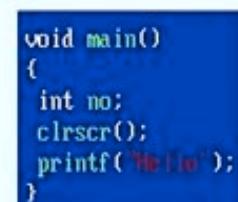





**College
Projects**


**Basic
Program** **C
Tutorial** 
**JAVA
Programming**



Tutorial4Us

Tutorials for Beginners



Tutorial4Us

Tutorials for Beginners



Tutorial4Us

Tutorials for Beginners

tutorial4us.com

A Perfect Place for All **Tutorials** Resources

Core Java | Servlet | JSP | JDBC | Struts | Hibernate | Spring

Java Projects | C | C++ | DS | Interview Questions | JavaScript

College Projects | eBooks | Interview Tips | Forums | Java Discussions

For More Tutorials Stuff Visit

www.tutorial4us.com

A Perfect Place for All **Tutorials** Resources

Hibernate

(Natraj Notes)

www.tutorial4us.com

HIBERNATE

15/7/10

The logic that is given to prepare user interface 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 persistence store.
Ex:- files, DataBase SQL etc.
- Insert, update, select and delete operations performed on the DB table - persistence store.
to maintain data are called persistence operation. The logic used for this purpose is called as persistence logic.
- persistence operations are also called as CURD (oos)
CRUD (oos) SCUP 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 persistence logic is called persistence layer technology.

Ex:- JDBC, Hibernate etc.

- JDBC, Hibernate code comes as persistence logic.
- persistence logic is always helps logic b/w Business logic.

Example applications -

STAKER TEST

Category

Test App

→ write signs out

→ read m₁, m₂, m₃ values → computation logic (Presentation logic)

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

→ store student details to DB-table (Persistence logic)

→ display results (Presentation logic)

(Servlet API) (JDBC) (Servlet & JSP)

* struts — hibernate — spring

* RMI — EJB (Programmatic code & JDBC)

* XML → web services (Servlets & JSP)

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

→ DB softwares are suitable as persistence stores in medium & large scale applications 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 codes OR Mapping

DB S/W [Oracle, Sybase]

Java app

Java code + OQL

ODB S/W (ext- power)

object query language

> logic

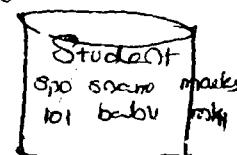
class marks

{

int m₁, m₂, m₃;

=

}

Marks m₁ = new Marks(45, 78, 89);(m₁)m₁ 45
m₂ 78
m₃ 89Marks class
objectLimitations with ODB S/W :-

- ① storing multiple values in a single column of db table
column is against normalization Rule no 2 - ①
- ② Generating reports from table of ODB S/W is complex process.

Limitations with files as persistence stores :-

- ① No security
- ② Can't store huge amount of data
- ③ No query language support

Ex:- manipulation ② ③ are very complex.

- Merging & comparison of data is complex.
- To solve problems with files & ODB S/W use RDBMS DB S/W like oracle, sqlbase, mysql & etc. as persistence ~~logic~~ stores.
- Java application can use JDBC code as persistence logic to interact with RDBMS database S/W's.
But following → limitations are these.

① SQL Queries are DB s/w dependent queries.

Since JDBC persistence logic use sql queries.

The JDBC code is DB dependent persistence 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 problem.

② JDBC Result Set 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.

To solve all these problems use ORMapping based persistence 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 OR Mapping persistence logic.

on table rows (CURD operations), we will use Java objects that are representing table rows, and there is no need of SQL queries. This makes OR-Mapping persistence logic as DB independent persistence logic.

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

class Student

```

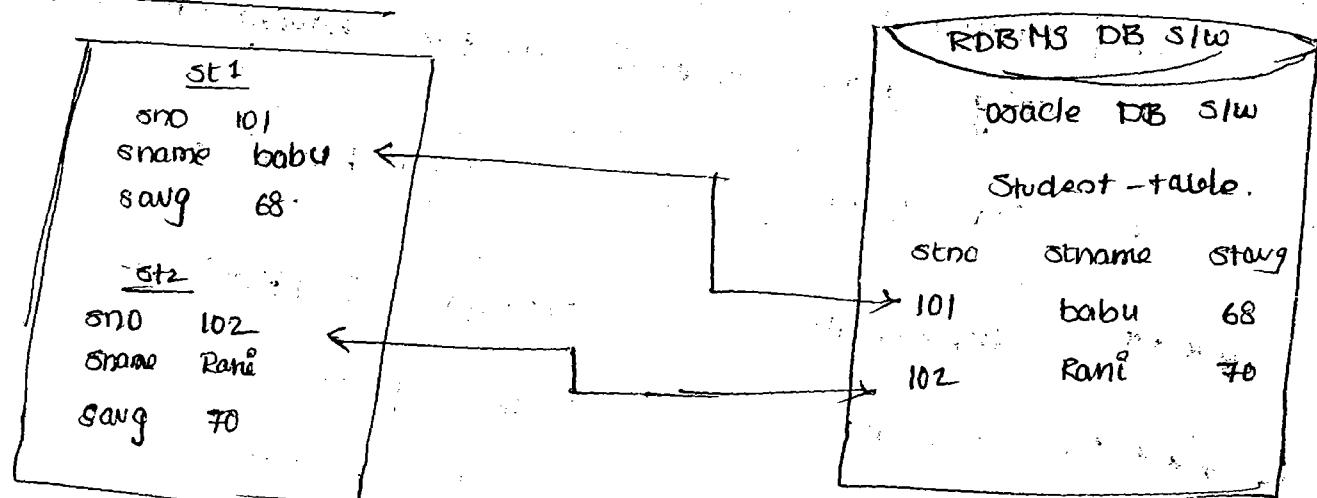
6
int sno;
String sname;
float avg;

```

OR Mapping work

members	$\{$	sno \rightarrow sno	db table columns
variables	\leftarrow	sname \rightarrow sname	
	\rightarrow	avg \rightarrow avg	

Java Application.



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

The special SW that provides abstraction layer on same category core (OS) basic technology is called **framework software**.

→ All ORM SW's internally use JDBC code to perform

persistence operation on DB table rows their are manipulated by manipulating objects

(hiding implementation) layers to programme on JDBC code. So ORM SW's are also called as Framework softwares.

- Struts is a web framework SW providing abstraction layer on Servlets, JSP Technologies.
- Spring is a Java - J2EE Framework SW providing abstraction layer on Java - J2EE technologies like JDBC, JNDI, JMS, EJB, Servlet - JSP & etc.
- Hibernate is a O-R mapping based framework SW providing abstraction layer on JDBC.
- Framework SW's generate common logics of the application development dynamically by using core - technologies so, Framework SW's improve productivity (doing more work in less time having accuracy).

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

- 1) ✓ Hibernate → salt tree (Red Hat)
- 2) ✓ Toplink → Oracle Corporation
- 3) EJB entity Beans → Sun Microsystems (Oracle Corporation)
- 4) JPA → sun microsystem (given from JEE5)
(Java Persistence 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
Persistence logic?

Ans:- while working with O-R mapping persistence
logic if 1000 records are there in the table
and they are not selected at a time, then
1000 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 programmers 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 SW's are given only to develop
persistence logic & they are not suitable to
develop other logics like presentation,
business logic and etc.

- type :- ORM s/w (oo) ORM tool (oo) ORM based framework.
- vendor :- soft tree (Red Hat)
- creator :- Ho. Gaving king & team.
- Version :- 3.5 (latest → compatible with J2SE 5.0 and 3.2 / 3.3 (regulatory used version))
compatible with J2SE 1.5
- 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.dev2dev.net,
www.precisejava.com,
www.javalinux.net
- Technical FAQ's :- www.forum.hibernate.org
- For Interview FAQ's :- www.geekinterview.org
- Reference books :- poof hibernate → apress (publisher name)
- Hibernate in Action → manning Series
- (published 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 java package.

- To develop hibernate persistence logic, programmes uses hibernate API & some helper resources like Java classes, and XML files.

Hibernate - def :- - hibernate is an open source, light weight java based ORM software as framework software to develop ORMapping style DB independent persistence logic in all kinds of java applications like stand alone, 2 tier, web application 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 components are heavy weight components

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

~~weight slows~~

- The Resources of Entity Bean components must be developed by using EJB API support.
- Learning & applying EJB entity bean component is always complex process.

~~Diff.~~

Hibernate is Lightweight SW -

Reasons -

- No servers, container are required to execute hibernate code.
- the basic J2SE DK SW 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 change.log.txt file of hibernate home directory (the directory where hibernate SW is installed).
for knowing difference b/w various versions of hibernate.
- Hibernate-home\hibernate 3.jar file represents the whole hibernate API.

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

=> MVC architecture is become industry standard
in the development of real world SW projects.

M - Model \rightarrow B.L + p.L \rightarrow (Exi - Account abbreveo)

$V - \text{View} \rightarrow \text{pol} \rightarrow [\text{Ex!- Beauftrag}]$

C - controller → Integration / connectivity logic.

[Ex: Traffic police]

→ logic that controls & monitors 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 → java bean → DB s/w.

(view) controllers (model)

JSP → servlet → java bean with DAO class → DB SQL.

The diagram illustrates a layered architecture with the following components from left to right:

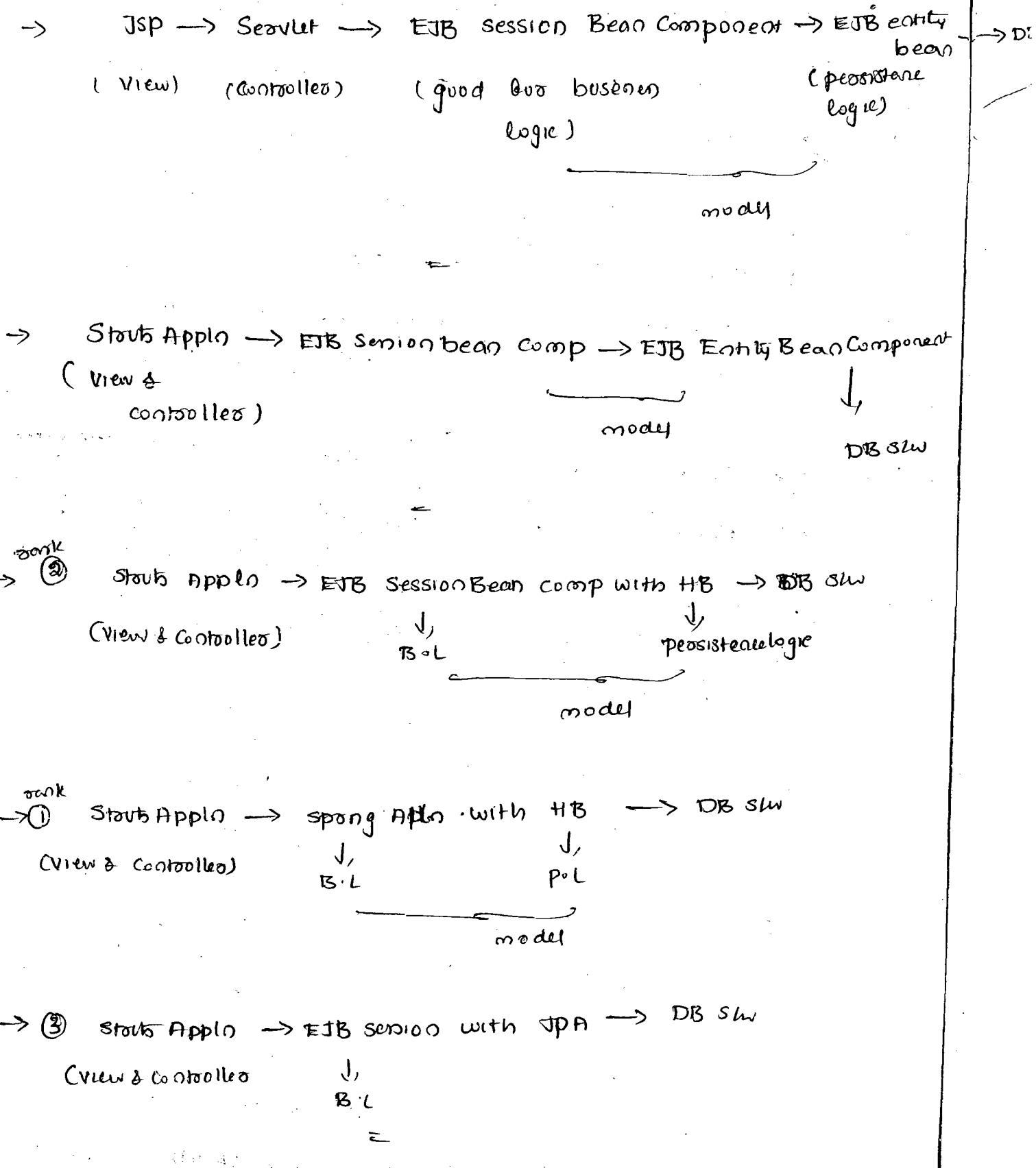
- (View)
- (Controller)
- business logic
- ↑
contains purely
persistance logic.

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. does not effect other logic of the application.



View layers "JSP, HTML, Velocity, FreeMarker, XSLT etc."

=> Various technologies to develop "integration logic" at controller layer "Servlet, Server 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 Sun Microsystems)

Webwork

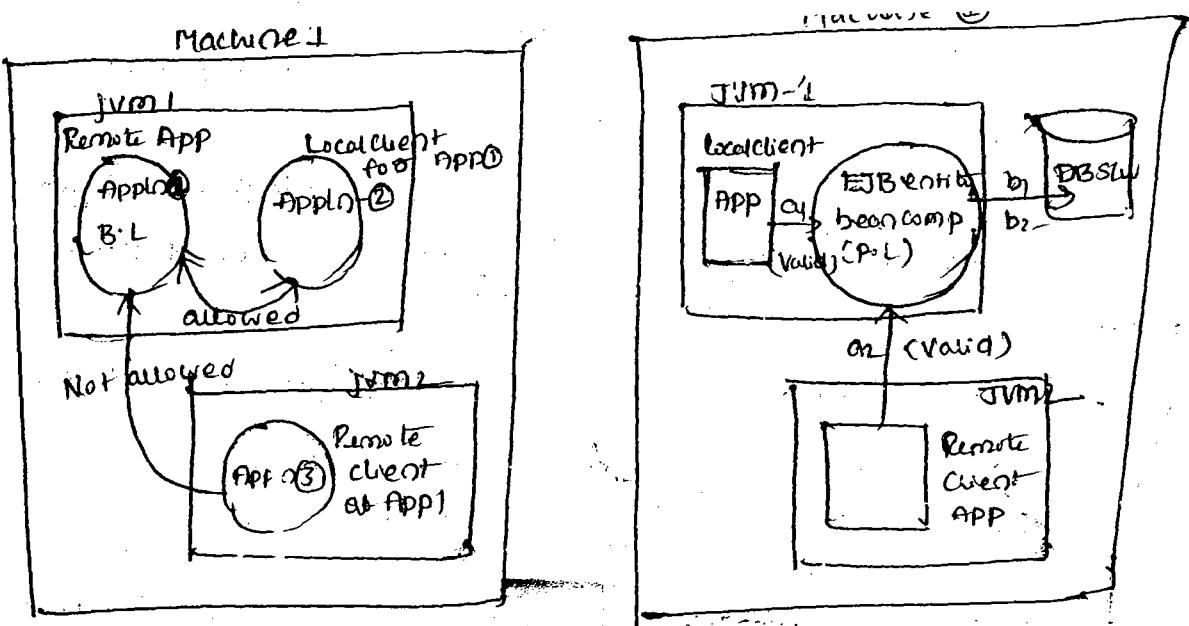
XWork, Spring MVC and etc.

=> Technologies to develop persistence logics of model layer JDBC, HIBERNATE, TopLink, iBatis, EJB entity beans, OJB, JPA and etc.

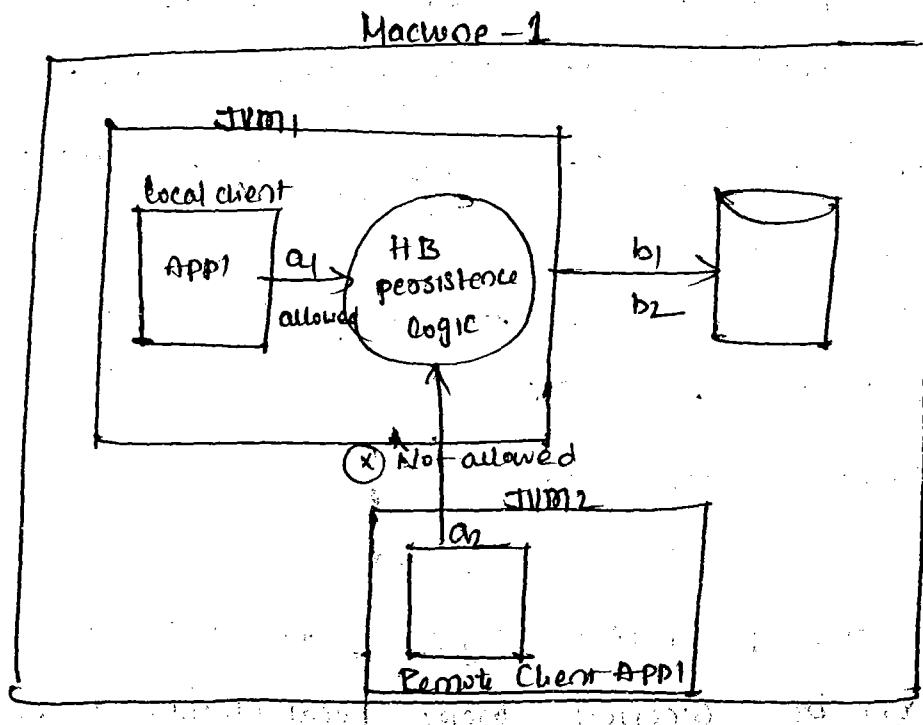
* → If application and its client reside on same JVM then application is called "Local application" and that client is called "Local client" to application.

→ If application and its client reside and execute on two different JVM's at same machine (so) 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 persistence logic of EJB entity bean component can be accessed from local clients (or) Remote clients.

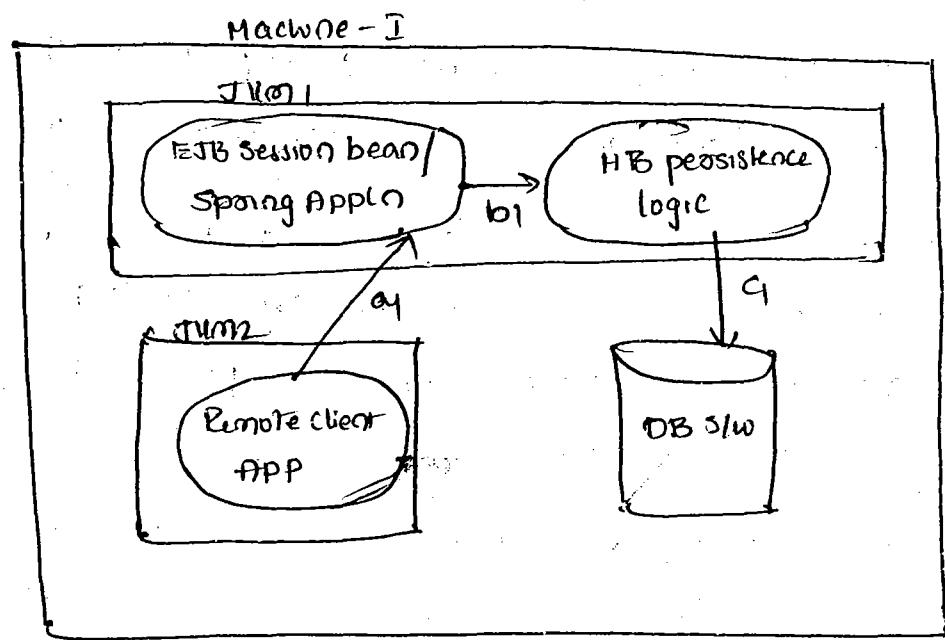


→ HB is not a distributed technology so that applications can't access hibernate persistence logic from remote places. Due to this the application and its HB persistence logic must be reside at execute from the same JVM. This is limitation of HB slow. This indicates HB persistence logic must be accessed only from local client apps.



multiple Java applications are executing parallelly and simultaneously.

→ 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)



$a_1 \rightarrow b_1 \rightarrow c_1 \rightarrow \text{DB S/W}$ (allowed)

→ since HB is not distributed technology, The HB persistence 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 appn wants to use HB P.L. to interact with DB S/W then XYZ appn and its HB persistence 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 appn.

↳ If we want to access DB S/W from different computer then we have to take help of mediator application.

Traditional client-server applications

These are location dependent. That means any change in the location of server application, we need to introspect 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) is an enterprise application.
- An enterprise application can be standalone (or) a-tier (or) distributed (or) web application.

Ex:-

- 1) Banking application is distributed & enterprise application.
- 2) Online shopping website is web application and enterprise & distributed application.
- 3) Hibernate supports POJO & POG 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, JSTL technologies also support POJO & POG model programming.

while developing a java class as resource at certain
S/W technology based java application, if that
class is not extending from a predefined class or
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

{

=

}

2)

public class Test extends Demo

{

=

}

Demo is user defined class so Test is POJO class

3)

public class Test extends Forum

{

=

}

Test is not a POJO class it is 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.

6
≡
3

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

7) It is only basic concept of Java.

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

6
≡

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

8) public class Test extends HttpServlet

6
≡
3

Test is Servlet API dependent. So Test is not
POJO class.

9) public class Test

6

public void bmt()

6
≡
3

3

Test is POJO class.

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

If that interface is not extending from predefined interfaces or 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.

Advantages of Hibernate

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

one record ab one child table.

iv) Easy to Learn and Easy to apply.

22/7/10

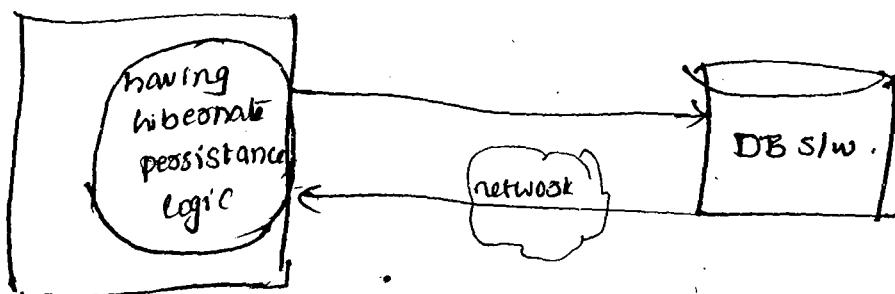
Browses → servlet → EJB component → DB Slw.

The above combination browses window is client to servlet. Servlet is client to EJB component and EJB component to browses window.

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

→ When java application uses JDBC code to interact with DB Slw, the java application is not client to JDBC code it is client to DB Slw.

Similarly, when java application uses Hibernate persistence logic to interact with DB Slw then the java application is not client to Hibernate Slw. It is client to DB Slw.



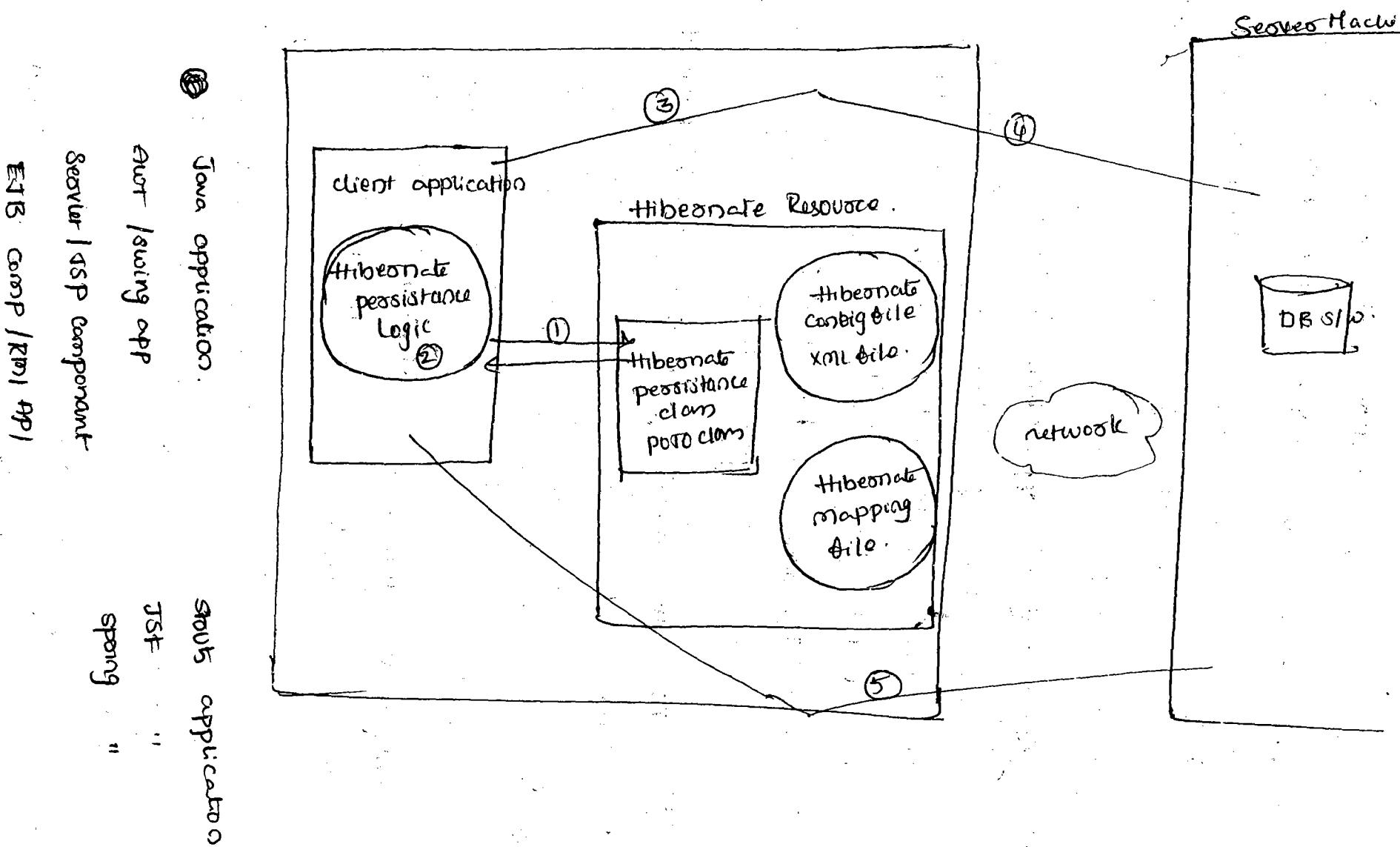
persistence logic to interact with DB s/w.

Note: the DB s/w can be local (or) remote to Java application but the hibernate persistence logic must be there along with java application.

→ different types of Java JEE applications need following resources directly & indirectly to develop OR Mapping based persistence logic as hibernate persistence logic.

- 1) JDBC drivers
- 2) hibernate configuration file (xml file)
- 3) hibernate persistence class (Generally POJO class based Java beans)
- 4) hibernate Mapping file (hbm file) (xml file)
- 5) hibernate API Available in hibernate 3.0 jar file

Hibernate Architecture.



- ① Client application activates Hibernate SW and makes that SW collecting Hibernate resources.
- ② programmer uses Hibernate API and Hibernate resources to develop objects base OR Mapping style persistence logic.
- ③ Java Client application interacts with DB SW by using this persistence logic (this OR Mapping persistence logic internally generates JDBC code to fulfill this requirement)
- ④ Data / DB SW will be manipulated based on the instructions given through persistence logic
- ⑤ DB SW sends generated result back to client application.

Hibernate configuration file :-

(Any file names .xml can act as hibernate configuration file but hibernate SW 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 SW explicitly.

→ This configuration file contains details to connect to database SW like driver class name Database URL, DataBase ~~URL~~ 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 too any DB s/w.

hibernate.connection.driver_class

hibernate.connection.url

hibernate.connection.username

hibernate.connection.password

hibernate.dialect

mappingfile name (hbm file name)

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

hibernate-home\etc\hibernate.properties file

(Q) chapter 3 of pdf file of hibernate

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

Ans Q) what is the use of hibernate.dialect property?

This property takes hibernate s/w supplied
DB s/w specific class name 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:-

i) oracle any version.

oog.hibernate.dialect.OracleDialect

ii) oracle 9i+10g

oog.hibernate.dialect.Oracle9Dialect

3) MySQL

oog.hibernate.dialect.MYSQLDialect

too more dialect class name refer chapter 5

as pdt file.

→ hibernate.dialect.property value helps hibernate s/w

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

a) makes hibernate s/w optimized SQL queries smoothly based on the DB s/w. to fulfill persistence operation requirement.

25/07/10

→ while developing HB persistence logic we can see 1 (or) more HB configuration files.

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

Hibernate Mapping file :-

Any filename.xml can act as HB mapping file. This file should be specified in HB configuration file.

There is no default name for this file,

this file contains various types of ORMapping

configuration like basic ORMapping, collection Mapping, Association Mapping, interface mapping and etc

- DB table members variables with table column names, this file is base file for HB S/W to understand OR Mapping configuration related to HB persistence classes.
- you can use example applications at Hibernate-Homework test folder to gather persistence example HB-Mapping files

HB persistence class :-

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

- This class is base class for the programmers to develop objects based OR Mapping style persistence operation in the client application.
- It is recommended to take java bean as HB persistence class but it is not mandatory.

client Application :-

This application is client to DB S/W and interacts ; manipulates DB table data by using ~~HB~~ HB persistence 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 jars of that one jar file.

→ If a java application uses two or more libraries 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 & dependent 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  
{  
    public static void main()  
    {  
        Test t = new Test();  
        t.m1();  
    }  
}
```

» `javac App1.java`

error :- can't invoke symbol.

solt :- add frost.jar file in classpath

» `java App1`

error :- java.lang.NoClassDefFoundError

solt :- add frost & second.jar files to classpath
where Demo class is used.

→ Hibernate is 3rd party SW so, it's API is called 3rd party API. When Java application uses Hibernate API, to develop Hibernate persistence 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 HIBAPI & it is having 7 no. of dependent jar files.

1) hibernate-core.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 at Hibernate-Home\lib
folders.

→ XML parsers are responsible to read &
manipulate data of XML files

EX! - SAX ~~SAX~~ (Simple API for XML processing),
DOM (Document object model), JDOM (Java DOM),
DOM4J and etc.

→ Hibernate ...

To read & manipulate content of XML files
called lib configuration file, HB Mapping files.

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

→ When Java web application based webresources
use Hibernate API to develop persistence logic
then the Hibernate API related main jar file
(HB3.jar) must be added to the classpath
and the HB API related Main and dependent
JAR files (HT.jar files) must be added in
the WEB-INF\lib folder of web-application.

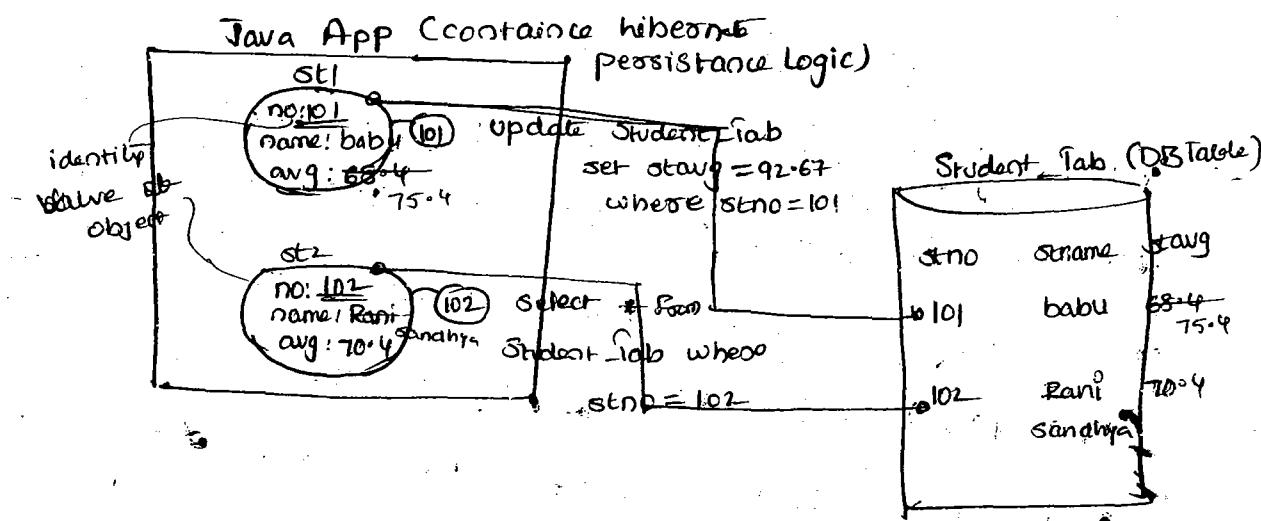
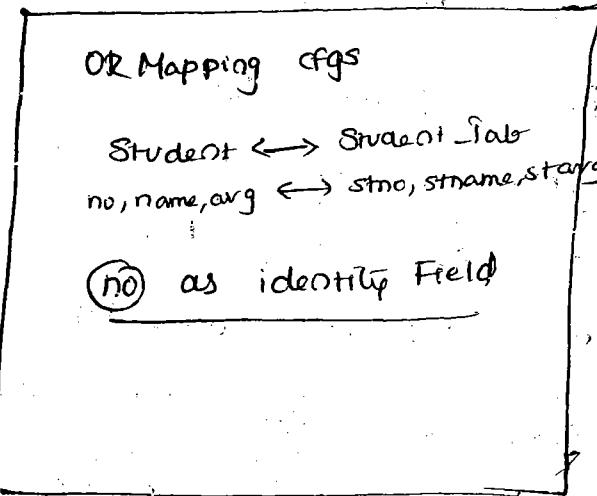
→ understanding synchronization b/w java object
 table rows while ~~etc~~ dealing with OR Mapping
 Persistence Logic.

public class Student

```

  int sno;
  String name;
  float avg;
  setXXX()
  {
    ...
  }
  getXXX()
  {
    ...
  }
}

```



→ every hibernate POJO class / persistence class object
 contains one identity value. hibernate b/w generates
 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 configured 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 select query by taking identity value as criteria value to reflect the changes in relevant record at 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 reflect 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 members variable value of the object (or) if the programmer modifies identity field ~~and~~ related table column value directly from sql prompt HB s/w perform Synchronization in a regular manner, in this case if necessary the HB s/w becomes ready to create new HB POJO class objects.

Identify field from the member variables of
Hibernate POJO class?

Ans :-

SC(1) :- If POJO class related DB table is having
one unique key constraint column or one
singular primary key constraint column then take
that column related member variable in
HB POJO class and configure that one as
singular Identity field of POJO class.

SC(2) :- If POJO class related DB Table is having
composite primary key constraint based on
multiple columns then take those multiple columns
related member variables of POJO class and
configure them as composite Identity field of
POJO class.

SC(3) :- If POJO class related DB table contains
no constraints programmer should ~~not~~ analyse
and gives 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
composite identity field.

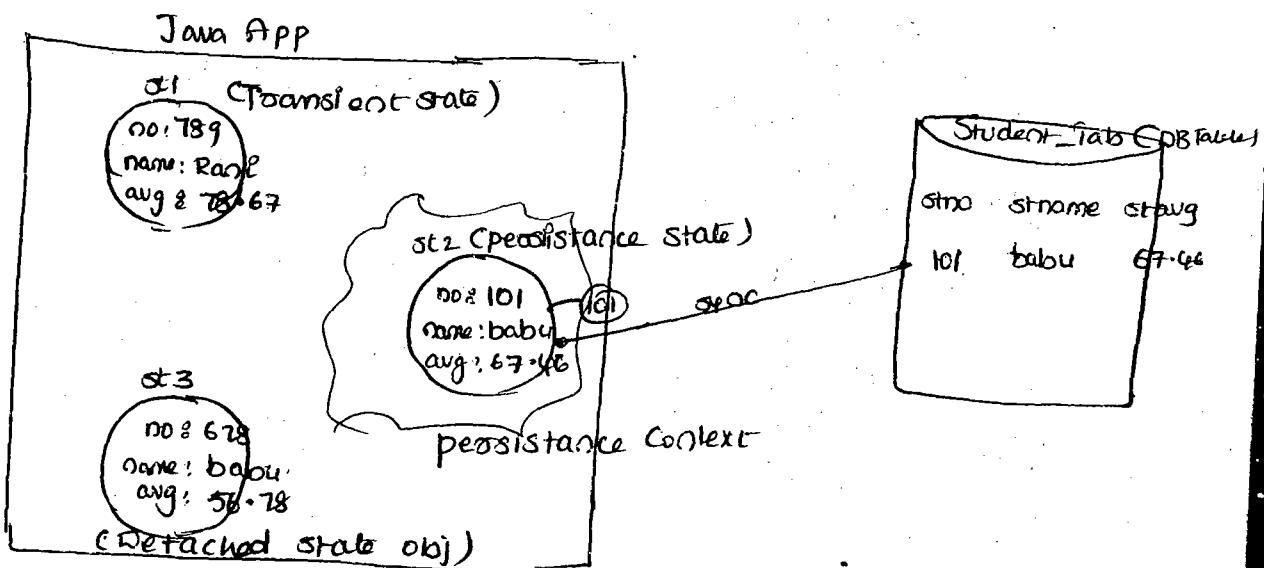
NOTE(1) :- If only one member variable of HB POJO
class is configured as Identity field then it is
called "singular identity field".

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

table columns are having constraints are not
identity field configuration must be performed
on hibernate POJO class number variables.

⇒ we can see HB POJO class object in ③ states

- ① transient state
- ② persistent state
- ③ detached state



Transient State :-

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

Persistent State :-

This state object represents table row and maintains synchronization with table row. This state object contains identity value. Programmers use this state object to perform persistence operation on table rows.

THE ENVIRONMENT

The maintenance persistence state object is called
Persistent context.

Detached State, Transient State objects
exists outside persistence context

Detached State:-

This state object contains identity value but doesn't represent a record. i.e either that record is deleted or the identity field column value of that record is modified.

Detached object doesn't perform synchronization with record.

Detached state object has represented the record earlier, but it is not currently representing the record. That is the reason to have identity value in Detached state object.

26/7/10

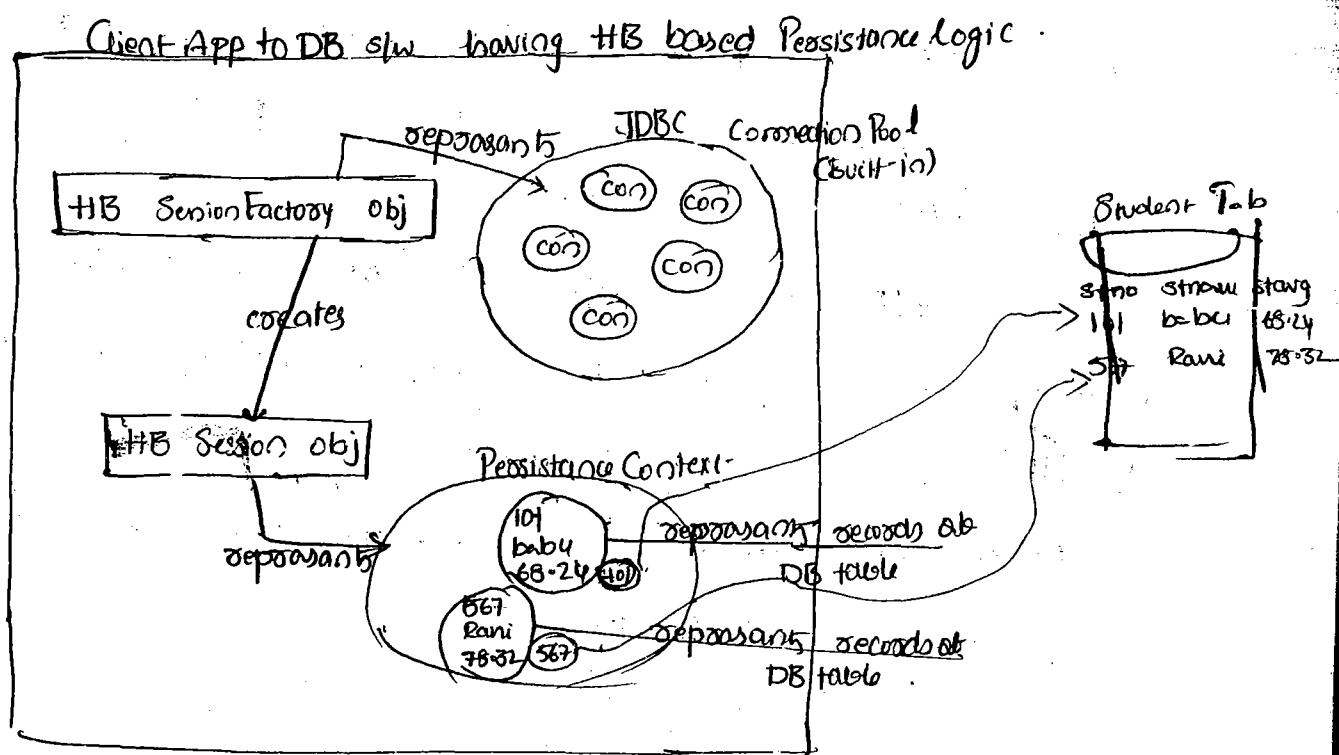
→ While developing hibernate Persistence logic in the client application, the programmer needs to deal with

① transient object-

② Hibernate SessionFactory object

③ Hibernate Session object

object?



Hibernate SessionFactory Object :-

- this object creates and represents

JDBC connection pool to 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 representing creating one or more HB Session objects.

Hibernate Session object :-

Session Factory object uses one \otimes JDBC connection object of connection pool and this connection object creates Statement object to create HB Session object.

This Session object represents Persistence Context where Persistence state HB POJO class objects reside representing

Programmed to provide instruction to HB SW
to perform persistence operations on the table
rows based on the operations performed on
the POJO class object.

HB Session object is no way related with
HibSession object ab Server programming.
HB Session object means it is the object
ab a class that implements org.hibernate.Session
interface.

⇒ Sample code of Client App (that contains hibernate
persistence logic) :-

// read cfg properties from hibernate.cfg.xml

// by activating HB SW.

→ Configuration cfg = new Configuration(); → ①

cfg.configure(); → ②

// create HB SessionFactory object

→ SessionFactory factory = cfg.buildSessionFactory(); → ③

// create HB Session object

→ Session ses = factory.openSession(); → ④

// write HB persistence logic.

// close HB Session object

→ ses.close();

cfg ① → it just calls HB S/W to return Persistence logic
and ② 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 HB SessionFactory object representing that connection pool.

④ openSession(), makes HB SessionFactory object to create one Hibernate Session object

⑤ ~~Session~~ session.close(), closes connection with DB S/W represented by Session object and also closes Persistence Context.

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

Deactivated in client apps?

- Ans: when Configuration class object is created,
HB S/W will be activated,
- when factory.close() is called HB S/W
will be deactivated.

→ Configuration cfg = new Configuration();
cfg = cfg.configure();

HB S/W looks, take hibernate.cfg.xml as hibernate
Configuration file,

Configuration cfg = new Configuration();
cfg = cfg.configure("myfile.xml");

HB S/W 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 no. of DB S/W involvement).

reated

Single Row Operations

Call following on HB Session obj

save(pojo obj) → to insert a record

persist(pojo obj) → to insert a record

Bulk operations

HQL

Native SQL

Criteria API

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 perform persistence operation, from HB environment will be HQL

27/07/2010: Develop the application to insert record into database

table by using hibernate persistence logic.

→ SW setup required

j2sdk 1.4 + SW, oracle sql plus, hibernate 3.x SW

→ Resource that are required.

1) DataBase table

Employee (Table).

Eid number primary key

FIRSTNAME varchar(20)

LASTNAME varchar(20)

EMAIL varchar(20)

→ create table employee (EID number primary key,

FIRSTNAME varchar(20),

LASTNAME varchar(20),

EMAIL varchar(20));

2) Hibernate Resource

EmpBeam.java → Hibernate persistence class

hibernate.cfg.xml → Hibernate configuration file.

Employee.hbm.xml → Hibernate mapping file.

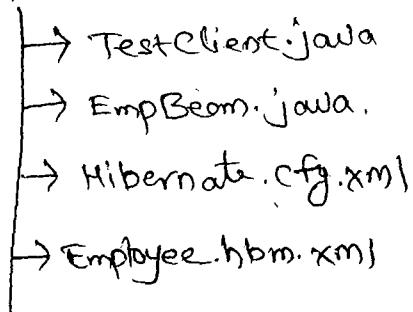
3) Client Application

TestClient.java → Client Application (java application)

→ Procedure to develop the above application.

Step 1: Develop the above given (java,xml) Resources and save them in a directory

E:\apps\hibernate\app1



hibernate.cfg.xml

<DOCTYPE -----

-----> hibernate-configuration3.0.dtd>

\hibernate.cfg.xml

<session-factory>

7 <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.oracle9
dialect </property>
<property resource="Employee.hbm.xml"/>

</session-factory>

</hibernate-configuration>

Note: hibernate property file, hibernate.cfg.xml file of hibernate-
home\dtc folder as reference example files.

Note: The above xml is developed against the dtd rules
available in hibernate-configuration3.dtd file.

II 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;
    }
```

```

8       return lname;
9
public void set setName(String lname)
8       this.lname = lname;
9
public String getLname()
8       return lname;
9
public void setMail(String mail)
8       this.mail = mail;
9
public String getMail()
8       return mail;
9
9

```

Note: The above class is java bean class that is taken as hibernate pojo class.

Note: Hibernate pojo class must contain a zero argument constructor (public) directly or indirectly
 program or generated one. → the java compiler generated one.

Employee.hbm.xml

```

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

```

<id name="id" column="EID"/> ⇒ singular identity field
configuration.

<property name="fname" column="FIRSTNAME"/>
<property name="lname" column="LASTNAME"/>
<property name="mail" column="EMAIL"/>

→ </class>
</hibernate-mapping>
→ O-R-Mapping configurations.

```

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 member variable with database table column
- <id> tag can also be used for this purpose but this tag also configures that member variable as singular identity field.

TestClient.java (client application to database s/w that
is using hibernate persistence logic)

```

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

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

```

config
 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.setId(10);
 eb.setFname("raja");
 eb.setLName("rao");
 eb.setEmail("rao@gmail.com");

'eb' object state
 is transient
 state here.

// now insert record.

Transaction tx = ses.beginTransaction();

'eb' object state is
 persistent state → ses.save(eb); // insert the record.
 here. → tx.commit();

// close session object

→ ses.close();

// close sessionfactory obj

→ factory.close();

if → close of main

if → close of class

'eb' object state is
 detached state
 here.

Note: All non select query, insert, update, delete, etc.
HB persistent logic must be executed as transactional
stmts as shown above.

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

28/07/2019

- Step 2: Add following jar files to the class path
- 1) hibernate3.jar (available in HB home 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) aom.jar
 - 8) antlr-2.7.6.jar
 - 9) cibc14.jar file.

2-8 => Dependent jar file to hibernate3.jar available in

HBhome/lib folder.

9 => Represent Oracle thin driver, available in oracle/oracle/jdbc/lib/folder

Note: Always use my computer Environment variable to add jar

files to class path.

";" symbol should be taken as separator b/w multiple

values added to my computer Environment variable, shouldn't
be taken as beginner or terminator.

Note: After adding new jar files to my computer Environment
variable class path ple open new command prompt in order to

Step 3: compile java resources of the application

E:\App\Hibernate\app2> javac *.java

Step 4: Execute the client application

E:\App\Hibernate\app2> java TestClient

⇒ To make HB SW showing "internally generated SQL queries" to fulfil our requirement we "show-sq!" property of HB Configuration file as shown below

<property name="show-sq!" value="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 onwards 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 +
This identity value comes as serializable object
value the programmer can evaluate whether a d is inserted
- or not.

with session.persist() we can't catch + use the
generated identity value, because the return type of this method
is void.

→ Example code of session.save() method.

```
Transaction tx = ses.beginTransaction();
Integer idval = (Integer) ses.save(eb);
System.out.println("Identity value "+ idval);
tx.commit();
```

→ Example code of session.persist()

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

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

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

new SWL's, Based on proprietary specification only certain company can develop the SWL's.

JDBC, servlets, EJB, JPA, etc--- are open specification.

o Net is the proprietary specification

Based on JPA specification supplied by SUM MICRO SYSTEM all ORMs
s/w's like hibernate, Toplink, iBatis and etc... are developed
→ hibernate 3.x version s/w's are given based on JPA specification

୨୭|୦୭|୨୦୧୦

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

```
EmpBean eb = new EmpBean();
eb.setNo(1080); // resistance identity value
eb.setName("newraja");
eb.setLname("newrao");
eb.setMail("newrao@gmail.com");
```

Transaction tx = ses.beginTransaction();

```
s.o.p(eb.getNoe) + " " + eb.getFrame() + " " +  
eb.getLname() + " " + eb.getMail());
```

```

Transaction tx = ses.beginTransaction();
ses.update(EB);
tx.commit();

```

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

whereas "session.update()" method update the record but can't 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 single row operation methods of hibernate session object, They can take only "identity value" of given POJO class object as criteria value. In order 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);
tx.commit(); Transaction state object

```

→ Here "eb" object state will be "detached" state.

persistence 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));

pojo class name
↑
persistence state pojo object
representing date of the selected record.
identity value as
a criteria value.

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

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

"EmpBean.class" state must kept in application
generates object of java.lang.Class degrading

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~~ for given
POJO class) and return that object.

- ⇒ "session.load()" methods performs "lazy loading"
to select the record that means until "getXX()"
are called on the return object (eb) record
"will not" be selected from the table.
→ session.get() method does not perform "lazy loading"
in any situation.

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

A) ⇒ 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 does not perform "lazy loading"

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

⇒ public Object get (Class name, Serializable id)
It does not perform "lazy loading"

class object identity field members variable value as
contains 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("Daya");
eb.setLname("Daoi");
eb.setMail("daoi@gmail.com");
```

Transaction tx = ses.beginTransaction();

```
ses.saveOrUpdate(eb);
tx.commit();
```

⇒ In serverside components like searlets, 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 SW

version: 4.0

Vendor: MySQL

open source DB SW

Default port no: 3306

default logical db names: test, mysql

Type: 4 mechanism based doneoname :- connectors

default admin username and password:

root (username)

root (password)

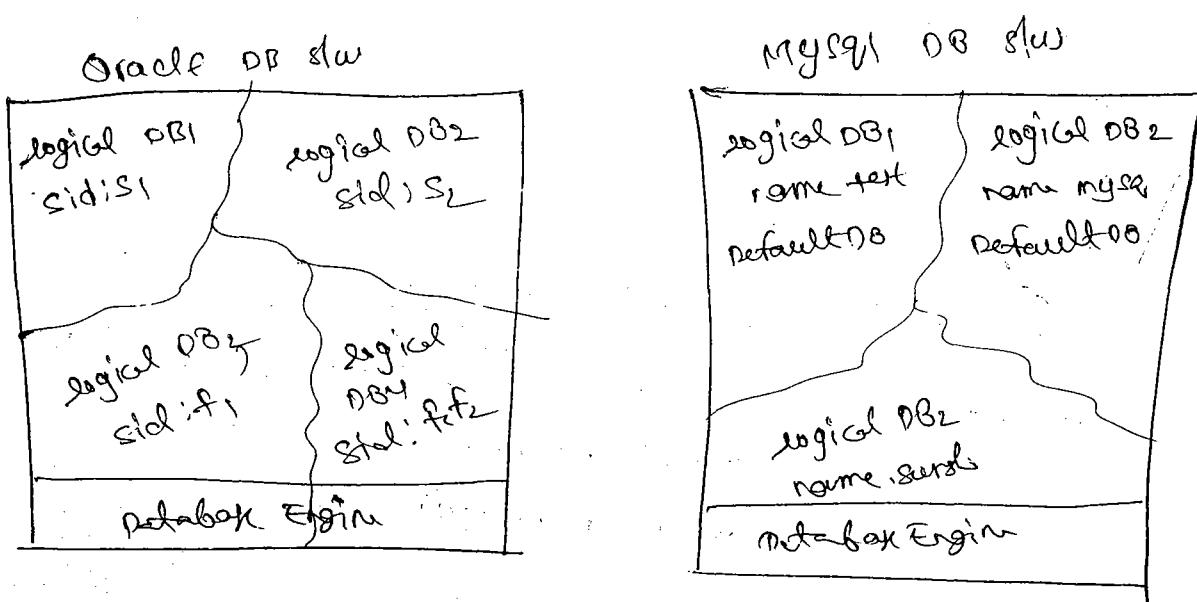
→ After installing mysql 4.x SW we need to activate that SW
separately by using winmysqladmin tool available in mysql/
home/bin directory (look for traffic signal symbol in the system
tray)

→ my SQL front is a third party supplied GUI tool to
perform operations by connecting to mysql database SW it is
like sqlplus tool of oracle DB SW

→ logical database are the logical partitions created in physical
database SW installed in computer. Each logical db will be
identified with one logical name. In oracle DB SW
this name is also called as sid (service id). In MySQL
DB SW each MBS acts as one logical database
and that MBS file name is logical database name.

In a company if multiple projects are using
same DB SW then the db SW will be installed

database will be created in that database & one per project base.



→ Procedure to create logical database and DB table in that by using MySQL front tool.

step 1 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: SunilHBDB → OK → select
 create table → Table name: Employee → Add → ID - INT: 5 primarykey, unique, key, normal
 Employee → Add → first name: varchar20 last name: varchar20 Email
 first name: varchar20 last name: varchar20 Email
 first name: varchar20 last name: varchar20 Email
 insert record 101 Rajesh rao rao@gmail.com
 1056 Ramya Chary chary@gmail.com

connector 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

→ The above JAR file is available in MySQL-Connector-Java-3.0.8-stable-bin.jar
→ Zip Extraction

Note : MS SQL Client Setup, MS SQL front end setup connecting driver connect etc folder of DVD.

→ Procedure to make our first HIB application → interact with MySQL database etc.

Step 1 Keep all the resource as it is, but do following modification in HIB configuration file.

hibernate.cfg.xml

```
<!DOCTYPE ----->
```

```
<hibernate-configuration>
```

```
  <session-factory>
```

```
    <property name =
```

```
>org.gjt.mm.
```

```
driver</property>
```

```
>jdbc:mysql://(suresh)
```

```
HIBO</property>
```

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

```
<property name = "connection.password">root</property>
```

```
<property name = "dialect">org.hibernate.dialect.mysql.  
database</property>
```

<session-factory>

<| hibernate-configuration>

step2) Add mysql connector jar file 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 as it's by using hb persistence logic becoz, the MS is 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 :-

② session.flush() method, forces the HB SW to synchronize with underlying DB SW records by using the date as persistent state POJO class object as persistence context.

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



// Select record from DB table into POJO object
 EmpBean eb = (EmpBean) ses.get(EmpBean.class, new
 Integer(1016));
 System.out.println("Employee No : " + eb.getFname() + " " +
 eb.getLname() + " " + eb.getMail());

 // modify "eb" (persistent state obj) object class
 eb.setMail("xyz1@xyz1.com");
 ses.flush("xyz1");
 ses.flush(); // two methods reflect changes done
 if (!"eb")
 // to DB table now through synchronization.
 // close session obj.
 ses.close();
 } } finally {
 ses.close(); // close sessionfactory
 }
 The above code demonstrates POJO object to DB table
 now synchronization.

DB table to POJO Object ? — (Synchronization)

Code in the client application to demonstrate
 DB table now to java object based ~~to~~ Synchronization
 [Modification done in DB table now from DB side will
 reflect in the associated persistent state object]

// Select record from db table into POJO obj.

EmpBean eb = (EmpBean) ses.get(EmpBean.class,
 new Integer(1016));
 System.out.println("Employee No : " + eb.getFname() + " " + eb.getLname() +
 eb.getMail());

System.out.println("Employee No : " + eb.getFname() + " " + eb.getLname() +
 eb.getMail());

Record "1016" to db table from sql prompt.

ses = refresh(eb);

// read & reads data from db table down into
// eb object for synchronization.

SOP(ebo.getNo() + " " + eb.getFname() + ",

eb.getLname() + " " + eb.getMatl());

Note :- Can you develop HB application (o) HB Persistence logic without HB Configuration file?

Ans :- Yes, it is possible and there are two approaches for this,

Approach(1) :- (By Using setProperty() at osgi hibernate.cfg.xml Configuration class.)

Code in client application :-

Configuration cfg = new Configuration();

cfg.setProperty("hibernate.connection.driver_class",
"oracle.jdbc.OracleDriver");

cfg.setProperty("hibernate.connection.url",
"jdbc:oracle:thin:@192.168.1.1521");

username
password cfg.setProperty

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

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

// Create SessionFactory obj

Session session = sessionFactory.buildSessionFactory();

→ since configuration property values are hard coded in the client 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

Persistance
roaches
ibemder
uDate
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 storing 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

props.txt :-

→ this is properties file

hibernate.connection.driver_class = oracle.jdbc.driver.OracleDriver

hibernate.connection.url = jdbc:oracle:thin:@192.168.1.52:1521:scott

hibernate.connection.username = scott

Show-Sql = true.

Code in Client application to create HIB 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.

String pToString());

// activate HIB Session and make the Session reading HIB config file.

Configuration cfg = new Configuration();

- cfg.setProperties(p);

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

== // create SessionFactory object -

--

--

Note:-

These approaches ① & ② practices are not industry standard practices always work with XML file as HIB 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 proun.

2/08/10

My Eclipse :-

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

Version : 6-x (compatible with j2sdk 1.5)

Vendor : - Eclipse

Commercial s/w : -

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

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

for documentation : - www.myeclipseide.com

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

plugin's come as jar files

My Eclipse IDE = Eclipse IDE + builtin plugins.

Eclipse IDE

1) Design to develop basic J2SDK application.

2) external plug-ins are required to develop advanced technologies based applications.

3) opensource SW

My Eclipse IDE

1) Design to develop all kinds of java, J2EE apps and other framework SW based applications

2) builtin plug-in's are there to ~~work~~ develop advanced technologies based java applications.

3) commercial SW .

Q) How to add plug-in's to the projects of Eclipse IDE

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 Create project in Eclipse IDE 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-ins and programmers can use the plug-ins applied environment to develop that technology based application.

Procedure to develop 1st HB Application by using MyEclipse IDE :-

Step①:- Launch MyEclipseIDE and choose workspace.

Note:- workspace is a folder where all projects created in MyEclipseIDE will be saved.

Step②:- update the subscription

MyEclipse Menu
 ↳ update subscription

 ↳ subscribers : - administrator
 subscription code : - XXXX - Cheat code

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

⇒ connected with ORACLE DB S/w. (DBNode)

Windows

 ↳ open perspective

 ↳ other

 ↳ MyEclipse DB Explorer

 OK ↳

 goto 'DB Browser' window

 Right click ↳

 New ↳

 Oracle thin ↳

 Databases

 Databases name : - osoap (logical name of profile)

 Connection url : jdbc:oracle:thin:@192.168.1.521:1521:soap

 username : - SCOTT

 password : - tiger

Save mode:-

Add jars:-

- checkbox save pars word.

↓
Next

→ Next

→ Finish

- Right click on **ORAP** at DB Browser window.

↓

Open connection.

↓

- Step ④:- Make sure that Employee table is available in oracle DB slw 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 hiberate capabilities to the project

Right click on project :-

→ MyEclipse

→ Add hibernate capabilities.

Select hibernate 3-1 core libraries

~~new~~ " " Advanced support libraries

↓
Next

↓
Next

Data Source :- use JDBC Driver

DB Driver - ojdbc (DB pool created above)

↓
Next

Create SessionFactory class?

→ ssc

→ java package ? New → pl

classname : MyHelper

↓

finish

↓

Goto hibernate.cfg.xml file

→ add miscellaneous properties

→ show-sql

→ ~~ps~~

→

Note :- the above step gives following things.

- a) adds hibernate API related jar files to -
- b) build path (or) classpath of the project
- c) hibernate.cfg.xml as hibernate configuration file
- d) use MyHelper.java file having code to create hibernate ~~or~~ SF object and Session object -

Java

⇒ Peftoom . Hibernate Reverse Engineering on Employee table to generate its POJO class (Employee.java), 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 ↴

Hibernate Reverse Engineering

java src boldo1 - /MyEclipse /sc

HB Mapping file.

update HB configuration with mapping file location.

java Date Object

Next

Type Mapping ↵

select HB types ↵

↳ Next

↳ Finish

↳ OK

- ① Employee.java as 3 POJO class
② Employee.hbm.xml as HBM mapping file

⇒ Develop client App in the project :-

Right click project
→ New
→ class

Name : Test

Main args[])

Finish

ctrl+shift+F10 // import statements

class TestClient

Transaction tx = session.beginTransaction();

Employee e = new Employee();

e.setEid(016);

e.setName("babu");

e.setLname("kashim");

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

// e.setEid(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 takes to
update the record if given POJO object record
is available in DB table, otherwise, (if not available)
this method inserts the record by using given
POJO class object data.

application being from my Eclipse IDE :-

Right click-on source code of application



Run As



open own dialog



choose app name (TestClient)



Arguments tab

Program Arguments

val1 <space> val2 <space> Val3 ...

args[0]

args[1]

args[2]



Apply

→ Run

MySQL

3/8/10

procedure to create DB pool in MyEclipse IDE pointing

to MySQL DB slave :-

Window menu

→ open perspective

→ Other

→ MyEclipse DB Explorer



DB Browser window

Right click
new

Right click

←

Driver template : MySQL Connector

Drivername : mysql

ConnectionURL : jdbc:mysql://localhost:3306/testDB.

username : root

password : — (NO PWD)

JAR file :-

Add Jar

... connectors-jdbc-3.0.8-stable-bin.jar

↓
Next
↳ finish
↓ =

Right click on mysqlp

↳ DB Browser window

↳ open connection
↳ ok

→ Example Applications to communicate with multiple DBs by using hibernate persistence logic & -

while working with Oracle log DB [XE] to communicate from HIB Persistence Logic also add ehcache-1.2.3 jar file in class path along with other regular jar files.

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

Resources Required :-

(1) in Oracle DB S/w :-

Employee (table)

EID	PK	NUMBER
FIRST NAME	VARCHAR2(20)	(20)
LAST NAME		(20)
EMAIL		

make sure that some records are available in this table.

Employee (table)

EID pk ~~int~~ int(5)

~~FIRSTNAME~~

FIRSTNAME

varchar(20)

LASTNAME

EMAIL

"

"

(3) files :-

mycfg-000.xml (oracle DB s/w related config file)
mycfg-mysql.xml (mysql DB s/w related config file)

Employee.hbm.xml (HBM file)

EmpBean.java (HIB Persistence class)

TestClient.java (Client app having persistence logic)

AIM :- AIM OF THE ABOVE APPLICATION IS SELECT A RECORD

FROM ~~DB~~ DB TABLE (Employee) OF ORACLE S/W AND
INSERT THAT RECORD INTO mysql DB TABLE (EMPLOYEE)

mycfg-000.xml :-

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

mycfg-mysql.xml :-

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

(4) Employee.hbm.xml -

← same as first I application →

← same as 1st application →

TestClient.java :-

```
import org.hibernate.cfg.*;
import org.hibernate.*;
public class TestClient
{
    void() throws Exception
{}
```

// get HB session object connected to oracle DB SW.

```
Configuration oorcfg = new Configuration();
oorcfg = oorcfg.configure("mycfg-oracle.xml");
SessionFactory oorfactory = oorcfg.buildSF();
Session oorases = oorfactory.openSession();
```

// get HB session object connected to mysql DB SW

```
Configuration mysqlcfg = new Configuration();
mysqlcfg = mysqlcfg.configure("mycfg-mysql.xml");
SessionFactory mysqlfactory = mysqlcfg.buildSF();
Session mysqlses = mysqlfactory.openSession();
```

// write HB persistence logic interacting with multiple DB SW

// select a record from oracle DB SW

```
EmpBean eb = (EmpBean) oorases.get(EmpBean.class,
new Integer(1016));
```

// insert the record into mysql DB SW

```
Transaction tx = mysqlses.beginTransaction();
```

```
mysqlses.save(eb);
```

```
tx.commit();
```

```
oorases.close(); } // closing session object
```

```
mysqlses.close(); }
```

```
oorfactory.close(); } // closing SessionFactory
```

```
mysqlfactory.close(); }
```

Add Regular Hibernate

add ~~hibernate~~ ojdbc14.jar, m

-connection-java-3.0.8-stable
-bin.jar

⇒ compile all java resource
application.

and execute the client

Session Factory object
but it is mutable (not a

or a singleton object
(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 HIB configuration
properties dynamically at time they will not be
reflected to the Session factory 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 means, modifications done 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 setDate1 (int a)
```

```
{ this.a = a;
```

```
public void setDate2 (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 date" + t.toString());
```

```
t.setDate1 (20);
```

```
t.setDate2 ("hello");
```

```
s.o.p ("t obj date" + t.toString());
```

Comment:- when setDate1(),
setDate2() methods are
called on "t" object
they directly modify
"t" object date

so, Test class

here is mutable class

```

class Test
{
    int a;
    String b;

    Test (int a, String b)
    {
        this.a = a;
        this.b = b;
    }

    public 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;
    }
}

```

Conclusion :-
when setDate1(),
setDate2() are called

on 't' object,

they are not modifying

't' obj date,

moreover they

are creating and

returning new objects

having modifications

related date, so

here

Test class

is called

immutable

class.

while developing user-defined immutable class, no java
method ab that class should have logic ab overriding
current invoking object date.

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

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

return temp;

4/8/10 IDENTIFIERS - (ALGORITHMS)

- Identity Value of HB POJO class object is the criteria.
Value too slow to perform synchronization b/w
HB POJO class object and table now.
- Hibernate supply lot of 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.Identifiers
Generator
all these classes having nick names (by) short
names to utilise.

<u>Nick Name</u>	<u>Algorithm class name</u>
1) assigned	\rightarrow org.hibernate.id.Assigned default algorithm.
2) increment	\rightarrow org.hibernate.id.IncrementGenerator
3) identity	\rightarrow org.hibernate.id.IdentityGenerator
4) sequence	\rightarrow org.hibernate.id.SequenceGenerator
5) hilo (high and low)	\rightarrow org.hibernate.id.Hilo TableHiloGenerator
6) seqhilo	\rightarrow org.hibernate.id.SequenceHiloGenerator

teoria
lehre

8) guid \rightarrow oog.hibernate.id. GUIDGenerator

9) native \rightarrow -

und

classe

10) select \rightarrow oog.hibernate.id. SelectGenerator

applied

strategies

11) foreign \rightarrow oog.hibernate.id. ForeignGenerator

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

If no algorithm is explicitly specified in mapping file
HIB SW ~~takes~~ take assigned as default algorithm.

\rightarrow 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 DB SW supports
the chosen algorithm.

ASSIGNED ALGORITHM -

This algorithm lets the application developer
to assign identity value to HB pojoian 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"/>

</id>

(i) assigned
org.hibernate.id. Assigned

<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 tx = ses.beginTransaction();

Integer idval = (Integer) ses.save(eb);

sop("id value is " + idval.toString());

tx.commit();

ses.close();

factory.close();

→ assigned algorithm can generate any type of identity values and assigned algorithm is compatible with all DB S/Ws.

⇒ Increment algorithm :- (existing value + 1)

This algorithm generates identity value of type long, short or int, this algorithm works with all DB 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 -->
  <generator class="increment"/>
</id>
```

⇒ Identity Algorithm :- X ORACLE (Hart!)

Supports identity columns 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 whether records in the table deleted or not that means it also considers deleted record values while picking-up ~~the~~ maximum value from the table column.

Ex:-

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

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

UNIQUE, NOTNULL, primary key, auto increment

* * * Q) 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

to
r
e
x
n
mybatis 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 ~~HIB~~ HIB S/W generates
identity value for HIB 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 HIB S/W
then HIB S/W assigns that
value as identity value for
POJO class object.

④ This algorithm works
with all DB S/Ws

④ this algorithm works with
only that DB S/Ws which
support identity columns.

5/8/10

SEQUENCE ALGORITHM:-

This algorithm uses sequence, created in DB2,
PostgreSQL, ORACLE, SAP, Mckoi, Interbase DB S/W to
generate identity 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 HIB mapping file use <param> tag as the subtag
of <generator> tag

→ create sequence my009_seq1 increment by 5; ORACLE
select * from user_sequences

Step①:- Create sequence in ORACLE DB SW as shown below :

SQL> Create sequence myseq_seq1 increment by 5;

Step②:- Write following code in HB Mapping file.

```
<id name="oo" column="EID">
  <generator column="sequence"> Algorithm name
    <param name="sequence"> myseq_seq1 </param>
  </generator>
</id>
```

↓ parameter name
sequence-name
Created in Oracle SW.

NOTE!- we can work with sequence algorithm only with those DB SWs which support sequences creation.

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

hilo :-

This algorithm uses hilo 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 SWs ; this algorithm expects the following ③ parameter values

- ① they are table (expects helper table name),
column (expects helper column name in the table),

59
shown how Table column value arrays depends
a number incrementation indicating how-many records
are inserted in the DB table by using hibo algorithm
based identity values.

example code :-

Step① :-

create helper table having helper column (make
sure that this helper column having 1 numeric value
~~so it's create table~~ as initial value).

SQL create table mytable(mycol number);

SQL insert into mytable values(10);

Step② :- configure hbo algorithm in HB mapping file
as shown below.

<id name="no" column="EID">

<generator class="hibo">

<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 :-

$(\text{max_lo} + 1) * \text{helper col val}$
(00)

$(\text{max_lo val}) * \text{helper col val} + \text{helper col val}$

formula to generate next values :-

$\text{max_lo val} + 1$

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

while working with this algorithm parameter passed ~~should~~ based on the capabilities of underlying DB S/w.

This algorithm works with all DB S/w's

when native algorithm is taken, by taking oracle as underlying ^{ORACLE} DB S/w, then this native algorithm internally uses sequence algorithm to generate identity values for HIB POJO objects.

Code in mapping file for ORACLE DB S/w :-

```
<id name="id" column="EID">  
  <generator class="native">  
    <param name="sequence">myseq</param>  
  </generator>  
</id>
```

while working with mysql DB S/w, 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 SW does not support both identity and sequence algorithms then the native algorithm internally uses hilo-algorithm.

Seq hilo 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 programmers, this algorithm works with only that DB SW which supports DB sequences.

(works with oracle, doesn't work with MySQL)

Parameter names for this algorithm are seq, max_lo

②

example code :-

step@1:- make sure that myora-seq1 is available in oracle DB SW.

step@2:- 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>
```

next val in sequence + increment by val of seq + 1)

$$* (\max_lo\ ray) + 1$$

Formula to generate next identity values -

previous identity val + ($\max_lo\ ray + 1$) + increment by
val of sequence.)

06/08/10

UUID algorithms - universal unique id

This algorithm uses, network 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 Schemas.

Example code -

① Take "no" number variable at EmpBean class as String variable.

② Change EID column datatype to varchar(35) in Employee table;

SQL → Alter table Employee modify EID varchar(35);

③ Configure uuid algorithm as shown below in mapping file.

```
<id name="EID" column="EID">
```

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

```
</id>
```

+1)

insert the below.

Transaction tx = session.beginTransaction();

String idval = (String) session.save(em);

System.out.println("id value is " + idval + idval.toString());

tx.commit();

quid :- global unique id

this algorithm generate uses DB generated GUID string

as identity value.

→ this only work with SQL SERVER, MySQL DBMS

scope

select :-

this algorithm generates identity value by executing triggers program in underlying DBMS.

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 algorithms :-

TL / PL decides identity value generator algorithm for POJO class object and HIS developer categorizes them programmatically.

If identity field DATA TYPE is numeric datatype

it is recommended to use increment(0) native algorithm.

numerical datatype then use assigned algorithm.

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

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

use <id> ab HBM file for singular identity

field configuration

use <composite-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 manipulation concept and use 'assigned' algorithm to set these values as identity values for POJO class objects.

=

If ~~DB~~ 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 HBM 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)

Name (VOL20)

Name (Voc (20))

email (VC(20))

for the above setup, we can write code in mapping

file as shown below.

~~hibernate - mapping~~

```
<class name = "EmpBean">
```

```
<ref name="no">
```

<generator class="Increment">

$\wedge \Phi / id$

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

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

<property name="mail"/>

</clan>

</hibernate-mapping>

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

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

<hibernate-mapping>

<class name = "PI.EmpBean">

≡ ↳ java package name

</class>

</hibernate-mapping>

(or)

<hibernate-mapping package = "PI">

<class name = "EmpBean">

≡

</class>

</hibernate-mapping>

≡

⇒

HIBERNATE SW SUPPLY SOME BUILT IN TOOLS TO

CREATE OR ALTER DB TABLES IN UNDERLYING DB SW

THESE TOOLS ARE ALSO USEFUL TO GENERATE HB POJO

CLASS DYNAMICALLY THEY ARE.

① Schema Export

② SchemaUpdate

③ CodeGenerator,

→ Schema Export tool always create new table

→ Schema 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 SW based on this mapping file.

name

new DB SW, ~~then~~ then these tools are very useful.

→ SchemaExport tool always create new table in DB SW.
If table is already available then it creates new
table by dropping 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="~~Employee~~" "EmpBean" table="Employee">

<id name="no" column="EID" length="10" type="int" />

<generator class="increment" /> ↳ size of the column

</ids>

HB Datatype

<property name="fname" column="FNAME" length="20"

not-null="true" unique="true" />

Applies Not null constraint on column. Unique constraint on column.

<property name="lname" column="LNAME" length="25"

type="string" />
small

<property name="mail" column="EMAIL" length="30"

type="string" />

</class>

</hibernate-mapping>

Step(2) :- execute SchemaExport command from command prompt
where HB configuration file and mapping file reside.

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

→ Hibernate Datatypes are the Bridge Datatypes b/w Java Datatypes and underlying DB SQL type Datatypes. They help the HB SQL to decide the Datatypes in DB SQL 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, float, double, byte, string ... etc

→ You can also specify Java wrapped 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 these datatypes (Hibernate, Java wrapped datatypes) is optional in hibernate mapping file. In this situation the Datatypes of POJO class member variables will be utilized as Reference Datatypes while creating columns in dynamic tables 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 SQL.

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 can alter existing table only by adding new
columns to table, based on ~~new~~ property configured
in hibernateMapping file.

example:-

step0:-

```
> java org.hibernate.tool.hbm2ddl SchemaUpdate [space]  
--config = mycfg.xml
```

CodeGenerator -

this tool is supplied in Hibernate 3.0, but not given
in 2.0 s/w. This tool is capable of generating
HIB persistence class (HIB POJO class) based on hibernate-
Mapping file.

sf = sourceForge

```
> java net.sf.hibernate.tool.hbm2java CodeGenerator
```

--properties = hibernate.properties [space]

Employee.hbm.xml

HIB Mapping file ← =

• properties file acting as
+ HIB configuration file.

- to perform hbm2ddl operations.
- ⇒ CodeGenerator tool design to perform hbm2ddl operations.
- ⇒ In Hibernate 2.0.x the HIB configuration file is Properties file.
- ⇒ we can use SchemaUpdate, SchemaExport tool along with HIB Application execution by parsing special properties in HIB 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 files-

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

Always create new table in each execution

Example②:-

→ configuration files-

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

Can create new table (or) can use existing table (or)

can alter existing table by using SchemaUpdate tool

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

<property name="hibernate.create-ddl" value="auto"/> create-dao

</property>

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

→ create-dao ~~of~~ hibernate.create-ddl.auto property
is quite useful in the testing mode of project
Table creation and destruction is required after testing
the project with Total (dummy) data.

Connection Pooling

⇒ Connection Pool is a Factory that contains set of
readily available JDBC Connection objects.
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 Connection pool.
(default)

② Third party supplied Connection Pools (JDBC)
(like C3P0, proxool)

③ web-server (or) application server managed JDBC
connection pools.

not suitable for production mode environment
in 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 HIB configuration file.

property name="hibernate.connection.pool-size">30</property

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

⇒ If the project type is stand-alone application, which can run outside the web servers and application servers use 3rd party connection pool like C3PO (or) PoolXool.

If project is Server Managed application like Web-application or EJB component use that Server Managed JDBC Connection pool.

NOTE:- Don't use HIB 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 HIB Configuration file.

- ② the connection s/w that it is used to create JDBC Connection Pool represented by Session Factory object.

⇒ This property takes a built-in class name & value
the possible values are.

① org.hibernate.connection.DriverManagerConnectionProvider
(default value uses hibernate s/w supplied built-in connection pool).

② org.hibernate.connection.DataSourceConnectionProvider
(uses Web server (or) Application Server managed JDBC connection Pool.)

③ org.hibernate.connection.C3P0ConnectionProvider
(uses C3P0 s/w managed Connection Pool.)

④ org.hibernate.connection.PoolConnectionProvider
(uses the pool s/w managed JDBC Connection Pool)

→ Step 1:- Add C3P0 Pool created Connection Providers Classname in HIB configuration file of the application.

→ <property name="hibernate.connection.provider-class">
org.hibernate.connection.C3P0ConnectionProvider
</property>

Hibernate 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
Sku).

Step①:- Run the client application in the regular
model.

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

Step①:- Add Poolable pool related Connection Provider
class in HB configuration file.

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

Step②:- Add Poolable Pool related property is in
HB Configuration Prop. file.

<property name="huberacle.pool.xml">

logic

</property>

(property)

- Note:- here we configure the another XML file ~~file~~ ~~as~~ ~~as~~
Helper XML file too \Rightarrow HB Config file.

(property)

> 2

tip>

ile

pool

8

Nonpool

108

>

ovideo

sty>

Note(1):- place this file in the directory where the same
HB configuration file resides.

Note(2):- since myfile.xml itself is containing
url, driver-class, ~~useoname~~ and password details
so there is no need of specifying same details.

Step①:- Add `hibernate-home\lib\poolxml-0.8.3.jar` file to the class path (This jar file represents poolxml SW)

Step②:- execute the client application.

Understanding Server Managed Connection Pools :-

Server Managed Connection Pool will be managed in web-server or Application Servers. 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 DataSource interface. To provide global visibility to DataSource object it will be registered in registry SW having nickname. Every webserver/Application Server ^{or your supplier} has one built-in registry SW.

→ Examples of registry SW's :-

RMI Registry :- ① COS (Common Object Service)

② LDAP Registry (Lightweight 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 object to registry (insert)

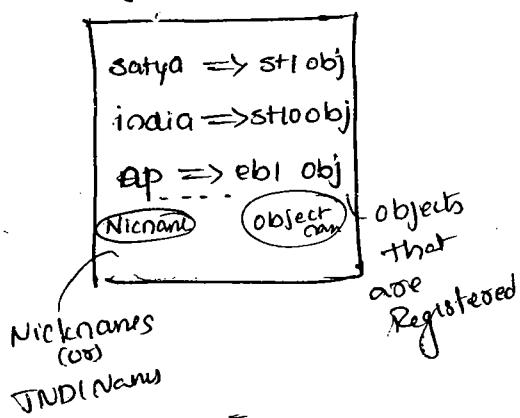
ReBind → Replacing existing object with new object (update)

UnBind → Removing object from Registry (delete)

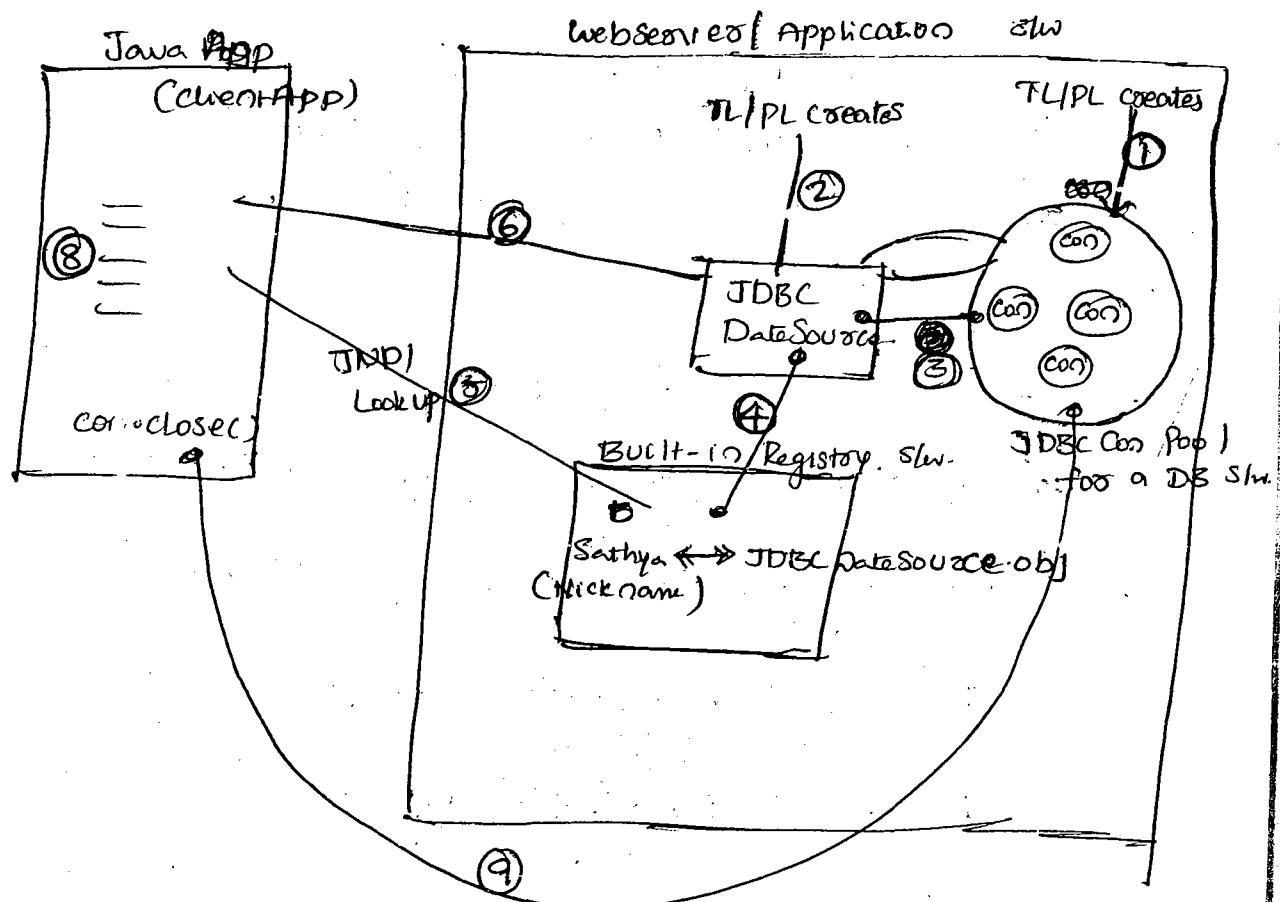
List operation → Retrieving all objects that are bound to Registry (select)

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

Registry S/w



→ To get access to JDBC Connection objects ab connection Pool (Server Manager) program must get access to JDBC Datasource object which is representing that connection.



- ① TL/PL creates JDBC Connection Pool for DB SW
- ②, ③, ④ TL/PL creates Datasource object pointing to connection pool and Registers the Datasource object with registry having nickname 'foo'
- ⑤ Client app uses JNDI and gets Datasource object from registry through lookup operation
- ⑥ Client application uses this Datasource object to get JDBC Connection object from connection pool.
- ⑦ Client app ~~creates~~ creates other object based on this connection object and writes the persistence data
- ⑧ JDBC Connection Pool releases connection back to pool.
- ⑨ Client app closes connection

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

to 8/10

WEBLOGIC :- (9/10)

- Application Server = web container / servlet container + EJB container + additional services
- Web Server = web container / servlet container + middleware services
- Application Server SW is enhancement of Web Server having enhanced facilities and features.

weblogic 8/86 -

Type :- Application Server SW

Version :- 9.0.x (compatible with J2SDK 1.5)

Vendor :- BEA Systems (Oracle SW).

Port no :- 7001 (default)

Commercial SW :-

To download SW :- www.commerce.bea.com

For document :- www.edocs.bea.com

Allows to create domains.

Default domain name is :- example.bea.com

If multiple projects of a companies are using

Same weblogic SW, then weblogic SW 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 Data Source In Example Server Domain

ab weblogic 9.x :-

Step 01:- Start example server domain ab web-logic.

start

→ programmes

→ BEA products

→ Examples

→ Weblogic Servers

↓
Start Example
Server.

Step 02:- open administration console ab

example servers domain.

→ Open Browser Window

↓

Type "http://lb:7001/console" w/o

user :- weblogic

password :- weblogic

↓
login.

Step 03:- create JDBC Data Source from Administration Console

pointing to JDBC Connection pool ab Oracle

Admin Console

→ Look & Edit

→ services

→ JDBC

→ Data Sources

→ New

~~domain~~

Name :- myorad

logical name

main

DBType :- oracle

DB Divers :- oracle then Divers

↳ Next

↳ Next

DB Name :- satya <sql> Select * from global_name

host name :- lh.

port no :- 1521

User :- SCOTT

~~Pass~~ Pass :- tigero

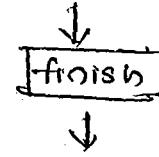
control pnd :- tigero

↳ Next

↳ Next

↳

Example Services.



Configure parameters of JDBC ConnectionPool :-

Services

↳ JDBC

↳ DataSources

↳ myoracles

↳ Connection Pool

↳ Lock & Edit

⋮

Capacity increment :-

Specify the no. of JDBC connection object it

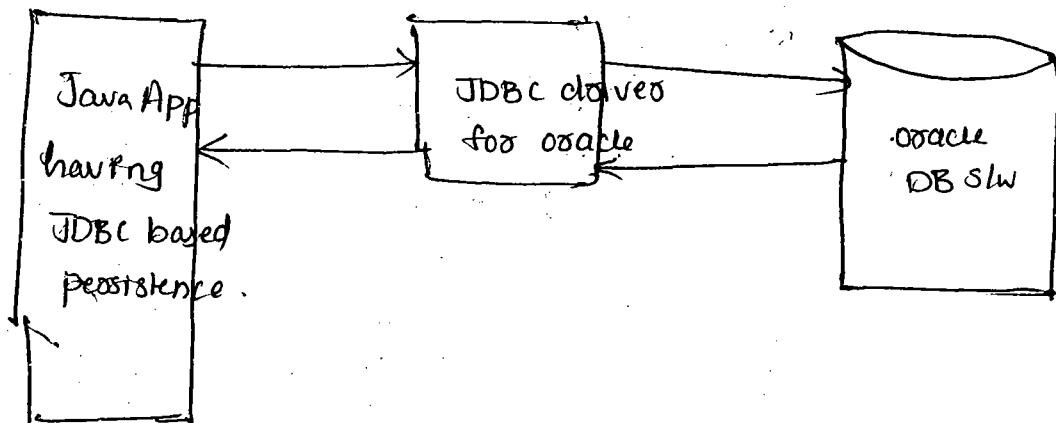
should be created in connection pool at a time. If there is a need of creating new JDBC connection object.

↓
Tennet

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 also 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 :-

deploys connectivity with DB S/W. To create this connection object the following details are required (JDBC Driver class name

- ① DB URL
- ② DB Username
- ③ 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 SLW

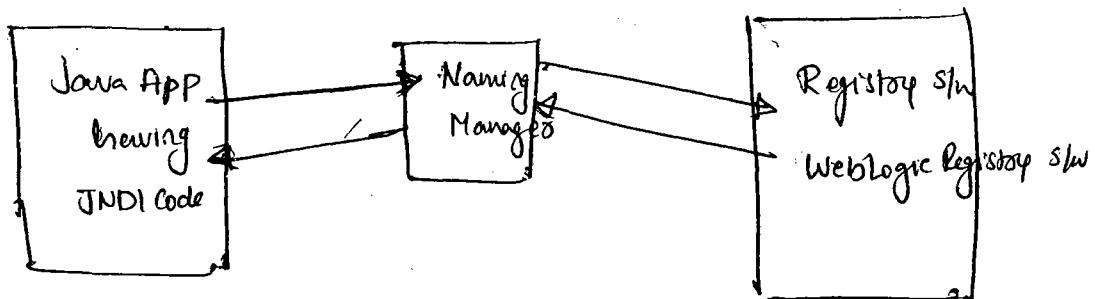
Every Naming Manager will be identified with its InitialContextFactory classname. This classname change based on Naming Manager and Registry SLW we used:

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

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

① InitialContextFactory class name of Naming Manager

② Provider URL of Registry SLW.



→ JNDI properties values of weblogic Directory Registry SLW

→ InitialContextFactory class name :-

weblogic.jndi.WLInitialContextFactory

This class is available in ~~BEA-HOME~~

TSEA-HOME\WEBLOGIC-N\sever\lib\weblogic.jar file

→ provider URL :- t3://<hostname>&<portno>

Connection Pool in stand alone Environment based

Hibernate Application :-

Step(1) :- Configure connection providers class, HB Configuration file.

<property name="hibernate.connection.provider-class">
org.hibernate.connection.DataSourceConnection
Provide </property>

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

<property name="hibernate.jndi.url"> t3://elb:7001 </property>

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

↓
InitialContextFactory class

(3)e - Specify JNDI Name (or) NickName of JDBC Datasource
created in weblogic server

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

Note :- since JDBC Datasource name, DBurl, DBusername,
DBpassword values are specified in server
admin console while creating JDBC Connection Pool

so there is no need of specifying them in
HB configuration file.

file to the class path.

→ Make sure that example scores domain of weblogic
weblogic.9.0.x is Running Mode and execute the class Application

can't

action

>

logic

use

s/pool

:27)

JEE

11/08/10

Procedure to create jndi datasource and jdbc connection

pool for Oracle in GlassFish application server :-

Step 1:- start GlassFish server

start → programs → sunmicrosystems → application servers
Start default server ↗

Step 2:- open admin console of GlassFish server

start → programs → sunmicrosystems → application servers → admin console
username : admin ↗
admin ↗

Step 5 :- Create JNDI

~~Admin~~ admin console

→ Resources

→ JDBC

→ connection pool

→ new

↓

name : mypool1

Resource type : javax.sql.DataSource

DB Vendor : oracle

→ next

→ Initial Capacity : 8 - 10.

Maximum Pool size : 20.

pool size : 2

DB name write "satya"

DataS

✓ DataSourceName : myds.

✗ port.

✓ DataBase Name : ~~satya~~ satya,

password : tiger

port name : 1521

server name : lh

✓ URL : jdbc:oracle:thin@lh:1521:satya

Users : SCOTT

[add property]

Driver Class : oracle.jdbc.driver.OracleDriver.

→ [Finish]

→ mypool1

→ [ping]

ping succeeded

→ [Save]

Glassfish | AppServer | lib is contain ~~@~~ `ojdbc14.jar`

Step④ \Rightarrow Create JDBC DataSource pointing to the above created connection pool

admin-console

\hookrightarrow Resources

\hookrightarrow JDBC

\hookrightarrow JDBC Resources

\hookrightarrow New

JNDI Name GSTD_Indi \leftarrow Any-name

\downarrow
OK

\Rightarrow Procedure to use GlassFish Server managed JDBC Connection Pool in stand alone environment based HIB application.

Step①:- same as Weblogic Server.

Step②:- specify JDBCDataSource JNDI name, in HIB configuration file.

`<property name="hibernate.connection.datasource"> SSD_Indi </property>`

\Rightarrow NOTE! - there is no need of adding InitialContextFactory class name, provider url value in HIB configuration file, while working with GlassFish Server supplied Registry file.

\Rightarrow Add following two jar files to the classpath.

AppServer.jar

② GlassFish Home \ AppServer \ appserver-rt.jar

javaee.jar

\Leftrightarrow execute the client DB Application.

WEB RESOURCE PROGRAMMES :-

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

Web-application :-

→ While developing HIB persistence logic in java application it is recommended to create HIB Session object in one time execution block and it is recommended to use this HIBSession object for persistence operations in repeatedly executing blocks.

Type	Place to create HIB Session obj	place to use HIB Session obj
1) AWT/swing Frame	static block / Constructor	Event Handling methods
2) Applet/JApplet	static block / Constructor / init()	Event Handling methods
3) servlet	static block / Constructor / init()	service(-,-)
4) JSP	<? public void jspInit() ?>	script-let code

5) EJB Session Bean Component

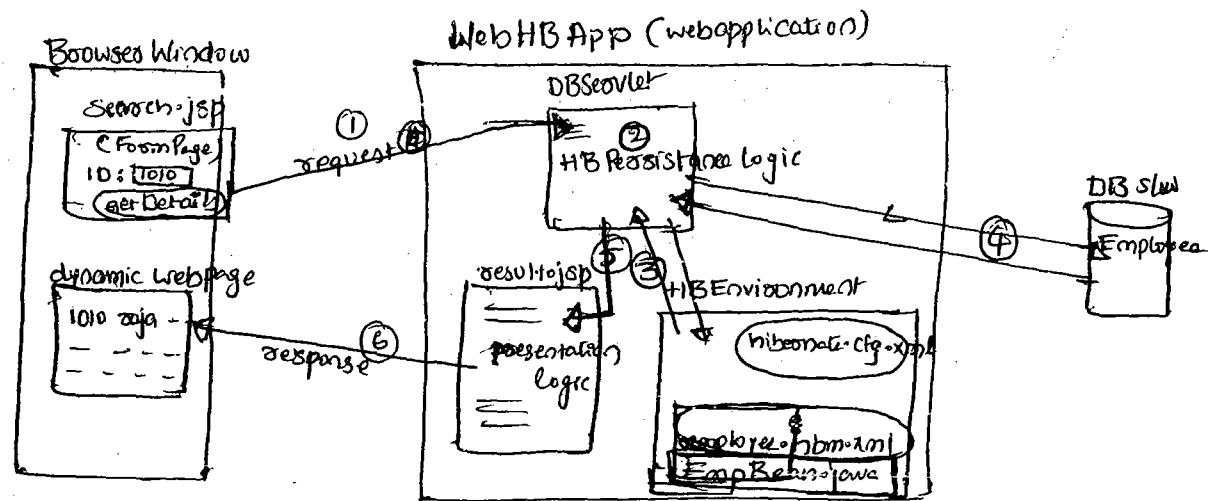
ejbCreate() / constructor
static block

Business Rule

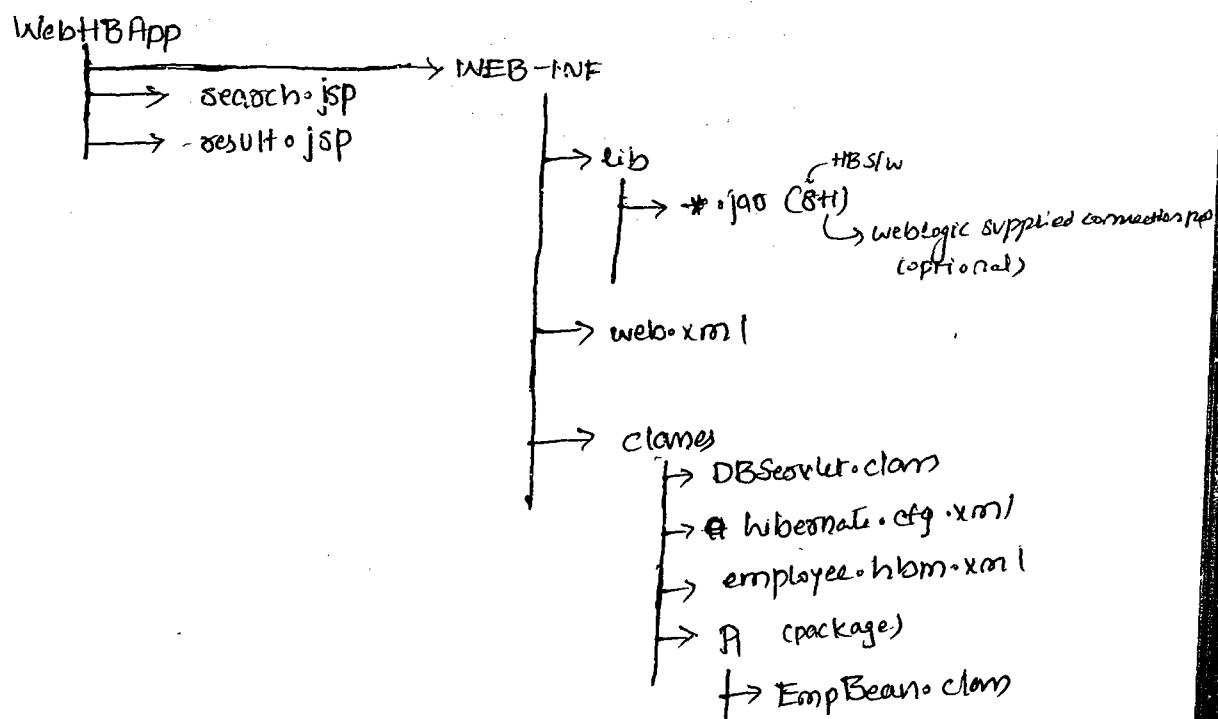
6) Struts App

static block / constructor

execute() ->



Deployment Directory structure:-



- In the above application DBSevlet is using **HIBPostgreSQL** logic to interact with DBS/W.
- In order to pass data from servlet to JSP when 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 use weblogic9.x server managed JDBC connection pool.
- If any application that is running outside the Weblogic servers wants to interact with weblogic directory Registry then the application must use JNDI Properties like InitialContextFactory class name, providers 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.connection.DriverManagerConnectionProvider </property>

<property name="hibernate.connection.datasource"> SathyaJdb

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

<session-factory>

<hibernate-configuration>

object

→ Do observe that configuration file not having hibernate-jndi class, hibernate-jndi.url properties holding JNDI properties. (Initial Context Factory class name, provider url)

12/08/10

EmpBean.java :-

same as first application but make sure that it is available in a package "P1"

web\BAPP\WEB-INF\classes > javac -d . EmpBean.java

Employee-hbm.xml :-

same as 1st app

Search.jsp :-

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

 Employee ID

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

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

</form>

=

">

hyp tool

```

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

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

public class DBServlet extends HttpServlet
{
    Session sess = null;

    public void init()
    {
        try
        {
            SessionFactory sess = new Configuration().configure().
                buildSessionFactory().openSession();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    public void doGet(HttpServletRequest req, HttpServletResponse res)
    {
        try
        {
            // read form data
            int no = Integer.parseInt(req.getParameter("tid"));
            // write to persistence layer
            EmpBean eb = (EmpBean) sess.get(EmpBean.class,
                new Integer(no));
            result = eb;
            res.setAttribute("result", eb);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}

```

If 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();

}

} // if doGet

public void doPost (HttpServletRequest req, HttpServletResponse res)

{ super.doPost (); // DBService
doGet (req, res);

}

catch (Exception e)

{
e.printStackTrace();

} // catch

} // doPost

public void destroy ()

{ super.destroy (); // DBService

} // try

res.close ();

} // catch (Exception e),

{

e.printStackTrace();

}

} // destroy

} // class

web.xml :-

Configure DBSeover by name ~~DB1~~ /DB01 as val pattern

result.jsp :-

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

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

<h> The Details are </h>

<% = eb.getNo() %>

<% = eb.getFname() %>

<% = eb.getLname() %>

<% = eb.getEmail() %>

⇒ In WEB-INF/lib folder of above web application
place the HBAPI related JAR files.

Procedure to Deploy web-app in weblogic 9.x servers

Step 01:- prepare war file on the deployment directory
structure of web-application.

> jar cf webHBApp.war <-----

above step gives webHBApp.war file representing
jar file

→ There are 3 ways to deploy application in servers

① Hand Deployment

② console Deployment

③ Tool Base Deployment

NOTE:- this application is using console Deployment process.

→ open administration console at example1 server Notebooks
weblogic 9.0.0.

→ Deploy the war file <webHBApp.war>

Admin Console

↳ Lock & Edit

↳ Deployment

↳ Install

↳ upload your files

↳ Browse and
select to webHBAPP.war

↓
Next

Select WebHBApp.war

Next

Next

Next

Finish

Activate changes

↳ Deployment

↳ select webHBAPP webApplication

↳ start

Test the webApplication

open Browser window:- <https://localhost:7001/WebHBAPP/search.jsp>

BULK OPERATION RELATED TECHNIQUES :-

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

But HQL Most Recommended Technique:

- 1) HQL
- 2) NativeSQL
- 3) Criteria API

HQL :-

It is the most popular persistence technique environment in Hibernate Programming

- 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 members 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 as Oracle.
- ⑤ HB SW converts HQL queries into SQL queries and sends them to DB SW for execution.
HQL Queries supports operators, expression, condition clauses, joins, subqueries, aggregate 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

Limitation:- ① position parameters ② named parameters.

① HQL query can't perform DDL operations

② HQL queries can't be used in PLSQL programming

③ HQL queries can't be used to insert single record into table.

④ HQL queries use negligible performance degradation bcz ab conversions than compare to SQL.

13/08/10

→ table name

SQL > Select * from Employee

HQL > Select eb from EmpBean as eb (0s)

HQL > ~~select~~ from EmpBean as eb (0s) → POJO class name

HQL > from EmpBean (0s)

HQL > from EmpBean eb

NOTE:-

when select keyword is there or POJO class member variables are there in HQL Query then creating alias name for POJO class is mandatory along operation.

~~HQL~~ select EID, LASTNAME from Employee

→ column names → table name

where EID >= 100
 → column name

~~HQL~~ select eb.no, eb.ulname from EmpBean as eb

where eb.no >= 100
 → column name
 → pojo class
 members variable ←

→ HQL keywords are not case sensitive but POJO class name and POJO class number variable used in HQL Query are case sensitive.

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

ex:- 'oaja', 'hyd'

~~SQL~~ select count(*) from Employee *

→ tablename

~~HQL~~ select count(*) from EmpBean

→ pojo class

~~SQL~~ delete from Employee where fname in ('oaja', 'oavi')

~~HQL~~ delete from EmpBean ~~as~~ eb where
 eb.fname in ('oaja', 'oavi')

- To execute select HQL Queries, use list()
iterate() or Query obj similarly to execute
 non-select HQL Queries call executeUpdate() on
 Query object.
- execution of HQL Query is nothing but converting
 HQL Query to the underlying DB specific SQL Query
 and sending that SQL Query to DB SW for execution.
- Hibernate 3.0.x SW internally uses, AST Query Translation
 to convert HQL Queries into DB SW equivalent SQL
 Queries.

⇒ PO ≡ imports

public class HQLTest

6

P S v main(String args[]) throws Exception.

6

//create HB Session object

SessionFactory factory = new Configuration()
configure("mycfg.xml").buildSF();

Session ses = factory.openSession();

Query q1 = ses.createQuery("select eb from EmpBean as
 ar1").
eb");

List l = q1.list(); // execute HQL

//display records

for (int i=0; i < l.size(); i++)

{ EmpBean eb = (EmpBean) l.get(i); }

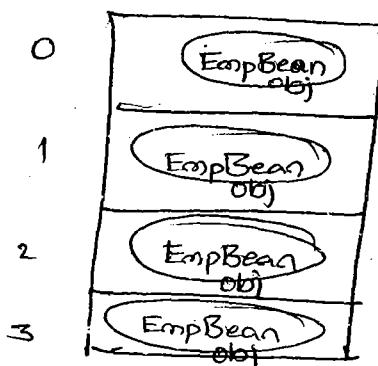
```
s.o.p ( e1.getNo() + " " + e1.getFinance() + " " + e1.getLname() +  
" " + e1.getMail());
```

{}

—

Object ~~copy~~ memory representation:-

```
l (java.util.List obj)
```



⇒ Executing ① HQL select Query by using ~~iterate~~ method.

Ex:-

///

Query q1 = ses.createQuery ("select eb from EmpBean
as eb");

Iterator it = q1.iterator(); // executes HQL

while (it.hasNext())

{

EmpBean e1 = (EmpBean) it.next();

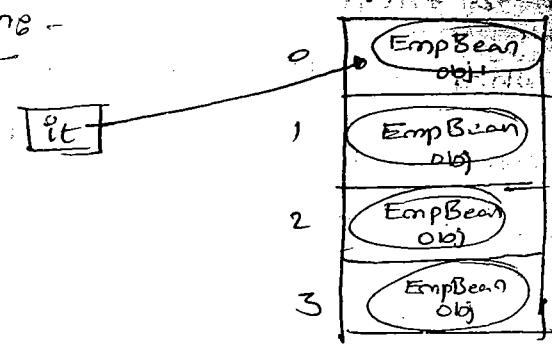
```
s.o.p ( e1.getNo() + " " + e1.getFinance() + " " +  
e1.getLname() + " " + e1.getEmai());
```

}

}

hand)

Memory diagrams -



Q) what is the difference between executing single SQL query by using list() & by using iterate()?

=> list() method generates result by selecting all records through the execution of single select SQL query -

d.
7
d.
7

(No Lazy Loading) that means if results objects are used (or) not used by the programmes (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 & initializes results related HB pojo class object on demand.

- List() method is recommended to use
- we can see the above two by list() and iterate().

of a table.

Q) what is the difference b/w executing

SQL Select Queries in JDBC & HQL Select Queries
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 DateStructure. Since List DateStructure object is serializable object by default, we can send the object over the network.

① Note:- all the collection Fw R's 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.fname like '%oy%'");

List l = q1.list();

for (int i = 0; i < l.size(); i++)

EmpBean e1 = (EmpBean) ls.get(i);

SOP (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 ~~DBMS~~ storage
→ when HQL Select Queries are selecting specific columns 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();

If display results

for(int i=0; i<l.size(); i++)

{

Object oow[] = (Object[]) l.get(i);

for(int k=0; k<oow.length; k++) {

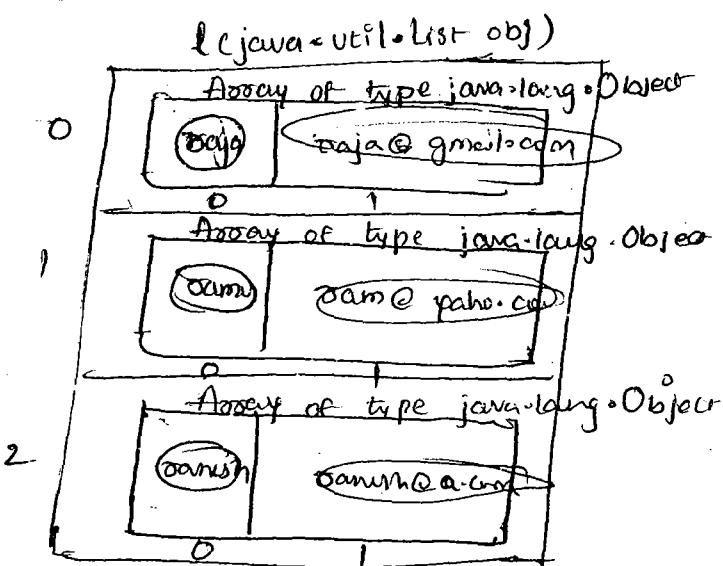
SOP (oow[k].toString() + " ");

} inner loop

SOP ();

} outer loop

Nearby Diagrams :-



→ 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 ~~Object~~ java.lang.Object class objects

Example - executing ~~the SQL Select Query that~~ ~~Select Specific~~ columns values by using iterateC()

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

```
| Heats > it = q1->iterate(); // execute TSQL Query
```

```
while (!it->hasNext())
```

6

Object :: doW [] = (@object []) it :: next () ;

```
for (int k=0; k<roww.length; j++)
```

S.O.P (now [k] to string() + " ") ;

When `iterate(C)` is used to select specific column values of HQL select query then no lazy loading will take place.

example:-

Executing single Aggregate function based HQL Select Query

Query `q1 = ses.createQuery("select count(*) from EmpBean eb");`

List `l = q1.list();`

S.O.P C "count of records in table are "+ l.toString());

Executing HQL query having multiple Aggregate functions and Utility functions

Query `q1 = ses.createQuery("select count(*), avg(eb.no), upper('rpa'), sum(eb.no), max(eb.no) from EmpBean eb");`

List `l = q1.list();`

S.O.P C "Results are "+ l.toString());

Object `oer[] = (Object[]) l.get(0);`

S.O.P ("count is "+ oer[0].toString()));

S.O.P ("avg is "+ oer[1].toString()));

S.O.P ("upper is "+ oer[2].toString()));

S.O.P ("sum is "+ oer[3].toString()));

S.O.P ("max is "+ oer[4].toString()));

Q `l(java.util.List obj)`

java.lang.Object class obj				
4	256	1024	RAJA	567

Example on Subqueries in HQL environment

Query $q_1 = ses.createQuery("select eb from EmpBean as eb$

~~where eb.no = (select max(eb.no) from~~
~~EmpBean as eb1)");~~

List l = q1.list();

for (int i=0; i < l.size(); i++)

{

EmpBean ei = (EmpBean) l.get(i);

System.out.println(" "+ei.getname()+" "+ei.getlname()+" "+
" "+ei.getHai());

}

=

- ⇒ To make HQL query flexible by setting input values
as Query from outside Query and to set Query if p
Values as javaside 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 SW generated JDBC code internally uses precompiled SQL Queries with the support of prepared statement object to perform persistence operations on the table.
- All HQL Queries related SQL Queries generated by HB SW are precompiled Queries by default.
 - Parameters in HQL

(1) positional parameters (?)

(2) Named Parameters (:<Name>) → Recommended 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, "y@gmail.com");

↓
parameters
index

↓
parameters value

List l = q1.list();

for (int i=0; i<l.size(); i++)

{

EmpBean eb = (EmpBean) l.get(i);

System.out.println(eb.getNo() + "");

}

⇒ we can'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 (00) invalid HQL query.

Query $q_1 = ses.createQuery("select eb from \underline{?} as eb where eb.name \geq ? \text{ and } eb.mail like ?")$; unexpected token.

$q_1.setString(0, "EmpBean");$

$q_1.setInt(1, 30);$

$q_1.setString(2, "%@.com");$

~~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 $q_1 = ses.createQuery("select eb from EmpBean as eb where eb.fname in (:P1, :P2) \text{ and } eb.mail like :P3");$

// setting values for parameters

$q_1.setString("P1", "Sajal");$

$q_1.setString("P2", "Soni");$

$q_1.setString("P3", "%.%@.com");$

List l = q1.list();

-for (-)

b
=

↓

=

~~in JDBC programming, we can't pass Named parameters in SQLQuery; To HQL Query both Named,~~

~~Positional parameters in a single Query.~~

But we must pass positional parameters before Named parameters.

Query $q_1 = ses.createQuery("select eb from EmpBean as eb where eb.fname @ in (?,?) \text{ and } eb.mail like ?:P3");$

$q_1.setString(0, "Sajal");$

$q_1.setString(1, "Soni");$

$q_1.setString("P3", "%.%@.com");$

Valid

Invalid HQL Query :-

Query $q_1 = \text{ses.createQuery} (" \text{select eb from EmpBean as eb} \text{ where eb.fname ' in } (\text{?P}_1, \text{?P}_2) \text{ and eb.email like } \%")$

Invalid {
 $q_1.setString ("P_1", "saya");$
 $q_1.setString ("P_2", "xx");$
 $q_1.setString (0, "%s.com");$

Executing Non-Select SQL Query (Insert/Update/Delete) :-

- must be executed as transaction 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 tx = ses.beginTransaction();
Query q1 = ses.createQuery (" delete from EmpBean as eb
where eb.no = (select min(eb.no) from EmpBean web)");
int res = q1.executeUpdate();
System.out.println ("no of record effected : " + res);
tx.commit();
```

HQL Non-Select * Query with Parameters :-

```
Transaction tx = ses.beginTransaction();
Query q1 = ses.createQuery (" update EmpBean as eb set
eb.mail = ? where eb.no >= ?P1 ");
q1.setString (0, "n@gmail.com");
q1.setString (1, "1000");
int res = q1.executeUpdate();
```

```
System.out.println ("no of record updated : " + res);
tx.commit();
```

⇒ HQL Insert Query is not given in this form Insert into

EmpBean as eb values (?, ?, ?, ?) \Rightarrow Wrong Query

⇒ Insert into EmpBean eb values (?, ?, ?, ?) \Rightarrow Wrong

⇒ The supported Form is HQL Insert Query is

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

i.e. using this syntax we can't ~~insert~~

⇒ we can't insert directly records to table,
we can insert record only by selecting from another
table.

Example:-

Source table <Employee>

③ same as first application.

destination table <Employee1>

EID → number PK

Name → varchar(20)

Email → varchar(20).

EmpBean.java:-

same as first appln.

EmpBean1.java:-

→ int no, String name, String email

→ we can configure multiple HIB 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="EmpBean" table="Employee1">
    <id name="eid" column="EID"/>
    <property name="name" column="NAME"/>
    <property name="mail" column="MAIL"/>
</class>
<hibernate-mapping>

```

Code in client appng - 'HQLtest.java'

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

Transaction tx = 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 >= ?1");
q1.setParameter("P1", 300);

int res = q1.executeUpdate();

s.out.println("No of records effected : " + res);

=

↓

For related information on HQL, See more example
on HQL defined chapter 13 & 14 in pdf file.

17/8/10

Pagination -

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

- Hibernate persistence logic provides Environment i.e required for pagination.
- All the 3 bulk operations techniques supports pagination.
- Even though select query selecting huge amounts of records in order to select specific range of records from those records use setFirstResult() & setMaxResults() methods are shown below.
- we can also use this method for Pagination.

Ex:-

Query q1 = session.createQuery ("select eb from EmpBean
as eb");

|| select ③ records from 300 records (0 based index)

q1.setFirstResult(2);

q1.setMaxResults(3);

List l = q1.list();

for (int i=0; i<l.size(); ++i)

6

EmpBean e1 = (EmpBean) l.get(i);

s.out.println(e1.getNo() + " " + e1.getName() + " " + e1.getNa() +
" " + e1.getHireDate());

g

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

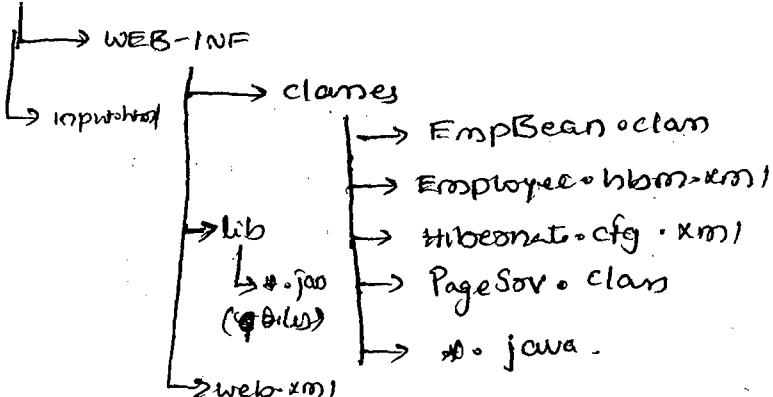
-> 'downum' is a invisible pseudo column in DB tables of Oracle SQL holding downo's as column values.

-> To perform pagination by using HIB persistence logic in web-applications, take fixed no for maxResult and change FrostResult value based on the RequestNo (00) page no., To make these operations specific to each client use session attribute to manage Frost result value ab each page.

Deployment Directory structure of web-appn that performs pagination by using HIB persistence logic :-

~~WEB-INF~~

page HIB APP



=
jar files in WEB-INF \ lib folder

- (2) → lib jar files
- (1) → jdbc jar files

jar files in class path:- → SERVLET-API.jar
→ hibernate3.jar

EmpBean.java :-

↔ same as 1st appn →

Employee.hbm.xml :-

↔ same as 1st appn →

hibernate.cfg.xml :-

↔ same as 1st appn →

Input.htm :-

<center> Generate Report </center>

↓ </center>

Use pattern of PageSov servlet

web.xml :-

Configure pageSov Servlet by having 1 pageSov as

Servlet

PageSov.java :-

```

import java.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import org.hibernate.*;
import org.hibernate.cfg.*;
import java.util.*;
```

public class PageSov extends HttpServlet

{

Session ses = null;

int interval;

int totrecords;

public void init()

{
try
{

}

}

}

```
catch (Exception e)
{
    e.printStackTrace();
}

}

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 eb");
    List l2 = q2.list();
    totorecords = l2.size();

    PrintWriter pw = res.getWriter();
    res.setContentType("text/html");

    // create / locates session for client
    HttpSession se = req.getSession();
    if (session.getAttribute("counted") == null)
    {
        // logic of first request
        interval = 0;
        ses.setAttribute("counted", new Integer(interval));
    }
    else
    {
        // logic for other than 1st record.
        Integer i1 = (Integer) session.getAttribute("counted");
        initial = i1.intValue() + 2;
        session.setAttribute("counted", new Integer(interval));
    }

    // logic to display records in form of html table
    // by applying pagination.

    Query q1 = ses.createQuery("select eb from EmpBean as eb");
}
```

```

q1 = getFirstResult (init val);
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("<td><td>" + e.getNo() + "</td> @
<td>" + e.getFname() + "</td>
<td>" + e.getLname() + "</td>
<td>" + e.getEmail() + "</td></tr>");
}

pw.println("</table><br>");

// logic to display home or next hyperlinks in the
// page
if ((interval+2) < totRecords)
{
    pw.println("<a href='page"+(interval+1)+"'>next</a>");

}
else
{
    pw.println("<a href='input.html'> home</a>");

    session.invalidate();
}

pw.close();

} // service.

public void destroy()
{
    try
    {
        ses.close();
    }
    catch (Exception e)
    {
        ...
    }
}

```

{ } || destroy

{ } || clear

18-08-10

Native SQL :-

- Native SQL queries are underlying DB specific SQL queries. These queries based persistence logic of HIB. It is DB dependent persistence 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.

- There 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 results.

Native SQL Query results must be mapped with Hibernate POJO classes or HIB Date 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 programmers to call PL/SQL procedures and functions.

ab DB slow from HIB ~~programming~~ language logic -

⇒ Example on Entity Native SQL Query (or) select Query
that select all column values :-

~~Step 1~~

SQLTestClient.java:-

← tableName



/ alias name

String sql = "select {e.*} from Employee e";

SQLQuery q1 = ses.createSQLQuery(sql);

// map entity query result with HIB 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 ei = (EmpBean)l.get(i);

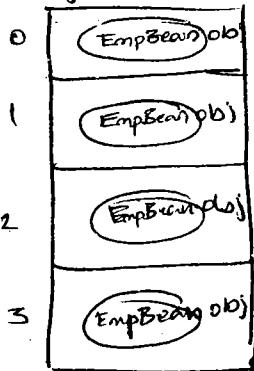
System.out.println(" "+ei.getFname()+" "

ei.getLname()+" "+ei.getHname());

}

}

java.util.List obj



→ we can't call `iterate()` on SQL Query object to execute entity Query (o) select Query or NativeSQL. This indicates lazy loading (o) lazy initialization of t1B 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 t1B 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 ei = (EmpBean)l.get(i);

System.out.println(ei);

}

=

⇒ Entity - NativeSQL must be mapped with t1B POJO class.

⇒ we can pass both Named and positional parameters in NativeSQL Queries.

In order 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 <=?";

SQLQuery q1 = ses.createSQLQuery(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++) {
        System.out.println(l.get(i));
    }
}

```

⇒ Like JDBC Date types in the form of `TYPES.INTEGER`, `TYPES.TIMESTAMP`, `TYPES.VARCHAR` and etc, HB also giving its own built-in datatypes as constants of `org.hibernate.Hibernate` class,

ex- `Hibernate.INTEGER`, `Hibernate.FLOAT`, `Hibernate.LONG`, `Hibernate.STRING` etc.

These Hibernate datatypes are useful while catching Scalar Query Results, Scalar NativeSQL Query results in Hibernate environment.

public static final members variables of java class or interface are called constant.

Example on Scalar Queries -

```

    // Native SQL scalar query.
    String sql = "select max(cid) as max from Employee";
    SQLQuery q1 = ses.createSQLQuery(sql);
    // mapping scalar query result with HB datatype

```

// executes Native SQL Query

List l = q1.list();

S-O-P C "max value of eid col is " + l[0].toString();

~~for~~

=

{

⇒ making Scalar Query returning more than one result.

1) Native SQL Scalar Query

String sq1 = "select max(eid) as mval, count(*) as cnt from Employee";

SQLQuery q1 = ses.createSQLQuery(sq1);

// mapping scalar query result with HIB date types

q1.addScalar("mval", Hibernate.INTEGER);

q1.addScalar("cnt", Hibernate.INTEGER);

// executes Native SQL Query.

List l = q1.list();

// display results

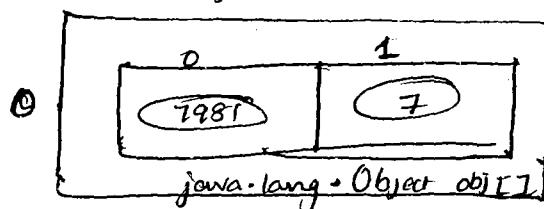
Object oes[] = (Object[]) l.get(0);

S-O-P C "max val in EID col is " + oes[0].toString();

S-O-P C "count of records" + oes[1].toString();

=

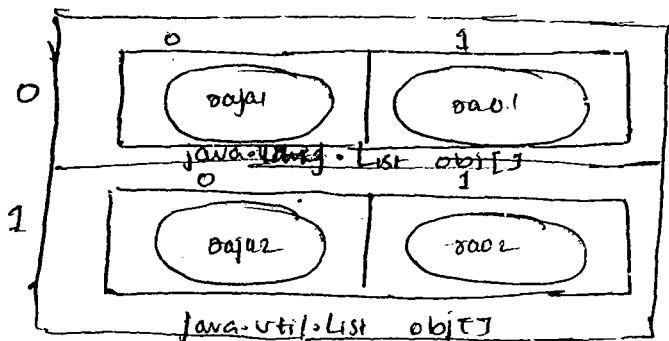
l [java.util.List obj]



Example on Native SQL Scalar Query that is selecting specific

columns values of table 3 -

{(java.util.List obj)}



String sql = "select FIRSTNAME, LASTNAME from employee
where email like ?";

SQLQuery q1 = ses.createSQLQuery(sql);

// set parameters values

q1.setString("?", "x@gmail.com");

// map scalar query results with +B 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 oout[] = (Object[])l.get(i);

for (int k=0; k<oout.length; ++k)

{
System.out.print(oout[k].toString() + " ");

}
System.out.println();

}

=

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

String sq1 = ses.createSQLQuery(sql);

q1.setIntegers("P");

String sq1 = "insert into Employee values (:P1,:P2,:P3,:P4);"

SQLQuery q1 = ses.createSQLQuery(sq1);

q1.setIntegers("P1", 824);

q1.setString("P2", "Oaja");

q1.setString("P3", "Oam");

q1.setString("P4", "oaja@gmail.com");

int res = q1.executeUpdate();

System.out.println("no. of records effected : " + res);

tx.commit();

—

⇒ we can execute DDL Queries as NativeSQL Queries,

Transaction tx = ses.beginTransaction();

String sq1 = "Drop table XYZ";

SQLQuery q1 = ses.createSQLQuery(sq1);

q1.executeUpdate();

System.out.println("table dropped");

tx.commit();

→ Native SQL Query, that is placed in HB Mapping file, having logical name is called Named NativeSQL Query, these Queries are required.

① To make Native SQLQuery 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 putting 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 the effect of developing DB independent persistence logic then use properties file support having SQL Query

⇒ NamedNativeSQL Queries will be written in ~~HB~~ HB mapping file but instructions of executing those queries will be given in HB Persistence logic of client application.

ex:- on NamedNativeSQLQuery (EntityQuery)

step 01:- propose NamedQuery in HB mapping file as shown below.

<hibernate-mapping>

<class Name="pi-EmpBean" table="Employee">

```

<property name="mail" column="EMAIL"/>
</class>
<sql-query name="my-test">
<return class="P1>EmpBean"/>
    select * from Employee where eid=?  

    and eid &lt;=:P1
    → logical Name of NamedQuery
    → Mapping EntityQuery result  

    with HIB P00D class
    → XML Entity giving
    <'symbol' = =

```

< ⇒ '' symbol if we give like this in XML
 it treats subtag in so keep "<"

Step@1:- write following code in Client application
 to execute NamedSQLQuery.

//represents named sql query
 Query q1 = ses.getNamedQuery("my-test");

//setting parameter values

q1.setInt(0, 100);

q1.setInt(1, 500);

//executes named native sql query

List l = q.list();

//display records

for(int i=0; i < l.size(); ++i)

6

EmpBean ei = (EmpBean) l.get(i);

S.o.p(ei.getNo());

=> NativeSQL Queries are written directly in client apps along with HIB persistence logic they become specific to one HIB session object using which ~~one~~ SQL Query object is created.

By Making Native SQL Query as Named Queries we make them visible to multiple session object in client application. (In ~~one~~ single client app)

```
// preparing named sql query  
Named  
Query q = ses.getQuery ("my-test");  
  
=====  
  
ses.close();  
Session ses1 = factory.openSession();  
  
=====  
Query q2 = ses1.getNamedQuery ("my-test");  
=====
```

Example on executing Scala - Named NativeSQL Query :-

Step 01:- Prepare NativeSQL query in mapping file.

```
<sql-query name="my-test">  
    <return class="PL-EmpBean"/>  
    select * from Employee where eid=? and  
    <?>=?  
</sql-query>  
  
<sql-query name="my-test1">  
    <return-scalar column="FIRSTNAME" type="String"/>  
    <return-scalar column="EMAIL" type="String"/>
```

up

select FIRSTNAME, EMAIL FROM employee where
email like ?

<sql-query>

<hibernate-mapping>

Step@:- write coding in client app to execute NamedSQL

~~NativeSQLQuery~~

Query q1 = ses.getNamedQuery("my-test1");

// setting parameter values

q1.setString(0, "y@gmail.com");

Execute named native sql query

List l = q1.list();

for (int i=0 ; i < l.size() ; ++i)

{

Object oow[] = (Object[]) l.get(i);

for (int k=0 ; k < oow.length ; ++k)

{

System.out.println(oow[k].toString());

} // inner for

System.out.println();

} // outer for

~

Note :- It is always recommended to write NativeSQL
queries as Named Queries.

NativeSQL Queries, Named NativeSQL Queries also support
pagination, use same methods of JDBC programming

=> we can keep namedNativeQuery 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 appn.

Transaction tx = ses.beginTransaction();

Query q1 = ses.getNamedQuery("my-test2");

int res = q1.executeUpdate();

s.o.p("no of effected records "+res);

tx.commit();

ses.close();

8

=

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 appn that deals with Named HQL Queries :-

① prepare NamedHQL Query in hbmapping file.

```
<query name="my-test2">
    delete from EmpBean as eb where eb.no = ?
```

Step①:- execute Named HQL Query in the client application.

To transaction tx = ses.beginTransaction();

Query q1 = ses.getNamedQuery("my-test2");

//setting parameter value

q1.setString("p1", 67);

int res = q1.executeUpdate();

System.out.println("no of records that are effected " + res);

tx.commit();

}

-

⇒ example on Select NamedHQLQuery

Step①:- Place HQLQuery in mapping file.

<query name="my-test3">

select eb from EmpBean as eb where eb.mail like ?
~~EMPLOYEE~~

</query>

Step②:- execute this query from client application

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);

System.out.println(eb.getNo() + " " + ...);

}

=

- In order to centralise certain persistence logic or Business logic to multiple modules of a project or for multiple projects of a company then place that logic in DB SW as PLSQL procedure (or) function
- HB persistence logic can call PLSQL procedure (or) function at DB SW from client application only when

→ These rules ab developing PL/SQL procedures or functions for HB will change based on the DB env we use.

→ A CURSOR in PL/SQL programming ab Oracle is a Datatype whose variable can store zero or more selected records from DB table.

→ common rules to prepare PL/SQL procedures or functions for this environment.

① pagination is not possible, so the results generated by queries ab HB specific PL/SQL procedure.

i.e. we can't use `setFirstResult()`, `setMaxResults()` methods in this environment.

② we must follow the following syntax, to call ~~procedure~~ PL/SQL procedures.

{ call procedure name (<parameters>) }

③ we must follow, following syntax to call PL/SQL function.

{ ? = call function name (<parameters>) }

Note:- PL/SQL procedure does not return 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 ab Oracle. having the ability to store selected records [one or more] like result set object ab JDBC programming.

→ For rules specific to sybase, Micsooft SQL DB etc
Chapter 16 Q6 PDF file.

NOTE! - HB applications use NativeSQL programming to call PL/SQL procedures or functions.

⇒ Example Appvn to call PL/SQL procedure of Oracle from HB persistence logic :-

Step①:- prepare PL/SQL procedure in oracle, in the angle it is required for HB.

[Refer above oracle rules]

procedure clauses
datatype in oracle

```
> Create or replace procedure get_empldetails(mycur out sys_refcursor,  
      ename in employee)  
begin  
  open mycur for  
    select * from Employee where first_name like ename;  
end;
```

→ * the nativeSQL Query generated records will be stored over

Step②:- execute the above PL/SQL procedure in oracle DB svr

* /

=

Step③:- call the above PL/SQL procedure from HB mapping by
as NamedNativeSQL Query. in employee.hbm.xml

```
<sql-query name = "my-test" callable = "true">  
  <setvar class = "EmpBean" /> ↳ mandatory  
HB POJO class  
  &call get_empldetails(?, ?);  
</sql-query>
```

* → this '?' marks represents our parameters of the above
PL/SQL procedure (cursor) it must always be '?' symbol
you can't place named parameter here.

~~(%P%)~~ - %P, represents 'in' parameter as above

PL/SQL procedure, it can be ~~be~~ either named or positional parameters.

when it is taken as positional parameter, its index becomes zero.

Step 1: - write following code in client application to ^{call &} execute PL/SQL procedure. (code in client appn)

Query q = ses.getNamedQuery("my-test");

1) setting parameter values

q.setString("P1", "xyz");

List l = q.list(); // execute ~~the~~ PL/SQL procedure.

for(int i=0; i<l.size(); ++i){

}

=

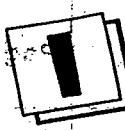
{

⑤ Develop remaining resources of the application like ~~etc~~

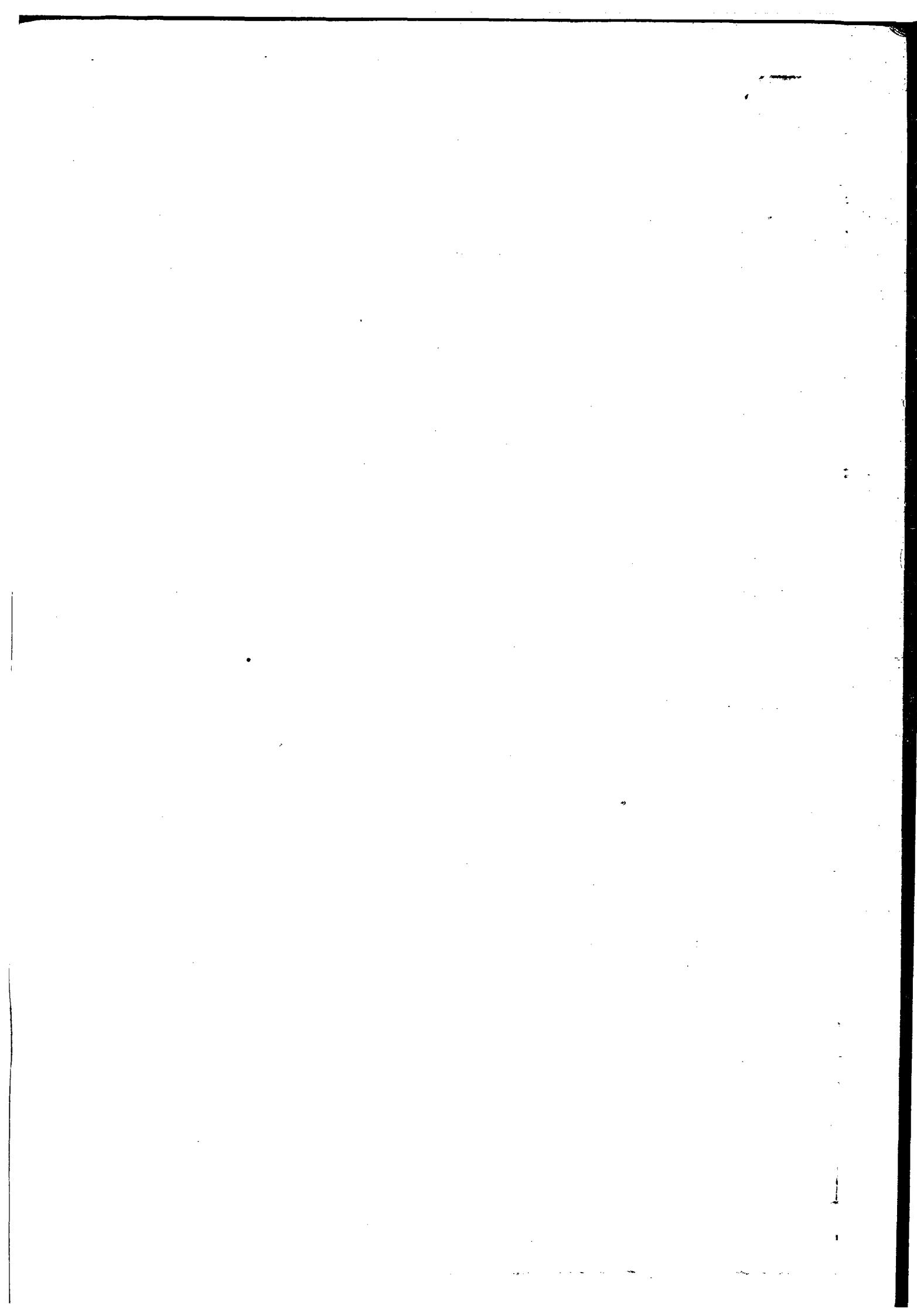
1st Application

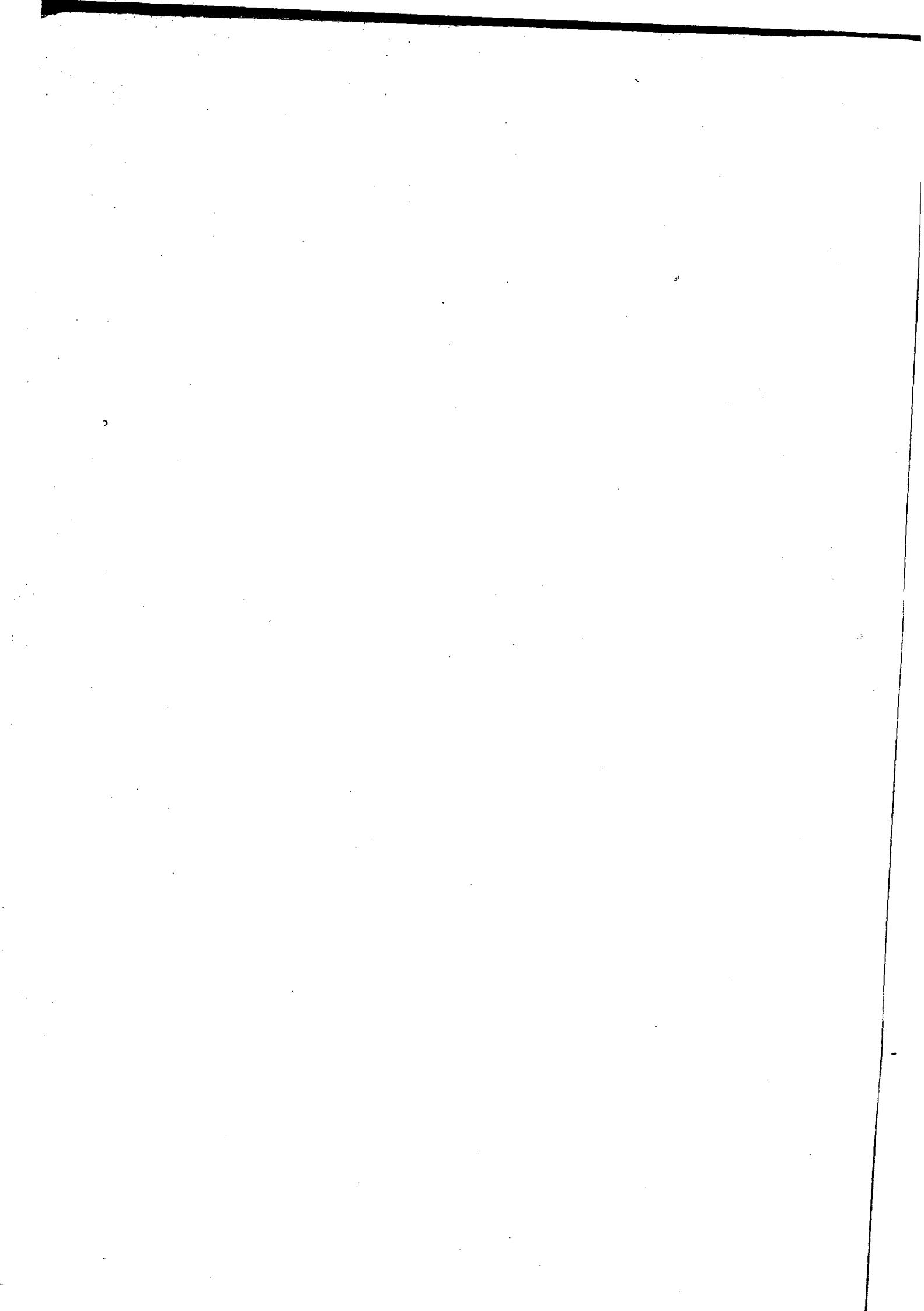
=

“શ્રી બાંધનેયમ्”



NAME : D Babu STD. : _____ SEC. : _____ ROLL NO. : 9966072198





21810

* procedure to call PLSQL function at Oracle DB via
by using hibernate persistence logic :-

Step 1 :- prepare PLSQL function in oracle DB w/o returning Result Set (cursor)

> create or replace function emp_function(no in number)
returns sys_refcursor

as

~~REFCURSOR~~ my_cursor SYS-REFCURSOR;

BEGIN

open my_cusor for

```
Select * from Employee where eid >= no9
```

between my_cousin;

END 3

Select * from user_procedures

Step(2) :- execute this PL/SQL function in Oracle DB SQL.

Step 5:- call the above PLSQL function from HB mapping file by using standard syntax.

In Employee :: bkm::xam1:-

三 24

```
<sql-query name="my_test" callable="true">
```

```
        class= "Employee"/>
```

L → mandatory

Can't be taken as
Named parameter

$f(?) = \text{call emp_function}(:PL)$

</SQLquery>

</hibernate-mapping>

Step 9: - write following persistence logic in client application to call and execute that PL/SQL procedure.

```
Query q1 = ses.getNamedQuery("my_query");
```

// setting parameters value

```
q1.setInteger("p1", 350);
```

List l = q1.list(); // execute PL/SQL function in DB & w.

```
for (int i = 0; i < l.size(); i++) {
```

```
    EmpBean eb = (EmpBean) l.get(i);
```

=

8

=

* We can't perform pagination / paging on the results generated by PL/SQL procedure or function called from HB persistence logic. (limitation of HB)

* Limitations of HB s/w -

- ① Since HB is not a distributed technology, the HB persistence logic is not distributed persistence logic. So this persistence logic cannot be used from remote client.
- ② Calling PL/SQL procedure (or) function from HB persistence logic is quite complex moreover these PL/SQL function or procedure must be written in the angle they are required (in 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/w's like Text file, MS Excel and etc (JDBC can do this)

⑤ HB can't interact with new conventional DB like MS Access

⑥ HB persistence logic use, negligible performance degradation compare to JDBC.

* * *
⇒ Calling PL/SQL procedure (or) function for HB persistence 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 persistence logic.

Example app:-

step①:- develop and execute normal PL/SQL function in oracle.

Y Create or replace function my_function(x in number)
returns 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.getConnection(); // gives JDBC obj

CallableStatement cs = con.prepareCall("{? =call my_function(?)}");

cs.setInt(2, 10); // this JDBC so starts from 1

cs.registerOutParameter(1, Types.INTEGER);

we are to execute C) 3. If execute PL/SQL function or DB SQL

S.O.P C "result is" + cs.getint(i);

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 as Oracle performing Non-select operation on the table:-

Step 01:- prepare & execute procedure 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 by the Non-select Query execution done in PL/SQL programming.

→ To call above PL/SQL function in ejb or applet as shown below.

Non-ORM mapping

```
Transaction tx = ses.beginTransaction();
```

// JDBC logic

Connection con = ses.getConnection(); // gives JDBC con obj

CallableStatement cs = con.prepareCall("{?=>call my_functions(?)");

```
cs.setInt(2, 567);
```

```
cs.registerOutParameter(1, Types.INTEGER);
```

cs.execute(); // executes PL/SQL function DB side

s.get("no. of records effected is " + cs.getInt(1));

```
tx.commit();
```

Note:- HQL Queries based PL/SQL programming is not possible

23/08/10

~~CRITERIA API~~

→ This API is given to develop HIB persistence logic plug-in java statements and with out using SQL, HQL Queries

→ Criteria API means working with classes & interfaces at org.hibernate.criterion package classes interface to develop P.L.

→ we can use Criteria API Only select Operations based persistence logic development.

→ we ~~can~~ 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 persistence logic POJO classes, ~~Member~~ memberVariable names will be utilised

→ Criteria API based persistence logic is DataBase independent persistence logic.

82 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);

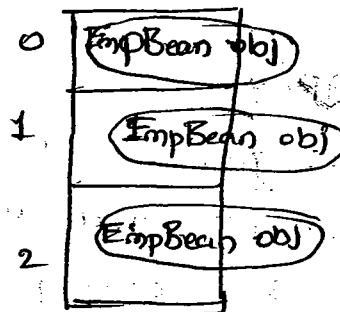
S.O.P ("orderNo:" + e1.getOrderNo() + " " + e1.getFname() + " " +);

}

8

Memory diagram :-

(java.util.List obj)



→ 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));

~~test also~~

ct.add(cond1);

List<Employee> list = session.createQuery("select * from employee
where eid >= 100").list();

for (int i = 0; i < list.size(); i++) {

System.out.println(list.get(i));

}

System.out.println(".....");

=

All methods of Restriction, Expression class are given

to frame condition while developing Criteria API based

persistence logic.

All these methods directly (or) indirectly return

Criteria object, these two classes are available

org.hibernate.Criteria package, Expression is the

sub class of Restrictions class.

Q) what is the diff b/w Criteria Object and
Criteria object?

a) Criteria object represent the total single
persistence ~~to~~ operation related persistence logic

in Criteria API. the conditions in this persist
logic will be represented by Criteria obj

→ we add Criteria 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.

Criteria object means, it is the object of class
that implements org.hibernate.Criteria.Criteria interface

⇒ Criteria object is mutable object, i.e.,

After creating Criteria object, if you add new data or if ~~add~~ 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~~ → 'and' class by default.

Criteria

Example with multiple conditions - (greaterthan equal / like)

Criteria ct = ses.createCriteria(EmpBean.class);

Criteria cond1 = Restrictions.ge("no", new Integer(100));

Criteria cond2 = Restrictions.like("mail", "x.gmail.com");

ct.add(cond1);

ct.add(cond2);

List l = ct.list(); // gives select * from employee

whose eid >= 100

and mail \sim x.gmail.com → like

using ~~or~~ in ~~as~~ conditions - (in, or)

Criteria ct = ses.createCriteria(EmpBean.class);

Criteria cond1 = Restrictions.lt("no", new Integer(100));

Criteria cond2 = Restrictions.in("fname", new String[]

{ "arya", "davi" });

~~Criteria~~

Criteria orcond = Restrictions.or(cond1, cond2);

ct.add (criteria);

List l = ct.list(); // select * from employee where
cid < 100 or fname in ("ooja", "oav")

foo(- -)

{

}

-

⇒ placing multiple conditions with OR clauses:

Criteria ct = ses.createCriteria (EmpBean.class);

Criteria cond1 = Restrictions.lt ("oo", new Integer(100));

Criteria cond2 = Restrictions.in ("fname", new String[]{"ooja", "oav"});

Criteria oocond1 = Restrictions.or (cond1, cond2);

Criteria cond3 = Restrictions.ne ("mail", new String("ooja@gmail.com"));

Criteria oocond2 = Restrictions.or (oocond1, cond3);

ct.add (oocond2);

List l = ct.list();

foo(- -)

-

{}

// select * from Employee where (cid < 100 or fname
("ooja", "oav")) or email > 'anil@guvi.com'

-

{},

Applying And, OR clauses based on condition in Query execution

→ Criteria ct = ses.createCriteria(EmpBean.class);

Criteria cond1 = Restrictions.lt("no", new Integer(100));

Criteria cond2 = Restrictions.in("fname", new String[]{"Sajal", "Saurav"});

Criteria andcond = Restrictions.and(cond1, cond2);

~~Criteria~~

Criteria cond3 = Restrictions.ne("email", new String("sajal@gmail.com"));

Criteria orcond = Restrictions.or(andcond, cond3);

ct.add(orcond);

List l = ct.list();

for
=

⇒ we can pass conditions in criteria API based

on persistence logic in the form of SQL Query statements by using expression = sql()

Criteria ct = ses.createCriteria(EmpBean.class);

Criteria cond1 = Expression.sql("email like '%.gmail.com'");

Criteria cond2 = Expression.sql("firstname in ('Sajal', 'Saurav')");

ct.add(cond1);

ct.add(cond2);

List l = ct.list();

for(-)
=

=

HQL is the most recommended technique to develop persistence logic in HB environment.

24/8/10

- HQL, NativeSQL, Criteria API based programming supports paging operation but the NativeSQL programming that deals with PLSQL procedures and functions does not support paging operation.

Filters:-

- HB Filters are hibernate 3.x feature that allows the programmer to specify predefined criteria condition for the persistence logic of hibernate programming.

~~24/8/10~~

- Filters represent where clause related condition in HB Mapping file and they are visible for multiple session objects of hibernate application.

~~The~~ 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 specific 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 persistence logic?

- ① Define filter in HB mapping file specifying its param names and types.

in

Step 1: employee.hbm.xml file

<hibernate-mapping>

<class name=

=

</class> ↳ logical name of filter.

<filter-def name="myfilter">

Parameters

Names & types

{ <filter-param name="myid1" type="int"/>

<filter-param name="myid2" type="int"/>

</filter-def>

<hibernate-mapping>

Step 2:-

link filter with specific hibernate POJO class and write condition.

<hibernate-mapping>

<class name="pl.EmpBeans" table="Employee">

<id name="no" column="eid" />

<property

=

the filter that is taken above.

<filter name="myfilter" condition=""

. condition = "eid >= ?myid1 and eid <=?

?myid2" />

</class>

<filter-def name="myfilters">

=

</filter-def>

</n-m>

=

Condition repository

key filter

Step 5 enable filter on hib session object and execute

HIB persistence logic.

Code in client application :-

~~Denote~~

ses = sf.openSession();

// enable filter on hibernate ses obj

oog.hibernate.Filter f1 = ses.enableFilter("myfilter");

// set parameter values

f1.setParameter("myid1", new Integer(100));

f1.setParameter("myid2", new Integer(900));

// 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.getId() + " " + e.getName());

}

// disabling filter on ses object

ses.disableFilter("myfilter");

Query q2 = ses.createQuery("from EmpBean");

List l1 = q2.list(); // execute HQL query with out condition

for (int i = 0; i < l1.size(); i++)

{

EmpBean e = (EmpBean) l1.get(i);

s.o.p("...");

}

=

-Hibernate Filters makes ~~it~~ conditions ab HB Filter logic as flexible condition by making them coming to app from outside the persistence logic comes from HB mapping file)

step 3:- Develop remaining resources ab the application like boost application.

*** For complete source code above application refer page appln (3) given in page no : 82-85

⇒ we can enable or disable filters for ~~the~~ ~~the~~ types ab HB persistence logic environment ~~HOL~~, colored API

→ it is not applicable for "NativeSQL"
- (we can't enable filters on NativeSQL programming related persistence 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 ab the project, this specification contains rules and guidelines to develop the project.
every project specification contains its own project API designed by PL, this API represents methods declared in the interfaces, abstract classes as rules and represents concrete method definitions ab abstract classes and concrete classes as guidelines.

while implementing project based on project specific rules will be implemented. If interface methods and guidelines will be followed (concrete method declarations).

- 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 ~~method~~ abstract methods representing rules and defines concrete methods representing guidelines.

→ In coding level of project development programmers 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 appn.
- Business Interface in RMI appn, EJB component.
- ⇒ =
- ⇒ SLW specification contains, rules and guidelines to develop new softwares, every SLW specification contains built-in API, the interfaces of this API having method representing rules of SLW specification having method declaration. the concrete methods of this API, available in concrete, abstract classes representing guidelines of SLW specification.
- Ex → JDBC specification contains rules and guideline to develop JDBC drivers.

② Servlets JSP specifications to make you develop sever
containers / web containers and you can do this

25-8-10

INHERITANCE MAPPING :-

Hibernate persistence classes are POJO class, so they can participate in inheritance to give the advantage of reusability and extensibility.

When HIB persistence classes are there in inheritance, i.e. DB table's Related to these persistence classes we also try to maintain data having relationship.

To make HIB understand this inheritance

With POJO classes, we need to configure them in HIB mapping file based inheritance mapping concepts.

→ Inheritance b/w two Servlet components is possible but inheritance b/w two EJB components is not possible.

→ HIB 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 shows use ③ approaches of mapping inheritance
 mapping to get Reusability, extensibility advantage
 of inheritance in different ways.

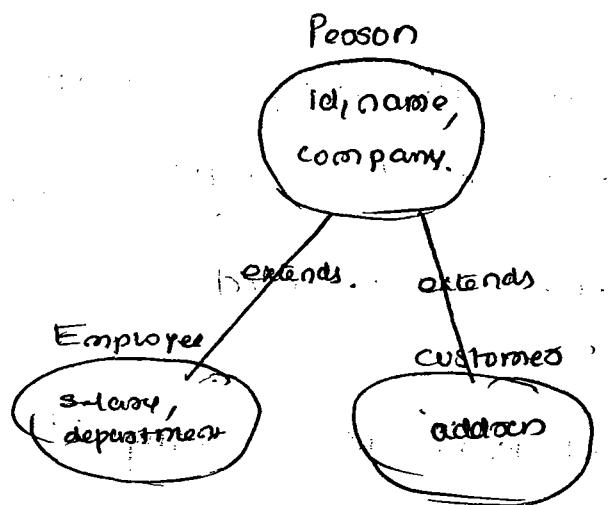
they are ① Table per class hierarchy

② Table per subclass

③ Table per concrete class.

① Table per class hierarchy :- (Pg No)

All the classes at inheritance, hierarchies use
 (as different)
 single DB table by having discrimination column.



all these classes
 inheritance based
 will use same DB +

discrimination
 column ↑

in-people						
Cpk	id	name	company	salary	department	address
101	ayyer		HCL	-	-	-
104	navi		TCS	10,000	1001	-
108	ram		Wipro	15000	-	Hyd
210	rakesh		HCL	-	-	Sec

Tables \rightarrow given in page no :- ⑥

Three-level model at higher level mapping

for example application on two PBO

be done in this mapping file.

Attribute PBO column but its configuration will

contain only people to design

in this approach distinction between the two

this subclauses is subclauses for

sub clause.

and <subclause> will be used to congregate

<clause> will be used to congregate super clause

In future per column hierarchy inheritance mapping

future contains other PBO-clauses related to codes

records separately for DB table even though DB

very useful to select specific PBO clauses related

the logic can do various of this discrimination values are

area => person-type column of above DB table

clause of higher nature hierarchy.

records to the table are inserted using which PBO

this column maintains logical values separating

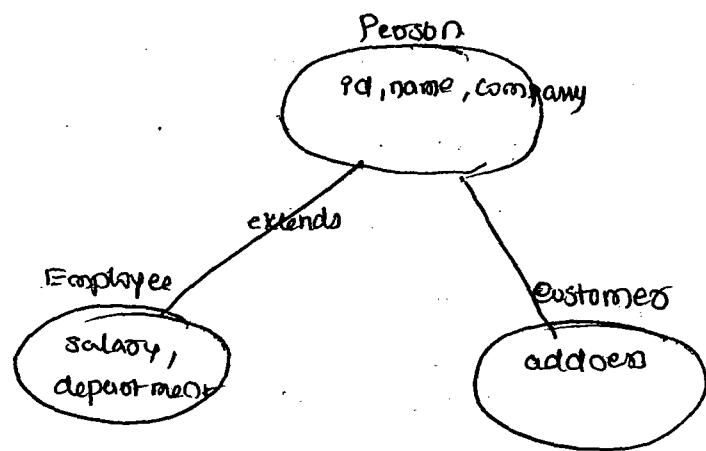
various DB table showing classification columns because

in table per column directly, this is recommended

→ While working with table per class hierarchy inheritance model the columns related to child POJO class properties in DB table can't be applied with not null constraint and disconnection column can't be applied with unique key or primary key constraint.

26/8/10
63 pages 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 its own database table. The table ~~takes~~ of child POJO classes will have 1/s with one-to-one parent POJO class table.

<u>InPerson</u> ↗ (parent table)		
id (pk) (N)	name (V2)	company (V2)
101	XXX	HCL
102*	YYX	POLARIS
103**	ZZZ	WIPRO

→ V2>Varchar 2 (20)
N → ~~Small~~ Number

<u>parent employee</u>		<u>child table</u>
<u>salary</u> (N)	<u>department</u> (N)	<u>person-id</u> (PK with ref. to (N) <u>in-person</u>)
6000	1001	121 *

N - ~~Number~~

in - customers 2 child table

<u>address</u> (N)	<u>person-id</u> (N)	PK with (ref) col ab in person
Hyd	131 *#*	=

→ In this model of inheritance mapping all column's Qb all DB table's can be applied with not null constraint and there is no necessity ab discriminator column.

One-one relationship b/w two tables is nothing but one record ab parent table with always dependent , one record ab child table.

→ In-table per subclass model ab 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 ab ~~<class>~~,

→ In this model @ inheritance, child pojo class object data will be splitted and will be inserted parent, child tables having relationships.

This relationship comes because @ Foreign Column @ 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 @ properties.
- ② It allows to design table having relationship satisfying DataBase designing principles.
- ③ Allows to apply Not Null constraint on all the columns @ all the tables

For Example application on Table per Subclass inheritance mapping see its application (7) in page 59-63

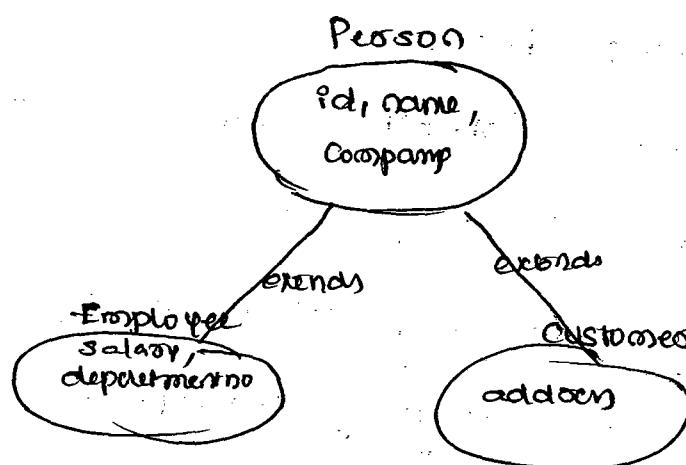
→ While working with multiple table, advanced ~~to~~ OR mapping concepts, it is recommended not to use reverse engineering @ MyEclipse IDE.

use ~~@~~ MyEclipse IDE for these operations but develop resources manually

③ Table per concrete class inheritance

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 with own 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's 3

<u>id (n) (pk)</u>	<u>name (vc2)</u>	<u>company (vc2)</u>
101	soja	xyz

In - Employees

<u>id (n) (pk)</u>	<u>name (vc2)</u>	<u>company (vc2)</u>	<u>salary (n)</u>	<u>deptno (n)</u>
102	soori	abc	8,000	1001

In - Customers 3

<u>id (n) (pk)</u>	<u>name (vc2)</u>	<u>company (vc2)</u>	<u>address (vc2)</u>
103	samish	mno	hyd

⇒ The above table designing is very poor designing because multiple tables in DBS are having same col so Table per concrete class model \Rightarrow Inheritance mapping is not industry standards.

⇒ In this model all columns in all tables can be applied with 'Not-Null' constraint.

→ For Example application on Table per Concrete Class APP
Page no 83-68

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 in a DB table is called component property.

→ A component property is a contained object in hibernate pojo class object representing specific multiple column values in a record.

→ component property of hibernate pojo class represent partial data in record whereas its hibernate pojo class object represents total record data including this partial data.

→ public class Person

{

int pid;

String pname;

JobType pjob; // component property

= getxxx(c)

} setxxx(c2)

→ prepared code for class JobType

String job;
double salary;
int department;
= getXXX()
setXXX()

Person-tab (DB table) :-

pid (n)	pname (vc2)	job (vc2)	salary (n)	department (n)
101	soja	SE	18000	1001

pid (property) → pid col

pname (" ") → pname col

pjob (" ") → job, salary, department cols

→ To configure "simple property".

in HIB POJO class use

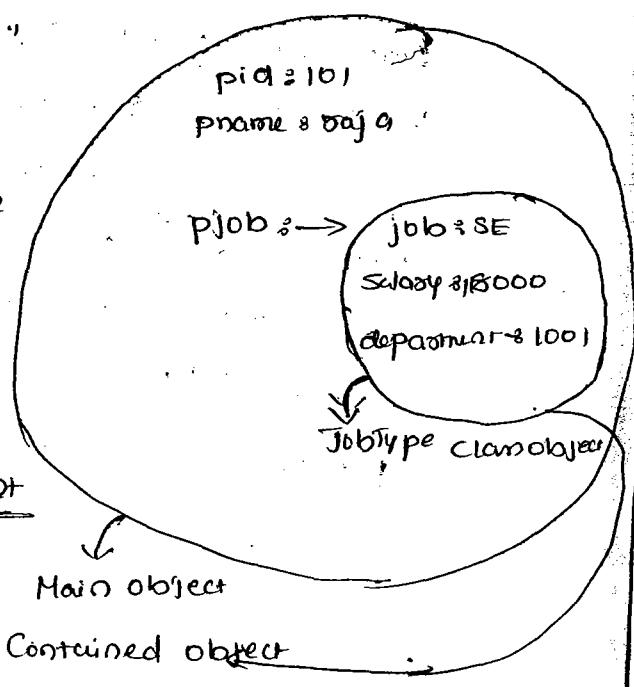
<property> tag in mapping file

similarly <component> tag
in mapping file.

→ For example application on component

mapping defined appn :- 5 ab

material in the pages :- 51-54



→ If you are looking to maintain parallel values ~~else see~~
in the form of separate java class object take component
property. In this class representing selected multiple
values in a table.

Q) How to find out no. of objects created by java class

Ans:- Take static member variable in java class and
increment that member variable in constructor.
So this static member variable keeps track of the
number of objects created by that class.

Ex:-

TestApp.java

Class Test

{

 public static ~~void main~~ int counter;

 Test()

{

 System.out.println("Zero obj constructed & Test class");

 counter++;

}

}

public class TestApp

{

 public void main(String args[])

{

 Test t1 = new Test();

 Test t2 = new Test();

 Test t3 = new Test();

 System.out.println("No. of objects are created = " + Test.counter);

}

}

=

28-08-10

Association Mapping :-

- * DB designing team designs the tables of project according to "Normalization Rules".
These are total six rules (or) norms. The Second normalization rule says design tables having integrity constraints that means tables should be designed having o/s like one-to-one, one-to-many, many-to-many and etc.

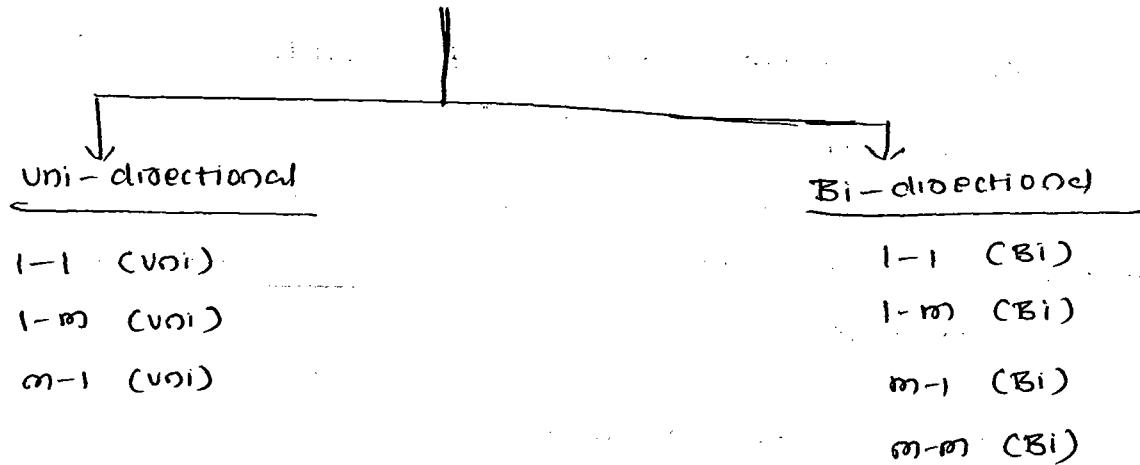
when tables are there in Association (o/s) RLS we can access one table data ~~over~~ based on another table data because records of one table represents the records of another table.

- 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 ~~not~~ 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 one user can have multiple phone numbers.
- The o/s b/w emp, dept tables is many to one because many employees can belong to one department.
- The o/s 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 EJS except many-to-many two tables are enough (parent, child) but to define many-many another three tables are required (table1, table2, relationship table)
- When two database tables are related in relationship their "HB POJO classes" must be designed and configured supporting that relationship. This work is called "Association mapping" in HB mapping file.
- When HB POJO classes are designed supporting EJS then the objects of these classes actually participates in object level relationship (or) association.

HB object level relationships



- By using parent POJO class object data we are able to access the associated child POJO class "object/objects" data and if reverse is not possible then it is called "Unidirectional" association. If reverse is also possible then it is called "bidirectional" association.

Ex:- If student POJO class object (parent) pointing one book POJO class object (child) having one-to-one association if programmer is able to get associated book object data through student object data and reverse is not possible then it is called "unidirectional" association.

If reverse is also possible then it is called

"bidirectional association".

→ The HIB programmers should always design HIB POJO classes based on the E-R diagrams given by db team.

→ ER Diagrams (Entity - Relationship diagrams)

To perform annotations 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 at these tags allows to enable bi-directional association / relationship.

inverse = "false" allows to enable unidirectional association.

⇒ Example HIB POJO classes for unidirectional one-to-one association :-

public class Student

b

int sno;

String name, address;

Rank r;

→ represents one object at Rank.

— —

POJO class

— — & no setXXX(), getXXX() methods.

}

public class Rank

b

int sno;

String subject;

String exam;

* Constructors & Getters

→ Example HIB 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

→ getters & setters.

class

}

public class Rank

{

int sno;

int sno;

String subject;

String exam;

Student st; → Represents one object of Student POJO class

→ getters & setters

}

→ Example HIB POJO classes for unidirectional one-to-many association :-

public class User

{

int uid; → represents foreign key column of child

String username;

List phones;

(0 or more) Set phones;

allows duplicates

represents one (or more) child User phone number class POJO objects

getxxx & setxxx

}

public class PhoneNumbers {

{

long ph;

String type;

int uid;

—

—

—

}

==> Example of two POJO classes for Bi-Directional one-to-many association :-

public class User

{

int uid;

String username;

Set phones;

==

g

public class PhoneNumbers

{

long ph;

String type;

int uid;

User u;

—

—

g

==

→ Example HIB POJO classes for UNIDIRECTIONAL MANY-TO-MANY

public class Emp

{

int empno;

String ename;

float sal;

int deptno; // represents Foreign key column at parent table
Department dept; → represents one object of Department
POJO class

setters & getters

}

public class Department

{

int deptno;

String name;

String loc;

setters & getters

}

- * In the above discussion "dep" properties belong to multiple Emp class objects respectively.
- (a) Same Department class object, to get the object is many-to-one association.

Example HIB POJO classes for Bi-directional many-to-one association e -

public class Emp

{

int empno;

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
annotation e -

public class Project

{

int projId;

String progname;

Set programmes;

↳ represents zero or more objects
of programmes 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 O/R 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 varchar(20));

Child table :-

create table PHONE_NUMBERS (NUMBER_TYPE varchar(20), phone number(10),
UNID number references USER_TABLE (USER-ID))

Foreign key column allowing duplicates

→ values to this foreign key UNID should not be inserted

as updated directly because these values comes automatically based on the values inserted in PK column of parent table (User-ID).

Application Resources :-

→ hibernate.cfg.xml

→ User.java

→ PhoneNumber.java

} Hibernate POJO classes

→ User.hbm.xml

→ PhoneNumber.hbm.xml

} HBM mapping files

→ TestClient.java — Client app ①

→ SelectTest.java — Client app ②

Hibernate.cfg.xml :-

Note :- configuration details for Oracle DB SW 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 useid;

private String firstName;

private Set phones; → To hold 2000 (or) more
objects at phone number - POJO class

//getters & setters

=

public Set getPhones()

{

return phones;

}

public void setPhones(Set phones)

{

this.phones = phones;

}

=

Phone Number Java :- (child POJO class)

```

public class PhoneNumbers {
    private String numberType;
    private long phone;
    private long id; → represents foreign key column
    // setters & 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 is child table.
        </set>
    </class>
</hibernate-mapping>

```

Note:- If this POJO class property type is collection like data structures like List, Set, Map and etc of java.util package. Then we need to configure those properties in mapping file through "Collection Mapping". For this we use <set> tag, <map> tag, <list> tag, <bag> tag and etc tags.

Note@C— since phones property should hold hand deprecant 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 of any tag (or) ~~at~~ <key> tag always points to foreignkey column name indicating that this is column that responsible for R/B bw 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 is "id" property is the FK column UNID and to disable direct updates and insertion in that column as insert operation `insert = "false"`, `update = "false"` values kept. Values to FK column should come dynamically based on the values inserted in the referenced PK column other tables.

TestClient.java :

```
import org.hibernate.cfg.*;
import java.util.*;
public class TestClient
{
    public static void main(String args[])
    {
        try
        {
            Session ses = null;
            Transaction tx = ses.beginTransaction();
            User u1 = new User();
            u1.setUserId(101);
            u1.setFirstName("Oaja");
            PhoneNumbers ph1 = new PhoneNumbers();
            ph1.setPhoneType("residence");
            ph1.setPhone(65538968);
            PhoneNumbers ph2 = new PhoneNumbers();
            ph2.setPhoneType("office");
            ph2.setPhone(65538958);
            Set s = new HashSet();
            s.add(ph1);
            s.add(ph2);
        }
    }
}
```

//Setting phoneNumber class object to phones property

to user obj

ui.setPhones(s)

170 insert record

ses.save (ui)

tx.commit()

ses.close()

finally

catch (Exception e)

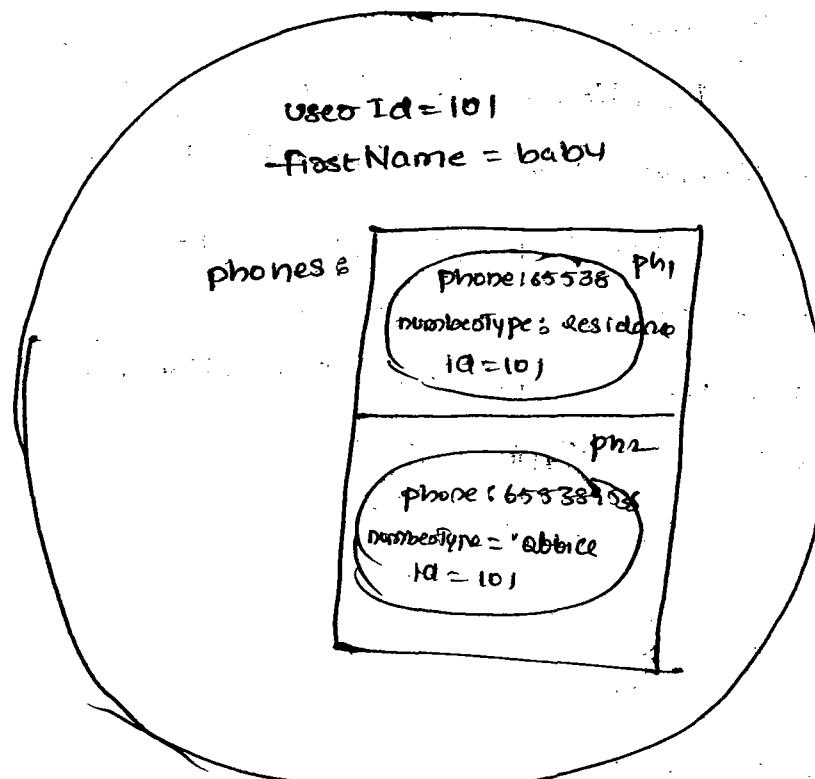
e

g

g

Memory Diagrams -

User obj(ui)



→ When the above application executed one record will be inserted in USER_TABLE, and two records will be inserted in PHONE_NUMBERS table having one-to-many association and values in foreign key column UNID will be set dynamically.

31/08/2010

SelectTest.java (gives parent table records and associated child table records) :-

public class SelectTest

{

 public static void 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);

 System.out.println("parent " + u1.getUserId() + " " + u1.getFirstName());

 Set s = u1.getPhones();

 Iterator it = s.iterator();

 while (it.hasNext())

 {

 PhoneNumber ph = (PhoneNumber) it.next();

 System.out.println("child " + ph.getPhoneNumber() + " " +

 ph.getNumberType() + " " +

 ph.getId());

Query q2 = ses.createQuery("from phoneNumbers");

It gives many child records

l = q2.list();

for (int i=0; i<l.size(); i++)

{

PhoneNumber ph= (PhoneNumber) 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 reverse is not possible. To make reverse operation
possible go for bidirectional one to many association.

* DeleteTest.java :- (To delete records)

public class DeleteTest

{

PSV main(String args[])

{

try

{

Session ses= —;

Transaction txn= ses.beginTransaction();

Query q1 = ses.createQuery("from User")
will 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) remains
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
- PhoneNumber.hbm.xml
- TestUser.java
- SelectTest.java
- DeleteTest.java

hibernate.cfg.xml -

→ same as above application ←

User.java -

→ same as above application ←

PhoneNumber.java -

```

public class PhoneNumber
{
    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 app.

PhoneNumbers-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"/>
    <many-to-one name="parent" class="User"
                  column="UNID" cascade="all"/>
        Foreign key column is ↪
        Child table having ability to
        perform operations from this DB
        ↪/class>
</hibernate-mapping>

```

operations performed on child
POJO class will be reflected in
the unassociated parent POJO
class object is necessary

↑
parent
POJO class
name.

TestClient.java:-

debes TestClient.java source code en pagina 2-798-80

Select Test.java:-

01/09/10 : debes ir a page no 8-81 & 82

DeleteTest.java:-

public class DeleteTest

{

 public static void main (String args[]){} 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(v2);
```

```
PhoneNumber q3 = new PhoneNumber();
```

```
q3.setPhone(91919191);
```

```
q3.setParent(v2);
```

```
Set s1 = new HashSet();
```

```
s1.add(q1);
```

```
s1.add(q2);
```

```
s1.add(q3);
```

```
v2.setPhones(s1);
```

```
ses.delete(q1);
```

```
ses.delete(q2);
```

```
ses.delete(q3);
```

```
tx.commit();
```

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

git add

} class

→ For complete Source Code of above sources representing Bidirectional one-to-many annotations refer app12 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 slw that the current property also participate in reverse annotation 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 this lazy loading as extra lazy loading we can use "lazy" attribute in the above said tags by having

lazy = "false" lazy = "extra" values.

disabled lazy loading ↙ ↘ enables extra lazy loading
and goes for eager loading

→ When lazy = "true" (default value) :-

When queryList() is called in annotation-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 slow only when programmer starts using those objects in his application.

→ When lazy = "false" :-

When queryList() called in annotation-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) \Rightarrow What is the difference b/w HQL queries with batch keyword and without batch keyword?

Ans:- When HQL queries are executed without batch 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 batch 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.
→ Detailed example appn on HQL Joins ~~in~~ HQLJoinsClient.java application in pages no e-80 & 81

QUESTION

You can go for unidirectional and bidirectional one-to-one annotation in 2 ways.

- ① Based on primary key based one-to-one annotation
- ② Based on Foreign key based one-to-one annotation

- For unidirectional one-one association it is recommended to work with primary key based 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 also. 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 DB their POJO classes and POJO objects are participating in session based on identity value.

Student-table -

<u>sid (pk)</u>	<u>name</u>	<u>address</u>
101	soya	hyd
102	charles	hyd

lib-membership:-

<u>sid (pk)</u>	<u>join date</u>
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;

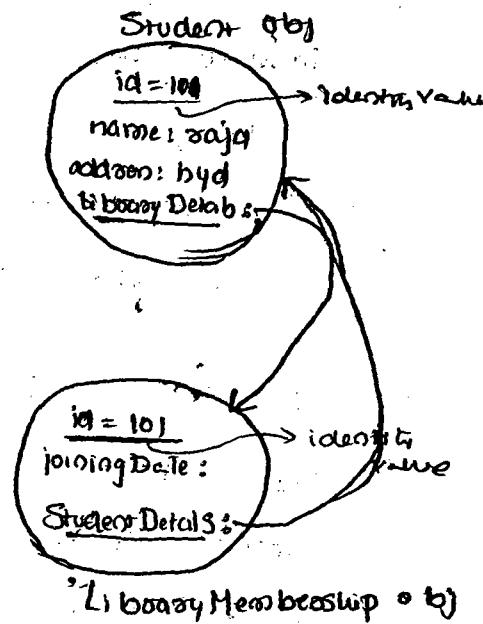
Student studentDetails;

getXXX(), setXXX();
=

8

→ 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 at this objects, objects are there in one-to-one as even though their detail tables are not having foreign key constrained column.

→ In one-to-one primary key association the Foreign key algorithm is utilized to generate identity value at child pojo object based on the identity value at parent pojo object. This foreign algorithm is identity value generator algorithm and no way related with fk constraint at DB side.



↳ Based on the above discussion the 4th member of library membership class should be categorized as library having borrowing aggregation as its primary mapping file having generation as its value generate.

↳ If there is a guarantee every pair of clear object go to will have one clear pair of clear object go to primary key based one-one association. If that guarantee is not there go to foreign key based one-to-one

↳ Every student of college will have one library member association. So we can go to primary key based one-to-one

↳ Every citizen of India need not to have license but every license belongs to one citizen. So we can go to foreign key based one-to-one association between citizen, license details.

↳ Foreign example app on pk based one-to-one application code given in page no 68-70 ↳ application code given in page no 68-70 ↳ supplementary clearing application for application-9 and doesed one-to-one association to select and

↳ value records.)

Select Test - Java

import org.hibernate.

II parent to child POJO ~~class allows to pass~~

Query q1 = session.createQuery ("from Student") ;

List l = q1.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.getLibraryMembership();

st.getLibraryDetails();

s.o.p ("child : " + lib.getId() + " " + lib.getJoiningDate());

}

- s.o.p ("-----");

II 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())`

parent to child

// for deletion of records (parent to child)

Transaction tx = _____;

LibraryMembership lb = new LibraryMembership();

lb.setID(1);

Student st = new Student();

st.setId(1);

st.setLibraryDetails(lb);

lb.setStudentDetails(st);

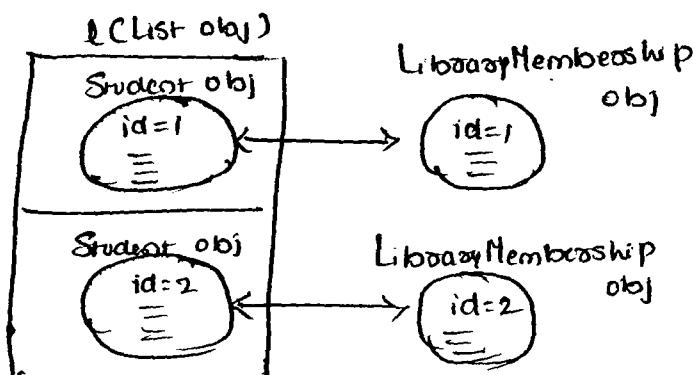
session.delete(st);

tx.commit();

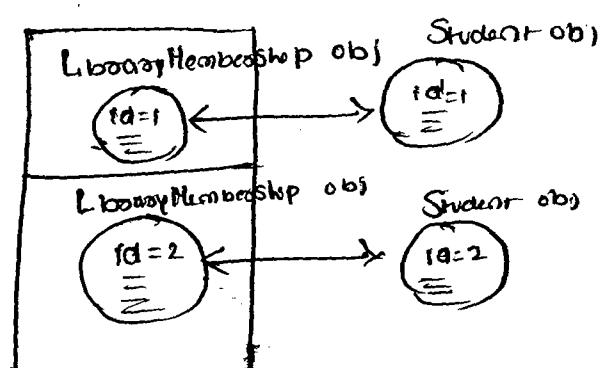
session.close();

}

parent to child



child to parent



Form en 6/9/10 Continen - from back-side

6/9/10

Q) what is the diff b/w Restrictions and Exclusions class while working with Criteria API?

Ans:- The methods of Restrictions class allows to pass conditions to Criteria API logic in the form of java statements, whereas the methods of Exclusions class allows to pass same condition as SQL Query statements.

Methods of Restrictions class are GE, LE, ID, GT and etc.

The methods of Exclusion class are sql (-)

One - One association (CFK) :-

In one-one(FK) association mapping, child table should contain Foreign key column, pointing to PK columns of parent table. This FK column should not 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 class 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 contains one-one o/s since there is no guarantee every citizen to have

er on license but there is a guarantee that every license points to one citizen, so go for
using of FK based one-one association mapping in this situation.

Example - For example application on FK based One-one
Unidirectional based association referred APP 16

Given in page no 8-70-73

=> If child POJO class object can take, this identity
value from the identity value of the associated
Parent POJO class object, then use one-one PK
annotation.

=> If child and Parent POJO class object are
looking take separate and independent identity
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 overcome this problem use many-to-one having
unique=true attribute.

To configure the property that is responsible
relationship see the line:- 1878 - 1878 of booklet.

Programmes - projects

programme_id (PK with project)	project_id (PK with period)	composite PK
101	1001	
101	1002	
102	1001	
103	1002	
102	1002	

- First table shows Parent data
 - Second table shows child data
 - Third table shows many-many associations b/w Parent table and child table data.
- ⇒ For example appln on many-many association mapping debbed app⑪ given in 73-77

Bag - the Bag DS, given by hibernate. It is same as List DS. i.e. Collection. So ~~not~~

Bag DS also allows ~~as~~ List DS 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 debbed App⑫

at page no :- 98-101

- The app ⑫ at booker, shows association b/w (one-to-many) speakers and his sessions, speakers and his phone numbers, that means one parent table is there having two child table.

- ⇒ If HIS POJO class property type is `java.util.List`
- (a) bag-hb mapping: Bag then we can combine that property in our HIS Mapping file entries by using `<List>` tag or `<bag>`.
- ⇒ DB table is having single column, then programme like the predetermined wrapped classes `Integer`, `Float` `String` as HIS POJO classes.
- especially this is useful, when associated table (child table) ab certain table is having only one col

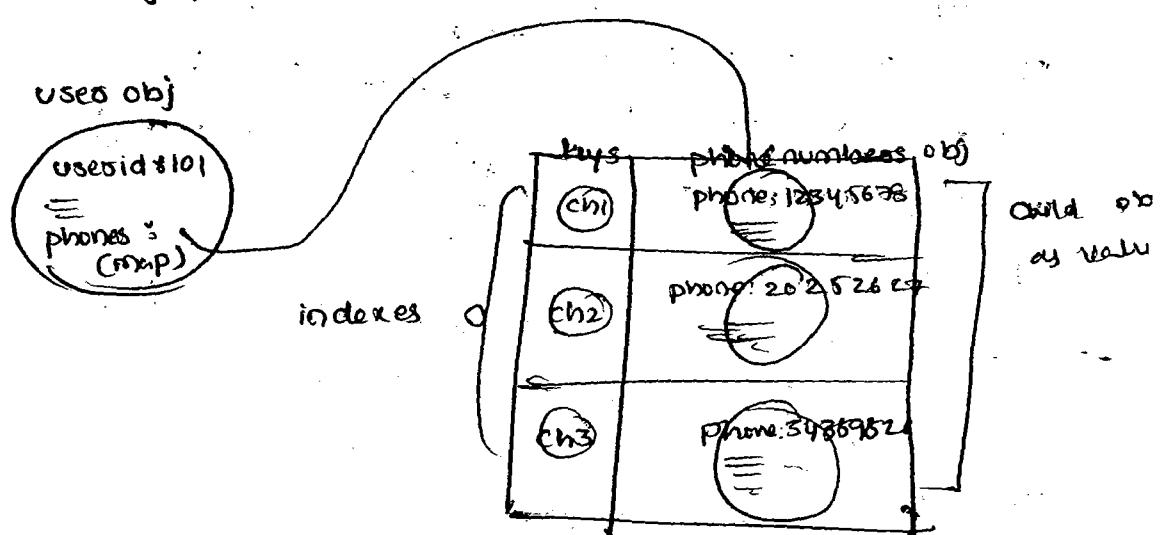
8/9/10

Map:

each element ab map DS, contains one key and one value. Key can be any object, and value can be any object, we can use key here to maintain index(0) count ab value objects.

In hibernate, we can use Map DS property ab parent poj to maintain one (0) many child pojo class objects to achieve one-many as.

in this program, we can provide index for the child pojo class objects that are linked with parent object.



→ If you want to link "child objects" with "parent objects" with out providing "id" of child object, we can use Set, List, Bag type property in Parent POJO class to link child POJ objects from child POJO class objects.

→ for example appin on ~~Map~~ Map + tag based collection mapping deleted supplementary handout :- 8/9/10

91910

Cache-Bubbed is a temporary memory, that holds result (o) date across the multiple and uses that date (o) result across the multiple same request given by application.

The process of storing result in Cache (o) bubbled and using that result across the multiple request or execution of the application is called "Caching".

(o) "Bubbling:

→ In client - severs application, if caching is enabled at client side, it reduces network round trips b/w client and servers applications.

→ every browsed window contains bubbles and stores, Request related response info, the bubbles ab browsed window Reduces N/W round trip b/w browsed window and webserver across the multiple same request given by browsed window.

→ The caching (o) bubbling done in Hib based application stores results collected from the DB s/w in the form of Hib pojo class objects, this Reduces N/W Round trip b/w Hib based client application and DB s/w across the multiple same request given by Hib based client appln.

→ HIB supports two levels of caching.

① 1st level caching \ L1 caching (HIB session object)

② 2nd level caching \ L2 caching (HIBSessionFactory 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.

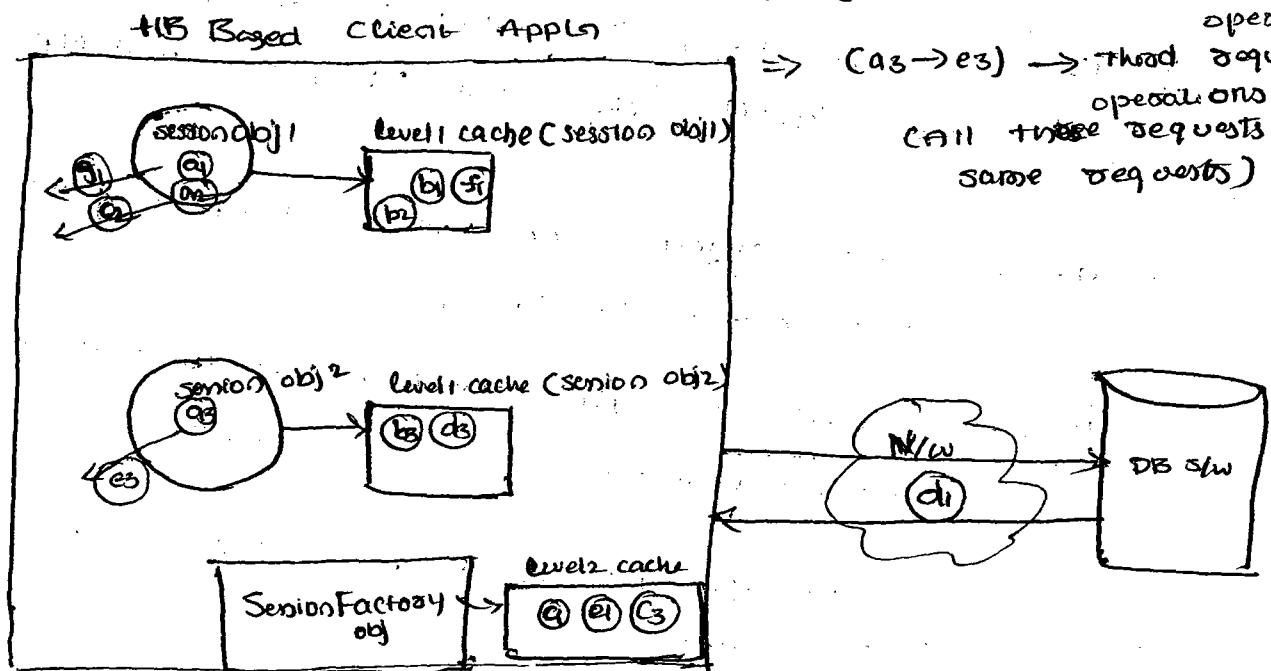
→ every session object contains one level1 cache storing its session object specific results.

→ Level II cache is called Global cache, bcz; we can store and manage the result generated by all HIB Session objects.

⇒ (a₁ → g₁) → first request operation

⇒ (a₂ → g₂) → second request operation

⇒ (a₃ → g₃) → third request operation
(all these requests are same requests)



- \Rightarrow session object ① gives initial request (client query)
- \Rightarrow b₁ \Rightarrow searches for that query request in level ① cache of session object ① (not available)
- \Rightarrow c₁ \Rightarrow searches for query result in level 2 cache (not available)
- \Rightarrow d₁ \Rightarrow HB application interacts with DB SW and gets query execution related result.
- \Rightarrow e₁ \Rightarrow HB SW registers the result in level ② cache
- \Rightarrow f₁ \Rightarrow HB SW also registers the result in level ① cache of session object ①
- \Rightarrow g₁ \Rightarrow session object ① gives the result to application
-
- \Rightarrow a₂ \Rightarrow session object ① gives same request once again
- \Rightarrow b₂ \Rightarrow searches for the result in level ① cache of session object ① (available)
- \Rightarrow c₂ \Rightarrow session object ① collects the results from its own level ① cache and passes to client appln.
-
- \Rightarrow a₃ \Rightarrow session object ② at same client appln gives same request to DB SW
- \Rightarrow b₃ \Rightarrow searches for the result in level ① cache of session object ② (not available).
- \Rightarrow c₃ \Rightarrow searches for the result in level ② cache (available)
- \Rightarrow d₃ \Rightarrow collects the result from level ② cache and registers with level ① cache of session obj ②.
- \Rightarrow e₃ \Rightarrow session obj ② gives the result to client appln

- ⇒ Results go to level①, level② cache will be stored in form of HIB POJO class objects.
- ⇒ The results are level① cache and level② cache updated automatically at regular intervals.
- ⇒ We can specify parameters to disable level② completely after certain amount of time or to delete results related POJO class objects from level② cache after certain amount of time.
- ⇒ The level① cache keeps track of all the changes on HIB POJO class in a transaction and instead of generating multiple SQL queries, for this multiple changes it generates only one SQL update query selecting all the changes done on that object at end of the transaction.
- ⇒ See Related info on Level① caching page 3106 of the booklet.
- ⇒ HIB also supports Querylevel cache, as sub portion of the cache. This query cache holds Native SQL Query Result directly in its original format not in the format of the class object.
- ⇒ It is recommended to empty level① and level② cache time to time to know the changes done in DB from time to time.
- ⇒ To get control on level① cache we need to work with bulk methods that are invocable on HIB Session object.

① session.evict(pojo obj):-

Removes given POJO object from level① cache.

② ses.clear():-

emptys level① cache associated with current session.

Removes all HIB POJO objects from cache.

close() — closes the session object
closes HB session, so also closes the level 0 cache
associated with current session object.

Ques (FAQ) what is dirty state object in HIS environment?

Ans— If you modify Date of persistent state HB POJO class object manually in the client application, these changes will be synchronised with associated records in the DB table, only when session is flushed by calling session.flush().

Before calling this flush() changes done in persistence state POJO class object will be pending mode to synchronised with DB in this situation [before calling session.flush()] ~~then~~ we can say HB POJO class object is there 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(10));
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().

13/9/16

1st level cache is configurable cache in HB level,
there are multiple vendors supplying SW to enable
Second level cache in HB applications.

some vendors are supplying the second level cache
related jar files along with HB SW installation.

⇒ some example second level and opensource cache SW's

are :-

→ EHCache

→ OSCache

→ SwoopCache (SwarmCache)

→ JBoss TreeCache and etc.

→ Most of the time developer works with EHCache,
bcz it is fast, light weight, easy to use cache.

→ for details about various second level cache SW's
and theories, caching strategies refer page 106

⇒ Procedure to enable 'EHCache' as second level cache in HB Application :-

Step 1:- every second level cache will have, one provider
class name, try to know the provider class name bcz
EHCache.

→ for EHCache, the provider class name is :-
org.hibernate.cache.EhCacheProvider

Note:- setters hibernate-home \ etc \ hibernate.properties file
for various second level classes related providers
classnames.

Step①:- In configuration EhCache provided class name, is written in Hibernate configuration file.

In Hibernate Configuration file:

```
<session-factory>
    <property name="hibernate.cache.provider">
        org.hibernate.cache.EhCacheProvider
    </property>
```

Step②:- Add `HIB-Home\lib\EhCache-1.2.3.jar` file to the class path environment variable.

Step③:- enable second level cache in HIB application
in Hibernate Configuration file:

```
⇒ <property name="hibernate.cache.use_second_level_cache">
    true </property>
```

Step④:- write `EhCache.xml` file along with other resources of HIB application, specifying the parameters related to EhCache.

② ehcache.xml :-

```
<ehcache>
    <diskStore path="useos,disk"/>
    <defaultCache maxElementsMemory="1000" ②
        eternal="false" ③
        timeToIdleSeconds="920" ④
        timeToLiveSeconds="200" ⑤
        overflowToDisk="true" ⑥
    />
</ehcache>
```

Note:- all second level cache uses primary memory (RAM), if needed we can also use disk memory (HDD) as an enhancement memory.

Note:- for details regarding ①-⑥ numbers see the XML comments given in HIB-Home/etc/cfg.xml.

Step ⑥:- compile and execute HIB 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 HIB Application.

Step ①:- enable Query Cache from HIB Configuration file.

<property name="hibernate.cache.use-query-cache">true</property>

Step ②:- make HQL, NativeSQL Query Results as Cacheable from Client Application.

Query q1 = ses.createQuery("from EmpBean");

q1.setCacheable(true);

List l = q1.list();

==

Step ③:- set logical name for QueryCacheRegion from Client Application.

Query q1 = ses.createQuery("from EmpBean");

Step (4) :- make sure that 2nd level cache also enabled, before enabling the QueryCache.

Performing various operation on secondlevelcache Client appn :-

Note:- use SessionFactory object for these operation

→ factory.close() → releases second-level cache, associated with session Factory object

→ factory.evict(EmpBean.class) → Removes all emp bean pojo class object, from 2nd level Cache.

→ factory.evictCollection("phones") → if phones property configured in hbm mapping file, is related to Collection Fw DS, all collection Fw DS objects depositing phones property will be removed from second-level-cache

factory

→ factory.evictQueries(); → Removes HQL, NativeSQL Query results, from current 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~~ 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 Slw.

A → (Automaticity), C → (Consistency), I → (Isolation),
D → (Durability)

① → The process of combining related sub operation into single unit is called 'Automaticity'

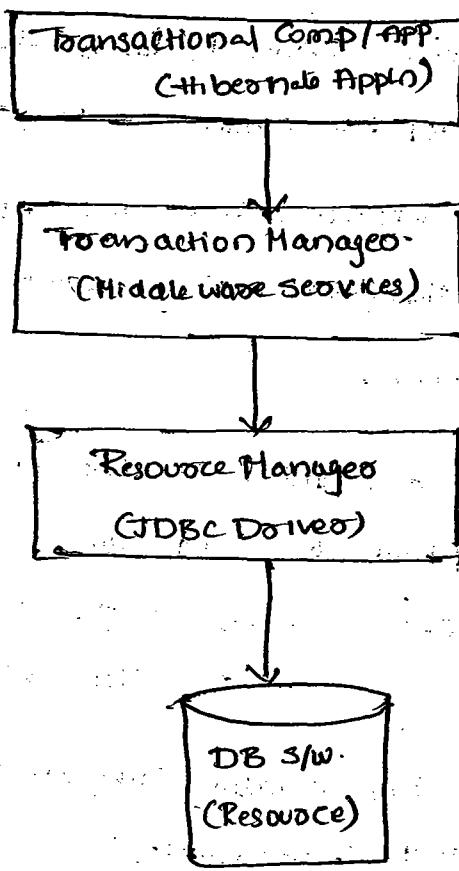
② → The process of getting guarantee, that rules kept on DB Slw. (like balance must not be negative) or not violated at the end of the transaction even though they are violated in the middle of Transaction is called "Consistency"

③ → The process of preventing concurrent operation on DB Slw from multiple users and application by applying 'locks' is called "Isolation"

④ → The ability of bringing the DB back to normal state by using log files and Backup & restore when DB is crashed and using DB data for long time is called "Durability".

14/9/10

Architecture of Transaction Management 2 -



- The app (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 that are involved, there are two types of transaction (DB SW's)

① Local Transaction:-

All the operations of application code, on which transaction management is enabled will deal with single resource (DB SW).

② Distributed

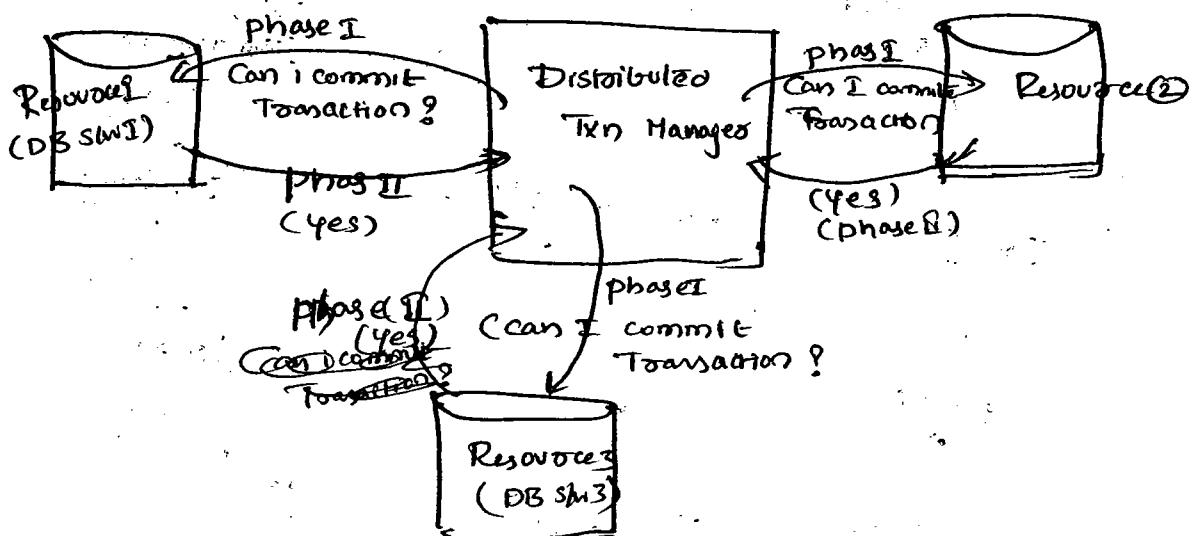
Ex:- Transfer money operation b/w same two accounts of same bank.

② Distributed Transaction:- If multiple resources or DB SW 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 uses based on 2PC protocol
(Two phase commit protocol)

③ 2PC protocol :-



According to 2PC protocol, in the phase I, Distributed Transaction manager asks all DB SWS permission to

commit the Transaction,

In phase II, all the DB SWS gives permission to commit the Transaction, then the distributed Transaction will be committed (OW) the distributed Transaction will be roll backed.

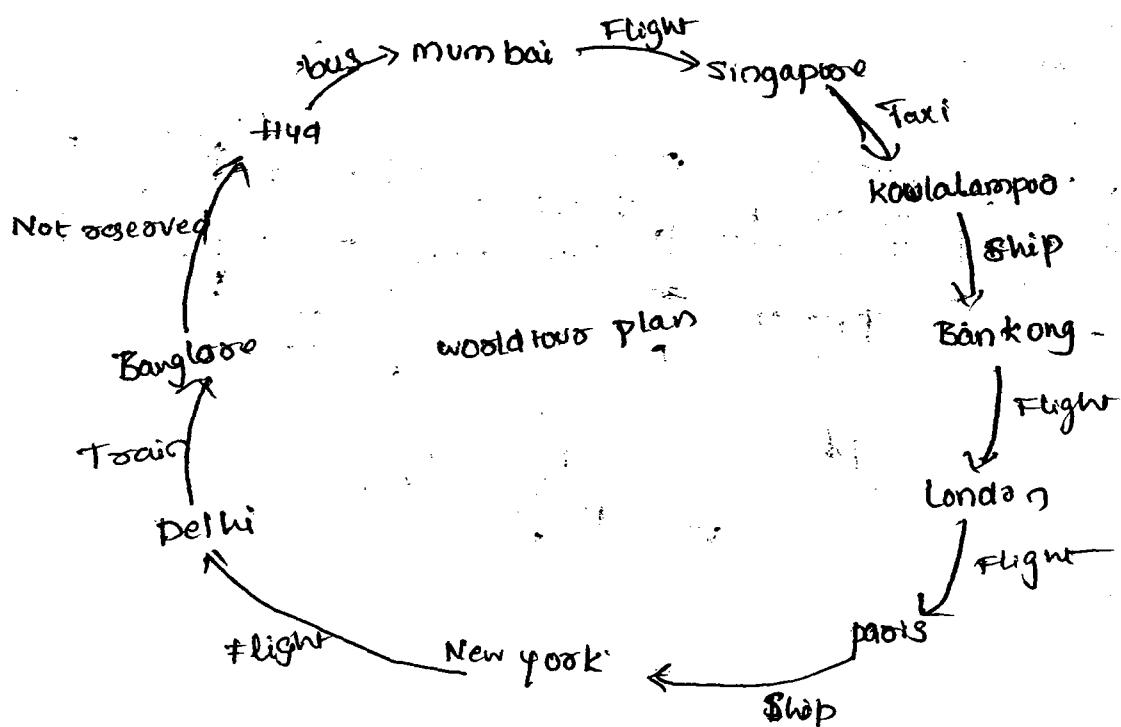
=> HIB support the local transaction management, but does not support distributed transaction management.

=> Spring, EJB Technology support both, distributed and local transaction management.

=> There are two transaction models

① Flat Transaction

② Nested Transaction.



Main Transaction .
does not effect other subtransactions etc
so two succns of dealing etc are as follows
as main transaction ,
booking operation will be taken as the sub trans
journey ticket because each journey ticket
it will be ~~cancel~~ commuted the remaining
one of others journey ticket are not available
to that runs with nested transaction, even tho'
when the above two plan application gives ←
—
5

journeys

journeys

Journeys Ticket booking ,

Main Transaction 6

Fair Transaction :-

will back.

so if one operation fails on other operations will
will be taken as direct operations of main transaction
because each journey ticket booking operation

it will carry on other journeys ticket .

If one (or) others journey ticket goes not available

Stu application that runs with ~~fair~~ Fair Transaction
when the above two plan is given to a ←

Nested Transaction :-

Main Transaction {

~~get name~~ subTxn 1 {

 ≡

 journey 1

 ≡

 ?

 subTxn 2 {

 ≡

 Journey 2

 ?

 ≡

 subTxn 3 {

 ≡

 Journey 3

 ?

 ?

≡

*** → EJB, HB, and spring support Flat Transactions.

→ EJB & HB doesn't support Nested Transaction.

→ spring support Nested Transaction.

⇒ Sample Code that performs Transaction management in HB environments :-

```

public void form() {
    Transaction txn=null;
    try {
        txn = ses.beginTransaction();
        persistence operation 1;
        persistence operation 2;
        persistence operation n;
    }
    catch (Exception e) {
        if(txn!=null)
            txn.rollback();
    }
}

```

```

    txn.commit();
}
try
{
    txn.rollback();
}
catch(Exception e)
{
    try
    {
        catch(Exception e1)
    }
    =
}
}

```

⇒ Example application to perform Transfer money operation b/w

two account b/w same bank having Transaction management

> alter table account add constraint xyz primary key (accno);

Step①:-

create account table in oracle db s/w.

> ~~select~~ create table account (accno number(5), pk,
accname varchar2(20), balc number(8,2));

Step②:- Launch MyEclipse IDE and create java project having
name "TxBApp"

Step③:- Create DB profile for oracle by using my eclipse DB
~~profile~~ explorer.

Step④:- Add TIB capabilities to the project.

Step⑤:- Perform reverse engineering on account table of
oracle DB s/w, 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 manager

out.java :-

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
public class TestClient {
    public static void main (String args[]) throws Exception {
        Session ses = HibernateSessionFactory.getSession();
        Transaction tx = null;
        try {
            tx = ses.beginTransaction();
            // operation 1 (with draw operation from acc amount)
            Query q1 = ses.createQuery("update Account set balance = (balance - ?) where acco = ?");
            q1.setLong(0, 4000);
            q1.setLong(1, 101);
            int res1 = q1.executeUpdate();
            // operation 2 (deposit amount operation to dest account)
            Query q2 = ses.createQuery("update Account set balance = (balance + ?) where acco = ?");
            q2.setLong(0, 4000);
            q2.setLong(1, 102);
            int res2 = q2.executeUpdate();
            if (res1 != 0 && res2 != 0) {
                tx.commit();
                System.out.println("Tx is committed");
            } else
                tx.rollback();
            System.out.println("Tx is rolled back");
        } catch (Exception e) {
            System.out.println("Caught Exception " + e);
            tx.rollback();
        }
    }
}
```

⇒ add objabc14.jar to built part of the project.

⇒ run the client application.

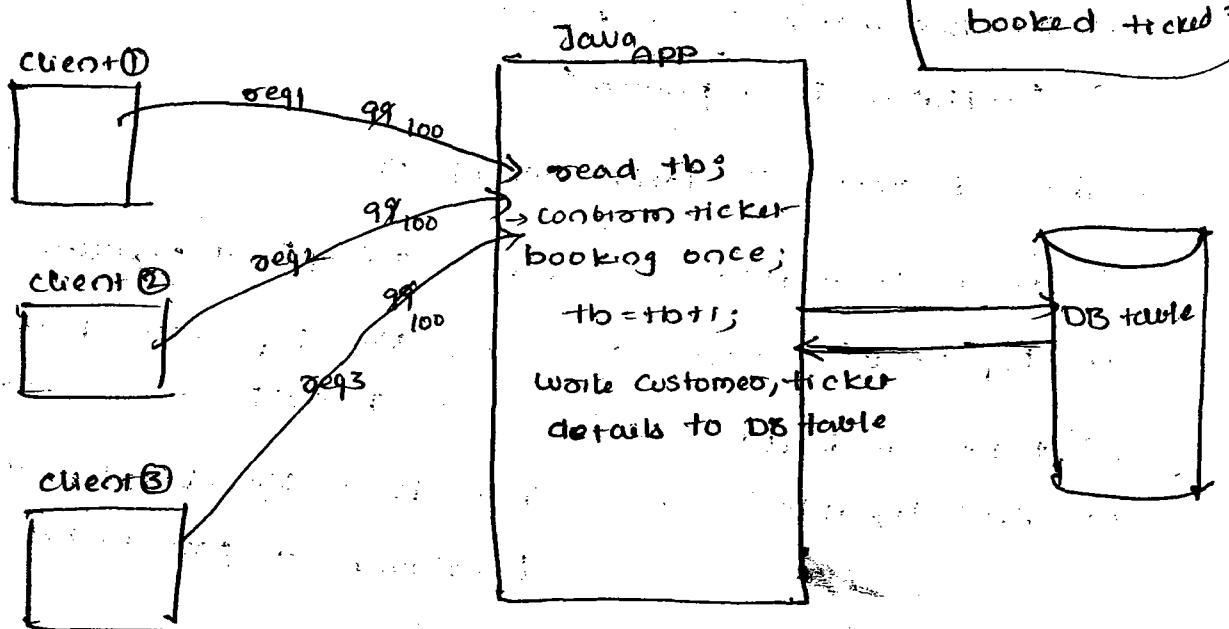
Exception

15/9/10

ISOLATION

→ If multiple users trying to access (or) manipulate DB at simultaneously & concurrently there is a possibility of getting corruption in the DB data. To prevent this problem apply locks on DB at various levels by using ISOLATION levels & allow only one user (or) one app to manipulate DB data.

Problem:-



In the above Diagram, Application and DB 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 SW by using ISOLATION levels.

→ If Application or DB is allowing another user (or) client to use the same data before complete

(a) If two or more operations, is called "concurrent operation" (b) "simultaneous operation" (c) "Interleaving operation"

→ when DB allows, these Interleaving or simultaneous database operations, there is a possibility of getting following type of problems.

To solve these problems we need to use different

Isolation levels

~~the~~ problems are:-

- ① Dirty Read problem
- ② Non-repeatable Read problem
- ③ Panther 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 ~~Panther Read~~ → Dirty Read, Non-Repeatable Read, Panther Read problems.

∴ we can set all these isolation levels on DB side, being from java applications by using JDBC code (a) HB Code (b) any other persistence logic code.

⇒ Dirty Read problem :-

- ⇒ If user A, user B have joint account holders to B.
⇒ DB SW is allowing uncommitted Reads.

UserA

UserB

1) User A reads the balance
of Joint a/c
(Rs: 5000)

2) UserA begin Txn and
deposits Rs: 3000 into joint
a/c then
 $bal = bal + amnt;$
(Rs: 8000)

(3) User B withdraws Rs:
from account (joinr)
 $bal = bal - amnt;$
(Rs: 1000)

④ User A abort / rolls back
Txn; so the balance becomes
Rs: 5000. This 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 makes
applications to read only committed data of DB

→ In most of the DB SWs Read Committed is
default Isolation level.

ex:- ORACLE, MySQL.

Non-Repeatable Read Problem :-

User A

User B

(1) At beginning of Tx UserA gives

Select query and gets five 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 UserA,

he issue same select query

he gets 10 records with modified

date. 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' mechanism on

DB. Due to this, ~~so~~ This Isolation levels also solve

Dirty Read problem.

Panther Read Problem

User A

User B

- ① At beginning of Tx user A gives select query and gets few records (let us assume 10 records) from DB Slw.

dirty records

- ② User B insert more data (let us assume 4 records) in same table which also satisfies select query condition of user A.

- 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 Panther Read Problem.

To solve Dirty Read, Non-Repeatable Read, Panther Problem Serializable Isolation level, i.e., it uses application

Read & Write Locks on DB.

PL

=> PL (or) DB designing team decides the Isolation level but programmer is responsible to configure apply this Isolation level on DB Slw.

-> Some DB Slws. doesn't support all Isolation level

for example

→ Oracle doesn't support Serializable Isolation level -

→ MySQL supports 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 :-

Connection.TRANSACTION-READ-COMMITTED	→ 2
Connection.TRANSACTION-REPEATABLE-READ	→ 4
Connection.TRANSACTION-SERIALIZABLE	→ 8

→ setting Isolation level <from HB application :-

In HB configuration file,

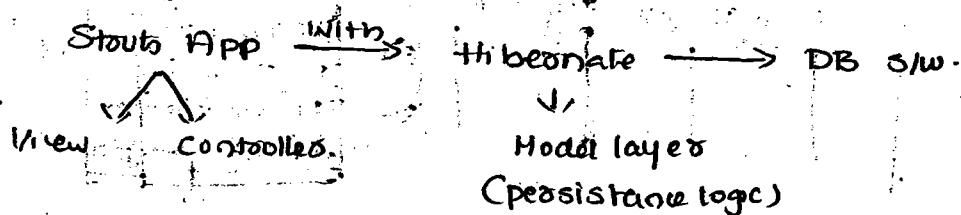
{ <property name = "hibernate.connection.isolation" > 4 </property>
(OR)
<property name = "hibernate.connection.isolation" >
REPEATABLE-READ </property>

Other possible values are :-

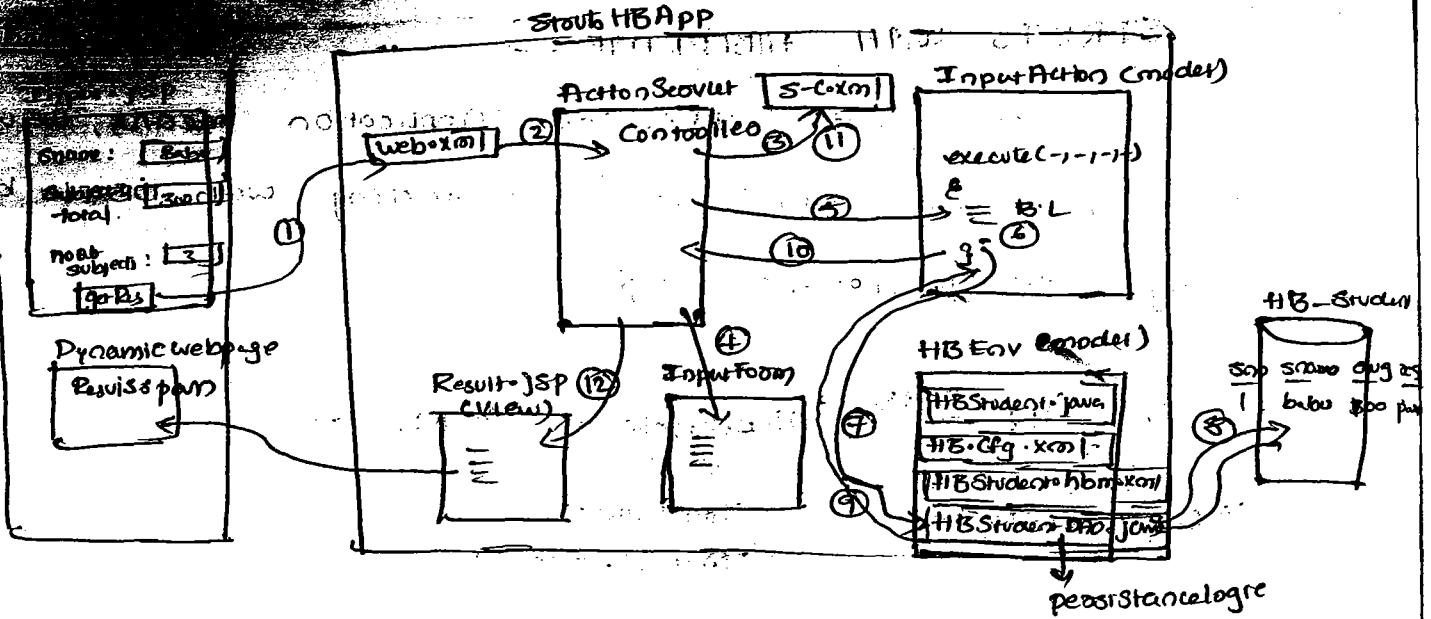
{ HIBERNATE
READ-COMMITTED → 2
REPEATABLE-READ → 4
SERIALIZABLE → 8

STRUTS WITH HIBERNATE :-

Struts with Hibernate Application means, we want to make Struts application interacting with DB s/w by using HB persistence 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 persistence 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 persistence 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.



These are two approaches, to develop stout with Hibernate Application.

Approach① :-

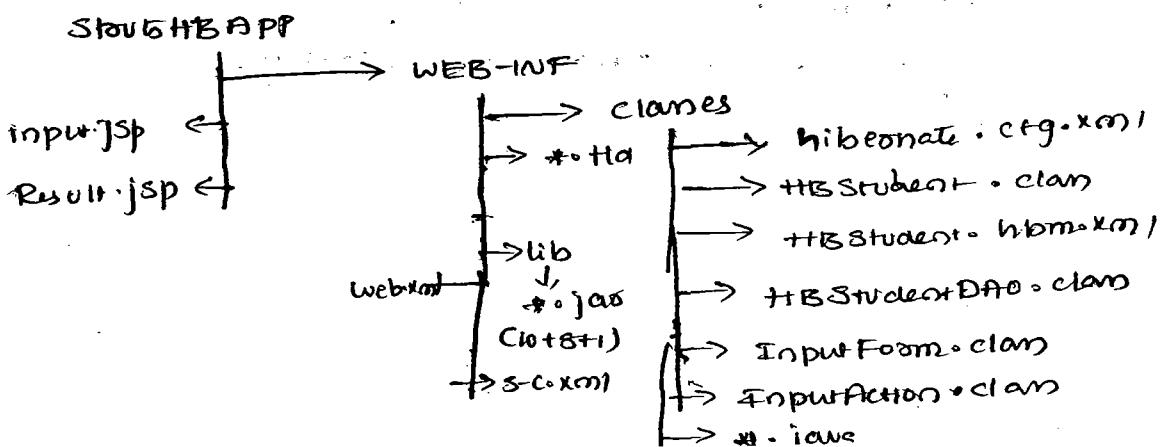
By creating HB Session object in stout action class or DAO class

Approach② :-

By creating HB Session object in user-defined plug-in class of stout

→ If above application is developed manually

based on Approach ① Deployment Directory Structure as shown below.



Jar files to the class path:-

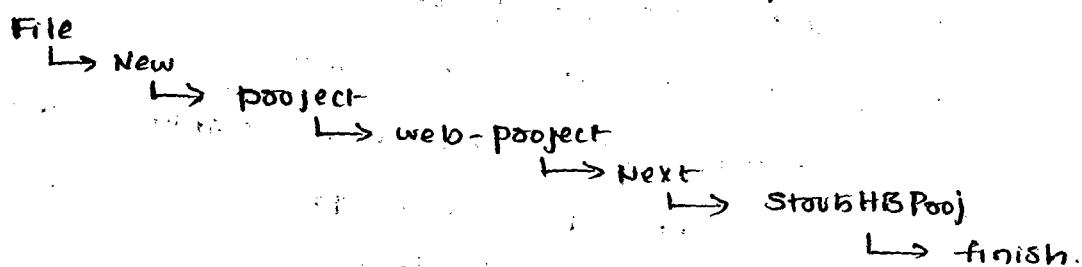
- Sevlet-api.jar
- slout-war-revision.jar
- hibernate3.jar

Jar files Required in WEB-INF\lib folder

- 8 tIB jar file
- 10 slout jar file
- 1 ojdbc14.jar.

→ Procedure to develop above diagram based App, by using
MyEclipse IDE

Step① - Create web project in MyEclipse IDE.



Make sure that HB-student table is there in oracle DB, by having pk constraint on sno column.

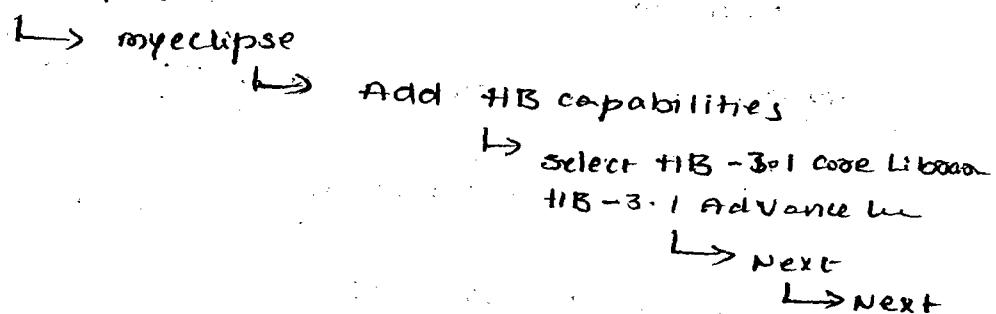
o HBStudent

sno(pk)	sname(varc)	total(num)	Avg(num)	Result(varc)

Step② Create ORACLE DB SLW Related DB poolfile in My-Eclipse IDE.

Step③ - Add tIB capabilities to the project

Right click on project



Step 5 : Create a DAO using (Create, create, DB properties)
Right click on project → Next
→ de-select Create SF class
→ Finish
→ add show.sql property

Step 6 :- perform HB Reverse engineering on HB_Student table.

go to DB Browser window

→ Right click on HB-STUDENT TABLE
→ HB Reverse engineering

- Java source folder : /Stout/HB/proj/dao
- select 1st, 2nd, 3rd check box.
- de-select 4th check box.
- select 5th check box. (JAVA DataAccess Object)

DAOType ← basic DAO

→ Next

→ HB DAO type.

→ 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.

Step 7 :- Add stout capabilities to the project.

Right click on project

→ My Eclipse

→ Add stout capabilities,

→ select stout4.0.x

↓
Finish.

Step 6 :- Add FormPage, FormBean Class, Action Class, ResultPage

↓ ↓ ↓ ↓
FormPage :- Input.jsp Inputform InputAction Result.jsp

~~Form~~

Right click on project

→ New

→ Other

→ HyEclipse

→ webstart

→ start 1-2

↓
start 1-2 Form, Actn
JSP

↓
[Next]

④ Create all mentioned class

Step 7 :- Right following code, in the execute (-,-,-,-) of InputAction class.

InputAction.java

=

execute (-,-,-,-)

6

```
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 = "pass"
dao =  

HBStudent st = new HBStudent();
st.setSname(name);
st.setTotal((long) total);
st.setAvg((double) avg);
st.setResult(res);
dao.save(st);

request.setAttribute("result", res);
return mapping.findForward("success");
```

Add Result.jsp Page to the web-Root folder at project.

 Student Result is <%= request.getAttribute("result") %>

Step ① :- Configure External Tomcat Server to MyEclipse IDE

window

→ preferences

→ my eclipse

→ application servers

→ tomcat

→ Tomcat 5.x

enable

② Tomcat 5.5

[Apply]

→ [OK]

Step ② :-

start Tomcat server from MyEclipse IDE.

goto server icon () tool bar

→ Tomcat 5.5

→ start

→ Deploy the project in Tomcat Server from MyEclipse IDE

goto Deploy () in the tool bar.

→ Add

→ server : Tomcat 5.5

[Finish]

→ [OK]

Step ③ :- Test the Application.

④ open browser window from tool bar :-

→

http://16.2020/MEStuts

ab
17/9/10

struts with hibernate using plug-in -
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-action.Plugin interface.

- every ~~for~~ 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 ActionServer object is created in struts F/W SW is
activated.
- If you want to make all Actionclasses in struts appln
using single HIB session object for all the request,
then place that logic in the user define Plugin class in
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 in web-app.

→ procedure to develop yesterday's application without
DAO class and by adding user defined struts-
plugin class :-

Note:- user define plugin class in struts application
always resides in WEB-INF\classes folder in
struts application.

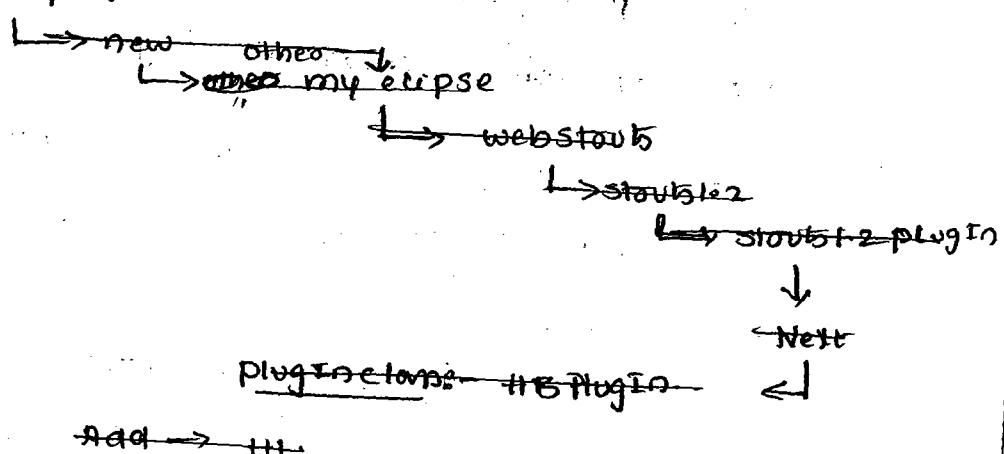
~~example of previous application.~~

perform HB Reverse engineering on HB student table. [don't select 'DAO' option]

Step 5 :- same as previous application.

Step 6 :- ~~Add~~ Add User defined stout-plugin class to the project. and place logic to create HBSession object in that class.

Right click on project



=> imports →
public class HBPlugin implements PlugIn {
 Session ses = null;

```
import javax.servlet.*;  
import org.apache.stout.action.*;  
import org.apache.stout.config.*;  
import org.hibernate.*;  
import org.hb.cfg.*;  
public class HBPlugin implements PlugIn {  
    Session ses = null;
```

```
    public void init(ActionServlet sevler, ModuleConfig mc)  
        throws SevlerException {  
        super.init();  
    }
```

SOP ("HBPlugin: init-> method")

```

toy 6
    if create HBSession obj
        ses = new Configuration().configure().buildSF().openSession()
        get SeaviewContext obj

        SeaviewContext sc = seavut.getSeaviewContext();
        if keep HB Session obj in SeaviewContext attr
            sc.getAttribute("HBses", ses);
        }

        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

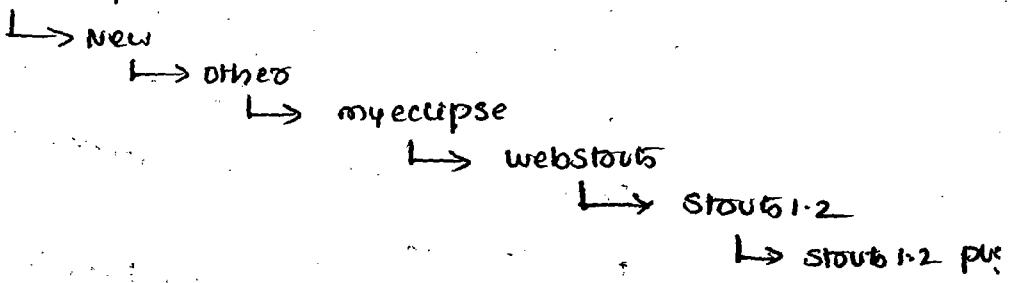
    public void destroy()
    {
        try
        {
            ses.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

```

Step ④:-

Configure above created plugin class with Struts appen, in Struts-configuration file.

Right click on project



Plugin Class:-
HBSPlugin

Finish ↵

→ The init() of struts plugin class, executes during the start up of the ~~Plug~~ Struts Appen, when ActionSet object is created.

→ The destroy() of struts plugin class, executes when Struts Application is un-deployed, stopped (or) reloaded.

same as step ⑥ of previous project

replied. (Copyright)

wrote 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 Accus ActionServer obj

ActionServer seover = getSeover();

//get accus to SeoverContext obj

SeoverContext sc = seover.getSeoverContext();

//read HIB session obj from sc.getAttribute

Session ses = (Session) sc.getAttribute("HIBses");

③ //B logic

double avg = (double) total / nosub;

String des = null;

if (avg < 35)

des = "fail";

else

des = "pass";

// write HIB Persistence Logic to insert the des

HIBStudent st = new HIBStudent();

~~st.setDes~~

st.setSname(name);

st.setTotal(total);

st.setAvg(avg);

st.setResult(des);

Step ⑪ :-

Step ⑪ → Step ⑫ are same as .

Step ⑬ → Step ⑫ of previous appn.

Transaction tx = ses.beginTransaction();

ses.save(st);

tx.commit();

// send result to result.jsp

request.setAttribute("result", res);

action mapping. find Forward ("success");

* * Note :- Plugin class creates session object only once and makes all Struts Action classes use that object

for all the request

→ The DAO class separate persistence logic from other logics of the application and also makes DB persistence logic as reusable logic in multiple Struts Action classes.

so you can add both plugin, DAO support in Struts with Hib application.

But the DAO class should use that Hib Session object that is created in Plugin class.

⇒ For EJB ~~or~~ SessionBean with Hib application added application given in page no 8- 92 to 96

Annotations Based on beans -

- 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 Server in web.xml file, to pass the details of server 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 to 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 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 JDE initial versions or 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

⇒ o @ Override, @ Inherit, @ 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.

⇒ H.B. 3.0.1-X onwards, Spring 2.5 onwards,
EJB 3.0 onwards, Struts 2.X onwards, Servlet 3.0 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 vs configurations, avoiding disturbing the source code.

→ we can apply Annotations at 3 levels

① Resource level (on class or Interface)

② Field level (on the member variables)

③ method level

⇒ To work with Annotations based HB programming
use either hibernate 3.4 SW completely

use hibernate 3.2.5 SW \oplus HB 3.3 annotations

same way to get the .

⇒ All ~~the~~ Annotations of HB programming are designed
based on JPA (Java persistence API), EJB 3.0 specification

⇒ All Annotations are special interface having
@interface keyword.

implementation of this interfaces having @interface keyword
which allows to use Annotations for metadata ~~annotation~~

operations

⇒ For API documentation integration annotation of HB
programming, go to ~~the~~ hibernate 3.3 annotation ~~home~~ - home/
doc\JPA-API\index.html file.

⇒ For Reference example application based on Annotations

debbos The test folder of HB 3.3 - annotations to mediate

⇒

Annotations :-

we can use generated algorithms to generate identity values as POJO class dynamically by using some algorithms.

→ public class Student

{

~~id~~ .

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

int sid ;

—

—

3

GenerationType = AUTO

GenerationType = AUTO

↓

picks up the algorithm dynamically

based on the capabilities of underlying

DB like native algorithm

Other possible values for strategy parameters :-

GenerationType =

GenerationType = SEQUENCE (uses sequence algorithm)

GenerationType = IDENTITY (uses identity algorithm)

GenerationType = TABLE (uses hilo algorithm)

class Stu

Procedure to configure uses define sequences as Identity generator :-

class Student

{

@Id

@Column(name = "sno")

@GeneratedValue(strategy = GenerationType.SEQUENCE,

generator = "my_seq")

Integer sno;

—

3

<Composite name = "addr1", column = "stradd", />
</Composite>
<Key-Composite key = "name" column = "strname", />
<Key-Composite key = "strno" column = "strno", />
<Composite -!d>
<Class name = "Student", table = "Student-Tab", />
<hibernate-mapping>

File Name file

g

at XXX () & set XX

for age

String address

String name

int score

g

public class Student

building the base composite identity field Configuration

algorithm such working with composite identity field.

Note:- we can not work with identity value generated

by

Identity field Configuration.

Configured as Identity fields then if is called composite

If more than one default is this two class are

use this operation.

use <id> at the mapping file are use @Id anno

identity field.

Configured as Identity field, then if is called singular

If only one member variable of the both class, is

Composite Identity Configuration -

Annotation Based Composite Identity Field Configuration :-

⇒ @Entity

@Table (name = "Student_Tab")

@Id Class (StudPk, class)

public class Student

{

 @StudP

 @Id

 @Column (name = "std_id")

 StudPk id;

 String address;

 int age;

=

 getters & setters

}

@Embeddable

public class StudPk implements Serializable

{

 int std_id;

 String name;

 getters & setters

=

{

=

← End of Hibernate →

C04