Maven Update....

**If it’s a Maven Project we can put Hibernate Related files...**

Src/main/resources create new file Stock.hbm.xml

Src/main/resources create new file hibernate.cfg.xml

v 1.2 Student.java

**private** **int** id;

**private** String firstName,lastName; (generate getter and setter)

student.hbm.xml

***Ignore DTD it will auto place in both file***

<?xml version=*'1.0'* encoding=*'UTF-8'*?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 5.3//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>

<class name=*"p1.Student"* table=*"stu"*>

<id name=*"sid"*>

<generator class=*"increment"*></generator>

</id>

<property name=*"sName"*></property>

<property name=*"roll"*></property>

</class>

</hibernate-mapping>

Hibernate.cfg.xml

<?xml version=*'1.0'* encoding=*'UTF-8'*?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 5.3//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

<session-factory>

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

<property name=*"connection.url"*>jdbc:mysql://localhost:3306/school</property>

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

<property name=*"connection.password"*></property>

<property name=*"hbm2ddl.auto"*>update</property>

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

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

<mapping resource=*"employee.hbm.xml"*/>

</session-factory>

</hibernate-configuration>

Client

ServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure().build();

MetadataSources mds = **new** MetadataSources(ssr);

Metadata md = mds.getMetadataBuilder().build();

SessionFactory sf = md.getSessionFactoryBuilder().build();

Session s = sf.openSession();

Student ss = **new** Student();

ss.setsName("SAM");

ss.setRoll(22);

s.save(ss);

Transaction t = s.beginTransaction();

t.commit();

System.***out***.println("Done");

If we re run that same program then we will get constraint violation problem because of Primary Key

or else if we dont want to use ***Hibernate.cfg.xml*** we can go with 2nd Approach

Configuration c = **new** Configuration();

Properties p = **new** Properties();

p.put(Environment.***DRIVER***, "com.mysql.jdbc.Driver");

p.put(Environment.***URL***, "jdbc:mysql://localhost:3306/School");

p.put(Environment.***USER***, "root");

p.put(Environment.***PASS***, "root");

p.put(Environment.***HBM2DDL\_AUTO***, "update");

p.put(Environment.***SHOW\_SQL***, "true");

p.put(Environment.***DIALECT***, "org.hibernate.dialect.MySQL5Dialect");

c.setProperties(p);

c.addResource("student.hbm.xml");

ServiceRegistry sr = **new** StandardServiceRegistryBuilder().applySettings(p).build();

SessionFactory sf = c.buildSessionFactory(sr);

Session s = sf.openSession();

Student ss = **new** Student();

ss.setsName("SAM");

ss.setRoll(22);

s.save(ss);

Transaction t = s.beginTransaction();

t.commit();

v 1.3 FetchData.java

ServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure().build();

MetadataSources mds = **new** MetadataSources(ssr);

Metadata md = mds.getMetadataBuilder().build();

SessionFactory sf = md.getSessionFactoryBuilder().build();

Session s = sf.openSession();

Student byID = s.byId(Student.**class**).getReference(1);

System.***out***.println("byId " + byID);

Student byGet = s.get(Student.**class**, 1);

System.***out***.println("byGet " + byGet);

Student byLoad = s.load(Student.**class**, 1);

System.***out***.println("byLoad " + byLoad);

v 1.4 Client.java

Student byID = s.byId(Student.**class**).getReference(1);

ss.setsName("XYZ");

ss.setRoll(22);

s.update(ss);

Transaction t = s.beginTransaction();

t.commit();

v X Client.java

Student byID = s.byId(Student.**class**).getReference(1);

s.delete(byID);

Transaction t = s.beginTransaction();

t.commit();

**Inheritence in Hibernate START HERE**

v 1.5

**1)Table Per Hierarchy Example**

Vehicle.java **int** vid; **String** vname>> gt st

TwoWheeler.java **String** tcompany; **int** tmileage;

FourWheeler.java **String** fcompany; **int** fmileage;

vehicle.hbm.xml

<?xml version=*'1.0'* encoding=*'UTF-8'*?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name=*"p1.Vehicle"* table=*"Vehicles"* discriminator-value=*"vehi"*>

<id name=*"vid"*>

<generator class=*"increment"*></generator>

</id>

<discriminator column=*"type"* type=*"string"*></discriminator>

<property name=*"vname"*></property>

<subclass name=*"p1.TwoWheeler"* discriminator-value=*"twoWheeler"*>

<property name=*"tcompany"*></property>

<property name=*"tmileage"*></property>

</subclass>

<subclass name=*"p1.FourWheeler"* discriminator-value=*"fourWheeler"*>

<property name=*"fcompany"*></property>

<property name=*"fmileage"*></property>

</subclass>

</class>

</hibernate-mapping>

Client.java

SessionFactory sf = HibernateUtils.getSessionFactory();

Vehicle v = **new** Vehicle();

V.setvName(“Honda”);

TwoWheeler t = **new** TwoWheeler();

t.setVname("Splender");

t.setTcompany("Honda");

t.setTmileage(70);

s.persist(v);

s.persist(t);

tx.commit();

**2)Table Per Concrete Example**

v 1.6 use class,Getter and setter of above EX

Vehicle.hbm.xml

<?xml version=*'1.0'* encoding=*'UTF-8'*?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name=*"p1.Vehicle"* table=*"vehicle"*>

<id name=*"vid"*>

<generator class=*"increment"*></generator>

</id>

<property name=*"vname"*></property>

<union-subclass name=*"p1.TwoWheeler"* table=*"twoWheeler"*>

<property name=*"tcompany"*></property>

<property name=*"tmileage"*></property>

</union-subclass>

<union-subclass name=*"p1.FourWheeler"* table=*"fourWheeler"*>

<property name=*"fcompany"*></property>

<property name=*"fmileage"*></property>

</union-subclass>

</class>

</hibernate-mapping>

Client.java

SessionFactory sf = HibernateUtils.getSessionFactory();

Session s = sf.openSession();

Transaction tt = s.beginTransaction();

Vehicle v = **new** Vehicle();

v.setVname("Splender");

TwoWheeler t = **new** TwoWheeler();

t.setVname("Bike");

t.setTcompany("Honda");

t.setTmileage(70);

FourWheeler f = **new** FourWheeler();

f.setVname("Car");

f.setFcompany("Maruti");

f.setFmileage(110);

s.save(v);

s.save(t);

s.save(f);

tt.commit();

**3)Table Per Sub-Class Example**

v 1.7 use class,Getter and setter of above EX

Vehicle.hbm.xml

<?xml version=*'1.0'* encoding=*'UTF-8'*?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name=*"p1.Vehicle"* table=*"vehicle"*>

<id name=*"vid"*>

<generator class=*"increment"*></generator>

</id>

<property name=*"vname"*></property>

<joined-subclass name=*"p1.TwoWheeler"* table=*"twoWheel"*>

<key column=*"tid"*></key>

<property name=*"tcompany"*></property>

<property name=*"tmileage"*></property>

</joined-subclass>

<joined-subclass name=*"p1.FourWheeler"*>

<key column=*"fid"*></key>

<property name=*"fcompany"*></property>

<property name=*"fmileage"*></property>

</joined-subclass>

</class>

</hibernate-mapping>

Client.java

SessionFactory sf = HibernateUtils.getSessionFactory();

Vehicle v = **new** Vehicle();

v.setVname("Splender");

TwoWheeler t = **new** TwoWheeler();

t.setVname("Bike");

t.setTcompany("Honda");

t.setTmileage(70);

FourWheeler f = **new** FourWheeler();

f.setVname("Car");

f.setFcompany("Maruti");

f.setFmileage(110);

s.save(v);

s.save(t);

s.save(f);

tt.commit();

**Component Mapping**

Address.java **int** pincode; String city; ***Getter and Setter***

Employee.java **int** id; String name; Address add;

employee.hbm.xml

<hibernate-mapping>

<class name=*"p2.Employee"* table=*"three"*>

<id name=*"id"*>

<generator class=*"increment"*></generator>

</id>

<property name=*"name"*></property>

**<component name=*"add"* class=*"p2.Address"*>**

**<property name=*"pincode"*></property>**

**<property name=*"city"*></property>**

**</component>**

</class>

</hibernate-mapping>

**Collection in Hibernate START HERE**

**Also called One to Many Relations...**

We can map collection elements of Persistent class in Hibernate. You need to declare the type of collection in Persistent class from one of the following types:

* java.util.List
* java.util.Set
* java.util.Map

Mapping collection in mapping file

There are many sub elements of **<class>** elements to map the collection. They are **<list>**, **<bag>**, **<set>** and **<map>**. Let's see how we implement the list for the above class:

<list name="answers" table="ans100">

          <key column="qid"></key>

          <index column="type"></index>

          <element column="answer" type="string"></element>

</list>

There are three sub elements used in the list:

* **<key>** element is used to define the foreign key in this table based on the Question class identifier.
* **<index>** element is used to identify the type. List and Map are indexed collection.
* **<element>** is used to define the element of the collection.

The key element is used to define the foreign key in the joined table based on the original identity. The foreign key element is nullable by default. So, for non-nullable foreign key, we need to specify not-null attribute such as:

<key column="qid" not-**null**="true" ></key>

### Indexed collections

The collection elements can be categorized in two forms:

* **indexed**, and
* **non-indexed**

The List and Map collection are indexed whereas set and bag collections are non-indexed. Here, indexed collection means List and Map requires an additional element **<index>**.

Collection Elements

The collection elements can have value or entity reference (another class object). We can use one of the 4 elements

* **element**
* **component-element (component-mapping we saw above)**
* **one-to-many**, or
* **many-to-many**

The **element** is used for normal value such as **string**, **int** **etc**. whereas one-to-many and many-to-many are used to map entity reference.

**Example of List by String**

Question.java **int** id; String qname; List<String> ans;

Question.hbm.xml

**<hibernate-mapping>**

<class name=*"p1.Question"* table=*"Question"*>

<id name=*"id"*>

<generator class=*"increment"*></generator>

</id>

<property name=*"qname"*></property>

**<list name=*"ans"* table=*"Answer"*>**

**<key column=*"id"*></key>**

**<index column=*"type"*></index>**

**<element column=*"answer"* type=*"string"*></element>**

**</list>**

</class>

**</hibernate-mapping>**

Client.java

SessionFactory sf = HibernateUtils.getSessionFactory();

Session s = f.openSession();

Transaction t = s.beginTransaction();

List<String> li = **new** ArrayList<String>();

li.add("Program");

li.add("language");

Question q = **new** Question();

q.setQname("Java");

q.setAns(li);

s.persist(q);

t.commit();

**Example of List by Entity Like Class Answer**

Question.java **int** id; String qname; List<Answer> ans;

Answer.java **int** id; String answer; String postedby;

Question.hbm.xml

<**class** name=*"p1.Question"* table=*"one"*>

<id name=*"id"*>

<generator class=*"increment"*></generator>

</id>

<property name=*"qname"*></property>

<**list** name=*"ans"* cascade=*"all"*>

<key column=*"eid"*></key>

<index column=*"type"*></index>

<one-to-many class=*"p1.Answer"*/>

</**list**>

</**class**>

<**class** name=*"p1.Answer"* table=*"two"*>

<id name=*"id"*>

<generator class=*"increment"*></generator>

</id>

<property name=*"answer"*></property>

<property name=*"postedby"*></property>

</**class**>

Client.java

SessionFactory sf = HibernateUtils.getSessionFactory();

Session s = f.openSession();

Transaction t = s.beginTransaction();

Answer a = **new** Answer();

a.setAnswer("Language");

a.setPostedby("Sam");

Answer a1 = **new** Answer();

a1.setAnswer("Program");

a1.setPostedby("Harry");

List<Answer> li = **new** ArrayList<Answer>();

li.add(a1);

li.add(a);

Question q = **new** Question();

q.setQname("Java");

q.setAns(li);

s.save(q);

t.commit();

**Example of List Fetching Data**

FatchData.java

SessionFactory sf = HibernateUtils.getSessionFactory();

Session s = f.openSession();

Query q = s.createQuery("from Question");

List<Question> lq = q.list();

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

**Example of List Delete Data**

FatchData.java

SessionFactory sf = HibernateUtils.getSessionFactory();

Session s = f.openSession();

Question q = s.load(Question.**class**, 1);

s.delete(q);

s.beginTransaction().commit();

**It will delete Parent and child object both Means Question and Answer Related to that Question.**

**Mapping Bag in Collection Mapping**

If our persistent class has List object, we can map the List by list or bag element in the mapping file. The bag is just like List but it **doesn't require index element**

***Difference between list and bag is List maintain insertion order whereas bag does not maintain any order(unordered)***

**Example of bag by String**

**Take previous example just change list to bag and remove index tag from hbm file**

**Example of bag by Entity Like Class Answer**

**Take previous example just change list to bag and remove index tag from hbm file**

**Mapping Set in Collection**

The set element doesn't require index element. The one difference between List and Set is that, it stores only unique values.

***A <set> is similar to <bag> except that it can only store unique objects.***

**Example of set by String**

**Take previous example just change list to bag and remove index tag from hbm file**

**Example of set by Entity Like Class Answer**

**Take previous example just change bag to set and remove index tag from hbm file**

**Mapping Map in collection mapping**

As we know, list and map are index-based collections. In case of map, index column works as the key and element column works as the value.

**Example of Map by String**

Question.java **int** id; String qname; Map<String,String> ans;

Question.hbm.xml

<hibernate-mapping>

<class name=*"p1.Question"* table=*"one"*>

<id name=*"id"*>

<generator class=*"increment"*></generator>

</id>

<property name=*"name"*></property>

**<map name=*"ans"* table=*"two"*>**

**<key column=*"eid"*></key>**

**<index column=*"type"* type=*"string"*></index>**

**<element column=*"answer"* type=*"string"*></element>**

**</map>**

</class>

</hibernate-mapping>

Client.java

SessionFactory sf = HibernateUtils.getSessionFactory();

Session s = f.openSession();

Transaction t = s.beginTransaction();

Map<String, String> m = **new** HashMap<String, String>();

m.put("1", "A");

m.put("2", "B");

Question q = **new** Question();

q.setName("SAM");

q.setAns(m);

s.save(q);

t.commit();

**Example of Map by Entity Like Class Answer**

Question.java **int** id; String qname; Map<String,Answer> ans;

Answer.java **int** id; String answer; String postedby;

Question.hbm.xml

<hibernate-mapping>

<class name=*"p1.Question"* table=*"one"*>

<id name=*"qid"*>

<generator class=*"increment"*></generator>

</id>

<property name=*"qname"*></property>

<property name=*"qFee"*></property>

<map name=*"ans"* cascade=*"all"*>

<key column=*"kid"*></key>

<index column=*"ind"* type=*"string"*></index>

<one-to-many class=*"p1.Answer"*></one-to-many>

</map>

</class>

<class name=*"p1.Answer"* table=*"two"*>

<id name=*"aid"*>

<generator class=*"increment"*></generator>

</id>

<property name=*"answer"*></property>

<property name=*"postedby"*></property>

</class>

</hibernate-mapping>

Client.java

SessionFactory sf = HibernateUtils.getSessionFactory();

Session s = f.openSession();

Transaction t = s.beginTransaction();

Question q = **new** Question();

q.setQname("What is Java");

q.setqFee(1000);

Answer a1 = **new** Answer();

a1.setAnswer("Program");

a1.setPostedby("Sam");

Answer a2 = **new** Answer();

a2.setAnswer("Language");

a2.setPostedby("Pam");

Map<String, Answer> m = **new** HashMap<>();

m.put("1", a1);

m.put("2", a2);

q.setAns(m);

s.save(q);

t.commit();