

Transaction'lar

Burak Bayramlı

Bu belge, \LaTeX ile üretilmiştir

Transaction

- Bir iş birimi, iş ünitesi
- Altında olan işlemlerin ya hepsi **başarılı**, ya da **başarısız** olması gereken bir grup

Örnek

Bir bankadaki X hesabından Y hesabına para aktarması gereken bir program düşünün.

X'den eksiltme ile Y'ye ekleme, aynı anda başarılı ya da başarısız olmalıdır.

Veri Tabanında Transaction

- Veri tabanında her işlem bir transaction altında yapılır
- İlişkisel veri tabanında JDBC ile her yeni SQL komutu bir transaction başlatır.
- Komut `commit` verildiği anda başlangıçtan o ana kadar yapılmış tüm işlemler aynı anda veri tabanına kalıcı şekilde yazılır.

Transaction ve Atomicity

Verilerin aynı anda yazılmasına atomicity denir.

Transaction ve İzolasyon

Bir transaction'ın commit edilmeden, öteki transaction'ların o yapılan değişiklikleri görmemesine **izolasyon** denir.

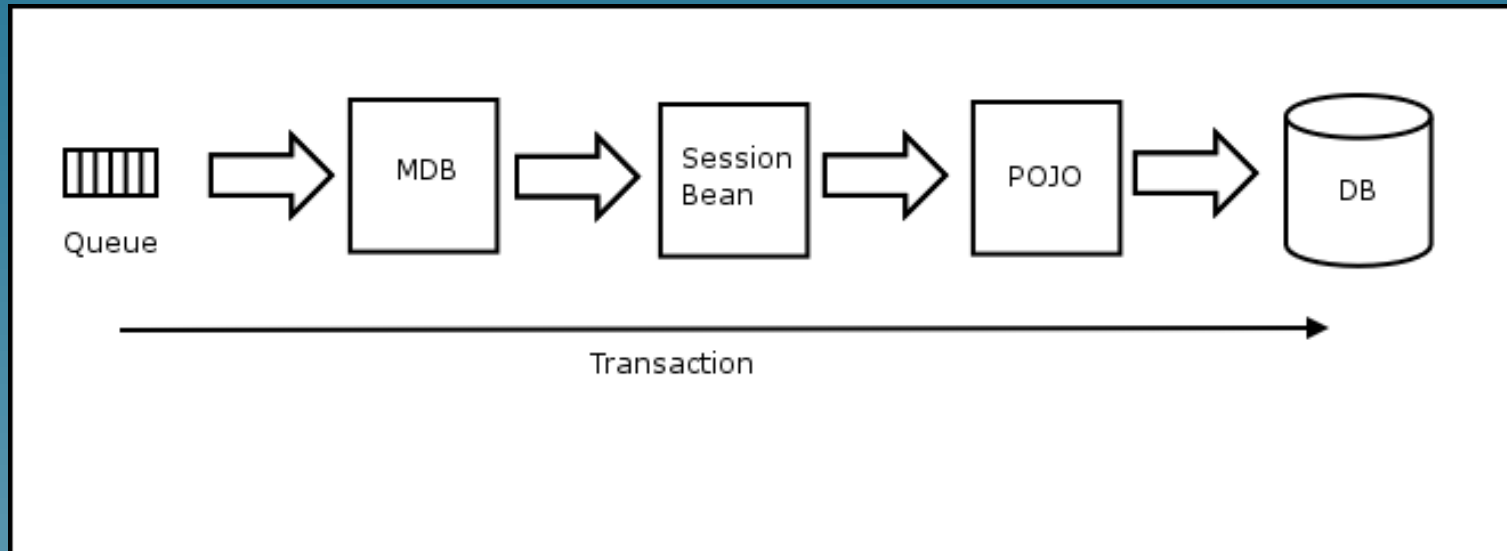
Session Bean ve Transaction

Session Bean'ler ve transaction, nesne seviyesinde izolasyon ile alakalı değildir. Çünkü paylaşılan veri, veri tabanında.

Session Bean ve Transaction

Session Bean'ler için istenen, veri tabanı, diğer Session Bean'ler, Pojo'lar ve queue'ları kapsayan transaction'ın metot bazlı başlatılabilmesi, ve bu işlemde tek bir aşamada Exception atılırsa tüm Session Bean, Pojo ve kaynaklardaki işlemin rollback edilmesidir.

Genel Resim



Session Bean ve Transaction

Bean üzerinde transaction gerekip gerekmediğini annotation ile belirtebiliyoruz.

Session Bean ve Transaction

```
@Stateful
public class ServiceXYZ implements XYZ, Serializable
{
    @TransactionAttribute
    (TransactionAttributeType.REQUIRED)
    public void metot1() {
        ...
    }
    ...
}
```

Session Bean ve Transaction

- **REQUIRED:** Eğer bu metoda gelmeden önce transaction var ise, ona dahil ol, yoksa yeni başlat.
- **MANDATORY:** Daha önceden bir transaction başlatılmış olmalı. Yok ise, hata ver.
- **REQUIRESNEW:** Yepyeni bir transaction lâzım. Eğer önceden var ise, o askıya alınır.

- **SUPPORTS:** Önceden transaction var ise, transaction içinde, yok ise, o hâlde işletilir.
- **NOT_SUPPORTED:** Eğer bean'e transaction ile gelinirse, hata ver.

MDB ve Transaction

Default olarak MDB'nin transaction ayarı `REQUIRED` (aynı Session Bean gibi) ve idare eden Container (yâni JBoss olacaktır). Bu yüzden `onMessage` transaction başlatır.

Hibernate

Hibernate

- Veriye nesnesel erişimi halleden teknoloji
- EJB3 annotation'ları artık persistence'i de hallediyor
- Hibernate bu annotation'lar için bir implementation motoru olmuştur.
- Tabii EJB3 Entity Bean tanımında Hibernate kurucusu Gavin King'in büyük etkisi oldu.

Bir Nesneyi Tabloya Yazmak

```
@Entity
public class XYZ implements Serializable {
    ...
    String col1;

    @Id
    @Column(name="col1")
    public String getCol1() {
        return col1;
    }
}
```

```
public void setCol1(String col1) {  
    this.col1 = col1;  
}  
...  
}
```

persistence.xml

```
<persistence>
  <persistence-unit name="kitapDemoDatabase">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>java:/kitapDemoDataSource</jta-data-source>
    <properties>
      <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.cache.provider_class"
        value="org.hibernate.cache.HashtableCacheProvider"/>
      <property name="hibernate.transaction.manager_lookup_class"
        value="org.hibernate.transaction.JBossTransactionManagerLookup"/>
      <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
    </properties>
  </persistence-unit>
</persistence>
```

persistence.xml

- `hibernate.hbm2ddl.auto` değeri `create-drop` ise, Hibernate eşlemelere bakarak, veri tabanındaki tabloları **otomatik olarak yaratır**.
- `dialect` tanımı her veri tabanı için değişiktir.
- `jta-data-source` bir JDBC data source tanımıdır.

kitapdemo-ds.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
    <jndi-name>kitapDemoDataSource</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/[___DB_ISMI___]</connection-url>
    <driver-class>org.gjt.mm.mysql.Driver</driver-class>
    <user-name>___USER___</user-name>
    <password>___PASSWORD___</password>
    <exception-sorter-class-name>
      org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter
    </exception-sorter-class-name>
    <metadata>
      <type-mapping>mySQL</type-mapping>
    </metadata>
  </local-tx-datasource>

```

</datasources>

Dosyalar

- `resources/kitapdemo-ds.xml`
- `resources/META-INF/persistence.xml`
- `resources/import.sql`

Hibernate ve Transaction

- `persistence.xml` üzerinde yaptığımız hibernate. → `transaction.manager.lookup.class` sayesinde JBoss Hibernate üzerinden yaptığımız tüm veri erişimleri, o anda işlemekte olan transaction'a dahil olacaktır.
- Hibernate ise bu transaction'ı alıp JDBC'ye aktarabilecektir.
- JDBC zaten veri tabanı için bir önyüzdür. Böylece DB

transaction'ı da bizim kontrolümüzde olacak.

Nesneler Birbirini Nasıl Bulur?

- MDB, EJB'yi
- EJB, Hibernate'i

EJB Enjekte Etmek

```
@MessageDriven (...)  
public class SampleMdb implements MessageListener {  
  
    @EJB XYZ xyz;  
    ...  
}
```

PersistenceContext Enjekte Etmek

```
@Stateful
public class XYZBean implements XYZ {

    @PersistenceContext
    EntityManager entityManager;
    ...
}
```

PersistenceContext

Bir nesneyi veri tabanına yazmak için Session Bean içinden `entityManager.persist(obj)` gibi bir çağrı yeterlidir.

Hata Olursa

- İşte işin güzel tarafı...
- Bu zincirde herhangi bir noktada Exception atılırsa
 - ★ Veri tabanına yazma işlemi rollback edilir
 - ★ Ve Queue'dan alınmış Message **geri konur**

Soru

Mesaj queue'ya geri konulunca ne olacak?

Görevler

- Kurmadıysanız MySQL veri tabanını kurun.
- Car adlı bir class yazın, üzerinde `licensePlate` ve `description` adında iki öge olsun. `licensePlate`, `license_plate` adlı kolona map etsin. Diğer öğeye kolon ismi vermeyin, default halini kullanın.
- `CarInterface` ve `CarBean` adlı iki class yaratın. Bu class üzerinde `add(Car car)` adlı bir metot olsun. Bu

metot'ta `EntityManager` kullanarak `car` nesnesini veri tabanına kaydedin.

- MDB kodlarını daha önceki projelerden alabilirsiniz. MDB, `Session Bean`'i, o da `Hibernate`'i çağırarak.
- MDB'ye test için `TextMessage` gönderin. `Message` içeriği 1,2,3 gibi sayılar içersin. Bu içerik `Car` objesinin `description` ve `licensePlate` öğelerine değer olarak konacak.
- Rollback testi için şunu yapın: Eğer bir `description`

(meselâ) değeri '2' ise, bir `RuntimeException` atın.

- Tabii aynı mesaj tekrar geri geleceği için, bu mesajın kaç kere geri geldiğini `Session Bean` içinde bir `static` sayaç ile sayın. 3 kereden sonra mesajı dikkate almayın.
- Dikkat: `onMessage` içindeki `catch(..)` ifadelerini çıkartın. Bunlar varsa `Exception`'larınızın `JBoss`'a çıkmasını engeller.

POJO Import

```
java.io.*;  
javax.persistence.Entity;  
javax.persistence.Column;  
javax.persistence.Id;
```

MDB Import

```
javax.ejb.MessageDriven;  
javax.ejb.ActivationConfigProperty;  
javax.jms.Message;  
javax.jms.MessageListener;  
javax.ejb.MessageDrivenContext;  
javax.annotation.Resource;  
org.apache.log4j.Logger;  
javax.ejb.EJB;  
javax.jms.TextMessage;
```

Session Bean Import

```
javax.ejb.Stateless;  
javax.ejb.Stateful;  
java.io.Serializable;  
javax.ejb.TransactionAttribute;  
javax.ejb.TransactionAttributeType;  
javax.persistence.PersistenceContext;  
javax.persistence.EntityManager;
```