

**Hibernate:**

Hibernate is a frame work. Hibernate frame work is used to develop a java application to interact with database server.

A frame work is a piece of software this piece of software contains solutions from commonly repeatedly occurred problems occurs multiple projects.

**IDE (Integrate Development Environment):**

As part of IDE all the components all integrated.

**Ex:** Editors, Compilers, Servers, Browser etc.

By using IDE we can develop project quickly.

IDE is will improve the productivity of the developer.

The following are the most popular IDE,s.

1. Eclipse
2. My Eclipse
3. Net Beans
4. RAD (Relational Application Development)
5. J Builder

To work with IDE we required a “workspace” folder.

A work space folder contains all the files which are related to that project.

When we double click on Eclipse.exe it will launch a dialog whose name is work space launch. To this work space launch we have to supply work space folder as input.

When we start the Eclipse IDE in the work space folder it has created a folder with the name.matadata. This folder contains all the files which are used by Eclipse IDE.

Eclipse IDE contains set of perspective by default Eclipse IDE launch’s JEE perspective.

A perspective contains set of views. A view is a Email window. A perspective contains set of views.

If we would like to perform any operations in IDE we must create a project.

**Procedure to create the project in IDE:**

Step 1: file                      new                      project.

Step 2: select the appropriate project and click on next button.

Step 3: in new java project window enter the project name and click on finish.



Step 4: we delete the project with out checking the check box. The files will not delete permanently.

As part of java prospective we use out line view to see all the variables and methods which are available in class.

Ctrl + O

The package explorer view is used to see all the files which are available in the project.

Select → Project → Properties → java Build path → library → add external jar files.

In IDE we can disable auto build process (Project → builds automatically).

**Note:**

We can change the short cuts of Eclipse IDE

(Window → Preferences → General → Keys).

As part of a Eclipse IDE we can replace or get the old code by using compare with or replace with local history.

To create a jar files we use an option export (files → export).

**\*Procedure to create web based application:**

Step 1: file → new → web project.

Step 2: The above step will display a dialog with the name 'new web project'. In that dialog enter the project name make sure that project name and web root folder name is same.

When we work with IDE to configure in deployment descriptor we have two options. They are like source and design. To see the contents of web.xml file we can use an option source.

By using graphical user interface if we want configure servlets. We can use design view.

When we use IDE we no need to deploy the project manually. We can configure IDE to deploy the project. As part of JEE respective. we can use server view to configure the project in IDE.

Step 1: Get the server view and right click chooses an option configure server connector.

Step 2: Choose the appropriate server the home directory of the server. Same servers required user name and password supply these values and click on "OK".

Step 3: To add the project to server we can use an option add deployment.

Step 4: They are some IDE's are available which are responsible to interact with database Servers. The popular IDE's for oracle database server are TOAD and SQL developer.



For the database server like MYSQL we can use “MYSQL workbench”.

Step 5: In the companies because of licenses issues we may not use the IDE like TOAD.

Eclipse gives has provided a feature to interact with database servers. We can use Eclipse/MYECLIPSE to interact with database servers.

**\*Procedures to configure IDE to interact with database server:**

1. Open MYECLIPSE database explorer of prospective.
2. In the database browser view when we right click it lunches a popup menu from that choose an option new.
3. The above step as lunched a window whose name is database driver from that choose driver template and provide a driver name. Now we have to supply driver class, url, username, password to interact with DB server.

**\*Hibernate is a frame which is used to interact with database server:**

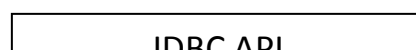
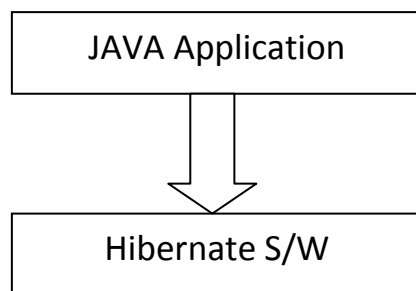
There are so many frameworks are available in the market there are some of struts, springs, JSF etc.

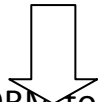
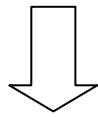
1. Struts, spring frame works are used to develop web-based applications.
2. Hibernate is a technology are a framework which is used to interact with database server this framework resolves all the problems of JDBC. In a project if he tries to use hibernate frame work if he develop the project quickly.
3. As part of JDBC we have to write lot of code to hardly checked Exception.
4. As part of Hibernate we no need to provide the code to handle checked Exception.
5. If we use JDBC a programmer is responsible to register the driver and get the connection. Programmer must provide the code to close the connection. As part of Hibernate he is responsible to open the connection and close the connection.
6. As part of Hibernate it's only starts the transaction and ends the transaction. Hibernate internally uses JTA (Java Transaction API). To start the transaction and end the transaction.



7. Hibernate support internal connection provides Hibernate uses C3B connection pool. DBCP connection pool. If we went to use external connection pool like web-logical we can configure it. If we use JDBC we have to provide the code to handle database specific error code.
8. If we use Hibernate we no need to provide the code to handle database specific errors.
9. Hibernate supports it's own query language HQL (Hibernate Query language) to resolve the problems of some java application to interact with multiple DB Servers without changing the queries.
10. In case of JDBC we have to provide the code to remove the hard coding. In case of Hibernate we provide the details in Hibernate configuration by reading the details from configuration file (.XML). It will interact with multiple Database Server.
11. If we use JDBC programmer is responsible to write the code to generate the primary key values. As part of Hibernate they have provided pre-defined class to generate primary key values.
12. By using Hibernate we can represent the queries in the form of object or we can use criteria API. As part of Hibernate we can achieve polymorphism between the tables and we can achieve association also.
13. \*by using JDBC we can't transfer result set object from one program to another program (result set can't be transferable) to resolve this problem we have to provide the code to retrieve data from result set object and store it in array list object if we use Hibernate. Hibernate does all these work.
14. \*we have the technology like EJB to interact with DB Server we can't run EJB Application without EJB container. We can't use EJB's in all type of application we can resolve all these problems we can use Hibernate.

\*The following is true Architecture of Hibernate framework.



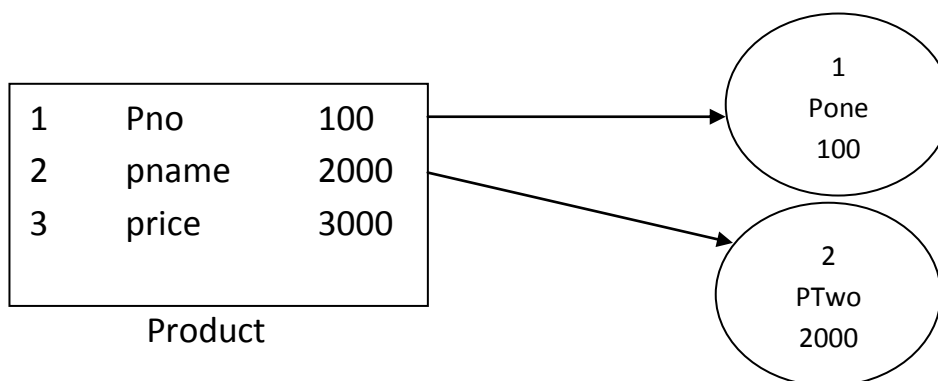


\*Hibernate is called as ORM tool are ORM framework ORM (Object Relational Mapping). In the market they are so many ORM frame works are available some of them ere Hibernate, JPA (java persistent API) JDO (java data object) i batches, STO (service data o/p), Top Link etc.

All these frame works are used as ORM Tools. The differences between the Tools are the names of the classes are different.

In the database server the data will be stored in the side the tables between the tables we establish a relationship because of this reason we call it is relational data.

\*Representations relational records in the form of object we call it has ORM:



\*can we use JDBC to represent records in the form of object?

A. Yes, we can use JDBC to represent the records in the form of object. In case of JDBC we have to provide huge amount of code to represent huge amount of data in the form of object.

B. As a Hibernate developer we need to develop a java application with uses Hibernate API and interact with database server. If we want to develop a java application we need to configure DB server and provide some additional files to Hibernate software.



C. If the Hibernate software to communicate with database server we required a user with special privileged (sequence) the tables in the database server must contain primary keys.

**\*Procedure to create user and assign some privileges:**

1. Login to database server using administrative user use the following command assign the user and privileges.

SQL> create user hib identified by abc;

Grant connect, resource to hib;

Grant create sequence to hib;

2. We have to create the table with primary key.

SQL> create table emp (eno number(4) primary key, ename varchar2(20),  
Address varchar2(20));

SQL> create table product (pid number(5), name varchar2(20), price number(10,2));

SQL> alter table product add primary key(pid);

3. We can get Hibernate software from Hibernate.org

4.

5. Hibernate software is a collection of jar files the most important jar file is Hibernate.jar

6. If we want to use Hibernate software we have to develop the following 3 files.

6.1. Hibernate configuration file

6.2. Java Beans (POJO classes) (plain Old Java Object)

6.3. Hibernate Mapping files (hbm)

7. Every framework uses configure file. This is the starting point to any framework generally the configuration files or xml files (we can use property files also as configuration files)



8. \*in Hibernate we use Hibernate.cfg.xml as the configuration file we can change the configuration file name according to requirement.

9. Hibernate configure file contains the entire information which is required to communicate with database server. They are:

- |    |              |
|----|--------------|
| a. | Driver class |
| b. | url          |
| c. | username     |
| d. | pwd          |
| e. | dialect      |

```
driver_class =oracle.jdbc
url = jdbc.ora....
username = hib
password = abc
dialect = OracleDialect
//information about HRM files
```

Hibernate.cfg.xml

10. Hibernate software required the java program to represent a record in the form of object. Because of this reason we have to develop couple of java programs this is based on the number of tables available in the database server.

11. These programs contains instance variables, setRow() and gerRow() methods. Developing these programs are easy because of this reason POJO classes are known as (plain old java objects).

12. In our database server we have two tables emp and product. We need to develop two proto classes.

- |    |              |
|----|--------------|
| a. | Emp.java     |
| b. | Product.java |

**Note:** There is no rule saying protocol names and tables name must be same.

```
public class employee{
    int eno;
    string ename;
    string eddress;
```

13.

We have to develop

Hibernate mapping files. These files contain. The information about which protocol class is mapped with which table and which properties mapped with which column.

Employee (POJO)	→	Emp (Table)
EMPNO	→	ENO
EMPNAME	→	NAME
EMPADDRESS	→	ADDRESS

emp.hbm.xml

**Note:**

IDE takes care of generating Hibernate configuration file, proto classes and Hibernate Mapping files.

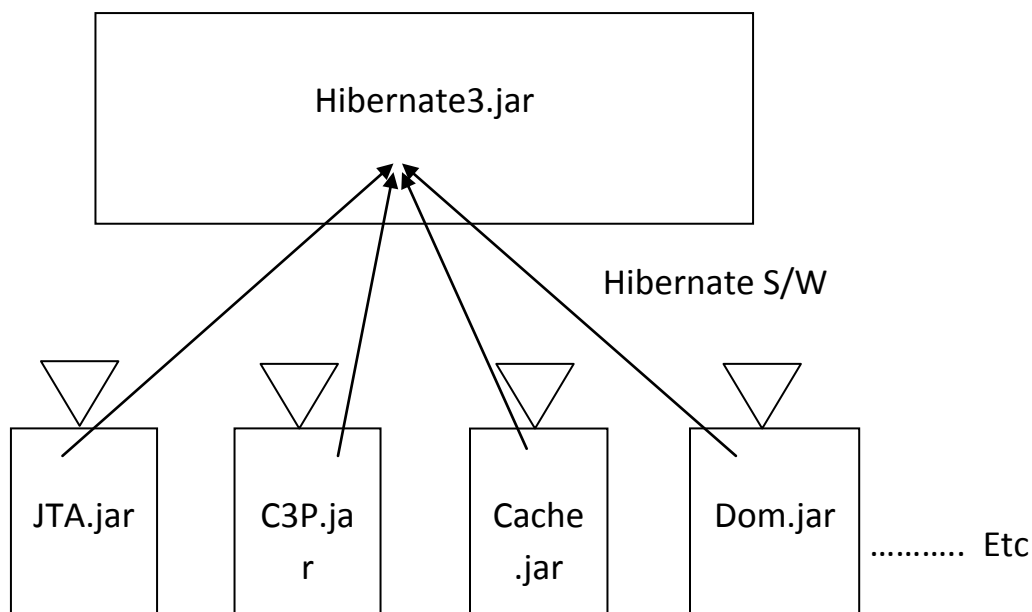
**\*The POJO classes must following rules:**

1. Must contain a default constructor.
2. Must be placed inside a package.
3. The POJO class supports properties. A setRow() and getRow() methods are called as properties.
4. When the Hibernate people are developing software they are started using the help of other software's. They are like Transaction API, Connection Tool, Dom Parse, cache jar files. If we would like to run the Hibernate software are





need to make sure that all these jar files generate the CLASSPATH other wise Hibernate software will fail to run.



\*\*\*Procedure to configure the Hibernate software and generate Hibernate configuration file by using MyEclipse IDE.

- I. Start MyEclipse IDE  
pointing to works place folder.
- II. Configure MyEclipse IDE  
to interact with the DB Server.(create DB Browser)
- III. Create a java project  
and add the package to it.
- IV. Add Hibernate project capabilities  
capabilities to the above project. MyEclipse  
→ Hibernate capabilities.



- V. Choose the required jar files to be added. (Core library and notations).
- VI. Choose new Hibernate configuration option and click on next button. From the list of available DB Driver select to which database server we have to connect.
- VII. Choose the package name and provide the class name as ORASF and click on finish button.

**\*Procedure to generate hbm files and POJO classes:**

Step 1: Go to MyEclipse db explorer perspective and establish the connection with Database server.

Step 2: Select the all required tables and right click on it generates @ launch pop-up menu from that choose on option Hibernate reverse Engineering.

Step 3: Select src and package and check the first 3 check boxes and click on finish button launches abstract class.

Step 4: The following is the configuration is Hibernate configuration file.

```
<hibernate-Configuration>
  <Session-Factory>
    <property name = "Connection.driver_class"> oracle.jdbc.driver.OracleDriver
                                          </property>
    <property name = "Connetion.url"> jdbc:oracle:thin:@localhost:1521:xe</property>
    <property name = "Connection.username">hib</property>
    <property name = "dialect">org.hibernate.dialect Oracle9iDialect</property>
    <mapping resource = "info/inetsolv/product.hbm.xml"/>
    <map resource = "info/inetSolv/Emp.hbm.xml"/>
  </Session-Factory>
</hibernate-configuration>           // Save hibernate.cfg.xml
```

Step 5: The following is the POJO class of emp tables.



```

public class Emp{
    private Integer eno;
    private String name;
    private Double salary;
// provide setters and getters methods
    above instance variables.
}      // Save Emp.java

```

Step 6: The following on the tags of Employee file.

```

<hibernate mapping>
    <class name = "info.inetsolv.Emp" table = "Emp">
        <id name = "eno" type = "java.lang.Integer">
            <column name = "ENO" precision = "5"/>
            <generator class = "assigned"/>
        </id>
        <property name = "name" type = "java.lang.String">
            <column name = "NAME"/>
        </property>
        <property name = "salary" type = "java.lang.Double">
            <column name = "SALARY"/>
        </property>
    </hibernate mapping>

```

\*The following are the most important interfaces and classes.

**\*Interfaces:**

- |    |                              |
|----|------------------------------|
| 1. | org.hibernate.SessionFactory |
| 2. | org.hibernate.Session        |
| 3. | org.hibernate.Transaction    |
| 4. | org.hibernate.Query          |
| 5. | org.hibernate.Criteria       |

**\*Classes:**



## 1. uration

org.hibernate.cfg.Config

\*The following are the steps which will be used to develop Hibernate application.

- 1) Create Configuration Object.
- 2) Call the Configuration() method by using the Configuration Object.
- 3) Get SessionFactory Object by using Configuration Object, we use a method build SessionFactory().
- 4) Get Session Object by using SessionFactory. Call the method openSession().
- 5) Get the Transaction Object.
- 6) Create the POJO class Object which we would like to perform the Operations.
- 7) Store the data in the POJO class Object.
- 8) Call the methods save/update/delete/load methods.
- 9) End Transaction by using commit/rollback.
- 10) Close Session Object.
- 11) Close SessionFactory Object.

### **\*Requirement:**

Develop a Hibernate application to insert records in to the emp table. The following is the java code which is used to store the data into emp table.

```
public class StoreEmpDetails{  
    public static void main(String args[]){  
        Configuration cfg = new Configuration();  
        cfg.configure();  
        SessionFactory sf = cfg.buildSessionFactory();  
        Session hsession = sf.openSession();
```



```

Transaction tx = hsession.beginTransaction();
Emp e = new Emp();
e.setEno(1);
e.setName("Raju");
e.setAddress("Hyd");
hsession.save(e);
tx.commit();
hsession.close();
sf.close();
    }
}

```

**Note:**

When we Run the same application for two times we got an Exception saying that ConstraintViolationException.

1. When we create the configuration object we are create an environment to store the configuration details. In this configuration object we can store Driver class, url, username, password and mapping information.
2. When we call `cfg.configure` it checks for `hibernate.cfg.xml` file in the CLASSPATH. If it is available it start reading the contains from hibernate configuration file. Now the hibernate file the corresponding hbm files it opens all the hbm files and try to read the contents from all hibernate mapping files. All this information stored a JVM's memory (configuration object). If the configuration object not available in the class objects it throw Exception `org.hibernate.hibernateException`.

**Note:**

This method is an Expensive operation. In a project it is recommended to call the configure method only once in the project life cycle.

```

public class StoreEmpDetails{
public static void main(String args[]){
Configuration cfg = new Configuration();
cfg.configure();
SessionFactory sf = cfg.buildSessionFactory();

```

```
Session hsession = sf.openSession();
Transaction tx = hsession.beginTransaction();
Emp e = new Emp();
Product p = new Product();
e.setEno(1);
e.setName("Raju");
e.setAddress("Hyd");
p.setId(1);
p.setName("Rice");
p.setAmount(1000);
hsession.save(e);
hsession.save(p);
tx.commit();
hsession.close();
sf.close();
    }
}
```

This method is an expensive operation in a project it is recommended to call the configure method only once in the project life cycle.

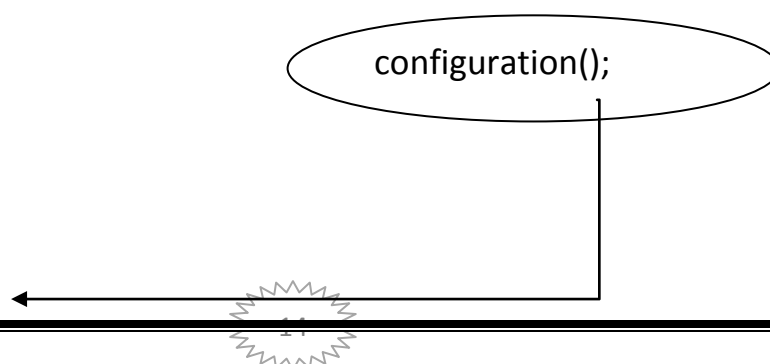
In any frame wrote we can change the configuration file name according to our requirement.

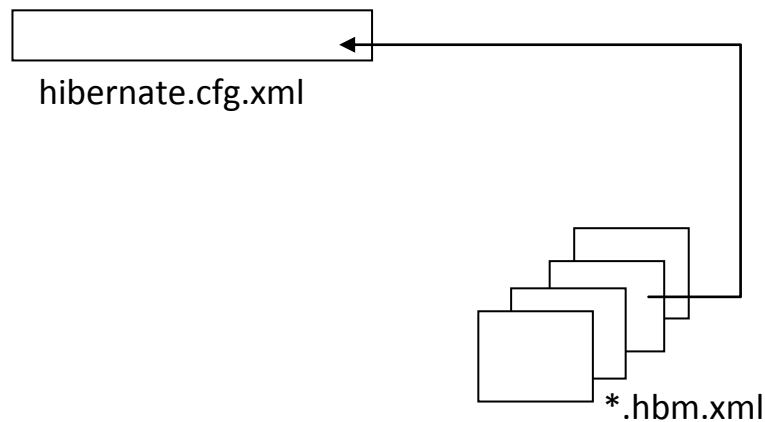
If you are using our own configuration file name we have to use the over loaded configuration method.

**Ex:** cfg.configure("myproject.xml");

**Note:**

The default configure method always check for hibernate cfg.xml file. It always recommends using .cfg in the configuration file name. Because of this we can easily recognize the configuration file name.





When we call `cfg.buildSessionFactory()` method it gets driver class, url, username and password these values are supplied as input to hibernate internal connection pool. Now the hibernate internal connection pool. Get the connections from database servers. Now the build `SessionFactory` method gets a connection from connection pool and establish the connection with database server. It will check whether all the required tables are available or not. If not available if required build `SessionFactory()` method create the tables. It is the responsibility of hibernate build `SessionFactory()` to create all the 'CURD' queries for every table and store the queries in JVM's memory now the build `SessionFactory` close the connection.

**Note:** calling build `SessionFactory()` method repeatedly in the project is not recommended. It is recommended to call only once in life time of the project.

By default build `SessionFactory()` method is not creating the tables. If you want to hibernate to create the tables we have to supply an additional property 'hbm2ddl.auto' these properties can take any of the following four values create, update and create-delete, validate.

The following is the tag which has to be added to Hibernate configuration file.

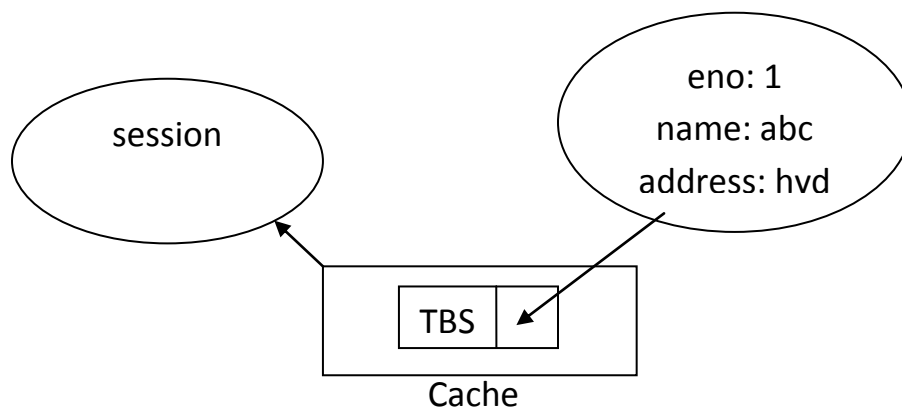
```
<property name = "hbm2ddl.auto">update</property>
```

It is always advisable to get a session Object when ever we would like to carry out any work by using Hibernate. It's always recommended to close the session Object after we finish the work. Getting a session object is similar to getting a connection object in JDBC.

When ever the session object is created immediately Hibernate starts a cache object and it will be associated to session object are all this cache objects as 1<sup>st</sup> level cache Hibernate remove the cache object when ever we close the session object.

Arrays we have to start transaction after the session object is created. We need to start the transaction only for insert/update/delete operation only. We no need to start the transactions for retrieve the records.

When we call hsession save hibernate takes the object and add it to first level cache by using a registration code is “TBS” to be save.



When we call “tx.commit” hibernate got first level cache and check are there any object are available in first level cache it the objects are available hibernate check. The registration code of the object hibernate find to which POJO class this object is created and to which table this POJO class is mapped.

Based on the table name and registration and hibernate get the insert query from the JVM’s memory. It is the responsibility hibernates to replace positional parameters with appropriate values from the object. Now the hibernate add the query to batch object.

Now the hibernate send the batch object to Database server if it got execute successfully it returns an identifies value. If the batch is failed it throws an exception batch update exception.

Hibernate send a sql query to Database server to see the sql query sent by hibernate we can add property “show-sql”. This attribute takes a Boolean value.



**Ex:**

```
<property name = "show_sql">true</property>
```

To see all the messages are work done by hibernate we can use log4j.properties.

```
log4j.root logger = DEBUG/A1  
log4j.appender.A1 = org.APAche.log4j.console Appender  
log4j.appender.A1.layout = org.apache.log4j.simple layout
```

We can use a method "persist" to store the data into database server.

**Syntax:** void persist(object)

**Ex:** serializable save(object)

When we generate hbm files and POJO classes in hibernate by using IDE based on the column data type IDE generate the appropriate data type in POJO class for example eno number(15) IDE uses big decimal data type for eno number(2) IDE uses byte and etc.

As part of java5.0 sun micro system as added a feature auto boxing. The advantage of auto boxing is we LAN directly convert primitive data type values to wrapper classes.

**Ex:**

```
int a = 10;  
integer i = a;  
system.out.println(i);
```

When we trying to dual with auto boxing for double data type as shown below we are getting the compilation error.

**Ex:** Double d = 20d;

\*develop a Hibernate application to retrieve a record from emp table where eno is 2?

```
public class RetrieveRecords{  
public static void main(String[] args){
```



```

Configuration cfg = new Configuration();
cfg.configure();
SessionFactory sf = cfg.buildSessionFactory();
Session hsession = sf.openSession();
Emp e = new Emp();
Hsession.load(e,new BigDecimal(2));
System.out.println("e.getEno());
System.out.println("e.getName());
System.out.println("e.getSalary());
hession.close();
    }
}

```

When we call the load() method the internal code of Hibernate has performed the following steps.

Step 1: It has checked the corresponding POJO class name for the supplied object.

Step 2: It has checked this POJO class object is mapped to which table for that table Hibernate has picked appropriate select query. The following is the query available in JVM's memory.

- Select eno, name, salary from emp where eno =? Now the Hibernate has replaced the positional parameter value with 2 and send the query to database server.
- Database server has executed the select query and represented the records in the result set object and given into hibernate software.
- Hibernate software has taken from ResultSet object and by using the getter method got the data from ResultSet object and stored it in POJO class object.
- Now the Hibernate added the POJO class object to 1<sup>st</sup> level cache.

**Note:** If we try to call a load method on a non available record Hibernate throw an exception saying 'org.hibernate.ObjectNotFoundException'.

To check weather any object is available in 1<sup>st</sup> level cache or not we can use a method 'contains()'.

**Syntax:** boolean Contains(object)

\*Write a hibernate application to delete the records from product table whose productID is 11. To delete the records from hibernate we can use two approaches.

**Approach 1:** Load the record and mark the object as to delete.

```
public class DeleteRecords{
    public static void main(String[] args){
        Configuration cfg = new Configuration();
        cfg.configure();
        SessionFactory sf = cfg.buildSessionFactory();
        Session hsession = sf.openSession();
        Transaction tx = hsession.beginTransaction();
        Product p = new Product();
        hsession.load(p,11);    // step 1
        hsession.delete(p);    // step 2
        tx.commit();           // step 3
        hsession.close();
    }
}
```

Step 1: when step 1 is executed it has retrieve the records whose primary key value is 11 and add into 1<sup>st</sup> level cache.

Step 2: When we call the method object is marked as to be deleted.

Step 3: when we call the commit method the hibernate software got the delete query and replaces the positional parameter with primary key value and send the query to database server.

In this approach first we are checked in whether the record is available or not if the record is not available. The load() method throws object not found exception.

**Approach 2:** Create the POJO class object and supply the primary key value. Mark the POJO class object as to be deleting by calling the delete method.

**Ex:**

```
public class DeleteRecords1{
    public static void main(String[] args){
```



```

Configuration cfg = new Configuration();
cfg.configure();
SessionFactory sf = cfg.buildSessionFactory();
Session hsession = sf.openSession();
Product p = new Product();
p.setpid(11);
hsession.delete(p);
tx.commit();
hsession.close();
    }
}

```

In this approach when we call the delete method object is marked as to be deleted. When we call the commit() method it has perform the following 3 steps.

Step 1: It check weather primary key value is available in the supplied object or not. If not available it will not carry out any work.

Step 2: If the primary key value is available a select query will be send to database server to check weather record is available or not.

Step 3: If the record is available in DB hibernate send the delete query. If the record is not available hibernate will not do any work.

**Approach 1:** Updating a record into database server which ever the record we would like to update load the record by calling load() method modify the values by using setter methods in the loaded POJO class object. Now mark the object as to be updated.

**Ex:**

```

public class UpdateRecord{
public static void main(String[] args){
Configuration cfg = new Configuration();
cfg.configure();
SessionFactory sf = cfg.buildSessionFactory();
Session hsession = sf.openSession();
Emp e = new Emp();
hsession.load(e, new BigDecimal(2));
e.setName("Raju");

```



```

hsession.update(e);
tx.commit();
hsession.close();
    }
}

```

**Approach 2:** Hibernate uses a directory object technique to check whether object value is modified or not. If the value is not modified. Hibernate will not send any update query to the database server. If the values are modified Hibernate send an update query to database server.

**Approach 3:** In this approach create POJO class object to the class which we would like to update the record and store the data into POJO class object. We need to mark the object as to be updated.

**Ex:**

```

public class UpdateRecord{
public static void main(String[] args){
Configuration cfg = new Configuration();
cfg.configure();
SessionFactory sf = cfg.buildSessionFactory();
Session hsession = sf.openSession();
Emp e = new Emp();
e.setEno(new BigDecimal(20));
e.setName("ttt");
hsession.update(e);
tx.commit();
hsession.close();
    }
}

```

\***evict()**: evict() method is used to remove a specified object from the 1<sup>st</sup> level cache.

**Ex:**

```

Transaction tx = hsession.beginTransaction();
hsession.load(e,new BigDecimal(1));
e.setName("Raju");

```



```
hsession.evict(e);  
tx.commit();
```

When we run the above application with out evict() method. It has update a record into database server. When we run the some application with evict() method. It has removed employee object from 1<sup>st</sup> level cache.

\***merge()**: merge method is used to add a specified object to the 1<sup>st</sup> level cache.

**Ex:**

```
Emp e = new Emp();  
e.setEno(new BigDecimal(22));  
e.setName("ABC modified");  
e.setSalary(1234d);  
hsession.merge(e);  
tx.commit();
```

When the merge() method is called the object is added to 1<sup>st</sup> level cache without registration code. When tx.commit() method is called it will get the object which does not contain the registration code. It will check weather the object is available in database server by sending select query. If the record is not available it will send an insert query to database server. If the record is already available it will send a an update query to database server.

\*There three states are available to hibernate objects they are:

1. Transient
2. Persistent
3. Detached

**Transient**: An object is which is not associated with any session object.

**Persistent**: An object which is added to 1<sup>st</sup> level cache is called as persistent state.

**Detached**: An object which is removed from 1<sup>st</sup> level cache is called detached state.

\***Clear()**:

Clear is used to remove all the objects from 1<sup>st</sup> level cache. This will remove unperformed operations like save and update also. The clear() method will evict all available objects.



### **\*Connection():**

This method is used to get legacy database connection. Generally this is not recommended approach in hibernate. We use this to perform some operations which can not be done by using hibernate.

#### **Ex:**

```
Session hsession = sf.openSession();  
Transaction tx = hsession.beginTransaction();  
Connection con = hsession.Connection();  
Statement stmt = con.createStatement();  
stmt.executeUpdate("Insert into emp values(2,'sadaf',234)");  
tx.commit();
```

In Hibernate when we get the connection object by default auto commit mode to false. We have multiple over loaded methods as for session interface they are:

```
void load(object, pid)  
object load(class, serializable)
```

#### **Ex:**

```
Class c = class.forName("info.inetsolv.product");  
Object o = hsession.load(c,l);  
Product p = (product)o;  
System.out.println(p.getPid());  
System.out.println(p.getName());  
System.out.println(p.getPrice());
```

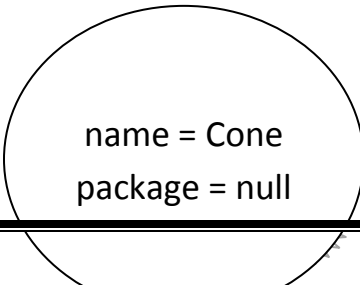
When we call the above load() method if the record is available load() method creates POJO class object and store the data and return POJO class object if the record is not available load() method will not create POJO class object.

We have a static variable class as part of object class when ever we call that variable by using class name it returns the calling class classobject.

#### **Ex:**

```
Cone.class
```

When the above code is executed it has return class object.



name = Cone  
package = null

## Class

### **\*get():**

get() method is also used to retrieve the record from database server if the record is available it returns that POJO class object. If record is not available it returns null value.

### **Ex:**

```
Object o = hsession.get(product.class,2);
If(o != null){
    Product p = (product)o;
    System.out.println(p.getPid());
    System.out.println(p.getName());
    System.out.println(p.getPrice());
}
else{
    System.out.println("Record is not available");
}
```

### **\*flush():**

When we call in flush() method all the objects which are available in 1<sup>st</sup> level cache will be converted into queries and send to database server. Flush will not store data permanently. When we call the commit() method the data is stored permanently.

Hbmaddl.auto property takes by different values

- |    |             |
|----|-------------|
| 1. | Update      |
| 2. | Create      |
| 3. | Create-drop |
| 4. | Validate    |

If hbmaddl.auto = update and weather the buildSessionFactory() method is executed it checks weather tables are available or not. If not available it create the takes.

If hbmaddl.auto = create if the tables are not available buildSessionFactory() method creates it. If the takes and create again if hbmaddl.auto = create-drop if the tables are not



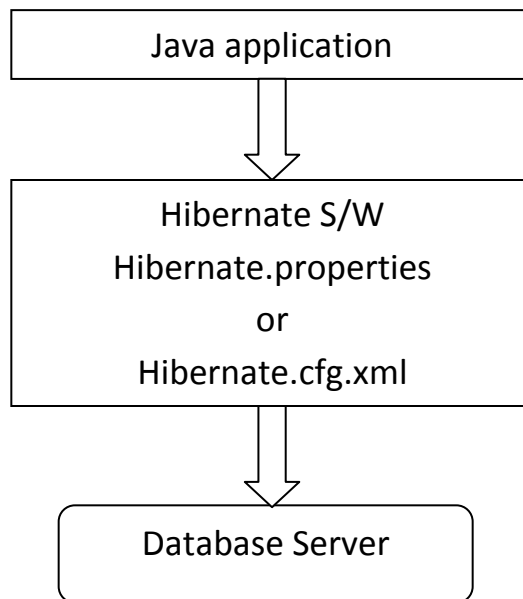


available it creates the tables. when we close the session factory object the tables will be dropped.

If hbmddl.auto = validate, buildSessionFactory() method check weather the tables are present in the database server or not. If not available it will throw an error message missing table.

**\*Developing Hibernate application by using manual procedure:**

The following is an architecture of hibernate application.



Step 1: Create a table in the data base server.

Step 2: Get the Hibernate software and placed in a lib folder (copy ojdbc14.jar also).

**Note:**

We can get the jar files from IDE.

Step 3: Develop a cmd file which contains the class path to all the Hibernate related jar files.

**Ex:** Set CLASSPATH=lib\antlr.2.7.6.jar;lib\C3PO-0.9.1.jar;

Step 4: Create the POJO class. We can use any class name as POJO class name for example.

public class Employee{



```

int empNo;
String empName;
double empSalary;
public void setEmployeeNo(int employeeNo){
this.empNo = employeeNo;
    }
public int getEmployeeNo(){
return empno;
    }

```

\*Create hbm file the name of hbm file can be any thing.

```

<?xml version = "1.0"?>
<! DOCTYPE hibernate.mapping public "something"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0dtt">
<hibernate-mapping>
<class name = "info.inetsolv.Employee"table = "emp">
<id name = "employeeNo">
<column name = "eno"/>
<generator class = "assigned"/>
</id>
<property name = "employeeName">
<column name = "name"/>
</property>
.....
</class>
</hibernate-mapping>

```

\*Develop hibernate configuration file.

```

<? xml version = "1.0"?>
<! DOCTYPE hibernate-configuration public "hibernate-configuration",
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session factory>
    .....
</session factory>

```



</hibernate-configuration>

Develop a java application to store the data into database server.

The parser program check for hibernate dtd file as part of hibernate file.

As part of hbm file we can remove the column tag or attribute if the POJO class properties and column names are same.

\*The following is sample configuration for emp table.

```
<hibernate-mapping>
<class name = "info.inetsolv.Employee">
<id name = "employeeNo"/>
<property name = "employeeName"/>
<property name = "employeeSalary"/>
</class>
</hibernate-mapping>
```

We can club multiple hbm files into a simple hbm file. But this approach is not recommended for big projects, it is recommended to use one hbm file BrOne one POJO class.

```
<hibernate-mapping>
<class name = "info.inetsolv.Employee">
<class name = "info.inetsolv.Product">
<id name = "Pid" access = "field"/>
<property name = "name" class = "field"/>
</class>
</hibernate-mapping>
```

We can develop hibernate application with out hibernate configuration file. But we have to provide the properties and mapping files through the java program.

**Ex:**

```
public class Store{
public static void main(String[] args){
Configuration cfg = new Configuration();
cfg.setProperty("hibernate.connection.driver_class","oracle.jdbc.driver.OracleDriver");
cfg.setProperty("hibernate.connection.url","jdbc:oracle:thin:@localhost:1521:xe:");
```



```

cfg.setProperty("hibernate.connection.username","hib");
cfg.setProperty("hibernate.connection.password","abc");
cfg.setProperty("hibernate.dialect","org.hibernate.dialect.Oracle9Dialect");
cfg.setProperty("hibernate.show_sql","true");
cfg.addResource("a.hbm.xml");
sessionFactory sf = cfg.buildSessionFactory();
session hession = sf.openSession();

.....

.....

....
    }
}

```

Instead of addResource() method we can use addClass() method.

**Ex:**

```
cfg.addClass(info.inetsolv.product.class);
```

when we use addClass() it will check for info.inetsolv.product.hbm.xml file.

The disadvantage of programmatic configuration is when Hard coding the values in the java program. If we want to communication with same hibernate application with different database server. We have to change the java code because of this reason this approach is not recommended.

Generally in the projects we use properties files to remove hard coding. Most of the projects uses property file end with an extension dot (.) properties inside the properties file we supply a data in the form of key and value.

**Ex:**

```
Key = value    //myproject.properties
```

We can configuration hibernate parameters as part of a property-file. Who's name is hibernate properties.

**Ex:**

```

hibernate.connection.driver_class = oracle.jdbc.driver.OracleDriver
hibernate.connection.url = jdbc:oracle:thin:@localhost:1521:xe
hibernate.connection.username = hib
hibernate.connection.password = abc
hibernate.dialect = org.hibernate.dialect.Oracle9Dialect

```



hibernate.show\_sql = true

//save hibernate.properties

It's not recommended to use property file as part of hibernate. This is because as part of the property file are can't configuration the mapping resource files.

We can supply the values by the properties using system properties when we run the application.

**Ex:** -Dhibernate.connection.driver\_class = oracle.jdbc.driver.OracleDriver

Meagerly hibernate is divided into three parts. They are:

Connection Management
ORM
Transaction Management

Hibernate S/W

Hibernate S/W is good at ORM as well as transaction management the internal hibernate code uses two connections pools C3P, DBCP. It's not recommended to use there connection pools. It's always recommended to use the external connection pool like **weblogic** connection pool.

\*Using procedure to use weblogic connection pool program in hibernate.

1. Configure weblogic server connection pool by specify JNDI name.
  2. Get the hibernate S/W place in lib folder and set the class path.
  3. Create POJO class and hbm file.
  4. Create hibernate configuration file with data source, username and password and jndi.class and jndi.url
- ```
<hibernate-configuration>  
<Session-Factory>  
<property name="hibernate.connection.datasource">mypool</property>
```



```

<property name="hibernate.connection.username">admin</property>
<property name="hibernate.connection.password">inetsolv</property>
<property name="hibernate.jndi.class">weblogic.jndi.WLInitialContextFactory
    </property>
<property name="hibernate.jndi.url">t3://localhost7001/</property>
<property name="hibernate.show_sql">true</property>
<mapping resource = "a.hbm.xml"/>
</Session-Factory>
</hibernate-configuration>

```

5. Develop the hibernate application to store the data into database server.
- Note:** we have set the class path to web.jar or set DomainEnv.cmd
6. When we are configuration are using connection pool as part of hibernate we have choose use jndi data source rather then jdbc driver.
7. To set the class path to resolve the problem of weblogic.jndi.WLInitialContextFactory we are added weblogic.jar in the class path.
8. Hibernate can be used to communicate with any DB server.

**\*Procedure to develop hibernate application to interact with MYSQL:**

When we communicate with MYSQL DB Server. We have change the url, username, driver class and etc.

We have to always use "wrapper classes" as part of POJO classes instead of primitive data types. Primitive data types occupy less amount of memory when compared with wrapper classes. Primitive data types can't hold a null value.

**Ex:** public class MyApp{  
 public static void main(String[] args){  
 int a = null;  
 System.out.println(a);  
 }  
}



When we compile the above program the compiler reports error message saying null value can not be assigned to primitive data types. we can assigned a null value to wrapper classes.

**Ex:**

```
public class MyApp{
    public static void main(String[] args){
        Integer a = null;
        System.out.println(a);
    }
}
```

As part of the IDE when we generating hbm files and POJO classes we have an option. We have primitive data types or wrapper classes. They are java types and hibernate types. When we choose java types it uses wrapper classes. When we choose hibernate types it uses primitive data types.

We are try to use primitive data types and trying to store data into tables. In these scenarios the default values of primitive data types are getting store into data base server.

**Ex:**

```
Product p = new Product();
p.setPid(Pid);
p.setName(Pname);
hsession.save(p);
```

When we execute the above code even though when the user is not supplied price value application has stored 0.0 values.

When we use primitive data types to retrieve records from a table which contains null values. then the application throwing an exception.

**“org.hibernate.propertyAccessException”.**

By using JDBC also we can represent the records in the form of objects.

The following is an example of representing data in the form of object by using JDBC?

**Ex:**

```
ResultSet rs = stmt.executeQuery(“select * from product”);
ArrayList list = new ArrayList();
While(rs.next()){
    Product p = new Product();
```



```

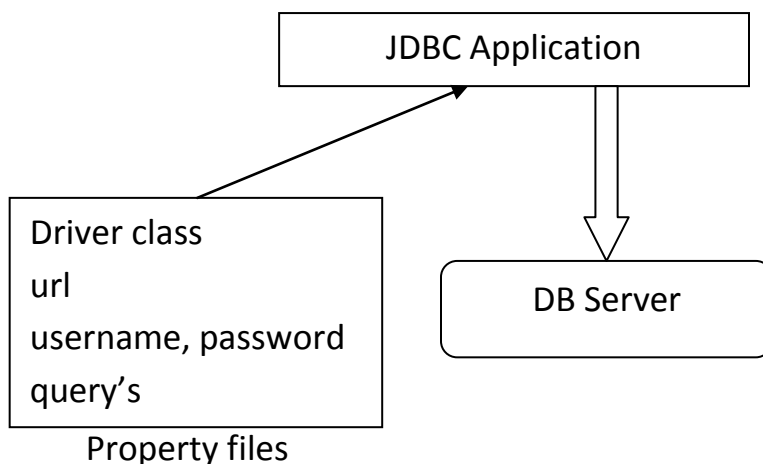
p.setPid(rs.getInt(1));
p.setPid(rs.getString(2));
p.setPid(rs.getDouble(3));
list.add(p);
}
System.out.println("After: " +list.size());
}
}

```

Hibernate guys has given the following three API's perform multiple row operations. They are:

1. HQL API
2. Criteria API
3. Native sql API

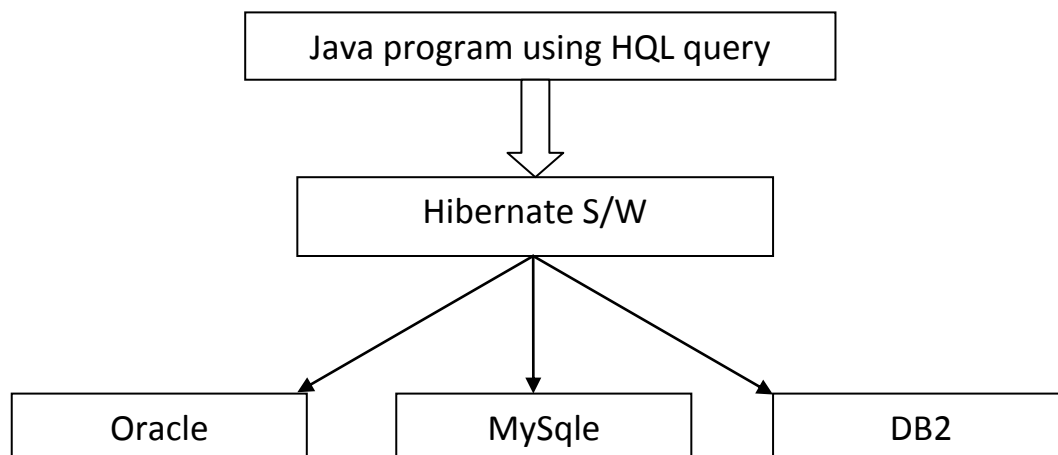
When we develop jdbc application to communicate with any database server to remove the hard coding we use property files. As part of the property files we have provided driver class, url, username, password and queries. If we would like to communicate with other database servers we change the values from property files.



HQL queries as given as input to hibernate S/W it is the responsibility of hibernate to convert HQL query into corresponding SQL queries and send into data base server.







Hibernate guys has provided so many dialect classes. As part of this dialect classes the code is provided to convert HQL query to corresponding sql queries.

The dialect classes are available in **org.hibernate.dialect** package. The following are some of the dialect classes. Oracle dialect, MySql dialect, SAP dialect etc.

The following are the features of sql.

- |    |                                                 |     |
|----|-------------------------------------------------|-----|
| 1. | Queries fully object oriented.                  | HQL |
| 2. | Queries supports inheritance and poly morphism. | HQL |
| 3. | Queries are case sensitive.                     | HQL |

If we want to write HQL queries instead of table names we have to use POJO calss names. Instead of column names we have to use property names HQL queries derived from sql queries. The following is HQL queries derived from sql queries. The following is HQL and sql queries to retrieve all the records from emp table.

SQL> select \* from emp;

→ Table name

HQL> from info.inetsolv.Employee

→ POJO calss name

Develop a hibernate application which uses HQL queries to retrieve all the records from emp table.

**\*Procedure to use HQL in herbernate:**

1. Represent  
HQL query in the form of query object.
2. Send the  
query object to hibernate software by calling the list method.
3. Hibernate  
S/W returns an ArrayList object render the ArrayList object and display the output to client.
4. Query is  
an interface which is available as port of org.hibernate package. We can not create the object to query interface. We can create a reference variable and it holds implementation class object.

```
public class RetrieveData{
public static void main(String args[]){
// Standard code
Session hsession = sf.openSession();
Query query = hsession.createQuery(hql query);
List l = query.list();
ArrayList emplist = (ArrayList)l;
Iterator i = emplist.iterator();
while(i.hasNext()){
Employee e = (Employee)i.next();
System.out.println(e.getEmployeeNo());
System.out.println(e.getEmployeeName());
System.out.println(e.getEmployeeAddress());
}
hsession.close();
}
}
```

When we call the list() method the following steps are carried out by list() method.

1. HQL query  
will be converted into corresponding SQL Query.
2. Hibernate  
S/W send SQL query to database server.

3. The database server executes select query and return the ResultSet object to hibernate S/W.
4. Hibernate S/W represents every record in the form of object and add it to array list object.
5. List() method converts ArrayList into super class reference variable list and it returns it we have to get ArrayList object and display records to client.

When we trying to add any element to ArrayList object it will be converted into super class object called as object. While retrieving data from ArrayList we need to typecast into appropriate object.

As part HQL queries we can add where conditions as well as order by clause, group by clause.

**\*using where clause as part of HQL:**

SQL> select \* from emp where eno>2;

HQL> from info.inetsolv.Employee where employee>2;

**\*using positional parameters in HQL:**

The following is an example of using positional parameters in HQL.

SQL> select \* from product where price >? And price<?

HQL> from product where price>? And price<?

**Ex:**

```
String query = "from product where price>? And price<?";
```

```
Query hqlquery = hsession.createQuery(query);
```

```
hqlquery.SetDouble(0,2000);
```

```
hqlquery.SetDouble(1,5000);
```

```
ArrayList List = (ArrayList)hqlquery.list();
```

In hibernate the HQL query positional parameter index starts with 0.

If we do not supply the values to all positional parameters hibernate display an exception query exception.

We can use alias names as part of HQL query by using a keyword as.

**Ex:**

```
String query = "from product as p where p.price>? and p.price<?";
```



We can use order by as part of HQL queries.

**Ex:**

from product order by price desc

**\*Retrieving specific columns from employee table:**

The following example get employeeNo, employeeName columns only from the database server.

```
String query = "select employeeNO,employeeName from info.inetsolv.Employee";
```

```
Query hql query = hsession.CreateQuery(Query);
```

```
ArrayList List = (ArrayList)hqlQuery.List();
```

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

```
While(i.hasNext()){
```

```
Object o[] = (object[]);
```

```
Next();
```

```
//code to display the data from arrays using index.
```

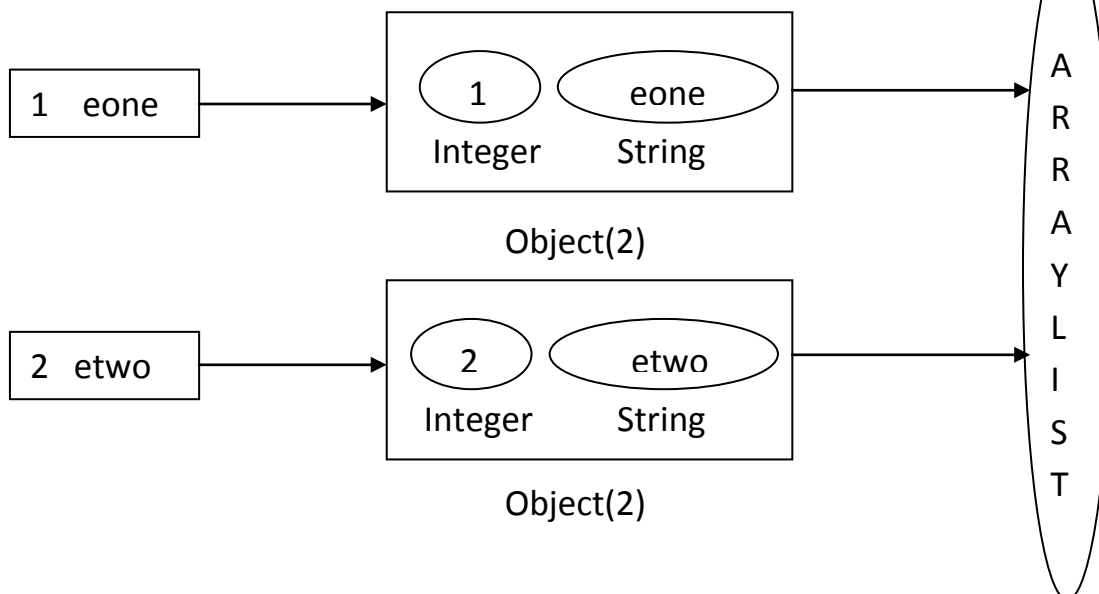
```
for(int j=0;j<0.lenth;j++){
```

```
System.out.println(o[j]);
```

```
}
```

```
}
```

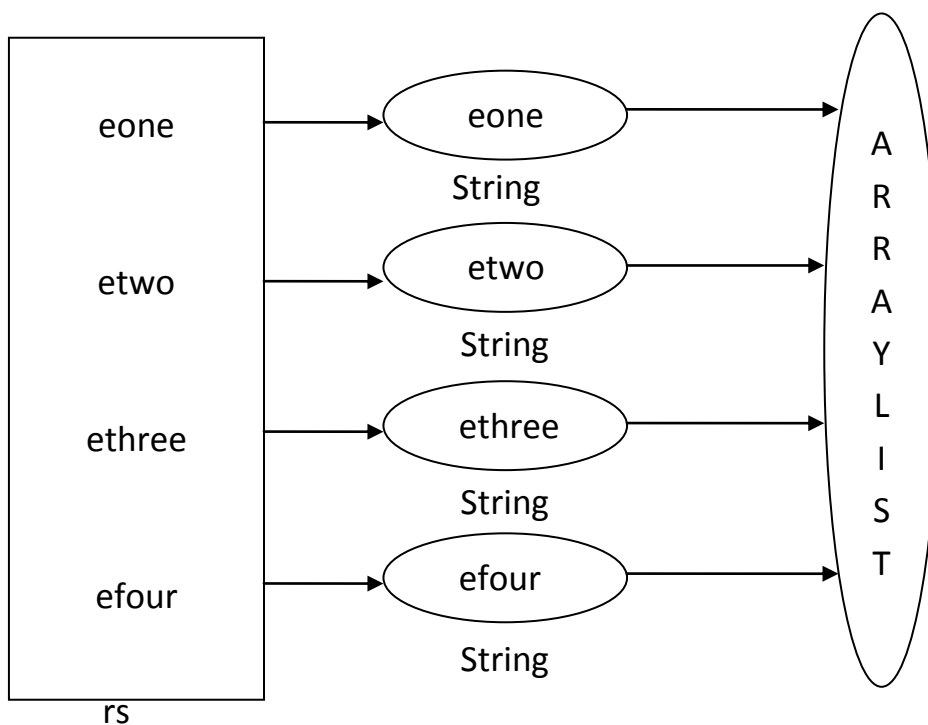
The following diagram shows how hibernate list() method as converted the appropriate data in the array list.



**Pro:** Develop a hibernate application to retrieve employee name column only from emp table.

```
String q = "select employee name from employee";  
Query que = hsession.CreateQuery(q);  
ArrayList list = (ArrayList)Query.List();  
Iterator i = list.iterator();  
while(i.hasNext()){  
Object o = i.next();  
Integer ii = (Integer)o;  
System.out.println(ii);}
```

When we have only one column in the select clause as part of the array list the data type will be added.



We can use aggregate functions in HQL queries. The following is an example using HQL to find number of records available in product table.

```
String q = "select count(*) from product as p";  
Query que = hsession.CreateQuery(q);  
ArrayList list = (ArrayList)que.List();  
Iterator i = list.iterator();
```

```

while(i.hasNext()){
Object o = i.next();
Long l = (Long)o;
System.out.println(o);
}

```

We can join multiple tables and retrieve the records the following is an example of join tables.

```

Query HQLQuery
Array list = (Arraylist)HQLQuery.List();
Iterator := list.iterator();
while(i.hasNext()){
Object o[] = (object[])i.next();
for(int j=0;j<0.length;j++){
Employee e = (Employee) o[0];
Product p = (Product) o[1];
System.out.println(e.getEmployee NO() + "\t");
System.out.println(e.getEmployee Name() + "\t");
System.out.println(p.getpid() + "\t");
System.out.println(p.getName() + "\t");
}
}

```

By using HQL we can perform updating, deletion and insert records into a database server. To send the update query we use a method executed update().

```

Transaction tx = hsession.beginTransaction();
String query = "update info.inetsolv.Employee set employee Address=! Where
employee No = ?";
Query hqlQuery = hsession.createQuery(Query);
hqlQuery.setString(0,"xyz");
hqlQuery.setInteger(1,1);
int no = hqlQuery.executeUpdate();
System.out.println(no);
tx.commit();
hsession.close();

```



```
}  
}
```

**Note:**

When we perform update, delete and insert operations. We must place the code in a transaction.

Hql will not support to insert record directly into table. if the data is already available in table we can copy from one table to another table.

```
Transaction tx = hsession.beginTransaction();  
String Query = "insert into Employee(employeeNo,employeeName) select pid,  
name from product";  
Query hql Query = hsession.createQuery(Query);  
int no = hql Query.executeUpdate();  
System.out.println(no);  
tx.commit();  
hsession.close();  
}
```

**Named positional parameters:**

When we write the hql Queries we can use positional parameters the problem with positional parameters is if somebody trying to read and understand it takes lot of time. Instead of positional parameters we use names. This will improve the readability of the queries.

Named positional parameters will not improve performance separately. This is because by default named positional parameters also uses prepare statements. The following is an example of named positional parameter.

**Ex:**

From product where price>:min value and price<:max value. The following is an example of using named positional parameters.

```
String Query = "from product where price>:minvalue and price<:maxvalue";  
Query hql Query = hsession.CreateQuery(Query);  
hql Query.setDouble("minvalue",2000);  
hql Query.setDouble("maxvalue",6000);  
ArrayList list = (ArrayList)hqlQuery.List();
```



In hibernate to retrieve the data from all the tables we can use hql query like from java.lang.object

**Ex:**

```
String Query = "from java.lang.object";  
Query hql Query = hsession.CreateQuery(Query);  
ArrayList list = (ArrayList)hqlQuery.List();
```

When the above list method is executed hibernate will send the queries to read the data from all the tables. The Array list object contains the POJO class object from table as well as emp tables.

### **Criteria API:**

Criteria API is used to retrieve the records from the data base server the advantage of this API is we are going to represent the queries in the form of objects.

We can not use Criteria API to insert the record, update the record and delete the record.

The following is an example of using Criteria API to retrieve the records of product table.

**Ex:**

```
//standard hibernate code  
Session hsession = sf.open Session();  
Criteria c = hsession.Create Criteria(Product.class);  
ArrayList list = (ArrayList)c.list();  
Iterator i = list.iterator();  
while(i.hasNext()){  
    Product p = (Product);  
    next();  
    System.out.println(p.getPId());  
    System.out.println(p.getName());  
    System.out.println(p.getPrice());  
}
```

To work with Criteria API we must follow the following two steps:

1. Represent the Query in the form of Criteria object.
2. Send the Criteria to data base server by using a method list();



Hibernate guys has given predefined classes to add the restrictions. They are available as part of org.hibernate.criteria package some of the important classes in that are restrictions,order, property etc.

The following is code to add restrictions to criteria.

```
Criteria c = hsession.create Criteria(Product.class);  
c.add(Restrictions.ge("price",2000d);  
c.add(Restrictions.le("price",4000d);
```

The Restrictions class contains couple of static factory methods. The internal code of these methods add() the where conditions to the Query as a developer we are responsible to use these methods based on the requirements.

To sort the records based on Criteria API they have provided predefined classes like order. This contains the methods like asc, desc.

**Ex:**

```
Criteria c = hsession.Create Criteria(product.class);  
c.addOrder(order.desc("price"));
```

To retrieve the specific columns from the data base table we use projections class.

**Ex:**

```
Criteria c = hsession.Create Criteria(product.class);  
c.setProjection(Projections.property("name"));
```

When we run the above java program the Array list object contains the columns corresponding data type.

#### **adding multiple projections:**

```
Criteria c = hsession.Create Criteria(Product.class);  
ProjectionList pl = projections.ProjectionList();  
Pl.add(Projections.property("name"));  
Pl.add(Projections.property("pid"));  
c.setProjection(pl);
```

**Note:**

Criteria API is not best sui table for the real time projects and to develop complicate Queries for example using the functions like di code, nul1, nul2 etc.



### **Native SQL API:**

The main advantage of Native SQL is we can write data base specific Queries as part of hibernate. By using Native SQL API we can perform the operations like insert Query, update, delete, retrieve and call the procedures and etc.

The following is an example of calling the procedure by using Native SQL.

#### **Ex:**

```
Transaction tx = hsession.beginTransaction();
SQL Query Query = hsession.creteSQLQuery("{call myproc}");
Query.executeUpdate();
tx.commit();
```

### **How do we use hibernate in web based applications:**

#### **How do we use hibernate in sturts:**

1. Create a web based application.
2. Copy all the hibernate related jar files into project lib folder.
3. Copy hibernate configuration file, POJO classes, hbm files and hibernate configuration files into classes folder.
4. Create hbm file and servlet to capture data and store data.

### **Generators:**

In all the applications which we developed as of new end user is suppose to enter the primary key values.

It is not recommended to ask the end user to enter to enter the primary key values. It's always recommended to generate primary key values by the application. This resolves the problem of end user remembering the primary keys.

By using JDBC we have to write code to generate primary key values. If we use hibernate internally it contains a logic to generate the primary key values.

### **Approach1:**

In this approach we find the maximum values from corresponding table and increment by one and use it as primary key value to find the next primary key value we can use the following query.

#### **Ex:**

```
Select max(eno) + 1 from emp;
```



This logic can be used in any data base server.

### **Approach2:**

Oracle data base server supports a feature sequences to generate the primary key values to create the sequence we use the following query.

#### **Ex:**

```
create sequence empseq  
min value 1  
max value 99999  
increment by 1  
start with 1;
```

To see all the available sequences in the data base server. We use the following query.

```
SQL> select * from user_sequences;
```

### **Approach3:**

In mysql there is a feature auto increment. If we use it mysql itself will increment the primary key values.

To use auto increment at the time of table is created we must specify auto increment.

#### **Ex:**

```
create table product(pid int(5) primary key auto_increment,  
name varchar(20),  
price decimal(10));
```

To insert the values we use the following query.

**Ex:** insert into product(name,price) values('abc',445);

auto increment will write in mysql and DB2. It does not work in oracle.

Hibernate has given set of predefined classes to deal with primary key values. These classes are available in a package. "org.hibernate.id"

The following are some of the classes of hibernate generators.

1. Assigned
2. Sequence generator
3. Increment generator
4. uuid generator
5. guid generator etc.

Hibernate guys has given an interface identifier generator this contain a method generator.

Identifier generator
Serializable generate(.....)

All the above generator classes must provide implementation to an interface identifier generator. As part of the generate method the logic is provided to generate the primary key value. This method returns primary key value in the form of serializable object.

As part of the hbm file there is a tag generator. This specify the generator which has to be used by hibernate.

**EX:**

```
<generator class = "org.hibernate.id .Sequence Generator"/>
```

Hibernate guys has provided short names for every generator. We can use the short names on be half of class names. The following are some of the short names. Assigned, sequence, increment, native, hilo, uuid etc.

The following is an example of using short names.

```
<generator class = "sequence"/>
```

Before we use a generator we have to check the following two steps.

1. Weather this generator can be used in the corresponding data base server or not.
2. We need to check weather the generator supports specific data type or not.

**Assigned generator:**

When we use assigned generator hibernate expect us to supply primary key value by default when we generator hbm and POJO classes generator is assigned with assigned value.

In the hbm file we can use generator with hibernate class name or short name.

```
<id name = "pid" type = "integer">  
  <column name = "pid"/>  
  <generator class = "org.hibernate.id.Assigned"/>
```



```
</id>    // product.hbm.xml
```

OR

```
<id name = "pid"/>  
    <column name = "pid"/>  
    <generator class = "assigned"/>  
</id>
```

In the hbm file if we does not supply the generator tag by default hibernate consider assigned generator.

If we supply a wrong class name or invalid generator class name hibernate throw an exception could not instantiate id generator.

### **Increment generator:**

When we use this generator it will increment the primary key value by one based on existing primary key value. This algorithm internally uses the existing primary key value. When we use this algorithm it will use the following query to find the maximum primary key value.

**Ex:** select max(pid) from product

The following is the configuration of increment generator.

```
<generator class = "org.hibernate.id.Increment Generator">  
</generator>  
<generator class = "Increment">  
</generator>
```

Increment generator can generate the primary key values for short, long, integer data types only.

We can use increment generator in mysql data base server also.

### **Identify generator:**

This generator can be used only in mysql data base server to use this generator compulsory the table must be auto increment. The following is the configuration of identify generator.

```
<generator class = "org.hibernate.id.dentity generator"/>
```

OR



```
<generator class = "identity"/>
```

**(sequencehilo):**

**Seq hilo:** This generator uses both sequence as well as hilo value to generate a primary key value. This is same as hilo generator. This uses the sequence instead of table. The following is the configuration for sequence hilo generator.

```
<generator class = "seqhilo">  
    <param name = "sequence">pidseq</param>  
    <param name = "mar_lo">5</param>  
</generator>
```

**Uuid/guid:** These generators generate the primary key value based on the IP address of the system and the start up time of JVM and convert it into 32 bit hexadecimal number and store in data base server.

To work with guid/uuid generators the primary key value data type must be var char with minimum size of 32 bit. The following is the configuration of generator.

```
<generator class = "uuid"/>
```

Instead of predefined generator we can use our own generator also to get the primary key values.

To develop our own generator class it must provide implementation to interface identifier generator. In this we need to provide the implementation to generator method.

**Ex:**

```
public class OurOwnGenerator implements Identifier Generator{  
    public serializable generator(SessionImplementor session, object object){  
        int eno = 0;  
        ResultSet rs = session.getbatcher().PrepareBatchStatement("select max(eno)  
from emp").executeQuery();  
        if(rs.next()){  
            eno = rs.getInt(1);  
            eno = eno + 1;  
        }  
        return eno;  
    }  
}
```

To use this generator in hibernate we have to configure it in hbm file as show below.



**Ex:** <generator class = "info.inetsolv.ourown Generator"/>

When we use hibernate in form based applications we have provided the configure method and build session factory method as part of service method.

### **Single ton design pattern:**

The main purpose of single ton design pattern is it make sure that any work execute only once or it make sure that the object is created to a class only once.

The following Cone class makes sure that it creates only one object for multiple people.

```
public class Cone{
private static Cone c;
Static{
System.out.println("creating Cone object only once");
c = new Cone();
}
public static Cone create Cone object(){
return c;
}
Private Cone(){
System.out.println("Cone object is created");
}
}
```

To get the Cone class object we use a method created Cone object

```
public class MyApp(){
public static void main(String args[]){
Cone c1 = Cone.create Cone object();
Cone c2 = Cone.create Cone object();
System.out.println(c1);
System.out.println(c2);
}
}
```

The following is hibernate single ton design pattern. This class make sure that session factory is created only once.

```
public class Hibernate SessionFactorySingleTon{
```



```

private Static SessionFactory sf = null;
    Static{
        Configuration cfg = new Configuration();
        cfg.configure();
sf = cfg.buildSessionFactory();
    }
public Static Session Factory getSessionFactory(){
    return sg;
}
private HibernateSessionFactorySingleTon(){
    }}

```

As part of a servelet we get the session object directly from single ton design pattern.

**Ex:**    public class StoreProductServlet extends HttpServlet{  
           public void service(. . . . . ){  
           Session hsession = HibernateSession factory single ton . getSession();  
           . . . . .  
           . . . . .  
           . . . .  
           hsession.close();  
           }  
         }

The following is an example hibernate temple design pattern. By using this any body can perform the some operation by using one line.

**Ex:**

```

public static void save(object o){
    Session hsession = Hibernate session factory.getSession();
    hsession.beginTransaction();
    hsession.Save(o);
    hsession.getTransaction().commit();
    Hibernate session factory.close Session();
    }
}

```

Develop a servlet to get a all the records from product table and display to the client.

### **Named Queries:**

Named Queries will improve the performance of java application.





When we develop a web based application to retrieve the records from product table and display to the client by using HQL Queries every time it try to convert HQL Query into corresponding SQL Query. We can improve the performance of by using named Queries.

**Procedure to use Named Queries:**

1. Configure Named Queries as part of hbm files.

**Ex:** <hibernate\_mapping>

<class--- >

----- >

</class>

<query name = "gpd">from info.inetsolv.product</query>

</hibernate\_mapping> //product.hbm.xml

2. From the java application to use the Named Queries use a method.getNamedQuery()

Query query = hsession.getNamedQuery("gpd");

ArrayList al = (ArrayList)query.list();

Iterator i = list.iterator();

When we are using Named Queries we can supply named positional parameters also.

**Ex:** <query name = "gpd">from info.inetsolv.product where name =: name</query>

Named native SQL we can configure native SQL also as part of hbm files. To configure native sql we use a tag <sql\_query>

**Ex:**

<hibernate\_mapping>

<class----- >

-----

</class>

<sql\_query>name = "hsq" select \* from product </sql\_query>

</hibernate\_mapping>

To call named sql from java application we use a getNamed Query this returns query object. When this send to data base server we getArrayList object with objectArray as the values.

**Ex:** Query query = hsession.getNamedQuery("hsq");



```

ArrayList al = (ArrayList)query.list();
Iterator i = al.iterator();
    while(i.hasNext()){
        object o[] = (object[])i.next();
        for(int j=0;j<=o.length;j++){
            System.out.println(o[j]);
        }
    }
}

```

Instead of object Array if we want to return a POJO class object we use return tag.in.hbm files as shown below.

```

<sql-query name = "hsql">
    <return class = "info.inetslov.product"/> select * from product </sql-query>

```

As an alternative to connection method hibernate guys has provided do work method.hibernate guys has given interface work. In this there is a method execute.

We provide any JDBC code has part of execute method.

```

public class JDBCWork implements works{
    public void execute(connection con){
        CallableStatemet cstmt = con.PrepareCall("{call myproc}");
        cstmt.Execute();
    }
}

```

When we write the business logic in the procedure they will improve the performance. The disadvantage of this approach. If we change the data base server the application will not work as a java programs we recommend to write the business logic in java application.

Hibernate is not the best solution when we write the business logic in procedure. It's not recommended to use procedures in the projects which uses hibernate. We are seen three approaches of calling procedures from hibernate.

1. By getting the connection from session object.
2. By using native SQL application.
3. By using do work method.

When we use do work method hibernate internally uses callback mechanism to call execute method the advantage of this is we are not opening the connection we are not



closing the connection. A procedure can write multiple records to achieve this we have to use cursors in procedure.

Cursors are a temporary memory area. Where the records will be stored the cursor object will be created by oracle. When ever select query is executed in oracle they are two types of cursors available. They are:

1. Implicit cursors
2. Explicit cursors

The following procedure returns a cursor/ResultSet object create or replace procedure MyProc( rs out sys\_refcursor)

As

Begin

open rs for select \* from product;

end myproc;

/

Following java application to call the procedure and display to the client.

```
public class GetRecords{
    public static void main(String args[]){
        //standard jdbc code
        CallableStatement cstmt = con.PrepareCall("{call myproc(?)}");
        cstmt.Execute();
        ResultSet rs = (ResultSet) cstmt.getObject(1);
        //code to display the records
    }
}
```

Hibernate guys are giving a way to call procedure which takes a parameter refcursor.

Configure procedure in hbm file as shown below.

```
<hibernate_mapping>
<class ----- >
-----
</class>
<sql_query name = "cp" callable = "true">
<return class = "info.inetsolv.product"></return>
{call myproc(?)}
```



```
</sql_query>
</hibernate_mapping>
```

To call the procedure from java we use named queries as shown below.

```
Query query = hession.getNamedQuery("cp");
ArrayList al = (ArrayList)query.list();
Iterator i = al.iterator();
    while(i.hasNext()){
        product p = (product)i.next();
    }
}
```

We have to follow limitations in hibernate to call the procedure.

1. The first parameter procedure must be out parameter and the data type must be ref cursor.
2. When we are configuring the procedure in we must use on attribute callable st.

When we specify callable true hibernate register. The first parameter as out parameter we must use native sql to call the procedure from hibernate application.

If you want to search the records based on given product id you have to supply a product id as parameter to procedure. We can call that the configure in hbm file as shown below.

We can develop a hibernate software to communicate with multiple data base servers from a single project. Procedure to use hibernate with multiple data base servers for the same project. Create multiple hibernate configuration files.

**Ex:**

```
hibernate.cfg.xml
hibernate.mysql.cfg.xml
```

To use multiple data base servers in the project we use multiple configuration files and multiple session factory objects. In the projects we do this work only once by using a single ton design pattern only.

```
public class HibernateSessionFactory{
    private static SessionFactory osf = null;
    private static SessionFactory msf = null;
    static{
```



```

Configuration ocfg = new Configuration();
ocfg.Configure();
Configuration mcfg = new Configuration();
mcfg.Configure("hibernate.mysql.cfg.xml");
ocfg.buildSessionFactory();
mcfg.buildSessionFactory();
    }

public static SessionFactory get mysqlSessionFactory{
return ocfg; }

public static SessionFactory get mysqlSessionFactory{
return mcfg; }

private HibernateSessionFactory(){}
```

In the java application we get the required Session Factory object and perform the operations.

**Ex:** Sun Micro System has released JPA API to interact with data base servers they are released this API as part of JEE standard API.

The following is the classes and interfaces as part of JPA.

#### **Classes**

#### **Interfaces**

Persistence

Hibernate guys provided the implementation to JPA API.

Create a java project and add JPA capabilities. There are so many people who are providing the implementation same of them or top link, hibernate and open JPA and etc. we can use any jar files to develop JPA application. When we add JPA capabilities it has created an xml file whose name is persistence.xml in this we have specified persistence unit name.

Generate an Entity Beans (POJO classes). Develop a java application to store the data into employee table.

```

public class StoreEmp{
public static void main(String args[]){
EntityManagerFactory emf = Persistence.CreateEntityManagerFactory("JPAORACLEP");
EntityManager em = emf.CreateEntityManager();
EntityTransaction tx = em.getTransaction();
tx.begin();
Emp e = new Emp();
```



```

e.setEno(1);
e.setEno(2);
e.setEno(3);

em.Persist(e);
tx.commit();
em.close();
    }
}

```

The top link provides the implementation of JPA API to work with top link we are to download the top link jar files and set the class path. When we are developing a project at the design phase we try to develop two types of design.

1. Data base design
2. Java design

As part of data base design we will decide the names of the tables and columns of the tables and etc. At the time of developing the project developer will not create any tables. In most of the projects we have to capture multiple address to an employee.

Eno

Name

Address:

Street	city	state
Am pet	HYD	AP
SR Nagar	HYD	AP

ADD another address

To implement above requirement we got the following design.

**Design 1:**

Eno	Name	Street	State	City
1	Naidu	Am pet	AP	HYD



As part of design we are adding two tables. Emp table and address table in the Emp table Eno is a primary key. In the address table Eno, Address no are forming the primary key.

**Design 2:**

Eno(PK)	Name	salary
1	Naidu	4000

Eno	Addno	Street	State	City
1	1	Am pet	AP	HYD

In the design2 the address table contain Eno, addressno as primary keys. If more than one column is valued in forming a primary key we call it as composite primary key. Hibernate will not give the best performance if the table contains composite primary key. Hibernate give the best performance if it's having only one primary key.

**Design 3:** In this design we have two tables. They are emp and address.

In the address table we will try to have a new column which acts as primary key.

Eno(PK)	Name	salary
1	Naidu	100000
2	Malli	200000

Aid	Eno(PK)	addno	Street	State	City
1	1	1	APet	HYD	AP
2	2	1	APet	HYD	AP
3	2	2	APet	HYD	AP

Generally by using hibernate we should be able to create the tables. but according to the project design first we are creating tables from the tables we are creating POJO classes. Creating the POJO classes from the table is called as hibernate reverse engineering.

There are some situations where we need to develop hibernate applications based on legacy data base servers.



**\*\*What are legacy data base servers?**

The data base server which uses composite primary key's is called as legacy data base servers. Procedure to develop hibernate applications based on legacy data base. Create the following the two tables.

**Table 1:** create table emp(eno number(5) primary key,

Name varchar2(20), salary number(10,2);

**Table 2:** create table address(eno number(5), addrno number(5),

State varchar2(20), primary key(eno, addrno));

When we generate hbm files and POJO classes we have observed that 3 POJO classes and two hbm files the POJO classes are:

Emp.java

AddressId.java

Address.java

We have not observed any change in emp POJO class. This is because this table is having only one primary key. We have observed that address table is created. Two POJO classes:

1. Composite primary key class(AddressId)
2. Regular POJO classes

The composite primary key class contain two properties this is because two columns involving in forming composite primary key.

```
public class AddressId{  
  
    Integer Eno;  
  
    Integer addrno;  
  
    //setters and getters  
  
}
```





The following is address.java which uses composite primary key class as a property.

```
public class Address{  
    AddressId id;  
    String street;  
    String city;  
    String state;  
    //provide setters and getters }  
}
```

We are generated two hbm files. They are:

1. Emp.hbm.xml
2. Address.hbm.xml

We have not observed any changes in Emp.hbm.xml files. We have observed lot of changes in address.hbm.xml file with respect to composite primary key.

<hibernate\_mapping>

<class name = "info.inetsolv.Address" table = "Address">

<composite-id name = "id" class = "info.inetsolv.AddressId">

<key-property name "Eno">

<column name = "Eno"/>

</key-property>

<key-property name = "addno">

<column name = "Addno"/>

</key-property></composite-id>

<property name = "street"/>



```
<property name = "city"/><property name = "state"/>  
</class>
```

```
<hibernate_mapping>
```

The following java application to store the data into data base server creating the emp obj.

1. AddressId obj
2. Address obj
3. Call save method for emp.address

To establish the relationship between tables we do it based on the data available in the tables. They are:

1. One to many
2. Many to one
3. One to one
4. Many to many

To establish the relationship between tables we do it based on the data available in the tables.

### **One to many relationship:**

To identify the relationship between the two tables we use the data based on the data available in the tables develop a hibernate application which uses one-to-many relationship b/w emp and address table. The following of the two tables are created by emp and address.

```
create table emp(Eno number(5), Name varchar2(20), salary number(10,2));
```

```
create table emp add primary key(Eno);
```

```
create table address(Aid number(5), Eno number(5), Addno number(5), city varchar2(10));
```

```
alter table address add primary key(Aid);
```

```
create table address(Aid number(5) primary key, Eno number(5) references emp(Eno)
```



when we generate hbm files and POJO classes we got two hbm files and POJO classes. The following is the POJO class from emp table.

```
public class emp{
```

```
Integer Emp;
```

```
String name;
```

```
Double salary;
```

```
Sent address = new HashSet(0);
```

```
//provide setting and getters
```

```
}
```

```
alter table address add
```

```
foreign key(Eno) references
```

```
emp(Eno);
```

The following is the configuration of the Emp.hbm.xml represents the many relationship.

```
<hibernate_mapping>
```

```
<class name = "info.inetsolv.Emp">
```

```
<id name = "Eno"/>
```

```
<property name = "name"/>
```

```
<property name = "salary"/>
```

```
<set name = "address" inverse = "true">
```

```
<key>
```

```
<column name = "ENO"/>
```

```
</key>
```

```
<one-to-many class = "info.inetsolv.Address"/>
```

```
</set>
```

```
</class>
```



<hibernate\_mapping>

The following is the POJO class for Address table.

```
public class Address{

    Integer aid;

    Emp emp;

    Integer addno;

    String street;

    String city;

    String state;

    // provide settress and gettress

}
```

The following hbm file for address table.

```
<hibernate_mapping>

<class name = "info.inetsolv.Address">

    <id name = "aid"/>

    <many-to-one name = "EMP" class = "info.inetsolv.Emp">

        <column name = "ENO"/>

    </many-to-one>

    <property name = "addrno"/>

    <property name = "street"/>

    <property name = "city"/>

    <property name = "state"/>
```



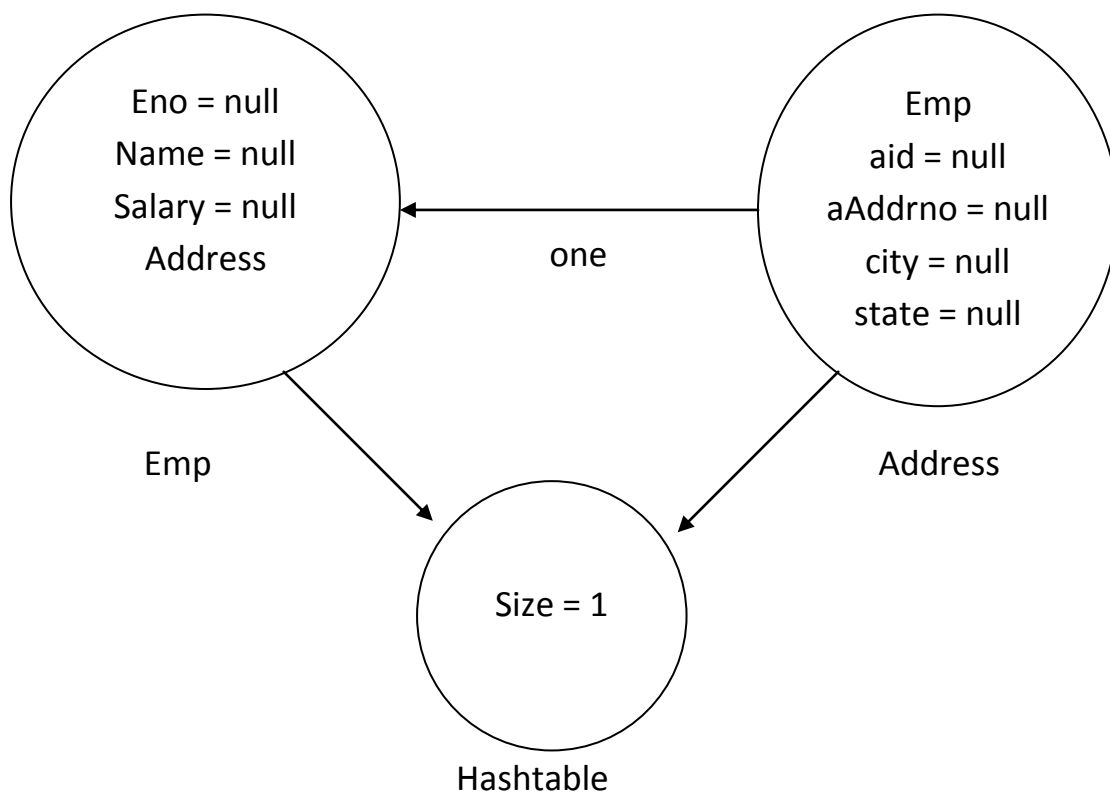
</class>

</hibernate\_mapping>

Generally IDE recognize the relationship between tables and generate hbm files and POJO classes. Hibernate understand the relationship between the tables based on the tags which are available in hbm file. To represent many relationship in hibernate we use collection objects. The following are collection objects of hibernate.

List Set Map Bag

These collection objects represents many relationships we use hibernate from the java side we have to create the objects store the data into objects and establish circular relationship.



The following java application to store the data into emp as well as address table.

```
public class store{  
public static void main(String str[]){
```

```

Session h = HibernateSessionFactory.getSession();
Transaction tx = h.beginTransaction();
Emp e = new Emp();
e.SetEno(1);
e.SetName("Naidu");
e.SetSalary(10,000);
Address a = new Address();
a.SetAid(1);
a.SetEmp(e);
a.SetAddno(1);
a.SetCity("HYD");
a.SetState("AP");
Set s = e.getAddress();
s.add(a);
h.Save(e);
h.getTransaction().Commit();
HibernateSessionFactory.CloseSession();

```

When we call save(e) hibernate can find the relationship between the tables and store the data into dependent on tables to do this we must specify two attributes inverse and cascade code.

```
<Set ----- inverse = "true" cascade = "all"/>
```

Develop a java application to retrieve the data from emp table who emp no is 2 (we would like to retrieve the corresponding address details). Hibernate checks two strategies to retrieve the record from data base server.

1. Lazy loading -----> Aggressive (eager) loading.

**\*\*What is lazy loading?**

When ever we perform an operation on one table and table has relationship with other tables and then we called a load method it will retrieve the data from that table only.

Hibernate retrieve the records from child table when even any operation is carried out of any other table. By default hibernate uses lazy loading(true).

**\*\*What is Aggressive loading?**



When ever we call a load method on a parent table it will try to retrieve record form child table also even though user doesn't access any properties. How to make hibernate as aggressive we have to configure lazy = "false" in hbm file of parent table.

```
<set ----- inverse = "true" lazy = "false">  
</set>
```

The following is the example of retrieve the data from data base server by using the load() method which is loading the relationship.

```
public class Retrieve{  
    public static void main(String args[]){  
        Session s = HibernateSessionFactory.getSession();  
        Emp e = new Emp();  
        e.getEno();  
        e.getName();  
        e.getSalary();  
        Set address = e.getAddress();  
        System.out.println(address.Size());  
        Iterator i = addresses.iterator();  
        while(i.hasNext());  
        Address a = (Address)i.next();  
        System.out.println(a.getAid());  
        System.out.println(a.getAddno());  
        System.out.println(a.getCity());  
    }  
    HibernateSessionFactory.getSession();  
}
```

As part of POJO classes and hbm file instead of set we can use only hibernate type List, map, set, bag.

**Ex:** We are using the list in the emp table is shown below.

```
public class Emp{  
    Integer eno;  
    String name;  
    Double salary;  
    List address = new ArrayList();  
}
```

The following configuration of Emp.hbm.xml



```

<hibernat_mapping>
<class name = "info.inetsolv.emp">
<id name = "Eno"/>
<property name = "name"/>
<property name = "salary"/>
<list name = "address" inverse = "+ve" cascade = "all">
<key column = "Eno"/>
<index>
</index>
<one-to-many class = "info.inetsolv.Address"/>
</list>
</class>
</hibernat_mapping>

```

By default hibernate uses inverse = "false" if inverse = "false" hibernate will not check the bi-directional relationship between tables. In this scenario if we try to send multiple insert queries and update query. In case of inverse = "true" he check the bi-directional relationship and store the data by using insert queries.

Cascade attribute takes multiple values. They are none, all, save-update, delete.

1. Cascade specifies when ever we perform any operation on parent table it perform the operation on child table also. By default in hibernate cascade attribute value is name.
2. When we specify cascade = none when ever we carry out any operation on parent table it does not carry out any operation on child table.
3. When we specify cascade = "save-update" when ever we perform save operation on parent table it will perform save operation on the child table also.
4. Cascade delete when we specify cascade delete when ever we delete the record from parent table we delete the record from child table also. Instead of specifying every operation we specify cascade = "all". We would like to maintain the customer and order details.

Cid(pk) Cname location

Customer table





order(pk) Cid(fk) quan price name
-----------------------------------

Order table

**many-to-one**: We would like to maintain the tables for multiple employees we would like to have departments and there are departments with out employees to achieve we use many-to-one relationship.

Eno(pk) name Did(fk)
----------------------

Emp

Did(pk) name
--------------

Dept

**one-to-one relationship**:

We would like to maintain one-to-one relationship between team and captain tables. The following are two takes which we are using to achieve this.

Tid(pk)	name	location
1	dc	'hyd'

Team

Tid(pk)	(pk)	Cname
1		Naidu

Captain

**Note**: When we create the above two tables and establish the relationships and generate hbm files and POJO classes IDE as un-necessarily created one-to-many relationship this is because of adding a foreign key relationship between captain and team table.

1. Create two tables team and captain with only primary key do not add the foreign in the beginning this is because if we had a foreign key IDE will generate one-to-many relationship.



2. Generate hbm files and POJO classes.
3. We have to add one-to-one relationship between team and captain by using properties

```
<hibernate-mapping>
    <class . . . . >
        . . . . .
        <one-to-one name = "captain" class = "info.inetsolv.captain" cascade = "all">
        </one-to-one>
    </class>
</hibernate-mapping>    //team.hbm.xml
```

Similar to team POJO class and hbm file, develop captain java and captain.hbm.xml

\*Develop the hibernate application to store data in to data base server.

```
public class StoreTeamDetails{
public static void main(String[] args){
Session hsession = HibernateSessionFactory.getSession();
Transaction tx = hsession.beginTransaction();
Team t = new Team();
t.setTid(tid);
t.setName(teamName);
t.setLocation(location);
Captain c = new Captain();
c.setTid(tid);
c.setName(captain);
c.setTeam(t);
t.setCaption(c);
hsession.save(c);
```



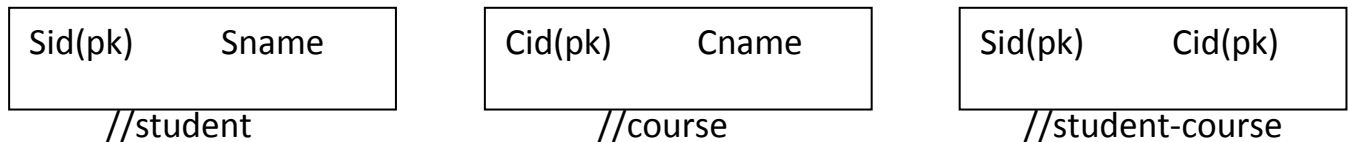
```

tx.commit();
HibernateSessionFactory.closeSession();
    }
}

```

### **many-to-many:**

We would like to implement many-to-many relationship b/w student and course tables.



To specify the many-to-many relationship b/w the tables we have to provide the following configuration in hbm files.

```

<set name = "StudentCourses" table = "Student_Course" CasCode = "all" inverse = "true">
    <key column = "Sid"/>
    <many-to-many column = "cid" class = "info.inetsolv.course"/>
</set>      //student.hbm.xml

```

\*\*\*The following is the java code to store data in the data base server.

```

public class StoreData{
public static void main(String args[]){
Session hession = HibernateSessionFactory.getSession();
Transaction tx = hsession.beginTransaction();
Course c = new Course();
c.setCid(1);
c.setCname("java");
Student s1 = new Student();
s1.setSid(1);
s1.setSname("raju");
Student s2 = new Student();
s2.setSid(2);
s2.setSname("naveen");
setStudents = c.getStudentCourse();
Students.add(s1);
Students.add(s2);
}
}

```



```
c.setStudentCourses(students);  
hsession.save(c);  
tx.commit();  
HibernateSessionFactory.closeSession();  
    }  
}
```