**JPA nedir ?**

JPA(Java Persistence Api) amacı : projede orm aracı ne olursa olsun,değişsin ben hepsiyle çalışırım der.

Ve bütün orm araçlarını tek bir çatı altında toplar JPA.

Aynı zamanda şöyle düşünülebilir JPA dans ise Hibernate dansçıdır. JPA ‘nın kendine has sorgulama dili vardır JPQL , bununla bütün veritabanlarına sorgu gönderebiliriz.

Eğer ki biz projemizi JPA ile geliştirirsek bütün orm araçlarından bağımsız olmuş ve yarın farklı bir ORM aracına geçmek istediğimizde geçebilmiş olacağız.Fakat biz JPA değil onun implementasyonu olan HİBERNATE kullanırsak tamamen o ORM framework’une bağımlı olmuş olacağız. Ve değişiklik mümkün olmayacak.

JPA ile de HİBERNATE ile de istediğimiz veritabanı üzerinde çalışabiliriz.hiç farketmez(mssql,mysql,postgresql,db2,oracle,access) vb.

Özet : **JPA İSKELET , HİBERNATE ONUN ETE KEĞİME BÜRÜNMÜŞ HALİ…..**

Transaction : Yap hep , ya hiç mantığıyla çalışır.

hibernate.config.xml : resource altında olmalıdır.

**JPA kullanacaksak;**

1-persistence.xml

2-JPQL-HQL hepsini kullanabilirsin.

3-EntityManagerFactory,EntityManager

**Hibernate kullanacaksak;**

1-hibernate.cfg.xml

2-HQL – JPQL hepsini kullanabilirsin.

3-SessionFactory,Session

En basit hali ile bu şekilde özetlenebilir hizmeti , amacı ,gayesi aynı söz dizimi çok ufak farklı

**ANATASYON(ANNOTATİON)**

**@Transient :** class içinde olmasını isteyipte kolon olarak yansımasını istemediğimiz durumlarda kullanılır

**@Basic :** class içinde ki alanın kesinlikle tabloya kolon olarak yansıması gerektiğini söyler.

**@Temporal:** tarihin,tarih veya zaman veya tarih-zaman şeklinde formatını sağlar.

**@Lob(Large object):** uzun textler yasmak için kullanılan anatasyondur.

**@Embeddable- @Embedded :** bir classı gömülebilir hale getirip entity classları içersine gömüp ,entity içerisine kolon olarak yansımasını sağlayabiliriz.

**@AttiribueOverrides :** gömmüş olduğumuz sınıfın içindeki alanların tabloya kolon olarak yansırken , kolon isimlerini istediğimiz gibi vermemizi sağlar.

**@ElementCollection :** adından da anlaşılacağı üzere elimizde ki listeyi ayrı bir tabloda ana tablo ile ilişkili tutmak istediğimiz durumlarda kullanılır.

**@JoinTable :** 2 entity’I birbiriyle ilişkilendirdiğimizde Jointable kullanırsak ilişkiyi harici ayrı bir 3.cü tablo oluşturarak sağlıyor.

**@JoinColumn :** 2 entity’I ilişkilendirdiğimizde JoinColumn kullanırsak ilişkiyi ek bir kolon ekleyerek sağlıyor 3. Tabloyu oluşturmuyor….

**FetchType.EAGER :** bir tabloya select attığımızda o tablonun ilişkili olduğu diğer bütün kayıtları da getir demiş oluruz.

**FetchType.LAZY :** bir tabloya select attığımızda sadece o tablonun kayıtlarını bize dönsün ilişkisi olan diğer hiçbir tablonun kaydını getirmesin demiş olduk.

**Mapped By** : İlişinin sahibini belirtmek için kullanılır.

**CascadeType.\*** : Bir entity üzerinde crud işlemleri gerçekleştirince o entity’nin bağlı olduğu diğer entity’de otomatik olarak tetikleyip crud işlemleri yaptırmamıza olanak sağlar.

**Apache Derby Veritabanı** : ücretsiz bir veritabanıdır küçüktür kurumsal projede kullanılmaz.Java ile yazılmıştır kullanım arayüzü yok eclipse içinden yönetiliyor.

**İlişkinin Sahibi Kavramı** : Bir class bir classın içindeyse ,içeriyorsa o sınıf ilişkinin sahibi , hakimidir. Veya şöyle diyebiliriz.

Bir entity diğer entity’den haberi varsa, bilgilerine erişebiliyorsa o taraf hakim taraftır.Hakim olmayan ise hakim olanın bilgilerine erişemez..

**İlişkide Hakim Taraf Nasıl Belirlenir ?**

Hakim olmayan tarafta gelip,hakim taraftaki anatasyon altında ki referansı alıp hakim olmayan taraftaki mapped by ‘a veriririz.Bu şekilde hakim taraf belirlenir hakim olmayanın bilgilerine erişilebilir.Hakim olmayan taraftan erişemeyiz ama.

Many’nin olduğu ilişkilerde listelerden faydalanıyoruz çünkü içlerinde birden fazla nesne tutabildikleri için.

Unidirectional(Tek yönlü erişim) : bir tablodan diğer tabloya erişilebilirken karşı tablodan diğer tabloya erişimin olmadığı erişim türüdür.

Bidirectional(Çift yönlü erişim) : her iki tablodan da birbirlerine erişebildikleri erişim türüdür.

Güzel bir bilgi : SQL ,tablolar üzerinde sorgulamalar yaparken , HQL ,JPQL gibi sorgulama dilleri ENTİTY’ler üzerinde sorgulama yapar tablo ‘da değil. Yani bir tablom varsa ve bunun Entity karşılığı yoksa HQL , JPQL kullanamazsın.

Her zaman config dosyamızın adını “hibernate.cfg.xml” vermek zorunda değiliz. Eğer ki değiştirmessek resource altında default olarak bu isimle arar. Eğer ki değiştirirsek SessionFactory oluşturduğum yerde onun ismini belirtmem gerekiyor.

new Configuration().configure(“hibernate-mysql.cfg.xml”);

setFirstResult() : kaçıncı kayıttan başlayayyım getirirken diyor.Başlangiç belirtmek için kullanıyoruz.Belirttiğimiz değer DAHİL değil! Ondan bir sonrakisinden almaya başlıyor.

setMaxResults() : max kaç tane kayıt getireyim sana diyor.

**HİBERNATE CACHE İŞLEMLERİ**

Cache(ön bellek) : veritabanına yaptığımız sorguların cache üzerinde depolanarak,tekrar aynı sorguyu tetiklediğimizde veritabanına gitmeden direkt cache üzerinde çekerek bize döndürür.

Bu da yöntemde gereksiz yere her seferinde veritabanına gitme sorununu çözmüş olur.

->Session nesnesi içerisinde persistence context( hamile anne-çocuk örneği unutma) vardır.

Burası cache ile ilgilidır.

->Biz session nesnesini kapatırsak(session.close();) o zaman cache deki tüm verileride kaybederiz.Çünkü Session ölürse persistence context’te ölür . Persistence context’in ölmesi cache’I kaybetmek demektir.