2. **Hibernate Architecture**

Hibernate architecture ထဲမွာ persistent object, session factory, transaction factory, connection factory, session, transaction etc လိုမ်ိဳး object ေပါင္းမ်ားစြာ ပါ၀င္ပါတယ္။

Hibernate ကို layer ၄ ခု ခြဲထားပါတယ္။

1. Java application layer
2. Hibernate framework layer
3. Backhand api layer
4. Database layer



ဒီပံုကေတာ့ hibernate ကို mapping file ရယ္ configuration file ေတြနဲ႔ ေတြ႕ၿမင္ရတဲ့ high level architecture ပံုပဲ ၿဖစ္ပါတယ္။



hibernate framework က session factory, session, transaction etc တို႔လို object မ်ားစြာကို java ရဲ႕ ရွိၿပီးသား api မ်ားၿဖစ္ၾကတဲ့ DBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface)တို႔နဲ႔ပါင္းစပ္အသံုးၿပဳတယ္။

Hibernate Architecture ရဲ႕ elements မ်ား

SessionFactory

session တစ္ခုရဲ႕ factory တစ္ခုၿဖစ္သလို ConnectionProvider ရဲ႕ client လည္းၿဖစ္တယ္။ သူက data ရဲ႕ second level cache (optional) ကို hold လုပ္ထားတယ္။ org.hibernate.SessionFactory interface က session ရဲ႕ object ကို ရရွိဖို႔ရန္ factory method ကို ထုတ္ေပးပါတယ္။

Session

session object က application နဲ႔ database ထဲမွာ သိမ္းထားတဲ့ data နဲ႔ၾကားထဲမွာ interface တစ္ခုကို ထုတ္ေပးပါတယ္။ သူက short-lived object တစ္ခုၿဖစ္ၿပီးေတာ့ JDBC object ကို wrap လုပ္ေပးပါတယ္။ session က Transaction, Query နဲ႔ Criteria တို႔ရဲ႕ factory လည္း ဟုတ္တယ္။ session က data ရဲ႕ first-level cache (mandatory) ကို hold လုပ္ထားတယ္။ org.hibernate.Session interface က object ေတြ insert, update နဲ႔ delete လုပ္ဖို႔ရန္ method မ်ားကို ထုတ္ေပးပါတယ္။ ၿပီးေတာ့ Transaction, Query နဲ႔ Criteria အတြက္လည္း factory method မ်ားကို ထုတ္ေပးပါတယ္။

Transaction

transaction object က work ရဲ႕ atomic unit ကို သတ္မွတ္ေပးတယ္။ org.hibernate.Transaction interface က transaction management အတြက္ methods မ်ားကို ထုတ္ေပးပါတယ္။

ConnectionProvider

JDBC connections ရဲ႕ factory တစ္ခုၿဖစ္တယ္။ connectionProvider က DriverManager သို႔မဟုတ္ DataSource မွ application ကို abstract လုပ္တယ္။ သူကေတာ့ optional ၿဖစ္တယ္။

TransactionFactory

Transaction ရဲ႕ factory တစ္ခုၿဖစ္တယ္။ သူကလည္း optional ၿဖစ္တယ္။

**Hibernate with Eclipse**

1. HB Using XML (main method နဲ႔ run ရမယ္)

2. HB Using Annotation (main method နဲ႔ run ရမယ္)

**1. HB Using XML**

Hibernate Example using XML in Eclipse

ေအာက္ပါအဆင့္မ်ားကို လုပ္ေဆာင္ရမွာ ၿဖစ္ပါတယ္။

1. persistence class ကို create လုပ္မယ္။
2. persistence class အတြက္ mapping file ကို create လုပ္မယ္။
3. configuration file ကို create လုပ္မယ္။
4. table ေဆာက္မယ္။
5. persistent object ကို retrieves သို႔မဟုတ္ stores လုပ္မယ့္ class တစ္ခုကို create လုပ္မယ္။
6. jar file ကို load လုပ္မယ္။
7. main method ပါတဲ့ class ကို run မယ္။

1. Create the Persistent class

simple Persistent class တစ္ခုကို create လုပ္ေတာ့မယ္ဆိုရင္ ေအာက္ပါ rules မ်ားကို လိုက္နာေဆာင္ရြက္သင့္ပါတယ္။

* **A no-arg constructor**: default constructor တစ္ခုကေတာ့ ပါကိုပါရမယ္ ဒါမွသာ hibernate က Persistent class ရဲ႕ instance ကို newInstance() method နဲ႔ ေဆာက္လို႔ရမွာ ၿဖစ္ပါတယ္။
* **identifier property တစ္ခုေပးပါ** : အဲ့တာက id လို attribute မ်ိဳးကို assign လုပ္လို႔ ပိုမိုေကာင္းမြန္ေစပါတယ္။ id ဆိုတဲ့ field ကေတာ့ db ထဲမွာ primary key ပံုစံနဲ႔ ရွိေနမွာ ၿဖစ္ပါတယ္။
* **getter and setter methods မ်ားထည့္သြင္းပါ** : Hibernate က default အားၿဖင့္ getter setter name မ်ားကို အသိအမွတ္ၿပဳထားေပးၿပီးသား။
* **non-final class ကို ဦးစားေပးပါ** : hibernate က proxies ရဲ႕ concept ကို အသံုးၿပဳတယ္ ဆိုလိုသည္မွာ အဲ့တာက persistence class အေပၚမွာ depends ၿဖစ္ေနပါတယ္။ application programmer ေတြကေတာ့ lazy association fetching အတြက္ proxies မ်ားကို သံုးဖို႔ မၿဖစ္ႏိုင္ပါဘူး။

Employee.java

**public** **class** Employee {

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

**private** String firstName, lastName;

**getter//setter**

2. Persistent class အတြက္ mapping file ကို create လုပ္ၿခင္း

mapping file ရဲ႕ နာမည္ကို အစဥ္အလာအတိုင္းပဲ ေပးသင့္ပါတယ္။ **class\_name.hbm.xml** အဲ့လို format အတိုင္းပဲ ၿဖစ္သင့္တယ္။

* **hibernate-mapping** : mapping file ရဲ႕ root element တစ္ခုၿဖစ္တယ္။ သူ႕ထဲမွာ mapping file အားလံုးပါ၀င္ပါတယ္။
* **class** : hibernate-mapping element ရဲ႕ sub element ၿဖစ္တယ္။ သူက Persistent class ကို သတ္မွတ္ေပးပါတယ္။
* **generator** : id ရဲ႕ sub element တစ္ခုၿဖစ္တယ္။ သူ႕ကို primary key generate လုပ္ဖို႔ရန္ အသံုးၿပဳတယ္။ သူထဲမွာ assigned, increment, hilo, sequence, native etc လိုမ်ိဳး generator class မ်ားစြာ ပါ၀င္ႏိုင္ပါတယ္။
* **property** : class ရဲ႕ sub element တစ္ခုၿဖစ္ပါတယ္။ သူက persistence class ရဲ႕ property name ကို သတ္မွတ္ေပးပါတယ္။

သူ႕ကို eclipse မွာ ယူမယ္ဆိုရင္ xml file type ကို မယူပါ။ file ဆိုတဲ့ type ကို ယူၿပီးမွာ name ေပးတဲ့အခါမွ format ၿဖစ္တဲ့ xml ကို ထည့္ရိုက္ေပးလိုက္တာ ၿဖစ္ပါတယ္။

ၿပီးေတာ့ သူ႕ကို local package ၿဖစ္တဲ့ hb ရဲ႕ အၿပင္ဘက္မွာ ေဆာက္ေပးပါ။ ရွင္းရွင္းေၿပာေၿပာရရင္ java file ေတြက local package အတြင္းထဲမွာ ၿဖစ္ၿပီးေတာ့ xml format မ်ားကေတာ့ local package ရဲ႕ အၿပင္ဘက္မွာ ၿဖစ္ရမွာ ၿဖစ္ပါတယ္။

employee.hbm.xml

<hibernate-mapping>

<class name=*"hibernate\_with\_eclipse.hb\_with\_xml.hb.Employee"* table=*"emp1000"*>

<id name=*"id"*>

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

</id>

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

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

</class>

</hibernate-mapping>

3. Configuration file ကို create လုပ္ၿခင္း

mapping file နဲ႔ database နဲ႔ပတ္သက္တဲ့ information မ်ားပါ၀င္ပါတယ္။ ဥပမာ connection\_url, driver\_class, username, password etc. တို႔ၿဖစ္ၾကပါတယ္။database ထဲမွာ table ကို automatically create လုပ္ဖို႔ရန္ hbm2ddl.auto property ကို အသံုးၿပဳပါတယ္။ Dialect class ရဲ႕ detail အေၾကာင္းကိုေတာ့ next topic မွာ ေၿပာပါမယ္။ သူ႕နာမည္ကေတာ့ hibernate.cfg.xml ၿဖစ္ရပါမယ္။သူ႕ကိုေတာ့ soucrce folder တစ္ခုေဆာက္ၿပီး ထည့္လိုက္ပါတယ္။ xml နဲ႔ေရးထားတဲ့အတြက္ေၾကာင့္ mapping resource ၿဖစ္ေနတာပါ။

hibernate.cfg.xml

<hibernate-configuration>

<session-factory>

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

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

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

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

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

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

<mapping resource=*"hibernate\_with\_eclipse/hb\_with\_xml/employee.hbm.xml"*/>

</session-factory>

</hibernate-configuration>

5. db မွာ သက္ဆိုင္ရာ table name အတိုင္း ေဆာက္မယ္။

CREATE TABLE `hibernate`.`emp1000` (

`id` INT NOT NULL,

`firstName` VARCHAR(45) NULL,

`secondName` VARCHAR(45) NULL,

PRIMARY KEY (`id`));

6. class တစ္ခုေဆာက္ၿပီး အဲ့ class ထဲကို objects မ်ား retrieves သို႔မဟုတ္ stores လုပ္ၿခင္း

StoreData.java (main method ပါ၀င္ပါတယ္)

**public** **class** StoreData {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Employee e1 = **new** Employee();

e1.setId(1);

e1.setFirstName("Gaurav");

e1.setLastName("Chawla");

session.save(e1);

t.commit();

System.***out***.println("successfully saved");

factory.close();

session.close();}}

7. hibernate အတြက္ လိုအပ္ေသာ jar file မ်ားကို download ဆြဲၿခင္း နဲ႔ eclipse ထဲထည့္သြင္းၿခင္း

လိုအပ္ေသာ jar file မ်ားကို download ဆြဲၿပီး resource folder ေအာက္မွာ jar ဆိုတဲ့ folder တစ္ခုေဆာက္ၿပီး ထည့္ေပးလိုက္တယ္။ ၿပီးရင္ build path မွာ add external jar ကို click ႏွိပ္လိုက္ရင္ folder dialog တက္လာတယ္ အဲ့တာဆိုရင္ ခုနက ကိုယ္ resource ေအာက္မွာထည့္ထားတဲ့ folder path လမ္းေၾကာင္းေအာက္က jar file ေတြကုို ေရြးေပးလိုက္ရံုပဲ။

**2. HB using Annotation**

hibernate application ကို xml နဲ႔ create လုပ္လို႔ရလို annotation နဲ႔လည္း create လုပ္လို႔ရပါတယ္။ ဥပမာ @Entity, @Id, @Table etc တို႔ ၿဖစ္ၾကပါတယ္။ Hibernate Annotations က JPA 2 specification ကို အေၿခခံထားတာၿဖစ္ၿပီး features ေတြအားလံုးကိုလည္း

support လုပ္ေပးထားပါတယ္။ JAP annotations အားလံုးကို javax.persistence package ထဲမွာ သတ္မွတ္ထားတာ ၿဖစ္ပါတယ္။ Hibernate EntityManager က JPA specification က သတ္မွတ္ထားတဲ့ interfaces နဲ႔ life cycle ကို implements လုပ္ထားပါတယ္။ annotation နဲ႔ hibernate ကို develop လုပ္ၿခင္းက ဘာေကာင္းလဲဆိုေတာ့ mapping (hbm) file ကို create လုပ္စရာမလိုေတာ့ပါ။ hibernate annotations က meta data ကို ေပးႏိုင္ရန္ လုပ္ေဆာင္ပါတယ္။

1. Create persistence class

@Entity annotation - ဒီ class က entity ၿဖစ္တဲ့ဆိုတာကို သိသာေစဖို႔အတြက္ ၿဖစ္တယ္။

@Table annotation – entity ထဲက data ေတြကို ဘယ္ table ထဲကို persist လုပ္ရမလဲဆိုတာက ေၿပာၿခင္းၿဖစ္ပါတယ္။ @Table annotation ဆိုတဲ့ annotation ကို အသံုးမၿပဳခဲ့ဘူးဆိုရင္ေတာ့ class name နဲ႔တူတဲ့ table name ကို default အားၿဖင့္ သြားၿပီး persist လုပ္ေပးလိမ့္မယ္။

Student.java

@Entity

@Table(name = "emp500")

**public** **class** Student {

@Id

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

**private** String firstName, lastName;

**getter//setter**

2. Create the Configuration file

annotation နဲ႔ေရးထားတဲ့အတြက္ေၾကာင့္ mapping class ၿဖစ္ေနတာပါ။

hibernate.cfg.xml

<hibernate-configuration>

<session-factory>

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

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

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

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

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

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

<mapping class=*"hibernate\_with\_eclipse.hb\_with\_annotation.Student"*/>

<mapping resource=*"hibernate\_with\_eclipse/hb\_with\_xml/employee.hbm.xml"*/>

</session-factory>

</hibernate-configuration>

3. db မွာ သက္ဆိုင္ရာ table name အတိုင္း ေဆာက္မယ္။

CREATE TABLE `hibernate`.`emp500` (

`id` INT NOT NULL,

`firstName` VARCHAR(45) NULL,

`lastName` VARCHAR(45) NULL,

PRIMARY KEY (`id`));

4. Create the class that retrieves or stores the persistent object.

StoredDataAnnotation.java

**public** **class** StoredDataAnnotation {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Student e1 = **new** Student();

e1.setId(101);

e1.setFirstName("Gaurav");

e1.setLastName("Chawla");

session.save(e1);

t.commit();

System.***out***.println("successfully saved");

factory.close();

session.close();}}

Hibernate Example

1. HB Web Application

2. HB Generator Class

3. HB Dialects

1. **HB Web Application**

Web Application with Hibernate (using XML)

presentation logic အတြက္ jsp ကို သံုးမယ္။ representing data အတြက္ Bean class ကို သံုးမယ္။ database codes အတြက္ DAO class ကုိ သံုးမယ္။ အခု example မွာဆိုရင္ JSP file ကိုသံုးၿပီး user ဆီက data ကို ရယူပါမယ္။

index.jsp

user ဆီက data ယူမယ္ ၿပီးရင္ register.jsp ဆီကို post method နဲ႔ သြားမယ္။

<form action=*"pages/hibernate\_example/hb\_web\_application/register.jsp"* method=*"post"*>

Name:<input type=*"text"* name=*"name"*/><br><br/>

Password:<input type=*"password"* name=*"password"*/><br><br/>

Email ID:<input type=*"text"* name=*"email"*/><br><br/>

<input type=*"submit"* value=*"register"*/>

</form>

register.jsp

register.jsp file ကို user class ကေနလာတဲ့ object ထဲက information ေတြကို သိမ္းထားဖို႔နဲ႔ request parameters အကုန္လံုးကို ရယူဖို႔ရအတြက္ ေရးသားထားၿခင္းၿဖစ္ပါတယ္။ သူက user entity class object ကို userDao ရဲ႕ register method ေခၚၿပီး pass လုပ္ထားပါတယ္။ register.jsp က process ကို အဓိကခ်ိတ္ဆက္ေပးတဲ့ file တစ္ခုၿဖစ္ပါတယ္။

<%@page import=*"hibernate\_example.hb\_web\_application.UserDao"*%>

<jsp:useBean id=*"obj"* class=*"hibernate\_example.hb\_web\_application.User"*>

</jsp:useBean>

<jsp:setProperty property=*"\*"* name=*"obj"* />

<%

**int** i = UserDao.register(obj);

**if** (i > 0)

out.print("You are successfully registered");

%>

User.java

**public** **class** User {

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

**private** String name, password, email;

**public** User() {}

//getter and setter

user.hbm.xml

database ထဲက table နဲ႔ user entity နဲ႔ map လုပ္ေပးတဲ့ file ၿဖစ္ပါတယ္။

<hibernate-mapping>

<class name=*"hibernate\_example.hb\_web\_application.User"* table=*"u400"*>

<id name=*"id"*>

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

</id>

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

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

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

</class>

</hibernate-mapping>

UserDao.java

parameter ပါလာတဲ့ user object ကို သိမ္းထားေပးၿပီး register page ကို return ၿပန္ေပးလိုက္တယ္။

**public** **class** UserDao {

**public** **static** **int** register(User u) {

**int** i = 0;

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

i = (Integer) session.save(u);

t.commit();

session.close();

**return** i;}}

hibernate.cfg.xml

database information နဲ႔ mapping file information ကို သိမ္းဆည္းေပးပါတယ္။

<mapping resource=*"hibernate\_example/hb\_web\_application/user.hbm.xml"*/>

web.xml

<welcome-file>pages/hibernate\_example/hb\_web\_application/index.jsp</welcome-file>

2. HB Generator Class

<generator> class က id ရဲ႕ sub element တစ္ခုၿဖစ္တယ္။ persistent class ရဲ႕ object ေတြအတြက္ unique identifie ကို ထုတ္ေပးဖို႔ရန္ အသံုးၿပဳပါတယ္။ Hibernate Framework ထဲမွာ generator class မ်ားစြာ သတ္မွတ္ေပးထားပါတယ္။

generator class အားလံုးက org.hibernate.id.IdentifierGenerator interface ကို implements လုပ္ထားပါတယ္။ application programme ေတြက IdentifierGenerator interface ကို implementing လုပ္ၿခင္းအားၿဖင့္ one's own generator classes ေတြကို create လုပ္တယ္။ Hibernate framework ေတြက built-in generator classes ေၿမာက္မ်ားစြာကို ထုတ္ထားေပးၿပီးသား ၿဖစ္ပါတယ္။

1. assigned
2. increment
3. sequence
4. hilo
5. native
6. identity
7. seqhilo
8. uuid
9. guid
10. select
11. foreign
12. sequence-identity

1) assigned

<generator> element မရွိခဲ့ဘူးဆိုရင္ ဒီေကာင္က default အေနနဲ႔၀င္ေနမွာ ၿဖစ္ပါတယ္။

....

 <hibernate-mapping>

  <**class** ...>

    <id ...>

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

    </id>

    .....

  </**class**>

 </hibernate-mapping>

2) increment

table တစ္လံုးထဲကို data ထည့္လိုက္ရင္ id မ်ား into တိုးလာေအာင္ လုပ္ေဆာင္တဲ့ class ၿဖစ္ပါတယ္။ အဲ့ class က short, int နဲ႔ long မ်ိဳးစံုေသာ data type မ်ားကို ထုတ္ေပးႏိုင္ပါတယ္။ အကယ္လို႔မ်ား table မွာ identifier ပါခဲ့မယ္ဆိုရင္ေတာ့ application အေနနဲ႔ maximum value ကို စဥ္းစားေပးဖို႔ လိုလိမ့္မယ္ အဲ့လိုမွမဟုတ္ဘူးဆိုရင္ေတာ့ application အေနနဲ႔ first generated identifier ကို 1 ကေနစၿပီး ထုတ္ေပးပါလိမ့္မယ္။

....

 <hibernate-mapping>

  <**class** ...>

    <id ...>

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

    </id>

    .....

  </**class**>

 </hibernate-mapping>

3) Sequence

database ရဲ႕ sequence ကို အသံုးၿပဳပါတယ္။ အကယ္လို႔ sequence ကို မသတ္မွတ္ထားရေသးဘူးဆိုရင္ auto သတ္မွတ္ေပးပါလိမ့္မယ္။ ဥပမာ oracle database မွာဆိုရင္ HIBERNATE\_SEQUENCE ဆိုတဲ့ sequence ကို auto သတ္မွတ္ေပးထားတယ္။ Oracle, DB2, SAP DB, Postgre SQL or McKoi တို႔မွာ sequence ကို သံုးတယ္ သို႔ေပမယ့္ interbase ထဲမွာရွိတဲ့ generator ကို အသံုးၿပဳပါတယ္။

.....

<id ...>

<generator class="sequence"></generator>

</id>

.....

ကိုယ္ပိုင္ sequence တစ္ခု။ generator ရဲ႕ param subelement ကို အသံုးၿပဳထားပါတယ္။

.....

<id ...>

<generator class="sequence">

<param name="sequence">your\_sequence\_name</param>

</generator>

</id>

.....

4) Hilo

short, int နဲ႔ long အစရွိတဲ့ type ရဲ႕ id မ်ားကို ထုတ္ရန္ high နဲ႔ low algorithm မ်ားကို အသံုးၿပဳပါတယ္။

.....

 <id ...>

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

 </id>

 .....

5) native

database vendor ေပၚမူတည္ၿပီး identity, sequence သို႔မဟုတ္ hilo အစရွိသည္တို႔ကို အသံုးၿပဳပါတယ္။

.....

<id ...>

<generator class="native"></generator>

</id>

.....

6) identity

id column ကို support လုပ္ရန္ Sybase, My SQL, MS SQL Server, DB2 နဲ႔ HypersonicSQL မ်ားထဲမွာ identity ကို အသံုးၿပဳပါတယ္။ returned id ကေတာ့ short, int or long တို႔ ၿဖစ္ၾကပါတယ္။ unique identifier ကို generate လုပ္ရန္ database ရဲ႕ responsibility ၿဖစ္ပါတယ္။

7) seqhilo

high and low algorithm ကို specified sequence name မွာ အသံုးၿပဳပါတယ္။ type short, int or long ရဲ႕ id ကို return ၿပန္ပါလိမ့္မယ္။

8) uuid

It uses 128-bit UUID algorithm to generate the id. The returned id is of type String, unique within a network (because IP is used). The UUID is represented in hexadecimal digits, 32 in length.

9) guid

It uses GUID generated by database of type string. It works on MS SQL Server and MySQL.

10) select

It uses the primary key returned by the database trigger.

11) foreign

It uses the id of another associated object, mostly used with <one-to-one> association.

12) sequence-identity

It uses a special sequence generation strategy. It is supported in Oracle 10g drivers only.

3. SQL Dialects in Hibernate

SQL statements ရဲ႕ appropriate type ကို hibernate က generate လုပ္ေပးပါတယ္။ hibernate မွာ dialect ဆိုတာက ဘယ္ DB ကို သံုးမလဲဆိုတာကို ဆံုးၿဖတ္ေပးတဲ့အတိုင္းပဲ ၿဖစ္ပါတယ္။ အဲ့တာေၾကာင့္ hibernate က ဘယ္ db ကို မဆိုခ်ိတ္ဆက္မယ္ဆိုရင္ dialects configuration ကို အရင္ခ်ဖို႔ လိုအပ္ပါလိမ့္မယ္။

Syntax of SQL Dialect

<property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>

List of SQL Dialects

|  |  |
| --- | --- |
| RDBMS | Dialect |
| Oracle (any version) | org.hibernate.dialect.OracleDialect |
| MySQL | org.hibernate.dialect.MySQLDialect |
| PostgreSQL | org.hibernate.dialect.PostgreSQLDialect |
| Microsoft SQL Server | org.hibernate.dialect.SQLServerDialect |

**Hibernate Log4j**

1. HB with Log4j 1
2. HB with Log4j 2

1. HB with Log4j 1

Hibernate Logging by Log4j using xml file

Logging ဆိုတာက file တစ္ခုတည္းမွာ log မ်ားကို permanently သြားရိုက္တဲ့ process ကို ေခၚပါတယ္။ hibernate မွာေတာ့ logging process အတြက္ Log4j နဲ႔ Logback frameworks တို႔ကို အသံုးၿပဳႏိုင္ပါတယ္။

log4j ကို နည္းလမ္း ၂ မ်ိဳးနဲ႔ အသံုးၿပဳလို႔ရပါတယ္။

1. xml file
2. properties.file

Levels of Logging

|  |  |
| --- | --- |
| Levels | Description |
| OFF | logging ကို ပိတ္ဖို႔ အသံုးၿပဳပါတယ္။ |
| WARNING | message level တစ္ခုၿဖစ္ၿပီးေတာ့ problem တစ္ခုကို ညႊန္ၿပပါလိမ့္မယ္ |
| SEVERE | message level တစ္ခုၿဖစ္ၿပီးေတာ့ failure တစ္ခုကို ညႊန္ၿပပါလိမ့္မယ္ |
| INFO | informational messages မ်ားအတြက္ အသံုးၿပဳတဲ့ level တစ္ခုၿဖစ္ပါတယ္ |
| CONFIG | static configuration messages မ်ားအတြက္ အသံုးၿပဳတဲ့ level တစ္ခုၿဖစ္ပါတယ္ |

Log4j xml file အသံုးၿပဳၿပီး Hibernate Logging ကိုလုပ္ေဆာင္ရန္ အဆင့္မ်ား

နည္းလမ္း ၂ မ်ိဳးရွိပါတယ္။

1. hibernate နဲ႔ log4j jar files မ်ားကို load ဆြဲတင္
2. src folder ထဲမွာ log4j.xml file ကို create လုပ္။အဲ့မွာ log ရိုက္ေစခ်င္တဲ့ log file ကို လမ္းေၾကာင္းနဲ႔နာမည္နဲ႔ တစ္ခါတည္းေပးထားခဲ့

Log4j xml file အသံုးၿပဳၿပီး Hibernate Logging ကိုလုပ္ေဆာင္တဲ့ example

Hibernate မွာ log ရိုက္မယ္ ဒီ step ၂ ခုကိုေတာ့ လုပ္ကို လုပ္ရမယ္။

Load the required jar files

slf4j.jar နဲ႔ log4j.jar files မ်ားကို hibernate jar files (load လုပ္ၿပီးသား) နဲ႔အတူ load လုပ္ရမွာ ၿဖစ္ပါတယ္။

Create log4j.xml file

log ရိုက္ဖို႔ လမ္းေၾကာင္းနဲ႔ file name ကို xml file ထဲမွာ တစ္ခါတည္း ထည့္ေပးလိုက္ဖို႔ လိုအပ္ပါလိမ့္မယ္။

\*\*ဒီ log file ကို စမ္းမယ္ဆိုရင္ main method ပါတဲ့ hibernate test တစ္ခုခုနဲ႔စမ္းပါ။

log4j.xml

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/"debug="false">

<appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="[%d{dd/MM/yy hh:mm:ss:sss z}] %5p %c{2}: %m%n" />

</layout>

</appender>

<appender name="ASYNC" class="org.apache.log4j.AsyncAppender">

<appender-ref ref="CONSOLE" />

<appender-ref ref="FILE" />

</appender>

<appender name="FILE" class="org.apache.log4j.RollingFileAppender">

<param name="File" value="C:/Users/Techfunmmr/AppData/javatpointlog.log" />

<param name="MaxBackupIndex" value="100" />

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="[%d{dd/MM/yy hh:mm:ss:sss z}] %5p %c{2}: %m%n" />

</layout>

</appender>

<category name="org.hibernate">

<priority value="DEBUG" />

</category>

<category name="java.sql">

<priority value="debug" />

</category>

<root>

<priority value="INFO" />

<appender-ref ref="FILE" />

</root>

</log4j:configuration>

2. HB with Log4j 2

Hibernate Logging by Log4j using properties file

Log4j xml file အသံုးၿပဳၿပီး Hibernate Logging ကိုလုပ္ေဆာင္ရန္ အဆင့္မ်ား

1. hibernate နဲ႔ log4j jar files မ်ားကို load ဆြဲတင္
2. src folder ထဲမွာ log4j.properties file ကို create လုပ္။အဲ့မွာ log ရိုက္ေစခ်င္တဲ့ log file ကို လမ္းေၾကာင္းနဲ႔နာမည္နဲ႔ တစ္ခါတည္းေပးထားခဲ့ပါ။

Log4j.properties file အသံုးၿပဳၿပီး Hibernate Logging ကိုလုပ္ေဆာင္တဲ့ example

Load the required jar files

slf4j.jar နဲ႔ log4j.jar files မ်ားကို hibernate jar files (load လုပ္ၿပီးသား) နဲ႔အတူ load လုပ္ရမွာ ၿဖစ္ပါတယ္။

Create log4j.xml file

log ရိုက္ဖို႔ လမ္းေၾကာင္းနဲ႔ file name ကို log4j.properties file ထဲမွာ တစ္ခါတည္း ထည့္ေပးလိုက္ဖို႔ လိုအပ္ပါလိမ့္မယ္။

\*\*ဒီ log file ကို စမ္းမယ္ဆိုရင္ main method ပါတဲ့ hibernate test တစ္ခုခုနဲ႔စမ္းပါ။ ဒါေပမယ့္ xml file ေရာ properties file တစ္ခါတည္း စမ္းလို႔ မရဘူး။ တစ္ခုတည္းထည့္စမ္းလို႔ ရမယ္။ ႏွစ္ခုလံုးထည့္ရင္ေတာ့ xml file ကိုပဲ သံုးသြားလိမ့္မယ္။

log4j.properties

# Direct log messages to a log file

log4j.appender.file=org.apache.log4j.RollingFileAppender

log4j.appender.file.File=C:\\Users\\Techfunmmr\\AppData\\log4jPropertiesTest.log

log4j.appender.file.MaxFileSize=1MB

log4j.appender.file.MaxBackupIndex=1

log4j.appender.file.layout=org.apache.log4j.PatternLayout

log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

# Direct log messages to stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.Target=System.out

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

# Root logger option

log4j.rootLogger=INFO, file, stdout

# Log everything. Good for troubleshooting

log4j.logger.org.hibernate=INFO

# Log all JDBC parameters

log4j.logger.org.hibernate.type=ALL

**Inheritance Mapping**

1. Inheritance Mapping
2. Table Per Hierarchy
3. TPH using Annotation
4. Table Per Concrete
5. TPC using Annotation
6. Table Per Subclass
7. TPS using Annotation

1. Inheritance Mapping

ကၽြန္ေတာ္တို႔အေနနဲ႔ inheritance hierarchy classes ေတြကို database ထဲက table နဲ႔ map လုပ္လို႔ ရႏိုင္ပါတယ္။ hibernate inheritance ထဲမွာ mapping strategies ၃ မ်ိဳး ရွိပါတယ္။

1. Table Per Hierarchy
2. Table Per Concrete class
3. Table Per Subclass

Table Per Hierarchy

hierarchy တစ္ခုလံုးကို map လုပ္ဖို႔အတြက္ table တစ္လံုးသာလွ်င္ လိုအပ္ပါတယ္။ class မွာ extra column (discriminator column လို႔ေခၚတဲ့) တစ္ခုကို class ကို identify လုပ္ဖို႔ရန္အတြက္ ထပ္ထည့္ဖို႔ေတာ့ လိုပါလိမ့္မယ္။ ဒါေပမယ့္ table ထဲမွာေတာ့ null value ေတြ ၀င္သြားလိမ့္မယ္။

Table Per Concrete class

tables are created as per class ကို ဆိုလိုသည္။ ဒါေပမယ့္ subclass tables ထဲမွာေတာ့ duplicate column ေပါင္းထည့္ရမယ္။

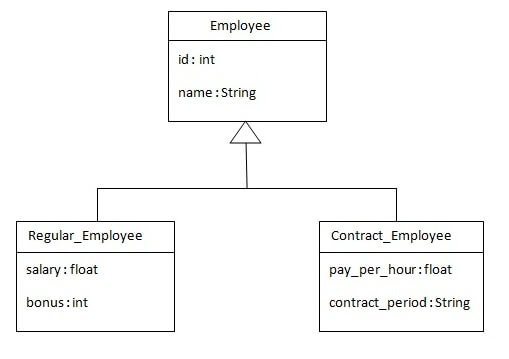
Table Per Subclass

tables are created as per class ကို ဆိုလိုသည္။ ဒါေပမယ့္ foreign key နဲ႔ relate လုပ္ထားပါတယ္။ ဒါေၾကာင့္ ဒီမွာ duplicate columns မ်ား မရွိပါဘူး။

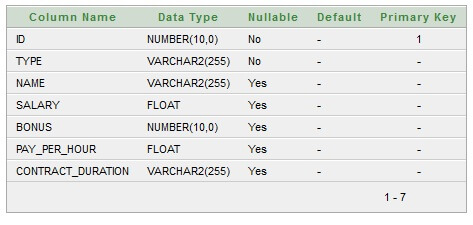
2. Table per Hierarchy (class အမ်ားၾကီး table တစ္လံုး - xml)

Hibernate Table Per Hierarchy using xml file

ကၽြန္ေတာ္တို႔အေနနဲ႔ hierarchy တစ္ခုလံုးကို single table only တစ္ခုတည္းနဲ႔ map လုပ္လို႔ရပါတယ္။ class ကုိ identify လုပ္ဖို႔ရန္ extra column (discriminator column လို႔ေခၚတဲ့) တစ္ခုကို table ထဲမွာေတာ့ create လုပ္ရမယ္။



hierarchy ထဲမွာ class ၃ ခုပါ၀င္ပါတယ္။ Regular\_Employee နဲ႔ Contract\_Employee classes တို႔ေတြအတြက္ Employee class က super class တစ္ခုၿဖစ္ပါတယ္။



Example of Table per class hierarchy

အခု example မွာ class ၃ ခုကို create လုပ္မယ္ ၿပီးရင္ employee.hbm.xml file ထဲမွာ အဲ့ file ၃ file ကို mapping လုပ္ပါမယ္။

1) Create the Persistent classes

Employee1.java

**public** **class** Employee1 {

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

**private** String name;

getter/setter

Regular\_Employee1.java

**public** **class** Regular\_Employee1 **extends** Employee1 {

**private** **float** salary;

**private** **int** bonus; **getter/setter**

Contract\_Employee1.java

**public** **class** Contract\_Employee1 **extends** Employee1 {

**private** **float** pay\_per\_hour;

**private** String contract\_duration; **getter and setter**

employee1.hbm.xml

<hibernate-mapping>

<class name=*"inheritance\_mapping.table\_per\_hierarchy.Employee1"* table=*"employee\_1"*

discriminator-value=*"emp"*>

<id name=*"id"*>

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

</id>

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

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

<subclass name=*"inheritance\_mapping.table\_per\_hierarchy.Regular\_Employee1"*

discriminator-value=*"reg\_emp"*>

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

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

</subclass>

<subclass name=*"inheritance\_mapping.table\_per\_hierarchy.Contract\_Employee1"*

discriminator-value=*"con\_emp"*>

<property name=*"pay\_per\_hour"*></property>

<property name=*"contract\_duration"*></property>

</subclass>

</class>

</hibernate-mapping>

hibernate.cgf.xml

<!-- inheritance\_mapping.table\_per\_hierarchy -->

<mapping resource=*"inheritance\_mapping/table\_per\_hierarchy/employee1.hbm.xml"* />

StoreData1.java

**public** **class** StoreData1 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Employee1 e1 = **new** Employee1();

e1.setName("Gaurav Chawla");

Regular\_Employee1 e2 = **new** Regular\_Employee1();

e2.setName("Vivek Kumar");

e2.setSalary(50000);

e2.setBonus(5);

Contract\_Employee1 e3 = **new** Contract\_Employee1();

e3.setName("Arjun Kumar");

e3.setPay\_per\_hour(1000);

e3.setContract\_duration("15 hours");

session.persist(e1);

session.persist(e2);

session.persist(e3);

t.commit();

session.close();

System.***out***.println("success");}}

3. TPH using Annotation (class အမ်ားၾကီး table တစ္လံုး - annotation)

Hibernate Table Per Hierarchy using annotation

Employee2.java

@Entity

@Table(name = "employee\_2")

@Inheritance(strategy = InheritanceType.***SINGLE\_TABLE***)

@DiscriminatorColumn(name = "type", discriminatorType = DiscriminatorType.***STRING***)

@DiscriminatorValue(value = "employee2")

**public** **class** Employee2 {

@Id

@GeneratedValue(strategy = GenerationType.***AUTO***)

@Column(name = "id")

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

@Column(name = "name")

**private** String name; **//getter and setter**

Regular\_Employee2.java

@Entity

@DiscriminatorValue("regular\_employee\_2")

**public** **class** Regular\_Employee2 **extends** Employee2 {

@Column(name = "salary")

**private** **float** salary;

@Column(name = "bonus")

**private** **int** bonus; **//getter and setter**

Contract\_Employee2.java

@Entity

@DiscriminatorValue("contract\_employee\_2")

**public** **class** Contract\_Employee2 **extends** Employee2 {

@Column(name = "pay\_per\_hour")

**private** **float** pay\_per\_hour;

@Column(name = "contract\_duration")

**private** String contract\_duration; **//getter and setter**

hibernate.cgf.xml

<!-- inheritance\_mapping.tph\_using\_annotation -->

<mapping class=*"inheritance\_mapping.tph\_using\_annotation.Employee2"*/>

<mapping class=*"inheritance\_mapping.tph\_using\_annotation.Regular\_Employee2"*/>

<mapping class=*"inheritance\_mapping.tph\_using\_annotation.Contract\_Employee2"*/>

StoreTest2.java

**public** **class** StoreTest2 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Employee2 e1 = **new** Employee2();

e1.setName("Gaurav Chawla");

Regular\_Employee2 e2 = **new** Regular\_Employee2();

e2.setName("Vivek Kumar");

e2.setSalary(50000);

e2.setBonus(5);

Contract\_Employee2 e3 = **new** Contract\_Employee2();

e3.setName("Arjun Kumar");

e3.setPay\_per\_hour(1000);

e3.setContract\_duration("15 hours");

session.persist(e1);

session.persist(e2);

session.persist(e3);

t.commit();

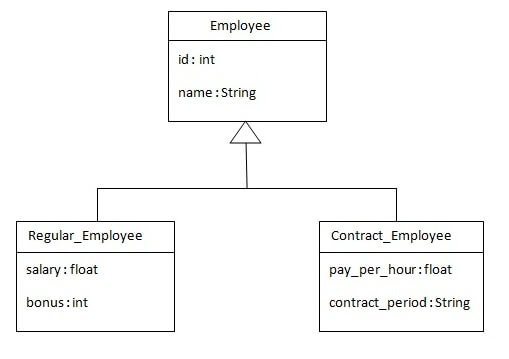
session.close();

System.***out***.println("success");}}

4. Table Per Concrete (class တစ္ခု table တစ္လံုး parent table field မ်ား child table ထဲေရာက္ - xml)

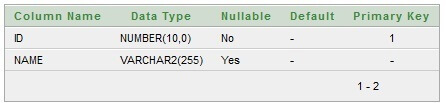
database ထဲမွာ တစ္ခုနဲ႔ တစ္ခု relation မရွိတဲ့ table ၃ လံုး ရွိေနလိမ့္မယ္။ table per concrete class strategy ကို နည္းလမ္း ၂ မ်ိဳးနဲ႔ table နဲ႔ map လုပ္လို႔ရႏိုင္တယ္။

* By union-subclass element
* By self creating the table for each class

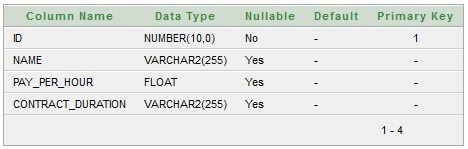


database ထဲမွာ table ၃ လံုးရွိၿပီးေတာ့ အဲ့ table တစ္လံုးခ်င္းစီက class တစ္ခုခ်င္းစီကို ကိုယ္စားၿပဳပါတယ္။ class ရဲ႕ union-subclass subelement က sub class ကို သတ္မွတ္ေပးပါတယ္။ parent class ရဲ႕ column ေတြၿဖစ္ၾကတဲ့ id နဲ႔ name ကို sub table မ်ားထဲမွာ သြားထည့္ေပးရမယ္။

Employee3 class ရဲ႕ table structure



Contract\_Employee3 class ရဲ႕ table structure



Regular\_Employee3 class ရဲ႕ table structure



Example of Table per concrete class

ဒီ example မွာဆိုရင္ class ၃ ခုကို create လုပ္မယ္။ employee3.hbm.xml file ထဲမွာ အဲ့ class ေတြကို mapping လုပ္ပါမယ္။

Employee3.java

**public** **class** Employee3 {

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

**private** String name;

Regular\_Employee3.java

**public** **class** Regular\_Employee3 **extends** Employee3{

**private** **float** salary;

**private** **int** bonus;

Contract\_Employee3.java

public class Contract\_Employee3 extends Employee3 {

private float pay\_per\_hour;

private String contract\_duration;

employee3.hbm.xml

<hibernate-mapping>

<class name=*"inheritance\_mapping.table\_per\_con\_concrete.Employee3"* table=*"employee\_3"*>

<id name=*"id"*>

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

</id>

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

<union-subclass

name=*"inheritance\_mapping.table\_per\_con\_concrete.Regular\_Employee3"* table=*"regular\_employee\_3"*>

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

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

</union-subclass>

<union-subclass

name=*"inheritance\_mapping.table\_per\_con\_concrete.Contract\_Employee3"* table=*"contract\_employee\_3"*>

<property name=*"pay\_per\_hour"*></property>

<property name=*"contract\_duration"*></property>

</union-subclass>

</class>

</hibernate-mapping>

hibernate.cfg.xml

<!-- inheritance\_mapping.table\_per\_con\_concrete -->

<mapping resource=*"inheritance\_mapping/table\_per\_con\_concrete/employee3.hbm.xml"* />

StoreData3.java

**public** **class** StoreData3 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Employee3 e1 = **new** Employee3();

e1.setName("Gaurav Chawla");

Regular\_Employee3 e2 = **new** Regular\_Employee3();

e2.setName("Vivek Kumar");

e2.setSalary(50000);

e2.setBonus(5);

Contract\_Employee3 e3 = **new** Contract\_Employee3();

e3.setName("Arjun Kumar");

e3.setPay\_per\_hour(1000);

e3.setContract\_duration("15 hours");

session.persist(e1);

session.persist(e2);

session.persist(e3);

t.commit();

session.close();

System.***out***.println("success");}}

5. TPC Using Annotation (class တစ္ခု table တစ္လံုး parent table field မ်ား child table ထဲေရာက္ - annotation)

Table Per Concrete class using Annotation

Table Per Concrete class မွာဆိုရင္ class တစ္ခုကို table တစ္လံုး create လုပ္ပါမယ္။ အဲ့တာေၾကာင့္ table ထဲမွာ nullable values မ်ား မရွိပါဘူး။ ဒီမွာ မေကာင္းတဲ့အခ်က္ကေတာ့ sub tables မ်ားထဲတြင္ duplicate columns မ်ား create လုပ္လို႔ရေနၿခင္းပဲ ၿဖစ္ပါတယ္။

parent class ထဲမွာ @Inheritance(strategy = InheritanceType.TABLE\_PER\_CLASS) ဆိုတဲ့ annotation ကို သံုးဖို႔ လိုအပ္ပါတယ္။

sub class ထဲမွာေတာ့ @AttributeOverrides ဆိုတဲ့ annotation ကို သံုးဖို႔လိုအပ္ပါတယ္။

* table per concrete class strategy ကို သံုးမယ္ဆိုရင္ @Inheritance(strategy = InheritanceType.TABLE\_PER\_CLASS) ဆိုတဲ့ annotation ကို သံုးကို သံုးေပးရမယ္။ သူ႕ကိုေတာ့ parent class ထဲမွာပဲ သံုးရမွာ ၿဖစ္ပါတယ္။
* sub class ေတြထဲမွာ parent class attributes မ်ားကို overridden လုပ္မယ္ဆိုရင္ @AttributeOverride ဆိုတဲ့ annotation ကို သံုးေပးရမယ္။ အဲ့လိုဆိုရင္ parent class ေတြရဲ႕ table column ေတြက subclass table ေတြထဲမွာ သြားေပါင္းလိမ့္မယ္။

Example of Table per concrete class

Employee4.java

@Entity

@Table(name = "employee\_4")

@Inheritance(strategy = InheritanceType.***TABLE\_PER\_CLASS***)

**public** **class** Employee4 {

@Id

@GeneratedValue(strategy = GenerationType.***AUTO***)

@Column(name = "id")

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

@Column(name = "name")

**private** String name;

Regular\_Employee4 .java

@Entity

@Table(name = "regular\_employee\_4")

@AttributeOverrides({ @AttributeOverride(name = "id", column = @Column(name = "id")),

@AttributeOverride(name = "name", column = @Column(name = "name")) })

**public** **class** Regular\_Employee4 **extends** Employee4 {

@Column(name = "salary")

**private** **float** salary;

@Column(name = "bonus")

**private** **int** bonus;

Contract\_Employee4 .java

@Entity

@Table(name = "contract\_employee\_4")

@AttributeOverrides({ @AttributeOverride(name = "id", column = @Column(name = "id")),

@AttributeOverride(name = "name", column = @Column(name = "name")) })

**public** **class** Contract\_Employee4 **extends** Employee4 {

@Column(name = "pay\_per\_hour")

**private** **float** pay\_per\_hour;

@Column(name = "contract\_duration")

**private** String contract\_duration;

hibernate.cfg.xml

<!-- inheritance\_mapping.tpc\_using\_annotation -->

<mapping class=*"inheritance\_mapping.tpc\_using\_annotation.Employee4"*/>

<mapping class=*"inheritance\_mapping.tpc\_using\_annotation.Contract\_Employee4"*/>

<mapping class=*"inheritance\_mapping.tpc\_using\_annotation.Regular\_Employee4"*/>

StoreData4 .java

**public** **class** StoreData4 {

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

StandardServiceRegistry ssr=**new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta=**new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory=meta.getSessionFactoryBuilder().build();

Session session=factory.openSession();

Transaction t=session.beginTransaction();

Employee4 e1=**new** Employee4();

e1.setName("Gaurav Chawla");

Regular\_Employee4 e2=**new** Regular\_Employee4();

e2.setName("Vivek Kumar");

e2.setSalary(50000);

e2.setBonus(5);

Contract\_Employee4 e3=**new** Contract\_Employee4();

e3.setName("Arjun Kumar");

e3.setPay\_per\_hour(1000);

e3.setContract\_duration("15 hours");

session.persist(e1);

session.persist(e2);

session.persist(e3);

t.commit();

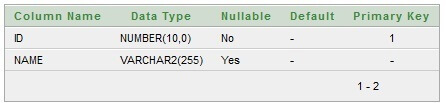
session.close();

System.***out***.println("success"); } }

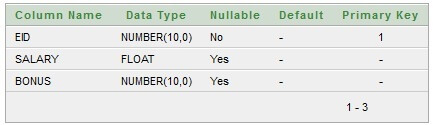
6. Table Per Subclass (class တစ္ခု table တစ္လံုး parent table ရဲ႕ id က child table ထဲမွာ fk ၿဖစ္ - xml)

table per subclass မွာဆိုရင္ primary key foreign key relationship ပံုစံနဲ႔သြားမွာ ၿဖစ္ပါတယ္။ အဲ့လို join မယ္ဆိုရင္ <joined-subclass> element ကို သံုးရမွာ ၿဖစ္ပါတယ္။

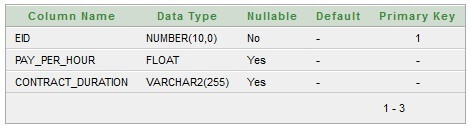
* joined-subclass ကို subclass အၿဖစ္ သတ္မွတ္ဖို႔ အသံုးၿပဳပါတယ္။
* key sub-element ကိုေတာ့ sub ထဲမွာ parent ရဲ႕ id ကို foreign key အၿဖစ္ သတ္မွတ္ဖို႔ရန္ အသံုးၿပဳပါတယ္။



Regular\_Employee5 class



Contract\_Employee5 class



Example of Table per subclass class

Employee5.java

**public** **class** Employee5 {

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

**private** String name;

Regular\_Employee5.java

**public** **class** Regular\_Employee5 **extends** Employee5 {

**private** **float** salary;

**private** **int** bonus;

Contract\_Employee5.java

**public** **class** Contract\_Employee5 **extends** Employee5 {

**private** **float** pay\_per\_hour;

**private** String contract\_duration;

employee5.hbm.xml

<hibernate-mapping>

<class name="inheritance\_mapping.table\_per\_subclass.Employee5" table="employee\_5">

<id name="id">

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

</id>

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

<joined-subclass

name="inheritance\_mapping.table\_per\_subclass.Regular\_Employee5" table="regular\_employee\_5">

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

<property name="salary"></property>

<property name="bonus"></property>

</joined-subclass>

<joined-subclass

name="inheritance\_mapping.table\_per\_subclass.Contract\_Employee5" table="contract\_employee\_5">

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

<property name="pay\_per\_hour"></property>

<property name="contract\_duration"></property>

</joined-subclass>

</class>

</hibernate-mapping>

hibernate.cfg.xml

<!-- inheritance\_mapping.table\_per\_subclass -->

<mapping resource=*"inheritance\_mapping/table\_per\_subclass/employee5.hbm.xml"*/>

StoreData5.java

**public** **class** StoreData5 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.buildSessionFactory();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Employee5 e1 = **new** Employee5();

e1.setName("Gaurav Chawla");

Regular\_Employee5 e2 = **new** Regular\_Employee5();

e2.setName("Vivek Kumar");

e2.setSalary(50000);

e2.setBonus(5);

Contract\_Employee5 e3 = **new** Contract\_Employee5();

e3.setName("Arjun Kumar");

e3.setPay\_per\_hour(1000);

e3.setContract\_duration("15 hours");

session.persist(e1);

session.persist(e2);

session.persist(e3);

t.commit();

session.close();

System.***out***.println("success");}}

7. TPS Using Annotation(class တစ္ခု table တစ္လံုး parent table ရဲ႕ id က child table ထဲမွာ fk ၿဖစ္ - annotation)

tps မွာဆိုရင္ parent class ထဲမွာ @Inheritance(strategy=InheritanceType.JOINED) လို႔ ေရးၿပီး sub class ထဲမွာဆိုရင္ @PrimaryKeyJoinColumn လို႔ေရးပါတယ္။

Example of Table per subclass class using Annotation

Employee6.java

@Entity

@Table(name = "employee\_6")

@Inheritance(strategy = InheritanceType.***JOINED***)

**public** **class** Employee6 {

@Id

@GeneratedValue(strategy = GenerationType.***AUTO***)

@Column(name = "id")

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

@Column(name = "name")

**private** String name;

Regular\_Employee6.java

@Entity

@Table(name = "regular\_employee\_6")

@PrimaryKeyJoinColumn(name = "ID")

**public** **class** Regular\_Employee6 **extends** Employee6 {

@Column(name = "salary")

**private** **float** salary;

@Column(name = "bonus")

**private** **int** bonus;

Contract\_Employee6.java

@Entity

@Table(name = "contract\_employee\_6")

@PrimaryKeyJoinColumn(name = "ID")

public class Contract\_Employee6 extends Employee6 {

@Column(name = "pay\_per\_hour")

private float pay\_per\_hour;

@Column(name = "contract\_duration")

private String contract\_duration;

hibernate.cfg.xml

<!-- inheritance\_mapping.tps\_using\_annotation -->

<mapping class=*"inheritance\_mapping.tps\_using\_annotation.Employee6"*/>

<mapping class=*"inheritance\_mapping.tps\_using\_annotation.Contract\_Employee6"*/>

<mapping class=*"inheritance\_mapping.tps\_using\_annotation.Regular\_Employee6"*/>

StoreData6.java

**public** **class** StoreData6 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Employee6 e1 = **new** Employee6();

e1.setName("Gaurav Chawla");

Regular\_Employee6 e2 = **new** Regular\_Employee6();

e2.setName("Vivek Kumar");

e2.setSalary(50000);

e2.setBonus(5);

Contract\_Employee6 e3 = **new** Contract\_Employee6();

e3.setName("Arjun Kumar");

e3.setPay\_per\_hour(1000);

e3.setContract\_duration("15 hours");

session.persist(e1);

session.persist(e2);

session.persist(e3);

t.commit();

session.close();

System.***out***.println("success");}}

**Hibernate Mapping**

1. Collection Mapping
2. Mapping List
3. Mapping Bag
4. Mapping Set
5. Mapping Map
6. One To Many XML
7. One To Many Annotation
8. Many To Many XML
9. Many To Many Annotation
10. One To One XML
11. One To One Annotation
12. Many To One XML
13. Many To One Annotation
14. Bidirectional
15. Lazy Collection
16. Component Mapping

1. Collection Mapping

ကၽြန္ေတာ္တို႔က hibernate ထဲမွာ persistent class ရဲ႕ collection elements မ်ားကို map လုပ္ေဆာင္ႏိုင္ပါတယ္။ ဒါေပမယ့္ ေအာက္က collection type တစ္ခုခုကိုေတာ့ Persistent class ထဲမွာ declare လုပ္ေပးဖို႔ေတာ့ လိုအပ္ပါလိမ့္မယ္။

1. java.util.List
2. java.util.Set
3. java.util.SortedSet
4. java.util.Map
5. java.util.SortedMap
6. java.util.Collection
7. or write the implementation of org.hibernate.usertype.UserCollectionType

collection element အတြက္ persistent class ကိုေအာက္ေဖာ္ၿပပါအတိုင္း ေရးႏိုင္ပါတယ္။

**import** java.util.List;

**public** **class** Question {

**private** **int** id;

**private** String qname;

**private** List<String> answers;//List can be of any type

  //getters and setters  }

mapping file ထဲက mapping collection

collection ကို map လုပ္ဖို႔ရန္ <class> elements ထဲမွာ subelements မ်ားစြာ ရွိပါတယ္။ သူတို႔ေတြကေတာ့ <list>, <bag>, <set> နဲ႔ <map> တို႔ ၿဖစ္ၾကပါတယ္။

<class name="com.javatpoint.Question" table="q100">

<id name="id">

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

</id>

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

<list name="answers" table="ans100">

<key column="qid"></key>

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

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

</list>

</class>

list ထဲမွာ sub elements ၃ ခုကို အသံုးၿပဳႏိုင္ပါတယ္။

1. <key> ဆိုတဲ့ element ကို ေအာက္မွာ ၿပထားတဲ့အတိုင္းေၿပာရရင္ Question class identifier ကို အေၿခခံၿပီး အဲ့ table ထဲမွာ foreign key သတ္မွတ္ရန္ အသံုးၿပဳပါတယ္။
2. <index> ဆိုတဲ့ element ကို type ကို identify လုပ္ရန္အသံုးၿပဳပါတယ္။ List နဲ႔ Map ဟာ indexed collection မ်ားၿဖစ္ၾကပါတယ္။
3. <element> collection ရဲ႕ element ကို define လုပ္ဖို႔ရန္ အသံုးၿပဳပါတယ္။

collection က string objects မ်ားကို store လုပ္ခဲ့မယ္ဆိုရင္ အဲ့တာက collection ရဲ႕ mapping ၿဖစ္ပါတယ္။ ဒါေပမယ့္ အကယ္လို႔မ်ား collection က entity reference (another class objects) မ်ားကို store လုပ္ခဲ့မယ္ဆိုရင္ ကၽြန္ေတာ္တို႔အေနနဲ႔ <one-to-many> သို႔မဟုတ္ <many-to-many> element ဆိုတဲ့ element မ်ားကို သတ္မွတ္ဖို႔ လိုအပ္ပါလိမ့္မယ္။

Question.java

import java.util.List;

public class Question {

private int id;

private String qname;

private List<Answer> answers;//Here, List stores the objects of Answer class

//getters and setters }

Answer.java

import java.util.List;

public class Answer {

private int id;

private String answer;

private String posterName;

//getters and setters

}

mapping file ကေတာ့ ဒီလို ၿဖစ္လိမ့္မယ္။

<class name="com.javatpoint.Question" table="q100">

<id name="id">

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

</id>

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

<list name="answers" >

<key column="qid"></key>

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

<one-to-many class="com.javatpoint.Answer" />

</list>

</class>

ဒီမွာ list ကို one to many relationship နဲ႔ခ်ိတ္ထားတယ္ ဆိုလိုတာက question တစ္ခုမွာ answer အမ်ားၾကီး ရွိရမယ္။

Understanding key element

key element ကို original identity ကို အေၿခခံထားတဲ့ joint table ထဲမွာ foreign key define လုပ္ဖို႔ရန္ အသံုးၿပဳပါတယ္။ foreign key က default အားၿဖင့္ nullable ၿဖစ္ပါတယ္။ ဆိုေတာ့ foreign key ကို non nullable ၿဖစ္ခ်င္တယ္ဆိုရင္ေတာ့ ေအာက္ကလိုမ်ိဳးသတ္မွတ္ေပးရမယ္။

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

key element ရဲ႕ attributes ေတြဟာ column, on-delete, property-ref, not-null, update နဲ႔ unique ၿဖစ္ၾကပါတယ္။

<key

column="columnname"

on-delete="noaction|cascade"

not-null="true|false"

property-ref="propertyName"

update="true|false"

unique="true|false"

/>

Indexed collections

indexed collections ကို form ၂ မ်ိဳးဆိုၿပီးေတာ့ category ခြဲထားပါတယ္။

1. indexed
2. non-indexed

List နဲ႔ Map collection က indexed မ်ားၿဖစ္ၾကၿပီးေတာ့ set နဲ႔ bag collections မ်ားကေတာ့ non-indexed မ်ားၿဖစ္ၾကပါတယ္။ indexed collection ဆိုတာက List နဲ႔ Map က additional element <index> တစ္ခုေပါင္းထည့္ဖို႔ လိုအပ္တယ္လို႔ ဆိုလိုပါတယ္။

Collection Elements

ollection elements ေတြ value သို႔မဟုတ္ entity reference (another class object) ရွိပါတယ္။ ကၽြန္ေတာ္တို႔အေနနဲ႔ element ၄ ခုကို အသံုးၿပဳႏိုင္ပါတယ္။

1. element
2. component-element
3. one-to-many, သို႔မဟုတ္
4. many-to-many

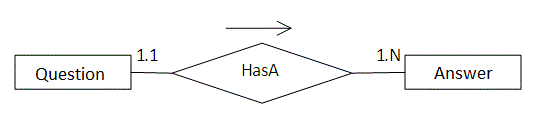
element နဲ႔ component-element ေတြကို သာမန္ value ေတြအတြက္သာ အသံုးၿပဳပါတယ္ ဥပမာ string, int etc။ one-to-many နဲ႔ many-to-many တို႔ဆိုရင္ေတာ့ map လုပ္ဖို႔ရန္ entity reference မ်ားကို အသံုးၿပဳပါတယ္။

2. Mapping List

Mapping List in Collection Mapping (using xml file)

ကၽြန္ေတာ္တို႔ရဲ႕ persistent object မွာ list object ရွိခဲ့တယ္ဆိုရင္ <list> element ကိုသံုးၿပီးေတာ့ mapping file ထဲမွာၿဖစ္ၿဖစ္ annotation ကိုသံုးၿပီးေတာ့ၿဖစ္ၿဖစ္ map လုပ္လိုက္လို႔ရပါတယ္။

question တစ္ခုမွာ answer ေတြအမ်ားၾကီး ရွိတဲ့သေဘာ။



List နဲ႔ Map ဟာဆိုရင္ index based collection ၿဖစ္တဲ့အတြက္ေၾကာင့္ column ထဲမွာ index အတြက္ column အပိုတစ္ခု ထည့္ေပးရမွာ ၿဖစ္ပါတယ္။

Example of mapping list in collection mapping

Question1.java

**public** **class** Question1 {

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

**private** String qname;

**private** List<String> answers;

question1.hbm.xml

<hibernate-mapping>

<class name=*"hibernate\_mapping.mapping\_list.Question1"* table=*"question\_1"*>

<id name=*"id"*>

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

</id>

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

<list name=*"answers"* table=*"answer\_1"*>

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

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

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

</list>

</class>

</hibernate-mapping>

hibernate.cfg.xml

<!-- hibernate\_mapping.mapping\_list -->

<mapping resource=*"hibernate\_mapping/mapping\_list/question1.hbm.xml"*/>

StoreData7.java

**public** **class** StoreData7 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

ArrayList<String> list1 = **new** ArrayList<String>();

list1.add("Java is a programming language");

list1.add("Java is a platform");

ArrayList<String> list2 = **new** ArrayList<String>();

list2.add("Servlet is an Interface");

list2.add("Servlet is an API");

Question1 question1 = **new** Question1();

question1.setQname("What is Java?");

question1.setAnswers(list1);

Question1 question2 = **new** Question1();

question2.setQname("What is Servlet?");

question2.setAnswers(list2);

session.persist(question1);

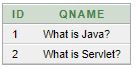
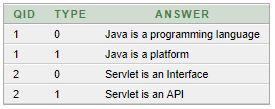
session.persist(question2);

t.commit();

session.close();

System.***out***.println("success");}}

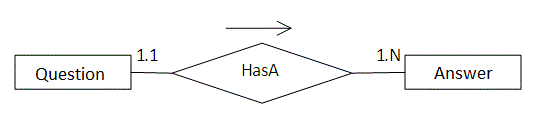
answer list ထဲက data မ်ားက answer\_1 table ထဲကို ၀င္သြားလိမ့္မယ္။ question ရဲ႕ id ကေတာ့ fb key အေနနဲ႔ ရွိေနလိမ့္မယ္။

3. Mapping Bag

Mapping Bag in Collection Mapping (using xml file)

ကၽြန္ေတာ္တို႔ရဲ႕ persistence class ထဲမွာ list object ရွိခဲ့မယ္ဆိုရင္ mapping file ထဲမွာ list သို႔မဟုတ္ bag element နဲ႔ map လုပ္လို႔ ရပါတယ္။ bag က list နဲ႔တူပါတယ္ သို႔ေပမယ့္ list လို index element မလိုပါဘူး။



Example of mapping bag in collection mapping

အခု example မွာဆိုရင္ bag နဲ႔ collection mapping ကို လုပ္ေဆာင္မွာ ၿဖစ္ပါတယ္။

Question2.java

**public** **class** Question2 {

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

**private** String qname;

**private** List<String> answers;

question2.hbm.xml

<hibernate-mapping>

<class name=*"hibernate\_mapping.mapping\_bag.Question2"* table=*"question\_2"*>

<id name=*"id"*>

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

</id>

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

<bag name=*"answers"* table=*"answer\_2"*>

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

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

</bag>

</class>

</hibernate-mapping>

hibernate.cfg.xml

<!-- hibernate\_mapping.mapping\_bag -->

<mapping resource=*"hibernate\_mapping/mapping\_bag/question2.hbm.xml"*/>

StoreData8.java

**public** **class** StoreData8 {

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

StandardServiceRegistry ssr=**new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta=**new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory=meta.buildSessionFactory();

Session session=factory.openSession();

Transaction t=session.beginTransaction();

ArrayList<String> list1=**new** ArrayList<String>();

list1.add("Java is a programming language");

list1.add("Java is a platform");

ArrayList<String> list2=**new** ArrayList<String>();

list2.add("Servlet is an Interface");

list2.add("Servlet is an API");

Question2 question1=**new** Question2();

question1.setQname("What is Java?");

question1.setAnswers(list1);

Question2 question2=**new** Question2();

question2.setQname("What is Servlet?");

question2.setAnswers(list2);

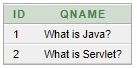
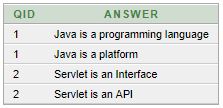
session.persist(question1);

session.persist(question2);

t.commit();

session.close();

System.***out***.println("success"); } }

How to fetch the data (table ထဲက data ေတြကို ၿပန္ၿပၿခင္း)

HQL ကိုသံုးၿပီး answers အပါအ၀င္ questions record အကုန္လံုးကို ဘယ္လို fetch လုပ္မလဲ။

FetchData2.java

**public** **class** FetchData2 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.buildSessionFactory();

Session session = factory.openSession();

TypedQuery query = session.createQuery("from hibernate\_mapping.mapping\_bag.Question2");

List<Question2> list = query.getResultList();

Iterator<Question2> itr = list.iterator();

**while** (itr.hasNext()) {

Question2 q = itr.next();

System.***out***.println("Question Name: " + q.getQname());

// printing answers

List<String> list2 = q.getAnswers();

Iterator<String> itr2 = list2.iterator();

**while** (itr2.hasNext()) {

System.***out***.println(itr2.next());}}

session.close();

System.***out***.println("success");}}

4. Mapping Set

Hibernate Mapping Set using XML

ကၽြန္ေတာ္တို႔ရဲ႕ persistence class မွာ set object ရွိခဲ့မယ္ဆိုရင္ mapping file ထဲမွာ set element နဲ႔ map လုပ္လို႔ရႏိုင္ပါတယ္။ set element က index element မလိုအပ္ပါဘူး။ list နဲ႔ set နဲ႔ အဓိက ကြာၿခားမွဳကေတာ့ set မွာ unique values မ်ားကိုသာ store လုပ္ေပးပါတယ္။

Example of mapping set in collection mapping

Question3.java

**public** **class** Question3 {

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

**private** String qname;

**private** Set<String> answers;

hibernate.cfg.xml

<!-- hibernate\_mapping.mapping\_set -->

<mapping resource=*"hibernate\_mapping/mapping\_set/question3.hbm.xml"*/>

question3.hbm.xml

<hibernate-mapping>

<class name=*"hibernate\_mapping.mapping\_set.Question3"* table=*"question\_3"*>

<id name=*"id"*>

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

</id>

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

<set name=*"answers"* table=*"answer\_3"*>

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

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

</set>

</class>

</hibernate-mapping>

StoreData9.java

**public** **class** StoreData9 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

HashSet<String> set1 = **new** HashSet<String>();

set1.add("Java is a programming language");

set1.add("Java is a platform");

HashSet<String> set2 = **new** HashSet<String>();

set2.add("Servlet is an Interface");

set2.add("Servlet is an API");

Question3 question1 = **new** Question3();

question1.setQname("What is Java?");

question1.setAnswers(set1);

Question3 question2 = **new** Question3();

question2.setQname("What is Servlet?");

question2.setAnswers(set2);

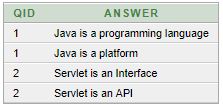
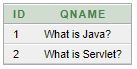
session.persist(question1);

session.persist(question2);

t.commit();

session.close();

System.***out***.println("success");}}



ကိုယ္က output ကိုၿမင္ခ်င္တယ္ဆိုရင္ေတာ့ fetchData class ကို run ၾကည့္ၿပီး ထုတ္လို႔ရပါတယ္။

5. Mapping Map

Hibernate Mapping Map using XML

RDBMS နဲ႔ Map elements မ်ားကို map လုပ္ရန္ hibernate က ခြင့္ၿပဳထားပါတယ္။သိထားသင့္တာကေတာ့ map နဲ႔ list ဆိုတာေတြက index-based collections မ်ားၿဖစ္ၾကပါတယ္။ map မွာ အလုပ္လုပ္တဲ့ပံုစံက index column ဟာဆိုရင္ key တစ္ခုအေနနဲ႔ လုပ္ေဆာင္ၿပီးေတာ့ element column ကေတာ့ value တစ္ခုအေနနဲ႔ လုပ္ေဆာင္ပါတယ္။

Example of Mapping Map in collection mapping using xml file

Question4.java

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

**private** String name,username;

**private** Map<String,String> answers;

**public** Question4() {}

**public** Question4(String name, String username, Map<String, String> answers) {

**this**.name = name;

**this**.username = username;

**this**.answers = answers;}

question4.hbm.xml

<hibernate-mapping>

<class name=*"hibernate\_mapping.mapping\_map.Question4"* table=*"question4"*>

<id name=*"id"*>

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

</id>

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

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

<map name=*"answers"* table=*"answer4"* cascade=*"all"*>

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

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

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

</map>

</class>

</hibernate-mapping>

hibernate.cfg.xml

<!-- hibernate\_mapping.mapping\_map -->

<mapping resource=*"hibernate\_mapping/mapping\_map/question4.hbm.xml"*/>

StoreData10.java

**public** **class** StoreData10 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

HashMap<String, String> map1 = **new** HashMap<String, String>();

map1.put("Java is a programming language", "John Milton");

map1.put("Java is a platform", "Ashok Kumar");

HashMap<String, String> map2 = **new** HashMap<String, String>();

map2.put("Servlet technology is a server side programming", "John Milton");

map2.put("Servlet is an Interface", "Ashok Kumar");

map2.put("Servlet is a package", "Rahul Kumar");

Question4 question1 = **new** Question4("What is Java?", "Alok", map1);

Question4 question2 = **new** Question4("What is Servlet?", "Jai Dixit", map2);

session.persist(question1);

session.persist(question2);

t.commit();

session.close();

System.***out***.println("successfully stored");}}

FetchTest.java

**public** **class** FetchTest {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

TypedQuery query = session.createQuery("from Question4 ");

List<Question4> list = query.getResultList();

Iterator<Question4> iterator = list.iterator();

**while** (iterator.hasNext()) {

Question4 question = iterator.next();

System.***out***.println("question id:" + question.getId());

System.***out***.println("question name:" + question.getName());

System.***out***.println("question posted by:" + question.getUsername());

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

Map<String, String> map = question.getAnswers();

Set<Map.Entry<String, String>> set = map.entrySet();

Iterator<Map.Entry<String, String>> iteratoranswer = set.iterator();

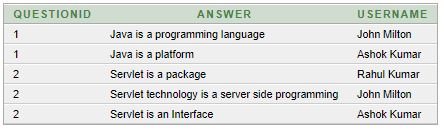
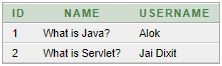
**while** (iteratoranswer.hasNext()) {

Map.Entry<String, String> entry = (Map.Entry<String, String>) iteratoranswer.next();

System.***out***.println("answer name:" + entry.getKey());

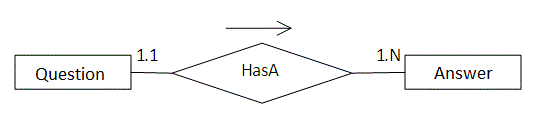
System.***out***.println("answer posted by:" + entry.getValue());}}

session.close(); }}



6. Hibernate One to Many Example using XML

persistence class ထဲမွာ list object ရွိေနမယ္ အဲ့ list object ထဲမွာလည္း entity reference ေတြပါ၀င္ေနခဲ့မယ္ဆိုရင္ ကၽြန္ေတာ္တို႔အေနနဲ႔ list element ကို map လုပ္ဖို႔ရန္ one-to-many association အသံုးၿပဳဖို႔ လိုအပ္ပါလိမ့္မယ္။



ပံုမွာၿပထားတဲ့အတိုင္းဆိုရင္ question တစ္ခုမွာ answer အမ်ားၾကီး ရွိေနတဲ့သေဘာၿဖစ္ပါတယ္။ answer တစ္ခုခ်င္းစီတိုင္းမွာလည္း ကိုယ္ပိုင္ informations မ်ားပါ၀င္မွာၿဖစ္တဲ့အတြက္ေၾကာင့္ persistent class ထဲမွာ list ကို အသံုးၿပဳမွာၿဖစ္ပါတယ္။ ဆိုလိုတဲ့သေဘာကေတာ့ Answer class ရဲ႕ reference မ်ားပါ၀င္ေနမယ္ဆိုတဲ့ သေဘာၿဖစ္ပါတယ္။ ဘာလို႔ပါ၀င္ရတာလဲဆိုေတာ့ answer မ်ားရဲ႕ collection တစ္ခုကို ကိုယ္စားၿပဳဖို႔အတြက္ ၿဖစ္ပါတယ္။

Question5.java

**public** **class** Question5 {

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

**private** String qname;

**private** List<Answer5> answers;

Answer5.java

**public** **class** Answer5 {

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

**private** String answername;

**private** String postedBy;

question5.hbm.xml

one-to-many ဆိုတဲ့ tab ကိုသံုးထားပါတယ္။

<hibernate-mapping>

<class name=*"hibernate\_mapping.one\_to\_many\_xml.Question5"* table=*"question5"*>

<id name=*"id"*>

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

</id>

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

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

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

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

<one-to-many class=*"hibernate\_mapping.one\_to\_many\_xml.Answer5"* />

</list>

</class>

<class name=*"hibernate\_mapping.one\_to\_many\_xml.Answer5"* table=*"answer5"*>

<id name=*"id"*>

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

</id>

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

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

</class>

</hibernate-mapping>

hibernate.cfg.xml

<!-- hibernate\_mapping.one\_to\_many\_xml -->

<mapping resource=*"hibernate\_mapping/one\_to\_many\_xml/question5.hbm.xml"*/>

StoreData11.java

**public** **class** StoreData11 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Answer5 ans1 = **new** Answer5();

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

ans1.setPostedBy("Ravi Malik");

Answer5 ans2 = **new** Answer5();

ans2.setAnswername("Java is a platform");

ans2.setPostedBy("Sudhir Kumar");

Answer5 ans3 = **new** Answer5();

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

ans3.setPostedBy("Jai Kumar");

Answer5 ans4 = **new** Answer5();

ans4.setAnswername("Servlet is an API");

ans4.setPostedBy("Arun");

ArrayList<Answer5> list1 = **new** ArrayList<Answer5>();

list1.add(ans1);

list1.add(ans2);

ArrayList<Answer5> list2 = **new** ArrayList<Answer5>();

list2.add(ans3);

list2.add(ans4);

Question5 question1 = **new** Question5();

question1.setQname("What is Java?");

question1.setAnswers(list1);

Question5 question2 = **new** Question5();

question2.setQname("What is Servlet?");

question2.setAnswers(list2);

session.persist(question1);

session.persist(question2);

t.commit();

session.close();

System.***out***.println("success");}}

FetchData11.java

**public** **class** FetchData11 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

TypedQuery query = session.createQuery("from Question5");

List<Question5> list = query.getResultList();

Iterator<Question5> itr = list.iterator();

**while** (itr.hasNext()) {

Question5 q = itr.next();

System.***out***.println("Question Name: " + q.getQname());

// printing answers

List<Answer5> list2 = q.getAnswers();

Iterator<Answer5> itr2 = list2.iterator();

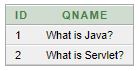
**while** (itr2.hasNext()) {

Answer5 a = itr2.next();

System.***out***.println(a.getAnswername() + ":" + a.getPostedBy()); }}

session.close();

System.***out***.println("success"); }}



7. Hibernate One to Many Example using Annotation

Title 6 က မွာ theory သေဘာမ်ားကို ေရးထားၿပီးသားၿဖစ္တဲ့အတြက္ေၾကာင့္ ထပ္မေရးေတာ့ပါ။

Question6.java

@Entity

@Table(name="question6")

**public** **class** Question6 {

@Id

@GeneratedValue(strategy=GenerationType.***TABLE***)

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

**private** String qname;

@OneToMany(cascade = CascadeType.***ALL***)

@JoinColumn(name="qid")

@OrderColumn(name="type")

**private** List<Answer6> answers6;

Answer6.java

@Entity

@Table(name = "answer6")

**public** **class** Answer6 {

@Id

@GeneratedValue(strategy = GenerationType.***TABLE***)

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

**private** String answername;

**private** String postedBy;

hibernate.cfg.xml

<!-- hibernate\_mapping.one\_to\_many\_annotation -->

<mapping class=*"hibernate\_mapping.one\_to\_many\_annotation.Question6"*/>

<mapping class=*"hibernate\_mapping.one\_to\_many\_annotation.Answer6"*/>

StoreData12.java

**public** **class** StoreData12 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Answer6 ans1 = **new** Answer6();

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

ans1.setPostedBy("Ravi Malik");

Answer6 ans2 = **new** Answer6();

ans2.setAnswername("Java is a platform");

ans2.setPostedBy("Sudhir Kumar");

Answer6 ans3 = **new** Answer6();

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

ans3.setPostedBy("Jai Kumar");

Answer6 ans4 = **new** Answer6();

ans4.setAnswername("Servlet is an API");

ans4.setPostedBy("Arun");

ArrayList<Answer6> list1 = **new** ArrayList<Answer6>();

list1.add(ans1);

list1.add(ans2);

ArrayList<Answer6> list2 = **new** ArrayList<Answer6>();

list2.add(ans3);

list2.add(ans4);

Question6 question1 = **new** Question6();

question1.setQname("What is Java?");

question1.setAnswers(list1);

Question6 question2 = **new** Question6();

question2.setQname("What is Servlet?");

question2.setAnswers(list2);

session.persist(question1);

session.persist(question2);

t.commit();

session.close();

System.***out***.println("success"); }}

FetchData12.java

**public** **class** FetchData12 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

TypedQuery query = session.createQuery("from Question6");

List<Question6> list = query.getResultList();

Iterator<Question6> itr = list.iterator();

**while** (itr.hasNext()) {

Question6 q = itr.next();

System.***out***.println("Question Name: " + q.getQname());

// printing answers

List<Answer6> list2 = q.getAnswers();

Iterator<Answer6> itr2 = list2.iterator();

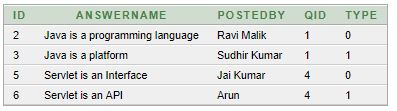
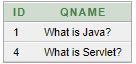
**while** (itr2.hasNext()) {

Answer6 a = itr2.next();

System.***out***.println(a.getAnswername() + ":" + a.getPostedBy()); }}

session.close();

System.***out***.println("success");}}



8. Hibernate Many to Many Example using XML

many to many relation ကို list, set, bag, map အစရွိသည္တို႔မွာ အသံုးခ်လို႔ရပါတယ္။ အခုဒီ example မွာေတာ့ list ကိုသံုးပါမယ္။ tables သံုးခု create လုပ္သြားမွာလည္း ၿဖစ္ပါတယ္။

Example of Many to Many Mapping

Question7.java

**public** **class** Question7 {

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

**private** String qname;

**private** List<Answer7> answers;

Answer7.java

**public** **class** Answer7 {

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

**private** String answername;

**private** String postedBy;

**private** List<Question7> questions;

question7.hbm.xml

<hibernate-mapping>

<class name=*"hibernate\_mapping.many\_to\_many\_xml.Question7"* table=*"question\_7"*>

<id name=*"id"* type=*"int"*>

<column name=*"q\_id"* />

<generator class=*"increment"* />

</id>

<property name=*"qname"* />

<list name=*"answers"* table=*"question\_answer\_7"* fetch=*"select"* cascade=*"all"*>

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

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

<many-to-many class=*"hibernate\_mapping.many\_to\_many\_xml.Answer7"* column=*"ans\_id"* />

</list>

</class>

</hibernate-mapping>

answer7.hbm.xml

<hibernate-mapping>

<class name=*"hibernate\_mapping.many\_to\_many\_xml.Answer7"* table=*"answer\_7"*>

<id name=*"id"* type=*"int"*>

<column name=*"ans\_id"* />

<generator class=*"increment"* />

</id>

<property name=*"answername"* />

<property name=*"postedBy"* />

</class>

</hibernate-mapping>

hibernate.cfg.xml

<!-- hibernate\_mapping.many\_to\_many\_xml -->

<mapping resource=*"hibernate\_mapping/many\_to\_many\_xml/question7.hbm.xml"*/>

<mapping resource=*"hibernate\_mapping/many\_to\_many\_xml/answer7.hbm.xml"*/>

StoreData13.java

**public** **class** StoreData13 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Answer7 ans1 = **new** Answer7();

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

ans1.setPostedBy("Ravi Malik");

Answer7 ans2 = **new** Answer7();

ans2.setAnswername("Java is a platform");

ans2.setPostedBy("Sudhir Kumar");

Question7 q1 = **new** Question7();

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

ArrayList<Answer7> l1 = **new** ArrayList<Answer7>();

l1.add(ans1);

l1.add(ans2);

q1.setAnswers(l1);

Answer7 ans3 = **new** Answer7();

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

ans3.setPostedBy("Jai Kumar");

Answer7 ans4 = **new** Answer7();

ans4.setAnswername("Servlet is an API");

ans4.setPostedBy("Arun");

Question7 q2 = **new** Question7();

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

ArrayList<Answer7> l2 = **new** ArrayList<Answer7>();

l2.add(ans3);

l2.add(ans4);

q2.setAnswers(l2);

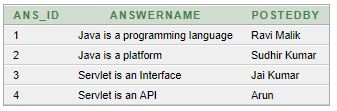
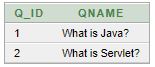
session.persist(q1);

session.persist(q2);

t.commit();

session.close();

System.***out***.println("success");}}



9. Hibernate Many to Many Example using Annotation

Title 7 က မွာ theory သေဘာမ်ားကို ေရးထားၿပီးသားၿဖစ္တဲ့အတြက္ေၾကာင့္ ထပ္မေရးေတာ့ပါ။

Question8.java

@Entity

@Table(name = "question\_8")

**public** **class** Question8 {

@Id

@GeneratedValue(strategy = GenerationType.***AUTO***)

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

**private** String qname;

@ManyToMany(targetEntity = Answer8.**class**, cascade = { CascadeType.***ALL*** })

@JoinTable(name = "question\_answer\_8",

joinColumns = { @JoinColumn(name = "q\_id") },

inverseJoinColumns = {@JoinColumn(name = "ans\_id") })

**private** List<Answer8> answers;

Answer8.java

@Entity

@Table(name = "answer\_8")

**public** **class** Answer8 {

@Id

@GeneratedValue(strategy = GenerationType.***AUTO***)

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

**private** String answername;

**private** String postedBy;

hibernate.cfg.xml

<!-- hibernate\_mapping.many\_to\_many\_annotation -->

<mapping class=*"hibernate\_mapping.many\_to\_many\_annotation.Question8"*/>

<mapping class=*"hibernate\_mapping.many\_to\_many\_annotation.Answer8"*/>

StoreData14.java

**public** **class** StoreData14 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Answer8 an1 = **new** Answer8();

an1.setAnswername("Java is a programming language");

an1.setPostedBy("Ravi Malik");

Answer8 an2 = **new** Answer8();

an2.setAnswername("Java is a platform");

an2.setPostedBy("Sudhir Kumar");

Question8 q1 = **new** Question8();

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

ArrayList<Answer8> l1 = **new** ArrayList<Answer8>();

l1.add(an1);

l1.add(an2);

q1.setAnswers(l1);

Answer8 ans3 = **new** Answer8();

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

ans3.setPostedBy("Jai Kumar");

Answer8 ans4 = **new** Answer8();

ans4.setAnswername("Servlet is an API");

ans4.setPostedBy("Arun");

Question8 q2 = **new** Question8();

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

ArrayList<Answer8> l2 = **new** ArrayList<Answer8>();

l2.add(ans3);

l2.add(ans4);

q2.setAnswers(l2);

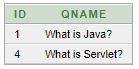
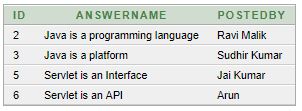
session.persist(q1);

session.persist(q2);

t.commit();

session.close();

System.***out***.println("success"); }}

9. Hibernate One to One Example using xml

hibernate မွာ one to one mapping ကိုလုပ္ေဆာင္မယ္ဆိုရင္ နည္းလမ္း ၂ မ်ိဳးနဲ႔ လုပ္ေဆာင္ဖို႔လိုအပ္ပါတယ္။

* By many-to-one element (unique="true" attribute အသံုးၿပဳၿခင္းအားၿဖင့္)
* By one-to-one element

အခု example မွာေတာ့ one-to-one element ကို အသံုးၿပဳပါမယ္။ အဲ့တာဆိုရင္ primary table ထဲမွာ foreign key ကို အသံုးၿပဳမွာ မဟုတ္ပါဘူး။ အခု example မွာ employee တစ္ေယာက္က address တစ္ခုရွိမယ္ address တစ္ခုမွာလည္း employee တစ္ခုရွိတယ္။ အခု bidirectional association သံုးပါမယ္။

Employee7.java

**public** **class** Employee7 {

**private** **int** employeeId;

**private** String name, email;

**private** Address7 address;

Address7.java

**public** **class** Address7 {

**private** **int** addressId;

**private** String addressLine1, city, state, country;

**private** **int** pincode;

**private** Employee7 employee;

employee7.hbm.xml

<hibernate-mapping>

<class name=*"hibernate\_mapping.one\_to\_one\_xml.Employee7"* table=*"employee7"*>

<id name=*"employeeId"*>

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

</id>

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

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

<one-to-one name=*"address"* cascade=*"all"*></one-to-one>

</class>

</hibernate-mapping>

address7.hbm.xml

ဒီ file မွာ အေရးအၾကီးဆံုးအပိုင္းကေတာ့ generator class ၿဖစ္ပါတယ္။ Employee class primary key အေပၚမူတည္ၿပီး foreign generator class ကၽြန္ေတာ္တို႔ အသံုးၿပဳမွာ ၿဖစ္ပါတယ္။

<hibernate-mapping>

<class name=*"hibernate\_mapping.one\_to\_one\_xml.Address7"* table=*"address\_7"*>

<id name=*"addressId"*>

<generator class=*"foreign"*>

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

</generator>

</id>

<property name=*"addressLine1"*></property>

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

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

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

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

<one-to-one name=*"employee"*></one-to-one>

</class>

</hibernate-mapping>

hibernate.cfg.xml

<!-- hibernate\_mapping.one\_to\_one\_xml -->

<mapping resource=*"hibernate\_mapping/one\_to\_one\_xml/employee7.hbm.xml"*/>

<mapping resource=*"hibernate\_mapping/one\_to\_one\_xml/address7.hbm.xml"*/>

StoreData15.java

**public** **class** StoreData15 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Employee7 e1 = **new** Employee7();

e1.setName("Ravi Malik");

e1.setEmail("ravi@gmail.com");

Address7 address1 = **new** Address7();

address1.setAddressLine1("G-21,Lohia nagar");

address1.setCity("Ghaziabad");

address1.setState("UP");

address1.setCountry("India");

address1.setPincode(201301);

e1.setAddress(address1);

address1.setEmployee(e1);

session.persist(e1);

t.commit();

session.close();

System.***out***.println("success"); }}

FetchData15.java

**public** **class** FetchData15 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

TypedQuery query = session.createQuery("from Employee7 e");

List<Employee7> list = query.getResultList();

Iterator<Employee7> itr = list.iterator();

**while** (itr.hasNext()) {

Employee7 emp = itr.next();

System.***out***.println(emp.getEmployeeId() + " " + emp.getName() + " " + emp.getEmail());

Address7 address = emp.getAddress();

System.***out***.println(address.getAddressLine1() + " " + address.getCity() + " " + address.getState() + " "

+ address.getCountry() + " " + address.getPincode());}

session.close();

System.***out***.println("success");}}

Hibernate One to One Example 1Hibernate One to One Example 2

10. Hibernate One to One Example using Annotation

theory ကေတာ့ title 9 အတိုင္းပဲ ၿဖစ္ပါတယ္။

Employee8.java

@Entity

@Table(name = "employee\_8")

**public** **class** Employee8 {

@Id

@GeneratedValue(strategy = GenerationType.***AUTO***)

@PrimaryKeyJoinColumn

**private** **int** employeeId;

**private** String name, email;

@OneToOne(targetEntity = Address8.**class**, cascade = CascadeType.***ALL***)

**private** Address8 address;

Address8.java

@Entity

@Table(name = "address\_8")

**public** **class** Address8 {

@Id

@GeneratedValue(strategy = GenerationType.***AUTO***)

**private** **int** addressId;

**private** String addressLine1, city, state, country;

**private** **int** pincode;

@OneToOne(targetEntity = Employee8.**class**)

**private** Employee8 employee;

Hibernate.cfg.xml

<!-- hibernate\_mapping.one\_to\_one\_annotation -->

<mapping class=*"hibernate\_mapping.one\_to\_one\_annotation.Employee8"*/>

<mapping class=*"hibernate\_mapping.one\_to\_one\_annotation.Address8"*/>

StoreData16.java

**public** **class** StoreData16 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Employee8 e1 = **new** Employee8();

e1.setName("Ravi Malik");

e1.setEmail("ravi@gmail.com");

Address8 address1 = **new** Address8();

address1.setAddressLine1("G-21,Lohia nagar");

address1.setCity("Ghaziabad");

address1.setState("UP");

address1.setCountry("India");

address1.setPincode(201301);

e1.setAddress(address1);

address1.setEmployee(e1);

session.persist(e1);

t.commit();

session.close();

System.***out***.println("success");}}

FetchData16.java

**public** **class** FetchData16 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

TypedQuery query = session.createQuery("from Employee8");

List<Employee8> list = query.getResultList();

Iterator<Employee8> itr = list.iterator();

**while** (itr.hasNext()) {

Employee8 emp = itr.next();

System.***out***.println(emp.getEmployeeId() + " " + emp.getName() + " " + emp.getEmail());

Address8 address = emp.getAddress();

System.***out***.println(address.getAddressLine1() + " " + address.getCity() + " " + address.getState() + " "

+ address.getCountry() + " " + address.getPincode());}

session.close();

System.***out***.println("success");}}

Hibernate One to One Example using Annotation 1Hibernate One to One Example using Annotation 2

11. Hibernate Many to One Mapping using XML

employee တိုင္းမွာ address တစ္ခုပဲ ရွိၿပီးေတာ့ address တစ္ခုမွာေတာ့ employee အမ်ားၾကီး ရွိပါတယ္။

Employee9.java

**public** **class** Employee9 {

**private** **int** employeeId;

**private** String name, email;

**private** Address9 address;

Address9.java

**public** **class** Address9 {

**private** **int** addressId;

**private** String addressLine1, city, state, country;

**private** **int** pincode;

**private** Employee9 employee;

employee9.hbm.xml

<hibernate-mapping>

<class name=*"hibernate\_mapping.many\_to\_one\_xml.Employee9"* table=*"employee\_9"*>

<id name=*"employeeId"*>

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

</id>

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

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

<many-to-one name=*"address"* cascade=*"all"*></many-to-one>

</class>

</hibernate-mapping>

address9.hbm.xml

<hibernate-mapping>

<class name=*"hibernate\_mapping.many\_to\_one\_xml.Address9"* table=*"address\_9"*>

<id name=*"addressId"*>

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

</id>

<property name=*"addressLine1"*></property>

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

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

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

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

</class>

</hibernate-mapping>

StoreData17.java

**public** **class** StoreData17 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Employee9 e1 = **new** Employee9();

e1.setName("Ravi Malik");

e1.setEmail("ravi@gmail.com");

Employee9 e2 = **new** Employee9();

e2.setName("Anuj Verma");

e2.setEmail("anuj@gmail.com");

Address9 address1 = **new** Address9();

address1.setAddressLine1("G-13,Sector 3");

address1.setCity("Noida");

address1.setState("UP");

address1.setCountry("India");

address1.setPincode(201301);

e1.setAddress(address1);

e2.setAddress(address1);

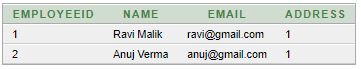
session.persist(e1);

session.persist(e2);

t.commit();

session.close();

System.***out***.println("success");}}

Hibernate Many to One Example 2

FetchData17.java

**public** **class** FetchData17 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

TypedQuery query = session.createQuery("from Employee9 e");

List<Employee9> list = query.getResultList();

Iterator<Employee9> itr = list.iterator();

**while** (itr.hasNext()) {

Employee9 emp = itr.next();

System.***out***.println(emp.getEmployeeId() + " " + emp.getName() + " " + emp.getEmail());

Address9 address = emp.getAddress();

System.***out***.println(address.getAddressLine1() + " " + address.getCity() + " " + address.getState() + " "+ address.getCountry() + " " + address.getPincode());}

session.close();

System.***out***.println("success");}}

12. Hibernate Many to One Mapping using Annotation

Employee10.java

@Entity

@Table(name = "employee\_10")

**public** **class** Employee10 {

@Id

@GeneratedValue(strategy = GenerationType.***AUTO***)

**private** **int** employeeId;

**private** String name, email;

@ManyToOne(cascade = CascadeType.***ALL***)

**private** Address10 address;

Address10.java

@Entity

@Table(name = "address\_10")

**public** **class** Address10 {

@Id

@GeneratedValue(strategy = GenerationType.***AUTO***)

**private** **int** addressId;

**private** String addressLine1, city, state, country;

**private** **int** pincode;

@OneToOne(cascade = CascadeType.***ALL***)

**private** Employee10 employee;

StoreData18.java

**public** **class** StoreData18 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Employee10 e1 = **new** Employee10();

e1.setName("Ravi Malik");

e1.setEmail("ravi@gmail.com");

Employee10 e2 = **new** Employee10();

e2.setName("Anuj Verma");

e2.setEmail("anuj@gmail.com");

Address10 address1 = **new** Address10();

address1.setAddressLine1("G-13,Sector 3");

address1.setCity("Noida");

address1.setState("UP");

address1.setCountry("India");

address1.setPincode(201301);

e1.setAddress(address1);

e2.setAddress(address1);

session.persist(e1);

session.persist(e2);

t.commit();

session.close();

System.***out***.println("success");}}

FetchData18.java

**public** **class** FetchData18 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

TypedQuery query = session.createQuery("from Employee10 e");

List<Employee10> list = query.getResultList();

Iterator<Employee10> itr = list.iterator();

**while** (itr.hasNext()) {

Employee10 emp = itr.next();

System.***out***.println(emp.getEmployeeId() + " " + emp.getName() + " " + emp.getEmail());

Address10 address = emp.getAddress();

System.***out***.println(address.getAddressLine1() + " " + address.getCity() + " " + address.getState() + " " + address.getCountry() + " " + address.getPincode());}

session.close();

System.***out***.println("success");}}

Hibernate Many to One Example using Annotation 2

13. Bidirectional Association

Bidirectional Association ဆိုတာက ႏွစ္ဖက္စလံုးမွာ ရွိတဲ့ dependent object မ်ားရဲ႕ detail ကို fetch လုပ္ႏုိင္တဲ့ process တစ္ခုပဲၿဖစ္ပါတယ္။ ဆိုလိုတာက ႏွစ္ဖက္စလံုးမွာ reference ေတြ ရွိၾကတယ္။ Address မွာလည္း Employee reference ရွိၿပီးေတာ့ Employee မွာလည္း Address reference ရွိမယ္ဆိုရင္ one-to-one သို႔မဟုတ္ one-to-many relationship ကိုအသံုးၿပဳႏိုင္ပါတယ္။

14. Lazy Collection

performance ေကာင္းေအာင္ သံုးၿခင္းၿဖစ္ပါတယ္။ ဘာလို႔လဲဆိုေတာ့ သူက child objects မ်ားကို demand အရသာ load ဆြဲတင္တဲ့ အတြက္ေၾကာင့္ ၿဖစ္တယ္။ အဲ့ lazy collecton ဆိုတာက Hibernate 3.0 မွာကတည္းက ရေနၿပီ။ lazy collection ကို သံုးခ်င္တယ္ဆိုလို႔ရွိရင္ collection ထဲမွာ lazy="true" ေပးလိုက္ရံုပဲ။ ဒါေပမယ့္ default က အဲ့အတိုင္းၿဖစ္ေနတဲ့အတြက္ ေရးစရာေတာင္မလိုဘူး auto lazy collection ကို အလုပ္လုပ္လိမ့္မယ္။ false လို႔သြားေပးလိုက္မယ္ဆိုရင္ေတာ့ init မွာ child objects အကုန္လံုးကို load သြားဆြဲေနမွာ ၿဖစ္တဲ့အတြက္ေၾကာင့္ performance က်သြားႏိုင္ပါတယ္။

**<list** name="answers" lazy="true"**>**

**<key** column="qid"**></key>**

**<index** column="type"**></index>**

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

**</list>**

15. Component Mapping

ကၽြန္ေတာ္တို႔က dependent object ကို component တစ္ခုအၿဖစ္ map လုပ္ၿခင္းကို component mapping လုပ္တယ္လို႔ ေခၚႏိုင္ပါတယ္။ component တစ္ခုဆိုတာက object တစ္ခုပဲ ဒါေပမယ့္သူ႕ကို entity reference တစ္ခုမဟုတ္ပဲ value တစ္ခုအၿဖစ္ သိ္္မ္းဆည္းထားၿခင္း ၿဖစ္တယ္။ သူ႕ကို dependent object မွာ primary key မရွိဘူးဆိုတဲ့ အေၿခအေနမွာ အဓိကအသံုးၿပဳပါတယ္။

Employee11.java

**public** **class** Employee11 {

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

**private** String name;

**private** Address11 address;// HAS-A

**public** Employee11(String name, Address11 address) {

**this**.name = name;

**this**.address = address;}

Address11.java

**public** **class** Address11 {

**private** String city, country;

**private** **int** pincode;

**public** Address11() {}

**public** Address11(String city, String country, **int** pincode) {

**super**();

**this**.city = city;

**this**.country = country;

**this**.pincode = pincode;}

employee11.hbm.xml

<hibernate-mapping>

<class name=*"hibernate\_mapping.component\_mapping.Employee11"* table=*"employee\_11"*>

<id name=*"id"*>

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

</id>

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

<component name=*"address"* class=*"hibernate\_mapping.component\_mapping.Address11"*>

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

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

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

</component>

</class>

</hibernate-mapping>

StoreData19.java

**public** **class** StoreData19 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

Employee11 e1=**new** Employee11("amit",**new** Address11("gzb","india",221233));

Employee11 e2=**new** Employee11("Vijay",**new** Address11("noida","india",224123));

session.persist(e1);

session.persist(e2);

t.commit();

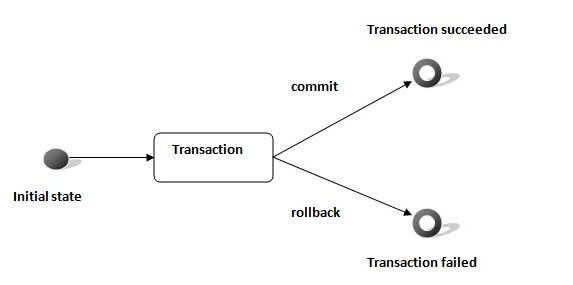
session.close();

System.***out***.println("success"); }}

**TX Management**

1. Transaction Management

Transacton တစ္ခုဆိုတာက unit of work တစ္ခုပဲၿဖစ္တယ္။ ဆိုလိုတာက step တစ္ခုမွာ fail ၿဖစ္တယ္ဆိုရင္ transaction တစ္ခုလံုး fail ၿဖစ္တယ္လို႔ ဆိုလိုၿခင္းၿဖစ္တယ္ (အဲ့တာကို atomicity လို႔ေခၚႏိုင္ပါတယ္)။ transaction တစ္ခုကို ACID properties မ်ားၿဖစ္အၿဖစ္ ေဖာ္ၿပႏိုင္တယ္ (Atomicity, Consistency, Isolation and Durability)။



Transaction Interface in Hibernate

hibernate framework ထဲမွာ transaction interface တစ္ခုၿဖစ္တယ္။ transaction implementation (JTA,JDBC) မွ abstraction ကို transaction interface က maintain လုပ္ပါတယ္။

transaction တစ္ခုဆိုတာ session နဲ႔ associated ၿဖစ္ေနၿပီး session.beginTransaction() ကို call လုပ္ၿခင္းအားၿဖင့္ instantiated လုပ္ပါတယ္။

Transaction interface ရဲ႕ method မ်ားကေတာ့…….

1. void begin() – transaction အသစ္တစ္ခုကို start လုပ္ခ်င္တဲ့အခါ
2. void commit() - ကၽြန္ေတာ္တို႔က FlushMode.NEVER လုပ္ခဲ့မယ္ဆိုရင္ေတာင္ ends the unit of work ၿဖစ္ပါတယ္။
3. void rollback() - အဲ့ transaction ကို rollback လုပ္ဖို႔ရန္ force လုပ္ရန္ သံုး
4. void setTimeout(int seconds) – မည္သည့္ transaction ကိုမဆို start လုပ္ရန္ transaction timeout ကို ထည့္ရန္အသံုးၿပဳ။
5. boolean isAlive() – transaction က alive ၿဖစ္ေသးလား မၿဖစ္ဘူးဆိုတာကို စစ္ေဆးဖုိ႔အတြက္ ၿဖစ္တယ္။
6. void registerSynchronization(Synchronization s) – transaction အတြက္ user synchronization callback တစ္ခုကို register လုပ္ရန္အတြက္ ၿဖစ္တယ္။
7. boolean wasCommited() – transaction တစ္ခုကို commit လုပ္တာ success ၿဖစ္မၿဖစ္ check ဖို႔အတြက္
8. boolean wasRolledBack() – transaction တစ္ခုကို rolledback လုပ္တာ success ၿဖစ္မၿဖစ္ check ဖို႔အတြက္

Example of Transaction Management in Hibernate

hibernate ထဲမွာ exception တစ္ခုခုေတြ႕ခဲ့ရင္ transaction ကို rollback လုပ္တာ ပိုေကာင္းပါတယ္။

Session session = **null**;

Transaction tx = **null**;

**try** {

session = sessionFactory.openSession();

tx = session.beginTransaction();

//some action

tx.commit();

}**catch** (Exception ex) {

ex.printStackTrace();

tx.rollback();

}

**finally** {session.close();}

**HQL**

1. Hibernate Query Language (HQL)

Hibernate Query Language (HQL) ဆိုတာက SQL (Structured Query Language) နဲ႔အတူတူပဲ ဒါေပမယ့္ သူက database ရဲ႕ table ေပၚမွာ depends မၿဖစ္ပါဘူး။ သူ႕မွာ table name အစား class name ကို အသံုးၿပဳပါတယ္။ အဲ့တာေၾကာင့္ hql ကို database independent query language လို႔ေခၚပါတယ္။

HQL ရဲ႕ advantage

HQL ရဲ႕ advantage ေတြ အမ်ားၾကီး ရွိပါတယ္။

1. database independent
2. polymorphic queries ကို support ေပးပါတယ္။
3. Java Programmer တစ္ေယာက္အတြက္ learn လုပ္ရတာ လြယ္ကူပါတယ္။

Query Interface

Query Interface ဆိုတာက Hibernate Query ရဲ႕ object oriented representation တစ္ခုၿဖစ္တယ္။ createQuery() method Session interface ကို call ၿခင္းအားၿဖင့္ query ရဲ႕ object ကပါ၀င္လာႏိုင္ပါတယ္။

query interface က method မ်ားစြာကို provide လုပ္ပါတယ္။အဲ့ထဲကမွာ အမ်ားသံုးတဲ့ method မ်ားကေတာ့…..

1. public int executeUpdate() - update သို႔မဟုတ္ delete query ကို execute လုပ္ဖို႔အတြက္ အသံုးၿပဳပါတယ္။
2. public List list() – relation ရဲ႕ result ကို list တစ္ခုအေနနဲ႔ return ၿပန္ေပးပါတယ္။
3. public Query setFirstResult(int rowno) – ဘယ္ row ကေန စၿပီးေတာ့ ထုတ္မလဲဆိုတာကို သတ္မွတ္ေပးၿခင္း ၿဖစ္တယ္။
4. public Query setMaxResult(int rowno) - ဘယ္ကေန ဘယ္အထိထုတ္မလဲဆိုတာမွာ ဘယ္ထိလဲဆိုတာကို သတ္မွတ္ဖို႔အတြက္ၿဖစ္ပါတယ္။
5. public Query setParameter(int position, Object value) - JDBC style query parameter ထည့္ေပးဖို႔အတြက္ ၿဖစ္ပါတယ္။
6. public Query setParameter(String name, Object value) - named query parameter တစ္ခုကို ထည့္ေပးဖို႔အတြက္ ၿဖစ္ပါတယ္။

Example of HQL to get all the records

Query query=session.createQuery("from Emp");//here persistent class name is Emp

List list=query.list();

Example of HQL to get records with pagination

Query query=session.createQuery("from Emp");

query.setFirstResult(5);

query.setMaxResult(10);

List list=query.list();//will return the records from 5 to 10th number

Example of HQL update query

Transaction tx=session.beginTransaction();

Query q=session.createQuery("update User set name=:n where id=:i");

q.setParameter("n","Udit Kumar");

q.setParameter("i",111);

int status=q.executeUpdate();

System.out.println(status);

tx.commit();

Example of HQL delete query

Query query=session.createQuery("delete from Emp where id=100");

//specifying class name (Emp) not tablename

query.executeUpdate();

HQL with Aggregate functions

HQL မွာ avg(), min(), max() အစရွိတဲ့ aggregate functions မ်ားကို ေခၚသံုးလို႔ရပါတယ္။

Example to get total salary of all the employees

Query q=session.createQuery("select sum(salary) from Emp");

List<Integer> list=q.list();

System.out.println(list.get(0));

Example to get maximum salary of employee

Query q=session.createQuery("select max(salary) from Emp");

Example to get minimum salary of employee

Query q=session.createQuery("select min(salary) from Emp");

Example to count total number of employee ID

Query q=session.createQuery("select count(id) from Emp");

Example to get average salary of each employees

Query q=session.createQuery("select avg(salary) from Emp");

**HCQL**

HCQL (Hibernate Criteria Query Language)

တိတိက်က်သတ္မွတ္ထားတဲ့ criteria ေပၚမူတည္ၿပီး data မ်ားကို ဆြဲထုတ္တာကို HCQL လို႔ေခၚပါတယ္။ Criteria interface က criteria မ်ားကို apply လုပ္ရန္ methods မ်ားကို provide လုပ္ပါတယ္။ ဘယ္လိုမ်ိဳးလဲဆိုေတာ့ ဘယ္သူက salary 50000 ထက္ပိုရေနသလဲဆိုတဲ့ criteria မ်ိဳးေတြ ၿဖစ္ပါတယ္။

HCQL ရဲ႕ advantage

HCQL က criteria မ်ားကို ထည့္ရန္ methods မ်ားကို provide လုပ္ေပးပါတယ္။ အဲ့တာေၾကာင့္ query တစ္ခုတည္းမွာ criteria အမ်ားၾကီးကို ေပါင္းထည့္ႏိုင္ပါတယ္။

Criteria Interface

criteria interface က criteria မ်ားကို specify လုပ္ဖို႔ရန္ method မ်ားစြာကို provide လုပ္ေပးပါတယ္။ Session interface ရဲ႕ createCriteria() method ကို call လုပ္ၿခင္းအားၿဖင့္ Criteria ရဲ႕ object ကို ရရွိႏိုင္ပါတယ္။

Syntax of createCriteria() method of Session interface

public Criteria createCriteria(Class c)

အသံုးမ်ားတဲ့ criteria interface ရဲ႕ method မ်ားကေတာ့….

1. public Criteria add(Criterion c) – restrictions မ်ားကို add လုပ္ရန္ သံုးပါတယ္။
2. public Criteria addOrder(Order o) – ordering မ်ားကို specifies လုပ္ဖို႔ရန္ သံုးပါတယ္။
3. public Criteria setFirstResult(int firstResult) – record ရဲ႕ first number ကို retrieved လုပ္ဖို႔ရန္ specifies လုပ္ေပးပါတယ္။
4. public Criteria setMaxResult(int totalResult) - record ရဲ႕ total number ကို retrieved လုပ္ဖို႔ရန္ specifies လုပ္ေပးပါတယ္။
5. public List list() – object ပါ၀င္တဲ့ list ကို return ၿပန္ေပးပါတယ္။
6. public Criteria setProjection(Projection projection) – projection ကို specifies လုပ္ေပးပါတယ္။

Restrictions class

Restrictions class က methods မ်ားစြာကို provide လုပ္ေပးပါတယ္ သူက Criterion အၿဖစ္ အသံုးၿပဳပါတယ္။ အသံုးမ်ားတဲ့ Restrictions class ရဲ႕ method မ်ားကေတာ့….

1. public static SimpleExpression lt(String propertyName,Object value) sets the less than constraint to the given property.
2. public static SimpleExpression le(String propertyName,Object value) sets the less than or equal constraint to the given property.
3. public static SimpleExpression gt(String propertyName,Object value) sets the greater than constraint to the given property.
4. public static SimpleExpression ge(String propertyName,Object value) sets the greater than or equal than constraint to the given property.
5. public static SimpleExpression ne(String propertyName,Object value) sets the not equal constraint to the given property.
6. public static SimpleExpression eq(String propertyName,Object value) sets the equal constraint to the given property.
7. public static Criterion between(String propertyName, Object low, Object high) sets the between constraint.
8. public static SimpleExpression like(String propertyName, Object value) sets the like constraint to the given property.

Order Class

order class က order ကို represent လုပ္ေပးပါတယ္။ order class ရဲ႕ အသံုးမ်ားတဲ့ method မ်ားကေတာ့…..

1. public static Order asc(String propertyName) applies the ascending order on the basis of given property.
2. public static Order desc(String propertyName) applies the descending order on the basis of given property.

Example of HCQL to get all the records

Crietria c=session.createCriteria(Emp.class);//passing Class class argument

List list=c.list();

Example of HCQL to get the 10th to 20th record

Crietria c=session.createCriteria(Emp.class);

c.setFirstResult(10);

c.setMaxResult(20);

List list=c.list();

Example of HCQL to get the records whose salary is greater than 10000

Crietria c=session.createCriteria(Emp.class);

c.add(Restrictions.gt("salary",10000));//salary is the propertyname

List list=c.list();

Example of HCQL to get the records in ascending order on the basis of salary

Crietria c=session.createCriteria(Emp.class);

c.addOrder(Order.asc("salary"));

List list=c.list();

HCQL with Projection

ကၽြန္ေတာ္တို႔က name ကဲ့သို႔ေသာ projection အားၿဖင့္ particular column တစ္ခုရဲ႕ data ကို fetch လုပ္ႏိုင္ပါတယ္။

Criteria c=session.createCriteria(Emp.class);

c.setProjection(Projections.property("name"));

List list=c.list();

**Named Query**

Hibernate Named Query

Hibernate Named Query ဆိုတာက meaningful name အားၿဖင့္ မည္သည့္ query ကိုမဆို အသံုးၿပဳႏိုင္တဲ့ နည္းလမ္းတစ္ခုၿဖစ္ပါတယ္။ alias names ကို အသံုးၿပဳၿခင္း သြားဆင္ေနတယ္။ Hibernate framework က named queries ရဲ႕ concept ကို provide လုပ္ပါတယ္ အဲ့ေတာ့ java programmer တစ္ေယာက္အေနနဲ႔ scatter queries မ်ားမလိုအပ္ေတာ့ပါ။

hibernate မွာ named query ကို နည္းလမ္း ၂ မ်ိဳးနဲ႔ အသံုးၿပဳႏိုင္ပါတယ္။

1. by annotation
2. by mapping file

Hibernate Named Query by annotation

ကၽြန္ေတာ္တို႔အေနနဲ႔ hibernate မွာ named query အသံုးၿပဳမယ္ဆိုလို႔ရွိရင္ @NamedQueries နဲ႔ @NamedQuery annotations မ်ားရဲ႕ knowledge မ်ားကို သိရွိထားဖို႔ လိုအပ္ပါတယ္။

@NameQueries annotation ကို multiple named queries မ်ား သတ္မွတ္ဖို႔ရန္ အသံုးၿပဳပါတယ္။

@NameQuery annotation ကို single named query မ်ား သတ္မွတ္ဖို႔ရန္ အသံုးၿပဳပါတယ္။

@NamedQueries(

{

@NamedQuery(

name = "findEmployeeByName",

query = "from Employee e where e.name = :name"

)

}

)

Example of Hibernate Named Query by annotation

persistence class ထဲမွာ named query ကို define လုပ္ဖို႔ရန္ annotation မ်ားကို အသံုးၿပဳပါမယ္။ ဒီ example မွာဆိုရင္ file ၃ file ပါ၀င္ပါမယ္။ 1.EmployeeNameQuery.java 2.FetchData20.java 3.hibernate.cfg.xml

table name က employee\_named\_query ၿဖစ္ၿပီးေတာ့ 4 columns ပါ၀င္ပါမယ္(id, name, job and salary).

EmployeeNameQuery.java

@NamedQueries({ @NamedQuery(name = "findEmployeeByName", query = "from EmployeeNameQuery e where e.name = :name") })

@Entity

@Table(name = "employee\_named\_query")

public class EmployeeNameQuery {

@Id

int id;

String name;

int salary;

String job;

@GeneratedValue(strategy = GenerationType.AUTO)

FetchData20.java

**public** **class** FetchData20 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

// Hibernate Named Query

TypedQuery query = session.getNamedQuery("findEmployeeByName");

query.setParameter("name", "amit");

List<EmployeeNameQuery> employees = query.getResultList();

Iterator<EmployeeNameQuery> itr = employees.iterator();

**while** (itr.hasNext()) {

EmployeeNameQuery e = itr.next();

System.***out***.println(e);}

session.close();}}

hibernate.cfg.xml

<!-- named\_query.name\_query\_annotation -->

<mapping class=*"named\_query.name\_query\_annotation.EmployeeNameQuery"*/>

Hibernate Named Query by mapping file

named query သတ္မွတ္ဖို႔ရန္ hibernate-mapping ရဲ႕ query element မ်ားကို အသံုးၿပဳရန္ လိုအပ္ပါတယ္။ hbm file ထဲမွာ name query ကို ေရးမွာ ၿဖစ္တယ္။

employee\_name\_query\_mapping\_file.hbm.xml

<hibernate-mapping>

<class name="named\_query.name\_query\_mapping\_file.EmployeeNameQueryMappingFile" table="employee\_name\_query\_mapping\_file">

<id name="id">

<generator class="native"></generator>

</id>

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

<property name="job"></property>

<property name="salary"></property>

</class>

<query name="findEmployeeByNameMappingFile">

<![CDATA[from EmployeeNameQueryMappingFile e where e.name = :name]]>

</query>

</hibernate-mapping>

EmployeeNameQueryMappingFile.java

public class EmployeeNameQueryMappingFile {

int id;

String name;

int salary;

String job;

FetchData21.java

**public** **class** FetchData21 {

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

StandardServiceRegistry ssr = **new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta = **new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();

Session session = factory.openSession();

// Hibernate Named Query

TypedQuery query = session.getNamedQuery("findEmployeeByNameMappingFile");

query.setParameter("name", "amit");

List<EmployeeNameQuery> employees = query.getResultList();

Iterator<EmployeeNameQuery> itr = employees.iterator();

**while** (itr.hasNext()) {

EmployeeNameQuery e = itr.next();

System.***out***.println(e);}

session.close();}}

**Hibernate Caching**

Caching in Hibernate

Hibernate caching က cache ထဲမွာ object ကို polling လုပ္ၿခင္းအားၿဖင့္ application ရဲ႕ performance က improve ၿဖစ္လာပါလိမ့္မယ္။ multiple times မွာ same data ကို fetch လုပ္ရင္ မ်ားစြာ အသံုး၀င္ပါတယ္။

cashing က အဓိကအားၿဖင့္ ႏွစ္မ်ိဳး ရွိတယ္။

1. First Level Cache, နဲ႔
2. Second Level Cache

First Level Cache

Session object က first level cache data ကို hold လုပ္ထားပါတယ္။ အဲ့တာက default ၿဖစ္ေနတာ။ first level cache data က entire application မွာ available မၿဖစ္ပါဘူး။ application တစ္ခုဟာ session object မ်ားစြာကို အသံုးၿပဳႏိုင္ပါတယ္။

Second Level Cache

SessionFactory object က second level cache data ကို hold လုပ္ထားပါတယ္။ second level cache data ကေတာ့ entire application မွာ available ၿဖစ္ပါတယ္။ ဒါေပမယ့္ အဲ့တာကို enable လုပ္ထားဖို႔ လိုအပ္ပါလိမ့္မယ္။

1. EH (Easy Hibernate) Cache
2. Swarm Cache
3. OS Cache
4. JBoss Cache

**Second Level Cache**

Hibernate second level cache က session factory တစ္ခုရဲ႕ session object အကုန္လံုးအတြက္ common cache တစ္ခုကို အသံုးၿပဳပါတယ္။ session factory တစ္ခုမွ multiple session objects မ်ားရွိခဲ့မယ္ဆိုရင္ အဲ့တာက useful ၿဖစ္ပါတယ္။

SessionFactory က second level cache data ကို hold လုပ္ထားပါတယ္။ session objects အားလံုးအတြက္ SessionFactory က global ၿဖစ္ပါတယ္။ default အားၿဖင့္ not enable ၿဖစ္ပါတယ္။

မတူညီတဲ့ vendors မ်ားက Second Level Cache ရဲ႕ implementation ကို provide လုပ္ပါတယ္။

1. EH Cache
2. OS Cache
3. Swarm Cache
4. JBoss Cache

implementation တစ္ခုခ်င္းစီတိုင္းက မတူညီတဲ့ cache usage functionality ကို provide လုပ္ပါတယ္။ second level cache ကို အသံုးၿပဳဖို႔ရန္ နည္းလမ္း ၄ ခု ရွိပါတယ္။

1. read-only: operation တစ္ခုတည္းကိုပဲ read ဖို႔အတြက္ cachsing က အလုပ္လုပ္လိမ့္မယ္။
2. nonstrict-read-write: cashing ကို read နဲ႔ write ဖို႔အတြက္ အလုပ္လုပ္လိမ့္မယ္ ဒါေပမယ့္ တစ္ၾကိမ္တည္း လုပ္တယ္။
3. read-write: caching က read and write အတြက္ အလုပ္လုပ္တယ္။
4. transactional: caching က transaction အတြက္ အလုပ္လုပ္တယ္။

hbm.xml file ထဲမွာ class or collection level ကို apply လုပ္ဖို႔ရန္အတြက္ cache-usage property ကို သံုးပါတယ္။

<cache usage="read-only" />

Hibernate Second Level Cache Example