Proof Of Concept

1) Dizajn šeme baze podataka

Nalazi se u fajlu ClinicalCentre.oom.

2) Particionisanje podataka

2.1) Logovanje u aplikaciju

Pretpostavljeni broj registrovanih korisnika aplikacije je 200 miliona. Od tog broja većina korisnika su pacijenti. Pacijent koristi aplikaciju tako što svaki put kada želi da pristupi nekoj funkcionalnosti aplikacije se loguje, uradi tu funkcionalnost i na kraju se izloguje. Zbog velikog broja logovanja pacijenata predlažemo horizontalno particionisanje tabele PATIENT po heš vrednosti kolone security_number. Predložen je security_number jer bi particionisanje na osnovu početnih slova imena ili prezimena dovelo do neravnomerne raspodele pacijenata po particijama zbog veće prisutnosti nekih početnih slova. Iako je distribucija po security_number ravnomernija, predlažemo njegovo heširanje da bi se osigurao balans medju particijama. Pored horizontalnog particionisanja tabele patient predložili bismo i vertikalno particionisanje kolona email i password.

2.2) Tabela CLINIC

Pretpostavljamo da broj klinika u našem kliničkom centru ne bi bio velik pa predlažemo samo vertikalno particionisanje. Razdvajanje tabele na kolone name, address, city, country, description, rating(relativno statični podaci) i ostale kolone.

2.3) Tabela CLINIC_ADMIN

Predlažemo vertikalno particionisanje na kolone email, name, surname, gender, date_of_birth, address, city, country, phone_number, security_number, clinic_id(relativno statične podatke) i ostale kolone.

2.4) Tabela CLINICAL_CENTRE_ADMIN

Predlažemo vertikalno particionisanje na kolone email, name, surname, gender, date_of_birth, address, city, country, phone_number, security_number (relativno statične podatke) i ostale kolone.

2.5) Tabela DOCTOR

Za ovu tabelu predlažemo horizontalno particionisanje po vrednosti kolone clinic_id. Takođe vertikalno particionisanje na kolone email, name, surname, gender, specialization, start_work, end_work, phone_number, address, country, city (relativno statične podatke) i ostale kolone.

2.6) Tabela ORDINATION

Za tabelu ordination predlažemo horizontalno particionisanje na osnovu vrednosti kolone clinic_id. Pored horizontalnog particionisanja može da se uradi i vertikalno particionisanje na kolone name, ordinaton_type i ordination_number (relativno statične podatke) i ostale kolone.

2.7) Tabela APPOINTMENT_TYPE

Particionisanje koje predlažemo za tabelu appointment_type je horizontalno particionisanje po vrednosti kolone clinic_id. Takođe predlažemo i vertikalno particionisanje na kolone name, duration, price (relativno statične podatke) i ostale kolone.

2.8) Tabela APPOINTMENT_REQUEST

Pretpostavljeni broj korisnika aplikacije je 200 miliona, od čega su većina korisnika pacijenti. Na osnovu toga možemo da pretpostavimo da će broj pristiglih zahteva za pregled biti velik. Zbog toga predlažemo horizotalno particionisanje tabele appointment_request na osnovu vrednosti kolone clinic_id. Zatim predlažemo horizontalno particionisanje na osnovu vrednosti kolone approved.

2.9) Tabela LEAVE_REQUEST

Predlažemo horizontalno particionisanje prema vrednosti kolone clinic_id i zatim horizontalno particionisanje prema vrednosti kolone approved.

2.10) Tabela MEDICAL_RECORDS

Broj entiteta u tabeli medical_records je direktno povezan sa brojem pacijenata u aplikaciji pa predlažemo horizontalno particionisanje na osnovu raspona vrednosti kolone patient_id.

2.11) Tabela APPOINTMENT

Uradili bismo horizontalno particionisanje na osnovu vrednosti kolone clinic_id.

2.12) Tabela MEDICAL_REPORT

Predlažemo vertikalno particionisanje na osnovu vrednosti kolone verified.

2.13) Tabela RATING

Broj entiteta u tabeli rating može maksimalno da bude broj registrovanih korisnika puta broj doktora plus broj klinika u klinickom centru, što je ukoliko imamo 200 miliona korisnika aplikacije veoma velik broj. Zato predlažemo horizontalno particionisanje na osnovu vrednosti kolone clinic_id i zatim horizontalno particionisanje na osnovu vrednosti raspona patient_id.

2.14) Sve tabele

Pošto se u aplikaciji za brisanje podataka koristi logičko brisanje predlažemo horizontalno particionisanje svih tabela na osnovu vrednosti kolone active, i zatim ostala particionisanja koja su gore predložena da se vrše na tabeli čija je vrednost kolone active “true”.

3) Replikacija baze i otpornost na greške

Predlažemo jednu glavnu bazu podataka u kojoj će se nalaziti svi podaci aplikacije. Zatim distribuciju te glavne baze podataka na više manjih baza od kojih će svaka pripadati jednoj klinici i sadržati podatke vezane za tu kliniku. Baze podataka od klinika treba da se nalaze geografski blizu tih klinika. Takođe potrebna je backup baza podataka koja će se nalaziti na nekoj udaljenoj geografskoj lokaciji dobro zaštićena od mogućih fizičkih oštećenja.

4) Keširanje podataka

Keširali bismo često korišćene podatke aplikacije. Pre pristupa bazi podataka prvo bi se proveralo da li traženi objekat postoji u kešu, ukoliko ne postoji tek onda bi se pristupalo bazi podatka. Za svrhu čuvanja celih objekata predlažemo korišćenje Memcached ili Redis. Predlažemo posmatranje rada aplikacije neko vreme i zatim keširanje klinika sa najviše poseta. Takođe, predlažemo keširanje predefinisanih pregleda klinika jer će pacijenti verovatno pri korišćenju aplikacije najvise pristupati njima. Pored pristupa predefinisanim pregledima pacijent će često pristupati svom medicinskom kartonu pa predlažemo i njegovo keširanje. Strategiju zamene keša koju predlažemo je Last Recently Used (LRU).

Strategija LRU pri zameni podataka iz keša zamenjuje podatke koji najduže nisu korišćeni novim podacima.

5) Procena hardverskih resursa u narednih 5 godina

Pošto u našoj aplikaciji čuvamo samo tipove podataka koji zauzimaju malo memorijskog prostora: String, int, double i boolean, a ne čuvamo memorijski zahtevne podatke kao što su veliki binarni fajlovi ili fotografije ili video zapisi naša procena je da će se dnevno u bazu unositi 100KB podataka po korisniku. To znači da bi za 5 godina bilo potrebno obezbediti skladištenje oko 34.000TB podataka.

6) Postavljanje load balansera

Predlažemo postavljanje load balansera između klijenata i aplikativnih servera. Algoritam load balansera za raspodelu zahteva koji predlažemo je Least Connections algoritam. Po načinu korišćenja aplikacije korisnike možemo podeliti u dve grupe. Pacijente, koji koriste aplikaciju tako što se uloguju, kratko se zadrže u njoj i zatim se izlogoju. Ostali korisnici aplikacije (doktori, medicinske sestre, admini klinike i admini kliničkog centra) se ulogoju u aplikaciju na početku randog vremena, i izloguju na kraju radnog vremena. Zbog takvog načina rada aplikacije predlažemo Least connections algoritam koji raspoređuje zahteve ka aplikaciji tako što pri svakom zahtevu proveri koji server u tom trenutku ima najmanje konekcija i njemu prosledi zahtev. Ukoliko uzimamo i kapacitete servera u obzir može se koristiti Weighted Least Connections algoritam. Jačim serverima se dodeljuje veći prioritet dok se slabijim dodeljuje manji. Ukoliko dva servera imaju isti broj konekcija u nekom trenutku zahtev će biti prosleđen onom serveru koji ima veći prioritet.

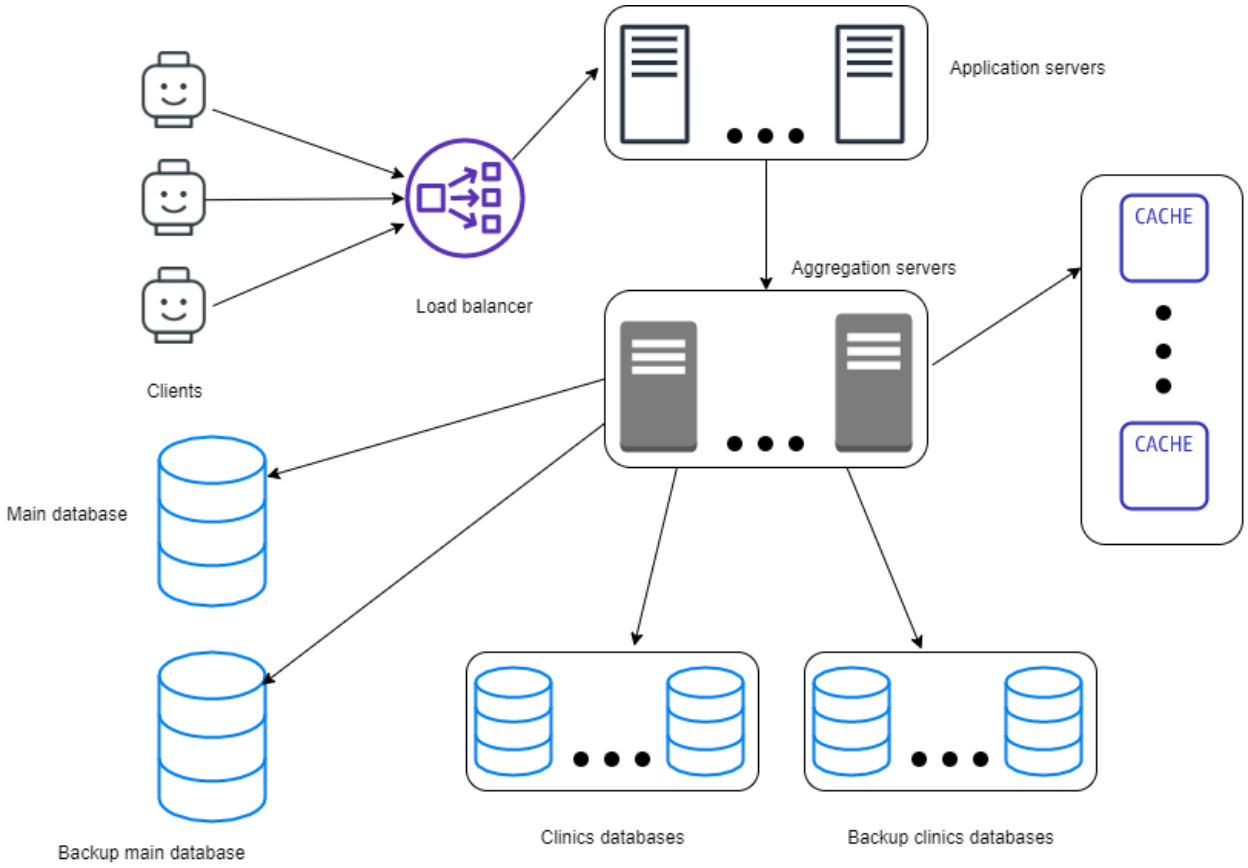
7) Operacije korisnika koje treba nadgledati u cilju poboljšanja sistema

Operacije korisnika koje treba nadgledati su:

- Broj pristiglih zahteva za novi pregled/operaciju
- Broj pristigih zahteva za registraciju
- Broj novoregistrovanih pacijenata
- Broj zakazivanja predefinisanog pregleda

Sve operacije treba posmatrati za svaki sat i na dnevnom nivou kroz period od nekoliko nedelja.

8) Dizajn predložene arhitekture



9) Dodatna unapređenja

Filtriranje podataka

Trenutno se filtriranje svih podataka radi direktno na frontu, nakon što se sa servera dobave podaci. Prvobitno smo se odlučili za ovaj pristup iz razloga male količine podataka. Pošto sada razmatramo dosta veću količinu podataka sa kojom je neophodno raditi, smatramo potrebnim da se uvede filtriranje na back-endu. Na ovaj način se korisniku neće vraćati svi entiteti i očekivati od njegove korisničke mašine da uspešno i brzo odradi filtriranje, već će se to raditi na serverima koji imaju znatno jače komponente.

Paging mehanizam tabela

Trenutno se sadržaj svih tabela dobavlja jednim pozivom na server. Za ovaj pristup smo se odlučili iz istog razloga kao i u prethodno navedenom slučaju, mala količina podataka. Takođe, iz istog razloga predlažemo uvođenje pagin mehanizma pri dobavljanju podataka iz baze, gde se ne bi dobavljali svi podaci nego određen manji broj. Uvođenjem ovog unapređenja doprineli bismo smanjenju uskog grla koje predstavlja transport podataka do korisnika i omogućili dosta brži rad same aplikacije.

Spring security

Trenutni pristup proveravanju tipa trenutno ulogovanog korisnika i njegovih kredencijala zahteva da aplikacija ne bude u potpunosti RESTful (postojanje sesije na koju se korisnik dodaje). Postojanje sesije otežava klasterovanje na servere i, time, smanjuje mogućnost optimizacije našeg rešenja. Predlažemo da se uvede Spring security koji bi uklonio potrebu za sesijom.