Rešavanje konfliktnih situacija kod definisanja loyalty programa

Autor: Nikola Dragić RA4-2017

Opis problema

Loyalty program je jedan deo našeg sistema zadužen za nagrađivanje klijenata koji su nam ukazali njihovo poverenje. Svaki pacijent ima svoj nalog koji pripada odredjenoj grupi. Svaka grupa je prethodno definisana od strane više administratora informacionog sistema i u zavisnosti od nje se određuje popust koji dajemo pacijentu za lekove, preglede i savetovanja u nasoj apoteci.

Kako su naše grupe sklone izmenama, moguće je doći u situaciju gde više administratora informacionog sistema pokuša da izmeni ili doda novu grupu i da na taj način dobijemo nevalidno stanje grupa zbog posledica konkuretnog pristupa istim resursima (pogledati Note #1 na drugoj strani).

Jedan od uslova kod izmene ili dodavanja nove grupe je da već ne postoji grupa sa istom vrednošću donje granice (minPoints u daljem tekstu) poena potrebnih da bi se loyalty program nalog ubacio u tu grupu.

Iako deluje kao trivijalna provera, kada imamo konkuretni pristup nailazimo na različite problem kao što su :

- Dva administratora u isto vreme podese minPoints na vrednost X, a pošto su oba prosla prethodnu validaciju, aplikacija će snimiti oba i onda imamo situaciju gde dve grupe imaju istu vrednost za minPoints. Postavlja se pitanje: U koju grupu ćemo staviti korisnika koji je nakupio X poena od ove dve?
- Jedan administrator želi da kreira novu grupu u koju ćemo stavljati pacijente koji sakupe X poena. Drugi administrator, u istom momentu, razmišlja kako bi bolja ideja bila da samo pomeri granicu već postojeće grupe na vrednost X. Kako, teorijski, dva administratora nisu međusobno komunicirali, šalju zahteve da se kreira grupa čija je vrednost minPoints polja X I da se polje minPoints već postojeće grupe digne na X. U ovom

slučaju bi opet dobili dve različite grupe sa istom vrednost minPoints, pa se postavlja isto pitanje kao I iz prošle stavke.

Rešenje problema

Iako je ovaj problem više teorijski, I nisu velike šanse da se dogodi u praksi, problem je se morao rešiti.

Inicijalna ideja je bila da se optimistički zaključa red koji menjamo, međutim to nije rešilo pravi problem već samo blokiralo dva administratora da u isto vreme menjaju istu grupu.

Kako nam je izuzetno bitno da podaci o grupama budu uvek validni I da potpuno eliminišemo mogućnost greške, korišćen je SERIALIZABLE tip izolacije. Iako u praksi treba izbegavati ovakav pristup kada god je moguće, u našem slučaju smo se odlučili baš za nju.

(**Note #1**: Ovaj problem je verovatno mogao biti rešen na lepši način, korišćenjem unique ograničenja, ali je izabran način o kojem bi moglo da se diskutuje, pošto su ostale potencijalne konfliktne situacije obrađene u zajedničkom delu ove tačke projektne specifikacije.)

Neki od razloga zašto smo doneli takav izbor:

- Same loyalty program grupe će se u praksi vrlo retko menjati pa samim tim "zaključavanje" ovih podataka neće uticati na performanse naše aplikacije.
- Postavili smo sebi pitanje: "Prilikom dodavanja/izmene grupe, da li ćemo imati problem ukoliko neko u međuvremenu doda novu grupu ili izmeni postojeću?". Odgovor na to je bio : "Da". Ukoliko neko u međuvremenu doda ili izmeni minPoints na istu onu vrednost koju mi pokušavamo da podesimo, dobićemo situaciju gde imamo dve različite grupe sa identicnom vrednošću.

Za izmenu podataka o postojećoj grupi, koristimo metodu "updateLoyaltyAccountMembership" koji se nalazi na putanji "com.hesoyam.pharmacy.user.service.impl.LoyaltyAccountService.java".

```java
@Override
@Transactional(propagation = Propagation.REQUIRED, isolation = Isolation.SERIALIZABLE)
@Retryable(value = {CannotAcquireLockException.class},maxAttempts = 3, backoff = @Backoff(750))
public LoyaltyAccountMembership updateLoyaltyAccountMembership(LoyaltyAccountMembershipDTO loyaltyAccountMembershipDTO) throws LoyaltyAccountMembershipIn
    validateLoyaltyAccountMembership(loyaltyAccountMembershipDTO);
    LoyaltyAccountMembership loyaltyAccountMembership = loyaltyAccountMembershipRepository.getOne(loyaltyAccountMembershipDTO.getId());
    loadLoyaltyAccountMembershipWithDTOData(loyaltyAccountMembership, loyaltyAccountMembershipDTO);
    try{
        loyaltyAccountMembership = loyaltyAccountMembershipRepository.save(loyaltyAccountMembership);
        refreshLoyaltyAccounts();
        return loyaltyAccountMembership;

    }catch (DataIntegrityViolationException e){
        throw new LoyaltyAccountMembershipInvalidUpdateException("Loyalty membership name must be unique.");
    }
}

@Override
```

Ona je anotirana sa @Transactional u kojoj definišemo nivo izolacije I tip propagacije. Kao nivo izolacije, izabran je SERIALIZABLE.
Prvi korak je da validiramo izmenu (metoda pod imenom “validateLoyaltyAccountMembership”). Upravo u toj metodu proveravamo da li već postoji grupa sa takvim podešavanjem(minPoints).

```java
    @Transactional(propagation = Propagation.REQUIRED)
    public void validateLoyaltyAccountMembership(LoyaltyAccountMembershipDTO loyaltyAccountMembershipDTO) throws LoyaltyAccountMembershipInvalidUpdateExcept
        int membershipsWithSameMinPointsCount = loyaltyAccountMembershipRepository.countAllByMinPointsAndIdIsNot(loyaltyAccountMembershipDTO.getMinPoints(),
        if(membershipsWithSameMinPointsCount != 0) throw new LoyaltyAccountMembershipInvalidUpdateException("Memberships with same min points value are not
    }


}
```

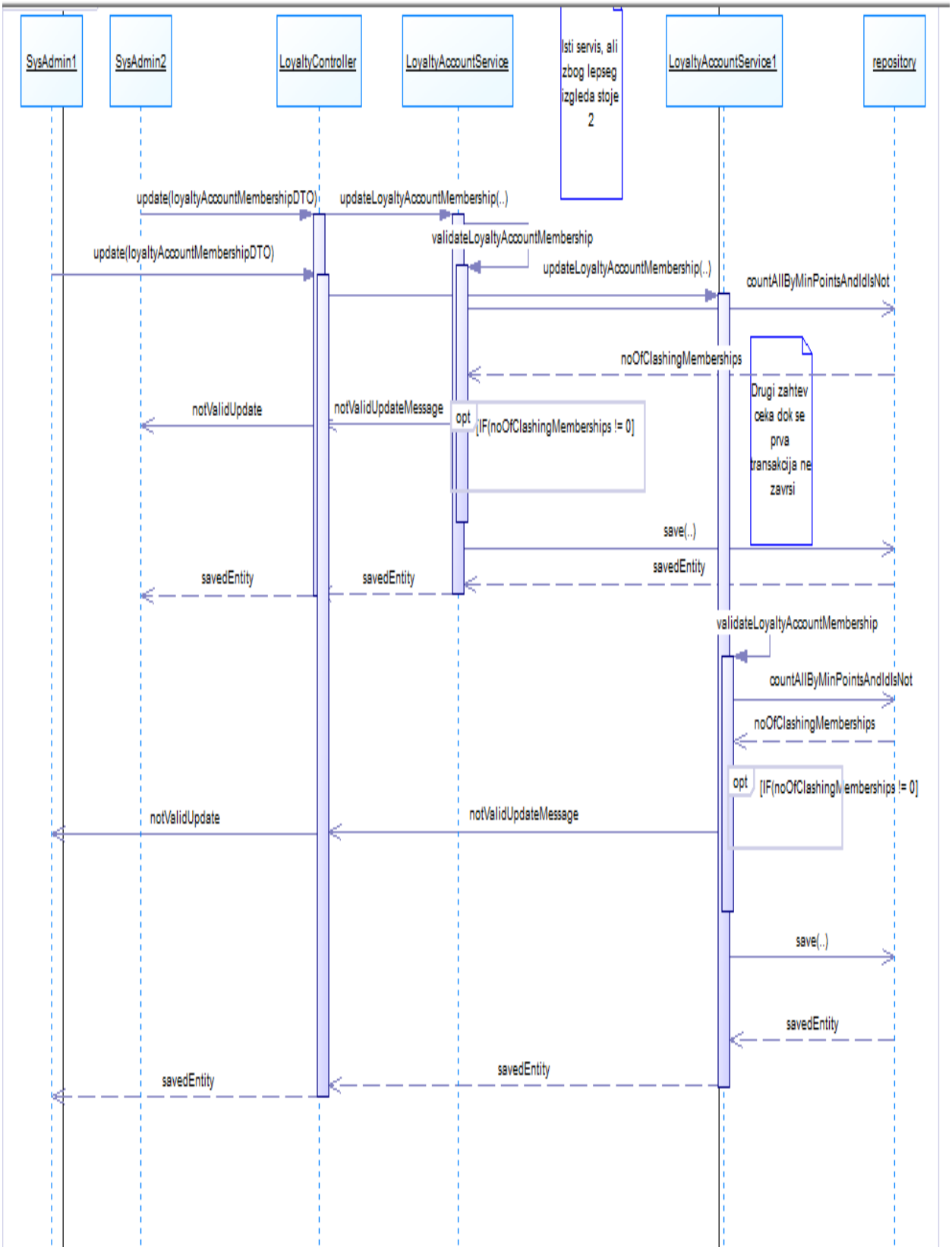
Ukoliko je sve u redu, snimamo tu grupu(membership).

Kada koristimo navedeni tip izolacije, DBMS će pokušati da izvrši više upita od jednom, ali tako da ispadne da su se upiti izvršili sekvencijalno. Kada to nije moguće, tj kada dve transakcije imaju zajedničke zavisnosti, dobijamo grešku koja nam sugeriše da ponovimo transakciju. Kako mi koristimo PostgreSQL, tačna greška koju dobijamo je "`org.postgresql.util.PSQLException: ERROR: could not serialize access due to read/write dependencies among transactions.`".

Ovaj problem rešavamo tako što pokušamo ponovo da pokrenemo transakciju. Probavamo maksimalno 3 puta, sa razmacima oko 750ms kako ne bi preopteretili bazu odjednom.

Kod kreiranja I brisanja grupe, sličan pristup je primenjen.

Na sledećoj strani se može naći vizuelni prikaz ovakve jedne situacije gde dva administratora informacionog sistema žele da izvrše promene nad grupama.


SysAdmin1
SysAdmin2
LoyaltyController
LoyaltyAccountService
Isti servis, ali zbog lepseg izgleda stoje 2
LoyaltyAccountService1
repository
update(loyaltyAccountMembershipDTO)
updateLoyaltyAccountMembership(..)
validateLoyaltyAccountMembership
update(loyaltyAccountMembershipDTO)
updateLoyaltyAccountMembership(..)
countAllByMinPointsAndIdIsNot
noOfClashingMemberships
Drugi zahtev ceka dok se prva transakcija ne zavrsi
notValidUpdate
notValidUpdateMessage
opt
[IF(noOfClashingMemberships != 0]
save(..)
savedEntity
savedEntity
savedEntity
validateLoyaltyAccountMembership
countAllByMinPointsAndIdIsNot
noOfClashingMemberships
opt
[IF(noOfClashingMemberships != 0]
notValidUpdate
notValidUpdateMessage
save(..)
savedEntity
savedEntity
savedEntity