

# Ek Konular

**Burak Bayramlı**

Bu belge,  $\text{\LaTeX}$  ile üretilmiştir

# Anahtarlar

# Anahtarlar

Anahtarlar hususunda ilginç iki konu var:

- Üretilen anahtarlar
- Bileşik anahtarlar

# Üretilen Anahtarlar

Bu anahtarlar, MySQL autoincrement ya da Oracle'da trigger'lar ile otomatik her yeni satır için **bir** arttırılan türden anahtarlardır. Hibernate bunları `@GeneratedValue` annotation'ı üzerinden kullanır.

# Üretilen Anahtarlar

@GeneratedValue'ya seçenek olarak “üretim metodu” verilir. Bu seçenekler

- AUTO
- TABLE
- IDENTITY
- SEQUENCE

# Üretilen Anahtarlar

Kullanmak için

```
@Id @GeneratedValue(strategy=GeneratorType.AUTO)
```

gibi bir kullanım yeterlidir. AUTO her tabanda kullanılacak en portable çözüm. TABLE için üretilen en son ID'yi taşıyan ayrı bir tablo yaratmanız gerekiyor. Ayrıca, kendiniz ID üretmeyi seçebilirsiniz.

```
UUID uuid = java.util.UUID.randomUUID(); uuid.toString();
```

# Bileşik Anahtarlar

## Bileşik Anahtarlar

Bazen veri tabanındaki anahtar yapısını direk koda yansıtmaya mecbur oluruz. Bu yapı da bazen bileşik (composite) olabilir. Meselâ bir Person için anahtarın isim ve soyadının birleşimi olması gibi.

# Ana class

```
@Entity
public class Class1 {
    ...
    @EmbeddedId Class1Pk id;
    ...
}
```

# Anahtar class

```
@Embeddable
public class Class1Pk implements Serializable {

    public Class1Pk(String prop1, String prop2, ..) {
        ..
        ...
    }

}
```

# İşaret eden ne olacak?

- Bileşik anahtar içeren bir objeye işaret eden nesnelerin değişmesi gerekir
- @JoinColumn ibareleri, bu çoklu kolon yapısını yansıtmalıdır.

# İşaret eden ne olacak?

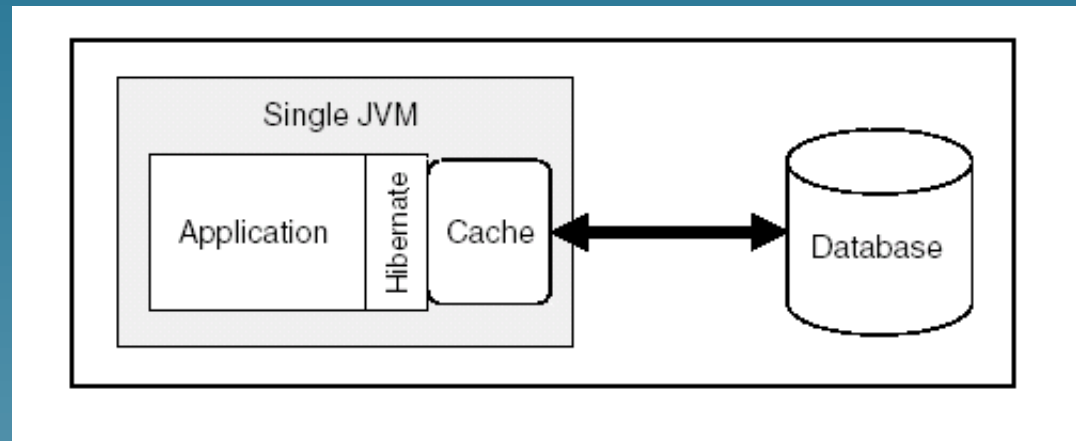
```
@Entity
public class Xyz implements Serializable {
    ...
    @OneToOne
    @JoinColumns({
        @JoinColumn(name="alan1",
                    referencedColumnName="prop1"),
        @JoinColumn(name="alan2",
                    referencedColumnName="prop2")})
    Class1 obj;
```

# İşaret eden ne olacak?

Burada

- alan1 işaret edenin bileşik foreign anahtarının birinci parçası
- alan2 işaret edenin bileşik foreign anahtarının ikinci parçası
- prop1 ve prop2 Pk class'ındaki öğelerin isimleri

# Caching



# Caching

Hibernate ile kullanılabilen birçok caching paketi var.

- TreeCache
- EhCache
- OSCache
- HashtableCache

# Caching

Bunlardan birini seçmek için `persistence.xml` içinde `hibernate.cache.provider_class` değeri set edilmeli.  
Meselâ

```
<property name="hibernate.cache.provider_class"  
    value="org.hibernate.cache.HashtableCacheProvider"/>
```

# Caching

Daha sonra hangi caching paketi seçildiyse, o paketin kendine has ayar dosyası üzerinde cache'in bazı özellikleri ayarlanabilir. Cache ne kadar büyük, kaç tane obje taşıyabilir, bu objeleri ne kadar zaman tutar (time out değeri) gibi. Meselâ TreeCache için bu ayarlar bir `-service.xml` dosyasından verilir.

# Obje Önbellesi

Şimdi bir nesneyi önbellesine atmak için

```
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
@Entity
public class Xyz { .. }
```

## Obje Önbelleği

Bundan sonra, `find` ile okunan nesne önbelleğe alınır. Takibindeki tüm `find`'lar önbelleğe gider.

Eğer biri, `find` ile alınan ID'deki objeye günceller ise, önbellekteki nesne silinir. Bundan sonraki ilk `find` tekrar veri tabanına gidecektir.

# Obje Önbelleđi

Obje önbelleđini kullanmak için hiçbir ek API çağrısı gerekmiyor.

# Sorgu Önbelleği

Sorgu önbelleği (query cache), HQL ile işlettiğiniz sorguların sonuçlarının önbelleğe atılmasını sağlar. Eğer sorgu yapılıp önbelleğe atıldıktan sonra o sorguyu ilgilendiren herhangi bir tabloya *tek* bir INSERT yapıldığı anda, o sorgunun cache'i invalidate olur.

# Sorgu Önbelleği

Kullanmak için `persistence.xml` dosyanıza

```
<property name="hibernate.cache.use_query_cache" value="true" />
```

ibaresini ekleyin.

# Sorgu Önbelleği

Sorgu önbelleğini kullanmak için bazı ek çağrılar gerekli. Meselâ `query` objeniz üzerinde, listeyi almadan önce, `setHint("org.hibernate.cacheable", true)` çağrısını yapmalısınız.

## 2. Seviye Önbellek

.. uygulama çapında, tüm JVM'e hitap eden bir önbellektir. Yâni, birden fazla EntityManager *aynı* önbellek ile konuşabilir. Bu demektir ki, bir Web ortamında, kullanıcı 1 ile 2 arasında önbellek paylaşılmaktadır, ki bu istediğimiz bir özelliktir.

# Stored Procedure Çağırma

# Stored Procedure Çağırma

- Eski (legacy) sistemlerimizde bazen stored procedure olabiliyor
- Neredeyse her büyük DB paketi SP destekliyor, MySQL 5'ten itibaren bunu desteklemeye başladı.
- Bu ihtiyaç sebebiyle Hibernate içinde de SP çağırma desteği var.

# Stored Procedure Çağırma

Örnek olarak ABC tablosu üzerine tanımlanmış `sp_get_abc` adlı bir SP görelim. MySQL 5'te bunu

```
create procedure sp_get_abc(in param1 type)
SELECT * from ABC where kolon1 = filterparam;
```

olarak yapabiliriz. Bu proc, bir filtre alıp geriye satır(lar)ın tamamını döndürüyor. `type`, `integer`, `varchar` vs olabilir.

# Stored Procedure Çağırma

Not: Eğer MySQL Front procedure yaratmakta problem çıkartıyorsa, <MYSQL>/bin dizininizi PATH'e ekleyin, ve komut satırından `mysql -u root` diyerek SQL komut satırından procedure yaratma komutlarını girin.

Procedure'ı test etmek için `call sp_get_abc('..')` kullanımı yeterli.

# Stored Procedure Çağırma

Hibernate ile bu proc'u çağırma için önce bir "isimlendirilmiş sorgu (named query)" tanımı yapmamız lazım. Bu da, Hibernate dünyasında herşeyin olduğu gibi yeni bir annotation üzerinden olacak.

# Sorgu Tanımlamak

```
@Entity
@NamedNativeQueries( {
    @javax.persistence.NamedNativeQuery
        (name="sp_get_abc",
         query="call sp_get_abc(:filtre)",
         resultClass = Abc.class
        ) }
)
public class Abc implements Serializable {
    ...
}
```

}

## Stored Procedure Çağırma

Bir proc'un çıktısını, eğer önceden map edilmiş bir class ile uyumlu ise, o objenin listesi olarak almak mümkün. Daha önceki örneklerimizi hatırlayın, `List<Car>` meselâ. Burada da aynısı mümkün olacak.

# Sorguyu Çağırarak

```
HibernateEntityManager hem = (HibernateEntityManager)em;  
org.hibernate.Query q =  
    hem.getSession().getNamedQuery("sp_get_abc");  
q.setInteger("filtre","test");  
List l = q.list();  
...
```

## Görevler

- SimpleHibernate projesini bir kopyasını çıkartın. Yeni bir @Id'yi üretilen türden bir ID'ye çevirin. Bunu yapınca artık bu değeri dışarıdan sizin set etmenize gerek kalmayacak. Bu şekildeki Car class'ınızı tabana ekleyin.
- SimpleHibernate projesinin yine bir kopyasını çıkartın. Bu projede Car class'ını “cache edilir” hale getirin. @Cache ile işaretleyin, persistence.xml değişikliklerini

yapın. Sonra, Aynı `EntityManagerFactory`'den *farklı* `EntityManager`'lar alarak, aynı test metodu içinde *aynı* nesneyi birkaç kez okuyun. Ekranda kaç tane SQL görüyorsunuz? Şimdi `Car` class'ının `@Cache` ibaresini comment edin. Aynı test'i tekrar işletin. Sonuç değişti mi?

- Test programınıza çok basit bir sorgu ekleyin. Bu sorguyu da `setHint` ifadesi ile önbelleğe attırın. Bu sorguyu farklı `EntityManager`'lara işlettirin. Sonuca bakın. Şimdi `setHint` ifadesini kapatın. Testi tekrar işletin.

- `Person` class'ına `firstName` ve `lastName`'den oluşan bir bileşik anahtar verin. `Car` class'ının `Person`'a bire bir bağlantısı olsun.
- `sp_get_person` adında bir stored procedure yazın. Bu procedure bir yaş (`age`) eşik değeri alsın ve sadece o yaştan büyük olan `Person`'ları listelesin. yani `NamedQuery`'iniz `Person` üzerinde tanımlanacak ve onu map edecek. Sonuçları `List` içinde `Person` objeleri olarak alabilmeniz lazım.

# Import'lar

```
import javax.persistence.Entity;  
import javax.persistence.Column;  
import javax.persistence.Id;  
import org.hibernate.annotations.Cache;  
import org.hibernate.annotations.CacheConcurrencyStrategy;
```

# Import'lar

```
import java.io.Serializable;  
import javax.persistence.IdClass;  
import javax.persistence.Id;  
import javax.persistence.Entity;  
import javax.persistence.EmbeddedId;
```

# Import'lar

```
import java.io.Serializable;  
import javax.persistence.Embeddable;  
import javax.persistence.Column;
```

# Import'lar

```
import javax.persistence.Persistence;  
import javax.persistence.EntityManagerFactory;  
import javax.persistence.EntityManager;  
import javax.persistence.EntityTransaction;  
import org.testng.annotations.Test;  
import javax.persistence.Query;  
import org.apache.log4j.BasicConfigurator;  
import org.hibernate.ejb.HibernateEntityManager;  
import java.util.List;
```

# Son