**Association Mapping(Part-3):-**

**Focus on below Program’s comment:-**

1. **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">

<hibernate-configuration>

<session-factory>

<!-- driver details -->

<property name=*"hibernate.connection.driver\_class"*>oracle.jdbc.driver.OracleDriver</property>

<property name=*"hibernate.connection.url"*>jdbc:oracle:thin:@localhost:1521:xe</property>

<property name=*"hibernate.connection.username"*>system</property>

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

<!-- Hibernate Properties -->

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

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

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

<!-- link mapping files -->

<mapping resource=*"com/dev/mapping/entity/Account.hbm.xml"*/>

<mapping resource=*"com/dev/mapping/entity/Person.hbm.xml"*/>

</session-factory>

</hibernate-configuration>

2.)**domain classes:-**

(i) **Account.java**

**package** com.dev.mapping.entity;

**public** **class** Account {

**private** **int** accountNo;

**private** String type;

**public** **int** getAccountNo() {

**return** accountNo;

}

**public** **void** setAccountNo(**int** accountNo) {

**this**.accountNo = accountNo;

}

**public** String getType() {

**return** type;

}

**public** **void** setType(String type) {

**this**.type = type;

}

@Override

**public** String toString() {

**return** "Account [accountNo=" + accountNo + ", type=" + type + "]";

}

}

(ii)**Person.java**

**package** com.dev.mapping.entity;

**import** java.util.Set;

**public** **class** Person {

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

**private** String name;

**private** **int** age;

**private** Set<Account> accounts;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** Set<Account> getAccounts() {

**return** accounts;

}

**public** **void** setAccounts(Set<Account> accounts) {

**this**.accounts = accounts;

}

@Override

**public** String toString() {

**return** "id:"+id+" ,name:"+name+" ,age:"+age;

}

}

Now Mapping Files for above 2 domain classes:-

3.)**Account.hbm.xml**

<?xml version=*"1.0"*?>

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

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

<hibernate-mapping>

<class name=*"com.dev.mapping.entity.Account"* table=*"ACCOUNT"*>

<!-- identifier value -->

<id name=*"accountNo"* column=*"ACCOUNT\_NO"*>

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

</id>

<property name=*"type"* column=*"ACCOUNT\_TYPE"*/>

</class>

</hibernate-mapping>

4.) **Person.hbm.xml**

<?xml version=*"1.0"*?>

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

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

<hibernate-mapping>

<class name=*"com.dev.mapping.entity.Person"* table=*"PERSON"*>

<!-- identifier value -->

<id name=*"id"* column=*"ID"*>

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

</id>

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

<property name=*"age"* column=*"AGE"*/>

<!-- collection type property -->

<set name=*"accounts"* cascade=*"all,evict"* lazy=*"true"*>

<!-- Foreign Key column of child table(Account Table) -->

<key column=*"person\_id"*/>

<!-- One To Many Association -->

<one-to-many class=*"com.dev.mapping.entity.Account"*/>

</set>

</class>

</hibernate-mapping>

5.)

**OneToManyDao.java**

**package** com.dev.mapping.dao;

**import** com.dev.mapping.entity.Person;

/\*\*

\*

\* **@author** ashutosh

\* Interface model Programming I am using to support Strategy Design Pattern...

\*/

**public** **interface** OneToManyDao {

//insert

**public** **int** save();

//select

**public** **void** loadData();

//add new element in set

**public** **void** addNewAccountForExistingPerson();

//delete(orphan Record)

**public** **void** deleteOneAccountForAUser();

}

6.) **OneToManyDaoImpl.java**

package com.dev.mapping.dao.impl;

import java.util.HashSet;

import java.util.List;

import java.util.Scanner;

import java.util.Set;

import org.hibernate.HibernateException;

import org.hibernate.Query;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.dev.mapping.dao.OneToManyDao;

import com.dev.mapping.entity.Account;

import com.dev.mapping.entity.Person;

import com.dev.mapping.util.HibernateUtil;

/\*\*

\*

\* @author ashutosh

\* Persistence logic put here

\*/

public class OneToManyDaoImpl implements OneToManyDao {

@Override

public int save() {

Session session=null;

Transaction transaction=null;

int idvalue=0;

Scanner scanner=null;

//System.out.println("Session::"+session);

/\*\*

\* one more important thing non select persistence operation must be

\* done under Transaction to ensure data integrity so here we must enable

\* Hibernate Transaction otherwise without Transaction data will not persist.

\*/

scanner=new Scanner(System.in);

//create First Child class

Account account=new Account();

//account.setAccountNo(accountNo);//natural id use this if generator is not specified

System.out.println("Enter Account Type::");

account.setType(scanner.nextLine());

//create Second object

Account account2=new Account();

System.out.println("Enter Account Type::");

account2.setType(scanner.nextLine());

//Now create one Set holding above created 2 Account Objects

Set<Account> accountSet=new HashSet<>(); //type inference(jdk-1.7) with generics(jdk-1.5)

accountSet.add(account);

accountSet.add(account2);

//now Create Parent class object

Person person=new Person();

/\*\*

\* person.setId(id); //natural id don't set as we are using Hibernate increment

\* Generator here....

\*/

System.out.println("Enter Name");

person.setName(scanner.nextLine());

System.out.println("Enter Age::");

person.setAge(scanner.nextInt());

scanner.nextLine();

//now set account object to Person

person.setAccounts(accountSet);

session=HibernateUtil.getSession();

try {

//begin Transaction

transaction=session.beginTransaction();

//now do persistence operation with help of Session

idvalue=(Integer)session.save(person); //AutoUnboxing concept(jdk-1.5)

//commit transaction

transaction.commit();

System.out.println("Data Inserted sucessfully");

}

catch (HibernateException he) { //try with multiple catch (jdk-1.7)

//rollback Transaction

transaction.rollback();

he.printStackTrace();

System.out.println("Data Insertion Failed");

}

catch (Exception e) {

transaction.rollback();

e.printStackTrace();

System.out.println("Data Insertion Failed");

}

finally {

//close session as it is one per persistent operation(client)

session.close();

}

//returning hibernate generator generated value(Surrogate column value)

return idvalue;

}

@Override

public void loadData() {

Session session=null;

Query query=null;

List<Person> personList=null;

/\*\*

\* Here I want to select multiple records from database tables so it means I am

\* interested to do bulk operation so I will go for HQL(Hibernate Query Language).

\*

\*/

//get Session object

session=HibernateUtil.getSession();

//Create Query Object

query=session.createQuery("from Person");

//now for select operation using HQL

personList=query.list();

//Get All Parents Records using Stream API

for(Person person:personList) {

System.out.println("Person Details::"+person);

Set<Account> accountSet=person.getAccounts();

//accountSet.isEmpty(); //for lazy="extra"

/\*\*

\* query for lazy="extra"

\* select

person0\_.ID as ID1\_1\_,

person0\_.NAME as NAME2\_1\_,

person0\_.AGE as AGE3\_1\_

from

PERSON person0\_

Person Details::id:1 ,name:Ashu ,age:23

Hibernate:

select

count(ACCOUNT\_NO)

from

ACCOUNT

where

person\_id =?

\*/

/\*for(Account account:accountSet) {

System.out.println(account);

}\*/

for(Account account:accountSet) {

System.out.println(account);

}

}

}

@Override

public void addNewAccountForExistingPerson() {

Session session=null;

Person person=null;

Set<Account> accountSet=null;

Transaction transaction=null;

//get Session

session=HibernateUtil.getSession();

//Load Parent Object(Person)

person=(Person) session.get(Person.class,1);

//get all its Associated child

accountSet=person.getAccounts();

//create New Account which you want to add in this Collection

Account account=new Account();

//session.clear();

//account.setAccountNo(1);

account.setAccountNo(5);

account.setType("Just");

//now add in the set

/\*\*

\* if u are adding element in collection under Transaction it will reflect

\* into db also.

\*/

try {

//begin Transaction

transaction=session.beginTransaction();

//session.evict(account);

accountSet.add(account);

//commit Transaction

// session.clear();

transaction.commit();

}

catch (Exception e) {

e.printStackTrace();

//rollback tx

transaction.rollback();

}

finally {

//close Session

HibernateUtil.closeSession(session);

}

}

@Override

public void deleteOneAccountForAUser() {

Transaction tx=null;

//Get Session

Session session=HibernateUtil.getSession();

//Load parent object

Person person=(Person) session.get(Person.class, 1);

//get all child objects associated with this parent object

Set<Account> accountSet=person.getAccounts();

/\*\*now load that child object which you want to delete and it will come from First Level

cache as cascade="all" so in above line when I loaded parent then all its associated

child also loaded into First Level cache...

\*/

Account account=(Account) session.get(Account.class,2);

/\*\*now I have to delete this above loaded child object from database

as we know if we do any changes into session(persistent state)(Collection) then

it will directly reflected into database table...\*/

/\*\*

\* but as delete(non select operation) so to maintain data consistency it must be done

\* into Transaction right so let me enable the Transaction....

\*/

try {

//begin the transaction

tx=session.beginTransaction();

//persistent operation

accountSet.remove(account); //now record-2 will be orphan record as it does not associated with any parent table key column value

tx.commit();

System.out.println("Loaded Child Object deleted successfully!!");

}

catch (Exception e) {

tx.rollback();

System.out.println("Loaded Child Object deletion failed!!");

}

finally {

//close session(as session is per client(one persistent operation)

HibernateUtil.closeSession(session);

}

}

}

7.)**Hibernate Util Class:-**

**package com.dev.mapping.util;**

**import org.hibernate.Session;**

**import org.hibernate.SessionFactory;**

**import org.hibernate.cfg.Configuration;**

**public class HibernateUtil {**

**private static SessionFactory factory;**

**private static Session session;**

**static {**

**//create Hibernate Configuration Object**

**Configuration configuration=new Configuration();**

**configuration=configuration.configure("com/dev/mapping/conf/hibernate.cfgs.xml");**

**/\*\***

**\* Create Session Factory object sothat Configuration data will dump to**

**\* SessionFactory object.......this object is one per jvm or one per application**

**\* or more specific one per classloader as we know in one jvm one class can be loaded**

**\* multiple times also as per classloader hierarchy....**

**\*/**

**factory=configuration.buildSessionFactory();**

**System.out.println("Factory::"+factory);**

**}**

**/\*\***

**\***

**\* Session object is one per client**

**\*/**

**public static Session getSession() {**

**if(factory!=null) {**

**System.out.println("Factory::"+factory);**

**session=factory.openSession();**

**}**

**return session;**

**}**

**public static void closeSession(Session session) {**

**if(session!=null)**

**session.close();**

**}**

**public static void closeSessionFactory() {**

**if(factory!=null)**

**factory.close();**

**}**

**}**

**8.)finally test/client:-**

**package com.dev.mapping.test;**

**import com.dev.mapping.dao.OneToManyDao;**

**import com.dev.mapping.dao.impl.OneToManyDaoImpl;**

**import com.dev.mapping.util.HibernateUtil;**

**public class OneToManyTest {**

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

**//invoke Dao**

**OneToManyDao dao=null;**

**dao=new OneToManyDaoImpl();**

**//insert**

**//dao.save(); //TransientObjectException**

**//select**

**//dao.loadData();**

**//update**

**//dao.addNewAccountForExistingPerson(); //StaleStateException,NonUniqueObjectException**

**//LazyInitializationException**

**//delete**

**dao.deleteOneAccountForAUser();**

**//close Session Factory**

**HibernateUtil.closeSessionFactory();**

**}**

**}**

Above Program Questions:-

1. **Cascade attribute and all its possible values**
2. **Lazy attribute and all its possible values**
3. **Hibernate Generator**
4. **Famous exception:- StaleStateException,NonUniqueObjectException,TransientObjectException**
5. **How to mention foreign key column in one to many association mapping**
6. **What is lazy and eager laoding in association mapping.**
7. **What is orphan record.**
8. **How many states are there for a domain class?**
9. **Session object and its scope?**
10. **First level cache?**
11. **SessionFactory usage?**
12. **GenericMapper**
13. **ASTQueryTranslator**
14. **Hbm2ddl.auto and possible values**