

Sorgulamak

Burak Bayramlı

Bu belge, \LaTeX ile üretilmiştir

Nasıl?

- Raporlama amaçlı ya da elimizde tekil bir nesne ID'si olmadığı zaman, sorgulama yaparak istediğimiz nesnelere ulaşmak isteriz.
- Veri tabanlarında `SELECT .. WHERE` kullanılarak, çoğu zaman `join` desteği ile işletilen `sQL` sorgulamasının karşılığı Hibernate'de nedir?

HQL

- Hibernate nesne isimleri ve alanlarını kullanarak bir sorgu yaratmanızı mümkün kılar. HQL denen dil bu çevirimi sağlıyor,
- Nesne ve alan isimlerine ek olarak, nesnesel ilişkiler de sorgulama sırasında kullanılabilir. Bu ilişkiler arka planda SQL JOIN komutlarına dönüşecektir.
- SQL'de bulunan fonksiyonlar, SUM, MAX, gruptama

özellikleri gibi, aynen HQL'de de mevcuttur.

Sonuçlar

HQL'den gelen sonuçlar üç şekilde olabilir;

- `query.getResultList()` ile bir nesne listesi
- `query.getSingleResult()` ile tek bir nesne
- Nesne değil kolon bazlı bir liste istendiyse, bir `Object []` listesi

Sorgu Sözdizimi

```
select obj from ClassIsmi obj  
where obj.property1 = :param1
```

Sorgu Sözdizimi

Bu sorguyu işletmek için

```
Query query = entityManager.createQuery('select ..');  
query.setParameter("description", description);  
List list = query.getResultList();
```

Sorgu Sözdizimi

Eğer gelecek sonucun **tek bir nesne** olacağından emin isek `getResultList()` yerine `getSingleResult()` çağrısını kullanmamız gerekir. Bu metot, `List` yerine `ClassIsmi` döndürecektir.

En Basit Sorgu

```
from ClassIsmi
```

... Tüm satırları döndürür

Join İçeren Sorgu

```
select x from Xyz x  
left join x.abcList abc  
where abc.property1 = :param1
```

İkinci Zıplama İle

```
select x from Xyz x
left join x.abclist a
where a.obje1.property1 > :param1
```

Dikkat edin, join *edilen* objeden üçüncü bir nesneye zıplıyoruz.

Like

HQL'de her türlü SQL-vari filtrelemeyi, fonksiyonu kullanabildiğimizden bahsetmiştik. “Like” komutu bunlardan biridir. Like, “benzer” kelimeleri bulmamızı sağlar.

```
from Abc abc where  
abc.property3 like '%kelime1%'
```

Sum

```
select sum(c.property2) from Class1 c
```

Max

```
select max(c.property2) from Class1 c
```

Sum ve Select

Bu iki fonksiyonu içeren çağrı çoğunlukla geriye tek sonuç (toplam ya da maksimum sayı) döndürecektir. Bu sonucu `getSingleResult` ile alabiliriz.

Criteria API

- Hibernate'in dinamik olarak sorgu üretebilmesini sağlar.
- HQL'i muhakkak String'leri birbirine yapıştırarak ta dinamik olarak üretebiliriz, fakat Criteria API bu işi daha rahat akılda kalacak, ve hata imkanını azaltabilecek API'ler üzerinden yapmanızı sağlar.
- Criteria Hibernate'e özel bir API'dir. Erişmeniz için

```
EntityManager hEm = (EntityManager)em;
```

```
Session s = hem.getSession();  
s.createCriteria(..);
```

Criteria API

Bu API'de her türlü filtreleme, HQL mekanizması bir Hibernate API çağrısı ile sorgunuza eklenebilir. Meselâ

```
Class1 c = (Class1)
    s.createCriteria(Class1.class)
      .add(Expression.eq("property1", new Integer(10)))
      .uniqueResult();
```

Burada bir *eşitlik* filtresini `Expression.eq` kullanarak eklemiş olduk.

Nesneler Yerine Alanlar Döndürmek

HQL'e "select obj" yerine

```
select obj.prop1, obj.prop2, ..
```

kullanmak mümkündür. Bu durumda geri gelen bir obje listesi olamaz, çünkü elimizde her bir List elemanı içinde, birden fazla öge var. Bu durumda bize bir *listenin listesi* döndürülüyor.

Nesneler Yerine Alanlar Döndürmek

Fakat bir `Object []` listesi bazen kod idaresi açısından istenmez olabilir. Bu durumda, `Object []`'i bir yardımcı diğer bir class'a map edebiliriz. Bu nesneye genelde `Summary` ismi verilir ve aynen bir POJO gibi çok basit bir class'tır. Üzerinde her `Object []` elemanı için bir alan (property) bulunur.

Nesneler Yerine Alanlar Döndürmek

Kullanmak için HQL şöyle değişir:

```
select new vs.vs.Summary(o1.prop1, o2.alan2, o1.alan3)
from Class2 as o1 " +
left join c2.xyzList as o2 " +
left join j1.abc as o3 " +
where o3.property > :param1";
```

Summary

Summary

```
public class Summary {  
    public Summary(String s1, String s2, Integer i1) {  
        ..  
    }  
    ..  
    ..  
}
```

Nesneler Yerine Alanlar Döndürmek

- Artık sorguyu işleten `Object []` listesi yerine `Summary` listesi olacaktır.
- Bu listeyi `Summary` tipini, onun öğelerini kullanarak “tip güvenli” bir şekilde rahatça gezebilir.

Native SQL

- Hibernate tüm ihtiyacımız olan SQL'i üretiyor olsa bile
- bazen kullandığımız tabana özel bazı komutları kullanmak gerekebilir
- Mesela MySQL'de AGAINST komutu gibi.. (fulltext aramaları için kullanılır)

Native SQL

Bu durumlar için `org.hibernate.session` üzerinde `createSQLQuery` komutu kullanılır. Bonus: Geri gelen sonuçlar *hala Hibernate tarafından sanki HQL sorgusuymuş gibi map edilebilir*. Tek fark, `select` ile döndürülen sonucun `{ .. }` ile sarmalanmasıdır.

```
select {obj.*} from XYZ obj where ..
```

```
List l = hm.getSession().createSQLQuery  
    ("select {obj.*} from ... ")
```

```
.addEntity("obj", Xyz.class)  
.list();
```

Görevler

- Bire çok Car ve Garage kodlarını alın. Bir garajın birden fazla arabası olsun.
- Person adında yeni bir class ekleyin. Üzerinde `firstName`, `lastName` ve `age` öğeleri olsun.
- Car'dan Person'a bire bir ilişki oluşturun.
- Tüm garajları listeleyin

- `car.description`'ı belli bir String olan tüm *garajları* döndürün.
- `Person`'daki yaşı belli bir sayı üzerinde olan *arabaların garajlarını* listeleyin.
- Araba tanım alanı bir kelimeye “benzeyen” arabaları döndürün.
- Tüm kişilerin yaş toplamını döndürün
- En yüksek yaşa sahip kişiyi döndürün

- Criteria API kullanarak yaşı 30 olan kişileri listeleyin
- Summary kavramını kullanarak `garage.description`, `car.description`, `person.age` öğelerini döndürün, işleyecek HQL ise yaşı belli bir parametrenin üstündeki kişileri filtrelemelidir.
- MySQL'e özel LIMIT komutu sayfalama yapmaya yarar. Hibernate'in kendine has bir sayfalama yöntemi var, ama biz burada, örnek olması amacıyla sayfalamayı LIMIT ile yapacağız. Garage üzerinde bir native SELECT yapın. Burada gelen sonuçları LIMIT 10 ile 10'ar 10'ar

listeleyin.

Import'lar

```
import java.util.List;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityManager;
import javax.persistence.Persistence;
import javax.persistence.Query;
import org.hibernate.Hibernate;
import org.hibernate.ejb.HibernateEntityManager;
import org.hibernate.criterion.Expression;
import org.hibernate.Criteria;
```

```
import org.hibernate.SQLQuery;  
import org.hibernate.Session;
```

Son