HIBERNATE

NOTES

By
Mr.NATRAJ

Sathya Technologies
2nd Floor, Sri Sai Arcade,
Beside: Aditya Trade Centre,
Ameerpet, Hyderabad - 500038
Tel: 65538058

HIBERNATE

15/7/10

The logic that is given to prepare UserInterface and to display results by applying styles is called presentation logic.

→ The main logic of the application that generates results based on given input values is called Business logic.

→ the permanent storage unit that can store data is called persistance store.

Ex:- Files, DataBase s/w etc.

→ Insert, update, select and delete operations performed on the DB table — persistance store.

to maintain data are called persistance operation.

The logic used for this purpose is called as persistance logic.

→ persistance operations are also called as CURD (or) SCUD operations

CRUD (or) SCUD operations

C → create (insert)
U → update (modify)
R → Read (select)
D → Delete (remove)

S → select
C → create
U → update
D → Delete

the technologies ie given to develop persistance logic is called persistance layer technology.

Ex:- JDBC, Hibernate etc.

→ JDBC, Hibernate code comes as persistance logic.

→ persistance logic is always helper logic for Business logic.

Example applications :-

Test App

→ read m1, m2, m3 values from end user (presentation logic)

→ total = m1 + m2 + m3 ; (Business logic)

→ store student details to DB table (peristance logic)

→ display results (presentation logic)

(ServletAPI) (JDBC) (Servlet & JSP)

* struts — hibernate — spring

* RMI — EJB (procedural code & JDBC)

* XML — webservices (Servlets & JSP)

→ Files are suitable as peristance stores in small scale applications like, desktop & mobile games etc).

→ DB softwares are suitable as peristance stores in medium & large scale application like websites, banking applications etc.

→ DB softwares that is capable of storing objects as table column values is called object DB softwares.

Java app —I/O streams→ Files

serialization / deserialization

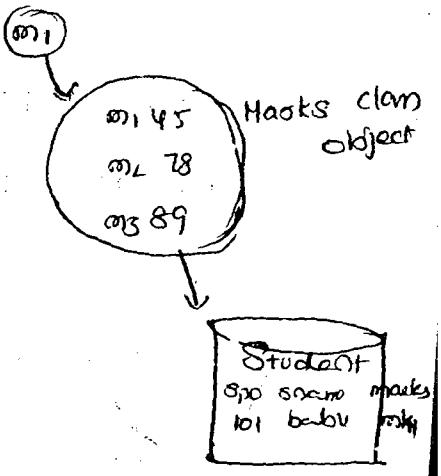
Java app —JDBC code & ORmapping→ DB s/w [oracle, sybase]

Java app —Java code + OQL→ ODB s/w (ex:- poet)

object query language

example .....

```
class marks
{
  int m1, m2, m3;
  =
}
Marks m1=new Marks(45,78,89);
```



Limitations with ODB s/w :-

① storing multiple values in a single column of db table column is against of Normalization Rule. no:-①

② Generating reports from table of ODB s/w is complex process.

Limitations with files as persistance stores :-

① No Security

② Can't store huge amount of data

③ No query Language support

Ex:- manipulation & ② are very complex.

→ Merging & comparision of data is complex.

→ To solve problems with files & ODB s/w use RDBMS D/B s/w like oracle, sybase, mySQL & etc. as persistance logic stores.

→ Java application can use JDBC code as persistance logic to interact with RDBMS database s/w's, But following ~~are~~ limitations are these.

① SQL Queries are D/B s/w dependent ~~...~~ since JDBC peristance logic use sql queries.

The JDBC code is DB dependent peristance logic due to this changing DB s/w in the middle of production environment (after release) and in the middle of development environment is complex proces

② : JDBC ResultSet object is not a serializable object so we cannot send object directly over the network.

③ Exception handling, transaction management, facilities in JDBC coding are not that much sufficient in large scale application development.

④ To solve all these problems use ORMapping based peristance logic to interact with RDBMS DB s/w s.

=

16/7/10

ORMapping :-

the process of mapping Java class with DB table, member variables with table columns & making that java class objects representing DB table rows by having synchronization b/w them is called OR-Mapping.

→ synchronization b/w object & table row is nothing but modification done in java object will reflect in table row & vice versa.

→ ORM softwares are responsible for this synchronization and to develop objects based ORMapping peristance-logic.

on table rows (CURD operations) will take place java objects that are representing table rows, and there is no need of SQL queries. This makes OR-Mapping persistance logic as DB independent persistance logic.

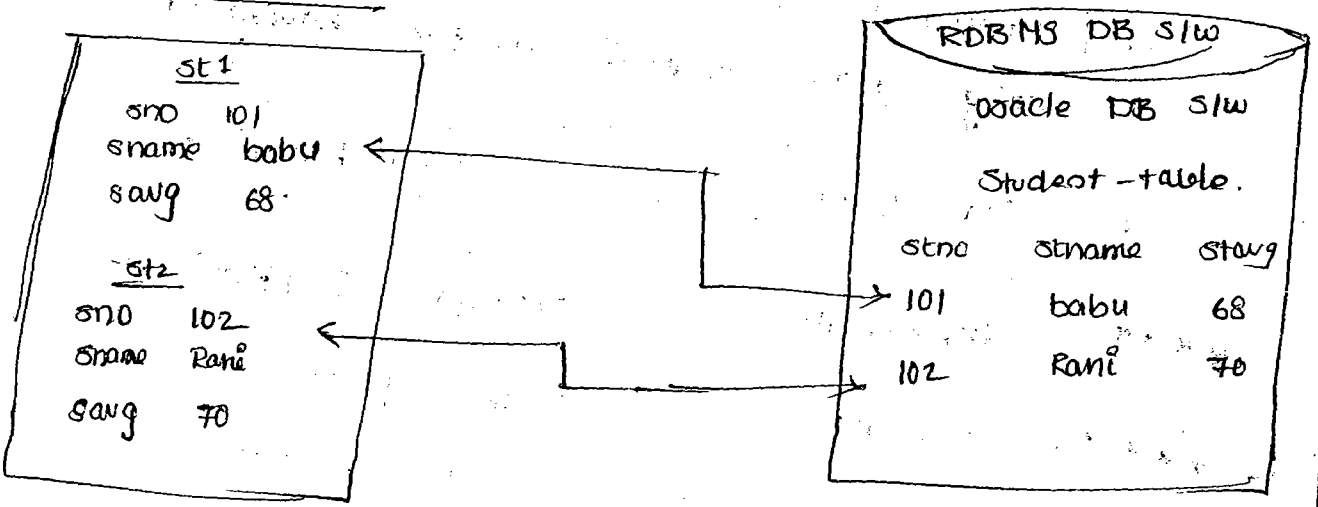
Due to this, we can change DB software in the middle of project development (or) utilization without disturbing the persistance logic.

```
class Student
{
  int sno;
  String sname;
  float avg;
  =
}
```

OR Mapping work

member variables { sno → stno, sname → stname, avg → stavg } } db table column

Java Application.



st1, st2 are student class objects representing → db table, rows.

the special s/w that provides abstraction layer on same category code (or) basic technology is called framework software.

→ All ORM s/w-s internally use JDBC code to perform persistance operation on DB table rows that are manipulating objects.

(hiding implementation) layer to programmed on JDBC code. so ORM s/w's are also called as Framework softwares.

→ Struts is a web frame work s/w providing Abstraction layer on Servlets, JSP Technologies.

→ spring is a Java - J2EE Framework s/w providing Abstraction layer on Java-J2EE technologies like JDBC, JNDI, JMS, EJB, Servlet-JSP & etc.

→ Hibernate is a O-R mapping based framework s/w providing Abstraction layer on JDBC.

→ Framework s/w's generate common logics of the application development dynamically by using code-technologies so, Framework s/w's improve productivity (doing more work in less time having accuracy).

All O-R Mapping s/w's (Java based) :-

1) ✓ Hibernate → softtree (Red Hat)

2) ✓ Toplink → oracle corporation

3) EJB entity Beans → Sun Micro Systems (oracle corporation)

4) ✓ JPA → sun microsystem (given from JEE5)
(Java Persistance API)

5) OJB → Apache Foundation.
[object Java Beans]

6) JDO → Adobe.
(Java Data Objects)

7) Ibatis → Apache Foundation.

1000 objects will be created in java application how can you justify this heavy weightness on Java application while working with O-R Mapping peristance logic ?

Ans:- while working with O-R mapping peristance logic if 1000 records are there in the table and they are not selected at a time, then 1,000 objects will not be created in java application If all 1000 records selected at a time then, 1000 objects are created in java application.

So, It is recommended to programmer to select less amount of records ~~page by page~~ and to use pagination (displaying huge amount of records page by page) to display the records,

Here based on programmers logic and the no" of records that is selecting at a time. the no of objects created in java application will be decided. so we must solve above problem by doing proper programming and pagination.

→ OR Mapping Fw's are given only to develop peristance logic & they are not suitable to develop other logics like presentation, business logic and etc.

29/-

→ type :- ORM s/w (oo) ORM tool (oo) orm based framework.

→ version :- sub tree (Red Hat)

→ creator :- Mr. Gavin king & team.

→ Version :- 3.5 (latest → compatable with J2sdk1.5 and 3.2/3.3 (regularly used version)
compatable with J2sdk1.5)

→ To download s/w :- Download s/w as zip file from www.hibernate.org (oo) www.sourceforge.net website.

→ online tutorial :- www.roseindia.net

→ good online articles :- www.onjava.net,
www.devzdev.net,
www.precise java.com,
www.javabear.net

→ Technical FAQ's :- www.forum.hibernate.org

→ For Interview FQQ's :- www.geekinterview.org

→ Rebarance books :- poo' hibernate → apress (publisher name)
:- Hibernate in Action → manning series
- (publisher name)

This API provides (Application programme Interface) base for the programmers to develop that technology based s/w application in java environment. API is nothing but set of classes and interfaces which come in the form of a java package.

→ To develop hibernate persistance logic, programmer uses hibernate API & some helper resources like Java classes and xml files.

Hibernate – def :- Hibernate is an opensource, lightweight java based ORM software as framework software to develop ORmapping style DB independent persistance logic in all kinds of java applications like stand alone, 2 tier, webapplication and distributed enterprise applications.

→ the application that deals with complex & heavy weight business logic and contains additional services like security, transaction management and etc is called an enterprise application.

Ex:- banking application, credit / debit card processing application etc.

since Hibernate s/w is free s/w and its source code will be supplied to programmers. it called open source s/w.

→ EJB entity bean componants are heavy weight componants

Reason:- They look for application servers s/w,

weight s/ws.

→ The Resources of Entity Bean components must be developed by using EJB API support.

→ Learning & applying EJB entity bean component is always complex process.

Hibernate is Lightweight s/w ?—

Reason? —

→ No servers, containers are required to execute hibernate code.

the basic J2SDK s/w is enough for execution.

→ certain resources of Hibernate application can be developed without using Hibernate API

→ Learning & applying Hibernate in projects development is always easy to perform.

→ Hibernate 3.2.5.ga.zip

→ use changelog.txt file of Hibernate Home directory (the directory where Hibernate s/w is installed).

for knowing difference b/w various versions of Hibernate.

→ Hibernate-Home\Hibernate3.jar file represents the whole Hibernate API.

-> The ORM persistence logic is DB independent even though the DB s/w is changed in the middle of project development or utilization in this process the DB table design must not be changed to get this effect.

=> MVC architecture is become industry's standard in the development of real world s/w projects.

M - Model -> B.L + P.L -> (Ex:- Account objects)

V - View -> P.L -> [Ex:- Beautician]

C - Controller -> Integration / Connecting logic.
[Ex:- Traffic police]

-> logic that controlles & monitor all the operations mvc architecture based applications is called "integration logic". This logic contains code to perform the communication b/w View layer resources and model layer resources.

JSP -> servlet -> javabean -> DB s/w.
(View) Controller (model)

JSP -> servlet -> javabean with DAO class -> DB s/w.
(View) (Controller) business logic contains purely persistance logic.
model

-> the java class that contains purely P.L and separate that P.L from other logics of the application is called 'DAO' class.

DAO class ...

that means any modification done in P.L doesn't effect other logic of the application.

→ JSP → Servlet → EJB session Bean Component → EJB entity bean → DB

(View) (Controller) (good for business logic) (persistance logic)

model

=

→ Struts Appln → EJB session bean comp → EJB Entity Bean Component → DB SW

(View & controller) model

=

→ bank ② Struts Appln → EJB SessionBean comp with HB → DB SW

(View & Controller) B.L persistance logic

model

→ bank ① Struts Appln → spring Appln with HB → DB SW

(View & Controller) B.L P.L

model

→ ③ Struts Appln → EJB session with JPA → DB SW

(View & Controller) B.L

=

View layer : JSP, html, Velocity, freemarker, XSLT etc

=> Various technologies to develop "integration logic" at controller layer : "servlet, servlet with filters"

=> DB S/w

=> Technologies to develop B.L at "model" layer EJB session bean, RMI, CORBA (common object request broker architecture) spring and etc.

=> web framework S/w to develop view, controller layer logics.

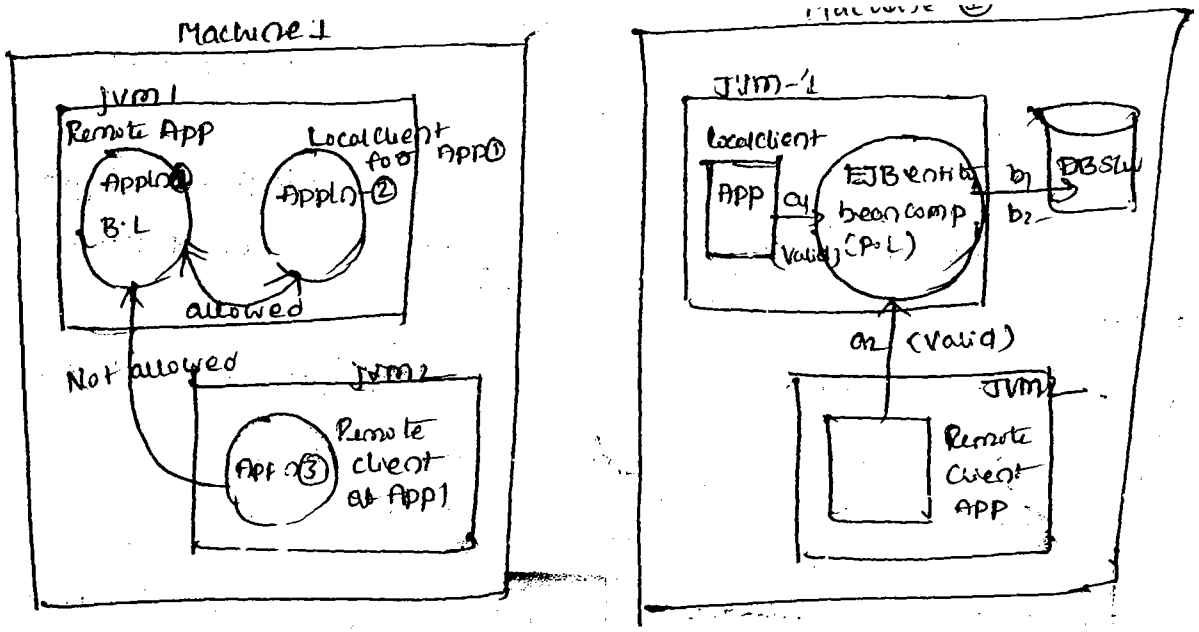
Struts
JSF (alternative for struts from Sunmicro Systems)
Webwork
Xwork, spring mvc and etc

=> Technologies to develop peristence logics of model layer JDBC, HB, toplink, iBatis, EJB entity bean, OJB, JPA and etc.

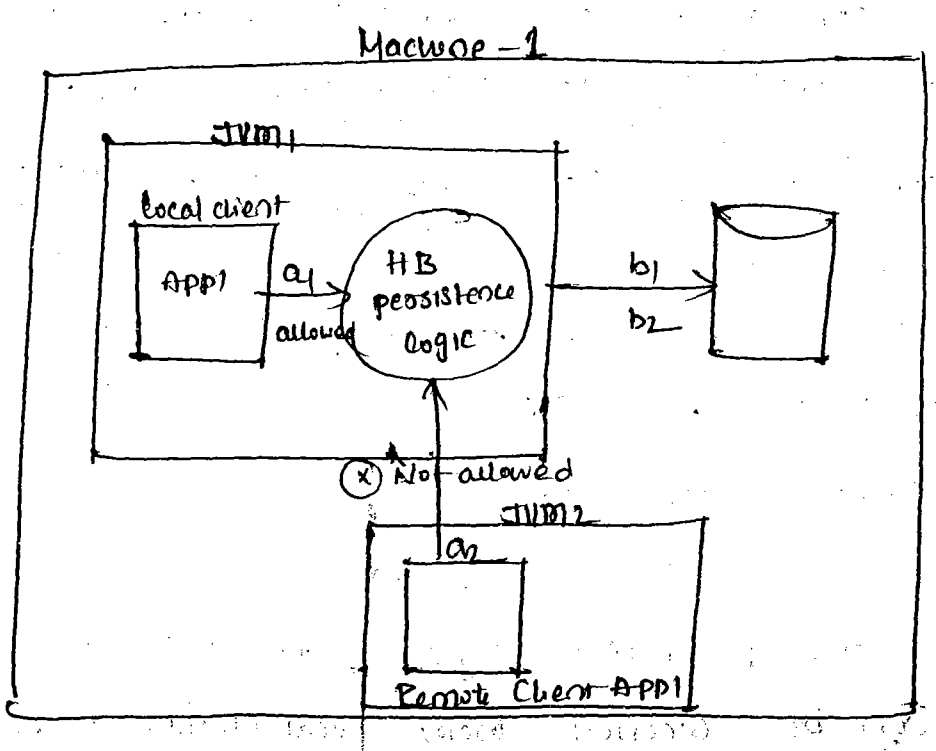
=> If application and its client reside on same JVM then application is called "local application" and that client is called "local client" to application.

-> If application and its client reside and execute on two different JVM's at same machine (or) different machine then that application is called "Remote application" and its client is called "Remote client"

-> distributed application allows both remote and local clients. EJB components are distributed components so the peristence logic of EJB entity bean component can be accessed from local clients (or) Remote client



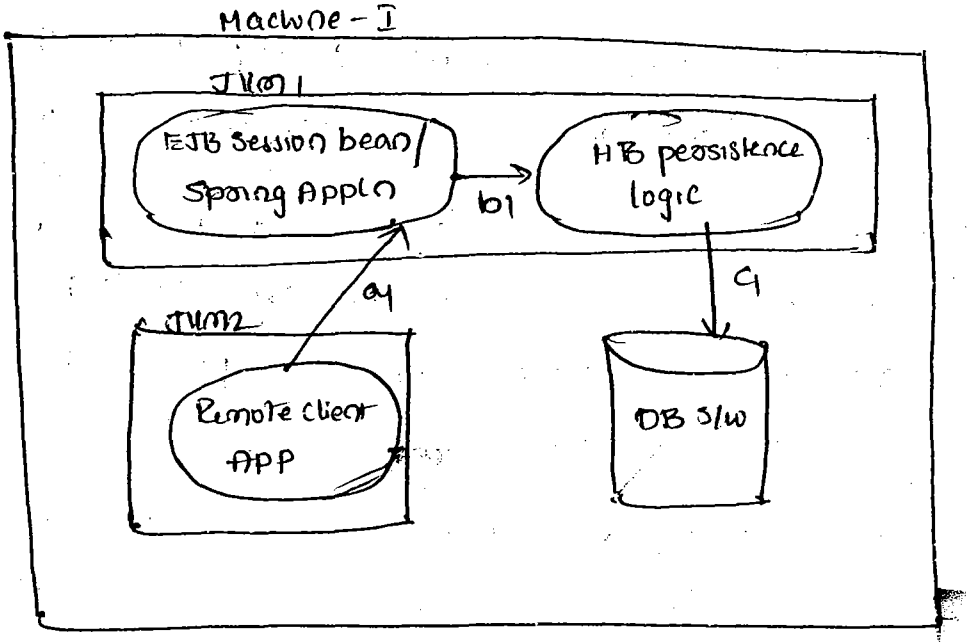
→ HB is not a Distributed technology so that applications cannot access hibernate persistance logic from Remote places. Due to this the application and its HB persistence logic must be reside at execute from the same JVM. This is limitation of HB SW. This indicates HB persistance Logic must be accessed only from local client applns.



multiple java applications are executing parallely and simultaniously.

⇒ we can access hibernate persistence logic from Remote client applications by taking the support of EJB Session bean (or) spring distributed application as mediator application as shown below

(EJB session bean component are distributed component by default)



a1 → b1 → c1 → DB s/w (allowed)

→ since HB is not distributed technology, the HB persistance logic must reside along with application that wants to interact with DB s/w by using this HB P.L.

For example if xyz java appln wants to use HB P.L to interact with DB s/w then xyz appln and its HB persistance logic must be reside and execute from same JVM but the destination DB s/w can be there in remote (or) local computer to xyz appln

Traditional client-server applications,

These are location dependent. That means any change in the location of server application, we need to inform to modify code in client application

→ Distributed applications are client-server applications having location transparency (location independency) that means change of server application location will be detected by client applications dynamically without modifying code.

→ web application can be developed as traditional client-server applications (or) as distributed applications

→ the application that contains complex Large scale business logic and handles multiple middleware services (security, connection pooling &, transaction management) an enterprise application.

⇒ An enterprise application can be: standalone (or) 2-tier (or) distributed (or) web application.

Ex:-
1) Banking application is distributed & enterprise application.
2) online shopping website is webapplication and enterprised & distributed application.
3) Hibernate supports POJO & POJI model programming that means we can take simple & regular java classes and java interfaces as resources while developing hibernate - persistence logic.
4) EJB 3.x, struts 2.x, spring, JSF, technologies also support POJO & POJI model programming.

while developing a java class as resource of certain s/w technology based java application, if that class is not extending from a predefined class of that technology API and if that class is not implementing a predefined interface of that technology API then that class is called POJO class.

Ex:- 1)
```
public class Test
{
==
}
```

Here test is POJO class because it is not extending from any other predefined class.

2)
```
public class Test extends Demo
{
==
}
```

Demo is user defined class, so Test is POJO class

3)
```
public class Test extends Frame
{
==
}
```

Test is not a POJO class it is awt API dependent.

4)
```
public class Test implements ABC
{
==
}
```

ABC is user defined interface so Test is POJO class

{
=
}

Test is a POJO class.

6) public class Test implements java.io.Serializable.

{
=
}

Test is a POJO class because Serializable is not a technology.

It is only basic concept of java.

7) public class Test implements java.rmi.Remote.

Test is RMI api dependent., so Test not POJO class.

8) public class Test extends HttpServlet

{
=
}

Test is Servlet API dependent. so test is not POJO class

9) public class Test

{
public void bmi()
{
=
}
}

Test is pojo class.

while developing java interface as resource of certain technology based java application.

if that interface is not extending from predefined interfaces of that technology API then that interface is called POJI.

Ex:-

1) public interface Demo
{
≡ declaration of methods
}

Demo is POJI

2) public interface Demo extends Xyz, Mno
{
≡
}

since Xyz, Mno interfaces are user defined interfaces so Demo is called POJI

3) public interface Demo extends java.rmi.Remote
{
≡
}

Demo is rmi API dependent so Demo is not POJI.

Features of Hibernate :-

1) supports POJI - POJO model programming.
2) Light weight technology to develop DB s/w, independent persistance logic.
3) Allows to work with any Java, JEE framework s/w s based applications to make them interacting with DB s/w.
4) use built in transaction management, connection pooling support.
5) Allows to work with third party JDBC connection pool s/w s like C3PO, ... etc.
6) supports two levels of caching (cos bubbling) to reduce network round trips b/w client application DataBase s/w.
7) Allows to call PL/SQL procedures & functions
8) Gives HQL (Hibernate Query Language) as DB s/w independent to perform persistance operations.
9) Allows to work with DB specific native SQL to perform persistance operations
10) Allows object level Relationship in development of persistance logic when tables are there in relationship like 1-1, 1-n, n-n etc.
11) Given special data structures like Bag, IdBag etc to support object level relationships.

one record of one child table.

12) Easy to Learn and Easy to apply.

22/7/10

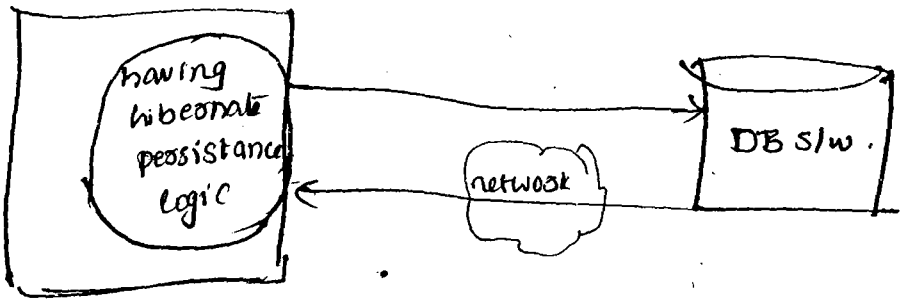
Browser → servlet → EJB Component → DB S/w.

the above combination browser window is client to servlet. Servlet is client to EJB component and EJB component to browser window.

This indicates there are no fixed client & server applications in s/w project. Based on their roles & logics we can call them as client / server applications.

→ when java application uses JDBC code to interact with DB S/w, the java application is not client to JDBC code it is client to DB S/w.

similarly, when java application uses Hibernate persistance logic to interact with DB S/w then the java application is not client to Hibernate S/w. It is client to DB S/w.



persistance logic to interact with DB s/w.

Here the DB s/w can be local (or) remote to Java application but the hibernate persistance logic must be there along with java application

→ different types of java JEE applications need following resources directly & indirectly to develop ORMapping based persistance logic as hibernate persistance logic.

1) JDBC drivers
2) hibernate configuration file (xml file)
3) hibernate persistance class (Generally POJO class based Java bean)
4) Hibernate Mapping file (hbm file) (xml file)
5) hibernate API (available in hibernate3.jar file)

=

Hibernate Architecture.

Hibernate Architecture.

- Java application.
  - AWT / swing app
  - Server / JSP component
  - EJB comp / RMI app
- Web application
  - JSF "
  - Spring "

① - client application activates hibernate s/w and makes that s/w collecting hibernate resources.

② programmer uses hibernate API and hibernate resources to develop objects base OR Mapping style persistance logic.

③ Java (client) application interacts with DB s/w by using this persistance logic (this ORMapping persistance logic internally generates JDBC code. to fullfill this requirement)

④ Data / DB s/w will be manipulated based on the instructions given through persistance logic

⑤ DB s/w sends generates result back to client application.

Hibernate configuration file :-

(Any file name.xml can act as hibernate configuration file but hibernate s/w looks to take hibernate.cfg.xml as default configuration file name.

If any other file name is taken as hibernate configuration file it must be informed to hibernate s/w explicitly.

→ This configuration file contains details to connect to database s/w like driver class name Database url, DataBase username, password and etc.
All these details you should pass as the value of hibernate configuration properties.

will be parsed based on the DB s/w ...

the following properties are minimum properties of Hibernate configuration file for any DB s/w.

Hibernate.connection.driver_class

Hibernate.connection.url

Hibernate.connection.username

Hibernate.connection.password

Hibernate.dialect

mappingfilename (hbm file name)

→ Lots of properties are there in HB configuration file you can collect their names from

Hibernate_Home | etc | hibernate.properties file

(or) chapter 3 of pdf file of Hibernate...

① ~~What is the use of Hibernate s/w supplied DB s/w specific class~~

Imp Q) what is the use of Hibernate.dialect property?

~~Ans:-~~ this property takes Hibernate s/w supplied DB s/w specific classname as the value.

so this class name will change based on the DB s/w and its version that be used in Hibernate application.

Ex:-

1) oracle any version.

org.hibernate.dialect.OracleDialect

2) oracle 9i-10g

org.hibernate.dialect.Oracle9Dialect

3) MySql

org.hibernate.dialect.MYSQLDialect

for more dialect class name refer chapter(3) of pdf file.

→ hibernate.dialect property value helps hibernate s/w

1) To generate and assign intelligent & sensible default values for some hibernate configuration properties (when they are not specified in config-uration file) based on the DB s/w.

2) makes hibernate s/w optimized sql queries smoothly based on the DB s/w. to fulfil persistance operation requirement.

25/07/10

→ while developing HB persistance logic we can see 1 (or) more HBconfiguration files.

this depends upon no of DB's that are involved in the persistance logic execution.

Hibernate Mapping file :-

<any-filename>.xml can act as HB mapping file. this file should be specified in HBConfigu-ration file.

There is no default name for this file, this file contains various types of ORMapping configuration like basic ORMapping, collection Mapping, Association Mapping, Inheritance mapping and etc

- DB table member variables with table column names, this file is base file for HB s/w to understand OR Mapping configuration related to HB persistance classes.

→ you can use example applications of Hibernate-Home\test folder to gather reference example HB-Mapping files

HB persistance class:-

A java class (or) java bean class that is taken as POJO class & mapped with DB table is called HB persistance class.

→ This class is base class for the programmer to develop objects based OR Mapping style persistance operation in the client application.

→ It is recommended to take java bean as HB persistance class but it is not mandatory.

client Application:-

This application is client to DB s/w and interacts ; manipulates DB table data by using ~~HBP~~ HB persistance logic.

→ To develop this logic programmer uses Hibernate API, Hibernate resources.

→ If classes of one jar file uses the classes of other jar files then other jar files are called dependent jar files of that one jar file.

→ If java application uses third party API's (other than J2SDK API's) then the 3rd party related main & dependent jar files must be added in the class path. then only our java application will be recognizing and using 3rd party API during compilation & the execution of Application.

⇒ example scenario to understand importance of Adding main & dependend jar files in the classpath when application uses 3rd party API.

first.jar (Main jar file)

```
public class Test
{
    public void m1()
    {
        Demo d1 = new Demo();
        d1.m2();
    }
}
```

Second.jar (Dependent jar file to First.jar file)

```
public class Demo
{
    public void m2()
    {
    =
    }
}
```

```
public class App1
{
  psvm main()
  {
  Test t = new Test();
   t.m1();
  }
}
```

> javac App1.java ↵

error :- cannot invoke symbol

sol :- add first.jar file in classpath

> java App1 ↵

error :- java.lang.NoClassDefinitionError

sol :- add first & second.jar files to classpath where Demo class is used.

→ Hibernate is 3rd party s/w so, it's API is called 3rd party API. when java application uses Hibernate API, to develop Hibernate persistance logic the following jar files we need to add in classpath as main and dependent jar files

→ Hibernate 3.jar is main jar file representing HBAPI & it is having 7 no. of dependent jar files.

1) hibernate3.jar

2) dom4j-1.6.1.jar

3) cglib-2.1.3.jar

4) commons-collections-2.1.1.jar

5) commons-logging-1.0.4.jar

6) jta.jar

7) asm.jar

8) antlr-2.7.6.jar

1) → Main.jar file available in Hibernate Home Dialect.

2-8) → hibernate3.jar file available in hibernate-Home\lib folder, these file names you can collect by seeing required class in -readme.txt file of hibernate-Home\lib folders.

→ XML parsers are responsible to read & manupalate data of XML files.

Ex!- SAX (Simple API xml procesing), DOM (Document object model), JDOM (JavaDom), DOM4J and etc.

→ Hibernate ...

to read & manipulate content of xml files called lib configuration file, HB Mapping files.

*Imp
→ The web resources of web application will be compiled from command prompt & will be exicuted from webserver (or) application server. So if these web-resources uses 3rd party API then the 3rd party API related main jar files should be added in classpath & the 3rd party API related main and dependent jar files must be added in WEB-INF\lib folder of web-application.

→ When java web application based web resources use Hibernate API to develop persistance logic then the Hibernate API related main jar file (HB3.jar) must be added in the classpath and the HBAPI related main and dependent jar files (17 jar files) must be added in the WEB-INF\lib folder of web-application.

⇒ understanding synchronization b/w java object table rows while ~~data~~ dealing with OR Mapping Peristance Logic.

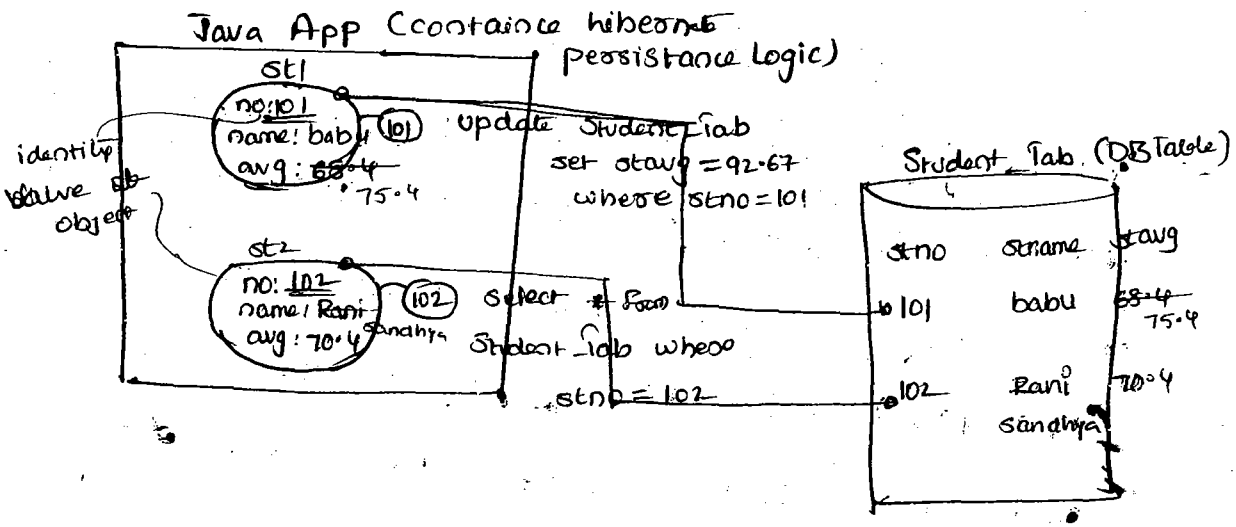
```
public class Student
{
  int no;
  String name;
  float avg;
  setX.:x()
  {
  }
  getXxx()
  {
  }
}
```

OR Mapping cfgs

Student ⟷ Student_Tab

no, name, avg ⟷ stno, stname, stavg

(no) as identity Field



→ every hibernate POJO class / peristance class object contains one identity value. Hibernate s/w Generate this identity value based on identity field configuration done by the programmer in Hibernate mapping file, programmer can configure one or more Hibernate POJO class member variables as identity field.

→ In the above Diagram no is configure as identity field, so that member variable value of st1, st2 (102) have become identity values of st1, st2 objects.

** when modification is done in POJO class object the Hibernate s/w internally generates Update query by taking identity value as criteria value to select the changes in relavant record of the DB table.

(refer above diagram)

* when modification is done in Table row (directly) then Hibernate s/w uses Select Query by taking object POJO class object identity value as criteria value to select that changes to the related java object.

(refer above diagram)

NOTE!- member variable of java class are technically called as fields.

NOTE!- Hibernate s/w identify each Hibernate POJO class object by using its identity value.

→ when the programmer modifies identity field member variable value of the object (or) if the programmer modifies identity field related to table column value directly from sql prompt HB s/w perform Synchronization in a regular manner, in this procan if necessary the HB s/w becomes ready to create new HB POJO class objects.

identity field from the member variables of Hibernate POJO class?

ans:-

SC①:- If POJO class related DB table is having one unique key constraint column or one singular primary key constrain column then take that column related member variable in HB POJO class and configure that one as singular identity field of POJO class.

SC②:- If POJO class related DB Table is having composit primary key constraint based on multiple columns then take those multiple columns related member variables of POJO class and configure them as composit identity field of POJO class.

SC③:- If POJO class related DB table contains no constraints programmer should analyse and given multiple columns in which duplicate values will not be there, then programmer has to take these multiple columns member variables of POJO class to configure them as composit identity field.

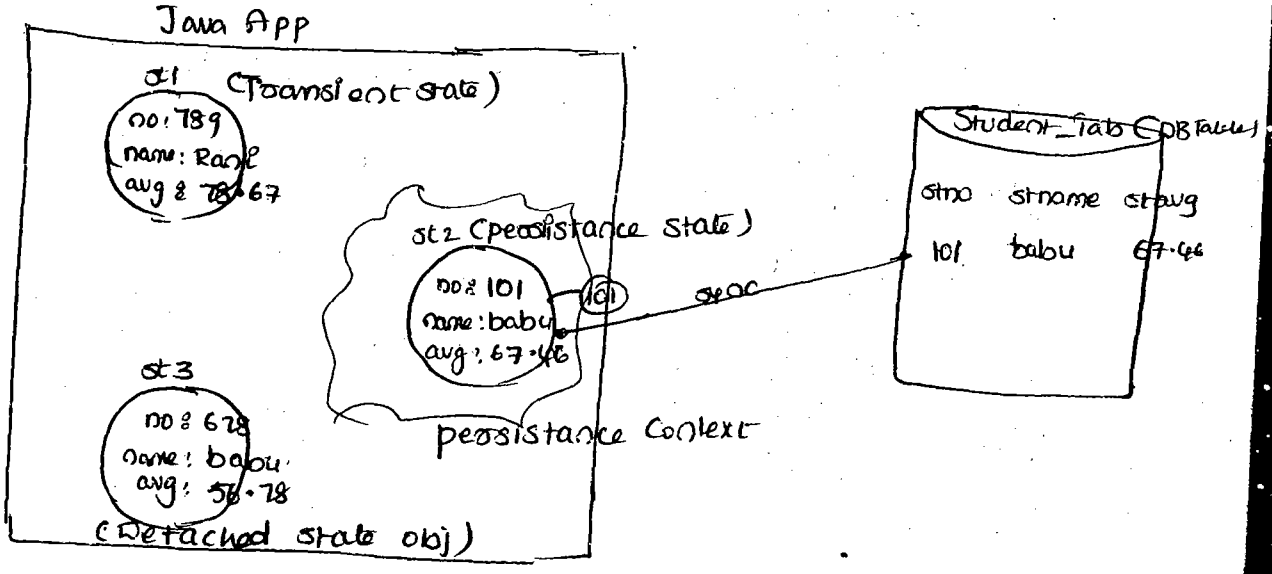
NOTE①:- If only one member variable of HB POJO class is configure as identity field then it is called "singular identity field"

If multiple member variables are configured then it is called composit identity field.

table columns are having constains are not identity field configuration must be performed on Hibernate POJO class member variables.

⇒ we can see HB POJO class object in ③ states

① Transient State
② Persistant State
③ Detached State



Transient State :-

In this state POJO class object does not represents record in the table and object doesnot contain identity value, it is just ordinary java class object of POJO class.

Persistant State :-

this state object represents table row and maintains synchronization with table row, this state object contains identity value. programmers uses this state object to perform persistance operation on table rows.

The environment ...

maintaine peosistance state object is called Peosistant context.

Detatched state, Transient state objects resides Outside peosistance context

Detatched State;-

this state object contains identity value but doesn't reprasant a record bcz either that record is deleted or the identity field colunn value of that record is modified.

Detached object doesn't peoform synchronization with record.

Detached state object has reprasant the record earlier, but it is not currently reprasenting the record. that is the reason to have identity value in Detached state object.

26|7|10

→ while developing hibernate Peosistance logic in the client application, the progoammer needs to deal with

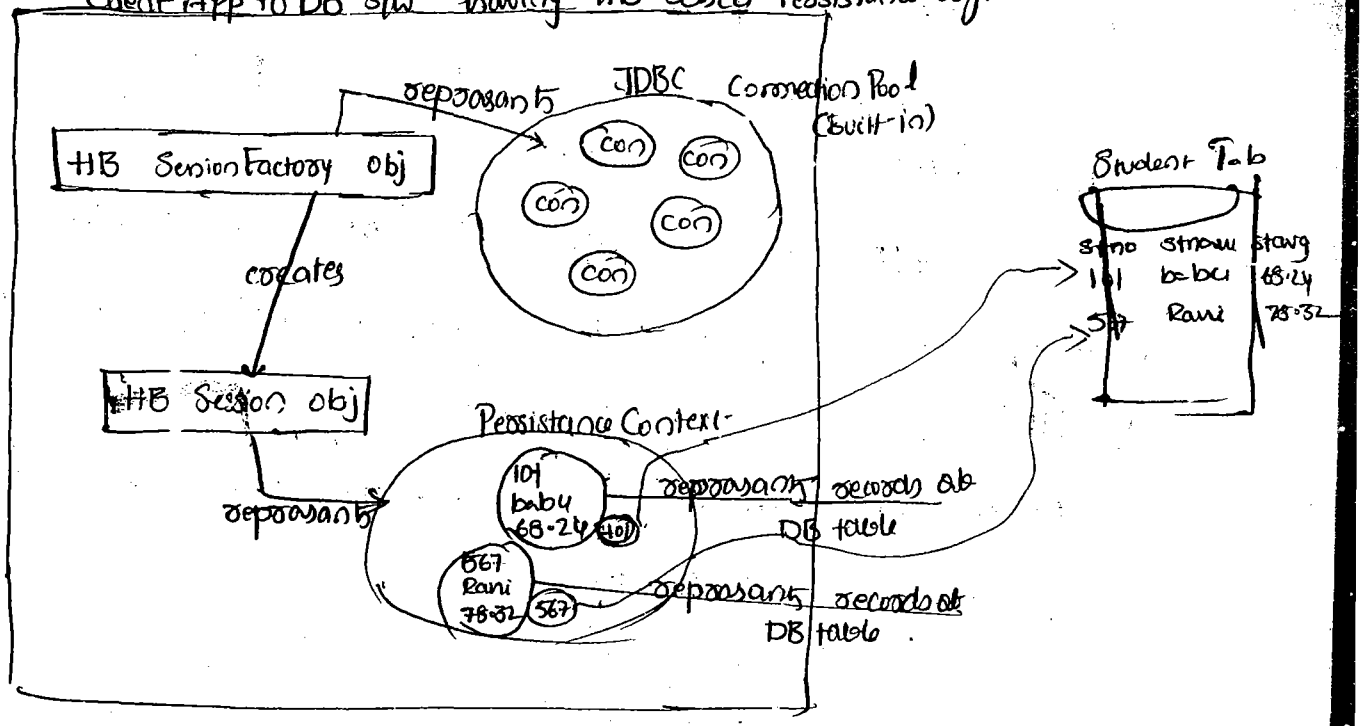
② important object-

① Hibernate SessionFactory object

② Hibernate Session object

→ ... object ?

Client App to DB s/w having HB based Persistance logic.



Hibernate SessionFactory object :-

this object creates and represents JDBC connection pool of DB, based on the details placed in HB configuration file.

This connection pool contains set of readily available JDBC connection objects., HB SessionFactory object means, it is a object of a class that implements org.hibernate.SessionFactory interface. this object is capable of ~~represent~~ creating one or more HB Session objects.

Hibernate Session object :-

SessionFactory object uses one JDBC connection object of connectionpool and this connection object based Statement object to create HB Session object.

this Session object represents Persistance Context where Persistance state HB POJO class objects reside representing

programmed to provide instruction to HB s/w to perform persistance operations on the table rows based on the operations performed on the POJO class object.

HB Session object is no way related with HttpSession object of Servlet programming.

HB Session object means it is the object of a class that implements org.hibernate.Session interface

=> Sample Code of Client App (that contains hibernate persistance logic) :-

```
// read cfg properties from hibernate.cfg.xml
// by activating HB s/w.
-> Configuration cfg = new Configuration();  ---①
   cfg = cfg.configure();                    ---②

// create HB SessionFactory object
-> SessionFactory factory = cfg.buildSessionFactory();  ---③

// create HB Session object
-> Session ses = factory.openSession();  ---④

// write HB persistance logic.

// close HB Session object
-> ses.close();
```

cfg ① → it just call HB s/w to perform Persistance logic and this object empty

cfg ② → it reads the xml file and contains details of cfg file.

① Configuration class object creation activates HB s/w.

② Configure() is the factory method of hibernate.cfg.Configuration class, which reads configuration properties from hibernate.cfg.xml file

③ buildSessionFactory(), uses configuration properties of cfg object

(a) creates JDBC connection pool

(b) creates and return HBSessionFactory object representing that connection pool.

④ openSession(), makes HBSessionFactory object to create one HibernateSession object

⑤ session.close(), closes connection with DB s/w represented by Session object and also closes Persistance Context.

→ factory.close(), closes SessionFactory object by releasing or cleaning JDBC connection pool.

Deactivated in client apps ?

Ans) when Configuration class object is created, HB SLW will be activated,

when factory.close() is called HB SLW will be deactivated.

=

⇒ Configuration cfg = new Configuration();

cfg = cfg.configure();

HB SLW looks, take hibernate.cfg.xml as Hibernate Configuration file,

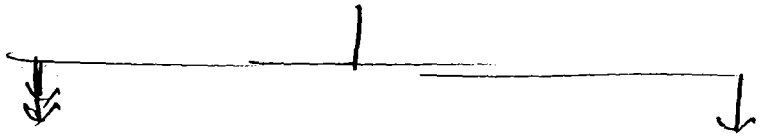
Configuration cfg = new Configuration();

cfg = cfg.configure("/myfile.xml");

HB SLW looks, take myfile.xml as HB Configuration file.

=

⇒ In the client application, we can see one or more SessionFactory object and one or more HB Session object based on the application requirement (depends on the non db DB SLW involvement).



Single Row Operations

Bulk operations

HQL
Native SQL
Criteria API

Call following on HB Session obj

save (pojo obj) → to insert a record

persist (pojo obj) → to insert a record

merge (pojo obj) → to update a record

update (pojo obj) → to update a record

save (or) update (pojo obj) → to insert or update a record

load (pojo obj, Identity value) → to select a record.

get (pojo obj, Identity value) → to select a record.

delete (pojo obj) → to delete record

**** Most recommended technique to performed persistance operation, from HB environment will be HQL

=

27/07/2010:- Develop the application to insert record into database table by using hibernate persistence logic.

→ s/w setup required

J2sdK 1.4 + s/w, oracle9i + s/w, Hibernate 3.x s/w

→ Resource that are required.

1) DataBase table

Employee (Table).

Eid	number primary key
FIRSTNAME	varchar2(20)
LASTNAME	varchar2(20)
EMAL	varchar2(20)

→ create table employee (EID number primary key,
FIRSTNAME Varchar2(20),
LASTNAME Varchar2(20),
EMAIL varchar2(20));

2) Hibernate Resource

EmpBean.java ⟶ Hibernate persistence class

hibernate.cfg.xml ⟶ Hibernate configaration file.

Employee.hbm.xml ⟶ Hibernate mapping file.

3) Client Application

TestClient.java ⟶ Client Application (java Application)

→ Procedure to develop the above application.

Step1:- Develop the above given java, xml, Resource and save them in a directory

E:\apps\hibernate\App1

- TestClient.java
- EmpBean.java.
- Hibernate.cfg.xml
- Employee.hbm.xml

hibernate.cfg.xml

<DOCTYPE .....
.....
hibernate-configaration 3.0 dtd>

<hibernate-configuration>

<session-factory>

<property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

<property name="hibernate.connection.url">jdbc:oracle:thin@localhost:1521:satya</property>.

<property name="hibernate.connection.username">scott</property>

<property name="hibernate.connection.password">tiger</property>

<property name="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</property>.

<property resource="Employee.hbm.xml"/>.

</session-factory>

</hibernate-configuration>.

Note1: Hibernate property file, hibernate.cfg.xml file of hibernate-home/dtd folder as reference example files.

Note2: The above xml is developed against the dtd rules available in hibernate-configuration-3.0.dtd file.

|| EmpBean.java (Hibernate pojo class / Hibernate persistence class).

```
public class EmpBean
{
   int no;
   String fname, lname, mail;

   public void setNO(int no)
     {
       this.no=no;
     }
   public int getNo()
     {
       return no;
     }
   public void setfname(String fname)
     {
       this.fname=fname;
```

```
    {
      return fname;
    }
  public void ~~str~~ setLname(String lname)
      {
        this.lname=lname;
      }
  public String getLname()
    {
      return lname;
    }
  public void setMail(String mail)
      {
        this.mail=mail;
      }
  public String getMail()
    { return mail;
    }
}
```

Note: The above class is javabean class that is taken as Hibernate pojo class.

Note: Hibernate pojo class must contain a zero argument constructor (public) directly or indirectly

↓ programmer generated one

↳ the java compiler generated one.

Employee.hbm.xml

```
<DOCTYPE-----
-----------
---------- hibernate-mapping-3.0.dtd">
```

<class ...>

<id name="no" column="EID"/> ⇒ singular identity field configaration.

<property name="fname" column="FIRSTNAME"/>

<property name="lname" column="LASTNAME"/>

<property name="mail" column="EMAIL"/>

</class>

</hibernate-mapping>

→ O-R-Mapping configarations.

Note: use hibernatehome/test-folder based example applications to develop above mapping file.

The above mapping is developed against the dtd rules of hibernate-mapping-3.0 dtd file

→ <property> tag is given to map membervariable with database table column

→ <id> tag can also be used for this perpose but this tag also configure that member variable as sigular identity field.

TestClient.java (Client Application to database slw that is using Hibernate persistence logic)

```
import org.hibernate.cfg.*; // For configaration class
import org.hibernate.*; // for session(i), sessionfactory(i),
                        Transaction(i).

public class TestClient {
    public static void main(String args[]) throws Exception {
    // Activate HB slw and make the slw reading HB cfg
```

configure()

```
cfg = cfg.configure();
// create sessionfactory object
SessionFactory factory = cfg.buildSessionFactory();
// create session object
Session ses = factory.openSession();
// write HB persistence logic
// create HB pojo class object having data.
EmpBean eb = new EmpBean();
eb.setEno(1010);
eb.setEname("raja");
eb.setLname("rao");
eb.setMail("rao@gmail.com");
// now insert record.
Transaction tx = ses.beginTransaction();
ses.save(eb); // insert the record.
tx.commit();
// close session object
ses.close();
// close sessionfactory obj
factory.close();
}// class of main
}// close of class
```

'eb' object state is transient state here.

'eb' object state is persistent state here.

'eb' object state is detached state here.

Note: All non select operations

HB persistent logic must be executed as transactional stmts as shown above.

Non select operations means insert, update, delete, - operations.

28/07/2017

Step2:- Add following jar files to the class path

1) Hibernate3.jar // available in HBhome directory representing - HB API.
2) dom4j-1.6.1.jar
3) cglib-2.1.3.jar
4) commons collection-2.1.1.jar
5) commons-logging-2.1.1.jar
6) jta.jar
7) asm.jar
8) antlr-2.7.6.jar
9) ojdbc14.jar file.

2-8 ⇒ Dependent jar file to hibernate3.jar available in HBhome/lib folder.

9 ⇒ Represent oracle thin driver, available in oracle/ora92/jdbc/lib/folder

Note: Always use my computer Environment veriable to add jar-files to class path.

";" symbol should be taken as seperator btw multiple values added to my computer Environment veriable, should not be taken as beggiiner or terminator.

Note: After adding new jar files to my computer Environment veriable class path ple open new command prompt inorder to

Step3:- compile java Resources of the Application

E:\App\Hibernate\App1> javac *.java↵

Step4:- Execute the client Application

E:\App\Hibernate\App1> java TestClient↵.

⇒ To make HB s/w showing internally generated sql queries to fullfil our requirement use "show-sql" property in HBConfigaration file as shown below

<property name = "show-sql"> true </property>.

⇒ when session.save() method is called by passing transient state HB pojo class object

a) That pojo class object will be made as persistent state object of persistence context by generating identity value to that object.

b) when application commit its transaction save() method related record insertion take place in database table Then on words that object and inserted record will be in synchronisation.

Q) What is the difference b/w inserting record by calling session.save() and by calling session.persist() method?

Ans:- Both methods are there to insert the record. Both methods are capable of generating identity value for HB pojo class object.

generated identity value as the return value of + ethod.
This identity value comes as seritilizable Object sed on this value the programer can evaluate wether a d is inserted-
- or not.

to with session.persist() we comit catch ease the

ration generated identity value, because the return typ f this method is void.

→ Example code of session.save() method.

cut-
```
Transaction tx = ses.beginTransaction();
Integer idval = (Integer) ses.save(eb);
s.o.p("Identity value"+ idval);
tx.commit();
```

ity

→ Example code of session.persist()

```
Transaction tx = ses.beginTransaction(
session.persist(eb); // insert the d.
tx.commit();
```

table

Note: with in the transaction, if HB pojo class ject is modified with multiple no.of times instead of generated erate sql query each time the HB s/w keep track of all the nges done to the object from begin of transaction to of transaction. Then generate final sql update query reflecting the changes when transaction is committed. This saves d reduces N/w round trips b/w application and database s/w

0

specification is a document containin rules and guidelines in the form of API's to develop n. s/w's

new s/w's, Based on propatary specification only certain company can develop the s/w's.

'jdbc, servlets, EJB, JPA, etc--- are open specification.

oNet 'is the propritary specification

Based on JPA specifion supplied by SUN MICRO SYSTEM all ORM s/w's like Hibernate, Toplink, Ibatis and etc--- are developed

→ Hibernate 3.x version s/w's are given based on JPA specification.

29/07/2010

* Code to "modify" the record by using "session.merge()" method :-

```
EmpBean eb = new EmpBean();
eb.setNo(1080);          //existance identity value
eb.setFname("newraja");
eb.setLname("newrao");
eb.setMail("newrao@gmail.com");
Transaction tx = ses.beginTransaction();
eb = (EmpBean) ses.merge(eb);   //update the record
s.o.p(eb.getNo()+" "+eb.getFname()+" "+
      eb.getLname()+" "+eb.getMail());
```

persistance state object

↓ Transient state object

```
Transaction tx = ses.beginTransaction();
    ses.update(eb);
    tx.commit();
```

-> "session.merge()" method updates the record and returns persistence state object representing the update record.

Where as "session.update()" method update the record but cannot return persistence state object of that record (the return type of session.update() method is void).

=> While performing single row operations from HB persistence logic by using singlerow operation methods of hibernate session object, They can take only "identity value" of given POJO class object as criteria value. Inorder to perform these operations based on other criteria values we need to work with HQL queries and other techniques.

* Code to "delete" the record :-

```
EmpBean eb = new EmpBean();
    eb.setNo(1080);   <-- Existing identity value.
// Now delete record
Transaction tx = ses.beginTransaction();
    ses.delete(eb);   <-- Transaction state object
    tx.commit();
```

-> Here "eb" object state will be "detached" state.

persistance logic can be executed as non-transactional operations.

* Selecting a record by using session.load() :-

```
EmpBean eb = (EmpBean) ses.load(EmpBean.class,
                                new Integer(1010));
```

(EmpBean: persistence state pojo object representing data of the selected record; EmpBean.class: pojo class name; new Integer(1010): identity value as a criteria value.)

```
s.o.p (eb.getNo() + " " + eb.getFname() + " " + eb.getLname()
       + " " + eb.getMail());
```

Note :- Comment s.o.p statement to observe the 'lazy loading' with session.load() method (it cannot apply select queries)

* Select a record by using session.get() :-

```
EmpBean eb = (EmpBean) ses.get(EmpBean.class,
                               new Integer(1010));

s.o.p (eb.getNo() + " " + eb.getFname() + " " + eb.getLname
       + " " + eb.getMail());
```

* ⇒ Object of java.lang.Class can represents a java class or an interface or an abstract class in a running java application.

"EmpBean.class" statement kept in application generates object of java.lang.class representing "E

given above are called they use given identity value a criteria value, They select record from the table, They store selected record into given pojo class object (for this, it dynamically creates object for given pojo class) and return that object.

***

=> "session.load()" methods performs "lazy loading" to select the record that means until "getXxx()" are called on the return object (eb) record "will not" be selected from the table.

-> session.get() method doesnot perform "lazy loading" in any situation.

=> Q) How can we avoid "lazy loading" even though you are working with "session.load()" method?

ans:- => public Object load (Class name, Serializable id)

This performs "lazy loading"

ex:- EmpBean eb = (EmpBean) ses.load (EmpBean.class, new Integer(1010));

=> public void load (Object obj, Serializable id)

It doesnot perform "Lazy loading"

ex:- EmpBean eb = new EmpBean();

ses.load (eb, new Integer(1010));

=> public Object get(Class name, Serializable id)

it doesnot perform "lazy loading"

The session.saveOrUpdate() method ...

class object identity field member variable value as criteria value to check whether that value based record is already available in the table or not. If available this method performs "update" operation otherwise this method performs "insert" operation.

Ex:-
```
EmpBean eb = new EmpBean();
    eb.setNo(1009);
    eb.setFname("raja1");
    eb.setLname("rao1");
    eb.setMail("rao1@gmail.com");
Transaction tx = ses.beginTransaction();
    ses.saveOrUpdate(eb);
    tx.commit();
```

=> In serverside components like servlets, JSP's, EJB components and etc it is recommended to use "session.get()" method to select a record.

In other situations it is upto the choice of programmer to use "session.load()" (or) "session.get()" method to select a record

MySql:

type: DB s/w

version: 4.0

vendor: Mysql

open source DB s/w

Default port no: 3306

default logical db names: test, mysql

typ: 4 mechanism based driveoname :- connector;

dbcwt admin useoname and password:

root (useoneme)

root (panwood)

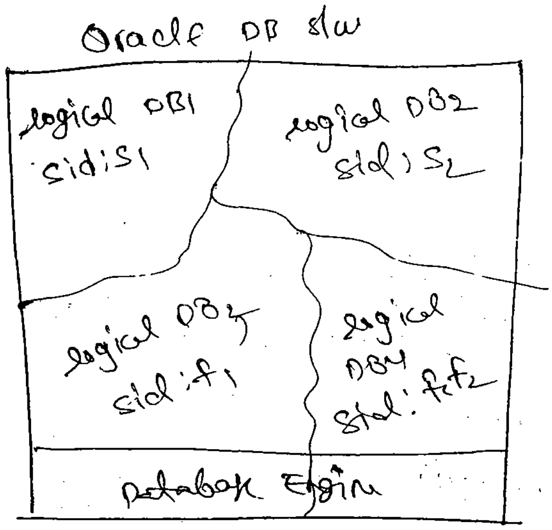
→ After installing mysql 4.x s/w we need to activate that s/w seperatly by using winmysqladmin tool available in mysql/home/bin directory (look for traffic signal symbol in the system tray)

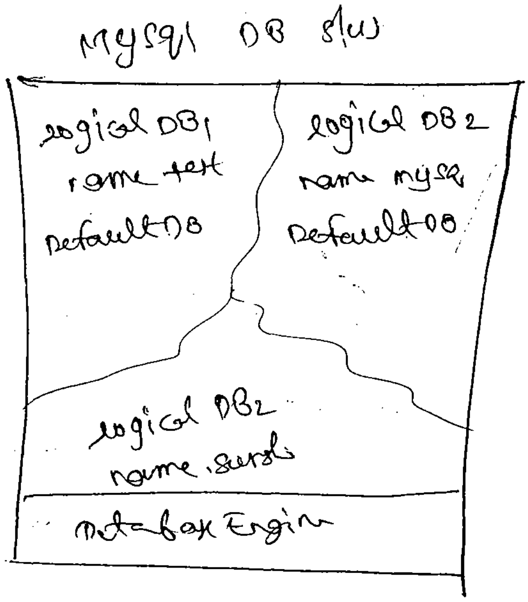
→ My sql front is a thirdparty supplied GUI tool to perform operations by connecting to mysql database s/w it is like sqlplustool of oracle DB s/w

→ logical database are the logical portitions created in physical database s/w installed in computer. Each logical db will be indentified with one logical name. In oracle DB s/w this name is also called as sid (service id). In my ms Access DB s/w each MDB acts as one logical database and that MDB file name is logical database name.

In a company if multiple projects are using same DB s/w then ~~that~~ db s/w will be installed

database will be created in that database s/w on one per project basis.





⇒ Procedure to create logical database and DB table in that by using mysql front tool.

connector/jdbc driver class

step1 launch mysql front tool

[New] → [my con name] → OK → host; name:- localhost, user:- root, password: ~~root~~ No password → ~~connect~~ port:3306 ↵ connect → Right click on root at localhost → create DB → Database name; sureshHBDB → OK → select sureshHBDB → Database tab → RC → create table → Table name: Employee → Add → EID — INT:5 primarykey, uniquic_key, non-null Firstname varchar20 lastname varchar20 Email varchar20 → create → select Employee table rc → insert record 101 Raja rao rao@gmail.com
456 Ramesh chary chary@gmail.com

connector/J driver:-

type type 4 mechanism based jdbc driver

website : www.devA.com.

driver class : org.gjt.mm.mysql.Driver

url :- jdbc:mysql:///<logical db name>.

jar file : mysql-connector-java-3.0.8-stable-bin.jar

Search Engine

→ The above jar file is available in mysql-connector-java-3.0.8-stable-.zip file Extraction

Note :- MS sql s/w setup, MS sql front s/w setup connector/j driver connect stu folder of DVD.

→ Procedure to make our first HB application to interact with MySql database s/w.

<u>step1 :-</u> Keep all the resources as it is, but do following modification in HB configaration file.

<u>hibernate.cfg.xml</u>

<DOCTYPE ----- >

<hibernate-configaration>

<session-factory>

<property name= >org.gjt.mm.

Driver</property>

< " " >jdbc:mysql:///suresh

HBDB</property>

<property name="connection.username">root</property>

< " = " .password"></property>

<property name="dialect">org.hibernate.dialect.mysql

database</property>

</session-factory>

</hibernate-configuration>.

Step2) Add mssql-connector-java-3.0.8-stable-bin.jar file to the class path along with already existing jar files

Note: it is always recommended to add new jar file to class path at the begining of existing jar file.

Step3) Run the client Application.

We can't interact with MS-Access DB s/w by using hb persistence logic because the HB s/w not providing dialect class for MS-Access.

The word HB is optional in HB configuration property name

=

31-07-2010 POJO object to DB table:-

⊙ session.flush() method, forces the HB s/w to synchronize with underlying DB s/w records by using the data in persistance state POJO class object in persistance context.

Write following code in client application, to see the modification done in object in associated record in the DB Table.

=
=
=
=

// Select record from DB table into POJO object

```
EmpBean eb= (EmpBean) ses.get(EmpBean.class, new
                Integer
    s.o.p( eb.getNo + " " +eb.getFname()+ " " +
            eb.getLname() + " " + eb.getMail()));
```

//modify "eb" (persistence state obj) object data

```
    eb.setMail( "xyz1@xyz1.com");
    ~~ses.flush ( " xyz1@~~
    ses.flush() ; // this method reflect changes done
// in "eb"
// to DB table row through synchronization.
// close session obj.
ses.close();
}} factory.close(); // close sessionfactory
```

the above code demonstrate POJO object to DB table row Synchronization.

stence

DB table to POJO Object ?— (Synchronization)

code in the client application to demonstrate DB table row to java object based Synchronization [Modification done in DB table row from DB side will reflect in the associated persistance state object]

// select record from db table into POJO obj.

```
EmpBean eb = (EmpBean) ses.get(EmpBean.class,
                new Integer (1016));
s.op( eb.getNo()+ " " + eb.getFname()+ " " + eb.getLname()+
        " " + eb.getMail());
```

//record "1016" in db table from sql prompt.

```
ses.refresh(eb);
```

// re-reads data from db table row into //eb object for synchronization.

```
S.O.P(eb.getNo()+" "+eb.getFname()+" "+
        eb.getLname()+" "+eb.getMail());
```

Note:- Can you develop HB application (or) HB Persistent logic without HB Configuration file?

ans:- yes, it is possible and there are two approaches for this,

approach(1):- (By using setProperty() of org.hibernate.cfg.Configuration class.

Code in client application:-

```
Configuration cfg = new Configuration();
  cfg.setProperty("hibernate.connection.driver_class",
                  "oracle.jdbc.driver.OracleDriver");

  cfg.setProperty("hibernate.connection.url",
                  "jdbc:oracle:thin:@lh:1521");

username
password  cfg.setProperty

dialect   cfg.setProperty("hibernate.dialect","org.hibernate.dialect.
                  Oracle9Dialect");

  cfg.addFile("Employee.hbm.xml");

  //create SessionFactory obj
  SF ........ cfg.buildSF();
```

→ since configuration property values are hard coded in the application the application will loose its flexibility

→ we cannot work with certain miscellaneous HB configuration properties like 'show_sql'

cfg.setProperty("show_sql","true"); //it cannot show query

to solve this problem no ① use Approach ②

Approach (2) :-

→ the text file that maintains the entries in the form of <key, value> pairs.

→ java.util.properties is a Map DataStructure each element of this DataStructure should have string values as <key and value> pair.

this class is a subclass of Hashtable class this Hashtable Map DataStructure elements can take any objects as keys and any objects as values

the element values of java.util.properties DataStructure can be collected from the Text Properties file.

Note:-

Example code based on Approach No:②

proops.txt:-

```
# this is properties file
hibernate.connection.driver_class = oracle.jdbc.driver.OracleDriver
hibernate.connection.url = jdbc:oracle:thin:@localhost:1521:satya
hibernate.connection.username = scott
```

show_sql = true.

Code in client application to create HB Session object :-

```
//read value from props.txt to java.util.Properties obj

FileInputStream fis = new FileInputStream("props.txt");
  Properties p = new Properties();
    p.load(fis); // load data from props.txt to
                    "p" object.
     S.o.P( p.toString());
//Activate HB s/w and make the s/w reading HB cfg file.
   Configuration cfg = new Configuration();
    cfg.setProperties(p);
     cfg.addFile("Employee.hbm.xml");

// create SessionFactory object
  ---
  ---
```

Note :- these approach ① & ② practices are not industry standard practices always work with xml file as HB Configuration file in real world applications.

object in the form of single line statement.

```
Session ses = new Configuration().configure("/myCfg.xml").
              buildSessionFactory().openSession();
```

the above statement demonstrate method chaining process.

=

2|08|10

MyEclipse :-

type : IDE s/w to Develop java, j2ee and other java f/w s/w's based application.

Version : 6.x (compatable with j2sdk 1.5)

Vendor :- Eclipse

Commercial s/w :-

(clear codes)
does not give built in server, but allows to ~~work with~~ configure all external servers.

To download s/w :- www.myeclipseide.com

for documention :- www.myeclipseide.com.

→ a plug in is a patch s/w (or) s/w application to enhance the functionalities of existing s/w (or) s/w application.

plug-in's come as jar files.

MyEclipse IDE = Eclipse IDE + builtin plugins.

Eclipse IDE	MyEclipseIDE
1) Design to develop basic J2sdk applicatio.	1) Design to develop all kinds of java, JEE apps and other Framework s/w based applications
2) external plug-ins are required to develop advanced technologies based applications.	2) built in plug-in's are these to develop advanced technologies based java applications.
3) opensource s/w	3) commercial s/w.

—

(Q) How to add plug-in's to the projects of EclipseIDE?

Ans:-

(1) download jar files that represent certain technologies related plug-in's. like hibernate plugin, spring plugin etc.

Note:- these plug-in provide the necessary environment required for the programmer to develop certain technology based application from Eclipse IDE.

(2) Creat project in EclipseIDE and observe that there will be plug-in's folder in the project.

(3) Add the downloaded plug-in related jar files to plug-in folder of project.

activates plug-in and programmer can use the plug-ins applied environment to develop that technology based application.

Procedure to develop 1st HB application by using MyEclipse 6.x IDE :-

Step①:- ~~Launch~~ Launch MyEclipse IDE and choose work space.

Note:- work space is a folder where all projects created in MyEclipse IDE will be saved.

** Step②:- update the subscription

MyEclipse Menu
↳ update subscription
↳ subscribe :- administrator
subscription code :- xxxx (Correct code)

Step③:- use MyEclipse IDE, to create DB profile :-

⇒ connected with ORACLE DB s/w. (DBMode)

Window
↳ open perspective
↳ other
↳ MyEclipse DB Explorer
ok
goto 'DB Browser window
Right click
New
Oracle thin Driver

Driver name :- oap (logical name of profile)
Connection url :- jdbc:oracle:thin:@localhost:1521:sid
username :- scott
password :- tiger

Derived mode:-

Add jars:-

☑ save password.

↓

Next

↳ Next

↳ finish

⇒ Right click on ORAP of DB Browser window.

↓

Open connection.

⇒

Step(4):- Make sure that Employee table is available in ORACLE DB s/w having atleast one primary key constraint column.

Create java project in MyEclipse IDE

- File
  - ↳ New
    - ↳ project
      - ↳ java project
        - ↳ Next

project name:- MyHB project

↓

Next

↳ finish

⇒ Add hibernate capabilities to the project

Right click on project :-

↳ MyEclipse

↳ Add Hibernate capabilities.

select ☑ Hibernate 3.1 core libraries

☑ ~~EEN~~ " " Advanced support library

↓

Next

↓

Next

DataSource :- use JDBC Driver

DB Drivers :- orap (DB profile created above)

⋮

[Next]

☑ Create SessionFactory class ?

⇒ src

⇒ java package : [New] → p1

classname : MyHelper

↓

[finish]

↓

Goto hibernate.cfg.xml file

↳ [add] miscellanious properties

→ show_sql

→

Note :- the above step gives following things.

a) Adds HibernateAPI related jar files to build path (or) classpath of the project

b) hibernate.cfg.xml as hibernate configuration file

c) use MyHelper.java file having code to create hibernate SF object and Session object.

⇒ Postgres + Hibernate Reverse Engineering on Employee table to generate HB POJO class (Employee), HB Mapping file (Employee.hbm.xml) dynamically :-

Window
↳ open perspective
↳ other
↳ MyEclipse DB Explorer
↳ DB Browser Window
Expand ORAP ↵
Expand Connected to ORAP ↵
Expand Scott ↵
↳ Expand table
↳ Right click on Employee
↓
HB Reverse Engineering

Java src folder :- /MyEclipse/src.

☑ HB Mapping file.
☑ update HB configuration with mapping file location.

☑ Java Data Object

Next

Type Mapping :- Select HB types ⓞ
↳ Next
↳ Finish
↳ OK

① Employee.java as a POJO class

② Employee.hbm.xml as HBMapping file

⇒ Develop client App in the project :-

Right click project

→ New

↳ class

Name : TestClient

☑ main(String[] args)

Finish

Ctrl+Shift+O // import statements

class TestClient

Transaction tx = session.beginTransaction();

Employee e1 = new Employee();

e1.setEid(016);

e1.setEname("babu");

e1.setLname("kasim");

e1.setEmail("a@b.com");

// e1.setEmail(args[0]);

```
            tx.commit();
            MyHelper.closeSession();
        }
        catch (HBException hb)
        {
        }
        catch (Exception e)
        {
        }
    }
}
```

⇒ Execute the client application :-

Right click on source code of : TestCode.java

↳ Run As

↳ java application.

=

⇒ What is the diff b/w session.update & session.merge ?

both methods take POJO class object as argument value,

session.update() :- update the record, only when given object related record is available in the table, otherwise update method fails in record updation. [if record is not available]

session.merge() :- merge method tries to update the record if given POJO object record is available in DB table. ow, (if not available) this method inserts the record by using given POJO class object data.

application being from myEclipse IDE :-

Right click-on source code of application
↓
Run AS
↓
open run dialog
↓
Chose app name (TestClient)
↓
Arguments tab

Program arguments: val1 <space> val2 <space> val3 :---.
args[0] args[1] args[2]

↓
Apply
↳ Run

=

MYSQL
8/8/19

procedure to create DB profile in MyEclipse IDE pointing to MySQL DB s/w :-

Window menu
↳ open perspective
↳ other
↳ MyEclipse DB Explorer
↓
DB Browser window
Right click
new
Driver template : MYSQL Connector
Driver name : mysql
Connection url : jdbc:mysql:///satyadbdb.
username : root
password : — (no pwd)
jar file :- — Add jar
...connector-java-5.1.8-stable-bin.jar

↓
Next
↳ finish
↓

Right click on mysqlp
↳ DB Browserwindow
↳ open connection
↳ ok

⇒ Example Application to communicate with multiple DB s/ws by using Hibernate persistance logic :-

- while working with Oracle 10g DB s/w [XE] (xpress Edition) to communicate from HB Persistance Logic also add "ehcache-1.2.3" jar file in class path along with other regular jar files.

→ Dialect class name for this environment
org.hibernate.dialect.Oracle10gDialect
(or)
org.hibernate.dialect.Oracle9Dialect.

Resources Required :-

(1) in oracle DB s/w :-

Employee (table)

EID	pk	number
FIRSTNAME		VARCHAR2 (20)
LAST NAME		(20)
EMAIL		(20)

make sure that some records are available in this table.

Employee (table)

EID pk	int(5)
FIRSTNAME	varchar(20)
LASTNAME	"
EMAIL	"

(3) files :-

mycfg-ora.xml	(oracle DB s/w related cfg file) } HB config files
mycfg-mysql.xml	(mysql DB s/w related cfg file)
Employee.hbm.xml	(HBM file)
EmpBean.java	(HB persistance class)
TestClient.java	(Client app having persistance logic)

AIM :- AIM OF THE ABOVE APPLICATION IS SELECT A RECORD FROM DB TABLE (Employee) OF ORACLE S/W AND INSERT THAT RECORD INTO MYSQL DB TABLE (EMPLOYEE)

mycfg-ora.xml :-

write minimum ⑥ properties to point into oracle DB s/w and take Employee.hbm.xml as mapping file.

mycfg-mysql.xml :-

write minimum ⑥ properties to point into mysql DB s/w and take Employee.hbm.xml as mapping file.

(4) Employee.hbm.xml :-

← same as first application →

← same as 1st application →

TestClient.java :-

```
import org.hibernate.cfg.*;
import org.hibernate.*;
public class TestClient
{
    main() throws Exception.
    {
// get HB session object connected to oracle DB s/w.
Configuration oracfg = new Configuration();
    oracfg = oracfg.configure("/mycfg-ora.xml");
SessionFactory orafactory = oracfg.buildSF();
 Session orases = orafactory.openSession();
// get HB session object connected to mysql DB s/w
Configuration mysqlcfg = new Configuration();
  mysqlcfg = mysqlcfg.configure("/mycfg-mysql.xml");
SessionFactory mysqlfactory = mysqlcfg.buildSF();
  Session mysqlses = mysqlfactory.openSession();
// write HB peasistance logic interacting with multiple DB s/w
  // select a record from oracle DB s/w
EmpBean eb=(EmpBean) orases.get(EmpBean.class,
                                new Integer(1016));
  // insert the record into mysql DB s/w
Transaction tx = mysqlses.beginTransaction();
    mysqlses.save(eb);
      tx.commit();
    orases.close();     } // closing session objects
    mysqlses.close();   }
    orafactory.close();     } // closing SessionFactory objects
    mysqlfactory.close();   }
```

Add Regular hibernate jar

add ojdbc14.jar, mysql-connector-java-3.0.8-stable-bin.jar

=> compile all java resources and execute the client Application.

> SessionFactory object is a singleton object but it is immutable (not changeable this object)

NOTE①:- saying the implementation class of org.hibernate.SessionFactory interface is singleton java class is a wrong statement - because we can create multiple SF objects in a single java application as shown above.

NOTE②:- the java class that allows to create only one object for jvm is called singleton java class

→ SessionFactory object is immutable object, that means after creating SF object if you modify HB configuration properties dynamically at time they will not be reflected to the SessionFactory object.

→ String class object is immutable object, StringBuffer class object is mutable object.

→ immutable object means when modification is done in the object it will not reflect in the same object, modification will be done by creating new object

→ mutable object mean, modifications done in the Node of the object will reflect in the same object.

Test.java (mutable class)

```
class Test
{
  int a;
  String b;

  Test(int a, String b)
  {
    this.a = a;
    this.b = b;
  }
  public void setData1(int a)
  {
    this.a = a;
  }
  public void setData2(String b)
  {
    this.b = b;
  }
  public String toString()
  {
    return "a=" + a + " b=" + b;
  }
  main(String[] args)
  {
    Test t = new Test(10, "babu");
    S.O.P("t obj data " + t.toString());
    t.setData1(20);
    t.setData2("hello");
    S.O.P("t obj data " + t.toString());
  }
}
```

Comment:- when setData1(), setData2() methods are called on "t" object they directly modify 't' object data so, Test class here is mutable class

```
class Test
{
  int a;
  String b;
 Test(int a, String b)
  {
    this.a=a;
    this.b=b;
  }
 Test()
  {
  }
 public Test setDate1(int a)
  {
    Test temp= new Test();
      temp.a=a;
      return temp;
  }
 public Test setDate(String b)
  {
    Test temp=new Test();
      temp.b=b;
      return temp;
  }
 main(String[] args)
  {
    Test t = new Test(10,"babu");
    S.O.P("t obj date"+t.toString());
    Test t1=t.setDate1(20);
    Test t2=t.setDate("hello");
    S.O.P("t obj date"+t.toString());
    S.O.P("t1 obj date"+t1.toString());
    S.O.P("t2 obj date"+t2.toString());
  }
}
```

```
public String toString()
{
  return "a="+a+"b="+b;
}
```

Conclusion:-
when setDate1(), setDate2() are called on 't' object, they are not modifying 't' obj date, more over they are creating and returning new objects having modification related data, so here Test class is called immutable class.

=> while developing user defined immutable class, no java method of that class should have logic of modifying current invoking object data.

If modification is required these method should construct new object and should keep modified data in that new object.

4/8/10 IDENTIFIERS :- (ALGORITHMS)

⇒ Identity value of HB POJO class object is the criteria value for HB sln to perform synchronization b/w HB POJO class object and table row.

→ Hibernate supply 10 predefined algorithm as identity value generator for POJO class object. Most of these algorithms generate dynamic and unique value as Identity values of HB POJO class objects.

→ These algorithms are predefined classes supplied by HB API, implementing org.hibernate.id.IdentifierGenerator

all these classes having nick names / short names to utilise.

Nick Name	Algorithm class name.
1) assigned	⟶ org.hibernate.id.Assigned
default algorithm.	
2) increment	⟶ org.hibernate.id.IncrementGenerator
3) identity	⟶ org.hibernate.id.IdentityGenerator
4) sequence	⟶ org.hibernate.id.SequenceGenerator
5) hilo (High and low)	→ org.hibernate.id.TableHiLoGenerator
6) seqhilo	→ org.hibernate.id.SequenceHiLoGenerator

8) guid ⟶ org.hibernate.id.GUIDGenerator

9) native ⟶ —

10) select ⟶ org.hibernate.id.SelectGenerator

11) foreign ⟶ org.hibernate.id.ForeignGenerator

=

→ To specify these algorithms use "<generator> tag (sub tag of <id>) in hibernate mapping file.

If no algorithm is explicitly specified in mapping file HB s/w ~~tooks~~ ~~to~~ take assigned as default algorithm.

→ while working with any identity value generator algorithm consider the following two rules

① make sure that the type of the identity value generated by algorithm is comparable with DataType of identity field member variable of POJO class.

② make sure that underlying D/B s/w supports the choosen algorithm.

ASSIGNED ALGORITHM :-

This algorithm lets the application developer to assign identity value to HB pojo class object manually before calling session.save() (or) other method to insert the record.

code in mapping file.

employee.hbm.xml :-

```
<hibernate-mapping>
    <class name="EmpBean" table="Employee">
  <id name="no" column="EID"> <!--singular identity field cfg-->
      <generator class="assigned"/>
                    (i) org.hibernate.id.Assigned
  </id>
  <property name="fname" column="FIRSTNAME"/>
     :
  </class>
</hibernate-mapping>
```

NOTE:- while testing all the algorithms, make sure that the client application is inserting record by calling session.save(), as shown below.

```
:

EmpBean eb=new EmpBean();
    eb.setNo(235);
    eb.setFname("x1");
    eb.setLname("y1");
    eb.setMail("x1@y1.com");

Transaction txn= ses.beginTransaction();
Integer idval=(Integer)ses.save(eb);
  s.o.p("idvalue is"+idval.toString());
   tx.commit();

  ses.close();
  factory.close();
```

8

Assigned algorithm can generate any type of identity values and assigned algorithm is compatable with all DB s/w.

⇒ Increment algorithm :- (existing value + 1)

This algorithm generates identity value of type long, short or int, this algorithm works with all D/B s/ws., this algorithm uses max value + 1 formula on identity field related table column values to generate new identity value.

Example code :-

```
<id name="no" column="EID" />  <!-- singular identity field cfg -->
  <generator class="increment" />
</id>
```

⇒ Identity Algorithm :- ✗ ORACLE (Mar-1)

Supports identity column's in DB2, MySQL, SQL SERVER, SYBASE, Hypersonic SQL DB s/w.

This algorithm returns identity value of type long (or) short (or) int, This algorithm does not work with Oracle DB s/w.

MySQL DB s/w table column value can be made as identity column by applying auto increment constraint in the column (To apply this constraint the column already should contain primary key constraint)

identity field related table column values to generate next identity value irrespective of wether records in the table deleted or not that means it also considers deleted record values while picking-up maximum value from the table column.

ex:-

```
<id name="no" column="EID">
    <generator class="identity" />
</id>
```

Note:- while testing above code use mySQL DB slw and apply following constraints on the EID column of Employee table.

UNIQUE, NOTNULL, primary key, auto increment

6) what is the diff b/w increment algorithm and identity algorithm ?

Increment	identity
1) identity field member Variable related column in DB, need not to have any constraint to work with this algorithm	1) identity field related column in DB table should be taken as identity column by applying auto-increment constraint to work with this algorithm.
2) when all records of the table are deleted increment	2) if all records of table are deleted after applying

algorithm generates 1 as the identity value.

and working with identity algorithm, this algorithm still considers deleted record values to generate new identity value as discussed above.

③ ~~HB~~ HB s/w generates identity value for HB POJO class object, when increment algorithm is used.

③ DB s/w generates identity value based on identity column behaviour and gives that value to HB s/w then HB s/w assigns that value as identity value for POJO class object.

④ This algorithm works with all DB s/w so

④ this algorithm works with only that DB s/w s which support identity columns.

5/8/10

SEQUENCE ALGORITHM:-

This algorithm uses sequence, created in DB2, postgreSQL, ORACLE, SAP, HCKOi, Interbase DB s/w to generate identified value of type long, short (or) int

NOTE:- this algorithm doesn't work with MYSQL DB s/w because the MYSQL DB s/w doesn't support sequences.

NOTE:- To pass input value to any algorithm configured in HB mapping file use <param> tag as the subtag of <generator> tag

→ create sequence mydb_seq1 increment by 5; ORACLE)

select * from user_sequence;

step①:- create sequence in ORACLE DB s/w as shown below.

SQL> create sequence myoog_seq1 increment by 5;

step②:- write following code in HB mapping file.

```
<id name="no" column="EID">
  <generator class="xxxxx.sequence">
    <param name="sequence">myoog_seq1</param>
  </generator>
</id>
```

→ Algorithm name

→ parameter name

sequence_name Created in oracle s/w.

NOTE:- we can work with sequence algorithm only with those DB s/ws which support sequences creation.

When sequence algorithm is used the HB s/w gets identity value from DB s/w by using the specified sequence name.

hilo:-

This algorithm uses hi/lo algorithm to efficiently generate identity values of type long, short or int by using helper table column value and given parameter values (Max_lo) as source of values. This algorithm works with all DB s/ws; this algorithm expects the following ③ parameter values ie they are table (expects helper table name), column (expects helper column name in the table),

Helper Table column value always represents a number incrementation indicating how-many records are inserted in the DB table by using hilo algorithm based identity values.

example code:-

step①:- create helper table having helper column (make sure that this helper column having 1 numeric value as initial value.

```
SQL> create table mytable(mycol number);
SQL> insert into mytable values(10);
```

step②:- configure hilo algorithm in HB mapping file as shown below.

```
<id name="no" column="EID">
  <generator class="hilo">
    <param name="table">mytable</param>
    <param name="column">mycol</param>
    <param name="max_lo">10</param>
  </generator>
</id>
```

=

This algorithm uses following formulas to generate initial identity value and next identity values.

formula to generate initial value:-

(max_lo + 1) * helper col val

(or)

(max_lo val) * helper col val) + helper col val

formula to generate next values:-

(max_lo val + 1)

this algorithm doesn't have its own behaviour, this algorithm dynamically picks up identity, sequence or hilo algorithm (one of these) depending upon the capabilities of underlying DB Sw. This algorithm does not have own class name because it does not have its own individual behaviour

while working with this algorithm parameter passed based on the capabilities of Underlying DB Sw.

This algorithm works with all DB Sws

When native algorithm is taken, by taking oracle as underlying (ORACLE) DB Sw, then this native algorithm internally uses sequence algorithm to generate identity values for HB POJO objects.

Code in mapping file for ORACLE DB Sw :-

```
<id name="no" column="EID">
    <generator class="native">
     <param name="sequence">myora_seq</param>
    </generator>
</id>
```

while working with mysql DB Sw, the native algorithm internally uses identity algorithm to generate identity value for pojo class object

```
<id name="no" column="EID">
  <generator class="native" />
</id>
```

if the underlying DB s/w doesnot support both identity and sequence algorithms then the native algorithm internally uses 'hilo-algorithm'

seqhilo algorithm :-

This algorithm is enhancement of hilo-algorithm which internally uses given sequence name and parameter values to generate identity values of type long, short or int this algorithm does not expect special table name from the programmer, this algorithm works with only that DB s/w. which supports DB sequences.

(works with oracle, doesn't works with mysql)

Parameter names for this algorithm are seq, max_lo

example code :-

step① :- make sure that myora_seq1 is available in oracle DB s/w.

step② :- configure seqhilo algorithm in hibernate mapping file as shown below.

```
<id name="no" column="EID">
  <generator class="seqhilo">
    <param name="sequence">myora_seq1</param>
    <param name="max_lo">10</param>
  </generator>
</id>
```

Cnext val in sequence + increment by val of seq +1)
* (max_lo val) +1

Formula to generate next identity value:-

previous identity val + (Cmax_lo val+1) * increment by val of sequence)
=

06/08/10

uuid algorithm:- universal unique id

This algorithm uses, net work IP address as base value to generate string type, hexadecimal notation value as identity value for POJO objects.

This value contains 32 digits in hexadecimal notation

→ This algorithm works with all DB s/ws.

Example code:-

① take "no" number variable in EmpBean class a String variable.

② Change EID column Datatype to varchar2(35) in Employee table;

SQL> Alter table Employee modify EID varchar2(35);

③ configure uuid algorithm as shown below in mapping file.

```
<id name="no" column="EID">
    <generator class="uuid"/>
</id>
```

insert the record.

```
Transaction tx = ses.beginTransaction();
String idval = (String) ses.save(eb);
S.O.P("id value is" + idval.toString());
tx.commit();
```

=

guid :- global unique id

this algorithm uses DB generated GUID String as identity value.

this only work with SQL SERVER, MY SQL DB's/w

=

Select :-

this algorithm generates identity value by executing trigger program in underlying DB s/w.

Foreign algorithm :-

In order to use, identity value of associated POJO object as the identity value of current POJO object use Foreign algorithm. This algorithm is very useful while working with one-to-one relationship.

Conclusion on algorithm :-

TL/PL decides identity value generated algorithm for POJO class object and HB developers configures them programatically.

If identity field DATATYPE is numeric datatype it is recommended to use increment (or) native algorithm.

numeric datatype then use assigned algorithm.

only on singular identity field configuration, we can use these algorithms.

while work with composit field identity configu there is no possibility ab working with these algorithms.

use <id> ab HBM file for singular identity field configuration.

use <composit-id> tag for composite identity field configuration.

=

*** Q) How to generate the following type values as identity values for POJO class object?

→ AB001, AB002, AB003.

ans- To generate these values write logic manually by using String manupulation concept and use 'assigned' algorithm to set these values as identity values for POJO class objects.

=

***

If DB Table name is same as POJO class name and If column names are same as POJO class no, variable name then there is no need ab specifying table name and column names in HB Mapping file while performing OR Mapping configuration.

```
public class EmpBean
{
  int no;
  String fname, lname, mail;
  setters & getters
}
```

DB table:-

```
EmpBean (table name)
  no  (numbers)
  Fname (VC2(20))
  lname (VC2(20))
  mail (VC2(20))
```

=

for the above setup, we can write code in mapping file as shown below.

```
<hibernate-mapping>
  <class  name = "EmpBean">
    <id  name = "no">
     <generator  class = "increment">
    </id>
    <property  name = "fname"/>
    <property  name = "lname" />
    <property  name = "mail" />
    </class>
</hibernate-mapping>
```

=> To observe that table name and column names are not specified in HB mapping file.

specify POJO class name as fully qualified class name as shown below.

```
<hibernate-mapping>
    <class name = "pl.EmpBean">
        ≡          ↳ java package name
    </class>
</hibernate-mapping>
```

(or)

```
<hibernate-mapping package = "pl">
    <class name = "EmpBean">
        ≡
    </class>
</hibernate-mapping>
```

⇒

HIBERNATE s/w SUPPLY SOME BUILT IN TOOLS TO CREATE OR ALTER DB TABLES IN UNDERLYING DB s/w THESE TOOLS ARE ALSO USEFUL TO GENARATE HB POJO CLASS DYNAMICALLY THEY ARE.

① Schema Export

② SchemaUpdate

③ CodeGenarator.

→ Scheema Export tool · always create new table

→ Scheema update tool can create new table or can alter existing table by adding new columns.

These tools are very useful to create tables dynamically in new DB s/w based on HB mapping fi

new DB S/w, then these tools are very useful.

→ SchemaExport tool always create new table in DB S/w. if table is already available then it creates new table by droping the existing table.

we can pass instruction to this tool from HB mapping file by using length, type, unique, not-null and etc attributes.

Example :-

Step(1):-

prepare HB mapping file by having ORMapping configuration and instructions

```
<hibernate-mapping>
    <class name="EmpBean" table="Employee">
    <id name="no" column="EID" length="10" type="int">
        <generator class="increment" />
    </id>
    <property name="fname" column="FNAME" length="20"
        not-null="true" unique="true" />
    <property name="lname" column="LNAME" length="25"
        type="string" />
    <property name="mail" column="EMAIL" length="30"
        type="string" />
    </class>
</hibernate-mapping>
```

length → size of the column

type → HB DataType

not-null="true" → Applies NotNull constraint on column.

unique="true" → unique constraint on column.

step(2):- execute SchExport tool from command prompt where HB configuration file and maping file reside.

> java org.hibernate.tool.hbm2ddl.SchemaExport <space>

→ Hibernate DataTypes are the Bridge DataTypes b/w java DataTypes and underlying DB s/w type DataTypes They help the HB s/w to decide the DataTypes in DB s/w while creating columns in DB table based on HB mapping file by using SchemaExport or SchemaUpdate Tool as shown above.

The HB DataTypes are :-

int, long, bloat, double, byte, String .... etc

→ You can also specify java wrapper DataTypes instead of HB DataTypes in mapping file They are

java.lang.Integer
java.lang.String
java.lang.Byte
java.lang.Float
java.sql.Date ..... etc

NOTE:- specifying both then datatypes (hibernate, java wrapper data types) is optional in hibernate mapping file

In this situation the DataTypes of POJO class members Variables will be utilized as Reference Data Types while creating columns in Dynamic table, while working with SchemaExport, SchemaUpdate tools.

7/8/10

SchemaExport, SchemaUpdate Tools are just given to create a new table or to alter existing table structure in underlying DB s/w.

Generally we use these tools to copy Data

purpose take the support of 3rd party Tools available in the internet.

=

⇒ SchemaUpdate tool can create the new table, if table is not already available

SchemaUpdate tool can use existing table or can alter existing table if table is already available, this tool alter existing table only by adding new columns to table., based on new property configured in HibernateMapping file.

example:-

step①:-

> java org.hibernate.tool.hbm2ddl.SchemaUpdate <space>
--config = mycfg.xml

CodeGenerator:-

this tool is supplied in HB 2.x, but not given in 3.x s/w. This tool is capable of generating HB persistance class (HB POJO class) based on Hibernate-Mapping file.

sf = sourceforge

> java net.sf.hibernate.tool.hbm2java.CodeGenerator
--properties = hibernate.properties <space>
Employee.hbm.xml

HB Mapping file ← =

.properties file acting as HB configuration file.

to perform hbm2ddl operations.

⇒ CodeGeneration tool design to perform hbm2ddl operations.

⇒ In Hibernate 2.x the HB configuration file is properties file.

⇒ we can use SchemaUpdate, SchemaExport tool along with HB Application execution by passing Special properties in HB configuration file.

hibernate.hbm2ddl.auto

possible values are.

1. create (uses SchemaUpdate tool internally)
2. update (default) (uses schema update tool internally)
3. create-drop (uses SchemaExport tool internally)

Example①:-

→ In configuration file:-

<property name="hibernate.hb2ddl.auto">create</property>

Always create new table in each execution.

Example②:-

→ configuration file:-

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

can create new table (or) can use existing table (or) can alter existing table by using schema update tool

⇒ (Most Recommended value for hbm2ddl.auto property is update.)

<property name="hibernate.hbm2ddl.auto">create-drop</property>

creates tables dynamically in DB s/w, when SessionFactory object is created & drops all these tables when SessionFactory object is closed from the application.

→ create-drop of hibernate.hbm2ddl.auto property is quite useful in the testing mode of project table creation & destruction is required after testing the project with Trial (dummy) data.

Connection Pooling :-

⇒ Connection Pool is a Factory that contains set of readily available JDBC Connection object.

SessionFactory of HB application represents this JDBC connection pool.

while creating each session object based on SF object one connection object of ConnectionPool will be utilized.

HB s/w can work with ③ types of Connection Pools

they are ① HB s/w supplied built-in ConnectionPool. (default)

② Third party supplied ConnectionPools (like C3P0, proxool)

③ web-server (or) application server manged JDBC Connection pools.

not suitable for production mode environment of project because of its poor performance.

⇒ The default max size of this built-in connection-pool is '20'.

This can be controlled by using following property of HB configuration file.

<property name = "hibernate.connection.pool_size"> 30 </property>

Q) what is the JDBC connection pool, that have used in your project where HB persistance logic is placed ?

⇒ If the project type is stand-alone application, which can run outside the webservers and application servers use 3rd party connection pool sws like C3P0 (or) Proxool.

If project is ServerManaged application like web-application or EJB component use that ServerManaged JDBC connection pool.

NOTE:- Don't use HB supplied built in connection pool any project (performance effected)

=

Our Hibernate Application:-

NOTE:- ① Based on the value passed to hibernate.connection.provider_class property in HB Configuration file. The HB SLW decides.

② the connection SLW that it as used to create JDBC connection pool represented by SessionFactory object.

⇒ This property takes a Built-in class name as value the possible values are.

① org.hibernate.connection.DriverManagerConnectionProvider (default value uses hibernate SLW supplied built-in connection pool.

** ② org.hibernate.connection.DatasourceConnectionProvider (uses WebServer (or) Application Server managed JDBC connection Pool.)

③ org.hibernate.connection.C3POConnectionProvider (uses C3PO SLW managed Connection Pool.)

④ org.hibernate.connection.PoolConnectionProvider. (uses the pool SLW managed JDBC Connection Pool)

→ Step①:- Add C3PO Pool related Connection Provider Classname in HB Configuration file as the application.

→ <property name = "hibernate.connection.provider_class">
org.hibernate.connection.C3POConnectionProvider
</property>

Step③:- Run ...

HB configuration file.

① <property name="hibernate.c3p0.max_size">30</property>

<property name="hibernate.c3p0.min_size">5</property>

<property name="hibernate.c3p0.timeout">5000</property>

<property name="hibernate.c3p0.acquire_increment">2</property>

=

→ Add hibernate-home\lib\c3p0-0.9.1.jar file to classpath.

(This jar file represents C3P0 ConnectionPool Sw).

Step④:- Run the client application in the regular mode.

=

Procedure to work with Proxool JDBC ConnectionPool in our HB Application:-

Step①:- Add Proxool pool related connection Provider class in HB configuration file.

<property name="hibernate.connection.provider_class">
org.hibernate.connection.ProxoolConnectionProvider
</property>

Step②:- Add Proxool Pool related property is in HB configuration file.

<property name="hibernate.proool.xml">mybile.xml</property>

Note:- here we configure the another xml file as Helper xml file for HB Config file.

step③:- develop above specified mybile.xml having properties of Pooxool s/w related connectionPool.

mybile.xml

```
<pooxool-config>
  <pooxool>
    <alias> mypool1 </alias>
    <driver-url> jdbc:oracle:thin:@lh:1521:sid
    </driver-url>
    <driver-class> oracle.jdbc.driver.OracleDriver
    </driver-class>
    <driver-properties>
      <property name="user" value="scott"/>
      <property name="password" value="tiger"/>
    </driver-properties>
    <minimum-connection-count> 10 </minimum-connection-count>
    <maximum-connection-count> 20 </maximum-connection-count>
  </pooxool>
</pooxool-config>
```

Note①:- place this file in the same directory where the HB configuration file resides.

Note②:- since mybile.xml itself is containing url, driverclass, username and password details so there is no need of specifying same details

Step①:- Add hibernate-home\lib\ proxool-0.8.3.jar file to the class path (This jar file represents proxool s/w)

Step②:- execute the client application.

Understanding Server Managed Connection Pools:-

Server Managed Connection Pool will be managed in web-server or Application Server s/w. JDBC DataSource object represents one JDBC ConnectionPool. In order to get access to each Connection object of JDBC connection Pool we need to use JDBC DataSource object.

JDBC DataSource object means it is the object of a class that implements javax.sql.DataSource interface. To provide global visibility to DataSource object it will be registered in registry s/w having nick name.

Every webserver/Application Server s/w supplies one built-in registry s/w.

→ Examples of registry s/ws:-

RMI-Registry :- ① COS (Common Object Service)
② LDAP Registry (Light weight Directory Access Protocol)
③ DNS (Domain Naming Service)
④ weblogic Directory Registry and etc....

with Registry s/w's :

* Java Application can perform insert, update, delete and select operation on DB table by using JDBC API.

* Similarly Java Application can perform Bind, ReBind, unBind, lookup and List operation on Registry s/w by using JNDI API.

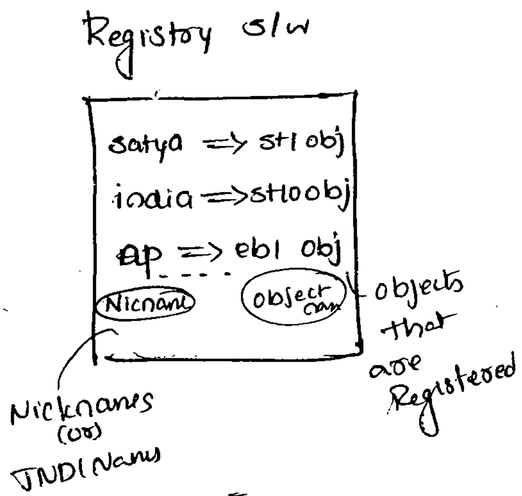
Bind → Adding ~~operatio~~ object to registry (insert)

ReBind → Replacing existing object with new object (update)

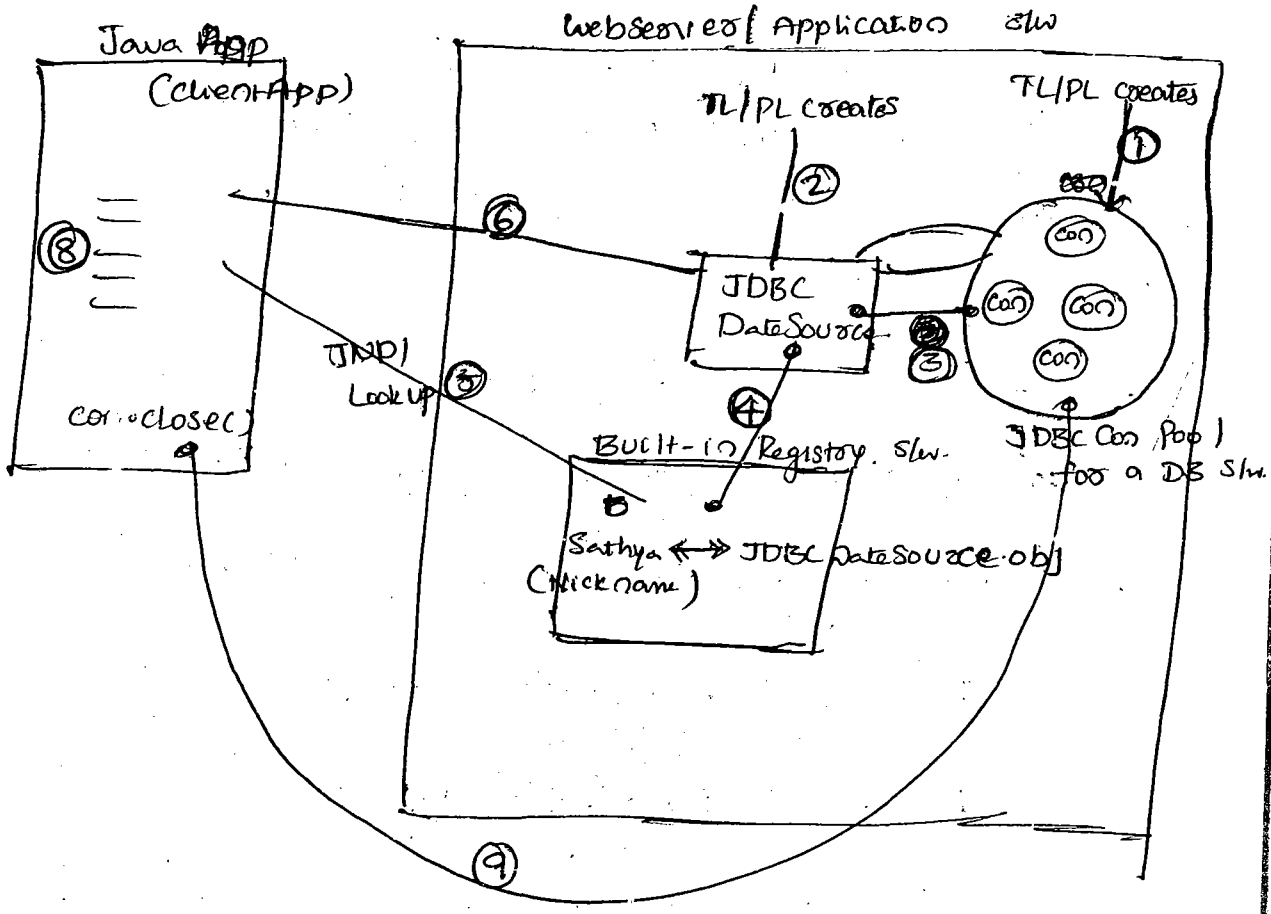
unBind → Removing object from Registry (delete)

List operation → Retrieving all objects that are bound to Registry (select)

Look up operation → Searching and getting object based on nickname/JNDI name (select)



→ To get Access To JDBC Connection objects of connection Pool (Server Managed) programmer must get access to JDBC DataSource object which is representing that connection pool.



① TL/PL creates JDBC ConnectionPool for DB s/w

②, ③, ④ TL/PL creates DataSource object pointing to ConnectionPool and Registers the DataSource object with registry having nickname for global visibility.

⑤ Client app uses JNDI and gets DataSource object from Registry through Lookup operation

⑥, ⑦ Client application uses this DataSource object to get JDBC Connection object from ConnectionPool.

⑧ Client app creates other object based on this connection object and writes the persistable data

object back to connection Pool so this connection becomes free to give service to client application

10/8/10

WEBLOGIC :- (9/10)

→ Application Server = webContainer / servlet Container + EJB Container + Additional services

→ webserver = webcontainer / servlet container + middleware services

→ Application server s/w is enhancement of webserver having enhanced facilities and features.

weblogic A/S :-

Type :- Application server s/w

Version :- 9.x (Compatable with J2SDK 1.5)

Vendor :- BEA systems (oracle s/w).

portno :- 7001 (default)

Commercial s/w :-

to down load s/w :- www.commerce.bea.com

for document :- www.edocs.bea.com

allows to create domains.

default domain name is :- examplesserver.

if multiple projects of a companies are using same weblogic s/w, then weblogic s/w will be installed only once on a common computer having multiple logical domains for multiple projects on one per project basis. Each logical domain acts as one Application

Oracle, JDBC DataSource in Example Servers Domain of weblogic 9.x :-

Step(1):- Start example servers domain of web-logic.

Start
↳ programmes
↳ BEA products
↳ Examples
↳ weblogic Servers
↓
Start Examples Servers.

Step(2):- open administration console of example servers domain.

Open Browser Window
↓
type "http://lb:7001/console" url
usr:- weblogic
pword:- weblogic
↓
login.

Step(3):- create JDBC DataSource from Administration console porting to JDBC Connection pool for Oracle

Admin Console
↳ lock & Edit
↳ services
↳ JDBC
↳ DataSources
↳ New
↳ New

Name:- myoracds

logical name

DBType :- oracle

DB Drivers :- oracleThinDrivers

↳ Next

↳ Next

DB Name :- satya <sid> select * from global_name

host name :- lh.

port no :- 1521

User :- SCOTT

pwd :- tiger

confirm pwd :- tiger

↳ Next

↳ Next

↳ ⊗

☑ ExampleServers.

↓

finish

↓

Activate changes

Configure parameters of JDBC ConnectionPool :-

services

↳ JDBC

↳ DataSources

↳ myoracle

↳ Connection Pool

↳ Lock & Edit

Capacity increment :- ____

<specify the no. of JDBC connection object, it should be created in connectionpool at a time if there is a need of creating new JDBC connection object.

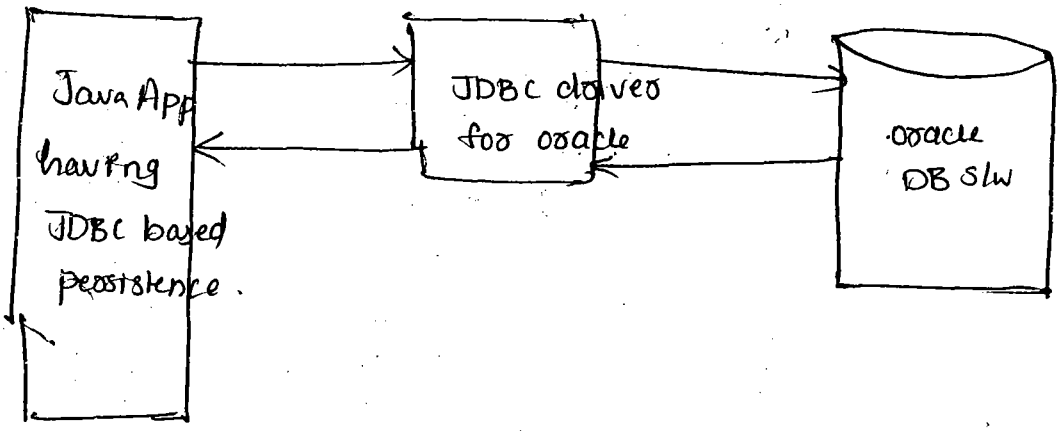
↓

Test

Activate changes

NOTE:- The Moment you create JDBC DataSource. the JDBC DataSource object will be automatically registered with the weblogic supplied built-in web-logic Directory Registry s/w having Nickname SatyaJNDI.

→ Communication b/w Java App and DB:-



JDBC Connection object in Java App

~~Communication b/w Java App and Registry s/w:-~~

represents connectivity with DB s/w. To create this connection object the following Details are required ① JDBC Driver class name

② DB url

③ DB username

④ DB password.

JDBC Driver is the Bridge b/w Java Applicat. and DB s/w, every JDBC Driver is identified with its Driver class name.

and Registry s/w

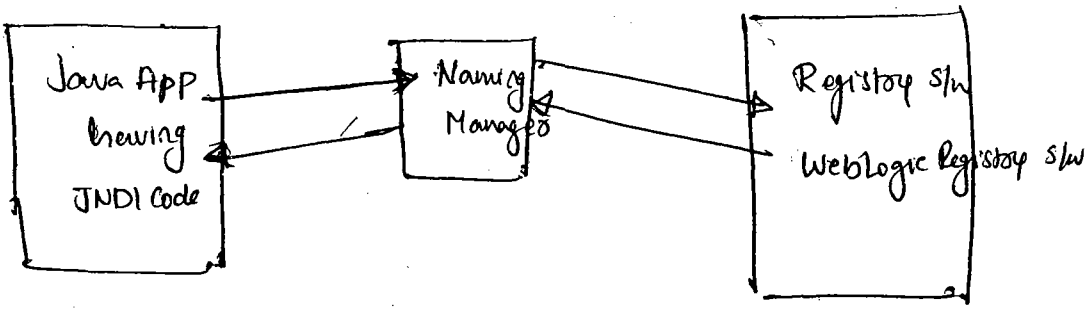
Every Naming Manager will be identified with its InitialContextFactory classname. This classname change based on Naming Manager and Registry s/w we used:

InitialContext object of java application represents connection with Registry s/w and provides environment to perform operations on Registry s/w.

To create this object we need following two details which are called JNDI properties:

① InitialContextFactory class name of NamingManager

② Provider url of Registry s/w.



→ JNDI properties values of weblogic Directory Registry s/w

→ InitialContextFactory class name :-

weblogic.jndi.WLInitialContextFactory

This class is available in

BEA-HOME\weblogic81\server\lib\weblogic.jar file

→ provider url :- t3://<hostname>:<portno>

Connection Pool in stand alone Environment based Hibernate Application :-

Step(1) :- configure connection provider class, HB configuration file.

<property name="hibernate.connection.provider_class"> org.hibernate.connection.DataSourceConnectionProvider </property>

(2) :- Specify JNDI property values, related to weblogic directory Registry in HB configuration file.

<property name="hibernate.jndi.url"> t3://eh:7001 </property> (provider url)

<property name="hibernate.jndi.class"> weblogic.jndi.WLInitialContextFactory </property>

(WLInitialContextFactory → InitialContextFactory class)

(3) :- Specify JNDI Name (or) Nickname of JDBC DataSource created in weblogic server

<property name="hibernate.connection.datasource"> SathyaJndi </property>

Note :- since JDBC Driverclassname, DBurl, DB username, DB password values are specified in server admin console while creating JDBC Connection Pool so there is no need of specifying them in HB configuration file.

bile to the class path.

→ Make sure that example servers Domain of weblogic weblogic.9.x in Running Mode and execute the client Application

---

GlassFish :-

type :- Application Server s/w

Version :- 2.0x (compatable with j2sdk1.5)

netbeans 6.x s/w use glassfish as built-in server.

Vendor :- sun microsystems (oracle corp)

open source s/w

default port no :- 4849 for Admin Console.
8081 for running w/a.

For downloadocs :- www.java.sun.com/javaee/5/docs.

jar file that represents J2EE API :- javaee.jar

registry s/w name : Glassfish Registry

11/08/10

Procedure to create jdbc datasource and jdbc connection pool for oracle in GlassFish application server :-

step① :- start GlassFish server

start → pgms → sunmicrosystems → Application Server

Start default server ←

step② :- open admin console of GlassFish server

start → pgms → sunmicrosystems → Application server → admin console

username :- admin

password :- admin

step ⑨ :- Create jdbc connection,

~~Ambad~~
admin console
↳ Resources
↳ Jdbc
↳ connection pool
↳ new
↳

name : mypool1

Resource type : javax.sql.DataSource

DB Vendor : oracle
↳ next

→ Initial Capacity : — 10.

Maximum Pool size : 20.

pool resize : 2

☑ ~~DB name write~~ "satya"

☑ Data S

✓ DataSourceName : myds.

e ~~pass~~.

✓ DataBase Name : ~~XE pass satya~~ satya.

Password : tiger

port name : 1521

Server name : lh

url : jdbc:oracle:thin@:lh:1521:satya

User : scott.

[Add property]

Driver Class : oracle.jdbc.driver.OracleDriver.

↳ [finish]

↳ mypool1

↳ [ping]

☑ ping succeeded

↳ [save]

GlassFish\AppServer\lib is contain ojdbc14.jar

step④
=> Create JDBC DataSource pointing to the above created connectionpool

admin_console
↳ Resources
↳ JDBC
↳ JDBC Resources
↳ New

JNDI Name: GSDS Jndi <any-name>

↓
OK

=> Procedure to use GlassFish Server managed JDBC Connection pool in standalone environment based HB Application.

step①:- same as weblogic server.

step②:- Specify JDBCDataSource JNDI name, in HB configuration file.

<property name="hibernate.connection.datasource">GSDsJndi</property>

=> NOTE:- there is no need of adding InitialContextFactory classname, provider url value in HB configuration file, while working with GlassFish Server supplied Registry s/w.

=> Add Following two jar files to the classpath.

~~AppServer~~:

① GlassFish Home\AppServer\appserv-ot.jar
\javaee.jar

=> execute the client Application.

WEB-RESOURCE PROGRAMMES :-

→ For this add HB API related main jar file (hibernate3.jar) in the classpath and add HB API related main and dependent jar files in WEB-INF/lib folder of web-application.

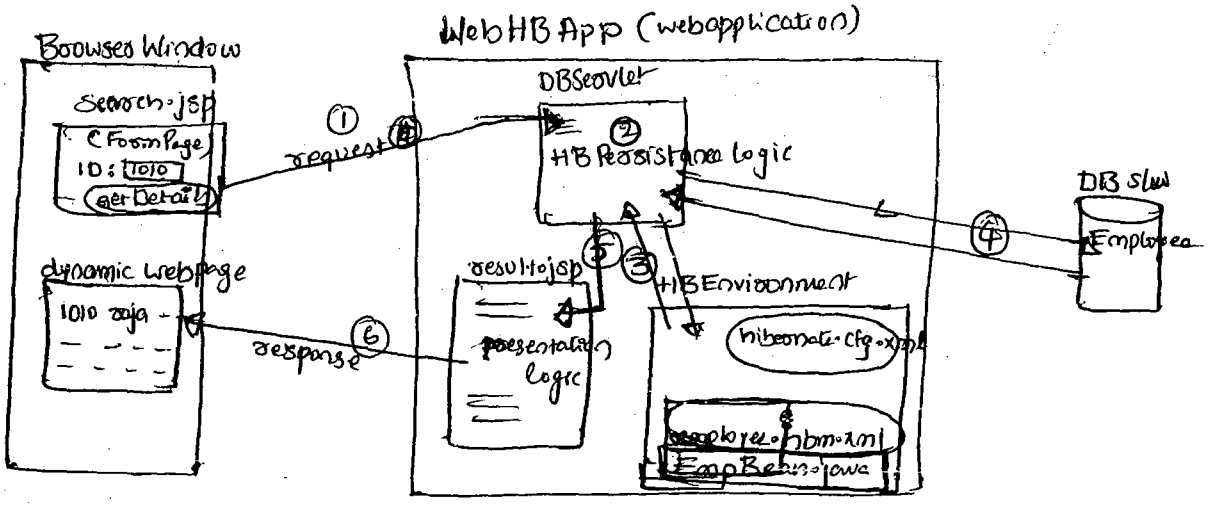
→ While developing HB persistance logic in java applications it is recommended to create HB Session object in one time execution block and it is recommended to use this HB Session object for persistance operations in repeatedly executing blocks.

→

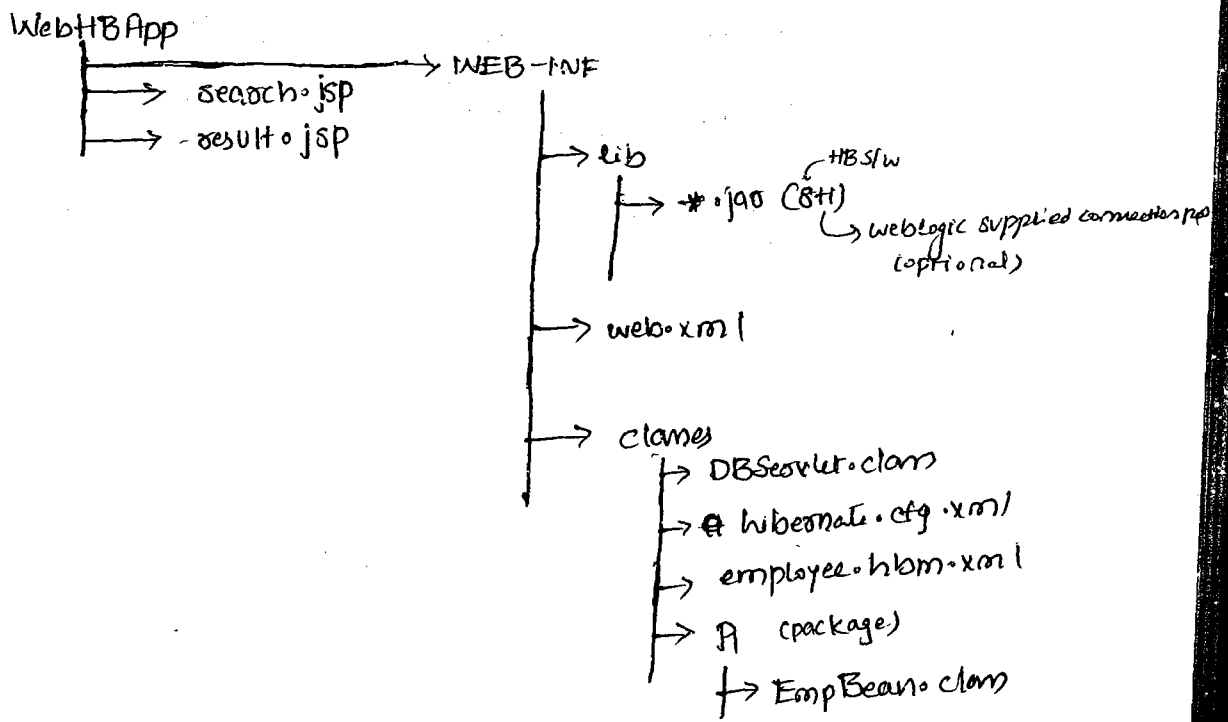
	Application Type	Place to create HB Session obj	place to use HB Session obj
1)	AWT / Swing Frame	static block / Constructor	Event Handling methods
2)	Applet / JApplet	static block / Constructor / init()	Event Handing methods
3)	Servlet	static block / Constructor / init()	service(-,-)
4)	JSP	<%! public void jspInit() { } %>	<% script-let Code %>

5) EJB Session Bean Component — ejbCreate() / constructor / static block — Business Method

6) Struts App — static block / constructor — execute()



Deployment Directory structure:-



→ In the above application DBServlet is using HBPersistence logic to interact with DB s/w.

→ In order to pass data from servlet to JSP when servlet is forwarding request to JSP the servlet can use request attributes.

that class WEB-INF / classes folder then the class must be there in package.

→ The above application is going to be deployed in weblogic 9.x server and it is going to used weblogic9.x server managed. JDBC connection Pool.

→ If any application that is running outside the weblogic server wants to interact with weblogic directory Registry SW then the application must use JNDI Properties like InitialContextFactory class name, provider url and etc..

→ If Application deployed in weblogic server wants to interact with weblogic directory Registry then there is no need to parsing the above said JNDI properties. According to this discussion there is no necessity of parsing JNDI property values in HB configuration file of above application.

Source Code:-

hibernate.cfg.xml :-

```
-DTD-
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">org.hibernate.
      dialect.Oracle9Dialect</property>
    <property name="hibernate.connection.provider_class">
      org.hibernate.
    <property name="hibernate.Connection.datasource">Sathya Jndi
```

<mapping resource = "Employee.hbm.xml" />

</session-factory>

</hibernate-configuration>

→ Do observe that the configuration file not having hibernate.jndi.class, hibernate.jndi.url properties holding JNDI properties. (InitialContextFactory class name, providerurl)

=

12/08/10

EmpBean.java :-

Same as first application but make sure that it is available in a package "p1"

webAPP\WEB-INF\classes > javac -d . EmpBean.java

Employee.hbm.xml :-

same as 1st appln

Search.jsp :-

<form action = "dburl" method = "get">

 Employee ID

<input type = "text" name = "tid" />

<input type = "submit" value = "Get Details" />

</form>

=

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

import org.hibernate.cfg.*;
import org.hibernate.*;

public class DBServlet extends HttpServlet
{
  Session s,s = null;
  public void init()
  {
   s.o.p("init() of DBServlet");
    try
    {
      ses = new Configuration().configure().
            buildSessionFactory().openSession();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
  } //init
  public void doGet(HttpServletRequest req, HttpServletResponse
                    res) throws SE,IOE
    {
      try
      { //read form data
      int no = Integer.parseInt(req.getParameter("tid"));
      // write HB persistance logic.
EmpBean eb = (EmpBean) ses.get(EmpBean.class,
                               new Integer(no));
// store result in request attribute to send to
   o result.jsp
req.setAttribute("result", eb);
```

```
//forward req to result.jsp.
RequestDispatcher rd = req.getRequestDispatcher("result.jsp");
if (rd != null)
    rd.forward(req, res);
}//try.
catch (HibernateException he)
{
    he.printStackTrace();
}
catch (ServletException se)
{
    se.printStackTrace();
}
} //doGet
public void doPost(HttpServletReq. req, HttpServletResponse res)
{ s.o.p("doPost() at DBServlet");
  doGet(req, res);
}
catch(Exception e)
{
  e.printStackTrace();
} //do
} // doPost
public void destroy()
{
  s.o.p(" destroy() at DBServlet");
  try
  {
    ses.close();
  }
  catch (Exception e){
  {
    e.printStackTrace();
  }
} //destroy
} //class
```

web.xml :-

Configure DBServlet by have /DBUrl as urlpattern

=

result.jsp :-

```
<%@page import="pl.EmpBean" %>

<% EmpBean eb = (EmpBean) request.getAttribute("result"); %>

<b> The Details are </b>

<%= eb.getNo() %><br>

<%= eb.getFname() %> <br>

<%= eb.getLname() %> <br>

<%= eb.getMail() %> <br>
```

=

⇒ In WEB-INF/lib folder of above web application place the HIBAPI related 4+ jar files.

=

Procedure to Deploy web-app in weblogic 9.x server :-

Step① :- prepare war file on the Deployment Directory structure of web-application.

> jar cf webHBApp.war<space>. ↵

above step gives a webHBApp.war file representing jar file

→ There are ③ ways to deploy applications in servers

① HotDeployment

② console Deployment

③ Tool Base Deployment

NOTE :- this application is using console Deployment process.

→ open administation console of example server (domain) weblogic 9.2:-

→ Deploy the war file <webHBApp.war>

Admin Console
- ↳ lock & Edit
  - ↳ Deployments
    - ↳ Install
      - ↳ upload your files
        - ↳ Browse and select webHBApp.war
        - ↓ Next
        - Select webHBApp.war
        - Next
        - Next
        - Next
        - Finish
- Activate Changes
  - ↳ Deployments
    - ↳ select webHBApp webApplication
      - ↳ start

Test the webApplication

open Browser window:- http://lh:7001/WebHBApp/search.jsp

=

BULK OPERATION RELATED TECHNIQUES:-

In order to manupulate single row, by taking our choice value as criteria value or to manupulate more than one row at a time we can use one of the following technique.

But HQL Most Recomended Technique:

1) HQL
2) NativeSQL
3) Criteria API

HQL:-

it is the most popular persistance technique environment in Hibernate Programing

① HQL's are DB independent queries, so these queries based independent logic is DB Independent.

② HQL queries will be written based on POJO classes and member variables of POJO classes.

③ HQL Queries are object level Queries, so they return HB POJO class objects as Results.

④ HQL Queries and keywords are very much similar to SQL Queries of oracle.

⑤ HB S/W converts HQL Queries into SQL queries and sends them to DB S/W for exicution.

HQL Queries supports operator, expression, condition clauses, joins, sub queries, aggrigate functions and etc

→ we can use HQL Queries for both select & non select operations.

→ Query Object represents HQL Query in Hibernate application, this is the object of a class that implements org.hibernate.Query Interface

→ all Non-select HQL Queries must be executed in transaction management environment.

→ HQL Query support two types of parameters ① positional parameters ② named parameters.

Limitations:-

① HQL Query cannot perform DDL operations

② HQL Queries cannot be used in PL/SQL programming

③ HQL Queries cannot be used to insert single record into table.

④ HQL Queries use negligable performance degradation because of conversions when compare to SQL.

13/08/10

SQL> select * from Employee ← table name

HQL> select eb from EmpBean as eb (or)

HQL> from EmpBean as eb (or) ← POJO class name

HQL> from EmpBean (or)

HQL> from EmpBean eb

NOTE:- When select keywood is there or POJO class member variables are there in HQL Query then creating alias name for POJO class is mandatory ~~thing~~ operation.

SQL> select EID, LASTNAME from employee where EID >= 100;

(EID, LASTNAME → column names; employee → table name; EID → column name)

HQL> select eb.no, eb.lname from EmpBean as eb where eb.no >= 100;

(EmpBean → POJO class; eb.no, eb.lname → POJO class member variable)

⇒ HQL keywords are not casesensitive but POJO class name and POJO class member variable used in HQL Query are casesensitive.

NOTE⇒ String values in HQL Query should be represented using single code.

ex:- 'raja', 'hyd'

=

SQL> select count(*) from Employee;

(Employee → tablename)

HQL> select count(*) from EmpBean

(EmpBean → POJO class)

=

SQL> delete from Employee where firstname in ('raja', 'ravi');

HQL> delete from EmpBean as eb where eb.fname in ('raja', 'ravi');

→ To execute select HQL Queries use list() iterate() or Query obj similary to execute non-select HQL Queries call executeUpdate() on Query objects.

=

→ execution of HQL Query is nothing but converting HQL Query to the underlying DB specific SQL Query and sending that SQL Query to DB S/w for execution.

→ Hibernate 3.x S/w internally uses, AST Query Translator to convert HQL Queries into DB S/w equivalent SQL Queries.

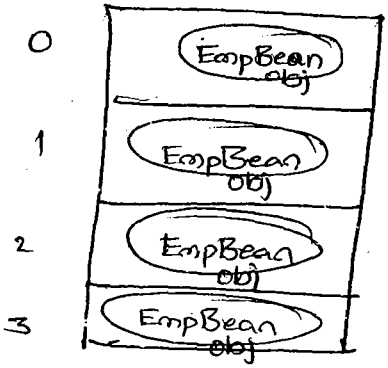
=>

```
// imports
public class HQLTest
{
  P S V main(String[] args) throws Exception.
  {
    // create HB Session object
SessionFactory factory = new Configuration().configure().buildSF();
    Session ses = factory.openSession();
Query q1 = ses.createQuery("select eb from EmpBean as eb");
List l = q1.list(); // execute HQL.
    // display records
for(int i=0; i<l.size(); ++i).
  {
    EmpBean eb = (EmpBean) l.get(i);
```

```
s.o.p (e1.getNo()+" "+e1.getFname()+" "+e1.getLname()
                    +" "+e1.getMail());
}
}
```

Object Memory representation:-

l (java.util.List obj)

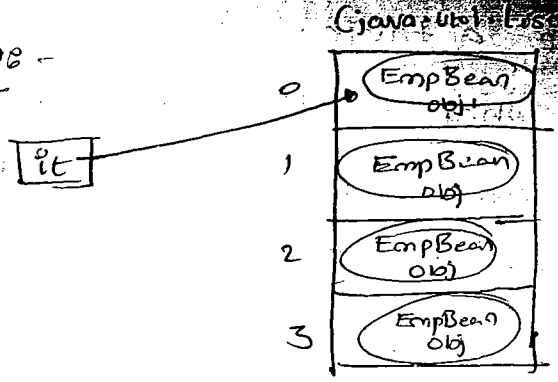


⇒ Executing ® HQL select Query by using Iterate method.

Ex:-

```
Query q1 = ses.createQuery("select eb from EmpBean
                            as eb");
Iterator it = q1.iterate(); //executes HQL
while(it.hasNext())
{
EmpBean e1=(EmpBean) it.next();
s.o.p (e1.getNo()+" "+e1.getFname()+" "+
        e1.getLname()+" "+e1.getEmail());
}
}
```

Memory diagram -



*** Q) what is the diff b/w executing Select HQL Query by using list() & by using iterate() ?

=> list() method generates result by selecting all records through the execution of single select SQL Query - (No Lazy Loading) that means if results objects are used (or) not used by the programmes (i by calling ~~getXxx~~ getXxx()), the POJO class object representing results will be initialized with data (records)

→ iterate() method, selects the records from DB table by executing multiple SQL select Queries (almost one select Query for record) and also generates first SQL select Query only to retrieve column names. iterate() performs lazy loading because it creates results related ~~Hibernate~~ HB and initializes results related HB pojo class object on demand.

* list() method is recomended to use

→ we can see the above diff b/w list() and iterate()

of a table.

Q) ⇒ what is the diffrence b/w executing select Queries in JDBC & HQL select Query in HB ?

Ans:- JDBC code based select Query execution gives result set object, which is not serializable object so we cannot send resultset object over the network.

Hibernate based HQL Query execution gives result in the form of collection Framework List DataStructure. since List DataStructure object is serializable object by default, we can send the object over the network.

① NOTE:- all the collection FW DS are serializable object by default.

② Note:- In order to get above said benefit in HB the programmer should make HB POJO class object implementing java.io.serializable interface

=

HQL Query with conditions -

⇒

⇒

```
Query q1 = ses.createQuery("select eb from EmpBean
                as eb where eb.no >=100 and eb
                eb.ename like 'A%'");
List l = q1.list();
for(int i=0;i<l.size();++i)
{
```

```
EmpBean e1 = (EmpBean) l.get(i);
S.O.P (e1.getNo() + " " + e1.getFname() + " " +
       e1.getLname() + " " + e1.getEmail());
}
}
}
```

14/08/10

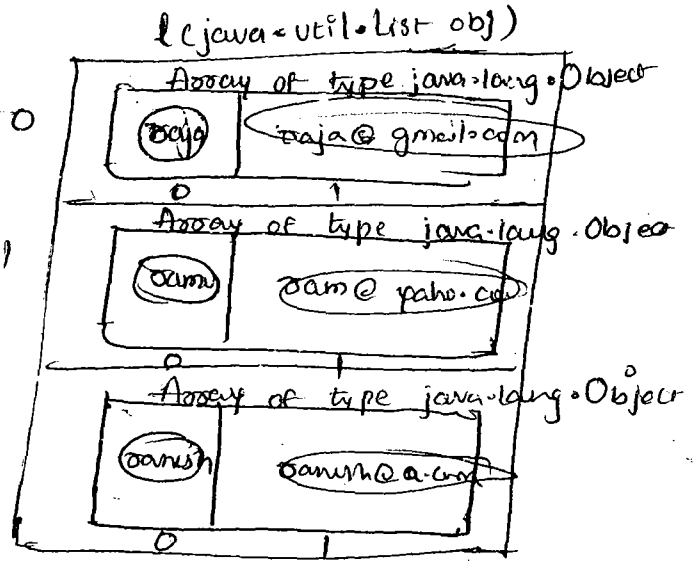
Comment :- Arrays are objects in java, so arrays can be added as element values of collection Framework DS

→ when HQL select Queries are selecting specific column values of a table then the result related List DS contains java.lang.Object class Object arrays as element values.

ex:-

```
Query q1 = ses.createQuery("select eb.fname, eb.mail
           from EmpBean as eb where eb.no>=100");
List l = q1.list();
//display results
for(int i=0; i<l.size(); ++i)
{
  Object row[] = (Object[]) l.get(i);
  for(int k=0; k<row.length; ++k) {
    S.O.P (row[k].toString() + " ");
  } //inner for loop
  S.O.P ();
} //out for loop
```

Memory Diagram :-



→ when Array type is Student class the elements in the Arrays are Student class objects

when Array type is java.lang.Object class the elements in that are java.lang.Object class objects

Example :- existing SQL select Query that select specific column values by using 'iterate()'

```
Query q1 = ses.createQuery("select eb.no, eb.mail
                from EmpBean as eb where (eb.no>=100)
                or (eb.mail like '%.com') or
                (eb.mail like '%gmail.com')");

Iterator it = q1.iterate();  // execute HQL Query

while(it.hasNext())
{
 Object obj[] = (Object[]) it.next();
  for(int k=0; k<obj.length; ++k)
  {
   s.o.p (obj[k].toString()+" ");
  }
```

⇒ When "iterate()" is used to select specific column values of HQL select Query then no lazy loading will take place.

=

example:-

Executing single Aggrigate function based HQL Select Query

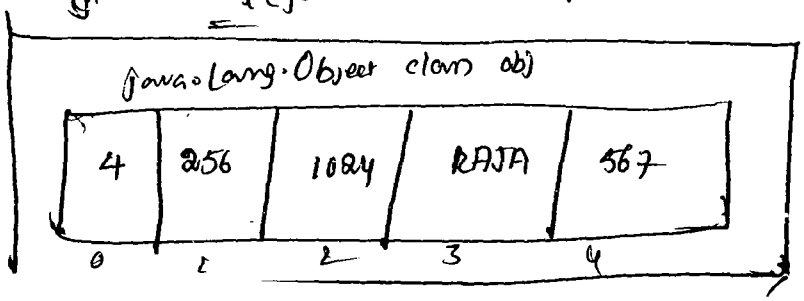
```
Query q1 = ses.createQuery("select count(*) from EmpBean eb");
List l = q1.list();
S.O.P("count of records in table are" + l.toString());
```

=

Executing HQL Query having multiple Aggrigate functions and Utility functions :-

```
Query q1 = ses.createQuery("Select count(*), avg(eb.no), upper('raja'),
                            sum(eb.no), max(eb.no) from EmpBean eb");
List l = q1.list();
s.o.p("Results are" + l.toString());
Object res[] = (Object[]) l.get(0);
s.o.p("count is" + res[0].toString());
s.o.p("avg is" + res[1].toString());
s.o.p("upper is" + res[2].toString());
s.o.p("sum is" + res[3].toString());
s.o.p("max is" + res[4].toString());
```

l (java.util.List obj)



Example on SUBQueries in HQL environments—

```
Query q1 = scs.createQuery("select eb from EmpBean as eb
                            where eb.no = (select max(eb1.no) from
                                           EmpBean as eb1)");
List l = q1.list();
for(int i=0; i<l.size(); ++i)
{
EmpBean e1=(EmpBean) l.get(i);
S.o.p(e1.getNo()+" "+e1.getFname()+" "+e1.getLname()+
                                    " "+e1.getMail());
}
```

=

⇒ To make HQL Query flexible by setting input values at Query from outside Query and to set Query i/p values as java side values without worrying about underlying DB sql syntax we can pass parameters to HQL Queries.

Parameters passed to HQL Query never makes HQL Query as precompiled Query. because HQL Query cannot directly go to DB sql.

16/08/10

→ In most of the situations, the HB s/w generated JDBC code internally uses precompiled SQL Queries with the support of prepared statement object to perform persistance operations on the table.

→ All HQL Queries related SQL Queries generated by HB s/w are precompiled Queries by default.

Parameters in HQL

(1) positional parameters (?)

(2) Named parameters (:<Name>) ⟶ Recomended to use

Example Code on positional parameters:-

```
Query q1 = ses.createQuery("select eb from EmpBean as
                 eb where eb.no >= ? and eb.mail like ?");

    // setting parameter values.
    q1.setInteger(0, 30);        → position starts from '0'
                                    in HQL
    q1.setString(1, "%gmail.com");
                 ↓          ↓
            parameter    parameter value.
              index
    List l = q1.list();
   for (int i=0; i<l.size(); ++i)
   {
     EmpBean e1 = (EmpBean) l.get(i);
     s.o.p(e1.getNo() + " " .......);
   }
```

⇒ we cann't pass HQL keywords, POJO classnames, POJO class number variable names as values of parameters.

HQL Queries, it is against of HQL Syntax.

Wrong HQL Query (or) invalid HQL Query.

Query q1 = ses.createQuery("select eb from (?) as eb where eb.no >= ? and eb.mail like ?");

unexpected token.

q1.setString(0, "EmpBean");

q1.setInteger(1, 30);

q1.setString(2, "x.gmail.com");

⇒ use parameters in HQL Query only to pass input values and condition values. [ie not table names]

Named Parameters:-

Example code on Named Parameters:-

Query q1 = ses.createQuery("select eb from EmpBean as eb where eb.fname in(:p1, :p2) and eb.mail like :p3");

// setting values for parameters

q1.setString("p1", "raja");

q1.setString("p2", "ravi");

q1.setString("p3", "x.y.com");

List l = q1.list();

for(--)

{

=

}

=

*** ⇒ in JDBC programming, we can't pass Named parameters in SQLQuery. But in HQL Query both Named, positional parameters in a single Query.

But we must pass positional parameters before Named parameters.

Valid:

Query q1 = ses.createQuery("select eb from EmpBean as eb where eb.fname in(?,?) and eb.mail like :p3");

q1.setString(0, "raja");

q1.setString(1, "ravi");

q1.setString("p3", "x.y.com");

Invalid HQL Query :-

Invalid

```
Query q1 = ses.createQuery("select eb from EmpBean as eb
        where eb.fname in (:P1, :P2) and eb.email like?");
q1.setString("P1", "raja");
q1.setString("P2", "xxx");
q1.setString(0, "y.s.com");
```

Executing Non-Select SQL Query (insert/update/delete) :-

→ must be executed as transactional statement

→ HQL insert Query is not given to insert only one record at a time, but to insert multiple records by selecting them another table (To solve this problem use ses.save())

Example:- Executing HQL delete Query

```
Transaction txn = ses.beginTransaction();
Query q1 = ses.createQuery("delete from EmpBean as eb
        where eb.no=(select min(eb.no) from EmpBean as eb)");
int res = q1.executeUpdate();
s.o.p("no of record effected :" + res);
txn.commit();
```

HQL Non-select Query with Parameters :-

```
Transaction txn = ses.beginTransaction();
Query q1 = ses.createQuery("update EmpBean as eb set
        eb.email=? where eb.no >= :P1");
q1.setString(0, "x@gamil.com");
q1.setInteger("P1", 1000);
int res = q1.executeUpdate();
s.o.p("no of rec updated :"+res);
txn.commit();
```

=> HQL Insert Query is not given in this form "insert into EmpBean as eb values (?,?,?,?)" ⟹ Wrong Query

(or)

insert into EmpBean eb values (:P1, :P2, :P3, :P4)" => Wrong

=> The supported Form of HQL insert Query is

** INSERT INTO <table1> SELECT <table2> FROM :-

ie using this syntax we can ~~insert~~

=> we cannot insert directly records to table, we can insert record only by selecting from another table.

Example:-

Source table <Employee>

same as first application.

destination table <Employee1>

EID → number pk
name → varchar2(20)
mail → varchar2(20).

=

EmpBean.java:-

same as First appln.

EmpBean1.java:-

int no, String name, String email

-> we can configure multiple HB pojo class in single mapping file by using <class> tag multiple times.

Employee.hbm.xml:-

```
<hibernate-mapping>
  <class name = "EmpBean" table = "Employee" >
    <id name = "no" column = "EID" />
```

=

```
<class name="EmpBean1" table="Employee1">
    <id name="eid" column="EID" />
    <property name="name" column="NAME" />
    <property name="mail" column="MAIL" />
</class>
</hibernate-mapping>
```

=

Code in client Applng - 'HQLTest.java'

→ To select records from one table and to insert them in another table.

```
Transaction txn = ses.beginTransaction();
Query q1 = ses.createQuery("insert into EmpBean1(no,name,
        mail) select eb.no,eb.name,eb.mail from
        EmpBean as eb where eb.no >= :P1");
q1.setInteger("P1",300);
int res = q1.executeUpdate();
S.O.P("no of records effected :" +res);
=
}
```

For related information on HQL, for more example on HQL refer chapter 13 & 14 of pdf file.

17/8/10

Pagenation -

when query selects huge no of records instead of displaying them in a single screen (or) single page, it is recommended to display them in multiple screens (or) in multiple pages through 'pagenation' concept

→ hibernate persistance logic provides Environment ie required for pagenation.

→ All the 3 bulk operations techniques supports pagenation.

→ Even though select query selecting huge amounts of records in order to select specific range of records from those records use setFirst_Result(), setMaxResult() methods are shown below.

→ we can also use this method for 'pagenation'

Ex:-

```
Query q1 = ses.createQuery("select eb from EmpBean
                            as eb");
// select 3 records from 399 records (0 based index)
q1.setFirstResult(2);
q1.setMaxResults(3);
List l = q1.list();
for(int i=0; i<l.size(); ++i)
{
EmpBean e1 = (EmpBean) l.get(i);
S.O.P(e1.getNo() + " " + e1.getName() + " " + e1.getAmt() +
                                    " " + e1.getMail());
}
```

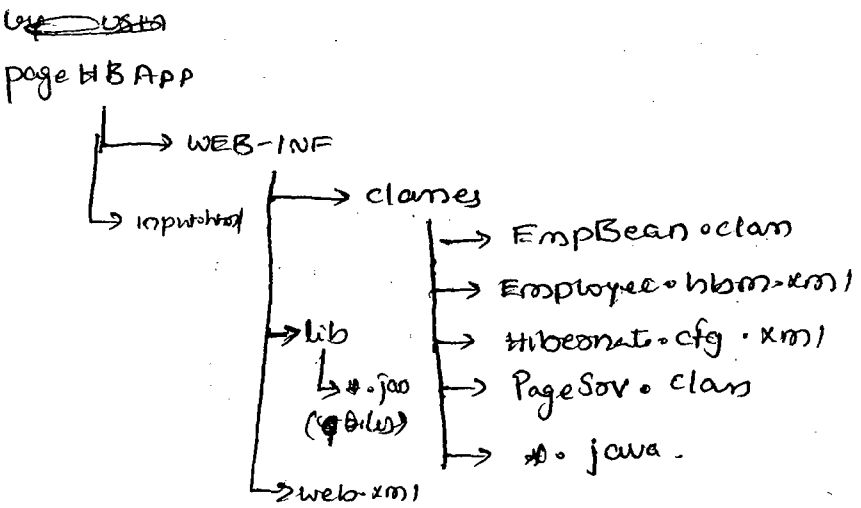
=> Hibernate internally uses the pseudo-column rownum while generating SQL Query for Oracle DB sw based on above code.

-> 'rownum' is a invisible pseudo column in DB tables of Oracle sw holding rownos as column values.

=

-> To perform pagenation by using HB persistence logic in web-applications, take fixed no for maxResult and change FirstResult value based on the request no (or) page no., To make these operations specific to each client use session attribute to manage first result value of each page.

Deployment Directory structure of web-appn that performs pagenation by using HB persistance logic :-



=

jar files in WEB-INF\lib folder

(8) -> lib jar files

(1) -> ojdbc jar files

jar files in classpath :- -> SERVLET-API.jar

-> hibernate3.jar

EmpBean.java :-

<-- same as 1st appln -->

Employee.hbm.xml :-

<-- same as 1st appln -->

hibernate.cfg.xml :-

<-- same as 1st appln -->

input.html :-

```
<center> <b> <a href="pageurl"> Generate Report </a> </b> </center>
```

↓

url pattern of PageSer servlet

web.xml :-

configure PageSer servlet by having /pageurl as url pattern

PageSer.java :-

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import org.hibernate.*;
import org.hibernate.cfg.*;
import java.util.*;
public class PageSer extends HttpServlet
{
   Session ses = null;
   int pnitval;
   int totrecords;
   public void init()
   {
      try
      {
         ses = new Configuration().configure().buildSessionFactory().openSession();
```

```
catch (Exception e)
 {
  e.printStackTrace();
 }
} //init
public void service(HttpServletRequest req, HttpServletResponse res)
                                        throws SE, IOE
{
  // logic to get count of records in db table for every request
Query q2 = ses.createQuery("select eb from EmpBean as e");
   List l2 = q2.list();
    totalrecords = l2.size();
 PrintWriter pw = res.getWriter();
   res.setContentType("text/html");
 //create / locates session for client
 HttpSession se = req.getSession();
 if (session.getAttribute("counter")== null)
  {
    // logic of first request
     intval=0;
    ses.setAttribute("counter", new Integer(initVal));
  }
  else
  {
    // logic for other than 1st record.
  Integer i1 = (Integer) session.getAttribute("counter");
    initVal = i1.intValue()+2;
    session.setAttribute("counter", new Integer(initVal));
  }
 // logic to display records in form of html table
by applying pagination.
Query q1 = ses.createQuery("select eb from EmpBean as eb");
```

```
q1.setFirstResult(initval);
q1.setMaxResults(2);
List l = q1.list();
pw.println("<table border='1'><tr><th>No</th>
        <th>FIRSTNAME</th><th>LASTNAME</th>
        <th>EMAIL</th></tr>");
for(int i=0;i<l.size();++i)
{
EmpBean e=(EmpBean)l.get(i);
pw.println("<tr><td>"+e.getNo()+"</td>
        <td>"+e.getFname()+"</td>
        <td>"+e.getLname()+"</td>
        <td>"+e.getMail()+"</td></tr>");
}
pw.println("</table></body>");
//logic to display home and next hyperlinks in the page.
if((initval+2)<totalrecords)
{
pw.println("<a href='pageurl'>next</a>");
}
else
{
pw.println("<a href='inputw.html'>home</a>");
session.invalidate();
}
pw.close();
} //service.
public void destroy()
{
try
{ ses.close();
}
catch(Exception e)
```

} // destroy

} // class

18-08-10

Native SQL :-

-> NativeSQL queries are underlying DB spec specific SQL queries. These queries based persistence logic of HB is DB dependent persistance logic. If programmer feels certain operation is complex (or) not possible with HQL Then it is recommended to use "NativeSQL" queries.

-> Native SQL Query will be written DB Table names and column names.

-> These are ② types of Native SQL Query.

① Entity Queries :-

these Queries return all the column values of a table.

② Scalar Queries :-

these Queries return specific column values or non-column values as a results.

Native SQL Query results must be mapped with Hibernate POJO classes or HB Data types.

Native SQL Queries allow both named, positional parameters.

-> SQLQuery object represents one NativeSQL Query. this is the object of a class, that implements org.hibernate.SQLQuery interface.

This interface is sub interface of org.hibernate.Query interface

-> NativeSQL Query programming environment is very useful for the programmer to call PL/SQL procedures and functions

ab DB s/w from HB ~~programming~~ ~~language~~ logic.

⇒ Example on Entity Native SQL Query (or) select Query that select all column values :-

~~TestClient~~

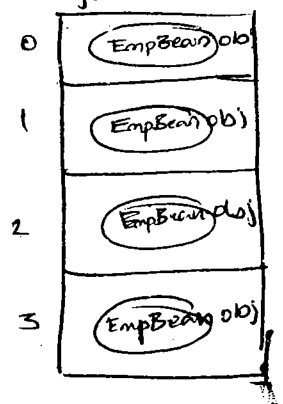
SQLTestClient.java :-

```
String sql = "select {e.*} from Employee e";
```

(tablename → Employee; aliyas name → e)

```
SQLQuery q1 = ses.createSQLQuery(sql);
  //map entity query result with HB POJO class
    q1.addEntity("e", EmpBean.class);
  // execute NativeSQL query
    List l = q1.list();
     // display result.
  for(int i=0; i<l.size(); i++)
   {
   EmpBean e1 = (EmpBean)l.get(i);
    s.o.p(e1.getNo() + " " + e1.getFname() + " "
                  e1.getLname() + " " + e1.getMail());
   }
  }
```

java.util.List obj



⇒ we can't call iterate() on SQL Query object to execute entity Query (or) select Query of NativeSQL. this indicates lazy loading (or) lazy initialization of HB pojo class objects is not possible with NativeSQL Query

Another form of above example:-

⇒

```
String sql = "select * from Employee";
SQLQuery q1 = ses.createSQLQuery(sql);
//map results with HB pojo class
q1.addEntity(EmpBean.class);
//execute Native SQL Query.
List l = q1.list();
//display Result.
for(int i=0; i<l.size(); i++)
{
EmpBean e1 = (EmpBean) l.get(i);
s.o.p( =
}
=
```

⇒ Entity - NativeSQL must be mapped with HB POJO class

⇒ we can pass both Named and positional parameters in NativeSQL Queries.

inorder to pass both types of parameters in a single NativeSQL Query, the positional parameters must be placed before Named Parameters.

```
String sql = "select * from Employee where eid >=? and
                                          eid <= :P1 ;
SQLQuery q1 = ses.creat SQLQuery(sql);
```

```
// set parameter values.
q1.setInteger(0,300);
q1.setInteger("p1",800);
   //map result with HB pojo class
   q1.addEntity(EmpBean.class);
   // execute NativeSQL Query
  List l = q1.list();
  // display results
  for(int i=0; i<l.size(); i++)
    {
    =
   }
==
```

=> Like JDBC Data types in the form of TYPES.INTEGER, Types, Types.VARCHAR and etc, HB also giving its own built-in data types as constants of org.hibernate.Hibernate class,

ex:- Hibernate.INTEGER, Hibernate.FLOAT, Hibernate.LONG, hibernate.STRING etc..

these Hibernate data types are useful while catching scalar Query Results, scalar NativeSQL Query results in Hibernate environment.

public static final member variables of java class or interface are called constants.

Example on Scalar Queries –

```
//Native SQL scalar query.
String sql = "select max(eid) as mval from Employee";
SQLQuery q1 = ses.createSQLQuery(sql);
// mapping scalar query result with HB data type
```

// executes Native SQL Query

List l = q1.list();

s.o.p ("max value of eid col is " + l.toString());

=

}

⇒ **making Scalar Query returning more than one result.**

// Native SQL scalar Query

String sql = "select max(eid) as mval, count(*) as cnt from Employee";

SQLQuery q1 = ses.createSQLQuery(sql);

//mapping scalar query result with HB data types

q1.addScalar("mval", Hibernate.INTEGER);

q1.addScalar("cnt", Hibernate.INTEGER);

// executes Native SQL Query.

List l = q1.list();

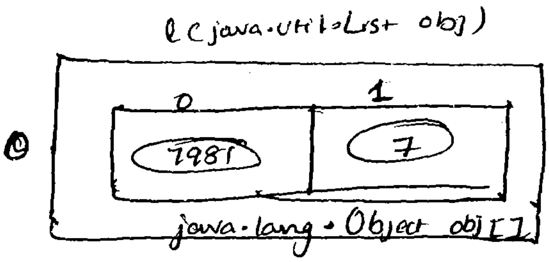
//display results

Object res[] = (Object[]) l.get(0);

s.o.p ("max val in EID col is " + res[0].toString());

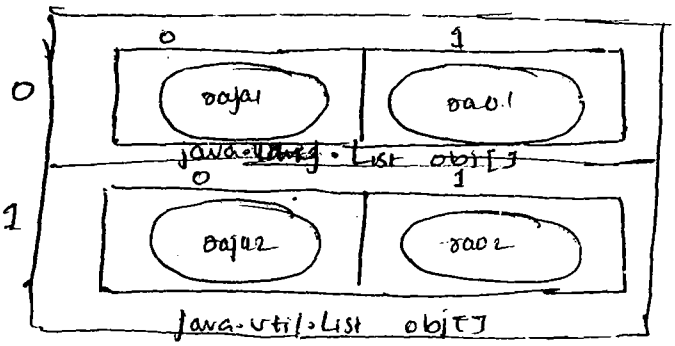
s.o.p ("count of records" + res[1].toString());

=



Example on Native SQL Scalar Query that is selecting specific Column values of table 3 -

(java.util.List obj)



String sql = "select FIRSTNAME, LASTNAME from employee where email like :P1";

SQLQuery q1 = ses.createSQLQuery(sql);

// set parameter values

q1.setString("P1", "%gmail.com");

// map scalar query results with HB data type

q1.addScalar("firstname", Hibernate.STRING);

q1.addScalar("lastname", Hibernate.STRING);

// execute Scalar Query.

List l = q1.list();

for(int i=0; i<l.size(); ++i)

{

Object row[] = (Object[]) l.get(i);

for(int k=0; k<row.length; ++k)

{

s.o.p(row[k].toString() + " ");

}

s.o.p();

}

}

19/08/10

⇒ All non-select NativeSQL Query must be executed as transactional statement.

⇒ By using NativeSQL Query we can directly insert single record into DB table, by directly passing values.

```
Transaction tx = ses.beginTransaction();
String sql = ses.createSQLQuery(sql);
sql.setInteger("p
String sql = "insert into Employee values(:p1,:p2,:p3,:p4)";
SQLQuery q1=ses.createSQLQuery(sql);
q1.setIntegers("p1",324);
q1.setString("p2","raja");
q1.setString("p3","ram");
q1.setString("p4","raja@gmail.com");
int res = q1.executeUpdate();
S.o.p("no.of records effected :"+res);
tx.commit();
```

⇒ we can execute DDL Queries as NativeSQLQueries,

```
Transaction tx = ses.beginTransaction();
String sql = "Drop table xyz";
SQLQuery q1 = ses.createSQLQuery(sql);
q1.executeUpdate();
S.o.p("table dropped");
tx.commit();
```

→ NativeSQLQuery, that is placed in HBMapping file, having logical name is called NamedNativeSQL Query, these Queries are required.

① To make NativeSQLQuery visible across the multiple session object of hibernate application
(All these session object must be created by using same SessionFactory object).

② To make NativeSQL Queries as flexible Queries by paring them from outside persistence logic.

→ The standard principle in the industry is don't hardcode any values in java application that are possible to change in future.

ex:- ① To make JDBC persistence logic as flexible logic and to get two effect of developing DB independent persistence logic then use properties file support having SQL Query

⇒ NamedNativeSQL Queries will be written in HB mapping file but instructions of executing those queries will be given in HB Persistance logic of client application.

ex:- on NamedNativeSQLQuery (EntityQuery)

step①:- propose NamedQuery in HBMapping file as shown below.

```
<hibernate-mapping>
<class name="pl.EmpBean" table="Employee">
```

```
<property name="mail" column="EMAIL"/>
</class>
                                    ──→ logical Name of Named Query
<sql-query name="my_test">
                                    ──→ Mapping Entity Query result
                                         with HB POJO class
  <return class="p1.EmpBean"/>
     select * from Employee where eid>=?
        and eid &lt=:P1
                                    ↳ XML Entity giving
                                      '<' symbol
</sql-query>
</hibernate-mapping>
```

&lt ⇒ '<' symbol if we give like this in XML it treats subtag in so keep "&lt"

Step②:- write following Code in Client application to execute NamedSQLQuery.

```
//represents named sql query
Query q1 = ses.getNamedQuery("my_test");
  //setting parameter values
 q1.setInteger(0,100);
 q1.setInteger("P1",500);
//executes named native sql query
 List l = q.list();
   //display records
for(int i=0; i<l.size(); ++i)
{
EmpBean e1 = (EmpBean) l.get(i);
S.o.p(e1.getNo()+ ..... );
}
```

=> ∞ NativeSQL Queries are written directly in client apps along with HB persistance logic they become specific to one HB session object using which SQL Query object is created

By Making Native SQL Query as Named Queries we make them visible for multip session object in client application. (In single client apps)

```
//represant named sql query
Query q = ses.getNamedQuery ("my-test");

≡

ses.close();
Session ses1 = factory.openSession();

Query q2 = ses1.getNamedQuery ("my-test");

≡
```

Example on executing Scalar-Named NativeSQL Query:-

Step①:- Prepare NativeSQL query in mapping file.

```
<sql-query name="my-test">
   <return class="p1.EmpBean"/>
   select * from Employee where eid>=? and
                                &lt;= :p,
</sql-query>

<sql-query name="mytest1">
   <return-scalar column="FIRSTNAME" type="string"/>
   <return-scalar column="EMAIL" type="string"/>
```

select FIRSTNAME, EMAIL FROM employee where
email like ?

<sql-query>
</hibernate-mapping>

step④ :- write coding in client App to execute NamedScal NativeSQLQuery.

```
Query q1 = ses.getNamedQuery("my_test1");
    //setting parameter values
    q1.setString(0, "%gmail.com");
    //execute named native sql query
    List l = q1.list();
for(int i=0; i<l.size(); ++i)
{
 Object row[] = (Object[]) l.get(i);
  for(int k=0; k<row.length; ++k)
   {
   s.o.p(row[k].toString()+"   ");
   } //inner for
   s.o.p();
} // outer for
```

Note1 :- it is always recommended to write NativeSQL Queries as NamedQueries.

NativeSQLQueries, Named NativeSQL Queries also support pagenation, use same methods of HQL programming.

=> we can keep NamedNativeSQLQuery in a single HBMapping file.

=

=> Executing Non-Select NativeSQLQuery as NamedQuery :-

Step①:- prepared named in HBMapping file.

```
<sql-query name="my_test2">
  delete from Employee where eid = (select max(eid)
                                    from Employee)
</sql-query>.
```

Step②:- execute NamedQuery in client appln.

```
Transaction txn = ses.beginTransaction();
Query q1 = ses.getNamedQuery("my_test2");
int res = q1.executeUpdate();
s.o.p("no of effected records "+res);
tx.commit();
ses.close();
}
```

=

20/8/10

-> we can make HQL Query as NamedQueries by passing them in HBmapping file using <query>

-> By making Queries as NamedHQLQueries, we can make them visible for multiple session object that are created by using SessionFactory object.

example appln that deals with NamedHQLQueries:-

① prepare NamedHQL Query in HBmapping file.

```
<query name="my_test2">
  from EmpBean as eb where eb.no = :p1
```

step②:- execute NamedHQL Query in the client application.

```
Transaction tx = ses.beginTransaction();
Query q1 = ses.getNamedQuery("my_test2");
   //setting parameter value
   q1.setString("p1",67);
    int res = q1.executeUpdate();
  S.o.p("no of records that are effected :" + res);
    tx.commit();
}
```

⇒ example on select NamedHQLQuery

step①:- place HQL Query in mapping file.

```
<query name = "my_test3" >
    select eb from EmpBean as eb where eb.mail like :p1
</query>
```

step②:- execute this query from client-appln

```
Query q1=ses.getNamedQuery("my_test3");
  q1.setString("p1", "y.gmail.com");
  List l = q1.list();
 for(int i=0;i<l.size();i++)
 {
  EmpBean eb = (EmpBean) l.get(i);
  S.o.p(eb.getNo() + " " + ....);
 }
```

⇒ In order to centralise certain persistance logic or Business logic for multiple modules of a projects or for multiple projects of a company then place that logic in DB s/w as PL/SQL procedure (or) function

→ HB persistance logic can call PL/SQL procedure (or) function of DB s/w from client application only when

→ These rules of developing PL/SQL procedure or function for HB will change based on the DB sw we use.

→ A cursor in PL/SQL programming of oracle is a Datatype whose variable can store zero or more selected records from DB table.

→ common rules to prepare PL/SQL procedure or function for thibernate.

① pagenation is not possible, so the results generated by Queries of HB specific PL/SQL procedure.
ie we cannot use setFirstResult(), setMaxResult() methods in this environment.

② we must follow the following syntax, to call ~~procedure~~ PL/SQL procedure.

```
{ call procedurename(<parameters>) }
```

③ we must follow, following syntax to call PL/SQL function.

```
{ ?= call functionname(<parameters>) }
```

Note:- PL/SQL procedure doesnot result value, where as PL/SQL function returns a value.

Rules specific to ORACLE :-

① PL/SQL procedure 1st parameter must be an out parameter returning resultset. [cursor having records]

② PL/SQL function must return a resultset [cursor having records]

Note :-
SYS_REFCURSOR is cursor type Datatype given by PL/SQL programming of oracle having the ability to store selected records [one or more] like resultset Object of JDBC programming

→ For rules specific to Sybase, Microsoft SQL DB Sw refer chapter 16 of PDF file.

NOTE:- HB applications use NativeSQL programming to call PL/SQL procedures or functions.

⇒ Example Appln to call PL/SQL procedure of oracle from HB persistance logic ? -

Step①:- prepare PL/SQL procedure in oracle, in the angle it is required for HB.

[refer above oracle rules]

```
> create or replace procedure get_empdetails(mycur out SYS_REFCURSOR,
                                              ename in employee)
  begin
    open mycur for
        select * from Employee where firstname like ename;
  end;
```

(procedure cursor DataType in oracle)

→ * the nativeSQL Query generated records will be stored over

Step②:- execute the above PL/SQL procedure in oracle DB Sw

```
> /
=
```

Step③:- call the above PL/SQL procedure from HB mapping file as Named NativeSQL Query. in employee.hbm.xml

```
<sql-query name="my_test" callable="true">
    <return class="EmpBean" />
    {call get_empdetails(?, :P1)}
</sql-query>
=
```

(callable → mandatory; EmpBean → HB POJO class; ? → *; :P1 → **)

* → this '?' marks represents out parameter of the above PL/SQL procedure (cursor) it must always be '?' symbol you cannot place named parameter here.

** [:P1] :- :P1 represents 'p1' parameter of above PL/SQL procedure, it can be either named or positional parameter.

when it is taken as positional parameter, it index becomes zero.

=

step 4 :- write following code in client application to call & execute PL/SQL procedure. (code in client applns)

```
Query q = ses.getNamedQuery("my_test");
// setting parameter values
q.setString("p1", "%x");
List l = q.list(); // execute .pl/sql procedure.
for(int i=0; i<l.size(); ++i)
{
  ≡
}
```

=

(5) Develop remaining resources of the application like I^st application

=

"श्री आंजनेयम्"

INDEX

NAME : D. Babu STD. : SEC. : ROLL NO. : 9966072198

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
		Hibernate - ②		

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks

21/8/10

* procedure to call PL/SQL function of Oracle DB slw by using hibernate peristance logic :-

step(1) :- prepare PL/SQL function in oracle DB slw returning ResultSet (cursor)

> create or replace function emp_function(no in number)
return SYS_REFCURSOR

as
my_cursor SYS_REFCURSOR;
BEGIN
open my_cursor for
select * from Employee where eid >= no;
return my_cursor;
END;

select * from user_procedures;

step(2) :- execute this PL/SQL function in oracle DB slw.

step(3) :- call the above PL/SQL function from HB mapping file by using standard syntax.

in Employee.hbm.xml :-

<sql-query name = "my_test" callable = "true"> → mandatory
<return class = "EmpBean" />
{(?) = call emp_function(:pi)}
</sql-query>
</hibernate-mapping>

Can't be taken as Named parameter

step(4) :- write following peristance logic in client appln to call and exicute that PL/SQL procedure.

```
Query q1 = ses.getNamedQuery("my_test");
    //setting parameter value
q1.setInteger("p1", 350);
List l = q1.list();   //execute PL/SQL function in DB s/w
for(int i=0; i<l.size(); i++){
  EmpBean eb = (EmpBean) l.get(i);
    =
}
=
```

** We can't perform pagination / paging on the results generated by PL/SQL procedure or function called from HB persistance logic. (limitation of HB)

** Limitations of HB s/w :-

① since HB is not a distributed technology, the HB persistance logic is not distributed persistance logic. So this persistance logic cannot be used from Remote client.

② calling PL/SQL procedure (or) function from HB persistance logic is quite complex more over these PL/SQL function or procedure must be written in the same angle they are required for HB.

③ pagination is not possible on the results not generated by PL/SQL procedures or functions.

④ HB can't be used to interact with Non-conventional DB s/ws like text file, MS Excel and etc

(JDBC can do this)

⑤ HB cannot interact with few conventional DB SW like MS-Access

⑥ HB persistance logic use, negligible performance degradation compare to JDBC.

***

⇒ → Calling PL/SQL procedure (or) function from HB persistance logic when it is not written based on HB rules is not directly possible but it is possible indirectly by injecting JDBC code in HB persistance logic.

Example app:-

step①:- develop and execute normal PL/SQL function in oracle.

```
create or replace function my_function(x in number)
                    return number as y number;
    begin
      y:=x*x;
      return y;
    end;
    /
```

step②:- write following code in client application to call that PL/SQL function.

```
Transaction tx = ses.beginTransaction();
    // jdbc logic.
Connection con = ses.connection(); // gives jdbc con obj
CallableStatement cs = con.prepareCall("{?=call my_function(?)}");
    cs.setInt(2,10);          // this JDBC code starts from '1'
cs.registerOutParameter(1, Types.INTEGER);
```

w.r.t DB cs.execute(); // execute PL/SQL function of DB s/w.

sop ("result is" + cs.getInt(1));

tx.commit();

cs.close();

=

=

NOTE:- to execute the above persistence logic there is no need of HB POJO class and HB mapping file, just the HB configuration file is enough.

the above coding scenario is very much needed to call already available and running PL/SQL procedures from HB application even though they are not developed in the angle they are required for Hibernate.

=

⇒ procedure to call a PL/SQL function of oracle, performing Non-select operation on the table:-

step(1):- prepare & execute PL/SQL function as shown below.

```
> CREATE OR REPLACE FUNCTION my_function1(no in number)
    RETURN NUMBER AS
  BEGIN
     delete from Employee where eid=no;
     return SQL% ROWCOUNT;
  END;
/
```

↓

Built-in cursor in oracle representing the no of record that are effected bec of Non-select Query execution done in PL/SQL programming.

→ to call above PLISQL function in client appln as shown below.

Non-ORMapping

```
Transaction tx = ses.beginTransaction();
    //jdbc logic
Connection con = ses.connection(); //gives jdbc con obj
CallableStatement cs = con.prepareCall("{ ? = call my_function(?,?) }");
    cs.setInt(2,567);
    cs.registerOutParameter(1,Types.INTEGER);
    cs.execute(); //executes PLISQL function DBSide
    S.O.P("no. of records effected :"+cs.getInt(1));
    tx.commit();
```

Note:- HQL Queries based PLISQL programming is not possible

23/08/10

CRITERIA API :-

→ This API is given to develop HB persistance logic plug-in java statements and with out using SQL, HQL Queries

→ Criteria API means working with classes & interfaces of org.hibernate.criteria.package classes & interface to develop P.L.

→ we can use Criteria API only select Operations based persistance logic development.

→ we ~~can use~~ have to use Criteria API, only to select all column values of a table. ie Criteria API is not design to select specific column values of a table.

→ while developing Criteria API based persistance logic POJO classes, ~~Member Vari~~ memberVariable names will be utilised

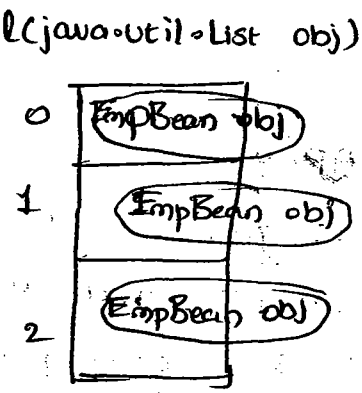
→ Criteria API based persistance logic is DataBase independent persistance logic.

Example on criteria API to select all records of a table :-

→ gives `select * from Employee` object.

```
Criteria ct = ses.createCriteria(EmpBean.class);
List l = ct.list(); // execute criteria api based PoL
for(int i=0;i<l.size();i++)
{
EmpBean e1=(EmpBean) l.get(i);
S.O.P(e1.getNo()+" "+e1.getFname()+---);
}
}
```

Memory Diagram :-



→ Iteration based select Query execution is not possible with criteria API., Because iterate() is not available in org.hibernate.Criteria interface

⇒ To execute logic with conditions - (greater than equal)

```
Criteria ct=ses.createCriteria(EmpBean.class);
Criteria cond1 = Restrictions.ge("no", new Integer(100));
ct = ct.add(cond1);
```

List l = cr.list(); //gives select * from employee where eid >=100 also

for(int i=0; i<l.size(); i++)

{

=

}

=

All methods of Restriction, Expression class are given to frame condition while developing Criteria API based persistance logic.

All these methods directly (or) indirectly return Criterian object, these two classes are available org.hibernate.Criterian package, Expression is the subclass of Restrictions class.

=

*** (Q) what is the diff b/w criteria object and criterian object?

ans) Criteria object represents the total single persistance operation related persistance logic of Criteria API. the conditions in this persis logic will be represented by Criterian obje

→ we add criterian objects to criteria object to add conditions to the basic select operation

→ criteria object means, it is the object of class that implements org.hibernate.criteria interface.

criterian object means, it is the object of class that implements org.hibernate.Criterian.Criterian inte

⇒ Criteria object is mutable object, i.e., After creating Criteria object, if you add new data or if modify existing data of that object changes will be reflected dynamically in the same object

⇒ when multiple conditions are added to criteria object these conditions will be executed having 'and' clause by default.

Example with multiple conditions :- (greaterthan equal / like)

```
Criteria ct = ses.createCriteria(EmpBean.class);
Criterion cond1 = Restrictions.ge("no", new Integer(100));
Criterion cond2 = Restrictions.like("mail", "%.gmail.com");
ct.add(cond1);
ct.add(cond2);
List l = ct.list(); // gives select * from employee
                    where eid >= 100
                    and email like '%.gmail.com';
```

using in conditions :- (in / or)

```
Criteria ct = ses.createCriteria(EmpBean.class);
Criterion cond1 = Restrictions.lt("no", new Integer(100));
Criterion cond2 = Restrictions.in("fname", new String[]
                                  {"raja", "ravi"});
Criterion orcond = Restrictions.or(cond1, cond2);
```

ct.add (oocond);

List l = ct.list(); // select * from employee where (eid<100) or (hostname in ("raja", "ravi"

foo(--);

{

}

=>

Placing multiple conditions with OR clauses :-

Criteria ct = ses.createCriteria (EmpBean.class);

Criterion cond1 = Restrictions.lt ("no", new Integer(100));

Criterion cond2 = Restrictions.in ("fname", new String[] {

"raja", "ravi" });

Criterion oocond1 = Restrictions.or (cond1, cond2);

Criterion cond3 = Restrictions.ne ("mail", new String ("rajag

gmail.

Criterion oocond2 = Restrictions.or (oocond1, cond3);

ct.add (oocond2);

List l = ct.list();

foo(--

}

// select * from Employee where (eid < 100 or fname

('raja', 'ravi')) or email <> 'anil@gmail.com'

Applying And, OR clauses based on condition in Query execution :-

→

```
Criteria ct = ses.createCriteria(EmpBean.class);

Criterion cond1 = Restrictions.lt("no", new Integer(100));

Criterion cond2 = Restrictions.in("fname", new String[]
                                  {"raja", "ravi"});

Criterion andcond = Restrictions.and(cond1, cond2);

Criterion cond3 = Restrictions.ne("email", new String
                                  ("raja@gmail.com"));

Criterion orcond = Restrictions.or(andcond, cond3);
    ct.add(orcond);
    List l = ct.list();
    for
    ==
    ==
```

=> we can pass conditions in criteria API based persistance logic in two forms of SQL Query statements by using expression.sql()

```
Criteria ct = ses.createCriteria(EmpBean.class);
Criterion cond1 = Expression.sql("email like '%.gmail.com'");
Criterion cond2 = Expression.sql("firstname' in ('raja', 'ravi')");
    ct.add(cond1);
    ct.add(cond2);
    List l = ct.list();
    for(-)
    ==
    =
```

HQL is the most recommended technique to develop persistance logic in HB environment.

24/8/10

→ HQL, NativeSQL, Criteria API based programming support paging operation but the NativeSQL programming that deals with PL/SQL procedures and functions doesnot support paging operation.

Filters :-

→ HB Filters are hibernate 3.x feature that allows the programmer to specify predefined criteria / condition for the persistence logic at hibernate programming.

~~24/8/10~~

→ Filters represent where condition clause related, condition in HB mapping file and they are visible for multiple session objects of hibernate application.

These filters can be enabled or disabled dynamically at runtime on each session object level.

→ A hibernate Filter is a Global Named parameterised condition that can be enabled or disabled per particular HB session, when enabled all Queries executed using that session object will be executed with condition of the Filter

Example Application to HB Filter on HB persistance logic :

① → define filter in HB mapping file specifying its param names and types.

in

in employee.hbm.xml file:-

```
<hibernate-mapping>
    <class name=
        ====
    </class>                         ↱ logical name of filter.
    <filter-def name="myfilter">
parameter   { <filter-param name="myid1" type="int"/>
names & types
              <filter-param name="myid2" type="int"/>
    </filter-def>
  </hibernate-mapping>
```

step②:-

link filter with specific hibernate POJO class and write condition.

```
<hibernate-mapping>
    <class name="p1.EmpBean" table="Employee">
        <id name="no" column="eid" />
        <property
           =                    the filter that is taken
                                          above.
                                  ↑
        <filter name="myfilter" condition="
           condition="eid >= :myid1 and eid &lt;=
                                    = :myid2"/>
      </class>                             ↓
      <filter-def name="myfilter">
         ≡                          condition represents
                                       by filter
      </filter-def>
    </h-m>
        =
```

Step④ :- enable filter on HB session object and execute HB peosistance logic.

Code in client application :-

```
Demo
    ses = sf.openSession();
  // enable filter on hibernate ses obj
org.hibernate.Filter f1 = ses.enableFilter("myfilter");

  //set parameter values
f1.setParameter("myeid1", new Integer(100));
f1.setParameter("myeid2", new Integer(400));

   // HQL Query

Query q1 = ses.createQuery("from EmpBean");
    List l = q1.list(); //execute HQL query with condition

for(int i=0; i<l.size(); i++)
  {
EmpBean e = (EmpBean) l.get(i);
  s.o.p(e.getNo()+" "+e.getName()+.....);
  }
// disabling filter on ses object
  ses.disableFilter("myfilter");

  Query q2 = ses.createQuery("from EmpBean");
List l1 = q1.list(); // execute HQL query with out condition

for(int i=0; i<l.size(); ++i)
{
  EmpBean e = (EmpBean) l1.get(i);
  s.o.p(....);
}
```

Hibernate Filters makes no conditions of HB Filter logic as flexible condition by making them coming to applns from outside the persistance logic (comes from HB mapping file)

step③ :- Develop remaining resources of the application like first application.

*** => For complete source code above application refer page applin ⑬ given in page no : 82-85

=> we can enable or disable filters only on ~~..~~ types of HB persistence logic environment HQL, ~~..~~ Criteria API

-> it is not applicable for "NativeSQL"
- (we cannot enable Filters on NativeSQL programming related persistance logic development).

***
Q) => when and where to use interfaces and abstract classes in your project development?

Ans -
For every project project specification will be design before starting coding of the project, this specification contains rules and guidelines to develop the project.

every project specification contains its own project API ~~API~~ designed by PL, this API represents methods declared in the interfaces, abstract classes as rules and represents concrete method definations of abstract classes and concrete classes as guidelines.

while implementing project based on project specifical rules will be implemented & interface methods and guidelines will be followed (concrete method definitions)

→ If PL wants to give everything as rule in project specification then he takes interfaces and declares method representing rules.

⇔ If PL wants to give rules and guidelines in project specification then he takes abstract classes and declares ~~methods~~ abstract methods representing rules and defines concrete methods representing guidelines

→ In coding level of project development programmer never takes userdefined interfaces and abstract classes.

But he takes userdefined interfaces only when they are required as resources of certain technology based application development resources.

→ spring interfaces in spring applin.

→ Busines interface in RMI applin, EJB component.

⇔ =

⇒ s/w specification contains, rules and guidelines to develop new softwares, every s/w specification contains built-in API, the interfaces of this API represent rules of s/w specification having method declaration. the concrete methods of this API, available in concrete, abstract classes represent guidelines of s/w specification

Ex → ① JDBC specification contains rules and guideline to deve JDBC drivers

② Servlet, JSP specification contains to develop servlet containers / web containers

25-8-10

INHERITANCE MAPPING :-

Hibernate persistance classes are POJO class, so they can participate in inheritance to give the advantage of reusability and extensibility.

When HB persistance classes are there in inheritance, ie DB table's Related to these persistance class we also say to maintain data having relation ship.

To make HB understanding this inheritance b/w POJO classes, we need to configure them in HB mapping file based inheritance mapping concepts.

→ Inheritance b/w two Servlet components is possible, but inheritance b/w two EJB components is not possible.

→ HB allows the following ORMapping operations in mapping file.

① Basic OR Mapping

② Inheritance Mapping

③ Collection Mapping

④ Association mapping ~~and etc...~~

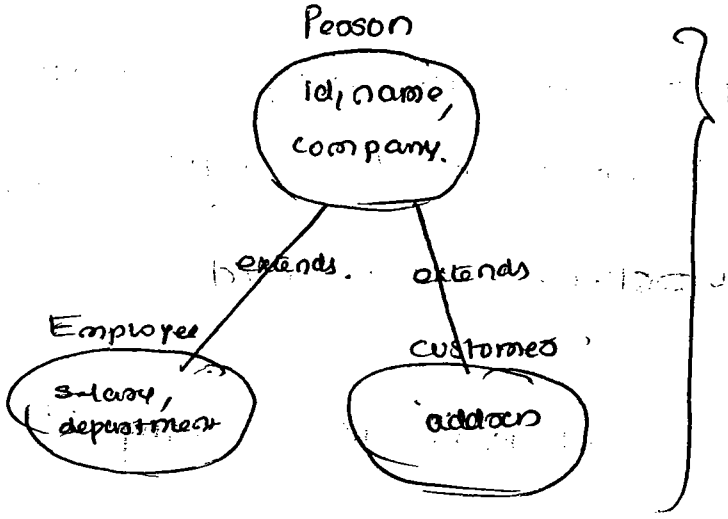
⑤ Component mapping and etc....

→ HB s/w use ③ approaches of mapping inheri
mapping to get Reusability, extensibility advantag
of inheritance in different ways.

they are ① Table per class hierarchy
② Table per subclass
③ Table per concrete class.

① Table per class hierarchy :- (59 page no)

All the classes of inheritance hierarchy use single DB table by having discriminator column (as different).



in_persons

Discrimin colu

(PK) id	name	company	salary	department	address	pers
101	raju	HCL	—	—	—	perso
104	ravi	TCS	10,000	1001	—	emp p
108	ram	Wipro	~~15,000~~	—	hyd	cust p
210	rakesh	HCL	—	—	sec	cust p

→ In table per class hierarchy, it is recommended to take DB table having discriminator column because this column maintains logical values representing records in the table are inserted using which POJO class of inheritance hierarchy.

ex:- person_type column of above DB table.

→ the logical values of this discriminator values are very useful to select specific POJO class related records separately for DB table even though DB table contains other POJO-classes related records.

→ In table per class hierarchy inheritance mapping <class> will be used to configure super class and <subclass> will be used to configure sub classes.

this <subclass> is subtag of <class> tag.

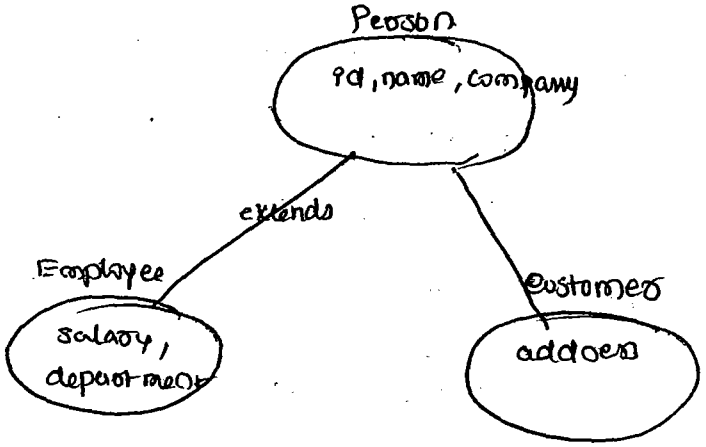
→ In this approach discriminator column of table doesn't contain any property in Hibernate POJO class but its configuration will be done in HB mapping file.

⇒ For Example Application on table per class Hierarchy model of inheritance mapping refer Application :- ⑥ given in page no :- 54-58

→ while working with table per class hierarchy inheritance model the columns related to child pojo class properties in DB table cannot be applied with 'not null' constraint and discriminator column cannot be applied with unique key or primary key constraint.

21/8/19
63page no ⇒ The advantage of any model of Inheritance mapping, is nothing but the ability of access child pojo class objects selected records through parent pojo class object.

② Table per subclass model of Inheritance mapping:—



→ In this model, every pojo class of inheritance will have it's own database table. The table of child pojo classes will have r/s with (one-to-one parent pojo class table.

inperson 2 (parent table)

id (pk) (N)	name (VC2)	company (VC2)
101	xxx	HCL
101*	yyy	POLARIS
81**	333	WIPRO

VC2 → Varchar 2 (20)
N → ~~not~~ null Number

58

in_employee (child table)

Salary (N)	department (N)	person_id (FK with id of in_person) (N)
6000	1001	121 *

N - Number

in_customer2 (child table)

address (N2)	person_id (N)
Hyd	131 **

FK with id column in person

=

→ In this model of inheritance mapping all column's of all DB table's can be applied with not null constraint and there is no necessity of descriminator column.

One-one relationship b/w two tables is nothing but one record of parent table will always represent one record of child table.

→ In table per subclass model of inheritance mapping <class> will be used configure parent POJO class and <joined-subclass> used to configure child_POJO classes.

<joined-subclass> is the sub tag of <class>,

—

→ In this model of inheritance, child POJO class object data will be splitted and will be inserted parent, child tables having relationships.

this relationship comes because of Foreign Column of child table, so this Foreign key column should be configured along with child POJO classes in HB Mapping File.

=

Table per subclass inheritance mapping model is most regularly used inheritance mapping model in company level programming. because.

① it recognises inheritance b/w POJO classes. and gives reusability of properties.

② it allows to design table having relationship as satisfying DataBase designing principles.

③ allows to apply, Not Null Constraint on all the columns of all the tables

For Example application on Table per Subclass inheritance mapping refer application ⑦ in page 59-63

→ while working with multiple table, advanced ORM mapping concepts, it is recommended not to use reverse engineering of MyEclipseIDE.

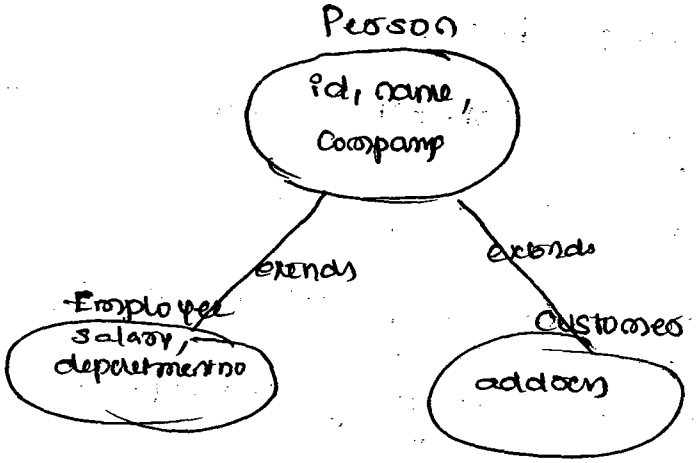
use MyEclipseIDE for these operations but develop resources manually

=

③ Table per concrete class

In this inheritance mapping, even though POJO classes are there in the inheritance, their tables will not participate in relationships and every POJO class in inheritance hierarchy will use one separate table without having relationship with other table.

→ These POJO classes will be configured in HBM mapping file as individual, independent, concrete classes without showing inheritance and without showing reusability of properties configurations.



in_person table

id (no) (pk)	name (vc2)	company (vc2)
101	raja	xyz

in_Employees table

id (no) (pk)	name (vc2)	company (vc2)	salary (no)	deptno (no)
102	ravi	abc	8,000	1001

in_Customer table

id (no) (pk)	name (vc2)	company (vc2)	address (vc2)
103	ramesh	mno	hyd

=> The above table designing is very poor designing because multiple tables of DBS are having same col so Table per concrete class model of of inheritance mapping is not industry standards.

=> In this model all columns of all tables can be applied with 'Not-Null' constraint.

=> For Example application on table per concrete class App@ page no 63-68

27/08/2010

* Component mapping :-

The property of HB POJO class that represents single column of DB table is called simple property

-> The property in hb pojo class whose type is class name (user defined) and represents multiple columns of a DB table is called component property.

-> A component property is a contained object in hibernate POJO class object representing specific multiple column values of a record.

-> component property of hibernate POJO class represent partial data of record where as its hibernate POJO cla object represents total record data including this partial data.

-> public class Person
{
int pid;
String pname;
JobType pjob; // component property

= getXXXX()

setXXXX()
}

→ public class JobType

{

String job;

double salary;

int department;

= getXxx()

setXxx()

}

Peoson-fab (DB table) :-

pid (n)	pname (VC2)	job (VC2)	salary (n)	department (n)
101	raja	SE	18000	1001

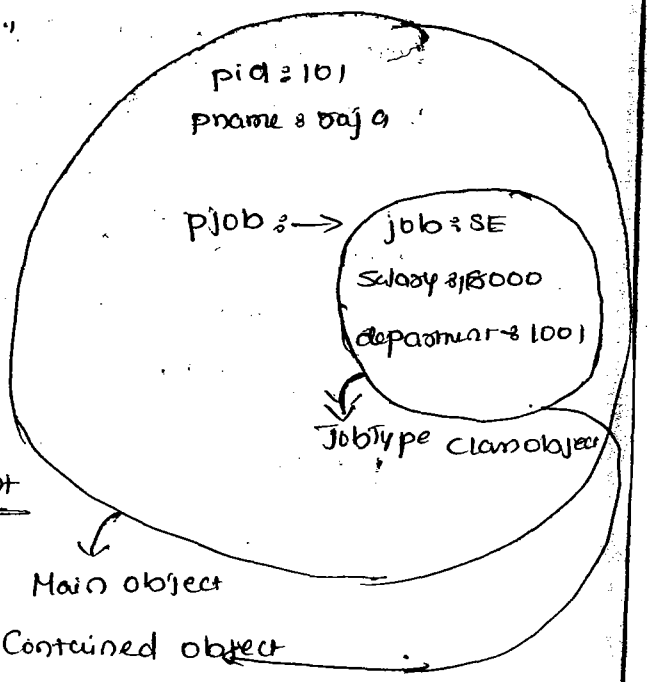
pid (property) → pid col

pname (") → pname col

pjob (") → job, salary, department cols

→ To configure "simple property" in HB POJO class use <property> tag in mapping file

similarly <component> tag in mapping file.



→ For example application on component mapping refer app no : 5 of material in the page no :- 51-54

→ If you are looking to maintain partial values of one in the form of separate java class object take compon property in this pojo class representing selected multiple values of a table.

Q) How to find out no. of objects created for java cla

ans:- Take static member variable in java class and increment that member variable in constructor. So this static member variable keeps track of th number of objects created for that class.

ex:- TestApp.java

```
class Test
 {
   public static int counter;

   Test ()
    {
     s.o.p ("from arg constructor : Test class");
     counter ++;
    }
 }
public class TestApp
 {
   P S V main (String args[])
    {
      Test t1 = new Test ();
      Test t2 = new Test();
      Test t3 = new Test();
     s.o.p (" No of objects are created = " + Test.
    }
 }
```

28-08-20

Association Mapping :-

* DB designing team designs the tables of project according to "Normalization Rules". There are total six Rules (or) forms. The Second normalization rule says design tables having integrity constraints that means tables should be designed having rls like one-to-one, one-to-many, many-to-many and etc.

when tables are there in Association (or) Rls we can accesses one table data based on another table data because records of one table represents the records of another table.

→ DB team take the support of primary key, Foreign key constraint to design the table having relationships

→ The relationship b/w student table and rank table is one-to-one because one student contains only one rank will be given or assigned to only one student.

→ The relationship b/w user, phonenumbers table is ~~one-to-one~~ one-to-many because because one user can have multiple phone numbers.

→ The rls b/w emp, dept tables is many to one because many employees can belong to one department.

→ The rls b/w programmers and project in small scale organization is many to many because in one project multiple programmers are there, one programmer can work for multiple projects, multiple projects can be

assigned to one programmer (or) multiple programmers.

→ To define any rls except many-to-many two tables are enough (parent, child) but to define many-many and three tables are required (table1, table2, relationship table)

→ When two database tables are in relationship their "HB pojo classes" must be designed and configured supporting that relationships. this work is called "Association mapping" in HB mapping file.

→ When HB pojo classes are designed supporting rls then the objects of these classes actually participates in object level relationship (or) Association.

HB object level relationships

Uni-directional	Bi-directional
1-1 (Uni)	1-1 (Bi)
1-m (Uni)	1-m (Bi)
m-1 (Uni)	m-1 (Bi)
	m-m (Bi)

→ By using parent pojo class object data we are able to access the associated child pojo class "object/objects" data and its reverse is not possible then it is called "unidirectional" association. Its reverse is also possible Then it is called "bidirectional" association.

Ex:- In student pojo class object (parent) pointing one bank pojo class object (child) having one-to-one association in programmer is able to get associated bank object data through student object data and reverse is not possible then it is called "unidirectional associa

If reverse is also possible then it is called "bidirectional association".

→ The HB programmer should always design HB pojo classes based on the E-R diagrams given by db team.

→ ER Diagrams: (Entity - Relationship diagrams)

To perform association mapping in hibernate mapping file we can work with following tags <one-to-one>, <one-to-many>, <many-to-many>, and <many-to-one> tags.

inverse = "true" attribute of these tags allows to enable bi-directional association / relationship.

inverse = "false" allows to enable unidirectional association.

⇒ Example HB POJO classes for unidirectional one-to-one association :-

```
public class Student
{
    int sno;
    String name, address;
    Rank r;  ⟶ represents one object of Rank POJO class
    - -
    - -
    - -   4 no of setXxx(), getXxx() methods.
}
public class Rank
{
    int sno;
    String subject;
    String exam;
    setters & getters
```

→ Example HB POJO classes for bidirectional one to one association :-

```
public class Student
{
    int sno;
    String name;
    String address;
    Rank r; → Represents one object of Rank pojo class

  → getters & setters

}
public class Rank
{
    int rno;
    int sno;
    String subject;
    String exam;
    Student st; → Represents one object of Student POJO class

  → getters & setters

}
```

→ Example HB POJO classes for unidirectional one-to-many association :-

```
public class User
{
    int uid; → represents foreign key column of child table
    String username;
    List phones; (or) Set phones; → doesn't allow duplicate
    ↓ (List: allows duplicates)
    represents one (or) more (or) zero phone number class POJO objects

    get xxx & set xxx

}
```

```
public class PhoneNumbers
{
    long pb;
    String type;
    int uid;
    - -
    - -
    - -
}
```

⇒ Example HB POJO classes for Bi-Directional one-to-many Association :-

```
public class User
{
    int uid;
    String username;
    Set phones;
    =
}
public class PhoneNumbers
{
    long pb;
    String type;
    int uid;
    User u1;
    -
    =
    -
}
=
```

→ Example HB POJO classes for UNIDIRECTIONAL MANY-TO
Association :-

```
public class Emp
{
  int eno;
  String ename;
  float sal;
  int deptno;   //represents Foreign key column of parent table
  Department dept; → represents one object of Department POJO class

  setters & getters
}
public class Department
{
  int deptno;
  String name;
  String loc;

    setters & getters

}
```

-many

* In the above discussion "dep" properties belong into multiple Emp class objects represents single (or) Same Department class object, to get the object of many-to-one association.

Example HB POJO classes for Bidirectional many-to one association :-

```
public class Emp
{
  int eno;
  String ename;
  float sal;
  int deptno;
  Department dept;
```

```
public class Department
{
    int deptno;
    String name;
    String loc;
    Set e;
    =
}
```

→ Example HB POJO classes for many-to-many association :-

```
public class Project
{
    int proj id;
    String projname;
    Set programmers;
        ↳ represents zero (or) more objects of programmers class.
    =
}
```

30/8/10

In Association mapping parent and child POJO classes that are representing parent child DB table respectively will not participate in any kind of DB inheritance. But there is a possibility of property of parent POJO class representing one or more objects of child POJO class and vice versa.

Example application on one-to-many unidirectional Association mapping based on user and phonenum information :-

DB tables :-

Parent table :-

```
create table USER_TABLE (USER_ID number
    primary key, FIRST_NAME varchar2(20));
```

Child table :-

```
create table PHONE_NUMBERS (NUMBERTYPE
    varchar2(20), phone number(10),
    UNID number references USER_TABLE(USER_
```

↑ Foreign key column allowing duplicates

→ values to this foreign key UNID should not be inserted or updated directly because these values comes automatically based on the values inserted in PK column of parent table.(user_id).

Application Resources :-

- → hb.cfg.xml
- → User.java
- → phonumber.java } Hibernate POJO classes
- → User.hbm.xml
- → phoneNumber.hbm.xml } HBM mapping files
- → TestClient.java — client appln ①
- → SelectTest.java — client appln ②

=

Hibernate.cfg.xml :-

Note :- configure details for oracle DB s/w connection having two mapping files configuration.

```
<hibernate-configuration>
  <session-factory>
    - -
    - -
    <!-- mapping files -->
  <mapping resource = "User.hbm.xml" />
  <mapping resource = "PhoneNumber.hbm.xml" />
  </session-factory>
  </hibernate-configuration>
```

User.java (parent POJO class)

```
import java.util.*;
public class User
{
 private long userid;
 private String firstname;
 private Set phones; → To hold zero (or) more
                       objects of phone number POJO class

 //getter & setters
 =
public Set getPhones()
 {
  return phones;
 }
public void setPhones(Set phones)
 {
  this.phones= phones;
 }
}
```

PhoneNumber.java :- (child POJO class)

```
public class PhoneNumber
{
  private String numberType;
  privat long phone;
  privat long id; ---> represent Foreign key column u
  // setter & getters
  =
}
```

/>

User.hbm.xml :-

```
<hibernate-mapping>
<class name="User" table="USER_TABLE">
 <id name="userid" column="user-id"/>
 <property name="firstName" column="FIRST_NAME"
 <set name="phones" table="PHONE_NUMBERS"
                    cascade="all">
  <key column="UNID"/>
        |--> FK Column in child table.
 <one-to-many class="PhoneNumber"/>
  </set>
 </class>
</hibernate-mapping>
```

=

Note①:- If POJO class property type is collection F/w data structures like List, Set, Map and etc of java.util package. Then we need to configure those properties in mapping file through "Collection Mappin for this we use <set> tag, <map> tag, <list> tag <bag> tag and etc tags

Note②:- since phones property should hold and represent zero (or) more child POJO class objects (phone number class objects) the child table name (phone_numbers) child POJO class name (phone numbers) are specified in mapping file while configuring "phones" property.

→ key attribute in any tag (or) <key> tag always points to foreignkey column name indicating that this is colun. that responsible for R/S b/w tables.

** cascade = "all" indicates any operation like insert, update, delete performed on the parent POJO class object will be reflected in child POJO class object if necessary.

PhoneNumber.hbm.xml:-

```
<hibernate-mapping>
<class name="PhoneNumber" table="PHONE_NUMBERS">
<id name="phone" column="PHONE" />
<property name="numberType" column="NUMBER_TYPE" />
<property name="id" column="UNID" insert="false" update="false" />
</class>
</hibernate-mapping>
```

* since the mapped column of "id" property is the FK column UNID and to disable direct updations and insertion in that column as manual operation insert = "false", update = "false" values kept. So values in FK column should come dynamically based on the values inserted on the referenced PK column other tables.

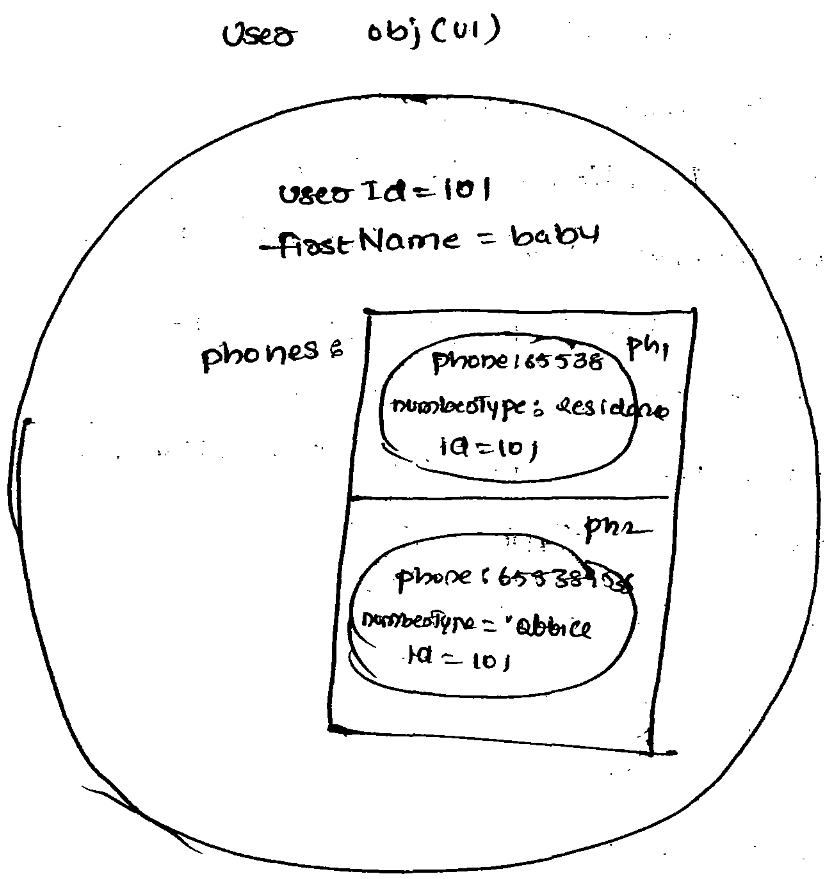
TestClient.java :

```
import org.hb.cfg.*;
import java.util.*;
public class TestClient
{
P S V main(String args[])
{
try
 {
 Session ses = - - - ;
Transaction tx = ses.beginTransaction();
  User u1 = new User();
    u1.setUserId(101);
    u1.setFirstName("aaja");
  PhoneNumber ph1 = new PhoneNumber();
    ph1.setPhoneType("residence");
    ph1.setPhone(65538968);
    PhoneNumber ph2 = new PhoneNumber();
    ph2.setPhoneType("office");
    ph2.setPhone(65538958);
  Set s = new HashSet();
     s.add(ph1);
     s.add(ph2);
```

//Setting phoNoNumber class object to phones property
~~to~~ user obj

```
        u1.setPhones(s);
        //To insert record
            ses.save(u1);
            tx.commit();
            ses.close();
    }//try
    catch(Exception e)
    {

    }
  }
}
```

Memory Diagrams -



→ When the above application executed one record wil inserted in USER_TABLE, and two records will be inser PHONE_NUMBERS table having one-to-many association a values in foreign key column UNID will be set dynami

31/08/2010

* SelectTest.java (gives parent table records and associated child table records):-

```
public class SelectTest
{
  p s v main(String args[])
  {
  try
   {
     Session ses = ---;
     Query q1 = ses.createQuery("from User");
  // gives parent table records and associated child
table records
     List l = q1.list();
  for(int i=0; i<l.size(); i++)
  {
  User u1 = (User) l.get(i);
  S.O.P("parent s" + u1.getUserId() + " " + u1.getFirstName());
  Set s = u1.getPhones();
  Iterator it = s.iterator();
  while (it.hasNext())
  {
  PhoneNumber ph = (PhoneNumber) it.next();
  S.O.P("\t\t\t child s" + ph.getPhone() + " " +
                ph.getNumberType() + " " +
                ph.getId());
  }
```

```
Query q2 = ses.createQuery("from PhoneNumbers");  <-
                    //gives only child records
    l = q2.list();
for(int i=0; i<l.size(); ++i)
 {
PhoneNumbers ph= (PhoneNumbers) l.get(i);
 S.O.p(ph.getNumberType()+" "+ph.getPhone()+" "
                    +ph.getId());
 }
 ses.close();
}//try
catch(Exception e)
 {
 =
 }
}//main
```

Note:- since the above application is having one to many unidirectional association. we can get access to child pojo class objects from parent pojo class object but access is not possible. To make reverse operation possible go for bidirectional one to many association.

* DeleteTest.java :- (To delete records)

```
public class DeleteTest
{
 P S V main(String args[])
 {
  try
  {
   Session ses= ---;
  Transaction txn= ses.beginTransaction();
```

```
Query q1 = ses.createQuery("from User");  // delete parent table records and associated records in child table.
List l = q1.list();
for(int i=0;i<l.size();++i)
{
User u1 = (User) l.get(i);
ses.delete(u1);
}
tx.commit();
}
}
```

=

Developing one-to-one Bidirectional association based web-applications using User, PhoneNumber details :-

DB tables :- same as above application

Note :- the designing of tables (parent & child) are same for unidirectional (or) bidirectional association but changes will come in pojo class and mapping file

Resources of the application :-

→ hibernate.cfg.xml
→ User.java
→ PhoneNumber.java
→ user.hbm.xml
→ PhoneNumbers.hbm.xml
→ TestClient.java
→ SelectTest.java
→ DeleteTest.java

hibernate.cfg.xml :-

→ same as above application ←

User.java :-

→ same as above application ←

PhoneNumbers.java :-

```
public class PhoneNumbers
{
 private String numberType;
 private long phone;
 private long id;
 User parent;
// write getXXX & setXXX
 public void setParent(User parent)
   {
     this.parent = parent;
   }
  public User getParent()
   {
     return parent;
   }
}
```

User.hbm.xml :-

same as previous appln.

PhoneNumbers.hbm.xml :-

```
<hibernate-mapping>
 <class name="PhoneNumbers" table="PHONE_NUMBERS">
 <id name="phone" column="PHONE" />
 <property name="numberType" column="NUMBER_TYPE" />
 <property name="id" column="UNID" insert="false"
                                   update="false" />
```

property is child pojo class to represents one object of parent pojo class.

```
 <many-to-one name="parent" class="User"
              column="UNID" cascade="all" />
```

↳ parent pojo class name.

Foreign key column in child table having ability to ~~perform~~ form this o/s

operations performed on child pojo class will be reflected in the associated parent pojo class object is necessary

```
 </class>
</hibernate-mapping>
```

TestClient.java :-

refer TestClient.java source code @ page no :- 79 & 80

SelectTest.java :-

01/09/10 refer 10 page no :- 81 & 82

⟹ DeleteTest.java :-

```
public class DeleteTest
{
  P S V main (String. arg[]) throws Exception
  {
    Session ses = ___ ;
  // parent to child
  Transaction tx = ses.beginTransaction() ;
    User u1 = new User() ;
      u1.setUserId (101) ;
    PhoneNumber p1 = new PhoneNumber() ;
      p1.setPhone (63636363) ;
      p1.setParent (u1) ;
    PhoneNumber p2 = new PhoneNumber() ;
      p2.setPhone (61616161) ;
      p2.setParent (u1) ;
    Set s = new HashSet() ;
      s.add (p1) ;
      s.add (p2) ;
    u1.setPhones (s) ;
      ses.delete (u1) ;
      tx.commit() ;
  // child to Parent
  User u2 = new User() ;
    u2.setUserId (102) ;
  PhoneNumber q1 = new PhoneNumber() ;
    q1.setPhone (81818181) ;
    q1.setParent (u2) ;
```

```
PhoneNumber q2 = new PhoneNumber();
    q2.setPhone(71717171);
    q2.setParent(u2);
PhoneNumber q3 = new PhoneNumber();
    q3.setPhone(91919191);
    q3.setParent(u2);
Set s1 = new HashSet();
    s1.add(q1);
    s1.add(q2);
    s1.add(q3);
  u2.setPhones(s1);
    ses.delete(q1);
    ses.delete(q2);
    ses.delete(q3);
  tx.commit();
 ses.close();
}//main
}//class
```

→ For complete source code of above sources representing Bidirectional one-to-many association refer appln (12) available in page no :- 77-82 in course material

→ inverse = "true" is optional in association-mapping, Collection-mapping related tags. But it is recommended to place to give hint to hibernate s/w that the current property also participate in reverse association mapping of Bi-directional mapping

→ By default all association mappings that are done by using association mapping tags and collection mapping tags will enable lazy loading by default

To disable this lazy loading (or) to make these lazy loading as extra lazy loading we can use "lazy" attribute in the above said tags by having

lazy = "false" lazy = "extra" values.

↓ disable lazy loading and goes for eager loading

↳ enables extra lazy loading

→ When lazy = "true" (default value) :-

when query.list() is called in association-mapping environment only empty parent pojo class objects will be created but the records into these objects and creation of child pojo class objects, records in to these child objects will be stored by hibernate s/w only when programmer starts using those objects in his application.

→ When lazy = "false" :-

When query.list() called in association-mapping environment all objects will be created and initialized with records selected from the table irrespective of whether these objects are used in the application (or) not

→ When lazy = "extra" :-

All objects creation and initialization with table records (even for parent pojo objects) only when application starts working with result data. This is more lazy than lazy = "true" mode.

Q) ⇒ What is the difference b/w HQL joins query with fetch keyword and without fetch keyword ?

Ans:- When HQL queries are executed without fetch keyword

① lazy initialization takes place for child pojo class or right side pojo class related objects.

② generates two separate SQL queries one for parent table records (left side table) another one for the associated child table records (right side)

When HQL queries are executed with fetch keyword

① Eager loading takes place for child pojo class object

② single select query will be executed to get parent ~~pojo~~ table and associated child table records.

→ refer example appln on HQL Joins HQLJoinsClient.java application in pages 2-80 & 81

⇒

You can go for unidirectional and bidirectional one-to-one association in 2 ways

① Based on primary key based one-to-one association

② Based on Foreign key based one-to-one association

→ For unidirection one-one association it is recommended to work with primary key base one-to-one association mapping.

→ For bidirectional one-one association it is recommended to work with "foreign key" based one-to-one association

→ If there is possibility of taking parent pojo class object identity value as child pojo class object identity value then we can go for one-to-one (FK) based association.

ex:- every Student of College should have library membership the relationship b/w student and library membership is one-to-one r/s. since student ID can be taken as library membership id, we can use one-to-one primary key association here. that means there is no need of foreign key column in child table or parent table. This indicates even though parent & child tables are not there in r/s their POJO classes and POJO class objects are participating in relship based on identity value.

Student-tab:-

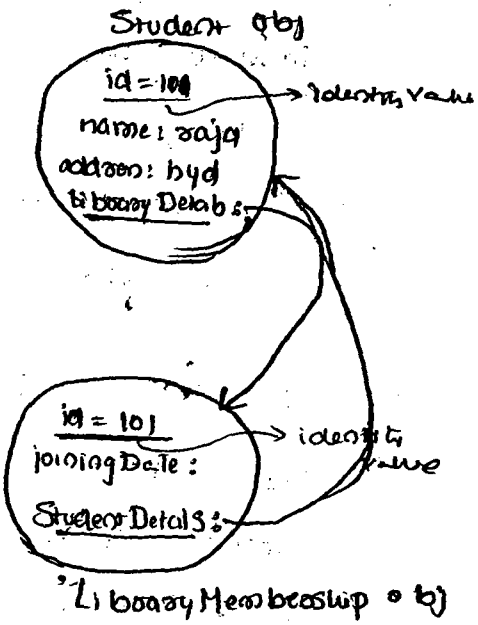
sid (pk)	name	address
101	raju	hyd
102	chakri	hyd

lib-membership:-

sid (pk)	jandate
101	28-jan-2009
102	21-dec-2009

related pojo classes :-

```
public class Student
{
  int id, String name, String address;
  LibraryMembership libraryDetails;
    getXXX(), setXXX
    ==
}
public class LibraryMembership
{
  int id; Date joiningDate;
  Student StudentDetails;
    getXXX, setXXX()
    ==
}
```



→ In the above diagram & discussion student object and library membership object are there in one-to-one relationship because the identity value of library membership obj is nothing but the identity value of student object. Because of this objects, objects are there in one-to-one r/s even though there related tables are not having foreign key constrained column.

→ In one-to-one primary key association the Foreign algorithm is utilized to generate identity value of child pojo object based on the identity value of parent pojo object. This foreign algorithm is identity value generator algorithm and no way related with FK constraint of DB s/w.

→ Based on the above discussion the id member variable of library membership class should be configured in mapping file having foreign algorithm as identity value generator.

=

4/9/10

→ If there is a guarantee every pojo class object will have one child pojo class object go for primary key based one-one association. If that guarantee is not there go for Foreignkey based one-to-one association.

→ Every student of college will have one library membership. so we can go for primary key based one-to-one association mapping b/w student, library membership details.

→ Every citizen of india need not to have license bu every license belongs to one citizen, so we can go for Foreign key based one-to-one association b/w citizen, licence details.

→ Fore example appln on pk based one-to-one association refer app-9 given in page no: 68-70

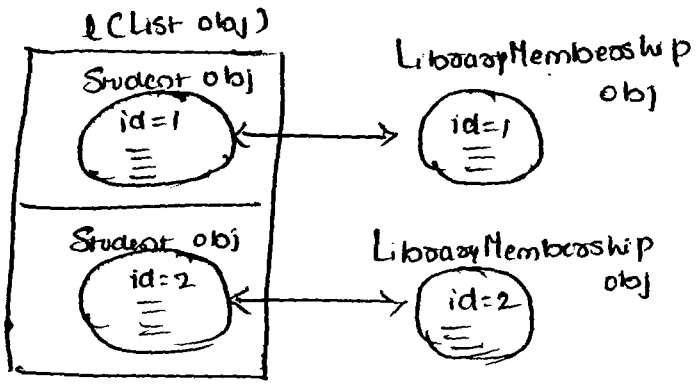
⇒ supplementary client application for application-9 (pk based one-to-one association to select and delete records.)

SelectTest.java

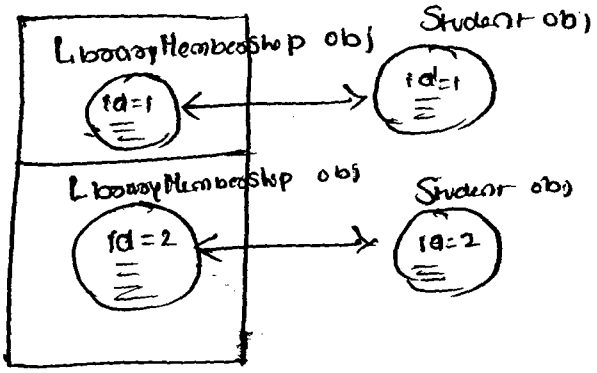
```
import org.hb...
=
// parent to child POJO ~~class allows to pass~~
Query q1 = session.createQuery("from Student");
List l = q.list();
for(int i=0; i<l.size(); i++)
{
Student st = (Student) l.get(i);
s.o.p("parent :" + st.getId()+" "+st.getName()+
" " + st.getAddress());

LibraryMembership lib = st.getLibraryDetails();

s.o.p("child : " + lib.getId() + " " + lib.getJoiningDate());

}
s.o.p("------");
// child to parent
q1 = session.createQuery("from LibraryMembership");
l = q1.list();
for(int i=0; i<l.size(); i++)
{
LibraryMembership lib = (LibraryMembership) l.get(i);
s.o.p("child :" + lib.getId() + " "
+ lib.getJoiningDate());
Student st = lib.getStudentDetails();
```

```
S.o.p ("parent" +st.getId() + "  " +st.getName() +"  " +st.getAddress());

// for deletion of records (parent to child)

Transaction tx = ____ ;

LibraryMembership lb = new LibraryMembership() ;

lb.setId (1);

Student st = new Student ();

st.getId (1);

st.setLibraryDetails (lb);

lb.setStudentDetails (st);

session.delete (st);

tx.commit ();

session.close();

}

}
```

parent to child



child to parent



From on 6/9/10 Continere from back-side

6/9/10

Q) what is the diff b/w Restrictions and Expressions clan while working with CriteriaAPI?

ans:- The methods of Restrictions clan allows to pass conditions to criteriaAPI logic in the form of java statements. , where as the methods of Expressions clan allows to pass same condition as SQL query statements.

methods of Restrictions clan are GE, LE, LD, GT and etc.

the methods of Expression clan are sql (-)

=

One-One association (FK) :-

in one-one(FK) association mapping, child table should contain Foreign key column, pointing to PK column of parent table. This FK column should not ~~have~~ allow duplicates and NULL values:, so this FK column should be applied with unique, not null constraint

=> If there is a guarantee every parent POJO clan object will have one associated child object then go for PK based one-one Association.

If there is no guarantee every parent POJO object will have one child object but there is a guarantee for every child object to have one association parent object, go for " FK based one-one association. mapping.

ex:- Citizen and licence containe one-one rls since there is no guarantee every citizen to have

license but there is a guarantee for every license points to one citizen., so go for FK based one-one association mapping in this situation

Eg:- For example application on FK based one-one Unidirection based association refer App (10) given in page no 8-70-73

=> If child POJO class object can take, its identity value from the identity value of, the associated Parent POJO class object, then use one-one PK association.

=> If child and Parent POJO class object are looking take separate and independent identity two values then use One-One FK Association.

** In FK based one-to-one Association we can't work with one-one tag because this tag is not having column attribute to specify Foreign key column name. based on which the association will be formed. to over come this problem use <many-to-one> having unique="true" attribute.

To configure the property that is representing relationship refer line:- 1878 - 1878 of booklet.

programmers_projects

programmer_id (CFK with parent PK)	project_id (CFK with parent pid)
101	1001
101	1002
102	1001
103	1002
103	1002

composite PK

→ First table shows Parent data

→ second table shows child data

→ third table shows many-many association b/w Parent table and child table data.

⇒ For example appln on many-many association mapping refers app(11) given in 73-77

Bag :- the Bag DS, given by hibernate. It is same as List DS of Collection F/w. So Bag DS also allows duplicates.

to configure Bag DS type property in our HB mapping file, we have to use the tag called 'Bag'

For example application on Bag DS refers Appli(17) of page no :- 98-101

→ The app (17) of book.hbm, shows association b/w (one-to-many) speaker and his sessions, speaker and his phone numbers, that means one parent table is there having two child table.

→ If HB POJO class property type is java.util.Li
(or) org.hb.mapping.Bag then we can conf
that property in our HB Mapping file either by
using <List> tag or <bag>.

⇒ DB table is having single column, then programmer
take the predefined wrapper classes Integer, Flo
String as HB POJO classes.

especially this is useful, when associated table
(child table) of certain table is having only one col

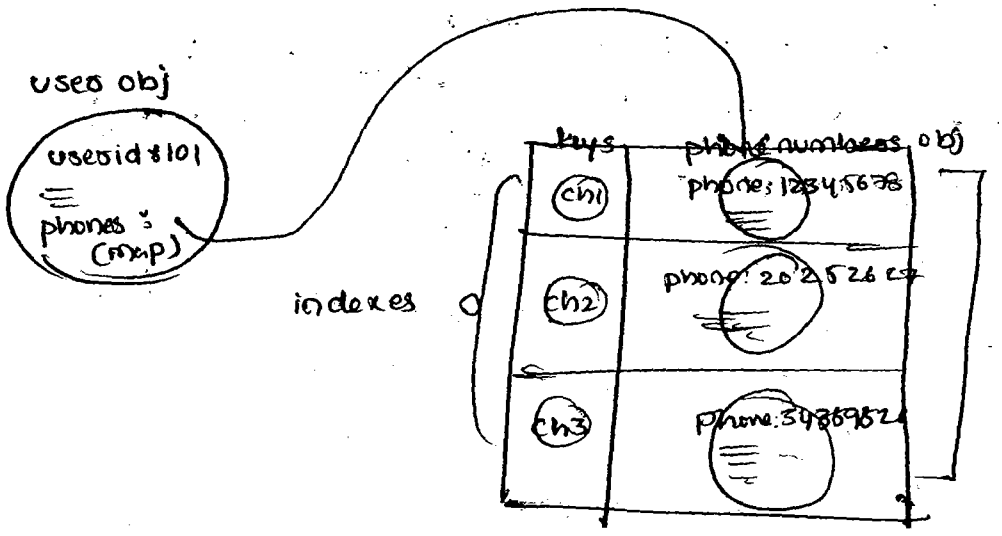
8/9/10

Map :-

each element of map DS, contains one key and one v
key can be any object, and value can be any object, we co
use key here to maintain index (or) count of value relat
objects.

In hibernate, we can use map DS property of parent POJ
to maintain one (or) many child POJO class object to achieve
one-many r/s.

in this process, we can provide index for the child P
class objects that are linked with parent object.



→ If you want to link child objects with parent objects without providing indexing for child object, we can use Set, List, Bag type property in Parent POJO class to ~~link child POJ~~ represent child POJO class objects.

=

⇒ for example applⁿ on Map ~~tagge~~ tag based collection mapping refer supplimentary handout :- 8/9/10

9/9/10

Cache - Buffer is a temporary memory, that holds result (or) data ~~across the multiple~~ and uses that data (or) result across the multiple same request given by application.

The process of storing result in cache (or) buffer and using that result across the multiple request or ~~execution~~ execution of the application is called "caching" (or) "Buffering".

→ in client-server application, if caching is enabled at client side, it reduces network round trips b/w client and server applications.

→ every browser window contains buffer and stores, Request related response result, the buffer of browser window Reduces N/w round trip b/w browser window and webserver across the multiple same request given by browser window.

→ The caching (or) buffering done in HB based application stores results collected from the DB s/w in the form of HB POJO class objects., this Reduces N/w Round trip b/w HB based client application and DB s/w across the multiple same request given by HB based client applⁿ

-> HB supports two levels of caching

① Ist level caching \ L1 caching (HB session object)

② IInd level caching \ L2 caching (HBSessionFactory object)

=> Ist level cache is built in cache.

=> IInd level cache is configurable cache.

-> IInd level cache will be enabled only when program configures it manually.

-> every session object contain one level I cache storing its session object specific results.

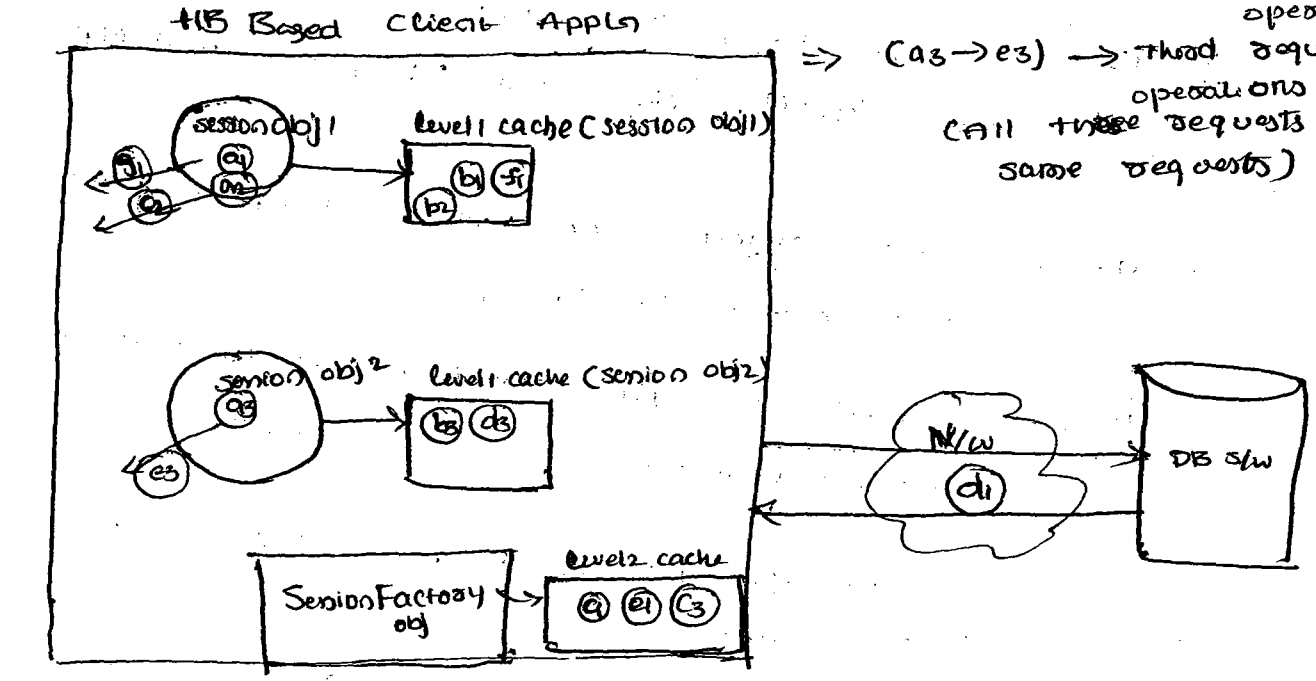
-> level II cache is called Global cache, bcz, we can store and manage the results generated by all HB Session objects.

=> $(a_1 \rightarrow g_1)$ -> First request operations

=> $(a_2 \rightarrow c_2)$ -> Second request operations

=> $(a_3 \rightarrow e_3)$ -> Third request operations

(All three requests are same requests)



a_1 => session object ① gives initial request (select query)

b_1 => searches for that query request in level ① cache of session object ① (not available)

=> c_1 => searches for query result in level 2 cache (not available)

=> d_1 => HB application interact with DB S/W and gets query execution related results.

=> e_1 => HB S/W registers the results in level ② cache

=> f_1 => HB S/W also registers the results in level ① cache of session object ①

=> g_1 => session object ① gives the result to application

=

=> a_2 => session object ① gives same request once again

=> b_2 => searches for the result in level ① cache of session object ①. (available)

=> c_2 => session object ① collect the results from its own level ① cache and pass to client appln.

=

=> a_3 => session object ② of same client appln gives same request to DB S/W

=> b_3 => searches for the result in level ② cache of session object ② (not available).

=> c_3 => searches for the result in level ② cache (available)

=> d_3 => collect the result from level ② cache and registers with level ① cache of session obj ②.

=> e_3 => session obj 2 gives the result to client appln

⇒ Results of level ①, level ② caches will be stored in form of HB pojo class objects.

⇒ the Results of level ① cache and level ② cache updated automatically at regular intervals.

⇒ we can specify parameters to disable level ② completely after certain amount of time or to d Results Related pojo class objects from level ② cac after certain amount of time.

⇒ the level ① cache, keeps track of all the changes on HB pojo class in a transaction and instead generation multiple SQL Queries, for this multiple cha it generates only one SQL update Query reflecting all changes done on that object at end of the transactio

⇒ for Related info on level ① caching page 106 of the booklet.

⇒ HB also support Query level cache, as sub position of le cache. this Query cache holds Native SQL Query Resu directly in its original format not in the format HB class object.

⇒ It is recommended to empty level ① and level ② cach time to time to know the changes done in DB sid time to time.

⇒ To get control on, level ① cache we need to work with built methods that are invocable on HB session object.

① session.evict(pojo obj) :-
Removes given pojo object, from level ① cac

② ses.clear() :-
empty's level ① cache associated with current s
Removes all HB pojo objects from cache.

close() :-

closes HB session, so also closes the level-1 cache associated with, current session object.

=

FAQ) what is dirty state object in HB environment?

as:- If you modify, Data of peristant state HB POJO class object manually in the client application, these changes will be synchronized with associated records in the DB table, only when session is flused by calling

session.flush();

Before calling this flush() changes done in peristance state POJO class object will be pending mode to synchronised with DB in this situation [before calling session.flush()] we can say HB POJO class object is these in dirty state.

session.isDirty() method return 'true' → if session object is having any dirty state objects.

=

```
EmpBean eb = (EmpBean) ses.get(EmpBean.class, new Integer(1010));
    S.O.P ("session is dirty mode?" + ses.isDirty()); → false
    eb.setFname("rakesh");
    S.O.P("session is dirty mode?" + ses.isDirty()); → true
    ses.flush();
    S.O.P("session is dirty mode?" + ses.isDirty()); → false.
```

=

In the above code state of 'eb' object is dirty state after calling eb.setFname(), before calling ses.flush();

=

13/9/10

IInd level cache is configurable cache in HB Level, there are multiple vendors supplying s/w to enable Second level cache in HB Applications.

some vendors are supplying the second level cac related jar files along with HB s/w installation.

⇒ some example second level and opensource cache s/w's are :-

→ EHCache

→ OSCache

→ SwarmCache (Swarm Cache)

→ JBOSS TreeCache and etc.

→ most of the time developers prefers with EHCache, bcz it is fast, light weight, easy to use cache.

→ for details about various second level cache s/w's and their, caching strategies refer page no

⇒ Procedure to enable 'EHCache' as second level cache in c HB Application :-

step①:- every second level cache will have, one provide class name, by to know the provided class name for EHCache.

→ for EHCache, the provided class name is :-

org.hibernate.cache.EhCacheProvider

Note:- refer hibernate-home \ etc \ HB.properties.file for various etc second level clames related providers class names.

Step②:- Configure EhCache provider class name, in HBConfiguration file.

In HBConfiguration file:-

<session-factory>

=

<property name="hibernate.cache.provider_class">

~~org.hibernate.cache.EhCacheProvider</property>~~

org.hibernate.cache.EhCacheProvider</property>

=

Step③:- Add HB-Home\lib\ehcache-1.2.3.jar file to the class path environment variable.

Step④:- enable second level cache in HB Application

In HBConfiguration file:-

⇒<property name="hibernate.cache.use_second_level_cache">

true </property>

=

Step⑤:- write Ehcache.xml file along with other resources of HB application, specifying the parameters related to Ehcache.

ehcache.xml :-

```
<ehcache>                                          (1)
   <diskStore path="user.dir" />
   <defaultCache maxElementsInMemory="1000"        (2)
                 eternal="false"                   (3)
                 timeToIdleSeconds="120"           (4)
                 timeToLiveSeconds="200"           (5)
                 overflowToDisk="true"             (6)
   />
</ehcache>
```

=

Note:- all second level caches uses primary memory (RAM), if needed we can also use disk memory (HD) as an enhancement memory.

Note:- for details regarding ①–⑥ numbers refer the xml comments given in HB-Home/etc/hibernate.cfg.xml file.

Step ⑥:- compile and execute HB client application.

=

⇒ Query Cache is part of second level cache, it basically stores HQL, NativeSQL Queries generated Results.

⇒ procedure to enable Query Cache on your HB Application.

Step①:- enable Query Cache., from HB Configuration file.

<property name = "hibernate.cache.use_query_cache"> true </property>

Step②:- make HQL, NativeSQL Query Results as cachable Region from client application.

Query q1 = ses.createQuery("from EmpBean");

q1.setCacheable(true);

List l = q1.list();

=

Step③:- set logical name for ~~Query~~ QueryCacheRegion from client application.

Query q1 = ses.createQuery("from EmpBean");

step (4) :- make sure that IInd level cache also enabled, before enabling the QueryCache.

Performing various operation on second level cache from Client appln :-

Note:- use SessionFactory object for these operation

→ factory.close() → releases second-level cache, associated with SessionFactory object

→ factory.evict(EmpBean.class) → Removes all emp bean pojo class object, from IInd level cache.

→ factory.evictCollection("phones") → If phones property configured in HB mapping file, is related to Collection f/w DS, all Collection f/w DS objects representing phones property will be removed from second-level-cache

~~factory.~~

→ factory.evictQueries(); :- Removes HQL, NativeSQL Query results from default Query Cache Region

→ @ factory.evictQueries("test"):- Removes HQL, NativeSQL Queries from the Named Query Cache Region called "test"

Transaction Management:

→ The process of combining set of related operations into single unit and executing them applying "do everything or nothing" principle is called transaction management.

while executing sensitive B.L and P.L, we must deal with Transaction Management.

ex:- Transfer money operation is composed with two operations ① withdraw amount from source account ② deposit amount into destination account,

so we need to execute these two operations by applying do everything or nothing principle through transaction management.

→ Transaction Management applied on the code, applies ACID properties support on DB s/w.

A → (Automaticity), C → (Consistency), I → (Isolation), D → (Durability)

① → The process of combining related sub operation into single unit is called 'Automacity'

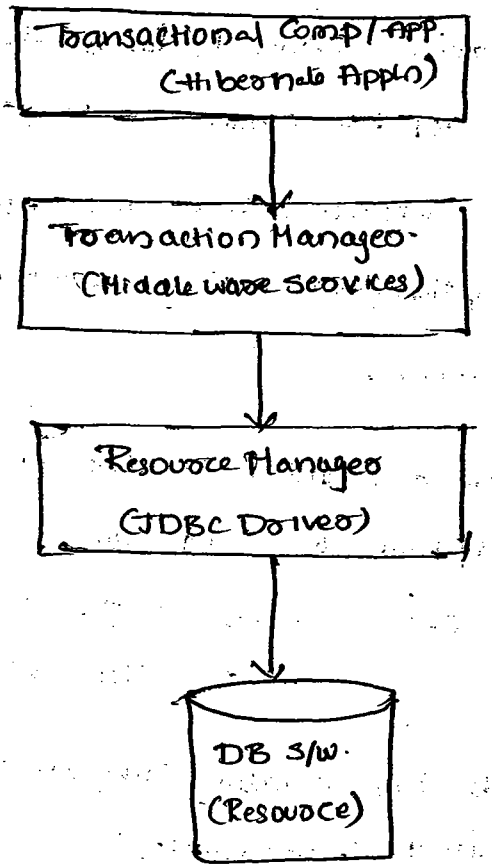
② → The process of getting guarantee, that rules kept on DB s/w (like balance must not be negative) or not violated at the end of the transaction even though they are violated in the middle of Transaction is called "Consistency"

③ → The process of preventing concurrent operation on DB s/w from multiple users and application by applying 'locks' is called "Isolation"

④ → The ability of bringing the DB s/w back to normal state by using log files and Backup files when DB is crashed and using DB data for long time is called "Durability".

14/9/10

Architecture of Transaction Management :-



→ The appln (or) component on which transaction management is enabled, is called Transactional Application/component.

→ Transaction Manager is responsible to begin Transaction, to commit (or) roll back the transaction on the application code.

→ Based on no. of resources (DB s/ws) that are involved, there are two types of transaction

(1)

① Local Transaction :-

all the operations of Application code, on which transaction management is enabled will deal with single resource (DB s/w).

② ~~Distributed~~

ex :- Transfer money operation b/w ~~same~~ two accounts of same bank.

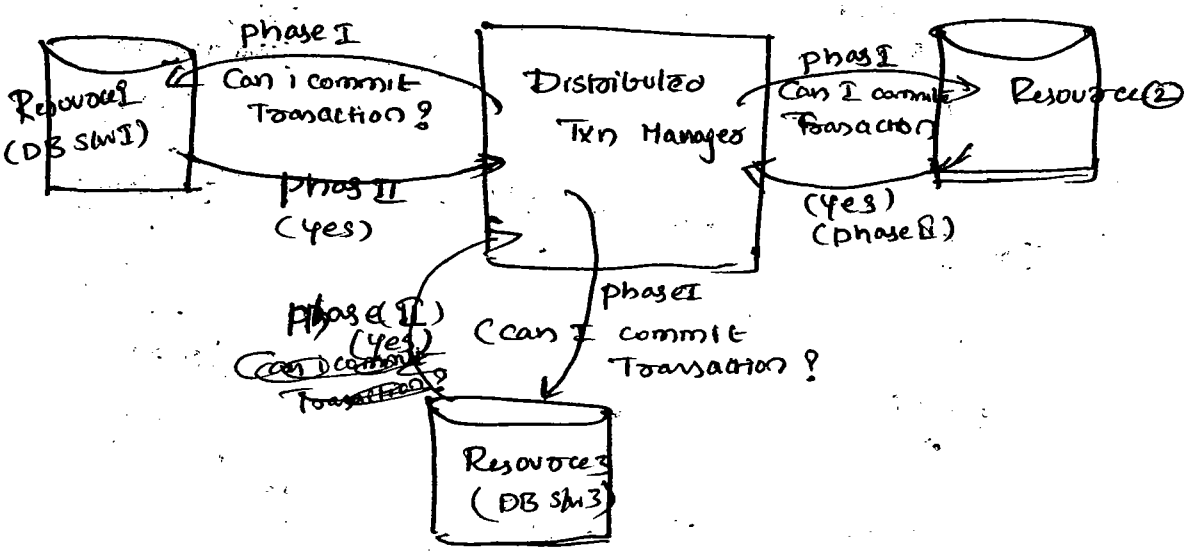
③ Distributed Transaction :- If multiple Resources or DB s/w are involved for various operations of Application code on which transaction management is enabled then that is called 'Distributed Transaction'

ex :- Transfer money operation b/w two accounts of two different banks.

Distributed transaction runs based on 2PC protocol (Two phase commit protocol).

=

① 2PC protocol :-



According to 2PC protocol, in the phase I, Distributed Transaction manager asks all DB S/W permission to commit the Transaction,

in phase II, all the DB S/W's gives permission to commit the Transaction, then the distributed Transaction will be committed (OR) the distributed Transaction will be roll backed.

=

=> HB support the local transaction management, but does not support distributed transaction management.

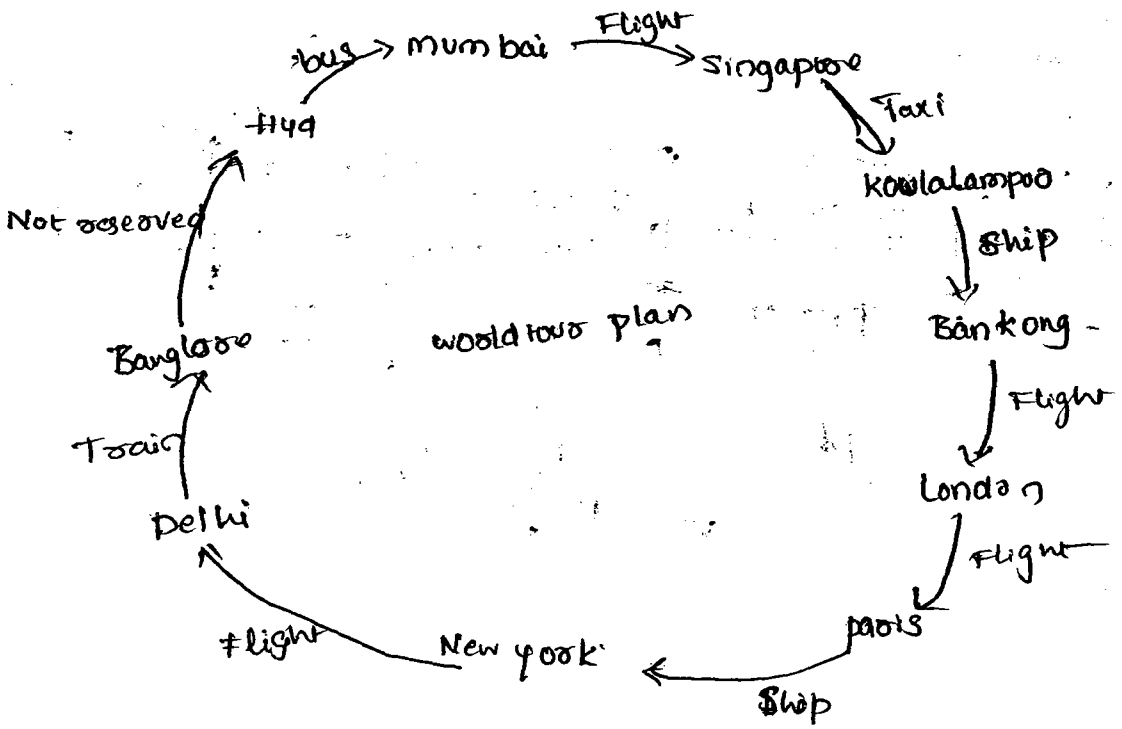
=> Spring, EJB Technology support both, distributed and local transaction management.

=

=> There are two transaction models

① Flat Transaction

② Nested Transaction.



→ When the above tour plan is given to a S/w application that runs with ~~nested~~ "Flat Transa[ction]" if one (or) other journey ticket are not available it will cancel all other journey tickets.

Because each journey ticket booking opera[tion] will be taken as direct operations of main transac[tion] so if one operation fails all other operations will [be] rollbacked.

Flat Transaction:-

```
MainTransaction {
        journey1 Ticket booking,
        journey2   "      ",
           :
        journeyn   "      ",
}
```

→ When the ~~above~~ tour plan application give[n] ~~to~~ to that runs with "nested transaction", eventho[ugh] one or other journey tickets are not available it will be ~~[struck]~~ conformed the remaining Journey ticket because, each journey ticket booking operation will be taken as the subtra[nsaction] of main transaction,

So the success or failure of one subtra[nsaction] does not effect other subtransactions of main Transaction.

Nested Transaction :-

```
Main Transaction {
    subTxn1 {
        ==
        journey1
        ==
    }
    subTxn2 {
        == Journey2
    }
    ==
    subTxn3 {
        == Journey3
    }
}
=
```

*** → EJB, HB, and spring support Flat Transaction.

→ EJB & HB doesn't support Nested Transaction.

→ Spring support Nested Transaction.

=

⇒ Sample code that performs Transaction management in HB environment :-

```
public void bm1 ()
{
    Transaction txn = null;
    try
    {
        tx = ses.beginTransaction();

        persistance operation 1;
        persistance operation 2;
        :
        persistance operation n;
```

```
        txn.commit();
    } //try
    catch(Exception e)
    {
      try
      {
        txn.rollback();
      }
      catch(Exception e1)
      {
        =
      }
    } //outer catch.
  } //method
    =
```

⇒ Example application to perform Transfer money operation b/w two account ob same bank having Transaction management?

> Alter table account add constraint xyz primary key(acno);

step①:- create account table in oracle db slw.

> ~~select~~ * create table account(acno number(5), pk, accname varchar2(20), bal number(8,2));

step②:- Lanch MyEclipseIDE and create java project having name "TxHBApp"

step③:- Create DB profile for oracle by using myeclipse DB explorer.

step④:- Add HB capabilities to the project.

step⑤:- perform reverse engineering on account table of oracle DB slw, by using DB profile.

NOTE:- After reverse engineering you get, account.java as POJO class, account.hbm.xml as mapping file.

step⑥:- develop the following client application, having the support of Transaction Management

...ent.java :-

```
import org.hibernate.*;
import p1.HibernateSessionFactory;
public class TestClient {
public static void main(String args[]) throws Exception

Sessions ses= HibernateSessionFactory.getSession();
  Transaction tx=null;
  try
   {
    tx = ses.beginTransaction();
    //operation1 (with draw operation from src amoun
Query q1 =ses.createQuery("update Account set balance =(balance-?)
                                        where acno=?");
   q1.setLong(0,4000);
   q1.setLong (1,101);
    int res1 = q1.executeUpdate();

// operation2 (deposit operation in dest amount)
  Query   q2 = ses.createQuery ("update Account set balance=
                              (balance+?) where acno=?");

   q2.setLong (0, 4000);
   q2.setLong (1, 102);
   int   res2 = q2.executeUpdate();
  if (res1!=0 && res2!=0)
   {
    tx.commit();
    s.o.p("Tx is commited");
   }
  else
   {
     tx.rollback();
    s.o.p
   }
  } //try
 catch(Exception e)
 {
  try
   { tx.rollback();
  } catch(Exception e1) {}
```

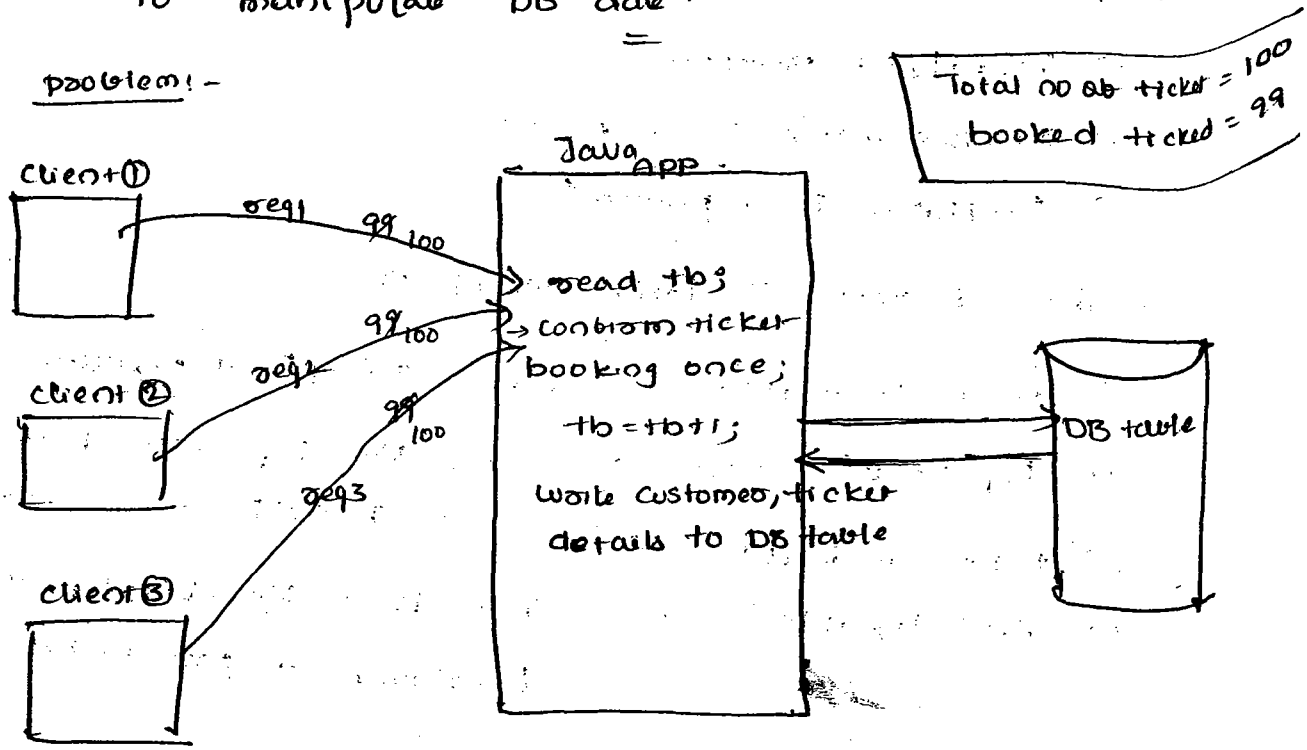
=> add ojdbc14.jar to built path of the project.

=> run the client application.

15/9/10 **ISOLATION :-**

-> If multiple users trying to access (or) manipulate DB data simultaneously & concurrently there is a possible of getting corruption to the DB data. To prevent this problem apply locks on DB at various levels by using 'ISOLATION' levels & allow only one user (or) one Appln at a time to manipulate DB data.

problem :-



In the above Diagram, Application and DB both are allowing concurrent operation, so there is a possibility of booking same ticket for multiple passengers.

To prevent this problem use Synchronization concept at java class level and apply locks on DB s/w by using Isolation levels.

-> If Application or DB is allowing another user (or) client to use the same data before completing

... operation is called "Concurrent operation" (or) "simeltanious operation" (or) "Interleaving operation"

→ when DB allows, these Interleaving or simeltanious operation there is a possibility of getting following problems.

To solve these problems we need to use different Isolation levels.

problems are:-

① Dirty Read Problem
② Non-repeatable Read problem
③ Panthem Read problem.

The solution Isolation levels are :-

① Read committed → solves Dirty Read problem
② Repeatable Read → solves Non-Repeatable Read problem, and Dirty-Read problem.
~~③ Panthom Read~~
③ Serializable → Dirty Read, Non-Repeatable Read, PanthomRead problems.

=> we can set all these Isolation levels on DB sides being from java Applications by using JDBC code (or) HB Code (or) any other persistance logic code.

⇒ Dirty Read problem :-

⇒ If user A, user B have joint account holders in B.

⇒ DB sw is allowing uncommitted Reads.

User A	User B
1) user A reads the balance of Joint a/c (Rs: 5000)	
2) user A begin Txn and deposits Rs: 3000 into joint A/c then bal = bal + amt; (Rs: 8000)	
	(3) user B withdraws Rs: from account (joint) bal = bal - amt; (Rs: 1000)
(4) user A abort / roll backs Txn, so the balance becomes Rs: 5000. Here withdraw operation done by user 'B' is called "Dirty Read operation"	

→ To solve above Dirty Read problem, Apply Read Committed Isolation level on DB sw. This mai Applications to read only committed data in DB

→ In most of the DB sw's Read Committed is default Isolation level.

ex:- ORACLE, MySQL.

Non-Repeatable Read Problem :-

User A

User B

(1) At begining of Tx User A gives select query and gets two records (let us assume 10 records) from DB s/w.

(2) User B executes update query that updates selected records of User A in DB s/w.

(3) At the end of Tx User A, re issue same select query he gets 10 records with modified data. this is called "Non-repeatable Read" problem.

=

To solve above Non-Repeatable read problem use Isolation level Repeatable Read, which applies 'write locks' and 'Read-committed' methanism on DB. Due to this, ~~so~~ This Isolation levels also solve Dirty Read problem.

=

=> Panthom-Read problem:-

User A	User B
① At begining of Tx user A gives select query and gets few records (let us assume 10 records) from DB Slw	② User B inserts more rec (let us assume 4 records) Same table which also sa select query condition of us
③ At the end of Tx user A re issues same select query and he gets 14 records. getting these 4 extra records is called panthom read problem.	

To solve Dirty Read, Non-Repeatable Read, Panthom problem serializable Isolation level, it uses appl Read & Write Locks on DB.

TL

=> TL (or) PL (or) DB designing team decides the Isol level but programmer is responsible to configure apply this Isolation level on DB Slw.

-> DB slws doesnot support all Isolation level for example

-> Oracle doesn't support serializable Isolation level.

-> mySQL support all Isolation levels.

⇒ Setting "Isolation level" on DB S/W, from JDBC code :-

≡

```
con.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
                    (OR)
con.setTransactionIsolation(8);
```

here all possible values are :-

JDBC

- Connection.TRANSACTION_READ_COMMITTED → 2
- Connection.TRANSACTION_REPEATABLE_READ → 4
- Connection.TRANSACTION_SERIALIZABLE → 8

→ setting Isolation level on DB S/W from HB application :-

In HB Configuration file,

```
<property name="hibernate.connection.isolation">4</property>
                    (or)
<property name="hibernate.connection.isolation">
        REPETABLE_READ </property>
```

Other possible values are :-

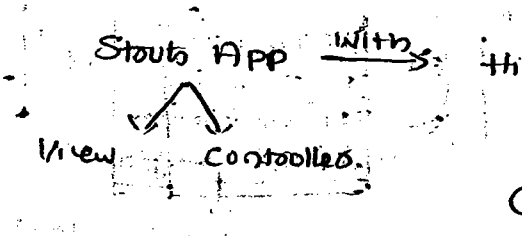
HIBERNATE

- READ_COMMITTED → 2
- REPETABLE_READ → 4
- SERIALIZABLE → 8

16/9/10

STRUTS WITH HIBERNATE :-

Struts with Hibernate Application means, we want to make struts application interacting with DB s/w by using HB persistance logic.



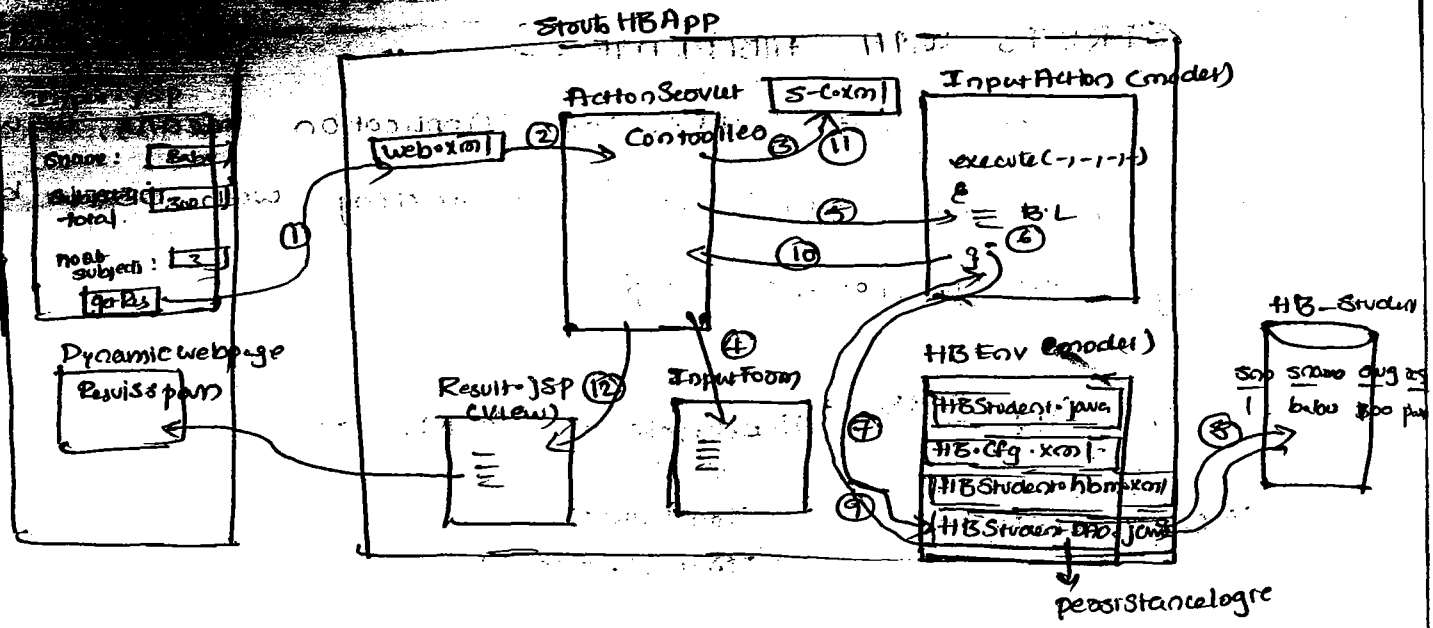
→ In struts with Hibernate Application, struts Action class contains Business Logic and uses P.L logic of HB to interact with DB s/w.

→ The java class are component that contains purely persistance logic and separate this P.L from other logics of the application is called DAO (Data Access Object).

→ the DAO makes P.L as reusable, flexible logic.

- Reusable means the persistance logic of DAO can be used by multiple Resources of the application.
- flexible means any modification in P.L, does not effects other logics.

→ The MyEclipse IDE can generate HB persistence logic based DAO dynamically.



=

There are two approaches, to develop Struts with Hibernate Application.

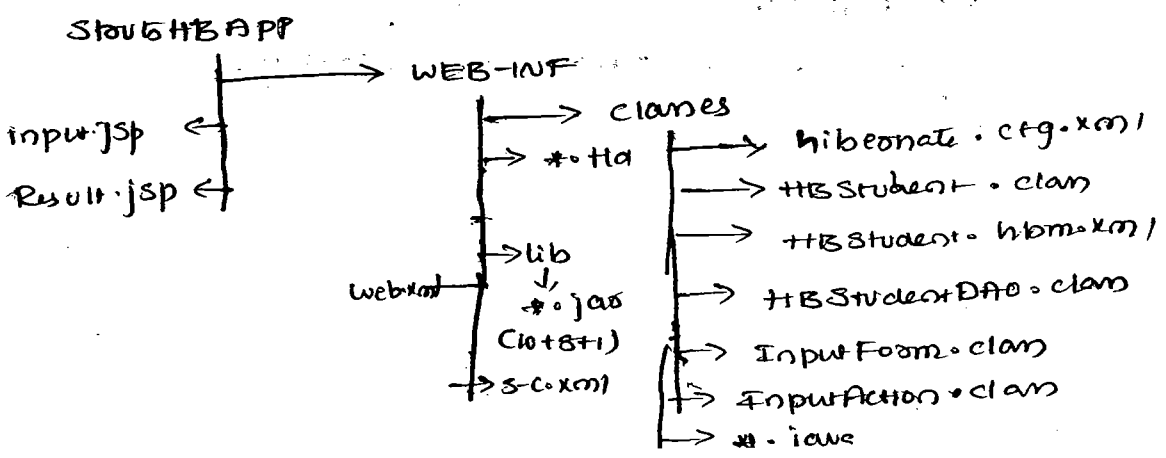
Approach ① :-

By creating HB session object in Struts Action class (or) DAO class

Approach ② :-

By creating HB session object in user-defined plug-in class of Struts.

→ If above application is developed manually based on Approach ① Deployment Directory Structure as shown below.



Jar files in the class path:-

→ servlet-api.jar

→ struts-core-version.jar

→ hibernate3.jar

jar files Required in WEB-INF\lib folder

→ 8 HB jar file

→ 10 struts jar file

→ 1 ojdbc14.jar.

→ Procedure to develop above diagram based Appln, by using MyEclipse IDE

Step①:- create web project in MyEclipse IDE.

File
↳ New
↳ project
↳ web-project
↳ Next
↳ StrutsHBProj
↳ finish.

Make sure that HB-student table is there in oracle DB slw, by having pk constraint on sno column.

HBStudent

sno (pk)	sname (vc2)	total (num)	AVG (num)	Result (vc2)

Step② create ORACLE DB slw Related DB profile in My-Eclipse IDE.

Step③:- Add HB capabilities to the project

Right click on project
↳ myeclipse
↳ Add HB capabilities
↳ select HB-3.1 core Libraries
HB-3.1 Advanced
↳ Next
↳ Next

& orap (already create DB profile)

└> Next

└> de-select Create SF class

└> finish

add show_sql property

Step④ :- perform HB Reverse engineering on HB_Student table.

goto DB Browser window

└> Right click on HB_STUDENT TABLE

└> HB Reverse engineering

→ Java src folder :- /Struts HB proj/src

→ select 1st, 2nd, 3rd check box.

→ deselect 4th check box.

→ select 5th check box. (JAVA DataAccess Object)

DAO Type :- o basic DAO

└> Next

└> ⊙ HB DType.

└> Next

└> finish

NOTE :- change IDE generate algorithm to increment algorithm in the generated mapping file (HBStudent.hbm.xml)

NOTE :- Add Transaction management support for non select operations related logics in the generated DAO class.

Step⑤ :- Add struts capabilities to the project.

Rightclick on project

└> My Eclipse

└> Add struts capabilities,

→ select struts 1.x

↓

finish.

Step⑥ :- Add FormPage, FormBean class, Action Class, ResultPage

FormPage :- input.jsp InputForm InputAction Result.js

~~Form~~

Right click on project
→ New
→ Others
→ MyEclipse
→ Web Struts
→ Struts 1.2
→ Struts 1.2 Form, Acti
Next

@ Create All mentioned class

=

Step⑦ :- Right bollowing code, in the execute(-,-,-,-) of InputAct class.

InputAction.java

=

execute(-,-,-,-)
{
InputForm if1 = (InputForm) form;
String name = if1.getSname();
int nosub = Integer.parseInt(if1.getNosub());
int total = Integer.parseInt(if1.getTotal());
float avg = (float) total/nosub; String res = null;
if(avg<35) res = "fail"
HBStudentDAO dao = new HBStudentDAO(); else res = "pas
HBStudent st = new HBStudent();
st.setSname(name);
st.setTotal((long)total);
st.setAvg((double)avg);
st.setResult(result);
dao.save(st);
request.setAttribute("result", res);
return mapping.findForward("success");
}

Add Result.jsp page to the Web-Root folder of project,

 Student Result is <%= request.getAttribute("result") %>

Step④:- Configure External Tomcat server to MyEclipse IDE

Window
↳ Preferences
↳ myeclipse
↳ Application servers
↳ Tomcat
↳ Tomcat 5.x
④ enable
D:\Tomcat 5.5
Apply
↳ OK

Step⑩:-
Start Tomcat server from MyEclipse IDE.

goto server icon ( ) Tool bar
↳ Tomcat 5.x
↳ start

⇒ Deploy the project in Tomcat Server from MyEclipse IDE

goto Deploy in the tool bar.
↳ Add
↳ server : Tomcat 5.x
Finish
↳ OK

Step⑪:- Test the Application.

⑧ open browser window from tool bar :-
↳
http://lh:2020/HBStudy

17/9/10 Struts with Hibernate using PlugIn :-

If we want to execute certain logic in struts appln, only for one time, for the all request coming to all the action classes, then used define plug-in class and place logic in that plug-in class.

The java class that implements org.apache.struts.action.PlugIn interface is called plug-in class in struts.

→ every PlugIn class must be configured in struts config file by using <plug-in> tag.

→ the struts PlugIn class execute only one time, that is when ActionServlet object is created in struts f/w s/w is Activated.

→ If you want to make all Actionclasses of struts appln using single HttpSession object for all the request, then place that logic in the used define PlugIn class of struts application. and use those objects in all action class by receiving them in the form of ServletContext attribute values.

** ServletContext attributes are global attributes in web-app so they are visible all the web-resources of web-appln.

→ procedure to develop yesterdays application without DAO class and by adding used defined struts-plugIn class :-

Note :- used define PlugIn class of struts application always resides in WEB-INF classes folder of struts application.

same as previous application.

Step④ :- perform HB Reverse engineering on HB_student table. [don't select 'DAO' option]

Step⑤ :- same as previous application.

Step⑥ :- Add user defined struts-plugin class to the project. and place logic to create HBSession object in that class.

Right click on project

→ new → other

→ other → my eclipse

→ web struts

→ struts 2

→ struts 2 plugin

→ Next

PlugIn class: HBPlugIn

Add → HB.

src

→ new

→ class

→ HBPlugIn.

→ Finish.

=> - imports →

public class HBPlugIn implements PlugIn {.

```
import javax.servlet.*;
import org.apache.struts.action.*;
import org.apache.struts.config.*;
import org.hibernate.*;
import org.hb.cfg.*;
public class HBplugIn implements PlugIn {
    Session ses = null;
public void init(ActionServlet servlet, ModuleConfig mcfg)
                    throws ServletException {
    s.o.p("HBPlugIn: init(-,-) method");
```

```
try {
    //create HBSession obj.
    ses = new Configuration().configure().buildSF().openSession
    //get ServletContext obj
  ServletContext sc = servlet.getServletContext();
   //keep HB session obj in ServletContext attributes
      sc.getAttribute("HBSes", ses);
  }
 Catch (Exception e)
  {
   e.printStackTrace();
  }
}//init()
public void destroy() {
     try
      {
       ses.close();
      }
     Catch(..)
      {
      }
    }//destroy
}//class
```

ugIn

Step ⑤:- configure above created PlugIn class with Struts appln, in Struts-configuration file.

right click on project
→ New
→ other
→ myeclipse
→ webstruts
→ Struts 1.2
→ struts 1.2 plu
↓
PlugIn Class:- HBPlugIn
[finish] ↵

→ The init() of struts PlugIn class, executes during the start up of the ~~Plug~~ Struts Appln, ~~when~~ ActionServlet object is created

→ The destroy() of Struts PlugIn class, executes when Struts Application is un-deployed, stopped (or) reloade

Same as step ⑥ of previous project.

:- write following logic in the execute(-,-,-) of Action class →

step ⑩ :-

class InputAction

{

=

AF execute(-,-,-) throws Exception.

{

InputForm if1 = (InputForm) form;

//read the data

long total = Long.parseLong(if1.getTotal());

String name = if1.getSname();

long nosub = Long.parseLong(if1.getNoSub());

// get Access ActionServlet obj

ActionServlet servlet = getServlet();

//get access to ServletContext obj

ServletContext sc = servlet.getServletContext();

//read HB Session obj from sc.attribute

Session ses = (Session) sc.getAttribute("HBSes");

⑧ //B.logic

double avg = (double) total/nosub;

String res = null;

if(avg < 35)

res = "fail";

else

res = "pass";

// write HB persistence logic to insert the rec.

HBStudent st = new HBStudent();

st.setSname(name);

st.setTotal(total);

st.setAvg(avg);

st.setResult(res);

Step (11) :- Step (10) → Step (14) are same as .

Step (8) → Step (12) of previous appln .

```
Transaction tx = ses.beginTransaction();
    ses.save(st);
    tx.commit();
    //send result to result.jsp
    request.setAttribute("result", res);
    return mapping.findForward("success");
  }
}
```

** Note (1) :- Plugin class creates session object only once and makes all Struts Action classes using that object for all the request

→ The DAO class separate persistance logic from other logics of the application and also makes the persistan logic as reusable logic in multiple Struts Action classes.

So you can add both plugin, DAO support in Struts with HB application.

But the DAO class should use that HB session object that is created in plugin class.

=

⇒ For EJB SessionBean with HB application refer application given in page no :- 92 to 96

=

Annotations Based Inheritance :-

→ Data about data is called "metadata".

→ configuring resources, to pass their details to underlying SW where the resources will be utilized for execution also comes under MetaData operation.

→ configuring servlet in web.xml file, to pass the details of servlet to underlying webserver comes under MetaData operation.

→ In earlier days programmers have taken, the support of xml file, for resources configurations and for MetaData operations.

→ In Recent days, we can also use the java statements called Annotation as alternate for xml files based Annotations.

→ The MetaData Operations done on the resources or on the code of Resources will apply marking and provide different identity for them during execution.

→ xml files based MetaData Operations gives flexibility towards modification, But does not give good performance because reading data from xml files is always quite complex operation.

→ since Annotations are java statements, they give better performance but does not provide flexibility towards modification.

→ Annotations are there in java from JDK initial versions as Documentation related Annotations like

@author, @param, @since and etc.

→ Annotations per programming are introduced from JDK 1.5

Syn:-

→ @ <annotation-name> (param1 = value1, param2 = value2 .....)

(No semicolon at end)

=> Each Annotation is like an XML tag and parameters of Annotations are like XML tag attributes

=> @Override, @Prelog, @Deplicated are built-in annotations of JDK 1.5

=> all high-end technologies that are given based on JDK 1.5, supply their own Annotations to configure Resources of the application.

=> HB 3.1.x onwards, Spring 2.5 onwards, EJB 3.x onwards, Struts 2.x onwards, Servlet 2.5 onwards are supporting Annotations.

=> If Annotations and XML file, both are configured for metadata operation, then settings done in XML file will be effected.

=> If programmer is using XML file, for metadata operations even though the application is already having Annotation based MetaData Configurations, then it indicates that programmer is interested

annotations to configurations, without disturbing the source code.

-> we can apply Annotations at 3 levels

① Resource level (on class or Interface)

② Field level (on the members variables)

③ method level

=

=> To work with Annotations based HB programming use either hibernate 3.4 s/w completely

or

use hibernate 3.2.5 s/w + HB 3.3 annotations software to gether.

=> All HB Annotations of HB programming are designed based on JPA (Java persistance API), EJB 3.x specification

=> all Annotations are special interface having @ interface keyword.

implementation of this interfaces having @ interface keyword who allows to use Annotations for metaData operations.

=> to for API documentation integration annotation of HB pogmming, go to hibernate 3.3 Annotation_home\ doc\ JPA-API\ index.html file.

=> For Reference example application based on Annotations refer the test folder of HB 3.3-annotations homedirectory

=>

19/9/10

There are three strategies to perform Inheritance

① Table per hierarchy

② " " subclass

③ " " concrete class

we can use @Inheritance annotation per Inheritance mapping, use @DescriminatorValue annotation to spec descriminator value.

use @DescriminatorColumn annotation to create certain column in DB table eg. @DescriminatorCol

→ Table per hierarchy strategy

→ @Inheritance (strategy = InheritanceType.SINGLE_TAB

→ Table per SubClass Strategy

→ @Inheritance (strategy = InheritanceType.JOINED)

→ Table per concrete class hierarchy :-

→ @Inheritance (strategy = InheritanceType.TABLE_PER_CLASS

For example application on TablePerClass hierarchy model of Inheritance mapping refer app ① in supplementary Handout given on 18/9/10

⇒ * FOR Example application on one-one PK association Refer application ② of supplementary booklet on !

@ → along with @One-To-One annotation, if you write @PrimaryKeyJoinColumn Annotation then that will One-To-One PK Association.

→ along with @One-To-One Annotation, @JoinColumn annotation is written specifying FK column, then the will act as One-To-One FK Association.

Generators in Annotations :-

we can we generator algorithm to generate identity values as POJO class dynamically by using some algorithms

→ public class Student

{

~~PK sid~~ ;

@Id

@GeneratedValue (strategy = GenerationType.AUTO)

int sid ;

=

=

}

GenerationType.AUTO

↓

picks up the algorithm dynamically based on the capabilities of underlay of DB like native algorithm

Other possible values for strategy parameter :-

GenerationType :

GenerationType.SEQUENCE (uses sequence algorithm)

GenerationType.IDENTITY (uses identity algorithm)

GenerationType.TABLE (uses hilo algorithm)

=

~~class Stu~~

Procedure to configure user define sequence as Identity generator :-

class Student

{

@Id

@Column (name = "sno")

@GeneratedValue (strategy = GenerationType.SEQUENCE,

generator = "my_seq1")

Integer sno ;

=

}

Composit Identity Configuration? -

If only one member variable of HB POJO class, is configured as Identity field, then it is called singular identity field.

use <id> in HB mapping file are use @Id anno for this operation.

If more than one variable of HB POJO class are configured as identity fields then it is called composit Identity field configuration.

Note:- we cannot work with identity value generato algorithms while working with composit identity field.

mapping file based composit identity field configuration:-

```
public class Student
{
  int sno;
  String sname;
  String address;
  int age;
  getXXX() & setXXX
}
```

hbm file:-

```
<hibernate-mapping>
  <class name="Student" table="Student_Tab">
    <composit-id>
      <key-property property="sno" column="stno"/>
      <key-property property="sname" column="stname"/>
    </composit-id>
    <property name="address" column="stadd"/>
    .....
  </class>
</hibernate-mapping>
```

Annotation Based Composit Identity field configuration :-

⇒

```
@Entity
@Table (name = "Student_Tab")
@IdClass (StudPk.class)
public class Student
{
    @Id
    @Column(name = "stid")
    StudPk id;
    String addrs;
    int age;
    ==  getters & setters
}

@Embeddable
public class StudPk implements Serializable
{
  int sno;
  String name;

  getXxx & setXxx

  ==
}
=
```

← End of Hibernate →