# **Hibernate Advanced Mapping**

## **Relationship**

### One To one

Bire bir ilişkidir, bir tabloda bulunan değere karşılık ilişkili olduğu diğer tabloda da yalnızca bir karşılığı olmalıdır. Aynı veri iki defa bulunamaz.

### One To Many

Bir tabloda bulunan bir değer, ilişkili olduğu diğer tabloda birden fazla kez bulunabilir.

### Many To Many

Bir tabloda bulunan değer ilişkili olduğu tabloda birden fazla kez bulunabilir aynı şekilde karşı tablodaki bir değer de ilgili tabloda birden çok defa bulunabilir.

## **Fetch Types**

### Eager

İlişkili tablolardan birisine ait nesneyi aldığımızda ilişkili olduğu tablolarda ki tüm verilerin alınmasıdır, örneğin Egitmen ve Kurs isimli iki tablo olduğunu varsayalım, bir eğitmenin birden fazla kursu olabilir, bu durumda bir eğitmen nesnesi aldığımızda kurs tablosundan o eğitmenin kurslarının tamamını fetch etmek istersek eager kullanmamız gerekir.

### Lazy

Aynı şekilde iki tablo arasındaki ilişkide bir tablonun nesnesini aldığımızda diğer tablodan hemen bütün verileri almak istemiyorsak istek üzerine almak istiyorsak lazy bağlantıyı kullanmamız gerekir.

## Uni-Directional

Bu bağlantı şeklinde tablolar arası ilişki tek yönlüdür, örneğin eğitmen tablosundan kurs tablosuna bir ilişki vardır, fakat kurs tablosunda eğitmene ulaşılabilecek bir referans yoktur.

## Bi-Directional

Bu bağlantı şeklinde iki tabloda da diğer tablo için bir referans değeri vardır ve herhangi birinden diğerine erişilebilir.

## **Entity Lifecycle**

### New Transient

Bir entity nesnesi oluşturulduğunda durumu **new transient**dir. New transient statede olan nesnelerin veritabanında herhangi bir kayıt karşılığı yoktur ve bu nesneler Persistence Context ile ilişkili değildir.

final MyObject myObject = new MyObject();

### Managed

Veritabanı karşılığı olan, aktif sessionda mevcut olan ve Persistence Context tarafından yönetilen entityler bu statededir. Managed statedeki entitylerin state değişikliği Persistence Context açık olduğu müddetçe takip edilir ve bu değişiklikler veritabanına belirli aşamada otomatik olarak yansıtılır.

### Detach

Veritabanı karşılığı olan fakat Persistence Context’de olmayan entitylerin statei detachdir. Persistence Context kapatıldığında managed statedeki entitylerin statei detach state olur. Detach statedeki entityler üzerinde yapılan değişiklikleri Hibernate takip etmez.

### Removed

Managed veya detached statedeki entityler silindiğinde, removed state’e geçer.

## **On To One Cascade Types**

### PERSIST

İlişki persist olarak tanımlandıysa kaydedilen nesnenin alt nesneside kaydedilir.

### MERGE

İlişki merge olarak tanımlandıysa nesne merge edilirse ilişkili olan nesne de merge edilir.

### REMOVE

İlişki remove olarak tanımlandıysa nesne silinirse ilişkili olan nesne de silinir.

### REFRESH

İlişki refresh olarak tanımlandıysa nesne yenilenirse bağlı olan nesne de yenilenir.

### ALL

İlişki all olarak tanımlandıysa yukarıdaki tüm işlemler için birlikte yapılır.

## **One To Many**

Bu ilişki türünde, bir nesnenin bağlı olduğu birden çok nesne olabilir. Örneğin bir eğitmen birden çok ders veriyor olabilir. Bu bağlantı çift yönlü de olabilir, örneğin bir eğitmenin birden fazla kursun eğitmeni olduğu durumda, birden fazla kurs tek bir eğitmenle ilişkili olmaktadır. Bu ilişki türüne On-To-Many ilişki denir.

### FetchTypes

İlişkili tablolarda, bir tabloyla ilgili işlem yaptığımızda onunla ilişkili diğer tablo/tablolarla ilgili nasıl bir işlem yapacağımızı belirlediğimiz ayardır.



#### Eager

Bu ayar, ilişkili tablolar üzerinde işlem yaparken bir tabloya ait bilgilere erişmek istendiğinde onunla ilişkili olan diğer tabloyu da veri tabanından çekmek ister. Örneğin kurs ve öğrenci tablosunun ilişkili olduğunu varsayalım, eğer bir kurs adını aratmak istersek kurslarla birlikte kursu alan tüm öğrenciler de veri tabanından çekilir, bu durum performansın düşmesine sebep olur.

#### Lazy

Genel kullanım lazydir çünkü performans sorunları yaşamaya sebep olmaz, yukarıdaki örnekteki gibi bir kursu arattığımızda veri tabanından yalnızca kurs bilgilerini çeker, kurs ile ilişkili öğrencilerin bilgilerini de çekmek, eğer kursu alan öğrencilere de ihtiyaç duyup böyle bir istekre bulunursak ancak o zaman veri tabanından öğrenci bilgilerini çeker.

## TypedQuery

Generic bir yapıdır, içerisine hangi entity üzerinde çalışacaksak onu vermemiz gerekir. Query yazarken entityManager kullanılır, entityManager.createQuery() metodu içerisine entity adlarını kullanarak (veritabanındaki tablo adı değil) sorgumuzu yazarız, fakat başına select \* koymayız yalnızca from ve sonraki kısımlar yazılır. Bu metod iki parametre alır biri sorgu diğeri ise hangi entity için çalışacağıdır. Sorguda eğer istersek değişken kullanabiliriz, bunun için :degiskenAdi şeklinde önüne : koyarak query içerisine yazarız. Query yazdıktan sonra query.setParameter() metodu ile değişkene değer atarız. Bu metod iki parametre alır, ilki query içerisinde yazdığımız değişkenin adı, buraya yazarken direkt değişken adını tırnak içinde yazarız, : kullanmayız. İkinci parametre ise bu değişkenin değeridir. Ve son olarak sorguyu çalıştırmak için query.getResaultList() metodunu çağırırız ve bize bir liste döner.

Bunu lazy bağlantıda ilişkili nesneleri çağırmak için ekstra bir tetikleyici olarak kullanabiliriz.

### Join Fetch

Bu kullanım, yukarıdaki işlemi daha kısa yoldan yapmak için, yani birbiriyle lazy ilişkili iki entity için gerektiğinde eager ilişki kurmanın yolu join kullanmaktır. Join ile yazdığımız sorgunun bize iki tablonun birleşimini dönmesini sağlarız. Örneğin:

select i from Instructor i

JOIN FETCH i.courses

Where i.id=:data

Bu sorguyu yazdığımızda, data değişkenine verdiğimiz değer ile Instructor tablosundaki idleri karşılaştırır, doğru değeri bulduğunda ona ait olan courses tablosundaki değerleri de alır ve hepsini birleştirerek bize kurslarıyla birlikte eğitmen döner, burada where koşulunu kaldırarak birden çok eğitmen de elde edilebilir.

## Many To Many

Many to many ilişkide, tablolar arası çoktan çoka bir ilişki vardır, örneğin bir kursu birden ok öğrenci alabileceği gibi bir öğrenci birden çok kurs alabilir, yani bir kursun birçok öğrencisi olur, bir öğrenci birçek kursa kayıt olabilir. Bu tür ilişkiler çoka çok ilişkilerdir.

### Join table

Bu tablolar çoka çok ilişkiyle birbirine bağlı olan tablolar için birer yabancı anahtar(foreign key) içermektedir, böyle aradaki ilişki tanımlanmış olur.