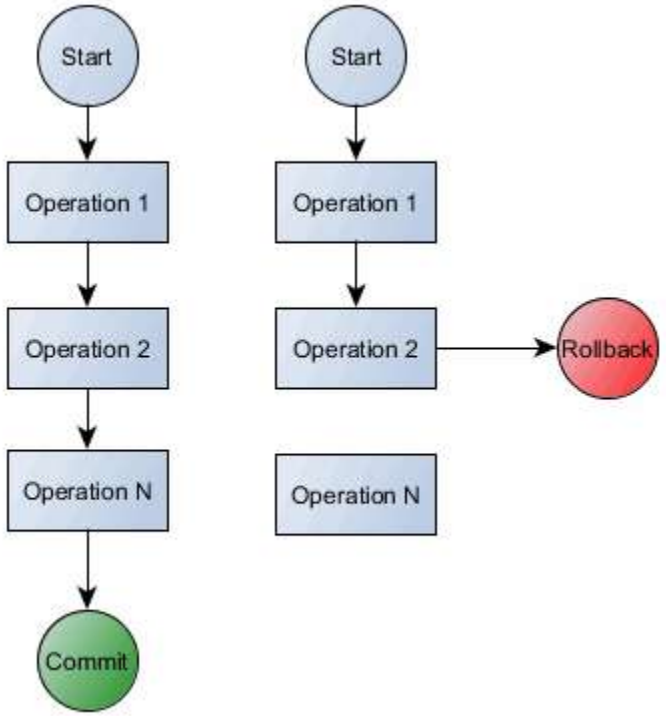
Transakcije

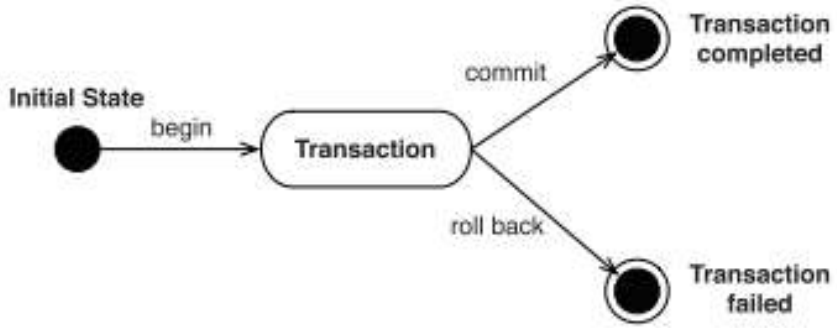
Transakcije

- Transakcije su prisutne u svim današnjim poslovnim sistemima, obezbeđujući integritet čak i u veoma konkurentnim okruženjima
- Transakcija je kolekcija read/write operacija koja je uspešna samo ako su sve sadržane operacije izvršene uspešno
- U relacionoj bazi podataka, svaki SQL upit se mora izvršiti u okviru transakcije
- Omogućavaju da više istovremenih korisnika rade konkurentno sa istim podacima bez da integritet podataka bude ugrožen



Transakcije

- Transakcija garantuje da će se završiti – ili uspešno (commit) ili neuspešno (rollback)
- Rollback vraća bazu u stanje pre početka. Za ovo mora postojati mehanizam da se zna koje je stanje bilo pre početka. Ovaj mehanizam se najčešće implementira kao transakcijski log gde pišu sve izmene koje su se načinile. Rollback čita te izmene i radi suprotne operacije



Životni ciklus transakcije

ACID – četiri svojstva transakcija

- **A**tomicity – transakcija se sastoji iz više read ili write operacija. Transakcija se smatra uspešnom samo ako su SVE operacije uspešno izvršene. Cela transakcija se posmatra kao jedna celina – ili će se sve operacije izvršiti, ili neće ni jedna
- **C**onsistency – jedna transakcija će da radi nad konzistentnim skupom podataka koji je sakriven od ostalih konkurentnih transakcija. Kada završi, podaci i dalje ostaju konzistentni jer se radi ili commit ili rollback
- **I**solaation – svaka transakcija mora da se radi u izolaciji, što omogućava da promene kod neuspešnih transakcija ne utiču na ostatak sistema. Ovo se postiže mehanizmimom zaključavanja (optimističko ili pesimističko)
- **D**urability – uspešna transakcija mora trajno da menja stanje sistema. Pre završetka, sve promene se beleže u transakcijski log

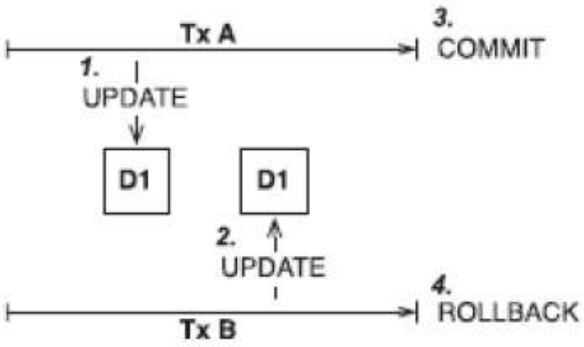
Konkurentni pristup podacima

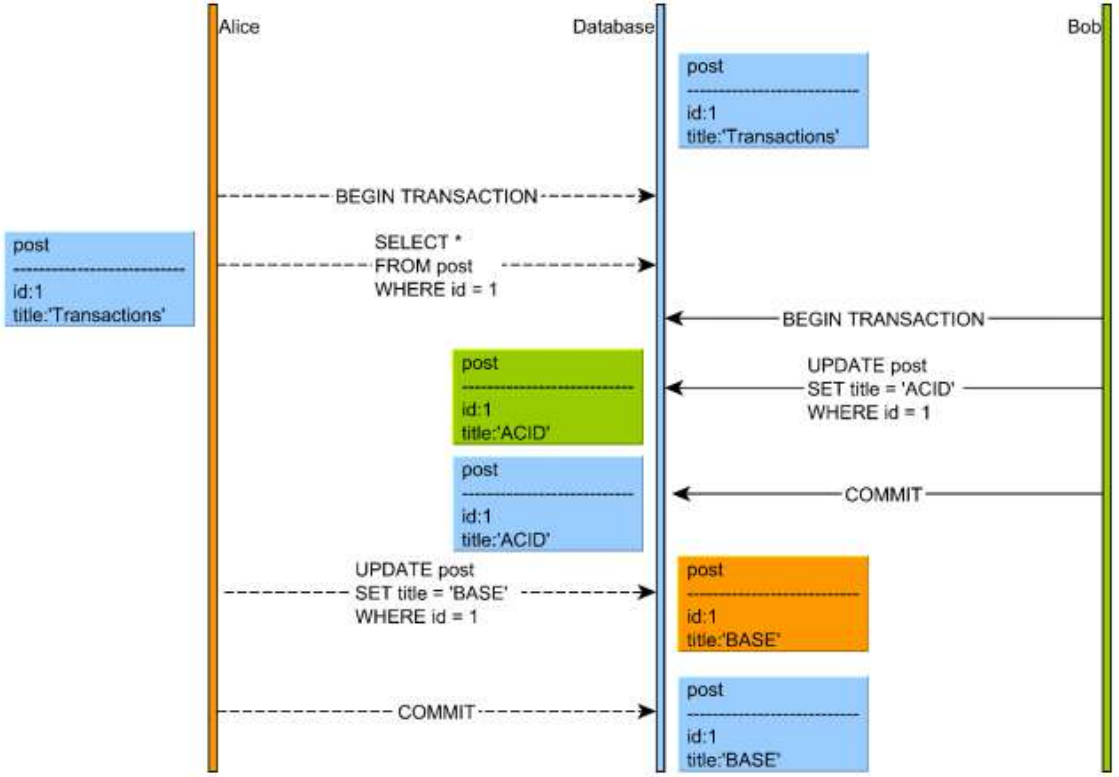
- Prilikom istovremenog pristupa podacima može da dođe do štetnog preplitanja rada više transakcija
- Tom prilikom može da se javi više problema:

1. Lost update
2. Dirty read
3. Unrepeatable read
4. Phantom read

Problem 1 – Lost update

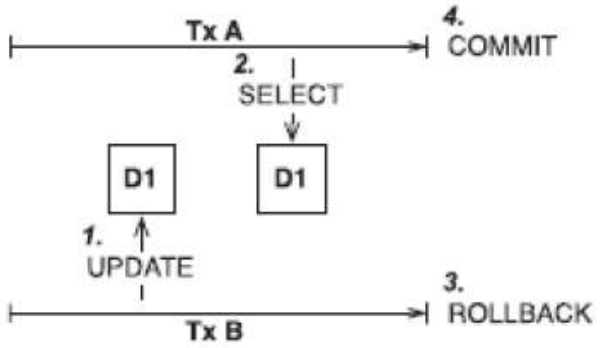
- Dve različite transakcije menjaju isti podatak bez zaključavanja
- Izmena koju je napravila transakcija se gubi, zato što druga transakcija nije bila svesna prethodne izmene, i ta izmena bude pregažena od strane druge transakcije

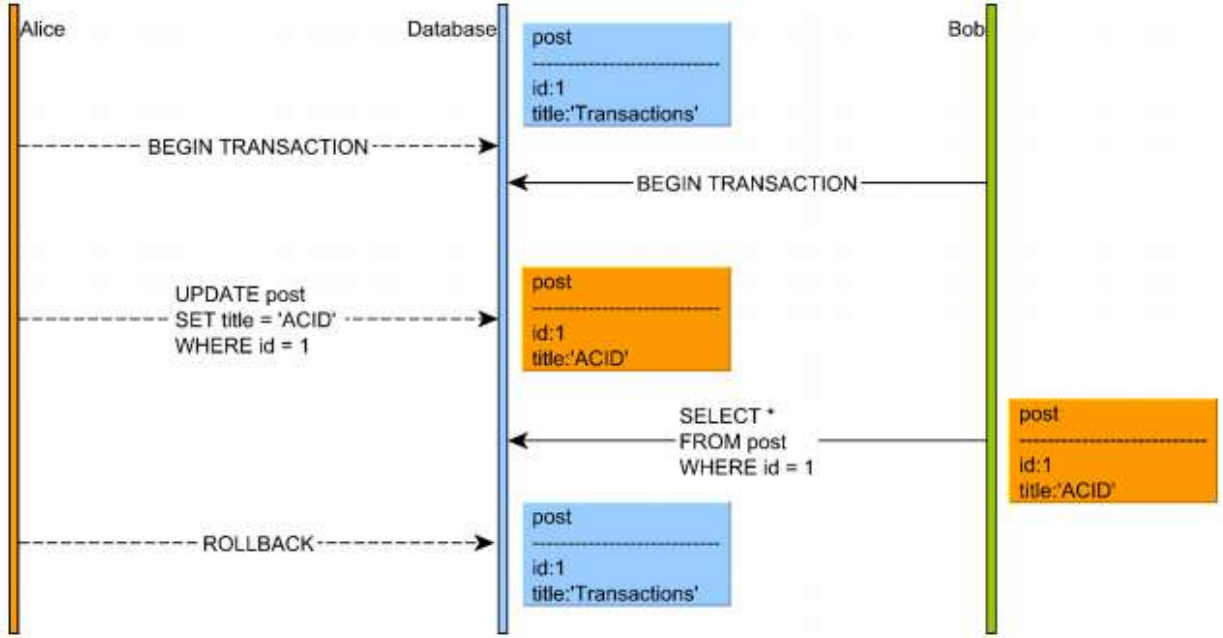




Problem 2 - Dirty read

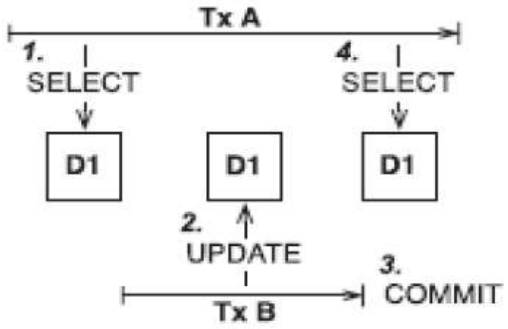
- Transakcija B menja podatke, transakcija A čita podatke pre nego što su izmene commit-ovane
- Ukoliko transakcija B uradi rollback, transakcija A će imati podatke koji ne postoje u bazi

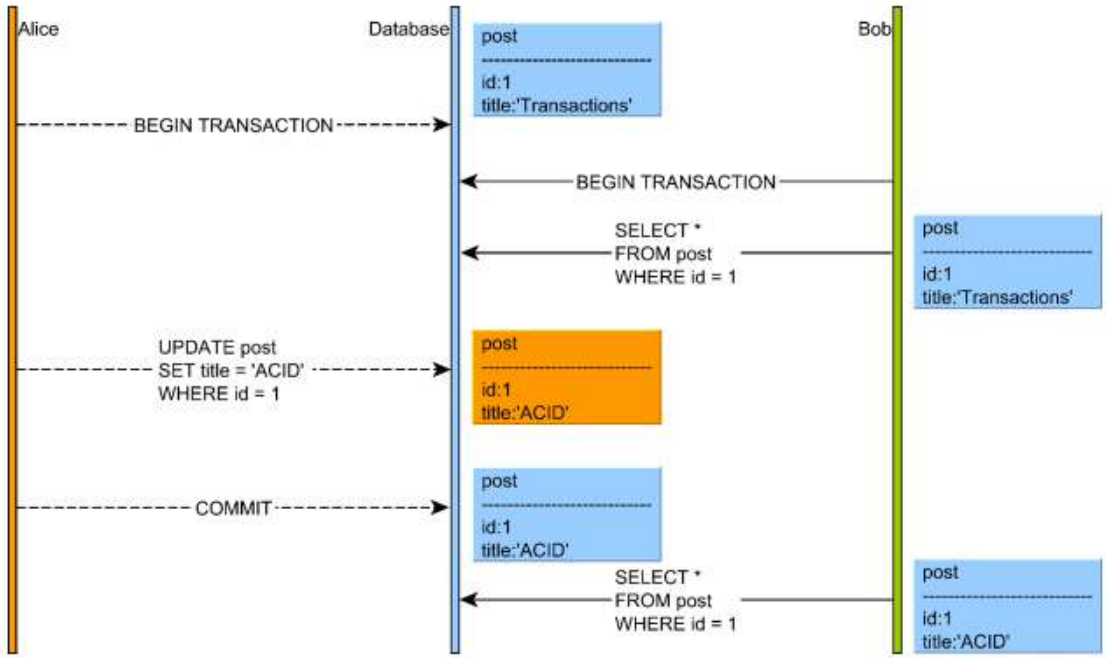




Problem 3 - Unrepeatable read

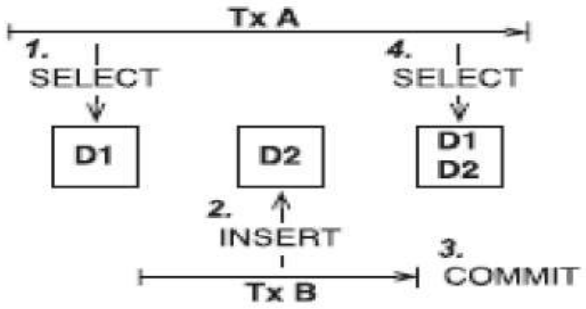
- Transakcija A dva puta čita iste podatke i dobija različiti sadržaj
- Transakcija A pročita jedan red, transakcija B izmeni ili obriše taj red i commit-uje izmene. Transakcija A ukoliko ponovo zatraži da pročita isti taj red, ili će dobiti različite podatke ili će dobiti informaciju da su podaci obrisani

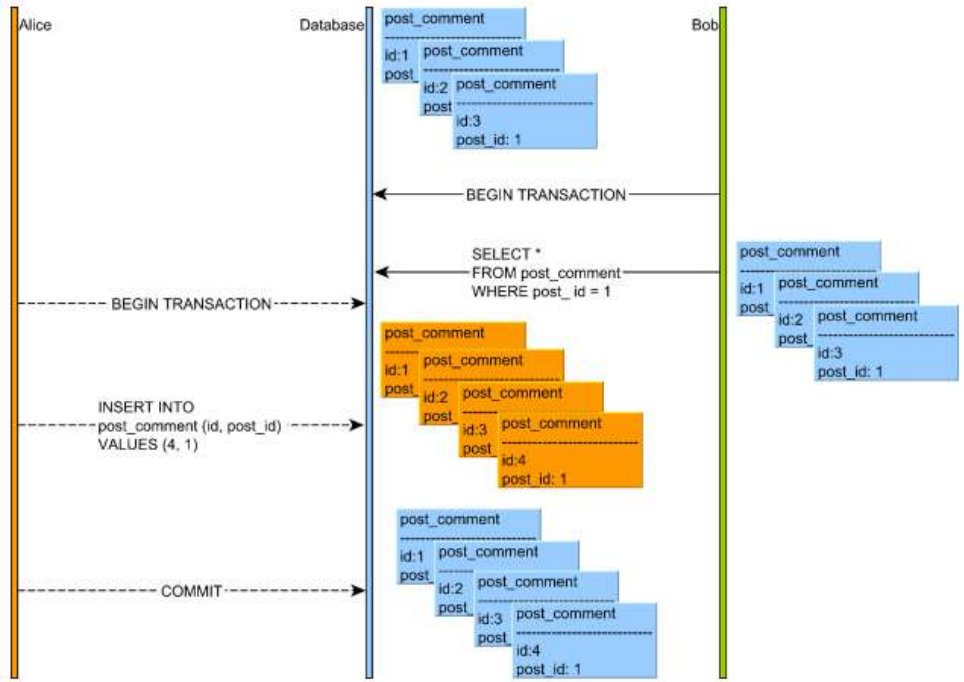




Problem 4 - Phantom read

- Transakcija A u drugom čitanju dobija i podatke kojih nije bilo prilikom prvog čitanja
- Primer: transakcija A šalje neki upit (recimo neki search criteria) i dobija odgovor koji zadovoljava postavljeni upit. Transakcija B kreira novi red u tabeli koji zadovoljava kriterijum pretrage za transakciju A. Ako transakcija A izvrši upit ponovo, različiti rezultati ulaze u odgovor





Nivoi izloacije

- Možemo da biramo nivo izolacije transakcija za svaku konekciju

1. READ_UNCOMMITTED – praktično deprecated
2. READ_COMMITTED - eliminiše problem "dirty read"
3. REPEATABLE_READ - eliminiše problem "unrepeatable read“
4. SERIALIZABLE - eliminiše problem "phantom read“

- Više informacija na linku https://vladmihalcea.com/2014/09/14/a-beginners-guide-to-database-locking-and-the-lost-update-phenomena/

1. Read uncommitted

- Transakcije nisu izolovane jedne od drugih
- Moguće je čitati izmene koje su načinile druge transakcije koje još nisu commit-ovane
- Mehanizmi zaključavanja koji se koriste da spreče tekuću transakciju od čitanja modifikovanih podataka ne blokiraju transakcije koje rade na ovom nivou izolacije

2. Read committed

- Transakcije ne mogu čitati podatke koji su modifikovani, a izmene još nisu commit-ovane od strane drugih transakcija
- Izmene se sakrivaju od konkurentnih transakcija, samo ih može videti ona transakcija koja ih je napravila
- Eliminiše problem „dirty read“

3. Repeatable read

- Transakcije ne mogu čitati podatke koji su modifikovani, a izmene još nisu commit-ovane od strane drugih transakcija
- Ni jedna druga transakcija ne može da menja podatke koji su pročitani od strane neke druge transakcije dokle god se ta transakcija ne završi
- Eliminiše problem "unrepeatable read“
- Neki ORM radni okviri (Jpa, Hibernate....) imaju podršku za *repeatbale reads* na aplikativnom nivou. Prva verzija svakog entitija koja se pročita iz baze se kešira u trenutni perzistentni kontekst. Svaki naredni SQL upit kojem u rezultat ulazi isti objekat će zapravo pročitati objekat koji je prethodno keširan, a ne iz baze. Na ovaj način, *Unrepeatable read* problem može da se reši i upotrebom *Read Committed* nivoa izolacije

4. Serializable

- Transakcije ne mogu čitati podatke koji su modifikovani, a izmene još nisu commit-ovane od strane drugih transakcija
- Ni jedna druga transakcija ne može da menja podatke koji su pročitani od strane neke druge transakcije dokle god se ta transakcija ne završi
- Druge transakcije ne mogu da unesu nove redove s ključevima čije vrednosti bi bi upadale u opseg ključeva koji su pročitani u tekućoj transakciji dokle god se tekuća transakcija ne završi
- Eliminiše problem "phantom read“

Nivoi izloacije

Isolation Level	Dirty read	Non-repeatable read	Phantom read
READ_UNCOMMITTED	allowed	allowed	allowed
READ_COMMITTED	prevented	allowed	allowed
REPEATABLE_READ	prevented	prevented	allowed
SERIALIZABLE	prevented	prevented	prevented

Pesimističko zaključavanje

- Svaka operacija treba da zaključa podatke **i za čitanje i za pisanje** sve dok se ne završi
- Garantuje ispravan rad
- Ima loše performanse: čak i ako dve transakcije pristupaju različitim redovima u tabeli može doći do blokiranja

Optimističko zaključavanje

- Svaka operacija pre izmene podataka treba da proveri da li je podatke neko drugi u međuvremenu menjao
- Poredi verziju podataka koje je pročitala sa onim što se trenutno nalazi u bazi. Ovo poređenje mora da se izvodi u režimu pesimističkog zaključavanja
- Ako su podaci menjani, prijavi se greška korisniku
- Polazi od pretpostavke da u praksi do kolizije dolazi jako retko, a situacije kada dođe do kolizije se otkrivaju i kontrola se vraća korisniku

Implementacija otimističkog zaključavanja

- Varijanta 1: poredimo sve vrednosti objekta sa vrednostima u bazi. Nezgodno ako tabela ima puno kolona.
- Varijanta 2: dodamo novu kolonu koja služi kao brojač izmena. Na svaku izmenu u datom redu tabele inkrementiramo njenu vrednost.

Granice transakcija

- Svaki upit mora da se izvrši u kontekstu transakcije, čak i ako klijent eksplicitno ne navede granice transakcije
- Ne postoji ni jedan način da se bilo koji upit izvrši van transakcije. Transakcija garantuje da će se završiti ili uspešno ili neuspešno i tom prilikom sve izmene koje je načinila će biti poništene
- Bez eksplicitno definisanja granica transakcije, baza podataka će koristiti implicitnu transakciju koja se odnosi na svaki pojedinačni upit. Implicitna transakcija počinje pre nego što se upit izvrši i završi se(commit ili rollback) nakon što se upit izvrši. Režim implicitnih transakcija poznat je pod nazivom *auto-commit*
- *Auto-Commit* treba izbegavati koliko god je moguće

Auto-Commit

- Primer prenos novca sa dva računa – istovremeno sa jednog računa određeni iznos mora biti skinut a na drugi dodat i ukupna količina novca mora ostati ista posle transakcije
- Ukoliko drugi update padne, samo se te izmene mogu poništiti. Sa prvog računa je novac skinut i izmene automatski commit-ovane i ne mogu se vratiti

```
try(Connection connection = dataSource.getConnection();
    PreparedStatement transferStatement = connection.prepareStatement(
        "UPDATE account SET balance = ? WHERE id = ?"
    )) {
    transferStatement.setLong(1, Math.negateExact(cents));
    transferStatement.setLong(2, fromAccountId);
    transferStatement.executeUpdate();

    transferStatement.setLong(1, cents);
    transferStatement.setLong(2, toAccountId);
    transferStatement.executeUpdate();
}
```

Definisanje granica

- Transakcija će biti commit-ovana jedino ako oba upita izmene uspeju
- Ako padne bar jedan upit, promene se poništavaju, i na oba računa ostaje ista količina novca

```
try(Connection connection = dataSource.getConnection()) {
    connection.setAutoCommit(false);
    try(PreparedStatement transferStatement = connection.prepareStatement(
        "UPDATE account SET balance = ? WHERE id = ?"
    )) {
        transferStatement.setLong(1, Math.negateExact(cents));
        transferStatement.setLong(2, fromAccountId);
        transferStatement.executeUpdate();

        transferStatement.setLong(1, cents);
        transferStatement.setLong(2, toAccountId);
        transferStatement.executeUpdate();

        connection.commit();
    } catch (SQLException e) {
        connection.rollback();
        throw e;
    }
}
```

Transakcije i Spring

@Transactional

- Anotacija koja se koristi za definisanje granica transakcije
- Deklarativno označavanje da se nešto izvršava u transakciji radi se upotrebom ove anotacije
- Može se iskorisititi na nivou klase što će značiti da će svaka metoda unutar klase biti obrađena u transakciji
- Pomoću ove anotacije možemo da konfigurišemo sledeće atribute: readOnly, timeout, rollbackFor, noRollbackFor, propagation, isolation

@Transactional - atributi

1. ReadOnly – označava da li će transakcija samo da čita podatke. Ukoliko je true, baza će raditi neke optimizacije, i ako se proba update, baca izuzetak
2. Timeout – označava koliko dugo će se transakcija izvršavati timeout pre nego što će se uraditi rollback
3. RollbackFor - označava za koje izuzetke (Exception klase) će se izvrsiti rollback
4. NoRollbackFor - označava za koje izuzetke (Exception klase) se neće izvrsiti rollback
5. Propagation - obično će se sav kod izvršiti unutar transakcije, međutim pomoću ove opcije može se naznačiti kako ce se metoda izvrsiti u zavisnosti od postojanja/nepostojanja transakcionog konteksta
6. Isolation - može se definisati drugačiji nivo izolacije u odnosu na podrazumevani za odgovarajuću bazu

Propagation

- REQUIRED - metoda se priključuje tekućoj transakciji, otvara novu ako transakcija ne postoji (podrazumevano)
- REQUIRES_NEW - metoda uvek pokreće novu transakciju, ako postoji tekuća transakcija ona se suspenduje
- MANDATORY - metoda mora da se izvršava u transakciji koja mora biti ranije pokrenuta; ako je nema jvlja se greška
- SUPPORTS - metoda će se priključiti tekućoj transakciji, ako ona postoji; ako ne postoji, izvršava se bez transakcije
- NOT_SUPPORTED - metoda se izvršava bez transakcije, čak i ako postoji tekuća transakcija
- NEVER - metoda se izvršava bez tranksacije; ako postoji tekuća transakcija, javlja se greška
- NESTED - metoda se izvršava u ugnježdenoj transakciji ako je trenutni thread povezan sa transakcijom u suprotnom startuje novu transakciju

Optimističko zaključavanje

- Spring (kao i EJB) koristi posebnu anotaciju @Version kojom se anotira (obično integer) polje koje će se pri svakoj promeni entiteta povećavati za 1
- Svaki klijent će dobiti i informaciju o verziji podatka
- Prilikom izmene podatka potrebno je proveriti da li je podatke neko drugi u međuvremenu menjao:
  - poredi se verzija podatka koju je klijent pročitao sa onim sto se trenutno nalazi u bazi
  - poređenje se mora izvoditi u režimu pesimističkog zaključavanja
  - ako su podaci menjani prijavljuje se greška korisniku

Pesimističko zaključavanje

- Resurs se može zaključati pozivom metode lock() nad EntityManager instancom i prosleđivanjem objekta koji se zaključava ili dodavanjem anotacije @Lock na metodu repozitorijuma (findOne()) koja vraća objekat za zaključavanje. Taj objekat ostaje zaključan do kraja transakcije.
- Dva osnovna lock-a (ima ih više) koja se mogu iskoristiti su:
  - PESSIMISTIC_READ – tzv. shared lock koji kada se dobije sprečava bilo koju drugu transakciju da uzme PESSIMISTIC_WRITE lock
  - PESSIMISTIC _WRITE lock – tzv. exclusive lock koji kada se dobije sprečava bilo koju drugu transakciju da uzme PESSIMISTIC_READ ili PESSIMISTIC_WRITE lock