Introduction

Hibernate :

Hibernate is ORM solution for Java Applications.
Hibernate is ORM tool given to the transfer the data between a java(object)application and a database(Relational) in the form of the objects. Hibernate is the open source, light weight tool given by **Gavin King**.
Hibernate is non-invasive framework, which means it doesn't forces the programmer to extend or implement any class or interface.
Hibernate can run with-in or with-out server, hence Hibernate is suitable for both stand-alone applications(desktop application) and web-applications.
Hibernate is purely meant for persistence.

Persistence : The process of storing enterprise data in to relational database.

JDBC Draw Backs

Agenda :

Draw Backs of JDBC :

In JDBC, to establish a simple database connection also, we need to write that code inside try block. And if any exceptions raised then we need to handle those exceptions inside catch block. Here try and catch blocks are mandatory.
After performing our required task, as a programmer we must close the connection manually, otherwise we may get out of connection error at runtime. Closing the connection and other resource-reallocating code we need to write inside finally block.

If we are not closing the connection manually then JDBC is not responsible to close the connection automatically.
After developing the JDBC program, if there is any changes occurred in the database table structure, then our JDBC program doesn't work because we will write static SQL Queries inside program.
So we need to rewrite the SQL commands according to the latest table structure. The re-compilation and re-deployment is mandatory.
Because of static SQL queries, JDBC will make our applications as database dependent. If we change the database then our applications doesn't work.
In JDBC, while retrieving the data from database, we will get that data in the form of ResultSet. Programmer is responsible to cover the data from ResultSet into our required objects.
JDBC will generate database specific exceptions and errors which are not familiar to java programmers.
In an Enterprise application, the data flow from one layer to another layer will be in the form of objects, but finally while transferring the object(data) from Data-Access layer to database, that object has to convert into text. Because JDBC cannot transfer the objects directly into the database.

Hibernate Advantages

Agenda

Advantages of Hibernate :

Hibernate persists java objects directly into the database.
Hibernate generates efficient queries at runtime.

Hibernate has its own query language called as HQL which is database independent.
Hibernate applications are database independent.
Hibernate supports implicit cashing mechanism.
Hibernate supports collections like List, Set, Map.
Hibernate have translators which will convert checked exceptions into unchecked exceptions. So that try, catch blocks are not required.
Hibernate have different types of algorithms to generate primary key implicitly , while storing the objects in to database.

Enterprise Application

Agenda

Enterprise is nothing but a business organization.
Business Organizations will provide their services to customers.
If any computer application that is used to computerize these business services is called as Enterprise Application.

Ex : Banking Application , Hospital management etc.,

To atomize these services of Business Organizations, we have to perform 4 common tasks in every enterprise application irrespective of the domain.

Persisting (storing) the enterprise data in a secured and easily retrieval manner.
Accessing stored data from the persistent storage.
Processing of data according to the enterprise rules.
Providing interaction to the end-user.

To perform these tasks industry is following Layered architecture.

According to this architecture each task in divided into 4 layers.

Data layer
Data-Access layer
Service layer
Presentation layer

If we use java technologies and java based frameworks to develop enterprise applications then it's called as java based enterprise application.
Java technologies and java application frameworks used in java based enterprise applications.

Presentation layer : Servlets, Jsp, Java Beans, JSF, Struts, Spring MVC, Wicket, Tapestry
Service layer : EJB, Spring
Data-Access layer : JDBC, JPA, Hibernate, Ibatis, Toplink, JDO

Data layer :

Collection of relational tables of an enterprise application is a Data layer.
Data layer is nothing but Data-base.
Almost always RDBMS acts as Data-Base.

Ex : Oracle , DB2, MS SQL, MySQL etc., Data Base professionals will develop this Data layer. Data Access layer will communicate with this Data layer and perform CURD operations with the Database.

Data-Access layer :

A collection of classes whose methods will perform CURD operations with the Database is known as Data-Access layer.
It will receive the request from Service layer.

Based on the request, it communicates with Data layer and performs CURD operations.
After performing CURD operations it will return appropriate response to Service layer.

Service layer :

A collection of classes whose methods performs data processing according to the business rules of an enterprise is known as Service layer.

For each service provided by the enterprise, functionality is implemented in this layer and hence it is called as Service layer.

It will receive the request from Presentation layer.
If required, it will communicate with the Data-Access layer and get the response.
Based on the response it will process the data according to the business rules.
Finally it will give the appropriate response to the Presentation layer.

Presentation layer :

A part of enterprise application which deals with providing interaction to end-user is nothing but presentation layer.

It will take the form data and validate the data according to business requirement.
It will send the request to Service layer.
Based on the response it will generate the response page for the end-user.

Hibernate Application Requirements

Agenda

POJO class
Mapping XML
Configuration XML
Client application to write our logic

Hibernate Application Requirements :

To develop a simple hibernate application we required 4 files.

POJO class
Mapping XML
Configuration XML
Client application to write our logic

These are minimum requirement to run any Hibernate application, and in fact we may require any number of POJO classes and any number of mapping xml files
(Number of POJO classes = that many number of mapping XMLs), and only one configuration xml and finally one java file to write our logic.

POJO class :

POJO is an acronym for Plain Old Java Object. The name is used to emphasize that a given object is an ordinary Java Object, not a special object.

The term "POJO" is mainly used to denote a java object which does not follow any of the major java object models, conventions, or frameworks.

A POJO is a java object not bound by any restriction other than those forced by the java language specification. i.e., a POJO **should not** have to

Extend pre-specified classes, as in

```
Ex :

public class Google extends
       javax.servlet.http.HttpServlet { .. }
```

Implement pre-specified interfaces, as in

```
Ex :

public class Bing extends
       javax.ejb.EntityBean { .. }
```

Contain pre-specified annotations , as in

```
Ex :

@javax.persistence.Entity public class Yahoo{ .. }
```

However, due to technical difficulties and other reasons , many software products or frameworks described as POJO - compliant actually still required the use of pre-specified annotations for features such as persistence to work property.

JavaBeans :

JavaBeans are reusable software components for java.
Practically, they are classes written in the java programming language conforming to a particular convention.
They are used to encapsulate many objects into a single object(the bean), so that they can be passed around as a single bean object instead of as multiple individual objects.

A JavaBean is a java object that is serializable, has a zero-argument constructor, and allows access to properties using getter and setter method.

Mapping and Configuration files in Hibernate :

Mapping :

Every ORM tool needs this mapping. Mapping is the mechanism of placing object properties into columns of a table.
Mapping can be given to an ORM tool either in the form of an XML or in the form of the annotations.
The mapping file contains mapping from a pojo class name to a table name and pojo class variable names to table column names.
While writing a hibernate application, we can construct one or more mapping files, mean hibernate application can contain any number of mapping files.

Generally an object contains 3 properties like

Identity (Object Name)
State (Object values)
Behavior (Object Methods)

But while storing an object into the database, we need to store only the values (State), But how to avoid identity, behaviour ? In order to inform what values of an object have to be stored in what column of the table, will be taking care by the mapping.

Mapping can be done using 2 ways :

XML
Annotations

Note : annotations are introduced into java from JDK 1.5

Syntax of mapping xml :

```
<hibernate-mapping>
<class-name name="POJO-class-name" table="table-name-in-
database">
 <id name="variable-name" column="column-name-in-database"/>
 <property name="variable-name" column="column-name-in-
database"/>
 <property name="variable-name" column="column-name-in-
database"/>
</class-name>
</hibernate-mapping>
```

Configuration :

Configuration is the file loaded into a hibernate application when working with hibernate, this configuration file contains 3 types of information.

Connection Properties
Hibernate Properties
Mapping file names

Note : We must create one configuration file for each database we are going to use, suppose if we want to connect with 2 databases, like Oracle, MySql then we must create 2 configuration files.

Syntax of Configuration xml :

```
<hibernate-configuration>

<session-factory>

<!-- Related to the connection properties -->
<property
name="dialect">org.hibernate.dialect.Oracle9Dialect</propert
y>
```

```
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</p
roperty>
<property name="connection.username">lms</property>
<property name="connection.password">scott</property>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriv
er</property>
<property
name="myeclipse.connection.profile">myJdbcDriver</property>

<!-- Related to the hibernate properties -->
<property name="show_sql">true/false</property>
<property name="dialect">Database dialect</property>
<property name="hbm2ddl.auto">create/update or
whatever</property>

<!-- Related to mapping -->
<mapping resource="hbm file1.xml" />
<mapping resource="hbm file2.xml" />

</session-factory>

</hibernate-configuration>
```

Example on CURD operations :

Pojo class :

```
package com.beans;

public class Employee {
private long employeeId;
private String employeeName;

public long getEmployeeId() {
     return employeeId;
}
public void setEmployeeId(long employeeId){
     this.employeeId = employeeId;
}
public String getEmployeeName(){
     return employeeName;
}
```

```
public void setEmployeeName(String employeeName){
     this.employeeName = employeeName;
}

}
```

Mapping xml for POJO :

Employee.hbm.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">

<!-- Related to hibernate mapping for POJO -->

<hibernate-mapping>

<class name="com.beans.Employee" table="EMPLOYEEDB" >
  <id name="employeeId" column="EID" />
  <property name="employeeName" column="ENAME" length="15"
/>
</class>

</hibernate-mapping>
```

Note : If the database column name is same with java bean property name then we can ignore the column.

Configuration XML :

hibernate.cfg.xml

```
<?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>
```

```
<!-- Related to the connection properties -->

<property
name="myeclipse.connection.profile">myJdbcDriver</property>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriv
er</property>
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</p
roperty>
<property name="connection.username">lms</property>
<property name="connection.password">scott</property>


<!-- Related to the hibernate properties -->
<property
name="dialect">org.hibernate.dialect.Oracle9Dialect</propert
y>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">create</property>

<!-- Related to hibernate mapping -->
<mapping resource="com/hiber/Employee.hbm.xml" />

</session-factory>

</hibernate-configuration>
```

Client Application 1 :

```
package com.client;

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

import com.beans.Employee;

public class ClientAppOne {

public static void main(String ar[]){
```

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

Employee e=new Employee();
e.setEmployeeId(1l);
e.setEmployeeName("Charan");
session.save(e);

tx.commit();

System.out.println("Success");

}
}
```

Client Application 2 :

Note :
Before running Client Application 2 , change the value of hbm2dll.auto to 'update' in hibernate.cfg.xml file .

Otherwise it will drop the table and recreate the new tables so you may get NullPointerException.

```
package com.client;

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

import com.beans.Employee;

public class ClientAppTwo {

public static void main(String ar[]){
```

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

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


Employee e=(Employee)session.get(Employee.class, 11);
System.out.println(e.getEmployeeId());
System.out.println(e.getEmployeeName());

System.out.println("---------");

}
}
```

Client Application 3 :

```
package com.client;

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

import com.beans.Employee;

public class ClientAppThree {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

Employee e=(Employee)session.get(Employee.class, 11);
e.setEmployeeName("Akshay");
session.update(e);

tx.commit();
```

```
System.out.println("update");

}
}
```

Client Application 4 :

```
package com.client;

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

import com.beans.Employee;

public class ClientAppDelete {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

Employee e=(Employee)session.get(Employee.class, 11);
session.delete(e);

tx.commit();

System.out.println("delete");

}
}
```

Hibernate

Agenda

Life cycle of POJO class
Transient state
Persistent state
Detached state

Life cycle of POJO class :

Our POJO class have 3 states.

Transient state
Persistent state
Detached state

1. Transient state

A newly created POJO class object is said to be in Transient state.

A Transient state object is not associated with the session.
Whenever POJO class object is in Transient state, it doesn't represent any record in the database.
Any modifications done to the Transient state object doesn't reflect in the database.

We have 2 approaches to move Transient state object to Persistent state object.

Save the Transient state object by calling
either **save()** or **persist()** method.
Load the object from the database by calling
either **load()** or **get()** metod.

2. Persistent state

Persistent state object is associated with the session. Whenever POJO class object is in Persistent state, then it represents one record in the database.
Whenever POJO class object is in Persistent state, then object is directly synchronized with the database record.
Any modifications done to the object will be directly reflect to the record in the database whenever we commit the transaction.
To move Persistent state object to Detached state, we need to remove the object from the session scope by calling **evict()** method, or we have to clear or close the session by calling
either **clear()** or **close()** method.
To move Persistent state object to Persistent state, we need to delete the record from the database by calling **delete()** method.

3. Detached state

Detached state object is not associated with the session but it represents the record in the database.
To move the Detached state object to Persistent state, we need to call update() method.

Note : POJO class, Mapping and Configuration files are same as above application.

Client Application 1 :

```
package com.client;

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

import com.beans.Employee;
```

```
public class ClientAppOne {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

// transient state
Employee e=new Employee();
e.setEmployeeId(2l);
e.setEmployeeName("charan");

// persistent state
session.save(e);
tx.commit();

System.out.println("---------");

}
}
```

Client Application 2 :

```
package com.client;

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

import com.beans.Employee;

public class ClientAppTwo {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
```

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

//persistent state by loading object
Employee e=(Employee)session.get(Employee.class, 21);
System.out.println(e.getEmployeeName());

//modifying employeeName
e.setEmployeeName("Akshay");

//not required to call update() method explicitly
// i.e., session.update(e);
tx.commit();

System.out.println("update");

}
}
```

Client Application 3 :

```
package com.client;

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

import com.beans.Employee;

public class ClientAppTwo {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

//persistent state by loading object
Employee e=(Employee)session.get(Employee.class, 21);
```

```
System.out.println("Object is in Transient state");

//detached state
session.evict(e);
System.out.println("Object is moved to detached state");

//modifying employeeName
e.setEmployeeName("Arun");

/* Modifications will not reflect with the database
 *  because Object is in Detached state.
 * If we want to save modified data to the database
 *  then we need to call update() explicitly
 */

//session.update(e);
tx.commit();

System.out.println("update");

}
}
```

Client Application 4 :

```
package com.client;

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

import com.beans.Employee;

public class ClientAppTwo {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();
```

```
//persistent state by loading object
Employee e=(Employee)session.get(Employee.class, 21);

//transient state
session.delete(e);
tx.commit();

System.out.println("delete");

}
}
```

Generators

Agenda

Generator classes
assigned
sequence
increment
identity
hilo
native
foreign

Generators :

Hibernate supports different types of algorithms for generating primary keys. For each algorithm, there is a separate class.
To let hibernate, generate the primary keys then we need to use <generator> class inside mapping file.
If we are not mentioning this tag then hibernate by default will assume Assigned as a generator.

In case of Assigned generator, programmer is responsible to assign the id value.
Except Assigned generator for all other generators, hibernate is responsible to generate primary key values.
Hibernate provides different primary key generator classes and all these classes are implemented **org.hibernate.id.IdentifierGenerator**
If we want to pass any values to generator then we need to use <param> tag.

Syntax :

```
<generator class="generator-class-name">
  <param name="parameter-name"> value </param>
</generator>
```

Generator classes :

assigned
sequence
increment
identity
hilo
native
foreign

assigned :

assigned is default generator.
We if didn't mention any generator then hibernate takes assigned as a generator.
In this case, programmer is responsible to give primary key value.
This generator supports in all databases.

sequence :

sequence is database dependent. MySql doesn't support sequence. Hibernate is responsible to generate id value. Even if programmer passes the id value, but hibernate won't take the programmer given value.
Hibernate will execute the sequence and assigns the value to the object when inserting a new record.
We need to pass our sequence to <generator> class.

Syntax :

```
<id name="employeeId" column="EID">
 <generator class="sequence">
  <param name="sequence">SequenceName</param>
 </generator>
</id>
```

If we didn't pass any sequence then hibernate will take default sequence. The default sequence taken by hibernate is hibernate_sequence.

Example :

Employee.java

```
package com.beans;

public class Employee {
private long employeeId;
private String employeeName;
private double salary;

public long getEmployeeId() {
     return employeeId;
}
public void setEmployeeId(long employeeId){
     this.employeeId = employeeId;
}
public String getEmployeeName(){
```

```
     return employeeName;
}
public void setEmployeeName(String employeeName){
     this.employeeName = employeeName;

}
public double getSalary() {
     return salary;
}
public void setSalary(double salary) {
     this.salary = salary;
}
}
```

Employee.hbm.xml

```
<hibernate-mapping>

<class name="com.beans.Employee" table="EMPLOYEEDB" >
  <id name="employeeId" column="EID">
    <generator class="sequence" />
  </id>
  <property name="employeeName" column="ENAME" length="30"/>
  <property name="salary" column="ESAL" />
</class>

</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>

<session-factory>

<!-- Related to the connection properties -->

<property
name="myeclipse.connection.profile">myJdbcDriver</property>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriv
er</property>
```

```
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</p
roperty>
<property name="connection.username">lms</property>
<property name="connection.password">scott</property>


<!-- Related to the hibernate properties -->
<property
name="dialect">org.hibernate.dialect.Oracle9Dialect</propert
y>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<!-- Related to hibernate mapping -->
<mapping resource="com/hiber/Employee.hbm.xml" />

</session-factory>

</hibernate-configuration>
```

Client Application :

```
import org.hibernate.cfg.Configuration;

import com.beans.Employee;

public class ClientAppOne {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

Employee e=new Employee();
e.setEmployeeName("charan");
e.setSalary(5000);

session.save(e);
tx.commit();
```

```
System.out.println("success");

}
}
```

increment :

Increment generator is database independent.
Increment generator will generate id value based on formula **Max of id value + 1**
It there are no records in database, then increment will generate 1 as an id value.

Note : POJO class, Mapping and Configuration files are same as above application.

Employee.hbm.xml

```
<!-- Related to hibernate mapping for POJO -->

<hibernate-mapping>

<class name="com.beans.Employee" table="EMPLOYEEDB" >
  <id name="employeeId" column="EID">
    <generator class="increment" />
  </id>
  <property name="employeeName" column="ENAME" length="30"/>
  <property name="salary" column="ESAL" />
</class>

</hibernate-mapping>
```

identity :

Identity is database dependent. Oracle database doesn't support identity generator.
Identity generator is similar to increment generator, but the difference is, increment generator is database independent and hibernate uses a select operation for selecting max of id before inserting new record.

But in case of identity, no select operation will be performed to insert id value for new record by hibernate.
Here id value is generated by the database but not by hibernate.

Syntax :

```
<id name="employeeId" column="EID">
  <generator class="identity" />
</id>
```

Note : POJO class, Mapping and Configuration files are same as above application.

Oracle doesn't support identity. So we will get exception if we use oracle database.

Employee.hbm.xml

```
<hibernate-mapping>

<class name="com.beans.Employee" table="EMPLOYEEDB" >
  <id name="employeeId" column="EID">
    <generator class="identity" />
  </id>
  <property name="employeeName" column="ENAME" length="30"/>
  <property name="salary" column="ESAL" />
</class>

</hibernate-mapping>
```

hilo :

hilo is database independent.
hilo will insert 1 as a id value for first record. For the second record it will insert 32768 as id value. And from next record it will increment the old id value by 32768 and stores the new value.

To store the count of id values, hibernate will use a separate table known as hibernate_unique_value. default column name in this table is next_hi.
To create our own table and column, we need to pass 2 parameters for hilo generator.

Syntax :

```
<id name="employeeId" column="EID">
 <generator class="hilo">
  <param name="table">table name</param>
  <param name="column">column name</param>
 </generator>
</id>
```

Note : POJO class, Mapping and Configuration files are same as above application.

Employee.hbm.xml

```
<hibernate-mapping>

<class name="com.beans.Employee" table="EMPLOYEEDB" >
  <id name="employeeId" column="EID">
    <generator class="hilo" />
  </id>
  <property name="employeeName" column="ENAME" length="30"/>
  <property name="salary" column="ESAL" />
</class>

</hibernate-mapping>
```

native :

native doesn't have a separate generator class. It just uses other generator classes. **identity, sequence, hilo**

First it will check whether the database supports identity or not.
If database not support identity then it will go for sequence.
If database not support sequence then it will go for hilo.

Note : POJO class, Mapping and Configuration files are same as above application.

Employee.hbm.xml

```
<hibernate-mapping>

<class name="com.beans.Employee" table="EMPLOYEEDB" >
  <id name="employeeId" column="EID">
    <generator class="native"/>
  </id>
  <property name="employeeName" column="ENAME" length="30"/>
  <property name="salary" column="ESAL" />
</class>

</hibernate-mapping>
```

foreign :

foreign generator is used to one-to-one relationship.

Hibernate Inheritance

Agenda :

Table per class
Table per sub-class
Table per concrete-class

Hibernate Inheritance :

Hibernate supports 3 types of Inheritance

Table per class
Table per sub-class
Table per concrete-class

In hibernate inheritance hierarchy if we save derived class object then base class object also will be stored into the database automatically.

Consider 3 POJO classes, Employee class as a base class and SalEmployee, HourEmployee as derived classes.
For all POJO classes we have only one mapping XML file which is nothing but Base class mapping file.
Both derived classes data has to map in base class mapping file only.

Table per class :

In Table per class hierarchy, both base class and derived class data will be stored into a single table which is base class related table.
For all 3 POJO classes we have single mapping file which is for base (Employee) class.
To map derived classes details into base class mapping file , we need to use <subclass> tag.
Here if we save either HourEmployee or SalEmployee then automatically Employee object also will be stored into database. Both derived class object (HourEmployee/SalEmployee) and base class object(Employee) will be stored into Employee related table only.
In Table per class hierarchy, we need to use one special column in the database table which is known as discriminator.
Discriminator column will help us to identify which derived class object is stored along with base class object.

Employee.java

```
package com.beans;

public class Employee {
private long employeeId;
private String employeeName;
```

```
public long getEmployeeId() {
     return employeeId;
}
public void setEmployeeId(long employeeId){
     this.employeeId = employeeId;
}
public String getEmployeeName(){
     return employeeName;
}
public void setEmployeeName(String employeeName){
     this.employeeName = employeeName;

}

}
```

SalEmployee.java

```
package com.beans;

public class SalEmployee extends Employee{
private double salary;

public double getSalary() {
     return salary;
}
public void setSalary(double salary) {
     this.salary = salary;
}

}
```

HoursEmployee.java

```
package com.beans;

public class HourEmployee extends Employee{

private int workHours;

public int getWorkHours() {
     return workHours;
```

```
}

public void setWorkHours(int workHours) {
     this.workHours = workHours;
}

}
```

Employee.hbm.xml

```
<hibernate-mapping>

<class name="com.beans.Employee" table="EMPLOYEEDB" >
  <id name="employeeId" column="EID"/>
  <discriminator column="discolumn" type="string"
length="15"/>
  <property name="employeeName" column="ENAME" length="30"/>

  <subclass name="com.beans.SalEmployee" discriminator-
value="se">
    <property name="salary" column="ESAL"/>
  </subclass>

  <subclass name="com.beans.HourEmployee" discriminator-
value="he">
  <property name="workHours" column="hours"/>
  </subclass>

</class>

</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>

<session-factory>

<!-- Related to the connection properties -->

<property
name="myeclipse.connection.profile">myJdbcDriver</property>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriv
er</property>
```

```
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</p
roperty>
<property name="connection.username">lms</property>
<property name="connection.password">scott</property>


<!-- Related to the hibernate properties -->
<property
name="dialect">org.hibernate.dialect.Oracle9Dialect</propert
y>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">create</property>

<!-- Related to hibernate mapping -->
<mapping resource="com/hiber/Employee.hbm.xml" />

</session-factory>

</hibernate-configuration>
```

Client Application :

```
package com.client;

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

import com.beans.HourEmployee;
import com.beans.SalEmployee;

public class ClientApp{

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();
```

```
SalEmployee sal=new SalEmployee();
sal.setEmployeeId(10L);
sal.setEmployeeName("charan");
sal.setSalary(5000);

HourEmployee hour=new HourEmployee();
hour.setEmployeeId(11L);
hour.setEmployeeName("Akshay");
hour.setWorkHours(6000);

session.save(sal);
session.save(hour);

tx.commit();

System.out.println("success");

}
}
```

Table per sub-class :

In Table per class, both derived class object and base class object are stored in base class related table only, which means for all POJO classes we have single table.
But in Table per sub-class, per each class we have a separate table(including subclass also).
X number of classes = X number of tables .
In Table per sub-class also for all POJO classes we have one mapping file which is base classes mapping file. Derived class details we have to map in base class mapping file only.
To map derived class details inside base class mapping file, we use <joined-subclass> tag.

In Table per sub-class, discriminator column not required because each class data is stored in separate table. But we need to take one <key> tag, inside <joined-subclass> tag.

Note : POJO class, Mapping and Configuration files are same as above application.

Employee.hbm.xml

```
<hibernate-mapping>

<class name="com.beans.Employee" table="EMPLOYEEDB" >
  <id name="employeeId" column="EID"/>
  <property name="employeeName" column="ENAME" length="30"/>

  <joined-subclass name="com.beans.SalEmployee"
table="SalEmployeeDB">
   <key column="refEID"/>
   <property name="salary" column="ESAL"/>
  </joined-subclass>

  <joined-subclass name="com.beans.HourEmployee"
table="HourEmployeeDB">
   <key column="refEID"/>
   <property name="workHours" column="HOURS"/>
  </joined-subclass>
</class>

</hibernate-mapping>
```

Client Application :

```
package com.client;

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

import com.beans.HourEmployee;
import com.beans.SalEmployee;
```

```
public class ClientApp {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

SalEmployee sal=new SalEmployee();
sal.setEmployeeId(10L);
sal.setEmployeeName("charan");
sal.setSalary(5000);

HourEmployee hour=new HourEmployee();
hour.setEmployeeId(11L);
hour.setEmployeeName("Akshay");
hour.setWorkHours(6000);

session.save(sal);
session.save(hour);

tx.commit();

System.out.println("success");

}
}
```

Table per concrete-class :

In Table per concrete class, we have tables only for derived classes.
X number of derived classes = X number of tables
Here Base class data also will be saved into derived class related table.
One mapping file for all POJO classes. Here also we need to take mapping file only for base class.

To map derived class details we need to take <union-subclass> tag inside base class related mapping file.

Note : POJO class, Mapping and Configuration files are same as above application.

Employee.hbm.xml

```
<hibernate-mapping>

<class name="com.beans.Employee">
  <id name="employeeId" column="EID"/>
  <property name="employeeName" column="ENAME" length="30"/>

  <union-subclass name="com.beans.SalEmployee"
table="SALEMPLOYEEDB">
   <property name="salary" column="ESAL"/>
  </union-subclass>

    <union-subclass name="com.beans.HourEmployee"
table="HOUREMPLOYEEDB">
   <property name="workHours" column="HOURS"/>
  </union-subclass>
</class>

</hibernate-mapping>
```

Client Application :

```
package com.client;

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

import com.beans.HourEmployee;
import com.beans.SalEmployee;

public class ClientAppOne {

public static void main(String ar[]){
```

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

SalEmployee sal=new SalEmployee();
sal.setEmployeeId(10L);
sal.setEmployeeName("charan");
sal.setSalary(5000);

HourEmployee hour=new HourEmployee();
hour.setEmployeeId(11L);
hour.setEmployeeName("Akshay");
hour.setWorkHours(6000);

session.save(sal);
session.save(hour);

tx.commit();

System.out.println("success");

}
}
```

Hibernate Query Language (HQL)

Agenda :

Hibernate Query Language (HQL) :

HQL is used to perform bulk operations. HQL is database independent.
An Object oriented from of SQL is nothing but HQL.
In HQL we will replace table names with POJO class names, and column names with POJO class variable names.

HQL Command to retrieve complete object :

SQL command for retrieving complete record from the database is

```
select * from employees
  // employees is table name in database
```

Using HQL if we want to retrieve complete object from the database then we need to replace class name in the place of table name, and reference variable name in the place of * symbol.

```
select e from Employee e

//here employee is a pojo class name, not a table name
,
// and 'e' is a reference  variable for Employee class.
```

We can start the command directly with from keyword also.

```
.    from Employee
```

HQL Command to retrieve partial object :

SQL command for retrieving partial object is

```
select empname, sal from employees

//empname and sal are column names
// and employees is table name in database
```

Using HQL, if we want to load partial object then we need to replace column names with pojo class variable names.

```
select e.employeeName, e.salary from Employee e

//Here employeeName and salary are pojo class variable names
// and Employee is a pojo class
```

Executing HQL Command :

In order to execute HQL command first we need to create Query object. Query is an interface and QueryImpl is its implemented class.
To get Query object, we need to call createQuery() method on Session interface.
On the query object call list() method, list() method will return java.util.List.

We need to iterate that list to get the required objects

Ex :

```
Query query=session.createQuery("HQL Command");
List list=query.list();
Iterator iterator=list.iterator();

while(iterator.hasNext()){
 Employee e=(Employee)iterator.next();
}
```

Example on selecting complete object :

While retrieving the complete object, hibernate internally converts each row of table into an object of pojo class type and hibernate stores all these pojo classes into List.
So while iterating the object inside while loop, we need to typecast with pojo class type only.

Employee.java

```
package com.beans;

public class Employee {
private long employeeId;
private String employeeName;
private double salary;


public long getEmployeeId() {
     return employeeId;
}
public void setEmployeeId(long employeeId){
     this.employeeId = employeeId;
}
public String getEmployeeName(){
     return employeeName;
}
public void setEmployeeName(String employeeName){
     this.employeeName = employeeName;
}
public double getSalary() {
     return salary;
}
public void setSalary(double salary) {
     this.salary = salary;
}

}
```

Employee.hbm.xml

```
<hibernate-mapping>

<class name="com.beans.Employee" table="EMPLOYEEDB">
  <id name="employeeId" column="EID"/>
  <property name="employeeName" column="ENAME" length="30"/>
  <property name="salary" column="ESAL"/>
</class>

</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>
```

```
<session-factory>

<!-- Related to the connection properties -->

<property
name="myeclipse.connection.profile">myJdbcDriver</property>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriv
er</property>
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</p
roperty>
<property name="connection.username">lms</property>
<property name="connection.password">scott</property>


<!-- Related to the hibernate properties -->
<property
name="dialect">org.hibernate.dialect.Oracle9Dialect</propert
y>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<!-- Related to hibernate mapping -->
<mapping resource="com/hiber/Employee.hbm.xml" />

</session-factory>

</hibernate-configuration>
```

Client Application 1 :

```
package com.client;

import java.util.Iterator;
import java.util.List;

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

import com.beans.Employee;
```

```
public class ClientAppOne {

public static void main(String ar[]){

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

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

String hqlQuery="from Employee";

Query query=session.createQuery(hqlQuery);
List list=query.list();
Iterator iterator=list.iterator();

while(iterator.hasNext()){
Employee e=(Employee)iterator.next();
System.out.println(e.getEmployeeId());
System.out.println(e.getEmployeeName());
System.out.println(e.getSalary());
System.out.println("----------");
}

}
}
```

Example on selecting partial object with single column :

While retrieving the object with single column, hibernate internally creates an object of that particular value column value type and stores all these objects into List.
So while iterating the list, we need to type cast with that particular column value type only.

Note : Employee class, mapping file and configuration files are same as above application.

Client Application 2 :

```
package com.client;

import java.util.Iterator;
import java.util.List;

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

public class ClientApp {

public static void main(String ar[]){

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

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

String hqlQuery="select e.salary from Employee e";

Query query=session.createQuery(hqlQuery);
List list=query.list();
Iterator iterator=list.iterator();

while(iterator.hasNext()){
Double sal=(Double)iterator.next();
System.out.println(sal);
System.out.println("success");
}

}
}
```

Example on selecting partial object with more than one column :

While retrieving the object with more than one column, hibernate internally stores these multiple column values into an object array and stores that object array into List.
So while iterating the object, we need to type cast with object array.

Ex : Employee class, mapping file and configuration files are same as above application.

Client Application 3 :

```
package com.client;

import java.util.Iterator;
import java.util.List;

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

public class ClientApp {

public static void main(String ar[]){

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

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

String hqlQuery="select e.employeeName,e.salary from
Employee e";

Query query=session.createQuery(hqlQuery);
List list=query.list();
Iterator iterator=list.iterator();

while(iterator.hasNext()){
Object[] o=(Object[])iterator.next();
System.out.println(o[0]);
System.out.println(o[1]);
System.out.println("success");
}

}
}
```

Passing runtime values :

If we want to pass runtime values to the query then we have to use either '?'(positional parameters) or label (named parameters) in HQL command.
To set the run time values to the query we need to call setParameter() method on Query object.

Note :

index number of positional parameter will start from zero.
But in JDBC index number for positional parameter will start from 1.
Any name prefixed with **:** symbol is said to be named parameter.

Positional parameter example :

Ex : Employee class, mapping file and configuration files are same as above application.

```
package com.client;

import java.util.Iterator;
import java.util.List;

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

public class ClientApp {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
```

```
String hqlQuery="select e.employeeName,e.salary" +
          "from Employee e where e.employeeId=?";

Query query=session.createQuery(hqlQuery);
query.setParameter(0, 10L);

List list=query.list();
Iterator iterator=list.iterator();

while(iterator.hasNext()){
Object[] o=(Object[])iterator.next();
System.out.println(o[0]);
System.out.println(o[1]);
System.out.println("success");
}

}
}
```

Named parameter example :

Ex : Employee class, mapping file and configuration files are same as above application.

```
package com.client;

import java.util.Iterator;
import java.util.List;

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

public class ClientApp {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
```

```
Session session=sf.openSession();

String hqlQuery="select e.employeeName,e.salary" +
          " from Employee e where e.employeeId=:id";

Query query=session.createQuery(hqlQuery);
query.setParameter("id", 10L);

List list=query.list();
Iterator iterator=list.iterator();

while(iterator.hasNext()){
Object[] values=(Object[])iterator.next();
System.out.println(values[0]);
System.out.println(values[1]);
System.out.println("success");
}

}
}
package com.client;

import java.util.Iterator;
import java.util.List;

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

public class ClientApp {

public static void main(String ar[]){

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

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

String hqlQuery="select e.employeeName,e.salary" +
          " from Employee e where e.salary=:id";
```

```
Query query=session.createQuery(hqlQuery);
query.setParameter("id", new Double(4000));

List list=query.list();
Iterator iterator=list.iterator();

while(iterator.hasNext()){
Object[] values=(Object[])iterator.next();
System.out.println(values[0]);
System.out.println(values[1]);
System.out.println("success");
}

}
}
```

Example for deleting the object using HQL :

Ex : Employee class, mapping file and configuration files are same as above application.

```
package com.client;

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

public class ClientAppDelete {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

String hqlQuery="delete from Employee e where
e.salary=:sal";
```

```
Query query=session.createQuery(hqlQuery);
query.setParameter("sal",5000D);

int i=query.executeUpdate();
tx.commit();

System.out.println("delete records :"+ i);
}
}
```

Example for updating the object using HQL :

Ex : Employee class, mapping file and configuration files are same as above application.

```
package com.client;

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

public class ClientAppUpdate {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

String hqlQuery="update Employee e set e.employeeName=?
where e.employeeId=?";

Query query=session.createQuery(hqlQuery);
query.setParameter(0, "arun");
query.setParameter(1, 11);

int i=query.executeUpdate();
tx.commit();
```

```
System.out.println("update records :"+ i);
}
}
```

Criteria

Agenda

Adding conditions to Criteria
Retrieving partial object with the help of Projections

Criteria :

Using criteria we can only retrieve the objects from the database. We can't perform DML operations.
And with criteria we can retrieve complete object only. We can't retrieve partial object.
To retrieve partial object we have to combine criteria with projections.

To execute Criteria first we need to create Criteria object.
To get Criteria object, we need to call createCriteria() method on Session interface. We need to pass our pojo class to this method.
On the query object call list() method , list() will return java.util.List.

We need to iterate that list to get the required objects

```
Criteria criteria=session.createCriteria(Employee.class);
List list=criteria.list();
Iterator iterator=list.iterator();

while(iterator.hasNext()){
Employee emp=(Employee)iterator.next();
}
```

Example :

Employee.java

```
package com.beans;

public class Employee {
private long employeeId;
private String employeeName;
private double salary;

public long getEmployeeId() {
     return employeeId;
}
public void setEmployeeId(long employeeId){
     this.employeeId = employeeId;
}
public String getEmployeeName(){
     return employeeName;
}
public void setEmployeeName(String employeeName){
     this.employeeName = employeeName;
}
public double getSalary() {
     return salary;
}
public void setSalary(double salary) {
     this.salary = salary;
}

}
```

Employee.hbm.xml

```
<hibernate-mapping>

<class name="com.beans.Employee" table="EMPLOYEEDB">
  <id name="employeeId" column="EID"/>
  <property name="employeeName" column="ENAME" length="30"/>
  <property name="salary" column="ESAL"/>
</class>

</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>

<session-factory>

<!-- Related to the connection properties -->

<property
name="myeclipse.connection.profile">myJdbcDriver</property>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriv
er</property>
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</p
roperty>
<property name="connection.username">lms</property>
<property name="connection.password">scott</property>

<!-- Related to the hibernate properties -->
<property
name="dialect">org.hibernate.dialect.Oracle9Dialect</propert
y>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<!-- Related to hibernate mapping -->
<mapping resource="com/hiber/Employee.hbm.xml" />

</session-factory>

</hibernate-configuration>
```

Client Application 1 :

```
package com.client;

import java.util.Iterator;
import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
```

```
import org.hibernate.cfg.Configuration;

import com.beans.Employee;

public class ClientApp {

public static void main(String ar[]){

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

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

Criteria criteria=session.createCriteria(Employee.class);
List list=criteria.list();
Iterator iterator=list.iterator();

while(iterator.hasNext()){
Employee emp=(Employee)iterator.next();
System.out.println(emp.getEmployeeId());
System.out.println(emp.getEmployeeName());
System.out.println(emp.getSalary());
}

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

}
```

Adding conditions to Criteria :

Using Criteria if we want to add conditions to the query, then we need to create one Criterian object and add this object to Criteria.
To add Criterion object to Criteria, we need to call add() method on criteria object.
To create Criterion object, we need to call static methods of Restrictions class, each method of this class returns Criterion object.

Ex:

```
Criteria criteria=session.createCriteria(Employee.class);
Criterion criterion=Restrictions.gt("salary", new
Double(3000));
 //here salary is pojo class variable name
criteria.add(criterion);
```

Note : POJO class, Mapping and Configuration files are same as above application.

Client Application 2 :

```
package com.client;

import java.util.Iterator;
import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.criterion.Criterion;
import org.hibernate.criterion.Restrictions;

import com.beans.Employee;

public class ClientApp {

public static void main(String ar[]){

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

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

Criteria criteria=session.createCriteria(Employee.class);
Criterion criterion=Restrictions.gt("salary", new
Double(3000));
criteria.add(criterion);

List list=criteria.list();
```

```
Iterator iterator=list.iterator();

while(iterator.hasNext()){
Employee emp=(Employee)iterator.next();
System.out.println(emp.getEmployeeId());
System.out.println(emp.getEmployeeName());
System.out.println(emp.getSalary());
}

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

}
```

Note : POJO class, Mapping and Configuration files are same as above application.

Client Application 3 :

```
package com.client;

import java.util.Iterator;
import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.criterion.Criterion;
import org.hibernate.criterion.Order;
import org.hibernate.criterion.Restrictions;

import com.beans.Employee;

public class ClientApp {

public static void main(String ar[]){

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

```
SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();

Criteria criteria=session.createCriteria(Employee.class);
Criterion criterion=Restrictions.gt("salary", new
Double(3000));
criteria.add(criterion);

Order order=Order.asc("salary");
criteria.addOrder(order);

List list=criteria.list();
Iterator iterator=list.iterator();

while(iterator.hasNext()){
Employee emp=(Employee)iterator.next();
System.out.println(emp.getEmployeeId());
System.out.println(emp.getEmployeeName());
System.out.println(emp.getSalary());
}

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

}
```

Retrieving partial object with the help of Projections :

Using Criteria we can retrieve only complete object, but it is possible to retrieve partial object if we combine Criteria with projects.
Projection is an interface and Projections is a class.
To get Projection object we need to call property() method on Projections class. It is static method and will return Projection object.
To add Projection object to Criteria , we need to call setProjection() method on Criteria object.
To retrieve partial object with single column, we need to create Projection object for that particular column value type.

Note : POJO class, Mapping and Configuration files are same as above application.

Client Application 4 :

```
package com.client;

import java.util.Iterator;
import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.criterion.Criterion;
import org.hibernate.criterion.Order;
import org.hibernate.criterion.Projection;
import org.hibernate.criterion.Projections;
import org.hibernate.criterion.Restrictions;

import com.beans.Employee;

public class ClientApp {

public static void main(String ar[]){

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

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

Criteria criteria=session.createCriteria(Employee.class);
Criterion criterion=Restrictions.gt("salary", new
Double(3000));
criteria.add(criterion);

Order order=Order.asc("salary");
criteria.addOrder(order);

Projection projection=Projections.property("salary");
criteria.setProjection(projection);
```

```
List list=criteria.list();
Iterator iterator=list.iterator();

while(iterator.hasNext()){
Double salary=(Double)iterator.next();
System.out.println(salary);
}

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

}
```

Note : if we add 2 Projection objects to Criteria then 2nd projection will override the first Projection.

To add multiple Projection objects to Criteria, we need to create ProjectionList. To create ProjectionList, we need to call projectionList() on Projections class.
Create multiple Projection objects and add these Projection objects to ProjectionList. Set this ProjectionList to Criteria. To set ProjectionList to Criteria we need to call setProjection() method on Criteria object. Finally while iterating the list, we need to type cast with Object[] because list contains partial object with multiple columns.

Note : POJO class, Mapping and Configuration files are same as above application.

Client Application 5 :

```
package com.client;

import java.util.Iterator;
import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
```

```
import org.hibernate.cfg.Configuration;
import org.hibernate.criterion.Criterion;
import org.hibernate.criterion.Order;
import org.hibernate.criterion.Projection;
import org.hibernate.criterion.ProjectionList;
import org.hibernate.criterion.Projections;
import org.hibernate.criterion.Restrictions;

import com.beans.Employee;

public class ClientApp {

public static void main(String ar[]){

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

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

Criteria criteria=session.createCriteria(Employee.class);
Criterion criterion=Restrictions.gt("salary", new
Double(3000));
criteria.add(criterion);

Order order=Order.asc("salary");
criteria.addOrder(order);

ProjectionList projectionList=Projections.projectionList();

Projection projection1=Projections.property("employeeName");
Projection projection2=Projections.property("salary");

projectionList.add(projection1);
projectionList.add(projection2);

criteria.setProjection(projectionList);

List list=criteria.list();
Iterator iterator=list.iterator();

while(iterator.hasNext()){
Object[] values=(Object[])iterator.next();
```

```
System.out.println(values[0]);
System.out.println(values[1]);
}

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

}
```

Associations

Agenda

Associations :

One to Many mapping :

One pojo class is related with multiple Pojo classes.
In parent pojo class we need to take one collection property. In the mapping file of parent pojo class we need to configure that collection details. Collection can be Set, List, or Map.
While mapping collection details, we need to add **cascade** attribute to transfer the operations on parent object to child objects.
Default value for cascade is none, which means even though relationship is exist, the operations we are doing on parent object will not transfer to child object.

In child class mapping file we need to take <key> tag to configure foreign key column.
Consider 2 pojo classes, Student and Course. For both pojo classes we need to take a separate mapping file. Configure these 2 mapping file details in hibernate.cfg.xml file.

Ex :

Here one student can join multiple courses. In Student class we need to take one collection property for courses and we need to configure these collection details inside Student.hbm.xml file.
In client application, set multiple courses to student object. Finally save student class object, then automatically course class objects also will be stored into the database.

Example : Student.java

```
package com.beans;

import java.util.Set;

public class Student {
private int studentId;
private String stuentName;
private Set courses;

public int getStudentId() {
     return studentId;
}
public void setStudentId(int studentId) {
     this.studentId = studentId;
}
public String getStuentName() {
     return stuentName;
}
public void setStuentName(String stuentName) {
     this.stuentName = stuentName;
}
public Set getCourses() {
```

```
    return courses;
}
public void setCourses(Set courses) {
    this.courses = courses;
}

}
```

Course.java

```
package com.beans;

public class Course {
private int courseId;
private String courseName;

public int getCourseId() {
    return courseId;
}
public void setCourseId(int courseId) {
    this.courseId = courseId;
}
public String getCourseName() {
    return courseName;
}
public void setCourseName(String courseName) {
    this.courseName = courseName;
}

}
```

Student.hbm.xml

```
<hibernate-mapping>

<class name="com.beans.Student" table="STUDENTDB">
  <id name="studentId" column="SID"/>
  <property name="stuentName" column="SNAME" length="15"/>
  <set name="courses" cascade="all">
   <key column="refSID"/>
   <one-to-many class="com.beans.Course"/>
  </set>
</class>

</hibernate-mapping>
```

Course.hbm.cml

```
<hibernate-mapping>

<class name="com.beans.Course" table="COURSEDB">
  <id name="courseId" column="CID"/>
  <property name="courseName" column="CNAME" length="15"/>
</class>

</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>

<session-factory>

<!-- Related to the connection properties -->

<property
name="myeclipse.connection.profile">myJdbcDriver</property>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriv
er</property>
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</p
roperty>
<property name="connection.username">lms</property>
<property name="connection.password">scott</property>


<!-- Related to the hibernate properties -->
<property
name="dialect">org.hibernate.dialect.Oracle9Dialect</propert
y>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">create</property>

<!-- Related to hibernate mapping -->
<mapping resource="com/hiber/Student.hbm.xml" />
<mapping resource="com/hiber/Course.hbm.xml" />

</session-factory>
```

```
</hibernate-configuration>
```

Client Application 1 :

```
package com.client;

import java.util.HashSet;
import java.util.Set;

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

import com.beans.Course;
import com.beans.Student;

public class ClientApp {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

Student student=new Student();
student.setStudentId(1);
student.setStuentName("Ashok");

Course course=new Course();
course.setCourseId(21);
course.setCourseName("hibernate");

Course course2=new Course();
course2.setCourseId(22);
course2.setCourseName("spring");

Set set=new HashSet();
set.add(course);
set.add(course2);
```

```
student.setCourses(set);

session.save(student);
tx.commit();

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

}
```

Client Application 2 :

Note :

```
package com.client;

import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

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

import com.beans.Course;
import com.beans.Student;

public class ClientApp {

public static void main(String ar[]){

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

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

Student student=(Student)session.get(Student.class, 1);
System.out.println(student.getStudentId());
System.out.println(student.getStuentName());

Set set=student.getCourses();
Iterator iterator=set.iterator();
```

```
while(iterator.hasNext()){
Course course=(Course)iterator.next();
System.out.println(course.getCourseId());
System.out.println(course.getCourseName());
}

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

}
```

Client Application 3 :

```
package com.client;

import java.util.Iterator;
import java.util.List;
import java.util.Set;

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

import com.beans.Course;
import com.beans.Student;

public class ClientApp {

public static void main(String ar[]){

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

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

String hqlQuery="from Student";
Query query=session.createQuery(hqlQuery);

List list=query.list();
Iterator iterator=list.iterator();
```

```
while(iterator.hasNext()){
Student student=(Student)iterator.next();
System.out.println(student.getStudentId());
System.out.println(student.getStuentName());

Set set=student.getCourses();
Iterator iterator2=set.iterator();
 while(iterator2.hasNext()){
 Course course=(Course)iterator2.next();
 System.out.println(course.getCourseId());
 System.out.println(course.getCourseName());
 }
}

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

}
```

Client Application 4 :

```
package com.client;

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

import com.beans.Student;

public class ClientApp {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

Student student=(Student)session.get(Student.class, 1);
session.delete(student);
tx.commit();
```

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

}
```

Client Application 5 :

```
package com.client;

import java.util.Iterator;
import java.util.List;

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

import com.beans.Student;

public class ClientApp {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

String hqlQuery="from Student";
Query query=session.createQuery(hqlQuery);

List list=query.list();
Iterator iterator=list.iterator();

while(iterator.hasNext()){
Student student=(Student)iterator.next();
session.delete(student);
}
tx.commit();
System.out.println("success");
}
```

```
}
```

Many to One mapping :

In many to one mapping, the relationship is from child class to parent class.
In child pojo class, we need to take one parent type property.
In child class mapping file, configure this parent type property with <many-to-one> tag.
In client application, set parent class object to child class object and finally save child class object. Then automatically parent class object also will be stored into the database.

Example :

Student.java

```
package com.beans;

public class Student {
private int studentId;
private String stuentName;

public int getStudentId() {
     return studentId;
}
public void setStudentId(int studentId) {
     this.studentId = studentId;
}
public String getStuentName() {
     return stuentName;
}
public void setStuentName(String stuentName) {
     this.stuentName = stuentName;
}

}
```

Course.java

```
package com.beans;
```

```
public class Course {
private int courseId;
private String courseName;

private Student student;

public int getCourseId() {
     return courseId;
}
public void setCourseId(int courseId) {
     this.courseId = courseId;
}
public String getCourseName() {
     return courseName;
}
public void setCourseName(String courseName) {
     this.courseName = courseName;
}
public Student getStudent() {
     return student;
}
public void setStudent(Student student) {
     this.student = student;
}

}
```

Student.hbm.xml

```
<!-- Related to hibernate mapping for POJO -->
<hibernate-mapping>

<class name="com.beans.Student" table="STUDENTDB">
  <id name="studentId" column="SID"/>
  <property name="stuentName" column="SNAME" length="15"/>
</class>

</hibernate-mapping>
```

Course.hbm.xml

```
<!-- Related to hibernate mapping for POJO -->
<hibernate-mapping>
```

```
<class name="com.beans.Course" table="COURSEDB">
  <id name="courseId" column="CID"/>
  <property name="courseName" column="CNAME" length="15"/>

  <many-to-one name="student" column="refSID"
                  class="com.beans.Student" cascade="all"/>
</class>

</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>

<session-factory>

<!-- Related to the connection properties -->

<property
name="myeclipse.connection.profile">myJdbcDriver</property>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriv
er</property>
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</p
roperty>
<property name="connection.username">lms</property>
<property name="connection.password">scott</property>


<!-- Related to the hibernate properties -->
<property
name="dialect">org.hibernate.dialect.Oracle9Dialect</propert
y>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">create</property>

<!-- Related to hibernate mapping -->
<mapping resource="com/hiber/Student.hbm.xml" />
<mapping resource="com/hiber/Course.hbm.xml" />

</session-factory>

</hibernate-configuration>
```

Client Application 1 :

ClientApp.java

```
package com.client;

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

import com.beans.Course;
import com.beans.Student;

public class ClientApp {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

Student student=new Student();
student.setStudentId(101);
student.setStuentName("Arun");

Course course=new Course();
course.setCourseId(201);
course.setCourseName("hibernate");
course.setStudent(student);

Course course2=new Course();
course2.setCourseId(202);
course2.setCourseName("spring");
course2.setStudent(student);

session.save(course);
session.save(course2);

tx.commit();
```

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

}
```

Client Application 2 :

ClientApp.java

```
package com.client;

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

import com.beans.Course;
import com.beans.Student;

public class ClientApp {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

Course course=(Course)session.get(Course.class, 201);
System.out.println(course.getCourseId());
System.out.println(course.getCourseName());

Student student=course.getStudent();
System.out.println(student.getStudentId());
System.out.println(student.getStuentName());

tx.commit();
System.out.println("success");
}

}
```

Client Application 3 :

ClientApp.java

```
package com.client;

import java.util.Iterator;
import java.util.List;

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

import com.beans.Course;
import com.beans.Student;

public class ClientApp {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

String hqlQuery="from Course";
Query query=session.createQuery(hqlQuery);
List list=query.list();
Iterator iterator=list.iterator();

while(iterator.hasNext()){
Course course=(Course)iterator.next();
System.out.println(course.getCourseId());
System.out.println(course.getCourseName());

Student student=course.getStudent();
System.out.println(student.getStudentId());
System.out.println(student.getStuentName());
}
```

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

}
```

Client Application 4 :

ClientApp.java

```
package com.client;

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

import com.beans.Course;
import com.beans.Student;

public class ClientApp {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

Course course=(Course)session.get(Course.class, 201);
session.delete(course);
tx.commit();

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

}
```

One to Many Bidirectional :

Applying both one to many and many to one is nothing but one to many bidirectional.

Example :

Student.java

```
package com.beans;

import java.util.Set;

public class Student {
private int studentId;
private String stuentName;

private Set courses;

public int getStudentId() {
     return studentId;
}
public void setStudentId(int studentId) {
     this.studentId = studentId;
}
public String getStuentName() {
     return stuentName;
}
public void setStuentName(String stuentName) {
     this.stuentName = stuentName;
}
public Set getCourses() {
     return courses;
}
public void setCourses(Set courses) {
     this.courses = courses;
}

}
```

Course.java

```
package com.beans;

public class Course {
private int courseId;
private String courseName;
```

```
private Student student;

public int getCourseId() {
     return courseId;
}
public void setCourseId(int courseId) {
     this.courseId = courseId;
}
public String getCourseName() {
     return courseName;
}
public void setCourseName(String courseName) {
     this.courseName = courseName;
}
public Student getStudent() {
     return student;
}
public void setStudent(Student student) {
     this.student = student;
}

}
```

Student.hbm.xml

```
<!-- Related to hibernate mapping for POJO -->
<hibernate-mapping>

<class name="com.beans.Student" table="STUDENTDB">
  <id name="studentId" column="SID"/>
  <property name="stuentName" column="SNAME" length="15"/>

  <set name="courses" cascade="all" inverse="true">
   <key column="refSID"/>
   <one-to-many class="com.beans.Course"/>
  </set>

</class>

</hibernate-mapping>
```

Course.hbm.xml

```
<!-- Related to hibernate mapping for POJO -->
```

```
<hibernate-mapping>

<class name="com.beans.Course" table="COURSEDB">
  <id name="courseId" column="CID"/>
  <property name="courseName" column="CNAME" length="15"/>

  <many-to-one name="student" column="refCID"
                  class="com.beans.Student" cascade="all"/>
</class>

</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>

<session-factory>

<!-- Related to the connection properties -->

<property
name="myeclipse.connection.profile">myJdbcDriver</property>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriv
er</property>
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</p
roperty>
<property name="connection.username">lms</property>
<property name="connection.password">scott</property>


<!-- Related to the hibernate properties -->
<property
name="dialect">org.hibernate.dialect.Oracle9Dialect</propert
y>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">create</property>

<!-- Related to hibernate mapping -->
<mapping resource="com/hiber/Student.hbm.xml" />
<mapping resource="com/hiber/Course.hbm.xml" />

</session-factory>
```

```
</hibernate-configuration>
```

Client Application 1 :

ClientApp.java

```
package com.client;

import java.util.HashSet;
import java.util.Set;

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

import com.beans.Course;
import com.beans.Student;

public class ClientApp {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

Student student=new Student();
student.setStudentId(101);
student.setStuentName("Ashok");

Course course=new Course();
course.setCourseId(201);
course.setCourseName("hibernate");

Course course2=new Course();
course2.setCourseId(202);
course2.setCourseName("spring");

Set set=new HashSet();
```

```
set.add(course);
set.add(course2);

student.setCourses(set);

session.save(student);
tx.commit();

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

}
```

Client Application 2 :

ClientApp.java

```
package com.client;

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

import com.beans.Course;
import com.beans.Student;

public class ClientApp {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

Student student=new Student();
student.setStudentId(101);
student.setStuentName("Arun");

Course course=new Course();
course.setCourseId(201);
```

```
course.setCourseName("hibernate");
course.setStudent(student);

Course course2=new Course();
course2.setCourseId(202);
course2.setCourseName("spring");
course2.setStudent(student);

session.save(course);
session.save(course2);

tx.commit();
System.out.println("success");
}

}
```

Client Application 3 :

ClientApp.java

```
package com.client;

import java.util.HashSet;
import java.util.Set;

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

import com.beans.Course;
import com.beans.Student;

public class ClientApp {

public static void main(String ar[]){



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

SessionFactory sf=cfg.buildSessionFactory();
```

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

Student student=new Student();
student.setStudentId(101);
student.setStuentName("Akshay");

Course course=new Course();
course.setCourseId(201);
course.setCourseName("hibernate");
course.setStudent(student);

Course course2=new Course();
course2.setCourseId(202);
course2.setCourseName("spring");
course2.setStudent(student);

Set set=new HashSet();
set.add(course);
set.add(course2);


session.save(course2);

tx.commit();
System.out.println("success");

}
}
```

Many to Many mapping :

Many to many is nothing but applying one to many relationship on both sides.

Ex: one student can join in multiple courses and one course can contain multiple students.

Here we need collection property in both pojo classes and we need to configure this collection details in both mapping files.

While configuring collection property details we need to use <many-to-one> to map the relationship.

Note : In many to many relationship a mediator table is mandatory. We can call this table as join table. This join table contains only foreign keys.

Example :

Student.java

```
package com.beans;

import java.util.Set;

public class Student {
private int studentId;
private String stuentName;

private Set courses;

public int getStudentId() {
     return studentId;
}
public void setStudentId(int studentId) {
     this.studentId = studentId;
}
public String getStuentName() {
     return stuentName;
}
public void setStuentName(String stuentName) {
     this.stuentName = stuentName;
}
public Set getCourses() {
     return courses;
}
public void setCourses(Set courses) {
     this.courses = courses;
}

}
```

Course.java

```
package com.beans;

import java.util.Set;

public class Course {
private int courseId;
private String courseName;

private Set students;

public int getCourseId() {
     return courseId;
}
public void setCourseId(int courseId) {
     this.courseId = courseId;
}
public String getCourseName() {
     return courseName;
}
public void setCourseName(String courseName) {
     this.courseName = courseName;
}
public Set getStudents() {
     return students;
}
public void setStudents(Set students) {
     this.students = students;
}

}
```

Student.hbm.xml

```
<!-- Related to hibernate mapping for POJO -->
<hibernate-mapping>

<class name="com.beans.Student" table="STUDENTDB">
  <id name="studentId" column="SID"/>
  <property name="stuentName" column="SNAME" length="15"/>
```

```
  <set name="courses" cascade="all" table="STUDENTCOURSE">
   <key column="refSID"/>
   <many-to-many class="com.beans.Course" column="refCID"/>
  </set>

</class>

</hibernate-mapping>
```

Course.hbm.xml

```
<!-- Related to hibernate mapping for POJO -->
<hibernate-mapping>

<class name="com.beans.Course" table="COURSEDB">
  <id name="courseId" column="CID"/>
  <property name="courseName" column="CNAME" length="15"/>

  <set name="students" cascade="all" table="STUDENTCOURSE">
   <key column="refCID"/>
   <many-to-many class="com.beans.Student" column="refSID"/>
  </set>
</class>

</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>

<session-factory>

<!-- Related to the connection properties -->

<property
name="myeclipse.connection.profile">myJdbcDriver</property>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriv
er</property>
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</p
roperty>
<property name="connection.username">lms</property>
<property name="connection.password">scott</property>
```

```
<!-- Related to the hibernate properties -->
<property
name="dialect">org.hibernate.dialect.Oracle9Dialect</propert
y>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">create</property>

<!-- Related to hibernate mapping -->
<mapping resource="com/hiber/Student.hbm.xml" />
<mapping resource="com/hiber/Course.hbm.xml" />

</session-factory>

</hibernate-configuration>
```

Client Application 1 :

ClientApp.java

```
package com.client;

import java.util.HashSet;
import java.util.Set;

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

import com.beans.Course;
import com.beans.Student;

public class ClientApp {

public static void main(String ar[]){



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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
```

```
Transaction tx=session.beginTransaction();

Student student=new Student();
student.setStudentId(101);
student.setStuentName("Akshay");

Student student2=new Student();
student2.setStudentId(102);
student2.setStuentName("Arun");

Course course=new Course();
course.setCourseId(201);
course.setCourseName("hibernate");

Course course2=new Course();
course2.setCourseId(202);
course2.setCourseName("spring");


Set set=new HashSet();
set.add(course);
set.add(course2);

student.setCourses(set);
student2.setCourses(set);

session.save(student);
session.save(student2);

tx.commit();
System.out.println("success");

}
}
```

One to One mapping :

In one to one mapping, one pojo class object is related with one pojo class object only.
Here foreign key is not required, we need to copy the primary key value of parent object into primary key value of child object.

To achieve this relationship we need to use foreign generator.

Example :

Student.java

```
package com.beans;

import java.util.Set;

public class Student {
private int studentId;
private String stuentName;

private Course course;

public int getStudentId() {
     return studentId;
}
public void setStudentId(int studentId) {
     this.studentId = studentId;
}
public String getStuentName() {
     return stuentName;
}
public void setStuentName(String stuentName) {
     this.stuentName = stuentName;
}
public Course getCourse() {
     return course;
}
public void setCourse(Course course) {
     this.course = course;
}

}
```

Course.java

```
package com.beans;

public class Course {
private int courseId;
```

```
private String courseName;

public int getCourseId() {
     return courseId;
}
public void setCourseId(int courseId) {
     this.courseId = courseId;
}
public String getCourseName() {
     return courseName;
}
public void setCourseName(String courseName) {
     this.courseName = courseName;
}

}
```

Student.hbm.xml

```
<!-- Related to hibernate mapping for POJO -->
<hibernate-mapping>

<class name="com.beans.Student" table="STUDENTDB">
  <id name="studentId" column="SID">
   <generator class="foreign">
     <param name="property">course</param>
   </generator>
  </id>

  <property name="stuentName" column="SNAME" length="15"/>

  <one-to-one name="course" class="com.beans.Course"
cascade="all"/>

</class>

</hibernate-mapping>
```

Course.hbm.xml

```
<!-- Related to hibernate mapping for POJO -->
<hibernate-mapping>

<class name="com.beans.Course" table="COURSEDB">
```

```
  <id name="courseId" column="CID"/>
  <property name="courseName" column="CNAME" length="15"/>

</class>

</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>

<session-factory>

<!-- Related to the connection properties -->

<property
name="myeclipse.connection.profile">myJdbcDriver</property>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriv
er</property>
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</p
roperty>
<property name="connection.username">lms</property>
<property name="connection.password">scott</property>


<!-- Related to the hibernate properties -->
<property
name="dialect">org.hibernate.dialect.Oracle9Dialect</propert
y>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">create</property>

<!-- Related to hibernate mapping -->
<mapping resource="com/hiber/Student.hbm.xml" />
<mapping resource="com/hiber/Course.hbm.xml" />

</session-factory>

</hibernate-configuration>
```

Client Application 1 :

ClientApp.java

```
package com.client;

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

import com.beans.Course;
import com.beans.Student;

public class ClientApp {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

Student student=new Student();
student.setStudentId(101);
student.setStuentName("Akshay");

Course course=new Course();
course.setCourseId(201);
course.setCourseName("Ashok");

student.setCourse(course);

session.save(student);

tx.commit();
System.out.println("success");

}
}
```

Client Application 2 :

ClientApp.java

```
package com.client;

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

import com.beans.Student;

public class ClientApp {

public static void main(String ar[]){

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

SessionFactory sf=cfg.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();

Student student=(Student)session.get(Student.class, 101);
session.delete(student);

tx.commit();
System.out.println("success");

}
}
```