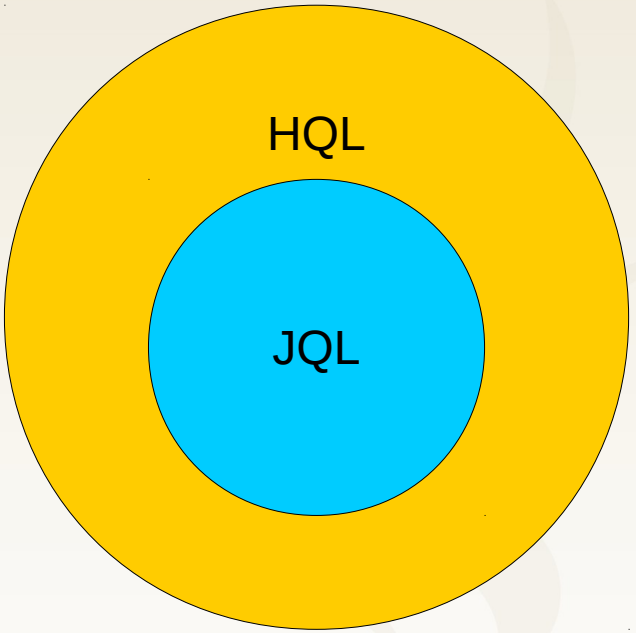
HQL/JQL İle Sorgulama

harezmi bilişim çözümleri

Sorgulamaya Giriş

- HQL, Hibernate'in Object Query Dili'dir
- JPA'nın sorgu dili **JQL**'dir
- HQL ve JQL yapı olarak **SQL'e benzer:** select...from...where
- Sorgu sonucunda dönen **persistent nesne**lerdir, persistence context (Session) ile ilişkilidirler



JQL, HQL'in bir alt kümesidir
JQL ile yapılabilen herşey ve daha fazlası HQL ile de yapılabilir

Örnek HQL/JQL Sorguları

- `from Pet`
- `from Pet as p` "as" elemanı opsiyoneldir
- `from Pet p`
- `from Person` polymorphic sorgudur
- `from Owner` sadece bu alt sınıfa ait nesneleri getirir
- `from java.lang.Object` bütün persistent nesneleri döner
- `from java.io.Serializable` bütün Serializable persistent nesneleri döner
- `from Owner o where o.email = 'ali@gmail.com'`

Örnek HQL/JQL Sorguları

- `from Owner o where o.email in ('foo@bar', 'bar@foo')`
- `from Owner o where o.email is null`
- `from Owner o where o.email is not null`
- `from Person p where p.firstName like 'G%'`
- `from Person p where p.firstName not like '%Foo B%'`
- `from Person p where p.firstName not like '\%Foo%' escape='\'`
- `from Person p order by p.firstName`
- `from Person p order by p.firstName desc`
- `from Person p order by p.lastName asc, p.firstName desc`

Örnek HQL/JQL Sorguları

- `from Person p where p.firstName like 'G%' and p.lastName like 'K%'`
- `from Owner o where (o.firstName like 'G%' and o.lastName like 'K%') or o.email in ('foo@bar', 'bar@foo')`
- `from Owner o where lower(o.email) = 'foo@hibernate.org'`
- `from Person p where concat(p.firstName, p.lastName) like 'G % K%'`
- `from Pet p where p.visits is not empty`
- `from Pet p where size(p.visits) > 0`
- `from Pet p, Owner o where p member of o.pets and p.name like '%Kitty%'`

Örnek HQL/JQL Sorguları

- `from Item i where i.price between 1 and 100`
- `from Item i where i.price > 100`
- `from Item i where ( i.price / 0.71 ) - 100.0 > 0.0`

Sorgunun Çalıştırılması

```
Query query = session.createQuery("from Pet p
order by p.name desc");

List pets = query.list();
```

Session.createQuery metodu Query nesnesi dönmektedir

Dönen Query nesnesinin list() metodu ile sorgu çalıştırılır. JPA'da karşılığı **getResultList()** metodudur

```
Pet pet = (Pet) session.createQuery("from Pet p where
p.id = 1").uniqueResult();
```

Eğer sorgu birden fazla kayıt dönerse exception fırlatır
Hiç kayıt dönmez ise NULL döner. JPA'da karşılığı **getSingleResult()** metodudur

Sorgunun Çalıştırılması (JPA)

```
Query query = entityManager.createQuery("from Pet p
order by p.name desc");

List pets = query.getResultList();
```

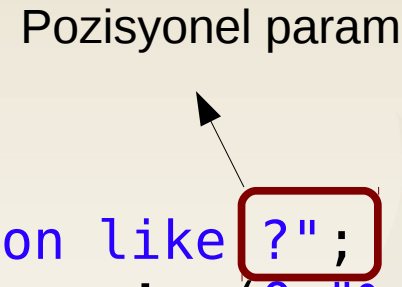
```
Pet pet = (Pet) entityManager.createQuery("from Pet p
where p.id = 1").getSingleResult();
```

Parametrelerin Bind Edilmesi

JAVA Eğitimleri

- Sorgu parametreleri **pozisyonel** ve **isimlendirilmiş** biçimde verilebilir
- Pozisyonel parametreler **soru işareti** ile belirtilirler
- Pozisyonel parametreler Hibernate'de 0'dan, JPA'da ise 1'den başlar
- JQL'de aynı zamanda **her soru işaretinin yanına parametrenin index'ini** koymak gerekmektedir

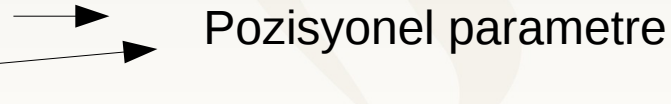
Parametrelerin Bind Edilmesi




```
String hql = "from Visit v where v.description like ?";
List result = session.createQuery(hql).setParameter(0,"%foo
%").list();
```

```
String jpql = "from Visit visit"
    + " where visit.description like ?1"
    + " and visit.date > ?2";

List result = entityManager.createQuery(jpql)
                .setParameter(1, searchString)
                .setParameter(2, mDate).getResultList();
```



Parametrelerin Bind Edilmesi

İsimlendirilmiş parametre

```
String hql =
        "from Visit v where v.description like :desc";
List result = session.createQuery(hql)
                    .setParameter("desc","%foo%").list();
```

```
String jpql = "from Visit visit"
    + " where visit.description like :search"
    + " and visit.date > :minDate";

List result = entityManager.createQuery(jpql)
                    .setParameter("search", searchString)
                    .setParameter("minDate", mDate)
                            .getResultList();
```

İsimlendirilmiş parametre

İlişkilerin Join Edilmesi

- HQL ve JQL farklı türde join yöntemlerini desteklemektedir
  - Explicit join
  - Implicit association join
  - Kartezyen çarpım
  - Ad-hoc join (sadece Hibernate)

İlişkilerin Join Edilmesi

- **Explicit** join
  - [inner] join
    - from Owner o **inner join** o.pets
  - left [outer] join
    - from Owner o **left outer join** o.pets
  - right [outer] join
    - from Owner o **right outer join** o.pets
  - full join
    - from Owner o **full join** o.pets

İlişkilerin Join Edilmesi

- **Implicit association** join
  - Sadece M:1 ve 1:1 ilişkiler için geçerlidir
  - from Pet p where **p.type.name** = ?
- **Kartezyen çarpım**
  - cross join olarak bilinir
  - from Owner o, Pet p
  - Aralarında FK ilişkisi olmayan iki tablo arasında da gerçekleşebilir (theta-style)
  - from User u, AuditLog a where u.email = log.username

İlişkilerin Join Edilmesi

- Ad-hoc join (**sadece Hibernate**)
  - Birbiri ile **ilişkisi olmayan entity'ler** arasında JOIN condition belirterek inner veya outer join yapılabilir
  - from User u **join** AuditLog a **on** u.email = a.username
  - from User u **left join** AuditLog a **on** u.email = a.username

İlişkilerin Join Edilmesi

OWNERS

OWNER_ID	OWNER_NAME
1	Owner 1
2	Owner 2
3	Owner 3

PETS

PET_ID	OWNER_ID	PET_NAME
1	1	Pet 1
2	1	Pet 2
3	1	Pet 3
4	2	Pet 4
5	NULL	Pet 5

İlişkilerin Join Edilmesi

INNER JOIN

OWNER_ID	OWNER_NAME	PET_ID	PET_NAME
1	Owner 1	1	Pet 1
1	Owner 1	2	Pet 2
1	Owner 1	3	Pet 3
2	Owner 2	4	Pet 4

İlişkilerin Join Edilmesi

LEFT OUTER JOIN

OWNER_ID	OWNER_NAME	PET_ID	OWNER_ID	PET_NAME
1	Owner 1	1	1	Pet 1
1	Owner 1	2	1	Pet 2
1	Owner 1	3	1	Pet 3
2	Owner 2	4	2	Pet 4
3	Owner 3	NULL	NULL	NULL

İlişkilerin Join Edilmesi

JAVA Eğitimleri

RIGHT OUTER JOIN

PET_ID	OWNER_ID	PET_NAME	OWNER_ID	OWNER_NAME
1	1	Pet 1	1	Owner 1
2	1	Pet 2	1	Owner 1
3	1	Pet 3	1	Owner 1
4	2	Pet 4	2	Owner 2
5	NULL	Pet 5	NULL	NULL

İlişkilerin Join Edilmesi

FULL OUTER JOIN

OWNER_ID	OWNER_NAME	PET_ID	OWNER_ID	PET_NAME
1	Owner 1	1	1	Pet 1
1	Owner 1	2	1	Pet 2
1	Owner 1	3	1	Pet 3
2	Owner 2	4	2	Pet 4
3	Owner 3	NULL	NULL	NULL
NULL	NULL	5	NULL	Pet 5

İlişkilerin Join Edilmesi

JAVA Eğitimleri

CROSS JOIN

OWNER_ID	OWNER_NAME	PET_ID	OWNER_ID	PET_NAME
1	Owner 1	1	1	Pet 1
1	Owner 1	2	1	Pet 2
1	Owner 1	3	1	Pet 3
1	Owner 1	4	2	Pet 4
1	Owner 1	5	NULL	Pet 5
2	Owner 2	1	1	Pet 1
2	Owner 2	2	1	Pet 2
2	Owner 2	3	1	Pet 3
2	Owner 2	4	2	Pet 4
2	Owner 2	5	NULL	Pet 5
3	Owner 3	1	1	Pet 1
3	Owner 3	2	1	Pet 2
3	Owner 3	3	1	Pet 3
3	Owner 3	4	2	Pet 4
3	Owner 3	5	NULL	Pet 5

Implicit ve Explicit Join'ler

- Implicit join'ler **her zaman M:1 veya 1:1 yönünde**dir
- ```
  from Pet pet where pet.type.name like '%Foo%'
  ```
- **Asla** collection-valued ilişki (1:M, N:M) yönünde **olmazlar**
  - **owner.pets.name** geçerli bir ifade değildir
- Explicit join'ler ise genellikle **from kısmında** yer alırlar
- ```
  from Vet v left join v.specialties s where
  s.name like '%Foo%'
  ```

Implicit ve Explicit Join'ler

- **with/on** ifadeleri ile ilave JOIN conditionlar belirtilebilir
- ```
  from Vet v left join v.specialties s with
  s.name = 'surgery' where v.firstName like '%J%'
  ```
  ```

Join İçeren Sorgu Sonuçlarının Ele Alınması

- Explicit join içeren sorguların sonucunda **List of Object[]** array dönülür
- ```
  from Vet v left join v.specialties s where s.name like '%Foo%
  ```
- Object array'in 0. index'i Vet, 1. index'i Specialty nesnesi içerir
- Eğer sorgu sonucunda Specialty nesneleri istenmiyorsa **SELECT** clause kullanılabilir
- ```
  select v from Vet v left join v.specialties s where s.name like '%Foo%
  ```

Join İçeren Sorgu Sonuçlarının Ele Alınması

- Explicit JOIN içeren sorgu sonuçlarında **tekrar eden kayıt** olma ihtimali he zaman vardır
- Tekrarlayan kayıtlar **distinct** anahtar kelimesi ile ayıklanabilir
- `select distinct v from Vet v left join v.specialties s ...`
- Diğer bir yol ise sorgu sonucunda dönen listeyi bir **LinkedHashSet** içerisine atmaktır

Sorgu Sonuçlarının Projeksiyonu

- Sorgu sonucunda **hangi property'lerin veya hangi nesnelerin dönüleceği** explicit biçimde belirtilir
- SELECT clause entity property'leri üzerinde **projeksiyon** işlemini gerçekleştirir
- ```
  select v.id, v.description from Visit v where v.date < current_date()
  ```
- Bu durumda sorgu sonucu **Object[]** içeren list şeklinde dönülür
- Ancak Object array içindeki elemanlar scalar değerlerdir (persistent entity değildir)
  ```

HQL/JQL ve Constructor İfadeleri

- Select ifadesinde sınıfın FQN'si ile başına **new operatörü** koyarak dönen sonuçlar **DTO nesnelerine** dönüştürülebilir
- Bu nesneler hiçbir şekilde **Session/EntityManager ile ilişkili değil**dir

```
Query query = session.createQuery(
    "select
        new com.javaegitimleri.petclinic.model.VisitDTO(
           v.id,
           v.description,
           v.date)
     from Visit v");

List<VisitDTO> result = query.list();
```

HQL ve ResultTransformer

- Constructor ifadelerine benzer biçimde **ResultTransformer** arayüzü ile de **sorgu sonuçlarından DTO nesneleri oluşturmak** mümkündür
- ResultTransformer kabiliyetini **sadece** Hibernate destekler

HQL ve ResultTransformer

```
Query query = session.createQuery(
     "select v.id as visitId,
             v.description as visitDescription,
             v.date as visitDate
      from Visit v");

List<VisitDTO> result = query.setResultTransformer(
          Transformers.aliasToBean(VisitDTO.class)).list();
```

```
public class VisitDTO {
      private Long visitId;
      private String visitDescription;
      private Date visitDate;

      public void setVisitId(Long visitId) {
            this.visitId = visitId;
      }

      public void setVisitDescription(String visitDescription) {
            this.visitDescription = visitDescription;
      }

      public void setVisitDate(Date visitDate) {
            this.visitDate = visitDate;
      }
}
```

AliasToBeanResultTransformer impl bir nesne oluşturur

! Sütun ve property isimleri eşleşse bile HQL içerisinde alias tanımlamak şarttır.

HQL ve ResultTransformer

- İstenirse sorgu sonuçları bir **Map**'in içerisine **key:value ikilileri** şeklinde de konabilir

```
Query query = session.createQuery(
    "select v.id as visitId,
            v.description as visitDescription,
            v.date as visitDate
     from Visit v");

List<Map<String,Object>> result = query
        .setResultTransformer(
            Transformers.ALIAS_TO_ENTITY_MAP).list();
```

```
for(Map<String,Object> row:pets) {
     long visitId = row.get("visitId");
     String visitDescription = row.get("visitDescription");
     Date visitDate = row.get("visitDate");
}
```

Select ifadesinde alias kullanılmaz ise map'in key değerleri 0,1,2 şeklinde indeks olacaktır

İsimlendirilmiş Sorgular

- ORM **Metadata içerisinde HQL/JQL** sorguları tanımlanabilir
- Bu sorgular tanımlanırken birde **isim** atanır
- Daha sonra uygulama içerisinde bu **isim kullanılarak sorguya erişilerek** çalıştırılabilir
- Bu tür sorgulara **isimlendirilmiş sorgu** denir
- Domain sınıflarında veya, **package-info.java** dosyası içerisinde tanımlanabilirler

İsimlendirilmiş Sorgular

Named query'lerin isimleri uygulama genelinde benzersiz olmalıdır

```
@NamedQueries({
     @NamedQuery(
      name = "findVisitsByDescription",
      query = "select v from Visit v where
v.description like :desc")
})
@Entity
@Table(name = "VISITS")
public class Visit {

}

session.getNamedQuery("findVisitsByDescription")
        .setParemeter("desc", description).list();
```

JPA ve Global Metadata

- İsimlendirilmiş sorgu, custom tip gibi tanımlar **global metadata** olarak adlandırılırlar
- **JPA** “**package visible**” annotasyonları **desteklemez**
- Bu nedenle JPA ile çalışırken **package-info.java** dosyası içerisinde JPA annotasyonları kullanılamaz
- JPA anotasyonları ile isimlendirilmiş sorgular vs **sadece domain sınıfları üzerinde** tanımlanabilir

Scalar Sorgular

- `select distinct v.description from Visit v`
- `select v.date, current_date() from Visit v`
- `select p.birthDate, upper(p.name) from Pet p`
- `select count(p.name) from Pet p`
- `select sum(i.price) from Item i`
- `select min(i.price), max(i.price) from Item i`
- `select count(distinct v.description) from Visit v`

Scalar Sorgular

- `select o.lastName, count(o) from Owner o group by o.lastName`
- `select i.id, avg(i.price) from Item i group by i.id`
- `select i.id, count(i), avg(i.price) from Item i group by i.id`
- `select i.id, count(i), avg(i.price) from Item i group by i.id having count(i) > 10`

Altsorgular (Subselects)

- ```
  from Pet p where (select count(v) from p.visits v where v.description is not null) > 10
  ```
- ```
  from Pet p where current_date() in (select v.date from p.visits v )
  ```
- ```
  from Vet v where :specialty in elements(v.specialties)
  ```
- ```
  from Vet v where :specialty in (select s from v.specialties s)
  ```

Identifier'ların Karşılaştırılması

- Bir identifier değerine sorgu içinden şu şekillerde erişilir
  - Ya **identifier property'nin ismi** ile
  - Ya da özel **“id” property ismi** ile
- HQL'de “id”, **farklı isimlendirilmiş identifier property**'si olabilir, ama JPA bunu desteklemez

İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com