Alati za razvoj softvera

Baze podataka, skaliranje

Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Relacione baze

- Mogu da posluže u velikom broju slučajeva
- Prilično stare ljudi ih korsite dovoljno dugo
- Podaci su struktuirani
- Podaci se nalaze u tabelarnom obliku
- Svaka torka ima jedinstven primarni ključ u kao
  - prirodno obeležje, neko od odeležja iz torke
  - surogat ključ, dodato obeležje da izigrava jedinstvenost

- Podaci se često nalaze u nekoliko tabela
- Ako treba da dobijemo kompletnu sliku moramo da uradimo operaciju spajanja (Join)
- U nekim situacijama ovo je usko grlo sistema
- Ako imam puno tabela, puno podataka i egotične spojeve to može da traje
- Moguće ubrzati pretrage dodavanjem indeksa
- Moderne relacione baze dopuštaju čak da se skladište i podaci u binarnom obliku, ili recimo JSON podaci

- Malo liče na objektni model
- Tablea je klasa, dok je jedna torka zapravo instanca klase
- Ova analogija je dovela do objektno-relacionog mapiranja
- Pokušati nekako klasu mapirati na tabelu, a atribute na kolone te tabele
- Objektno-relaciono mapiranje je tehnika konvertovanja podataka koji se cuvaju kao objekti u memoriji u
  - podatke koji se mogu poslati u bazu podataka
  - ali se mogu i podaci iz baze konvertovati u objekte odgovaraju ceg tipa u memoriji

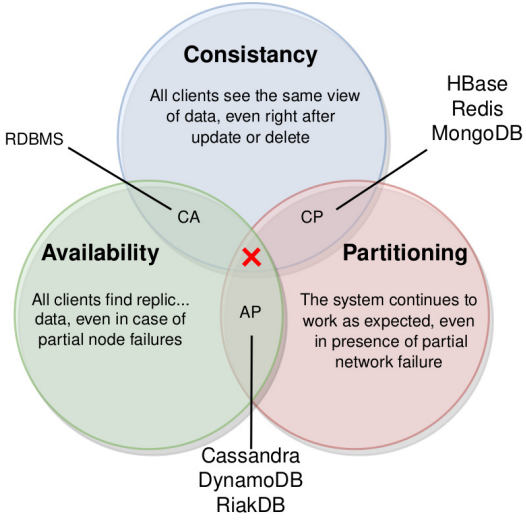
- Prednost ovog pristupa je u tome što možemo raditi sa klasa omiljenog programskog jezika bez da drljamo sa goilm SQL naredbama
- Ovo je korisno za većinu aplikacija, ali ako trebate da radite neke kompleksne upite i analitiku možda he bolje koristiti klasičan SQL
- Pored toga neke od koncepata objektne paradigme nije tako jednostano prebaciti u OR mapere
- Posto razne strategije koje se bve ovim, ali generalno toga korisnici moraju biti svesni
- Prilično popularna i korisna tehnika kod razvoja aplikacija

NoSQL baze

- NoSQL pokret reprezentuje sve tipove baza podatka koje nisu relacione baze podataka
- Njihov princip modelovanja, model skladištenja, upita i svih ostalih operacija razlikuje se dosta od relacionih baza
- NoSQL baze imaju nekoliko modela podataka
  - Key-value
  - Column
  - Graf
  - Document
  - Time series
  - Mixed models

- Koriste se u raznim situacijama
  - kada model podataka sa kojim se radi nije relacioni po prirodi
  - radimo sa velikim količinama podataka
  - ima potrebe za intenzivnim skaliranjem
  - kada je potrebno da samo serijalizujemo/deserijalizujemo nekakv podaataka (JSON, XML, itd.)
- Često se koriste u cloud okruženju
- Dosta se koriste u mikroservisnim arhitekturama

- Na važnosti počinju da dobijaju jos u ranim nastancima cloud computing-a
- Razlog za to je nemogućnost relacionih baza da se primene i distribuiranim sistemima zbog cuvene CAP teoreme
- Ova teorema kaze da ne mozemo zadovoljiti sve tri osobine u isto veme:
  - Konzistentnost
  - Dostupnost
  - Particioniranje
- Uvek možemo da dobijemo najviše dve od tri osobine
- Na korisnicima je da odluče šta od tih osobina zele da imaju
- Particioniranje nije moguće izbeći u distribuniram sistemima


Consistancy
All clients see the same view of data, even right after update or delete
HBase
Redis
MongoDB
RDBMS
CA
CP
Availability
All clients find replic... data, even in case of partial node failures
AP
Partitioning
The system continues to work as expected, even in presence of partial network failure
Cassandra
DynamoDB
RiakDB

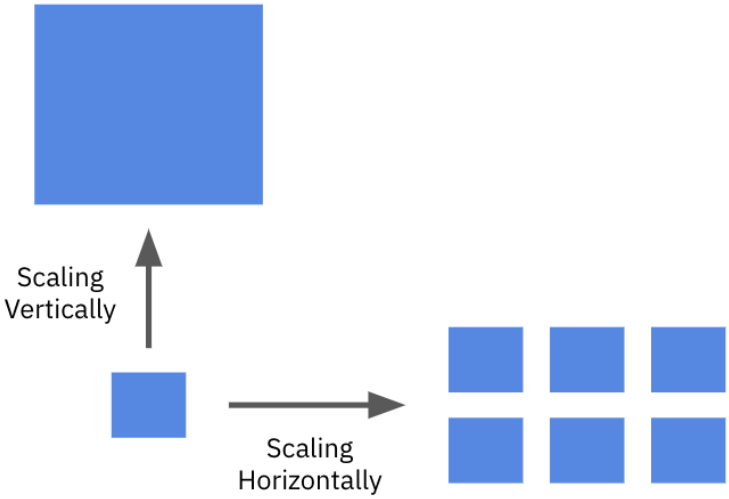
(CAP teorema)

- Ključna stvar kod NoSQL baza podataka jeste da poznajemo svoje upite
- Unapred treba znati kakve upite želimo da radimo nad podacima
- Koristeći to znanje, biramo model i bazu podataka
- Ovi tipovi baza obično ne podrzavaju join operacije, imaju restriktivnije transakcije od relacionih baza i često slbodmniju formu podataka
- Zbog ovih osobina izuzetno se lako koriste iz prostog razloga mogućnosti da domen problema prilagodimo bazi
- Izbegavamo dovijanje u fiksnom modelu relacionih baza

- To ne znači da transakcije nije moguće implementirati u NoSQL bazama
- One obično implementiraju na aplikativnom sloju (npr. SAGA patern)
- Moguće je i da budu deo baze kao recimo distribuirane transkacije (2 phase commit, 3 phase commit, razni drugi algoritmi distribiurnaih sisitema)
- Distribuirane trasnakcije, nisu nižno jednostavne za postići i treba ih izbegavati
- Druga stvar koju treba izbegavati jeste da forsirate model podataka spram baze koju volite!
- Kada ste u dilemi uvek konsultujte domen problema, i shodno tome izaberite najbolji/najprirodniji model!!
- Još jedna napomena je da su kod ovih baza ključevi UUID a ne brojevi ili obeležja

Uvod

- Skaliranje je proces u kom omogućavamo našoj aplikaciji/sistmeu da opslužuje rastući broj korisnika
- Kada se priča o skaliranju obično se misli na jedan od dva načina skaliranja koja su bitno različita
  - Vertikalno skaliranje je proces u kom se *uzima jača mašina*, tj. više resursa i u ovom principu ne postoji redudancija ako sistem padne naša aplikacija je nedostupna — preferenrani način skaliranja relacionih baza
  - Horizontalno skaliranje je proces skaliranja za *veće* aplikacije, gde se aplikacija izvršava na više mašina a zahtevi se balansiraju. Ako jedna mašina padne, uvek imamo druge koje mogu da prihvate zahtev — preferiran način skaliranja u cloud sistemima i NoSQL bazama



(Horizontalno i Vertikalno skaliranje)

Uvod

- Consul je jedna od mnoštvo NoSQL baza podataka koje koriste Key-Value model podataka
- Ovaj model je jedan od najstarijih modelima korišćen je dosta i u samom UNIX operativnom OS-u
- Sve što nam ovaj model omogućava jeste da pod nekakvim klju'čem tipa string, sačuvamo proizvoljan niz bajtova tj. bilo koji podatak
- Isto tako, ako tražite vrednost pod odgovarajućim ključem dobićete nazad kompletan podatak koji se čuva pod tim ključem
- Izuzetno jednostavan model, izuzeno koristan model, izuzetno moćan model za razne alate i servise a vrlo sličan strukturi podataka *hash map*

Naprednije funkcije

- Ovaj model ima i *Time to Live (TTL)* mehanizam
- Odnosno vremenski čuva podatak
- Ovo može biti zgodno za implementaciju funkcionalnosti vaših aplikacija
- Ova operacija nije tako jednostavna za implementriati u relacionij bazi, i obično zahteva upotrebu još minimalno jednog alata

- Ova baza, kao i mnoge druge Key-Value podržavaju i *prefix scan*
- Pretraga svih ključeva na osnovu prefixa — nešto slično select naredbi
- Koristeći ovu operaciju možete jednostavno vratiti sve podatke korisnika
- Naravno, moramo voditi računa kako dizajniramo ključeve da bi podržali ovu opciju
- Na primer:
  - key - user123_name, value - Miloš
  - key - user123_lastname, value - Simić
  - Ako radimo prefix skan sa user123, dobijamo nazad oba podatka!

Rad sa Consul-om u Go-u

- Prvo ćemo napraviti strukturu koja ima konekriju ka bazi, i na nju ćemo nakačiti potrebne funkcije (slično kao kod servisa)
- Odgovarajući endpoint će samo pozivati ove funkcije da bi imao interakciju sa bazom — podela nadležnosti

```
package poststore

type PostStore struct {
        cli *api.Client
}
```

- Zatim ćemo prebaciti *model.go* u paket *poststore* zarad lakšeg rada

- Nakon toga ćemo napraviti funkciju koja pravi novu promenljivu i uspostavlja konekciju ka bazi

```
func New() (*PostStore, error) {
        db := os.Getenv("DB")
        dbport := os.Getenv("DBPORT")

        config := api.DefaultConfig()
        config.Address = fmt.Sprintf("%s:%s", db, dbport)
        client, err := api.NewClient(config)
        if err != nil {
                return nil, err
        }

        return &PostStore{cli: client}, nil
}
```

- Napravićemo funkciju koja generiše ključ (uuid) za nas

```
func generateKey() (string, string) {
        id := uuid.New().String()
        return fmt.Sprintf("posts/%s", id), id
}
```

- Napravićemo funkciju koja rekonstruiše ključa nas

```
func constructKey(id string) string {
        return fmt.Sprintf("posts/%s", id)
}
```

- Funkcija za kreiranje post-a

```
func (ps *PostStore) Post(post *RequestPost) (*RequestPost, error) {
        kv := ps.cli.KV()
        sid, rid := generateKey()
        post.Id = rid

        data, err := json.Marshal(post)
        if err != nil {
                return nil, err
        }

        p := &api.KVPair{Key: sid, Value: data}
        _, err = kv.Put(p, nil)
        if err != nil {
                return nil, err
        }
        return post, nil
}
```

- Funkcija za vraćanje post-a

```
func (ps *PostStore) Get(id string) (*RequestPost, error) {
        kv := ps.cli.KV()
        pair, _, err := kv.Get(constructKey(id), nil)
        if err != nil {
                return nil, err
        }

        post := &RequestPost{}
        err = json.Unmarshal(pair.Value, post)
        if err != nil {
                return nil, err
        }
        return post, nil
}
```

- Funkcija za vraćanje svih postova

```
func (ps *PostStore) Get(id string) (*RequestPost, error) {
        kv := ps.cli.KV()
        pair, _, err := kv.Get(constructKey(id), nil)
        if err != nil {
                return nil, err
        }

        post := &RequestPost{}
        err = json.Unmarshal(pair.Value, post)
        if err != nil {
                return nil, err
        }
        return post, nil
}
```

- Funkcija za brisanje posta

```
func (ps *PostStore) Delete(id string) (map[string]string, error) {
        kv := ps.cli.KV()
        _, err := kv.Delete(constructKey(id), nil)
        if err != nil {
                return nil, err
        }
        return map[string]string{"Deleted":id}, nil
}
```

Dodatni materijali

- Introduction to NoSql by Martin Fowler
- NoSQL Distilled to an hour by Martin Fowler
- Introduction to database scalability
- SQL vs NoSQL
- Consul docs

Kraj predavanja

Pitanja? :)