**Hibernate**

**Object Relational Mapping (ORM):** ORM is a theoretical approach whichprovides a concept of converting object to DB record and vice-version.

If follows are concept i.e., mapping it will be done in between java class and Database table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Java Class |  |  |  | DB Table |
| (Model Class) |  | Mapping (XML) |  |  |
|  |  |  |  |  |

**Framework:** Framework is a technologies and API’s. Which provide a new concept by simply existing for a better programming is development. Frameworks are mainly designed for to reduce burden on programmer and faster development of application.

**HIBERNATE:** Hibernate is a ORM Framework. It follows java API’s like JDBC,JNDI, JTA.

**JDBC:** Java Database Connectivity.

**JNDI:** Java Naming Directory Interface.

**JTA:** Java Transaction API.

And also uses XML (non-Java Technology).

* Hibernate provides In-built Persistency API to perform Single Row Operations with out SQL by programmer. Like insert (Save(), Delete(), Update()), read (get() or load()).
* Hibernate supports Transaction Management begin Transaction, Commit, rollback transaction etc...(in-built).
* Hibernate supports Connection Pooling (Multiple Connection are Maintained as a memory for re-using).
* Hibernate provides HQL (Hibernate Query Language) which is DB Independent.
* Dialect is a class/concept that generates SQL queries for Databases. Ex: Oracle Dialect, MYSQL Dialect etc….
* Hibernate Supports collection Mapping and using, like List, Set, Map etc…
* Hibernate provides combinations of designing of DB tables for Inheritance and Associations.
* Hibernate provides Caching for Reducing DB Network calls between program to Database.

**Model Class:**

**Syntax:**

private <DT> <V>

Setter Method

Public void set<V> (<DT> <V>)

{

**Here DT: Data Type**

**V: Variable**

this.<V>=<V>;

**Syntax:**

}

Getter Method

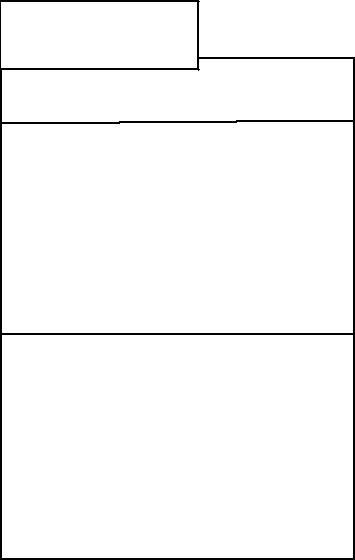
Public <DT> get<V> ()

{

Return <V>;

}

**Example:**



Com.app

+Ticket

-ticId : Int

-show: String

-Cost : double

// Setters and Getters

Methods ()

**package** com.app;

**public class** Ticket {

**private int** ticId;

**private** Stringshow;

**private double** cost;

**public int** getTicId() {

**return** ticId;

}

**public void** setTicId(**int** ticId) {

**this**.ticId=ticId;

}

**public** String getShow() {

**return** show;

}

**public void** setShow(Stringshow) {

**this**.show=show;

}

**public double** getCost() {

**return** cost;

}

**public void** setCost(**double** cost) {

**this**.cost=cost;

}

@Override

**public** String toString() {

**return** "Ticket [ticId="+ticId+", show="+show+", cost="+

cost

* "]";

}

}

**Example:**

Package com.one

Class +Details

Variables

-dtlId: int

-name: String

* val : double
* data: List<String>
* addrObj : Address

**package** com.app;

**public class** Details {

**private int** dtlId;

**private** Stringname;

**private double** val;

**private** List<String>data;

**private** AddressaddrObj;

**public int** getDtlId() {

**return** dtlId;

}

**public void** setDtlId(**int** dtlId) {

**this**.dtlId=dtlId;

}

**public** StringgetName() {

**return** name;

}

**public void** setName(Stringname) {

**this**.name=name;

}

**public double** getVal() {

**return** val;

}

**public void** setVal(**double** val) {

**this**.val=val;

}

**public** List<String> getData() {

**return** data;

}

**public void** setData(List<String>data) { **this**.data=data;

}

**public** AddressgetAddrObj() {

**return** addrObj;

}

**public void** setAddrObj(AddressaddrObj) { **this**.addrObj=addrObj;

}

@Override

**public** String toString() {

**return** "Details [dtlId="+dtlId+", name="+name+", val="+

val

* "]";

}

}

**Hibernate Mapping File:** To bind Java Class with database table, one XML file(with Format<file-name>.hbm.xml) is required to specify the mapping details.

**Example:** employee.hbm.xml

This file mapp’s Employee class with empTab.

For this equailant xml mapping code is

<!DOCTYPE hibernate-mapping PUBLIC

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

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

<hibernate-mapping>

<class name="com.app.Empoyee" table=" ">

<id name="empNo" column=""/>

<property name="empName" length="20"/>

</class>

</hibernate-mapping>

**Note:** Here <id> indicates primary key of table to be binded. And <property>indicates field – column binding.

**Hibernate Configuration:** To provide details like connection properties,hibernate properties, database level properties, we need to specify. Key value pairs as Hibernate configuration (properties) file.

**File name follows naming conversion:** hibernate.cfg.xml.

**hibernate.cfg.xml:** To specify the configuration of the database hibernate.

Database details-- driver\_class, url, username, password, dialect, show\_sql, hbm2ddl.autos.

**Basic Connection Properties Are:**

1. **Driver\_class:** To specify in configuration file to unique to provide asa property with key as connection.driver\_class and the value should be driver\_class.

**Example:** <propertyname=”connection.driver\_class”>com.mysql.jdbc.Driver </property>

1. **URL:** It is a location of Resource. Which provides combination of URL=IP+ PORTNUMBER + RESOURCENAME.

**Example:** http://198.126.0.1:8089/TestService

Jdbc.Oracle:@198.126.0.1:1521:XE.

Here http: and Jdbc, oracles are protocols.

**Note:** To specify this key is connection.url.

**Example:** <property

name=”connection.url”>jdbc:mysql://localhost:3306/test

</property>

1. **Username and Password:** These are known as resource accessauthentication details.

To specify these keys are: Connection. username, Connection. password.

1. **Dialect:** It is classes defined by hibernate that generates SQL Queriesto be executed at DB level based on Hibernate programs and HQL. **Example:** org.hibernate.dialect.MySQLDialect.

org.hibernate.dialect.OracleDialect.

1. **Show\_sql:** It display the generated query by dialect, by default thisfeature is turned off (false), to activate provide value as true.
2. **Hbm2ddl.auto:** From hibernate mapping file a DB table create/modification/validation will be performed.

**Possible values:**

* 1. create
  2. create-drop
  3. update
  4. validate

Configuration object and loading cfg file writing hibernate main program.

To writing hibernate that perform basic

Save, update, and delete, sets.

Step by step: configuration object

sesionFactory object

session object

transaction object

operation (like save, update….)

commit transaction

close session.

**Writing Hibernate Program to Save Java Object as Record in DB:**

**To write Hibernate code 4 files ( 2 java, 2 xml) are required they are:**

1. Java class that represents Database Table. Example: Account.java, Employee.java.
2. Mapping XML file that shows connectivity of java class and DB table. Example: mapping.hbm.xml, accout.hbm.xml.
3. Configurartion xml file, That provides details like properties and Mapping resourses.

Config = properties + Mapping resources

* + **It must follow a naming rule like: hibernate.cfg.xml.**

1. Main program to perform Specific operation.

Example: save().

**POJO (Plane Old Java Object):** It is a simple class, which is not binded withany technology.

**It follows below rules likes:**

* POJO class must not extend or implements any class or interface (Technology related API like Servlet, Swing, and Struts etc…)
* POJO class must not have any annotation applied on class/ members level. (Technology related @Id, @Controller, @WebServlet etc…)

**Java Bean:**

* Clas must be public (recommended to write package structure and no modifiers like final, abstract etc….)
* Always provide Default / No Arguments Constructors even parameterized constructors also allowed. But these must be public type.
* Writing fields (instance variables are optional, but if defined, they must private type.
* For every field setter and getter methods are required (Alt+Shift+s r).
* hashCode(), equals(), toString() -- All methods class. Java bean class override super class (object class) method like.

toString():String, hashCode():int, equals():Boolean.

**Note:** This is a Optional RULE.

**Shortcuts are:** ALT+Shift+s s  toString()

ALT+Shift+s h  hashCode and equals()

ALT+Shift+s o  defaultContructors()

ALT+Shift+s r  setters and getters()

Ctrl+Shift+ T  Open type (predefined class)

EX: \*STR\*Buff\*.

Ctrl+Shift+ r  Open Resource

EX: \*emp\*dao\*impl\*hibernate.

**Note:** on printing reference variable to console using system.out.println()internally it calls object class toString() method.

Object class toString() return class name and @ Symbol appended with hashcode in hexa-decimal format.

**Mapping File concepts Questions:**

1. <property name=”empName” column=”ename”/>

For column ename default data is null. And for this not-null is false, that means it accepts null data table.

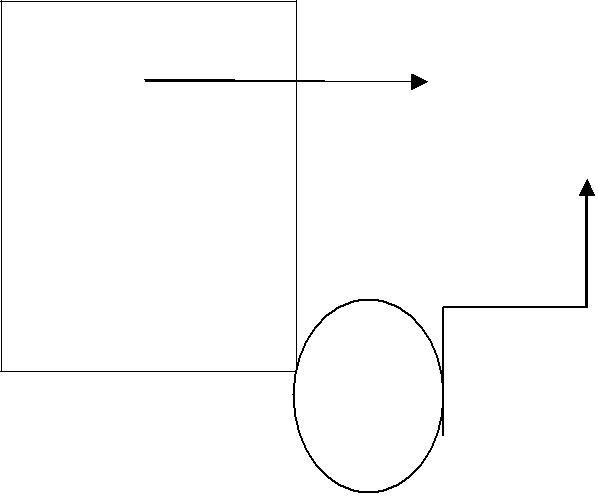
To avoid null data into table provide condition not-null=”true”. <property name=”empName” column=”ename” not-null=”true”/>.

1. If no mapping found between fields and columns table will not read data even through sent by program that means reads default data.

Example: null data

1. Class must have either <id> or <Compositeid> and <property> tag is optional.

In DTD id |compositeid is having no symbol, where property is \*(0-N). Here 0 mean zero, N mean null.



F2x (mapping)

F3x (mapping)

10

A

500

|  |  |  |
| --- | --- | --- |
| C1 | C2 | C3 |
|  |  |  |
| 10 | NULL | NULL |
|  |  |  |

1. If c2, c3 are accepting null then record is inserted.

ii)if one of c2, c3 are not-null then record is rejected.

1. On proving no DB values at Mapping code like No table name and column names, in such case, it takes class details to Constructor validate tables.

<hibernate-mapping package=”com.app”>

<class name=”statusMaster”>

<id name=”statId”/>

<property name=”details”/>

<property name=”costforData”>

</class>

</hibernate-mapping>

<!.... for this table name is status master and column name are

statid, details, costfordata>

Note: In all Data base – SQL query is case insensitive.

Example: Select \* From Employee;

Select \* from employee;

Select \* FROM EMPLOYEE;

Return same data. But data is case sensitive.

In case of MySQL:

Data also case insensitive

Select \* from employee where ename=’ABCD’; Select \* from employee where ename=’abcd’;

Return same data.

**File Locations:** If mapping file is available under a package or folder than,location must be specified as

**<mapping resource=”com/app/status.hbm.xml”/>**

**If Configuration file:**

**Case i:** Having name as hibernate.cfg.xml and location is src.

Then in main code is cfg.configure() is required.

**Case ii:** Having name as details.cfg.xml and location is src.

Then main code is **cfg.configure(“details.cfg.xml”).**

**Case iii:** Having name as details.cfg.xml and location is com/app.

Main code is cfg.configure(“com/app/detail.cfg.xml”).

**Java Bean:** java bean does not support writing of annotation if should not beor implements any class Serializable interface.

**Auto boxing:** A primitive value can be stored in wrapper type; automaticallyvalue is group to be converted into object format.

**Example:** int x=10; // primitive type

Interger s=10; //wrapper to object-auto boxing.

**Note:** Auto boxing available only after jdk 1.5 version.

**Up casting:** A super can store sub class or Implemented object.

**Example:** class A { }

Class B extends A { }

A a1=new B();

**Auto Boxing & Up Casting:**

Serilizable s1=10;

Here 10 is Wrapped to Integer type in

**Step 1:** Integer x=new Integer (10); or x=10;

Serilizable s1= \_\_\_\_\_\_\_\_\_\_\_\_

Then it is assigned to serilizable type as upcasted object in

step2.

**Step 2:** Serilizable s1=new Integer (10);

(or)

Serilizable s1=x;

Up casting and down casting: If two classes are inheritance relation, then internal conversion can be done from subclass to super class to stub.

Example: class A { }

Class B extends A { }

New for statement

B b1=new B(); create memory for itself and super types.

**Save():** This is defined in Session.

**Save (object):** Serilizable this method, takes input as object, which is tobe stored in DB, and returns primary key data in serilizable type. That we can downcast to required type.

**Main:** Employee emp=new Employee();

emp.setEmpId(100);

.

.

Serilizable s1=session.save (emp);

Int id=(integer) s1; //downcasting and auto unboxing

**Update ():** This method updates the object based on primary key value.

**Update (object):** void

**Example:** Employee emp=new Employee();

emp.setEmpId(101);

.

.

Session.update(emp);

**Save or update():** This method will try to check record exit or not bywriting select query.

**Get():** This method defined in session API. It returns DB record to javaobject based on primary key in serilizable format.

**Example:** getmethod takes two parameters, class notation and primary keyvalue.

**Format 1:** get(String,Serilizable): object.

Example: object ob=ses.get(“com.app.Student”,101);

**Format 2:** get(class,Serilizable): object.

Example: object ob=ses.get(“Student.class”,101);

**Object ob=null;**

**Note:** In case of No data found based on given primary key input, and thenget method returns null data.

**Program Main Method:**

Public class Test{

Public static void main (String[] arg){

Configuration cfg=null;

SessionFactory factory=null;

Session ses=null;

//Transaction tx=null;

Try{

Cfg=new Configuration();

Cfg.configuration();

Ses=factory.OpenSession();

//Tx=ses.beginTranscation();

//Tx.begin();

Object ob=ses.get(“com.app.Student”,101);

Student s1=(Student)ob;

//Tx.commit();

}catch(Exception e){

//tx.rollback();

e.printStackTrace();

}finally{

If(ses!=null){

Ses.close();

} }

System.out.println(“Done”);

}

**Note:** get method internally query select query for this operationtransaction management this optional. For non-select queries like save(insert), update, delete etc….is transaction is required.

**Load():** This method hits session cache to get the object, if object foundthen returns object, else it returns.

**Excetion: ObjectNotFounException:** No row with the given identifier exists.

In case of get, it hits DB table and returns object to session memory, the loads into application, if no data found then returns null.

Example: for load method

**Student std=(Student)ses.load(student.class,101);**

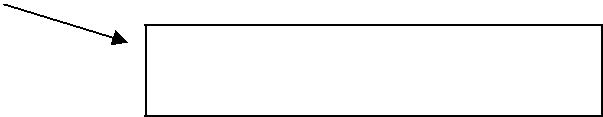
Note: 1) In Transaction Management **begin()** method will be called automatically.

1. To remove one object from session memory use **evict(object)** method.
2. To remove all objects from session memory use clear(), that will not close session.
3. To remove object from session memory and close session, use close() method from session.
4. Object states in Hibernate:
   1. Newly created: transient
   2. Available in session: persistence
   3. Removed from session: Detached
5. hibernate configuration keys can be specified as Hibernate.connection.driver\_class (or) driver\_class

Here hibernate keyword is optional at configuration level.

**Hibernate Generator:** To generator a primary key instead of reading from enduser either predefined class or user defined class is used even through value is provided by end client (or) program. Hibernate Generator will **omit** the value. To define generator to primary key column use below.

Syntax: <generator class=”\_\_\_\_\_\_\_\_\_\_\_\_\_\_ “/>



Generator class Name

**Types of Generators:** In hibernate Generators are categories into two types.

1. Pre-define Generator
2. User-define Generator

**Pre-define Generator:** i) Sequence Generator (oracle sequence)

1. Identity Generator
2. increment Generator (max+1).
3. hilow Generator, Assigned, native.

Example Program: In this stdId is considered as primary key. For this to specify Generator code should be provided in mapping file(\_\_\_\_\_\_\_\_\_\_.hbm.xml).

If we don’t provide any generator the primary key value should be entered manually.

Ex: <class name=”com.app.Student” table=”stdTab”>

<id name=”stdId” column=”sid”>

<generator class=”increment”/>

</id>

<property name=” ” column=” ” />

</class>

**User-define Generator:** A Programmer can define his own generation logic forprimary key value. To define user define generator.

**Step 1:** create a class that implement identifier generator.

IdentifierGenerator(org.hibernate.id.IdentiferGenerator)

**Step 2:** override generation method.

**Step 3:** Define Generation Logic and return value.

Public class MyGen implements IdentiferGenerators

{

Public Serializable generate(SessionImplementor arg0,Object arg1) throws HibernateException{

Date d=new Date();

String sid=”GEN”;

Random r=new Random();

Int x=r.nextInt(1000);

Sid=sid+(d.getYear()+1900)+”-“+(d.getMonth()+1)+d.getDate()+” “ + x;

Return sid;

}

}

**Configuration code for providing user define generator:**

This should be mapped in hibernate mapping file we have to specify fully qualified name of generate class.

Ex: <id name=”stdId” column=”sid”>

<generator class=”com.app.MyGen”/>

**Creating a Sequence in Oracle:**

Create SEQUENCE “HIBERNATE\_SEQUENCE” increment by 20 start with 140;

**Oracle 10gXE:**

Startall programs  oracle 10gXE  go to Data base Homepage (It opens a browser to login)  Enter user name and password (Ex: system / system)  click on administration  Data base User  create button  provide username password and confirm password Select DBA (check box) and click on create button  logout and login with new user  select object Browser (or) SQL>> SQL Command.

**Predefined Generators:**

1. **Assigned:** This is the default generator. It indicates read values fromobject for primary key column. In case if we do not provide any sequence this will be activated by default as below.

Ex: <id name=”stdId” column=”sid”/>

Equals to

<id name=”stdId” column=”sid”>

<generator class=”assigned”/>

</id>

1. **Increment:** It provides next primary key value based on max number ofexited values and increments +1.

Ex: <id name=”stdId” coloumn=”sid”>

<generator class=”increment”/>

</id>

3. **Sequence:** It follows a data base sequence to generate a number series.

Note: Hibernate search by default a sequence name as hibernate\_seqence.

Ex: <id name=”stdId” coloumn=”sid”>

<generator class=”sequence”/>

</id>

Note: To provide different sequence name, use a parameter, to generator as below.

Ex: <id name=”stdId” coloumn=”sid”>

<generator class=”sequence”>

<param name=”sequence”>Custom\_Sequence</param>

</generator>

</id>

1. **Identity Generator:** This generator is data base dependent, this isimplemented by MySQL (Auto-increment), Sybase (max+1), DB (Max+1,Max+n).

Note: In case of MySQL it generates numbers with +1 of previous number.

Ex: <id name=”stdId” coloumn=”sid”>

<generator class=”identity”/>

</id>

Identity Generator is not applicable for oracle DB. This generator uses integer types.

1. **UUID:** Universal Unique Id. It is an hexadecimal represent is stringdatatype, which is generator based on same dynamic properties like, Database details, date and time, IPaddress, OS proportion, along with record no etc….

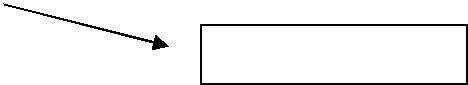
Ex: <id name=”stdId” coloumn=”sid”>

<generator class=”uuid”/>

</id>

Public class Student {

Private String empid;



For UUID

}

1. **Hilo:** This is a number generation algorithm. which follows initialvalue (20=1) and the next value 215 of previous value.

Hilo=Hibernate unique key.

Next level  value + 2 15

Here hibernate\_unique\_key represent the current level data increased to table.

Ex: if 5 records are available in DB, then the next value is 5. For that main table data, next record value will be added with

* 1. i.e.

1. **Native:** It represent generators concept at DB level, related todatabase and which is default in database are, in this case hibernate component (or) programmer are not going to provide any value for different DBs, different algorithm will selected automatically.

Ex: <id name=”stdId” coloumn=”sid”>

<generator class=”native”/>

</id>

For oracle database: user sequence (hibernate\_sequence).

For MySQL: user identity (auto\_increment).

**User defined Generators:**

**Composite-Primary Key:** Combination of more than one column behaves likeprimary key is known as composite primary key.

Note: composite-primary key class must implement serializable interface.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Java code:** |  |  |  |  |
| Public class Employee implements serializable { | |  |  |  |
| Private int empId; |  |  |  |  |
|  |  |  |  |  |
| Private String empName; | Eid | Ename | Esal |  |
|  |  |  |  |  |
| Private double empSal; | 101 | A | - |  |
| // Setters and Getters Method | 101 | B | - |  |
| } | 101 | C | - |  |
| **XML code:** | 102 | A | - |  |
| <hibernate-mapping | 102 | A | - |  |
| package=”com.app”> | 102 | B | - |  |
|  |  |

<class name=”Employee”

table=”etab”>

<Composite-id>

<key-property name=”empId” column=”eid”/> <key-property name=”empName” column=”ename”/>

</composite-id>

<property name=”empSal” column=”esal”>

</class>

</hibernate-mapping>

First Program: Hibernate jars files add in eclise Bulidpath.

TWO JAVA FILES:

TWO XML FILES:

package com.app;

public class Student {

private int stdId;

private String stdName;

private double stdFee;

public int getStdId() {

return stdId;

}

public void setStdId(int stdId) {

this.stdId = stdId;

}

public String getStdName() {

return stdName;

}

public void setStdName(String stdName) { this.stdName = stdName;

}

public double getStdFee() {

return stdFee;

}

public void setStdFee(double stdFee) { this.stdFee = stdFee;

}

@Override

public String toString() {

return "Student [stdId=" + stdId + ", stdName=" + stdName + ",

stdFee="

+ stdFee + "]";

} }

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

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd"> <hibernate-mapping package="com.app">

<class name="Student" table="stutab">

<id name="stdId" column="stdid"/>

<property name="stdName" column="stdname"/> <property name="stdFee" column="stdfee"/>

</class>

</hibernate-mapping>

<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property

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

<property

name="connection.url">jdbc:oracle:thin:@localhost:1521:XE</property> <property name="connection.username">system</property> <property name="connection.password">system</property>

<property

name="dialect">org.hibernate.dialect.OracleDialect</property> <property name="show\_sql">true</property> <property name="format\_sql">true</property> <property name="hbm2ddl.auto">create</property>

<mapping resource = "student.hbm.xml"/>

</session-factory>

</hibernate-configuration>

package com.app;

import org.hibernate.Session;

import org.hibernate.SessionFactory; import org.hibernate.Transaction; import org.hibernate.cfg.Configuration;

public class Test {

public static void main (String[] arg){ Configuration cfg=new Configuration(); cfg=cfg.configure();

SessionFactory sf=cfg.buildSessionFactory(); Session ses=sf.openSession();

Transaction tx=ses.beginTransaction(); tx.begin();

Student stu = new Student();

stu.setStdId(910);

stu.setStdName("AAAbbbb");

stu.setStdFee(54533.42);

ses.save(stu);

tx.commit();

ses.close();

System.out.println(stu);

System.out.println("done");

}

}

Output:

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

Hibernate:insert into stutab(stdname, stdfee, stdid) values(?, ?, ?) Student [stdId=910, stdName=AAAbbbb, stdFee=54533.42] done

**HQL**

**HQL (Hibernate Query Language):** SQL Queries are database dependent. ThisQueries uses table names and column names to get the data and these Queries are case insensitive.

Ex: select \* from emptab;

Select \* from EMPTAB; the same

HQL Queries are database independent. HQL Query construction uses class name and field names.

Ex: public class com.app.Employee{

Public int empId;

Public String empName; }

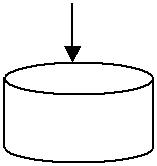
**HQL Query:** SelectempNamefromcom.app.Employee



This is fieldname This is Qualified class name.

Note: HQL Query will be connected into SQL Query by using Dialect. That will be executed on Database.

|  |  |  |  |
| --- | --- | --- | --- |
| HQL Queries are case sensitive at class name and Field name etc…. but | | |  |
| not at SQL classes. |  |  |  |
| HQL |  |  |
| Ex: select, from, where, group by, having etc…. |  |  |
|  |  |  |
| To write HQL Queries and to get the data use Query APS. |  |  |  |
| Dialect |  |  |
| Query (org.hibernate). |  |  |
|  |  |  |
|  |  |  |
| To create Query object use methods create Query. |  |  |  |
| SQL |  |  |
|  |  |  |
| createQuery (String): Query. |  |  |  |
|  |  |  |



To list out all the records in “Query” class use a method.

List():List. DB

This List () return data in the form of list of objects.

Eid Ename Esal

101 A 500

HQL

1. Model class Student.
2. Mapping code (XML) student.hbm.xml.
3. Configuration code (XML) hibernate.cfg.xml.
4. Main Program.

Main program:

Public class Main{

Public static void main (String[] arg){ //common steps -begin

Configuration cfg=null;

SessionFactory factory=null;

Session ses=null;

Transaction tx=null;

Try{

101

A 500

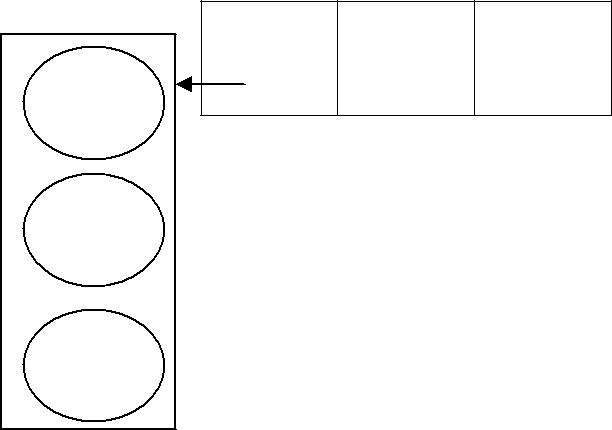
102

B 600

101

1. 700

102 B 600



103 C 700

Cfg=new Configuration();

Cfg.configuration();

Ses=factory.OpenSession();

Tx=ses.beginTranscation();

//common steps end.

//For HQL

String hql=”from com.app.Student”; Query qry=ses.createQuery(hql); List<Student> stdList=qry.list(); Iterator<Student> stditr=stdList.iterator(); /\* End HQL

//\* Display Data

//display format-1 using with while iterator.

While(stditr.hasNext())

{

Student student=stditr.next(); System.out.println(Student.getStdName()); }

//display format-2

System.out.println (stdList);

//format 3

For( Student std:stdList)

{

System.out.println(std);

}

//common steps -begin

Tx.commit();

}catch(Exception e){

//tx.rollback();

e.printStackTrace();

}finally{

If(ses!=null){

Ses.close();

} }//end-common steps System.out.println(“Done”);

}

}

**Pagination using Hibernate:** In Query interface hibernate two methods.

setFirstResult(int);

setMaxResult(int);

Here setFirstResult index Number start from zero.

And MaxResult specifies Max number of records in the result, if can be less than specified length but not going to exceed the length.

Inputs: pageSize=(int type);

pageNumber=(int type); firstNumber=(pageNumber-1)\*pageSize; NoOfPages=(int type)TotalRecords/pageSize;

Ex program: I. Student.java (stdId,stdName,stdFee)

1. student.hbm.xml.

III. Hibernate.cfg.xml. iv. Main program.

Main program:

Public class Test{

Public static void main (String[] arg){ //pagination inputs

int pageSize=5;

int pageNumber=2;

//common steps -begin

Configuration cfg=null;

SessionFactory factory=null;

Session ses=null;

Transaction tx=null;

Try{

Cfg=new Configuration();

Cfg.configuration();

Ses=factory.OpenSession();

Tx=ses.beginTranscation();

//common steps end.

//For HQL with program

String hql=”from Student.class.getName()”; Query qry=ses.createQuery(hql); qry.setFirstResult((pageNumber-1)\*pageSize); qry.setMaxResult(pageSize);

List<Student> listObj=qry.list(); Iterator<Student> stditr=listObj.iterator(); /\* End HQL

//\* Display Data

//display format-1 using with while iterator.

While(stditr.hasNext())

{

Student student=(Student)stdItr.next();

System.out.println(Student);

}

//common steps -begin

Tx.commit();

}catch(Exception e){

//tx.rollback();

e.printStackTrace();

}finally{

If(ses!=null){

Ses.close();

} }//end-common steps System.out.println(“Done”);

}

}

**Partial data loading from table:**

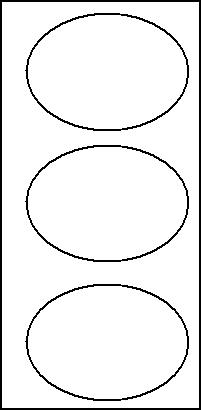
Instead of reading all columns data, we can even specify few columns required. Then only specified columns data will be loaded, this concept is known as partial loading.

Data will be returned in list of object array format (List <object[]>)

**Case 1:** full record loading (with all columns data)

hql=”from com.app.Student”

\*\*Data comes in Student Object Format.



101

A 500

102

B 600

101

1. 700

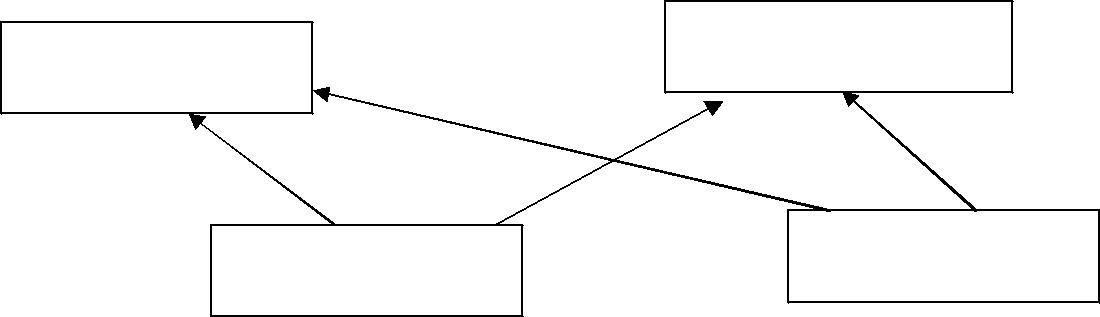
|  |  |  |
| --- | --- | --- |
| Eid | Ename | Esal |
|  |  |  |
| 101 | A | 500 |
| 102 | B | 600 |
| 103 | C | 700 |
|  |  |  |

**Case 2:** Partial record loading (with few columns data)

Hql= “select stdId, stdName from com.app.Student”

\*\*Data will be returned in ObjectArrayFormat.

**Super Types for Wrappers & String class:**

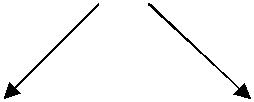


Java.io.serializable

Java.lang.object

Java.lang.String

Java.lang.Number (Class)



Integer Double ….

Case 3: Partial Data loading (only one column Data) can be represented in list object format (on list of its own type format).

Ex: stdId - Int tye can be represented as

1. List<Integer>
2. List<Number>
3. List <Object> \*\*\*\*
4. List <serializable>

Note: Here Hibernate stores data in List as Object format only, later it will be “down casted” to specific type.

Named Query: A HQL Query can be declared in Mapping file (Global Declaration) and if can be read any location.

Ex: Student.hbm.xml

<hibernate-mapping package=” ”>

<class> ….</class>

<Query name=”stdDataQry”>

Select stdName from com.app.Student

</Query>

</hibernate-mapping>

To read / use this query in program, use a method getNamedQuery(String):

Query.

Ex: Program

//configuration

//sessionfactory

//session

//transaction

Query qry=ses.getNamedQuery(“stdDataQry”);

List<object> listData=qry.list();

//display data using iterator.

Passing parameters to HQL Query: In hibernate Data can be passed to Query in two ways. 1. Positional Parameters(?) 2. Named Parameters(:name)

1. **Positional Parameters(?):** Positions always starts with number zero andwe can specify multiple parameters also.

Ex: String hql=”select stdId,stdName,stdFee from com.app.Student where stdId=? And stdName=?”;

Query qry=ses.createQuery(hql);

qry.setParameter(0,16);

qry.setParameter(1,”ABCD”);

List data=qry.list();

Note: setParameter (int,object) method takes values in object format (up casted Data) instead this, we can setXXX() (SetterMethods).

Query qry=ses.createQuery(hql);

qry.setInteger(0,16);

qry.setString(1,”ABCD”);

List data=qry.list();

1. **Named Parameters(:name):** Instead of position number we can specify aname to pass a value (indexing), which is more better than sequencing. To specify use colon (:) with name.

Ex: from com.app.Employee where eid:DataId

To set the value, we can use setParameter(string, object) Method.

Program: String hql=”select stdId,stdName,stdFee from com.app.Student where stdId=:a And stdName=:b”;

Query qry=ses.createQuery(hql);

qry.setParameter(“a”,16);

qry.setParameter(“b”,”ABCDEF”);

(or)

qry.setInteger(“a”,16);

qry.setString(“b”,”ABCD”);

NOTE: Combinations of Positional and Named Parameters:

* ? ? : a (allowed)
* : a ?: b (Not allowed)

: a ?(Not allowed)

* :a(allowed)

Positional parameters are not allowed after named parameters.

* In Case positional parameters are used after named parameters, then hibernate throws QuerySyntaxError.

**Counting numbers of records HQL Query:**

**Format 1:** String hql=”Select Count(\*) from com.app.student”;

**Format 2:** String hql=”select count(stdId) from com.app.student”;

Note: i. Count(1) not valid at HQL level but valid at SQL level.

1. In case of Query return one object / on value than it can stored in list or object. To store list use list() method. To store in object use uniqueResult() method.

Ex: string hql=”select count(\*) from com.app.Student”;

Query qry=ses.createQuery(hql);

List<object> list=qry=qrylist();

System.out.println(list);

(or)

Number ob=(Number)qry.uniqueResult();

**Unique Result ():(Object):** This method is used to store one value/one objecttype object (Employee, Student type etc…). In case of Query return more than one value than it will throw exception.

**List ():** List method can be used event for one record storage also, butdefault list will 10 size, memory will be used for one object, reaming wasted.

HQL Query for Non-Select Operations: HQL Queries can also be written for non-select operation like update, delete, insert etc….

But for basic operations like save, update, delete core API is used in Hibernate. HQL bulk updations and bulk deletions or partial updations can be done using Non-Select Operations.

Ex: Update one column value based on id. To perform this on query object call a method “execut update():int”. It returns no of records updated in DB.

Ex: Program

String hql=”update com.app.Student set “+” stdName=:name, stdFee=:fee “+” where stdId=:Id;

Query query=ses.createQuery(hql);

query.setParameter(“name”,”IIjkl”);

query.setParameter(“fee”,652.32);

query.setParameter(“id”,3);

int x1=query.executeUpdate();

System.out.println(x1);

Tx.commit();

Dynamic Query Construction using String Buffer and Conditional Based:

Configure above class to relate the mapping table in mapping file.

Ex: Student.hbm.xml

Configure hibernate properties and mapping resources in “hibernate.hbm.xml” file.

Ex: hibernate.hbm.xml.

Define main class, by using a conditional based Query construction and data passing.

For Query construction string buffers are used later string buffer can be converted to String format using toString() Method.

In case of String Buffer append () method is used to add or concatenate type inputs or Query object (Buffer object).

Ex: Query construction using String Buffer class.

Select stdName from com.app.Student where stdId=:id.

Above Query is constructed using String Buffer as specified

below.

Code:

1. StringBuffer hql=new StringBuffer(“select stdId”); or

(“select sobj.stdId”).append(“from”+ Student.class.getName()+”as sObj”).append(“where”).append(“sObj.stdId=:sid”);

System.out.println(hql);

1. Public static void main(String[] args)

{ StringBuffer hql=new StringBuffer (“select sobj.stdName)

.append(“from”+Student.class.getName()+”as

sobj”).append(“where”).append(“sobj.stdId:=sid”);

System.out.println(hql);

Output for the above query is : select sobj.stdName from com.app.Student as sobj where sobj.stdId=:sid

**Criteria API:** This API is designed in hibernate to read the data (selectdata) from a DB table with out any query construction by programmer. Instead programmer hibernate will construct Query. It follows internally connected with 2 APIs they are I) Restrictions II) Projections.

I) **Restrictions:** These indicates simple where clause conditions

Ex: >(gt), <(lt), <=(ge), >=(le, =eg, !=ne, between, and, or etc….

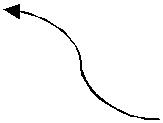
Every restriction returns “criterion” object. That could be added to criteria object.

Design Model:

(interface) Certeria



(interface) Certerion (add)



(class) Restrictions methods

**Code Sample:**

//config

//sesstonfactory

//session

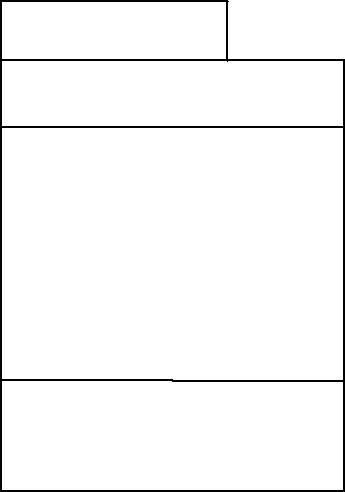
//transactions operations

//commit/rollback

// session close

**Operations Over criteria: By using create criteria (class):**

Criteria methods, table data can be selected that reads data in the list format.



|  |  |
| --- | --- |
| EX: Stdtab | Com.app |
|  |  |

+Student

Sid Sname Sfee

-stdId: int

-stdName: String

-stdFee: double

//setters & getters

**Main program:**

Criteria criteria=ses.createCriteria(Student.class)

//above line meaning= select \* from stdtab;

List list=criteria.list();

System.out.println(list);

(or)

List<Student> list=criteria.list(); Iterator<student> iterator=list.iterator(); While(iterator.hasNext())

{ Student student=(student)iterator.next(); System.out.println(student);

}

**Applying restrictions on criteria:**

Code: 1. empId >= 200

Select \* from emp where empId>=200;

StdId>=200

**Format 1:**

Criteria criteria=ses.criteria(Student.class); Criteria.add(Restrictins.ge(“stdID”,200));

**Format 2:**

Criteria criteria=Restrictions.get(“stdId”,200);

Criteria.add(criteria);

2.Multile Conditons: StdId>200 or stdName like ‘%D’.

**Format 1**

Criteria criteria=ses.create Criteria(Student.class); Criterion criterion1=Restrictions.gt(“stdId”,200); Criterion criterion2=Restrictions.ilike(“stdName”,”%D”); Criterion criterion3=Restrictions.Or(criterion1,criterion2); Criteria.add(criterion3);

**Format 2**

Criteria criteria=ses.create Criteria(Student.class);

Criteria.add(Restricitons.or(Restrictions.gt(“std”,200),Restrictions,il

ike(“stdName”,”%D”)));

**Examples:**

1. Eid>=100 || ename like %AJ%
   * (esak=”5000.36” || estatus ilike “%A%”));

Criteria.add(Restrictions.ge(“eid”, 100);

Restrictions.like(“ename”,”%AJ”);

Restrictions.eq(“esal”,500.36);

Restrictions.ilike(“estatus”, “%A%”));

II Multile conditions

stdId>200 or std Name Ignore case like ‘%D’

1. stdId between 20 and 30 Criteria.add(Restrictions.between(“stdId”,20,30));

iv) where stdName is Ajay

criteria.add(Restrictions.eq(“stdName”,”Ajay”); V) stdId!=10(notequals)

Restrictions.ne(“stdId”,10);

VI) get all records where stdName is not null

Restrictions.isNotNull(“StdId”,10);

SQL: where stdId!=null or where stdId is not null;

VII) get all records where stdName is null

Restrictions.isNull(“stdName”);

SQL: where stdId=null;

VIII) get all records where stdName is empty

Restrictions.isempty(“stdName”);

SQL: where stdName=’ ‘;

IX) get all records where stdName is not empty

Restrictions.isNotEmpty(“stdName”);

SQL: where stdId!=’ ‘;

1. **Projections:** It specifies, specifies columns need to be selected insteadof all columns data. Criteria select all columns data that can be restricted to specific columns using projections.

To specify one projection: Use method property () in projections class and that can be set to criteria object using set Projection.

In case if we specify more than one projection using projection. Property (“--”) than last projection will be considered as only one projection need to be applied.

Code: create criteria object

Use criteria add method to add projections use

Projections.property(“FiledName”);

1. Criteria criteria=ses.createCriteria(Student.class); Criteria.setProjection(Projections.property(“stdId”));

List list=criteria.list();

1. Criteria.setProjection(projections.property(“stdId”));

Criteria.setProjection(projections.property(“stdName”)); Criteria.setProjection(projections.property(“stdFee”));

List list=criteria.list();

**Projection to specify more than one column / Field Data:**

Create projections list object.

Add all projections properties to listSet list to criteria object.

Criteria criteria=ses.createCriteria(Student.class); ProectionList list=proections.projectionList();

Plist.add(Projections.property(“stdName”)); Plist.add(Projections.property(“stdFee”)); Criteria.setProjection(plist); List<Object[]> list=criteria.list();

**Hibernate collection Mapping:**

Hibernate supports collection like

List (list and Bag)

Set

Map as basic collection

And also supports one-to-many etc… as association collections.

Here list and Map are index based collection where set in non-index based collection.

For collection field a new table will be created to store data as supperate entity, and this data will be linked as FK.Pk data.

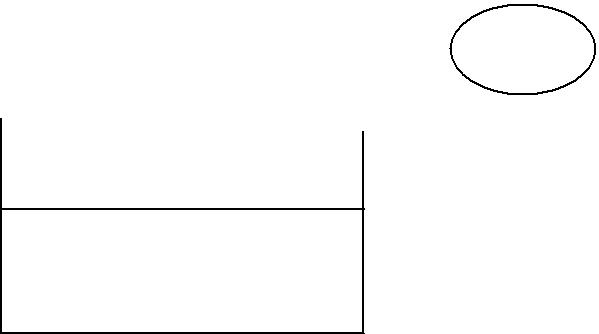
**Every collection table must contain:**

I) Key Column

II) Element column with type

III) Index column (only for List and Map)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ex: Design | |  | Pk | StdTab |  |
|  |  |  |
|  |  |  |  |  |
|  | Com.app |  |  |  |  |
|  |  |  |  |  |  |

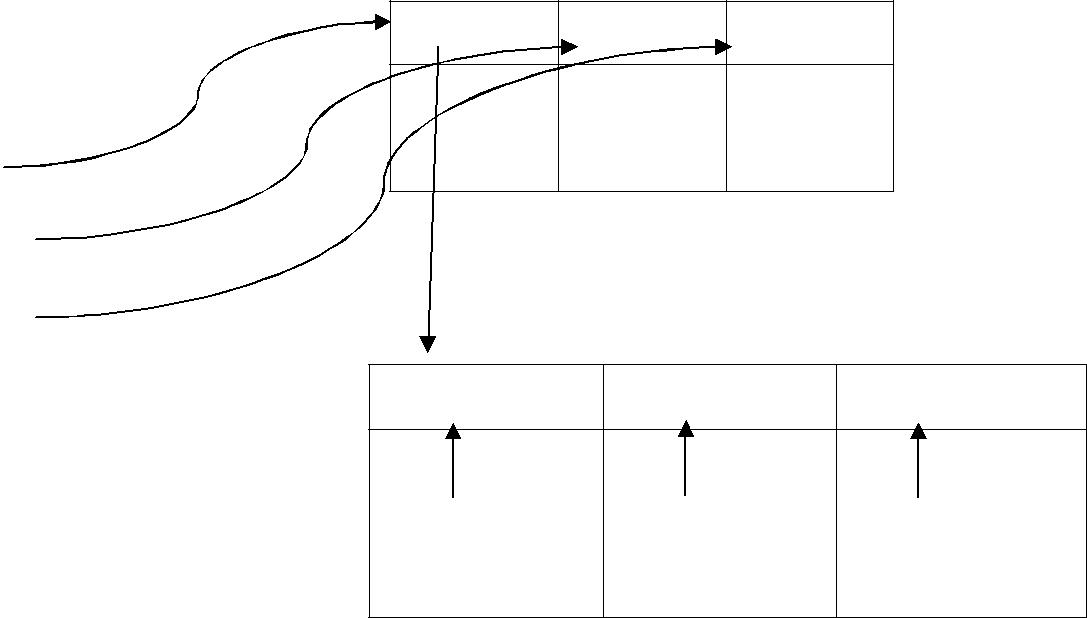


+Student

-stdId: int

-stdName: String

Sid Sname Sfee



SfkId StdAddr Position

|  |  |
| --- | --- |
| Key column Element | Index |
| column | column |

**Coding:**

XML Mapping Code (student.hbm.xml):

<hibernate-mapping package=”com.app”>

<class name=”Studen” table=”stdtab”>

<id name=”stdId” column=”sid”>

<!--<generator class=”increment”/>-->

</id>

<property name=”stdName” column=”sname”/> <property name=”stdFee” column=”sfee”/> <list name=”stdAddr” table=”stdaddrtab”>

<key column=”sfkId”/>

<index column=”posion”/>

<element column=”addrdata” type=”String”/> </list> </class> </hibernate-mapping>

**Main code:**

Tx=ses.beginTransaction();

Tx.begin();

List<String> listAddr=new ArrayList<String>(); listAddr.add(“Address 101-1”);

listAddr.add(“Address 101-2”);

listAddr.add(“Address 101-3”);

Student stdObj=new Student();

stdObj.setStdId(103);

stdObj.setStdName(“SAJ”);

stdObj.setStdFee(3546.23);

stdObj.setAddr(listAddr);

ses.save(stdObj);

**Map Collection:** Map Collection takes 3 inputsI)Key Column (as Fk, refers Pk in parent) II)Index column (key from Map) III)element column (value from Map)

Ex: package com.app;

Public class Student {

Private int stdId;

Private String stdName;

Private double stdFee;

Private Map<String,String> addrMap;

//setters and getters

//toString methods

}

**Student.hbm.xml:**

<hibernate-mapping package=”com.app”>

<class name=”Studen” table=”stdtab”>

<id name=”stdId” column=”sid”/>

<property name=”stdName” column=”sname”/>

<property name=”stdFee” column=”sfee”/>

<map name=”addrMap” table=”stdaddrtab”>

<key column=”sfkId”/>

<index column=”mapkey”/>

<element column=”mapval” type=”String”/> </map> </class> </hibernate-mapping>

**Main Code:**

//configuration

* sessionfactory, session, transaction //operation

Map<string,string> mapAddr=new HashMap<String,String>(); mapAddr.put(“key-addr1”,”val-addr1”); mapAddr.put(“key-addr2”,”val-addr2”); mapAddr.put(“key-addr3”,”val-addr3”);

Student stdObj=new Student(); stdObj.setstdId(103); stdObj.setstdName(“SAH”); stdObj.setstdFee(2435.24); stdObj.setstdAddr(mapAddr); ses.save(stdObj);

Note: Map Data will be stored in child table as

Map key will be stored in  Index Column

Map value will be stored in  element column

Hibernate Collections Mapping:

Hibernate supports all basic collection mapping, that stores as new table. Basic collections are divided into two types.

1. Index based: List, Map.
2. Non-Index based: Set.

Every collection-mapping table must contains

Key column - (refers primary key of parent)

Element column – (represent data/map value)

And index based collection contains

Index column – (represents position/map key)

Note:

1. In case of list index type is int so not required to specify any type for index and need to specify type of only element.
2. In case of map type is required to specify for both index column (map key) and element column (map value)
3. Set never contains index. So, type is required only for element column.
4. For ever child table providing name is optional hibernate by default consider parent table name\_variable name as table name, to specify our own table name use table attribute as collection tag level.

package com.app;

import java.util.List;

public class Student {

private int stdId;

private String stdName;

private double stdFee;

private List<String> stdAddr;

public Student() {

super();

}

public Student(int stdId, String stdName, double stdFee,

List<String> stdAddr) {

super();

this.stdId = stdId;

this.stdName = stdName;

this.stdFee = stdFee;

this.stdAddr = stdAddr;

}

public int getStdId() {

return stdId;

}

public void setStdId(int stdId) {

this.stdId = stdId;

}

public String getStdName() {

return stdName;

}

public void setStdName(String stdName) { this.stdName = stdName;

}

public double getStdFee() {

return stdFee;

}

public void setStdFee(double stdFee) { this.stdFee = stdFee;

}

public List<String> getStdAddr() {

return stdAddr;

}

public void setStdAddr(List<String> stdAddr) { this.stdAddr = stdAddr;

}

@Override

public String toString() {

return "Employee [stdId=" + stdId + ", stdName=" + stdName

+ ", stdFee=" + stdFee + ", stdAddr=" + stdAddr + "]";

}

}

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

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property

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

<property

name="connection.url">jdbc:oracle:thin:@localhost:1521:XE</property> <property name="connection.username">system</property> <property name="connection.password">system</property>

<property

name="dialect">org.hibernate.dialect.OracleDialect</property> <property name="show\_sql">true</property> <property name="format\_sql">true</property> <property name="hbm2ddl.auto">create</property>

<mapping resource = "student.hbm.xml"/>

</session-factory>

</hibernate-configuration>

<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd"> <hibernate-mapping package="com.app">

<class name="Student" table="stdtab1">

<id name="stdId" column="sid"/>

<property name="stdName" column="sname" />

<property name="stdFee" column="sfee" />

<list name="stdAddr" table="stdaddr">

<key column="sfkId"/>

<index column="pos"/>

<element column="addrdata" type="string"/> </list>

</class>

</hibernate-mapping>

package com.app;

import java.util.ArrayList;

import java.util.List;

import org.hibernate.Session;

import org.hibernate.SessionFactory; import org.hibernate.Transaction; import org.hibernate.cfg.Configuration;

public class Main {

public static void main(String[] args) {

Configuration cfg=new Configuration().configure();

SessionFactory sf=cfg.buildSessionFactory();

Session session=sf.openSession(); Transaction tx=session.beginTransaction(); tx.begin();

Student std=new Student();

std.setStdId(101);

std.setStdName("Venu");

std.setStdFee(3000);

List<String> list=new ArrayList<String>();

list.add("Bangalore");

list.add("Hyderabad");

list.add("Kurnool");

list.add("Mysore");

std.setStdAddr(list);

session.save(std);

tx.commit();

session.close();

System.out.println("Record update successfully");

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| } } |  |  |  |  |  |  |
| OUTPUT: | SID | SNAME | SFEE |  |  |  |
|  |  | SFKID | ADDRDATA | POS |  |
|  |  |  |  |
|  | 101 | Venu | 3000 |  |  |  |
|  |  |  | 101 | Bangalore | 0 |  |
|  |  |  | 101 | Hyderabad | 1 |  |
|  |  |  | 101 | Kurnool | 2 |  |
|  |  |  | 101 | Mysore | 3 |  |

**\*\*Bag Collection (Hibernate specific):** A Bag is collection of items whichcan even store duplicate items.

A bag is list type implementation where list maintains index at table level and bag will not maintain any index.

Bag is most performance effective than all collections.

Map is slower than all collection (configuration and reading).

EX:

package com.app;

Public class Student {

Private int stdId;

Private String stdName;

Private double stdFee;

Private List<String > addrList;

//setters and getters

//toString methods

}

**Student.hbm.xml:**

<hibernate-mapping package=”com.app”> <class name=”Studen” table=”stdtab”> <id name=”stdId” column=”sid”/>

<property name=”stdName” column=”sname”/> <property name=”stdFee” column=”sfee”/> <bag name=”addrList” table=”stdaddrtab”>

<key column=”sfkId”/>

<element column=”value” type=”String”/>

</bag> </class> </hibernate-mapping>

**Main Code:**

//configuration

* sessionfactory, session, transaction //operation

List<string> ListAddr=new ArrayList<String>(); ListAddr.add(“Addr1”); ListAddr.add(“Addr2”); ListAddr.add(“Addr3”);

Student stdObj=new Student(); stdObj.setstdId(103); stdObj.setstdName(“SAH”); stdObj.setstdFee(2435.24); stdObj.setstdAddr(listAddr); ses.save(stdObj);

**Inheritance Mapping:** Model classes can follow inheritance method, for thistable can be created in below ways.

I) Table per class

* 1. Table per Concret class
  2. Table per subclass

1. **Table per class:** It create single table per complete hierarchy. It bindsall columns into single table with extra column know as discriminator.

For this we have to specify discriminator-value at class level to identify what type of record created in DB.

For Example:

Design:

To represent parent class in mapping code we have to specify <class> tag.

To represent child class we have to specify <subclass> tag.

Java Code:

1. Public class Employee { Private int empId;

Private String empName;

….}

1. Public class Manager extends Employee { Private String projStatus;

Private String version;

III. Public class Developer extends Employee

Private String tech;

Private String tools;

….}

{

**Employee.hbm.xml:**

<class name=”Employee” table=”emptab” discriminator-value=”0”> <id name=”empId” column=”eid”/>

<discriminator column=”format” type=”String”/> <property name=”empName” column=”ename”/> <subclass name=”Manager” discriminator-value=”1”>

<property name=”projstatus” column=”pstatus”/> <property name=”version” column=”version”/>

</subclass>

<subclass name=”Developer” discriminator-value=”2”> <property name=”tech” column=”tech”/> <property name=”tools” column=”tools”/>

</subclass>

</class>

**Main Code:**

//configuration

* sessionfactory, session, transaction //operation

//Employee Object

Employee emp=new Employee(); emp.setEmpId(101); emp.setEmpName(“A”); //Manager Object

Manager mgr=new Manager(); mgr.setEmpId(102); mgr.setEmpName(“B”); mgr.setProjStatus(“RSL”); mgr.setVersion(“1.2”); //Developer Object

Developer dev=new Developer(); dev.setEmpId(103); dev.setEmpName(“C”); dev.setTech(“Java”); dev.setTools(“Eclipse,Ant”); ses.save(emp); ses.save(mgr); ses.save(dev);

tx.commit();

**Pagination Logic:**

Inputs: PageNumber

PageSize

Recordcount

Totalpages=(recordscount/pageSize)+((recordCount%PageSize>)0?1:0)

Emplist=Session.createCriteria(Employee.class).setFirstResult((pageNumb er-1)\*pageSize).setMaxResult(pagesize).list());

**Hibernate-joins:**

Hibernate supports 4 types of joins

* 1. Inner join or join
  2. Left Outer join (or) left join
  3. Right outer join (or) right join
  4. Full outer join (or) full join

1. **Inner join or join:**

**UML for classes:** for example, consider two classes which arehaving mapping relation (like one-to-many etc….) for those, to read data from specific column then joins are used.

Joins only used to read data by making connection between 2 tables.

Classes: public class com.app.Employee{

Private int empId;

Private String empName;

Private double empSal;

Private Set<Address> addrSet;

* //setter and getters

}

Public class Address {

Private int addrId;

Private String location;

Private String hno;

…//setters and getters

}

And Consider same data under the tables as below,

|  |  |  |
| --- | --- | --- |
| EID | ENNAME | ESAL |
|  |  |  |
| 101 | Abcd | 256.36 |
| 102 | MNOP | 156.36 |
|  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 103 | IJKL | 456.36 |  |  |  |  |  |  |
| 104 | xyz | 856.36 |  | AID | LOC | HNO | EFKID |  |
|  |  |  |  |  |  |
|  |  |  |  | 10 | HYD | 1-21 | 101 |  |
|  |  |  |  |
|  |  |  | | 11 | HYD1 | 1-213 | 101 |  |
|  |  |  | | 12 | BAN | 1-21 | 102 |  |
|  |  |  | | 13 | MNO | 996-6 | NULL |  |
|  |  |  | | 14 | PQR | 12/85 | NULL |  |
|  |  |  |  |  |  |  |  |  |

Query for Inner join/join

Select emp.empId, emp.empName, addr.location, addr.hno from com.app.Employee as emp

Join emp.addrSet as addr.

Note: To join two classes / tables use join field in the parent class and use alias name to specify the join of child.

Ex: from <parentclass> as <alias-name1> <join-type>

<alias-names> <join-field as <alias-name>>

Here alias-name1 indicates parent class and alias-name2 indicates child classes.

For above Query, data is:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 101 | abcd | 256.36 | 10 | hyd | 1-21 |
| 101 | abcd | 256.36 | 11 | hyd1 | 1-213 |
| 102 | MNop | 156.36 | 12 | BAN | 1-21 |

|  |  |  |  |
| --- | --- | --- | --- |
| 101 | abcd | hyd | 1-21 |
| 101 | abcd | hyd1 | 1-213 |
| 102 | MNOP | BAN | 1-21 |

**Left Join:**

**Query:** select emp.empId, addr.location from com.app.Employee as empleft outer join emp.addrSet as addr

**Data:**

101 hyd

1. hyd1
2. BAN
3. Null
4. Null

**Right Join:**

**Query:** select emp.empId, addr.location from com.app.Employee as emp rightjoin emp.addrSet as addr

**Data:**

101 10 hyd

101 11 hyd1

102 12 BAN

Null 13 MNO

Null 14 PQR

**Cache Concept in Hibernate:**

It a method of main ting (temporary memory at application level).

**Cache types:**

* 1. First Level (session cache)
  2. Second Level (session Factory cache)

1. **First Level:** session cache maintained is taken care by hibernate only.As programmer we can not disable (or) enable this cache.

Once Session close and opened data will be loaded database only.

1. **Session Factory cache:** Instead going to data base every time, we cancreate a temporary memory at application side (which can be enabled and disabled by programmer) to store data for some time.

**Annotation in Hibernate:** To specify configuration at java code (Model classlevel) 2 required annotations are need to be applied they are:

@ Entity indicates class

@ Id primary key column

To specify the column names and tables names use

* Table (name = “ “)
* Column (name=” “)

To configuration this class in cfg files use below:

SYN: <mapping class=” “/>

Ex: <mapping class=”com.app.Employee”/>

Session Factory as Singleton Object: For every Application we maintain SF as single object. To convert SF as Singleton, we use singleton Design pattern. as below format.

**Format 1:**

Using Static Block: Static block will be executed only once, at the time loading the class into JVM, or on server Startup.

Public class HibernateUtill {

Private static SessionFactory factory=null;

Static {

Configuration cfg=new configuration().configure(); Factory=cfg.buildSessionFactory(); }

Public static Session getSessionFactoryObject() {

Return factory;

}

**Format 2:** /\*\* code as static method explicit call \*\*/

Public class HibernateUtill {

Private static SessionFactory factory=null;

Public Static sessionfactory getSessionFactoryObject(){

If(factory==null)

{

Configuration cfg=new configuration().configure();

Factory=cfg.buildSessionFactory();

}

Return factory;

}

}

**Format 3:** /\*\* code as static method implicit call\*\*/

Public class HibernateUtill {

Private static SessionFactory factory=null;

Public Static sessionfactory assignFirstValue(){

Return new configuration().configure().buildSessionFactory();

}

Public static Session getSessionFactoryObject() {

Return factory;

}

Sample Programs

**package** com.app;

**public class** Account {

**private int** accNum;

**private** Stringname;

**private double** balance;

**public int** getAccNum() {

**return** accNum;

}

**public void** setAccNum(**int** accNum) {

**this**.accNum=accNum;

}

**public** String getName() {

**return** name;

}

**public void** setName(Stringname) {

**this**.name=name;

}

**public double** getBalance() {

**return** balance;

}

**public void** setBalance(**double** balance) {

**this**.balance=balance;

}

@Override

**public** String toString() {

**return** "Account [accNum="+accNum+", name="+name+",

balance="

+ balance + "]";

}

}

<?xml version=*"1.0"* encoding=*"UTF-8"*?> <!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property

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

name=*"connection.url"*>jdbc:oracle:thin:@localhost:1521:XE</property> <property name=*"connection.username"*>system</property> <property name=*"connection.password"*>system</property> <property

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

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

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

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

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

</session-factory>

</hibernate-configuration>

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

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd"> <hibernate-mapping package=*"com.app"*>

<class name=*"Account"* table=*"accounttab"*>

<id name=*"accNum"* column=*"acctno"*/>

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

<property name=*"balance"* column=*"Balance"*/> </class>

</hibernate-mapping>

**package** com.app;

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

**import** org.hibernate.Transaction;

**public class** Test {

**public static void** main(String[]args) {Configuration cfg=**new** Configuration(); cfg.configure("hibernate.cfg.xml"); SessionFactory factory=cfg.buildSessionFactory(); Session session=factory.openSession(); Transaction tx=session.beginTransaction(); tx.begin();

Account stu = **new** Account();

stu.setAccNum(2454);

stu.setBalance(252244.54);

stu.setName("Raju");;

session.save(stu);

tx.commit();

session.close();

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

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

session.close();

}

}

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

Hibernate:

insert

into

accounttab

(Name, Balance, acctno)

values

(?, ?, ?)

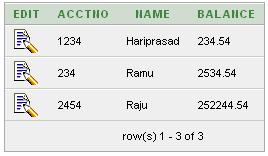
Account [accNum=2454, name=Raju, balance=252244.54] done

Exception in thread "main" org.hibernate.SessionException: Session was already closed

at org.hibernate.impl.SessionImpl.close(SessionImpl.java:320)

at com.app.Test.main(Test.java:25)

http://127.0.0.1:8080/apex/f?p=4500:1001:2475510258818860::NO:::



Hibernate Collections

http://127.0.0.1:8080/apex/f?p=4500:1001:1583328310102601::NO:::

package com.app;

import java.util.List;

public class Student {

private int stdId;

private String stdName;

private double stdFee;

private List<String> stdAddr;

public Student() {

super();

}

public Student(int stdId, String stdName, double stdFee, List<String> stdAddr) {

super();

this.stdId = stdId;

this.stdName = stdName;

this.stdFee = stdFee;

this.stdAddr = stdAddr;

}

public int getStdId() {

return stdId;

}

public void setStdId(int stdId) {

this.stdId = stdId;

}

public String getStdName() {

return stdName;

}

public void setStdName(String stdName) { this.stdName = stdName;

}

public double getStdFee() {

return stdFee;

}

public void setStdFee(double stdFee) { this.stdFee = stdFee;

}

public List<String> getStdAddr() {

return stdAddr;

}

public void setStdAddr(List<String> stdAddr) {

this.stdAddr = stdAddr;

}

@Override

public String toString() {

return "Employee [stdId=" + stdId + ", stdName=" + stdName

+ ", stdFee=" + stdFee + ", stdAddr=" + stdAddr + "]";

}

}

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

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property

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

<property

name="connection.url">jdbc:oracle:thin:@localhost:1521:XE</property> <property name="connection.username">system</property> <property name="connection.password">system</property>

<property

name="dialect">org.hibernate.dialect.OracleDialect</property> <property name="show\_sql">true</property> <property name="format\_sql">true</property> <property name="hbm2ddl.auto">create</property>

<mapping resource = "student.hbm.xml"/>

</session-factory>

</hibernate-configuration>

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

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd"> <hibernate-mapping package="com.app">

<class name="Student" table="stdtab1">

<id name="stdId" column="sid"/>

<property name="stdName" column="sname" />

<property name="stdFee" column="sfee" />

<list name="stdAddr" table="stdaddr">

<key column="sfkId"/>

<index column="pos"/>

<element column="addrdata" type="string"/> </list>

</class>

</hibernate-mapping>

package com.app;

import java.util.ArrayList;

import java.util.List;

import org.hibernate.Session;

import org.hibernate.SessionFactory; import org.hibernate.Transaction; import org.hibernate.cfg.Configuration;

public class Main {

public static void main(String[] args) {

Configuration cfg=new Configuration().configure();

SessionFactory sf=cfg.buildSessionFactory();

Session session=sf.openSession(); Transaction tx=session.beginTransaction(); tx.begin();

Student std=new Student();

std.setStdId(101);

std.setStdName("Venu");

std.setStdFee(3000);

List<String> list=new ArrayList<String>();

list.add("Bangalore");

list.add("Hyderabad");

list.add("Kurnool");

list.add("Mysore");

std.setStdAddr(list);

session.save(std);

tx.commit();

session.close();

System.out.println("Record update successfully");

}

}

OutPut:

SID SNAME SFEE

1. Venu 3000

row(s) 1 - 1 of 1

SFKID ADDRDATA

POS

101 Bangalore

0

|  |  |  |
| --- | --- | --- |
| 101 | Hyderabad | 1 |
| 101 | Kurnool | 2 |
| 101 | Mysore | 3 |
|  | row(s) 1 - 4 of 4 | |

package com.app;

import java.util.List;

public class StudentDetails {

private int stdId;

private String stdName;

private double stdFee;

private List<Integer> stdMarks;

public StudentDetails() {

super();

}

public StudentDetails(int stdId, String stdName, double stdFee, List<Integer> stdMarks) {

super();

this.stdId = stdId;

this.stdName = stdName;

this.stdFee = stdFee;

this.stdMarks = stdMarks;

}

public int getStdId() {

return stdId;

}

public void setStdId(int stdId) {

this.stdId = stdId;

}

public String getStdName() {

return stdName;

}

public void setStdName(String stdName) { this.stdName = stdName;

}

public double getStdFee() {

return stdFee;

}

public void setStdFee(double stdFee) { this.stdFee = stdFee;

}

public List<Integer> getStdMarks() {

return stdMarks;

}

public void setStdMarks(List<Integer> stdMarks) { this.stdMarks = stdMarks;

}

@Override

public String toString() {

return "StudentDetails [stdId=" + stdId + ", stdName=" + stdName

+ ", stdFee=" + stdFee + ", stdMarks=" + stdMarks +

"]";

}

}

<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd"> <hibernate-mapping>

<class name="com.app.StudentDetails" table="stdTab">

<id name="stdId" column="sid"/> <property name="stdName" column="sname"/> <property name="stdFee" column="sfee"/> <list name="stdMarks" table="stdmarks"> <key column="sidfk"/> <index column="No"/>

<element column="Marks" type="int"/>

</list>

</class>

</hibernate-mapping>

<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property

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

operty>

<property

name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:XE</propert

y>

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

<property name="hibernate.connection.password">system</property>

<property

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

<property name="show\_sql">true</property>

<property name="format\_sql">true</property>

<property name="hbm2ddl.auto">create</property>

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

</session-factory>

</hibernate-configuration>

package com.app;

import java.util.ArrayList;

import java.util.List;

import org.hibernate.Session;

import org.hibernate.SessionFactory; import org.hibernate.Transaction; import org.hibernate.cfg.Configuration;

public class Main {

public static void main (String[] arg){ Configuration cfg=new Configuration().configure(); SessionFactory sf=cfg.buildSessionFactory(); Session ses=sf.openSession();

Transaction tx=ses.beginTransaction(); tx.begin();

List<Integer> stdlist=new ArrayList<Integer>();

stdlist.add(75);

stdlist.add(79);

stdlist.add(85);

stdlist.add(56);

StudentDetails std=new StudentDetails();

std.setStdId(20);

std.setStdName("Sivakumar");

std.setStdFee(3000);

std.setStdMarks(stdlist);

ses.save(std);

tx.commit();

ses.close();

}

}

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

Hibernate:

insert into stdTab(sname, sfee, sid) values(?, ?, ?)

Hibernate:

insert into stdmark(sidfk, No, Marks) values(?, ?, ?)

Hibernate:

insert into stdmarks(sidfk, No, Marks) values(?, ?, ?)

Hibernate:

insert into stdmarks(sidfk, No, Marks) values(?, ?, ?)

Hibernate:

insert into stdmarks(sidfk, No, Marks) values(?, ?, ?)

StdTab

SID SNAME

SFEE

20 Sivakumar

3000

row(s) 1 - 1 of 1

stdMarks

Insert Row

|  |  |  |  |
| --- | --- | --- | --- |
| EDIT | SIDFK MARKS NO | | |
| Edit | 20 | 75 | 0 |
| Edit | 20 | 79 | 1 |
| Edit | 20 | 85 | 2 |
| Edit | 20 | 56 | 3 |

**This program collections (List,Set,Map)**

package com.app;

import java.util.List;

import java.util.Map;

import java.util.Set;

public class Employee {

private int empId;

private String empName;

private double empSal;

private List<String> empProj;

private Set<String> sampleOne;

private Map<String,String> sampleTwo;

public Employee() {

super();

}

public Employee(int empId, String empName, double empSal, List<String> empProj, Set<String> sampleOne, Map<String, String> sampleTwo) {

super();

this.empId = empId;

this.empName = empName;

this.empSal = empSal;

this.empProj = empProj;

this.sampleOne = sampleOne;

this.sampleTwo = sampleTwo;

}

public int getEmpId() {

return empId;

}

public void setEmpId(int empId) {

this.empId = empId;

}

public String getEmpName() {

return empName;

}

public void setEmpName(String empName) { this.empName = empName;

}

public double getEmpSal() {

return empSal;

}

public void setEmpSal(double empSal) { this.empSal = empSal;

}

public List<String> getEmpProj() {

return empProj;

}

public void setEmpProj(List<String> empProj) { this.empProj = empProj;

}

public Set<String> getSampleOne() {

return sampleOne;

}

public void setSampleOne(Set<String> sampleOne) { this.sampleOne = sampleOne;

}

public Map<String, String> getSampleTwo() { return sampleTwo;

}

public void setSampleTwo(Map<String, String> sampleTwo) { this.sampleTwo = sampleTwo;

}

@Override

public String toString() {

return "Employee [empId=" + empId + ", empName=" + empName

* ", empSal=" + empSal + ", empProj=" + empProj
* ", sampleOne=" + sampleOne + ", sampleTwo=" +

sampleTwo + "]";

}

}

<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

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

<hibernate-configuration>

<session-factory>

<property

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

operty>

<property

name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:XE</propert

y>

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

name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property> <property name="show\_sql">true</property>

<property name="format\_sql">true</property> <property name="hbm2ddl.auto">create</property>

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

</session-factory>

</hibernate-configuration>

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

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd"> <hibernate-mapping>

<class name="com.app.Employee" table="empTab11"> <id name="empId" column="eid"/>

<property name="empName" column="ename"/> <property name="empSal" column="esal"/> <list name="empProj" table="empproject">

<key column="eidfk"/>

<index column="position"/>

<element column="data" type="string"/> </list>

<set name="sampleOne" table="samOne">

<key column="eidfk" />

<element column="data" type="string"/> </set>

<map name="sampleTwo" table="samTwo">

<key column="eidfk"/>

<index column="position" type="string"/> <element column="data" type="string"/> </map>

</class>

</hibernate-mapping>

package com.app;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.HashSet;

import java.util.List;

import java.util.Map;

import java.util.Set;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction; import org.hibernate.cfg.Configuration;

public class Main {

public static void main (String[] arg){ Configuration cfg=new Configuration().configure(); SessionFactory sf=cfg.buildSessionFactory(); Session ses=sf.openSession();

Transaction tx=ses.beginTransaction(); tx.begin();

List<String> emplist=new ArrayList<String>();

emplist.add("asdf");

emplist.add("mnop");

emplist.add("ghij");

emplist.add("pqrst");

Set<String> empset.add("A"); empset.add("B"); empset.add("C"); empset.add("D");

empset=new HashSet<String>();

Map<String,String> empmap=new HashMap<String,String>();

empmap.put("k1", "v1");

empmap.put("k2", "v2");

empmap.put("k3", "v3");

empmap.put("k4", "v4");

Employee emp=new Employee();

emp.setEmpId(101);

emp.setEmpName("Ramesh");

emp.setEmpSal(34555.00);

emp.setEmpProj(emplist);

emp.setSampleOne(empset);

emp.setSampleTwo(empmap);

ses.save(emp);

tx.commit();

ses.close();

}

}

Output:

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

|  |  |  |
| --- | --- | --- |
| SLF4J: Defaulting | | to no-operation (NOP) logger implementation |
| SLF4J: See | http://www.slf4j.org/codes.html#StaticLoggerBinder for further | |
| details. |  |  |
| Hibernate: insert | | into empTab11(ename, esal, eid) values(?, ?, ?) |
| Hibernate: insert | | into empproject(eidfk, position, data) values(?, ?, ?) |
| Hibernate: insert | | into empproject(eidfk, position, data) values(?, ?, ?) |
| Hibernate: insert | | into empproject(eidfk, position, data) values(?, ?, ?) |
| Hibernate: insert | | into empproject(eidfk, position, data) values(?, ?, ?) |
| Hibernate: insert | | into samOne(eidfk, data) values(?, ?) |
| Hibernate: insert | | into samOne(eidfk, data) values(?, ?) |
| Hibernate: insert | | into samOne(eidfk, data) values(?, ?) |
| Hibernate: insert | | into samOne(eidfk, data) values(?, ?) |
| Hibernate: insert | | into samTwo(eidfk, position, data) values(?, ?, ?) |

Hibernate: insert into samTwo(eidfk, position, data) values(?, ?, ?)

Hibernate: insert into samTwo(eidfk, position, data) values(?, ?, ?)

Hibernate: insert into samTwo(eidfk, position, data) values(?, ?, ?)

EmpTab11

EID ENAME ESAL

101 Ramesh 34555

EmpProject

EIDFK DATA POSITION

1. asdf 0
2. mnop 1
3. ghij 2
4. pqrst 3

samOne

EIDFK DATA

1. D
2. A
3. B
4. C

samTo

EIDFK DATA POSITION

101 v3 k3

101 v4 k4

101 v1 k1

101 v2 k2

============================================================================ ===========================

package com.app;

import java.util.List;

import java.util.Set;

public class Product {

private int prdId;

private List<String> parts;

private Set<String> buildItems;

private List<Integer> content;

public Product() {

super();

}

public Product(int prdId, List<String> parts, Set<String> buildItems, List<Integer> content) {

super();

this.prdId = prdId;

this.parts = parts;

this.buildItems = buildItems;

this.content = content;

}

public int getPrdId() {

return prdId;

}

public void setPrdId(int prdId) {

this.prdId = prdId;

}

public List<String> getParts() {

return parts;

}

public void setParts(List<String> parts) { this.parts = parts;

}

public Set<String> getBuildItems() {

return buildItems;

}

public void setBuildItems(Set<String> buildItems) { this.buildItems = buildItems;

}

public List<Integer> getContent() {

return content;

}

public void setContent(List<Integer> content) { this.content = content;

}

@Override

public String toString() {

return "Product [prdId=" + prdId + ", parts=" + parts + ", buildItems="

+ buildItems + ", content=" + content + "]";

}

}

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

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property

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

<property

name="connection.url">jdbc:oracle:thin:@localhost:1521:XE</property> <property name="connection.username">system</property> <property name="connection.password">system</property>

<property

name="dialect">org.hibernate.dialect.OracleDialect</property> <property name="show\_sql">true</property> <property name="format\_sql">true</property> <property name="hbm2ddl.auto">create</property>

<mapping resource = "product.hbm.xml"/>

</session-factory>

</hibernate-configuration>

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

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd"> <hibernate-mapping package="com.app">

<class name="Product" table="productTab"> <id name="prdId" column="pid"/> <list name="parts" table="partsTab">

<key column="pidfk"/>

<index column="pos"/>

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

</list>

<set name="buildItems" table="buildTab">

<key column="pidfk"/>

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

</set>

<list name="content" table="contenTab"> <key column="pidfk"/>

<index column="pos"/>

<element column="content" type="int"/>

</list>

</class>

</hibernate-mapping>

package com.app;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import org.hibernate.Session;

import org.hibernate.SessionFactory; import org.hibernate.Transaction; import org.hibernate.cfg.Configuration;

public class Main {

public static void main(String[] args) {

Configuration cfg=new Configuration().configure();

SessionFactory sf=cfg.buildSessionFactory();

Session session=sf.openSession(); Transaction tx=session.beginTransaction(); tx.begin();

Product p=new Product();

p.setPrdId(100);

List<String> list=new ArrayList<String>();

list.add("adf");

list.add("hjkl");

list.add("mnop");

p.setParts(list);

Set<String> bn=new HashSet<String>();

bn.add("M");

bn.add("N");

bn.add("O");

p.setBuildItems(bn);

List<Integer> list1=new ArrayList<Integer>();

list1.add(20);

list1.add(2);

list1.add(89);

p.setContent(list1);

session.save(p);

tx.commit();

session.close();

System.out.println("successfully");

}

}

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

Hibernate: insert into productTab(pid) values(?)

Hibernate: insert into partsTab(pidfk, pos, parts) values(?, ?, ?)

Hibernate: insert into partsTab(pidfk, pos, parts) values(?, ?, ?)

Hibernate: insert into partsTab(pidfk, pos, parts) values(?, ?, ?)

Hibernate: insert into buildTab(pidfk, buildItems) values(?, ?)

Hibernate: insert into buildTab(pidfk, buildItems) values(?, ?)

Hibernate: insert into buildTab(pidfk, buildItems) values(?, ?)

Hibernate: insert into contenTab(pidfk, pos, content) values(?, ?, ?)

Hibernate: insert into contenTab(pidfk, pos, content) values(?, ?, ?)

Hibernate: insert into contenTab(pidfk, pos, content) values (?, ?, ?)

successfully

PRODUCTTAB

PID

100

PARTSTAB

PIDFK PARTS POS

100 adf 0

1. hjkl 1
2. mnop 2

BUILDTAB

PIDFK BUILDITEMS

1. M
2. N
3. O

|  |  |  |
| --- | --- | --- |
| CONTENTTAB |  |  |
| PIDFK CONTENT | | POS |
| 100 | 20 | 0 |
| 100 | 2 | 1 |

100 89 2

**Hibernate Mapping:**

One - Association Mapping:

I) One-to-one (non-collection)

1. one-to-many(collection dependent)
2. many-to-many(non-collection)

IV) many-to-many(collection dependent)

\*) Two classes can have relation with each other using HAS-A relation, here 2 tables are created for 2 classes those contains fkcolumn to link each other table.

\*) Always code, XML changes comes at parent class level and table changes comes at \* level.

**Many-to-one:** One child record from child table can have connection withmultiple (zero or more) parent table records.

\*) many-to-many is non-collection configuration.

\*) For this case changes comes at parent table child table primary key (PK) will become parent table Foregin key (FK).

**Many-to-One Example Program:**

package com.app.model;

public class Employee {

private int empId;

private String empName;

private double empSal;

private Address addr;

public Employee() {

super();

}

public Employee(int empId, String empName, double empSal, Address addr)

{

super();

this.empId = empId;

this.empName = empName;

this.empSal = empSal;

this.addr = addr;

}

public int getEmpId() {

return empId;

}

public void setEmpId(int empId) {

this.empId = empId;

}

public String getEmpName() {

return empName;

}

public void setEmpName(String empName) { this.empName = empName;

}

public double getEmpSal() {

return empSal;

}

public void setEmpSal(double empSal) { this.empSal = empSal;

}

public Address getAddr() {

return addr;

}

public void setAddr(Address addr) {

this.addr = addr;

}

@Override

public String toString() {

return "Employee [empId=" + empId + ", empName=" + empName + ", empSal=" + empSal + ", addr=" + addr + "]";

}

}

package com.app.model;

public class Address {

private int addrId;

private String loc;

public Address() {

super();

}

public Address(int addrId, String loc) { super();

this.addrId = addrId;

this.loc = loc;

}

public int getAddrId() {

return addrId;

}

public void setAddrId(int addrId) {

this.addrId = addrId;

}

public String getLoc() {

return loc;

}

public void setLoc(String loc) {

this.loc = loc;

}

@Override

public String toString() {

return "Address [addrId=" + addrId + ", loc=" + loc + "]";

}

}

package com.app.model;

import org.hibernate.Session;

import org.hibernate.SessionFactory; import org.hibernate.Transaction; import org.hibernate.cfg.Configuration;

public class Main {

public static void main(String[] args) {

Configuration cfg=new Configuration().configure();

SessionFactory sf=cfg.buildSessionFactory();

Session ses=sf.openSession(); Transaction tx=ses.beginTransaction(); tx.begin();

Address addr=new Address();

addr.setAddrId(99);

addr.setLoc("hyd");

Employee emp=new Employee();

emp.setEmpId(50);

emp.setEmpName("viswa");

emp.setEmpSal(36000);

emp.setAddr(addr);

System.out.println(emp);

ses.save(emp);

tx.commit();

ses.close();

}

}

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

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd"> <hibernate-mapping>

<class name="com.app.model.Employee" table="empModel"> <id name="empId" column="eid"/>

<property name="empName" column="ename"/> <property name="empSal" column="esal"/>

<many-to-one name="addr" column="aidpk" class="com.app.model.Address" cascade="all"/>

</class>

<class name="com.app.model.Address" table="addrModel"> <id name="addrId" column="aid"/> <property name="loc" column="loc"/>

</class>

</hibernate-mapping>

<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property

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

operty>

<property

name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:XE</propert

y>

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

name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property> <property name="show\_sql">true</property>

<property name="format\_sql">true</property> <property name="hbm2ddl.auto">create</property>

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

</session-factory>

</hibernate-configuration>

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

Employee [empId=50, empName=viswa, empSal=36000.0, addr=Address [addrId=99, loc=hyd]]

Hibernate: select address\_.aid, address\_.loc as loc1\_ from addrModel address\_ where address\_.aid=?

Hibernate: insert into addrModel(loc, aid) values(?, ?)

Hibernate: insert into empModel(ename, esal, aidpk, eid) values(?, ?, ?, ?)

============================================================================ ====================================================================

**Inheritance releation mapping in Hibernate (IS-A) :**

package com.app.model;

public class Person {

private int personId;

private String personName;

public Person() {

super();

}

public Person(int personId, String personName) { super();

this.personId = personId;

this.personName = personName;

}

public int getPersonId() {

return personId;

}

public void setPersonId(int personId) { this.personId = personId;

}

public String getPersonName() {

return personName;

}

public void setPersonName(String personName) { this.personName = personName;

}

@Override

public String toString() {

return "Person [personId=" + personId + ", personName=" +

personName

+ "]";

}

}

package com.app.model;

public class Employee extends Person{

private int empSal;

private String empName;

public Employee() {

super();

}

public Employee(int empSal, String empName) { super();

this.empSal = empSal;

this.empName = empName;

}

public int getEmpSal() {

return empSal;

}

public void setEmpSal(int empSal) {

this.empSal = empSal;

}

public String getEmpName() {

return empName;

}

public void setEmpName(String empName) {

this.empName = empName;

}

@Override

public String toString() {

return "Employee [empSal=" + empSal + ", empName=" + empName +

"]";

}

}

package com.app.model;

public class Student extends Person{

private double stdFee;

private String collegeName;

public Student() {

super();

}

public Student(double stdFee, String collegeName) { super();

this.stdFee = stdFee;

this.collegeName = collegeName;

}

public double getStdFee() {

return stdFee;

}

public void setStdFee(double stdFee) { this.stdFee = stdFee;

}

public String getCollegeName() {

return collegeName;

}

public void setCollegeName(String collegeName) { this.collegeName = collegeName;

}

@Override

public String toString() {

return "Student [stdFee=" + stdFee + ", collegeName=" + collegeName

+ "]";

}

}

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

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd"> <hibernate-mapping>

<class name="com.app.model.Person" table="commantab" discriminator-value="person" >

<id name="personId" column="pid"/> <discriminator column="objType" type="string"/> <property name="personName" column="pname"/>

<subclass name="com.app.model.Employee" discriminator-value="emp"> <property name="empSal" column="empsal"/>

<property name="empName" column="empname"/> </subclass>

<subclass name="com.app.model.Student" discriminator-value="std"> <property name="stdFee" column="sfee"/>

<property name="collegeName" column="cname"/> </subclass>

</class>

</hibernate-mapping>

<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property

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

operty>

<property

name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:XE</propert

y>

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

name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property> <property name="show\_sql">true</property>

<property name="format\_sql">true</property> <property name="hbm2ddl.auto">create</property>

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

</session-factory>

</hibernate-configuration>

package com.app.model;

import org.hibernate.Session;

import org.hibernate.SessionFactory; import org.hibernate.Transaction; import org.hibernate.cfg.Configuration;

public class Main {

public static void main(String[] args) {

Configuration cfg=new Configuration().configure();

SessionFactory sf=cfg.buildSessionFactory();

Session ses=sf.openSession(); Transaction tx=ses.beginTransaction(); tx.begin();

Person p=new Person();

p.setPersonId(10);

p.setPersonName("A");

Employee e=new Employee();

e.setPersonId(20);

e.setPersonName("B");

e.setEmpName("ABCD");

e.setEmpSal(3000);

Student s=new Student();

s.setPersonId(30);

s.setPersonName("C");

s.setStdFee(345.56);

s.setCollegeName("UBIO School");

ses.save(p);

ses.save(e);

ses.save(s);

tx.commit();

ses.close();

}

}

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

|  |  |  |
| --- | --- | --- |
| Hibernate: insert into | | commantab(pname, objType, pid) values(?, 'person', ?) |
| Hibernate: insert | into | commantab(pname, empsal, empname, objType, pid) |
| values(?, ?, ?, 'emp', | | ?) |
| Hibernate: insert | into | commantab(pname, sfee, cname, objType, pid) |
| values(?, ?, ?, 'std', | | ?) |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EDIT | PID | OBJTYPE | | PNAME EMPSAL | | | EMPNAME | | SFEE CNAME |
| Edit | 10 | person | | A | - | - |  | - | - |
| Edit | 20 | emp | B | 3000 | ABCD |  | - | - |  |
| Edit | 30 | std | C | - | - |  | 345.56 | | UBIO School |

package com.app;

public class Employee {

private int empId;

private String empName;

public Employee() {

super();

}

public Employee(int empId, String empName) { super();

this.empId = empId;

this.empName = empName;

}

public int getEmpId() {

return empId;

}

public void setEmpId(int empId) {

this.empId = empId;

}

public String getEmpName() {

return empName;

}

public void setEmpName(String empName) {

this.empName = empName;

}

@Override

public String toString() {

return "Employee [empId=" + empId + ", empName=" + empName + "]";

}

}

package com.app;

public class Manager extends Employee{

private int projectVersion;

private String projectStatus;

public Manager() {

super();

}

public Manager(int projectVersion, String projectStatus) { super();

this.projectVersion = projectVersion;

this.projectStatus = projectStatus;

}

public int getProjectVersion() {

return projectVersion;

}

public void setProjectVersion(int projectVersion) { this.projectVersion = projectVersion;

}

public String getProjectStatus() {

return projectStatus;

}

public void setProjectStatus(String projectStatus) { this.projectStatus = projectStatus;

}

@Override

public String toString() {

return "Manager [projectVersion=" + projectVersion + ", projectStatus="

+ projectStatus + "]";

}

}

package com.app;

public class Developer extends Employee{

private String tech;

private String tools;

public Developer() {

super();

}

public Developer(String tech, String tools) { super();

this.tech = tech;

this.tools = tools;

}

public String getTech() {

return tech;

}

public void setTech(String tech) {

this.tech = tech;

}

public String getTools() {

return tools;

}

public void setTools(String tools) {

this.tools = tools;

}

@Override

public String toString() {

return "Developer [tech=" + tech + ", tools=" + tools + "]";

}

}

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

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd"> <hibernate-mapping>

<class name="com.app.Employee" table="commantab1" discriminator-value="emp" >

<id name="empId" column="eid"/>

<discriminator column="objType" type="string"/> <property name="empName" column="ename"/>

<subclass name="com.app.Manager" discriminator-value="manager"> <property name="projectVersion" column="pversion"/> <property name="projectStatus" column="pstatus"/>

</subclass>

<subclass name="com.app.Developer" discriminator-value="developer"> <property name="tech" column="tech"/>

<property name="tools" column="tools"/> </subclass>

</class>

</hibernate-mapping>

<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property

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

operty>

<property

name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:XE</propert

y>

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

name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property> <property name="show\_sql">true</property>

<property name="format\_sql">true</property> <property name="hbm2ddl.auto">create</property>

<mapping resource = "employee.hbm.xml"/> </session-factory>

</hibernate-configuration>

package com.app;

import org.hibernate.Session;

import org.hibernate.SessionFactory; import org.hibernate.Transaction; import org.hibernate.cfg.Configuration;

public class Main {

public static void main(String[] args) {

Configuration cfg=new Configuration().configure();

SessionFactory sf=cfg.buildSessionFactory();

Session ses=sf.openSession(); Transaction tx=ses.beginTransaction(); tx.begin();

Employee e=new Employee();

e.setEmpId(101);

e.setEmpName("Madhu");

Manager m=new Manager();

m.setEmpId(102);

m.setEmpName("Ramesh");

m.setProjectStatus("ABC");

m.setProjectVersion(3);

Developer d=new Developer();

d.setEmpId(103);

d.setEmpName("Raju");

d.setTech("java");

d.setTools("eclipse");

ses.save(m);

ses.save(e);

ses.save(d);

tx.commit();

ses.close();

}

}

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

Hibernate:

insert

into

commantab1

(ename, pversion, pstatus, objType, eid)

values

(?, ?, ?, 'manager', ?)

Hibernate:

insert

into

commantab1

(ename, objType, eid)

values

(?, 'emp', ?)

Hibernate:

insert

into

commantab1

(ename, tech, tools, objType, eid)

values

(?, ?, ?, 'developer', ?)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| EDIT | EID | OBJTYPE | ENAME | PVERSION | PSTATUS | TECH | TOOLS |
| Edit | 102 | manager | Ramesh | 3 | ABC | - | - |
| Edit | 101 | emp | Madhu | - | - | - | - |
| Edit | 103 | developer | Raju | - | - | java | eclipse |

=========================ALL THE BEST==================================