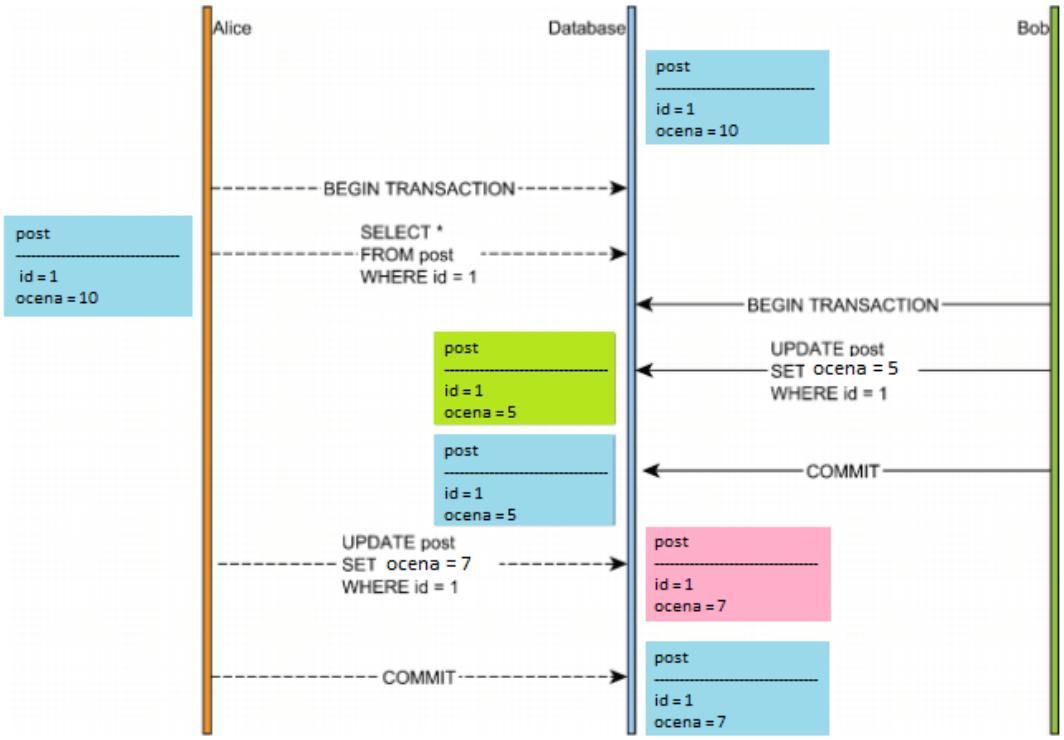
KONKURENTNI PRISTUP RESURSIMA U BAZI

KONFLIKTNE SITUACIJE I NJIHOVO REŠAVANJE

1. **opis konfliktne situacije:**

   Situacija u kojoj dva ili više pacijenta pokušavaju da pošalju upit za pregled u jednom te istom terminu kod istog doktora se smatra konfliktnom situacijom. S obzirom da jedan lekar ne može u isto vreme prisustvovati na više različitih pregleda, potrebno je izbeći pojavljivanje ovih i sličnih situacija. Prethodno navedeni problem se javlja kao *Lost update* (pokušaj menjanja podatka od strane više transakcija, bez njegovog zaključavanja).

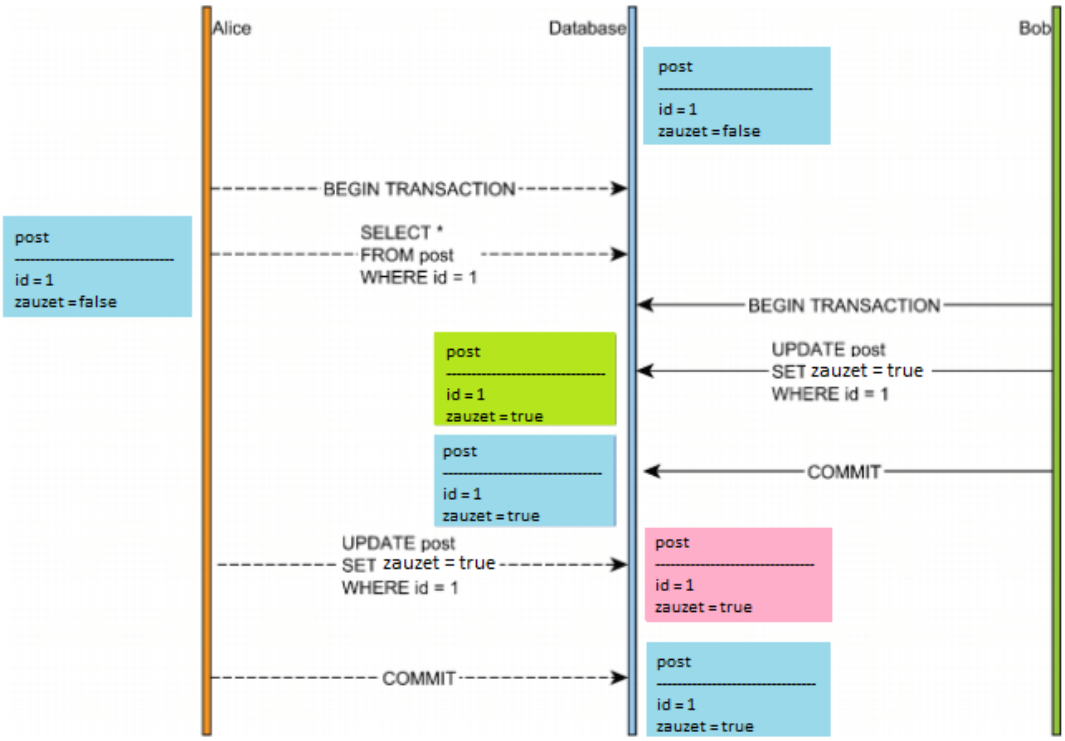


   opis načina rešavanja situacije:

   Uključivanje mehanizma optimističkog zaključavanja predstavlja rešenje za problem ove vrste. Kako se mora osigurati da podaci ostanu konzistentni između konkurentnih čitanja i menjanja podataka, optimističko zaključavanje omogućava konkurentni pristup bazi podataka na pravi način. Dodavanjem anotacije *@Version* iznad atributa klase *Lekar*, mogu se upoređivati vrednosti pročitanih podataka sa onima koji se nalaze u bazi. Konkretan podatak se ovde odnosi na listu termina pregleda koji svaki lekar poseduje i stoga je prethodno primenjeno u klasi Lekar.

2. **opis konfliktne situacije:**

Potrebno je onemogućiti zakazivanje istog termina iz liste unapred definisanih termina od strane dva ili više pacijenta. Ova konfliktna situacija se javlja usled pokušaja rezervisanje termina koji predstavlja deljeni resurs, od strane više pacijenata, bez odgovarajućeg nivoa izolacije transakcija. Kako je zakazivanje pregleda iz liste pregleda koji su već unapred definisani zamišljeno kao logičko brisanje entiteta, tako ovaj problem možemo klasifikovati pod problemom *Lost update*.

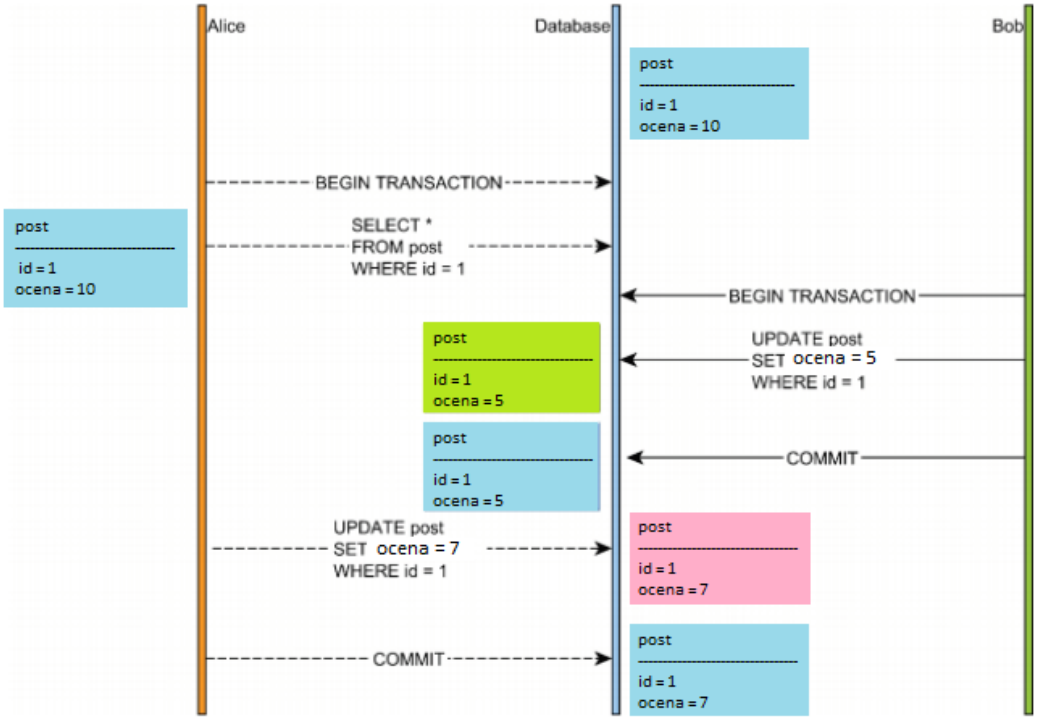


opis načina rešavanja situacije:

Kada više pacijenata pokuša da zakaže jedan te isti unapred definisani pregled, to predstavlja situaciju koja se može opisati kao konfliktna jer se pristupa deljenom resursu. Da bismo obezbedili efikasno zakazivanje pregleda, potrebno je implementirati zaključavanje deljenih podataka prilikom menjanja istih. Jedno od mogućih rešenja je optimističko zaključavanje. Ovaj pristup, kao što je pomenuto, omogućava konkurentno čitanje podataka, međutim, ukoliko je potrebna izmena istih, operacija prvog korisnika će uspeti, a ostali će proizvesti *OptimisticLockException* pokušavajući da ih promene. Jedan od načina, koji je korišćen i u ovom slučaju, podrazumeva dodavanje anotacije *@Version* iznad atrbuta će predstavljati verziju određenih podataka. U ovom slučaju klasa *SlobodanTermin* se proširuje dodavanje pomenutog atributa. Optimističko zaključavanje eliminiše i jedan drugi problem nazvan *Non-repeatable reads* (*Unrepeatable reads*) Ovaj pristup ima za prednost to što pravog zaključavanja nema, što dodatno poboljšava skalabilnost.

3. **opis konfliktne situacije:**

Pokušaj ocenjivanja jedne klinike od strane više pacijenata u isto vreme. Neophodno je omogućiti da više istovremenih korisnika ocene kliniku konkurentno, a da pritom integritet samih podataka ne bude ugrožen. Tokom ove situacije dve različite transakcije menjaju određeni podatak bez zaključavanja istog. Ukoliko se deljeni resurs ne zaključa, izmena jedne transakcije će se izgubiti usled izvršenja druge transakcije koja nije imala podatka o prethodnim izmenama. Ovaj problem se naziva *Lost update*.



opis načina rešavanja situacije:

Prilikom rešavanja ovog konflikta neophodno je zaključati deljeni resurs dok se menjaju podaci. Mehanizmi zaključavanja koji se koriste da spreče trenutnu transakciju od čitanja izmenjenih podataka ne blokiraju one transakcije koje rade na ovom nivou izolacije. Takođe, pošto je moguće čitanje modifikacija koje su načinile druge transakcije, a da još nisu *commit*-ovane, eliminisaće se gore spomenuti problem koristeći *Read uncommitted* nivo izolacije. U cilju rešavanja navedene konfliktne situacije korišćeno je optimističko zaključavanje deljenih resursa. Potrebno je da operacije pre modifikacije podataka provere da li je podatak neko drugi menjao u međuvremenu. Verzija pročitanih podataka se poredi sa verzijom podaka koji se nalaze u bazi. Ukoliko se primeni neslaganje verzija, prijavljuje se greška korisniku. Poređenje verzija je implementirano dodavanjem kolone koja će predstavljati brojač izmena. Tokom svake izmene podataka, odgovarajuća vrednost se povećava za jedan. Pored toga, definisane su granice određenih transakcija na nivou metoda i klasa uz pomoć odgovarajućih anotacija.