# HİBERNATE DERS NOTLARI

1. JDBC, bir istemcinin bir veritabanına nasıl erişebileceğini tanımlayan bir API'dir; Hibernate, nesneye yönelik alan modelini geleneksel bir ilişkisel veritabanına eşlemek için bir çerçeve sağlayan bir ORM kitaplığıdır.

2. \* ***ORM*** : İlişkisel veritabanında yer alan verilerin nesne yönelimli diller içerisinde nesne olarak kullanılmasına olanak tanıyan bir teknolojidir.

3. JDBC, çeşitli uygulamaların varolmasına ve aynı uygulama tarafından kullanılmasına izin verir; Hibernate, Java sınıflarından veritabanı tablolarına eşleme yapar.

4. Hibernate bir JPA uygulamasıdır.

5.Hibernate -> HQL JDBC -> SQL demektir.

\*Hibernate kullanabilmek için öncelikle şu sitenin sağından indirmemiz lazım.

https://hibernate.org/orm/releases/5.4/

Daha sonra indirdirğimiz dosyada hibernate-release-5.4.27.Final\lib\required klasörünün içine girip içindekilerle beraber mysql connector jarıda projemize build path\add jar yoluyla inject etmeliyiz.Aynı şekilde içindeki kodların aşağıda yazılı olduğu hibernate.cfg.xml(googleye hibernate configuration xml yazınca çıkıyor) dosyasınıda projenin src sine eklemeliyiz.

Sol tarafta resources klasörürün içine META-INF isimli folder açıyoruz.Ve hibernate xml persistence xml gibi konfigürasyon dosyalarını bunun içinde tutuyoruz.

Engin hoca session,ferhat hoca entitymanager gibi terimler kullandılar.Aşağıda göreceksin ikisininde kullanımını.Peki farkları ne?Session bir hibernate terimidir.Entity ise JPA.Ve Hibernate JPA’nın altında olduğunu düşünürsek Entity sessionu kapsar.Şayet eclipseLink yada toplink gibi JPA’nın farklı apilerine sonradan geçilecekse Entity kullanmalısın.Sen Entity kullan.

<property name="hibernate.hbm2dll.auto" value="create"/> persistence.xml’deki create kodu çok önemli,eğer veri güncelleme yaparken value update değil create ise veri tabanındaki bütün verileri siler yalnızca güncellenen veriyi ekler.

\*Transaction diye bir şey kullanacağız bunun mantığı şudur:Try bloğu içine şu aşağıdaki kodu yazacağız ilkini tryin başına yani Transaction başlat diğerini tryin sonuna yani işleme sok(commit) anlamına geliyor.Ve bu iki kod arasında herhangi bir yerde problem çıkarsa bu iki kod arasındaki hiçbir şey çalışmaz,hiçbir kod run olmaz.

Session.beginTransaction(); ve session.getTransaction().commit();

**ENGİN HOCA------------------------------------------------**

**HİBERNATE.CFG.XML DOSYASININ İÇİ**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/world?useSSL=false&amp;serverTimezone=UTC</property> //world falan hangi database’i kullanıyosak onu yazmalıyız

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">1234</property>

<property name="hibernate.connection.pool\_size">1</property>

<property name="hibernate.current\_session\_context\_class">thread</property>

<property name="hibernate.show\_sql">true</property>

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

</session-factory>

</hibernate-configuration>

Örnek bir uygulama:

**City Classı**

**package** com.hibernate;

**import** javax.persistence.\*;

@Entity

@Table(name="city") //Databasenin içindeki hangi tablo olduğu

**public** **class** city {

@Id

@Column(name="ID") //Databasenin içindeki kolon isimleri

**private** **int** ID;

@Column(name="Name")

**private** String name;

@Column(name="CountryCode")

**private** String countryCode;

@Column(name="District")

**private** String district;

@Column(name="Population")

**private** **int** population;

<<<-! Burdan aşağısında parametreli parametresiz constructor,set-get metodları ve to string metodu var ->>>

**Main Classı**

**public** **static** **void** **main**(String[] args) {

SessionFactory **factory** = **new** Configuration().

configure("hibernate.cfg.xml"). //hibernate config bağlamayı unutma

addAnnotatedClass(city.**class**). //city classını vermeyi unutma

buildSessionFactory();

Session **session** = factory.getCurrentSession();

//Buraya kadar olan kısım ve alttaki List cities-iteratör dışındaki kalan kısımlar standart yazılıyor.

**try** {

session.beginTransaction();

List<city> **cities** = session.createQuery("from city").getResultList();

**for**(city **city** : cities) {

System.**out**.println(city.getName());

}

session.getTransaction().commit();

} **finally** {

factory.close(); //Factory’i kapatmayı unutma } }

**Hibernate ile Koşullu Sorgular**

**public** **static** **void** **main**(String[] args) {

SessionFactory **factory** = **new** Configuration().

configure("hibernate.cfg.xml").

addAnnotatedClass(city.**class**).

buildSessionFactory();

Session **session** = factory.getCurrentSession();

//Buraya kadar standart zaten

**try** {

session.beginTransaction(); //Session Başlat

List<city> **cities** = session.createQuery("from city where CountryCode='TUR'").getResultList(); //CountryCode u TUR olanlar

List<city> **cities1** = session.createQuery("from city where CountryCode='TUR' OR CountryCode='USA'").getResultList(); //CountryCode u TUR yada USA olanlar çünkü OR kullandık

List<city> **cities2** = session.createQuery("from city where CountryCode='TUR' AND District='Ankara'").getResultList(); //CountryCode u TUR olanlar ve District’i Ankara olanlar çünkü and kullandık

List<city> **cities3** = session.createQuery("from city where name LIKE '%kar%'").getResultList(); //city tablosundan içinde kar geçen herşey

List<city> **cities4** = session.createQuery("from city where name LIKE '%kar'").getResultList(); //city tablosundan sonunda kar geçen herşey

List<city> **cities5** = session.createQuery("from city where name LIKE 'kar%'").getResultList(); //city tablosundan başında kar geçen herşey

**for**(city **city** : cities4) {

System.**out**.println(city.getName());

}

session.getTransaction().commit();

}

**finally** {

factory.close(); //factory kapatmayı unutma

}

**Hibernate ile Order By Kulanımı**

**Üstteki kodlarda List<city> ile başlayan yere sadece alttaki kodu yazdık.**

List<city> **cities** = session.createQuery("FROM city ORDER BY name DESC"). getResultList();

//City’deki bütün verileri DESC yani Sondan Başa sıralamasıyla getir.Eğer ASC yazsaydık Baştan Sona getirirdi

\*ASC = Ascending

\*DESC = Descending

**Hibernate ile Group By Kullanımı**

**try** {

session.beginTransaction();

List<String> **countryCodes** = session.createQuery("Select countryCode FROM city Group BY countryCode").getResultList(); //Listeyi String’e çevirdik çünkü

**for**(String **countryCode** : countryCodes) {

System.**out**.println(countryCode);

}

session.getTransaction().commit();

}

**Hibernate ile Crud İşlemler**

INSERT İSLEMLERİ

city city = new city();

city.setCountryCode("TUR");

city.setDistrict("DoguAnadolu");

city.setName("URFASAN");

city.setPopulation(1);

System.out.println("Ekleme Başarılı");

session.save(city); //kaydetmeyi unutma

UPDATE İSLEMLERİ

Sehrin ilk ismi SametCity olsun.

city city = session.get(city.class,4085); //Veritabanındaki 4085 ID’li veri

city.setName("SametSehir");

session.save(city); //kaydetmeyi unutma

DELETE İSLEMLERİ

city city = session.get(city.class,4000); //Veritabanındaki 4000 ID’li veri

session.delete(city); //Session.save(kaydetme yok bunda ona göre)

**FERHAT HOCA-----------------------------------------------**

NOT=Aşağıdaki persistence.xml’de tablo oluştururken hata çıkıyorsa.Dialecti sil yada persistence.xml2 yi kullan.

<property name="hibernate.hbm2dll.auto" value="create"/> persistence.xml’deki create kodu çok önemli,eğer veri güncelleme yaparken value update değil create ise veri tabanındaki bütün verileri siler yalnızca güncellenen veriyi ekler.

Persistence.xml1

<?xml version="1.0" encoding="UTF-8"?>

<persistence version="2.1"

xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence\_2\_1.xsd">

<persistence-unit name="TestPersistence" transaction-type="RESOURCE\_LOCAL">

<class>com.project.model.Personel</class> <!--Hangi classtan sql’de tablo yapacaksak onu buraya kesinlikle yazıyoruz. -->

<properties>

<property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/project" />

<property name="javax.persistence.jdbc.user" value="root" />

<property name="javax.persistence.jdbc.password" value="1234" />

<!--Dialect :Uygulamamızın veritabanı ile hangi veritabanı dilinde konuşacağı bilgisini gösterir.

Örnek verirsek uygulama üzerinden MSSQL veritabanına bağlanıyorsak , MSSQL betikleri(script) kullanmamız gerekir . -->

<property name="hibernate.dialect" value ="org.hibernate.dialect.MySQLDialect"/>

<!--Bu alttaki kod ile Veri Tabanı ile alışverişin thread yöntemi ile olacağını belirtiyoruz. -->

<property name="hibernate.current\_session\_context\_class" value="thread"/>

<!--Tabloların her seferinde baştan mı oluşturulacağını yoksa sadece veri mi güncelleceneği belirtiyoruz.

Eğer value'ye update yazsak sadece güncelleme işlerini yapardı. -->

<property name="hibernate.hbm2dll.auto" value="create"/>

<!--Uygulama tarafından veritabanına gönderilen sql betiklerinin console ekranında gösterilmesini sağlar -->

<property name="hibernate.show\_sql" value="true"/>

<!--Sorgular sql formatında olsun.Eğer false yaparsak yada yazmazsak,sorgular hibernate formatında olur. -->

<property name="hibernate.format\_sql" value="true"/>

</properties>

</persistence-unit>

</persistence>

----------------------------------------------------------

Persistence.xml2

<persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence\_2\_0.xsd" version="2.0">

<persistence-unit name="persistence-test" transaction-type="RESOURCE\_LOCAL">

<class>com.project.model.Product</class> //Kullancağımız classları belirtiyoruz.

<properties>

<property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>

<property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/jpa?useUnicode=true&amp;useLegacyDatetimeCode=false&amp;serverTimezone=Turkey" />

<property name="javax.persistence.jdbc.user" value="root"/>

<property name="javax.persistence.jdbc.password" value="1234"/>

<property name="hibernate.hbm2ddl.auto" value="update"/>

<property name="hibernate.show\_sql" value="true"/>

<property name="hibernate.format\_sql" value="true"/>

</properties>

</persistence-unit>

</persistence>

**--------------------------------------------------------------------**

**---C3P0 Provider---**

**Dependency**

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-c3p0 -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-c3p0</artifactId>

<version>5.4.21.Final</version>

</dependency>

**Persistence.xml’e eklenecek kod**

<!-- C3P0 PROVİDER -->

<property name="hibernate.connection.provider\_class" value="org.hibernate.connection.C3P0ConnectionProvider"/>

<property name="hibernate.c3p0.min\_size" value ="5"/>

<property name="hibernate.c3p0.max\_size" value = "20"/>

<property name="hibernate.c3p0.acquire\_increment" value="5"/>

<property name="hibernate.c3p0.timeout" value="600"/>

<property name="hibernate.c3p0.idle\_test\_period" value="1000"/>

<property name="hibernate.c3p0.max\_statements" value="30"/>

---------------------------------------------------------------------------------

**1.Create Table**

**Personal Classı**

@Entity

@Table(name = "personel") //oluşturulacak tablomuzun adı

public class Personel {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) //IDyi otomatik arttır dedik.Bu sebepten ötürü aşşağıda parametreli constructor içine ve set metoduna Id yazmicaz

@Column(name = "personelId" ,unique=true,length=10,nullable=true,updatable = false) //Column adını vermezsek bile değişkenin adıyla oluşturur.Unique=Benzersiz olması aynı Id’den bir tane daha olmaz.Tabi bu primary key olduğu için zaten benzersiz.Length=karakter uzunluğu,nullable=true = boş bırakılabilir,updatable=false = daha sonra güncellenemez demek//

private int personelId;

private String firstName;

private String lastName;

@Temporal(TemporalType.TIMESTAMP) //DATE Sadece Tarih,TIME sadece saat,TIMESTAMP ikisinide tutar.

private Date birthOfDate;

@Lob //About uzun olacağı için,uzun veri alabilir demek

private String about;

@Transient //Veritabanında böyle bir alan açma demek

private double salary;

private String personelNumber;

<<<-! Burdan aşağısında parametreli parametresiz constructor,set-get metodları ve to string metodu var ->>>

**MAİN Class**

EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("persistence-test");

EntityManager entityManager = entityManagerFactory.createEntityManager();

**2.Persist(Veri Tabanına Veri Ekleme(insert into)**

* Kullanacağımız classı persistence.xml’de belirtiyoruz = <class>com.project.model.Product</class>
* Product classı yukarıda ve konular için ortak kullanımda
* <property name="hibernate.hbm2dll.auto" value="create"/> persistence.xml’deki create kodu çok önemli,eğer veri güncelleme yaparken value update değil create ise veri tabanındaki bütün verileri siler yalnızca güncellenen veriyi ekler.

EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("persistence-test");

EntityManager entityManager = entityManagerFactory.createEntityManager();

EntityTransaction entityTransaction = entityManager.getTransaction();

Product product = new Product(3,"ProdNew",1.00,500,new Date(),null);

entityTransaction.begin();

entityManager.persist(product);

entityTransaction.commit();

**3.Update Merge (Veri Tabanındaki Veriyi Güncelleme)**

* Kullanacağımız classı persistence.xml’de belirtiyoruz = <class>com.project.model.Product</class>
* Product classı yukarıda ve konular için ortak kullanımda
* <property name="hibernate.hbm2dll.auto" value="create"/> persistence.xml’deki create kodu çok önemli,eğer veri güncelleme yaparken value update değil create ise veri tabanındaki bütün verileri siler yalnızca güncellenen veriyi ekler.

EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("persistence-test");

EntityManager entityManager = entityManagerFactory.createEntityManager();

EntityTransaction entityTransaction = entityManager.getTransaction();

entityTransaction.begin(); //Transaction başlat

Product product = entityManager.find(Product.class, 1);

//key value ilişkisi-keyimiz çekeceğimiz verinin classı,value ise Idsi ile veriyi javaya çektik.

product.setUnitPrice(250.00); //Fiyatını Güncelledik

product.setUpdateDate(new Date()); //Güncelleme Tarihini ekledik.

entityManager.merge(product); //Burada güncellenmiş veriyi geri yolladık

entityTransaction.commit(); //Transaction işle

**4.Find(Veri Tabanındaki Verileri Çekme Gösterme(select))**

* Kullanacağımız classı persistence.xml’de belirtiyoruz = <class>com.project.model.Product</class>
* Product classı yukarıda ve konular için ortak kullanımda
* <property name="hibernate.hbm2dll.auto" value="create"/> persistence.xml’deki create kodu çok önemli,eğer veri güncelleme yaparken value update değil create ise veri tabanındaki bütün verileri siler yalnızca güncellenen veriyi ekler.

EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory ("persistence-test");

EntityManager entityManager = entityManagerFactory.createEntityManager();

Product product = entityManager.find(Product.class, 1);

//key value ilişkisi-keyimiz çekeceğimiz verinin classı,value ise Idsi ile veriyi javaya çektik.

if(product != null) {

System.out.println("ÜRÜN BİLGİLERİ");

System.out.println("Product ID =" +product.getProductId());

System.out.println("Product İsim =" +product.getName());

System.out.println("Product Fiyat =" +product.getUnitPrice());

System.out.println("Product Adet =" +product.getAvailable());

System.out.println("Product Eklenme Tarihi =" +product.getAddDate());

System.out.println("Product Güncellenme Tarihi =" +product.getUpdateDate());

}

else {

System.out.println("Ürün Bulunamadı");

}

**5.Remove(Veri Tabanındaki Verileri Silme(delete))**

* Kullanacağımız classı persistence.xml’de belirtiyoruz = <class>com.project.model.Product</class>
* Product classı yukarıda ve konular için ortak kullanımda
* <property name="hibernate.hbm2dll.auto" value="create"/> persistence.xml’deki create kodu çok önemli,eğer veri güncelleme yaparken value update değil create ise veri tabanındaki bütün verileri siler yalnızca güncellenen veriyi ekler.

EntityManagerFactory entityManagerFactory = Persistence.createEntityManager Factory("persistence-test");

EntityManager entityManager = entityManagerFactory.createEntityManager();

EntityTransaction entityTransaction = entityManager.getTransaction();

entityTransaction.begin();

Product product = entityManager.find(Product.class,3);

//key value ilişkisi-keyimiz çekeceğimiz verinin classı,value ise Idsi ile veriyi javaya çektik.

entityManager.remove(product); //Veriyi Sildik.

entityTransaction.commit();

**Bazı Annotationlar**

* @Column = adını vermezsek bile değişkenin adıyla oluşturur.Aralarına virgül koyarak alttaki özellikleride ekleyebiliriz.

Unique=true =Benzersiz olması aynı Id’den bir tane daha olmaz.Primary keyler zaten benzersiz. Column anotasyonu içine yazılır.

Length=10 =karakter uzunluğu, Column anotasyonu içine yazılır.

nullable=true = boş bırakılabilir, Column anotasyonu içine yazılır.

updatable=false =daha sonra güncellenemez demek,Column anotasyonu içine yazılır.

* @GeneratedValue(strategy = GenerationType.IDENTITY) = IDyi otomatik arttırma.Bu sebepten ötürü aşşağıda parametreli constructor içine ve set metodunu Id için oluşturmayız.
* @GeneratedValue(strategy = GenerationType.AUTO) = Buda üsttekiyle aynı işi görüyor.Fakat sql’de bir tane daha tablo açıyor.Oranın içine de sıradaki gelecek ıd numarasını yazıyor.
* @Embeddable , @Embedded ve @AttributeOverrides Anotasyonları = Assocation ilişkisi olan classlarda kullanırız.Örnek verecek olursak.Diyelim ki Author ve Book pojosu oluşturduk.Book pojosunun içinde kitapAdi,kitapSayfaSayisi,kitapTürü ve son olarak Author türünden kitapYazari diye bir değişken oluşturduk.Author classında da yazarIsim,yazarSoyisim adında değişkenler var.Author pojosunun üstüne @Embeddable yazıyoruz.Ve Book pojosunun içindeki author isimle değişkenin üstüne de @Embedded yazıyoruz.Bu şekilde veri tabanına kayıt eklersek eğer hem book pojosunun hem de author pojosunun içindeki columnlarda oluşur.Peki @AttributeOverrides nedir?Bunuda yine embedded anotasyonunun hemen altına yazıyoruz.Ve Author pojosundan gelecek değişken isimlerinden değil de kendi koymak istediğimiz isimlerde column oluşturucağımızı belirtiyoruz. NOT: Yalnızca bir tane column değiştiriceksek eğer attributeOverrides içine almaya gerek yok, @AttributeOverride yeterli olur.Kullanımı aşağıda.

ÖRNEK:

@Embedded

@AttributeOverrides({

@AttributeOverride(name ="authorName" ,column = @Column(name ="yazarAdi",length = 25 )) ,

@AttributeOverride(name = "authorSurname" ,column = @Column(name ="yazarSoyadi", length = 25 ))

})

//yalnızca bir tane column değiştiriceksek eğer attributeOverrides içine almaya gerek yok,üstteki anotasyonu tek yazsak yeterli olur.

private Author author; //Etki edeceği değişken

* @ElementCollection ve @CollectionTable Anotasyonu = ElementCollection anotasyonunun görevi şudur.Diyelim Personel diye bir pojo-sql tablosu oluşturucaz.Ve personelin birden fazla numarası var.Ee bunu sql column’a nasıl sığdırıcaz?İşte O List türündeki değişkenin üstüne bunu yazarsak personel numaraları diye başka bir tablo oluşturuyor otomatik. Peki CollectionTable anotasyonu nedir?Çok gerekli değil bu kullanmasakta olur ama bilelim.Bu anotasyonda otomatik oluşucak telefon numaraları adlı sql tablosunun adı ve Id nin column ismini ayarlama.

@ElementCollection

List<String> phoneNumbers;

--------

@ElementCollection

@CollectionTable(name ="personelNumbers",joinColumns = @JoinColumn(name= "PersonelId")) //Tablonun Adı //Yeni oluşacak tabloyla asıl tablo arasındaki ilişkili columnun adı(ID columnun adı)

List<String> phoneNumbers;

ElementCollection ile MAP Kullanımı

@CollectionTable(name ="personelNumbers",joinColumns = @JoinColumn(name="PersonelId"))

@MapKeyColumn(name ="Numaralar")

@Column(name ="telefonNumaraları")

private Map<String, String> phoneNumbers = new HashMap<String, String>();

* @Temporal(TemporalType.TIMESTAMP) Anotasyonu = DATE Sadece Tarih,TIME sadece saat,TIMESTAMP ikisinide tutar. Date türündeki değişkenlerin üstüne gelir.

* @Lob Anotasyony = About gibi uzun metinler için,uzun veri alabilir demek
* @Transient Anotasyonu = Veritabanında böyle bir column açma demek

**Hibernate ile İlişkisel Veri Tabanları**

**@OneToOne Anotasyonu**

Bire bir ilişkisi.Mesela bir kullanıcının birr adresi olabilir.Diyelim ki Adres diye bir pojo var ve onu assocation eden bir User Pojosu var.Adres classınıda bir tablo gibi entity ıd anotasyonları ile donatıyoruz.Ve User classına gelip Adres türünden oluşturuğumuz değişkenin üstüne @OneToOne(cascade = CascadeType.PERSIST) anotasyonunu ekliyoruz.Bu kadar.Fakat bu pojoyu sadece bu user kullanabilir.Demek bu.Ona göre

Persistence.xml’de her iki classıda <class> içine yazmayı unutma!

@OneToOne(cascade = CascadeType.PERSIST) //Bu cascadeType özelliği şudur.Bunu yazmasaydık mainde persist ederken transaction içine ilk user pojosunu sonra adres pojosunu persist etmeliydik.Ama Bu özellik sayesinde sadece useri persist ediyoruz.Bu cascadeType özelliği içinde persist harici remove özelliği falanda var.O da bu useri sil dediğimizde o userin adresini de siliyor.One to One de kullanılıyor fakat Many to one’de falan remove özelliği tehlikeli.

**@OneToMany Anotasyonu**

Bire çok ilişkisi.Mesela bir satıcının birden çok ürünü var.Diyelim ki Product diye bir pojo var.Id Entity anotasyonları donatılmış bir entity bu.Brir tanede user var.Bu user birden fazla product satıyor.O zaman product pojosunun içine geliyoruz ve product generic bağlantılı bir List oluşturuyoruz.(list olmasının sebebi birden fazla ürünü içine atabilelim).Daha sonra bu list türündeki değişkenin üstünde @OneToMany(cascade = CascadeType.PERSIST) yazıyoruz.Bu kadar.Ama şunu kesinlikle unutmuyoruz.Bu ürünleri sadece bu user satabilir.

**@ManyToOne Anotasyonu**

Çoka bir ilişkisi.Mesela Department ve Employee adında bir pojo ve veri tabanı tablomuz var.Birden çok employee bir departmanda çalışabilir mantığı ile gider.Department entitysini oluşturuyoruz.Employee entitysinin içine ise department türünden değişken oluşturup üstüne @ManyToOne yazıyoruz.

ÖNEMLİ NOT:Bu anotasyonu kullanırken cascadeType özelliği kullanmak sakıncalı.Bu sebepten ötürü ilk önce department yani ManyToOne anotasyonunun sağ kısmı olan One pojosunu persist ediyoruz.Daha sonra Many kısmını yani Employeeleri persist ediyoruz.

**@ManyToMany Anotasyonu**

Çoka çok ilişkisi.Mesela bir userin birden çok ürünü var aynı zamanda ürünlerinde birden çok useri var.O zaman şöyle yapıyoruz.Hem user adında hemde product adında entity oluşturuyoruz.Daha sonra Userin productlarını görebilmek için User entitisi içinde List<Product> products = new Arraylist<>(); değişkenini oluşturup bunun üzerine @ManyToMany yazıyoruz.Aynısını Product entitisi içine gelip List<User> users = new Arraylist<>(); değişkenine de yapıyoruz.Üstüne ManyToMany yazıyoruz.Bu kadar fakat böyle kaydedersek eğer veri tabanında Users,Products,User\_Products ve Products\_User adında tablo oluşur.Bir tane yardımcı tabloyu silmek için iki entitiden birinin içine gelip @ManyToMany Anotasyonunun yanına parantez açıp içine mappedBy ile seçmediğimiz diğer ManyToMany anoyastonu ile donatılmış değişkenin adını tırnak içinde yazıyoruz. Örnek;

@ManyToMany(mappedBy =”users”,cascade = CascadeType.PERSIST) //Diğer clasın içinde değişken

List<Product> products = new Arraylist<Product>();

**JPQL(Jakarta Persistence Query Language)**

Bu bir sql programlama dilidir.Bunu neden kullanıyoruz.?Daha çok filtreli sorgular için kullanılıyor.Fiyatı bu aralıkta olan ürünleri getir.Ürünleri fiyatını azalandan artana göre sırala,ismi şu olan ürünleri getir gibi sorgularda kullanılıyor.

**ÖRNEK SORGULAR**

1.public static final String getAllProduct ="SELECT p FROM Product p";

2.public static final String getOnlyProductNames ="SELECT p.productName FROM Product p ";

3.public static final String getProductById ="SELECT p FROM Product p WHERE p.productId=:productId";

4.public static final String getProductNamesAndPrice ="SELECT p.productName, p.unitPrice FROM Product p";

5.public static final String getGreatPrice ="SELECT p FROM Product p WHERE p.unitPrice > :unitPrice";

6.public static final String getBetweenPrice ="SELECT p FROM Product p WHERE p.unitPrice BETWEEN :minUnitPrice AND :maxUnitPrice";

7.public static final String getLikeNameProduct ="SELECT p FROM Product p WHERE p.productName LIKE :productName";

8.public static final String getOrderByDESC ="SELECT p FROM Product p ORDER BY p.unitPrice DESC";

9.public static final String getOrderByASC ="SELECT p FROM Product p ORDER BY p.unitPrice ASC";

10.public static final String getStatisticFromProduct = "SELECT AVG(p.unitPrice), SUM(p.unitPrice), MAX(p.unitPrice), MIN(p.unitPrice), COUNT(p) FROM Product p";

**YUKARIDAKİ SORGULARI ÇAĞIRDIĞIMIZ REPOSİTORY CLASSI**

IJPAFactory jpafactory = new JPAFactory();

EntityManager entityManager = jpafactory.getEntityManager();

EntityTransaction entityTransaction = jpafactory.getEntityTransaction();

1.public List<Product> getAllProduct() {

TypedQuery<Product> query = this.entityManager.createQuery (Queries.getAllProduct,Product.class);

List<Product> products = query.getResultList();

return products; }

2.public List<String> getOnlyProductNames() {

TypedQuery<String> query = this.entityManager.createQuery (Queries.getOnlyProductNames,String.class);

List<String> products = query.getResultList();

return products; }

3.public Product getProductById(int productId) {

TypedQuery<Product> query = this.entityManager.createQuery (Queries.getProductById,Product.class);

query.setParameter("productId", productId);

Product product = query.getSingleResult();

return product; }

4.public List<Object[]> getProductNamesAndPrice() {

TypedQuery<Object[]> query = this.entityManager.createQuery (Queries.getProductNamesAndPrice,Object[].class);

// Object esnek bağlantılı bir list olduğu için typequery kullanılamaz

// Burada query kullanacağız.

List<Object[]> products = query.getResultList();

return products; }

5.public List<Product> getGreatPrice(int unitPrice) {

TypedQuery<Product> query = this.entityManager.createQuery (Queries.getGreatPrice,Product.class);

query.setParameter("unitPrice", unitPrice);

List<Product> products = query.getResultList();

return products; }

6.public List<Product> getBetweenPrice(int minUnitPrice, int maxUnitPrice) {

TypedQuery<Product> query = this.entityManager.createQuery (Queries.getBetweenPrice,Product.class);

query.setParameter("minUnitPrice", minUnitPrice);

query.setParameter("maxUnitPrice", maxUnitPrice);

List<Product> products = query.getResultList();

return products; }

7.public List<Product> getLikeNameProduct(String productName) {

TypedQuery<Product> query = this.entityManager.createQuery (Queries.getLikeNameProduct,Product.class);

query.setParameter("productName","%"+productName+"%");

List<Product> products = query.getResultList();

return products; }

8.public List<Product> getOrderByDESC() {

TypedQuery<Product> query = this.entityManager.createQuery (Queries.getOrderByDESC,Product.class);

List<Product> products = query.getResultList();

return products; }

9.public List<Product> getOrderByASC() {

TypedQuery<Product> query = this.entityManager.createQuery (Queries.getOrderByASC,Product.class);

List<Product> products = query.getResultList();

return products; }

@SuppressWarnings("unchecked")

10.public List<Object[]> getStatisticFromProduct() {

// Object esnek bağlantılı bir list olduğu için typequery kullanılamaz

// Burada query kullanacağız.

Query query = this.entityManager.createQuery (Queries.getStatisticFromProduct);

List<Object[]> list = query.getResultList();

return list; }

**Named Query Kullanımı**

Buda yukarıdakilerle aynı işlemi görür sadece daha hızlı daha dinamiktir.Bide queiries gibi classlar oluşturmaya gerek kalmaz.

ÖRNEK

[@Entity](https://www.objectdb.com/api/java/jpa/Entity)

@NamedQueries({

[@NamedQuery](https://www.objectdb.com/api/java/jpa/NamedQuery)([name](https://www.objectdb.com/api/java/jpa/NamedQuery/name)="Country.findAll",[query](https://www.objectdb.com/api/java/jpa/NamedQuery/query)="SELECT c FROM Country c"),

[@NamedQuery](https://www.objectdb.com/api/java/jpa/NamedQuery)([name](https://www.objectdb.com/api/java/jpa/NamedQuery/name)="Country.findByName",[query](https://www.objectdb.com/api/java/jpa/NamedQuery/query)="SELECT c FROM Country c WHERE c.name = :name"),

})

public class Country {

}

MAİN CLASS

EntityManagerFactory

EntityManager oluştur ilk önce

TypedQuery<Country> typedQuery = this.entityManager.createNamedQuery (“Country.findAll”,Country. class);

List< Country> countiries = typedQuery.getResultList();

TypedQuery<Country> typedQuery1 = this.entityManager.createNamedQuery (“Country.findByName”,Country. class);

typedQuery1.setParameter(“name”,ÜlkeİsmiGir);

List< Country> countiries = typedQuery1.getResultList();