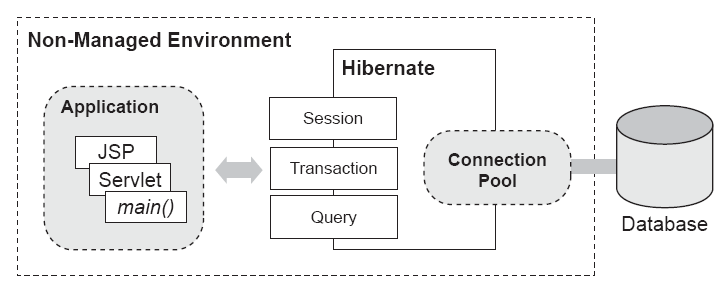
**Hibernate**

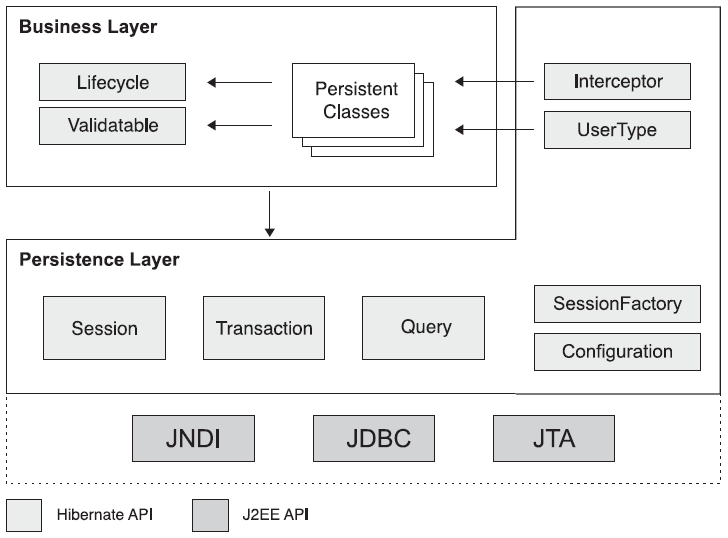
**Hibernate**, [Java](http://tr.wikipedia.org/wiki/Java_programlama_dili) platformunda yazılmış bir [ORM](http://tr.wikipedia.org/w/index.php?title=ORM&action=edit) (Object/Relational Mapping) aracıdır. ORM, [nesne odaklı](http://tr.wikipedia.org/w/index.php?title=Nesne_odakl%C4%B1&action=edit) (object oriented) dillerdeki nesnelerin, [ilişkisel veritabanlarındaki](http://tr.wikipedia.org/w/index.php?title=%C4%B0li%C5%9Fkisel_veritabanlar%C4%B1&action=edit) (relational databases) kayıtlara nasıl karşılık geldiğini yürüten bir teknolojidir.

Gavin King tarafından 2001 yılının sonlarına doğru geliştirilmiştir.

Hibernate gibi ORM araçlarıyla, bir nesneyi veritabanına kaydetmek, yeni halini güncellemek ve sorgulama yapmak düz [SQL](http://tr.wikipedia.org/wiki/SQL) bağlantılarına göre çok kolaydır.

Hibernate gibi ORM araçlarının en önemli faydası, kod yazımını kısaltmak veya kolaylaştırmaktan öte, yazılım bakımını kolaylaştırmasıdır. Veritabanı temelli uygulamalarda, kodun 1/3´ü veritabanı erişimine yöneliktir. Veritabanındaki bir kolonunun tipinin değişmesi, yeni bir kolon eklenmesi gibi değişiklikler, bütün veri erişim kodunu tekrar gözden geçirmeyi gerektirir. Hibernate ile bu gözden geçirmeden çok yüksek oranda tasarruf edilir. Hibernate kullanılan yazılımlarda, veritabanındaki değişikliklerde yapılması gereken sadece nesnelerle tabloların birbirine nasıl eşleştirildiğinin (mapping) gözden geçirilmesidir.



****

**SessionFactory**

SessionFactory derlenmiş basit bir veri tabanı adreslemelerinin tutulduğu alan olarak tanımlanabilir

**Session**

Uygulama ve kalıcı veriler arasında ki tek iş parçacıklı kısa süreli bir görüşmeyi temsil eder.

**Kalıcı nesneler ve koleksiyonlar (*Persistent Objects and Collections*)**

Tek iş parçacıklı (*single-thread*) , kısa ömürlü nesnelerdir ve kalıcı durumu (*persistent state*) ve iş metodlarını (*business functions*) içerirler. Bu nesneler sıradan *java Beanler* olabilirler, ama en önemli özellikleri halihazırda tek bir *Session* nesnesiyle ilişkili olmalarıdır. *Session* kapanır kapanmaz, bu nesnelerin Session ile birliktelikleri sona erer ve uygulama katmanından yönlendirilmeye, kullanılmaya hazır durumu geçerler.

**Gelip-geçici Nesneler ve Koleksiyonlar (*Transient Objects and Collections*)**

Bu türde nesneler henüz bir *Session* nesnesine ilişiklendirilmemiş kalıcı (*persistent*) nesneleri temsil ederler. Bu nesneler bir *Session* nesnesi tarafından, bir uygulama tarafından yada bir kapanan bir *Session* nesnesi tarafından başlatılmış olabilirler. Bununla birlikte henüz kalıcılaştırılmamış, kaydedilmemiş bir kalıcı nesne (persistent object) olabilirler.

**Hareketler (*Transactions*)**

Tek iş parçacıklı (*single-thread*), kısa ömürlü nesnelerden oluşan bir katmandır. Uygulamayı altta yatan ayrıntılı JDBC, JTA veya CORBA katmanlarından yalıtır. Bir Session nesnesi aynı anda birden çok Hareketle iş görebilir.

**Bağlantı Sağlayıcı (*ConnectionProvider*)**

JDBC bağlantıları için bir fabrika (factory) sınıfını temsil eder. Uygulamayı altta yatan *Datasource* ve *DriverManager*’den yalıtır.

**Hareket Fabrikası (*TransactionFactory*)**

Hareket varlıkları için bir fabrika konumu üstlenir. Uygulama tarafından kullanımı açık bırakılmamıştır, ama istenirse kullanıcı tarafından genişletilip (extend), gerçekleştirilebilir (implement).

***SESSIONFACTORY* YAPILANDIRMASI**

Hibernatin pek çok değişik yazılımsal ortamla birlikte kullanımı söz konusu olduğu için, birçok yapılandırma parametresinin kullanılması sözkonusudur. Hibernate kurulumuyla birlikte gelen önbelirli (*default*) değiştirgeler (*parameter*) makul, mantıklı bir şekilde ayarlanmışlardır. Hibernate için gerekli ayarlar *hibernate.properties* kütüğü aracılığıyla değiştirilir, yapılandırılır. Bu kütüğün sistemin sınıf yoluna (*classpath*) eklenmesi gerekmektedir.

***Programlama yoluyla yapılandırma***

*net.sf.hibernate.Configuration* paketi, uygulama içinde kullanılan tüm java veri türlerinin (java *types*) SQL veri türlerine (types) eşlemesini gerçekleştirir. *Configuration* bir *SessionFactory* nesnesi elde etmek için kullanılır. Eşlemeler (mapping) çeşitli XML kütüklerine dayalı olarak belirlenir.

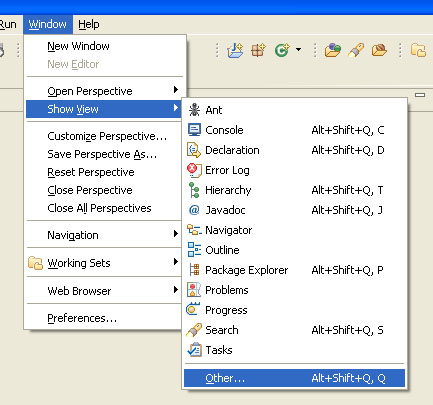
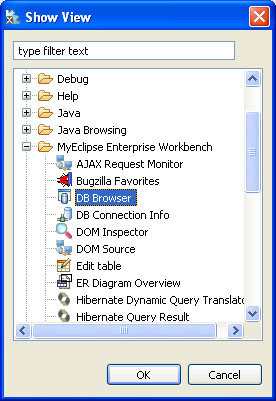
***SessionFactory elde etme***

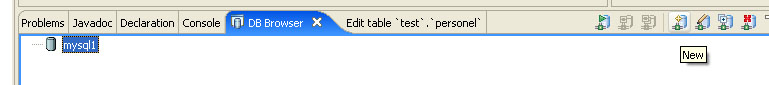
Tüm eşlemeler (mapping) *Configuration* tarafından ayrıştırıldıktan sonra uygulama bir *SessionFactory* nesnesi elde etmelidir. Bu nesne tüm uygulama iş parçacıkları (thread) tarafından ortak bir şekilde kullanılır. Birden fazla veritabanına erişim varsa her biri için SessionFactory gereklidir. SessionFactory, üretilen SQL cümlelerini ve Hibernate’in çalışma zamanında kullandığı diğer haritalama metadata’larını önbellekte (cache) tutar.

***JDBC baglantıları***

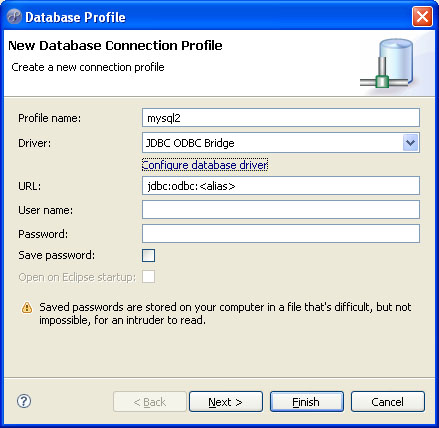
Hibernate kullanıcılara iki çeşit JDBC bağlantısı yapma imkanı verir; kullanıcı tanımlı JDBC bağlantısı ve *SessionFactory* yoluyla JDBC bağlantısı. SessionFactory yoluyla JDBC bağlantısı kurmak için hibernate yapılandırılmalıdır. Hibernate bağlantı havuzu oluşturmada (connection pooling) ve bu havuzu yönetmede kendisiyle birlikte gelen *C3P0ConnectionProvider*’i kullanır.

**MyEclips’te veri tabanı araçlarını kullanma**

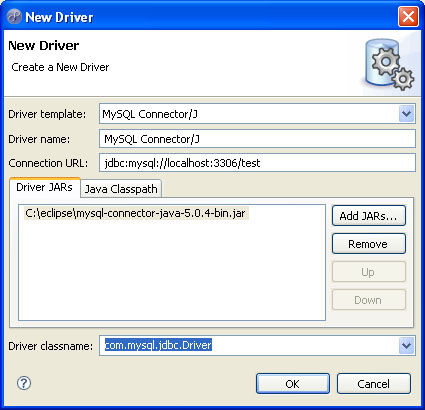
 



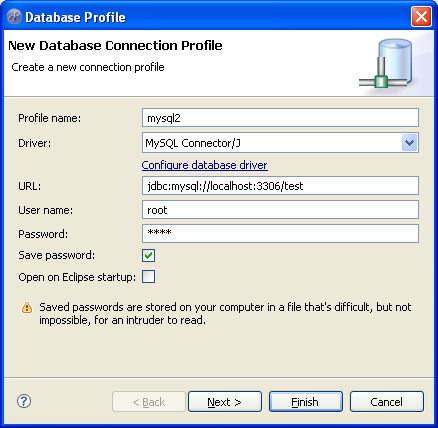
**Yeni connection tanımlama**



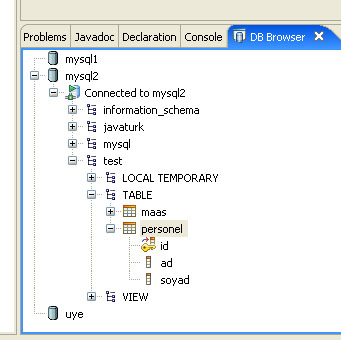
Bütün değerler girilirse şu şekilde olmalıdır.



Bu tanımdan sonra mevcut profiller üzerinden sağ tuşla **open connection** seçilir ve veri tabanına bağlantı yapılır.



Veri tabanına bağlanılınca aşağıdaki ekran görünür.



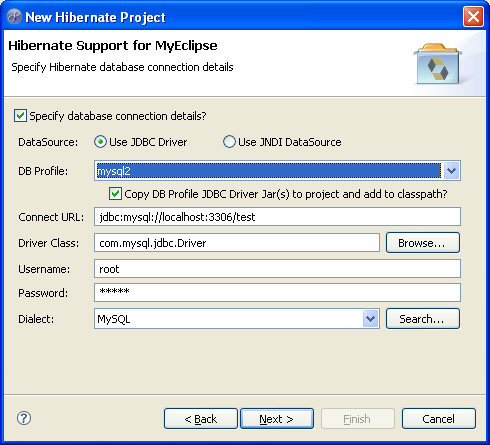
**Hibernate Projesi Oluşturma**

* Düz java projesi
* Strut projesi

**Düz java projesi tanımlama**

* New / Project… (Java Project)
* HibernatePrj1 (Proje ismi)
* Paket oluşturulur (src)
* MyEclipse / Add Hibernate Capabilities

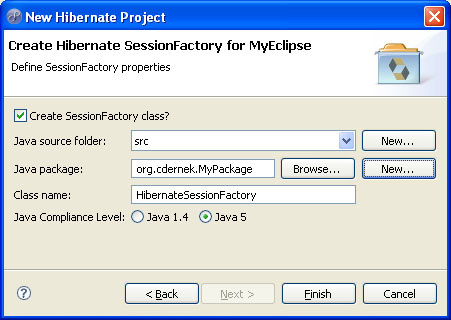
|  |  |
| --- | --- |
| Untitled-1 | Untitled-1 |



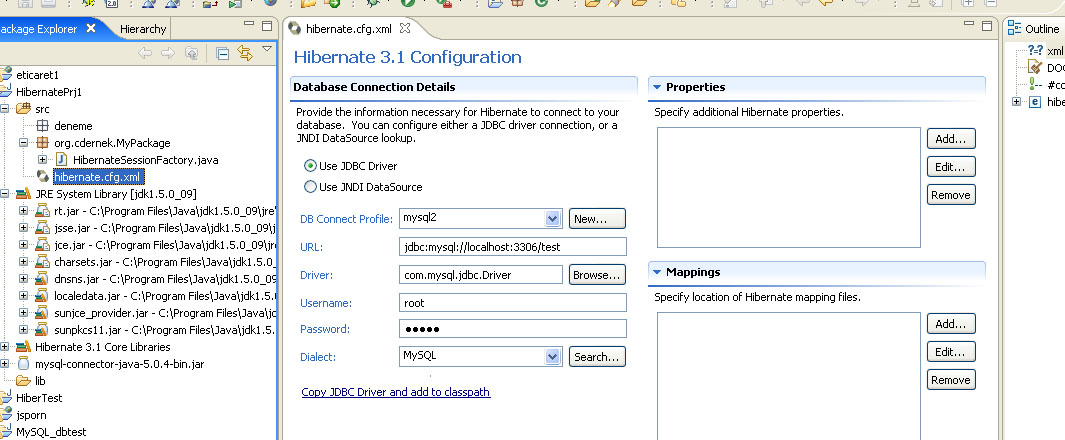
**seçili olmalı!!!**

**önceden tanımlanmış DB profile seçilmeli**

Veritabanı bağlantısı ile ilgili ayrıntılı bilgiler burada girilir (Next)



**seçili olmalı!!!**

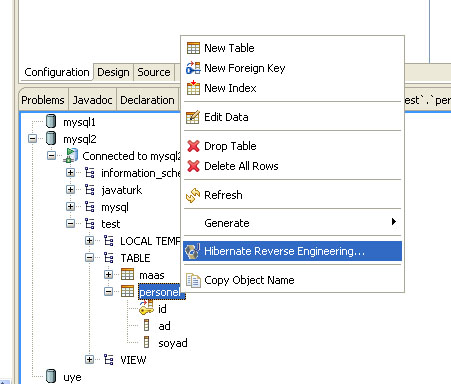


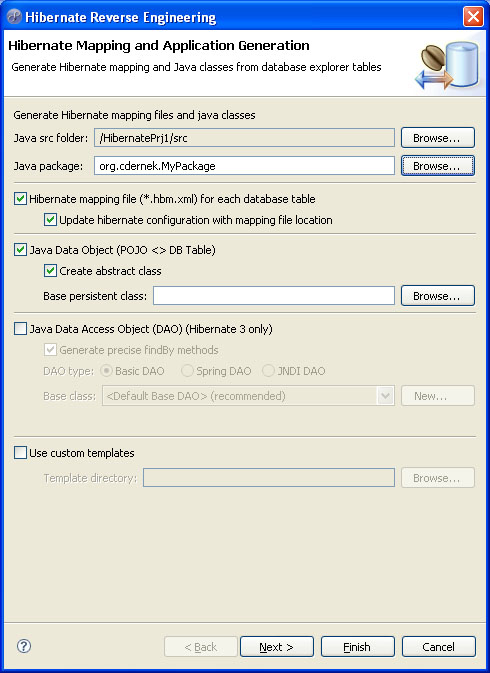
**hibernate.cfg.xml** dosyasının dizayn görünümü

**hibernate.cfg.xml** kaynak kodu

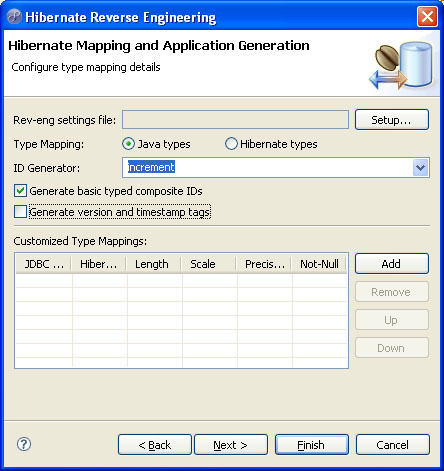
|  |
| --- |
| ……………………….  <hibernate-configuration>  <session-factory>  <property name="connection.username">root</property>  <property name="connection.url">jdbc:mysql://localhost:3306/test</property>  <property name="dialect">org.hibernate.dialect.MySQLDialect</property>  <property name="myeclipse.connection.profile">mysql2</property>  <property name="connection.password">root1</property>  <property name="connection.driver\_class">com.mysql.jdbc.Driver</property>  </session-factory>  </hibernate-configuration> |

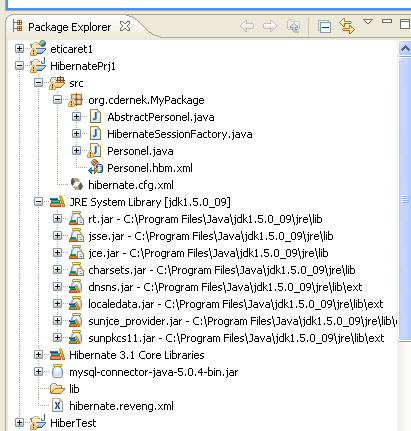
**Tablo (Tablolar) ile hibernate mappingi oluşturma**





**Bug var!!**





Package Explorer’in son hali

**derby**

|  |
| --- |
| create table "CLASSICCARS"."PERSONEL"(  "ID" INTEGER not null generated always as identity,  "AD" VARCHAR(40),  "SOYAD" VARCHAR(40),  "MAAS" INTEGER,  constraint "SQL110713062903210" primary key ("ID")  );  create unique index "SQL110713062903210" on "CLASSICCARS"."PERSONEL"("ID"); |

**mysql**

|  |
| --- |
| create table `test`.`personel`(  `no` INT not null auto\_increment,  `ad` VARCHAR(20),  primary key (`no`)  );  create unique index `PRIMARY` on `test`.`personel`(`no`); |

**Personel.java**

|  |
| --- |
| package org.cdernek.MyPackage;  public class Personel extends AbstractPersonel implements java.io.Serializable {  // Constructors  /\*\* default constructor \*/  public Personel() {  }    /\*\* full constructor \*/  public Personel(String ad, String soyad) {  super(ad, soyad);  }    } |

**AbstractPersonel.java**

|  |
| --- |
| package org.cdernek.MyPackage;  public abstract class AbstractPersonel implements java.io.Serializable {  private Integer id;  private String ad;  private String soyad;  public AbstractPersonel() {  }    public AbstractPersonel(String ad, String soyad) {  this.ad = ad;  this.soyad = soyad;  }    public Integer getId() {  return this.id;  }  public void setId(Integer id) {  this.id = id;  }  public String getAd() {  return this.ad;  }    public void setAd(String ad) {  this.ad = ad;  }  public String getSoyad() {  return this.soyad;  }    public void setSoyad(String soyad) {  this.soyad = soyad;  }  } |

**hibernate.cfg.xml** dosyasının son hali

|  |
| --- |
| <session-factory>  <property name="connection.username">root</property>  <property name="connection.url">  jdbc:mysql://localhost:3306/test  </property>  <property name="dialect">  org.hibernate.dialect.MySQLDialect  </property>  <property name="myeclipse.connection.profile">mysql2</property>  <property name="connection.password">root1</property>  <property name="connection.driver\_class">  com.mysql.jdbc.Driver  </property>  <**mapping resource="org/cdernek/MyPackage/Personel.hbm.xml" />**  </session-factory>  </hibernate-configuration> |

Eğer veritabanında primary key auto increment olarak ayarlanmışsa <generator class=”identity”> belirtilmeli. Eğer auto increment değil de biz bu özelliği istiyorsak <generator class=”increment”>

Tanımlanmalı. Eğer hem auto increment tanımlanmamış ama hem de biz herhangi bir değeri atamak istiyorsak bu durumda generator class satırı silinmeli.

**HiberTestOku.java**

|  |
| --- |
| package org.cdernek.MyPackage;  import java.util.Iterator;  import org.hibernate.Query;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  public class HiberTestOku {  public static void main(String[] args) {  Session session=null;  try{  **SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();**  **session=sessionFactory.openSession();**  **(Sorgu !!!)**  Query query=session.createQuery( **"from Personel"** );  List result = query.list();  Iterator iter = result.iterator();  while(iter.hasNext()){  Personel per=(Personel)iter.next();  System.out.println(per.getAd());  }    System.out.println("Kayit okutuldu...");  }catch(Exception e){  System.out.println(e);  }  finally{  session.flush();  session.close();  }  }  } |

**HiberOku.java**

|  |
| --- |
| package org.csystem.hiber;  import java.util.List;  import org.hibernate.Query;  import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.cfg.Configuration;  public class HiberOku {  public static void main(String[] args) {  Session session = null;  try {    **session = HibernateSessionFactory.getSession();**  Query query = session.createQuery("from Personel");  List<Personel> result = query.list();  for (Personel per : result) {  System.out.println(per.getAd());  }  System.out.println("Kayit okutuldu...");  } catch (Exception e) {  System.out.println(e);  }  finally {  // session.flush();  // session.close();  }  }  } |

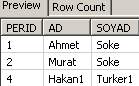
**HiberTestYaz.java**

|  |
| --- |
| **package** org.csystem.hiber;  **import** java.util.Date;  **import** org.hibernate.SessionFactory;  **import** org.hibernate.Transaction;  **import** org.hibernate.cfg.Configuration;  **import** org.hibernate.Session;  **public** **class** hiberYaz {  /\*\*  \* **@param** args  \*/  **public** **static** **void** main(String[] args) {  Session session =  HibernateSessionFactory.*getSession*();  Transaction tx = session.beginTransaction();  Personel per = **new** Personel();  per.setAd("murat");  per.setSoyad("soke");  session.save(per);  tx.commit();  System.*out*.println("kayit yapildi..");  }  } |

**Update**

****

|  |
| --- |
| Session session = HibernateSessionFactory.*getSession*();  Transaction tx = session.beginTransaction();  Personel per=(Personel)session.load(Personel.**class**, 4);  //Personel per=new Personel();  per.setAd("Hakan1");  per.setSoyad("Turker1");  session.save(per);  tx.commit();  System.*out*.println("kayit yapildi.."); |



**Annotation Kullanımı**

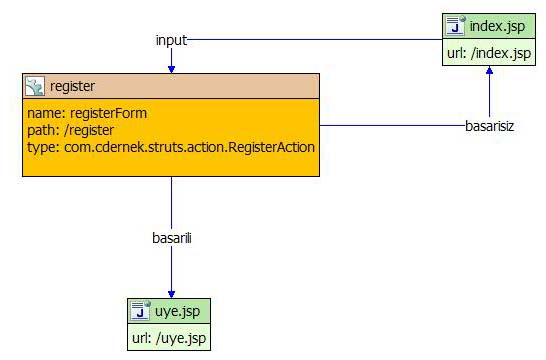
|  |
| --- |
| package org.csystem.hiber;  import javax.persistence.Column;  import javax.persistence.Entity;  import javax.persistence.GeneratedValue;  import static javax.persistence.GenerationType.IDENTITY;  import javax.persistence.Id;  import javax.persistence.Table;  /\*\*  \* Personel1 entity. @author MyEclipse Persistence Tools  \*/  @Entity  @Table(name = "PERSONEL1", schema = "CLASSICCARS")  public class Personel implements java.io.Serializable {  // Fields  private Integer perid;  private String ad;  private String soyad;  // Constructors  /\*\* default constructor \*/  public Personel() {  }  /\*\* full constructor \*/  public Personel(String ad, String soyad) {  this.ad = ad;  this.soyad = soyad;  }  // Property accessors  @Id  @GeneratedValue(strategy = IDENTITY)  @Column(name = "PERID", unique = true, nullable = false)  public Integer getPerid() {  return this.perid;  }  public void setPerid(Integer perid) {  this.perid = perid;  }  @Column(name = "AD", length = 20)  public String getAd() {  return this.ad;  }  public void setAd(String ad) {  this.ad = ad;  }  @Column(name = "SOYAD", length = 20)  public String getSoyad() {  return this.soyad;  }  public void setSoyad(String soyad) {  this.soyad = soyad;  }  } |

|  |
| --- |
| <?xml version=*'1.0'* encoding=*'UTF-8'*?>  <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">  <!-- Generated by MyEclipse Hibernate Tools. -->  <hibernate-configuration>  <session-factory>  <property name=*"dialect"*>  org.hibernate.dialect.DerbyDialect  </property>  <property name=*"connection.url"*>  jdbc:derby://localhost:1527/myeclipse  </property>  <property name=*"connection.username"*>classiccars</property>  <property name=*"connection.password"*>myeclipse</property>  <property name=*"connection.driver\_class"*>  org.apache.derby.jdbc.ClientDriver  </property>  <property name=*"myeclipse.connection.profile"*>  Personel.hbm.xml yerine Personel sınıfı geldiğine dikkat  MyEclipse Derby  </property>  <mapping class=*"org.csystem.hiber.Personel"* />  </session-factory>  </hibernate-configuration> |

|  |
| --- |
| SessionFactory sessionFactory = new AnnotationConfiguration().configure()  .buildSessionFactory();  Session session = sessionFactory.openSession();  Query query = session.createQuery("from Product");  List<Product> products = query.list();  **for** (Product product : products) {  System.*out*.println(product.getProductname());  } |

**Struts ve Hibernate’in beraber kullanıldığı bir örnek**

User ve password girildikten sonra hibernate kullanılarak uye tablosundan kişinin üye olup olmadığı kontrol ediliyor. Eğer üye ise **uye.jsp’ye** değilse **index.jsp’ye** yani başlangıca tekrar gönderiliyor.



**Strut’ta sonuçların Görüntülenmesi**

|  |
| --- |
| request.setAttribute("users", list); |

|  |
| --- |
| <logic:present name="users">  . . .  <logic:iterate id**="user"** name="users">  <bean:write name**="user"** property="ad"/><br>  </logic:iterate>  . . .  </logic:present> |

**struts-config.xml**

|  |
| --- |
| **<form-beans >**  **<form-bean name="registerForm" type="com.cdernek.struts.form.RegisterForm" />**  **</form-beans>**  **<global-exceptions />**  **<global-forwards />**  **<action-mappings >**  **<action**  **attribute="registerForm"**  **input="/index.jsp"**  **name="registerForm"**  **path="/register"**  **scope="request"**  **type="com.cdernek.struts.action.RegisterAction">**  **<forward name="basarisiz" path="/index.jsp" />**  **<forward name="basarili" path="/uye.jsp" />**  **</action>**  **</action-mappings>** |

**package explorer görüntüsü**

|  |  |  |
| --- | --- | --- |
|  |  |  |

**hibernate.cfg.xml**

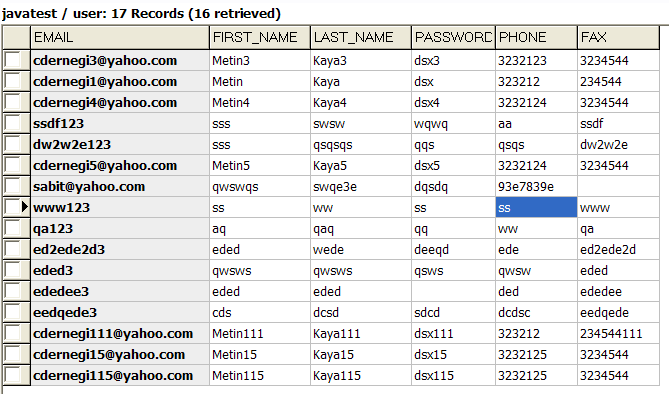
|  |
| --- |
| **<hibernate-configuration>**  **<session-factory>**  **<property name="connection.username">root</property>**  **<property name="connection.url">**  **jdbc:mysql://localhost:3306/javatest**  **</property>**  **<property name="dialect">**  **org.hibernate.dialect.MySQLDialect**  **</property>**  **<property name="myeclipse.connection.profile">mysql</property>**  **<property name="connection.driver\_class">**  **com.mysql.jdbc.Driver**  **</property>**  **<mapping resource="org/hiber/Uye.hbm.xml" />**  **</session-factory>**  **</hibernate-configuration>** |

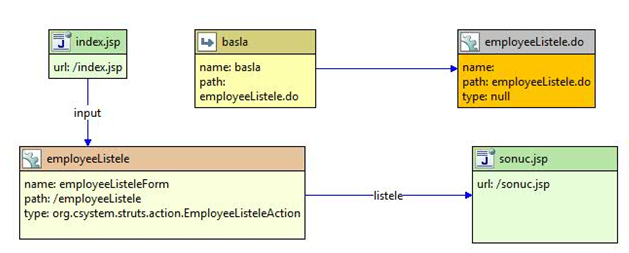
**Uye.hbm.xml**

|  |
| --- |
| **<hibernate-mapping>**  **<class name="org.hiber.Uye" table="uye" catalog="javatest">**  **<id name="id" type="java.lang.Integer">**  **<column name="id" />**  **<generator class="increment" />**  **</id>**  **<property name="user" type="java.lang.String">**  **<column name="user" length="10" not-null="true" />**  **</property>**  **<property name="password" type="java.lang.String">**  **<column name="password" length="10" not-null="true" />**  **</property>**  **</class>**  **</hibernate-mapping>** |

**RegisterAction.java**

|  |
| --- |
| **import java.util.Iterator;**  **import org.apache.struts.action.\*Action;**  **import org.hiber.Uye;**  **import org.hibernate.\*;**  **import org.hibernate.cfg.Configuration;**  **import com.cdernek.struts.form.RegisterForm;**  **...**  **public class RegisterAction extends Action {**    **public ActionForward execute(ActionMapping mapping, ActionForm form,**  **HttpServletRequest request, HttpServletResponse response) {**  **RegisterForm registerForm = (RegisterForm) form;// TODO Auto-generated method stub**  **String user=registerForm.getUser();**  **String password=registerForm.getPassword();**  **Session session=null;**  **try{**  **SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();**  **session=sessionFactory.openSession();**  **Query query=session.createQuery("from Uye where user='"+user+"' and password='"+password+"'");**  **List result = query.list();**  **Iterator iter = result.iterator();**  **if (iter.hasNext())**  **return**  **mapping.findForward("basarili");**  **}catch(Exception e){**  **System.*out*.println(e);**  **}**  **return mapping.findForward("basarisiz");**  **}**  **}** |

****

****

**EmployeeListeleAction**

|  |
| --- |
| **public** ActionForward execute(ActionMapping mapping, ActionForm form,  HttpServletRequest request, HttpServletResponse response) {  EmployeeListeleForm employeeListeleForm = (EmployeeListeleForm) form;  EmployeeDAO eD = **new** EmployeeDAO();  List<Employee> employees = eD.findAll();  request.setAttribute("employees", employees);  /\*for (Employee employee : employees) {  System.out.println(employee.getFirstname());  }\*/    **return** mapping.findForward("listele");  } |

**index.jsp**

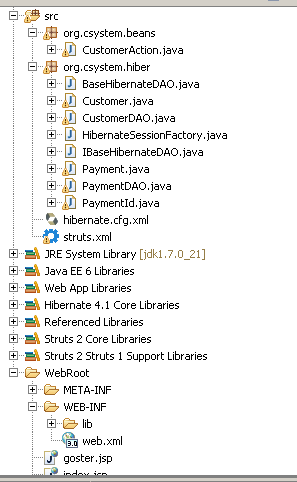
|  |
| --- |
| <logic:forward name="basla"/> |

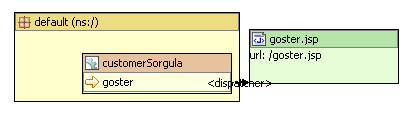
**sonuc.jsp**

|  |
| --- |
| <body>  <logic:iterate id=*"employee"* name=*"employees"*>  <bean:write name=*"employee"* property=*"firstname"*/><br>  </logic:iterate> |

|  |
| --- |
| <global-forwards >  <forward name=*"basla"* path=*"employeeListele.do"* />  </global-forwards>  <action-mappings >  <action  attribute=*"employeeListeleForm"*  input=*"/index.jsp"*  name=*"employeeListeleForm"*  path=*"/employeeListele"*  scope=*"request"*  type=*"org.csystem.struts.action.EmployeeListeleAction"*>  <set-property property=*"cancellable"* value=*"true"* />  <forward name=*"listele"* path=*"/sonuc.jsp"* />  </action>  </action-mappings> |

**Struts2 Örneği**





|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.1//EN" "http://struts.apache.org/dtds/struts-2.1.dtd">  <struts>  <package name=*"default"* namespace=*"/"* extends=*"struts-default"*>  <action name=*"customerSorgula"*  class=*"org.csystem.beans.CustomerAction"*>  <result name=*"goster"*>/goster.jsp</result>  </action></package></struts> |

|  |
| --- |
| **package** org.csystem.beans;  **import** java.util.ArrayList;  **import** java.util.List;  **import** org.csystem.hiber.Customer;  **import** org.csystem.hiber.CustomerDAO;  **import** com.opensymphony.xwork2.ActionSupport;  **public** **class** CustomerAction **extends** ActionSupport {  List<String> custNames = **new** ArrayList<String>();  **public** List<String> getCustNames() {  **return** custNames;  }  **public** **void** setCustNames(List<String> custNames) {  **this**.custNames = custNames;  }  **public** String execute() {  CustomerDAO cDAO = **new** CustomerDAO();  List<Customer> custs = cDAO.findAll();  **for** (Customer customer : custs) {  custNames.add(customer.getContactfırstname());  }  **return** "goster";  }  } |

|  |
| --- |
| **package** org.csystem.hiber;  **import** java.util.HashSet;  **import** java.util.Set;  **import** javax.persistence.CascadeType;  **import** javax.persistence.Column;  **import** javax.persistence.Entity;  **import** javax.persistence.FetchType;  **import** javax.persistence.GeneratedValue;  **import** **static** javax.persistence.GenerationType.*IDENTITY*;  **import** javax.persistence.Id;  **import** javax.persistence.OneToMany;  **import** javax.persistence.Table;  /\*\*  \* Customer entity. **@author** MyEclipse Persistence Tools  \*/  @Entity  @Table(name = "CUSTOMER", schema = "CLASSICCARS")  **public** **class** Customer **implements** java.io.Serializable {  // Fields  **private** Integer customernumber;  **private** String customername;  **private** String contactlastname;  **private** String contactfırstname;  **private** String phone;  **private** String addresslıne1;  **private** String addresslıne2;  **private** String cıty;  **private** String state;  **private** String postalcode;  **private** String country;  **private** Integer salesrepemployeenumber;  **private** Double credıtlımıt;  **private** Set<Payment> payments = **new** HashSet<Payment>(0);  // Constructors  /\*\* default constructor \*/  **public** Customer() {  }  /\*\* full constructor \*/  **public** Customer(String customername, String contactlastname,  String contactfırstname, String phone, String addresslıne1,  String addresslıne2, String cıty, String state, String postalcode,  String country, Integer salesrepemployeenumber, Double credıtlımıt,  Set<Payment> payments) {  **this**.customername = customername;  **this**.contactlastname = contactlastname;  **this**.contactfırstname = contactfırstname;  **this**.phone = phone;  **this**.addresslıne1 = addresslıne1;  **this**.addresslıne2 = addresslıne2;  **this**.cıty = cıty;  **this**.state = state;  **this**.postalcode = postalcode;  **this**.country = country;  **this**.salesrepemployeenumber = salesrepemployeenumber;  **this**.credıtlımıt = credıtlımıt;  **this**.payments = payments;  }  // Property accessors  @Id  @GeneratedValue(strategy = *IDENTITY*)  @Column(name = "CUSTOMERNUMBER", unique = **true**, nullable = **false**)  **public** Integer getCustomernumber() {  **return** **this**.customernumber;  }  **public** **void** setCustomernumber(Integer customernumber) {  **this**.customernumber = customernumber;  }  @Column(name = "CUSTOMERNAME", length = 50)  **public** String getCustomername() {  **return** **this**.customername;  }  **public** **void** setCustomername(String customername) {  **this**.customername = customername;  }  @Column(name = "CONTACTLASTNAME", length = 50)  **public** String getContactlastname() {  **return** **this**.contactlastname;  }  **public** **void** setContactlastname(String contactlastname) {  **this**.contactlastname = contactlastname;  }  @Column(name = "CONTACTFIRSTNAME", length = 50)  **public** String getContactfırstname() {  **return** **this**.contactfırstname;  }  **public** **void** setContactfırstname(String contactfırstname) {  **this**.contactfırstname = contactfırstname;  }  @Column(name = "PHONE", length = 50)  **public** String getPhone() {  **return** **this**.phone;  }  **public** **void** setPhone(String phone) {  **this**.phone = phone;  }  @Column(name = "ADDRESSLINE1", length = 50)  **public** String getAddresslıne1() {  **return** **this**.addresslıne1;  }  **public** **void** setAddresslıne1(String addresslıne1) {  **this**.addresslıne1 = addresslıne1;  }  @Column(name = "ADDRESSLINE2", length = 50)  **public** String getAddresslıne2() {  **return** **this**.addresslıne2;  }  **public** **void** setAddresslıne2(String addresslıne2) {  **this**.addresslıne2 = addresslıne2;  }  @Column(name = "CITY", length = 50)  **public** String getCıty() {  **return** **this**.cıty;  }  **public** **void** setCıty(String cıty) {  **this**.cıty = cıty;  }  @Column(name = "STATE", length = 50)  **public** String getState() {  **return** **this**.state;  }  **public** **void** setState(String state) {  **this**.state = state;  }  @Column(name = "POSTALCODE", length = 15)  **public** String getPostalcode() {  **return** **this**.postalcode;  }  **public** **void** setPostalcode(String postalcode) {  **this**.postalcode = postalcode;  }  @Column(name = "COUNTRY", length = 50)  **public** String getCountry() {  **return** **this**.country;  }  **public** **void** setCountry(String country) {  **this**.country = country;  }  @Column(name = "SALESREPEMPLOYEENUMBER")  **public** Integer getSalesrepemployeenumber() {  **return** **this**.salesrepemployeenumber;  }  **public** **void** setSalesrepemployeenumber(Integer salesrepemployeenumber) {  **this**.salesrepemployeenumber = salesrepemployeenumber;  }  @Column(name = "CREDITLIMIT", precision = 52, scale = 0)  **public** Double getCredıtlımıt() {  **return** **this**.credıtlımıt;  }  **public** **void** setCredıtlımıt(Double credıtlımıt) {  **this**.credıtlımıt = credıtlımıt;  }  @OneToMany(cascade = CascadeType.*ALL*, fetch = FetchType.*LAZY*, mappedBy = "customer")  **public** Set<Payment> getPayments() {  **return** **this**.payments;  }  **public** **void** setPayments(Set<Payment> payments) {  **this**.payments = payments;  }  } |

hibernate-cfg.xml

|  |
| --- |
| <?xml version=*'1.0'* encoding=*'UTF-8'*?>  <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">  <!-- Generated by MyEclipse Hibernate Tools. -->  <hibernate-configuration>  <session-factory>  <property name=*"dialect"*>  org.hibernate.dialect.DerbyDialect  </property>  <property name=*"connection.url"*>  jdbc:derby://localhost:1527/myeclipse  </property>  <property name=*"connection.username"*>classiccars</property>  <property name=*"connection.password"*>myeclipse</property>  <property name=*"connection.driver\_class"*>  org.apache.derby.jdbc.ClientDriver  </property>  <property name=*"myeclipse.connection.profile"*>  MyEclipse Derby  </property>  <mapping class=*"org.csystem.hiber.Payment"* />  <mapping class=*"org.csystem.hiber.Customer"* />  </session-factory>  </hibernate-configuration> |

index.jsp

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"ISO-8859-1"*%>  <%@taglib uri=*"/struts-tags"* prefix=*"s"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <body>  <s:form action=*"customerSorgula"*>  <s:submit value=*"sorgula"* />  </s:form>  </body>  </html> |

goster.jsp

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"ISO-8859-1"*%>  <%@taglib uri=*"/struts-tags"* prefix=*"s"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <body>  <s:form>  <s:select name=*"custName"* list=*"custNames"*></s:select>  <s:submit />  </s:form>  </body>  </html> |

**Hibernate Query Language (HQL)**

Önce HQL’e bakalım:

**session.createQuery("from Category c where c.name like 'Laptop%'");**

Bir kriter vererek arama (*query by criteria* (QBC) ) ve örnek vererek arama (and *query by example* (QBE)) yapmak için Criteria API’sini görelim:

**session.createCriteria(Category.class)**

**.add( Expression.like("name", "Laptop%") );**

***Sorguların İcrası***

Query ve Criteria arayüzleri sorgu için birçok metoda sahiptirler. Uygulamamızda bu arayüzlerden birini kullanmak istersek Session’dan bir metod yardımıyla nesne elde edilir.

***Query Arayüzü***

Yeni bir Query nesnesi oluşturmak için, createQuery() veya createSQLQuery()metodu çağrılır.

createQuery()metodu HQL sorgusu hazırlar:

**Query hqlQuery = session.createQuery("from User");**

Mevcut veritabanının doğal (native) sözdizimini (syntax ) kullanarak, bir SQL sorgusunu oluşturmak için createSQLQuery()metodu kullanılımı **deprecate** olmuştur.

**~~Query sqlQuery = session.createSQLQuery(~~**

**~~"select {u.\*} from USERS {u}", "u",User.class);~~**

**Sayfa Sayfa Sonuç Alma (Paging)**

Her defasında 10 satır alınacağı varsayılsın:

**Query query =**

**session.createQuery("from User u order by u.firstName asc");**

**query.setFirstResult(1);**

**query.setMaxResults(10);**

setMaxResults(10), ilk 10 nesneyi çağırır.

**Criteria crit = session.createCriteria(User.class);**

**crit.addOrder( Order.asc("firstName") );**

**crit.setFirstResult(40);**

**crit.setMaxResults(20);**

**List results = crit.list();**

Kırkıncı nesneden başlayarak bir sonraki 20 nesneyi çekeriz. Görüldüğü gibi, SQL’de sayfalama için standart bir yol yoktur. Ancak Hibernate, herhangi özel bir veri tabanı için işlemi en verimli halde uygular.

Query ve Criteria arayüzleri ile metod–zincirleme tarzı kullanılabilir (Metod, void yerine nesne geri döndürmektedir. Evvelden yazılmış örnekleri metod-zincirleme tarzıyla yeniden yazalım:

**java.util.List results =**

**session.createQuery("from User u order by u.firstName asc").**

**setFirstResult(1).setMaxResults(5).list();**

**java.util.List results = session.createCriteria(User.class).addOrder( Order.asc("firstName") ).setFirstResult(1)**

**.setMaxResults(5).list();**

**İsimli Sorgulama**

Gerekmedikçe HQL stringlerinin java kodunun içinde çeşitli yerlere yayılmış olarak görünmesi istenmeyebilir. Hibernate isimli sorgulama adı verilen teknikle sorgulamaları dışarıda bir XML dosyasının içine gömülmesine izin vermektedir.

**User.hbm.xml**

|  |
| --- |
| **...**  **</class>**  **<query name="byName">from User u where u.firstName like :isim</query>**  **</hibernate-mapping>**  **...** |

**Java kodu içinde çağırma**

|  |
| --- |
| **...**  **java.util.List results = session.getNamedQuery("byName").setString("isim", "Mehmet").list();**  **Iterator iter=results.iterator();**  **while(iter.hasNext()){**  **User per=(User)iter.next(); System.*out*.println(per.getFirstName()+".."+per.getLastName());**  **}**  **...** |

**Kısıt Koyma**

**Bilinen yöntem:**

**from User u order by u.firstName**

|  |
| --- |
| **...**  **Criterion adCr = Expression.*eq*("firstName", "Metin");**  **Criteria crit = session.createCriteria(User.class);**  **crit.add(adCr);**  **User user = (User) crit.uniqueResult();**  **System.out.println(user.getFirstName());**  **...** |

**Karşılaştırma Operatörleri**

|  |
| --- |
| **Query query=session.createQuery("from User u where u.phone between 3232124 and 3232125");** |

|  |
| --- |
| **java.util.List results=session.createCriteria(User.class).add( Expression.between("phone",new String("3232124"), new String("3232125"))).list();** |

**from User user where user.firstname like "G%" and user.lastname like "K%"**

**session.createCriteria(User.class)**

**.add( Expression.like("firstname", "G%") )**

**.add( Expression.like("lastname", "K%") )**

**from User user where ( user.firstname like "G%" and user.lastname like "K%" ) or user.email in ( "foo@hibernate.org", "bar@hibernate.org" )**

**Criteria crit = session.createCriteria(User.class)**

**.add( Expression.or( Expression.and(**

**Expression.like("firstname", "G%"),**

**Expression.like("lastname", "K%")),**

**Expression.in("email", emails)));**

**Çoklu Tablo İlişkileri**

|  |  |
| --- | --- |
|  |  |

Örnek uygulamada takım ve oyuncu olarak iki sınıf tanımlanacaktır. Bir takımda birden fazla oyuncu olabileceği için aradaki ilişki bire-birçok ilişkidir.

|  |
| --- |
| <hibernate-mapping>  <class name="example.Team" table="takımlar">  <id name="id" column="takım\_id" type="long" unsaved-value="null">  <generator class="hilo"/> </id>  <property name="name" column="team\_name" type="string" length="15" not-null="true"/>  <property name="şehir" column="şehir" type="string" length="15" not-null="true"/>  <set name="oyuncular" cascade="all" inverse="true" lazy="true">  <key column="takım\_id"/>  <one-to-many class="example.Player"/>  </set>  </class>  hibernate-mapping> |

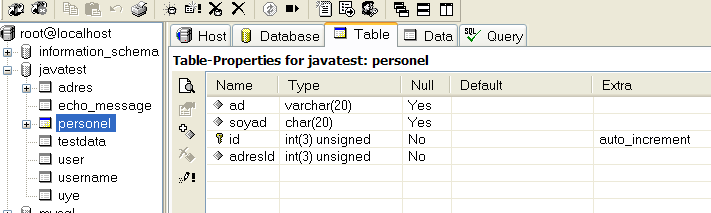
|  |
| --- |
| <hibernate-mapping>  <class name="example.Player" table="oyuncular">  <id name="id" column="oyuncu\_id" type="long" unsaved-value="null"> <generator class="hilo"/> </id>  <property name="firstName" column="adı" type="string" length="12" not-null="true"/>  <property name="lastName" column="soyadı" type="string" length="15" not-null="true"/>  <property name="draftDate" column="draft\_date" type="date"/>  <property name="annualMaaş" column="maaş" type="float"/>  <property name="jerseyNumber" column="sırt\_numarası" type="integer" length="2" not-null="true"/>  <many-to-one name="team" class="example.Team" column="takım\_id"/> </class>  </hibernate-mapping> |

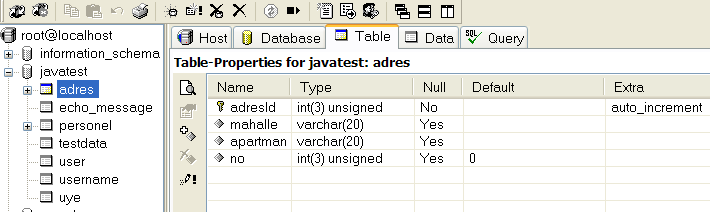
|  |
| --- |
| <set name="oyuncular" cascade="all" inverse="true" lazy="true"> <key column="takım\_id"/> <one-to-many class="example.Player"/> </set> |

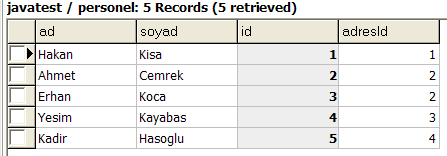
yukarıda görülen *lazy=”true” ifadesi* bir takım veri tabanından çekildiğinde, takımın oyuncularının kümesi uygulama takıma erişene kadar oluşturulmayacaktır. i*nverse=”true”* kesimi ise bir oyuncudan , o oyuncunun oynadığı takımı belirleyebileceğimizi gösterir. Bunu yapabilmemizi oyuncular için eşleme kütüğüne yazdığımız şu kesim sağlar:

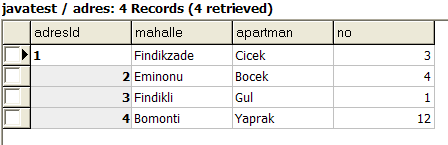
|  |
| --- |
| <many-to-one name="team" class="example.Team" column="takım\_id"/> |

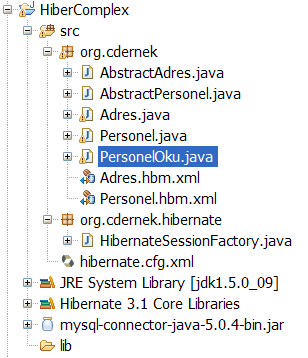
**Uygulama**











|  |
| --- |
| **public abstract class AbstractAdres implements java.io.Serializable {**  **private Integer adresId;**  **private String mahalle;**  **private String apartman;**  **private Integer no;**  **private Personel person;**  **...**  **public Personel getPersonel() {**  **return person;**  **}**  **public void setPersonel(Personel person) {**  **this.person = person;**  **}** |

|  |
| --- |
| **public abstract class AbstractPersonel implements java.io.Serializable {**  **private Integer id;**  **private String ad;**  **private String soyad;**  **private Integer adresId;**  **private Adres adres;**  **...**  **public Adres getAdres() {**  **return adres;**  **}**  **public void setAdres(Adres adres) {**  **this.adres = adres;**  **}** |

|  |
| --- |
| **<hibernate-configuration>**  **<session-factory>**  **<property name="connection.username">root</property>**  **<property name="connection.url">**  **jdbc:mysql://localhost:3306/javatest**  **</property>**  **<property name="dialect">**  **org.hibernate.dialect.MySQLDialect**  **</property>**  **<property name="myeclipse.connection.profile">**  **mysql(PersonelAdres)**  **</property>**  **<property name="connection.driver\_class">**  **com.mysql.jdbc.Driver**  **</property>**  **<mapping resource="org/cdernek/Adres.hbm.xml" />**  **<mapping resource="org/cdernek/Personel.hbm.xml" />**  **</session-factory>**  **</hibernate-configuration>** |

**Adres.hbm.xml**

|  |
| --- |
| **<hibernate-mapping>**  **<class name="org.cdernek.Adres" table="adres" catalog="javatest">**  **<id name="adresId" type="java.lang.Integer">**  **<column name="adresId" />**  **<generator class="foreign">**  **<param name="property">personel</param>**  **</generator>**  **</id>**  **<property name="mahalle"**  **type="java.lang.String">**  **<column name="mahalle" length="20" />**  **</property>**  **<property name="apartman"**  **type="java.lang.String">**  **<column name="apartman" length="20" />**  **</property>**  **<property name="no"**  **type="java.lang.Integer">**  **<column name="no" />**  **</property>**  **<many-to-one name="personel"**  **class="org.cdernek.Personel"**  **column="adresId" insert="false"**  **update="false"/>**  **</class>**  **</hibernate-mapping>** |

|  |
| --- |
| **<hibernate-mapping>**  **<class name="org.cdernek.Personel" table="personel"**  **Personel.hbm.xml**  **catalog="javatest">**  **<id name="id" type="java.lang.Integer">**  **<column name="id" />**  **<generator class="increment" />**  **</id>**  **<property name="ad" type="java.lang.String">**  **<column name="ad" length="20" />**  **</property>**  **<property name="soyad" type="java.lang.String">**  **<column name="soyad" length="20" />**  **</property>**  **<property name="adresId" type="java.lang.Integer">**  **<column name="adresId" not-null="true" />**  **</property>**  **<many-to-one name="adres" class="org.cdernek.Adres" column="adresId" insert="false" update="false"/>**  **</class>**  **</hibernate-mapping>** |

**PersonelOku.java**

|  |
| --- |
| **Session session=null;**  **try{**  **SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();**  **session=sessionFactory.openSession();**    **Query query=session.createQuery("from Personel");**    **List result = query.list();**  **Iterator iter = result.iterator();**    **while(iter.hasNext()){**  **Personel per=(Personel)iter.next();**        **System.*out*.println(per.getAd()+" "+per.getAdres().getApartman());**  **}**    **System.*out*.println("Kayit okutuldu...");**  **}catch(Exception e){**  **System.*out*.println(e);**  **}**  **finally{**  **session.flush();**  **session.close();**  **}** |

**Örnekler**

**hibernate.cfg.xml**

|  |
| --- |
| **<?xml version='1.0' encoding='UTF-8'?>**  **<!DOCTYPE hibernate-configuration PUBLIC**  **"-//Hibernate/Hibernate Configuration DTD 3.0//EN"**  **"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">**  **<!-- Generated by MyEclipse Hibernate Tools. -->**  **<hibernate-configuration>**  **<session-factory>**  **<property name="connection.username">root</property>**  **<property name="connection.url">**  **jdbc:mysql://localhost:3306/test**  **</property>**  **<property name="dialect">**  **org.hibernate.dialect.MySQLDialect**  **</property>**  **<property name="myeclipse.connection.profile">**  **test(MySQL)**  **</property>**  **<property name="connection.driver\_class">**  **com.mysql.jdbc.Driver**  **</property>**  **<mapping resource="org/cdernek/hiber/Personel.hbm.xml" />**  **</session-factory>**  **</hibernate-configuration>** |

**Personel.hbm.xml**

|  |
| --- |
| **<hibernate-mapping>**  **<class name="org.cdernek.hiber.Personel" table="personel" catalog="test">**  **<id name="perId" type="java.lang.Integer">**  **<column name="perID" />**  **<generator class="native" />**  **</id>**  **<property name="ad" type="java.lang.String">**  **<column name="ad" length="20" not-null="true" />**  **</property>**  **<property name="soyad" type="java.lang.String">**  **<column name="soyad" length="20" not-null="true" />**  **</property>**  **</class>**  **</hibernate-mapping>** |

|  |
| --- |
| CREATE TABLE `personel` (  `ad` varchar(20) NOT NULL,  `soyad` varchar(20) NOT NULL,  `perID` int(3) NOT NULL AUTO\_INCREMENT,  PRIMARY KEY (`perID`)  ) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=latin1;  INSERT INTO test.personel (ad, soyad)  VALUES ('Hakan', 'Kisa');    INSERT INTO test.personel (ad, soyad)  VALUES ('Osman', 'Halas');  INSERT INTO test.personel (ad, soyad)  VALUES ('Burak', 'Kisa'); |

**hibernateOku.java**

|  |
| --- |
| **SessionFactory sessionFactory = new Configuration().configure()**  **.buildSessionFactory();**  **session = sessionFactory.openSession();**  **Query query = session.createQuery("from Personel");**  **List result = query.list();**  **Iterator iter = result.iterator();**  **while (iter.hasNext()) {**  **Personel per = (Personel) iter.next();**    **String ad=per.getAd();**  **String soyad=per.getSoyad();**  **System.*out*.println(ad+" "+soyad);**    **}** |

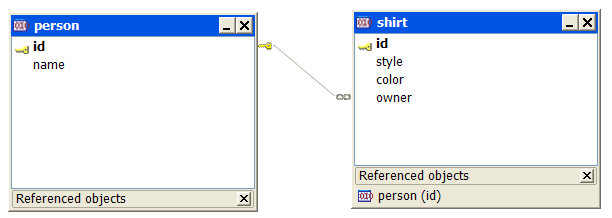
**HiberTestYaz.java**

|  |
| --- |
| **package org.cdernek.hiber;**  **import org.hibernate.Session;**  **import org.hibernate.SessionFactory;**  **import org.hibernate.Transaction;**  **import org.hibernate.cfg.Configuration;**  **public class HiberTestYaz {**  **/\*\***  **\* @param args**  **\*/**  **public static void main(String[] args) {**  **// TODO Auto-generated method stub**  **Session session = null;**  **try {**  **SessionFactory sessionFactory = new Configuration().configure()**  **.buildSessionFactory();**  **session = sessionFactory.openSession();**  **Transaction trns = session.beginTransaction();**  **Personel personel = new Personel();**  **personel.setAd("Huseyin");**  **personel.setSoyad("Burak");**  **session.save(personel);**  **trns.commit();**  **System.*out*.println("Kayit Yazildi...");**  **} catch (Exception e) {**  **System.*out*.println(e);**  **}**  **finally {**  **session.flush();**  **session.close();**  **}**  **}**  **}** |

**personShirt.sql**

|  |
| --- |
| CREATE TABLE `person` (  `id` smallint(5) UNSIGNED NOT NULL AUTO\_INCREMENT,  `name` char(60) NOT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=latin1;  CREATE TABLE `shirt` (  `id` smallint(5) UNSIGNED NOT NULL AUTO\_INCREMENT,  `style` ENUM('t-shirt','polo','dress') NOT NULL,  `color` ENUM('red','blue','orange','white','black') NOT NULL,  `owner` smallint(5) UNSIGNED NOT NULL,  PRIMARY KEY (`id`),  KEY `fk\_owner` (`owner`),  CONSTRAINT `fk\_owner` FOREIGN KEY (`owner`) REFERENCES `person` (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=8 DEFAULT CHARSET=latin1; |

|  |
| --- |
| CREATE TABLE shirt (  id **INTEGER** **UNSIGNED** NOT NULL AUTO\_INCREMENT,  style **VARCHAR**(**10**) NOT NULL,  color **VARCHAR**(**10**)NOT NULL,  owner **INTEGER** **UNSIGNED** NOT NULL,  PRIMARY KEY (id),  KEY fk\_owner (owner),  CONSTRAINT fk\_owner FOREIGN KEY (owner) REFERENCES person (id)  ) ENGINE=InnoDB ; |



**shirt person**

|  |  |
| --- | --- |
| shirt | person |

**Person.hbm.xml**

|  |
| --- |
| **<hibernate-mapping>**  **<class name="org.cdernek.hiber.Person" table="person" catalog="hibernate">**  **<id name="id" type="java.lang.Short">**  **<column name="id" />**  **<generator class="native" />**  **</id>**  **<property name="name" type="java.lang.String">**  **<column name="name" length="60" not-null="true" />**  **</property>**  **<set name="shirts" inverse="true">**  **<key>**  **<column name="owner" not-null="true" />**  **</key>**  **<one-to-many class="org.cdernek.hiber.Shirt" />**  **</set>**  **</class>**  **</hibernate-mapping>** |

**Shirt.hbm.xml**

|  |
| --- |
| **<hibernate-mapping>**  **<class name="org.cdernek.hiber.Shirt" table="shirt" catalog="hibernate">**  **<id name="id" type="java.lang.Short">**  **<column name="id" />**  **<generator class="native" />**  **</id>**  **<many-to-one name="person" class="org.cdernek.hiber.Person" fetch="select">**  **<column name="owner" not-null="true" />**  **</many-to-one>**  **<property name="style" type="java.lang.String">**  **<column name="style" length="8" not-null="true" />**  **</property>**  **<property name="color" type="java.lang.String">**  **<column name="color" length="6" not-null="true" />**  **</property>**  **</class>**  **</hibernate-mapping>** |

**HibernateOku.java**

|  |
| --- |
| package org.cdernek.hiber;  import java.util.Iterator;  import java.util.List;  import java.util.Set;  import org.hibernate.\*;  import org.hibernate.cfg.\*;  //import org.apache.log4j.Logger;  public class HibernateOku {  //private Logger log =Logger.getLogger(this.getClass());  public static void main(String[] args) {  Session session = null;  try {  SessionFactory sessionFactory = new Configuration().configure()  .buildSessionFactory();  session = sessionFactory.openSession();  Query query = session.createQuery("from Person");  List result = query.list();  Iterator iter = result.iterator();  while (iter.hasNext()) {  Person per = (Person) iter.next();  System.out.println(per.getName());  System.out.println("----------------------");  Set set = per.getShirts();  Iterator itr = set.iterator();  while (itr.hasNext()) {  Shirt sh = (Shirt) itr.next();  System.out.println(sh.getStyle() + " " + sh.getColor());  }  System.out.println();    }  } // try  catch (Exception e) {  System.out.println(e.getMessage());  } finally {  // Actual contact insertion will happen at this step  session.flush();  session.close();  }  }  } |

**Ekran Çıktısı**

Antonio Paz

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

dress white

polo blue

t-shirt blue

Lilliana Angelovska

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

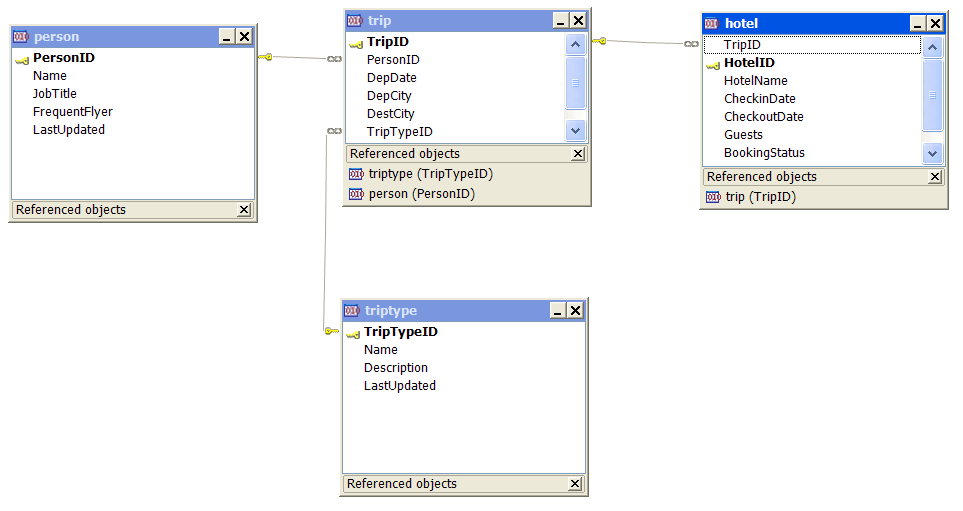
dress blue

t-shirt white

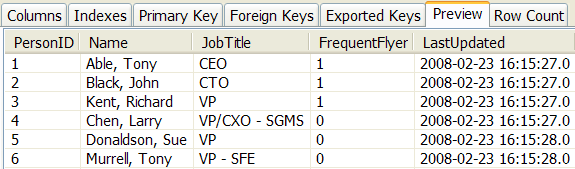
polo red

dress orange

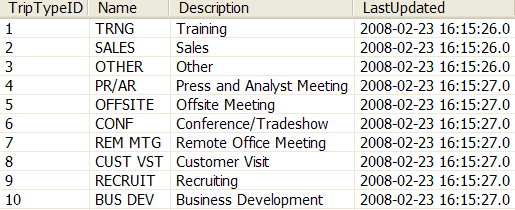
**Travel Örneği**



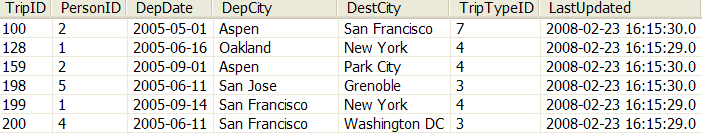
**person**



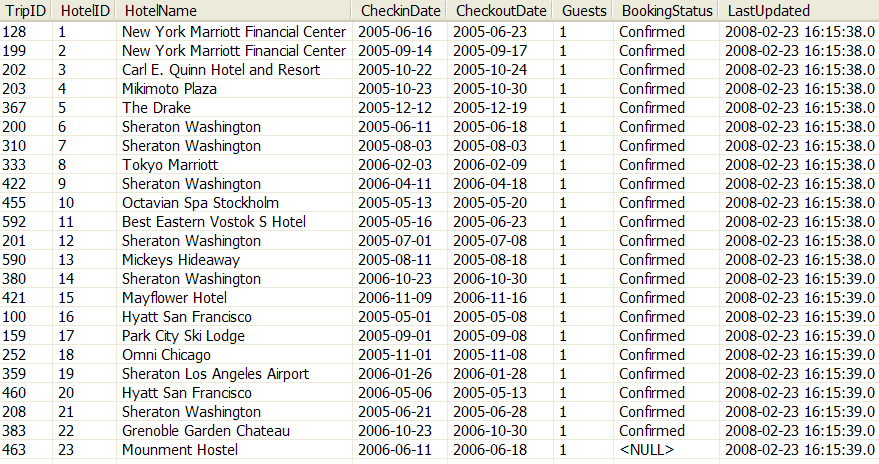
**triptype**



**trip**



**hotel**



**Person.hbm.xml**

|  |
| --- |
| <hibernate-mapping>  <class name=*"org.cdernek.hiber.Person"* table=*"PERSON"* schema=*"TRAVEL"*>  <id name=*"personid"* type=*"java.lang.Integer"*>  <column name=*"PERSONID"* />  <generator class=*"native"* />  </id>  <property name=*"name"* type=*"java.lang.String"*>  <column name=*"NAME"* length=*"50"* />  </property>  <property name=*"jobtitle"* type=*"java.lang.String"*>  <column name=*"JOBTITLE"* length=*"50"* />  </property>  <property name=*"frequentflyer"* type=*"java.lang.Short"*>  <column name=*"FREQUENTFLYER"* />  </property>  <property name=*"lastupdated"* type=*"java.sql.Timestamp"*>  <column name=*"LASTUPDATED"* length=*"26"* />  </property>  <set name=*"trips"* inverse=*"true"*>  <key>  <column name=*"PERSONID"* not-null=*"true"* />  </key>  <one-to-many class=*"org.cdernek.hiber.Trip"* />  </set>  </class>  </hibernate-mapping> |

**Trip.hbm.xml**

|  |
| --- |
| <hibernate-mapping>  <class name=*"org.cdernek.hiber.Trip"* table=*"TRIP"* schema=*"TRAVEL"*>  <id name=*"tripid"* type=*"java.lang.Integer"*>  <column name=*"TRIPID"* />  <generator class=*"native"* />  </id>  <many-to-one name=*"triptype"* class=*"org.cdernek.hiber.Triptype"* fetch=*"select"*>  <column name=*"TRIPTYPEID"* not-null=*"true"* />  </many-to-one>  <many-to-one name=*"person"* class=*"org.cdernek.hiber.Person"* fetch=*"select"*>  <column name=*"PERSONID"* not-null=*"true"* />  </many-to-one>  <property name=*"depdate"* type=*"java.util.Date"*>  <column name=*"DEPDATE"* length=*"10"* />  </property>  <property name=*"depcity"* type=*"java.lang.String"*>  <column name=*"DEPCITY"* length=*"32"* />  </property>  <property name=*"destcity"* type=*"java.lang.String"*>  <column name=*"DESTCITY"* length=*"32"* />  </property>  <property name=*"lastupdated"* type=*"java.sql.Timestamp"*>  <column name=*"LASTUPDATED"* length=*"26"* />  </property>  <set name=*"hotels"* inverse=*"true"*>  <key>  <column name=*"TRIPID"* not-null=*"true"* />  </key>  <one-to-many class=*"org.cdernek.hiber.Hotel"* />  </set>  <set name=*"flights"* inverse=*"true"*>  <key>  <column name=*"TRIPID"* not-null=*"true"* />  </key>  <one-to-many class=*"org.cdernek.hiber.Flight"* />  </set>  <set name=*"carrentals"* inverse=*"true"*>  <key>  <column name=*"TRIPID"* not-null=*"true"* />  </key>  <one-to-many class=*"org.cdernek.hiber.Carrental"* />  </set>  </class>  </hibernate-mapping> |

**PersonList.java**

|  |
| --- |
| SessionFactory sesFact = **new** Configuration().configure().buildSessionFactory();  Session session = sesFact.openSession();  Query query = session.createQuery("from Person");  List<Person> persons = query.list();  **for** (Person person : persons) {  System.*out*.println(person.getName());  System.*out*.println("\*\*\*\*trips\*\*\*\*");  Set<Trip> trips = person.getTrips();  **for** (Trip trip : trips) {  System.*out*.println(trip.getDepcity());  Set <Flight> flights=trip.getFlights();  System.*out*.println("\*\*\*flights\*\*\*\*\*");  **for** (Flight flight : flights) {  System.*out*.println(flight.getAirlinename());  }  System.*out*.println("------------------");  }  } |

**Eticaret Örneği**

|  |
| --- |
| create table "APP"."URUN"(  "URUNID" INTEGER not null generated always as identity,  "URUNKOD" VARCHAR(20),  "URUNAD" VARCHAR(20),  "FIYAT" INTEGER,  constraint "SQL110908085703490" primary key ("URUNID")  );  create unique index "SQL110908085703490" on "APP"."URUN"("URUNID"); |

|  |
| --- |
| insert into APP.urun (urunAd, urunKod, fiyat) values ('elma','elm1',5);  insert into APP.urun (urunAd, urunKod, fiyat) values ('portakal','port1',4);  insert into APP.urun (urunAd, urunKod, fiyat) values ('nar','nar1',3); |

|  |  |
| --- | --- |
|  |  |

