Hibernate

**1.What is ORM ?**

ORM stands for object/relational mapping. ORM is the automated persistence of objects in a Java application to the tables in a relational database.

**2.What does ORM consists of ?**

An ORM solution consists of the followig four pieces:

* API for performing basic CRUD operations
* API to express ries refering to classes
* Facilities to specify metadata
* Optimization facilities : dirty checking,lazy associations fetching

**3.What are the ORM levels ?**

The ORM levels are:

* Pure relational (stored procedure.)
* Light objects mapping (JDBC)
* Medium object mapping
* Full object Mapping (composition,inheritance, polymorphism, persistence by reachability)

**4.What is Hibernate?**

Hibernate is a pure Java object-relational mapping (ORM) and persistence framework that allows you to map plain old Java objects to relational database tables using (XML) configuration files.Its purpose is to relieve the developer from a significant amount of relational data persistence-related programming tasks.

**5.Why do you need ORM tools like hibernate?**

The main advantage of ORM like hibernate is that it shields developers from messy [SQL](http://www.developersbook.com/hibernate/interview-questions/hibernate-interview-questions-faqs.php). Apart from this, ORM provides following benefits:

* **Improved productivity**
  + High-level object-oriented API
  + Less Java code to write
  + No SQL to write
* **Improved performance**
  + Sophisticated caching
  + Lazy loading
  + Eager loading
* **Improved maintainability**
  + A lot less code to write
* **Improved portability**
  + ORM framework generates database-specific SQL for you

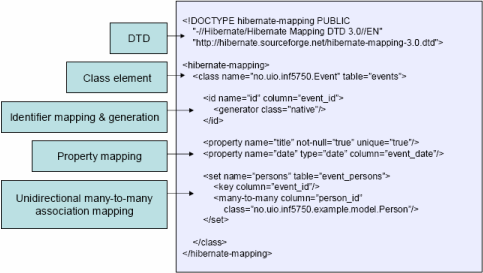
**6.What Does Hibernate Simplify?**

Hibernate simplifies:

* Saving and retrieving your domain objects
* Making database column and table name changes
* Centralizing pre save and post retrieve logic
* Complex joins for retrieving related items
* Schema creation from object model

**7.What is the need for Hibernate xml mapping file?**

Hibernate mapping file tells Hibernate which tables and columns to use to load and store objects. Typical mapping file look as follows:

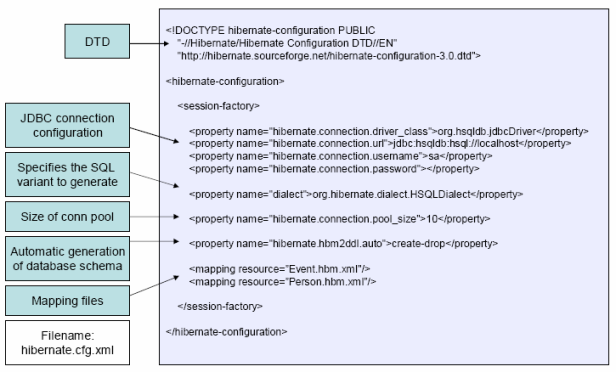
  
  
**8.What are the most common methods of Hibernate configuration?**

The most common methods of Hibernate configuration are:

* Programmatic configuration
* XML configuration (hibernate.cfg.xml)

**9.What are the important tags of hibernate.cfg.xml?**

An Action Class is an adapter between the contents of an incoming HTTP rest and the corresponding business logic that should be executed to process this rest.

  
  
**10.What are the Core interfaces are of Hibernate framework?**

The five core interfaces are used in just about every Hibernate application. Using these interfaces, you can store and retrieve persistent objects and control transactions.

* Session interface
* SessionFactory interface
* Configuration interface
* Transaction interface
* Query and Criteria interfaces

**11.What role does the Session interface play in Hibernate?**

The Session interface is the primary interface used by Hibernate applications. It is a single-threaded, short-lived object representing a conversation between the application and the persistent store. It allows you to create query objects to retrieve persistent objects.  
  
Session session = sessionFactory.openSession();

**Session interface role**:

* Wraps a JDBC connection
* Factory for Transaction
* Holds a mandatory (first-level) cache of persistent objects, used when navigating the object graph or looking up objects by identifier

**12.What role does the SessionFactory interface play in Hibernate?**

The application obtains Session instances from a SessionFactory. There is typically a single SessionFactory for the whole application—created during application initialization. The SessionFactory caches generate [SQL statements](http://www.developersbook.com/hibernate/interview-questions/hibernate-interview-questions-faqs.php) and other mapping metadata that Hibernate uses at runtime. It also holds cached data that has been read in one unit of work and may be reused in a future unit of work  
  
SessionFactory sessionFactory = configuration.buildSessionFactory();

**13.What is the general flow of Hibernate communication with RDBMS?**

The general flow of Hibernate communication with RDBMS is :

* Load the Hibernate configuration file and create configuration object. It will automatically load all hbm mapping files
* Create session factory from configuration object
* Get one session from this session factory
* Create HQL Query
* Execute query to get list containing Java objects

**14.What is Hibernate Query Language (HQL)?**

Hibernate offers a query language that embodies a very powerful and flexible mechanism to query, store, update, and retrieve objects from a database. This language, the Hibernate query Language (HQL), is an object-oriented extension to SQL.

**15.How do you map Java Objects with Database tables?**

* First we need to write [Java](http://www.developersbook.com/hibernate/interview-questions/hibernate-interview-questions-faqs.php) domain objects (beans with setter and getter). The variables should be same as database columns.
* Write hbm.xml, where we map [java class](http://www.developersbook.com/hibernate/interview-questions/hibernate-interview-questions-faqs.php) to table and database columns to Java class variables.

**Example** :

<hibernate-mapping>  
  <class name="com.test.User"  table="user">  
   <property  column="USER\_NAME" length="255"   
      name="userName" not-null="true"  type="java.lang.String"/>  
   <property  column="USER\_PASSWORD" length="255"  
 name="userPassword" not-null="true"  type="java.lang.String"/>  
 </class>  
</hibernate-mapping>

**16.What’s the difference between load() and get()?**

load() vs. get() :-

|  |  |
| --- | --- |
| **load()** | **get()** |
| Only use the load() method if you are sure that the object exists. | If you are not sure that the object exists, then use one of the get() methods. |
| load() method will throw an exception if the unique id is not found in the database. | get() method will return null if the unique id is not found in the database. |
| load() just returns a proxy by default and database won’t be hit until the proxy is first invoked. | get() will hit the database immediately. |

**17.What is the difference between and merge and update ?**

Use update() if you are sure that the session does not contain an already persistent instance with the same identifier, and merge() if you want to merge your modifications at any time without consideration of the state of the session.

**18.How do you define sequence generated primary key in hibernate?**

Using <generator> tag.  
**Example**:-

<id column="USER\_ID" name="id" type="[java](http://www.developersbook.com/hibernate/interview-questions/hibernate-interview-questions-faqs-2.php).lang.Long">   
 <generator class="sequence">   
 <param name="table">SEQUENCE\_NAME</param>  
 <generator>  
</id>

|  |
| --- |
|  |

**19.Define cascade and inverse option in one-many mapping?**

cascade - enable operations to cascade to child entities.  
cascade="all|none|save-update|delete|all-delete-orphan"  
  
inverse - mark this collection as the "inverse" end of a bidirectional association.  
inverse="true|false"   
Essentially "inverse" indicates which end of a relationship should be ignored, so when persisting a parent who has a collection of children, should you ask the parent for its list of children, or ask the children who the parents are?

**20.What do you mean by Named – SQL query?**

Named SQL queries are defined in the mapping [xml document](http://www.developersbook.com/hibernate/interview-questions/hibernate-interview-questions-faqs-2.php) and called wherever required.  
**Example:**

<sql-query name = "empdetails">  
   <return alias="emp" class="com.test.Employee"/>  
      SELECT emp.EMP\_ID AS {emp.empid},  
                 emp.EMP\_ADDRESS AS {emp.address},  
                 emp.EMP\_NAME AS {emp.name}   
 FROM Employee EMP WHERE emp.NAME LIKE :name  
</sql-query>

Invoke Named Query :

List people = session.getNamedQuery("empdetails")  
 .setString("TomBrady", name)  
 .setMaxResults(50)  
 .list();

**21.How do you invoke Stored Procedures?**

<sql-query name="selectAllEmployees\_SP" callable="true">  
 <return alias="emp" class="employee">  
  <return-property name="empid" column="EMP\_ID"/>

<return-property name="name" column="EMP\_NAME"/>         
 <return-property name="address" column="EMP\_ADDRESS"/>  
    { ? = call selectAllEmployees() }  
 </return>  
</sql-query>

**22.Explain Criteria API**

Criteria is a simplified API for retrieving entities by composing Criterion objects. This is a very convenient approach for functionality like "search" screens where there is a variable number of conditions to be placed upon the result set.  
**Example** :

List employees = session.createCriteria(Employee.class)  
        .add(Restrictions.like("name", "a%") )  
          .add(Restrictions.like("address", "Boston"))  
 .addOrder(Order.asc("name") )  
 .list();

**23.Define HibernateTemplate?**

org.springframework.orm.hibernate.HibernateTemplate is a helper class which provides different methods for querying/retrieving data from the database. It also converts checked HibernateExceptions into unchecked DataAccessExceptions.

**24.What are the benefits does HibernateTemplate provide?**

The benefits of HibernateTemplate are :

* HibernateTemplate, a Spring Template class simplifies interactions with Hibernate Session.
* Common functions are simplified to single method calls.
* Sessions are automatically closed.
* Exceptions are automatically caught and converted to runtime exceptions.

**25.How do you switch between relational databases without code changes?**

Using Hibernate [SQL](http://www.developersbook.com/hibernate/interview-questions/hibernate-interview-questions-faqs-2.php) Dialects , we can switch databases. Hibernate will generate appropriate hql queries based on the dialect defined.

**26.If you want to see the Hibernate generated** [**SQL statements**](http://www.developersbook.com/hibernate/interview-questions/hibernate-interview-questions-faqs-2.php) **on console, what should we do?**

In Hibernate configuration file set as follows:   
<property name="show\_sql">true</property>

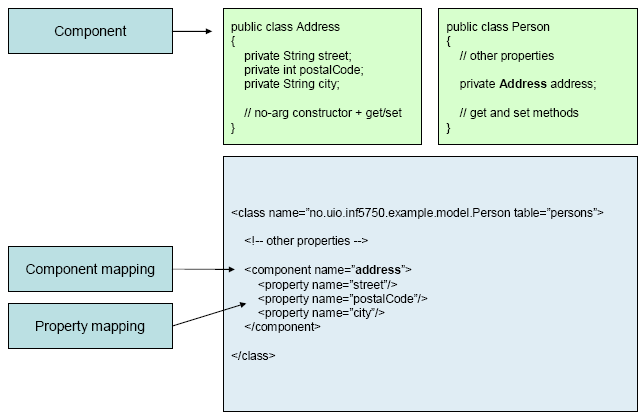
**27.What are derived properties?**

The properties that are not mapped to a column, but calculated at runtime by evaluation of an expression are called derived properties. The expression can be defined using the formula attribute of the element.

**28.What is component mapping in Hibernate?**

* A component is an object saved as a value, not as a reference
* A component can be saved directly without needing to declare interfaces or identifier properties
* Required to define an empty constructor
* Shared references not supported

**Example**:

  
  
**29.What is the difference between sorted and ordered collection in hibernate?**

**sorted collection vs. order collection** :-

|  |  |
| --- | --- |
| **sorted collection** | **order collection** |
| A sorted collection is sorting a collection by utilizing the sorting features provided by the Java collections framework. The sorting occurs in the memory of JVM which running Hibernate, after the data being read from database using java comparator. | Order collection is sorting a collection by specifying the order-by clause for sorting this collection when retrieval. |
| If your collection is not large, it will be more efficient way to sort it. | If your collection is very large, it will be more efficient way to sort it . |

**.What is the advantage of Hibernate over jdbc?**

Hibernate Vs. JDBC :-

|  |  |
| --- | --- |
| **JDBC** | **Hibernate** |
| With JDBC, developer has to write code to map an object model's data representation to a relational data model and its corresponding database schema. | Hibernate is flexible and powerful ORM solution to map Java classes to database tables. Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this. |
| With JDBC, the automatic mapping of Java objects with database tables and vice versa conversion is to be taken care of by the developer manually with lines of code. | Hibernate provides transparent persistence and developer does not need to write code explicitly to map database tables tuples to application objects during interaction with RDBMS. |
| JDBC supports only native Structured Query Language (SQL). Developer has to find out the efficient way to access database, i.e. to select effective query from a number of queries to perform same task. | Hibernate provides a powerful query language Hibernate Query Language (independent from type of database) that is expressed in a familiar SQL like syntax and includes full support for polymorphic queries. Hibernate also supports native [SQL statements](http://www.developersbook.com/hibernate/interview-questions/hibernate-interview-questions-faqs-3.php). It also selects an effective way to perform a database manipulation task for an application. |
| Application using JDBC to handle persistent data (database tables) having database specific code in large amount. The code written to map table data to application objects and vice versa is actually to map table fields to object properties. As table changed or database changed then it’s essential to change object structure as well as to change code written to map table-to-object/object-to-table. | Hibernate provides this mapping itself. The actual mapping between tables and application objects is done in XML files. If there is change in Database or in any table then the only need to change XML file properties. |
| With JDBC, it is developer’s responsibility to handle JDBC result set and convert it to Java objects through code to use this persistent data in application. So with JDBC, mapping between Java objects and database tables is done manually. | Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects. It relieves programmer from manual handling of persistent data, hence reducing the development time and maintenance cost. |
| With JDBC, caching is maintained by hand-coding. | Hibernate, with Transparent Persistence, cache is set to application work space. Relational tuples are moved to this cache as a result of query. It improves performance if client application reads same data many times for same write. Automatic Transparent Persistence allows the developer to concentrate more on business logic rather than this application code. |
| In JDBC there is no check that always every user has updated data. This check has to be added by the developer. | Hibernate enables developer to define version type field to application, due to this defined field Hibernate updates version field of database table every time relational tuple is updated in form of [Java class](http://www.developersbook.com/hibernate/interview-questions/hibernate-interview-questions-faqs-3.php) object to that table. So if two users retrieve same tuple and then modify it and one user save this modified tuple to database, version is automatically updated for this tuple by Hibernate. When other user tries to save updated tuple to database then it does not allow saving it because this user does not have updated data. |

**32.What are the Collection types in Hibernate ?**

* Bag
* Set
* List
* Array
* Map

**33.What are the ways to express joins in HQL?**

HQL provides four ways of expressing (inner and outer) joins:-

* An *implicit* association join
* An ordinary join in the FROM clause
* A fetch join in the FROM clause.
* A *theta-style* join in the WHERE clause.

**34.Define cascade and inverse option in one-many mapping?**

cascade - enable operations to cascade to child entities.  
cascade="all|none|save-update|delete|all-delete-orphan"  
  
inverse - mark this collection as the "inverse" end of a bidirectional association.  
inverse="true|false"   
Essentially "inverse" indicates which end of a relationship should be ignored, so when persisting a parent who has a collection of children, should you ask the parent for its list of children, or ask the children who the parents are?

**35.What is Hibernate proxy?**

The proxy attribute enables lazy initialization of persistent instances of the class. Hibernate will initially return CGLIB proxies which implement the named interface. The actual persistent object will be loaded when a method of the proxy is invoked.

**36.How can Hibernate be configured to access an instance variable directly and not through a setter method ?**

By mapping the property with access="field" in Hibernate metadata. This forces hibernate to bypass the setter method and access the instance variable directly while initializing a newly loaded object.

**37.How can a whole class be mapped as immutable?**

Mark the class as mutable="false" (Default is true),. This specifies that instances of the class are (not) mutable. Immutable classes, may not be updated or deleted by the application.

**38.What is the use of dynamic-insert and dynamic-update attributes in a class mapping?**

Criteria is a simplified API for retrieving entities by composing Criterion objects. This is a very convenient approach for functionality like "search" screens where there is a variable number of conditions to be placed upon the result set.

* dynamic-update (defaults to false): Specifies that UPDATE SQL should be generated at
* runtime and contain only those columns whose values have changed
* dynamic-insert (defaults to false): Specifies that INSERT SQL should be generated at runtime and contain only the columns whose values are not null.

**39.What do you mean by fetching strategy ?**

A *fetching strategy* is the strategy Hibernate will use for retrieving associated objects if the application needs to navigate the association. Fetch strategies may be declared in the O/R mapping metadata, or over-ridden by a particular HQL or Criteria query.

**40.What is automatic dirty checking?**

Automatic dirty checking is a feature that saves us the effort of explicitly asking Hibernate to update the database when we modify the state of an object inside a transaction.

**41.What is transactional write-behind?**

Hibernate uses a sophisticated algorithm to determine an efficient ordering that avoids database foreign key constraint violations but is still sufficiently predictable to the user. This feature is called transactional write-behind.

**42.What are Callback interfaces?**

Callback interfaces allow the application to receive a notification when something interesting happens to an object—for example, when an object is loaded, saved, or deleted. Hibernate applications don't need to implement these callbacks, but they're useful for implementing certain kinds of generic functionality.

**43.What are the types of Hibernate instance states ?**

Three types of instance states:

* Transient -The instance is not associated with any persistence context
* Persistent -The instance is associated with a persistence context
* Detached -The instance was associated with a persistence context which has been closed – currently not associated

**44.What are the differences between EJB 3.0 & Hibernate**

Hibernate Vs EJB 3.0 :-

|  |  |
| --- | --- |
| **Hibernate** | **EJB 3.0** |
| **Session**–Cache or collection of loaded objects relating to a single unit of work | **Persistence Context**-Set of entities that can be managed by a given EntityManager is defined by a persistence unit |
| **XDoclet Annotations** used to support Attribute Oriented Programming | [**Java**](http://www.developersbook.com/hibernate/interview-questions/hibernate-interview-questions-faqs-3.php) **5.0 Annotations** used to support Attribute Oriented Programming |
| **Defines HQL** for expressing queries to the database | **Defines EJB QL** for expressing queries |
| **Supports Entity Relationships** through mapping files and annotations in JavaDoc | **Support Entity Relationships** through Java 5.0 annotations |
| **Provides a Persistence Manager API** exposed via the Session, Query, Criteria, and Transaction API | **Provides and Entity Manager Interface** for managing CRUD operations for an Entity |
| **Provides callback support** through lifecycle, interceptor, and validatable interfaces | **Provides callback support** through Entity Listener and Callback methods |
| **Entity Relationships are unidirectional**. Bidirectional relationships are implemented by two unidirectional relationships | **Entity Relationships are bidirectional or unidirectional** |

**45.What are the types of inheritance models in Hibernate?**

There are three types of inheritance models in Hibernate:

* Table per class hierarchy
* Table per subclass
* Table per concrete class

**Difference between session.save() , session.saveOrUpdate() and session.persist()?**

session.save() : Save does an insert and will fail if the primary key is already persistent.   
  
session.saveOrUpdate() : saveOrUpdate does a select first to determine if it needs to do an insert or an update.   
Insert data if primary key not exist otherwise update data.   
  
session.persist() : Does the same like session.save().   
But session.save() return Serializable object but session.persist() return void.   
         session.save() returns the generated identifier (Serializable object) and session.persist() doesn't.   
For Example :   
         if you do :-   
         System.out.println(session.save(question));   
         This will print the generated primary key.   
         if you do :-   
         System.out.println(session.persist(question));   
         Compile time error because session.persist() return void.

**.What is the difference between hibernate and jdbc ?**

There are so many   
  
1) Hibernate is data base independent, your code will work for all ORACLE,MySQL ,SQLServer etc.   
  
In case of JDBC query must be data base specific.   
  
2) As Hibernate is set of Objects , you don?t need to learn SQL language.   
  
You can treat TABLE as a Object . Only Java knowledge is need.   
  
In case of JDBC you need to learn SQL.   
  
3) Don?t need Query tuning in case of Hibernate. If you use Criteria Quires in Hibernate then hibernate automatically tuned your query and return best result with performance.   
  
In case of JDBC you need to tune your queries.   
  
4) You will get benefit of Cache. Hibernate support two level of cache. First level and 2nd level. So you can store your data into Cache for better performance.   
  
In case of JDBC you need to implement your java cache .   
  
  
  
5) Hibernate supports Query cache and It will provide the statistics about your query and database status.   
  
JDBC Not provides any statistics.   
  
6) Development fast in case of Hibernate because you don?t need to write queries   
  
7) No need to create any connection pool in case of Hibernate. You can use c3p0.   
  
In case of JDBC you need to write your own connection pool   
  
8) In the xml file you can see all the relations between tables in case of Hibernate. Easy readability.   
  
9) You can load your objects on start up using lazy=false in case of Hibernate.   
  
JDBC Don?t have such support.   
  
  
  
10 ) Hibernate Supports automatic versioning of rows but JDBC Not.

### **What is lazy fetching in Hibernate? With Example .**

Lazy fetching decides whether to load child objects while loading the Parent Object.   
You need to do this setting respective hibernate mapping file of the parent class.   
Lazy = true (means not to load child)   
By default the lazy loading of the child objects is true.   
This make sure that the child objects are not loaded unless they are explicitly invoked in the application by calling getChild() method on parent.In this case hibernate issues a fresh database call to load the child when getChild() is actully called on the Parent object   
.But in some cases you do need to load the child objects when parent is loaded.   
Just make the lazy=false and hibernate will load the child when parent is loaded from the database.   
Example :   
If you have a TABLE ? EMPLOYEE mapped to Employee object and contains set of Address objects.   
Parent Class : Employee class   
Child class : Address Class   
public class Employee {   
private Set address = new HashSet(); // contains set of child Address objects   
public Set getAddress () {   
return address;   
}   
public void setAddresss(Set address) {   
this. address = address;   
}   
}   
In the Employee.hbm.xml file   
<set name="address" inverse="true" cascade="delete" lazy="false">   
<key column="a\_id" />   
<one-to-many class="beans Address"/>   
</set>   
In the above configuration.   
If lazy="false" : - when you load the Employee object that time child object Adress is also loaded and set to setAddresss() method.   
If you call employee.getAdress() then loaded data returns.No fresh database call.   
  
If lazy="true" :- This the default configuration. If you don?t mention then hibernate consider lazy=true.   
when you load the Employee object that time child object Adress is not loaded. You need extra call to data base to get address objects.   
If you call employee.getAdress() then that time database query fires and return results. Fresh database call.

**what is the advantage of Hibernate over jdbc?**

**what is the advantage of Hibernate over jdbc?**

There are so many   
1)    Hibernate is data base independent, your code will work for all ORACLE,MySQL ,SQLServer etc.   
In case of JDBC query must be data base specific.   
2)    As Hibernate is set of Objects , you don?t need to learn SQL language.   
You can treat TABLE as a Object . Only Java knowledge is need.   
In case of JDBC you need to learn SQL.   
3)    Don?t need Query tuning in case of Hibernate. If you use Criteria Quires in Hibernate then hibernate automatically tuned your query and return best result with performance.   
In case of JDBC you need to tune your queries.   
4)     You will get benefit of Cache. Hibernate support two level of cache. First level and 2nd level. So you can store your data into Cache for better performance.   
In case of JDBC you need to implement your java cache .   
  
5)    Hibernate supports Query cache and It will provide the statistics about your query and database status.   
JDBC Not provides any statistics.   
6)    Development fast in case of Hibernate because you don?t need to write queries   
7)    No need to create any connection pool in case of Hibernate. You can use c3p0.   
In case of JDBC you need to write your own connection pool   
8)    In the xml file you can see all the relations between tables in case of Hibernate. Easy readability.   
9)    You can load your objects on start up using lazy=false in case of Hibernate.   
JDBC Don?t have such support.   
  
10 ) Hibernate Supports automatic versioning of rows but JDBC Not.

### **How to prevent concurrent update in Hibernate?**

version checking used in hibernate when more then one thread trying to access same data.   
For example :   
User A edit the row of the TABLE for update ( In the User Interface changing data - This is user thinking time)   
and in the same time User B edit the same record for update and click the update.   
Then User A click the Update and update done. Chnage made by user B is gone.   
  
In hibernate you can perevent slate object updatation using version checking.   
  
Check the version of the row when you are upding the row.   
Get the version of the row when you are fetching the row of the TABLE for update.   
On the time of updation just fetch the version number and match with your version number ( on the time of fetching).   
  
This way you can prevent slate object updatation.   
  
Steps 1:   
Declare a variable "versionId" in your Class with setter and getter.   
public class Campign {   
private Long versionId;   
private Long campignId;   
private String name;   
public Long getVersionId() {   
return versionId;   
}   
public void setVersionId(Long versionId) {   
this.versionId = versionId;   
}   
  
public String getName() {   
return name;   
}   
public void setName(String name) {   
this.name = name;   
}   
  
public Long getCampignId() {   
        return campignId;   
    }   
private void setCampignId(Long campignId) {   
        this.campignId = campignId;   
    }   
  
}   
  
Step 2.   
In the .hbm.xml file   
<class name="beans.Campign" table="CAMPIGN" optimistic-lock="version">   
  
<id name="campignId" type="long" column="cid">   
<generator class="sequence">   
<param name="sequence">CAMPIGN\_ID\_SEQ</param>   
  
</generator>   
     </id>   
    <version name="versionId" type="long" column="version" />   
  
<property name="name" column="c\_name"/>   
  
</class>   
  
Step 3.   
Create a coulmn name "version" in the CAMPIGN table.   
  
Step 4.   
In the code   
// foo is an instance loaded by a previous Session   
session = sf.openSession();   
int oldVersion = foo.getVersion();   
session.load( foo, foo.getKey() );   
if ( oldVersion!=foo.getVersion ) throw new StaleObjectStateException();   
foo.setProperty("bar");   
session.flush();   
session.connection().commit();   
session.close();   
  
  
You can handle StaleObjectStateException() and do what ever you want.   
You can display error message.   
  
Hibernate autumatically create/update the version number when you update/insert any row in the table.

### **How to perevent slate object updatation in Hibernate ?**

version checking used in hibernate when more then one thread trying to access same data.   
For example :   
User A edit the row of the TABLE for update ( In the User Interface changing data - This is user thinking time)   
and in the same time User B edit the same record for update and click the update.   
Then User A click the Update and update done. Chnage made by user B is gone.   
  
In hibernate you can perevent slate object updatation using version checking.   
  
Check the version of the row when you are upding the row.   
Get the version of the row when you are fetching the row of the TABLE for update.   
On the time of updation just fetch the version number and match with your version number ( on the time of fetching).   
  
This way you can prevent slate object updatation.   
  
Steps 1:   
Declare a variable "versionId" in your Class with setter and getter.   
public class Campign {   
private Long versionId;   
private Long campignId;   
private String name;   
public Long getVersionId() {   
return versionId;   
}   
public void setVersionId(Long versionId) {   
this.versionId = versionId;   
}   
  
public String getName() {   
return name;   
}   
public void setName(String name) {   
this.name = name;   
}   
  
public Long getCampignId() {   
        return campignId;   
    }   
private void setCampignId(Long campignId) {   
        this.campignId = campignId;   
    }   
  
}   
  
Step 2.   
In the .hbm.xml file   
<class name="beans.Campign" table="CAMPIGN" optimistic-lock="version">   
  
<id name="campignId" type="long" column="cid">   
<generator class="sequence">   
<param name="sequence">CAMPIGN\_ID\_SEQ</param>   
  
</generator>   
     </id>   
    <version name="versionId" type="long" column="version" />   
  
<property name="name" column="c\_name"/>   
  
</class>   
  
Step 3.   
Create a coulmn name "version" in the CAMPIGN table.   
  
Step 4.   
In the code   
// foo is an instance loaded by a previous Session   
session = sf.openSession();   
int oldVersion = foo.getVersion();   
session.load( foo, foo.getKey() );   
if ( oldVersion!=foo.getVersion ) throw new StaleObjectStateException();   
foo.setProperty("bar");   
session.flush();   
session.connection().commit();   
session.close();   
  
  
You can handle StaleObjectStateException() and do what ever you want.   
You can display error message.   
  
Hibernate autumatically create/update the version number when you update/insert any row in the table.

### **What is version checking in Hibernate ?**

|  |
| --- |
| version checking used in hibernate when more then one thread trying to access same data.  For example :  User A edit the row of the TABLE for update ( In the User Interface changing data - This is user thinking time)  and in the same time User B edit the same record for update and click the update.  Then User A click the Update and update done. Chnage made by user B is gone.   In hibernate you can perevent slate object updatation using version checking.   Check the version of the row when you are upding the row.  Get the version of the row when you are fetching the row of the TABLE for update.  On the time of updation just fetch the version number and match with your version number ( on the time of fetching).   This way you can prevent slate object updatation.   Steps 1:  Declare a variable "versionId" in your Class with setter and getter.  public class Campign {  private Long versionId;  private Long campignId;  private String name;  public Long getVersionId() {  return versionId;  }  public void setVersionId(Long versionId) {  this.versionId = versionId;  }   public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }   public Long getCampignId() {          return campignId;      }  private void setCampignId(Long campignId) {          this.campignId = campignId;      }   }   Step 2.  In the .hbm.xml file  <class name="beans.Campign" table="CAMPIGN" optimistic-lock="version">   <id name="campignId" type="long" column="cid">  <generator class="sequence">  <param name="sequence">CAMPIGN\_ID\_SEQ</param>   </generator>       </id>      <version name="versionId" type="long" column="version" />   <property name="name" column="c\_name"/>   </class>   Step 3.  Create a coulmn name "version" in the CAMPIGN table.   Step 4.  In the code  // foo is an instance loaded by a previous Session  session = sf.openSession();  int oldVersion = foo.getVersion();  session.load( foo, foo.getKey() );  if ( oldVersion!=foo.getVersion ) throw new StaleObjectStateException();  foo.setProperty("bar");  session.flush();  session.connection().commit();  session.close();    You can handle StaleObjectStateException() and do what ever you want.  You can display error message.   Hibernate autumatically create/update the version number when you update/insert any row in the table. |

|  |
| --- |
|  |

**How to handle user think time using hibernate** ?

version checking used in hibernate when more then one thread trying to access same data.   
For example :   
User A edit the row of the TABLE for update ( In the User Interface changing data - This is user thinking time)   
and in the same time User B edit the same record for update and click the update.   
Then User A click the Update and update done. Chnage made by user B is gone.   
  
In hibernate you can perevent slate object updatation using version checking.   
  
Check the version of the row when you are upding the row.   
Get the version of the row when you are fetching the row of the TABLE for update.   
On the time of updation just fetch the version number and match with your version number ( on the time of fetching).   
  
This way you can prevent slate object updatation.   
  
Steps 1:   
Declare a variable "versionId" in your Class with setter and getter.   
public class Campign {   
private Long versionId;   
private Long campignId;   
private String name;   
public Long getVersionId() {   
return versionId;   
}   
public void setVersionId(Long versionId) {   
this.versionId = versionId;   
}   
  
public String getName() {   
return name;   
}   
public void setName(String name) {   
this.name = name;   
}   
  
public Long getCampignId() {   
        return campignId;   
    }   
private void setCampignId(Long campignId) {   
        this.campignId = campignId;   
    }   
  
}   
  
Step 2.   
In the .hbm.xml file   
<class name="beans.Campign" table="CAMPIGN" optimistic-lock="version">   
  
<id name="campignId" type="long" column="cid">   
<generator class="sequence">   
<param name="sequence">CAMPIGN\_ID\_SEQ</param>   
  
</generator>   
     </id>   
    <version name="versionId" type="long" column="version" />   
  
<property name="name" column="c\_name"/>   
  
</class>   
  
Step 3.   
Create a coulmn name "version" in the CAMPIGN table.   
  
Step 4.   
In the code   
// foo is an instance loaded by a previous Session   
session = sf.openSession();   
int oldVersion = foo.getVersion();   
session.load( foo, foo.getKey() );   
if ( oldVersion!=foo.getVersion ) throw new StaleObjectStateException();   
foo.setProperty("bar");   
session.flush();   
session.connection().commit();   
session.close();   
  
  
You can handle StaleObjectStateException() and do what ever you want.   
You can display error message.   
  
Hibernate autumatically create/update the version number when you update/insert any row in the table

**.Transaction with plain JDBC in Hibernate ?**

If you don't have JTA and don't want to deploy it along with your application, you will usually have to fall back to JDBC transaction demarcation. Instead of calling the JDBC API you better use Hibernate's Transaction and the built-in session-per-request functionality:   
  
To enable the thread-bound strategy in your Hibernate configuration:   
  
set hibernate.transaction.factory\_class to org.hibernate.transaction.JDBCTransactionFactory   
set hibernate.current\_session\_context\_class to thread   
  
Session session = factory.openSession();   
Transaction tx = null;   
try {   
tx = session.beginTransaction();   
  
// Do some work   
session.load(...);   
session.persist(...);   
  
tx.commit(); // Flush happens automatically   
}   
catch (RuntimeException e) {   
tx.rollback();   
throw e; // or display error message   
}   
finally {   
session.close();   
}

### **What are the general considerations or best practices for defining your Hibernate persistent classes?**

1.You must have a default no-argument constructor for your persistent classes and there should be getXXX() (i.e accessor/getter) and setXXX( i.e. mutator/setter) methods for all your persistable instance variables.   
  
2.You should implement the equals() and hashCode() methods based on your business key and it is important not to use the id field in your equals() and hashCode() definition if the id field is a surrogate key (i.e. Hibernate managed identifier). This is because the Hibernate only generates and sets the field when saving the object.   
  
  
3. It is recommended to implement the Serializable interface. This is potentially useful if you want to migrate around a multi-processor cluster.   
  
4.The persistent class should not be final because if it is final then lazy loading cannot be used by creating proxy objects.

**Difference between session.update() and session.lock() in Hibernate ?**

Both of these methods and saveOrUpdate() method are intended for reattaching a detached object.   
The session.lock() method simply reattaches the object to the session without checking or updating the database on the assumption that the database in sync with the detached object.   
It is the best practice to use either session.update(..) or session.saveOrUpdate().   
Use session.lock() only if you are absolutely sure that the   
detached object is in sync with your detached object or if it does not matter because   
you will be overwriting all the columns that would have changed later on within the same transaction.   
  
Each interaction with the persistent store occurs in a new Session. However, the same persistent instances are reused for each interaction with the database. The application manipulates the state of detached instances originally loaded in another Session and then "reassociates" them using Session.update() or Session.saveOrUpdate().   
  
// foo is an instance loaded by a previous Session   
foo.setProperty("bar");   
session = factory.openSession();   
session.saveOrUpdate(foo);   
session.flush();   
session.connection().commit();   
session.close();   
You may also call lock() instead of update() and use LockMode.READ (performing a version check, bypassing all caches) if you are sure that the object has not been modified.

.**Difference between getCurrentSession() and openSession() in Hibernate ?**

**getCurrentSession() :**The "current session" refers to a Hibernate Session bound by Hibernate behind the scenes, to the transaction scope.   
A Session is opened when getCurrentSession() is called for the first time and closed when the transaction ends.   
It is also flushed automatically before the transaction commits. You can call getCurrentSession() as often and anywhere you want as long as the transaction runs.   
To enable this strategy in your Hibernate configuration:   
  
set hibernate.transaction.manager\_lookup\_class to a lookup strategy for your JEE container   
set hibernate.transaction.factory\_class to org.hibernate.transaction.JTATransactionFactory   
  
Only the Session that you obtained with sf.getCurrentSession() is flushed and closed automatically.   
  
Example :   
try {   
UserTransaction tx = (UserTransaction)new InitialContext()   
.lookup("java:comp/UserTransaction");   
  
tx.begin();   
  
// Do some work   
sf.getCurrentSession().createQuery(...);   
sf.getCurrentSession().persist(...);   
  
tx.commit();   
}   
catch (RuntimeException e) {   
tx.rollback();   
throw e; // or display error message   
}   
  
openSession() :   
If you decide to use manage the Session yourself the go for sf.openSession() , you have to flush() and close() it.   
It does not flush and close() automatically.   
Example :   
UserTransaction tx = (UserTransaction)new InitialContext()   
.lookup("java:comp/UserTransaction");   
  
Session session = factory.openSession();   
  
try {   
tx.begin();   
  
// Do some work   
session.createQuery(...);   
session.persist(...);   
  
session.flush(); // Extra work you need to do   
  
tx.commit();   
}   
catch (RuntimeException e) {   
tx.rollback();   
throw e; // or display error message   
}   
finally {   
session.close(); // Extra work you need to do   
}

### **Difference between session.saveOrUpdate() and session.merge()?**

<b>saveOrUpdate() </b>does the following:   
  
? if the object is already persistent in this session, do nothing   
  
? if another object associated with the session has the same identifier, throw an exception   
  
? if the object has no identifier property, save() it   
  
? if the object's identifier has the value assigned to a newly instantiated object, save() it   
  
? if the object is versioned (by a <version> or <timestamp>), and the version property value is the same   
  
value assigned to a newly instantiated object, save() it   
  
? otherwise update() the object   
  
  
  
<b>merge() </b>is very different:   
  
? if there is a persistent instance with the same identifier currently associated with the session, copy the state of the given object onto the persistent instance   
  
? if there is no persistent instance currently associated with the session, try to load it from the database, or   
  
create a new persistent instance   
  
? the persistent instance is returned   
  
? the given instance does not become associated with the session, it remains detached

### **Filter in Hibernate with Example?**

Filter in Hibernate ------   
  
USER ( ID INT, USERNAME VARCHAR, ACTIVATED BOOLEAN) - TABLE   
  
  
  
public class User   
  
{   
  
private int id;   
  
private String username;   
  
private boolean activated;   
  
  
  
  
  
public boolean isActivated()   
  
{   
  
return activated;   
  
}   
  
public void setActivated(boolean activated)   
  
{   
  
this.activated = activated;   
  
}   
  
public int getId()   
  
{   
  
return id;   
  
}   
  
public void setId(int id)   
  
{   
  
this.id = id;   
  
}   
  
public String getUsername()   
  
{   
  
return username;   
  
}   
  
public void setUsername(String username)   
  
{   
  
this.username = username;   
  
}   
  
}   
  
-----------------------------------------------------------------   
  
<?xml version='1.0' encoding='utf-8'?>   
  
<!DOCTYPE hibernate-mapping   
  
PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"   
  
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">   
  
  
  
<hibernate-mapping>   
  
<class name="User">   
  
<id name="id" type="int">   
  
<generator class="native"/>   
  
</id>   
  
  
  
<property name="username" type="string" length="32"/>   
  
<property name="activated" type="boolean"/>   
  
<filter name="activatedFilter" condition=":activatedParam = activated"/>   
  
</class>   
  
<filter-def name="activatedFilter">   
  
<filter-param name="activatedParam" type="boolean"/>   
  
</filter-def>   
  
</hibernate-mapping>   
  
  
  
--------------------------------------------------------------------   
  
Save and Fetch using filter example   
  
  
  
User user1 = new User();   
  
user1.setUsername("name1");   
  
user1.setActivated(false);   
  
  
  
session.save(user1);   
  
  
  
User user2 = new User();   
  
user2.setUsername("name2");   
  
user2.setActivated(true);   
  
  
  
session.save(user2);   
  
  
  
User user3 = new User();   
  
user3.setUsername("name3");   
  
user3.setActivated(true);   
  
  
  
session.save(user3);   
  
  
  
User user4 = new User();   
  
user4.setUsername("name4");   
  
user4.setActivated(false);   
  
  
  
session.save(user4);   
  
  
  
All the four user saved to Data Base User Table.   
  
  
  
Now Fetch the User using Filter..   
  
  
  
Filter filter = session.enableFilter("activatedFilter");   
  
filter.setParameter("activatedParam",new Boolean(true));   
  
Query query = session.createQuery("from User");   
  
Iterator results = query.iterate();   
  
while (results.hasNext())   
  
{   
  
User user = (User) results.next();   
  
System.out.print(user.getUsername() + " is ");   
  
  
  
}   
  
  
  
  
  
Guess the Result :   
  
name2 name3   
  
  
  
Because Filer is filtering ( only true value) data before query execute.

**Q.How does Value replacement in Message Resource Bundle work?**

In the resource bundle file, you can define a template like:   
errors.required={0} is required.   
ActionErrors errors = new ActionErrors();   
errors.add(ActionErrors.GLOBAL\_ERROR,   
new ActionError("error.custform","First Name"));   
  
Then the Error message is : First Name is required.   
  
Other constructors are   
  
public ActionError(String key, Object value0, Object value1)   
. . .   
public ActionError(String key, Object[] values);

**Difference between list() and iterate() i9n Hibernate?**

If instances are already be in the session or second-level cache iterate() will give better performance.   
If they are not already cached, iterate() will be slower   
than list() and might require many database hits for a simple query.

### **Difference between session.load() and session.get() ?**

load() will throw an unrecoverable exception if there is no matching database row.   
get() will return null if there is no matching database row.   
  
  
Cat fritz = (Cat) session.load(Cat.class, "1");   
  
Return the Cat Object with key 1. If there is no Cat Object with key 1 then throw will throw an unrecoverable exception.   
  
If the class is mapped with a proxy, load() just returns an uninitialized proxy and does not actually hit the database until you invoke a method of the proxy. This behaviour is very useful if you wish to create an association to an object without actually loading it from the database. It also allows multiple instances to be loaded as a batch if batchsize is defined for the class mapping.   
  
  
Cat fritz = (Cat) session.get(Cat.class, "1");   
If you are not certain that a matching row exists, you should use the get() method, which hits the database immediately and returns null if there is no matching row.

Deleting persistent objects Session.delete() will remove an object's state from the database. Of course, your application might still hold a reference to a deleted object. It's best to think of delete() as making a persistent instance transient. sess.delete(cat);

**SQL statements execution order.**

* 1. all entity insertions, in the same order the corresponding objects were saved using Session.save()   
     2. all entity updates   
     3. all collection deletions   
     4. all collection element deletions, updates and insertions   
     5. all collection insertions   
     6. all entity deletions, in the same order the corresponding objects were deleted using Session.delete()

**Modifying persistent objects?**

DomesticCat cat = (DomesticCat) sess.load( Cat.class, new Long(69) );   
cat.setName("PK");   
sess.flush(); // changes to cat are automatically detected and persisted To Data Base.   
  
No need any session.update() call.

**SQL Queries In Hibernate..**

You may express a query in SQL, using createSQLQuery() and let Hibernate take care of the mapping from result sets to objects. Note that you may at any time call session.connection() and use the JDBC Connection directly. If you chose to use the Hibernate API, you must enclose SQL aliases in braces:   
List cats = session.createSQLQuery(   
"SELECT {cat.\*} FROM CAT {cat} WHERE ROWNUM<10",   
"cat",   
Cat.class   
).list();   
List cats = session.createSQLQuery(   
"SELECT {cat}.ID AS {cat.id}, {cat}.SEX AS {cat.sex}, " +   
"{cat}.MATE AS {cat.mate}, {cat}.SUBCLASS AS {cat.class}, ... " +   
"FROM CAT {cat} WHERE ROWNUM<10",   
"cat",   
Cat.class   
).list()   
SQL queries may contain named and positional parameters, just like Hibernate queries.

**Criteria Query Two Condition**

Criteria Query Two Condition- Example   
<class name="com.bean.Organization" table="ORGANIZATION">   
<id name="orgId" column="ORG\_ID" type="long">   
<generator class="native"/>   
</id>   
<property name="organizationName" column="ORGANISATION\_NAME" type="string" length="500"/>   
<property name="town" column="TOWN" type="string" length="200"/>   
<property name="statusCode" column="STATUS" type="string" length="1"/>   
</class>   
List of organisation where town equals to pune and status = "A".   
  
List organizationList = session.createCriteria(Organization.class)   
.add(Restrictions.eq("town","pune"))   
.add(Restrictions.eq("statusCode","A"))   
.list();

**Equal and Not Equal criteria query**.

Equal and Not Equal criteria query- Example   
<class name="com.bean.Organization" table="ORGANIZATION">   
<id name="orgId" column="ORG\_ID" type="long">   
<generator class="native"/>   
</id>   
<property name="organizationName" column="ORGANISATION\_NAME" type="string" length="500"/>   
<property name="town" column="TOWN" type="string" length="200"/>   
</class>   
List of organisation where town equals to pune.   
  
List organizationList = session.createCriteria(Organization.class).add(Restrictions.eq("town","pune")).list();   
  
List of organisation where town not equals pune.   
  
List organizationList = session.createCriteria(Organization.class).add(Restrictions.ne("town","pune")).list();

**Cascade Save or Update in Hibernate** ?

Cascade Save or Update - In one to Many- EXAMPLE   
PROCESS\_TYPE\_LOV (PROCESS\_TYPE\_ID number, PROCESS\_TYPE\_NAME varchar) - TABLE   
PROCESS (PROCESS\_ID number,PROCESS\_NAME varchar,PROCESS\_TYPE\_ID number)- TABLE   
  
public class ProcessTypeBean {   
       
    private Long processTypeId;   
    private String processTypeName;   
  
           
    /\*\*   
     \* @return Returns the processTypeId.   
     \*/   
    public Long getProcessTypeId() {   
        return processTypeId;   
    }   
    /\*\*   
     \* @param processTypeId The processTypeId to set.   
     \*/   
    public void setProcessTypeId(Long processTypeId) {   
        this.processTypeId = processTypeId;   
    }   
    /\*\*   
     \* @return Returns the processTypeName.   
     \*/   
    public String getProcessTypeName() {   
        return processTypeName;   
    }   
    /\*\*   
     \* @param processTypeName The processTypeName to set.   
     \*/   
    public void setProcessTypeName(String processTypeName) {   
        this.processTypeName = processTypeName;   
    }   
       
}   
  
public class ProcessBean {   
       
    private Long processId;   
    private String processName = "";   
    private ProcessTypeBean processType;   
  
    public Long getProcessId() {   
        return processId;   
    }   
    /\*\*   
     \* @param processId The processId to set.   
     \*/   
    public void setProcessId(Long processId) {   
        this.processId = processId;   
    }   
    /\*\*   
     \* @return Returns the processName.   
     \*/   
    public String getProcessName() {   
        return processName;   
    }   
    /\*\*   
     \* @param processName The processName to set.   
     \*/   
    public void setProcessName(String processName) {   
        this.processName = processName;   
    }   
    /\*\*   
     \* @return Returns the processType.   
     \*/   
    public ProcessTypeBean getProcessType() {   
        return processType;   
    }   
    /\*\*   
     \* @param processType The processType to set.   
     \*/   
    public void setProcessType(ProcessTypeBean processType) {   
        this.processType = processType;   
    }   
}   
  
<class name="com.bean.ProcessBean"   
        table="PROCESS">   
        <id name="processId" type="long" column="PROCESS\_ID" />   
        <property name="processName" column="PROCESS\_NAME" type="string"   
            length="50" />   
        <many-to-one name="processType" column="PROCESS\_TYPE\_ID" class="ProcessTypeBean" cascade="save-update" />   
           
</class>   
  
<class name="com.bean.ProcessTypeBean"   
        table="PROCESS\_TYPE\_LOV">   
        <id name="processTypeId" type="long" column="PROCESS\_TYPE\_ID" />   
        <property name="processTypeName" column="PROCESS\_TYPE\_NAME"   
            type="string" length="50" />   
           
    </class>   
---------------------------------------------------------------------------------   
Save Example Code -   
  
ProcessTypeBean pstype = new ProcessTypeBean();   
pstype.setProcessTypeName("Java Process");   
  
ProcessBean process = new ProcessBean();   
process.setProcessName("Production")   
ProcessBean.setProcessType(pstype);   
  
// session.save(pstype); -- This save not required because of in the mapping file cascade="save-update"   
session.save(process); - This will insert both ProcessBean and ProcessTypeBean;

**One To Many Bi-directional Relation in Hibernate?**

Bi-DireCtional One to Many Relation- EXAMPLE   
PROCESS\_TYPE\_LOV (PROCESS\_TYPE\_ID number, PROCESS\_TYPE\_NAME varchar) - TABLE   
PROCESS (PROCESS\_ID number,PROCESS\_NAME varchar,PROCESS\_TYPE\_ID number)- TABLE   
  
public class ProcessTypeBean {   
       
    private Long processTypeId;   
    private String processTypeName;   
private List processes = null;   
  
    /\*\*   
     \* @return Returns the processes.   
     \*/   
    public List getProcesses() {   
        return processes;   
    }   
    /\*\*   
     \* @param processes The processes to set.   
     \*/   
    public void setProcesses(List processes) {   
        this.processes = processes;   
    }   
       
    /\*\*   
     \* @return Returns the processTypeId.   
     \*/   
    public Long getProcessTypeId() {   
        return processTypeId;   
    }   
    /\*\*   
     \* @param processTypeId The processTypeId to set.   
     \*/   
    public void setProcessTypeId(Long processTypeId) {   
        this.processTypeId = processTypeId;   
    }   
    /\*\*   
     \* @return Returns the processTypeName.   
     \*/   
    public String getProcessTypeName() {   
        return processTypeName;   
    }   
    /\*\*   
     \* @param processTypeName The processTypeName to set.   
     \*/   
    public void setProcessTypeName(String processTypeName) {   
        this.processTypeName = processTypeName;   
    }   
       
}   
  
public class ProcessBean {   
       
    private Long processId;   
    private String processName = "";   
    private ProcessTypeBean processType;   
  
    public Long getProcessId() {   
        return processId;   
    }   
    /\*\*   
     \* @param processId The processId to set.   
     \*/   
    public void setProcessId(Long processId) {   
        this.processId = processId;   
    }   
    /\*\*   
     \* @return Returns the processName.   
     \*/   
    public String getProcessName() {   
        return processName;   
    }   
    /\*\*   
     \* @param processName The processName to set.   
     \*/   
    public void setProcessName(String processName) {   
        this.processName = processName;   
    }   
    /\*\*   
     \* @return Returns the processType.   
     \*/   
    public ProcessTypeBean getProcessType() {   
        return processType;   
    }   
    /\*\*   
     \* @param processType The processType to set.   
     \*/   
    public void setProcessType(ProcessTypeBean processType) {   
        this.processType = processType;   
    }   
}   
  
<class name="com.bean.ProcessBean"   
        table="PROCESS">   
        <id name="processId" type="long" column="PROCESS\_ID" />   
        <property name="processName" column="PROCESS\_NAME" type="string"   
            length="50" />   
        <many-to-one name="processType" column="PROCESS\_TYPE\_ID" lazy="false" />   
           
</class>   
  
<class name="com.bean.ProcessTypeBean"   
        table="PROCESS\_TYPE\_LOV">   
        <id name="processTypeId" type="long" column="PROCESS\_TYPE\_ID" />   
        <property name="processTypeName" column="PROCESS\_TYPE\_NAME"   
            type="string" length="50" />   
           
        <bag name="processes" inverse="true" cascade="delete" lazy="false">   
            <key column="PROCESS\_TYPE\_ID" />   
            <one-to-many   
                class="com.bean.ProcessBean" />   
        </bag>   
  
    </class>

**One To Many Mapping Using List ?**

WRITER (ID INT,NAME VARCHAR) - TABLE   
STORY (ID INT,INFO VARCHAR,PARENT\_ID INT) - TABLE   
One writer can have multiple stories..   
-------------------------------------------------------------   
Mapping File...   
<?xml version="1.0" encoding="UTF-8"?>   
<!DOCTYPE hibernate-mapping   
PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"   
"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">   
  
<hibernate-mapping>   
<class name="Writer" table="WRITER">   
<id name="id" unsaved-value="0">   
<generator class="increment"/>   
</id>   
  
<list name="stories" cascade="all">   
<key column="parent\_id"/>   
<one-to-many class="Story"/>   
</list>   
<property name="name" type="string"/>   
</class>   
<class name="Story"   
table="story">   
<id name="id" unsaved-value="0">   
<generator class="increment"/>   
</id>   
<property name="info"/>   
</class>   
</hibernate-mapping>   
-------------------------------------------------------   
public class Writer {   
private int id;   
private String name;   
private List stories;   
  
  
public void setId(int i) {   
id = i;   
}   
  
public int getId() {   
return id;   
}   
  
public void setName(String n) {   
name = n;   
}   
  
public String getName() {   
return name;   
}   
  
public void setStories(List l) {   
stories = l;   
}   
  
public List getStories() {   
return stories;   
}   
}   
  
---------------------------------------------------   
public class Story {   
private int id;   
private String info;   
  
public Story(){   
}   
  
public Story(String info) {   
this.info = info;   
}   
  
public void setId(int i) {   
id = i;   
}   
  
public int getId() {   
return id;   
}   
  
public void setInfo(String n) {   
info = n;   
}   
  
public String getInfo() {   
return info;   
}   
}   
----------------------------------------------------   
  
Save Example ..   
Writer wr = new Writer();   
wr.setName("Das");   
  
ArrayList list = new ArrayList();   
list.add(new Story("Story Name 1"));   
list.add(new Story("Story Name 2"));   
wr.setStories(list);   
  
Transaction transaction = null;   
  
try {   
transaction = session.beginTransaction();   
session.save(sp);   
transaction.commit();   
} catch (Exception e) {   
if (transaction != null) {   
transaction.rollback();   
throw e;   
}   
} finally {   
session.close();   
}

**Many To Many Relation In Hibernate ?**

Best Example..for Many to Many in Hibernate ..   
EVENTS ( uid int, name VARCHAR) Table   
SPEAKERS ( uid int, firstName VARCHAR) Table   
EVENT\_SPEAKERS (elt int, event\_id int, speaker\_id int) Table   
-----------------------------------------------------------   
import java.util.Set;   
import java.util.HashSet;   
  
public class Speaker{   
  
private Long id;   
private String firstName;   
private Set events;   
  
public Long getId() {   
return id;   
}   
  
public void setId(Long id) {   
this.id = id;   
}   
  
public String getFirstName() {   
return firstName;   
}   
  
public void setFirstName(String firstName) {   
this.firstName = firstName;   
}   
  
public Set getEvents() {   
return this.events;   
}   
  
public void setEvents(Set events) {   
this.events = events;   
}   
  
private void addEvent(Event event) {   
if (events == null) {   
events = new HashSet();   
}   
events.add(event);   
}   
}   
--------------------------------------------------------   
import java.util.Date;   
import java.util.Set;   
  
public class Event{   
  
private Long id;   
private String name;   
private Set speakers;   
  
public void setId(Long id) {   
this.id = id;   
}   
  
public Long getId() {   
return id;   
}   
  
public String getName() {   
return name;   
}   
  
public void setName(String name) {   
this.name = name;   
}   
  
public void setSpeakers(Set speakers) {   
this.speakers = speakers;   
}   
  
public Set getSpeakers() {   
return speakers;   
}   
  
}   
--------------------------------------------------------------   
Event.hbm.xml   
<?xml version="1.0"?>   
<!DOCTYPE hibernate-mapping PUBLIC   
"-//Hibernate/Hibernate Mapping DTD 2.0//EN"   
"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">   
  
<hibernate-mapping>   
<class name="Event" table="events">   
<id name="id" column="uid" type="long" unsaved-value="null">   
<generator class="increment"/>   
</id>   
<property name="name" type="string" length="100"/>   
<set name="speakers" table="event\_speakers" cascade="all">   
<key column="event\_id"/>   
<many-to-many class="Speaker"/>   
</set>   
</class>   
</hibernate-mapping>   
------------------------------------------------------------------   
Speaker.hbm.xml   
  
<?xml version="1.0"?>   
<!DOCTYPE hibernate-mapping PUBLIC   
"-//Hibernate/Hibernate Mapping DTD 2.0//EN"   
"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">   
  
<hibernate-mapping>   
<class name="Speaker" table="speakers">   
<id name="id" column="uid" type="long">   
<generator class="increment"/>   
</id>   
<property name="firstName" type="string" length="20"/>   
<set name="events" table="event\_speakers" cascade="all">   
<key column="speaker\_id"/>   
<many-to-many class="Event"/>   
</set>   
</class>   
</hibernate-mapping>   
----------------------------------------------------------------------   
Save and Fetch Example   
Event event = new Event();   
event.setName("Inverse test");   
event.setSpeakers(new HashSet());   
event.getSpeakers().add(new Speaker("Ram", event));   
event.getSpeakers().add(new SpeakerManyToMany("Syam", event));   
event.getSpeakers().add(new SpeakerManyToMany("Jadu", event));   
session.save(event); /// Save All the Data   
  
event = (Event) session.load(Event.class, event.getId());   
Set speakers = event.getSpeakers();   
  
for (Iterator i = speakers.iterator(); i.hasNext();) {   
Speaker speaker = (Speaker) i.next();   
System.out.println(speaker.getFirstName());   
System.out.println(speaker.getId());   
}

**What does session.refresh() do ?**

It is possible to re-load an object and all its collections at any time, using the refresh() method. This is useful   
when database triggers are used to initialize some of the properties of the object.   
For Example - Triger on cat name coulmn. Trigger is updating hit\_count coulmn in the same Cat Table. When Insert data into Cat TABLE trigger update hit\_count coulmn to 1. sess.refresh() reload all the data. No need again to select call.   
sess.save(cat);   
sess.flush(); //force the SQL INSERT   
sess.refresh(cat); //re-read the state (after the trigger executes)

Hibernate setup using .cfg.xml file

The XML configuration file is by default expected to be in the root o your CLASSPATH. Here is an example:   
<?xml version='1.0' encoding='utf-8'?>   
<!DOCTYPE hibernate-configuration PUBLIC   
"-//Hibernate/Hibernate Configuration DTD//EN"   
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">   
<hibernate-configuration>   
<!-- a SessionFactory instance listed as /jndi/name -->   
<session-factory   
name="java:hibernate/SessionFactory">   
<!-- properties -->   
<property name="connection.datasource">java:/comp/env/jdbc/MyDB</property>   
<property name="dialect">org.hibernate.dialect.MySQLDialect</property>   
<property name="show\_sql">false</property>   
<property name="transaction.factory\_class">   
org.hibernate.transaction.JTATransactionFactory   
</property>   
<property name="jta.UserTransaction">java:comp/UserTransaction</property>   
<!-- mapping files -->   
<mapping resource="org/hibernate/auction/Cost.hbm.xml"/>   
</session-factory>   
</hibernate-configuration>   
As you can see, the advantage of this approach is the externalization of the mapping file names to configuration.   
The hibernate.cfg.xml is also more convenient once you have to tune the Hibernate cache. Note that is your choice to use either hibernate.properties or hibernate.cfg.xml, both are equivalent, except for the above mentioned benefits of using the XML syntax.   
With the XML configuration, starting Hibernate is then as simple as   
SessionFactory sf = new Configuration().configure().buildSessionFactory();   
You can pick a different XML configuration file using   
SessionFactory sf = new Configuration() .configure("catdb.cfg.xml")   
.buildSessionFactory();   
  
Cost.hbm.xml -----> looks like   
  
<?xml version="1.0"?>   
<!DOCTYPE hibernate-mapping PUBLIC   
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"   
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">   
  
<hibernate-mapping>   
<class name="com.bean.Cost" table="COST">   
  
<id name="id" column="ID">   
</id>   
<property name="isQueued" type="int" column="IS\_QUEUED"/>   
<property name="queueDate" type="timestamp" column="QUEUE\_DATE"/>   
<property name="lastModifiedDate" type="timestamp" column="LAST\_MODIFIED\_DATE"/>   
<property name="lastModifiedBy" column="LAST\_MODIFIED\_BY"/>   
<property name="amount" column="AMOUNT" type="double"/>   
<property name="currencyCode" column="CURRENCY\_CODE" />   
<property name="year" column="YEAR"/>   
<property name="quarter" column="QUARTER"/>   
<property name="costModFlag" type="int" column="COST\_MOD\_FLAG"/>   
<property name="parentId" column="PARENT\_ID"/>   
<property name="oldParentId" column="OLD\_PARENT\_ID"/>   
<property name="parentIdModFlag" type="int" column="PARENT\_ID\_MOD\_FLAG"/>   
<property name="dateIncurred" type="timestamp" column="DATE\_INCURRED"/>   
<property name="USDAmount" column="USD\_AMOUNT" type="double"/>   
<property name="isDeleted" type="int" column="IS\_DELETED"/>   
</class>   
  
</hibernate-mapping>

How to add .hbm.xml file in sessionFactory?

SessionFactory sf = new Configuration()   
.addFile("Item.hbm.xml")   
.addFile("Bid.hbm.xml")   
.buildSessionFactory();

**How to set 2nd level cache in hibernate with EHCache?**

When you are creating SessionFactory just add the below steps   
  
String ecache = appHome+File.separatorChar+"ehcache.xml";   
try {   
CacheManager.create(ecache);   
} catch (CacheException e) {   
// logger.logError(e);   
}\*/   
  
Then   
sessionFactory = configuration.buildSessionFactory();   
  
ECache.xml is like   
<ehcache>   
<diskStore path="java.io.tmpdir"/>   
<defaultCache   
maxElementsInMemory="10000"   
eternal="false"   
timeToIdleSeconds="120"   
timeToLiveSeconds="120"   
overflowToDisk="true"   
diskPersistent="false"   
diskExpiryThreadIntervalSeconds="120"   
/>   
  
<cache name="bean.ApplicationBean"   
maxElementsInMemory="300"   
eternal="false"   
overflowToDisk="false"   
/>   
</ehcache>   
  
ApplicationBean will be avilable in 2nd level cache

How will you configure Hibernate?

Step 1> Put Hibernate properties in the classpath.   
Step 2> Put .hbm.xml in class path ?   
Code is Here to create session ...   
  
  
package com.dao;   
  
  
  
  
import java.io.File;   
import java.io.FileInputStream;   
  
import java.util.Properties;   
import org.apache.log4j.Logger;   
import org.hibernate.HibernateException;   
import org.hibernate.Session;   
import org.hibernate.SessionFactory;   
import org.hibernate.Transaction;   
import org.hibernate.cfg.Configuration;   
  
/\*\*   
\*   
\* @author Satya Das   
\*/   
public class HibernateUtil {   
    protected static final Logger logger=Logger.getLogger(HibernateUtil.class);   
    public static String appHome = "No";   
       
       
    private static SessionFactory sessionFactory;   
       
       
    private static final ThreadLocal threadSession = new ThreadLocal();   
    private static final ThreadLocal threadTransaction = new ThreadLocal();   
       
       
       
    /\*\*   
     \* Initialize Hibernate Configuration   
     \*/   
    public static void initMonitor(){   
        logger.info("Hibernate configure");   
        try {   
            logger.info("appHome"+appHome);   
            String path\_properties = appHome+File.separatorChar+"hibernate.properties";   
            String path\_mapping = appHome+File.separatorChar+"mapping\_classes.mysql.hbm.xml";   
            //String ecache = appHome+File.separatorChar+"ehcache.xml";   
               
               
            Properties propHibernate = new Properties();   
            propHibernate.load(new FileInputStream(path\_properties));   
               
            Configuration configuration = new Configuration();   
            configuration.addFile(path\_mapping);   
            configuration.setProperties(propHibernate);   
               
            /\* try {   
CacheManager.create(ecache);   
} catch (CacheException e) {   
// logger.logError(e);   
}\*/   
               
            sessionFactory = configuration.buildSessionFactory();   
               
               
               
        } catch (Throwable ex) {   
            logger.error("Exception in initMonitor",ex);   
            throw new ExceptionInInitializerError(ex);   
        }   
    }   
    /\*\*   
     \* @return a Session Factory Object   
     \*/   
    public static SessionFactory getSessionFactory() {   
        logger.info("Inside getSessionFactory method");   
        try {   
               
            if (sessionFactory == null) {   
                initMonitor();   
            }else {   
                   
                //sessionFactory.getStatistics().logSummary();   
            }   
               
               
        } catch (Exception e) {   
            logger.error("Exception in getSessionFactory",e);   
        }   
           
        return sessionFactory;   
    }   
       
    /\*\*   
     \* @return Session . Start a Session   
     \*/   
    public static Session getSession() {   
           
        Session s = (Session) threadSession.get();   
        logger.debug("session"+s);   
        if (s == null) {   
               
            s = getSessionFactory().openSession();   
            threadSession.set(s);   
            logger.debug("session 1 $"+s);   
        }   
        return s;   
    }   
       
    /\*\*   
     \* Close Session   
     \*/   
    public static void closeSession(){   
           
        Session s = (Session) threadSession.get();   
        threadSession.set(null);   
        if (s != null && s.isOpen()) {   
            s.flush();   
            s.close();   
        }   
    }   
       
       
    /\*\*   
     \* Start a new database transaction.   
     \*/   
    public static void beginTransaction(){   
        Transaction tx = null;   
           
        if (tx == null) {   
            tx = getSession().beginTransaction();   
            threadTransaction.set(tx);   
        }   
    }   
       
    /\*\*   
     \* Commit the database transaction.   
     \*/   
    public static void commitTransaction(){   
        Transaction tx = (Transaction) threadTransaction.get();   
        try {   
            if ( tx != null ) {   
                tx.commit();   
            }   
               
            threadTransaction.set(null);   
               
        } catch (HibernateException ex) {   
            rollbackTransaction();   
               
            throw ex;   
        }   
    }   
       
       
    /\*\*   
     \* Rollback the database transaction.   
     \*/   
    public static void rollbackTransaction(){   
           
        Transaction tx = (Transaction) threadTransaction.get();   
        try {   
            threadTransaction.set(null);   
            if ( tx != null && !tx.wasCommitted() && !tx.wasRolledBack() ) {   
                tx.rollback();   
            }   
        } finally {   
            closeSession();   
        }   
    }   
       
       
       
       
}

**What are the Instance states in Hibernate?**

|  |
| --- |
|  |

transient   
The instance is not, and has never been associated with any persistence context. It has no persistent identity   
(primary key value).   
persistent   
The instance is currently associated with a persistence context. It has a persistent identity (primary key value) and, perhaps, a corresponding row in the database. For a particular persistence context, Hibernate guarantees that persistent identity is equivalent to Java identity (in-memory location of the object).   
detached   
The instance was once associated with a persistence context, but that context was closed, or the instance was serialized to another process. It has a persistent identity and, , perhaps a corrsponding row in the database. For detached instances, Hibernate makes no guarantees about the relationship between persistent identity and Java identity

**What are the core components in Hibernate ?**

SessionFactory (org.hibernate.SessionFactory)   
A threadsafe (immutable) cache of compiled mappings for a single database. A factory for Session and a   
client of ConnectionProvider. Might hold an optional (second-level) cache of data that is reusable   
between transactions, at a process- or cluster-level.   
Session (org.hibernate.Session)   
A single-threaded, short-lived object representing a conversation between the application and the persistent   
store. Wraps a JDBC connection. Factory for Transaction. Holds a mandatory (first-level) cache of persistent   
objects, used when navigating the object graph or looking up objects by identifier.   
Persistent objects and collections   
Short-lived, single threaded objects containing persistent state and business function. These might be ordinary   
JavaBeans/POJOs, the only special thing about them is that they are currently associated with (exactly   
one) Session. As soon as the Session is closed, they will be detached and free to use in any application   
layer (e.g. directly as data transfer objects to and from presentation).   
Transient and detached objects and collections   
Instances of persistent classes that are not currently associated with a Session. They may have been instantiated   
by the application and not (yet) persisted or they may have been instantiated by a closed Session.   
Transaction (org.hibernate.Transaction)   
(Optional) A single-threaded, short-lived object used by the application to specify atomic units of work.   
Abstracts application from underlying JDBC, JTA or CORBA transaction. A Session might span several   
Transactions in some cases. However, transaction demarcation, either using the underlying API or Transaction,   
is never optional!   
Architecture   
Hibernate 3.0.2 9   
ConnectionProvider (org.hibernate.connection.ConnectionProvider)   
(Optional) A factory for (and pool of) JDBC connections. Abstracts application from underlying Datasource   
or DriverManager. Not exposed to application, but can be extended/implemented by the developer.   
TransactionFactory (org.hibernate.TransactionFactory)   
(Optional) A factory for Transaction instances. Not exposed to the application, but can be extended/   
implemented by the developer.   
Extension Interfaces   
Hibernate offers many optional extension interfaces you can implement to customize the behavior of your   
persistence layer. See the API documentation for details.

**What is a Hibernate Session? Can you share a session object between different theads?**

|  |
| --- |
| Session is a light weight and a non-threadsafe object (No, you cannot share it between threads) that represents a single unit-of-work with the database. Sessions are opened by a SessionFactory and then are closed when all work is complete. Session is the primary interface for the persistence service. A session obtains a database connection lazily (i.e. only when required). To avoid creating too many sessions ThreadLocal class can be used as shown below to get the current session no matter how many times you make call to the currentSession() method.   ?  public class HibernateUtil {  ?  public static final ThreadLocal local = new ThreadLocal();   public static Session currentSession() throws HibernateException {  Session session = (Session) local.get();  //open a new session if this thread has no session  if(session == null) {  session = sessionFactory.openSession();  local.set(session);  }  return session;  }  } |

|  |
| --- |
|  |

addScalar() method in hibernate

Double max = (Double) sess.createSQLQuery("select max(cat.weight) as maxWeight from cats cat")   
.addScalar("maxWeight", Hibernate.DOUBLE);   
.uniqueResult();   
  
addScalar() method confim that maxWeight is always double type.   
  
This way you don't need to check for it is double or not.

Hibernate session.close does \_not\_ call session.flush ?

session.close() don't call session.flush() before closing the session.   
  
This is the session.close() code in hibernate.jar   
  
public Connection close() throws HibernateException {   
        log.trace( "closing session" );   
        if ( isClosed() ) {   
            throw new SessionException( "Session was already closed" );   
        }   
           
  
        if ( factory.getStatistics().isStatisticsEnabled() ) {   
            factory.getStatisticsImplementor().closeSession();   
        }   
  
        try {   
            try {   
                if ( childSessionsByEntityMode != null ) {   
                    Iterator childSessions = childSessionsByEntityMode.values().iterator();   
                    while ( childSessions.hasNext() ) {   
                        final SessionImpl child = ( SessionImpl ) childSessions.next();   
                        child.close();   
                    }   
                }   
            }   
            catch( Throwable t ) {   
                // just ignore   
            }   
  
            if ( rootSession == null ) {   
                return jdbcContext.getConnectionManager().close();   
            }   
            else {   
                return null;   
            }   
        }   
        finally {   
            setClosed();   
            cleanup();   
        }   
    }

**What is the main difference between Entity Beans and Hibernate ?**

)In Entity Bean at a time we can interact with only one data Base. Where as in Hibernate we can able to establishes the connections to more than One Data Base. Only thing we need to write one more configuration file.   
  
2) EJB need container like Weblogic, WebSphare but hibernate don't nned. It can be run on tomcat.   
  
  
3) Entity Beans does not support OOPS concepts where as Hibernate does.   
  
4) Hibernate supports multi level cacheing, where as Entity Beans doesn't.   
  
5) In Hibernate C3P0 can be used as a connection pool.   
  
6) Hibernate is container independent. EJB not.

**How are joins handled using Hinernate.**

|  |
| --- |
| Best is use Criteria query  Example -  You have parent class  public class Organization {  private long orgId;  private List messages;  }  Child class  public class Message {      private long messageId;  private Organization organization;  }   .hbm.xml file   <class name="com.bean.Organization" table="ORGANIZATION">  <bag name="messages" inverse="true" cascade="delete" lazy="false">              <key column="MSG\_ID" />              <one-to-many                  class="com.bean.Message" />          </bag>     </class>   <class name="com.bean.Message" table="MESSAGE">      <many-to-one name="organization" column="ORG\_ID" lazy="false"/>   </class>  Get all the messages from message table where organisation id = <any id>   Criteria query is :  session.createCriteria(Message.class).createAlias("organization","org").              add(Restrictions.eq("org.orgId",new Long(orgId))).add(Restrictions.in("statusCode",status)).list();   you can get all the details in hibernate website.  <a href="http://www.hibernate.org/hib\_docs/reference/en/html/associations.html">http://www.hibernate.org/hib\_docs/reference/en/html/associations.html</a> |

|  |
| --- |
|  |

The information you are posting should be related to java and ORACLE technology. Not political. 

What is Hibernate proxy?

By default Hibernate creates a proxy for each of the class you map in mapping file. This class contain the code to invoke JDBC. This class is created by hibernate using CGLIB.   
  
Proxies are created dynamically by subclassing your object at runtime. The subclass has all the methods of the parent, and when any of the methods are accessed, the proxy loads up the real object from the DB and calls the method for you. Very nice in simple cases with no object hierarchy. Typecasting and instanceof work perfectly on the proxy in this case since it is a direct subclass.

### **What is the main advantage of using the hibernate than using the sql ?**

1) If you are using Hibernate then you don't need to learn specific SQL (like oracle,mysql), You have to user POJO class object as a table.   
  
2) Don't need to learn query tuning..Hibernate criteria query automatically tuned the query for best performance.   
  
3) You can use inbuild cache for storing data   
  
4) No need to create own connection pool , we can use c3po . It will give best result...   
  
5> Don't need any join query which reduce performance and complex. Using hibernate you have to define in bean and hbm.xml file.   
  
6> You can add filter in Hibernate which exceute before you query fires and get the best performance   
  
7> EhCache is used for 2nd level cache to store all the redefind data like country table ..

how to create primary key using hibernate?

<id name="userId" column="USER\_ID" type="int">   
<generator class="increment"/>   
</id>   
  
increment generator class automatically generate the primary key for you.

### **How to Execute Stored procedure in Hibernate ?**

Option 1:   
Connection con = null;   
  
  
try {   
con = session.connection();   
  
CallableStatement st = con   
.prepareCall("{call your\_sp(?,?)}");   
st.registerOutParameter(2, Types.INTEGER);   
st.setString(1, "some\_Seq");   
  
st.executeUpdate();   
  
Option 2:   
<sql-query name="selectAllEmployees\_SP" callable="true">   
<return alias="emp" class="employee">   
<return-property name="empid" column="EMP\_ID"/>   
  
<return-property name="name" column="EMP\_NAME"/>   
<return-property name="address" column="EMP\_ADDRESS"/>   
{ ? = call selectAllEmployees() }   
</return>   
</sql-query>   
  
code :   
  
SQLQuery sq = (SQLQuery) session.getNamedQuery("selectAllEmployees\_SP");   
  
List results = sq.list();

what is lazy fetching in hibernate?

Lazy setting decides whether to load child objects while loading the Parent Object.You need to do this setting respective hibernate mapping file of the parent class.Lazy = true (means not to load child)By default the lazy loading of the child objects is true. This make sure that the child objects are not loaded unless they are explicitly invoked in the application by calling getChild() method on parent.In this case hibernate issues a fresh database call to load the child when getChild() is actully called on the Parent object.But in some cases you do need to load the child objects when parent is loaded. Just make the lazy=false and hibernate will load the child when parent is loaded from the database.Exampleslazy=true (default)Address child of User class can be made lazy if it is not required frequently.lazy=falseBut you may need to load the Author object for Book parent whenever you deal with the book for online bookshop

**Difference b/w first level cache/ second level cache**

1.1) First-level cache

First-level cache always Associates with the Session object. Hibernate uses this cache by default. Here, it processes one transaction after another one, means wont process one transaction many times. Mainly it reduces the number of SQL queries it needs to generate within a given transaction. That is instead of updating after every modification done in the transaction, it updates the transaction only at the end of the transaction.

1.2) Second-level cache

Second-level cache always associates with the Session Factory object. While running the transactions, in between it loads the objects at the Session Factory level, so that those objects will available to the entire application, don’t bounds to single user. Since the objects are already loaded in the cache, whenever an object is returned by the query, at that time no need to go for a database transaction. In this way the second level cache works. Here we can use query level cache also. Later we will discuss about it.

#### **What is ORM?**

Object Relational Model is a programming technique. This technique is used to convert the data from an incompatible type to that of a relational database. The mapping is done by using the OOP languages. For example every table in a table is represented by an object.

The purpose of ORM is an application is allowed and written OOP language that is to deal with the data that is manipulates in the forms objects. The class level attributes are ‘mapped’ to tables.

#### **What is Hibernate?**

Hibernate is an ORM tool. Hibernate is a solution for object-relational mapping and a persistence management layer. For example a java application is used to save data of an object to a database. Hibernate provides a solution for this persistence by mapping database tables to a class. The database data is copied to the object. The copying the data as objects and vice versa is known as object-relational mapping. A query that uses select statement and its operations of fetching tuples is done by a simple query object. And being an object, the same object can be utilized several times, which reduces the ‘query defining time’. Usage of simple java classes [ Plain Old Java Objects ] makes this task simple instead of using entity beans / JDBC calls.

#### **Why do you need ORM tools like hibernate?**

Hibernate is open source ORM tool. Hibernate reduces the time to perform database operations. Hibernate has advantages over using entity beans or JDBC calls. The implementation is done by just using POJOs. Creating a POJO may implement the Java bean mechanism of using setters and getters methods for assigning values and they can be used to persist on the database. By using a simple mapping file, the fields of class and the fields of the table is done. This mapping file is a XML file. So the flexibility in mapping is attained, instead of hard coding. IDEs like eclipse supports to create the description files.

#### **What are the main advantages of ORM like hibernate?**

Advantages of ORM tools:

* The SQL code / statements in the application can be eliminated without writing complex JDBC / Entity Bean code.
* Distributed transaction can simply be performed by using ORM tools.
* Hibernate is an open source ORM tool and a robust framework to perform ORMapping.
* Without much of SQL knowledge, one can master Hibernate easily

#### **What are the core interfaces of Hibernate framework?**

1. **Session Interface:** The basic interface for all hibernate applications. The instances are light weighted and can be created and destroyed without expensive process.

2. **SessionFactory interface:** The delivery of session objects to hibernate applications is done by this interface. For the whole application, there will be generally one SessionFactory and can be shared by all the application threads.

3. **Configuration Interface:** Hibernate bootstrap action is configured by this interface. The location specification is specified by specific mapping documents, is done by the instance of this interface.

4. **Transaction Interface:** This is an optional interface. This interface is used to abstract the code from a transaction that is implemented such as a JDBC / JTA transaction.

5. **Query and Criteria interface:** The queries from the user are allowed by this interface apart from controlling the flow of the query execution.

#### **Explain how to configure Hibernate.**

Hibernate uses a file by name hibernate.cfg.xml. This file creates the connection pool and establishes the required environment. A file named .hbm.xml is used to author mappings. The bootstrap action is configured by using Configuration interface.

There are two types of environment to configure hibernate:

1. **Managed Environment:** The definitions of database operations such as connections, transaction boundaries, security levels. This environment is provided by application servers such as Jboss,Weblogic,Websphere.

2. **Non-managed Environment:** The basic configuration template is provided by this interface. Tomcat is one of the examples that best supports this environment.

#### **Define HibernateTemplate.**

HibernateTemplate is a helper class that is used to simplify the data access code. This class supports automatically converts HibernateExceptions which is a checked exception into DataAccessExceptions which is an unchecked exception. HibernateTemplate is typically used to implement data access or business logic services. The central method is execute(), that supports the Hibernate code that implements HibernateCallback interface.

#### **What are the benefits of HibernateTemplate?**

The benefits of HibernateTemplate are:

* 1. HibernateTemplate, which is a Spring Template class, can simplify the interactions with Hibernate Sessions.
* 2. Various common functions are simplified into single method invocations.
* 3. The sessions of hibernate are closed automatically
* 4. The exceptions will be caught automatically, and converts them into runtime exceptions.

#### **What is Hibernate proxy?**

Mapping of classes can be made into a proxy instead of a table. A proxy is returned when actually a load is called on a session. The proxy contains actual method to load the data. The proxy is created by default by Hibernate, for mapping a class to a file. The code to invoke Jdbc is contained in this class.

#### **Explain the types of Hibernate instance states.**

The persistent class’s instance can be in any one of the three different states. These states are defined with a persistence context. The Hibernate has the following instance states:

* **Transient:** This instance is never been associated with any one of the persistence process. This does not have persistent identity like primary key value.
* **Persistent:** A persistent context is made to associate with the current instance. It has persistent identity like primary key value and a corresponding row of a table in the data base. Hibernate guarantees the persistent identity is equivalent to the java Identity [object], for a particular persistence context
* **Detatched:** This instance association with a persistence context is only once and the context was closed or serialized to another process. The persistent identity is retained and it can be a corresponding row in a database.

#### **Explain the Collection types in Hibernate.**

A collection is defined as a one-to-many reference. The simplest collection type in Hibernate is <bag>. This collection is a list of unordered objects and can contain duplicates. This is similar to java.util.List. The content of a collection is required for a SQL query. This won’t work until the code is actually accessed. This process has a benefit that allows the developer to separate the database access logic from that of the object traversal logic. This is called lazy collection.

Filtering logic can be applied with lazy collection. This can alter the SQL which invokes when the actual collection is invoked.

#### **What is lazy initialization in hibernate?**

The delaying the object creation or calculating a value or some process until the first time it is needed. The retrieval of particular information only at the time when the object is accessed, is lazy initialization in hibernate. A scenario for lazy initialization is:

When the field creation is expensive, a field may or may not be invoked.

In this scenario the creation of a field can be deferred until the actual moment is arise to use it. The performance is increased using this technique, by avoiding unnecessary creation of objects which is expensive and consumes the memory space.

#### **What is lazy fetching in hibernate?**

Lazy fetching is associated with child objects loading for its parents. While loading the parent, the selection of loading a child object is to be specified / mentioned in the hbm.xml file. Hibernate does not load the whole child objects by default. Lazy=true means not to load the child objects.

#### **What is the difference between sorted and ordered collection in hibernate?**

**Sorted Collection**The sorted collection is a collection that is sorted using the Java collections framework. The sorting is done in the memory of JVM that is running hibernate, soon after reading the data from the database using Java Comparator

The less the collection the more the efficient of sorting

**Ordered Collection**The order collections will also sorts a collection by using the order by clause for the results fetched.

The more the collection, the more efficient of sorting

#### **Explain the difference between transient (i.e. newly instantiated) and detached objects in hibernate.**

Transient objects do not have association with the databases and session objects. They are simple objects and not persisted to the database. Once the last reference is lost, that means the object itself is lost. And of course , garbage collected. The commits and rollbacks will have no effects on these objects. They can become into persistent objects through the save method calls of Session object.

The detached object have corresponding entries in the database. These are persistent and not connected to the Session object. These objects have the synchronized data with the database when the session was closed. Since then, the change may be done in the database which makes this object stale. The detached object can be reattached after certain time to another object in order to become persistent again.

#### **Explain the advantages and disadvantages of detached objects.**

**Advantages:**

* Detached objects passing can be done across layers upto the presentation layer without using Data Transfer Objects.
* At the time of using long transactions by the user which needs long think-time, it is suggested to split these transactions into some transactions. The detached objects get modified apart from the transaction scope which then can be re-attached to a new transaction.

**Disadvantages:**

* The usage of detached objects are cumbersome and cryptic. It is suggested not to be cluttered with the session, if possible.
* It is recommended to use DataTransferObjects and DomainObjects that is used to maintain separation between the user interfaces and the Service.

#### **What is Hibernate Query Language (HQL)?**

Hibernate Query Language is designed for data management using Hibernate technology. It is completely object oriented and hence has notions like inheritance, polymorphism and abstraction. The queries are case-sensitive. This has an exception for Java classes and properties. The query operations are through objects. HQL acts as a bridge between Objects and RDBMS.

**HQL: The Hibernate Query Language** The Hibernate Query Language is executed using session.createQuery(). This tutorial includes from clause,Associations and joins, Aggregate functions,The order by clause,The group by clause,Subqueries.

**The from clause**

|  |
| --- |
| from Employee // Employee is class name mapped to EMPLOYEE TABLE or from Employee as e  or from Employee e where e.empId = 3; List empList = session.createQuery("from Employee").list(); |

**Associations and joins**

|  |
| --- |
| from Employee e where e.scopeModFlag = 1 and pc.isDeleted != 1  List empList = session.createQuery("from Employee e where e.scopeModFlag = 1 and pc.isDeleted != 1").list(); |

**Aggregate functions**

|  |
| --- |
| select avg(cat.weight), sum(cat.weight), max(cat.weight), count(cat) from Cat cat ScrollableResults rs = session.createQuery("select avg(cat.weight), sum(cat.weight), max(cat.weight), count(cat) from Cat cat").scroll(); if(rs.next()){ System.out.println(rs.get(0)); System.out.println(rs.get(1)); System.out.println(rs.get(2)); System.out.println(rs.get(3)); } |

**The order by clause**

|  |
| --- |
| from Employee e order by e.name desc  List empList = session.createQuery("from Employee e order by e.name desc").list(); asc or desc indicate ascending or descending order respectively. |

**The group by clause**

|  |
| --- |
| select e.dept, sum(e.salary), count(e) Employee e group by cat.dept |

**Subqueries**

|  |
| --- |
| from Employee as e  where e.name = some (  select name.nickName from Name as name  ) |

#### **Explain the general flow of Hibernate communication with RDBMS?**

The general flow of Hibernate communication with RDBMS is :

* The Hibernate configuration is to be loaded and creation of configuration object is done. The mapping of all hbm files will be performed automatically.
* Creation of session factory from the configuration object.
* Obtain a session from the session factory.
* Creation of HQL Query
* Execution of the query in order to get the list of containing java objects.

#### **Explain the role of Session interface in Hibernate.**

Session interface is a single threaded object. The representation of single unit of work with the Java application and the persistence database is done by this object. The wrapping of a JDBC connection is the emphasized role of a Session interface object. This is the major interface between Java application and Hibernate. This is a runtime interface. The beginning and ending of a transaction cycle is bounded by the Session interface. The purpose of this interface is to perform create, read and delete operations for the objects that are mapped with entity classes.

#### **What is a SessionFactory?**

The SessionFactory is the concept that is a single data store and thread safe. Because of this feature, many threads can access this concurrently and the sessions are requested, and also the cache that is immutable of compiled mappings for a specific database. A SessionFactory will be built only at the time of its startup. In order to access it in the application code, it should be wrapped in singleton. This wrapping makes the easy accessibility to it in an application code.

#### **State the role of SessionFactory interface plays in Hibernate.**

The SessionFactory is used to create Sessions. Each application is having usually only one SessionFactory. The requests from the servicing client threads obtain the Sessions from the session factory. SessionFactory is thread safe, so that many threads can access it concurrently and requests the sessions.

#### **Explain the difference between load() and get() in Hibernate.**

Difference between load() and get()

**load()**Use this method if it is sure that the objects exist.

The load() method throws an exception,when the unique id could not found in the database.

The load() method returns proxy by default and the data base will not be effected until the invocation of the proxy.

**get()**Use this method if it is not sure that the objects exist.

Returns null when the unique id is unavailable in the database.

The data base will be effected immediately

#### **What is the difference between merge and update?**

update () : When the session does not contain an persistent instance with the same identifier, and if it is sure use update for the data persistence in hibernate.

merge (): Irrespective of the state of a session, if there is a need to save the modifications at any given time, use merge().

#### **What is the advantage of Hibernate over jdbc?**

The advantages of Hibernate over JDBC are:

* Hibernate code will work well for all databases, for ex: Oracle,MySQL, etc. where as JDBC is database specific.
* No knowledge of SQL is needed because Hibernate is a set of objects and a table is treated as an object, where as to work with JDBC, one need to know SQL.
* Query tuning is not required in Hibernate. The query tuning is automatic in hibernate by using criteria queries, and the result of performance is at its best. Where as in JDBC the query tuning is to be done by the database authors.
* With the support of cache of hibernate, the data can be placed in the cache for better performance. Where as in JDBC the java cache is to be implemented.

#### **Why hibernate is advantageous over Entity Beans & JDBC?**

An entity bean always works under the EJB container, which allows reusing of the object external to the container. An object can not be detached in entity beans and in hibernate detached objects are supported.

Hibernate is not database dependent where as JDBC is database dependent. Query tuning is not needed for hibernate as JDBC is needed. Data can be placed in multiple cache which is supported by hibernate, whereas in JDBC the cache is to be implemented.

#### **Explain the main difference between Entity Beans and Hibernate.**

Entity beans are to be implemented by containers, classes, descriptors. Hibernate is just a tool that quickly persist the object tree to a class hierarchy in a database and without using a single SQL statement. The inheritance and polymorphism is quite simply implemented in hibernate which is out of the box of EJB and a big drawback.

#### **Explain the difference between hibernate and Spring.**

Hibernate is an ORM tool for data persistency. Spring is a framework for enterprise applications. Spring supports hibernate and provides the different classes which are templates that contains the common code.

## ***Caching***

TopLink caching improves application performance by automatically storing data returned as objects from the database for future use. This caching provides several advantages:

* Reusing Java objects that have been previously read from the database minimizes database access
* Minimizing SQL calls to the database when objects already exist in the cache
* Minimizing network access to the database
* Setting caching policies a class-by-class and bean-by-bean basis
* Basing caching options and behavior on Java garbage collection

TopLink supports several caching polices to provide extensive flexibility. Developers can fine-tune the cache for maximum performance, based on individual application performance. Refer to [Part XVIII, "Cache"](http://download-west.oracle.com/docs/cd/B25221_04/web.1013/b13593/prt_cac.htm#BABHJHECIV) for complete information.