by LINKgroup
ITAcademy™
Škola za profitabilnu IT karijeru

Upravljanje transakcijama

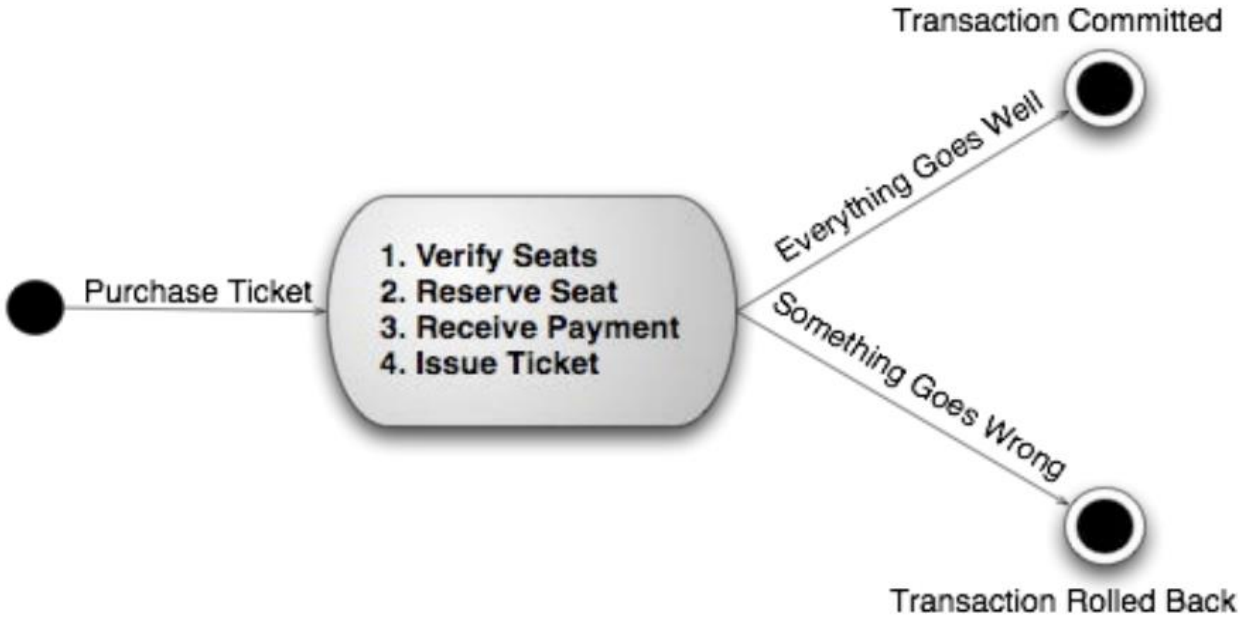
DARKO.ZIVANOVIC@IT-AKADEMIJA.COM

Uvod (1)

- U razvoju softvera, "sve ili ništa" operacije se nazivaju transakcijama.
- Transakcije nam omogućuju da grupišemo nekoliko operacija u jedinstvenu jedinicu posla koja će biti ili u potpunosti izvršena ili u potpunosti neće biti izvršena.
- Ako sve prođe dobro, transakcija je uspešna. Ako nešto krene naopako, sve se vraća u pređašnje stanje, i kao da se ništa nije ni dogodilo.

Uvod (2)

Koraci pri kupovini bioskopske karte:



Koraci koji su uključeni u kupovinu karte moraju biti "sve ili ništa".

Transakcije u četiri reči (*ACID*)

- Atomičnost (***A**tomic*) - obezbeđuje da se sve operacije u transakciji izvrše ili da se nijedna od njih ne izvrši.
- Konzistentnost (***C**onsistent*) - kada se transakcija završi (bilo to uspešno ili ne), sistem se ostavlja u ispravnom stanju.
- Izolovanost (***I**solated*) - transakcije moraju biti međusobno izolovane da bi se sprečilo istovremeno čitanje i upis istih podataka.
- Trajnost (***D**urable*) - Kada se transakcija završi, rezultati transakcije treba da postanu trajni, što obično znači smeštanje rezultata u bazu podataka ili neko drugo trajno skladište.

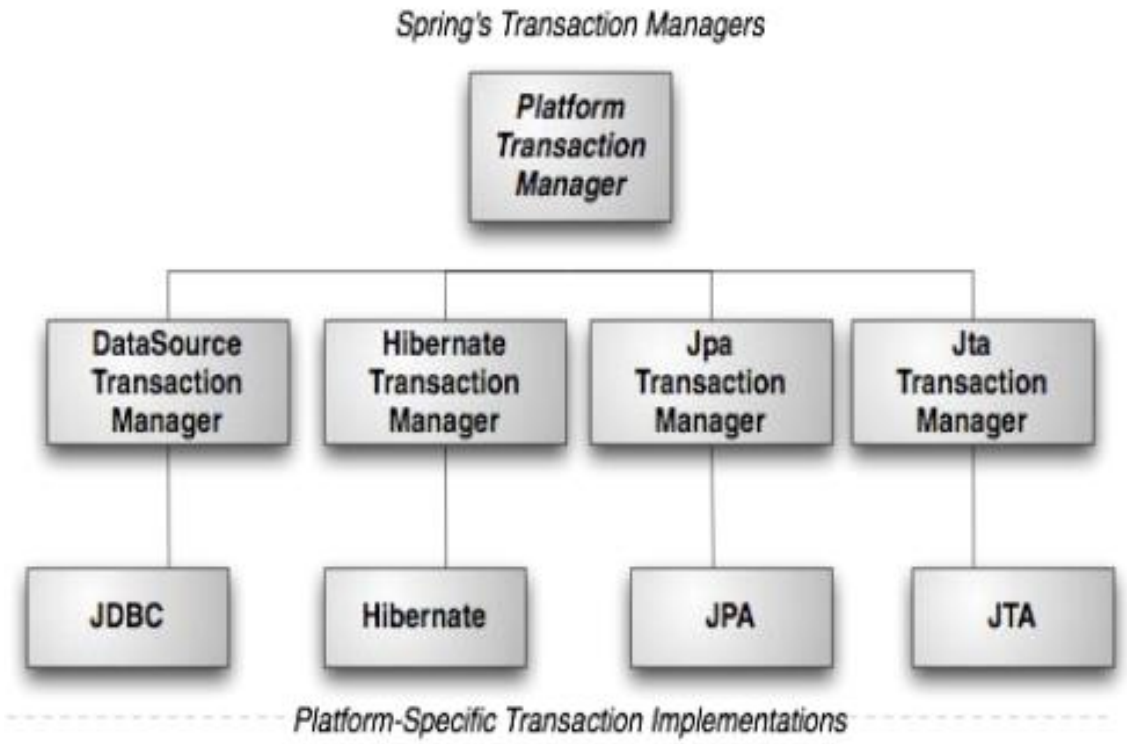
Vrste upravljača transakcijama (1)

Transaction manager (`org.springframework.*`)	**Use it when...**
`jca.cci.connection.CciLocalTransactionManager`	Using Spring's support for J2EE Connector Architecture (JCA) and the Common Client Interface (CCI).
`jdbc.datasource.DataSourceTransactionManager`	Working with Spring's JDBC abstraction support. Also useful when using iBATIS for persistence.
`jms.connection.JmsTransactionManager`	Using JMS 1.1+.
`jms.connection.JmsTransactionManager102`	Using JMS 1.0.2.
`orm.hibernate3.HibernateTransactionManager`	Using Hibernate 3 for persistence.

Vrste upravljača transakcijama (2)

`orm.jdo.JdoTransactionManager`	Using JDO for persistence.
`orm.jpa.JpaTransactionManager`	Using the Java Persistence API (JPA) for persistence.
`transaction.jta.JtaTransactionManager`	You need distributed transactions or when no other transaction manager fits the need.
`transaction.jta.OC4JJtaTransactionManager`	Using Oracle's OC4J JEE container.
`transaction.jta.WebLogicJtaTransactionManager`	You need distributed transactions and your application is running within WebLogic.
`transaction.jta.WebSphereUowTransactionManager`	You need transactions managed by a `UOWManager` in WebSphere.

Upravljači transakcijama u *Spring*-u



JDBC transakcije

- Ako koristimo JDBC u aplikaciji, **DataSourceTransactionManager** će se pobrinuti da odredi granice transakcija umesto nas.
- U pozadini, **DataSourceTransactionManager** upravlja transakcijama pozivajući metode **java.sql.Connection** objekta: **setAutoCommit(false)**, **commit()**, **rollback()**.
- Upravljač transakcija dobija **java.sql.Connection** objekat od **DataSource**-a.

```
<bean id="transactionManager" class="org.springframework.jdbc.
       datasource.DataSourceTransactionManager">
  <property name="dataSource" ref="dataSource"/>
</bean>
```

Hibernate transakcije

- *Hibernate* upravljač transakcija ima referencu na objekat tipa **`SessionFactory`**.
- **`SessionFactory`** generiše objekat tipa **`org.hibernate.Session`**.
- Sesija ima referencu na objekat tipa **`org.hibernate.Transaction`** nad kojim je moguće pozvati metode **`commit()`** i **`rollback()`**.

```
<bean id="transactionManager" class="org.springframework.
        orm.hibernate3.HibernateTransactionManager">
  <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

JPA transakcije

- **JpaTransactionManager** mora samo biti povezan sa bilo kojom implementacijom **javax.persistence.EntityManagerFactory**, a **JpaTransactionManager** će sarađivati sa JPA **EntityManager**-om da bi izvršio transakciju.

```
<bean id="transactionManager"
    class="org.springframework.orm.jpa.JpaTransactionManager">
  <property name="entityManagerFactory"
      ref="entityManagerFactory" />
</bean>
```

Deklarativno upravljanje transakcijama

- **`@Transactional`** – transakcija na nivou metode.
- Upravljač transakcijama obavija metodu kodom koji će je izvršiti kao transakcionu jedinicu.

```
UserTransaction utx = entityManager.getTransaction();

try {
    utx.begin();

    ourBusinessLogicMethod();

    utx.commit();
} catch(Exception ex) {
    utx.rollback();
    throw ex;
}
```

- Deklarativne transakcije redukuju količinu ponavljajućeg koda, ali zato smanjuju fleksibilnost.
- Ukoliko je veći deo metode "pomoćni" kod, a manji deo metode se odnosi na kod koji treba izvršiti u transakciji, pogodnije je programsko upravljanje transakcijama.

Konkurentne transakcije

- Kod klasičnih aplikacija više transakcija nad istim skupom podataka se izvršava istovremeno.
- Problemi koji mogu nastati:
  - Prljavo čitanje (*dirty read*)
  - Neponovljivo čitanje (*nonrepeatable read*)
  - Fantomsko čitanje (*phantom read)*
- U idealnom slučaju, transakcije bi bile potpuno međusobno izolovane i izbegli bi se ovi problemi.
- Perfektna izolacija podrazumeva zaključavanje redova ili kompletnih tabela u bazi, što dramatično utiče na performanse aplikacije.

Ostale karakteristike transakcije:

- *Read-only* - govori da li je u pitanju transakcija koja samo čita podatke.
- *Transaction timeout* - mogućnost da se transakcija automatski poništi posle određenog broja sekundi.
- *Rollback rules* - skup pravila koja definišu koji izuzeci dovode do poništenja transakcije, a koji ne dovode.

Deklarisanje transakcija anotacijama

- Kao dopuna na **<tx:advice>** element postoji **<tx:annotation-driven>** element, čija upotreba je vrlo jednostavna:

```
<tx:annotation-driven />
```

- Konfigurisanje **<tx:annotation-driven>** elementa govori Spring-u da ispita sva zrna u aplikacionom kontekstu i da traži ona oznacena sa **@Transactional**, bilo na nivou klase ili na nivou metode.

```
@Transactional(propagation=Propagation.SUPPORTS, readOnly=true)
public class SpitterServiceImpl implements SpitterService {
...
  @Transactional(propagation=Propagation.REQUIRED, readOnly=false)
  public void addSpitter(Spitter spitter) {
...
  }
...
}
```