What is Hibernate – Hibernate Introduction

Download jar file:-

<https://sourceforge.net/projects/hibernate/files/hibernate3/>

In this tutorial am going to explain, why Hibernate came into picture though we have JDBC for connecting to the database, and what is this hibernate frame work first let us see what are the draw backs of JDBC

## ****Draw Backs of JDBC:****

* In JDBC, if we open a database connection we need to write in try, and if any exceptions occurred catch block will takers about it, and finally used to close the connections.
* here as a programmer we must close the connection, or we may get a chance to get our of connections message…!
* Actually if we didn’t close the connection in the finally block, then jdbc doesn’t responsible to close that connection.
* In JDBC we need to write Sql commands in various places, after the program has created if the table structure is modified then the JDBC program doesn’t work, again we need to modify and compile and re-deploy required, which is tedious.
* JDBC used to generate database related error codes if an exception will occurs, but java programmers are unknown about this error codes right.
* In the Enterprise applications, the data flow with in an application from class to class will be in the form of objects, but while storing data finally in a database using JDBC then that object will be converted into text.  Because JDBC doesn’t transfer objects directly.

In order to overcome above problems,  Hibernate came into picture..!

## What is Hibernate:

Hibernate is the ORM tool given to 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 a non-invasive framework,  means it wont forces the programmers to extend/implement any class/interface, and in hibernate we have all POJO classes so its light weight.

Hibernate can runs with in or with out server, i mean it will suitable for all types of java applications (stand alone or desktop or any servlets.)

Hibernate is purely for persistence (to store/retrieve data from Database).

# Main Advantage And Disadvantages Of Hibernates

## Advantages of hibernates:

* Hibernate supports Inheritance, Associations, Collections
* In hibernate if we save the derived class object,  then its base class object will also be stored into the database, it means hibernate supporting inheritance
* Hibernate supports relationships like One-To-Many,One-To-One, Many-To-Many-to-Many, Many-To-One
* This will also supports collections like List,Set,Map (Only new collections)
* In jdbc all exceptions are checked exceptions, so we must write code in try, catch and throws, but in hibernate we only have Un-checked exceptions, so no need to write try, catch, or no need to write throws.  Actually in hibernate we have the translator which converts checked to Un-checked ;)
* Hibernate has capability to generate primary keys automatically while we are storing the records into database
* Hibernate has its own query language, i.e hibernate query language which is database independent
* So if we change the database, then also our application will works as HQL is database independent
* HQL contains database independent commands
* While we are inserting any record, if we don’t have any particular table in the database, JDBC will rises an error like “View not exist”, and throws exception, but in case of hibernate, if it not found any table in the database this will create the table for us ;)
* Hibernate supports caching mechanism by this, the number of round trips between an application and the database will be reduced, by using this caching technique an application performance will be increased automatically.
* Hibernate supports annotations, apart from XML
* Hibernate provided Dialect classes, so we no need to write sql queries in hibernate, instead we use the methods provided by that API.
* Getting pagination in hibernate is quite simple.

 Disadvantages of hibernates:

* I don’t think there are disadvantages in hibernate
* You know some thing.., Its saying hibernate is little slower than pure JDBC, actually the reason being hibernate used to generate many SQL statements in run time, but i guess this is not the disadvantage :-)
* But there is one major disadvantage, which was boilerplate code issue, actually we need to write samecode in several files in the same application, but spring eliminated this

# Hibernate Versioning Example, Hibernate Versioning Of Objects

Once an object is saved in a database, we can modify that object any number of times right, If we want to know how many no of times that an object is modified then we need to apply this versioning concept.  
When ever we use versioning then hibernate inserts version number as **zero**, when ever object is saved for the first time in the database.  Later hibernate increments that version no by one automatically when ever a modification is done on that particular object.  
In order to use this versioning concept, we need the following two changes in our application

* Add one property of type int in our pojo class
* In hibernate mapping file, add an element called version soon after id element

**Note**:

* Remember friends, first we must run the logic to **save** the object then hibernate will inset 0 (Zero) by default in the version column of the database, its very important point in the interview point of view also
* First save logic to let the hibernate to insert zero in the version column, then any number of updatelogic’s (programs) we run, hibernate will increments +1 to the previous value
* But if we run the update logic for the first time, hibernate will not insert zero..! it will try to incrementthe previous value which is NULL in the database so we will get the exception.

# Hibernate Lifecycle Of pojo Class Objects

Actually our POJO class object having 3 states like…

* Transient state
* Persistent state
* Detached state

## Transient & Persistent states:

* When ever an object of a pojo class is created then it will be in the Transient state
* When the object is in a Transient state it doesn’t represent any row of the database, i mean not associated with any Session object, if we speak more we can say no relation with the database its just an normal object
* If we modify the data of a pojo class object, when it is in transient state then it doesn’t effect on the database table
* When the object is in persistent state, then it represent one row of the database, if the object is in persistent state then it is associated with the unique Session
* if we want to move an object from persistent to detached state, we need to do either closing that session or need to clear the cache of the session
* if we want to move an object from persistent state into transient state then we need to delete that object permanently from the database

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | import org.hibernate.\*;  import org.hibernate.cfg.\*;    public class ClientProgram {    public static void main(String[] args)  {    Configuration cfg = new Configuration();  cfg.configure("hibernate.cfg.xml");    SessionFactory factory = cfg.buildSessionFactory();  Session session = factory.openSession();             // Transient state\_\_\_\_\_start  Product p=new Product();  p.setProductId(101);  p.setProName("iPhone");  p.setPrice(25000);           // Transient state\_\_\_\_\_end             // Persistent state\_\_\_\_\_start  Transaction tx = session.beginTransaction();  session.save(p);  System.out.println("Object saved successfully.....!!");  tx.commit();           // Persistent state\_\_\_\_\_end    session.close();  factory.close();  }    } |

Note:

* see the above client program, line numbers **16** to **19** we just loaded the object and called the corresponding setter methods, its not related to the database row
* if you see, line number **24** we called save method in the Session Interface, means the object is now having the relation with the database
* if we want to convert the object from Transient state to Persistentstate we can do in 2 ways
  + By saving that object like above
  + By loading object from database

If we do any modifications all the changes will first applied to the object in session cache only (Let\_\_ we do the modifications 5 times, then 5 times we need to save the changes into the database right, which means number of round trips from our application to database will be increased, Actually if we load an object from the database, first it will saves in the cache-memory so if we do any number of changes all will be effected at cache level only and finally we can call save or update method so with the single call of save or update method the data will be saved into the database.

If we want to save an object into database then we need to call any one of the following 3 methods

* save()
* persist()
* saveOrUpdate()

i will explain about persist, saveOrUpdate methods later….

If we want to load an object from database, then we need to call either load() or get() methods

## Transient:

One newly created object,with out having any relation with the database, means never persistent, not associated with any Session object

## Persistent:

Having the relation with the database, associated with a unique Session object

## Detached:

previously having relation with the database [persistent ], now not associated with any Session

see the next sessions for the better understanding of the life cycle states of pojo class object(s) the hibernate

# Hibernate Converting Object From Detached to Persistent state

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | import org.hibernate.\*;  import org.hibernate.cfg.\*;    public class ClientLogicProgram {        public static void main(String... args)      {            Configuration cfg = new Configuration();          cfg.configure("hibernate.cfg.xml");            SessionFactory factory = cfg.buildSessionFactory();            Session session1 = factory.openSession();             Product p=null;          //Transient state..           Object o=session1.get(Product.class, new Integer(1001));           p=(Product)o;           //now p is in Persistent state..            session1.close();            p.setPrice(36000);            // p is in Detached state            Session session2=factory.openSession();             Transaction tx=session2.beginTransaction();              session2.update(p);      // now p reached to Persistent state           tx.commit();            session2.close();            factory.close();  }  } |

Notes:

* We have opened the session1 at line number **14** and closed at line number **20**, see i have been loaded the Product class object by using get(-,-) method
* **Remember** get() method always returns the super class object (Object)
* so i typecast into my pojo class object type, so now we can use print any value from this object so its in the Persistent state
* see line number **22**,  am trying to change the Price,  but it wont effect the database because its not in the session cache so i need to take one more session to update this value in the database, so for that reason i took one more session from line numbers 24 – 30
* Gun short point is, things related to database must go with in the session only that’s it

# Inheritance Mapping In Hibernate – Introduction

Compared to JDBC we have one main advantage in hibernate, which is hibernate inheritance.  Suppose if we have base and derived classes, now if we save derived(sub) class object, base class object will also be stored into the database.  
But the thing is we must specify in what table we need to save which object data ( i will explain about this point later, just remember as of now).

**Hibernate supports 3 types of Inheritance Mappings**:

* Table per class hierarchy
* Table per sub-class hierarchy
* Table per concrete class hierarchy

# Generators <generator> In Hibernate

<generator /> is one of main element we are using in the hibernate framework [in the mapping file],  let us see the concept behind this generators.

* Up to now in our hibernate mapping file, we used to write <generator /> in the id element scope, actually this is default like whether you write this assigned generator or not hibernate will takes automatically
* In fact this assigned means hibernate will understand that, while saving any object hibernate is not responsible to create any primary key value for the current inserting object, user has to take the response
* The thing is, while saving an object into the database, the generator informs to the hibernate that, how the primary key value for the new record is going to generate
* hibernate using different primary key generator algorithms, for each algorithm internally a class is created by hibernate for its implementation
* hibernate provided different primary key generator classes and all these classes are implemented from

org.hibernate.id.IdentifierGeneratar Interface

* while configuring <generator /> element in mapping file, we need to pass parameters if that generator class need any parameters, actually one sub element of <generator /> element is <param />, will talk more about this

|  |  |
| --- | --- |
| 1  2  3 | <generator class="">  <param name=""> value </param>  </generator> |

## List of generators

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

* assigned
* increment
* sequence
* identity
* hilo
* native
* foregin
* uuid.hex
* uuid.string

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

## assigned

* This generator supports in all the databases
* This is the default generator class used by the hibernate, if we do not specify <generator –> element under id element then hibernate by default assumes it as “assigned”
* If generator class is assigned, then the programmer is responsible for assigning the primary key value to object which is going to save into the database

## Increment

* This generator supports in all the databases, database independent
* This generator is used for generating the id value for the new record by using the formula

Max of id value in Database + 1

* if we manually assigned the value for primary key for an object, then hibernate doesn’t considers that value and uses **max value of id in database + 1** concept only :-)
* If there is no record initially in the database, then for the first time this will saves primary key value as 1, as…

max of id value in database + 1  
0 + 1  
result -> 1

## sequence

* Not has the support with MySql
* This generator class is database dependent it means, we cannot use this generator class for all the database, we should know whether the database supports sequence or not before we are working with it
* while inserting a new record in a database, hibernate gets next value from the sequence under assigns that value for the new record
* If programmer has created a sequence in the database then that sequence name should be passed as the generator
* If the programmer has not passed any sequence name, then hibernate creates its own sequence with name “**Hibernate-Sequence**” and gets next value from that sequence, and than assigns that id value for new record
* But remember, if hibernate want’s to create its own sequence, in hibernate configuration file, **hbm2ddl.auto** property must be set enabled

sql> create sequence MySequence incremented by 5;

* first it will starts with 1 by default
* though you send the primary key value., hibernate uses this sequence concept only
* But if we not create any sequence, then first 1 and increments by 1..bla bla. in this case hibernate creating right..? so ensure we have hbm2ddl.auto enabled in the configuration file

## identity

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

As this is not working in Oracle, if you would like to check this in MySql you must change the configuration file as…….

**class**: com.mysql.jdbc.Driver  
**url**: jdbc:mysql://www.java4s.com:3306/test (test is default database)  
**user**: root (default)  
**pass**: (default)  
**dialet**: org.hibernate.dialet.MySQLDialet

**Note**:

* jar file required (in class path we must set..  
  **mysql-connector-java-3.0.8-stable-bin.jar** (version number may change)
* Actually this jar will never come along with mysql database software,  to get this jar file we need to download the following file, and unzip it.**mysql-connectar-java-3.0.8-stable.zip**

## hilo

* This generator is database independent
* for the first record, the id value will be inserted as 1
* for the second record the id value will be inserted as 32768
* for the next records the id value will be incremented by 32768 and will stores into the database (i mean adds to the previous)
* actually this hibernate stores the count of id values generated in a column of separated table, with name “**hibernate\_unique\_key**” by default with the column name “**next\_hi”**
* if we want to modify the table and column names theen wee need to pass 2 parameter’s for the hilo generators

## native

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

* identity
* sequence
* hilo

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

## foreign

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

# Part 1 Hibernate Query Language Introduction

So far we done the operations on single object (single row), here we will see modifications, updates on multiple rows of data (multiple objects) at a time.  In hibernate we can perform the operations on asingle row (or) multiple rows at a time, if we do operations on multiple rows at once, then we can call this as bulk operations.

* HQL is the own query language of hibernate and it is used to perform bulk operations on hibernate programs
* An object oriented form of SQL is called HQL
* here we are going to replace table column names  with POJO class variable names and table nameswith POJO class names in order to get HQL commands

## Advantages Of HQL:

* HQL is database independent, means if we write any program using HQL commands then our program will be able to execute in all the databases with out doing any further changes to it
* HQL supports object oriented features like ***Inheritance***, ***polymorphism***, ***Associations***(Relation ships)
* HQL is initially given for selecting object from database and in hibernate 3.x we can do DML operations ( insert, update…) too

## Different Ways Of Construction HQL Select

* If we want to select a **Complete Object** from the database, we use POJO class reference in place of   **\***  while constructing the query
* In this case (select a complete object from the database) we can directly start our HQL command from,  **from** key word

**Example:**

|  |
| --- |
| // In SQL  sql> select \* from Product  Note: Product is the table name right....!!!    // In HQL  hql> select p from Product p       [ or ]       from Product p  Note: here p is the reference...!! |

 If we want to load the **Partial Object** from the database that is only selective properties (selected columns) of an objects then we need to replace column names with POJO class variable names.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | // In SQL  sql> select pid,pname from Product  Note: pid, pname are the columns Product is the table name right..!    // In HQL  hql> select p.productid,p.productName from Product p       [ or ]       from Product p ( we should not start from, from key word here because  we selecting the columns hope you are getting me )    Note: here p is the reference...!!             productid,productName are POJO variables |

* It is also possible to **load** or **select** the object from the database **by passing run time values** into the query,  in this case we can use either ” ? ” symbol or label in an HQL command, the index number of  ” ? ” will starts from zero but not one ( Remember this, little important regarding interview point of view)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | // In SQL  sql> select \* from Product where pid=?  Note: Product is the table name right..!    // In HQL  hql> select p from Product p where p.productid=?       [ or ]       select p from Product p where p.productid=:java4s       [ or ]       from Product p where p.productid=?       [ or ]       from Product p where p.productid=:java4s    Note: Here p is the reference...!! |

* Here :java4s refers that, java4s is a label(colon ‘:’ symbol represents that its the label)
* we used to store some values in this label in run time (in our class)

# Part 2 Hibernate Query Language, Executing HQL Commands

## Procedure To Execute HQL Command:

* If we want to execute execute an HQL query on a database, we need to create a query object
* ” Query ” is an interface given in org.hibernate package
* In order to get query object, we need to call createQuery() method in the session Interface
* Query is an interface, QueryImpl is the implemented class
* we need to call list method for executing an HQL command on database, it returns java.util.List
* we need to use java.util.Iterator for iterating the List collection

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Query qry = session.createQuery("--- HQL command ---");  List l = qry.list();  Iterator it = l.iterator();  while(it.hasNext())  {    Object o = it.next();    Product p = (Product)o;    ----- ------- ---------  } |

**Notes:**

* line 1: Getting the Query object
* line 2: Executing the object content (which is HQL command)
* line 3: Iterating, then while loop and type cast into our class type that’s it

# Part 3 HQL, Different Ways Of Executing HQL Commands

we can execute our HQL command in 3 ways,  like by selecting total object, partial object (more than one column), partial object (with single column).  Let us see..

## Different Ways Of Executing HQL

**Case 1: [ Selecting Complete Object ]**

* In this approach, we are going to select complete object from the database, so while iterating the collection, we need to typecast each object into our  POJO class type only
* Internally hibernate converts each row selected from the table into an object of POJO class and hibernate stores all these POJO class objects into list so while iterating the collection, we typecast into POJO class type

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Query qry = session.createQuery("select \* from Product p ");  List l =qry.list();  Iterator it = l.iterator();  while(it.hasNext())  {     Object o = it.next();     Project p = (Product)o;     ----- ------- ------  } |

**Case 2: [ Selecting Partial Object ]**

* In this approach we are going to select partial object, (selected columns, i mean more than one column not single column)
* In this case hibernate internally stores the multiple column values of each row into an object array and stores these object arrays into List collection
* At the time of iterating the collection, we need to typecast the result into an object arrays

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Query qry = session.createQuery("select p.pid,p.pname from Product p");  List l =qry.list();  Iterator it = l.iterator();  while(it.hasNext())  {     Object o[] = (Object o[])it.next();     System.out.println("-------");     ----- ------- ------  } |

**Case 3: [ Selecting Partial Object ]**

* In this case we are going to select partial object with single column from the database
* In this case hibernate internally creates an object of that value type and stores all these objects into the list collection
* At the time of iterating the collection, we need to typecast into that object type only

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Query qry = session.createQuery("select p.pid from Product p");  List l =qry.list();  Iterator it = l.iterator();  while(it.hasNext())  {     Integer i = (Integer)it.next();     System.out.println(i.intValue());     ----- ------- ------  } |

**Note**:  it.next() return type is always **Object**

# Criteria Query, Hibernate Criteria Query Introduction

Unlike HQL, Criteria is only for selecting the data from the database, that to we can select complete objects only not partial objects, in fact by combining criteria and projections concept we can select partial objects too we will see this angle later,  ;) but for now see how we are using criteria for selecting complete objects form the database. We cant perform non-select operations using this criteria.  Criteria is suitable for executing dynamic queries too, let us see how to use this criteria queries in the hibernate..

|  |
| --- |
| Criteria crit = session.createCriteria(Employee.class);  // let Product is our pojo class  List l = crit.list()  // need to call list() to execute criteria  Iterator it = l.iterator();  while(it.hasNext())  {  Object o = it.next();  Product p = (Product)o;  ------ ----- -----  } |

## Adding Conditions To Criteria

* If we want to put conditions to load data from database, using criteria then we need to create one Criterion Interface object and we need to add this object to Criteria Class object

crit.add(**Criterion Interface Object**)  
crit = criteria class object

* Criterion is an interface given in “org.hibernate.criterion” package
* In order to get Criterion object, we need to use Restrictions class
* Restrictions is the factory for producing Criterion objects, but friends there is no explicit relation between Criterion interface and Restrictions class, it means Restrictions class is not implemented from Criterion Interface
* In Restrictions class, we have all static methods and each method of this class returns Criterion object
* Restrictions class is also given in “org.hibernate.criterion” package

|  |  |
| --- | --- |
| 1  2  3  4  5 | Criteria crit = session.createCriteria(Products.class);  Criterion c1=Restrictions.gt("price", new Integer(12000));  //price is our pojo class variable  crit.add(c1); // adding criterion object to criteria class object  List l = crit.list(); // executing criteria query |

Note: See line number 2, am calling gt(-,-) method of Restrictions class, (means greater than), in our above example am fetching the data by comparing price greater than (>) 12000

* If we want to put more conditions on the data (multiple conditions) then we can use **and** method , **or**method give by the Restrictions class

|  |  |
| --- | --- |
| 1  2  3  4 | crit.add(Restrictions.and(Restrictions.like("proName","%R%"),  Restrictions.eq("price",new Integer(12000))));  List l=crit.list();  Iterator it = l.iterator(); |

Like this we can add any number of conditions…

# Hibernate Projections Introduction

So far in criteria, we are able to load complete object right….! let us see how to load the partial objects while working with criteria.  The projections concept is introduced in hibernate 3.0 and mainly we can do the following 2 operations using the projection

* We can load partial object from the database
* We can find the Result of Aggregate functions

*Projection* is an Interface given in “org.hibernate.criterion” package, *Projections* is an class given in same package,  actually Projection is an interface, and Projections is an class and is a factory for producing projection objects.

In Projections class, we have all static methods and each method of this class returns Projection interface object.

If we want to add a Projection object to Criteria then we need to call a method **setProjection()**

**Remember**, while adding projection object to criteria, it is possible to add one object at a time.  It means if we add 2nd projection object then this 2nd one will overrides the first one (first one wont be work), so at a time we can only one projection object to criteria object.

Using criteria, if we want to load partial object from the database, then we need to create a projection object for property that is to be loaded from the database

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Criteria crit = session.createCriteria(Products.class);  crit.setProjection(Projections.proparty("proName"));  List l=crit.list();  Iterator it=l.iterator();  while(it.hasNext())  {  String s = (String)it.next();  // ---- print -----  } |

If we add multiple projections to criteria then the last projection added will be considered to execute see…

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | Criteria crit = session.createCriteria(Products.class);    Projection p1 = Projection.property("proName");  Projection p2 = Projection.property("price");    crit.setProjection(p1):  crit.setProjection(p2):  List l=crit.list();  -------------- ----  ------- - ---  ---- |

Here collections list l, is going to contain the price in the form of Double objects, but product names are over ridden,  second projection over rides the first one, i mean p2 only will works p1 will not works, actually there is a way to add multiple projections to criteria to select more than one column from the database i will tell you in the next example :-)

# Difference between HQL and Criteria Query in Hibernate

* HQL is to perform both select and non-select operations on the data,  but Criteria is only for selecting the data, we cannot perform non-select operations using criteria
* HQL is suitable for executing Static Queries, where as Criteria is suitable for executing Dynamic Queries
* HQL doesn’t support pagination concept, but we can achieve pagination with Criteria
* Criteria used to take more time to execute then HQL
* With Criteria we are safe with SQL Injection because of its dynamic query generation but in HQL as your queries are either fixed or parametrized, there is no safe from SQL Injection.

# Hibernate Native SQL Query Example

Native SQL is another technique of performing bulk operations on the data using hibernate

* By using Native SQL, we can perform both select, non-select operations on the data
* In face Native SQL means using the direct SQL command specific to the particular (current using) database and executing it with using hibernate

## Advantages and Disadvantages of Native SQL

* We can use the database specific keywords (commands), to get the data from the database
* While migrating a JDBC program into hibernate, the task becomes very simple because JDBC uses direct SQL commands and hibernate also supports the same commands by using this Native SQL
* The main draw back of Native SQL is, some times it makes the hibernate application as database dependent one

If we want to execute Native SQL Queries on the database then, we need to construct an object of SQLQuery, actually this SQLQuery is an interface extended from Query and it is given in ” org.hibernate package ”

In order to get an object of SQLQuery, we need to use a method createSQLQuery() given by session interface.

While executing native sql queries on the database, we use directly tables, column names directly in our command.

**Remember**, while executing Native SQL Queries, even though we are selecting complete objects from teh database we need to type cast into object array only, not into our pojo class type, because we are giving direct table, column names in the Native SQL Querie so it does’nt know our class name

If we execute the command, always first it will put’s data in **ResultSet** and from there List

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | SQLQuery qry = session.createSQLQuery("select \* from PRODUCTS");  // Here PRODUCTS is the table in the database...  List l = qry.list();  Iterator it = l.iterator();  while(it.hasNext())  {  Object row[] = (Object[])it.next();  --- -------  } |

* while selecting data from the table, even though you are selecting the complete object from the table, in while loop still we type cast into object array only right
* See the above code, we typecast into the object[] arrays right..,  in case if we want to type cast into our POJO class (i mean to get POJO class obj), then we need to go with entityQuery concept
* In order to inform the hibernate that convert each row of ResultSet into an object of the POJO class back, we need to make the query as an entityQuery
* to make the query as an entityQuery, we need to call addEntity() method

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | //We are letting hibernate to know our pojo class too  SQLQuery qry = session.createSQLQuery("select \* from PRODUCTs").addEntity(Product.class);  List l = qry.list();  Iterator it = l.iterator();  while(it.hasNext())  {  Product p = (Product)it.next();  --- -------  } |

**Notes:**

* See line number 2, i have been added addEntity(Product.class) at the end, which will let the hibernate to know about our POJO class, so now we can typecast into our POJO class type like what i have done at  line number 7
* And that’s it, this is the total concept on this Native SQL, am not going to give any example on this separately hope you understood the concept

# Hibernate Named Query Introduction Tutorial

Let us see few points, before going to see an example on Named Queries in HIbernate..

* While executing either HQL, NativeSQL Queries if we want to execute the same queries for multiple times and in more than one client program application then we can use the Named Queries mechanism
* In this Named Queries concept, we use some name for the query configuration, and that name will be used when ever the same query is required to execute
* In hibernate mapping file we need to configure a query by putting some name for it and in the client application, we need to use getNamedQuery() given by session interface, for getting the Query reference and we need to execute that query by calling list()
* If you want to create Named Query then we need to use **query element** in the hibernate mapping file

## Syntax Of hibernate mapping file [For HQL]

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <hibernate-mapping>    <class name="---" table="---">     <id name="---" column="---"  />     <property name="---" column="---" length="10"/>     <property name="---" column="---" />     --- --- --- ---  </class>    <query name="Give Query Name">      <![CDATA[from Product p where p.price = :java4s]]>  </query>    </hibernate-mapping> |

**Notes:**

* See line numbers 10,11,12, this is the new element we have to add to work with Named Queries
* there colon (:) java4s is the label, i will pass the value into that label in the run time.., or let us see the client program logic too

## Example Logic in Application:

|  |  |
| --- | --- |
| 1  2  3 | Query qry = session.getNamedQuery("Name we given in hibernate mapping xml");  qry.setParameter("java4s",new Integer(1022));  List l = qry.list(); |

**Notes**:

* Line number 1, getting the query from hibernate mapping file to our client program
* Line number 2, passing run time value to that query
* Line number 3, calling list() method to execute the query

Up to now this is the case if we use HQL query in hibernate mapping file, let us see the case if we would like to use nativeSQL query

## Syntax Of hibernate mapping file [For Native SQL]

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <hibernate-mapping>    <class name="---" table="---">     <id name="---" column="---"  />     <property name="---" column="---" length="10"/>     <property name="---" column="---" />     --- --- --- ---  </class>    <sql-query name="Give Query Name">      select \* from PRODUCTS  </sql-query>    </hibernate-mapping> |

**Notes:**

* If we want to give HQL query in hiberante mapping file, we need to use <query/> element, but we have to use <sql-query /> element in case of Native SQL
* See line number 11, its the normal sql command, and PRODUCTS is the table name, not the pojo class name :-)

Done…!!!!!

# Hibernate Caching Mechanism, Hibernate Cache

very fresh session having its own cache memory, Caching is a mechanism for storing the loaded objects into a cache memory.  The advantage of cache mechanism is, whenever again we want to load the same object from the database then instead of hitting the database once again, it loads from the local cache memory only, so that the no. of round trips between an application and a database server got decreased.  It means caching mechanism increases the performance of the application.

In hibernate we have two levels of caching

* First Level Cache [ or ] Session Cache
* Second Level Cache [ or ] Session Factory Cache [ or  ] JVM Level Cache

# Hibernate First Level Cache Example

Let us try to understand the first level cache in hibernate,  actually i tried to give almost all the concept about this first level cache hope you will enjoy this :-)

* By default, for each hibernate application, the first level cache is automatically been enabled
* As a programmer, we no need to have any settings to enable the first level cache and also we cannot disable this first level cache
* the first level cache is associated with the session object and scope of the cache is limited to one session only
* When we load an object for the first time from the database then the object will be loaded from the database and the loaded object will be stored in the cache memory maintained by that session object
* If we load the same object once again, with in the same session, then the object will be loaded from the local cache memory not from the database
* If we load the same object by opening other session then again the object will loads from the database and the loaded object will be stored in the cache memory maintained by this new session