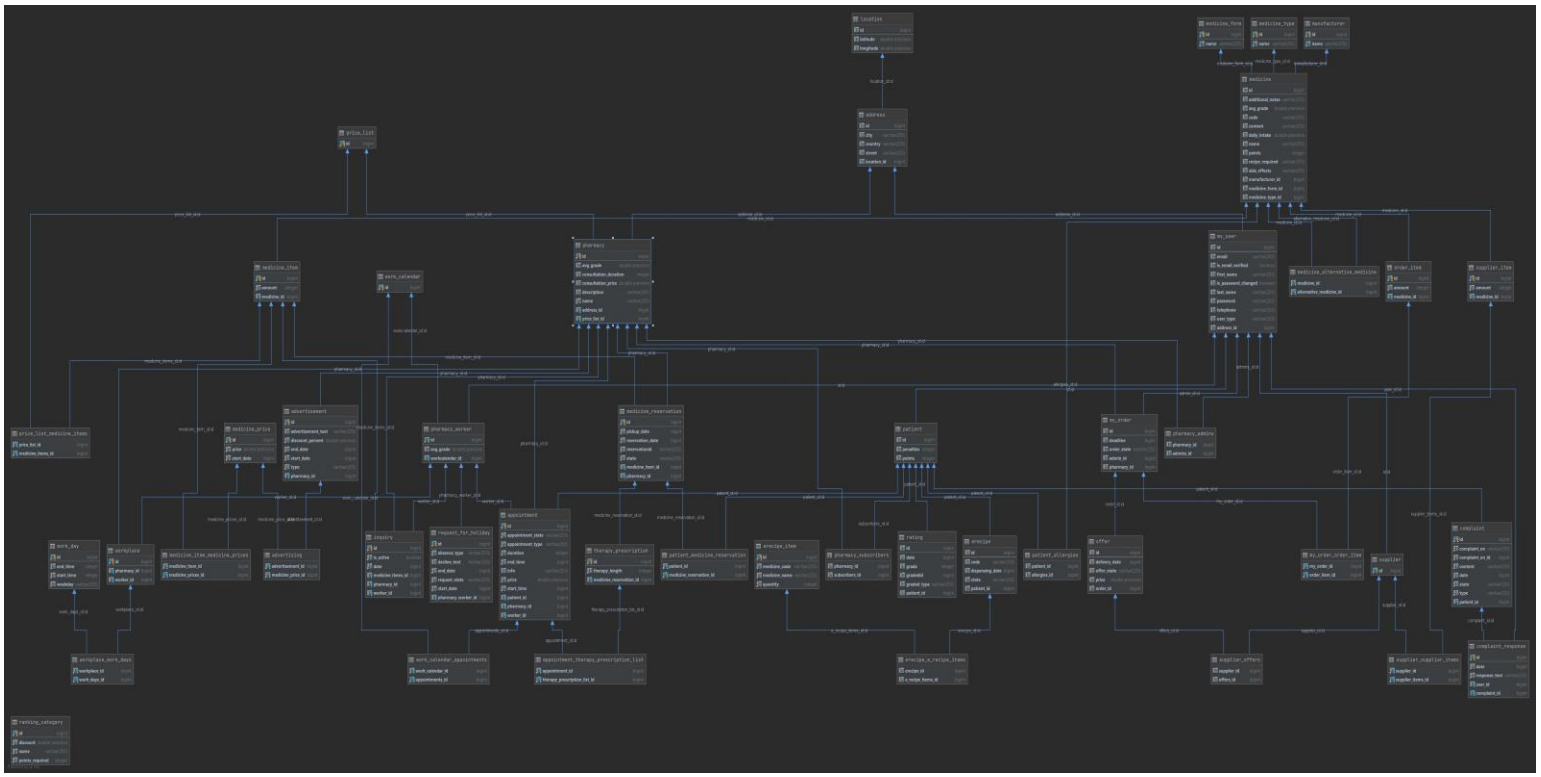
Skalabilnost

1. Šema baze podatke



2. Particionisanje podataka

Kako je predviđeni broj korisnika aplikacije velik (oko 200 miliona), što predstavlja znatnu količinu podataka smeštenih u jednu bazu, može doći do stvaranja tabela sa ogromnim brojem slogova. Usled toga, može doći do usporavanja procesa čitanja, što predstavlja čestu akciju u sistemu i može dovesti do zastoja prilikom korišćenja. Jedno od rešenja ovakvog problema predstavlja horizontalno i vertikalno particionisanje.

Uzevši u obzir model podataka prikazan na prethodnom dijagramu, kao i njihov sadržaj u okviru implementacije sistema, vertikalno particionisanje nije od velike pomoći. Kako se podaci uglavnom sastoje od relativno kratkih atributa, nema potrebe za njihovim vertikalnim particionisanjem u posebne tabele. Stoga, bolje rešenje u ovom programu predstavlja horizontalno particionisanje u ključnim delovima sistema, pri čemu će se smanjiti broj elemenata po tabelama, i na taj način ubrzati proces čitanja.

Uzevši u obzir procenjen broj korisnika, kao i broj pregleda i rezervacija na mesečnom nivou, neke od bithih tabela u kojima bi bilo uvedeno horizontalno particionisanje su:

- Appointment
- Medicine
- MedicineItem
- My_user
- Pharmacy

Ove tabele predstavljaju grupe podataka kojima se često pristupa od strane korisnika sistema. Radnici (farmaceuti i dermatolozi) često pristupaju svojim sastancima, budućim ili prethodnim, kao i klijenti koji su ih rezervisali. Pored toga, čest je pristup i entitetima *medicine* i *medicineItem* koji predstavljaju lekove koji se nalaze na sistemu, kao i njihovo stanje u svakoj apoteci. To je posledica toga što će klijenti prilikom rezervacije leka morati da izrvše njihovu pretragu, a to će isto morati i da urade admini apoteka ili radnici prilikom definisanja terapija. Njihov broj može bitno znatno velik, te horizontalno particionisanje može znatno da ubrza njihov pristup. Isto važi i za samu tabelu korisnika, kao i za listu apoteka.

Horizonaltno particionisanje će se u ovom slučaju vršiti po opsegu. Ukoliko bismo se odlučili za particionisanje na osnovu heširanja, potencijalan problem koji bi nastao u budućnosti jeste potreba za znatnim brojem prebacivanja elemenata između tabela usled novih particionisanja, kao i zbog toga što će slogovi biti nasumično raspoređeni po particionisanim tabelama, čime se gubi lokalnost podataka. Sa druge strane, particionisanje po opsegu daje mnogo jednostavnije rešenje za raspodelu slogova, pri čemu se njihovo pozicioniranje po particionisanim tabelama vrši po određenom pravilu. Na taj način, omogućićemo brzu pretragu po kriterijumu raspoređivanja, po cenu pojavljivanja tabela koja sadrže veći broj podataka od drugih, usled nebalansiranosti kriterijuma particionisanja (ukoliko pravilom raspodele bude određeno da jedna particija tabele sadrži znatno veći broj slogova od drugih), a čime nastaje usko grlo.

3. Replikacija i obezbeđivanje od grešaka

Kako je nužno održavati replikacije baze podataka u cilju ubrzavanja procesa čitanja/pisanja, kao i u cilju obezbeđivanja od grešaka odnosno pada baze podataka, potrebno je kreirati više baza sa identičnim podacima. U ovom slučaju, ideja je da se kreira jedna glavna baza (master) preko koje će se vršiti upis i čitanje (u krucijalnim situacijama), kao i više pomoćnih baza (slave). U okviru pomoćnih baza, koje će registrovati promene nastale prilikom upisa na glavnu, i ažurirati svoj sadržaj tako da održe konzistentnost, biće vršeno isključivo čitanje. Na taj način, postojanjem više pomoćnih baza omogućiće se brži pristup podacima i smanjenje opterećenja sa jedne baze na više njih. Takođe, na ovaj način se baza obezbeđuje od grešaka, odnosno u slučaju greške ili pada sistema na glavnoj bazi, u bilo kom trenutku neka od pomoćnih će moći da preuzme njenu ulogu pisanja. Jedna od mana ovog pristupa je ta da ukoliko dođe do pristupa podacima sa neke od pomoćnih baza u momentu kada glavna počinje da im šalje informacije o izmenama, te izmene neće biti poslate do korisnika.

4. Keširanje podataka

Jedan od bitnih načina ubrzanja sistema jeste kreiranje servera za keširanje podataka iz baze. U slučaju čitanja podataka, koristićemo Cache-Aside metod, koji se zasniva na čitanju keš podataka pre pristupa bazi, i dobijanju podataka od njega u slučaju pogotka, odnosno od baze u slučaju da se zahtevani podaci ne nalaze u keš serveru. Ovaj pristup je pogodan za našu aplikaciju s obzirom da se veći deo aplikacije zasniva na prikupljanju podataka iz baze: lekovi, apoteke, istorija sastanaka itd. Mana ovakvog pristupa jeste to što ukoliko se podaci ne nalaze u keš serveru, sam zahtev za podacima mora dalje da se pošalje bazi, što može da uspori tok podataka. Međutim, ovakav pristup prioritizuje brzinu pre svega, što je korisno za klijenta koji ima potrebu da pretraži razne lekove, ili da vidi istoriju sastanaka ili zakaže novi. Takođe, ovakav pristup je koristan zbog dostupnosti sistema, jer ukoliko dođe do greške ili pada baze ili keš servera, sistem će lako moći da se oporavi, tako što će zahtev biti poslat na neki drugi bazni server.

Što se tiče čitanja, odabrana metoda jeste Write-Around. Ova strategija se zasniva na tome da se upis podataka šalje direktno ka bazi, dok se u kešu nalaze samo podaci koji su iščitani iz baze. Ovim pristupom, obezbedićemo da se u kešu nalaze podaci kojima se najčešće pristupa. Ovo je veoma korisno za sisteme naručivanja slične ovome, u okviru koga se korisniku nude lekovi ili apoteke kojima se najčešće pristupa, odnosno popularni entiteti. Jedan od problema ovakvog pristupa je taj što se podaci u keš memoriji ažuriraju samo prilikom neuspešnog traženja podataka u kešu, što može dovesti do problema nekonzistentnosti iščitanih podataka. Da bi se rešio ovaj problem, koristiće se TTL (time to live) pristup, koji će označiti podatke zastarelim nakon određenog vremena, nakon čega će keš server tražiti nove podatke iz baze i na taj način ih ažurirati. Kao jedan od popularnijih sistema za keširanje, biće korišćen Redis (Remote Dictionary Server).

5. Okvirna procena za hardverske resurse u narednih pet godina

Na osnovu okvirnih računanja glavnih komponenti baze, dobili smo kao rezultat sledeće procene:

- Sastanci:
    - po našem proračunu, svaki sastanak u bazi zauzima oko 0.6KB
    - kada se taj broj pomnoži sa brojem meseci u narednih 5 godina, kao i sa prosečnim brojem sastanaka na mesečnom nivou (1 000 000) dobije se **36GB**
- Korisnici
    - po našem proračunu, svaki korisnik u bazi zauzima oko 0.8KB
    - kada se taj broj pomnoži sa brojem korisnika (200 000 000) dobije se **156GB**
- Rezervacije
    - po našem proračunu, svaka rezervacija u bazi zauzima oko 0.5KB
    - kada se taj broj pomnoži sa brojem meseci u narednih 5 godina, kao i sa prosečnim brojem rezervacija na mesečnom nivou (1 000 000) dobije se **30GB**
- Žalbe
    - po našem proračunu, svaka žalba u bazi zauzima oko 0.7KB

- kada se taj broj pomnoži sa očvekivanim brojem žalbi (100 000 000 – polovina broja korisnika) dobije se **70GB**
- za svaku žalbu je potrebno napisati i odgovor na žalbu, što je otrpilike još toliko zauzeća memorije (**70GB**)

Ukupno očekivano zauzeće od prethodno navedenih entiteta je otprilike **370GB**. Kako u tu procenu nisu uračunati svi ostali entiteti iz baze, kao ni potencijalno veći broj korisnika, ova procena će biti pomnožena sa 5, što daje oko **1800GB**. Stoga, naša procena za hardversko memorijsko zauzeće jeste oko **2-3TB**.

6. Strategija za postavljanje load balansera

Kad je u pitanju *load balancer*, strategija za koju smo se odlučili je *Least Connection*. Proučavajući i ostale tipove, uočili smo da svaki od njih ima negativne strane. Round Robin ne uzima u obzir, dužinu konekcija i time može doprineti mnogo većem saobraćaju na jednom serveru. Dok algoritam heširanja IP adrese ima manu da sav saobraćaj sa jedne IP adrese, uvek ide na isti server i ukoliko je količina zahteva poslata sa te adrese velika, server može biti preopterećen. Kod Least Connection-a je problem dužih konekcija takođe moguć, međutim on se sa njima bolje izbori i nikad nije drastična razlika između broja konekcija na tom serveru i ostalima.

7. Operacije korisnika koje treba nadgledati u cilju poboljšanja sistema

Posmatranje akcija korisnika u okviru našeg sistema je veoma bitno iz razloga što često ne možemo da predpostavimo na koji način će se ponašati korisnici, a samim tim i sistem koji smo inicijalno postavili. Informacije koje prikupljamo nadgledanjem akcija korisnika možemo iskoristiti u te svrhe da unapredimo sistem i time poboljšamo iskustvo koje korisnici imaju dok koriste našu aplikaciju.

Neophodno je posmatrati odnos broja čitanja spram broja pisanja u bazu podataka. Ukoliko je broj čitanja mnogo veći nego što je očekivano, možemo još dodati pomoćnih (slave) baza.

Takođe u obzir treba uzeti i broj čitanja nekonzistentnih podataka iz keša. Ukoliko taj broj pređe prihvatljivu granicu, možemo prilagoditi naš prvobitni izbor strategije keširanja i zameniti je nekom prikladnijom za tu konkretno situaciju.

Ukoliko broj korisnika naglo poraste, i time naše baze postanu nedovoljne za potrebe servera, možemo uzeti u obzir kreiranje još jedne glavne (master) baze, koja bi mogla biti geografski bliža korisnicima i time bi smanjili opterećenje konekcija na prvobitnu glavnu bazu. Ukoliko bi obe glavne baze bile na različitim geografskim područjima i samim tim opremale zahteve različitih korisnika, konfliktne situacije između glavnih baza ne bi bile toliko česte. Deljenje baze (*Database Sharding*) ima svoju cenu, gubimo mogućnost korišćenja transakcija zavisnih od oba servera, kao i mogućnost spajanja tabela koje se

nalaze na različitim serverima. Ukoliko su nam podaci previše isprepletani, upotrebljavanje ovog pristupa nije baš najbolja ideja.

8. Dizajn arhitekture