Hibernate ve Veritabanı Constraint Tanımları

harezmi bilişim çözümleri

Hibernate ve Veritabanı Constraint Tanımları

JAVA Eğitimleri

- ORM mapping metadata vasıtası ile veritabanı şeması üzerinde **çeşitli düzeylerde** (sütun, satır, tablo vb) **kısıtlar** tanımlanabilir
- Constraint tanımları sadece **şema DDL** ifadelerinin üretilmesinde faydalıdır
- Bu sayede Hibernate yardımı ile geliştirme veya test ortamlarında **bütün şema otomatik biçimde** constraintleri ile beraber yaratılabilir
- **Runtime**'da bir fonksiyon icra etmezler

Constraint Türleri

- **Sütun** kısıtları
  - Sütunun veri tipi ile, NOT NULL özel bir kısıt
- **Satır** kısıtları
  - Bitiş tarihi > başlangıç tarihi tek satırı bağlar
- **Tablo** kısıtları
  - Bir veya daha fazla kaydı etkiler
  - UNIQUE birden fazla kaydı etkiler
- **Veritabanı** kısıtları
  - Kuralın birden fazla tabloyu etkilemesidir
  - FK yaygın bir veritabanı kısıtıdır

Sütun Düzeyinde Constraint Eklenmesi

```
@Column(name = "FIRST_NAME", length = 16,
        nullable = false, unique = true)
```

Length, nullable, unique hepsi sütun düzeyinde constraint demektir

```
@org.hibernate.annotations.Check(constraints =
"regexp_like(FIRST_NAME,'^[[:alpha:]]+$')")
private String firstName;
```

```
create table PERSONS (
    …
    FIRST_NAME varchar(16) not null
        check(regexp_like(FIRST_NAME,'^[[:alpha:]]+$')),
    …
);
```

```
<property name="firstName" type="string">
    <column name="FIRST_NAME" check="regexp_like(FIRST_NAME,'^[[:alpha:]]+$')"/>
</property>
```

Satır Düzeyinde Constraint Eklenmesi

```
@Entity
@org.hibernate.annotations.Check(constraints="START_DATE < END_DATE")
public class Meeting {
//...
}
```

Birden fazla sütun'un yer aldığı constraint'lerdir

```
create table MEETING (
    …
    START_DATE timestamp not null,
    END_DATE timestamp not null,
    …
                    check (START_DATE < END_DATE));
```

```
<class name="com.javaegitimleri.petclinic.model.Meeting" table="MEETING"
                                                    check="START_DATE < END_DATE">
</class>
```

Tablo Düzeyinde Constraint Tanımı

```
@Entity
@Table(name = "CATEGORY",
   uniqueConstraints = {
        @UniqueConstraint(columnNames ={"CAT_NAME", "PARENT_CATEGORY_ID"} )
   })
public class Category {
//...
}
```

```
create table CATEGORY (
    …
    CAT_NAME varchar(255) not null,
    PARENT_CATEGORY_ID integer,
    …
                unique (CAT_NAME, PARENT_CATEGORY_ID)
);
```

```
<class name="com.javaegitimleri.petclinic.model.Category" table="CATEGORY">
    <id name="id" column="ID"/>
    <properties name="cat_name_parent_cat_id" unique="true">
        <property name="name" column="CAT_NAME"/>
        <many-to-one name="parentCategory" column="PARENT_CATEGORY_ID"/>
    </properties>
    ...
</class>
```

Tablo Düzeyinde Constraint Tanımı

```
@Entity
@Table(name="ITEMS", indexes = @Index(
      name = "IDX_INITIAL_PRICE_CURRENCY",
      columnList = { "INITIAL_PRICE", "INITIAL_CURRENCY" }    ))
public class Item {
    @Column(name = "INITIAL_PRICE")
    private BigDecimal price;

    @Column(name = "INITIAL_CURRENCY")
    private Currency currency;
}
```

JPA 2.1'de @**Table** içerisinde kullanılan @**Index** annotasyonu gelmiştir. Bu annotasyon ile bir tablonun bütün index'leri tek bir yerde tanımlanmaktadır.

```
create index IDX_INITIAL_PRICE_CURRENCY on
ITEMS (INITIAL_PRICE,INITIAL_CURRENCY);
```

Veritabanı Düzeyinde Constraint Tanımı

JAVA Eğitimleri

```
@Entity
public class Pet {
    @ManyToOne
    @JoinColumn(name = "OWNER_ID",
            foreignKey = @ForeignKey(name = "FK_PET_OWNER_ID"))
    private Owner owner;
}

@Entity
public class Vet {
    @ManyToMany
    @JoinTable(name = "T_VET_SPECIALTY",
            foreignKey = @ForeignKey(name = "FK_VET_ID"),
inverseForeignKey = @ForeignKey(name = "FK_SPECIALTY_ID"))

    private Set<Specialty> specialties =new HashSet<Specialty>();

}
```

JPA 2.1'de @JoinColumn ve @JoinTable içerisinde kullanılan @ForeignKey annotasyonu gelmiştir. Ancak bununla ilgili Hibernate 4'de bazı bug'lar mevcuttur. Örneğin M:N de tanımlanan FK'lar dikkate alınmamaktadır. Hibernate 5'de bu bug fixlenmiştir

Collection İlişkileri ve Cascade On Delete

```java
@Entity
public class Owner {

  @OneToMany
  @JoinColumn(name = "OWNER_ID")
  @org.hibernate.annotations.OnDelete(OnDelete.CASCADE)
  private Set<Pet> pets = new HashSet <Pet>();

}
```

Herhangi bir Owner silindiği vakit ona bağlı Pet'lerin de ilgili Tablodan SQL DELETE CASCADE ile silinmesini sağlar

```java
@Entity
public class Pet {

}
```

```
alter table T_PET add constraint
FK_h14un5v94coafqonc6medfpv8 foreign key (owner_id)
references T_OWNER on delete cascade
```

İletişim

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