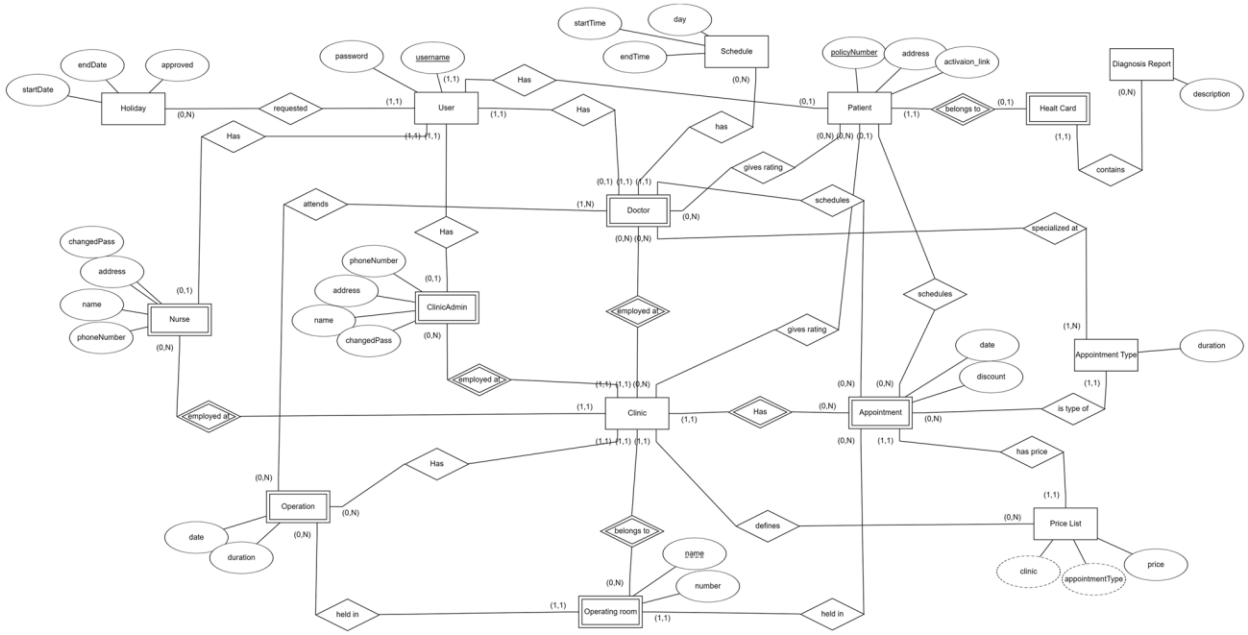
Proof Of Concept za visoku skalabilnost aplikacije Clinical Center

Šema baze podataka



Strategija za particionisanje podataka

Glavna akcija u aplikaciji je zakazivanje pregleda. Samim tim najbolji skok u performansama bismo vidjeli pri particionisanju tabele za preglede i operacije. Baza koju smo koristili u deploymentu je PostgresQL. Ova SQL baza podržava tri tipa deklarativnog particionisanja : Range (na osnovu broja ili datuma), List (grupe koje mi definišemo na osnovu vrijednosti nekog atributa), i hash.

1.Tabele Appointment i Operation imaju u sebi atribut date. Kako je u našoj aplikaciji, pri provjeri slobodnih termina za pregled, upravo ovo najskuplji zahtjev, prvo bismo odradili horizontalno Range particionisanje na osnovu polja date. Dalje particionisanje bi možda moglo biti odrađeno na osnovu polja doctor.

2.Po trenutnoj implementaciji aplikacije, pri slanju zahtjeva za pregled od strane pacijenta, prvo se provjerava tabela Doctors. Nad ovom tabelom se rade JOINovi sa drugim relevantim

tabelama za upit. Tako da smatramo da bi prioritet bio da se particioniše ova tabela, i to po atributu specialization koji nam govori za koje preglede je doktor specijalizovan.

3. Pod pretpostavkom da aplikacija ima 200 miliona korisnika, većina će biti pacijenti. Ovdje bi nam najveći dio zahtjeva bio za dobijanjem JSON web tokena na osnovu username i passworda. Za tabelu Patient bismo koristili hash particionisanje, i to nad atributom username.

4. Istu strategiju kao kod particionisanja tabele Patient bismo mogli iskoristiti kod tabela ClinicAdmin i Nurse, za admine klinike i medicinske sestre.

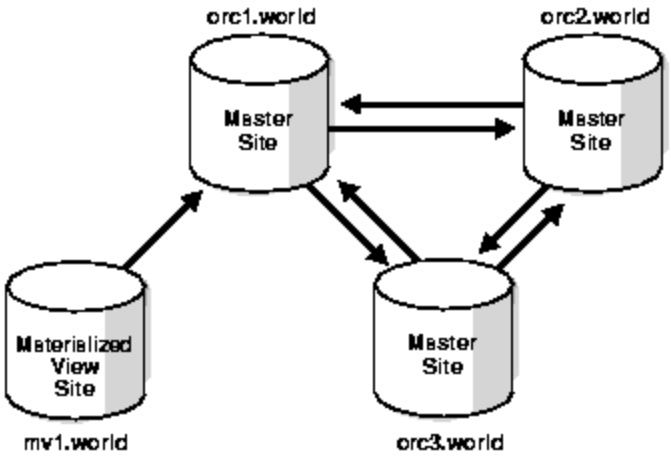
5. Tabelu koja sadrži odmore radnika bismo takođe particionisali na Range način. Konkretno na osnovu atributa za početak odmora. Kako nam uglavnom stariji podaci od prošlih godina nisu bitni, na ovaj način bismo dobili bolje performanse.

6. Tabela koja sadrži radno vrijeme za svakog radnika je takođe pogodna za particionisanje. Kako imamo polje day, koja predstavlja dan u nedelju za koji vezujemo radno vrijeme, a tipa je integer, uradićemo Range particionisanje.

7. Poslednje dvije tabele koje nismo nabrojili, su identične. Naime imamo tabelu ClinicRating i DoctorRating. Kako je broj korisnika 200 miliona, možemo očekivati ogroman broj ocjena. Obije tabele sadrže polje pacijenta i ocjenu, i njih bismo particionisali na osnovu trećeg preostalog polja, clinic i doctor.

Strategija za replikaciju baze i obezbjeđivanje otpronosti na greške

Naša aplikacija pod pretpostavkom da ima 200 miliona korisnika, i milion zakazanih operacija i pregleda mjesečno, sadrži bitne informacije. Samim tim moramo obezbijediti jako dobru otpornost na greške i imati replicirane podatke. Brzina upisa write operacija u našu bazu podataka nam nije od presudne važnosti, tako da predlažemo jednu vrstu Master Master sistema. Tačnije kako bismo izbjegli Split Brain, uveli bismo više glavnih čvorova koji međusobno komuniciraju i updateuju jedni druge. Pored ovoga svaki od tih glavnih čvorova bi imao jednog ili više Slave čvorova sa kojima je povezan i na koje bi se čuvali podaci. U slučaju otkaza glavnog čvora, imali bismo računar koordinator koji bi neki od Slave računara postavio kao glavnog na mjesto Mastera koji je prestao sa radom.



Strategija za keširanje podataka

Za keširanje podataka bismo koristili Redis. Za ovaj sistem keširanja smo se odlučili i zbog asinhronog izvršavanja zadataka. To izvršavanje bi implementirano bilo pomoću Celery biblioteke, dok bi Redis bio korišćen kao task queue. Podaci koji se rijetko mijenjaju a pristupa im se na svakom refreshu stranica su tipovi pregleda, i njih bi čuvali u keš memoriji. Kako su unaprijed definisani pregledi usko vezani za kliniku, pacijent mora prvo otići na profil klinike. Tako da bismo keširali prvo klinike, potencijalno sa svojim unaprijed definisanim pregledima. To zavisi isključivo od ponašanja korisnika i koji procenat definisanih pregleda u stvari predstavljaju unaprijed definisani termini. Pored ovoga smatramo da bi trebali keširati istoriju pregleda i operacija pacijenta, s obzirom da se ovi podaci ne mijenjaju tako često, a pacijent će vjerovatno često pristupati.

Okvirna procjena hardverskih resursa neophodnih za skladištenje podataka u narednih 5 godina

Najveći broj korisnika nam predstavljaju pacijenti. U profilu pacijenta ne postoji slika, niti bilo gdje postoje multimedijalni fajlovi. Svi podaci koje pacijent može da generiše su ocjene za klinike i doktore, i to isključivo s kojim je imao veze tokom medicinske istorije, i zahtjevi za zakazivanje pregleda. Imamo pretpostavku da imamo mjesečno milion zahtjeva za pregled ili operaciju. U tabeli Appointment imamo 2 *date* polja i jedan *time*, u PostgresQL dokumentaciji je navedeno da zauzimaju po 4 bajta, a polje *time* 8 bajtova. Imamo 4 polja za strane ključeve koji su predstavljeni brojem, to je još 16 bajtova. I imamo 2 strana ključa koji su predstavljeni email adresama. Pretpostavimo da je email adresa u prosjeku dužine 20 karaktera, za njih imamo 2 puta po 2*20 bajtova, a to je 80. Za jednu instancu modela Appointment smo izračunali da nam treba oko 108 bajtova. Za 5 godina dobijamo oko 7GB podataka za preglede. Najveći „potrošač“ memorije nam je zapravo tabela sa korisnicima. Imamo ih oko 200 miliona, a sad ćemo uraditi račun za pacijenta. Većina polja čuvaju stringove i pretpostavićemo da je za svaki

karakter potrebno 2 bajta za čuvanje. Otprilike za jednog pacijenta nam je onda potrebno recimo 200 bajtova. Dobijamo broj da nam je za čuvanje 200 miliona takvih naloga potrebno oko 40GB podataka.

Recimo da sveukupni troškovi u pogledu memorije ne prelaze 50GB.

Strategija za postavljanje Load Balancera

Kako smo implementirali sistem za čuvanje podataka sa više MASTER stanica, trebala bi nam dva load balancera. Jedna koji bi bio usmjeren ka aplikativnim serverima od klijenta, a drugi ka serverima baze. Jednostavniji vid implementacije load balancera bi nam bio Round Robin, koji bi kružno dodjeljivao zahtjeve pristigle od klijenata. Međutim malo kompleksniji, ali potencijalno bolji način bi bio load balancer na osnovu IP adrese korisnika. Ukoliko se ispostavi da nam je geografska distribuiranost korisnika povoljna za implementaciju podsistema koji bi ih opsluživao (aplikativni i bazni serveri), ovakvi load balanceri bi bili potencijalno bolji.

Koje operacije korisnika nadgledati u cilju poboljšanja sistema

Kao što je navedeno već ranije, najčešće korišćena akcija će biti rezervacija pregleda, kao i provjera slobodnih termina. S toga ćemo taksativno navesti operacije koje bi trebalo nadgledati:

-[PACIJENT] : zakazivanje unaprijed definisanog pregleda. Ovu operaciju smatramo da bi trebalo nadgledati, da utvrdimo koliko često se koristi u odnosu na ručno zakazivanje od strane pacijenta. Ako se ne koristi i ne mijenja toliko često, ovi podaci bi bili kandidat za keširanje

-[PACIJENT]: pristup istoriji pregleda i operacija. Isti razlog kao i prethodne operacije.

-[PACIJENT]: ocjenjivanje doktora i klinika. U trenutnoj implementaciji aplikacije, korisnik lako može da mijenja ocjenu koju je dao klinici ili doktoru. Ako bismo vidjeli da se ova opcija zloupotrebljava i šalje se veliki broj zahtjeva, morala bi se rekonfigurisati.

-[DOKTOR]: zakazivanje sledećeg pregleda ili operacije za korisnika.

-[ADMIN KLINIKE]: definisanje slobodnih termina.

-[ADMIN KLINIKE]: dodjela sale.

Gore navedene operacije su potencijalno implementirane pomoću kompleksnijih upita ka bazi podataka. U slučaju velikog broja korišćenja ovih operacija, neke bi se trebale refaktorisati.

Dizajn arhitekture