Java Projelerinde Hibernate

Java ile veritabanı işlemleri yapıyorsanız ve hibernate ile tanışmadıysanız bu konu sizin için devrim niteliğinde olacaktır. Hibernate kısaca java nesnelerinin ilişkisel veritabanında birebir karşılığının tutulmasını sağlar. Her tutulan nesne veritabanında ki bir tabloya karşılık gelmektedir. Bu yöntem ile karmaşık sql sorgularına gerek kalmadan tablolar üzerinde her türlü işlem gerçekleştirilebilir.

Hibernate Kullanmanın Avantajları Nelerdir?

- Hibernate ile veritabanında tabloları oluşturmaya ve tabloların birbirleriyle ilişkilerini tanımlamaya gerek kalmaz.
- Projenin bakımını ve güncellenmesini kolaylaştırır.
- Tabloya yeni bir alan eklemek gerektiğinde kod kısmında oluşacak hatalar en aza indirilir.
- Karmaşık sql sorguları ile uğraşmadan verilere erişim sağlanır.
- Projenin farklı veritabanlarına taşıma işlemi basit ve hızlıdır.

Günümüzde kurumsal java uygulamalarında hibernate ve spring kullanımı standart hale gelmiştir.

ORM'ye Daha Yakından Bir Bakış

ORM Object Relational Mapping anlamına gelmektedir. ORM toolları nesne tabanlı programlamada bulunan objeler ile veritabanı sistemimizdeki tablolar arasında köprü görevi kurulmasını sağlar. Objeleri ilişkisel veritabanında mapping yapmaya yarar. ORM toollarının kullanımının en büyük avantajı OOP'da yer alan inheritance, polymorphism gibi konseptleri veritabanımız ile kolaylıkla kullanabilmektir. Ayrıca bazı durumlarda projelerde SQL sorguları yazılımcıya büyük yük olabiliyor. Özellikle proje büyüdüğünde bu işlemler zorlaşıyor. ORM

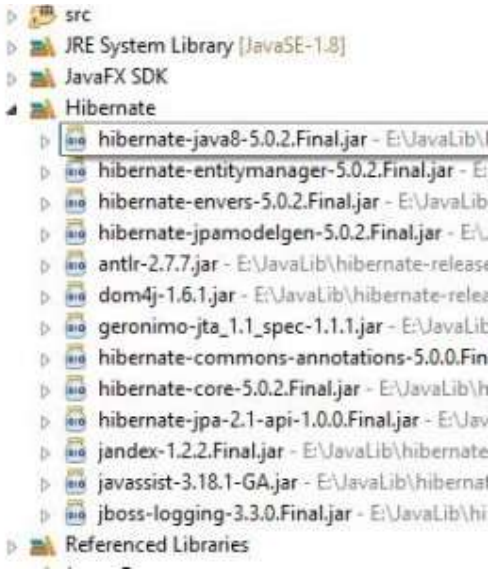
toollarından önce yazılımcılar kendileri objeleri veritabanındaki tablolarla eşleştirecek kodlar yazıyorlardı. Hibernate gibi toollar yazılımcıları bu yükten kurtardı. Burada bir not: ORM bu konseptin adıdır, bir araç değildir. Hibernate bir ORM aracıdır ve EclipseLink, OpenJPA, TopLink gibi birçok farklı ORM aracı bulunmaktadır. Hibernate, Java dünyasındaki popüler ORM araçlarından biridir.

```java
@Entity
@Table(name = "Students")
public class Student {

	@Id
	@Column(name = "id")
	private int id;

	@Column(name = "first_name")
	private String firstName;

	@Column(name = "last_name")
	private String lastName;

	@Column(name = "email")
	private String email;
```

Hibernate'in Java Projesine Eklenmesi

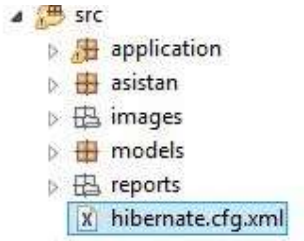
Hibernate açık kaynak kodlu ve ücretsiz bir ORM'dir. Hibernate'i kullanmaya başlamak için gerekli jar dosyalarını http://hibernate.org/orm/ adresinden indirerek yada maven yardımı ile java projesine eklemeniz gerekir.

```xml
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-agroal</artifactId>
  <version>5.3.5.Final</version>
  <type>pom</type>
</dependency>
```



Yukarıdaki hibernate kütüphanelerini indirdiğiniz dosyanın içerisinde lib klasörü altında bulabilirsiniz.

Bu işlemden sonra bir tane hibernate ayar dosyası oluşturmanız gerekir. Bu dosya hibernate'in projeniz ve veritabanıyla haberleşmesini sağlayan tüm ayarların yer aldığı dosyadır. Örnek hibernate ayar dosyaları indirdiğiniz klasör içerisinde **hibernate.cfg.xml** adıyla bulunmaktadır.




```
<hibernate-configuration>

 <session-factory>

  <!-- Database connection settings -->

  <property name="connection.driver_class">com.mysql.jdbc.Driver</property>

```

```
9     <property
10 name="connection.url">jdbc:mysql://localhost:3306/koorddb?createDatabaseIfNotExist=true</property>
11
12    <property name="connection.username">root</property>
13
14    <property name="connection.password">sifre</property>
15
16    <!-- JDBC connection pool (use the built-in) -->
17
18    <property name="connection.pool_size">1</property>
19
20    <!-- SQL dialect -->
21
22    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
23
24    <!-- Enable Hibernate's automatic session context management -->
25
26    <property name="current_session_context_class">thread</property>
27
28    <!-- Disable the second-level cache  -->
29
30    <property name="cache.provider_class">org.hibernate.cache.internal.NoCacheProvider</property>
31
32    <!-- Echo all executed SQL to stdout -->
33
34    <property name="show_sql">true</property>
35
36    <!-- Drop and re-create the database schema on startup -->
37
38    <property name="hbm2ddl.auto">update</property>
39
40    <mapping class="models.Tanimlamalar"></mapping>
41
```

```
    <mapping class="models.Alanlar"></mapping>

    <mapping class="models.Koordinatorler"></mapping>

    <mapping class="models.Isletmeler"></mapping>

    <mapping class="models.Ogrenciler"></mapping>

    <mapping class="models.HaftalikProgram"></mapping>

    <mapping class="models.GorevKayitlari"></mapping>

  </session-factory>

</hibernate-configuration>
```

Yukarıdaki ayar dosyası mysql veritabanına bağlanmak için oluşturulmuştur. Proje derlendiğinde veritabanı yoksa en alttaki tablolar ve içerisindeki alanlar model nesnelerinden otomatik oluşturulur.

Aşağıda nesne bir okuldaki alanları modellemek için kullanılmıştır.

```
@Entity
public class Alanlar {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int alanId;
    private String alanAdi;

    public int getAlanId() {
        return alanId;
```

```
    }

    public void setAlanId(int alanId) {
        this.alanId = alanId;
    }

    public String getAlanAdi() {
        return alanAdi;
    }

    public void setAlanAdi(String alanAdi) {
        this.alanAdi = alanAdi;
    }

    @OneToMany(mappedBy="alan")
     private Set<Koordinatorler> koordinator;

     @OneToMany(mappedBy="alan")
     private Set<Ogrenciler> alanlar;

}
```

Model incelendiğinde bu nesnenin koordinatorler ve ogrenciler nesnelerinde ki bir alana karşılık geldiği görülmektedir. Bu durum ilişkisel veritabanında foreign key olarak gerçekleştirilir.

Modelleme işlemini de yaptıktan sonra geriye en önemli nokta olan verilerin kaydedilmesi, çekilmesi silinmesi ve güncellenmesi işlemi kaldı.

Aşağıda veritabanı üzerinde bu işlemlerin gerçekleştirilmesi için gerekli kodları metotlar halinde gösterdim.

```
SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
Session session;
private static List<Alanlar> alanlar;
//Alan bilgilerini veritabanından alarak bir List içerisine aktaralım.

public void alanBilgileriniGetir(){
        session=sessionFactory.openSession();
        session.beginTransaction();
        Criteria cr=session.createCriteria(Alanlar.class);
        alanlar=cr.list();
        session.close();
    }
//Bir alanı veritabanına ekleyelim.
private void alanKaydet (){

    Alanlar alan= new Alanlar();
    alan.setAlanAdi(txtAlanAdi.getText().toUpperCase());
    session=sessionFactory.openSession();
    session.beginTransaction();
    session.save(alan);
    session.getTransaction().commit();
    session.close();
}
//Bir alanı veritabanından silelim.
private void alanSil(){

    session=sessionFactory.openSession();
    session.beginTransaction();
    Alanlar alan=new Alanlar();
    alan.setAlanId(1);
    session.delete(alan);
    session.getTransaction().commit();
    session.close();    }
```

Hibernate'i başta anlamak biraz zor olsa da nesne tabanlı çalışan kişilerin veritabanıyla ilgili kod yazma işlemini %50 oranında azaltmaktadır. Konuya ilgili sorularınız bana iletebilirsiniz.

KAYNAKLAR:

1. https://www.kodlamamerkezi.com/java/java-projelerinde-hibernate-kullanimi/
2. https://tugrulbayrak.medium.com/hibernate-1-orm-kavram%C4%B1na-giri%C5%9F-c2ba2f2a3bfe