveda notranje strukture vrednosti, ki jih vsebuje [33]. Vrednost je lahko objekt, ki je lahko strukturiran, vendar ne kot množica parov tipa ključ/vrednost. Interpretacija vrednosti in upravljanje njene vsebine sta prepuščeni odjemalni aplikaciji. Vrednost lahko zapišemo le kot celoto. Če torej odjemalec shranjuje dokument JSON in želi posodobiti samo eno njegovo polje, mora pridobiti celoten dokument, posodobiti polje v njem in celoten dokument shraniti nazaj.

Tipični predstavniki

Čeprav obstajajo že dolgo časa (npr. Berkeley DB), je v zadnjih nekaj letih na pojavitev velikega števila NoSQL shramb tega tipa močno vplival Amazon Dynamo. Gre za podatkovno bazo, ki jo Amazon uporablja za svoje storitve, in do nedavnega ni bila na voljo za uporabo širši javnosti. Sedaj je na voljo kot storitev podatkovne baze v oblaku in se imenuje DynamoDB.

Poleg Dynama poznamo še shrambe kot so Project Voldemort, Riak, Redis, Scalaris, Tokyo Cabinet, Membase in MemcacheDB.
Voldemort (LinkedIn), Riak (Mozilla, GitHub) in Kai so odprtokodni kloni Dynama.

Membase, Tokyo Cabinet in Redis so tri hitre implementacije shramb tipa ključ/vrednost. Redis je poseben primer, ker ključ lahko nastopa v obliki različnih podatkovnih struktur kot so nizi, seznami in množice. Prav tako omogoča zelo bogat nabor operacij za dostopanje do podatkov teh različnih tipov podatkovnih struktur.

Membase in MemcacheDB temeljita na odprtokodnem in visoko zmogljivem predpomnilniškem sistemu Memcached, ki se uporablja v spletnih aplikacijah. Dodatno omogočata tudi obstojno shranjevanje podatkov na disk, replikacije in druge funkcije. Lahko jih jemljemo kot hiter in učinkovit predpomnilnik za shrambe tipa razširljiv zapis [34].

Model poizvedb

Za operacije nad podatki imajo shrambe tipa ključ/vrednost na voljo preprost API. Ta omogoča uporabo dveh temeljnih operacij:

- *put(key,value)*, ki shrani vrednost skupaj s pripadajočim ključem
- *value=get(key)*, ki omogoča dostop do vrednosti na podlagi podanega ključa

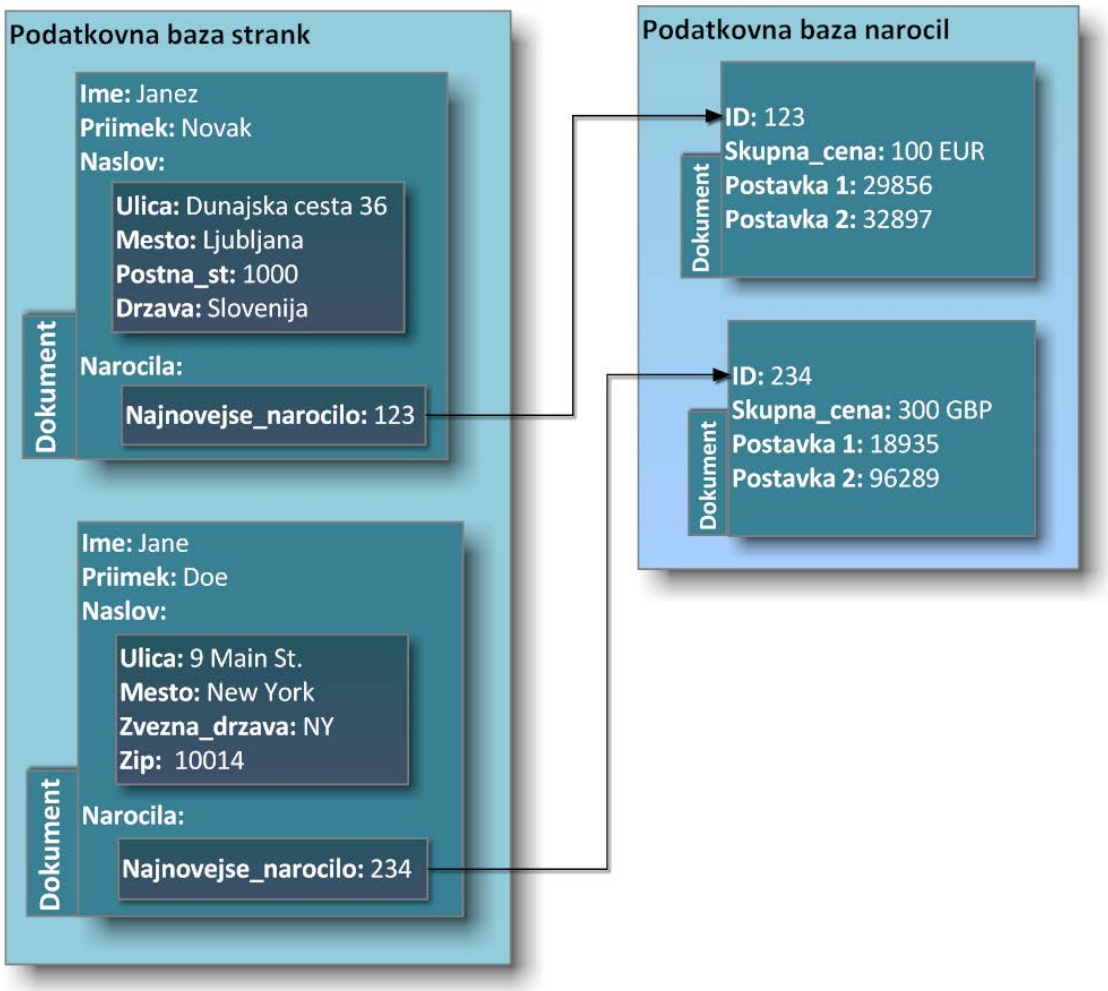
Shrambe tipa ključ/vrednost so uporabne predvsem za shranjevanje preprostih podatkov, kot so seje uporabnikov, enostavni podatki o profilih, predhodno izračunane vrednosti in rezultati. Čeprav je možno shraniti tudi bolj kompleksne podatke, to otežuje delo programerjev, ki morajo v aplikaciji poskrbeti za ročno definiranje vseh indeksov, potrebnih za bolj kompleksne poizvedbe.

4.3.2 Dokumentne shrambe

Dokumentne podatkovne baze so v osnovi shrambe tipa ključ/vrednost, ki za shranjevanje podatkov uporabljajo bolj kompleksne podatkovne strukture, to je dokumente.

Podatkovni model

Beseda *dokument* v primeru dokumentne podatkovne baze predstavlja nabor parov tipa ključ/vrednost, strukturiranih v obliko dokumenta. Povod za tako strukturo so podatki, ki danes ne nastopajo več v tako enostavnih oblikah kot so preproste vrstice ali stolpci. Pogosto so namreč zastopani v obliki XML ali JSON datotek, saj so te tehnologije visoko prenosljive, kompaktne in standardizirane. Namesto da se ti XML in JSON dokumenti preslikajo v relacijsko obliko, je veliko bolj smiselno uporabiti dokumentno shrambo. Te podatkovne baze so brez sheme, saj ni vnaprej določenega formata za XML/JSON dokument, tako da je vsak dokument neodvisen od drugih. Podatkovni model prikazuje slika 4.2 [7].



Slika 4.2. Primer podatkovnega modela za dokumentno shrambo

Dokumentne podatkovne baze niso sistemi za upravljanje dokumentov (*Document management systems*). Čeprav lahko shranijo »dokumente« v tradicionalnem smislu (članki, datoteke Microsoft Word, PDF, itd.), je lahko dokument v teh sistemih katerekoli vrste objekt [34].

Dokumentne podatkovne baze obravnavajo dokument kot celoto in se izogibajo deljenju dokumenta v njegove sestavne dele, to je v pare tipa ključ/vrednost. To omogoča povezovanje več različnih dokumentov v eno samo zbirko. Omogočajo indeksiranje dokumentov na osnovi njihovega primarnega identifikatorja, pa tudi na osnovi njihovih lastnostih [34]. Nekatere dokumentne shrambe podpirajo tudi verzije dokumentov (stare verzije se ohranijo, vse verzije pa so oštevilčene).

Dokumentne shrambe se zavedajo notranje strukture dokumenta, kar omogoča neposredno podporo sekundarnim indeksom in s tem učinkovitejše poizvedbe na kateremkoli polju.

Za razliko od shramb tipa ključ-vrednost ti sistemi tako v splošnem podpirajo sekundarne indekse in več vrst dokumentov na podatkovno bazo, ter ugnezdene dokumente. Vrednosti so lahko reference na dokumente ali pa dokumenti sami.

Tipični predstavniki

Danes je na voljo nekaj različnih odprtokodnih dokumentnih podatkovnih baz, npr. SimpleDB, Terrastore, najbolj obetavni med njimi pa sta MongoDB, ki ga uporabljajo podjetja kot so Foursquare, bit.ly, SourceForge, Disney itd., in CouchDB, ki ga uporabljata Apple, BBC, itd. Kot zanimivost naj še omenimo, da za raziskave na velikem hadronskem trkalniku v CERN-u uporabljajo CouchDB [14].

Podjetji CouchDB in Membase sta se združili v Couchbase. S tem poskušata zagotoviti prednosti obeh rešitev, npr. bogatejši CouchDB podatkovni model in hitrost ter elastično skalabilnost Membase.

Model poizvedb

Modeli poizvedb se v posameznih implementacijah dokumentnih shramb med seboj razlikujejo. Za poizvedovanje se lahko uporabljajo vmesniki API ali pa lastni poizvedovalni jeziki podobni SQL (MongoDB). Nekatere rešitve omogočajo tudi uporabo ogrodja MapReduce (CouchDB). Za razliko od shramb tipa ključ/vrednost, dokumentne shrambe omogočajo poizvedovanje na podlagi ključa kot tudi na podlagi vsebine dokumenta.

4.3.3 Shrambe tipa razširljiv zapis

Shrambam tipa razširljiv zapis (*Extensible record stores*) rečemo tudi shrambe s širokimi stolpci (*Wide column datastores*), vendar večina avtorjev uporablja prvo poimenovanje. Uvrščamo jih v skupino stolpično usmerjenih shramb (*Column oriented datastores*). Sem

spadajo tudi stolpično usmerjene podatkovne baze (*Column oriented databases*), ki pa ne spadajo v skupino NoSQL. Gre namreč za relacijske podatkovne baze, kjer se operacije izvajajo nad stolpci in ne nad vrsticami [18].

Podatkovni model

Shrambe tipa razširljiv zapis so osnovane na podlagi BigTable, Googlovega porazdeljenega sistema za shranjevanje. Osnovni koncept njihovega podatkovnega modela je delitev tako vrstic kot stolpcev preko več vozlišč (vertikalno in horizontalno particioniranje):

- Vrstice so razdeljene po vozliščih na podlagi delitve primarnega ključa (ali t.i. ključ vrstice). Namesto zgoščevalne funkcije se delitev izvede glede na določeno zalogo vrednosti ključev. Posledično to pomeni, da v primeru poizvedovanja po obsegu vrednosti ni potrebno obiskati vsakega vozlišča.
- Stolpci tabele so porazdeljeni čez več vozlišč na podlagi koncepta družin stolpcev (*Column families*). Družine stolpcev so način, s katerim lahko definiramo, katere stolpce in s tem podatke je najbolje hraniti skupaj.

Slika 4.3 [7] prikazuje primer podatkovnega modela za shrambo tipa razširljiv zapis, ki vsebuje tabele (navedeno s črko "T"), družine stolpcev ("DS") in stolpce ("S"). Tabele vsebujejo ključ in stolpce, ki so sestavljeni iz parov tipa ime/vrednost.

Slika 4.3. Primer podatkovnega modela za shrambo tipa razširljiv zapis

Horizontalno kot tudi vertikalno particioniranje se lahko uporablja istočasno na isti tabeli. Na primer, če je tabela strank razdeljena v tri družine stolpcev (osebni podatki, finančne informacije, informacije za prijavo), potem je vsaka od teh treh družin stolpcev obravnavana kot ločena tabela. Vrstice prvotne tabele so po teh ločenih tabelah porazdeljene po ID-ju stranke.

Družine stolpcev morajo pri biti pri shrambah tipa razširljiv zapis vnaprej določene. Vendar pa to ne predstavlja omejitve, saj so lahko novi atributi definirani kadarkoli.

Vrstice so podobne dokumentom: lahko imajo spremenljivo število atributov, imena atributov pa morajo biti enolična. Neobstoječe vrednosti (null) niso shranjene.

Shrambe tipa razširljiv zapis ne podpirajo sekundarnih indeksov, z izjemo redkih izjem kot je Cassandra in Hypertable od lanskega leta [34].

Tipični predstavniki
Poleg Google BigTable, ki predstavlja temelj večini shramb tipa razširljiv zapis, sem spadajo še Cassandra, HBase in Hypertable.

Cassandra je ena izmed bolj popularnih NoSQL podatkovnih baz, saj jo uporablja več znanih podjetij [19]:

- Twitter, za analitiko,
- Facebook, za iskanje po osebnih sporočilih
- Cloudkick, za spremljanje statistike in analitike
- Reddit, za implementacijo obstojnega predpomnilnika.

Model poizvedb
Vsaka implementacija shrambe tipa razširljiv zapis ima nek svoj vmesnik API, ki omogoča različne operacije nad podatki. Google BigTable API omogoča bralne, pisalne in administrativne operacije, Cassandra ima na voljo samo tri operacije: get za dostopanje, insert za pisanje in delete za brisanje. Hypertable ima na voljo svoj poizvedovalni jezik HQL (*Hypertable Query Language*) kot tudi svoj API. Tudi Cassandra ima od lanskega leta svojo alternativo SQL, to je poizvedovalni jezik CQL (*Cassandra Query Language*) [30].

Na splošno imajo shrambe tipa razširljiv zapis precej omejene zmogljivosti poizvedb. Programer mora v aplikaciji sam poskrbeti za vzdrževanje indeksov, ki omogočajo bolj kompleksne poizvedbe.

4.4 Primerjava razredov NoSQL

V spodnji preglednici 4.2 [18] je prikazana primerjava med posameznimi razredi NoSQL in relacijsko podatkovno bazo. Kot je razvidno, so vsem razredom NoSQL skupne naslednje lastnosti:

- visoka zmogljivost
- visoka skalabilnost
- visoka razpoložljivost
- majhna kompleksnost
- programerju prijazne

Preglednica 4.2. Primerjava osnovnih značilnosti posameznih razredov podatkovnih baz NoSQL in relacijskih podatkovnih baz

Podatkovna baza	Zmogljivost	Skalabilnost	Razpoložljivost	Prilagodljivost podatkovnega modela	Kompleksnost PB
Ključ/vrednost	visoka	visoka	visoka	visoka	nizka
Razširljiv zapis	visoka	visoka	visoka	zmerna	nizka
Dokumentne	visoka	spremenljiva (visoka)	visoka	visoka	nizka
Relacijske	spremenljiva	spremenljiva	spremenljiva	nizka	zmerna

Med posameznimi razredi pa obstajajo tudi razlike, ki so v večji meri povezane z različnimi podatkovnimi modeli in njihovimi lastnostmi:

- Shrambe tipa razširljiv zapis imajo nekoliko manj prilagodljiv podatkovni model zaradi vnaprejšnje določitve družin stolpcev.
- Dokumentne shrambe imajo bolj kompleksen podatkovni model, zato je njihova skalabilnost v primerjavi z ostalima dvema razredoma lahko manjša.
- Sekundarne indekse podpirajo samo dokumentne shrambe in nekaj izjem iz shramb tipa razširljiv zapis.

Velja omeniti še dejstvo, da vseh podatkovnih baz NoSQL ni možno uvrstiti samo v eno izmed treh kategorij. Riak, na primer, ni stroga shramba tipa ključ/vrednost, ker ima tudi nekatere lastnosti dokumentnih shramb kot je na primer možnost shranjevanja polstrukturiranih dokumentov. Še en primer je Cassandra, ki ima lastnosti tako shramb tipa ključ/vrednost kot tudi shramb tipa razširljiv zapis.

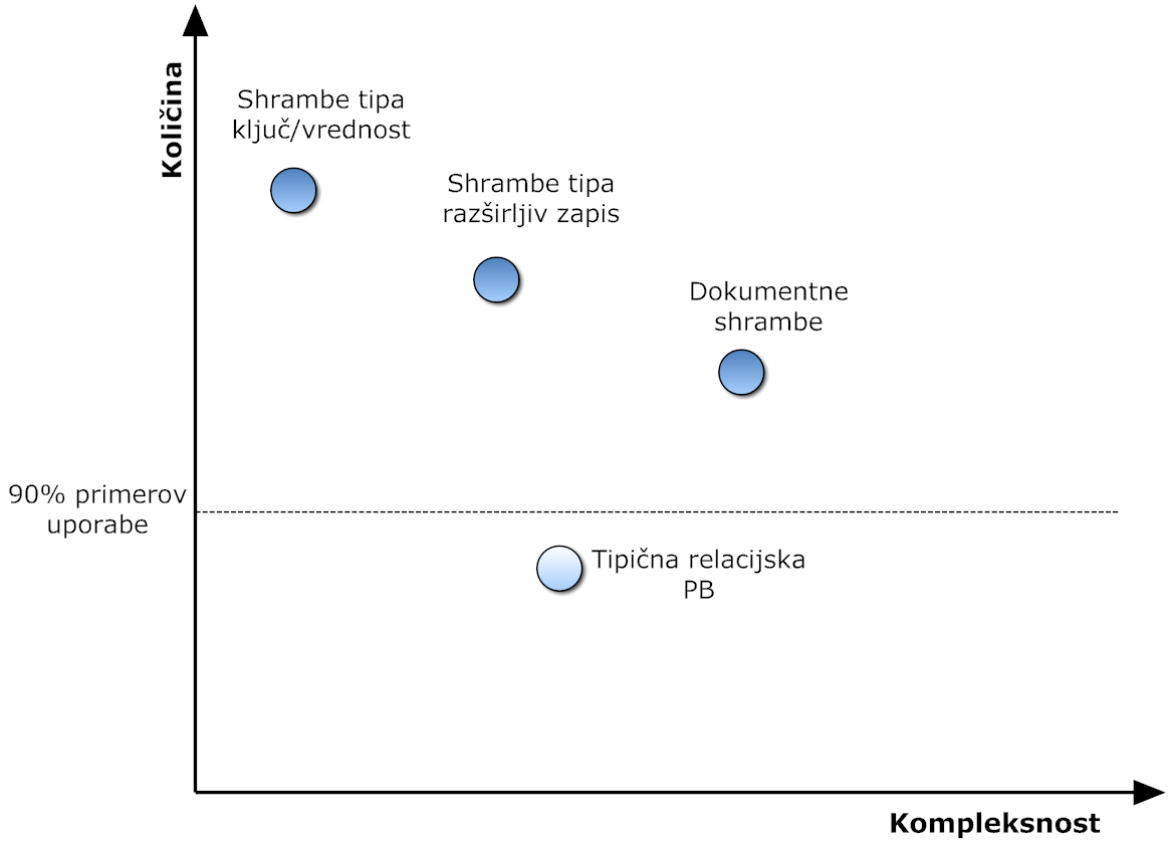
4.4.1 Obravnavanje kompleksnosti in količine podatkov

Graf na sliki 4.4 [13,22] prikazuje kako različni podatkovni modeli obravnavajo količino in kompleksnost podatkov.

Shrambe tipa ključ/vrednost in **shrambe tipa razširljiv zapis** se v primeru velikih količin podatkov zelo dobro obnesejo. To je zato, ker imajo preproste podatkovne modele, ki se jih da enostavno particionirati po večih vozliščih, kar omogoča visoko horizontalno skalabilnost. Slaba stran pa je, da preprostost njihovih podatkovnih modelov pomeni, da je potrebno za obravnavanje podatkov, ki nimajo tako preproste strukture, v aplikaciji implementirati bolj kompleksne funkcije (npr. stične operacije).

Dokumentne shrambe pa imajo na drugi strani bogatejši podatkovni model, ki omogoča shranjevanje tudi bolj kompleksnih hierarhij podatkov (vgnezdeni dokumenti). Vendar posledično to vodi k večji povezanosti podatkov znotraj posameznega dokumenta, kar predstavlja večji izziv za skalabilnost.

Relacijske podatkovne baze se na grafu nahajajo v območju, ki predstavlja 90% vseh primerov uporabe. So namreč množično razširjene in večina spletnih podjetjih jih zaenkrat še vedno uporablja, saj ne upravljajo s tolikšnimi količinami podatkov, kot jih 10% najuspešnejših podjetjih (npr. Facebook, Google, Twitter, itd.).



Slika 4.4. Graf kompleksnosti v odvisnosti od količine podatkov za različne podatkovne modele

Poglavje 5

Opredelitev kriterijev za pomoč pri izbiri najprimernejše podatkovne baze

5.1 Opredelitev problema

Danes je na voljo ogromno novih, izpopolnjenih podatkovnih baz, tako da je precej težko dobiti hiter pregled nad njihovimi značilnostmi in medsebojnimi razlikami. Še težje je izbrati pravo podatkovno bazo za shranjevanje in uporabo podatkov na način, ki bo ustrezal našim poslovnim zahtevam. Prihod podatkovnih baz NoSQL tako še dodatno otežuje izbiro in hkrati prinaša odločitev, ali uporabiti relacijsko ali NoSQL podatkovno bazo.

Organizacija, ki je soočena s takšno odločitvijo, ima težko nalogo, saj ponudniki rešitev NoSQL ponavadi dajejo precej splošne argumente za uporabo njihovih rešitev, kot npr. da so dobre za velike količine podatkov ali za hitre dostope tipa ključ/vrednost.

Pomembno je, da za izbiro najboljše podatkovne baze izhajamo predvsem iz naših zahtev, saj obstajajo velike razlike tako med relacijskimi in NoSQL podatkovnimi bazami, kot tudi med posameznimi rešitvami znotraj NoSQL podatkovnih baz. Razvijalci in arhitekti potrebujejo določene kriterije, ki jim bodo v prvi fazi odločanja pomagali organizirati in razlikovati te podatkovne shrambe glede na njihove lastnosti, prednosti in pomanjkljivosti. Na ta način bodo nato lahko ugotovili, kam usmeriti svojo bolj poglobljeno raziskavo za sprejem odločitve glede izbire prave podatkovne baze, ki bo izpolnjevala potrebe poslovanja organizacije.

Kriterije za pomoč pri izbiri najprimernejše podatkovne baze, ki jih predlagamo v diplomskem delu, so na prvem nivoju namenjeni odločanju med relacijsko in NoSQL podatkovno bazo, na drugem pa gre za odločanje med tremi različnimi razredi znotraj

NoSQL. Na tem mestu naj poudarimo, da kriteriji niso namenjeni kot pomoč pri izbiri med posameznimi NoSQL podatkovnimi bazami, saj so si te medseboj še vedno precej različne in dobre za specifičen problem, za razliko od relacijskih podatkovnih baz, ki naj bi ustrezale večini primerov uporabe.

5.2 Kriteriji odločanja med relacijsko in NoSQL podatkovno bazo

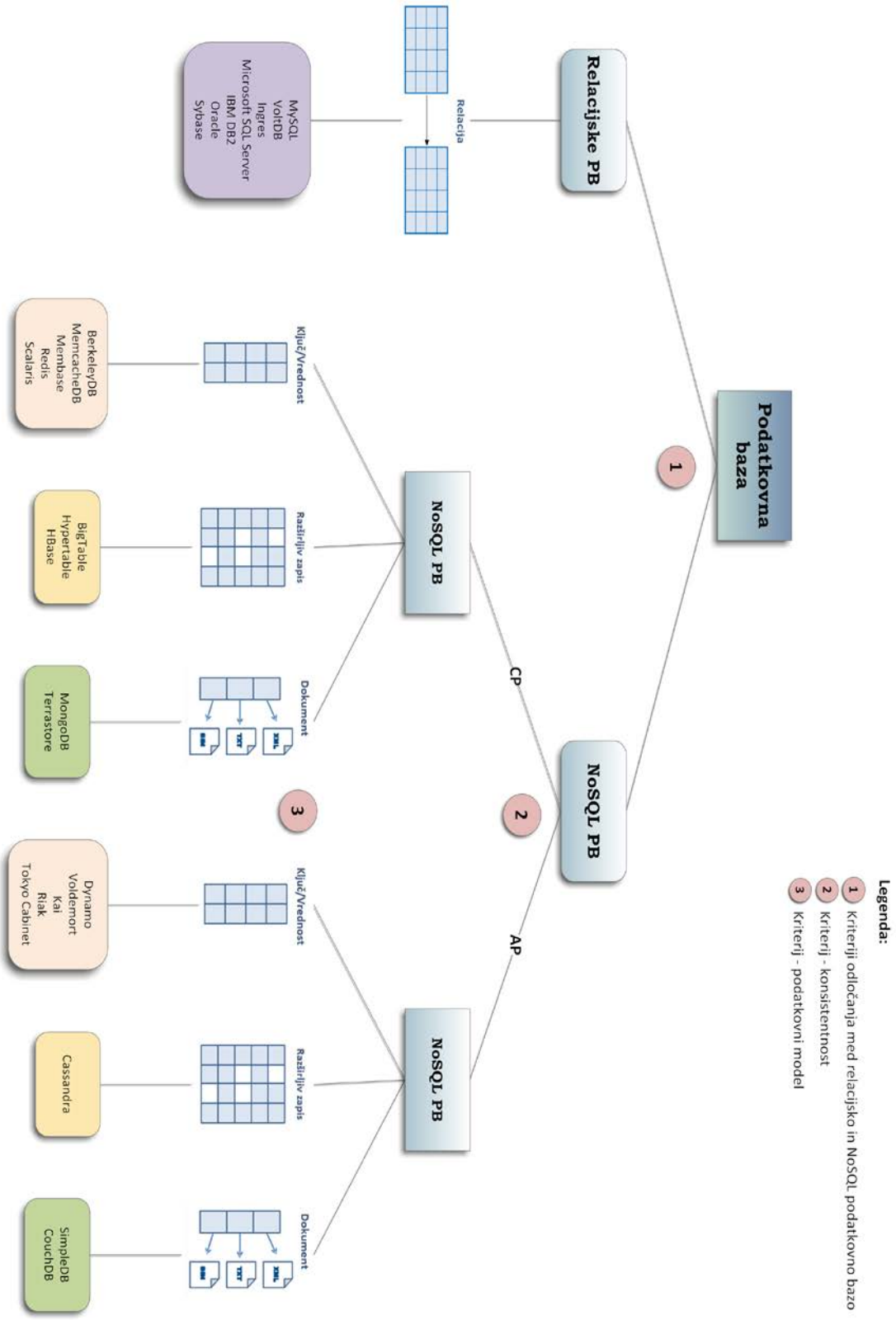
Za opredelitev možnih kriterijev za potrebe odločanja med relacijsko in NoSQL podatkovno bazo smo izhajali iz primerjave njunih značilnosti, ki smo jo podali v tretjem poglavju. Kriteriji so prikazani v spodnji preglednici 5.1, njihovi opisi pa so obrazloženi v nadaljevanju.

Na tem mestu omenimo še, da je vrstni red pomembnosti kriterijev odvisen od posameznega primera uporabe.

Na sliki 5.1 je prikazan potek odločanja na podlagi opredeljenih kriterijev po posameznih nivojih. Kot primere NoSQL podatkovnih baz smo navedli samo najvidnejše in največkrat omenjene produkte. Kriteriji odločanja med relacijsko in NoSQL podatkovno bazo so na sliki 5.1 ponazorjeni s številko 1.

Preglednica 5.1. Kriteriji odločanja med relacijsko in NoSQL podatkovno bazo

Kriteriji		Relacijska PB	NoSQL
Podatkovna shema		Definirana, statična	Ni vnaprej določena; prilagodljiva
Struktura in vrednost podatkov	**Vrsta podatkov**	Občutljivi podatki	Nekrtični, začasni podatki
	Povezave med podatki	Da, imamo kompleksna razmerja, pomembna je celovitost podatkov	Ne
	Lastnosti ACID	Da, potrebujemo. Ne moremo si privoščiti izgube podatkov.	Ne potrebujemo, šibka konsistentnost
	Strukturiranost podatkov	Strukturirani (tabele)	Polstrukturirani, nestrukturirani
Dostop do podatkov	**Poizvedbe**	Zapletene, dinamične, ad hoc	Enostavne, vnaprej definirane; nimamo stikov med podatki
	Sekundarni indeksi	Da	Omogoča jih peščica produktov
Količina podatkov	**Faktor velikosti podatkov**	Terabajti	Petabajti, neprestano narašča
	Število uporabnikov	Nekaj tisoč	Več milijonov
	Hitrost obdelave podatkov	Hitra	Visoka
Lastnosti bralnih in pisalnih operacij		Veliko posodobitev in novih pisanj	Visoka zmogljivost pisanj in nizka latenca branj
Horizontalna skalabilnost in stroški doseganja le-te		Možna, vendar draga	Enostavna, cenejša
Podpora		Zagotovljena, potreben administrator	Vprašljiva; administracija ni potrebna
Varnost		Dobra	Slaba



Slika 5.1. Kriteriji odločanja med relacijskimi in NoSQL podatkovnimi bazami ter posameznimi razredi NoSQL podatkovnih baz

5.2.1 Opis kriterijev

1. PODATKOVNA SHEMA

V primeru, da imamo podatkovni model že vnaprej dobro definiran in statičen, ki ima omejitve (za njih skrbi podatkovna baza in s tem preprečuje nezaželene anomalije) ter povezave, ki ohranjajo podatkovno celovitost, ne želimo podvojenih podatkov in je neodvisnost podatkov od aplikacije za nas pomembna, se odločimo za relacijsko podatkovno bazo.

Če želimo dinamično, nedefinirano podatkovno shemo, ki bo preprosta in prilagodljiva**,** brez večjih omejitev, in ki bo omogočala shranjevanje kakršnekoli strukture v podatkovni element, potem izberemo podatkovno bazo NoSQL.

2. STRUKTURA IN VREDNOST PODATKOV

Vrsta podatkov, povezave med podatki in lastnosti ACID
Poslovno kritični podatki potrebujejo strukturo zaradi konsistentnosti, neredundančnosti in kompleksnih razmerij med njimi. Povezani so z lastnostmi ACID in posledično z močno konsistentnostjo. Če naš sistem obravnava finančne podatke, zdravstvene zapise ali druge občutljive informacije, kjer si ne moremo privoščiti izgube podatkov, potrebujemo robusten transakcijski sistem. In če imamo veliko sočasnih uporabnikov, ki bi lahko poskušali posodobiti isti podatek ob istem času, kot je na primer vnos v tabelo delnic v sistemu e-poslovanja, potem sta izolacija in zaklepanje nujno potrebna. Naštete lastnosti zagotavlja relacijska podatkovna baza.

Kadar torej potrebujemo močno konsistentnost, izberemo relacijsko podatkovno bazo, saj je z NoSQL to veliko težje zagotoviti. Po vsej verjetnosti tudi Google uporablja relacijsko podatkovno bazo za svoje finančne podatke in podatke o zaposlencih.

Primer: Registracija vozil
Dober primer za relacijske podatkovne baze je na primer kompleksnejša aplikacija za registracijo vozil, ki lahko interaktivno iščejo po barvi vozila, znamki, modelu, letu, delnih številkah registrskih tablic in lastnostih lastnika kot so osebni podatki, bivališče, itd. ACID transakcije se lahko prav tako izkažejo kot koristne za podatkovno bazo, ki se posodablja iz večih lokacij.

V ostalih primerih, ko izgubljeni in nekonsistentni podatki niso tako kritični oziroma lahko dopuščamo nekaj podatkovne nekonsistentnosti, lahko izberemo podatkovno bazo NoSQL. Gre predvsem za spletne strani, kjer ne potrebujemo transakcij z lastnostmi ACID in konsistentnosti, ampak večinoma samo predstavljamo neke podatke. To velja za večino družabnih strani, kjer je celovitost podatkov v veliki meri opcijska in so stroški za zagotavljanje le-te nepotrebni. Dejstvo je, da ne potrebujemo ACID za pisanje statusov na

Facebooku ali tvitov na Twitterju. Ker NoSQL ni ACID skladna, lahko zagotavlja različne rezultate različnim uporabnikom ob istem času. Posledično ni široko uporabna v poslovnih aplikacijah. Če uporabnik izgubi nastavitve prilagojene spletne strani, ni tako velik problem, v nasprotju s tem, če izgubimo občutljive poslovne transakcijske podatke.

Naslednji primer je blog, ki prejema objave komentarjev. Ni potrebno, da takoj objavimo komentar, in ker je komentar v glavnem besedilo, resnično ni potrebe po relacijski podatkovni bazi za shranjevanje komentarjev. Namesto tega lahko vse komentarje zadnje ure ali dneva obdelamo naenkrat v paketu, zato da bi bile naše operacije tudi bolj poenostavljene.

Tovrstni našteti podatki so običajno zapisani samo enkrat in večkrat prebrani, zato nimajo ACID zahtev.

NoSQL izberemo tudi v primeru, če naš poslovni model ustvarja npr. veliko začasnih podatkov, ki v resnici ne sodijo v glavno podatkovno shrambo. Primer so nakupovalni vozički, zgodovina iskanja, prilagojene strani, dnevniške datoteke, seje uporabnikov.
Sem spadajo tudi sistemi za upravljanje dokumentov, katalogov in vsebin. Ti so poenostavljeni s sposobnostjo shranjevanja zapletenih dokumentov kot celote, ne pa organiziranih v relacijskih tabelah.

Strukturiranost podatkov
Kadar shranjujemo polstrukturirane (npr. XML, e-pošta, EDI, itd.) ali nestrukturirane podatke (npr. golo besedilo, dejanski dokumenti, slike, avdio, video, itd.), velike BLOB-e podatkov, ki nimajo strukture ali želimo imeti podatkovne strukture, ki so bližje tistim v programskih jezikih, se odločimo za NoSQL podatkovne baze. Tovrstne podatke največkrat srečamo pri spletnih aplikacijah kot so trgovine, novice, družabna omrežja, blogi ipd., saj v splošnem niti ne potrebujejo strogo strukturiranih podatkov.

Na drugi strani imamo lahko podatke strukturirane v tabele. Takšno podatkovno strukturo potrebujemo, kadar želimo imeti informacije organizirane, kadar želimo ohranjati celovitost in preprečiti neredundančnost podatkov ter imamo med njimi kompleksna razmerja. V takih primerih izberemo relacijsko podatkovno bazo. Struktura tabel je dobra rešitev za dolgoročno dokumentacijo podatkov, za arhiv podatkov in podatke z visokimi zahtevami glede konsistentnosti.

Primer: Craigslist *(spletni portal za objavo različnih kategorij informacij)*
Craigslist je pred kratkim preselil več kot 2 milijardi arhiviranih objav iz gruč MySQL v nabor repliciranih strežnikov MongoDB. Še vedno uporabljajo MySQL za vse aktivne objave na svoji strani, vendar pa za arhivirane objave uporabljajo MongoDB. Za Craigslist sta bili skalabilnost in zanesljivost osrednji zahtevi, ampak ena od zanimivejših funkcij, ki so jo pridobili s prehodom na MongoDB, je bila prilagodljivost sheme. Njihova MySQL podatkovna baza je bila tako velika, da bi katerakoli sprememba sheme (npr. sprememba tabele s stavkom alter table) potrebovala približno dva meseca preden bi se lahko uveljavila [1]. V primeru MongoDB je to veliko lažje in hitreje, saj so podatki shranjeni kot dokumenti

JSON brez zahtevane sheme. Z ločitvijo arhiviranih objav od trenutnih, so na Craigslist poenostavili svojo arhitekturo in omogočili lažji in hitrejši odziv na spremembe.

3. DOSTOP DO PODATKOV

Poizvedbe

V primeru, da so naše poizvedbe zapletene, dinamične, ad hoc in temeljijo na več iskalnih pogojih s stiki, potem izberemo relacijsko podatkovno bazo. Tovrstne poizvedbe največkrat potrebujemo v poslovnem svetu, kjer želimo hitre rezultate in v realnem času pridobiti nek podatek (npr. takojšnja analiza nekega poslovnega dogodka).
Mnogo razvijalcev že pozna SQL in mnogi zagovarjajo, da je uporaba SQL enostavnejša kot pa nižjenivojski ukazi, ki jih zagotavljajo sistemi NoSQL.

Če imamo enostavne poizvedbe, ki jih lahko vnaprej opredelimo, nimamo stikov med podatki, potem je bolje izbrati NoSQL podatkovne baze. Namreč, če že vnaprej vemo, kako bomo dostopali do podatkov ali potrebujemo spremembe aplikacije v primeru, ko želimo drugače dostopati do nje, potem prednosti relacijske podatkovne baze postanejo ovira. Še posebej to velja za splet, kjer podatki niso niti strogo strukturirani, niti niso potrebne dinamične poizvedbe, saj večina aplikacij že uporablja vnaprej pripravljene poizvedbe ali shranjene procedure. V tem primeru bodo naše poizvedbe, in ne sami podatki, narekovale tudi najboljšo organiziranost podatkov.

Pomembno je torej, da najprej definiramo na kakšen način bomo dostopali do podatkov in jih v skladu s tem shranimo, ne pa obratno.

Sekundarni indeksi

Da bi podprli bogatejša podatkovna razmerja in hitrejše, učinkovitejše iskanje po drugih pogojih ter vračanje določenih podatkov, do katerih pogosto dostopamo, bi morda potrebovali podporo sekundarnih indeksov.

Najboljša podpora sekundarnim indeksom je zagotovljena v relacijskih podatkovnih bazah. NoSQL pa v splošnem ne podpirajo sekundarnih indeksov, obstajajo pa redke izjeme kot so MongoDB, HBase in Cassandra.

4. KOLIČINA PODATKOV

Podatkovne baze NoSQL so namenjene ogromnim količinam podatkov. Zaradi samega načina shranjevanja lahko podatke shranjujejo in vračajo hitreje od relacijskih podatkovnih baz. S pomočjo MapReduce lahko mnogo hitreje (vzporedno) obdelajo ogromne količine podatkov za opravljanje analitičnih poizvedb.

Podatkovne baze NoSQL zato izberemo takrat, kadar količina podatkov, ki jih moramo shraniti, neprestano narašča ali ko imamo vse večjo potrebo po obdelavi večjih količin

podatkov v krajšem času. Na primer, število (npr. milijon) uporabnikov naše spletne aplikacije se povečuje in posledično se ustvarjajo večje količine podatkov (nekaj petabajtov) in večje število dostopov. Paketne obdelave podatkov največkrat potrebujemo pri ad hoc analizah v podatkovnih skladiščih.

Primer za količino podatkov
Twitter ima problem shranjevanja več kot 12TB podatkov na dan, ki jih ustvarijo uporabniki, z upoštevanjem, da se te zahteve podvojijo večkrat na leto [32]. To so podatki, ki so preveliki, da bi zanje skrbelo eno samo vozlišče. Pri hitrosti 80MB/s je potreben en dan za shranjevanje 7TB, tako da morajo biti pisanja porazdeljena čez gručo vozlišč, kar pomeni dostop tipa ključ/vrednost, MapReduce, replikacije, toleranco proti okvaram, probleme konsistentnosti in vse ostalo. Za hitrejša pisanja se lahko uporabi sisteme, ki shranjujejo podatke v pomnilnik. Tudi če bi Twitter uporabljal najhitrejše diske, bi potreboval več kot 40 ur za zapis teh informacij. Podjetje se je odločilo za kombinacijo uporabe podatkovne baze HBase in Hadoop HDFS (*Hadoop Distributed File System*), porazdeljenega datotečnega sistema s samodejno replikacijo in toleranco proti okvaram, ki omogoča porazdelitev pisanj vseh tvitov čez več poceni strežnikov.

Primer za hitrost obdelave podatkov
Nabori podatkov so lahko v praksi res ogromni. Pri ponudniku kreditnih kartic Visa so za obdelavo dvoletnih transakcij kreditnih kartic (približno 70 milijard transakcij) uporabili ogrodje Hadoop MapReduce [32]. Posledično so zmanjšali čas obdelave iz enega meseca, kolikor so porabili z uporabo relacijske podatkovne baze, na samo 13 minut.

Torej nekatere NoSQL podatkovne baze zagotavljajo bistveno večjo prepustnost podatkov kot tradicionalni SUPB-ji in hitrejše odzivne čase pri vseh obremenitvah.

Če imamo le nekaj tisoč uporabnikov in podatke velikosti nekaj terabajtov, izberemo relacijsko podatkovno bazo.

5. LASTNOSTI BRALNIH IN PISALNIH OPERACIJ

Razvijalci morajo razčleniti svoje potrebe podatkovne baze v različne komponente. Najpomembnejše je, da definiramo kakšno je razmerje med številom bralnih in pisalnih operacij glede na odzivnost podatkovne baze. Obstaja več različic NoSQL, ki so optimizirane za branje ali pisanje podatkov.

V primeru, da imamo predvsem veliko posodobitev obstoječih podatkov, in novih pisanj (vnosov) glede na zahtevano konsistetnost ter obstojnih pisanj, raje izberemo relacijske podatkovne baze. Torej imamo več pisalnih kot bralnih operacij.

NoSQL podatkovne baze so načrtovane za visoko zmogljivost pisanj in veliko branj (npr. družabna omrežja, spletne trgovine, kjer imamo veliko iskanj) ali pa samo za branja (npr.

novice, podatkovna skladišča, kjer opravljamo analize), ki potrebujejo nizko latenco. Torej posodobitev ni ali pa je teh izredno malo. Zaradi horizontalne skalabilnosti in porazdeljene shrambe lahko vzporedno pišemo v več vozlišč v gruči.

6. HORIZONTALNA SKALABILNOST IN STROŠKI DOSEGANJA LE-TE

Ključna značilnost NoSQL sistemov je horizontalna skalabilnost. Pri komercialnih relacijskih podatkovnih bazah sicer lahko dosežemo večjo skalabilnost z nakupom večjih in hitrejših strežnikov in shrambe, kar seveda prinaša večje stroške, prav tako pa ne morejo zlahka doseči samodejne delitve podatkov, zaradi tabelno osnovane narave in ACID lastnosti. To je eden od glavnih razlogov, da se odločamo za NoSQL podatkovno bazo, saj so načrtovane za lažjo in cenejšo horizontalno skalabilnost.

Če torej naša relacijska podatkovna baza ne bo skalabilna glede na naš promet pri sprejemljivih stroških, če je količina naših podatkov tako velika, da je za njihovo obravnavo potrebno particioniranje in če naš SUPB ne zmore več obravnavati velikih obremenitev, se odločimo za NoSQL.

Kadar pa lahko vse naše podatke upravlja podatkovna baza na enem vozlišču in jih ni potrebno particionirati, pa izberemo relacijsko podatkovno bazo.

7. PODPORA

Podjetja velikokrat želijo zagotovila, da bodo v primeru odpovedi ključnega sistema, dobila pravočasno in ustrezno podporo. Ponudnik mora ponujati tudi možnost vzdrževanja, nadgrajevanja podatkovne baze s funkcijami in orodji, ki bi jih morda potrebovali kasneje in ki bodo zadovoljili naše potrebe danes in jutri. Pomembna je tudi možnost izobraževanja in vrsta podpore ponudnika, to je npr. elektronska pošta, telefon ali splet.

Vsi ponudniki relacijskih podatkovnih baz zagotavljajo visoko raven podpore podjetjem (24 ur na dan), ker jih zavezuje licenčna pogodba. V primeru okvare ali napake sistema nam bodo le-to zagotovo odpravili v roku 24 ur.

Nasprotno je večina podatkovnih baz NoSQL odprtokodnih projektov, zato so zelo dobro sprejete s strani razvijalcev, ki jim ni potrebno skrbeti glede licenc in problemov oglaševalske podpore. Podjetja, ki so ustvarila te sisteme, so pogosto majhna novoustanovljena mlada podjetja (start-up) brez globalnega dosega, sredstev za podporo ali ugleda, ki jih imajo veliki ponudniki SUPB-jev kot so Oracle, Microsoft in IBM. V primeru izpada sistema se lahko zgodi, da bomo morali iskati pomoč pri dragem zunanjem vzdrževalcu.

Možne so tudi prilagoditve podatkovne baze NoSQL za naše potrebe, vendar bomo za to storitev morali plačati.

Izbrati moramo torej ponudnika z dobro dokumentirano podatkovno bazo z aktivno podporo in s stabilnim delovanjem.

NoSQL podatkovne baze zaenkrat še vedno potrebujejo visoko raven znanja za namestitev. Na tem mestu se je potrebno vprašati, koliko časa bo torej potrebnega za uvedbo in integracijo podatkovne baze NoSQL.

Naslednje ključno vprašanje je, koliko administracije bo nova podatkovna baza potrebovala. V poslovnem svetu ima večina podjetij pri uporabi SUPB-jev (npr. za sistemske nastavitve) strokovnjake in administratorje podatkovnih baz. Pri NoSQL sta administracija in nastavljanje zmogljivosti drugačni. Aplikacije so tiste, ki upravljajo z nastavitvami in zmogljivostjo, ne pa optimizacije podatkovnih sistemskih virov, za katere skrbijo za to izučeni administratorji.

Večina razvijalcev je v tem trenutku seznanjena le s koncepti NoSQL. To pomeni, da je skoraj vsak razvijalec NoSQL še v stanju učenja. To se bo sčasoma seveda spremenilo, vendar je v tem trenutku lažje najti izkušenega programerja ali administratorja za relacijsko podatkovno bazo kot pa strokovnjaka za NoSQL. Vendar pa so lahko NoSQL podatkovne baze lažje razumljive novim razvijalcem, ki niso obremenjeni z logiko relacijskih podatkovnih baz.

8. VARNOST

Relacijske podatkovne baze zagotavljajo veliko mehanizmov za dobro podporo varnosti dostopa do podatkov in sledenju informacij kdaj in do katerih podatkov je nekdo dostopal. (V Sloveniji je to pomembno zaradi zakona o varstvu osebnih podatkov - ZVOP). Podpora podatkovnih baz NoSQL na tem področju je zelo slaba. Veliko sistemov je starih komaj nekaj let in so zaenkrat še nezanesljivi.

Predno preklopimo na NoSQL podatkovno bazo moramo oceniti naše tveganje. Razmisliti moramo o resnih posledicah glede varnosti in količini časa ter ljudi, ki sta potrebni za upravljanje in razumevanje varnostnih vidikov novega načina varovanja podatkov (npr. omejen dostop, varnostne kopije, ognjevarne omare, itd.).

5.2.2 Splošni primeri uporabe

Če povzamemo, so najprimernejše situacije za uporabo NoSQL podatkovnih baz sledeče:

- preprost podatkovni model
- prilagodljivost je pomembnejša od strogega nadzora nad opredeljeno podatkovno strukturo
- zahtevana visoka zmogljivost

- stroga podatkovna konsistentnost ni nujna
- računalništvo v oblaku

Ne gre za vprašanje, ali so podatkovne baze NoSQL boljše od relacijskih, ampak za vprašanje, kdaj je dobro uporabiti NoSQL. Odgovor je, kadar imamo zahtevo za poizvedbe po obsežnih podatkovnih naborih z veliko hitrostjo izvajanja poizvedb. Tu nastopi NoSQL. Gre za podatkovne nabore, kamor spadajo npr. spletni dnevniki, transakcije nakupov, proizvodni podatki iz naprav na tekočem traku, znanstvene zbirke podatkov za izvajanje različnih analiz itd., ki se kopičijo v velikem številu vsako sekundo in za njihovo shranjevanje in obdelavo SUPB ni dovolj zmogljiva in hitra rešitev.

Na drugi strani, če potrebujemo konsistentne podatke, obstojna pisanja, ACID transakcije in imamo zapletene podatkovne strukture in razmerja, je tradicionalna relacijska podatkovna baza še vedno najboljša rešitev. Področja, ki so primerna za relacijske podatkovne baze lahko vključujejo katerekoli zapletene poslovne aplikacije, zlasti finančno usmerjene aplikacije, kjer so celovitost transakcij, konsistentnost in trajnost nujne zahteve. Primeri so tudi OLTP obdelave, kjer še vedno zmaguje kombinacija kakovosti podatkov in zmogljivosti dobro načrtovanega SUPB-ja.

5.3 Kombinacija relacijske in NoSQL podatkovne baze

Veliko poslovnih področij lahko obsega različne vrste strukturiranih in nestrukturiranih podatkov. V teh primerih je lahko dobra rešitev kombinacija uporabe relacijske in NoSQL podatkovnih baz.

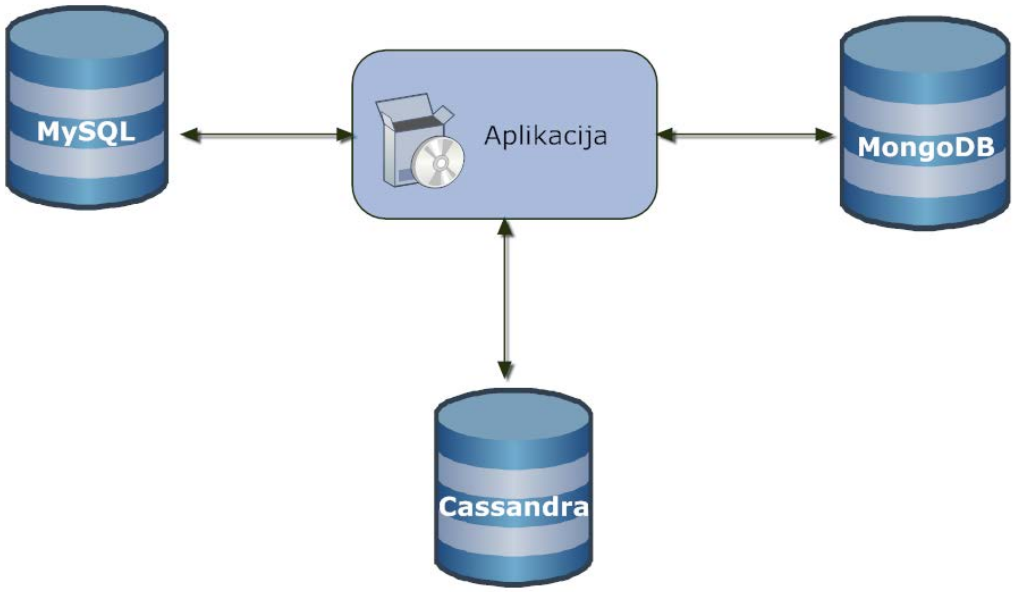
Slabost takšne rešitve je, da je potrebno namestiti dve podatkovni bazi, morebitno potrebno integracijo podatkov med njima pa moramo implementirati v sami aplikaciji (saj stiki niso možni), kar prinaša še dodatno kompleksnost.
Kombinacija obojega po drugi strani lahko omogoča lažji prehod na uporabo NoSQL. Ni potrebe, da selimo naše obstoječe podatke, ampak podatkovno bazo NoSQL uporabimo kot dopolnilno shrambo za ustrezne vrste nekritičnih podatkov kot so podatki seje ali podatki za analize.

Najboljši pristop je torej uporaba prednosti, ki jih prinašata oba svetova – relacijska in NoSQL podatkovna baza (slika 5.2).

Primer tipa aplikacije (spletna trgovina), ki uporablja več vrst podatkovnih baz:

- Relacijske podatkovne baze: MySQL za kritične podatke kot so podatki o plačilu.

- Dokumentne shrambe: MongoDB za veliko količino podatkov z nizko vrednostjo kot so števci zadetkov ali dnevniki.
- Shrambe tipa razširljiv zapis: Cassandra za podatke uporabnikov kot so uporabniški profili.



Slika 5.2. Kombinacija uporabe relacijske in NoSQL podatkovnih baz

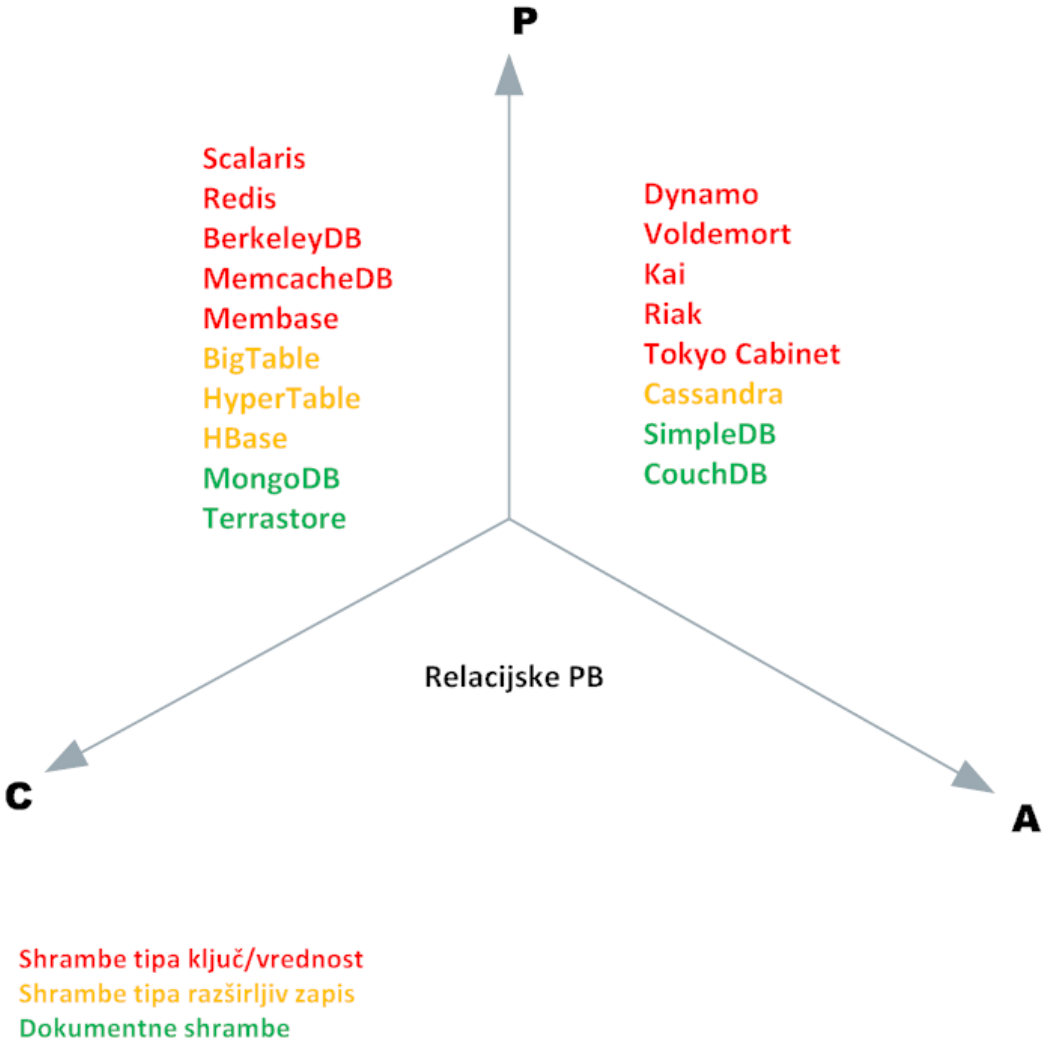
5.4 Kriteriji odločanja med razredi podatkovnih baz NoSQL

Naš prvi kriterij za odločanje med posameznimi razredi podatkovnih baz NoSQL izhaja iz CAP teorema, opisanega v drugem poglavju, ki pravi, da lahko vsaka podatkovna baza pokriva le dve od treh lastnosti CAP.

Eden od primarnih ciljev NoSQL je že večkrat omenjena horizontalna skalabilnost. Zanjo potrebujemo particijsko toleranco omrežja, ki zahteva opustitev ali konsistentnosti ali razpoložljivosti.

Z vidika poslovanja je razpoložljivost bolj dragocena kot konsistentnost. Strankam ne moremo preprosto zavrniti njihovih naročil, ker so nam odpovedali računalniki.

Slika 5.3 prikazuje splošno usmeritev nekaj različnih podatkovnih baz glede na teorem CAP. Upoštevati je potrebno, da se prikaz podatkovnih baz na sliki lahko spremeni glede na nastavitve podatkovnih baz (uporabljena je privzeta nastavitev).



Slika 5.3. Prikaz podatkovnih baz glede na teorem CAP

Iz slike je razvidno, da je zasnova sistema glede na pozicijo CAP neodvisna od tipa podatkovnega modela NoSQL.

V tej upodobitvi so relacijske podatkovne baze na liniji med konsistentnostjo in razpoložljivostjo **(CA)**, kar pomeni, da niso odporne na izpad omrežja. Razlog za to je lahko izpad glavnega strežnika, ali pa da imamo polje strežnikov, ki preprosto nima dovolj vgrajenih mehanizmov za nemoteno delovanje v primeru izpada omrežja. Tovrstni sistemi imajo torej težave s particijami omrežja, rešujejo pa jih običajno z replikacijo.
Da v prvi vrsti podpiramo konsistentnost in razpoložljivost pomeni, da po vsej verjetnosti uporabljamo dvofazno potrjevanje za porazdeljene transakcije. Ko se pojavi mrežna particija, sistem blokira. Tak sistem je omejen na en sam podatkovni center, saj lažje zagotovimo, da so vsa vozlišča povezana [19].

Googlova podatkovna baza BigTable, MongoDB, HBase, Hypertable, Redis in ostali, se vsi osredotočajo na nekoliko manjšo razpoložljivost in bolj na zagotavljanje konsistentnosti in

particijske tolerance **(CP)**. Naši podatki bodo konsistentni, v primeru izpada vozlišča pa bo sistem še vedno lahko nerazpoložljiv, vendar za zelo kratek čas [19].
Če sistem v primeru particij zagotavlja kompromis glede razpoložljivosti, torej podpira močno konsistentnost, potem zagotavlja atomarnost branj in pisanj tako, da se bodisi v celoti zaustavi, bodisi zavrne pisanja ali pa se odzove samo na tista branja in pisanja podatkov, katerih gospodar se nahaja v particiji [16].

Podatkovne baze izpeljane iz Amazon Dynamo vključujejo Cassandro, projekt Voldemort, CouchDB, Riak in ostale. Te so bolj osredotočene na razpoložljivost in particijsko toleranco **(AP)** in se raztezajo čez več podatkovnih centrov. Vendar to ne pomeni, da označujejo konsistentnost kot nepomembno. Gre za sisteme, kjer je eventuelna konsistentnost dopustna in kjer je »eventuelno« lahko stvar milisekund, prednost pa je nizka latenca dostopa in visoka prepustnost.
Sistem bo v primeru particij omrežja vedno na voljo. Odzval se bo na vse zahteve, posledično pa lahko s tem vrne zastarele, netočne podatke in sprejme konfliktna pisanja. Takšne probleme nekonsistenčnosti se pogosto rešuje z vektorskimi urami [16].

Natančneje se torej pri podatkovnih bazah NoSQL osredotočamo in izbiramo med močno (CP) in eventuelno konsistentnostjo (AP). Na sliki 5.1 je ta kriterij ponazorjen s številko 2. Torej gre za poslovno in ne tehnično odločitev, zato si moramo zastaviti sledeči vprašanji:

- Kako pomembna je za našo aplikacijo konsistentnost?
- Ali je tveganje, da je naša podatkovna baza nerazpoložljiva večje od njene nekonsistentnosti?

5.4.1 Opis kriterijev

1. KONSISTENTNOST

Kadar obstaja majhna verjetnost, da se bodo podatki sočasno posodabljali in če uporabnik lahko tolerira začasno nekonsisentnost podatkov, torej ni odvisen od sprememb, ki jih povzročijo drugi uporabniki, je eventuelna konsistentnost dobra rešitev. V nasprotnem primeru je potrebna močna konsistentnost.

Primer uporabe eventuelne konsistentnosti: Nakupovalni voziček v spletni trgovini
V primeru velike spletne trgovine nam je najbolj pomembno, da lahko naše stranke oddajo naročila, torej da je podatkovna baza visoko razpoložljiva. V nasprotnem primeru lahko izgubimo na tisoče naročil. Konsistentnost nam v tem primeru ni toliko pomembna, to je ali so prikazani podatki o zalogi posameznih izdelkov točni ali ne. Lahko da podatkovna baza vseh izdelkov, ki so na voljo, za nekaj sekund niti ne prikaže. Tudi če nek izdelek ni na zalogi, lahko naročilo izpolnimo, vendar malce kasneje. Realnost je sama po sebi že eventuelno konsistentna, saj če na primer podatkovna baza shranjuje število razpoložljivih

izdelkov v naši trgovini in se nam eden razbije, potrebujemo nekaj časa, da posodobimo ta podatek. To ne predstavlja velikega problema, saj lahko stranko obvestimo o roku dobave in izvedemo naročilo.

Primer uporabe močne konsistentnosti: Facebook

Facebook se je odločil razširiti svojo infrastrukturo pošiljanja sporočil, ki vključuje klepet, SMS, e-pošto in sporočila za pogovore v realnem času [25,26,34].

Trenutna infrastruktura za pošiljanje sporočil upravlja z več kot 350 milijoni uporabnikov, ki pošiljajo več kot 15 milijard osebnih sporočil na mesec. Njihova storitev za klepet podpira več kot 300 milijonov uporabnikov, ki pošiljajo preko 120 milijard sporočil na mesec. S spremljanjem uporabe so ugotovili dva podatkovna vzorca:

- Majhen nabor začasnih podatkov, ki niso obstojni.
- Naraščajoč nabor podatkov, do katerih se redko dostopa.

Vedeli so, da potrebujejo porazdeljeno podatkovno bazo, ki bo odporna na napake, sposobna upravljati s petabajti sporočil in v večini primerov opravljati pisalne operacije.

Za shranjevanje takšnih podatkov so se v glavnem odločali med tremi podatkovnimi bazami, to je MySQL, Cassandra in HBase. Pri testiranju vseh treh baz se je MySQL izkazal za neučinkovitega v primeru velikih podatkov. Ko so indeksi in podatkovni nabori zelo zrastli, se je zmogljivost poslabšala. Pri Cassandri se ugotovili, da je model eventuelne konsistentnosti težko uskladiti z njihovo novo infrastrukturo za pošiljanje sporočil. HBase je zelo dobro skalabilen in zmogljiv za njihovo delovno obremenitev ter ima enostavnejši model konsistentnosti kot Cassandra. Ugotovili so tudi, da nudi funkcionalnosti, ki ustrezajo njihovim zahtevam (samodejna izravnava obremenitev in preklop na drugo vozlišče v primeru odpovedi, podpora kompresiji, itd.).

Facebook je torej na koncu izbral HBase, ker so ugotovili, da gre za sistem, ki bi lahko bil najbolj primeren za obravnavo omenjenih dveh tipov vzorcev podatkov, in ker ponuja model močne konsistentnosti. Na primer v primeru pošiljanju e-pošte so želeli zagotoviti, da ko uporabnik pošlje sporočilo, dobi nemudoma obvestilo o uspešnem pošiljanju. V nasprotnem primeru bo uporabnik poskušal ponovno poslati sporočilo, naslednji dan pa bo ugotovil, da je poslal dve sporočili.

2. PODATKOVNI MODEL

Poleg konfiguracije CAP je drug pomemben kriterij za odločanje med podatkovnimi bazami NoSQL sam podatkovni model, ki ga uporabljajo. Na sliki 5.1 ta kriterij ponazarja številka 3. Omenili smo že, da lahko definiramo tri tipe podatkovnih modelov in sicer:

- ključ/vrednost,
- dokument in
- razširljiv zapis.

Sledi predstavitev njihovih tipičnih primerov uporabe za lažjo usmeritev in odločanje.

5.4.2 Splošni primeri uporabe med posameznimi razredi NoSQL podatkovnih baz

Nobena od NoSQL podatkovnih shramb ni najboljša za vse vrste uporab. Različni uporabniki dajejo glede na svoje zahteve različne prioritete funkcionalnostim podatkovnih baz NoSQL. Čeprav popoln vodič za izbiro določene podatkovne shrambe presega namen naše diplomske naloge, si bomo ogledali nekatere primere uporabe, ki ustrezajo posameznim razredom NoSQL podatkovnih shramb.

5.4.2.1 Shrambe tipa ključ/vrednost

Shrambe tipa ključ/vrednost so najbolj primerne za uporabo v tistih aplikacijah, ki uporabljajo sezname nekih vrednosti ali pa samo posamezno vrednost, in v katerih opravljamo poizvedbe na podlagi enega samega atributa. Primeri seznamov so npr. kategorije izdelkov, atributi posameznih izdelkov, vsebina nakupovalnega vozička, najbolj prodajani izdelki, itd. V primeru posameznih vrednosti pa gre lahko za barvne sheme, URL naslov glavne strani spletne aplikacije, kamor se usmeri uporabnika, ID računa uporabnika, itd. Za shrambe tipa ključ/vrednost velja, da lahko vrednosti zavzamejo različne podatkovne tipe in so lahko poljubne velikosti. To pomeni, da so primerne tudi za shranjevanje komentarjev, ocen, statusnih sporočil ali e-pošte. Ker večina takih podatkov ni v hierarhičnem razmerju, odsotnost razmerij in stikov ne predstavlja omejitve.

Shrambe tipa ključ/vrednost bi lahko uporabili tudi za poizvedbe na osnovi več atributov s kreiranjem dodatnega indeksa ključ/vrednost, ki bi ga sami vzdrževali. Vendar pa bi bilo v takem primeru bolje, da uporabimo dokumentno shrambo.
Tovrstne shrambe se lahko uporabijo tudi kot predpomnilniki shramb tipa razširljiv zapis in relacijskih podatkovnih baz.

Ostali primeri uporabe:

- štetje normaliziranih podatkov (glasovi, ocene, nekatera statistika)
- shranjevanje podatkov o nastavitvah uporabnikov
- shranjevanje začasnih podatkov (kot so spletne seje, zaklepanja ali kratkotrajna statistika)
- implementacija vgrajenega iskanja
- shranjevanje zapletenih objektov, ki so bili dragi za stike v relacijski podatkovni bazi.

5.4.2.2 Dokumentne shrambe

V splošnem se dokumentne shrambe uporabljajo v situacijah, ko želimo poizvedovati po več atributih. Nekatere rešitve omogočajo tudi verzioniranje dokumentov in ohranjajo stare verzije dokumentov.

Primeri uporabe:

- shranjevanje dokumentov PDF in Word, slik, preglednic Excel, itd.
- dnevniške datoteke
- koledarji
- pogovorne aplikacije
- imenik
- arhiviranje

5.4.2.3 Shrambe tipa razširljiv zapis

Primeri uporabe za shrambe razširljivih zapisov so podobni tistim pri dokumentnih shrambah: več vrst objektov in s poizvedbami, ki temeljijo na kateremkoli atributu.

Shrambe tipa razširljiv zapis se dobro obnesejo v primerih, ko želimo entitete pogrupirati po njihovih skupnih lastnostih, ki pa so lahko glede na posamezno entiteto sestavljene iz različnih atributov.

Primer je katalog izdelkov, kjer imamo zbirko različnih izdelkov. Vsak izdelek ima določene lastnosti, mi pa želimo v eno tabelo pogrupirati vse izdelke po lastnostih »Velikost« in »Ocena«. Nekateri izdelki imajo lahko velikost podano z merami širine, višine in globine, drugi s premerom, ostali pa s težo. Ocena je lahko podana s številko od 1 do 5 (npr. za knjige) ali pa je sestavljena iz večih podkategorij, kot so barva, okus, svežina, itd. V primeru relacijskih podatkovnih baz bi morali biti atributi, ki so specifični za nek izdelek, vsi posebej shranjeni v ločenih tabelah, ki bi vsebovale tuji ključ na tabelo lastnosti. Posledično bi bile za poizvedovanje potrebne stične operacije.

Drug primer uporabe je spletna trgovina, v kateri vzdržujemo podatke o strankah. Ker želimo npr. učinkovito iskati po strankah določene države, jih pogrupiramo po lastnosti »Država«. Za izboljšanje zmogljivosti želimo tudi ločiti podatke o strankah, ki se redko spreminjajo (npr. osebni podatki) od podatkov, ki se pogosto spreminjajo (podatki o nakupih).

5.5 Uporaba kriterijev za izbiro podatkovne baze na hipotetičnem primeru

5.5.1 Opredelitev problema

Spletna aplikacija za podporo storitvam trženja in oglaševanja

Podjetje OglašujMe razvija spletno aplikacijo, ki bo njihovim strankam nudila informacijsko podporo storitvam trženja in oglaševanja, kamor spadajo kreiranje novih naročnin na različne storitve, vodenje evidence o naročnikih in statistika o naročninah. Za slednjo iščejo rešitev za izbiro najprimernejše podatkovne baze glede na njihove potrebe.

Njihove stranke bodo večinoma pravne osebe, to je različna podjetja, zavarovalnice, banke, študentski servis, itd. Za primer banke se lahko z določeno ključno besedo preko SMS-a ali pa na spletni strani in s potrditveno e-pošto naročimo na neko storitev, na primer: obvestila o prekoračenem limitu, dvigih na bankomatu, obvestila o raznih predavanjih s področja financ, itd. Vsaka naročnina se zabeleži v podatkovno bazo, števec naročnin pa se za posamezno vrsto poveča.

Različna podjetja imajo lahko različne načine trženja in oglaševanja in za vsakega želijo zasledovati sledeče:

- število naročnin po posameznih vrstah storitev za določen dan
- število preklicanih naročnin

Spremlja se lahko tudi razloge za preklic in statistiko vseh naročnin do določenega datuma za posamezno vrsto naročnine. Vsakokrat, ko se poveča dnevna statistika, se poveča tudi sumarna statistika, ki vključuje celotno statistisko do tega dne.

Večina poizvedb so vstavljanja z minimalnim številom branj. Statistiko morajo spremljati sproti, saj lahko uporabniki isti dan potrdijo naročnino in jo tudi prekličejo.

Podjetje OglašujMe pričakuje, da se bo podatkovni model lahko veliko spreminjal. Neka stranka se lahko npr. odloči, da bo za novo trženje oziroma oglaševalsko kampanjo dodala novo vrsto naročnine in v tem primeru bo potrebno poslati sms z novo ključno besedo. Lahko pa v svojo oglaševalsko kampanjo vključi kakšno drugo sredstvo obveščanja (poleg SMS in e-pošte tudi družabna omrežja, npr. Facebook).

Stranke, ki so uporabljale njihovo spletno aplikacijo, so imele pred nekaj leti le nekaj 10.000 naročnikov. Danes to aplikacijo uporabljajo podjetja, ki imajo lahko tudi več kot 10 milijonov naročnikov. Pričakujejo, da se bo to število v prihodnosti le še povečevalo.

Ker je OglašujMe še vedno dokaj mlado podjetje, si ne morejo privoščiti dragih podatkovnih baz. Usmerjajo se predvsem v cenejše, odprtokodne rešitve in nove tehnologije. V podjetju je zaposleno manjše število razvijalcev aplikacij, zato stremijo k lažji administraciji. Želijo si tehnologijo, ki bi omogočala razvijalcem lažji začetek in hitre spremembe.

5.5.1.1 Kriteriji odločanja na prvem nivoju

1. Podatkovna shema

Glede na podani opis bi potrebovali prilagodljiv podatkovni model, ki ga zagotavljajo NoSQL podatkovne baze, saj se lahko načini oglaševanja in trženja spreminjajo. V primeru spremembe podatkovnega modela, aplikaciji samo dodamo kodo, ki bo shranila dodatne atribute. Razvoj aplikacij bo tako lažji in hitrejši, spremenjene poslovne zahteve pa hitreje uveljavljene.

2. Struktura in vrednost podatkov

Podatki so sumarni (spremljanje statistike), torej nekritični, med njimi ne nastopajo kompleksna razmerja. So polstrukturirani, saj imajo lahko entitete (npr. oglaševalska kampanja) različne atribute. Kljub temu pa se zahteva atomarnost in izolacija podatkov, saj se lahko istočasno spremeni isti podatek (npr. število dnevnih naročnin). Vse našteto zagotavljajo določene NoSQL podatkovne baze.

3. Dostop do podatkov

Poizvedbe so že vnaprej opredeljene in enostavne. Med podatki ne nastopajo stiki. Gre za poizvedovanje po statistiki za določen dan, po določeni oglaševalski kampanji. Podpora sekundarnih indeksov ni potrebna. Kriteriju najbolje ustrezajo NoSQL podatkovne baze, saj število podatkov nenehno narašča, zato bi bila izdelava statističnih poročil s pomočjo poizvedb SQL na vedno večjih tabelah časovno potratna.

4. Količina podatkov

Število uporabnikov je veliko, po pričakovanjih bo še naraščalo, s tem pa se bo večala tudi količina podatkov in večje število dostopov, zato se raje odločimo za NoSQL.

5. Lastnosti bralnih in pisalnih operacij

Imajo predvsem veliko pisanj in malo branj (večinoma takrat, ko pregledujejo statistiko). Kriteriju ustreza NoSQL.

6. Horizontalna skalabilnost in stroški doseganje le-te

Zaradi velikega števila uporabnikov, velike količine podatkov in napovedi naraščanja, je za učinkovito obravnavo podatkov potrebno particioniranje. NoSQL podatkovne baze zagotavljajo lažjo in cenejšo horizontalno skalabilnost v primerjavi z relacijskimi.

7. Podpora

Pri podpori je v primeru odločitve za NoSQL podatkovno bazo potrebno sprejeti kompromis, saj se ta razlikuje med posameznimi skupnostmi NoSQL. Prav tako je potrebno upoštevati čas za uvedbo NoSQL podatkovne baze.

NoSQL podatkovne baze zahtevajo manj upravljanja pri administraciji, za katero bodo lahko skrbeli razvijalci, zaposleni v podjetju.

8. Varnost

Ker ne gre za upravljanje s kritičnimi podatki, ni potrebe po zagotavljanju visoke varnosti, zato ustreza izbor NoSQL podatkovne baze.

Na podlagi utemeljitve kriterijev za pomoč pri izbiri med relacijsko in NoSQL podatkovno bazo smo ugotovili, da bi hipotetičnemu primeru uporabe najbolje ustrezala NoSQL podatkovna baza. Ugotovitve so povzete tudi v spodnji preglednici 5.2.

Preglednica 5.2. Ugotovitve pri izbiranju med relacijsko in NoSQL podatkovno bazo

Kriteriji	Relacijska PB	NoSQL
Podatkovna shema		✓
Struktura in vrednost podatkov		✓
Dostop do podatkov		✓
Količina podatkov		✓
Lastnosti bralnih in pisalnih operacij		✓
Horizontalna skalabilnost		✓
Podpora	✓	✓
Varnost		✓

5.5.1.2 Kriterij na drugem nivoju

Konsistentnost

Podatki morajo biti točni, konsistentni, saj se lahko istočasno spremeni isti podatek. Zagotoviti je potrebno atomarnost in izolacijo za vsako operacijo, ki dostopa do podatkov znotraj objekta, da se ne pokvarijo pari tipa ključ/vrednost. Torej vsi podatki, ki se povezujejo, morajo biti vsebovani v istem objektu, zaradi potreb atomarnih operacij. Razpoložljivost v tem primeru ni tako pomembna. Zato se raje odločimo za tiste NoSQL podatkovne baze, ki se osredotočajo na zagotavljanje konsistentnosti in particijske tolerance (CP).

5.5.1.3 Kriterij na tretjem nivoju

Podatkovni model

Podatki nakazujejo, da je najprimernejša uporaba dokumentne podatkovne shrambe. Za vsak dan imamo lahko nov dokument (skupaj 365 dokumentov), ki vsebuje statistiko naročnin za ta dan. Takšna podatkovna struktura je boljša, saj je lažje vzdrževati en sam dokument, ki vsebuje vse informacije o dnevni aktivnosti naročnin, kot pa da bi imeli podatke razdeljene po večih relacijskih tabelah. Še en razlog, ki govori v prid dokumentnim shrambam je zahteva po možnosti gnezdenja vrednosti, saj je potrebno npr. znotraj obveščanja po SMS-ih spremljati statistiko še po podatkih kot so ključne besede, omrežna številka mobitela itd.

Poglavje 6

Zaključek

Podatkovne baze NoSQL so s svojim pojavom na trg podatkovnih baz dodobra pretresle domeno relacijskih podatkovnih baz. Postajajo vse bolj pomemben in priljubljen način shranjevanja in obdelave podatkov. Motive za njihov razvoj in uporabo je mogoče povzeti s potrebo po visoki skalabilnosti, hitri obdelavi ogromnih količin podatkov, sposobnosti za porazdelitev podatkov med večimi poceni strežniki ter prilagodljivejšem podatkovnem modelu. Gre za zahteve, ki jim relacijske podatkovne baze ne morejo ustreči oziroma vsaj ne na enostaven in stroškovno učinkovit način.

Relacijske podatkovne baze so zelo dobre pri reševanju določenih problemov za shranjevanje podatkov, toda s tem, ko pojav in uporaba spletnih aplikacij, še posebej socialnih omrežij, strmo naraščata, se hitro povečuje tudi količina podatkov, ki jih je potrebno učinkovito shraniti in obdelati. Čeprav je možno za to uporabiti relacijske podatkovne baze, so podatkovne baze NoSQL bistveno cenejša in enostavnejša rešitev.

V diplomski nalogi smo podali temeljit pregled podatkovnih baz NoSQL. Spoznali smo, da so razlike med NoSQL podatkovnimi bazami veliko večje, kot pa med posameznimi relacijskimi podatkovnimi bazami. Kar pa ni presenetljiv rezultat, saj nobena izmed njih ne predstavlja popolne in univerzalne rešitve vseh problemov shranjevanja in upravljanja s podatki. Vsaka podatkovna baza ima svoje lastnosti in prednosti.

Pri opredelitvi kriterijev odločanja se tako nismo odločili za izbiranje med posameznimi podatkovnimi bazami NoSQL, saj so si med seboj zelo različne. Kriteriji, ki smo jih predlagali, predstavljajo osnovo za bistveno razlikovanje med različnimi podatkovnimi modeli in so uporabno orodje za nadaljnjo usmeritev našega odločanja. Izbor NoSQL podatkovnih baz omejimo na precej manjšo množico, kar podjetju omogoča, da v naslednji fazi mnogo lažje izvede še dejanske meritve na lastni strojni in programski opremi in s tem

najpreprosteje ugotovi in se prepriča, če določena NoSQL podatkovna baza v resnici izpolnjuje njegove zahteve.

Pomembno je, da pri izbiri podatkovne baze izhajamo iz našega problema in na podlagi naših zahtev ugotovimo, kakšno podatkovno bazo potrebujemo. V splošnem bi lahko rekli, da se relacijske podatkovne baze lahko uporabijo za visoko konsistentnost, NoSQL pa predvsem za visoko razpoložljivost in skalabilnost.

Glavna ovira pri izdelavi diplomskega dela je bila pomanjkljiva strokovna literatura na področju NoSQL podatkovnih baz. Omejeni smo bili predvsem na iskanje člankov po spletu, forumih, blogih in dokumentacijo posameznih NoSQL podatkovnih baz. Težko je bilo najti verodostojna gradiva, saj je tehnologija zelo nova in se spreminja iz meseca v mesec, prav tako še ni sprejetih standardov.

6.1 Sklep

Količina raznovrstnih podatkov strmo narašča in s tem tudi zahteve po novih podatkovnih bazah, ki lahko hitro in učinkovito upravljajo ter analizirajo tovrstne podatke. Dejstvo, da se računalništvo vedno bolj seli v oblak, pa k temu dodaja še zahtevo po visoko skalabilnih in visoko razpoložljivih podatkovnih bazah.

NoSQL podatkovne baze so dobra rešitev za tovrstne probleme shranjevanja in obdelave podatkov. S svojim pojavom so ustvarile veliko navdušenja in zapolnile tiste funkcionalnosti, ki manjkajo relacijskim podatkovnim bazam. NoSQL tako ne smemo razumeti kot popolno zavračanje relacijskih (SQL) podatkovnih baz, ampak kot njihovo dopolnitev. Nekatera podjetja, ki uporabljajo in se zavzemajo za NoSQL podatkovne baze, še vedno namreč za poslovno kritične podatke uporabljajo relacijske podatkovne baze.

Po drugi strani pa tudi ponudniki relacijskih podatkovnih baz, predvsem tisti večji kot sta Oracle in IBM, ne zavračajo NoSQL, ampak stremijo k načrtovanju in izgradnji lastnih nerelacijskih rešitev, ki jih bodo lahko uporabljali v kombinaciji z obstoječimi relacijskimi podatkovnimi bazami.

Čeprav so podatkovne baze NoSQL trend na področju shranjevanja ogromnih količin podatkov, obstajajo številne ovire, ki jih morajo še premagati. Bistvo je, da so še vedno nova in zapletena tehnologija. Izbira med njimi je velika, po drugi strani pa zaenkrat še ni sprejetih nobenih standardov. Polega tega so cena in stroški vzdrževanja težko opredeljivi kot tudi sama varnost uporabe. Relacijska podatkovna baza je zaenkrat še vedno dovolj dobra za večino klasičnih poslovnih organizacij.

Vsekakor pa predstavlja dejstvo, da imajo zasluge za pojav in vedno bolj priljubljeno uporabo podatkovnih baz NoSQL velike računalniške organizacije kot so Google, Amazon, Facebook,

Twitter in drugi, močan dokaz, da se bo tehnologija NoSQL še razvijala in da je njena prihodnost svetla.

Kazalo slik

Kazalo preglednic

Literatura in viri

[1] (2012) 10gen. "Craigslist." Dostopno na: http://www.10gen.com/customers/craigslist

[2] (2012) Daniel Abadi, Alexander Thomson. "The problems with ACID, and how to fix them without going NoSQL." Dostopno na: http://dbmsmusings.blogspot.com/2010/08/problems-with-acid-and-how-to-fix-them.html

[3] (2012) Roi Aldaag, Nadav Wiener. "Apples, Oranges and NoSQL." Dostopno na: http://www.slideshare.net/roialdaag/seminar2010nosql

[4] Chris Andersen, Jan Lehnardt, Noah Slater, *CouchDB: The Definitive Guide*, 1st ed. Sebastopol: O'Reilly Media, Inc., 2010, pogl. 2.

[5] (2012) Tony Bain. "Is the Relational Database Doomed?" Dostopno na: http://www.readwriteweb.com/enterprise/2009/02/is-the-relational-database-doomed.php

[6] (2012) Eric Brewer. "Towards Robust Distributed Systems." Dostopno na: http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

[7] (2012) Andrew Brust. "NoSQL and the Windows Azure platform." Dostopno na: http://download.microsoft.com/download/9/E/9/9E9F240D-0EB6-472E-B4DE-6D9FCBB505DD/Windows%20Azure%20No%20SQL%20White%20Paper.pdf

[8] (2012) Greg Burd. "NoSQL." Dostopno na: http://www.usenix.org/publications/login/2011-10/openpdfs/Burd.pdf

[9] (2012) Rick Cattell. "Scalable SQL and NoSQL Data Stores." Dostopno na: http://cattell.net/datastores/Datastores.pdf

[10] (2012) Michael Cobb. "Comparing relational database security and NoSQL security." Dostopno na: http://searchsecurity.techtarget.com/answer/Comparing-relational-database-security-and-NoSQL-security

[11] Thomas Connolly, Carolyn Begg, Database Systems: A Practical Approach to Design, Implementation, and Management, 4th ed.: Addison Wesley, 2005, pogl. 1, 2.

[12] (2012) Stefan Edlich. "NOSQL Databases." Dostopno na: http://nosql-database.org/

[13] (2012) Emil Eifrem. "NOSQL: scaling to size and scaling to complexity." Dostopno na: http://blogs.neotechnology.com/emil/2009/11/nosql-scaling-to-size-and-scaling-to-complexity.html

[14] (2012) Klint Finley. "Why Large Hadron Collider Scientists are Using CouchDB." Dostopno na: http://www.readwriteweb.com/enterprise/2010/08/lhc-couchdb.php

[15] (2012) Google. "Google Trends - NoSQL." Dostopno na: http://www.google.com/trends/?q=nosql

[16] (2012) Coda Hale. "You Can’t Sacrifice Partition Tolerance." Dostopno na: http://codahale.com/you-cant-sacrifice-partition-tolerance/

[17] (2012) Derrick Harris. "Couchbase goes 2.0, pushes SQL for NoSQL." Dostopno na: http://gigaom.com/cloud/couchbase-2-0-unql-sql-nosql/

[18] (2012) Tom Haughey. "The Role of Data Architecture in NoSQL." Dostopno na: http://www.dama-phila.org/HaugheyNOSQL.pdf

[19] Eben Hewitt, *Cassandra: The Definitive Guide*, 1st ed. Sebastopol: O'Reilly Media, Inc., 2010, pogl 1.

[20] (2012) IDC. "The 2011 Digital Universe Study: Extracting Value from Chaos." Dostopno na: http://www.emc.com/collateral/demos/microsites/emc-digital-universe-2011/index.htm

[21] (2012) InformationWeek. "New York Stock Exchange Ticks on Data Warehouse Appliances." Dostopno na: http://www.informationweek.com/news/software/bi/207800705

[22] (2012) Tobias Ivarsson. "NoSQL for Dummies." Dostopno na: http://www.slideshare.net/thobe/nosql-for-dummies

[23] (2012) Joab Jackson. "CouchBase, SQLite launch unified NoSQL query language." Dostopno na: http://www.arnnet.com.au/article/395469/couchbase_sqlite_launch_unified_nosql_query_language#closeme

[24] (2012) Michael Kopp. "NoSQL or RDBMS? – Are we asking the right questions?" Dostopno na: http://blog.dynatrace.com/2011/10/05/nosql-or-rdbms-are-we-asking-the-right-questions/

[25] (2012) Cade Metz. "Facebook: Why our 'next-gen' comms ditched MySQL." Dostopno na: http://www.theregister.co.uk/2010/12/17/facebook_messages_tech/

[26] (2012) Kannan Muthukkaruppan. "The Underlying Technology of Messages." Dostopno na: https://www.facebook.com/notes/facebook-engineering/the-underlying-technology-of-messages/454991608919

[27] (2012) Kai Orend. "Analysis and Classification of NoSQL Databases and Evaluation of their Ability to Replace an Object-Relational Persistence Layer." Dostopno na: http://wwwmatthes.in.tum.de/file/Publications/2010/Or10/Or10.pdf

[28] (2012) Rick Osborne. "Playing around with MongoDB and MapReduce functions." Dostopno na: http://rickosborne.org/blog/2010/02/playing-around-with-mongodb-and-mapreduce-functions/

[29] (2012) Srini Penchikala. "Virtual Panel: Security Considerations in Accessing NoSQL Databases." Dostopno na: http://www.infoq.com/articles/nosql-data-security-virtual-panel

[30] (2012) Courtney Robinson, Eric Evans."CQL - Cassandra Query Language." Dostopno na: http://www.slideshare.net/shotaz/cql-cassandra-query-language

[31] (2012) Christof Strauch. "NoSQL Databases." Dostopno na: http://www.christof-strauch.de/nosqldbs.pdf

[32] (2012) David Strom. "NoSQL: Breaking free of structured data." Dostopno na: http://www.itworld.com/data-centerservers/172477/nosql-breaking-free-structured-data

[33] (2012) The Book Of Brilliant Things. "Data Models." Dostopno na: https://www.bookofbrilliantthings.com/book/rts/data-models

[34] Shashank Tiwari, *Professional NoSQL*, 1st ed. Indianapolis: John Wiley & Sons, Inc., 2011, pogl. 1, 9, 14, 15.

[35] Tom White, *Hadoop: The Definitive Guide*, 1st ed. Sebastopol: O'Reilly Media, Inc., 2009, pogl 1.

[36] (2012) Wikipedia. "Apache Hadoop." Dostopno na: http://en.wikipedia.org/wiki/Apache_Hadoop

[37] (2012) Wikipedia. "NoSQL." Dostopno na: http://en.wikipedia.org/wiki/NoSQL

[38] (2012) Wikipedia. "Podatkovna baza." Dostopno na: http://sl.wikipedia.org/wiki/Podatkovna_baza