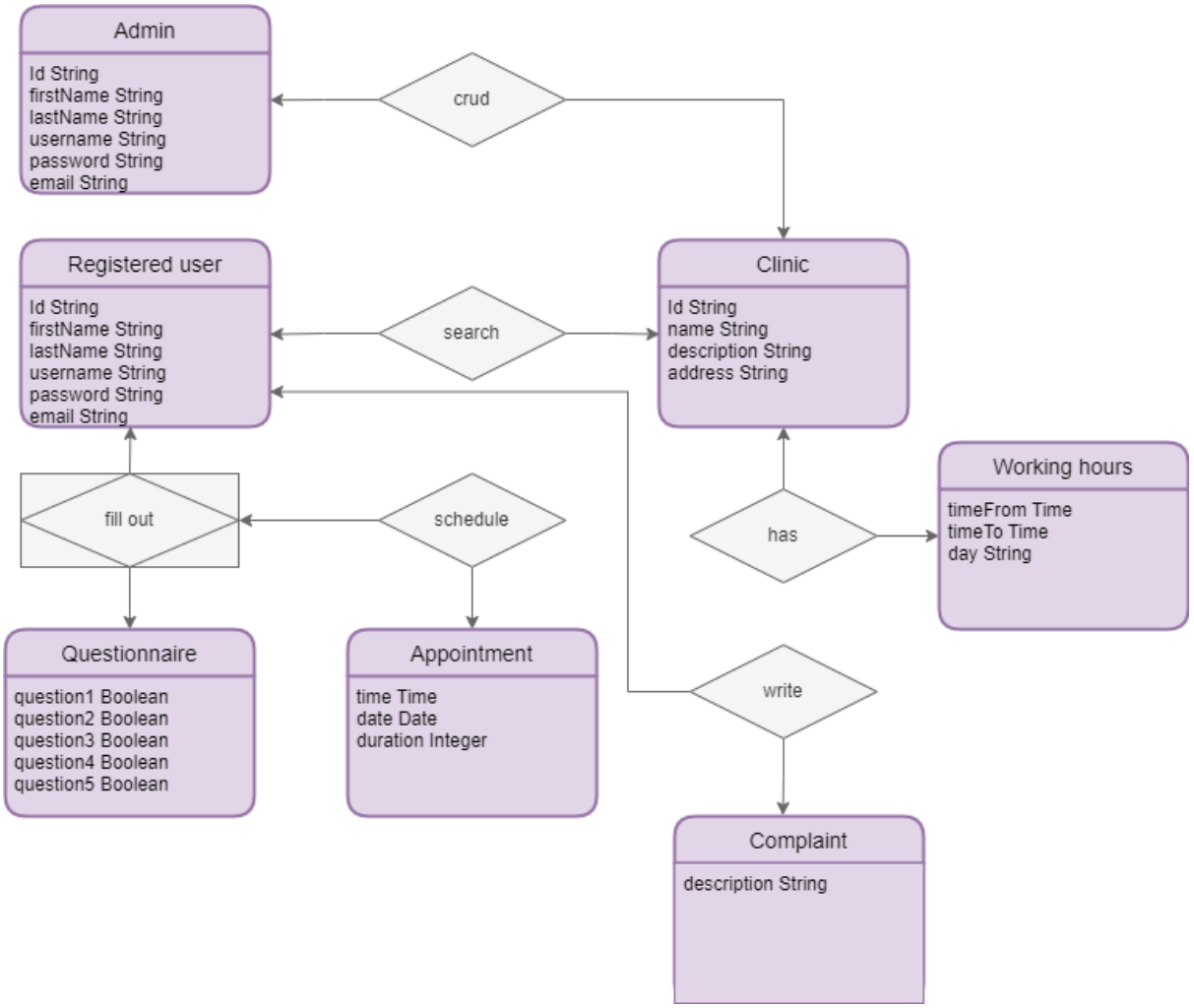
Proof of Concept(PoC)

Sadržaj

Uradila:

Anamarija Popović Dolovac IN34/2018

1. Dizajn šeme baze podataka



2. Predlog strategije za particionisanje podataka

U našem slučaju je neophodno primeniti particionisanje. Neki od načina optimizacije su: dobar server, dobro iskonfigurisan MySql, poboljšanje upita… Kada kažem poboljšanje upita mislim isključivo na njihovo "brže" izvršavanje ali i to nekada ne mora mnogo da utiče na sporost sistema, pa onda je najbolje da se usredsredimo na samu strukturu i kvalitet podataka.

Sama baza sistema ne sadrži mnogo tabela ni mnogo kolona po horizontali, što znači da je baza dobro normalizovana. Atributske tabele mogu biti reda velicine par stotina do par hiljada slogova sto nije puno, ali „glavna" velika tabela, iako dobro normalizovana, može imati preko milion slogova. Upiti koji se izvršavaju nad tom tabelom i množe sa atributskim tabelama zahtevaju prolazak kroz veliki broj slogova i ma koliko indexi bili korisni ako imamo agregatnu funkciju nad velikom količinom slogova, to prosto znači da prolazimo kroz slogove, pročitamo svaki, uzmemo vrednost, ubacimo je u agregatnu funkciju(što nije efikasan način) .

MySql podržava partioning od verzije 5.1. MySQL partitioning radi tako da tabelu podeli u male tabele (svaka particija fizicki na disku postane zasebna tabela) i kada optimizer prepozna da može upit da izvrši nad samo jednom particijom, operaciju vrši nad tom jednom tabelom, ako upit mora da izvršava nad više tabela, onda se određuje da li može da sekvencijalno koristi jednu pa drugu ili mora da pravi „join" između particija. U slučaju da se pravi join tada je pristup particionisanim podaci nesto sporiji nego kada oni nisu particionisani. Ono sto je kod particionisanja vrlo zgodno je to što možemo brisati podatke iz particije i popraviti jednu particiju ukoliko dođe do korupcije.

Efekti operacija particionisanja na performanse:

1. File system operations - brzina operacija za particionisanje i reparticionisanje tipa (ALTER TABLE with PARTITION BY) zavisi dosta od tipa i karakteristika fajl sistema, brzine diska, brzine rukovođenja fajlovima od strane operativnog sistema
2. Table locks - proces koji izvršava operaciju particionisanja na tabelu upisuje lock i zaključava tabelu, tako da su na čekanju sve operacije tipa INSERT, UPDATE dok se operacija ne završi.
3. Indexes, partitioning pruning - upotreba odgovarajućih indeksa znatno može ubrzati upite nad bazom i samim tim i rad sistema.

3. Predlog strategije za replikaciju baze i obezbeđivanje otpornosti na greške

Snapshot replication - je jedna od najkorišćenijih strategija za replikaciju kod onih sistema koji imaju neredovne izmene podataka (korisnicima će se čuvati podati prilikom registracije, i neće svaki korisnik vršiti moguću izmenu istih), replicira malu količinu podataka (što bi nam i odgovaralo).

Ukoliko nam se promeni adresa centra, što je malo ređi slučaj, ali na primer jednom godišnje, nakon te izmene je potrebno uraditi snapshot. Sve izmene u toku dana koje su ključne kao što su izmena lozinke, profila korisnika, radno vreme centra, otkazivanje termina, neophodno je odmah uraditi snapshot replikaciju baze/tabele kako bismo imali sačuvane verzije podataka koje su od velikog značaja za dalje funkcionisanje sistema.

Par koraka kojih se treba pridržavati radi poboljšanja bezbednosti softvera i povećanja otpornosti na greške:

1. Standardi konzistentnog programiranja - pridržavati se file-naming konvencije, reprezentacije ne-ASCII karaktera
2. Korišćenje automatskih alata za testiranje
3. Negativno testiranje i pozitivno testiranje
4. Primena biblioteka za jednostavne taskove (npr. validacija unosa korisnika)
5. Pregled koda
6. Implementirati use cases, misuse cases i abuse cases od samog početka procesa

4. Predlog strategije za keširanje podataka

Strategija koju sam odabrala za keširanje podataka je Cache-Aside (Lazy Loading). Strategija funkcioniše tako što se ne šalje svaki korisnikov zahtev bazi, nego se prvo proverava da li su podaci dostupni iz keša. Ukoliko su podaci dostupni prvo u kešu (a cache hit), keširani podaci se vraćaju korisniku kao odgovor na zahtev. Ukoliko podaci nisu dostupni iz keša (a cache miss), upit šaljemo bazi i podaci se vraćaju korisniku kao odgovor na zahtev.

Na ovaj način bismo uštedeli na vremenu izvršavanja upita, odnosno na brzini odgovora na korisnikov zahtev. Kao rezultat ovog pristupa imamo direktno poboljšanje performansi.

5. Okvirna procena za hardverske resurse potrebne za skladištenje svih podataka u narednih 5 godina

Predložena infrastruktura koja može da skladišti količinu podataka koja će akumulirati kroz 5 godina bi bili serveri koji bi čuvali desetine PB podataka. Takođe ti podaci bi bili replicirani na pomoćni server kako bi se sačuvali usled pada glavnog servera. Ne bih odabrala nekog Cloud vendora jer su podaci privatni i bolje je da se nalaze na našem serveru.

Takođe ako bi naši korisnici bili amerikanci morali bi da ispoštujemo HIPAA zakon i to je još jedan razlog za čuvanje podataka na našim serverima.

6. Predlog strategije za postavljanje load balansera

Strategija koju sam odabrala za postavljanje load balancera je Weighted Round Robin, load balancing algoritam, koji ima mogućnost da rasporedi klijentske zahteve "farmi" servera na osnovu njihovog kapaciteta.

Na primer, aplikacija ima 3 servera, gde je prvi server znatno jači od druga dva. Ukoliko dobijemo pet sekvencijalnih zahteva klijenata, prva dva zahteva idu prvom serveru (zbog njegove jačine), treći zahtev ide drugom serveru, četvrti zahtev trećem serveru i opet peti zahtev prvom serveru. Na ovaj način lepo raspoređujemo zahteve po njihovoj težini i sprečavamo preopterećenje servera kao i mnoge druge komplikacije (npr. velika količine zahteva za registraciju i aktiviranje email servisa za verifikaciju, želimo da omogućimo odmah korisnicima pristup našem sistemu bez imalo čekanja, a to ćemo uraditi samim raspoređivanjem zahteva po serverima).

7. Predlog koje operacije korisnika treba nadgledati u cilju poboljšanja sistema

Operacije korisnika koje je potrebno nadgledati su:

1. Brzina izvršavanja zahteva korisnika - kako bismo poboljšali performanse
2. Snalaženje korisnika kroz sistema - da li se lako kreću i snalaze kroz sistem
3. Jednostavnost korišćenja sistema
4. Prilagođenost sistema korisnicima - da li je sistem prilagođen ulogama korisnika u stvarnom životu