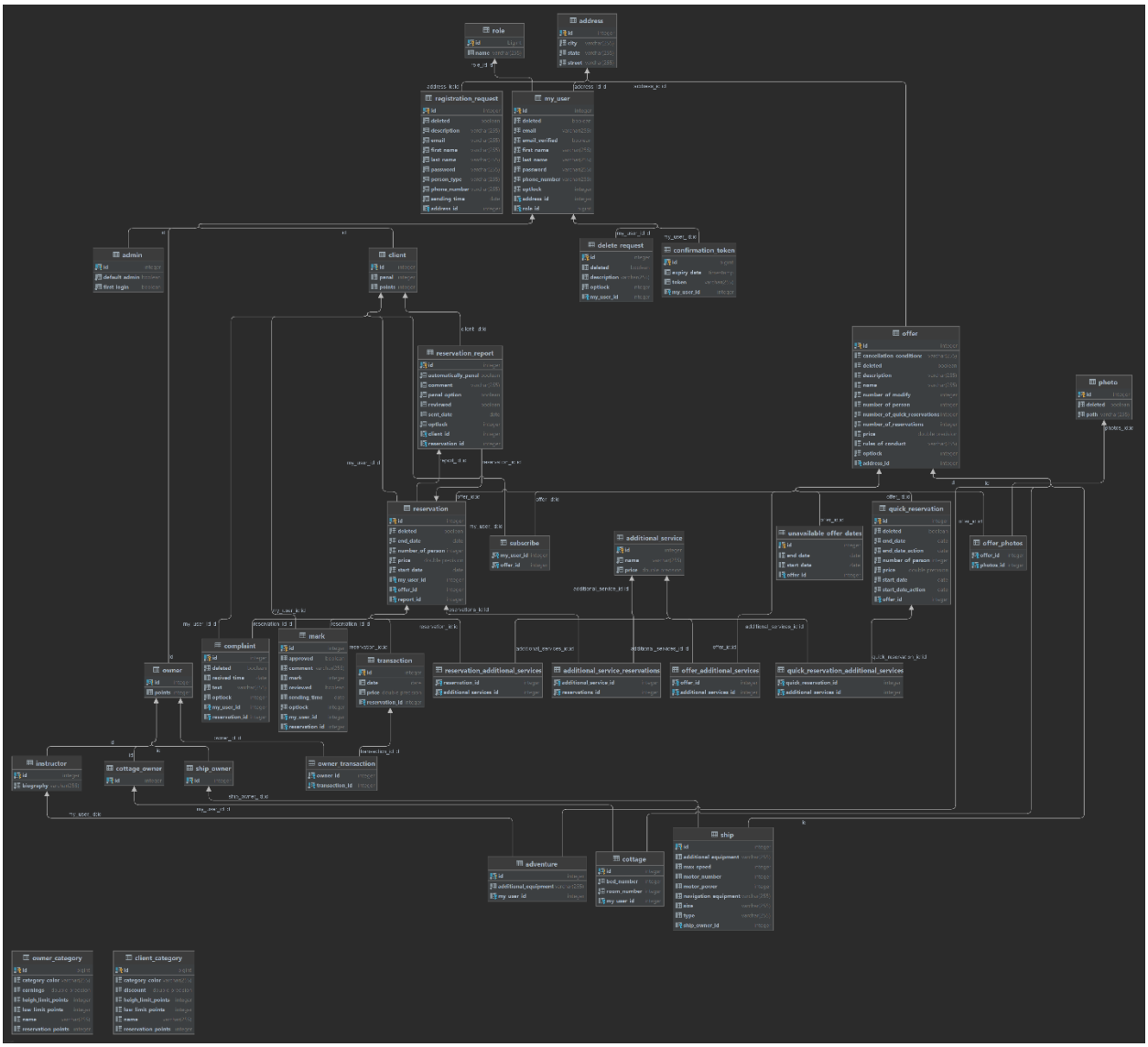
Skalabilnost

1. Šema baze podataka



2. Strategija za particionisanje podataka

S obzirom da je pretpostavljeni broj korisnika naše aplikacije oko 100 miliona, potrebno je čuvati ogromnu količinu podataka koja treba da se nalazi u bazi. Usled nagomilavanja podataka, proces čitanja, koji je procenjeno da predstavlja otprilike 95% operacija ka bazi u web aplikacijama, može značajno usporiti. Samim tim naša aplikacija će biti spora, a klijenti nezadovoljni.

Uzevši u obzir način na koji se koriste podaci iz naše baze i gledajući u model podataka sa slike došli smo do zaključka da vertikalno particionisanje ne bi imalo smisla, jer ne postoje kolone koje su nam često potrebne, a druge retko. Kada bismo naše podatke vertikalno podelili, uneli bismo samo kašnjenje koje bi proizašlo iz join operacija nad tabelama.

Stoga, bolje rešenje za našu aplikaciju bi bilo horizontalno skaliranje, gde bismo torke jedne tabele podelili u više tabele i time skratili vreme čitanja. Ulogovani klijenti kao i neautentifikovani korisnici koji će biti dominantni korisnici naše aplikacije će često vršiti operacije pregleda i pretrage ponuda, kao i pregled svojih rezervacija u slučaju klijenata. Zato su *Cottage, Ship, Instructor, Reservation, Adventure* neke od tabela u kojima bi se isplatilo uvođenje horizontalnog particionisanja.

3. Strategija za replikaciju baze i obezbeđivanje otpornosti na greške

U cilju ubrzavanja procesa čitanja/pisanja, kao i obezbeđivanja od pada baze podataka i gubitka podataka, potrebno je kreirati replike baze podataka. Ideja je da se koristi *master-slave* arhitektura. Primarna baza(*master*) bila bi ona koja preko koje će se vršiti upis, dok bi sekundarne baze(*slave*) predstavljale repliku master baze i one bi služile isključivo za čitanje. Na ovaj način kao što smo rekli najčešća operacija, čitanje, biće ubrzana jer postoji više baza koje će opsluživati milionske korisnike. Sekundarne baze registrovaće promene kada se dese u master bazi i tada će biti ažurirane, čime se neće narušiti konzistentnost.

Takođe ovom arhitekturom je povećana bezbednost podataka i dostupnost aplikacije, jer ih imamo na više mesta. Ukoliko se desi da master baza padne, neka od sekundarnih baza će se izabrati da postane master sve dok se prethodna master baza ne oporavi.

4. Strategija za keširanje podataka

Još jedno od poboljšanja aplikacije koje bi značajno došlo do izražaja pri velikom broju korisnika jeste keširanje podataka.

Strategija koju bismo koristili za keširanje u slučaju čitanja podataka je *Cache-Aside(Lazy loading).* Ovaj pristup je pogodan za našu aplikaciju s obzirom da se većina operacija zasniva na prikupljanju podataka iz baze. Takođe ovaj pristup bi efektivno

koristio keš i u njemu bi se uvek nalazilo samo ono što je korisnik zapravo zahtevao. Dodatno, pretpostavka je da bi korisnik uvek pristupao malom skupu podataka njemu od interesa te je to još jedan razlog zašto je ovaj pristup pogodan.

Uz ovu strategiju dodatno bismo koristili *Write-Around* strategiju za upis podataka direktno u bazu. Tako da bi samo ono što je pročitano išlo u keš. Ovo će prouzrokovati da podaci u kešu mogu biti nekonzistentni ukoliko dođe do njihove promene u bazi. Ovo bismo rešili time što bi praznili keš prilikom operacija koje dovode do izmene podataka kao što su na primer izmena ili brisanje ponuda itd. Smatramo da bi ovo bilo efikasnije od *TTL(time to live*) pristupa jer su ove operacije retke, samim tim će i podaci u kešu retko biti zastareli.

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

Na osnovu okvirnih računanja za najmnogobrojnije podatke iz baze dobili smo sledeće rezultate:

- Rezervacije
  - svaka rezervacija zauzima okvirno 0.05KB
  - ovaj broj pomnožen sa brojem meseci u narednih 5 godina i sa prosečnim brojem rezervacija na mesečnom nivou (1 000 000) daje **3GB** memorijskog zauzeća
- Korisnici
  - svaki korisnik zauima okvirno 0.14KB
  - ovaj broj pomnožen sa pretpostavljenim brojem korisnika (100 000 000) daje **14GB** memorijskog zauzeća
- Ponude
  - svaka ponuda zauzima okvirno 0.4KB
  - ovaj broj pomnožen sa pretpostavljenim brojem ponuda (10 000 000) daje **4GB** memorijskog zauzeća
- Ocene
  - svaka ocena zauzima okvirno 0.1KB
  - ovaj broj pomnožen sa brojem meseci u narednih 5 godina i sa pretpostavljenim brojem ocena vezanih za rezervaciju na mesečnom nivou (500 000) daje **3GB** memorijskog zauzeća
- Slike
  - svaka slika zauzima okvirno 1000KB
  - ovaj broj pomnožen sa pretpostavljenim brojem ponuda (10 000 000) i pretpostavljenim prosečnim brojem slika po ponudi (5) daje **50TB** memorijskog zauzeća

Ukupno očekivano zauzeće prethodnih entiteta je okvirno 50TB**.** S obzirom da ovde nismo uračunali sve ostale entitete iz baze na ovu procenu dodali bismo još 1TB. Tako da je naša krajnja procena memorijskog zauzeća za našu aplikaciju kroz narednih 5 godina **51TB.**

6. Strategija za postavljanje load balansera

Strategije *Round Robin* ili *Weighted Round Robin* jesu povoljne za implementaciju jer su jednostavne, ali ne bismo se odlučili za njih jer je moguće da jedni klijenti ostanu duže konektovani na jednom serveru u odnosu na druge, što bi izazvalo problem za taj server i stvorilo mogućnost da on padne.

Strategija koja bi ovaj problem rešila bila bi *Least Connections*, ova strategija gleda koji server ima trenutnno najmanje konekcija i dodeljuje novu konekciju tom serveru, jer je on najmanje opterećen.

7. Predlog koje operacije korisnika treba nadgledati u cilju poboljšanja sistema

Vrlo bitan aspekt održavanje sistema nakon puštanja istog u produkciju je njegovo nadgledanje. Teško je inicijalno dobro pretpostaviti i shvatiti buduće zahteve naših klijenata. Zbog toga sam *monitoring* nam može dati vredne informacije koje ćemo iskoristiti kako bismo unapredili naš sistem.

Ono za šta bismo se mi opredelili jeste posmatranje odnosa broja čitanja naspram broj pisanja u bazu podataka. Ukoliko je broj čitanja veći nego što je pretpostavljeno, možemo poboljšati performanse time što ćemo dodati jos sekundarnih (*slave*) baza podataka.

Još jedan parametar vredan posmatranja bio bi broj pogodaka vezanih za keš (odnosno koliko se puta ono što smo tražili u kešu zaista tamo i nalazilo). Ukoliko vidimo loše performanse, možemo izmeniti inicijalno odabranu strategiju keširanja.

Ukoliko bi vremenom nadgledanjem zaključili da je većina operacija vezana za određeni skup atributa jednog entiteta, mogli bismo uvesti i vertikalno particionisanje podataka, kako bi omogućili brži rad sa frekventno korišćenim podacima.

8. Dizajn predložene arhitekture