HIBERNATE INTERVIEW QUESTIONS, PART- II

Q: What are core interfaces for Hibernate framework?

A: Most Hibernate-related application code primarily interacts with four interfaces provided by Hibernate Core:

org.hibernate.Configuration

org.hibernate.Session

org.hibernate.SessionFactory

org.hibernate.Criteria

org.hibernate.Query

The configuration interface is used to load the hibernate configuration.

The Session is a persistence manager that manages operation like storing and retrieving objects. Instances of Session are inexpensive to create and destroy. They are not thread safe. The application obtains Session instances from a SessionFactory. SessionFactory instances are not lightweight and typically one instance is created for the whole application. If the application accesses multiple databases, it needs one per database. The Criteria provides a provision for conditional search over the resultset.One can retrieve entities by composing criterion objects. The Session is a factory for Criteria. Criterion instances are usually obtained via the factory methods on Restrictions. Query represents object oriented representation of a Hibernate query. A Query instance is obtained by calling

Session.createQuery().

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

A: 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.

Q: What is the difference between hibernate and jdbc ?

A: 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.

Q: What is lazy fetching in Hibernate? With Example.

A: 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.

Q: what is the advantage of Hibernate over jdbc?

A: 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) Dont 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 dont 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 Dont have such support.

10) Hibernate Supports automatic versioning of rows but JDBC Not.

Q: What is c3p0? How To configure the C3P0 connection pool?

A: c3p0 is an easy-to-use library for augmenting traditional (DriverManager-based) JDBC drivers with JNDI-bindable DataSources, including DataSources that implement Connection and Statement Pooling, as described by the jdbc3 spec and jdbc2 std extension. In simple you can say c3p0 is an open source database connection pool used in hivernate.

Configuration: We need to add the library C3P0 jar to our application lib. then we have to configure it in our hibernate configuration file.

Here is a sample of C3P0 configuration. This is an extract of hibernate.cfg.xml:

```
<!-- configuration pool via c3p0-->
<property name="c3p0.acquire_increment">1</property>
<property name="c3p0.idle_test_period">100</property> <!-- seconds -->
<property name="c3p0.max_size">100</property>
<property name="c3p0.max_statements">0</property>
<property name="c3p0.min_size">10</property>
<property name="c3p0.timeout">100</property> <!-- seconds -->
<!-- DEPRECATED very expensive property name="c3p0.validate">-->
```

You also can set extra c3p0 properties using c3p0.properties. Put this file in the classpath (WEB-INF/classes for example), but be careful, the previous values will be overridden by Hibernate whether set or not (see below for more details). For more information on C3P0 configuration, please have a look at http://sourceforge.net/projects/c3p0 and unzip the download, there is a doc folder where you'll find everything you need.

read more : http://www.hibernate.org/214.html

Q: What is dirty checking in Hibernate?

A: Hibernate automatically detects object state changes in order to synchronize the updated state with the database, this is called dirty checking. An important note here is, Hibernate will compare objects by value, except for Collections, which are compared by identity. For this reason you should return exactly the same collection instance as Hibernate passed to the setter method to prevent unnecessary database updates.

Q: What are different fetch strategies Hibernate have?

A: A fetching strategy in Hibernate is used for retrieving associated objects if the application needs to navigate the asociation. They may be declared in the O/R mapping metadata, or over-ridden by a particular HQL or Criteria query.

Hibernate3 defines the following fetching strategies:

Join fetching - Hibernate retrieves the associated instance or collection in the same SELECT, using an OUTER JOIN.

Select fetching - a second SELECT is used to retrieve the associated entity or collection. Unless you explicitly disable lazy fetching by specifying lazy="false", this second select will only be executed when you actually access the association.

Subselect fetching - a second SELECT is used to retrieve the associated collections for all entities retrieved in a previous query or fetch. Unless you explicitly disable lazy fetching by specifying lazy="false", this second select will only be executed when you actually access the association.

Batch fetching - an optimization strategy for select fetching - Hibernate retrieves a batch of entity instances or collections in a single SELECT, by specifying a list of primary keys or foreign keys.

Hibernate also distinguishes between:

Immediate fetching - an association, collection or attribute is fetched immediately, when the owner is loaded.

Lazy collection fetching - a collection is fetched when the application invokes an operation upon that collection. (This is the default for collections.)

"Extra-lazy" collection fetching - individual elements of the collection are accessed from the database as needed. Hibernate tries not to fetch the whole collection into memory unless absolutely needed (suitable for very large collections)

Proxy fetching - a single-valued association is fetched when a method other than the identifier getter is invoked upon the associated object.

"No-proxy" fetching - a single-valued association is fetched when the instance variable is accessed. Compared to proxy fetching, this approach is less lazy (the association is fetched even when only the

identifier is accessed) but more transparent, since no proxy is visible to the application. This approach requires buildtime bytecode instrumentation and is rarely necessary.

Lazy attribute fetching - an attribute or single valued association is fetched when the instance variable is accessed. This approach requires buildtime bytecode instrumentation and is rarely necessary.

We use fetch to tune performance. We may use lazy to define a contract for what data is always available in any detached instance of a particular class.

Q: Explain different inheritance mapping models in Hibernate?

A: There can be three kinds of inheritance mapping in hibernate

1. Table per concrete class with unions

2. Table per class hierarchy

3. Table per subclass

Example:

We can take an example of three Java classes like Vehicle, which is an abstract class and two subclasses of Vehicle as Car and UtilityVan.

1. Table per concrete class with unions, In this scenario there will be 2 tables

Tables: Car, UtilityVan, here in this case all common attributes will be duplicated.

2. Table per class hierarchy

Single Table can be mapped to a class hierarchy

There will be only one table in database named 'Vehicle' which will represent all attributes required for all three classes.

Here it is be taken care of that discriminating columns to differentiate between Car and UtilityVan

3. Table per subclass

Simply there will be three tables representing Vehicle, Car and UtilityVan.

Q: How to Integrate Struts1.1 Spring and Hibernate ?

A: Step 1:

In the struts-config.xml add plugin

```
<plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
<set-property property="contextConfigLocation"
value="/WEB-INF/applicationContext.xml"/>
</plug-in>
```

Step 2:

In the applicationContext.xml file

Configure datasourse

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name="driverClassName"><value>oracle.jdbc.driver.OracleDriver</value>
</property>
<property name="url"><value>jdbc:oracle:thin:@10.10.01.24:1541:ebizd</value>
</property>
<property name="username"><value>sa</value></property>
    <property name="password"><value></value></property>
</bean>
```

Step 3.

Configure SessionFactory

```
<!-- Hibernate SessionFactory -->
<bean id="sessionFactory" class="org.springframework.orm.hibernate.LocalSessionFactoryBean">
<property name="dataSource"><ref local="dataSource"/></property>
<property name="mappingResources">
<list>
<value>com/test/dbxml/User.hbm.xml</value>
</list>
</property>
<property name="hibernateProperties">
```

```
<props>
<prop key="hibernate.dialect">net.sf.hibernate.dialect.OracleDialect </prop>
</props>
</property>
</bean>
```

Step 4.

Configure User.hbm.xml

```
<hibernate-mapping>
<class name="com.garnaik.model.User" table="app_user">
<id name="id" column="id" >
<generator class="increment"/>
</id>
<property name="firstName" column="first_name" not-null="true"/>
<property name="lastName" column="last_name" not-null="true"/>
</class>
</hibernate-mapping>
```

Step 5.

In the applicationContext.xml ? configure for DAO

```
<bean id="userDAO" class="com.garnaik.dao.hibernate.UserDAOHibernate">
<property name="sessionFactory"><ref local="sessionFactory"/></property>
</bean>
```

Step 6.

DAO Class

```
public class UserDAOHibernate extends HibernateDaoSupport implements UserDAO {
private static Log log = LogFactory.getLog(UserDAOHibernate.class);
public List getUsers() {
return getHibernateTemplate().find("from User");
}
public User getUser(Long id) {
return (User) getHibernateTemplate().get(User.class, id);
}
public void saveUser(User user) {
getHibernateTemplate().saveOrUpdate(user);
if (log.isDebugEnabled()) {
log.debug("userId set to: " + user.getId());
}
}
public void removeUser(Long id) {
Object user = getHibernateTemplate().load(User.class, id);
getHibernateTemplate().delete(user);
}
}
```

Q: How to prevent concurrent update in Hibernate?

A: 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. Change made by user B is gone. In hibernate you can

prevent slate object updatation using version checking. Check the version of the row when you are updating 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 column 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.

Q: What is version checking in Hibernate or How to handle user think time using hibernate ?

A: version checking used in hibernate when more then one thread trying to access same data. As we already have discussed in the previous 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.

Q: Transaction with plain JDBC in Hibernate ?

A: 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();
}
```

Q: What are the general considerations or best practices for defining your Hibernate persistent classes?

A: 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.

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

A: 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.

Q: Difference between getCurrentSession() and openSession() in Hibernate ?

A: 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
}
```

Q: Difference between session.saveOrUpdate() and session.merge()?

A: **saveOrUpdate() 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

merge() 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

Q: Filter in Hibernate with Example?

A: 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: Criteria Query Two Condition

A: 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 organization 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();

Q: How can I avoid n+1 SQL SELECT queries when running a Hibernate query?

A: Follow the best practices guide! Ensure that all <class> and <collection> mappings specify lazy="true" in Hibernate2 (this is the new default in Hibernate3). Use HQL LEFT JOIN FETCH to specify which associations you need to be retrieved in the initial SQL SELECT.

A second way to avoid the n+1 selects problem is to use fetch="subselect" in Hibernate3.

If you are still unsure, refer to the Hibernate documentation and Hibernate in Action.

Q: Explain the hibernate N+1 problem and its solution?

A:

Problem

In a previous pitfall (Avoiding The N+1 Selects Problem) we showed how the N+1 selects problem could inadvertently arise when not using lazy initialization. A more subtle cause for the N+1 selects problem is from not using database joins correctly to return the data our application uses. If an application does use the correct fetching strategy to load the data it needs, it may end up making more round trips to the database than necessary.

We illustrate by returning to the example of the class `Contact`, which has a one-to-many relationship with `Manufacturer` (that is, there is one `Contact` for many `Manufacturers`) . Since the `Contract` is uses lazy initialization in its *hbm.xml* file

```
<class name="example.domain.Contact" table="CONTACT" lazy = "true">
...
</class>
```

it will not automatically be lazily initialized as part of the execution of the HQL "`from Manufacturer manufacturer`". This query will not load the data for `Contact`, but will instead load a proxy to the real data.

The problem is when you run this query but decide in writing your application that you do want to retrieve all the contacts for the returned set of manufacturers

```
Query query = getSupport().getSession().createQuery("from Manufacturer
manufacturer");
List list = query.list();
for (Iterator iter = list.iterator(); iter.hasNext();)
{
  Manufacturer manufacturer = (Manufacturer) iter.next();
  System.out.println(manufacturer.getContact().getName());
}
```

Since the initial query "`from Manufacturer manufacturer`" does not initialize the `Contact` instances, an additional separate query is needed to do so for each `Contact` loaded. Again, you get the N+1 selects problem.

Solution

We solve this problem by making sure that the initial query fetches all the data needed to load the objects we need in their appropriately initialized state. One way of doing this is using an HQL fetch join.

We use the HQL

```
"from Manufacturer manufacturer join fetch manufacturer.contact
contact"
```

with the fetch statement. This results in an inner join:

```
select MANUFACTURER.id from manufacturer and contact ... from
MANUFACTURER inner join CONTACT on MANUFACTURER.CONTACT_ID=CONTACT.id
```

Using a *Criteria* query we can get the same result from

```
Criteria criteria = session.createCriteria(Manufacturer.class);
criteria.setFetchMode("contact", FetchMode.EAGER);
```

which creates the SQL

```
select MANUFACTURER.id from MANUFACTURER left outer join CONTACT on
MANUFACTURER.CONTACT_ID=CONTACT.id where 1=1
```

In both cases, our query returns a list of `Manufacturer` objects with the contact initialized. Only one query needs to be run to return all the contact and manufacturer information required for the example.

Q: How does Value replacement in Message Resource Bundle work?

A: 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);

Q: Difference between list() and iterate() i9n Hibernate?

A: 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.

Q: Deleting persistent objects

A: 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);

Q: SQL statements execution order.

A: 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

Q: Modifying persistent objects?

A: 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.

Q: SQL Queries In Hibernate..

A: 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.

Q: Equal and Not Equal criteria query.

A: 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();

Q: What is bi- directional mapping, Give example of Bidirectional one-to-many with an indexed collection?

A: Hibernate 2 does not support bidirectional (inverse="true") one-to-many associations with an indexed collection (list, map or array) as the "many" end.

If you want to keep the inverse="true" attribute and want to use an indexed collection, you have to handle the index of the collection manually.The following solution works with a list.

The same method can be used when using an array or a map. (See below)

Mapping

```
<class name="net.sf.test.Parent" table="parent">
  <id name="id" column="id" type="long" unsaved-value="null">
    <generator class="sequence">
      <param name="sequence">SEQ_DEFAULT</param>
    </generator>
  </id>
  <list name="children" lazy="true" inverse="true">
      <key column="parent_id"/>
      <index column="index_col"/>
      <one-to-many class="net.sf.test.Child"/>
  </list>
</class>
<class name="net.sf.test.Child" table="child">
  <id name="id" column="id" type="long" unsaved-value="null">
    <generator class="sequence">
      <param name="sequence">SEQ_DEFAULT</param>
    </generator>
  </id>
  <many-to-one name="parent" column="parent_id" not-null="true"/>
  <property name="index" column="index_col" type="int" update="true" insert="true"/>
</class>
```

The inverse="true" is set to the one side.

The column name for the index property on the child is the same as the index column of the one-to-many mapping on the parent.

Q: What is hibernate entity manager?

A: Hibernate EntityManager implements:

* The standard Java Persistence management API

* The standard Java Persistence Query Language

* The standard Java Persistence object lifecycle rules

* The standard Java Persistence configuration and packaging

Hibernate EntityManager wraps the powerful and mature Hibernate Core. You can fall back to Hibernate native APIs, native SQL, and native JDBC whenever necessary.

The Hibernate Java Persistence provider is the default persistence provider of the JBoss EJB 3.0 implementation and bundled with the JBoss Application Server.

Q: What is Hibernate Annotations?

A: Hibernate, like all other object/relational mapping tools, requires meta data that governs the transformation of data from one representation to the other (and vice versa). As an option, you can now use JDK 5.0 annotations for object/relational mapping with Hibernate 3.2. You can use annotations in addition to or as a replacement of XML mapping metadata.

The Hibernate Annotations package includes:

Standardized Java Persistence and EJB 3.0 (JSR 220) object/relational mapping annotations

Hibernate-specific extension annotations for performance optimization and special mappings

You can use Hibernate extension annotations on top of standardized Java Persistence annotations to utilize all native Hibernate features.

Requirements: At a minimum, you need JDK 5.0 and Hibernate Core, but no application server or EJB 3.0 container. You can use Hibernate Core and Hibernate Annotations in any Java EE 5.0 or Java SE 5.0 environment.

Q: Cascade Save or Update in Hibernate ?

A: 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;
```

Q: One To Many Bi-directional Relation in Hibernate?

A: 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>
```

Q: One To Many Mapping Using List ?

A: 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();
}
```

Q: Many To Many Relation In Hibernate ?

A: 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());
}
```

Q: What does session.refresh() do ?

A: 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)

Q: Hibernate setup using .cfg.xml file ?

A: 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"/; n will be avilable in 2nd level cache

Q: How to get JDBC connections in hibernate?

A: User Session.connection() method to get JDBC Connection.

Q: How will you configure Hibernate?

A: 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;
```

```
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();
```

```
        }
    }
}
```

Q: What are the Instance states in Hibernate?

A: The instance states in hibernate are:

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 corresponding row in the database.

For detached instances, Hibernate makes no guarantees about the relationship between persistent identity and Java identity.

Q: What are the core components in Hibernate ?

A: **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.

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

A: 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;
}
}
```

Q: addScalar() method in hibernate...

A: 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.

Q: Hibernate session.close does _not_ call session.flush ?

A: 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();
    }
}
```

Q: What is the main difference between Entity Beans and Hibernate ?

A: 1)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 need. 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.

Q: How are joins handled using Hibernate?

A: 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.

http://www.hibernate.org/hib_docs/reference/en/html/associations.html

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

Q: What is Hibernate proxy?

A: 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 sub classing 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 instance of work perfectly on the proxy in this case since it is a direct subclass.

Q: What is the main advantage of using the hibernate than using the sql ?

A: 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 ..

Q: how to create primary key using hibernate?

A: <id name="userId" column="USER_ID" type="int">

<generator class="increment"/>

</id>

increment generator class automatically generate the primary key for you.

Q: How to Execute Stored procedure in Hibernate ?

A: 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();
```

Q: what is lazy fetching in hibernate?

A: 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 actually 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. Examples lazy=true (default)Address child of User class can be made lazy if it is not required frequently. lazy=false But you may need to load the Author object for Book parent whenever you deal with the book for on line bookshop

Q: What are Hibernate Fetch Strategies?

● A fetching strategy is the strategy Hibernate will use for retrieving associated objects if the application needs to navigate the association.

● Fetching strategy will have performance impact

● Fetch strategies may be declared in the mapping files, or over-ridden by a particular HQL or Criteria query.

Types of Fetching Strategies

● **How fetching is done**

– Join

– Select (default)

– Subselect

– Batch

● **When fetching is done**

– immediate

– lazy (default)

Q: What are Cascades in hibernate?

A: Cascades in hibernate are:

save-update, delete, all, all-delete-orphan, delete-orphan and none.

1) cascade="none", the default, tells Hibernate to ignore the association.

2) cascade="save-update" tells Hibernate to navigate the association when the transaction is committed and when an object is passed to save() or update() and save newly instantiated transient instances and

persist changes to detached instances.

3) cascade="delete" tells Hibernate to navigate the association and delete persistent instances when an object is passed to delete().

4) cascade="all" means to cascade both save-update and delete, as well as calls to evict and lock.

5) cascade="all-delete-orphan" means the same as cascade="all" but, in addition, Hibernate deletes any persistent entity instance that has been removed (dereferenced) from the association (for example, from a collection).

6) cascade="delete-orphan" Hibernate will delete any persistent entity instance that has been removed (dereferenced) from the association (for example, from a collection).

Read more Hibernate API for cascades.

Q: what is the use of cascade in hbm file?

A: cascade specifies which operations should be casecaded from the parent object to the associated object. The meaningfull values would be persist , merge, delete, save_update, evict , replicate, lock , refresh , all , delete_orphan.

Q: What is Inverse in Hibernate?

A:

Q: Example of hibernate inheritance with examples?

- Table-per-class hierarchy
- Table-per-subclass
- Table-per-concrete class

Mapping example for Table-per-class hierarchy

```
mysql> select * from cd;
+--------+---------------------+--------+---------------------+------+-------------+-----------+--------+-----------------+
|id      | title               | artist | purchasedate        | cost | newfeatures | languages | region | cd_type         |
+--------+---------------------+--------+---------------------+------+-------------+-----------+--------+-----------------+
|360449  | Grace Under Pressure | Rush   | 2004-04-16 00:00:00 | 9.99 | NULL        | NULL      |   NULL | cd              |
| 360450 | Grace Under Pressure | Rush   | 2004-04-16 00:00:00 | 9.99 | Widescreen  | NULL      |   NULL | SpecialEditionCD |
| 360451 | Grace Under Pressure | Rush   | 2004-04-16 00:00:00 | 9.99 | NULL        | Spanish   |      4 | InternationalCD |
+--------+---------------------+--------+---------------------+------+-------------+-----------+--------+-----------------+
3 rows in set (0.00 sec)
```

```
<?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 package="example.products">
<classname="CD" table="cd" discriminator-value="cd">
<id name="id" type="integer" unsaved-value="0">
<generator class="hilo"/>
</id>
<discriminator column="cd_type"type= "string"/>
<property name="title"/>
<property name="artist"/>
<property name="purchasedate" type="date"/>
<property name="cost" type="double"/>
<subclass name="SpecialEditionCD"
discriminator-value="SpecialEditionCD">
<property name="newfeatures" type="string"/>
</subclass>
<subclass name="InternationalCD" discriminator-value="InternationalCD">
<property name="languages"/>
<property name="region"/>
</subclass>
</class>
</hibernate-mapping>
```

Mapping example for Table-per-subclass

```
mysql> select * from cd;
+--------+---------------------+--------+--------------------+------+
| id | title | artist | purchasedate | cost |
```

```
+--------+----------------------+--------+---------------------+------+
| 425985 | Grace Under Pressure | Rush | 2004-04-16 00:00:00 | 9.99 |
| 425986 | Grace Under Pressure | Rush | 2004-04-16 00:00:00 | 9.99 |
| 425987 | Grace Under Pressure | Rush | 2004-04-16 00:00:00 | 9.99 |
+--------+----------------------+--------+---------------------+------+
3 rows in set (0.00 sec)

mysql> select * from secd;
+--------+-------------+
| id | newfeatures |
+--------+-------------+
| 425986 | Widescreen |
+--------+-------------+
1 row in set (0.00 sec)

mysql> select * from icd;
+--------+-----------+--------+
| id | languages | region |
+--------+-----------+--------+
| 425987 | Spanish | 4 |
+--------+-----------+--------+
1 row in set (0.00 sec)
```

```
<?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 package="example.products">
<class name="CD" table="cd">
<idname="id" type="integer"
unsaved-value="0">
<generator class="hilo"/>
</id>
<property name="title"/>
<property name="artist"/>
<property name="purchasedate" type="date"/>
<property name="cost" type="double"/>
<joined-subclass name="SpecialEditionCD"
table="secd">
```

```
<key column="id"/>
<property name="newfeatures" type="string"/>
</joined-subclass>
<joined-subclass name="InternationalCD"
table="icd">
<key column="id"/>
<property name="languages"/>
<property name="region"/>
Creating Persistent Classes
</joined-subclass>
</class>
</hibernate-mapping>
```

Mapping example for Table-per-concrete class

```
<?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 package="example.products">
<classname="CD" table="cd" discriminator-value="cd">
<id name="id" type="integer"
unsaved-value="0">
<generator class="hilo"/>
</id>
<property name="title"/>
<property name="artist"/>
<property name="purchasedate" type="date"/>
<property name="cost" type="double"/>
```

```
</class>
<class name="SpecialEditionCD" table="secd">
<id name="id" type="integer"
unsaved-value="0">
<generator class="hilo"/>
</id>
<property name="title"/>
<property name="artist"/>
<property name="purchasedate" type="date"/>
<property name="cost" type="double"/>
<property name="newfeatures" type="string"/>
</class>
<class name="InternationalCD" table="icd">
<id name="id" type="integer" unsaved-value="0">
<generator class="hilo"/>
</id>
<property name="title"/>
<property name="artist"/>
<property name="purchasedate" type="date"/>
<property name="cost" type="double"/>
<property name="languages"/>
<property name="region"/>
</class>
</hibernate-mapping>
```

Using the same example program as in the last two sections, we obtain the following rows in the three databases:

```
mysql> select * from cd;
+--------+----------------------+--------+---------------------+------+
| id | title | artist | purchasedate | cost |
+--------+----------------------+--------+---------------------+------+
| 458753 | Grace Under Pressure | Rush | 2004-04-16 00:00:00 | 9.99 |
+--------+----------------------+--------+---------------------+------+
1 row in set (0.00 sec)


mysql> select * from secd;
+-------+----------------------+--------+-------------------+------+-----------+
|id | title | artist | purchasedate | cost |newfeatures|
+-------+----------------------+--------+-------------------+------+-----------+
|491521 | Grace Under Pressure | Rush | 2004-04-16 00:00:00| 9.99 | Widescreen|
+-------+----------------------+--------+-------------------+------+-----------+
1 row in set (0.00 sec)


mysql> select * from icd;
+-------+-------------------+-------+------------------+----+---------+--------+
|id |title |artist |purchasedate |cost|languages| region |
+-------+-------------------+-------+------------------+----+---------+--------+
|524289 |Grace Under Pressure|Rush |2004-04-16 00:00:00|9.99|Spanish | 4 |
+-------+-------------------+-------+------------------+----+---------+--------+
1 row in set (0.00 sec)
```

Q: One-to-One and Many-to-One example?

A:

many-to-one

An ordinary association to another persistent class is declared using a many-to-one element. The relational model is a many-to-one association: a foreign key in one table is referencing the primary key column(s) of the target table.

A typical many-to-one declaration looks as simple as this:

<many-to-one name="product" class="Product" column="PRODUCT_ID"/>

one-to-one

A one-to-one association to another persistent class is declared using a one-to-one element. ex:

<many-to-one name="department" column="departmentid" class="com.garnaik.fr.pojo.Department"/>

Q: Locking In Hibernate?

Hibernate is having two types of locking: Optimistic and Pessimistic

Optimistic – If you are using versioning or timestamp in your application then by default hibernate will use the locking. Hibernate will always check the version number before updating the persistence object.

Pessimistic – Explicitly we can implement the locking in hibernate by using the API. Locking of rows using SELECT FOR UPDATE syntax. We have to set the lock while we are loading the persistence object from the db.