**Cascade Attribute in Hibernate**

Main concept of hibernate relations is to getting the relation between **parent** and **child** class objects **Cascade attribute is mandatory**, whenever we apply relationship between objects, cascade attribute transfers operations done on one object onto its related child objects

***Example***: if we apply insert (or update or delete) operation on parent class object, then child class objects will not be affected, if cascade = “none”

***Cascade having the values…….***

***ALL***

***DETACH***

***MERGE***

***PERSIST***

***REFERESH***

***REMOVE***

***We will see in more details about cascade in Hibernate Relation Concepts***

*Relations in Hibernate START HERE*

**One to One Relations..........**

**1)** One object is associated with one object only

**2)** In this relationship, one object of the one pojo class contains association with one object of the another pojo class

**3)** To apply one to one relationship between two pojo class objects it is possible by without taking a separate foreign key column in the child table of the database

**4)** To apply one to one relationship, we copy the primary key value of parent object into primary key value of the child object. So that the relationship between two objects is one to one

**5)** If we want to copy parent object primary key value into child object primary key, we need to use a special generator class given by hibernate called foreign.

**6)** Actually, this foreign generator is only used in one-to-one relationship only

Laptop.java **int** lid; String lName; Student s;

Student.java **int** sid; String sName; Laptop l;

Laptop.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=*"p2.Laptop"* table=*"nine"*>

<id name=*"lid"*>

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

</id>

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

<one-to-one name=*"s"* class=*"p2.Student"* cascade=*"all"*></one-to-one>

</class>

<class name=*"p2.Student"* table=*"ten"*>

<id name=*"sid"*>

<generator class=*"foreign"*>

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

</generator>

</id>

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

<one-to-one name=*"l"* class=*"p2.Laptop"* constrained=*"true"*></one-to-one>

</class>

</hibernate-mapping>

Client.java

Student st = **new** Student();

Laptop lap = **new** Laptop();

st.setsName("SAM");

lap.setlName("DELL");

st.setL(lap);

lap.setS(st);

**s.save(st);**

**Update in One to One Relations..........**

FetchData.java

Session s = sf.openSession();

Laptop lap = s.load(Laptop.**class**,1);

lap.setlName("HP");

lap.getS().setsName("PAM");

s.save(lap);

s.beginTransaction().commit();

**Many to One Relations..........**

In the many to one relationship, the relationship is applied from child object to parent object, but in one-to-many parent object to child object right...! just remember

Vendor.java **int** vid; String vName;

Customer.java **int** cid; String cName; Vendor v;

Vendor.hbm.xml

<class name=*"p1.Vendor"* table=*"nine"*>

<id name=*"vid"*></id>

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

</class>

<class name=*"p1.Customer"* table=*"ten"*>

<id name=*"cid"*></id>

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

<many-to-one name=*"v"* column=*"Dummy"* class=*"p1.Vendor"* cascade=*"all"*></manyto-one>

</class>

Client.java

Session session = factory.openSession();

Vendor v =**new** Vendor();

v.setVid(101);

v.setVname("java4s");

Customer c1=**new** Customer();

c1.setCid(504);

c1.setCname("customer4");

c1.setV(v);

Customer c2=**new** Customer();

c2.setCid(505);

c2.setCname("customer5");

c2.setV(v);

Customer c3=**new** Customer();

c3.setCid(506);

c3.setCname("customer6");

c3.setV(v);

Transaction tx = session.beginTransaction();

session.save(c1);

session.save(c2);

session.save(c3);

tx.commit();

session.close();

System.***out***.println("One To Many is Done..!!");

factory.close();

**Many to Many Relations..........**

It will show relations from both side means one question has many answers and one answer can have many questions. **It will create three tables** one for Question One for Answer and one for where it will show the link of Question and Answer

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

Answer.java **int** id; String answer; String postedby; List<Question> q;

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"* table = *"one\_two"* fetch = *"select"*>**

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

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

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

**</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

Answer ans1=**new** Answer();

ans1.setAnswer("Java is a programming language");

ans1.setPostedby("Ravi Malik");

Answer ans2=**new** Answer();

ans2.setAnswer("Platform Independent");

ans2.setPostedby("Sudhir Kumar");

Question q1=**new** Question();

q1.setQname("What is Java?");

ArrayList<Answer> l1=**new** ArrayList<Answer>();

l1.add(ans1);

l1.add(ans2);

q1.setAns(l1);

Answer ans3=**new** Answer();

ans3.setAnswer("Servlet is an Interface");

ans3.setPostedby("Jai Kumar");

Question q2=**new** Question();

q2.setQname("What is Servlet?");

ArrayList<Answer> l2=**new** ArrayList<Answer>();

l2.add(ans3);

l2.add(ans2);

q2.setAns(l2);

s.persist(q1);

s.persist(q2);

t.commit();

**Composite Primary Keys in Hibernate**

Composite primary keys mean having more than one primary key,

If the table has a primary key, then in the hibernate mapping file, we need to configure that column by using <id /> element.

Even though the database table doesn’t have any primary key, we must configure one column as id (one primary key is must)

If the database table has more than one column as primary key then we call it as composite primary key, so if the table has multiple primary key columns, in order to configure these primary key columns in the hibernate mapping file we need to use one new element called

**<composite-id** ….> **</composite-id>**

if we want to use the composite primary keys, we must implement our pojo class

with **Serializable** interface

**It will enter data like**

**1 – 1**

**2 – 1**

**3 – 2**

**4 – 2**

**3 – 2 we cannot enter this bcuz these combinations already done above**

Product.java **int** productId; **int** proVersion; **int** price;

product.hbm.xml

<class name = *"p1.Product"* table = *"pro"*>

<composite-id>

<key-property name = *"productId"* column = *"pid"*></key-property>

<key-property name = *"proVersion"* column = *"proVersion"*></key-property>

</composite-id>

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

</class>

**List of generators**

The following are the **list of main generators** we are using in the hibernate framework

assigned

increment

sequence

identity

hilo

native

foregin

uuid.hex

uuid.string

In the above generators list, the first 7 are used for **int,long,short types of primary keys**, and last **2** are used when the primary key column type is **String type (varchar2)**

## sequence

* Nor has the support with **MySQL**
* This generator class is database dependent it means, we cannot use this generator class for all the database, we should know whether the database supports sequence or not before we are working with it
* while inserting a new record in a database, hibernate gets next value from the sequence under assigns that value for the new record
* If programmer has created a sequence in the database, then that sequence name should be passed as the generator

## identity

* This is database dependent, actually it’s not working in oracle
* In this case (identity generator) the id value is generated by the database, but not by the hibernate, but in case of increment hibernate will take over this
* this identity generator doesn’t need any parameters to pass
* this identity generator is similar to increment generator, but the difference was increment generator is database independent and hibernate uses a select operation for selecting max of id before inserting new record
* But in case of identity, no select operation will be generated in order to insert an id value for new record by the hibernate

## native

when we use this generator class, it first checks whether the database supports identity or not, if not checks for sequence and if not, then hilo will be used finally the order will be.

* identity
* sequence
* hilo

For example, if we are connecting with oracle, if we use generator class as native then it is equal to the generator class sequence.

## foreign

we will see about this generator in one-to-one relationship, else you may not understand.

## Hilo

When we configure Hilo generator class in hbm.xml file, along with the generator name we can pass 3 parameters:

1. **table**
2. **column**
3. **max\_lo**

<id name=*"qid"*>

<generator class=*"hilo"*>

<param name = *"table"*>one</param>

<param name = *"column"*>hlo</param>

<param name = *"max\_lo"*>4</param> //next Increment Value

</generator>

</id>

Here we took table name as **mytable** , column name as **col1** and max\_lo value is **10.**

* When first time hilo generator executed then id is ‘1’.
* Second time onwards hilo uses formula for generating the Id. So by applying the above formula the Id is “11”

**max\_lo \* next\_hi + next\_hi**

here,  max\_lo = 10 (in hbm.xml we configured)

next\_hi = 1 (already existed value in the column col1)

then **10 \* 1 + 1 = 11**and in the col1 it updates the value as “2” (Phase 2)

* For third time hilo generates id as “22”

**10 \* 2 + 2 = 22**and in the col1 it updates the value as “3”  (Phase 3)

This is how Hilo generator works. And Hilo generator is database independent generator, It will work on all databases.

**Support for 'Hilo' generator has been removed Hibernate 5**

**Call Stored Procedure**

**Fetch record from DB and Bind that all record (Columns) in DTO, Procedure Must Return record**

**Like Select \* from product;**

Session s = sf.openSession();

NativeQuery<Product> n = s.createSQLQuery("{call selectPrice(:id)}");

n.addEntity(Product.**class**);

n.setParameter("id", 1);

List<Product> li = n.list();

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

**It’s a NativeQuery means we are calling procedure the same way we use to call in SQL**

**Fetch records from DB in Hibernate 5(JPA) without writing SQL Code**

**First Create Procedure in SQL........**

**mysql> create procedure showCount()**

**-> begin**

**-> select \* from pro;**

**-> end;**

**-> //**

StoredProcedureQuery sp = s.createStoredProcedureQuery("showCount");

sp.execute();

List<Object[]> li = sp.getResultList();

li.forEach(i -> System.***out***.println(Arrays.*toString*(i)));

**ByDefault it will Return List<Object[]> if we want to bind all result into DTO**

StoredProcedureQuery sp = s.createStoredProcedureQuery("showCount",Product.**class**);

sp.execute();

List<Product> li = sp.getResultList();

li.forEach(i -> System.***out***.println(Arrays.*toString*(i)));

**Fetch records from DB in Hibernate 5(JPA) with IN and OUT Parameter**

**First Create Procedure in SQL........(Drop old Procedure showCount....)**

**Drop procedure showCount// then create procedure again with same Name.**

**mysql> create procedure showCount(IN id int, OUT pco int)**

**-> begin**

**-> select COUNT(\*) into pco from pro where productId = id;**

**-> end;**

**-> //**

**Call in MySQL.....**

**mysql> call showCount(1,@pco)//**

**mysql> select @pco//**

**Now we can call it through Hibernate**

StoredProcedureQuery sp = s.createStoredProcedureQuery("showCount");

sp.registerStoredProcedureParameter("id", Long.**class**, ParameterMode.***IN***);

sp.registerStoredProcedureParameter("pco", Long.**class**, ParameterMode.***OUT***);

sp.setParameter("id", 1L);

sp.execute();

Long data = (Long)sp.getOutputParameterValue("pco");

System.***out***.println("Data " + data);

We can use something called ***REF\_CURSOR*** above code can be

StoredProcedureQuery sp = s.createStoredProcedureQuery("showCont");

sp.registerStoredProcedureParameter(1, Product.**class**, ParameterMode.***REF\_CURSOR***);

sp.execute();

List<Product> li = sp.getResultList();

li.forEach(i -> System.***out***.println(i));

**Means we are output value into Particular variable of MySql and thats called REF\_CURSOR**

**Hibernate throws the following exception:**

org.hibernate.QueryException: java.lang.IllegalArgumentException:

org.hibernate.QueryException: Dialect [org.hibernate.dialect.MySQL57InnoDBDialect] not known to support REF\_CURSOR parameters

**Even though this stored procedure is**[**working properly on Oracle**](https://vladmihalcea.com/how-to-call-oracle-stored-procedures-and-functions-from-hibernate/)**and**[**PostgreSQL**](https://vladmihalcea.com/how-to-call-postgresql-functions-from-hibernate/)**, on MySQL, it does not work because the MySQL driver does not support REFCURSOR outside of stored procedure.**