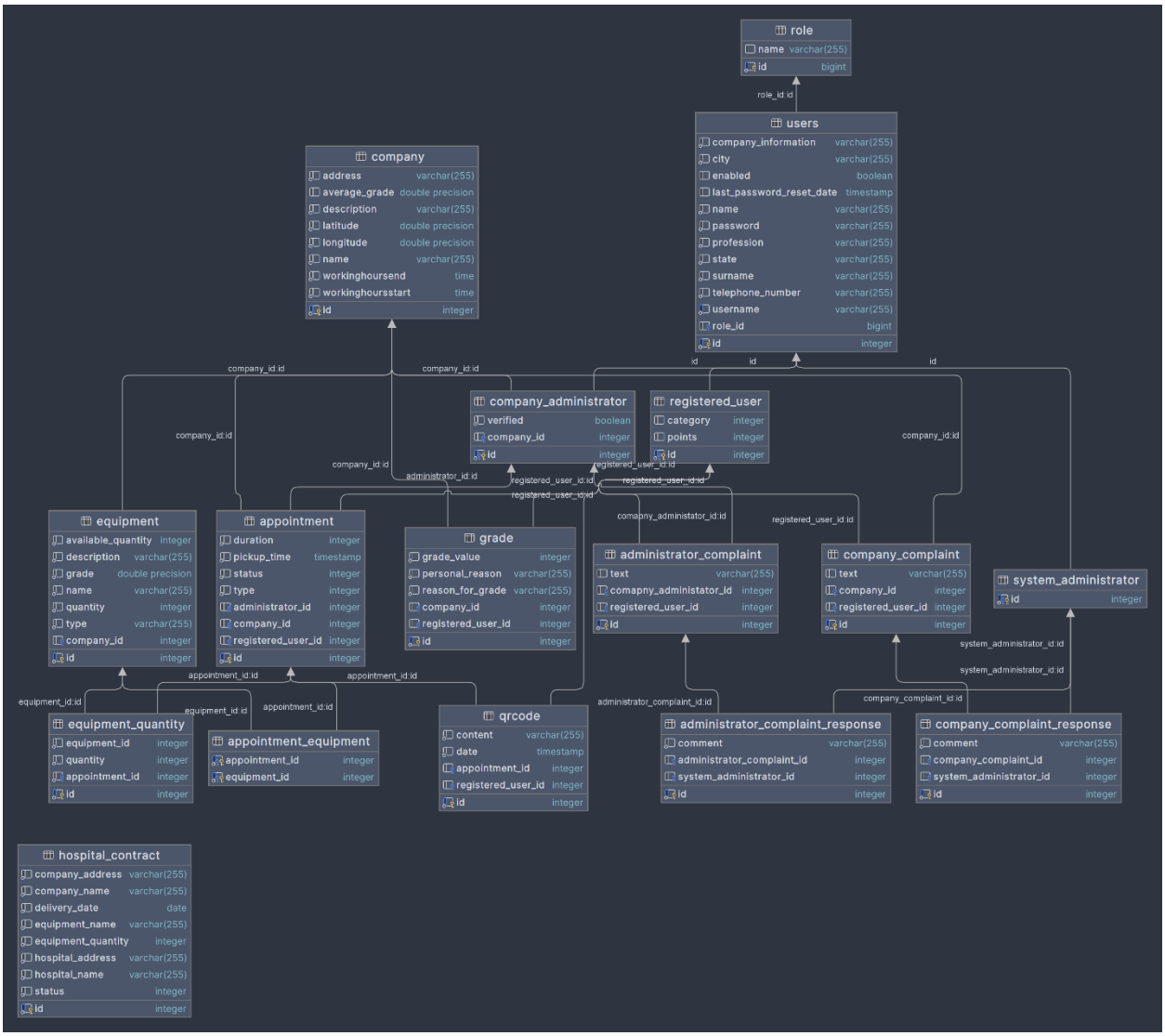
Proof of concept

ISA - TEAM 71 2023/2024

Dizajn šeme baze podataka – Klasni dijagram



Predlog strategije za particionisanje podataka

Particionisanje tabele može značajno poboljšati performanse i omogućiti lakše upravljanje velikim setovima podataka. Neke od mogućih strategija pristupa particionisanju koje bismo primenili na naš sistem:

1. Particionisanje po datumu
    - Tabela rezervacija može se particionisati po datumu rezervacije.
    - Na primer, svaki mesec može imati svoju particiju.
    - Ovo omogućava efikasnije upravljanje podacima i brže upite koji zahtevaju podatke za određeni vremenski period.
    - Povećana efikasnost ispisa radnog kalendara
2. Horizontalno particionisanje po geografskoj lokaciji
    - Ukoliko bi naša aplikacija imala globalno prisustvo, mogli bi razmisliti o horizontalnom particionisanju po geografskoj lokaciji.
    - Tabela rezervacija može se podeliti na particije koje pokrivaju različite regione ili države.
    - Ovo omogućava distribuirano čuvanje podataka i poboljšava odziv za korisnike širom sveta

Predlog strategije za replikaciju baze i obezbeđivanje otpornosti na greške

PostgreSQL je tradicionalno dizajniran kao jedna instanca sistema za upravljanje bazama podataka (DBMS), što znači da postoji samo jedan server koji rukuje svim upitima i transakcijama.

Ovo može biti ograničavajuće u smislu horizontalnog skaliranja jer dodavanje resursa (kao što su CPU i RAM) na jednu instancu može imati svoje granice. Vertikalno skaliranje, tj. dodavanje više resursa jednom serveru, može biti skupo i može dostići svoje granice.

Problem single instance može se rešiti korišćenjem replikacije baze podataka. Replikacija se koristi kako bi se omogućila distribucija podataka i opterećenja između više instanci baze podataka.

Naš izbor strategije je Read Replication koja podrazumeva postojanje jednog glavnog ("master") servera koji prima upite za pisanje, dok se replike koriste za čitanje podataka. To pomaže u smanjenju opterećenja na glavnom serveru jer replike mogu rukovati zahtevima za čitanje.

Ova vrsta replikacije koristi se kada su potrebne visoke performanse čitanja, ali može doći do određenog vremenskog zakašnjenja između ažuriranja na master čvoru i čitanja sa replika.

Distribuirano čitanje sa replika može smanjiti opterećenje na glavnom (master) čvoru, što rezultira bržim odgovorima na čitanje upita.

Obezbeđivanje otpornosti na greške prilikom replikacije u PostgreSQL uključuje niz praksi i strategija kako bismo se nosili sa mogućim problemima i osigurali da sistem ostane pouzdan i dostupan. Naš

predlog je postojanje više replika kako bismo obezbedili redundantnost. Ako jedna replika doživi grešku, ostale replike i dalje mogu pružati usluge. Ovo se odnosi i na master i na replike. Dodatni mehanizam otpornosti na greške omogućava da se od replika zatraži povratna informacija o uspešnosti ažuriranja. Nakon toga, operacija propagacije može biti ponovljena u slučaju neuspeha kako bi se osiguralo ispravno ažuriranje sistema.

Uvođenje rate limiter-a bi trebalo biti deo strategije za replikaciju baze. Korišćenjem rate limiter-a možemo kontrolisati brzinu ažuriranja i čitanja podataka iz replika, čime se obezbeđuje ravnoteža opterećenja i smanjuje rizik od preopterećenja sistema. Ovo takođe pomaže u sprečavanju prevelikog broja zahteva koji bi mogli dovesti do gubitka performansi ili otkaza u radu, doprinoseći tako opštoj otpornosti na greške.

Predlog strategije za keširanje podataka

Level 1 keširanje - L1

Ovaj nivo keširanja je podržan od strane Hibernate-a, nije potrebna nikakva dodatna konfiguracija i ne može se isključiti. L1 keširanje se tiče Hibernate sesije. Kada se objekat učita u sesiju, prilikom svakog sledećeg upita za taj isti objekat, Hibernate neće slati upit ka bazi, već će objekat dobavljati iz keša. L1 keš omogućava da, unutar sesije, zahtev za objektom iz baze uvek vraća istu instancu objekta i tako sprečava konflikte u podacima i sprečava Hibernate da učita isti objekat više puta.

Level 2 keširanje - L2

Ovaj nivo keširanja je podržan od Hibernate-a, ali je neophodan eksterni provajder. Naš izbor bi bio EhCache, jer podržava različite strategije keširanja kao što su: Read Only, Read Write, Nonrestricted Read Write i Transactional. Zbog toga predstavlja jedan od najboljih i najpopularnijih provajdera za L2 keširanje u Spring aplikacijama.

Naša odabrana sttrategija bi bila Transactional - strategija koja se koristi kod visoko konkurentnih sistema gde je ključno sprečiti da se u bazi podataka nalaze zastareli podaci.

EhCache pruža mogućnost čuvanja keširanih objekata na Java heap-u, u RAM memoriji kao i na disku

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

1.Potrebni hardverski resursi za čuvanje korisnika

Što se tiče zauzimanja memorije kod čuvanja korisnika(pod pretpostavkom da u sistemu postoji million korisnika), potreban memorijski prostor za čuvanje bi bio približno oko 24 GB, gdje je za čuvanje podataka o jendom korisniku potrebno 252B što je oko 0,241 KB.

2. potrebni hardverski resursi za čuvanje podataka o kompanijama i opremi

Podaci o pojedinačnoj kompaniji iznose 144B. Pod pretpostavkom da će naša aplikacija, za pet godina, posedovati informacije od deset hiljada kompanija, dolazimo do brojke 1,4 MB za čuvanje informacija o kompanijama.

Smatraćemo da svaka kompanija u proseku sadrži oko stotinu različitih vrsta opreme. Za čuvanje svake pojedinačne vrste opreme je potrebno 100B, što dovodi do brojke od 95.4 MB za skladištenje podataka o opremi.

Kombinacijom ova dva rezultata dobijemo konačan potreban kapacitet od 96.8 MB za čuvanje kompanija i njihove opreme

3. Potrebni hardverski resursi za čuvanje podataka o rezervacijama

Za čuvanje podataka o jednoj rezervaciji je potrebno, ukoliko pretpostavimo da u proseku jedna rezervacija u sebi sadrži pedeset različitih vrsta opreme, iznosi 56 + 40*50 B, što je 2,01 KB.

Uz pretpostavku da mešečno imamo petsto hidljada rezervacija, za pet godina dolazimo do brojke 11.5 GB.

4. Potrebni hardverski resursi za čuvanje podataka o ugovorima

Za čuvanje podataka o ugovorima, pod pretpostavkom da se svaka kompanije u proseku pravi ugovore sa dvadeset bolnica, je potrebno oko 19.1 MB jer je potrebno 100B za čuvanje jendog ugovora.

Predlog strategije za postavljanje load balansera

Najjefikasnija strategija za load balanser bi bila takozvana “sticky sessions”. Kada korisnik pristupi aplikiciji i uspešno se uloguje, load balancer će zapamtiti sa kojeg server je korisnik došao. Za svaki sledeći zahtev tog korisnika, load balacer će usmeriti zahtev na isti server. Stanje sesije se čuva na klijentskoj strain aplikacije, kroz kolačiće ili kroz JWT token.

Druga strategija, koja je dosta jednostavnija bi bila round robin koja ravnomerno raspoređuje zahteve između dostupnih servera. Svaki novi zahtev bi se usmeravao na sledeći server u listi. Može biti veoma efikasan ako su podaci na serverima ravnomerno rasporedjeni.

Koristićemo nginx-ov, round robin load balancer koji može ispuniti naše zahteve za horizontalnim skaliranjem.

Predlog koje operacije korisnika treba nadgledati u cilju poboljšanja Sistema

U cilju poboljšanja aplikacije, kao i potencijalnog proširenja postojećih funkcionalnosti, dosta su bitne žalbe korisnika. U okviru žalbi korisnici prijavljuju greške u funkcionisanju softvera ali često i nedostatke koji predstavljaju izvor informacija za razvoj novih funkcionalnosti.

Takođe, sve informacije o korisnicima, njihovim pretragama, preferencama, i trendu kompanija iz kojih naručuju opremu bi trebali da se nadgledaju kako bi se poboljšao user experience samih korisnika.

Potencijalno bismo uveli eksterni API koji ima mogućnost nadgledanja podataka. Primjer takve aplikacije je Google Analytics ili Open Web Analytics.

Kompletan crtež dizajna predložene arhitekture