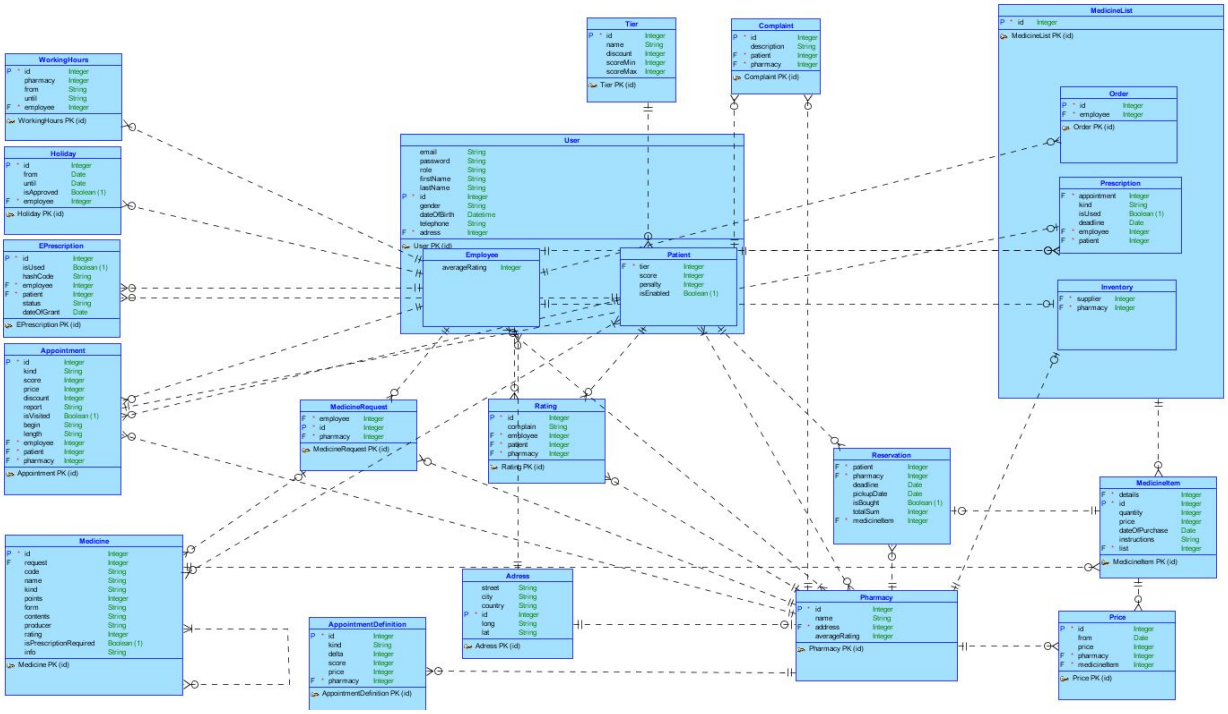
PROOF OF CONCEPT

Kako broj korisnika (200 miliona) i zahteva za rezervaciju lekova i zakazivanja pregleda (milion mesecno) prevazilazi kapacitete jednog servera oslonicemo se na *horizontalno skaliranja* odnosno formiracemo *klastere* servera i uvesti *load balancer*. Glavni ciljevi su nam da izbegnemo single point of failure arhitekturu kao i da efikasnije opsluzimo zahteve korisnika.

Dizajn šeme baze podataka je dat na sledecem dijagramu. Korisceni su koncepti EER modela podataka.



Izostavljena su *version* polja koja se koriste za potrebe optimistickog zakljucavanja.

Predlog strategije za particionisanje podataka

Koristimo kombinaciju horizontalne i vertikalne particije podataka

1. Horizontalna particija (*sharding*)- Podaci o korisnicima (pacijentima i zaposlenima) kao i o inventarima apoteka i dobavljaca mogu, na osnovu poznatih adresa, da se rasporede na manje klastere na razlicitim geografskim lokacijama kako bi se minimizovalo vreme potrebno za obradu zahteva za ovim podacima. Svi klasteri imaju iste seme samo torke koje sadrze su “vezane” za tu geografsku oblast.

2. Vertikalna particija- Nakon sto smo rasopredili podatke na osnovu lokacije mozemo na jednoj lokaciji dalje izvrsiti particiju podataka o istim entitetima na vise servera. Ovo cinimo kako bismo minimizovali broj I/O operacija kao i fetch-ovanje nepotrebnih podataka pri istim. Tako npr. osnovni staticki podaci o lekovima (naziv, sifra, tip, proizvodjac) bi se nalazili na jednom serveru a dinamicki podaci o konkretnim inventarima, rezerva-

cijama i receptima na drugom. Slicna resenja se mogu napraviti i za podatke za razlicite tipove zaposlenih kao i za podatke vezane za termine pregleda.Ovim bi se poboljsale performanse za ceste operacije kao sto je pregled lekova jer ne bismo morali da dobavljamo suvisne podatke o kolicinama leka u razlicitim apotekama, rezervacijama itd. Takodje smanjujemo broj konkurentnih pristupa bazi i imamo dodatnu mogucnost formiranja posebnih particija za osetljive podatke kao sto su istorija pregleda pacijenata.

Dodatno, ukoliko se vremenom u nekoj od particija dobijenih vertikalnim particionisanjem primeti jasna podela unutar podataka potrebnih za razlicite funkcionalnosti moze se uvesti i dalje funkcionalno particionisanje podataka. Ovo bi se vrsilo tek nakon analiziranja kako “prosecan korisnik” koristi sistem.

Predlog strategije za replikaciju baze i obezbeđivanje otpornosti na greške.

Strategija- *Partial replication*- Svaki server baze ce imati po jedan backup server na istoj lokaciji. Izbor ove strategije je direktna posledica ranije horizontane particije baze.

Metode za replikaciju podataka:

1. Za podatke koji se retko menjaju - *Full table replication*
   Za prethodno pomenute servere, dobijene vertiklanom particijom baze, ukoliko pretezno sadrze staticke/osetljive podatke backup serveru cemo slati sve izmene cim nastanu. Zbog kolicine saobracaja koju ova metoda stvara ne koristimo je u ostatku sistema.

2. Za podatke koji se cesce menjaju - *Key-based incremental replication*
   Backup server baze vrsi povremeno azuriranje podataka tj. uvek sadrzi podatke koji su konzistentni sa poslednjim updatom. Kako se operacije kao sto su rezervisanje lekova i zakazivanje pregleda sastoje iz niza podkoraka koji mogu proizvesti vise uzastopnih azuriranja efikasnije je vrsiti njihovo povremeno repliciranje na backup server. Ovim se smanjuje kolicinu saobracaja i procesorsko vreme potrebno za repliciranje podataka.

Predlog strategije za keširanje podataka

Read-only cache-iskljucivo za staticke podatke kao sto je email korisnika

Nonrestricted Read-Write cache- za podatke koji se najcesce vise puta traze i time stvaraju opterecenje kada je u pitanju broj identicnih SQL upita ka bazi-to su liste lekova/ apoteka/ farmaceuta/ dermatologa sortiranih po oceni/nazivu itd. Kako su ove liste iste za sve korisnike (za razliku od npr liste najblizih apoteka) kesiramo prvih N rezultata (npr. 10) za svaku od njih. Dodatni razlog za kesiranje ovih listi na ovaj nacin je taj sto se one relativno sporo menjaju u odnosu na npr. liste termina a ukoliko nisu azurne ne stvaraju kriticne situacije.

Transactional cache- Za liste slobodnih (predefinisanih) termina i inventare apoteka. Koristi se ovaj tip kesa kako bi izbegli situacije gde korisnici pokusavaju da zakazu vec zauzete termine ili rezervisu nepostojece lekove.

Okvirna procena za hardverske resurse potrebne za skladištenje svih podataka u narednih 5 godina
Na osnovu broja korisnika kao i broja zakazivanja pregleda/rezervacije leka potrebno je cuvati:

- 200 miliona zapisa o korisnicima
- 60 miliona zapisa o rezervaciji leka- rezervacije se cuvaju i nakon sto su ispunjene- za potrebe izvestaja
- 60 miliona zapisa o terminima- termini se cuvaju nakon sto su odrzani- istorija pregleda/ savetovanja

Kako koristimo Postgres bazu integer vrednosti zauzimaju po 4 bajta, Date po 8, Boolean 1 a ako pretpostavimo da ce u proseku stringovi biti duzine 20 karaktera (20 bajta) mozemo doci do procene o memoriji koju zauzimaju pojedinacni zapisi.

Podaci o korisniku-221 bajt po korisniku	Ukupno 221*2*10^8=4,42*10^10 bajta
Rezervacija leka-41 bajt po rezervaciji	Ukupno 41*6*10^7=2,46*10^9 bajta
Zakazani termin-125 bajta po terminu	Ukupno 125*6*10^7=7,5*10^9 bajta

Za reference ka drugim entitetima gde je veza bila OneToMany ili ManyToMany pretpostavili smo da ce u proseku postojati po 2 zapisa sa vezanim entitetom sto se u bazi svodi na po 2 integer vrednosti u medjutabeli ili kao strani kljuc u povezanom entitetu.
Kako vrsimo replikaciju svakog zapisa na tacno jedan backup server potrebna kolicina memorije se mnozi sa 2.

2*(44,2*10^9+2,46*10^9+7,5*10^9)=108,32*10^9

Ukupna kolicina memorije potrebna za skladistenje podataka iznosi problizno 101GB.

Predlog strategije za postavljanje load balansera

Inicijalne zahteve klijenata *load balancer* rasporedjuje po *Weighted Least Connection* algoritmu odnosno nova sesija se uspostavlja sa serverom koji trenutno ima najmanje aktivnih konekcija pri cemu mi mozemo zadati tezine (verovatnoce izbora koje se mnoze sa brojem konekcija) za svaki server u klasteru. Glavna prednost ovog algoritam je ravnomerna raspodela zahteva a nudi i dodatnu fleksibilnost kroz mogucnost menjanja tezina po potrebi.

Resenje za otkazivanje servera u klasteru- Svaka sesija se nalazi na dva servera (primarni i backup). Backup server nije fiksan, vazno nam je samo da se svaka sesija nalazi na jos jednom serveru. Load balancer/dispatcher pamti za svaku sesiju koji je sekundarni server i preusmera-va zahtev na njega ukoliko dodje do otkazivanja primarnog. Ovako, intenzitet saobracaja izmedju servera se ne menja sa povecanjem broja servera i ne postoje posebni cvorovi cije otkazivanje bi dovelo do drasticnog pada u performansama/ pouzdanosti sistema.

Predlog koje operacije korisnika treba nadgledati u cilju poboljšanja sistema

Trenutno, najkriticnije operacije u pogledu broja zahteva koji stizu od korisnika su ***zakazivanje termina*** (pregleda/savetovanja) kao i ***rezervacija leka*** (ukoliko dozvoljavamo istovremenu rezervaciju vise lekova).

Analiziranjem kriterijuma po kom prosecan korisnik donosi konacnu odluku kao i broja koraka (“premisljanja”) u okviru ovih aktivnosti mozemo zakljuciti koje podatke mozemo kesirati kako bi smanjili broj pristupa bazi. Npr. ako bismo vremenom primetili jasnu korelaciju izmedju lekova koji se uzimaju zajedno mogli bismo prilikom zahteva o podacima o jednom leku da serveru za kesiranje odmah prosledimo i podatke o drugom leku koji ce korisnik zeleti da vidi nakon njega. Slicno ukoliko su dve apoteke cesto gledane kao alternative prilikom zakazivanja pregleda mogli bismo to da iskoristimo na identican nacin cime bismo dodatno smanjili pristupe bazi.

Dodatno, kako su particije podataka bazirane i na pretpostavkama o ovim aktivnosima posmatranje korisnika je nuzno kako bismo odredili da li su predlozene particije podataka adekvatne. Takodje, moze nam dati uvid u to gde mozemo uvesti funkcionalne particije.

Kompletan crtež dizajna predložene arhitekture