HIBERNATE
The Legend Java

BY
Mr.NATARAJ Sir

NARESH TECHNOLOGY

SRI RAGHAVENDRA XEROX

Software Languages Material Available

Beside Bangalore Ayyangar Bakery, Opp. C DAC, Ameerpet, Hyderabad.

Cell: 9951596199

Hibernate

RS = 100 =

Introduction

⇒ Java is a programming language.
⇒ Jdbc, Sewlets, JSP, EJB, JMS, Java Mail, JTA and etc.. are technologies.
⇒ Hibernate, Struts, Toplinks, Spring, JSF and etc.. are frameworks.

Java framework important technologies

a) Persistance store
b) Persistance Data.
c) Persistance operations
d) Persistance Logic
e) Persistance Technologies

CRUD/CURD operations

C- create
U- Update
R- Read
D- Delete

SCUD operations

S- Select
C- create
U- Update
D- Delete

a) Persistance store

⇒ The store where data can be saved and managed for long time is called Persistance store.
ex: Flat files (Directoly managed by OS), DB SW.

b) Persistance Data

⇒ The data of Persistance store is called Persistance Data
ex: Content of file, Db Table records

c) Persistance Operations

⇒ The insert, delete, select, update operations are called Persistance operations. These are called CURD/CRUD/SCUD operations.

d) Persistance Logic

⇒ The Logic to perform persistance operations is called Persistance Logic.
ex: Jdbc code, Hibernate code.

e) Persistance technology

⇒The technologies that can be used to develop persistance logic is called Persistance technology.

Ex: JDBC, Hibernate Framework.

* Java Applications → IO STEAMS PERFORMS → Flat files
(Not Associated with any technology)

* Java Applications → JDBC, O-R Mapping → RDBMS DB SW
(Oracle, Sybase, PostgreSQL, MySQL, --)

Files are having the following limitations are persistance stores

a) No Security
b) Cannot maintain huge amount of data
c) Data Redundancy Problem (Duplications)
d) No SQL Support
e) No support for constraints and etc...

⇒ Still files are good as Persistance store in small scale Apps like Desktop Games, Mobile Games and etc.

⇒ To overcome the above problems use DB SWs as Persistance stores in Large scale, Medium scale Apps.

Ex: Websites, Banking Projects and etc..

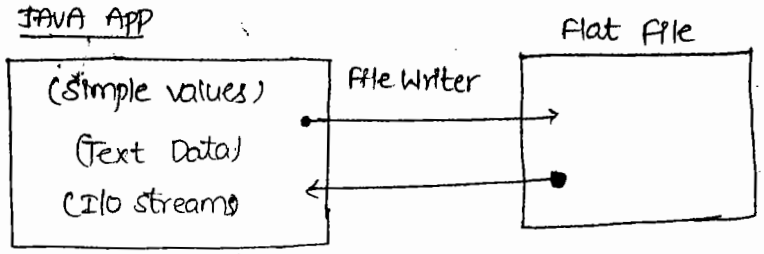
⇒ There are multiple approaches to perform persistance operations on files, Database softwares from Java Applications

⇒ In Java, Data can represented either in the form of multiple simple values (text data) or in the form of objects.

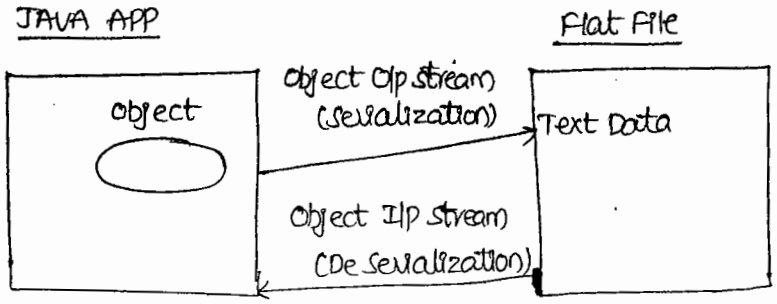
```
int sno=101;
String name=rajesh;
Student st=new Student(101,"rajesh"); // as a student class object
                                      (constructor)
```

Approach1 : If data is there as text data (simple value) and flat file is persistence store.

store



Approach2 If data is there in the form of object and flat file is Persistence store.



⇒ The process of capturing object data and storing that data to a file is called "Serialization".

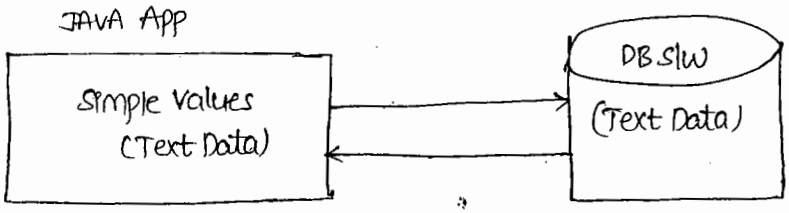
⇒ Reading data from a file and creating object having the data is called "DeSerialization".

⇒ In real time after capturing simple values from end users they will be store in objects and used as Objects through-out application.

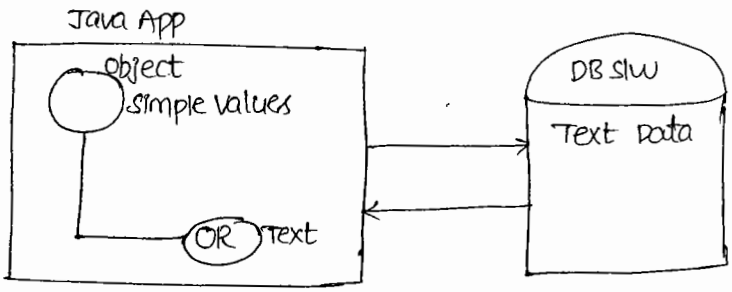
⇒ Even persistence Operations will perform on through objects

NOTE Approach1 and Approach2 we are using files as Persistence stores which are having the above said repetitions.

Approach3 Java Application contains text data/simple values and Database software as Persistence store.



Approach4 Java Application contains data in the objects and Database sw is Persistance store(use JDBC).



⇒ To persist object we need to get text data/simple values from object (ie, To insert object data we need to convert object data to text data/simple values to use them in insert SQL Query).

⇒ Jdbc uses SQL Queries for persistence operations, but these SQL Queries are Database sw dependent queries, so Jdbc persistence logic is database sw dependent.

⇒ changing database sw in the middle of development/production is very complex while working with Jdbc.

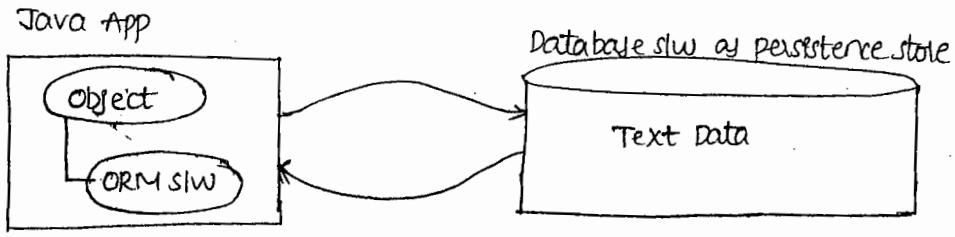
⇒ Jdbc Resultset is not a serializable object so we cannot send that object over the networking

⇒ There is no proper support for Transaction Management in Jdbc Program.

NOTE

⇒ To solve all the problems of Approach1 to Approach4, use Approach5 that makes programmers to use ORM software objects based persistence logic.

Approaches: Java Application contains data in the form of objects and uses O-R Mapping.



⇒ ORM slw allows programmer to develop O-R Mapping persistence logic where all persistence operation can be done directly through objects.

List of ORM softwares

- Hibernate ⟶ From SoftTree (RedHat) (1)
- TopLink ⟶ From Oracle Corporation (3)
- OJB ⟶ From Apache
- EJB Entity Beans ⟶ From Sun MS (Oracle Corp) (5)
- JPA ⟶ From Sun MS (Oracle Corp) (4)
- JDO ⟶ From Apache
- iBatis ⟶ From Apache (2)

Q What is O-R Mapping?

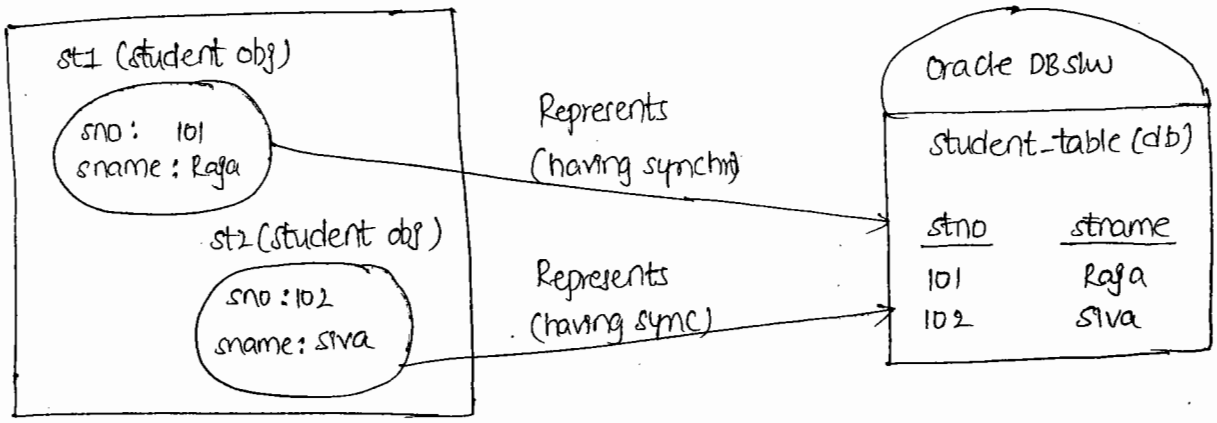
A The process of mapping java class with Database Table, java class member variables with Database Table columns and making the objects of java class representing database table records having synchronization between them is called O-R mapping.

⇒ In O-R mapping synchronization means the modifications done in object data will reflect

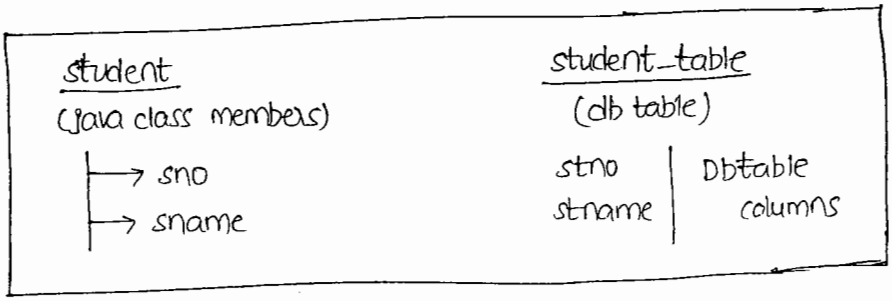
Ex:

```
public class Student {
  //properties
  private int sno;
  private String sname;
  //methods
  |==
}
```


SRI RAGHAVENDRA XEROX
Software Languages Material Available
Beside Bangalore Ayyagar Bakery,
Opp. CDAC, Balkampet Road,
Ameerpet, Hyderabad.



O-R mapping cfgs (Either through XML or through annotations)



⇒ In O-R mapping, we can *map multiple Java classes with multiple Database tables as per object requirement.

1) We design Java classes or O-R mapping through object oriented programming principles and we design database table through normalization principles. In this process some impudence mismatches may come between classes and database tables, we can overcome mismatchy through or mapping cfgs.

2) For example, we can design Java classes of o-r mapping through inheritence but the database tables of some Java classes cannot be inheritedee. we can overcome this mismatches of o-r mapping and we can design the classes of inheritance pointing to the DB tables.

⇒ Every ORM Fw internally uses jdbc code + sql queries to complete persistence operations but it never makes programmer to know and use Jdbc code.

⇒ This indicates hibernate provides abstraction layer on jdbc programming (hides the implementation).

Q What is the difference b/w programming languages, s/w technology, s/w frameworks ?

Programming Language

1) These are directly install-able s/w as raw materials having basic features to develop s/w applications

2) Programming languages defines syntaxes and semantics to develop Application.
(rules) (structure)

3) These are for creating technologies, fw, tools, DB s/ws, OS and etc...

Ex : C, C++, Java, C#, VC++ and etc..

S/W Technologies

1) It is a s/w specification that contains set of rules and guidelines to develop the s/w using programming language. These technologies are not installable.

Ex: Jdbc is a Java Technology, that contains set of rules and guidelines to develop Jdbc driver s/w using programming language. Jdbc is not install-able but Jdbc driver is install-able.

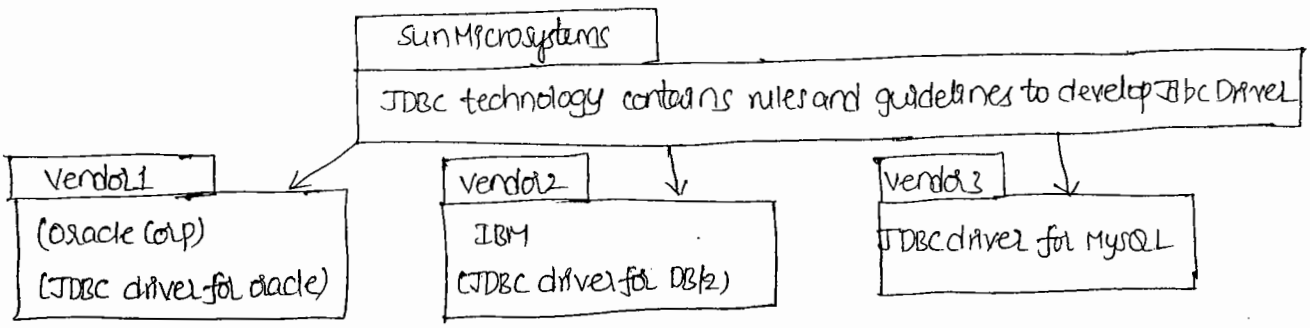
2) Types of Technologies

a) Open Technologies : The Rules and guidelines are open all vendor companies to develop s/w's.

ex: JDBC, Sewlet, ODBC, JSP, EJB and etc..

b) Proprietary Technologies: These Rules and guidelines are specific to one vendor company. So that vendor is only allowed to develop s/w's.

ex: Microsoft Technologies.



⇒ In Technologies SunMS gives the interfaces representing technologies rules & other vendor companies gives implementation classes for those interfaces.

⇒ Jdbc Technology gives interfaces & vendor who develop Jdbc Driver provide implementation classes for those interfaces.

FrameWork

⇒ These are directly installable s/w's having capability to generate common logics of the application dynamically based on application specific logics given by programmer.

⇒ These are directly installable s/w providing abstraction layer on existing technologies to simplify the application development process.

⇒ Abstraction layer means every fw internally uses technologies but it never makes the programmer to know about the application development.

⇒ If work with technologies programmers to develop both common logics & application specific logics.

⇒ If work with Programmers just need to develop application specific logic because the common logics with will be generated by fw dynamically using technologies.

⇒ Using fw application development, testing, releasing & maintains can be done fast & easily.

* In Java, based on the kind of applications we develop there are 3 types of fw's.

a) Web-Application Fw : Provides abstraction layer on sewlet, JSP & simplifies web-application development.

ex: struts, jsf, spring, mvc etc...

b) ORM Fw : Provides abstraction layer on jdbc & etc.. to simplify OR-Mapping Persistence logic development.

ex: Hibernate

c) Application Fw

⇒ Provides abstraction layer on multiple technologies to simplify all kinds of application development.

ex: spring.

NOTE while working with fw's & languages we can observe interfaces and these implementation classes are given by language / framework it self directly. Because they are directly installable.

Plain JDBC Application

1) Register JDBC Driver
2) Establish the connection || (common logics)
3) Create statement object
4) Send and execute SQL Query || (Application specific logics)
5) Gather results and process results
6) close JDBC objects || (common logics)
7) Handle Exceptions

⇒ In JDBC Application we need to write both common logics and application specific logics.

⇒ The multiple JDBC Apps contains the same common logics repeatly, so this problem is called Boiler plate code problem.

⇒ While working with FIWs we can avoid this boiler plate code problem because they generate common logics of the internally and automatically.

* Based on the mode of Application development we do there are 2 types of FIWs.

a) Invasive framework

⇒ Here fiw Apps should implement or extend from framework api interfaces or class respectively. Here Appln will be tightly coupled with fiw api's, so there is no possibility replacing one fiw with another fiw.

Ex: Struts 1.x

b) Non-Invasive Framework

⇒ Here fiw application need not to implement or need not extend from fiw application interfaces or class respectively. Here applications are not tightly coupled with fiw api, so we can replace one fiw with another fiw by changing the libraries ("jar-files")

Ex: HB (Hibernate), Struts 2.x, spring and etc..

⇒ while developing offline appln (end user involvement will not be there) that deals with huge amount of data transverse prefer using jdbc because ORM fiw like HB create huge amount of sw while develo dealing huge number of records use this system may crash.

Ex: Gather senses info and storing oracle db siws.

⇒ While developing online applications (that application ie., operated by endusers) as internet or intranet web application that deals with small amounts of data we need to use hibernate to take the advantage of hibernate features.

Ex: website that display stock market share values, website that displays shopping items.

Hibernate

Type : Java Based ORM Fw.

Version : 4.3.x (compatable with jdk 1.6)

Vendor : SoftTree (Red Hat)

Open source fw

Creator : MR. Gavin King

⇒ To download sw : download as zip file from www.hibernate.org or www.sourceforge.org (hibernate-release-4.3.5.Final.zip) (2014.04.02)

⇒ To install sw : Extract zip file

<HB_home> | changelog.txt contains various releases info of hibernate

<HB_home> | project → contains simple HB application, utility files, source code and etc.

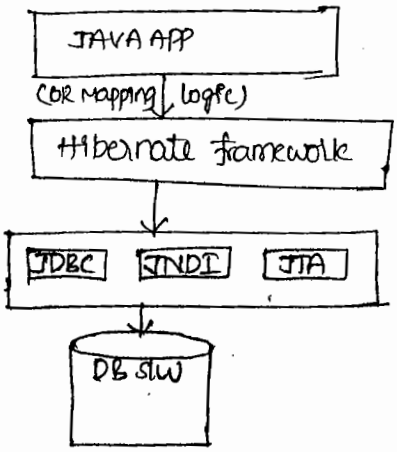
<HB_home> | documentation → contains api documents, pdf manuals and etc..

<HB_home> | lib → HB libraries (api jar files)

HB-core-4.3.5.Final.jar main jar file of hb api and it contains multiple dependent jar files

⇒ If the classes and interfaces of one jar file are using the classes and interfaces of another jar files the another jar files are called dependent jar files to first jar files (one jar file)

⇒ If java app uses new api's | third party api's (like hb api) then we need both main and dependent jar files of that new api or third api to class path or buildpath (eclipse)



JNDI : Java naming and directory interfaces

JTA : Java transaction API.

⇒ HB is an open source, light weight, non-invasive java based ORM FlW to develop objects based DB slw independent o-r mapping persistence logic in all kinds of java, jee and java flws apps.

⇒ Along with HB slw installation we get source code this makes HB slw as open source.
(<HB_home> |project |hibernate-core)

⇒ HB is light weight
a) Installation of HB slw takes less memory in the MBS
b) To execute HB code we do not need heavy weight containers or server slws.
c) we can develop logics in some HB resources with out using HB api.

⇒ HB is non-invasive became the classes of HB application can develop with implementing extending from the interfaces or class of HB api.

⇒ HB is ORM FlW because it allows to develop objects based HB gives as HQL as DB slw independent Query language.

⇒ HB supports lazy loading (Gets data from DB slw lazily on demand ie., we will not load data from DB table immediately. It will load data from DB slw only when need is there).

⇒ Allows to develop objects based persistence-logic in standalone environment and also in web application environment (J2EE environment).

⇒ Does not need the heavy weight containers, server to execute HB code.

⇒ HB o-r mapping solves the mismatches blw java programming and DB programming.
ex: java supports inheritance but database tables do not support inheritance, still we can make the java classes of inheritance interacting with DB table).

⇒ HB supports both xml and annotations based O-R mapping configurations.

NOTE: Making underlying server, flw recognizing the resource class or interface based on the given details of the resource is called resource configuration.

⇒ HB supports two levels of caching to reduce nlw round trips blw HB application & DB slw.

NOTE cache|buffer is a temporary memory that holds data for temporary period.

⇒ HB persistence logics can be written in all kinds of java applications, j2ee technologies applications and java flws applications (Easy Integrations).

⇒ HB supports versioning
(allows to keep track of how many times the object|record is modified).

⇒ HB supports timestamping
(keeps track of when object/record is saved or modified with data and time values)

⇒ Gives supports to certain API which generates performance tuned SQL Queries internally based the given objects based persistence logic.

⇒ Supports object level relationships like one to one, one-to-many, many-to-one, many-to-many.

⇒ Since HB allows to develop objects based DB sw independent persistence logic we can change DB sw the middle of development or production environment easily.

⇒ HB easy to learn and easy to use and etc...

⇒ MVC is industry defacto standard architecture to develop java based websites.

⇒ M → Model → Represents Business logic + persistence logic + Data of the application.

⇒ V → View → Represents Presentation logic (It is like Beautician)

⇒ C → Control → Represents Integration logic (The logic that monitors all the activities of web application).

(The logic that makes view layer and model layer components interacting with each other)

V C M
JSP → servlet → javaclass ⟶ Database sw.

V C M
JSP → SERVLETS → EJB Component → DB sw

V C M
JSP → SERVLET ⟶ EJB SESSIONBEAN → EJB ENTITY BEAN —
(Business logic) (persistence logic)

← Model →
STRUTS → EJB SESSION BEAN → HIBERNATE → DB sw
(V)(C) (Business logic) (persistence logic)

← Model →
STRUTS → SPRING ⟶ HIBERNATE ⟶ DB sw
(V)(C) (Business logic) (persistence logic)

← Model →
SPRING MVC → SPRING JEE → SPRING DAO / SPRING ORM → DB sw
(V)(C) (Business logic) (persistence logic)

← Model →
SPRINGMVC → WEBSERVICE → SPRING DAO / SPRING ORM / Hibernate → DB sw
(V)(C) (Business logic) (persistence logic)

NOTE: HB is useful only to develop persistence logic in project, it cannot be used to develop other logics in projects.

List of View Layer Technologies

html, jsp, ajax, velocity, free marker, jquery, dojo

List of Controller Layer Technologies

sewlet, sewletfilter

List of Model Layer Technologies to develop business logic

RMI, Java classes, EJB session Bean, WebServices, Spring JEE, Spring core.

List of Model Layer Technologies (Flw) to develop persistance logic

jdbc, hb, toplink, jdo, iBatis, jsf, webwork, ojb, EJB Entity Beans, open JTA...etc.

List of web application flw to develop view and controller layer logics

struts, jsf, webwork, Spring MVC, Tapstrey, ADF.

Q What is Pojo class, Pojo Java Bean, Domain class/Entity class/B.O class?

Pojo class: The ordinary class with out any special behaviour is called Pojo class.

⇒ The java class that can be compiled by using jdk libraries (no third party libraries) is called POJO class.

⇒ The java class that is not implementing any technology/framework api interfaces and not extending from any technology/framework api class is called POJO class.

⇒ POJO and POJI Model programming is all about developing the resources of the Application as ordinary class/interfaces. This allows is to develop application as light weight application

```
class Demo{
    ---
    ---
}
```

POJO class

```
class Demo implements Serializable{
    ---
    ---
}
```

```
class Demo implements javax.sewlets.Servlets{
    ---
    ---
}
```

```
class Demo extends Thread{
    ---
    ---
}
```

Not a POJO class :

```
class Demo extends GenericServlet{
    ----
    ---
}
```

```
class Demo extends Test{
    ---
    ---
}
class Test{
    ---
    ---
}
```

```
class Test extends HttpServlet{
    ---
    ---
}
```

"Demo" ⟶ non pojo class

"Test" ⟶ non pojo class

⇒ The simple an regular java interface without any special properties/ behaviour is called POJI.

⇒ The java interfaces that can be compiled by using jdk libraries is called POJI.

⇒ The java interfaces that is not extending from any technology/ framework specifies interfaces is called POJI.

```
interface Demo{
    ---
    ---
}
```

POJI

```
interfaces Demo extends Serializable{
    - - -
    - - -
}
```

```
interfaces Demo extends javax.sewlet.Sewlets{
    - - -
    - - -
}
```

Not POJI

⇒ NOTE: Framework is not invasive framework then it supports POJO and POJI Model Programs Application became lightweight application when its support POJO and POJI Model supports.

Java Bean

⇒ The java class is, developed with some standards is called Java bean. It is a helper class in application development to represent data in the form of objects & to send the over the network.

⇒ If the java class satisfies the following standards then it is called Java Bean.

i.e.,

a) Class must be public class & must implement java.io.Serializable (I)
b) Must have private properties (member variables)
c) Every property must have one setter method & one getter method
d) Must have explicitly placed or implicitly generated 0-param constructor (By java compiler)

NOTE: setter methods are useful to set data bean properties & getter methods are useful to read data from bean properties.

Example

StudentBean.java

```
Package com.nt.bean;
public class StudentBean implements java.io.Serializable{
   //Bean properties
  private int no;
  private String name;
  //0-param constructor
  public StudentBean(){
  } -- }
```


SRI RAGHAVENDRA XEROX
Software Languages Material Available
Beside Bangalore Ayyagar Bakery,
Opp. CDAC, Balkampet Road,
Ameerpet, Hyderabad.

Q. Difference between POJO class & JAVA Bean?

⇒ POJO class need not to have accessor methods (getter and setter) & need not to fulfill other standards of Java Bean. Java Bean must have accessor methods & must satisfy other standards.

⇒ Even though it is satisfied java bean standards its looks like POJO class.

CONCLUSION: Every Java Bean is a POJO class. But every class need not be, Java Bean.

Domain class:

⇒ The java class whose object can represents DB Table record values (Column values) is called Domain class | Business Object class | Entity class | Model class.

⇒ This class should have properties to hold the column values of DB Table record.

Domain Class: a) May or may not be a public class
b) May or may not implements Serializable (I)
c) Properties (member variables) can be private or public or protected or default variables.
d) Must have explicit or implicit 0-param constructor
e) Must have setter & getter methods for properties.

we can take java bean class as Domain Class.
we can say every Domain class is POJO class.

⇒ In HB the class whose object represents DB table records in ORMapping will be taken as Domain class.

⇒ Domain class will be mapped with Table, its properties will be mapped with DB table columns & its objects represents db table records having synchronization.

DB Table

~~Serializable~~

student
- sno (number)
- sname (varchar2)

Domain class

```
public class Student implements Serializable {
    private int no;
    private String name;
    public Student() { }
    //write setters & getters
}
```

HB programming can be done in 2 ways.

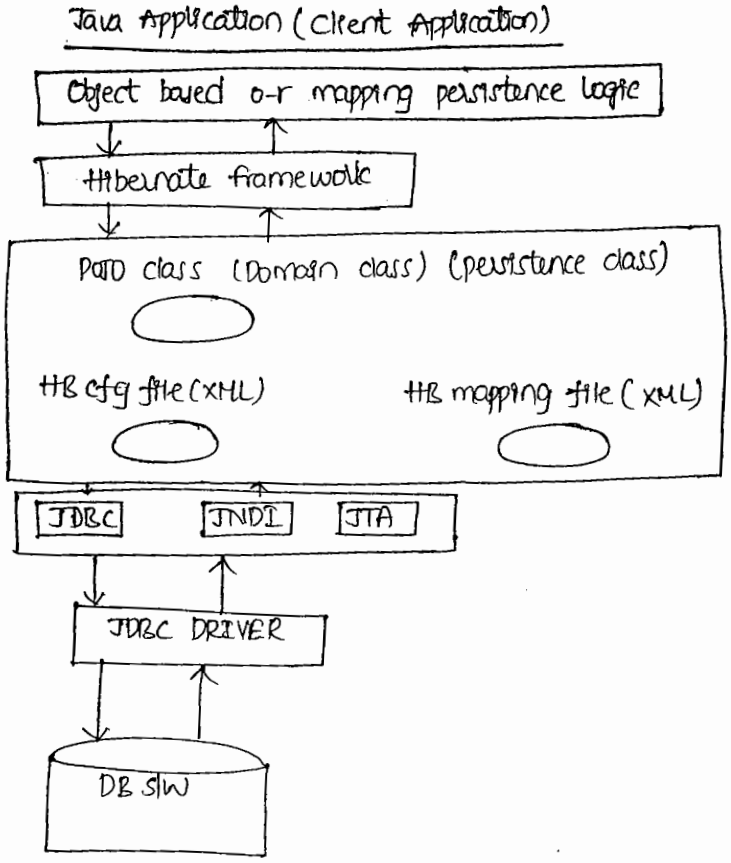
1) Using xml configuration
2) Using Annotation configuration

Using XML configuration

⇒ For this we need the following resources to develop objects based OR Mapping persistence logics. They are

1) HB configuration file (XML)
2) HB Mapping File (XML)
3) HB Persistence Class (.java file)

(Domain class | Business object class | Model class | Entity class)



HB Configuration file

⇒ It is the file contains information to establish connection with db fw & to provide instruction to hibernate fw. It contains information in the form of property names & values. The property names are fixed but values can be changed based on the DB fw, jdbc driver & application we are developing

⇒ This file contains 3 kind of details.

a) Connection Properties (TO establish the connection like JDBC properties)

b) HB Properties (TO instruct HB fw like enabling auto commit)

c) HB Mapping filenames

⇒ Any filename.xml can be taken as HB cfg file. If no filename is specified then the hibernate fw use "hibernate.cfg.xml" as default file name, JDBC driver class name, url, db username, db password are called JDBC properties.

Domain class / Property class

⇒ It is a class whose objects represents db table record in ORMapping persistence logic. It is a pojo class & also java bean.

⇒ Client application develop persistence logics by using objects this class.

⇒ In Hibernate Programming this class is called as Entity / Model / Business Object / Hibernate POJO class & etc.

⇒ In Hibernate Programming saving Domain class object means inserting Data of that object in DB Table as record,

⇒ Loading Domain class object means select record into object from dbtable.

⇒ Deleting Domain class object means removing record from dbtable i.e, represented by that object.

⇒ Updating Domain class object means updating the table record of db table i.e, represented by that object.

NOTE Domain class objects are actors which can play the roles of inserting / updating / deleting the records.

Hibernate Mapping File

⇒This file contain ORMapping cfg like mapping Domain class with DB Table, mapping Domain class properties with DBTable columns & etc..

⇒ HB framework uses this file content internally to generate jdbc code & sql queries that are required to perform persistence operations.

⇒ Any filename.xml can be taken as hibernate mapping file. There is no default name for this filename.

⇒ HB framework internally uses some .xml parses like dom4j & sax & process xml documents.

SAX : SIMPLE API FOR XML PROCESSING

DOM4J : DOCUMENT OBJECT MODEL FOR JAVA.

NOTE : XML Parser is a sw program or application that can validate & process the XML documents.

Client Application

⇒This is the application that contains HB persistence logic as object based ORMapping persistence logic.

⇒ This is client to db sw not the client to hb fw. This application can be any java, jee & java fw appln having hibernate ~~properties~~ persistence logic like standalone application, awt/swing appln, sewlet component/jsp component, strutsappln, jsf appln, spring app & etc..

⇒This appln activates HB fws, uses hibernate api & domain class object to develop object based ORMapping persistence logics.

⇒In one HB Application we can have one or more hibernate cfg files on 1 per DB file notation is <filename>.cfg.xml

⇒In a HB appln we can have one or more Domain classes on 1 per DBTable class.

⇒In a HB appln we can have one or more HBMapping files on 1 per DBTable basis file notation is <filename>.hbm.xml

Naming Conventions : (Recommendations)

⇒ If DBtable name is Employee then the recommended,

cfg filename : hibernate.cfg.xml

Domain class name : Employee / EmployeeBean / EmployeeDetails and etc..

HB Mapping filename : Employee.hbm.xml

⇒ In DBtable if primary key (p.k) is applied on one column is called singular PK constraint.

⇒ In DBtable if PK is applied on multiple columns then it is called composite primary key.

Q How synchronization takes place between objects and db table rows in hibernate?

1) In ORMapping Domain class objects represents dbtable records with synchronization i.e., if modification done in Domain class object that will reflect to DB table record & vice-versa.

2) Underlying ORM fw like hibernate is responsible for this synchronization.

Domain class

```
class student {
    int no;
    String name;
    DB Table || getters and getters
    columns
    - - -
}
```

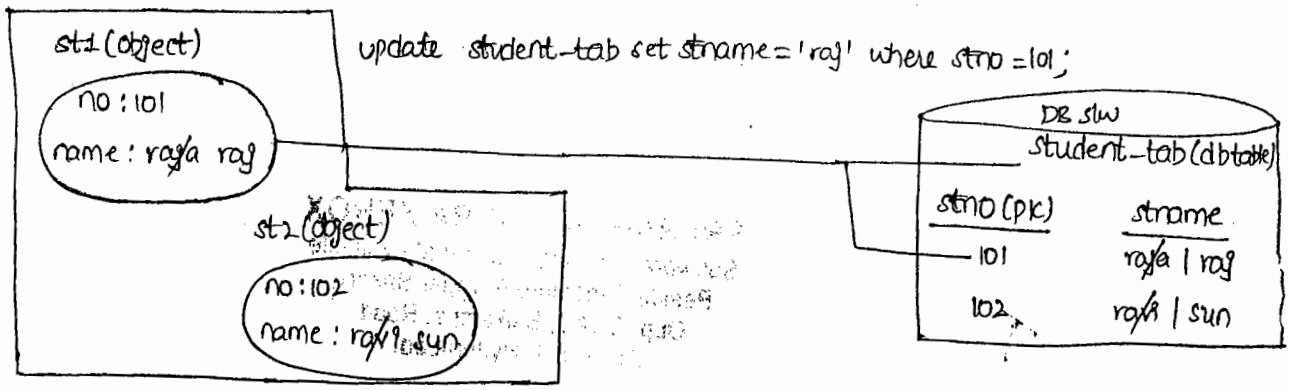
Hibernate Mapping file (XML)

Student ⟶ student_tab (dbtable)
(Java class)

no ⟷ stno | DB Table columns
name ⟷ stname |

IdField : no (singular Id field)

ClientAppln
(HB OR-Mapping Logic)



(1) : select * from student_tab where stno = 102;

3) Every Java Object is identified by JVM through its hashcode. Similarly every object of Domain class is identified by hibernate fw through identity value. In HBM file we cfg one or another property of Domain class as identifier field. That property value in every Domain class object acts as identity value & this value will be used HB fw as criteria value to perform synchronize b/w Objects and DB Table rows.

4) HB FW generates update Query having identity value as the criteria value to synchronize the modification done in Domain class object with DBtable row.

5) Similarly HB FW generates "select query" having identity value as the criteria value to synchronize the modification done in DBtable row to object as shown above.

Q. What kind of analyzation on strategies should be followed by programmer while cfg identify field in mapping file?

A ⇒ If only one property Domain class is cfg as "Idfield" then it is called as "singular identity field".

⇒ If more than one property of Domain class is cfg as "Idfield" then its called as "Composite Identity field".

case(1): If DB Team gives DBTable having "singular primary key column" then take that column related property in Domain class & configuration it is as "Singular Idfield" in mapping file using <id> tag.

case(2): If DB Team gives DB Table having "Composite Primary key column" then take that columns related multiple properties in Domain class & cfg them as "Composite Idfield" in mapping file using <composite-id> tag.

case(3): If DB Team gives DB Table with out constraints then reject that table to use HB. programming.

⇒The 3 important object of HB programming in client Application Development.

a) Configuration object
b) HBSessionFactory object
c) HBSession Object

a) Configuration Object: When this object is created HB Fw will be activated based on the Jar files that are added to classPath or BuildPath.

⇒This object also reads & verifies given cfg file & mapping file entries and stores the info as "Inmemory metadata" at ram level in the form of "HashMap Object". Now onwards if configuration file, mapping file info is mapped required during the execution of appln it will collect from that "In-memory metadata (HashMap)" instead of going to those physical XML files.

Sample code

```
//Activate HB Fw
Configuration cfg = new Configuration();
//verifies the and read both cfg & mapping file data & stores the data as in memory metadata.
cfg = cfg.configure("/hibernate.cfg.xml");
```

cfg object means it is the object of org.hibernate.cfg.Configuration class.

NOTE If cfg.configure() is called with argument then it takes hibernate.cfg.xml as default hibernate.cfg.xml file. If the same method is called with argument then it takes specified filename as Hibernate cfg file.

cfg = cfg.configure(); → Takes hibernate.cfg.xml as cfg filename
cfg = cfg.configure("/myfile.html") → Takes myfile.xml as cfg filename.

b) SessionFactory object

1) It will be created based on cfg object. This object holds connection properties, hibernate cfg file information, mapping file information & other details so it is called heavy weight object in HB programming.

2) It is Immutable object. i.e., once the data is placed in the object it can't be modified.

3) It is Thread Safe Object because all immutable objects are Thread Safe Objects by default. (Eventhough multiple threads are started on this object data will not be corrupted).

4) It is a factory to create HB session objects

5) It represents jdbc con pool i.e, created based hibernate cfg file jdbc properties.

NOTE: JDBC con pool contains set of readily available jdbc con objects

6) It is object of java class that implements org.hibernate.SessionFactory (I).

code

```
SessionFactory factory = cfg.buildSessionFactory();
```

c) Session Object

1) SessionFactory object gets one connection object from its jdbc connection pool & uses that object to create HB session object.

2) This object open connection blw java appln & DB slw through HB flw.

3) It is the object of java class that implements org.hibernate.Session (I)

4) It is the object for programmer to give persistence instruction of HB flw using Domain class obj.

5) It is not Thread Safe object by default.

6) This object is no way related to HttpSevlet object for Sevlet Programming.

code

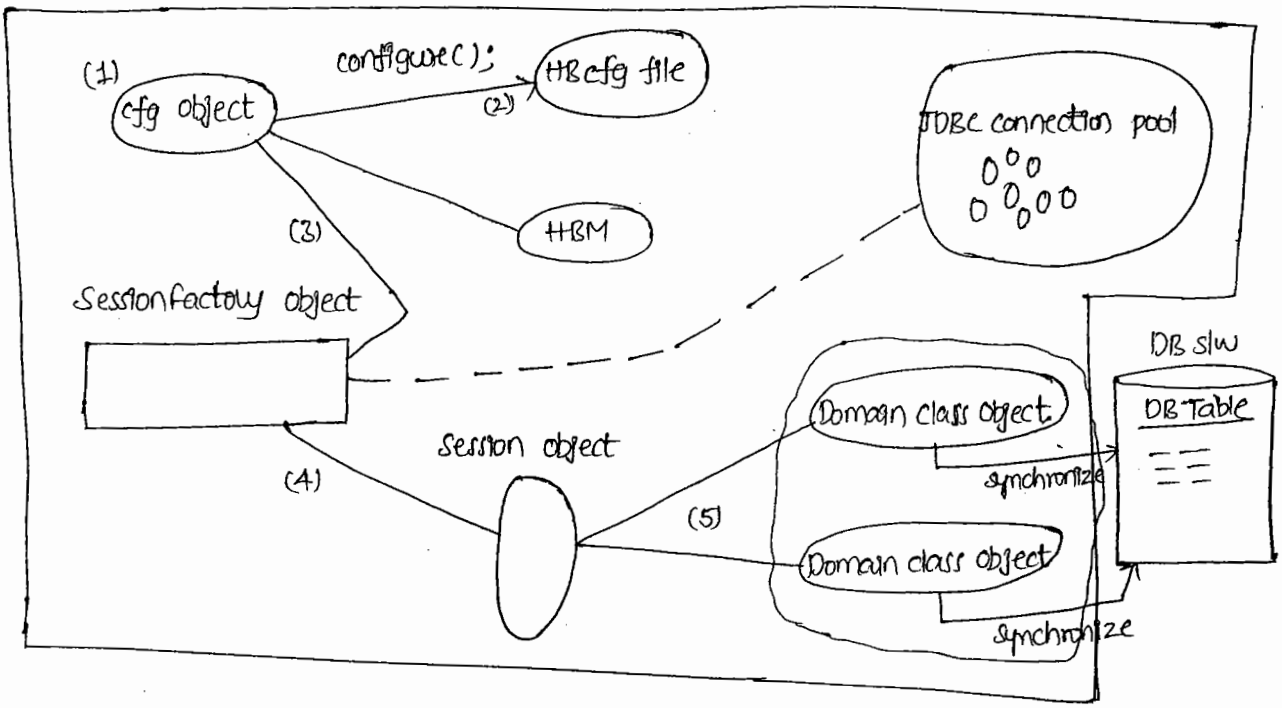
```
Session ses = factory.openSession();
```

⇒ In a project that talks with one DB slw using HB

a) Create 1 cfg object, 1 SessionFactory Object

b) Create multiple Session object if that project contains multiple client/Appln interacting with DB slw (on 1 per client/Appln basis)

Hibernate Appln



// Activate HB Fw

Configuration cfg = new Configuration(); ⟶ (1)

// Read & verify HB cfg file, HB Mapping file entries

cfg = cfg.configure("/hibernate.cfg.xml"); ⟶ (2)

// create HB Session factory object

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

// create Session object

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

// perform persistence operations using Domain class objects

ses.save(student object) ⟶ (5) // Here student object is the Domain class object

⇒ SessionFactory Object is heavy weight because it contains more data.

⇒ Session object is lightweight object.

Bulk Operations

1) HQL (Hibernate Query Language)
2) Native SQL queries (Direct Sql Queries)
3) Criteria API (can manipulate more than 1 record at a time)

single Row operations

1) Can manipulate 1 record at a time
2) Can following on HB session object for this

ses.save(-) : Insertion
ses.persist(-) : Insertion
ses.update(-) : updation
ses.delete(-) : delete
ses.saveOrUpdate(-) : insertion | deletion
ses.merge(-) : insertion | updation
ses.load(-) : selection
ses.get(-) : selection
and etc..

Example

⇒ FirstHBAppln to save object (insert record) in DB Table.

Software setup:

Hibernate FW : HB4.3.5
Eclipse IDE : Luna or Kepler (JEE mode)
Oracle 10g | 11g | 12c Data base software
JDK1.7 SW

Steps)

create DB table in oracle

Employee
- Eid number(5) primary key
- FirstName varchar2(20)
- LastName varchar2(20)
- Email varchar2(20)

Step(2)

⇒ Lauch Eclipse IDE by choosing workspace folder (Nt HB27)

step(3)

⇒ create java project in eclipse IDE having the following resources

HBProj1

```
|-> src
    |->com.nt.cfgs
        |-> hibernate.cfg.xml (HB cfg file)
        |-> Employee.hbm.xml (HB mapping file)
    |-> com.nt.domain
        |-> EmpDetails.java (HB Entity/ Domain/ POJO class)
    |-> com.nt.test
        |-> ClientApp.java (standalone app as client having o-r-mapping logic
                            to intelact with DB slw)
```

EmpDetails.java

```
package com.nt.domain;
public class EmpDetails implements java.io.Serializable{
    private int no;
    private String name;
    public EmpDetails(){}
```

//full constructor → right click → source → generate constructor using fields → select all properties → ok.

on Eclipse IDE

```
//write setter and getter methods
```

→ right click → source → generate setter and getter methods

```
//place tostring
```

⇒ right click → source → generate toString() → select all →

```
}
```

Employee.hbm.xml

```
<!DOCTYPE ---- copy paste ---- |hibernate-mapping-3.0.dtd ">

<hibernate-mapping>
<class name="com.nt.domain.EmpDetails " table="scott.Employee">
<id name="no" column="EID"/>
<property name="fname" column="FNAME"/>
<property name="lname" column="LNAME"/>
<property name="mail" column="EMAIL"/>

</class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="connection.username">scott</property>
<property name="connection.password">tiger</property>

<mapping resource="com/nt/cfgs/Employee.hbm.xml"/>

</session-factory>
</hibernate-configuration>
```

Step(4)

⇒ Add the following jar-files (libraries) to the build path of the project

⇒ all the jar files of <HB 4.3.5_home>/lib/requiredfiles

⇒ ojdbc14.jar / ojdbc6.jar

step(5)

⇒ Develop the client App.

clientApp.java

```
public class clientApp{
    public static void main(String args[]){
        Configuration cfg=new Configuration();
        cfg = cfg.configure("|com|nt|cfgs|hibernate.cfg.xml");
      SessionFactory factory = cfg.buildSessionFactory();
      Session ses = factory.openSession();
     Transaction tx=null;
     try{
         tx=ses.beginTransaction();
      EmpDetails details = new EmpDetails (101, "raja", "rao", "rajarao@gmail.com");
        ses.save(details);
        tx.commit();
      }
    catch (Exception e)
      {
        tx.rollback();
       }
    ses.close();
    factory.close();
  }
}
```

step(6) Run the clientApp

Observations : Placing <!DOCTYPE --> in xml file, HB is mandatory but same is optional in web.xml.

⇒ xml parser is a sw application/program that can read & process xml documents.

Ex: SAX, DOM, DOM4J & ETC..

⇒ Configuration of object of HB internally uses both SAX, DOM4J xml parser to read & process HB cfg, mapping file entries & to create "In Memory-Meta Data".

⇒ Transaction is a unit that can be committed (or) rollback. Every transaction contains 3 operations.

a) Begin Transaction
b) proceed Transaction
c) commit (or) rollback transaction (tx)

⇒ All non-select persistence operations through HB must executed as Transactional logics because HB makes underlying jdbc code to execute the generated SQL queries in DB slw by disabling auto commit mode on DB slw.

⇒ While working with java technologies the implementation classes for api interfaces will be given by vendor companies who creates slw's based on technologies.

⇒ While working with language, flw's the implementation classes for api interfaces will be given by language or flw vendor companies itself.

⇒ In HB Sessionfactory object means it is the object of flw supplied java class that implements.

org.hibernate.Sessionfactory (I) i.e.,
org.hibernate.internal.SessionfactoryImpl

⇒ HB Session object means it is the object of flw supplied java class that implements org.hibernate.Session (I). ie, org.hibernate.internal.SessionImpl.

⇒ By default HB internally uses Jdbc Transaction Management that means when

session.beginTransaction() → It calls con.setAutoCommit(false)
for session.commit() → It calls con.commit();
for tx.rollback() → It calls con.rollback();

⇒ Session object keeps track of all persistence instructions by maintaining given objects in buffer once tx.commit() is called the HB flw internally uses jdbc+sql queries to perform persistence operations on db slw.

⇒ Ex: When ses.save(-) methods is called the record will not be inserted in DB table. Actually the session object takes given Domain class Object & keeps that object in buffer/cache.

⇒ Once tx.commit() is called the HB flw uses jdbc code + insert SQL query to insert record in DB Table.

⇒ To make HB flw showing the internally generated SQL Query as log message by placing "show-sql" property in HB cfg file.

In hibernate.cfg.xml

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

(It is HB property)

⇒ HB internally uses JDBC prepared statement object while performing persistence operation

Date 27-04-2015 (24-04-2015 to 26-04-2015 class is not there)

ses.close() : → close the session by releasing jdbc connection. Instead of this we can also call.

ses.disconnect() method.

⇒ When HBSession is closed all the active HB domain class object become inactive objects.

factory.close() : → Releases all the resources that are associated SessionFactory like jdbc con pools, caches, etc.. We must ensure that all sessions are closed properly before closing SessionFactory.

Q What is the use of hibernate.dialect property in hibernate cfg file?

(or)

What is hibernate dialect?

A It is an optional property of hibernate cfg file but it is recommended property to place.

This classname will change based on the DB sw & its version we use...

Oracle any version : org.hibernate.dialect.OracleDialect
Oracle 8i version : org.hibernate.dialect.Oracle8iDialect
Oracle 9i version : org.hibernate.dialect.Oracle9iDialect
Oracle 10g & later : org.hibernate.dialect.Oracle10gDialect
MySQL any version : MySQLDialect
MySQL5 : MySQL5Dialect
PostgreSQL (any version) : PostgreSQLDialect
PostgreSQL8.1 : PostgreSQL8.1Dialect
PostgreSQL9 or later : PostgreSQL9Dialect & etc...

Ex: <property name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>

NOTE

Most of the times the HB Fw ~~internally~~ automatically picks up the Dialect class name. If failed then set explicitly.

Advantages

a) Helps HB fw to decide the destination DB sw to communicate.
b) Helps HB fw to generate optimised SQL as required for underlying DB sw.
c) Assigns sensible, intelligent default values for HB properties of cfg file based on underlying DB sw we use even though they are not placed in cfg file.

ses.save(-) not only gives persistence instructions to HB fw to insert record it also generates serializable object as the identity value based on the identity field that is cfg using <id> tag.

All wrapper classes are serializable classes.

Prototype of ses.save(-) method

```
public Serializable save(Object obj)
```

```
Transaction tx=null;
  try{
      tx=ses.beginTransaction();
      EmpDetails details = new EmpDetails();
      int id=(Integer)ses.save(details) ⟶!! (auto unboxing is used here)
      S.O.P("Generated id : "+id);
      tx.commit();
    }
  catch(Exception e){
      tx.rollback();
    }
```


⇒ses.save(-) returns wrapper class object as serializable object representing the identity value.

⇒ses.save(-) method first generate identity value irrespective of object is saved or not in DB table (record is not inserted or not in DB Table) without identity value no domain class object can participate in persistence operations.....

SAVING OBJ / INSERTING RECORD USING SES.PERSIST(-) METHOD:

```
Transaction tx=null;
  try{
      tx=ses.beginTransaction();
      EmpDetails details =new EmpDetails();
          ses.persist(details);
    }     tx.commit();
```

```
catch(Exception e){
    tx.rollback();
  }
```

Q What is the difference between ses.save(-) & ses.persist(-) method?

A ses.save(-) returns the generated identity value but ses.persist(-) doesn't return the generated identity value. Because return type is void.

```
public Serializable save(Object obj)
public void persist(Object obj)
```

⇒ All the methods that are invokable on session object performs persistence operations only by taking the identity value of given domain class object as the criterion value.

Deleting the object/ Deleting record using ses.delete(-) method:

```
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    EmpDetails details =new EmpDetails();
        details.setNo(102);
        ses.delete(details);
        tx.commit();
    S.o.p(" record/ object are deleted ");
}
catch(Exception e){
    tx.rollback();
}
```

Prototype: public void delete(Object object)

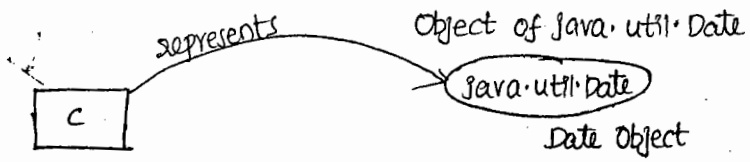
Knowing about java.lang.class

⇒ In a running appln to hold numeric values we use int, float etc.. Data type variables to hold string values we use of object of java.lang.String, similarly we can use object of java.lang.class to hold class or interface.

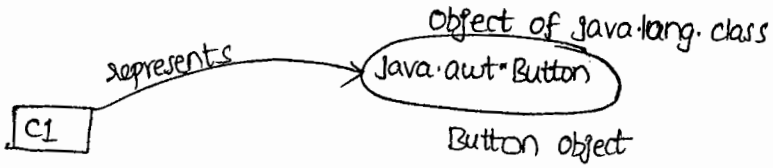
⇒ In a running Java Appln the object of java.lang.class can represent a class or interface.

a) Using Class.forName(-) method

Ex: Class c = Class.forName("java.util.Date");



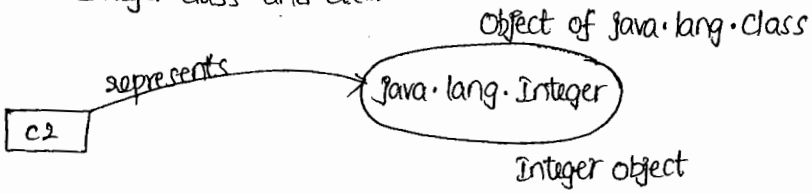
b) Using getClass() method of java.lang.Object class



Ex: Button b = new Button("OK");
Class c1 = b.getClass();

c) Using "class" property

Class c2 = Integer.class and etc..



NOTE

1) class, length are built-in properties of java
2) main, garbage collector are the built-in threads of java
3) this, super are built-in reference variable in java.

Loading object from DB Table

⇒ This is nothing but select a record from DB Table & storing that record in a object of Domain class.

⇒ For this 2 methods are given.

1) ses.load(-) method (performs lazy loading)

public Object load(Class class, Serializable id)

2) ses.get(-,-) (Performs eager/easly loading)

public Object get(Class class, Serializable id)

Select operations in HB can be executed as non-transactional operations.

```
//Loading object/record using ses.get(-) method
EmpDetails details = (EmpDetails) ses.get(EmpDetails.class, 1001);
//Display object data
if(details != null)
{
   S.O.P(details.getNO() + "  " + details.getfname() + "  " + details.getLname() + "
    *  " + details.getMail() );
}
else
{
   S.O.P(" Record object not found ");
}
```

Loading object/record using ses.load(-,-) method :

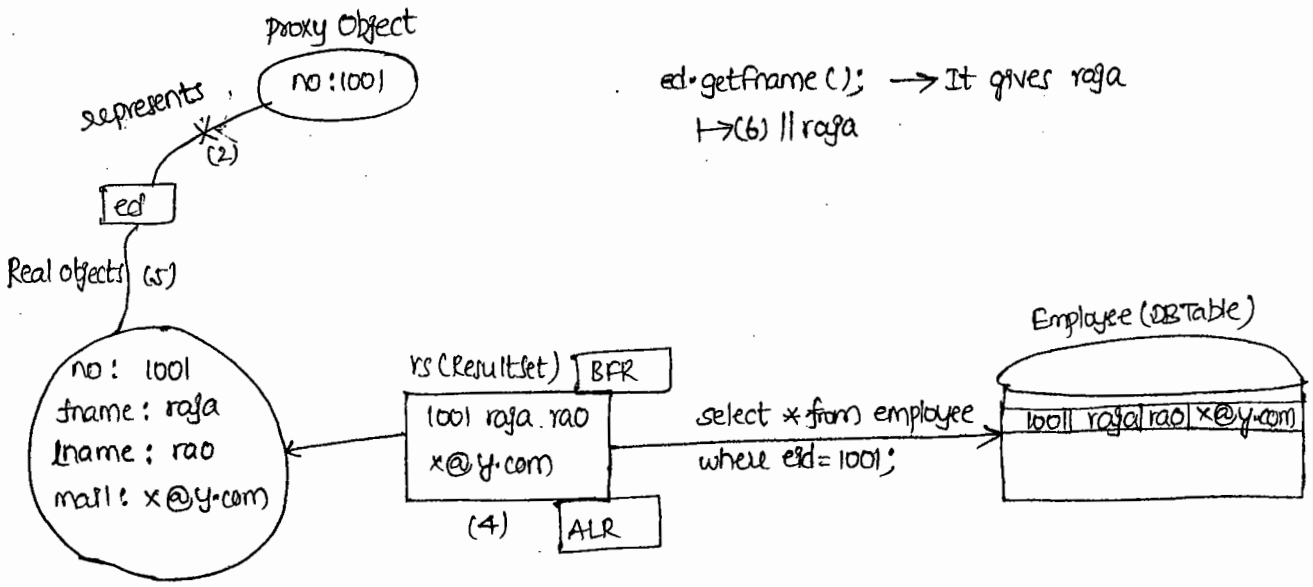
```
// load object or select a record
EmpDetails ed = (EmpDetails) ses.load(EmpDetails.class, 1001);
// Display record
S.O.P(ed.getNO() + "  " ed.getfname() + "  " + ed.getLname() + "  " + ed.getMail());
```

⇒ ses.get(-,-) perfoms eager/early loading that means it will get record from DBTable to Domain class object irrespective of weather that object is used or not in the Application.

⇒ ses.load(-,-) perfoms lazy loading that means it will get record from DBTable to Domain class object only when we start the utilization of Domain class object in the Application. This method also returns proxy object to represent the presents in the Application until real object comes as shown below.

NOTE: The Dummy object/Domain Duplicate object that represents until real objects comes is called proxy object.

```
EmpDetails ed = (EmpDetails) ses.load (EmpDetails.class, 1001);
                                    (1)
```



ed.getfname(); → It gives raja
↦(6) // raja

1) Appln calls ses.load(-,-) method
2) Hibernate creates & returns proxy object having identity value the reference variable "ed" represents this proxy object
3) Appln calls non-identifier method on "ed" like ed.getfname();
4) HB FW uses select SQL Query to get record from DBTable & to store that record by creating real object of Domain class
5) The Reference variable 'ed' stops refering proxy object & starts refering object.
6) Appln gets ed.getfname(-) result from real object.

FAQ: What is difference b/w ses.load(-) & ses.get(-)

ses.load()	ses.get()
1) Lazy loading	1) eager/ early loading (check by commenting System.out.println)
2) creates Proxy object	2) Does not creates Proxy object
3) Creates 2 objects for Domain class to load one record (proxy + real object)	3) Creates only one object for Domain class to load 1 record
4) Throws Exception if record not found to load.	4) Does not throw any exception if record not found to load.

5) Not suitable, call this method by assuming record is available

6) Use this method when load object/ record will be used lately.

7) Multi-layer Appln

5) Suitable to check whether record is available or not.

6) Use this method when loaded by object/ record will be used immediately.

7) Suitable single layer Appln.

HB Fw can maintain Domain class object in 3 states

a) Transient state
b) Persistent state
c) Detached state

a) Transient state

⇒ Never Persistence not associated with session does not contain identity value & does not represent record in DB Table.

b) Persistent state

⇒ Associated with unique state contains identity value & represents DB Table records. Maintains Synchronization with DB Table record. IN O-R Mapping persistence logic we use persistent objects to develop persistance logic.

c) Detached state

previously persistent, not associated with session.

⇒ It contains (Detached state) identity value but does not represent DB Table record & does not maintain synchronization with DB Table record. When session is closed all persistent objects become "Detached state objects".

Ex:

```
EmpDetails ed=new EmpDetails (101,"x","y","x@y.com");
  Transaction tx=null;
try{
    tx=ses.beginTransaction();
      ses.save(ed);
    tx.commit();  ——→ // persistent state object (ed become)
  }
catch(Exception e){
  tx.rollback();
}
ses.close();  ——→ //ed become Detached state object.
```

Updating Record/Object:

We can use 3 approaches for this:

1) By creating object for Domain class having existing identity value & new Data.
2) By loading object & modifying the object in a Transaction.
3) By loading object & by calling ses.update(-) method.

Approach(1) (using ses.update(-) method)

```
EmpDetails ed = new EmpDetails(1002, "rajesh", "rao", "x@gmail.com");
```

(existing value) (New value)

```
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.update(ed);
    tx.commit();
}
catch(Exception e){
    tx.rollback();
}
```


NOTE

Preparing object having existing id value & new data is very complex.

Proto Type: public void update(Object obj)

Approach(2) (By modifying loaded object in a Transaction)

```
EmpDetails ed=(EmpDetails) ses.get(EmpDetails.class,1003);
if(ed==null)
{
  S.O.P("record/object not found");
}
else{
    Transaction tx=null;
    try{ tx=ses.beginTransaction();
```

This modification will be synchronized with DB Table record

```
ed.setMail("x@yahoo.com");
```

→ No ses.update(-) method here

```
        tx.commit();
        }
        catch(Exception e){
            tx.rollback();
        }
    }
```

Note 1

⇒ ses.get(-) / ses.load(-) methods returns persistent state object. If we modify these objects in a transaction then those modifications will be synchronized automatically with DB Table record.

Q) can update record without calling ses.update(-), ses.save(-) or update(-) or ses.merge() methods?

A) yes, refer above code.

Approach (3)

⇒ By modifying loaded object & by calling ses.update(-) method.

Ex: same as approach 2 but call ses.update(ed); before tx.commit(); method

NOTE

ses.save(-), ses.persist(-) methods throw Exception if object is not saved.

But ses.update(-) method does not throw Exception eventhough object is not updated.

Saving or updating using one saveOrUpdate(-) method:

```
EmpDetails ed = new EmpDetails(1003, "rajesh", "rao", "x@y.com");
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.saveOrUpdate(ed);
    tx.commit();
}
catch(Exception e){
    tx.rollback();
}
```

prototype: public void saveOrUpdate(Object object)

Q What is the Difference b/w ses.update(-) & ses.saveOrUpdate(-) methods?

A ⇒ ses.update(-) method just updates the record by using update sql query.

⇒ ses.saveOrUpdate(-) method updates the ~~record~~ object of the record is already available. Otherwise it will insert the record/ object for this it internally generates select query & insert (or) update query.

Attaching HB source code & API Docs with Eclipse Project:

Right click on Project → Build Path → configure buildpath → Expand Hibernate core 4.3.5.final.jar → source attachment → edit → External location → <HB_home/project/hibernate-core 4.3.5.final> → ok

javadoc location → Edit → javadoc in archieve → External file

→ Archieve path : Original HBhome zip file

Path within archieve : HB_home.zip/documentation/javadocs → validate → ok

```
//Saving or updating object/record using ses.merge(-)
EmpDetails ed = new EmpDetails (1003, "rajesh", "rao", "x@gmail.com");
    Transaction tx=null;
     try {
          tx= ses.beginTransaction();
          EmpDetails ed1 = (EmpDetails) ses.update(ed);
            tx.commit();
            s.o.p(ed1);
          }
       catch( Exception e) {
          tx.rollback();
          }
```

Prototype : public void merge (Object obj)

Q) What is the Difference b/w ses.saveOrUpdate(-) & ses.merge(-) method?

ses.saveOrUpdate(-)	ses.merge(-)
1) Both Methods are same (functionality wise)	
2) It does not return any object representing the record that will be inserted/updated because return type is void.	2) Returns object representing the record that will be inserted/updated. (persistent state object).

Deleting object/record

⇒ There are 2 Approaches

1) By using ses.delete(-) method directly.

⇒ This is not recommended approach because we can't confirmation about the deletion of record.

2) By loading object & by calling ses.delete(-) method.

⇒ This is recommended approach because we can get confirmation about deletion of record.

Approach(1)

```
EmpDetails ed=new EmpDetails();
        ed.setNO(1003);
Transaction tx=null;
 try{
        tx=ses.beginTransaction();
          ses.delete(ed);
          tx.commit();
        s.o.p("record/object deleted");
    }
catch(Exception e){
        tx.rollback();
    }
```

Prototype: public void delete(Object obj)

Approach(2)

```
EmpDetails ed=(EmpDetails)ses.get(EmpDetails.class, 1003);
        if(ed==null)
        {
          S.o.p("record/object not found");
        }
       else{
           Transaction tx=null;
           try{
               tx=ses.beginTransaction();
                ses.delete(ed);
                tx.commit();
             S.o.p("Record/object deleted");
              }
           catch(Exception e){
                 tx.rollback();
               }
       }
```


NOTE Approach(2) is recommended

Q How many ways are there to supply hibernate configuration (cfg) properties to our Appln?

a) using xml file (hibernate.cfg.xml file (or) some other file) (Best Practice)
b) using properties file (hibernate.properties file (or) some other file)
c) using ~~properties~~ programmatic approach (By placing Java code directly in client Appln)

b) using hibernate properties:

step(1): keep first appln ready
step(2): Remove HB cfg file
step(3): Add hibernate.properties file in "src" folder of the project.

In hibernate.properties (code as shown below)

```
hibernate.connection.driver_class = oracle.jdbc.driver.OracleDriver
---
---
hibernate.show_sql=true
```

step(4): write the following code in client Appln to get HB Session object.

Code:

```
Configuration cfg = new Configuration();
    cfg.addFile("src/com/nt/cfgs/Employee.hbm.xml");
SessionFactory factory = cfg.buildSessionFactory();
   Session session = factory.openSession();
```

Limitations

1) The word hibernate is mandatory in property file.
2) Specifying mapping file name in clientAppln comes under hard coding.

c) Programmatic Approach

step(1) : Keep first appln ready with out cfg file & properties file.

step(2) : Write the following code in client Appln to create session object.

```
Configuration cfg = new Configuration();
  cfg.setProperty("hibernate.connection.driver_class", "oracle.jdbc.driver.OracleDriver");
  = = =
  - - -
  cfg.addFile("src/com/nt/cfgs/Employee.hbm.xml");
SessionFactory factory = cfg.buildSessionFactory();
Session ses = factory.openSession();
```

Limitations

1) The word hibernate is not optimal in property names.
2) Since we are hard coding values directly in client Appln we loose the flexibility of modifications

NOTE : We get this flexibility of modifications through xml file & properties file.

Q) What happens if we place all the above 3 approaches in 1 appln to supply HB cfg properties?
A) If cfg.setXxx(-) methods are called after cfg.configure(-) method then the cfgs done in programmatic approach will override remaining 2 approaches data. Actually, in the above discussion xml file configurations override property file cfg & programmatic approach cfgs override xml file configurations (cfgs).

Q can we Develop HB appln without cfg file?
A Possible by using properties file, programmatic approach but not recommended.

Q How many ways are there to perform O-R Mapping configurations in HB Appln.

A 1) Using xml File (Mapping xml files)
2) Using Mapping Annotations (will be placed directly in Domain class)

⇒Logical DB is a logical partition of physical db slw & it will be created on one per project basis.

⇒ Every logical db contain views, Tables, PL/SQL procedures & functions etc..

⇒ The DBA of db team creates these logical DB...

Physical DB slw

Logical DB1 (for proj1) Name : NtDB1	Logical DB2 (for proj2) Name : NtDB2
Logical DB3 (for proj3) Name : NtDB3	
DB Engg	

⇒In Real time the GUI DB tools will be used a lot to perform DB operations. These tools must be installed seperately. They are Toad for Oracle, Toad for MySQL, SQL Yog for mysql and etc..

Procedure to create Logical DB having DB Table with records in mySQL using SQLyog.

step(1) : Lauch SQL yog & create a connection
new → connection
name : con1
username : root
password : root
connect

step(2) : Create Logical DB with table having records...

right click on to root (on localhost → create DataBase

Enter new Database name : HBNtDB4

right click on HBNtDB4 → create table

Field Name :
EID :
First Name :
Last Name : Email :

create table → Enter table Name : Employee

Right click on employee table → Insert/Update Data

⇒ Type 4 mechanism jdbc driver for Mysql is called Connection (or) Jdbc driver --- @)

The details are : →

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

(or)

com.mysql.jdbc.Driver

Jdbc Url : jdbc:mysql:///<logicalDB> (for local mysql)

(or)

jdbc:mygql://<host>:<portno>/<logicalDB> <for remote mysql>

jar files : mysql-connector-java-5.1.6.jar

(download it seperately)

hibernate Dialect : org.hibernate.dialect.MYSQL5Dialect

Procedure to make first Appln talking with MySQL DB sw.

step(1) : Make sure that hibernate.cfg.xml file is having following content.

Hibernate.cfg.xml

```
<hibernate-configuration>
 <session-factory>
  <property name = "connection.driver_class"> com.mysql.jdbc.Driver</property>
                = "connection.url"> jdbc:mysql:///HIBNTDB4 </property>
                = "connection.username"> root </property>
                = "connection.password"> root</property>
                = "show_sql"> true </property>
                = "hibernate.dialect"> org.hibernate.dialect.MYSQL5Dialect</property>
 <mapping resource = "com/nt/cfgs/Employee.hbm.xml"/>
 </session-factory>
</hibernate-configuration>
```

step(2) : add mysql-connector-java-5.1.6.jar file to build path

step(3) : Run client Appln.

First Level Cache in HB

1) Cache/Buffer is a temporary memory that holds data from temporary period. In client-sewer environment the cache at client side holds data given by server & uses that data across the multiple same requests to reduce network rounds trips between client & server.

Q What is the difference between pool & cache?

⇒ pool holds set of semantics

⇒ cache holds set of different items.

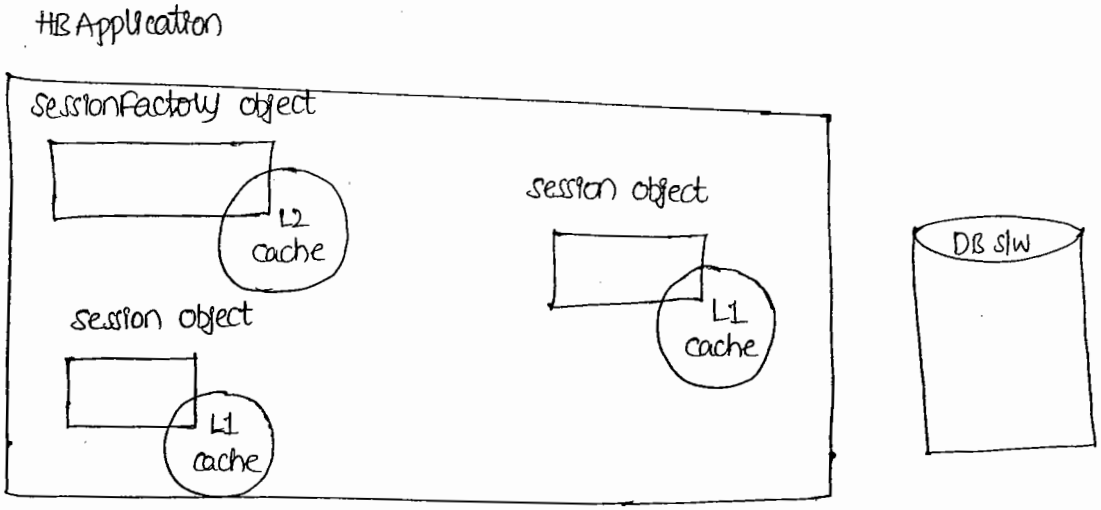
⇒ In client-server environment the cache at client side should emptied at regular intervals to get update to date results from server.

HB supports 2 level of caching

a) First level cache/session cache/L1 cache

b) Second level cache/session factory cache/L2 cache

⇒ L1 cache associated with session object on 1 per session object basis. It is a built-in cache of every session object. The data of one session cache cannot be used in another session cache.



⇒ HB uses L1 cache in 2 angles:

1) When we ask HB to load same object/record from DB for multiple times it will load that object/record only 1 time from DB s/w & keeps that object in L1 cache to use it for multiple times. This avoids multiple SQL queries generation object creation & also reduces Network round trips...

```
EmpDetails ed = (EmpDetails) ses.get(EmpDetails.class, 1234);
    //gets from DB & keeps in L1 cache.
        S.O.P(ed);
```

```
EmpDetails ed1 = (EmpDetails) ses.get(EmpDetails.class, 1234);
            //gets from L1 cache
           s.o.p(ed1);
EmpDetails ed2 = (EmpDetails) ses.get(EmpDetails.class, 1234);
            //gets from L2 cache
            s.o.p(ed2);
```

⇒ Hits the DB slw only for 1 time by generated 1 select query & uses that object for multiple times by keeping L1 cache.

⇒ In a transaction if persistant state object is modified for multiple times instead generating multiple update queries to reflect the modifications in DB tables. It will keep track of all the modifications of that object L1 cache till the end of transaction. Once the transaction is committed it generates only update query having all the modifications to reflect modifications in DB Tables....

```
EmpDetails ed = (EmpDetails) ses.get(EmpDetails.class, 1234);
    Transaction tx=null;
      try{
          tx = ses.beginTransaction();
           ed.setMail("x1@y1.com");
           ed.setMail("x2@y2.com");
           ed.setLname("rao1");
           tx.commit(); ---> //Generates update query here having all modifications
        }
      catch(Exception e){
           tx.commite);
          }
```

To control L1 cache

(a) ses.evict(ed) : Remove 1 object from L1 cache

(b) ses.clear() : Remove all objects from L1 cache

(c) ses.close() : closes L1 cache along with session

Q Can we perform non-select persistent operations in HB without taking Transaction environment?

A Possible, but not recommended because it works only in some DB sw.

NOTE: use ses.flush() method to synchronized persistent state Domain class object data with DB Table record.

```
EmpDetails ed = new EmpDetails(1001, "raja", "rao", "x@y.com");
ses.save(ed);
ses.flush();
ses.close();
factory.close();
```
→ //Try work with Oracle setup.

NOTE: when tx.commit() is internally uses ses.flush() method.

Q Is it possible to have primary key constraint on the DB Table column i.e, Mapped by Identity field property of Domain class?

A optional, but it is recommended to have either pk constraint on unique key constraint. Because without pk constraint we will get synchronization issues towards duplicate values.

Working with hibernate.hbm2ddl.auto property:

⇒HB allows to create/alter DB tables based on the cfgs (configuration) done in HB mapping file. This option is useful to add new columns in the middle of project development (or) in the production environment of the project.

⇒This is the also useful to create DB Tables dynamically new in DB sw if the db sw is changed in the middle of the project development or in the production environment dynamically.

⇒Do not use this option to design DB Tables of project because this will be taken care by DB Designing Team before coding of the project.

* To work with this facility we need to "hibernate.hbm.2ddl.auto" property having one of the following 4 values. They are

(a) create
(b) update
(c) validate (default)
(d) create-drop

(a) create

1) Always creates new DB table & also drops db table if DB table is already available.

2) This tool internally uses "schema expot tool"

3) To control column Dimensions we can use length, unique, not-null attributes in <property-tags>.

Ex:

In mapping-file

<property name="mail" column="email" unique="true" not-null="true" length="10" type="string">

4) Example in cfg file

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

Note 1) The column is specified in <id> tag automically gets PK constraint.

Usage:

useful to clean up DB Table data when ever project is restarted.

Ex:

When Railway Reservation application is restarted. The passengers info will be cleaned up.

It will be done for every 2 years.

(b) update:

1) This is multipurpose value. To create new DB table if db table is not already available. Uses the existing DB Table. If db table is already available alters existing db table by adding new column if new property is cfg in mapping file.

2) It can't alterable by dropping existing cols (or) by changing the data types of existing cols.

3) It internally uses "schemaupdate" tool for this operation.

Ex

In HB cfg file

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

4) To make HB adding new column in existing DB Table.

step(1) Add the new property in Domain class along with setter & getter method in EmpDetails.java

//private String xyz;

setter & getter methods

step(2) : cfg this property in mapping file

In Employee.hbm.xml

<property name="xyz" column="xyz"/>

5) It is most regularly used value of "hbm2ddl.auto" property.

Usage

Useful to add new column in DBtable in the middle of the project development or in the middle of production environment...

(c) validate

1) It is default value for "hbm2ddl.auto" property if no value is specified.
2) It checks whether DB Table is available according to mapping file cfgs or not.
   If not available it throws exception (HibernateException)
3) It internally uses "schema validator" tool.

Ex

In hibernate.cfg.xml

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

(d) create-drop

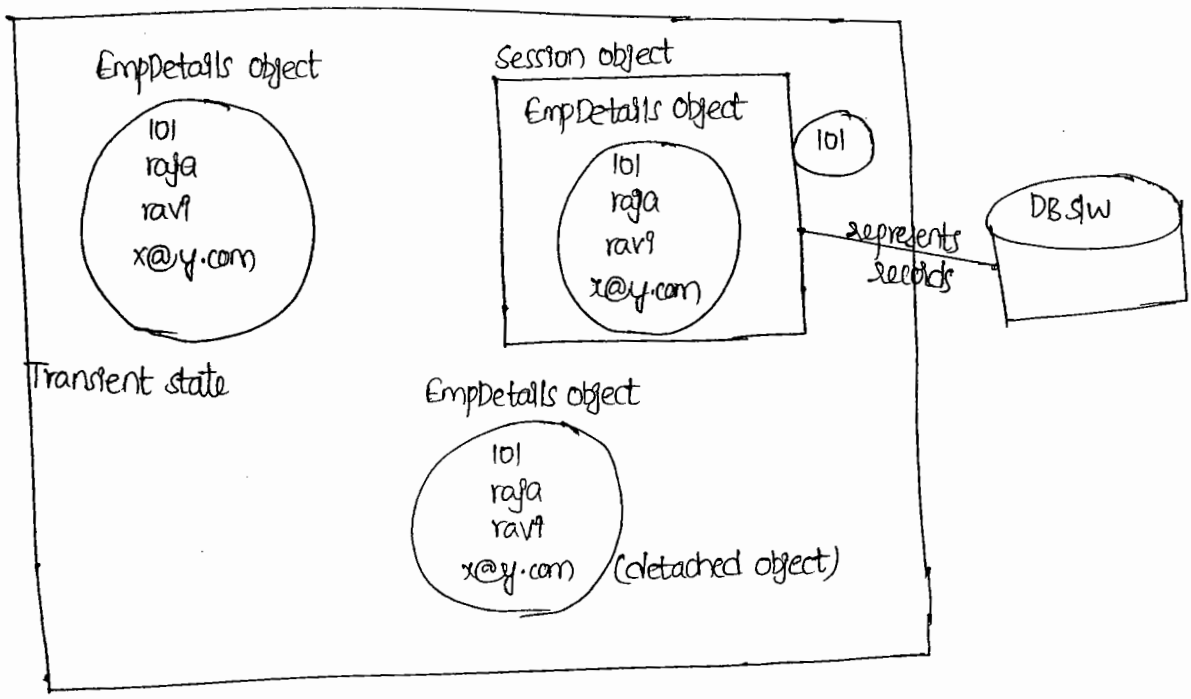
1) It creates DB Table based on appl mapping file when session factory object is created & drops that table when session factory object is closed.
2) It is very useful in testing mode of project where DB table will created temporarily at the beginning of testing & destroyed at the end of testing.

ex

In hibernate.cfg.xml

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

Hibernate Appln



Domain class object can be there in 3 states:

a) Transient state (not associated with session)

b) Persistent state (Associated with unique session)

c) Detached state (previously associated with session)

a) Transient state

⇒ Does not contain id value. Does not represent record of DB Table, so any modifications on object does not reflect in DB Table. Any assign "null" to Domain class object/ refer or when we create new object to Domain class then its state becomes Transient state.

```
EmpDetails ed = null; // transient state
EmpDetails ed = new EmpDetails(-,-,-,-); // transient state
```

We can make transient object as persistent state object by calling save(-) or persist(-), update(-), saveOrUpdate(-) or merge(-), get(-), load(-) methods.

We can make persistent state object as Transient state object by calling delete(-).

(b) Persistent state

contains id value. Associated with session represents db table record and maintains synchronization with DB table record (any modification in object reflects to DB Table record).

⇒ In the development of OR mapping persistence logic we use persistent state object.

⇒ When "null" is there in transient state object then we call get(-,-) or load(-,-) to make that object as persistent state object.

⇒ When transient state object does not contain null value then we can call save(-) on update(-) on saveOrUpdate(-) on persist(-) or merge(-) method to make that object as persistent state object.

```
EmpDetails ed = new EmpDetails(-,-,-,-); // Transient state
Transaction tx = ses.beginTransaction();
    ses.save(ed); // persistent state
    tx.commit();
EmpDetails ed = null;
(EmpDetails) ed = (EmpDetails)ses.load/ses.get(EmpDetails.class, 101); // Persistent state
```

Note: persistent state object resides in First Level cache/ L1 cache of session.

(c) Detached state

⇒ Previously associated with session not currently contains id value & does not represent record and does not maintain synchronization with record. So any modifications done in object does not reflect to DB Table record.

⇒ When close the session all persistent state objects become detached state objects.

⇒ By calling close(), evict() or clear() methods we can make persistent state object as detached state object.

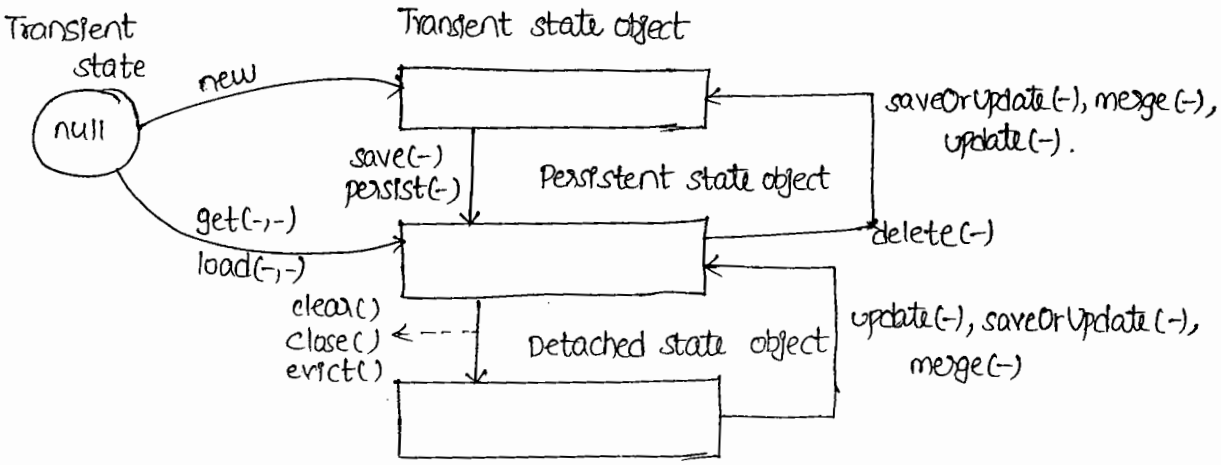
⇒ By calling update() or saveOrUpdate(), merge() methods we can make detached state objects as persistent state object.

```
EmpDetails ed = new EmpDetails(-,-,-,-); // Transient state
Transaction tx = ses.beginTransaction();
        ses.save(ed); // persistent state
      tx.commit();
    ses.close(); (or) ses.clear() (or) ses.evict(ed);
  Here on words "ed" detached state ------
```

ClientApps

```
EmpDetails ed = null; // Transient state
      ed = (EmpDetails) ses.get(EmpDetails.class, 101); // persistent state
Transaction tx = ses.beginTransaction();
     ed.setMail("xx@y.com"); // persistent state
      tx.commit();
     ses.clear(); / ses.close();
    Thread.sleep(20000);
    ed.setMail("xx@yy.com"); // Detached state
  Transaction tx = ses.beginTransaction();
          (modifications will not reflect to DB Table)
    ses.merge(ed) (or) ses.update(ed); (or) ses.saveOrUpdate(); // persistent state
      tx.commit();
```

Life Cycle Diagram of HB Domain class object



Note

⇒ HB never deletes Domain class object.

⇒ ses.delete(-) does not delete Domain class object it deletes record represented by object & makes the object as Transient state Object like any other normal object JVM deletes Domain class object when no reference are there for object.

Q What is the difference between ses.update(-), ses.saveOrUpdate(-), ses.merge(-) method?

1) The first level cache of each session object allows domain class objects with unique identity values.

2) If you call ses.update(-) method to update the object with different values when that object with same identity values is already there in first level cache of session object. Then we will get exception (non-unique object exception).

3) In the above situation if we call ses.merge(-) then no exception will be raised more ever given object modifications will be synchronized DB Table record.

4) If we modify same record by loading that record/object in multiple sessions then to merge all the modifications and to reflect the changes DB Table use ses.merge(-) method.

5) In the above discussion ses.saveOrUpdate(-) is same as ses.update(-) method.

Example

Client App6.java

```
Session  ses1 = factory.openSession();
EmpDetails  ed1 = (EmpDetails) ses.get(EmpDetails.class,1001); // persistent state(ed1)
      ses1.close();
     ed1.setMail("xx@yy.com");
Session  ses2 = factory.openSession();
EmpDetails ed2=(EmpDetails)ses2.get(EmpDetails.class, 1001);
Transaction tx=null;
 try{
      tx=ses.beginTransaction();
     ses2.saveOrUpdate(ed1); (or) ses2.update(ed1);
      //ses2.merge(ed2);
     tx.commit();
   }
 catch(Exception e)
 {
   e.printStackTrace();
   tx.rollback();
 }
```

⇒ The above code gives exception because ses2 was already got ed2 object having the same id value of ed1 object (1001);

⇒ To overcome this problem & to merge the modifications done in both ed1 & ed2 objects we can use ses.merge(-) method as shown below.

```
Transaction tx=null;
  try{  tx=ses2.beginTransaction();
        ses2.merge(ed1);
        ses2.setLname("chari");
        tx.commit();
    }
  catch(Exception e)
  {
    tx.rollback();
   e.printStackTrace();
  }
```

Conclusion

⇒ Detached state object means, the object is, previously attached with session to making detached state object as persistent state object is nothing but re-attaching object with session. In this process if session contains another object with same identity value & if we are using ses.update(-) or ses.saveOrUpdate(-) for this reattachment then non-unique object exception will be raised.

⇒ To overcome this problem use ses.merge(-) method for this re-attachment.

⇒ If you want to modify object with in a single session then use ses.update(-) or ses.saveOrUpdate(-) method.

⇒ If you want to modify same object across the multiple session object & you want to merge the all modifications then use ses.merge(-) method.

⇒ ses.update(-) just updates the object and ses.merge(-) or ses.saveOrUpdate(-) can insert(or) update the object.

DESIGN PATTERNS

⇒ Design patterns are set of rules that comes as best solutions for reoccuring problems of application development.

⇒ Design patterns are best solutions/ practices to work with various s/w technologies effectively.

⇒ ISO organization maintains the documentation both design patterns & anti patterns.

⇒ Most of design patterns are given by GOF (Gang of four) in s/w environment.

Ex: singleton, DAO, MVC, factory & etc..

⇒ DAO (Data Access Object) is pattern that seperates persistence logic from other logics of the applications (specially from business logic) & gives the following advantages.

a) Makes persistence logics reusable in multiple Business components / service components.

b) Gives flexibility to modify persistence logic.

c) Allows to implement different versions of same persistence logic. Finding totals, avg based on student marks in business logic, Inserting student record having those details is persistence logics.

⇒ Generally industry uses "table per dao" strategy while DAO pattern is, they write multiple DAO classes for multiple DAO table on 1 DAO class per table basis.

⇒ Every DAO class contain query part (as string constants) & code part (Persistence logic).

⇒ In implementation DAO we use 3 resource.

(a) DAO Interface (Declaration of persistence methods)

(b) DAO Implementation class (Implementation of persistence methods)

(c) DAO factory (Returns one of the several DAO implementation classes).

NOTE we can write several DAO implementation classes for 1 DAO interfaces to write some persistence logic in different versions. In HB this is not required because HB persistence logic is portal logic across the multiple DB sw.

NOTE we can write DAO persistence logic in any persistence technology like JDBC, HB etc...

Single line code to create HB session object using method cheining concept.

```
Session ses = new Configuration().configure("com\nt\cfgs\hibernate.cfg.xml").
                                    buildSessionFactory().openSession();
```

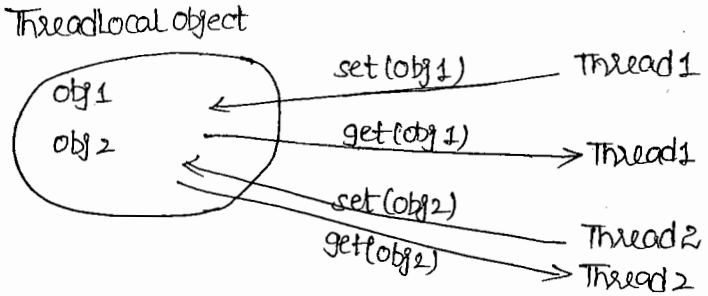
⇒In RealTime instead of writting the logic of creating session factory, session in every DAO class (or) client Application its recommended to write in a seperate utility class called "HBUtility.java".

⇒This class contains logic to create session factory in state block & methods to create the session object & to close the session object.

⇒SessionFactory is immutable object. So it is ThreadSafe by default. To make session object as ThreadSafe it is recommended to place that object in ThreadLocalObject.

⇒If we want to make any object/variable as Threadsafe without using synchronization concept & without making that object/variable as local to method then we can place in ThreadLocalObject.

⇒ThreadLocalObject allows each thread to place one object but doesnot allow other thread access that object. Eventhough multiple threads are acting on the ThreadLocalObject.



Thread1 can not get obj2 at any cost.

Thread2 can not get obj1 at any cost.

NOTE Each Thread can place only one object at a time to ThreadLocalObject.

⇒Factory pattern is capable of creating & returning one of the several implementation class object (or one of the several subclass objects based on the data ie, (that is) supplied.

Program

⇒ Project having DAO pattern, HB util class implementation.

HBProj2 (DAO Pattern)

```
|--> src
     |--> com.nt.cfgs
          |--> hibernate.cfg.xml
          |--> Employee.hbm.xml
     |--> com.nt.domain
          |--> EmpDetails.java
     |--> com.nt.dao
          |--> EmpDAO.java
          |--> EmpDAOImpl.java
          |--> EmpDAOFactory.java
     |--> com.nt.test
          |--> ClientApp.java
     |--> com.nt.utility
          |--> HBUtil.java
```

Refer: App2 of the Booklet (new).

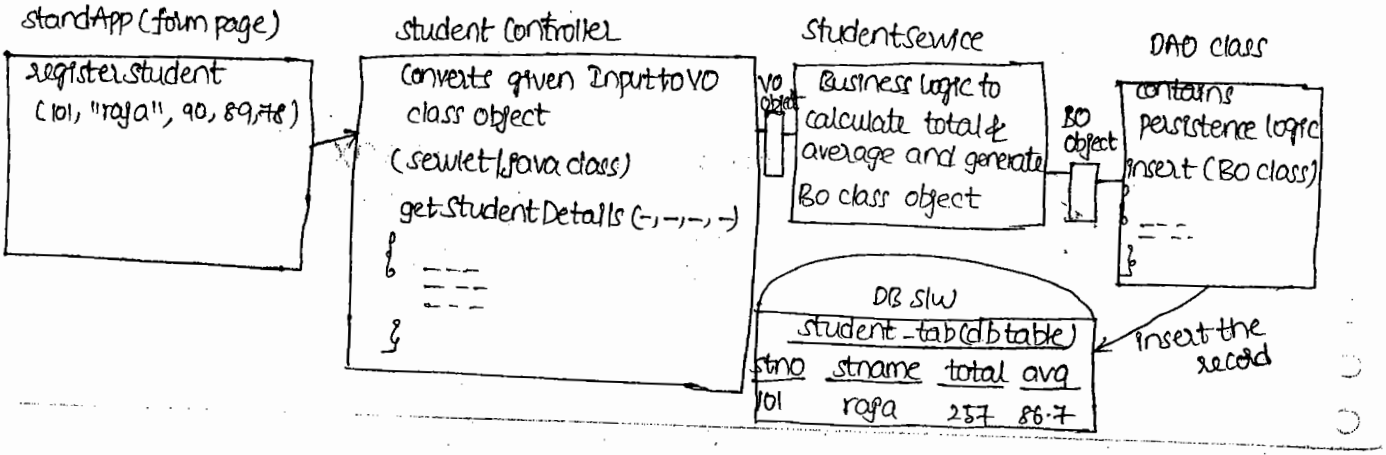
⇒ Static block is class level one time execution block.

⇒ Constructor is object level one time execution block, we generally HBSessionFactory object on one per project & DB basis. So we prefer creating this object in static block of HBUtility class.

⇒ In real time SessionFactory object will be taken on 1 per project basis and DB basis whereas session object will be taken 1 per application / client application basis.

⇒ In Java Bean class objects can hold client supplied (the end user supplied data) is called VO class. (Generally holds form data).

⇒ The Java Bean class object that can hold data as required for DB persistence operations is called BO/Domain class/Entity class.



HB projs (DAO with VO|BO)

- src
  - com.nt.cfgs
    - hibernate.cfg.xml
    - Employee.hbm.xml
  - com.nt.service
    - StudentService.java
  - com.nt.vo
    - StudentVO.java
  - com.nt.bo
    - StudentBo.java
  - com.nt.dao
    - StudentDAO.java
    - StudentDAOImpl.java
    - StudentDAOFactory.java
  - com.nt.test
    - clientApp.java
  - com.nt.utility
    - HbUtility.java
  - com.nt.controller
    - StudentController.java

⇒ If Domain class name is matching with DB Table name & Domain class property names are matching with DB table column names then specifying DB Table name, col names in HB mapping file is optional.

⇒ In a project there will be only one controller class talking with multiple service classes. These multiple service classes take the support of multiple DAO classes to interact with DB sw. All client applications using this single controller class to interact with multiple service classes.

⇒ Interact with multiple DB sw by creating multiple session factory objects with the support of multiple HB cfg files. We can make an HB application talking with more than 1 DB sw.

⇒ Application to transfer a record of oracle employee table to mysql employee table.

Note In mobile portability the customer details of one mobile nw db should be transferred to another mobile network.

DB Tables :

Employee in oracle (same as first applm)

Employee in sql (same as first appln)

HBProj4 (DAO Pattern)

```
|→src
    |→ com.nt.cfgs
        |→ ora-hibernate.cfg.xml,
        |→ mysql-hibernate.cfg.xml
        |→ Employee.hbm.xml
    |→ com.nt.domain
        |→ EmpDetails.java
    |→ com.nt.dao
        |→ TransferDAO.java
        |→ TransferDAOImpl.java
        |→ TransferDAOFactoy.java
    |→ com.nt.test
        |→ ClientApp.java
    |→ com.nt.utility
        |→ Ora-HibernateUtil.java, Mysql-HibernateUtil.java
```

Ora-hibernateutil.java

same as previous appln but create sessionfactoy object in static block as shown below...

```
static
{ try{
    factoy = new Configuration();
    configure ("ora-hibernate.cfg.xml");
    buildSessionFactoy();
  }
 catch(Exception e){
    e.printStackTrace();
  }
}
```

mysql-hibernateutil.java

same as previous appln but create session factoy object in static block by taking mysql-hibernate.cfg.xml as cfg filename

TransferDAO.java

```
package com.nt.dao;
public interface TransferDAO
{
    public boolean transferObject(int no);
}
```

TransferDAOImpl.java

```
public class TransferDaoImpl implements TransferDAO{
    public boolean transferObject(int no){
        boolean flag=true;
        //load object from oracle
      Session oraSes=OraHibernateUtil.getSession();
      EmpDetails ed=(EmpDetails) oraSes.get(EmpDetails.class,no);
        //save object to mysql
      Session mysqlSes=MysqlHibernateUtil.getSession();
       Transaction tx=null;
        try{
            tx=ses.beginTransaction();
            mysqlSes.save(ed);
             tx.commit();
             s.o.p("Record transferred");
             flag=true;
          }
        catch(Exception e){
              tx.rollback();
               flag=false;
           }
        OraHibernateUtil.closeSession();
        MysqlHibernateUtil.closeSession();
    }
}
```

TransferDaoFactoly.java

```
public class TransferDaoFactoly
{
  public static TransferDao getDaoInstance()
   {
     return new TransferDaoImpl();
   }
}
```

ClientApp.java

```
public class ClientApp{
  public static void main(String args[]){
TransferDao dao=TransferDaoFactoly.getDaoInstance();
boolean flag=dao.transferObject(1001);
   s.o.p("Record transfered"+flag);
  }
}
```

Object Versioning

⇒The process of keeping track of how many times the loaded object is modified through HBPersistence logic is called versioning. To implement these features manually through JDBC we need to write lot of complex logic but in HB in feature that will work by enabling few settings.

⇒The Real time use cases are:

1) keeping track of how many transactions are done on a bankacc in a month. (To apply charges for more than 10 transactions).
2) keeping track of how many times the add on services are used in a month (like changing caller tune)
3) keeping track of how many times personal info is changed in a month (like changing address, email id etc..).

Example App

1) keep first app ready
2) add a new column in db table "ver_col" in db table "employee".

```
sql> alter table employee add ver_col number(5);
```

3) add an extra numeric property in domain class "EmpDetails" to keep track of object versioning along with getter & setter methods.

In EmpDetails.java

```
private int ver;
//getter & setter methods
```

4) cfg the above property in mapping file using <version> tag.

In Employee.hbm.xml (after <id> tag)

```
<version name="ver" column="ver_col"/>
```

5) Write following code in clientApp to insert record & observe "ver_col" value in that record.

clientApp.java

```
//get session
Session ses = HibernateUtil.getSession();
EmpDetails ed = new EmpDetails (1002, "siva", "reddy", "vsiva@yf.com");
Transaction tx=null;
try {
    tx=ses.beginTransaction();
    ses.save(ed);
    tx.commit();
}
catch(Exception e) { tx.rollback(); }
```

O/P : 1002 siva reddy vsiva@yf.com 0 (ver_col)

6) write following code in clientApp to load the object & to modify the object.

NOTE

observer changes happening in "ver" property & "ver_col" column.

```
EmpDetails ed=(EmpDetails)ses.get(EmpDetails.class,1002);
Transaction tx=null;
  try{
      tx=ses.beginTransaction();
      ses.setMail("x@y.com");
       tx.commit();
      S.O.P("object ver : "+ed.getVer()+"times ");
      }
   catch(Exception e)
     {
       tx.rollback();
     }
```

O/P: 1002 raja rao x@y.com 1 (ver_col)

NOTE

⇒ The versioning feature of HB does not keep track of the modifications done on DB table record through sql prompt (or) other applications. It will keep track of only the modification done on the loaded objects through HB persistence logics.

Timestamp

1) This feature of HB keeps track of when each object/record inserted or updated ie, it holds the date & time of record/object insertion/updation.

2) versioning feature keeps track of how many times the object/record is modified where as timestamp feature keeps track of when each object/record is modified.

Usecase1

If banking project maintains seperate DB table having all daily transactions by enabling timestamp feature we can keep track of the date & time of each transaction.

Usecase2

While maintaining stock market share value in DB table as record we can also maintain the lastly updated time & date through timestamp feature.

NOTE

While working with xml based HB cfgs we can't apply both version, timestamp features from single application because we can place either <version> or <timestamp> in our HB mapping file.

Example program

1) Keep first project ready with basic employee table (eid, fname, lname, email).

2) add new column (last-updated) in employee table of type "timestamp".

```
sql> alter table employee add last-updated timestamp;
     alter table employee drop column vel-col;
```

3) add new property in Domain class with setter & getter methods.

In EmpDetails.java

```
public void setLastUpdated (Timestamp lastUpdated)
 {   this.lastUpdated = lastUpdated;
 }
public Timestamp getLastUpdated()
 {  return lastUpdated;
 }
```

4) cfg the above property in mapping file with <timestamp> tag.

In Employee.hbm.xml (after <id> tag)

```
<timestamp name="lastUpdated" column="last-updated"/>
```

5) Write code in ClientApp to save Object/record

NOTE Observe the value of "last-updated" column.

```
Session ses = HibernateUtil.getSession();
EmpDetails ed = new EmpDetails(1003, "siva", "reddy", "x@y.com");
Transaction tx=null;
try{  tx=ses.beginTransaction();
      ses.save(ed);
      tx.commit();
   }
catch(Exception e){ tx.rollback(); }
```

o/p : 1003 siva reddy x@y.com 10-MAY-15 02.50.34.932023 PM (last-updated)

6) Write code in client app to load object/record to modify the object/record.

```
EmpDetails ed = (EmpDetails) ses.get(EmpDetails.class, 1003);
Transaction tx=null;
  try{
      tx=ses.beginTransaction();
       ses.setMail("x@y1.com");
       tx.commit();
      s.o.p("object is modified at:"+ed.getLastUpdated());
     }
catch(Exception e){
   tx.rollback();
  }
```

Composite Identity Field cfg

⇒ If primary key (pk) constraints is applied on 1 column of db table then it is called "singular pk" constraint.

⇒ If pk constraint is applied on column of db table then it is called "composite pk" constraint

⇒ If 1 property of domain class is configured as identity field then it is called "singular id field" (use <id> tag).

⇒ If more than 1 property ---> "composite id field" (use <composite-id> tag)

⇒ If db team gives db table with singular pk constraint then go for singular id field cfg.

⇒ If db team gives composite pk constraint ⟶ composite id field cfg.

⇒ If you have a table maintaining both programmer id's & project id's we can't apply singular pk constraint on both columns because both columns should allows duplicates. The reason is 1 or more programmer can work for a project. In 1 project multiple programmer can be there & 1 programmer can for multiple projects. In this situation we need to apply composite pk.

Programmers_projects

proj id	prog id (composite id on both columns)
1001	101
1002	101
1003	102
1001	103

Example App on Composite Id field cfg

1) Create db table having composite primary key constraint on eid, firstname columns

```
sql> create table employee1(eid number(5), firstname varchar2(20), lastname varchar2(20),
                            email varchar2(20), primary key (eid, firstname));
```

2) Keep first project ready.

3) Design HB domain class as shown below having the support of helper class.

In EmployeeId.java

```
package com.nt.domain;

public class EmployeeId implements Serializable
{
    private int no;
    private String fname;
    private EmployeeId()
    {
    }
}
```

```
public EmployeeId (int no, String fname)
{
  this.no=no;
  this.fname = fname;
}
//write setter & getter methods
}
```

NOTE:

This helper class contain properties to act as composite Identity field. Since HB expects or return id value as Serializable object, we must take this class as Serializable class.

EmpDetails.java

```
package com.nt.domain;

public class EmpDetails implements java.io.Serializable
{
    private EmployeeId empId;
    private String lname,mail;
    //constructor
   //setter & getter methods
}
```

4) cfg composite id field in mapping file as shown below <composite-id> tag.

Employee.hbm.xml

```
<hibernate-mapping>
<class name ="com.nt.domain.EmpDetails" table ="employee1">
<composite-id name ="empId" class ="com.nt.domain.EmployeeId">
<key-property name ="no" column="EID" />
<key-property name ="fname" column ="firstname" />
</composite-id>
<property name="lname" column="lastname" />
<property name="mail" column="email" />
</class>
</hibernate-mapping>
```

5) Write the following code in clientApp to insert the record.

clientApp.java

```
Session ses = HibernateUtil.getSession();
EmployeeId id = new EmployeeId (101, "siva");
EmpDetails ed = new EmpDetails (id, "reddy", "x@y.com");
Transaction tx=null;
    try {
        tx=ses.beginTransaction();
      (EmployeeId) no =(EmployeeId)ses.save(ed);
        tx.commit();
        S.O.P("Record inserted");
       }
    catch(Exception e)
     {
       tx.commite();
      }
```

6) Load the object in clientApp.

```
EmployeeId id = new EmployeeId (101, "siva");
EmpDetails ed = (EmpDetails)ses.load(EmpDetails.class,id);
```

Generators in HB

1) HB gives set of predefined algorithms as identity value generator to generate identity value for HB domain class object while saving them to db sw.

2) All these generator are the classes of org.hibernate.id pkg implementing org.id.IdentifierGenerator (I).

3) We can also develop custom generator by implementing "IdentifierGenerator (I)".

4) All predefined generators are having nicknames (or) code name to use in the cfg.

They are:

- select
- increment
- hilo
- sequence
- sequencehilo
- native
- assigned (default)
- foreign
- uuid
- guid
- seqhilo and etc...

5) All these generators can cfg only while working with singular Identity field cfg.

6) While choosing these generators we must consider

(a) Make sure that the value generated by the identity value is compatible to store in "Identityfield" i.e, cfg

(b) Make sure that underlying db sw provides facilitates that are required for "generator" to generate the identity value.

Usecase

⇒ While perform customer/student registration the registration should be generated dynamically while inserting the records/saving objects. For this we can use the "generator" support.

1) Assigned (algorithm)

⇒ It is default generator if no generator is configured.

⇒ This generator class name "org.hibernate.id.Assigned".

⇒ This generator take value from identity field i.e, configured & makes that value as the identity value while saving the object.

⇒ This generator works with all db sw. so it is db sw independent.

⇒ This generator can generate any kind of value as the identity value.

Example

1) keep first appln ready

2) cfg assigned generator by using one of the following 3 ways in mapping file.

Employee.hbm.xml

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

(or)

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

(or)

```
<id name="no" column="EID">
<generator class="org.hibernate.id.Assigned"/>
</id>
```

3) Write the following code in client App to save object/record

clientApp.java

```
Transaction tx=null;
try{
    tx=ses.beginTransaction();
  EmpDetails ed = new EmpDetails(1001, "siva", "reddy", "x@y.com");
    int id=(Integer)ses.save(ed);
    S.O.P("generated id value:"+id);
   tx.commit();
 }
 catch(Exception e){
   tx.rollback();
   }
```

2) increment (algorithm)

1) Generate the identity value of type long (or) int (or) short

2) It works in all db sw. Indicates this generator db sw independent.

3) uses [max val+1] formula to generate identity value.

4) If db table is empty it takes '0' as max value to generate identity value.

5) Does not consider the deleted records while generating the identity value.

Ex

In Employee.hbm.xml

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

(3) Identity(algorithm)

1) uses Identity column(column with auto increment) in DB2, mysql, mssqlsewer, sybase & hypersonicsql to generate identity value of type int, short or long.

2) Since oracle does not suppolt identity columns, this algorithm does not work with oracle db sw.

3) uses [max val+1] formula to generate id(identity) value.

4) considers deleted record values while generating new identity value.

Example

1) Make ur appln pointing to mysql db sw

2) Apply "autoincrement" constraint on EID column to make that "column "identity" column.

SQLyog → employee → EID column → right click → Manage columns →
select "auto increment," check box.

3) cfg "identity" generator in mapping (Employee.hbm.xml)

Employee.hbm.xml

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

4) class name org.hibernate.id.IdentityGenerator

Identity	increment
1) DB sw dependent	1) DB sw independent
2) Id field mapping column must have auto increment constraint	2) Not required
3) Considers deleted value while generating new id value	3) Does not considers
4) DB sw give id value after inserting record	4) HB sw generates id value

4) Sequence (algorithm)

1) Uses sequence in db2, postgresql, oracle, sapdb, makoi to generated id value of type long, short or int.
2) It works in oracle because oracle supports "sequences", but does not work mysql because "mysql" does not "sequences".
3) If no sequences is specified the Fw generates "hibernate-sequence" as default sequence having "increment by 1" logic. For this "hbm2ddl.auto" property should create/update.
4) If "sequence" name is specified then that name will be used to generate the identity value.

Example 1

(uses default sequence hibernate_sequence) (use oracle setup)

In mapping file (Employee.hbm.xml)

```
<id name="no" column="EId">
<generator class="sequence" />
</id>
```

In cfg file

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

Example 2

(working with user defined sequence)

1) create sequence in oracle db slw.

```
sql> create sequence EmpNO_sequence increment by 5 start with 1000;
```

2) specify the above sequence in mapping file (in Employee.hbm.xml)

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

5) hilo(algorithm)

1) It is db slw independent generator generating identity value of type int or short or long.

2) It uses hilo algorithm. To generate the identity values uses given helper table column value as the source of hi value & max_lo param value as the source of low value.

Params are: table; column, max_lo

⇒ The value of helper table column will be incremented by 1 for every "hilo" algorithm generator object saving/record insertion.

The class name: org.hibernate.id.TableHiloGenerator

Example

1) create helper table in oracle having helper column.

```
sql> create table mytable (mycolumn number(4));
     insert into mytable values (10);
     commit;
```

2) cfg the "hilo" generator as shown below:

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

formula

hi value * { low value+1 }

(table column value) * (max_lo param+1)

10 * (10+1) = 110

11 * (10+1) = 121

```
sql> select * from mytable;
```

10 > 11 > 12 > 13

6) seqhilo

1) uses hilo algorithm internally by using "given" sequence generated value as the source of hi value & "max_lo" param value as the source of low value.

2) Generates the identity value of type int or short or long.

3) It works only in that db sw that supports sequence like oracle. (Does not work in mysql)

4) params are : sequence, max_lo

5) class name : org.hibernate.id.SequenceHiloGenerator.

Example

a) create sequence in oracle db sw.

sql> create sequence mysql increment by 5 start with 100;

b) cfg "seqhilo" generator in mapping file as shown below.

```
<id name="no" column="EID">
<generator class="seqhilo">
<param name="sequence">mysql</param>
<param name="max_lo">5</param>
</generator>
</id>
```

Here no helper table is required

Formula

sequence generated value * max_lo value
(hi value) (low value)

100 * (5+1) = 600
105 * (5+1) = 630
110 * (5+1) = 660

7) native (algorithm)

1) This generator uses identity (or) sequence (or) hilo generator. based on
(mysql) (oracle)

based on the limitations or capabilities of underlying db sw.

2) If db sw supports identity column then it uses identity generator if db sw supports "sequences" it uses "sequence" generator. If db sw does not support both "sequences", "identity" columns then it uses "hilo" generator.

3) There is no class for this generator.

Example

In oracle

```
<id name="no" column="EID">
  <generator class="native" /> uses "hibernate_sequence" internally.
</id>
```

In oracle

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

In Mysql

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

⇒ This generator makes HB as patable towards generating the identity value.

6) foreign (algorithm)

1) In order to use the identity value of associated object as the identity value of current object we need to use "foreign" generator.

2) Generally the student id will be taken as "Library membership" id in one to one association. For this we can use "foreign algorithm".

3) This generator is useful in one-to-one association.

9) select (algorithm)

uses db sw based trigger execution to generate the identity values of type long(or) int(or)short.

Use cases

⇒ In custom registration, mobile number registration use increment generator (deleted no's will not be considered).

⇒ In Employee registration bank account generation use identity/sequence algorithm.

⇒ To generate coupon nos in lucky draws we hilo/seqhilo algorithm. To make or generator pottable to work with db sws use native algorithm. In one-to-one relationships to use the identity value of parent object as the identity value of child object use foreign generator.

b) uuid (universal unique id) algorithm

⇒ Generates 32 bit hexadecimal string value as the identity value. Uses the following details while generating the identity value.

a) System IP Address
b) System Date & Time
c) JVM Time Ticker
& etc...

Example

1) keep first project ready
2) change EID column type to varchar2(40) by creating new DB table employee2.
3) change the "no" property value type to java.lang.String in "EmpDetails" class & reflect the change in setter & getter methods.
4) cfg "uuid" generator in mapping files as shown below.

```
<class name="com.nt.domain.EmpDetails" table="Employee2">
 <id name="no" column="EID">
 <generator class="uuid"/>
 </id>
 <property name="fname" column="FIRSTNAME" length="20"/>
 ----
 ----
</class>
```

5) call ses.save(-) method in client appln as shown below

```
String id = (String) ses.save(details);
    SOP("Generated id value :"+id);
```

NOTE

⇒ All the classes of pre-defined "generator" implements "org.hibernate.id.IdentifierGenerator(I)". If we are not comfortable with pre-defined "generator" then we can use user-defined generator support.

⇒ Every custom/user-defined generator must implement "IdentifierGenerator(I)" directly or indirectly. All these generators must be cfg in mapping file directly with the class name.

Procedure to generate psuedo random number as identity value through custom generator.

1) keep first appln ready
2) develop custom generator by implementing identifier generator interface.

package com.nt.generator;

RandomIdGenerator.java

```
public class RandomIdGenerator implements IdentifierGenerator
{
   public Serializable generate(SessionImplementor sesImpl, Object obj) throws
                                                                HibernateException
   {
      Random rd = new Random();
      int val = rd.nextInt(100000);
      return val;
   }
}
/* sesImpl --> Control blw session object & other parts
   obj      --> represents HBdomain class objects
*/
```

3) cfg the above as "generator" in hibernate mapping file.

In Employee.hbm.xml

```
<id name="no" column="EID" />
<generator class="com.nt.generator.RandomIdGenerator"/>
</id>
```

4) write code in clientApp to save the object/record.

```
int id = (Integer)ses.save(details);
S.O.P("The generated id value is :"+id);
```

Develop Custom Generator giving values FF001, FF002, ---FF010, FF011, --- FF100, ----

Note: It should work with all db tables.

Example

1) keep first project ready as shown below.

HBproj11 (CustomGenerator2)

- src
  - com.nt.cfgs
    - hibernate.cfg.xml
    - Employee.hbm.xml
  - com.nt.domain
    - EmpDetails.java

```
|-> com.nt.dao
      |-> EmpDAO.java
      |-> EmpDAOImpl.java
      |-> EmpDAOfactory.java
|-> com.nt.test
      |-> ClientApp.java
|-> com.nt.generator
      |-> FFIDGenerator.java
|-> com.nt.utility
      |-> HibernateUtil.java
```

2) Create employee3 table same as employee having EID column with varchar2(20) column data type.

3) change "no" property type to java.lang.String in EmpDetails.java

4) Create sequence in oracle having increment by 1 logic.

```
SQL> create sequence ffGen_seq increment by 1;
     sequence created;
```

5) Develop custom generator.

FFIdGenerator.java

```
public class FFIdGenerator implements IdentifierGenerator
{
  public Serializable generate(SessionImplementor sesImpl, Object obj) throws HibernateException
  {
    try{
        Connection con = sesImpl.connection();
        Statement st = con.createStatement();
       ResultSet rs = st.executeQuery("select ffGen_seq.nextVal from dual");
         int val=0;
        if(rs.next())
          val = rs.getInt(1);
        else if(val<10)
          return "FF00"+val;
        else if(val<100)
          return "FF0"+val;
        else
          return "FF"+val;
    }
    catch(Exception e){
      e.printStackTrace();
      return "can not generated id value";
    }
    finally{ /*do not close connection obj;
              close st, rs obj */
    }
  }
}
```

6) cfg the above generator in mapping file

In Employee.hbm.xml

```
<id name="no" column="EID" >
<generator class ="com.nt.generator.FFIDGenerator"/>
</id>
```

7) Develop the DAO pattern classes & interfaces.

EmpDAO.java

```
public interface EmpDao
 {
  public String registerEmp(EmpDetails details);
 }
```

EmpDAOImpl.java

```
public class EmpDAOImpl implements EmpDAO
 {
  public String registerEmp(EmpDetails details)
   //get HB session
  Session ses = HibernateUtil.getSession();
    //save object
   Transaction tx=null;
  try{
      tx=ses.beginTransaction();
     String id=(String) ses.save(details);
      tx.commit();
    return id;
    }
  catch(Exception e)
   {
    e.printStackTrace();
    return "Registration Failed";
   }
  finally{
        HibernateUtil.closeSession();
       }
  }
 }
```

EmpDAOFactoly.java

```
public class EmpDAOFactoly
{
    public static EmpDAO getInstance()
    {
        return new EmpDAOImpl();
    }
}
```

ClientApp.java

```
public class ClientApp
{
    public static void main(String[] args)
    {
        EmpDAO dao = EmpDAOFactoly.getInstance();
        //prepare domain class object
        EmpDetails details = new EmpDetails (" ","Siva","Reddy","x@y.com");
        String id = dao.registerEmp(details);
        S.O.P("Emp Registered with:"+id);
    }
}
```

Connection pooling in Hibernate

⇒ Connection pool is a factory that contains set of readily available jdbc connection object before actually being used.

⇒ SessionFactory represents one jdbc connection pool, while creating each session object one connection object of jdbc connection pool will be utilized.

Q What is the difference between pool & cache?

Pool contains set of same items having reusability.

Cache contains set of different items having reusability.

⇒ If no jdbc pool is configured HB FW uses the built-in jdbc connection pool given by HB FW having max capacity 20 & minimum capacity 1.

⇒ To control max size we can use "connection.pool-size" property as shown below.

In hibernate.cfg.xml

```
<property name="connection.pool_size">10</property>
```

⇒ The built-in connection pool of HB is not good for production environment. So use other jdbc connection pools.

Q What is the jdbc connection pool ie, used in your project?

⇒ If our HB project is standalone-App (running outside the server) then use either C3P0 pool (or) Proxool pool.

⇒ If our HB App/ logic is running from the server then use that server managed /side jdbc connection pool.

NOTE Do not use HB's built-in jdbc connection pool.

⇒ We can use specific connection provider class in the .HB cfg file, to make HB flw working with certain jdbc connection pool.

	Pool	Connection Provider class
1)	C3P0	org.hibernate.c3p0.internal.C3P0ConnectionProvider
2)	proxool	org.hibernate.proxool.internal.ProxoolConnectionProvider
3)	Built-in con pool	org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl
4)	sewedManaged con pool	org.hibernate.engine.jdbc.connections.internal.DataSourceConnectionProviderImpl

Procedure to Configure C3P0 connection Pool in our HB Application

1) Keep any standalone HB Appln ready.
2) Add <HB4.3_home>\lib\optional\c3p0 folder jar files to build path of the project
3) cfg C3P0 connection pool provider class & C3P0 connection pool properties in HB cfg file.

In hibernate.cfg.xml

```
<property name="connection.provider_class">org.hibernate.c3p0.internal.C3P0ConnectionProvider</property>
<property name="c3p0.max_size">20</property>
<property name="c3p0.min_size">5</property>
<property name="c3p0.acquire_increment">2</property>
```
(*1)

*1: Specifies the number of new JDBC connection objects that should be created when there is a need of creating new connection objects.

NOTE

1) we can collect all the above properties of cfg file from <HB home>\project\etc\hibernate.properties
2) Even though connection provider class is not configured. If we just cfg "hibernate.c3p0.*" properties in the hibernate cfg file then the HB flw uses C3P0 connection pool.

Assignments

1) Develop custom generator that gives FF001... identity values & works with all db slw's & tables
2) Make HB application interact with postgreSQL?
3) What is the Boiler plate code problem?

Procedure to configure "proxool" jdbc connection pool in our standalone HB application

1) Keep first Appln ready

2) Add <HB_home>\lib\optional\proxool folder jar files (2) to build path

3) Prepare seperate xml file having proxool configurations.

src\com\nt\cfgs\proxool.xml

```
<proxool-config>
 <proxool>
  <alias>pool1</alias>
  <driver-url>jdbc:oracle:thin:@localhost:1521:xe</driver-url>
  <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
  <driver-properties>
  <property name="user" value="scott"/>
  <property name="password" value="tiger"/>
  </driver-properties>
 <minimum-connection-count>10</minimum-connection-count>
 <maximum-connection-count>20</maximum-connection-count>
 </proxool>
</proxool-config>
```

4) Write the following entries in HB cfg

hibernate.cfg.xml

```
<hibernate-configuration>
<session-factory>
 <property name="connection.provider_class"> org.hibernate.proxool.internal.ProxoolConnectionProvider</property>
 <property name="proxool.pool_alias"> pool1 </property>
 <property name="proxool.pool xml"> com\nt\cfgs\proxool\cfgs.xml </property>
<mapping resource="com\nt\cfgs\Employee.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

NOTE

In standalone application it is recommended to use "C3P0" jdbc connection pool.

JNDI Registry

1) It is a s/w service which provides global visibility to objects or object references.
If you want objects (or) object reference globally in multiple places then that object | object references in JNDI Registry.

Ex: RMI Registry, COS Registry, LDAP Registry & etc..

2) Every webserver | Application server gives 1 built-in JNDI Registry.

3) To interact with JNDI Registry from our Java Applns we can use JNDIAPI (javax.naming & its subpackages) (part JSE Module)

Terminology:

Bind: keeping object | object reference in jndi registry
ReBind: replacing object | object reference
unBind: removing object | object reference
lookup: search object | object reference.

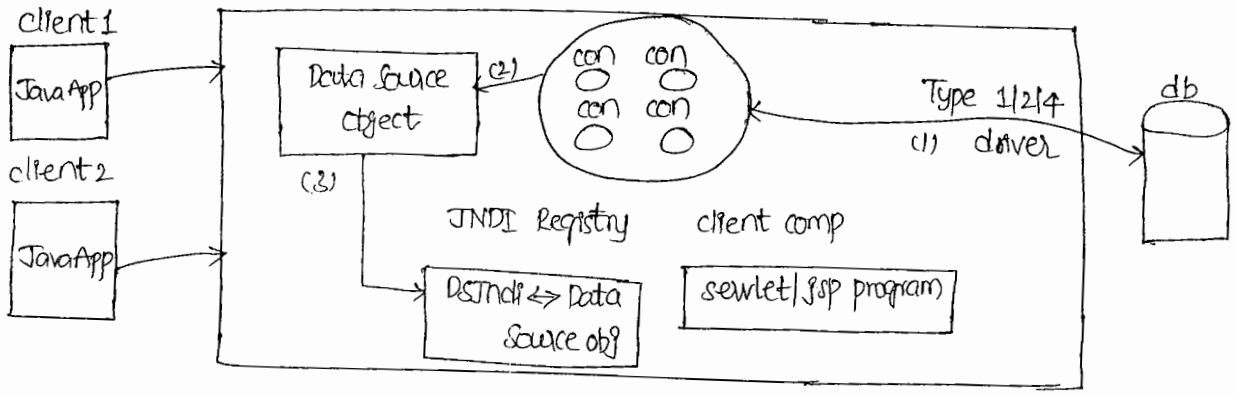
Jdbc Connection Pools

2 Types:

(a) server side jdbc con pools --> Managed in webserver & Application server
(b) client side jdbc con pools --> Ex: C3PO, proxool, Apache DBCP & etc.

⇒ In server side Jdbc con pool the DataSource object represents jdbc con pool & its reference will be in JNDI Registry for global visibility.

Understanding server managed connection



1) Team Leader (TL) | project Leader (PL) uses Type 1|2|4 jdbc driver to create Jdbc conpool server having set of connection objects

2) TL|PL creates DataSource object representing jdbc con pool (It is the object of a class that implements datasource interface).

(3) The datasource object reference will be placed in JNDI Registry for global visibility having nickname or alias name | JNDI name.

NOTE

All client application context JNDI Registry to get datasource object reference & uses that reference to get Jdbc connection objects from connection. Once the work is completed the same connection objects will be returned back to con pool.

Procedure to create domain in weblogic server 12c

1) Every domain of weblogic act as one domain server / Appln server

2) If multiple projects of a company uses weblogic then the weblogic sw will be installed only one in a common computer, In the multiple domains will be created for multiple projects on 1 per project basis.

3) start → All programs → Oracle → Oracle home → Weblogic Server → Tools → Configuration Wizard.

Create a new domain → next → next → name : --- ; password : --- ; confirm password : --- ;

> Development mode → next → Administration server → listen port No: 7070 → next → create → next → finish.

1) start "NTHB27Domain" server.

<oracleweblogic_home> \domains\NTHB27domain\startweblogic cmd

2) open admin console of weblogic domain server

http://localhost:7070/console →

username : [ ]
password : [ ]
(Login)

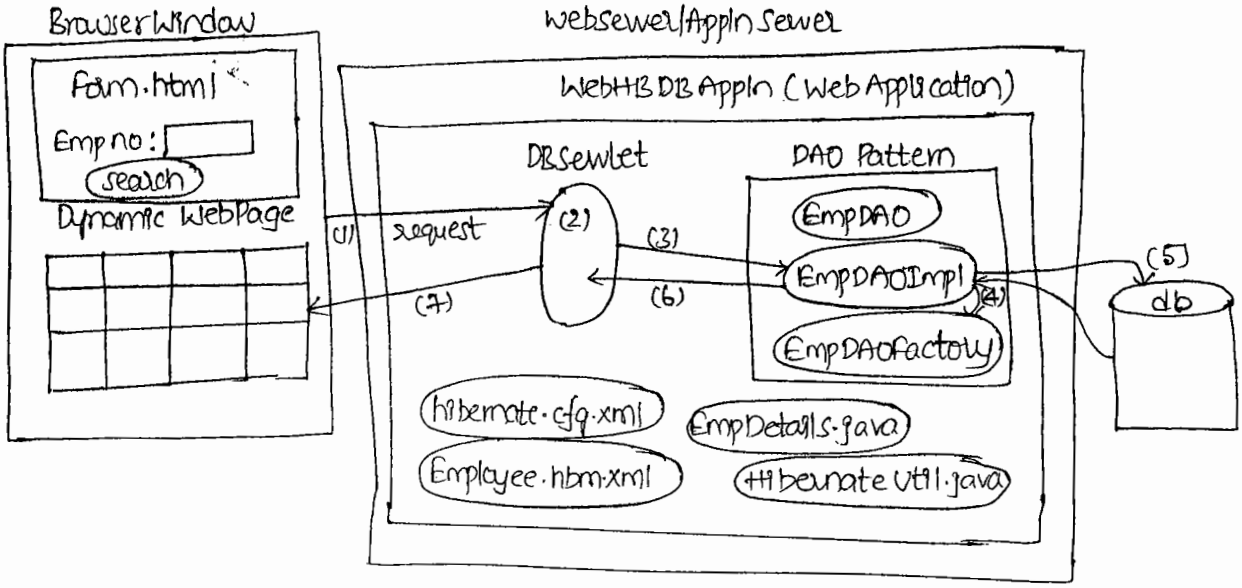
3) create datasource pointing to jdbc con pool oracle

Admin console screen → services → datasources → New → Generic Datasource → <user defined> → next → Databasedriver → oracleDriver (thin) for services connections → next → database name : xe, hostname : localhost; username : scott; port : 1521; pwd : tiger; confirm pwd : tiger → next → Test Configuration → Connection Test success → Next → select Admin server → Finish.

launch hds1 → connection pool → Initial capacity : 5; Max capacity : 1000; Min capacity : 10; → save → Advanced → shrink frequency : 800 → save.

*800 → Destroy the idle Jdbc con objects after every 800 seconds.

⇒ The moment finish button is clicked, the datasource object references that points to connection pool will be placed in JNDI Registry automatically.



WebHBDBAPPln (Dynamic WebProject)

HBPrg13

- ⊢→ JavaResources
  - ⊢→ src
    - ⊢→ com.nt.cfgs
      - ⊢→ hibernate.cfg.xml (HB cfg file)
      - ⊢→ Employee.hbm.xml (HB mapping file)
    - ⊢→ com.nt.domain
      - ⊢→ EmpDetails.java (HB Entity | Domain | POJO class)
    - ⊢→ com.nt.utility
      - ⊢→ HibernateUtil.java
    - ⊢→ com.nt.web
      - ⊢→ DBServlet.java
    - ⊢→ com.nt.dao
      - ⊢→ EmpDao.java, EmpDaoImpl.java, EmpDaoFactory.java
- ⊢→ WebContent
  - ⊢→ form.html
  - ⊢→ WEB-INF → web.xml

Jar files: HB Basic Library... !

⇒This application is deployable in the server. so use server managed jdbc connection pool.

Procedure for development.

1) Configure Tomcat 7x server with Eclipse IDE
2) Create Dynamic web project
3) Develop the resource of the project.

HibernateUtil.java

```
package com.nt.utility;
  import org.hibernate.Session;
                .SessionFactory;
                .cfg.Configuration;

public class HibernateUtil {

private static SessionFactory factory=null;
   static{
        try{
            factory = new Configuration().configure("com/nt/cfgs/hibernate.cfg.xml").
                                                buildSessionFactory();
            }
        catch(Exception e){
            e.printStackTrace();
            }
      }//static
static ThreadLocal<Session> tl=new ThreadLocal<Session>();
 static Session ses=null;
 public static Session getSession()
 {
    try{
        if(tl.get()==null)
        {
          ses=factory.openSession();
          tl.set(ses);
        }
        else{
            return tl.get();
        }
       }//try
                                    catch(Exception e){
                                     } return null;
                                  }//method
```

```
public static void closeSession()
  {
    try{
        ses.close();
       }
      catch(Exception e){
         e.printStackTrace();
         }
      }
  }
```

Employee.hbm.xml

```
<hibernate-mapping>
<class name="com.nt.domain.EmpDetails" table="Employee">
<id name="no" column="EID" />
<property name="fname" column="FIRSTNAME" />
<property name="lname" column="LASTNAME" />
<property name="mail" column="EMAIL" />
</class>
</hibernate-mapping>
```

EmpDetails.java

```
package com.nt.domain;
public class EmpDetails{
  private int no;
  private String fname, lname, mail;
   public EmpDetails(){
        S.O.P("EmpDetails(0)-param constructor");
       }
   //setter & getter methods;
  public String toString()
    { return "EmpDetails are:[no="+no+", fname="+fname+", lname="+lname+",
                                                    mail="+mail+"];
    }
}
```

hibernate.cfg.xml

```
<hibernate-configuration>
 <session-factory>
 <property name="connection.provider_class">org.hibernate.engine.jdbc.connections.
                                    internal.DatasourceConnectionProviderImpl</property>
 <property name="connection.datasource">DsJndi</property>
 <property name="show_sql">true</property>
 <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
 </session-factory>
</hibernate-configuration>
```

EmpDao.java

```
package com.nt.dao;
import com.nt.domain.EmpDetails;
public interface EmpDao{
        public EmpDetails seach(int no);}
```

EmpDaoImpl.java

```
package com.nt.dao;
import org.hibernate.Session;
import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;
public class EmpDaoImpl implements EmpDao{
    public EmpDetails seach (int no)
    { Session ses = HibernateUtil.getSession();
      EmpDetails details = (EmpDetails)ses.get (EmpDetails.class,no);
       HibernateUtil.closeSession();
        if(details!=null)
        return details;
        else
        return null;
    }
}
```

EmpDaofactoly.java

```
package com.nt.dao;
public class EmpDaofactoly
{
 public static EmpDao getInstance();
  {
   return new EmpDaoImpl();
  }
}
```

DBSewlet.java

```
public class DBSewlet extends HttpSewlet{
    public void doGet(HttpSewletRequest req, HttpsewletResponse res) throws
                                        SewletException.IOException
    {
      res.setContentType("text/html");
      PrintWriter pw=res.getWriter();
      int no = Integer.paseInt(req.getParameter("no"));
      EmpDao dao = EmpDaofactoly.getInstance();
      EmpDetails details = dao.search(int no);
      if(details!=null)
         pw.println("<font color='red'>"+details+"</font>");
      else
         pw.println("<font color='red'>Emp details not found</font>");
         pw.close();
    }
   public void doPost(HttpSewletRequest req,HttpSewletResponse res) throws
                                        SewletException.IOException
      {
        doGet(req,res);
      }
}
```

web.xml

```
<web-app>
 <sewlet>
<sewlet-name> abc </sewlet-name>
<sewlet-class> com.nt.web.DBsewlet </sewlet-class>
</sewlet>
<sewlet-mapping>
<sewlet-name> abc </sewlet-name>
<url-pattern> /dburl </url-pattern>
</sewlet-mapping>
<welcome-file-list>
<welcome-file> form.html </welcome-file>
</welcome-file-list>
</web-app>
```

1) Configure "com.nt.web.DBsewlet" with "/dburl" url pattern

2) Configure "form.html" as welcome-page.

form.html

```
<h1> Employee Seach </h1>
<form action ="dburl" method="get">
 Empno : <input type="text" name="no"> <br>
 <input type="submit" value="search"/>
 </form>
```

NOTE

⇒ If sewlet/jsp program uses third party api like hibernate api then the api related jar file should added to classpath & should be added to WEB-INF/lib folder.

⇒ In eclipse the jar files to build path of the project will not be moved WEB-INF/lib of webproject directly. For this we need to some process ie.,

Right click on the project → cfg build path → Deployment Assembly → Java BuildPath entries → HB Basic lib (select all) → Finish.

⇒ Export war file to <WEBLOGIC_HOME>\user_projects\domains\NTHB27Domain\autodeploy folder.

Right click → Export → war file → Web Project: <userdefined>
Destination file: <Weblogic domain path>

⇒ start "NTHB27Domain" server
<WEBLOGIC_HOME>\user_projects\domains\NTHB27Domain\startWeblogic.cmd file.

⇒ Test the application.
http://localhost:7070/WebHBDBAPP13.

Assignments

1) Write a web based HB Application to perform insert, update, delete & view operations from one form page by taking multiple search boxes.....

GLASSFISH

type : Application Server s/w.
Vendor : SunMs (oracle corp)
Version : 4.x (Compatable with JDK1.7)
Open source
default ports : for Admin Console : 4545
for Http Access : 8080
Allows to create domains.
Default Domain name : domain1
domain1 username : admin
domain1 password : adminadmin

To download s/w : download as zip file from www.glashfish.org (glassfish-4.0.zip)

(or)

download this location.

(Extract the zip file for installation).

Procedure to create user-defined domain in Glassfish4

<Glassfish_home>\bin\asadmin create-domain --user=testuser --adminport=4343 HBGFDomain1

Enter password (Choose password) : testuser

Re-Enter password (choose password) : testuser

Domain created successfully.

To change the "http" port no. of the above "HBGFDomain1" server

<Glassfish_home>\domains\HBGFDomain1\config\domain.xml and modify the port attribute value of first <network-listener> tag → restart domain server.

Procedure to create jdbc con pool for oracle DataSource in "HBGFDomain1" server of Glassfish

Step(1): place ojdbc14.jar/ojdbc6.jar file in <Glassfish_home>\domain\HBGFDomain\lib\ext folder

Step(2): start "HBGFDomain1" server.

<Glassfish_home>\bin\asadmin start-domain HBGFDomain1

Step(3): open Admin console of "HBGFDomain1" server.

http://localhost:4343

user name : testuser

password : testuser

Step(4): create jdbc con pool for oracle

Admin console → Resource → Jdbc → JDBC connection pools → pool name : pool1

Resource type : javax.sql.datasource

DB Driver vendor : oracle

→next→ Application properties : User, DBNAME : Password, servername, Driver Type, sevice name, url, portno → finish.

lauch pool1 → ping (ping succeeded) → save (New values successfully saved).

Step(5): create jdbc datasource pointing to the above jdbc con pool

admin console → resources → jdbc resources(Datasource) → new → Jndi name : DsJndi

pool name : pool1 → OK.

To deploy "WebHBDBApp13" in Glassfish "HBFGDomain1" server.

Rightclick on the project → export → warfile → web project → Finish.

To stop Glassfish Domain server

<Glassfish_home>\bin\asadmin stop-admin HBGFDomain1

Tomcat Sewer

Procedure to create jdbc con pool for oracle in Tomcat 6|7|8 sewers

step(1): Download apache-tomcat-jdbc-1.1.0.1-bin.zip & extract that zip file to get tomcat-jdbc.jar

step(2): Add tomcat-jdbc.jar, ojdbc14.jar / ojdbc6.jar files to <Tomcat_home>|lib folder.

step(3): Write the following <Resource> tag under <context> in <Tomcat_home>|conf|context.xml file.

```
<Resource name="DsJndi" type="javax.sql.DataSource" factory="org.apache.tomcat.jdbc.pool.
DataSourceFactory" driverClassName="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1521:xe" username="scott" password="tiger"
initialSize="10" maxActive="2"/>
```

step(4): Restart/start the sewer.

Procedure to deploy "WebHBDBApp" folder in Tomcat Server

step(1): change the Jndi name to "java:|comp|env|DsJndi" in "connection.datasource" property of hibernate.cfg.xml.

step(2): Deploy the webapplication

Right click on project → export → war file

Project name : WebHBDBApp13

Destination : → Finish

step(3): Test the Application

step(4): http://localhost:3030/WebHBDBApp13.

Annotations Based HB programming

MetaData:

Data about data is called MetaData. Writting more info about existing info is called "MetaData"

⇒ In java programming meta data operations can be done in 3 ways.

(a) using comments;

```
//holds person age
int age = 40;
```

(b) using modifiers

```
public static String name = "raja";
```

(c) using xml files

It is all about resourcing cfg like sewlet programming configuration. Here xml tags provide more information about sewlet program to underlying sewlet container | webserver.

xml files based Metadata operations give good flexibility of modification but gives bad perfomance because the xml paser ie, there to read & process xml doc is quiet heavy.

(d) using Annotations

Annotations java statements which can be used as alternate to xml files fol metadata operations & resource cfgs.

Annotations gives good perfolmance & bad flexibility of modification because they will be written directly in .java file.

NOTE: In most the technology/frameworks the xml files will be utilized overide the cfgs done in annotations.

Annotations fol programming are introduced from jdk1.5 so all technologies/frameworks that are compatible with jdk1.5 suppot annotations.

Syntax: @<Annotations>(param1=val1, param2=val2,--)

⇒ Annotations can be applied class/interface level, middle level, constructol level, param level, return type level, field level, and etc...

HB Annotations

Annotations support in HB is introduced from Hibernate 3.2 based on JPA specification.

JPA is slw specification having set of rules & guidelines to develop ORM slw like HB, IBatis, toplink & etc..

If all ORM slws are designed based on JPA then it becomes easy to move from 1 ORM flw to another ORM flw.

All ORM flws support JPA annotations, due to this even though we change ORM libraries we can walk with same annotations of JPA.

HB supports all the common annotations of JPA & its also having its own annotations supporting those features of HB which are not there in JPA (javax.persistence pkg)

Common JPA Annotations: (JEE Module)

@Entity : To make java class as domain class
@Table : To map java class with DB table
@Column : To java class property with DB table column.
@Id : singular identity field configuration
@Transient : Make the field as non-persistence fields etc..

HB Specific annotations

@GenericGenerator
@Filters
@Filter
@FilterDefs
@FilterDef and etc..

In addition annotations based HB programming there is no need of mapping file (xml) because we add mapping annotation in Domain class, But HB cfg file is required.

By adding mapping file we can override the cfgs done in Domain class through Annotations...

To cfg Annotations based domain classes in HB cfgs file we can use <mapping> with "class" attribute.

```
<mapping class="com.nt.domain.Student"/>
<mapping class="com.nt.domain.Employee"/>
```

Eventhough there are same annotations in JPA & HB we should give first priority to working with to JPA Annotations. If certain operations can not be done using JPA Annotations then we need to walk with HB Annotations.

HB Prg 14 (Basic Annotation)

```
|--> src
      |--> com.nt.cfgs
            |--> hibernate.cfg.xml
      |--> com.nt.model
            |--> Student.java
      |--> com.nt.test
            |--> ClientApp.java
```

Ref App 18 of the page no: 307 & 308

The Annotations of mapping can applied at field level or it getter method level. If table name & domain class is same then applying @Table is optional.

If Domain class property names & DB Tables col names are same then applying @Column is optional.

The Domain class property ie, annotated with @Transient will not participate in any kind of persistence operation.

While working with Annotations the entire persistence logic in client application is going to be same. But we need to place mapping annotations either on the top of fields or on the top of getter methods.

In project contains more mapping files are annotation classes then they will be placed in a jar file & that jar file name will be mentioned in the cfg file as shown below.

```
<mapping jar="mappings.jar"/>
```

The mappings.jar file contains more HB mapping files & more annotations classes.

Generators configuration using annotations

JPA supports 4 generators to generate the identity value they are "Auto", "Sequence", "Identity", "Table". But these generators may not be sufficient in HB programming. So we prefer working with HB supplied generators like "Increment", "seqhilo", "native", "sequence" & etc....

for this we need to use

@GeneratorId value

@GenericGenerator along with @Id (HB)

1) Increment generator configuration

```
@Id
@Column(name="sno")
@GenericGenerator(name="gen1", strategy="increment")
@GeneratedValue(generator="gen1")
private int sno;
```

2) Identity

1) change to my sql & make sure that student-tab db table is available in mysql having "sno" column with pk constraint + auto increment constraint

2) cfg "identity" generator as shown below in student.java

```
@Id
@Column(name="stno")
@GenericGenerator(name="gen1" strategy="identity")
@GeneratedValue(generator="gen1")
private int sno;
```

3) hilo

create helper dbtable with value in any db sw.

mytable

mycol

10

cfg "hilo" generator in student.java as shown below

```
@Id
@GenericGenerator(name="gen1", strategy="hilo", parameters={@Parameter(name="table",
value="mytable"), @Parameter(name="column", value="mycol")
    @Parameter(name="max_lo", value="5")})
@GeneratedValue(generator="gen1")
private int sno;
```

Object versioning through Annotations

1) @Version : JPA Annotation

(keeps track of the no.of times the object/record is modified through HB persistence logic).

2) keep HB Basic Annotation ready.

3) Add the additional column in student_tab table to hold the version value.

SQL> alter table student_tab add ver_col number(5);

4) Add additional property "ver" to keep track of version of the object, having @version, @Column & getter, setter methods.

In student.java

```
@Version
@Column(name="ver_col")
private int ver;
```

5) write code in client application to insert record/ save object.

```
Student st =new Student(1001,"raja","hyd");
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    int id=(Integer) ses.save(st);
    tx.commit();
  s.o.p("object is saved with :"+id);
   }
catch(Exception e)
 {
   tx.rollback();
   e.printStackTrace();
 }
```

SQL> select * from student_tab;

1001 raja hyd 0 (ver_col)

For the above steps based complete Application Ref Appln 19 of the page no : 308

Object Timestamping through Annotations

NOTE : In xml cfg's we can't apply versioning & Timestamp features at a Time. But it is possible through annotations.

@Temporal → for Timestamp enabling (keeps track of when the object is saved/updated)

1) keeps previous versioning Appln ready
2) Add additional column in student_tab of type timestamp.
   SQL> alter table student_tab add lastUpd timestamp.
3) Add additional property in student class having of type java.util.Date having @Temporal with setter & getter method.

In Student.java

```
@Temporal (TemporalType.DATE)
@Column (name="lastUpd")
private Date lastUpd = new Date();
  //setter & getter methods
```

4) Place code in client Appln to save the object
   --- same as previous ----
   --- check the db table ---
5) Place code in client Appln to load & update the object
   --- same as previous ---
   --- check the db table ---

Composite Id field using Annotations

⇒ To configure singular Identity field use @Id

⇒ To configure composite Identity field use @Embeddable, @EmbeddedId

1) Create Student1 Table having composite PK constraint applied on the stno, stname columns.
   SQL> create table student_tab1 (stno number(5), stname varchar2(20), stadd varchar2(20), primary key (stno, stname));
2) Keep App18 of Booklet Ready
3) Design class for Composite PK & Domain class shown below.

StudentPk.java

```
package com.nt.model;

  @Embeddable
Public class StudentPK implements Serializable
{     @Column(name="stno")
      private int sno;
```

```
@Column(name="sname")
private String sname;
//0-param constructor
//2-arg constructor
//setter & getter methods
//toString(-) method
}
```

Student.java

```
package com.nt.model;
@Table(name="Student_tab1")
@Entity
public class Student
{
  @EmbeddedId
  private StudentPK pk;
  @Column(name="stadd")
  private String address;
//0-param constructor
//2arg constructor
//setter & getter methods
//toString(-) method
}
```

4) write the following code in the ClientApp to insert the record

ClientApp.java

```
//save object
StudentPK pk=new StudentPK(1002,"ravi");
Student st=new Student(pk,"hyd");
Transaction tx=null;
try{ tx=ses.beginTransaction();
  StudentPK id=(StudentPK)ses.save(st);
    tx.commit();
    s.o.p("object is saved with:"+id);
}
catch(Exception e)
{ tx.rollback();
  e.printStackTrace();
}
```

5) write the code in ClientAppln to load the object.

ClientApp.java

```
StudentPK pk = new StudentPK (1002, "ravi");
    Student st = (Student) ses.get (Student.class, pk);
      s.o.p(st);
```

⇒ From HB 4.x onwards ServiceRegistry is introduced to hold multiple services that are requiring HB Application execution. This ServiceRegistry can have services like ConnectionPool, mapping, configuration details, interceptors, listeners and etc...

⇒ All these Services can be made available to SessionFactory object through ServiceRegistry. To support this ServiceRegistry concept the cfg.buildSessionFactory() method is depricated from HB 4x & cfg.buildSessionFactory(registry) method is given alternative.

⇒ Every Registry is the implementation of "ServiceRegistry" interface. In HB 4.3 we write the following code to build ServiceRegistry & SessionFactory, Session objects.

```
    //Active HB flw
Configuration cfg = new Configuration();
  //Read mapping.cfg details
    cfg = cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
  //Create Service Registry Builder
StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder();
  // use builder to create Service Registry
ServiceRegistry registry = builder.applySettings(cfg.getProperties()).build();
  //create Session factory object
SessionFactory factory = cfg.buildSessionFactory(registry);
  //create Session
Session ses = factory.openSession();
```

⇒ To perform Bulk operations & to perform single row (or) Bulk operations by using our choices criteria value we need to work with the following 3 concepts of HB.

(a) HQL (HB Query Language)

(b) Native SQL (Direct SQL queries)

(c) Criteria API (or) QBC (Query By Criteria)

HQL

1) It is object based query language
2) It is db s/w independent query language
3) Support both select & non-select operations
4) These queries will be written based on Domain class & its properties
5) Every HQL Query will be converted into SQL Query by HB using ASTL QueryTranslatorFactory.
6) Supports both positional params (?), named params (:<name>)
7) Each HQL query will be represented by query object. It is the object of Java class that implements org.hibernate.Query (I).
8) Gives support for conditional clauses, relational operators, joins, aggregate functions, sub queries & etc..
9) Use list() or iterate() methods to execute HQL select query & use executeUpdate() to execute HQL-non select query.
10) we can't perform DDL operations using HQL.
11) HQL based pl/sql programming is not possible.
12) HQL degrades the performance little bit in the process of generating SQL queries internally.

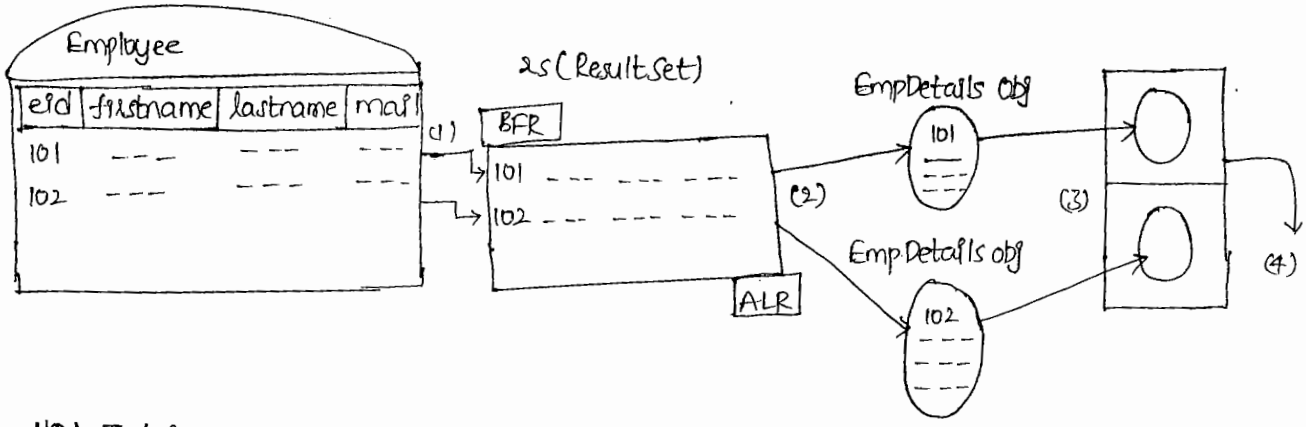
SQL > select * from Employee (table name)
HQL > from EmpDetails (Domain class)
HQL > from EmpDetails as ed (alias name)
HQL > from EmpDetails ed
HQL > from Emp select ed from EmpDetails ed

SQL > select eid, firstname from employee;
HQL > select no, fname from EmpDetails;
HQL > select ed.no, ed.fname from EmpDetails ed;
SQL > select * from employee where eid >=100 and eid <=100
HQL > from EmpDetails where no >=100 and no <=1000
HQL > from EmpDetails ed where ed.no >=100 and ed.no <=1000

SQL > delete from employee where email like '%gmail.com'
HQL > delete from EmpDetails where mail like '%gmail.com'

HBProj18 (HQL)

- Java Resources
  - src
    - com.nt.cfgs
      - hibernate.cfg.xml (HB cfg file)
      - Employee.hbm.xml (HB mapping file)
    - com.nt.domain
      - EmpDetails.java (HB Entity | Domain | POJO class)
    - com.nt.utility
      - HibernateUtil.java
    - com.nt.test
      - HQLTest.java



HQLTest.java

```
package com.nt.test;

public class HQLTest
{
  public static void main(String args[])
  {
    Session ses = HibernateUtil.getSession();
    Query query = ses.createQuery("from EmpDetails");
    List<EmpDetails> list = query.list();
    for(EmpDetails ed:list)
    {
      S.O.P(ed);
    }
  }
}
```

Employee.hbm.xml

```
<!DOCTYPE hibernate-mapping SYSTEM "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="com.nt.domain.EmpDetails" table="Employee">
<id name="no" column="EID" /> <!-- Singular Id field cfg -->
<property name="fname" column="FIRSTNAME" length="20" />
<property name="lname" column="LASTNAME" length="30" />
<property name="mail" column="EMAIL" length="20" unique="true" not-null="true" />
</class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="connection.username">system</property>
<property name="connection.password">siva</property>
<property name="show_sql">true</property>
<mapping resource="com/nt/cfgs/Employee.hbm.xml" />
</session-factory>
</hibernate-configuration>
```

EmpDetails.java

```
package com.nt.domain;
public class EmpDetails {
    private int no;
    private String fname, lname, mail;
    public EmpDetails() {
        S.O.P("EmpDetails:0-param constructor");
    }
//setter and getter methods
@Override
public String toString()
{
    return "EmpDetails [no="+no+", fname="+fname+", lname="+lname+", mail="+mail+"]";
}
}
```

HibernateUtil.java

```
package com.nt.utility;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;

public class HibernateUtil{
    private static SessionFactory factory=null;
   static{
        try{
       Configuration cfg=new Configuration();
          cfg=cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
  StandardServiceRegistryBuilder builder=new StandardServiceRegistryBuilder();
  ServiceRegistry registry=builder.applySettings(cfg.getProperties()).build();
       factory=cfg.buildSessionFactory(registry);
      S.O.P("sessionfactory complete successfully");
   }
   catch(Exception e)
     { S.O.P("enter into factory exception");
      e.printStackTrace();
     }
}
static ThreadLocal<Session> tl=new ThreadLocal<Session>();
static Session ses=null;
  public static Session getSession()
   {  try{
        If(tl.get()==null){
          ses=factory.openSession();
            S.O.P("enter into session");
          tl.set(ses);
         return ses;
        }
      else{
          S.O.P("get into session");
```

```
            return tl.get();
          }
        }//try
        catch (Exception e)
        {
           s.o.p("session exception");
         return null;
        }
    } //getSession();

    public static void closeSession()
    {
        try{
            ses.close();
          }
        catch(Exception e)
        {
          e.printStackTrace();
        }
    }

} // class
```

The points about the diagram:

1) HB fw translate given HQL query into sql query & execute that query in db s/w to Jdbc resultset object.

2) HB fw reads the record from Resultset into Domain class object

3) & 4) These domain class object will be placed in list collection & that will be given to client application.

⇒ In Jdbc to place Resultset records into list collection we need to write lot of additional logic where in HB same thing happens internally & we get list Ds directly.

Executing HQL Query with positional parameters based condition

```
Query query1 = ses.createQuery("from EmpDetails where no>=? and no<=? ");
    query1.setInteger(0,100);
    query1.setInteger(1,200);
List<EmpDetails> list = query.list();
    for(EmpDetails ed: list)
    {
      s.o.p(ed);
    }
```

In Jdbc the positional params(?) index is 1 based indexes. In HB same index is '0' based index. When HQL select query select all cols/properties of records/objects then the generated list collection holds domain class object element values.

From HB 4.0 positional are deprecated & its recommanded to use Named(:<name>) params because if multiple positional params are there we may confuse to identify their index.

```
Query query2 = ses.createQuery("from EmpDetails where mail like :domain");
    query2.setString("domain","%gmail.com");
List<EmpDetails> list = query2.list();
    for(EmpDetails ed:list)
    {
      s.o.p(ed);
    }
```

Assignment

Develop HQL Query that gives employee details based on given 2 last names.

```
Query query3 = ses.createQuery("from EmpDetails where lname in(:ln1,:ln2)");
    query3.setString("ln1","chau");
    query3.setString("ln2","rao");
Iterator<EmpDetails> it = query3.iterate();
    while(it.hasNext())
    {
      EmpDetails ed = it.next();
      s.o.p(ed);
    }
```

The iterate() execute internally like this:

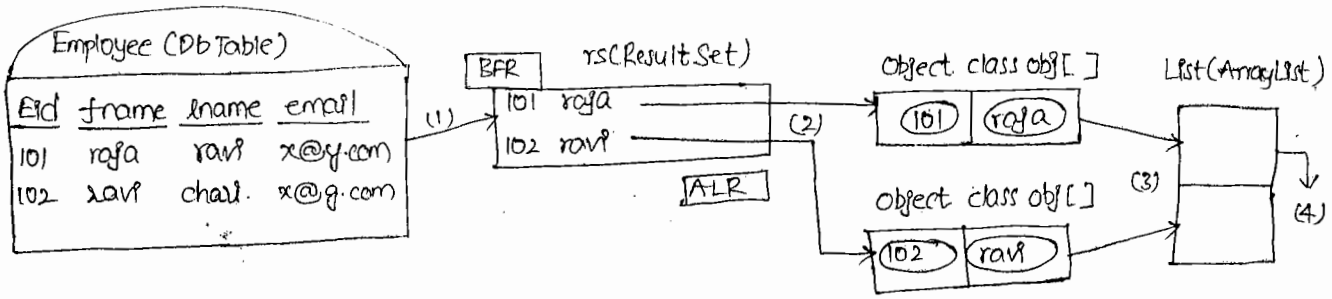
```
select EID from Employee where ----
    101
    102
    103
select * from Employee where eid=101;
    101, ---, ----
select * from Employee where eid=102;
  102 --- --- ---
```

First SQL Query gives only the identity value & the remaining SQL queries gives uses those identity values as the criteria values to get records from DB table lazily (on demand basis).

list()	iterate()
1) generates 1 select query to get n records	1) Generates n+1 query to get n records as shown above.
2) Performs eager loading of objects/records	2) Performs lazy loading of object/records
3) Does not create any proxy obj	3) creates the proxy object
4) Gives list collection	4) Gives Iterator that object pointing to list collection
5) Useful in single layer appln	5) Useful in multilayer Applications

//Executing HQL select query that gives specific multiple column values.

```
Query query4 = ses.createQuery("select no,fname from EmpDetails");
List<Object[]> list = query4.list();
for(Object[] row : list)
{
   for(Object value : row)
   {
      s.o.p("value +" ");
   }
  s.o.p();
}
```



When we use HQL select Query to get Specific multiple column values of DB table (or) Specific multiple property values of objects. The generated list data structure ~~constructs~~ contains java.lang.Object class obj[ ] as the elements.

Arrays are objects in Java so they can be placed as element value of any collection in Java.

If iterate() method is used to execute HQL select query i.e, not selecting all columns/properties of Table/objects then iterate() method does not perform Lazy loading. It behaves same as list method.

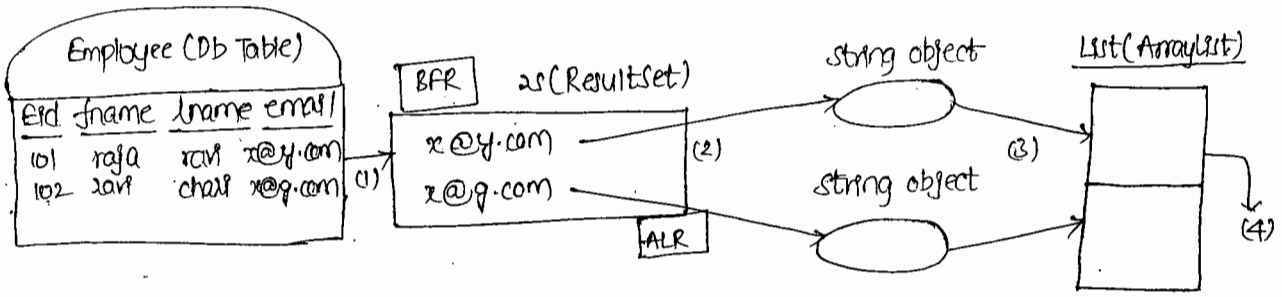
NOTE: If the above code executing using iterate() method no Lazy loading takes place.

```
Query query4 = ses.createQuery("select no,fname from EmpDetails");
Iterator<Object[]> it = query4.iterate();
    while(it.hasNext())
    {
      Object row[] = it.next();
      for(Object value : row)
       {
         S.o.p(value+" ");
       }
      S.o.p("\n");
    }
```

⇒HB Performs lazy loading only while dealing with Domain class object not while dealing with other objects.

```
Query q = ses.createQuery("select mail from EmpDetails");
    List<String> list = q.list();
      for(String email : list)
      {
        S.o.p(email);
      }
```

If HQL select query is used to retrieve property/column value then the generated list data structure contain the property related objects or wrapper objects as the element value.




```
//Executing the HQL select query with subquery, aggregate function

Query query5 = ses.createQuery("from EmpDetails where no =(select max(no) from EmpDetails)");

    List<EmpDetails>  list = query5.list();
        if(list != null)
        {
          EmpDetails ed = list.get(0);
            S.o.p(ed);
        }

//Executing HQL query that gives count of records

Query query6 = ses.createQuery("select count(*) from EmpDetails");

    List<long> list = query.list();
      long cnt = list.get(0);
     S.o.p("Records count "+cnt);
```

Assignment

Write HQL select query that gives Employee details whose having nth highest employee..

HQLTest.java

```
    public class HQLTest {
        public static void main(String args[])
        {   Session  ses = HibernateUtil.getSession();
          Query query = ses.createQuery("from EmpDetails where no = (select max(no)
                                                    from EmpDetails)");
            List<EmpDetails> list = query.list();
             for(EmpDetails ed: list)
               { S.o.p(ed);
               }
    }
```

We can use executeUpdate() method to execute non-select HQL Queries. This method returns a numeric value representing no.of records that are effected. These non-select HQL queries must be executed as Transactional statements.

```
//Execute HQL Delete Query

Query query6 = ses.createQuery("delete from EmpDetails where lname like :ln");
    query6.setString("ln", "chari");
Transaction tx=null;
  try{
      tx = ses.beginTransaction();
     int result = query6.executeUpdate();
      tx.commit();
   S.O.P("No. of records that are effected : "+result);
    }
  catch(Exception e)
  {
    tx.rollback();
   e.printStackTrace();
  }
```

HQL insert query is not given to insert records into DB Table with direct values. It is given to insert records into DB table by selecting records from another DB Table.

Only the inserted into... select... form is supported. not the insert into..values... form.

Usages:

1) Transferring employees from employee DB Table to NGOMember table based on some conditions.
2) Transferring Daily transaction records to from master DBTable another table if the transaction amounts are high etc...

Example Application

employee(same)

NGOMember (cols are same as employee)

HBProg19(HQL Insert)

- Java Resources
  - src
    - com.nt.cfgs
      - hibernate.cfg.xml (HB cfg file)
      - Employee.hbm.xml (HB mapping file)
      - NGOMember.hbm.xml
    - com.nt.domain
      - EmpDetails.java (HB Entity/Domain/POJO class)
      - NGOMember.java
    - com.nt.utility
      - HibernateUtil.java
    - com.nt.test
      - HQLInsertTest.java

hibernate.cfg.xml

same but cfgs & mapping files

```
SQL> create table NGOMember as select * from employee;
SQL> truncate table NGOMember;
```

Employee.hbm.xml

same

EmpDetails.java

same

NGOMember.hbm.xml

same but write class tag as shown below

```
<class name="com.nt.domain.NGOMember" table="NGOMember">
---
</class>
```

NGOMember.java

```
public class NGOMember
{
  same as EmpDetails.java
}
```

Hibernate Util.java

same

HQL InsertTest.java

In main(-) method

```
//Execute HQL Query to copy the records of Employee DB Table to NGOMember DB Table
Query query = ses.createQuery("insert into NGOMember(no,fname,lname,mail)
                    select ed.no, ed.fname, ed.lname, ed.mail from EmpDetails ed
                                where ed.lname = :ln");
    query.setString("ln","rao");
Transaction tx=null;
try{
    tx=ses.beginTransaction();
   int res=query.executeUpdate();
    tx.commit();
   s.o.p("no.of records effected:"+res);
    }
catch(Exception e){
    tx.rollback();
    e.printStackTrace();
    }
```


NOTE: use ses.save(-) or ses.persist(-) for inserting single record in DB table.

Named HQL queries:

The HQL query is, placed in mapping file having logical name is called NamedHQLQuery.
This query is useful to provide global visibility to HQL query in multiple sessions of a one(or) more client applications. This query is also useful to get the flexibility of modification without disturbing the java code. In Real world the HQL query will not be placed directly in client Applications or DAO classes rather they will be placed in mapping file as "NamedHQLQueries" using query tags.

Ex:

place NamedHQLQuery in Mapping file.

Employee.hbm.xml

```
<class...
</class>
<query name="get_Emp_Details"> from EmpDetails where mail like :domain </query>
```

In one (or) more client application write following code to access & execute the NamedHQLQuery

```
Query query = ses.getNamedQuery("get_Emp_Details");
    query.setString("domain", "%x.com");
List<EmpDetails> list = query.list();
for(EmpDetails ed:list)
{
    S.O.P(ed);
}
```

Employee.hbm.xml

```
<query name="change_lastName">
<![CDATA[ from EmpDetails set lname='rao' where no>=:min and no<=:max ]]>
</query>
```

//HQLInsertTest.java

```
Query query = ses.getNamedQuery("change_lastName");
    query.setInteger("min", 1000);
    query.setInteger("max", 2000);
Transaction tx=null;
try{
    tx = ses.beginTransaction();
    int res = query.executeUpdate();
    tx.commit();
    S.O.P("no.of records updated:"+res);
}
catch(Exception e)
{
    tx.rollback();
    e.printStackTrace();
}
```

⇒ While placing NamedHQL Query in mapping file we should take care of '<' symbol problem Because it applies xml meaning instead of the conditional operator meaning.

⇒To overcome this problem use either '<' or <![CDATA[---]]> as shown above we can place multiple NamedHQLQueries in Mapping file....

⇒If make HQL queries as NamedHQL Queries are

@NamedQueries
@NamedQuery Annotations(JPA)

use @NamedQuery --> To perform each NamedHQLQuery on the top of the class.

use @NamedQueries --> To hold multiple @NamedQuery annotations. upto jdk1.7 we can't place same annotations for multiple times on the top of same class or method or field etc...

To overcome this problem they give another annotation to hold other annotations for multiple times.

@NamedQueries annotation is given to hold multiple @NamedQuery annotations.

Example

```
@Table(name="Student_Tab")
@Entity
@NamedQueries(values={@NamedQuery(name="get_Addrs", query="select address from
                    student"),
        @NamedQuery(name="update_Addrs", query="update student set address=:ad
                                    where sno=:no")})
public class Student
{
 ---
 ---
}
```

ClientApp.java

```
Query query1 = ses.getNamedQuery("get_Addrs");
  List<String> list = query.list();
    for(Student address:list)
     {
       s.o.p(address);
     }
Query query2 = ses.getNamedQuery("Update_Addrs");
      query2.setString("ad", "new delhi");
      query2.setInteger("no", 1001);
  Transaction tx=null;
    try{ tx=ses.beginTransaction();
         int res = query2.executeUpdate();
           tx.commit();
      } s.o.p("No.of records updated"+res);
catch(Exception e)
{
   tx.rollback();
   e.printStackTrace();
}
```

NATIVE SQL QUERIES :

⇒ These are DB s/w specific SQL Queries, If certain operations are not possible with HQL(or) complex with HQL then use native SQL Query. These operations are like calling PL/SQL procedures/functions insert record with values, performing DDL operations etc...

1) These queries based persistence logic is DB s/w dependent.
2) SQL query object represents each native SQL query. It is the object of java class that implements SQLQuery(I) & It is sub interface of Query(I).
3) Allows both select, non-select & DDL operations
4) Useful to call pl/sql procedures & functions.
5) Allows to make the queries as named native SQL queries
6) Allows both named & positional params use list() (no interfaces) to execute select queries & use executeUpdate() method for non-select Queries.
7) The select query that implemen returns all column values is called Entity Query & The select query that returns single column values is called Scalar Query.
8) These are real sql queries will be written based on the table names & column names.

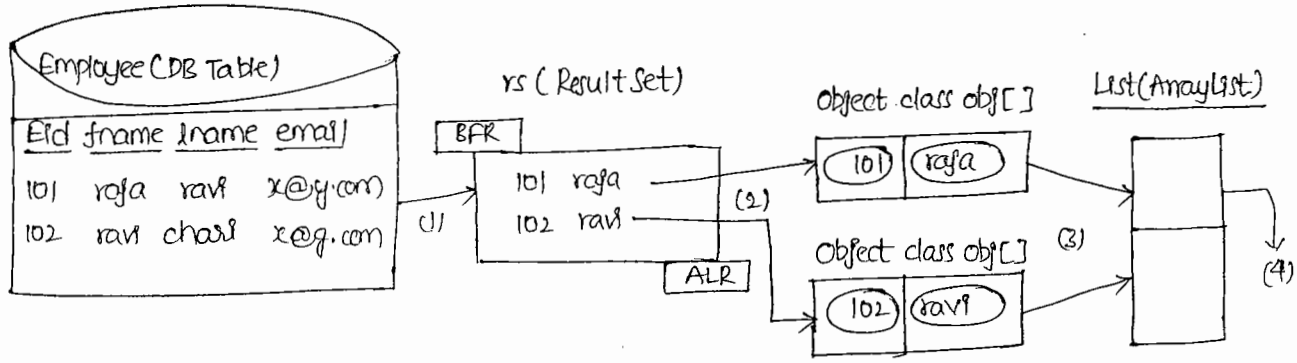
HB Proj 21

```
|→ JavaResources
    |→ src
        |→ com.nt.cfgs
            |→ hibernate.cfg.xml (HB cfg file)
            |→ Employee.hbm.xml (HB mapping file)
        |→ com.nt.domain
            |→ EmpDetails.java (HB Entity/Domain/POJO class)
        |→ com.nt.utility
            |→ HibernateUtil.java
        |→ com.nt.test
            |→ NativeSQLTest.java
```

NativeSQLTest.java

```
//prepare NativeSQLQuery(EntityQuery)
SQLQuery query = ses.createSQLQuery("select * from employee where email like :domain");
        query.setString("domain", "%.y.com");
List<Object[]> list = query.list();
for(Object[] row:list)
{
 for(Object val:row)
  {
   S.O.P(val+" ");
  }
 S.O.P();
}
```

⇒The above code gives list object having Object class object arrays. Because we did not specify domain class name to map each record into domain class object.



⇒In the above code if you specify Domain class name using addEntity() method then each record retrieve from table will be stored into one object of Domain class (DC) with respect to above code

```
query1.addEntity(EmpDetails.class);
List<EmpDetails> list = query1.list();
for(EmpDetails ed:list)
{
 S.O.P(ed);
}
```

Here the generated list object contains generated "domain class" object.

⇒ We can't execute NativeSQL select query by using iterate() method because it sends & execute query in database slw directly. So there is no provision to get the records from DBTable lazily.

// Executing Native SQL scalar query (selecting specific column values of DB Table)

In the above environment the generated list datastructure contains java.lang.Object class object array(obj[])

NativeSQLQueryTest.java

```
SQLQuery query1 = ses.createSQLQuery("select eid,mail from employee where lastname
                                                                    like :ln");
    query1.setString("ln","rao");
List<Object[]> list = query1.list();
for(Object[] row:list)
 {
   for(Object val:row)
    {
      S.O.P(val+" ");
    }
   S.O.P();
 }
```

1) In NativeSQL,HQL queries we can place both named & positional parameters but we must place all positional parameters before any named parameters is defined.

2) We can take parameters in the query representing only the input values not representing table|domain class & column names|property names.

//Executing Native SQL select query that gives single column values both named & positional params..

```
SQLQuery query1 = ses.createQuery("select fname from employee where eid>=? and
                                                                    eid<=max");
    query1.setInteger(0,1000);
    query1.setInteger("max",2000);
List<String> list = query1.list();
    for(String fname:list)
    {
      S.O.P(fname);
    }
```

```
//Executing Non-select SQL Query

SQLQuery query2 = ses.createQuery("insert into employee values( :val1, :val2, :val3, :val4 )");
        query2.setInteger ("val1", 111);
        query2.setString  ("val2", "raja");
        query2.setString  ("val3", "rao");
        query2.setString  ("val4", "rao@x.com");

Transaction tx=null;
try{
    tx=ses.beginTransaction();
   int res = query2.executeUpdate();
     tx.commit();
   if(res==0)
     S.O.P("record not inserted");
   else
     S.O.P("record inserted");
  }
 catch(Exception e)
  { tx.rollback();
    e.printStackTrace();
  }
```

The NativeSQLQuery fe, placed in HB mapping file having logical name is called NamedNativeSQLQuery. This query gives global visibility in multiple session object of a one or more client Applns.

for this we need to use <sql-query> tags of mapping file.

Example

1) Placed NamedNative SQL Query in Mapping file

In Employee.hbm.xml

```
<sql-query name = "get_EmpDetails">
<return class = "com.nt.domain.EmpDetails"/>
<![CDATA[select * from employee where eid>=:min and eid<=:max]]>
</sql-query>
```

2) Access NativeSQLQuery & execute the query from Client Appln

```
Query query = ses.getNamedQuery("get_EmpDetails");
        query.setInteger ("min", 1000);
        query.setInteger ("max", 2000);
List<EmpDetails> list = query.list();
    for(EmpDetails ed:list)
    { S.O.P(ed);
```

Annotations based NativeSQL

To make Native SQL queries as NamedQueries using annotations we need to work with

@NamedNativeQueries

@NamedNativeQuery annotations.

Example

1) keep appln ready (Annotations Basic)

2) Place the following annotations on the top of the class having NamedNativeQueries

```
@Table (name="student_Tab")
@Entity
@NamedNativeQueries ( value={ @NamedNativeQuery (name="getStuds",
        query="select * from student_tab where stadd=:city",
            resultClass=com.nt.model.Student.class),
    @NamedNativeQuery( name="delstuds", query="delete from student_tab
                    where stadd=:city") } )
public class student
{
  ----
  ----
}
```

3) Access & execute NamedSQL Queries from the client Application

```
Query q1=ses.getNamedQuery("getStuds");
    q1.setString("city","hyd");
List<Student> list=q1.list();
for(Student st:list)
{
  s.o.p(st);
}

Query q2=ses.getNamedQuery("delStuds");
    q2.setString("city","new delhi");
Transaction tx=null;
    try{
        tx=ses.beginTransaction();
        int res=q2.executeUpdate();
        tx.commit();
        s.o.p("no of records deleted:"+res);
    }
    catch(Exception e){
        tx.rollback();
        e.printStackTrace();
```

⇒ In order to centralize persistence logic & to make persistence logic reusable in multiple applications of a module & in multiple modules of project we need to work with PL/SQL procedure function.

⇒ PL/SQL procedure does not return a value where function returns a value. Instead of authentication logic in every module as SQL Query it is recommended to write as pl/sql procedure (or) in multiple modules of a project.

⇒ HB 3.x onwards we call pl/sql procedures/functions but they must be developed in the style they are required for HB by following set of rules will vary based on the db sw we use.

For Oracle Specific rules

1) PL/SQL procedure must have first parameter as out parameter of type sys_refcursor
2) PL/SQL functions must return cursor having sys_refcursor as return type.

Some Common Rules:

Use SQL92 syntax to call pl/sql procedure or function

{ call <procedure name> (<param>) }

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

⇒ Pagenation can't be applied on the results (displaying records part by part)

⇒ Cursor is a variable in oracle pl/sql programming that can hold bunch of records it is like "jdbc" result set objects...

Example Appln on calling PL/SQL function from HB Appln

1) Keep any HB Appln ready
2) Develop pl/sql function in oracle satisfying hibernate rules

```
SQL> create or replace function Get_Emp_Details(stno in number, endno in number) return
     sys_refcursor as details sys_refcursor;
     begin
      open details for select * from employee where eid >= stno and eid <= endno;
     return details;
     end;
SQL> show errors --- If any errors occured use this
```

The above select query generated o/p (output) will be stored in the cursor.

3) Prepare NamedNativeSQLQuery in HBMapping file having "callable = true" value.

In Employee.hbm.xml

```
<sql-query name="fxtest" callable="true">
<return class="com.nt.domain.EmpDetails"/>
```

```
<!= call Get_Emp_Details (:min, :max) }>
</sql-query>
```

4) Access & execute NamedSQLQuery in client Appln to call pl/sql functions of DB Sw.

In client Application

```
Query q1 = ses.getNamedQuery("Get_Emp_Details");
        q1.setInteger("min", 1000);
        q1.setInteger("max", 2000);
List<EmpDetails> list = q1.list();
for(EmpDetails ed:list)
  {
  s.o.p(ed);
  }
```

For above steps based appln refer appln No: 20 page no: 62 (supplementary booklet)

Example on calling pl/sql Procedure (scalar query)

1) Keep any HB Appln ready

2) Create pl/sql procedure in oracle as required for HB.

```
SQL> create or replace procedure Get_Emp_Details_For_Mail (details out Sys_refcursor
                                                    domain in varchar) as
begin
  open details for
    select firstname, lastname from employee where email like domain;
    end;
SQL> show errors--- If any errors occured use this
```

3) Prepare NamedNativeSQLQuery in HB Mapping file having "callable = true" value.

In Employee.hbm.xml

```
<sql-query name="proc_test" callable="true">
<return scalar column="firstname" type="string"/>
<return scalar column="lastname" type="string"/>
<!= call Get_Emp_Details_For_Mail (?, :domain) >
</sql-query>
```

4) Access & execute NamedSQLQuery in client Appln to call PL/SQL functions of DB Sw.

In client Appln

```
Query q1 = ses.getNamedQuery("Get_Emp_Details");
        q1.setString("domain", "%x.com");
      List<Object[]> list = q1.list();
       for(Object[] raw : list)
        {
          for(Object val : raw)
           {
             S.o.p(val);
           }
        }
```

For above steps based appln ref app NO: 21 page no: 65 (supplementary booklet)

Calling PLISQL Procedure fe, not satis-fying HB rules:

for this we can use either ses.doWork or) ses.doReturningWork(-) methods:

```
ses.doWork(new Work(){
   public void execute(Connection con)
    {
      ---
      ---//jdbc code...
    }
});
```

⇒Here ses.doWork() is called Annonymous inner class object that implements Work(I) overriding execute(-) method of Work(I). use this when u don't want to return any value from the jdbc code of execute() method.

```
String res=ses.doReturningWork(new ReturningWork<String>(){
        public String execute(Connection con)
         {
           ---
           ---
           return...;
         }
});
```


use this when the jdbc code of execute() wants to return a value.

Example

1) create procedure in oracle db sw having authentication logic.

DB Table:(NO HB rules)

Userlist

Uname	pwd
raja	rani
King	kingdom

```
SQL> create or replace procedure Auth_pro (user in varchar, pass in varchar, result out
                                                                    varchar) as
        cnt number;
    begin
    Select count(*) into cnt from userlist where uname=user and pwd=pass;
    if(cnt<>0) then
    result := 'valid credentials';
    else
    result := 'In valid credentials';
    end if;
   end;
```

2) Develop the Appln as shown below

HBProg25

```
|->javaresources
    |->src
        |->com.nt.cfgs
            |->hibernate.cfg.xml
        |->com.nt.utility
            |->HibernateUtil.java
        |->com.nt.test
            |->ProcedureTest.java
```

hibernate.cfg.xml

same pointing to oracle but no mapping file is required

HBUtil.java

same

3) Write following code in the client Appln (ProcedureTest.java)

In ProcedureTest.java

```
String res = ses.doReturningWork(new ReturningWork<String>(){
  public String execute(Connection con) throws SQLException{
  CallableStatement cs = con.prepareCall("{call Auth-pro(?,?,?)}");
      cs.registerOutParameter(3, Types.VARCHAR);
      cs.setString(
      cs.setString(
      cs.execute();
String res= cs.getString(3);
  return res;
   }
  });
S.O.P(result);
```

Assignment

⇒ call plsql procedure of DBslw by using Annotations

Hint : JPa's @NamedNativeQuery annotation is not having callable param. so use HB's @NamedNativeQuery Annotation.

Criteria API (QBC : Query By Criteria)

1) Allows to work with objects & method calls to develop DB s/w independent persistence logic.
2) HB generates optimised SQL queries to improve the performance based on the given QBC logic.
3) This is very useful in page nation & report generation.
   (Displaying records part by part in multiple pages).
4) Allows to perform both single row & bulk operations....
5) QBC logic will be developed without any Queries support
6) QBC can be used to retrieve single (or) multiple specific or all column's/property values...
7) Support to perform aggregate operations.
8) Does not support non-select operations.
9) Each criteria object represents 1 QBC logic. In that we can add multiple conditions in the form of criterian objects.
10) Criteria API means org.hibernate.Criterian pkg & its sub package...

HBProg26 (QBC)

```
|→ javaresources
   |→ src
      |→ com.nt.cfgs(same)
         |→ hibernate.cfg.xml (HB cfg file)
      |→ com.nt.domain(same)
         |→ EmpDetails.java (HB Entity/Domain/POJO class)
      |→ com.nt.utility(same)
         |→ HibernateUtil.java
      |→ com.nt.test
         |→ QBCTest.java
```


QBCTest.java

```
Criteria ct = ses.createCriteria(EmpDetails.class);
//It gives select * from employee;
List<EmpDetails> list = ct.list();
for(EmpDetails ed:list)
{ s.o.p(ed);
}
```

NOTE : Criteria object means it is object of a class that implements org.hibernate.Criteria(I).

⇒ Criteria logic (QBC Logic) internally uses Metadata concepts on given class to know class name, properties and uses that information to get table, column names by using this table name and column names queries will be generated.

⇒ Conditions in QBC can be taken as criterion objects. These criterion objects will be created by using various static methods of "Restriction" class.

```
Criteria ct=ses.createCriteria(EmpDetails.class);
Criterion cond1=Restrictions.between("no",1000,2000);
    ct.add(cond1);
List<EmpDetails> list=ct.listC();
 for(EmpDetails ed:list)
  { s.o.p(ed);
  }
//Executing QBC logic with conditions.
/* Criteria ct=ses.createCriteria(EmpDetails.class);
Criterion cond1=Restrictions.in("lname", new Object[]{"chau","rao"});
Criterion cond2=Restrictions.like("mail","%x.com");
   ct.add(cond1);
   ct.add(cond2);
    (or)
Criterion cond3=Restriction.and(cond1,cond2);
     ct.add(cond3);
List<EmpDetails> list=ct.listC();
 for(EmpDetails ed:list)
  { s.o.p(ed);
  }
*/
Criteria ct=ses.createCriteria(EmpDetails.class);
Criterion cond1 = Restrictions.ge("no",1000);
Criterion cond2 = Restrictions.le("no1",2000);
Criterion cond3 = Restrictions.like("mail","%x.com");
Criterion andcond = Restrictions.and(cond1,cond2);
Criterion finalcond=Restrictions.or(andcond,cond3);
  ct.add(finalcond);
   List<EmpDetails> list=ct.list();
    for(EmpDetails ed:list)
     { s.o.p(ed); }
```

⇒If the methods of Restrictions class are not allowing to frame conditions based on Domain class & its properties then we can frame such conditions by using SQL.

For that use Restrictions.sqlRestriction() method.

```
Criteria ct=ses.create Criteria(EmpDetails.class);
Criterion cond1 = Restrictions.sqlRestriction("email like '%x.com' ");
        ct.add(cond1);
List<EmpDetails> list = ct.list();
   for(EmpDetails ed:list)
   {
      s.o.p(ed);
   }
```

We can add order object to criteria object to specify order of retrieving content either in ascending order (or) descending order.

```
Criteria ct=ses.create Criteria(EmpDetails.class);
Criterion cond1 = Restrictions.sqlRestriction("email like '%x.com' ");
        ct.add(cond1);
Order od = Order.desc("frame");
        ct.addOrder(od);
List<EmpDetails> list = ct.list();
 for(EmpDetails ed:list)
 {
    s.o.p(ed);
 }
```

⇒ In QBC to get specific values & we can work projection for that perform the following operation.

a) prepare projection objects pointing to properties
b) add position objects to projectionList objects
c) set projectionList to criteria objects
d) execute QBC logic

```
Projection p1 = Projections.property("no");
Projection p2 = Projections.property("fname");
ProjectionList p3 = Projections.projectionList();
    p3.add(p1);
    p3.add(p2);
    ct.setProjection(p3);
   List<Object[]> list = ct.list();
    for(Object[] row : list)
     {
       for(Object val:row)
       {
         S.O.P(val+" ");
       }
       S.O.P();
     }
```


⇒ We can also use ~~properties~~ projections to perform aggregate operations on the date of table objs. we can add criterian object, order object & projections object ~~to~~ on criteria objects.

```
Projection p1 = Projections.count("no");
Projection p2 = Projections.min("no");
Projection p3 = Projections.max("no");
Projection p4 = Projections.avg("no");
ProjectionList p5 = Projections.projectionList();
 p5.add(p1);
    ct.setProjection(p5);
List<Object[]> list = ct.list();
   Object[] row = list.get(0);
  S.O.P("count"+row[0]+"min"+row[1]+"max"+row[2]+"avg"+row[3]);
```

For complete Ex.appln on QBC Refer Appln 23 of Pageno: 69 (new booklet).

NOTE Related to QBC logics there is no annotations because the whole QBC logic will be written as Javacode & no cfgs are required in mapping file.

HIBERNATE FILTER

1) Hibernate filter is a global, named and parameterized condition placed in mapping file having the ability to apply on multiple session objects.

2) Once filter is enabled on the HB session all the queries of that session will be executed having filter condition until that filter is disabled.

3) org.hibernate.Filter(I) implementation class object represents each HB filter.

4) To define filter name & param names will be we use <filter-def> tag & to specify the filter condition we use <filter> tag.

5) To enable filter on session object use ses.enableFilter(-) method, similarly to disable filter use ses.disableFilter(-) method.

6) HB filter is no way related with Sewlet filter.

7) In order to provide global visibility to complex conditions we can use HB filter.

Example

1) Keep any HB Appln ready

2) Place filter definition in mapping file specifying the name of the filter & param names of the filter using <filter-def> tag.

In Employee.hbm.xml

```
<filter-def name="Emp_NO-Filter">
<filter-def name="min" type="int"/>
<filter-param name="max" type="int"/>
</filter-def>
```

3) prepare filter condition by using filter param names with suppoit of <filter> tag.

```
<class name="com.nt.domain.EmpDetails" table="Employee">
---
---
---
<property name="mail" column="Email"/>
<filter name="Emp_NO_Filter" condition="eid">=:min and eid &lt;=:max">
</class>
```

4) In client Appln enable & disable filter as shown below.

In ClientAppln

```
Filter filter = ses.enableFilter("Emp_NO_Filter");
        filter.setParameter("min", 1000);
        filter.setParameter("max", 2000);
Query query = ses.createQuery("from EmpDetails"); //executes with FilterCondition
List<EmpDetails> list = query.list();
for(EmpDetails ed: list)
 { s.o.p(ed);
 }
ses.disableFilter("Emp_NO_Filter");
Query query1 = ses.createQuery("select count(*) from EmpDetails"); //executes with Filter Cond.
List<Long> list1 = query1.list();
    s.o.p("Record count" + list1.get(0));
```

⇒We can apply HB Filters on HQL Query logics & QBC logics. But it can't applied on native SQL logics because native SQL query goes to db slw directly for execution without any conversion.

For above steps based complete Example Appln 24 on 72 pageno.

Annotations based HB Filter

⇒There are no JPA annotations to work with HB Flws because the concept filter is specific to HB and not there in JPA. The HB supplied annotations are

- @FilterDef - To define each filter definition
- @FilterDefs - To group multiple filter Definitions
- @Filter - To specify each filter condition
- @Filters - To group multiple filter conditions.
- @ParamDef - To define each filter parameter

Example

1) Keep any annotation Example Ready
2) Add the following annotations on the top of Student class.

Student.java

```
@Table (name ="student_Tab")
@ Entity
@ FilterDef(name="stno_filter", parameters :{ @ParamDef(name ="min", type ="int"),
                                              @ParamDef(name="max", type="int")})
@ filter( name = "stno_filter" condition ="stno>:min and stno <=:max")
public class Student
{
---
---
}
```

In clientApp.java

```
Filter filter = ses.enableFilter("stno_filter");
    filter.setParameter ("min", 1000);
    filter.setParameter("max", 2000);
Query query = ses.createQuery( "from EmpDetails"); //executes with FilterCondition
  List<Student> list = query.list();
    for (Student ed: list)
      { s.o.p(ed);
      }
  ses.disableFilter("stno_filter");
```

The above steps based Appln 25 of pageNO:74

⇒ Working with HB Filters is not industry standard because they make the programmer to write query & conditions seperately. Due to this debugging problem & fixing of problems becomes quite complex.

⇒ HB allows to perform the following O-R mapping cfgs. They are

a) Basic o-r mapping
b) Component mapping
c) Inheritance Mapping
d) Collection Mapping
e) Association Mapping (Relationships in HB)

Component Mapping

⇒ To make one class using another class we can go for either composition (or) inheritance.

Composition (has-A relationship)

```
class B
{
---
----
}
class A
{
B b = new B();
---
}
```

Inheritance (IS-A relationship)

```
class B
{
---
---
}
class A extends B
{
---

}
```


⇒ When Domain classes are in composition & wants to be mapped with single DB table then we can not use basic o-r mapping & that needs special O-R mapping is called component mapping.

Q) What is the problem of Granularity?

⇒ When Domain classes are there in composition and if they want to be mapped with single DB table raises the situation of more granuals (more classes) pointing to single table. This situation is nlb problem of granularity. This problem can be solved by using basic o-r mapping & we should use component mapping for that problem.

⇒The Domain class property i.e, mapping single column is called simple property (use <property> tag).

The property that contains multiple sub properties & mapped with multiple columns of DBTable is called component property.

(use <component> tag for that property etc)

Example

JobType.java

```
public class JobType
{
   private String job;
   private int department;
   private long salary;
  //write setter & getter methods;
  ---
}
```

Person.java

```
public class Person
{
   private int pid;
   private String pname;
   private JobType pjob;
   //write setter & getter methods;
  ---
}
```

DBTable

person_tab

pid (PK)
pname
job
department
salary

In hibernate Mapping file

```
<class name="<pkg>.Person" table="person_tab">
<id name="pid" />
<property name="pname" />
<component name="pjob" class="<pkg>.JobType">
<property name="job" /> <property name="department" /> <property name="salary" />
</component> </class>
```

⇒ Complete Example Appln 26 pageno: 76 to 79 (new booklet).

Annotations based Component Mapping

use @Embeddable @Embedded annotations (JPA)

```
@Embeddable
public class JobType
{
  @Column(name="job")
  private String job;
  @Column(name="department")
  private int department;
  @Column(name="salary")
  private double salary;
  //setter & getter methods
  ---
  ---
}
@Entity
@Table(name="person_tab")
public class Person
{
  @Id
  @Column(name="pid")
  private int pid;
  @Column(name="pname")
  private String pname;
  @Embedded
  private JobType pjob;
  //write setter & getter methods.
  ---
}
```


HBProj30(Component Mapping annotations)

- Java Resources
  - src
    - com.nt.cfgs → hibernate.cfg.xml (HB cfg file)
    - com.nt.domain (same) → Person.java, JobType.java
    - com.nt.utility (same) → HibernateUtil.java
    - com.nt.test → InsertTest.java, SelectTest.java

Refer Appln 27 of page no : 79 (new booklet)

Inheritance Mapping

⇒ Since Domain classes are pojo classes we can place them in inheritance to take the advantage of extensibility, but we can't place DB tables in the inheritance. We can't use basic O-R mapping to map the domain classes of inheritance getting mapped with one or more DB tables. This is called subtypes problem.

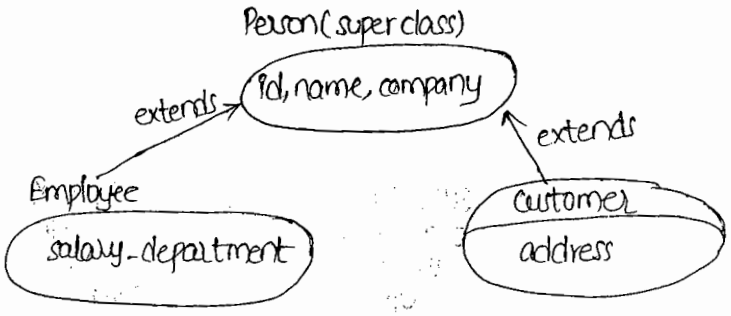
⇒ In order to overcome the "subtypes" problem n|b mapping the classes of inheritance with one(or) more DB tables, we can use inheritance mapping.

The 3 strategies of inheritance mapping

a) Table per class hierarchy (All the classes of inheritance hierarchy will use single DB table)

b) Table per sub class. (Every class of inheritance hierarchy will use its own DB table & those DB tables will be there in relationships)

c) Table per concrete class (same as cb), but db tables will not there in the relationship).



a) Table per class hierarchy

1) All the classes of inheritance will be mapped with single DB-table.

2) DB table contains columns to store the property values of superclass & subclasses.

3) DB table can have descriminator (seperation) column to know which record is inserted by using which Domain class and this column will not be mapped with any property.

4) Here <class> tag will be used cfg superclass and <subclass> tags will be used to cfg subclasses. For every class cfg we can specify <descriminator> and that value will be inserted in descriminator column when that class is used to insert the record/save the obj.

5) To cfg descriminator column use <descriminator-column> tag.

in_persons

id(pk)	name	company	salary	department	address	person_type (Desc column)
101	raja	HCL	----	----	----	person
102	ravi	Wipro	80000	1001	----	employee
103	ramesh	Samsung	----	----	Hyd	customer

Mapping File

```
<class name="<pkg>.Person" table="in-persons" descriminator-value="person">
<id name="id"/>
<descriminator-column="person_type" type="string"/>
<property name="native"/>
<property name="company"/>
<subclass name="<pkg>.Employee" descriminator-value="employee">
<property name="salary"/>
<property name="department"/>
</subclass>
</class>
```

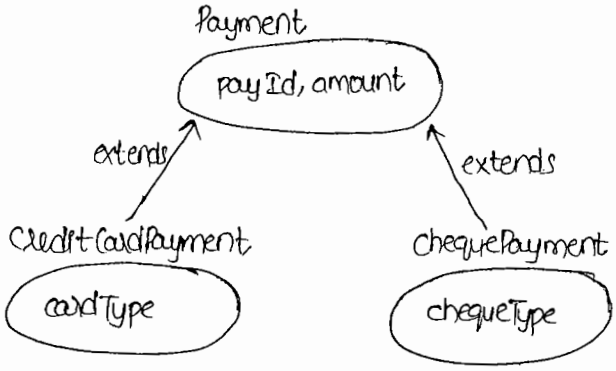
Limitations with Table per class hierarchy model of inheritance mapping

1) we need to take big table with huge columns to hold the values all the classes inheritance hierarchy. This is against of Real world DB Design.
2) We can't apply not null constraint on the column of subclass properties.
3) To work with descriminator column values we need to use NativeSQL support.

HBProj31

- src
  - com.nt.cfgs
    - hibernate.cfg.xml
    - person.hbm.xml
  - com.nt.domain
    - Person.java
    - Employee.java
    - Customer.java
  - com.nt.utility
    - HBUtil.java
  - com.nt.test
    - InsertTest.java
    - SelectTest.java

b) Table Per SubClass:



⇒ Every class of inheritance hierarchy will have own DB table, But all these db tables participate in relationships.

⇒ The DB tables of super class & the DB tables of subclasses will be placed in association (one-to-one) through FK column.

⇒ Discriminator column is optional because every class is having its own DB Table.

⇒ use <class> to cfg superclass & use <joined-subclass> tags to cfg subclasses as shown below.

In_persons2 (parent table)

Id(PK)	name	company
101	raja	HCL
102	ravi	Wipro
103	roja	Info

In_employees (child table1)

department	salary	person_id (FK)
1001	8000	102

In_customers (child table2)

address	person_id
hyd	103

The Mapping File

```
<class name="com.nt.model.Person" table="in_person2">
<id name="id"/>
<property name="name"/>
<company name="company"/>
<joined-subclass name="com.nt.model.Employee" table="in_employee2">
<key column="person_id"/> (fk id) (90% of 'key' will used in foreign key)
<property name="salary"/>
<property name="department"/>
</joined-subclass>
<joined-subclass name="com.nt.model.customer" table="in_customer2">
<key column="person_id"/>
<property name="address"/>
</joined-subclass>
</class>
```

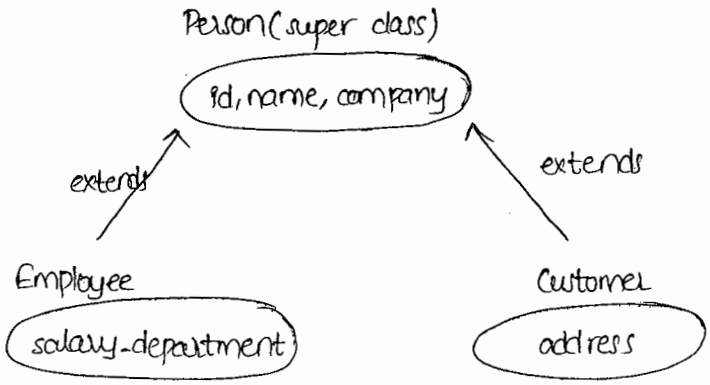
⇒ This model of inheritance mapping is industry standard & recommanded to use because of instead of maintaining huge amount of data in 1 table. It is maintaining the data in multiple tables keeping them in Association. The columns of child class properties can be applied with not null constraint.

⇒ There is no need of working with Native SQL.

HBProj32 (Inheritance Mapping 2-TPSC)

```
|→ src
    |→ com.nt.cfgs
        |→ hibernate.cfg.xml
        |→ person.hbm.xml
    |→ com.nt.domain
        |→ Person.java, Employee.java, Customer.java
    |→ com.ut.utility
        |→ HBUtil.java
    |→ com.nt.test
        |→ InsertTest.java
        |→ SelectTest.java
```

Table Per concrete class



⇒ Here every class inheritance hierarchy will have its own DB table but these DB Tables will not be there in Association. Note ever the DBtables of subclasses will maintain columns representing super class properties & their subclass properties

⇒ In the above discussion there is a possibility of having duplicate columns in multiple DBtable which is against of industry standard.

⇒ Use multiple <class> tags to cfg the multiple classes of inheritance hierarchy. In this process while cfg sub class we need to cfg super class properties & sub class properties.

⇒ As alternate to above mapping we can use <class> to cfg superclass & <union-subclass> to cfg subclasses. Here while cfg subclass we need to cfg only subclass properties.

In persons3 (table 1)

Id(pk)	name	company
101	raja	HCL(person)

In_employees3 (table2)

Id(pk)	name	company	department	salary
345	ravi	wipro	1001	9000 (Employee)

In customers3 (table3)

Id(pk)	name	company	address
342	karan	Infosys	Hyd(Customer)

NOTE

table2, table3 are having duplicate columns of table 1, so it is not industry standard to follow.

Mapping file: (version 1)

```
<class name = "pkg.Person" table="in-persons3 ">
  ---   //super class properties to cfg
</class>
<class name = "pkg.Employee " table = "in-employees3">
  ---
  ---  //superclass properties to cfg
  ---
  ------  //sub class properties cfg
</class>
```

Mapping file: (version 2)

```
<class name="pkg.Person" table = "in-persons3">
  ---
  ---  // super class properties to cfg
</class>
<union-subclass name = "pkg.Employee" table = "in-customers3">
  ---
  ---  //subclass1 properties to cfg
  ---
</union-subclass>
<union-subclass name = "pkg.Employee" table = "in-customers3">
  ------  //subclass 1 properties to cfg
</union-subclass>
```

HBProj33 (Inheritance Mapping 3 - TPCC)
Same as deployment structure

⇒ Table for sub class strategy of inheritance mapping is the recommended approach to follow while implementing real time scenarios.

Annotations based Inheritance Mapping:

JPA Annotations are

@Inheritance → To specify Inheritance Mapping strategy

@Inheritance (strategy = InheritanceType.SINGLE_TABLE)

(Table per class hierarchy)

@Inheritance (strategy = InheritanceType.JOINED)

(Table per subclass)

@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)

(Table per concrete class)

@DiscriminatorColumn → To specify the name of discriminator column

@DiscriminatorValue → To specify discriminator value

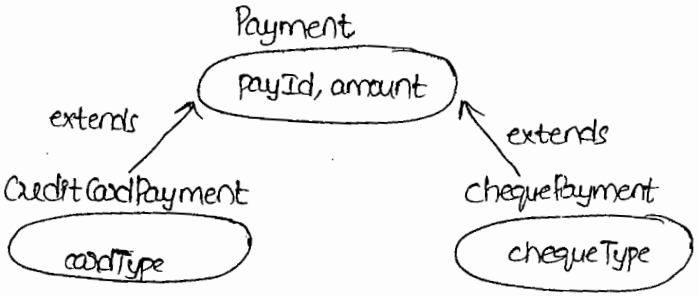
@PrimaryKeyJoinColumn → To specify FK column in Table subclass

NOTE:

1) We need @Inheritance annotation on the top of root class to specify the inheritance mapping strategy.

2) If @Inheritance annotation is not specified by default the table per class hierarchy model will be applied.

Table per class Hierarchy:



Payment.java

```
@Entity
@Table(name="payment")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="pay_type")
public class Payment
{
  private int payId, amount;
  //write setter & getter methods
  ---
}
```

CreditCardType.java

```
@DiscriminatorValue("CREDIT")
public class CreditCardPayment extends Payment
{
  private String cardType;
  //write setter & getter methods
  ---
  ---
}
```

ChequePayment.java

```
@DiscriminatorValue("CHEQUE")
public class ChequePayment extends Payment
{
  private String chequeType;
  //write setter & getter methods
  ---
}
```

For example Appln refer Appln:31 of the pageno:89

HBProj34(Anno-InheritanceMapping1-TPCH)

```
|→ src
    |→ com.nt.cfgs
        |→ hibernate.cfg.xml
    |→ com.nt.model
        |→ Payment.java,
        |→ CreditCardPayment.java
        |→ ChequePayment.java
    |→ com.nt.utility
        |→ HBUtil.java
    |→ com.nt.test
        |→ InsertTest.java
        |→ SelectTest.java
```

Annotations Based Table per Subclass:

@Inheritance (strategy = InheritanceType.JOINED)
@PrimaryKeyJoinColumn (To specify FK column)

Payment2 (parent table)

PayId(PK)	amount
101	9000
102	8000

Credit_payment2(childTable1)

cctype	payment_id(fk)
VISA	101

cheque_payment2(childTable2)

chtype	payment_id(fk)
ORDER	102

Payment.java

```
@Table (name="payment2")
@Entity
@Inheritance (strategy = InheritanceType.JOINED)
public class Payment
{
  ---
  --- (superclass)
}
```

CreditCardPayment.java

```
@Entity
@Table( name="Credit_payment2")
@ PrimaryKeyJoinColumn( name="payment_id")
public class CreditCardPayment extends Payment
{
  ---
  --- // (subclass1)
}
```

ChequePayment.java

```
@Entity
@ Table (name="cheque_payment2")
@PrimaryKeyJoinColumn (name="payment_id")
public class ChequePayment extends Payment
{
  ---
  --- (subclass2)
}
```

HBProj85(Anno-InheritanceMapping2-TPSC)

- src
  - com.nt.cfgs
    - hibernate.cfg.xml
  - com.nt.model
    - Payment.java
    - CreditCardPayment.java
    - ChequePayment.java
  - com.nt.utility
    - HBUtil.java
  - com.nt.test
    - InsertTest.java
    - SelectTest.java

For the above based example ref appln:33 of page no:95

Table per Concrete Class (Annotations based Inheritance Mapping)

1) Every class of inheritance hierarchy will have its own DB table but those DBtables will not be there in Association.

Payment3 (table1)		Credit_Payment3 (table2)			Cheque_Payment3 (table3)		
payId(PK)	amount	payId(PK)	amount	ccType	payId(PK)	amount	chType
		2	9000	ORDER	1	8000	VISA

⇒ Duplicate columns in DBTable so it is not recommended to use.

Payment.java

```
@Entity
@Table(name="strategy")
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Payment
{
  ---
  ---
}
```

CreditCardPayment.java

```
@Entity
@Table(name="credit_payments")
public class CreditCardPayment extends Payment
{
  ---
  ---
}
```

ChequePayment.java

```
@Entity
@Table(name="cheque_Payments")
public class ChequePayment extends Payment
{
  ---
  ---
}
```

Q) What is implicit polymorphism in Hibernate (HB)?

Generally in Table per concrete class model of inheritance mapping we use <cfg> tag to cfg super class & we use <union-subclass> tags to cfg subclasses. But there is possibility to cfg same classes of inheritance mapping by multiple independent <class> tag as shown below.

```
<class name="Payments" table="Payments">
  ---
  --- //super class properties cfg
</class>
<class name="Credit_Payment" table="Credit-Payments">
  ---
  --- //super class and subclass properties tag cfg
</class>
<class name="cheque_Payment" table="cheque_payements">
  ---
  --- // super class & subclass properties cfg
</class>
```

Though classes of inheritance are configured (cfg) as ordinary classes. Hibernate recognize their inheritance based on inheritance that is there physically b/w the classes. Due to this we can use them classes of inheritance even though their inheritance is not highlighted in the mapping file (or) though Annotations. This is called implicit polymorphism.

NOTE: We can not deal with implicit polymorphism while working with annotations based inheritance mapping. Because there JPA annotation & if no @Inheritance is placed by default TABLE_PER_CLASS hierarchy model of inheritance mapping takes place.

Working with large Objects

Files are called large objects. There are 2 types of large objects

a) BLOB (b) CLOB

BLOB : Binary Large Objects

Ex: audio files, video files, avi files, Images etc

CLOB : Character Large Objects

Ex : text files, rich text files, doc files, & etc.

All most all major db slws are giving support for large objects. While developing job portal, matrimony, profile management applications we need to work with large objects.

Hibernate 3.x supports LOBs insertion & retrieving but we need to take support of jdbc & streams of that.

For inserting BLOB values:

BLOB file (Image file) → byte[] → java.sql.Blob object → set to domain class object → ses.save(-) → db slw.

For inserting CLOB values:

(text file) CLOB file → char[] → java.sql.clob object → set to domain class object → ses.save(-) → DB slw.

For Retrieving BLOB values:

DB slw → ses.load(-,-) → Domain class object → java.sql.Blob object → byte[] → file

For Retrieving CLOB values:

DB slw → ses.load(-,-) → Domain class object → char[] → file.

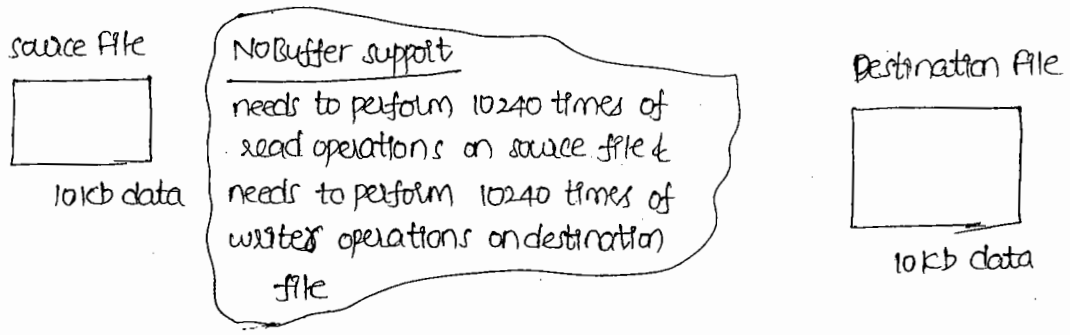
HB Proj35 (Anno-InheritanceMapping2-TPSC)

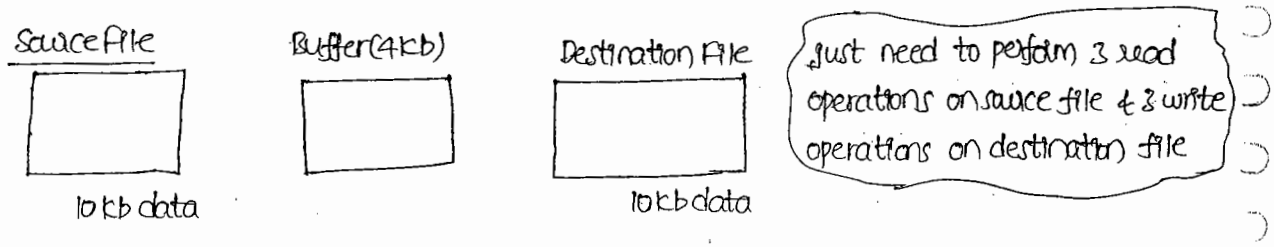
```
↦ src
    ↦ com.nt.cfgs
        ↦ hibernate.cfg.xml
        ↦ Employee.hbm.xml
    ↦ com.nt.model
        ↦ EmpDetails.java
    ↦ com.nt.utility
        ↦ HBUtil.java
    ↦ com.nt.test
        ↦ InsertClient.java, RetrieveClient.java
```

Appln 35 of page no: 97 for the above application.

⇒ Buffer is a temporary memory that holds data for temporary period. While transferring huge amount of data from source file to destination file. If we take the support of buffer we can reduce the no of read operations on source file & no. of write operations on destination file.

⇒ Generally buffer is collection object or byte[] or char[].





In Hibernate 3.x version

⇒ Hibernate.createClob(-), Hibernate.createBlob(-) methods are deprecated & the alternate methods are LobHelper.createBlob(-) and LobHelper.createClob(-) methods.

Ex:

```
LobHelper helper = ses.getLobHelper();
java.sql.Blob photo = helper.createBlob(byte[]);
java.sql.Clob resume = helper.createClob(String);
```

Association Mapping (Relationships in Hibernate)

⇒ Storing the data of multiple entities (items) in single DB tables gives lot of problems. They are

a) Data redundancy problem (Data duplication)

b) Complexity in managing huge amount of data

Example

storing faculties, student details in single db tables gives problems string users, phone numbers details in single db tables gives problems.

User_phone_info (db tables)

userId	firstName	address	phone	numbertype	provider
101	raja	hyd	9999	personal	airtel
101	raja	hyd	8888	office	Vodafone
102	ravi	Mumbai	7777	personal	airtel
102	ravi	Mumbai	6666	residence	airtel

Data redundancy problem

Solution 1

Maintain user details & phone number details seperately

User_Table

userId (pk)	firstName	address
101	raja	hyd
102	ravi	Mumbai

Phone_numbers

Phone	number_type	provider
9999	office	airtel
8888	personal	Vodafone
7777	personal	airtel
6666	personal residence	airtel

Limitations

Though 2 db tables are taken, since these are not placed in relationships. So we cannot access user details from phone numbers & phone numbers from users.

Solution 2:

Keep the db tables in Relationships

User_table (parent table)

userId (pk)	first_name	address
101	----	----
102	----	-----

Phone_numbers (child table)

Phone (pk)	number-type	provide	unid (fk with userid)
9999	----	----	101
8888	----	-----	101
7777	-----	-----	102
6666	----	-----	102

⇒ The solution2 solves the basic problem and the limitations of solution 1.

⇒ In hibernate programming we need to take domain classes participating in relationships when there db tables in relationships due to this we get navigate from one object to another object of domain classes.

⇒ The domain classes of relationships (association) must be cfg in xml file to express their relationships by using association mapping. Hibernate supports 4 models of association mapping.

a) one to one (ex: Student → Bank)
b) one to many (ex: user → phonenumbers)
c) Many to one (ex: phone numbers → user)
d) Many to many (ex: programmer → projects)

⇒ These relationships can be unidirectional (or) bi-directional.

⇒ In uni-directional parent object can access child object or child object can access parent object. In bi-directional parent object can access child object & child objects can access parent obj.

parent ⇒ child or child ⇒ parent (uni-directional)

parent ⇒ child (bi-directional)

One to many uni directional Association : (parent to child)

1) Here each parent object hold set of child objects (1 or more child objects)

2) For this parent class should have collection type property to hold one or more child objects.

3) While working with one to many and many to many associations the Domain classes can have collection type properties.

4) To cfg these collection type properties specifying the association we can use <set>|<list>|<map>|<bag> and etc.. tags (This is called collection mapping)

5) The association b/w users & phone numbers is one to many. Because one user can have multiple phone numbers.

Ex: one to many uni-directional

DB Tables:

User_table (parent)

User_id (PK)	first_name (VC2)
101	raja
102	ravi

Phone_numbers

Phone (PK)	number_type	uid (FK)
9999	office	101
8888	residence	101
7777	office	102
6666	personal	102

User.java

```
public class User
{
  private int userId;
  private String firstName;
  //write setter & getter methods
  ----
}
```

PhoneNumber.java

```
public class PhoneNumber
{
  private long phone;
  private String numberType;
  //write setter & getter methods
  ----
}
```

User.hbm.xml

```
<class name="pkg.User" table="user_table">
<id name="userId" column="user_id"/>
<propelty name="firstName" column="first_Name"/>
<set name="phones" cascade="all">
<key column="unId"/>
<one-to-many class="pkg.PhoneNumber"/>
</set>
</class>
```

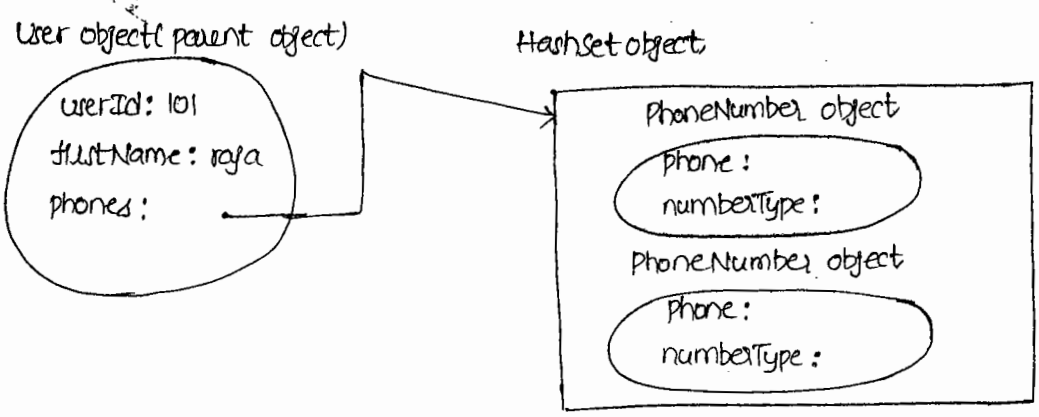
PhoneNumber.hbm.xml

```
<class name="pkg.PhoneNumber" table="phone_numbers">
<id name="phone"/>
<propelty name="numberType" column="number_Type"/>
</class>
```

To save objects (in clientAppln)

```
    //parents objects
  User user1 = new User();
      user1.setUserId(101);
      user1.setFirstName("raja");
      //child objects
  PhoneNumber  phone1 = new PhoneNumber();
      phone1.setPhone(8888);
      phone1.setNumberType("personal");
  PhoneNumber  phone2 = new PhoneNumber();
      phone2.setPhone(9999);
      phone2.setNumberType("residence");
    //set child objects to parents objects
  Set<PhoneNumber> set= new HashSet<PhoneNumber>();
      set.add(phone1);
      set.add(phone2);
      user1.setPhone(set);
    Transaction tx=null;
      try{
          tx=ses.beginTransaction();
          ses.save(user1);
          tx.commit();
      }
```

```
catch(Exception e)
{ tx.rollback();
}
```



HB Progs(OToM-Uni)

```
|→src
   |→com.nt.cfgs
        |→hibernate.cfg.xml
        |→User.hbm.xml
        |→PhoneNumber.hbm.xml
   |→com.nt.domain
        |→User.java,
        |→PhoneNumber.java
   |→com.nt.utility
        |→ HBUtil.java
   |→com.nt.test
        |→ InsertTest.java
        |→ SelectTest.java
        |→ DeleteTest.java
```


For the above example appln refer appln 36 page no:101

⇒ In association mapping, the class whose objects can hold the objects of other class is called Parent class and the class whose object can be associated with other class objects is called child class.

⇒ <one-to-many>, <many-to-many>, <one-to-many> and <many-to-many> tags are given for xml based association mapping.

Insert operation in one to many association (uni-directional)

⇒ For this we need to create parent object set of child objects in a collection and add that collection to parent objects. If we take cascade="all" or cascade="persist" or cascade="save" or cascade="save-update" then the save operation done on parent object will also be cascaded to associated child objects.

⇒ Irrespective of the value we have taken for cascade attribute all the select operations will be cascaded.

```
<set name="phones" cascade="all">
---
---
</set>
```


Select operation in one to many Association (uni-directional)

⇒ Here hibernate does not load associated child objects along with parent objects ie, it loads parent objects normally but the associated child objects will be loaded on demand basis (Lazy loading).

⇒ In association mapping lazy loading means the main/parent objects will be loaded normally but associated child objects will be loaded lately/lazily on demand basis.

⇒ Here we can enable or disable lazy loading of associated child objects using "lazy" attribute. In <one-to-many> tag the lazy attribute allows two values "true/false".

Lazy="true" → Performs lazy loading (default)
Lazy="false" → Performs eager/early loading

Ex:
```
<set name="phones" lazy="true/false">
---
---
</set>
```

NOTE

⇒ The attribute cascade does not show any cascading related to select operation. It always performs cascading of non-select persistence operations.

Q) How can we achieve lazy loading in Hibernate?

a) By using ses.load(--)
b) By using list() of HQL, criteriaAPI (QBC) logics
c) In Association Mapping

a) By using lazy="true" of <set>/<list>/<map>/<bag> and etc.
b) By using lazy="proxy/no-proxy" of <one-to-one>, <many-to-one> tags.

Delete operations in one-to-many Association (uni-directional)

⇒ Here when delete parent object the associated child objects will also be deleted automatically (cascade="all", cascade="delete")

Q What is orphan record?

1) Deleting one child from collection of childs belonging to a parent

```
// load parent object
User user = (User) ses.get(User.class, 101);
// get all childs to a parent object
Set<PhoneNumber> childs = user.getPhones();
// load 1 child object that u want to delete
PhoneNumber ph = (PhoneNumber) ses.get(PhoneNumber.class, (long) 99999);
// delete 1 child object from collection
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    childs.remove(ph);
    tx.commit();
  }
catch(Exception e){
    tx.rollback();
  }
```


⇒ In the above code is executed to delete 1 child from collection of childs belongings to a parent it does not delete that child record completely but removes the relation (or) link with parent record by emptying fk column for that child record. Such record is called orphan record. This happens when cascade = all (or) delete.

User_table

user_id	first_name
101	raja

Phone_Numbers (child table)

phone	number_type	child (fk)
99999	office	— (orphan record)
88888	residence	101

NOTE

⇒ To instruct HB (Hibernate) to delete orphan records of above situation we can use cascade="delete-orphan" (or) cascade="all-delete-orphan" → recommanded.

Sample Code

```
<set name="phones" cascade="all-delete-olphan" lazy="true">
- - -
- - -
</set>
```

Note cascade attribute is purely related to non-select operations.

⇒ Lazy attribute is purely related to select operations.

Q What is cascading in hibernate?

⇒ The process of navigating/propagating non-select persistent operations performed on the main/parent objects to the associated child objects is called cascading. The possible values are

cascade="none" (default) → any persistence operations on main object does not cascade to associated objects.

cascade="persist" → cascade insert operations to associated objects

cascade="saveUpdate" → cascades insert and update operations to associated objects

cascade="delete" → cascades the delete operations to associated objects

cascade="merge" → cascades the merge operations to associated objects

cascade="all" → cascade insert, update, delete operations to the associated objects

cascade="delete-olphan" → cascades the delete operations and also deletes the olphan records

cascade="all-delete-olphan" → cascades all the persistence operations and also deletes olphan records and etc...

⇒ When cascade="persist" → we must use ses.persist() method to save parent objects. when when cascade="save-update" → we must use ses.save() to save objects (or) we must use ses.update() to update objects (or) we must use ses.saveOrUpdate() to save or update objects.

Taking Collection type of java.util.List in one to many Association:

1) In associating we can take collection property as java.util.List/Set/Map type property
2) We can use <set>/<list>/<map>/<bag> tags to cfg those properties
3) The collection type set does not allow duplicates and does not maintain the indexing of elements.
4) The collection type list allows duplicates and maintains the elements with indexing
5) In one to many association if we take collection type as java.util.List each parent can maintain its childs having indexes but same thing not possible if we take collection type as the "set".
6) To store the index of child objects for every parent object we need one extra index column in child table and use <list-index> tag to cfg that index column.

⇒ Converting previous one to many (User_PhoneNumbers) (Uni-directional) example app by taking collection type as java.util.List.

1) Create tables

User_table1 (parent table)

user_id	first_name
101	raja

Phone_numbers1 (child table)

Phone	number_type	unid(fk)	last_index
9999	office	101	0
8888	residence	102	1

2) Change collection type of java.util.List in User.java

```
private List<PhoneNumber> phones;
//reflect the change in setter and getter methods
```

3) Place <list> tag instead of <set> in user.hbm.xml

```
<list name="phones" cascade="all-delete-orphan">
<key column="unid" />
<list_index column="1st_index" /> <!-- mandatoly -->
<one-to-many class="com.nt.domain.PhoneNumber" />
</list>
```

4) write the following code in client App.

```
User user1 = new User();
  user1.setUserId(101);
  user1.setFirstName("raja");
PhoneNumber ph1 = new PhoneNumber();
- - -
- - -
PhoneNumber ph2 = new PhoneNumber();
- - -
- - -
List<PhoneNumber> childs = new ArrayList<PhoneNumber>();
    childs.add(ph1);
    childs.add(ph2);
  //add childs to parent
  user1.setPhones(childs);
  Transaction tx=null;
  try{
      tx=ses.beginTransaction();
      ses.save(user1);
      tx.commit();
  }
```

```
catch (Exception e){
        tx.rollback();
      }
```

For above steps based Ex.appln refer Appln 86 of page no: 105

⇒ While working with java.util.Set type collection if we want to delete one child from collection of childs belongings to a parent we must load that child object. But while working with java.util.List type collection there is no need of loading child objects for the same operation refer below code. Because we can delete List collection for the same element through index.

```
//load parent object
User user=(User)ses.get(User.class, 101);
//get all child of parent objects
List<PhoneNumber> childs = user.getPhones();
//delete 1 child object from collection
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    childs.remove(0);
    tx.commit();
  }
catch (Exception e)
  { tx.rollback();
  }
```

Deleting the childs of a parent in one-to-many association

```
User user=(User)ses.get(User.class, 102);
List<PhoneNumber> childs = user.getPhones();
Transaction tx=null;
  try{
      tx=ses.beginTransaction();
      childs.removeAll(childs);
    tx.commit();
    }
  catch (Exception e)
  { tx.rollback();
  }
```

Working with <bag> tag:

⇒ When we cfg java.util.List collection by using <list> tag index column in child table is mandatory. If we are interested in that index column then cfg same java.util.List collection by using <bag> tag.

Ex

(a) create same table with out index column in child table
(b) Replace <list> tag with <bag> tag in user.hbm.xml file

```
<bag name="phones" cascade="all-delete-orphan">
<key-column="unid"/>
<one-to-many class="com.nt.domain.PhoneNumber"/>
</bag>
```

User_table3(same)

phone_numbers2(same but no lst_index column)

⇒ W.R.T Appln36 of material adding new child to the existing childs of a parent

```
//get parent object
User user=(User) ses.get(User.class,101);
//get child objects of parent
List<PhoneNumber> phones = ses.getPhones();
//new child object
PhoneNumber ph3 = new PhoneNumber();
    ph3.setPhone(8888);
   ph3.setNumberType("personal");
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    phones.add(ph3);
   tx.commit();
  }
catch(Exception e)
  {
   tx.rollback();
  }
```


NOTE

⇒ In HB any persistence operations performed on the persistent state collection in a transaction will be reflected to DB Table automatically as shown above.

Taking Collection type as java.util.Map in one to many association mapping:

1) If the collection type is java.util.Set. The child objects of a parent object contains no index.
2) If the collection type is java.util.List. The child objects of a parent objects holds numeric values as the indexes.
3) If the collection type is java.util.Map. The child objects of parent objects holds keys of map collection as the indexes (ie, user defined indexes can be maintained).
4) To cfg such index column use <map-index> tag and also use <map> tag to cfg the collection property.

Ex

DB-tables

user_table (parent table)

User_id (PK)	first_name (vc2)
101	raja

phone_numbers (child table)

Phone	number_type	unid (fk)	map_index (vc2)
999999	office	101	phone1
8888888	personal	101	phone2

Step 1

change collection type java.util.Map from java.util.List/Set in User.java

```
private Map<String, PhoneNumber> phones;
  //write setter and getter methods
```

Step 2

Replace <set>/<list>/<bag> with <map> tag as shown below in user.hbm.xml file

```
<map name="phones" cascade="all">
<key column="unid" />
<map-key column="map_index" type="string" />
<one-to-many class="com.nt.domain.PhoneNumber" />
</map>
```

Step 3

Place the following code in ClientAppln to save the objects

```
// create parent object
User user1 = new User();
  user1.setUserId(101);
  user1.setFirstName("rajesh");
```

```
//create child objects
PhoneNumber ph1 = new PhoneNumber();
    ph1.setNumberType ("office");
    ph1.setPhone (99999);
PhoneNumber ph2 = new PhoneNumber();
    ph2.setNumberType ("residence");
    ph2.setPhone (8888);

//add child objects to parent object
Map<String, PhoneNumber> childs = new HashMap<String, PhoneNumber> ();
    childs.put ("phone1", ph1);
    childs.put ("phone2", ph2);
    user1.setPhones (childs);
//save the object
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.save (user1);
    tx.commit();
}
catch(Exception e){
    tx.rollback();
}
```

Q What is the difference b/w List Collection (or) Set Collection (or) Map Collection in Association Mapping?

Set	List	Map
Does not allow duplicates	Allows	Allows duplicates values, but not keys
Does not maintain the index of elements	maintains the index of elements	keys are the indexes
use <set> tag for configuration	use <list>/<bag> tag	use <map> tag
Index column cfg is not required	Is required for <list> tag & not required for <bag> tag	Is required
we can not delete one child from group of childs without loading that child object	We can delete with out loading child object	We can delete without loading child objects

Many to one Uni-directional Association (child to parent)

1) Generally one-to-many association is parent to child Association and many to one association is child to parent Association.

2) Here child domain class contains parent class reference variable pointing to parent class object like this multiple child objects will point to single parent class object.

3) To cfg special property use <many-to-one> with cascade, lazy attributes

4) The Association between EmpDetails (Emp) and Department (Dept) is many to one from EmpDetails point of view (child) and the same association is one to many from department point of view (parent).

   1) Department can contain multiple Employees (1 to Many)
      Multiple Employees can belong to one department (many to 1).

In <one-to-one>, <many-to-one> Association no need of collection type property.

SQL > select * from empdetails;

ENO	ENAME	SALARY	DEPTNO
1001	raja	9000	101
1002	ravi	8000	101

SQL> select * from department;

DEPTNO	DEPTNAME	DEPTHEAD
101	Accounts	SMITH

HBProg42(MTO Uni-Directional Association)

```
|→ src
    |→ com.nt.cfgs
          |→ hibernate.cfg.xml
          |→ Department.hbm.xml
          |→ EmpDetails.hbm.xml
    |→ com.nt.domain
          |→ EmpDetails.java, Department.java
    |→ com.nt.dao
          |→ M2ODao.java, M2ODaoFactory.java, M2ODaoImpl.java
    |→ com.nt.utility
          |→ HBUtil.java
    |→ com.nt.test
          |→ ClientApp.java
```

Domain classes

Department.java(parent class)

```
package com.nt.domain;
public class Department
 {
   private int deptno;
   private String deptname;
   private String depthead;

     public Department()
      {  s.o.p("0-param constructor");
      }
   public Department(int deptno, String deptname, String depthead){
         this.deptno=deptno;
         this.deptname=deptname;
         this.depthead=depthead;
        }
   //write setter and getter methods

   @override
  public String toString(){
   return "Department [deptno="+deptno+", deptname="+deptname+",
                                        depthead="+depthead+"]";
        }
 }
```

EmpDetails.java(child class)

```
package com.nt.domain;
public class EmpDetails
  {  private int eno;
     private String ename;
     private long salary;
     private Department dept;
  //write setter and getter methods
    @override
    public String toString(){
  } } return "EmpDetails [eno="+eno+", ename="+ename+", salary="+salary+"]";
```

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
  <property name = "connection.driver_class">oracle.jdbc.OracleDriver</property>
  <property name= "connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
  <property name = "connection.username">system</property>
  <property name= "connection.password">system</property>
  <property name= "show_sql">true</property>
  <mapping resource ="com/nt/cfgs/EmpDetails.hbm.xml"/>
  <mapping resource ="com/nt/cfgs/Department.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Department.hbm.xml

```
<hibernate-mapping>
  <class name ="com.nt.domain.Department" table="Department">
   <id name="deptno" /> <!--Singular Id field cfg-->
   <property name="deptname" />
  <property name="dept head" />
  </class>
</hibernate-mapping>
```

EmpDetails.hbm.xml

```
<hibernate-mapping>
 <class name="com.nt.domain.EmpDetails" table="EmpDetails">
  <id name="eno" />
  <property name="ename" />
  <property name="salary" />
 <many-to-one name="dept" class="com.nt.domain.Department" column="deptno" cascade="none" />
 </class>
</hibernate-mapping>
```

com.nt.dao

M2ODao.java

```
public interface M2ODao{

    public void createEmployeesWithDept();
    public void addNewEmployeeInDept();
    public void listEmployeesWithDept();
    public void deleteEmployeesWithDept();
    public void addExistingEmployeeToNewDept();
    public void deleteEmployeeWithoutDeletingDept();

  }
```

M2ODaoFactory.java

```
package com.nt.dao;
public class M2ODaoFactory{
    public static M2ODao getDAOInstance()
    { return new M2ODaoImpl();
    }
}
```

M2ODaoImpl.java

```
package com.nt.dao;
public class M2ODaoImpl implements M2ODao{
      @Override
    public void createEmployeesWithDept(){
        //Get Session object
     Session ses=HibernateUtil.getSession();
       //create parent object
    Department dept=new Department(101,"Accounts","SMITH");
       //create employee objects(child objects)
    EmpDetails emp1=new EmpDetails(1001,"raja",9000);
    EmpDetails emp2=new EmpDetails(1002,"ravi",8000);
        //Build Many to One relation
          emp1.setDept(dept);
          emp2.setDept(dept);
```

```
        //save object
    Transaction tx=null;
    try{
          tx=ses.beginTransaction();
           ses.save(emp1);
           ses.save(emp2);
           tx.commit();
        }
catch(Exception e){
          tx.rollback();
        }
  }
   @override
   public void addNewEmployeeInDept(){
         //Get Session object
      Session ses = HibernateUtil.getSession();
         //load existing department object
    Department dept=(Department)ses.get(Department.class,101);
       // create new Employee
      EmpDetails emp= new EmpDetails (1003,"Ramesh",7000);
          emp.setDept(dept);
       Transaction tx=null;
       try{
            tx=ses.beginTransaction();
             ses.save(emp);
            tx.commit();
         }
       catch(Exception e)
        {
           tx.rollback();
        }
  }
```

```
@Override
public void listEmployeesWithDept(){
    //Get Session object
    Session ses = HibernateUtil.getSession();
    //get childs with parents
    Query qry = ses.createQuery("from EmpDetails");
    List<EmpDetails> list = query.list();
    for(EmpDetails ed : list)
    {
        S.O.P(" child --->"+ed);
        //get department of each employee
        Department dept = ed.getDept();
        S.O.P("parent"--> "+dept);
    }
}
@Override
public void deleteEmployeesWithDept(){
    //Get Session object
    Session ses = HibernateUtil.getSession();
    //load one child object
    EmpDetails ed =(EmpDetails)ses.get(EmpDetails.class,1003);
    Transaction tx=null;
    try{
        tx=ses.beginTransaction();
        ses.delete(ed);
        tx.commit();
    }
    catch(Exception e)
    {
        tx.rollback();
    }
}
```

```
    @Override
    public void addExistingEmployeeToNewDept()
    {
        //get Session
        Session ses = HibernateUtil.getSession();
        //create new Department
        Department dept = new Department(102, "IT", "ALLEN");
        //get existing employee
        EmpDetails ed = (EmpDetails) ses.get(EmpDetails.class,1002);
            ed.setDept(dept);
        Transaction tx=null;
            try{
                tx=ses.beginTransaction();
                ses.update(ed);
                tx.commit();
               }
           catch(Exception e)
            {
              tx.rollback();
            }
    }
  @Override
public void deleteEmployeeWithOutDeletingDept()
  {
      //get session object
     Session ses= HibernateUtil.getSession();
     //To delete only child object
    Query q1 = ses.createQuery("delete from EmpDetails where eno=:id");
         q1.setInteger("id", 1002);
      Transaction tx=null;
        try{ tx=ses.beginTransaction();
           int result = q1.executeUpdate();
          s.o.p("result +"no. of records are deleted");
           tx.commit();
         }
        catch(Exception e)
         { tx.rollback();
         }
```

```
/*  //get child object (Execute this code with cascade="none" to delete only child)
    EmpDetails ed1=(EmpDetails) ses.get(EmpDetails.class, 1002);
     Transaction tx1=null;
     try{
          tx1 = ses.beginTransaction();
            ses.delete(ed1);
          tx.commit();
       }
      catch(Exception e)
        {
          tx.rollback();
        }
    */
  }
}
```

com.nt.utility

HibernateUtil.java

```
package com.nt.utility;
public class HibernateUtil
 {
   private static SessionFactory factory=null;
    static{
          try{
           Configuration cfg=new Configuration();
           //Read mapping cfg details
          cfg = cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");
          //create Service Registry Builder
      StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder();
          //use builder to create Service Registry
      ServiceRegistry registry= builder.applySettings(cfg.getProperties()).build();
          //create Session Factory obj
        factory = cfg.buildSessionFactory(registry);
           S.o.p("session factory complete successfully");
        }
       catch(Exception e)
        {  S.o.p("enter into factory exception");
        } //static
```

```
static ThreadLocal<Session> t1 = new ThreadLocal<Session>();
    static Session ses=null;
  public static Session getSession()
      {
        try{
            if(t1.get()==null)
             {
               //get from ThreadLocal
             ses=factoty.openSession();
              s.o.p("enter into session");
             t1.set(ses) ; //set into ThreadLocal
            return ses;
             }
             else{
                 s.o.p("get into session");
              return t1.get(); //get from ThreadLocal
              }
           }//try
        catch(Exception e)
         {
           s.o.p("session exception");
         return null;
         }
     }//method
 public static void closeSession()
  {  try { ses.close();
        }
     catch(Exception e)
       {
         e.printStackTrace();
       }
    }//closeSession();
}// class
```

com.nt.test

ClientApp.java

```
package com.nt.test;
import com.nt.dao.H2ODao;
import com.nt.dao.H2ODaoFactory;
public class ClientApp{
    public static void main(String[] args)
    {
        //get DAO object
    H2ODao dao = H2ODaoFactory.getDAOInstance();
        //call methods
        //dao.createEmployeesWithDept();
        //dao.addNewEmployeeInDept();
        //dao.addExistingEmployeeToNewDept();
        //dao.listEmployeesWithDept();
        dao.deleteEmployeesWithDept();
        //dao.deleteEmployeeWithOutDeletingDept();
    }
}
```

⇒In <many-to-one> association if we save child objects the HB FW first checks weather the associated in parent table is inserted or not. If not inserted first it inserts the record in parent table and associated records in child table. If parent table record is already available it inserts record in child table & links them with parent table record through foreign key columns.

⇒ In <one-to-many> relationship (uni-directional parent to child). If lazy loading is enabled parent object will be loaded but the associated child objects will be loaded directly.

⇒In <many-to-one> association (uni-directional child to parent). If lazy loading is enabled child objects will be loaded normally & the associated parent object will be loaded lazily.

⇒Here we can use "lazy" attribute of <many-to-one> tag to specify weather the associated parent objects should be loaded with child objects or not. Here lazy="true" is not supported, but lazy="proxy|no-proxy|false" values are supported.

⇒lazy="false" → performs early loading/fetching ie, the associated parent object will be loaded along with child objects.

⇒lazy="no-proxy" → performs lazy loading with out generating proxy objects. For this we need "Built time byte code instrumentation" support.

⇒ Lazy = "proxy" → perfoms lazy loading by generating proxy object ie, when child objects are loaded the associated parent objects proxy object will be created. When we start accessing parent through child object then real parent object will be created on demand basis.

Delete operations in <many-to-one> association:

1) Here if we try to delete child objects (having cascade = "all" "delete") then the associated parent object will also be attempted for deletion, but if that parent object is having other child objects then "constraint violation Exception" will be raised, otherwise both child object & the associated parent object will be deleted.

2) If we attempt to delete all child objects then the associated parent object will be deleted automatically.

3) If we want to delete one child object from multiple child objects belonging to a parent object, then we can use cascade = "none" to load & delete that child object.

```
EmpDetails emp = (EmpDetails) ses.get (EmpDetails.class, 1001);
   //deleted child objects
Transaction tx = null;
try {
     tx = ses.beginTransaction();
      ses.delete(emp);
      tx.commit();
   }
catch (Exception e)
  {  tx.rollback();
  }
```

⇒ In <many-to-one> association there is no possibilities of getting orphan records. so cascade = "delete-orphan" / "all-delete-orphan" cannot be used.

⇒ In the above code changing to cascade = "none" in the middle of application development practically impossible. so use "HQLDelete" query as shown below to solve the problem.

```
//Get HB session
Session ses = HibernateUtil.getSession();
//Use HQL Delete query
Query query = ses.createQuery ("delete from EmpDetails where eid = :id");
    query.setInteger("id", 1002);
Transaction tx = null;
```

```
try{
    tx=ses.beginTransaction();
    int result=query.executeUpdate();
    S.O.P("no. of records deleted "+result);
    tx.commit();
  }
catch(Exception e)
  {
   tx.rollback();
  }
```

⇒ Generally we call ses.delete(-) method by loading child object. In this process the associated parent object will also be loaded & deleted. When we use HQL query it won't load child objects moreover it directly deletes child objects.

<one-to-many> Bi-directional / <Many-to-one> Bi-directional Association :

1) It is the combination of <one-to-many> unidirectional & <many-to-one> unidirectional associations.

2) For this parent class should have collection type property to hold/more child objects & child class should have parent class reference to make multiple child objects holding single parent object.

3) In Bi-directional association we access child objects from parent objects & vice-versa. i.e., parent to child and child to parent navigation is possible

4) Though it is Bi-directional association we can create tables for this association having single FK column in child table. i.e, there is no need of having 2 fk columns in 2 tables (parent table, child table). Because we can access the records child to parent & parent to child by using single FK column.

Example

User.java(parent class)

```
public class User
{
  private int userId;
  private String firstName;
  private Set<PhoneNumber> phones;
  //write setter and getter methods
  ---
  ---
}
```

PhoneNumber.java

```
public class PhoneNumber
{
  private long phone;
  private String numberType;
  private User user;
  //write setter and getter methods;
  ---
}
```

User.hbm.xml

```
<hibernate-mapping>
<class name="pkg.User" table="user_table">
<id name="userId" column="user_id"/>
<property name="firstName" column="first_Name"/>
<set name="phones" cascade="all-delete-orphan" lazy="true">
<key column="unId"/>
<one-to-many class="pkg.PhoneNumber"/>
<lset>
<lclass>
<lhibernate-mapping>
```

PhoneNumber.hbm.xml

```
<hibernate-mapping>
<class name="pkg.PhoneNumber" table="phone_numbers">
<id name="phone"/>
<property name="numberType" column="number_Type"/>
<many-to-one class="pkg.User" column="unId" cascade="all"/>
<lclass>
<lhibernate-mapping>
```

DB tables

User_table (parent)

- user_id (pk)
- first_name (vc2)

Phone_numbers (child)

- phone (pk)
- number_Type (vc2)
- unId (fk)

HBProj43 (O TO M Bi-Directional Association)

```
|-> src
    |-> com.nt.cfgs
        |-> hibernate.cfg.xml
        |-> user.hbm.xml
        |-> phoneNumber.hbm.xml
    |-> com.nt.domain
        |-> User.java
        |-> PhoneNumber.java
    |-> com.nt.dao
        |-> O2MDao.java,
        |-> O2MDaofactoy.java
        |-> O2MDaoImpl.java
    |-> com.nt.utility
        |-> HBUtil.java
    |-> com.nt.test
        |-> ClientApp.java
```

Q) What is inverse association in HB?

A) While working with Bi-directional association we can keep inverse="true" to tell to HB the other side association is mirror of current side. This helps HB to generate SQL queries appropriately by keeping Bi-directional association in mind and avoids the generation of unnecessary additional queries to update the fk column values that means certain queries generation is reduced to improve the performance.

⇒ In <one-to-many> Bi-directional association enable this at any side (in <set>/<list>/<map> tags)

⇒ In <many-to-many> association we can enable this either side.

Ex

In oneto Many Bi-directional association Example

```
<set name="phones" cascade="all-delete-orphan" inverse="true">
---
---
</set>
```

inverse="true" indicates this association is Bi-directional and avoids extra update query generation to update fk column values.

HQL Joins

⇒ We use SQL joins to get records of more than one db table by using single select SQL query with implicit conditions

⇒ To work with SQL joins DB tables need not be there in relationship.

⇒ To get more than 1 domain class objects (more than 1 db table records) by using single HQL select query we can use HQL joins. These joins internally use some implicit conditions to fetch the data. To work with HQL joins our domain classes must be there in relationship like one to one or one to many or many to one or many to many.

HB supports 4 types of joins

a) Inner join (default)
b) Left join / left outer join
c) Right join / right outer join
d) Full join

Inner join

⇒ Gives the common data of left side table and right side table.

Left join

⇒ Gives common data of both tables and also gives the uncommon data of left side table

Right join

⇒ Gives common data of both tables also gives uncommon data of right side table

full join

⇒ Gives both common & uncommon data of both left side & right side db tables

Joinqueries

Inner join : select u.userId, p.phone from User u inner join u.phones p
Left join : select u.userId, p.phone from User u left join u.phones p
Right join : select u.userId, p.phone from User u right join u.phones p
Full join : select u.userId, p.phone from User u full join u.phones p

Q What is 1+n select problem?

⇒ In <one-to-many> association HB fw generates one select query to retrieve all parent records and n-select queries to retrieve child records of n-parent records that means it generates 1+n select queries totally.

Ex

User_table

101	raja
102	raja

Phone_numbers (child)

888888	office	101
9999	residence	101
7777	office	102
666666	residence	102

⇒ HB generates 1 select query to get all the records of parent table and generates n select queries to get child records of all (n) parent records as shown below.

```
select * from user_table ; (1 query)
101 - - -
102. - - -
select * from phone_numbers where unId = 102;
select * from phone_numbers where unId = 101 ; (2 n queries)
```

⇒ Generating more SQL queries degrades the performance of the application. So the above queries generation process looks like (1+n select problem) problem.

Q) How to solve 1+n select problem?

Solution 1:

By using "fetch" join

```
String qry = "select u from User u inner join fetch u.phones";
Query query = ses.createQuery(qry);
List<User> list = query.list();
for( User u: list)
{
   S.o.p("parent" -> " + u);
   Set<PhoneNumber> phones = u.getPhones();
   for(PhoneNumber ph: phones)
   {
      S.o.p("child" -> " + ph);
   }
}
```

⇒ fetch join disables lazy loading and enables eagly loading by generating single SQL select query to get all parent table records and the associated child table records.

Solution 2

⇒ By using QBC (Criteria API) with FetchMode API.

```
Criteria ct = ses.createCriteria(User.class);
    ct.setFetchMode("phones", FetchMode.JOIN);
List<User> list = ct.list();
for(User u: list)
{
  ---
  ---
}
```

NOTE: 1) Here also it generates single select query to get all parent records & the associated child records

⇒ We cannot specify "fetchmodes" while working with direct HQL queries of NativeSQLQuery.

Fetching strategies / modes in HB

⇒ We can use these fetching modes to specify fetch modes while retrieving the associated child objects along with the parent objects

The 3 select strategies are:

1) Select :→ generates 1 select query to get all parent records and generates n select queries to get child records of n parent records raise Hn select problem

b) Sub-select: → generates 1 select query to get all parent records and 1 select query to get child records of all parent records (1+1) queries. Does not raise Hn select problem.

c) Join: → generates 1 select query to get all parent records and the associated child records. Solves Hn select queries problem.

⇒ While working with QBC (Criteria API) we can specify the fetching strategies by using "fetch" attributes of <collection> tags (like <set>, <list>, <map>, <bag>) or we can use ct.setFetchMode(-) method.

Ex:

```
<set names="phones" cascade="all" lazy="true" fetch="join">
--
</set>
```

NOTE: ⇒ While working with ct.setFetchMode(-) method we can specify FetchMode.SELECT, FetchMode.DEFAULT, FetchMode.JOIN (There is no provision to specify "subselect" here)

⇒ FetchMode.DEFAULT will use "fetch" attribute value ie, set in xml mapping file.

⇒ If we specify 2 different strategies through xml file & QBC code (ct.setFetchMode(-)) then the fetching strategy specified in the QBC code will be applied.

Many to Many Association

1) It is one to many association from parent & 1 to many association from child. so together is called Many to Many Association.
2) For this association parent class must have collection type property to hold one or more childs and child class must have collection type property to hold one or more parents.
3) In many to many association 1 or more parents will be associated with 1 or more childs and vice-versa
4) The relationship b/w programmers and projects is many-to-many association. 1 or more programmers can work for 1 more parents and vice versa.
5) To build many to many association we need 3 db-tables (maximum) table1 holds parent records, table2 holds child records and table3 (joins table) specifies the relation b/w parent & child.

Example

Programmer.java

```
public class Programmer
{
   private int pid;
   private String pname;
   private long salary;
   private set<Project> projects;
   //3-param constructor
   //setter and getter methods
   ---
   ---
}
```

Project.java

```
public class Project
{
   private int proId;
   private String proname;
   private set<Programmer> programmers;
   //2-param constructor
   //setter and getter methods
   ---
   ---
   //toString
}
```

Tables

Programmers (table 1)

pid (pk)	pname	salary
101	raja	9000
102	ravi	8000
103	ramesh	7000

Projects (table 2)

proid (pk)	proname
1001	proj 1
1002	proj 2

Programmers_projects (table 3 - join - tables)

programmer_id (fk)	project_id (fk)
101	1001
101	1002
102	1001
103	1001
103	1002

Programmer.hbm.xml

```
<hibernate-mapping>
<class name="pkg.Programmer" table="programmers">
<id name="pid" />
<property name="pname" />
<property name="salary" />
<set name="projects" table="programmers_projects" cascade="all" lazy="true">
<key column="programmer_id" />
<many-to-many class="pkg.Project" column="project_id" />
</set>
</class>
</hibernate-mapping>
```

Project.hbm.xml

```
<hibernate-mapping>
<class name="pkg.Project" table="projects">
<id name="proid" />
<property name="proname" />
<set name="programmers" table="programmers_projects" cascade="all" lazy="true">
<key column="project_id" />
<many-to-many class="pkg.Programmer" column="programmer_id" /> </set></class>
</hibernate-mapping>
```

HBProj44(MTOM-Bi-directional Association)

- src
  - com.nt.cfgs
    - hibernate.cfg.xml
    - programmer.hbm.xml
    - project.hbm.xml
  - com.nt.domain
    - Project.java
    - Programmer.java
  - com.nt.dao
    - M2MDao.java, M2MDaoFactoty.java, M2MDaoImpl.java
  - com.nt.utility
    - HBUtil.java
  - com.nt.test
    - ClientApp.java

⇒In many to many association there is no question of getting orphan records so cascade="delete-orphan" or cascade="all-delete-orphan" not required.

⇒In many to many association there is no question of getting orphan records so cascade="delete-orphan" or cascade="all-delete-orphan" not required.

<idbag> tag :

⇒While working with many to many association there is no uniqueness in 3rd table (join table) i.e, there is no column in 3rd table that holds unique values. To overcome this problem use <idbag> tag.

⇒The above column that contains unique values can be used to access the records of 3rd table and to manipulate the records of 3rd table.

Example

1) keep many to many example appln ready
2) Add an extra column in 3rd table to hold the unique value

SQL >alter table programmers_projects add prog_prgmr_index number(5);

3) change collection type to List from set in Project.java
4) cfg <idbag> in project.hbm.xml instead of <set> tag as shown below.

Project.hbm.xml

```
<idbag name="programmers" table="programmer_projects" cascade="all">
<collection-id name="prog_prgmr_index" type="int">
<generator class="increment" />
</collection-id>
```

```
<key-column="proj_id"/>
<many-to-many class="com.nt.domain.Programmer" column="programmer_id"/>
</idbag>
```

5) Write the code in DAO class accordingly to save the objects

If we execute code with the above steps

SQL> select * from programmers_projects;

Programmers_Id	Projects_Id	Proj_prgmr_index
101	1001	1
102	1001	2
103	1001	3
102	1002	4
103	1002	5

One To One Association

⇒If one parent contains one child then it is called one to one association. It can be implemented in 2 ways.

a) One To One PK (One To One Primary key)

b) One To One FK (One To One foreign key)

Q) What should we use One To One PK and One To One FK?

One To One PK

⇒If the relationship is bidirectional. If parent object is ready to use child object identity value as its identity value (or) vice versa. Here no need of maintaining FK column in DB tables.

Ex:

Every "Student" contains "LibraryMembership" more over we can use "StudentId" as the "LibraryMembership Id", i.e., we do not need seperate Id for LibraryMembership.

One To One FK

⇒If the relationship is uni-directional. If parent object & child objects want to have seperate id values.

⇒If every child contains parent but there is no guarantee every parent should have children vice versa. Here we need FK column support to build relationship

Ex

Every "DrivingLicense" belongs to a person and Every Person need not to have "DrivingLicense", More over we do not have want to use personId as DrivingLicenseId.

One To One FK Sample Code:

Parent class

```
public class Person
{
   private int id;
   private String firstName, lastName;
   private byte age;
   //setter and getters methods
  ---
  ---
}
```

child class

```
public class License
{
   private long id;
   Private String type;
   private Date dateFrom, dateTo;
   Private Person LicenseHolder;
   //setters and getters method
  ---
  ---
}
```

Person (parent table)

Person_id	firstName	lastName	age
101	raja	rao	30
102	ravi	rao	12

License (child table)

License_id	type	valid_from	valid_to	license_holder
1001	2	30-Jun-15	29-Jun-15	101

⇒ <onetoone> tag is dedicated for building <one-to-one> (one to one FK) association where there is no provision to specify foreign key column. So this tag can't be used to build one To one FK association.

⇒ To overcome this problem we use <many-to-one> tag with unique="true", not-null="true" and there is a provision to specify FK column using column attributes. This helps to avoid duplicates in FK column (Because of unique="true", not-null="true") and builds many to one association as one To one FK Association.

Person.hbm.xml

```
<hibernate-mapping>
<class name="pkg.Person" table="person">
<id name="id" column="person_id">
<generator class="increment" />
</id>
<property name="age" />
<property name="firstName" />
<property name="lastName" />
</class>
</hibernate-mapping>
```

License.hbm.xml

```
<hibernate-mapping>
<class name="pkg.License" table="license">
<id name="id" column="license_id">
<generator class="sequence">
<param name="sequence">my_seq_1</param>
</generator>
</id>
<property name="type" />
<property name="dateFrom" column="valid_from" />
<property name="dateTo" column="valid_to" />
<many-to-one name="licenseHolder" class="pkg.Person" column="license_holder" unique="true"
not-null="true" cascade="all" lazy="proxy" />
</class>
</hibernate-mapping>
```

Sequence

```
Create sequence my_seq_1 increment by 1 start with 1001;
```

HBProj47(OTOO FK)

- src
  - com.nt.cfgs
    - hibernate.cfg.xml, person.hbm.xml, license.hbm.xml
  - com.nt.domain
    - Person.java, License.java
  - com.nt.dao
    - O2ODao.java, O2ODaoFactory.java, O2ODaoImpl.java
  - com.nt.utility
    - HBUtil.java
  - com.nt.test
    - ClientApp.java

Sample code of One To One PK:

1) Here association will be build on the id values not based on the FK column values.

2) Here parent object & associated child object must have the same identity values, ie, child object should get the identity value of parent object (or) vice-versa. For this we need to use "foreign" generator

3) Foreign generator is useful to use the identity value of associated object as the identity value of current object.

Student.java (Parent class)

```
public class Student
{
   private int id;
   private String name;
   private String address;
   private LibraryMembership libraryDetails; // to hold associated child object
   ---
   ---
   //setters and getters methods
}
```

LibraryMembership.java

```
public class LibraryMembership
{
   private int id;
   private Date joiningDate;
   private Student studentDetails; // to hold associated parent object
  //setter and getter methods
  ---
  ---
}
```

DB Tables :

Student (Parent)

Sid (pk)	name	address
101	raja	hyd
102	ravi	hyd

Lib_membership (child)

Lid (pk)	doj (date)
101	20-OCT-1987
102	31-NOV-1990

Student.hbm.xml

```
<hibernate-mapping>
<class name="pkg.Student" table="student">
<id name="id" column="sid">
<generator class="increment"/>
</id>
<property name="name"/>
<property name="address"/>
<one-to-one name="libraryDetails" class="pkg.LibraryMembership" cascade="all"/>
</class>
</hibernate-mapping>
```

Library.hbm.xml

```
<hibernate-mapping>
<class name="pkg.LibraryMembership" table="Lib-Membership">
<id name="id" column="lid">
<generator class="foreign">
<param name="property">studentDetails</param>
</generator>
</id>
<property name="joiningDate" column="doj"/>
<one-to-one name="studentDetails" class="pkg.Student" cascade="all"/>
</class>
</hibernate-mapping>
```

⇒ In <one-to-one> there is no column attribute to specify foreign key column so we can't use <one-to-one> tag for building one to one association. In one to many, many to one association we can place lazy="true|false" values.

⇒ In many to one, one to one association we can place lazy="false|proxy|no-proxy" values (lazy="true" not available)

Annotations based Association Mapping/Relationship

@OneToMany → To build association with the support of collection property

@JoinColumn → To specify FK column

Enums

FetchType.Lazy/EAGER → For enabling Lazy/EAGER loading

CascadeType. ALL/PERSIST/MERGE/REFRESH/REMOVE/DETACH (To specify the cascading Types)

MERGE → cascade the merge operations done on main object to the associated object

DETACH → If main object becomes Detached object the associated object also becomes detached object

REFRESH → If main object is reloaded from DB, the associated child object will also be reloaded from DB.

Example

User_table (parent)

user_id (PK)	first_name

Phone_numbers (child)

phone (PK)	number_type	unid (FK)

User.java

```
@Entity
@Table (name="user_table")
public class User
{
   @Id
   @Column(name="user_id")
   private int userId;
   private String firstName;
   @OneToMany (targetEntity=PhoneNumber.class, cascade=cascadeType.ALL,
                fetch=FetchType.LAZY, orphanRemoval=true)
   @JoinColumn(name="unid", referencedColumnName="user_id")
   private Set<PhoneNumber> phones;
   //setter and getter method
   ---
   ---
   ---
}
```

PhoneNumber.java

```
@Table (name = "phone_numbers")
@Entity
public class PhoneNumber
{
    @Id
    private long phone;
    @Column (name = "number_type")
    private String numberType;
    public long getPhone()
    { return phone;
    }
    public void setPhone(long phone)
    { this.phone = phone;
    }
    public String getNumberType()
    { return numberType;
    }
    public void setNumberType(String numberType)
    { this.numberType = numberType;
    }
}
```

⇒ While working with Annotations the default FetchType in OneToMany is FetchType.Lazy & default fetchType is ManyToOne is FetchType.EAGER.

HBProj4(Anno-OneToMany)Uni

- src
  - com.s.h.cfgs
    - hibernate.cfg.xml
  - com.s.h.domain
    - User.java, PhoneNumber.java
  - com.s.h.dao
    - OneToManyDao.java, OneToManyDaoFactory.java, OneToManyDaoImpl.java
  - com.s.h.utility
    - HibernateUtil.java
  - com.s.h.test
    - Main.java

Working with java.util.List type Collection

⇒ In OneToMany association if the collection type is java.util.List we need to maintain an extra index column in child table and we can use

@IndexColumn (Hibernate Annotation that is deprecated) or

@OrderColumn (JPA Annotation) to specify that index column

Example

User.java

```
@OneToMany (targetEntity = PhoneNumber.class, cascade = CascadeType.ALL,
                    orphanRemoval = true)
@JoinColumn (name = "unId", referencedColumnName = "user_id")
@IndexColumn (name = "lst_indx") --> Deprecated or
@OrderColumn (name = "lst_indx")
private List<PhoneNumber> phones;
```

DB Tables : User_table (parent)

User_id (pk)	first_name
101	rafa

Phone_numbers (child)

Phone	number_Type	unId	lst_index (Index column)
9999999999	office	101	0
8888888888	residence	101	1

NOTE: As of now there is no Annotation alternate for <bag> tag.

Taking collection type as java.util.Map

⇒ In One To Many Association if the collection type is java.util.Map then the child table must contain an extra index column to hold the keys of map elements. We need to cfg this column by using @MapKeyColumn annotation (JPA).

Example

User.java

```
@OneToMany(---)
@JoinColumn(---)
@MapKeyColumn (name="map_index")
private Map<String, PhoneNumber> phones;
```

DB Tables

User_Table

User_id	first_name
101	raja

Phone_numbers

Phone	number_type	uid(fk)	map_indx
---			ph1
---			ph2

Annotations based ManyToOne uni-directional (Child parent)

Annotations

@ManyToOne : To refer parent class object being from child class
@ JoinedColumn : To specify FK column

Enums

CascadeType, FetchType, --

Example

Department.java(parent)

```
@Table (name="Department")
@Entity
public class Department
{   @Id
    private int deptno;
    private String deptname,depthead;
 //setters and getters
}
```

EmpDetails.java

```
@Table (name = "EmpDetails")
@Entity
public class EmpDetails
{
  @Id
  private int eno;
  private string ename;
  private long salary;
  @OneToMany( targetEntity = Department.class , cascade = CascadeType.ALL, fetch = FetchType.
                                                                                EAGER)
  @JoinedColumn (name = "deptno", referencedColumnName = "deptno")
  private Department dept ;
  //setters and getters
  ---
}
```

DB Tables

Department (parent)

deptno (pk)	deptname	depthead
1001	accounts	smith

EmpDetails (child)

eno	ename	salary	deptno (fk)
101	raja	9000	1001
102	ravi	9000	1001

OneToMany Bi-Directional

Annotations

@OneToMany : To make parent class property to hold set of child objects

@ManyToOne : To make child class property to hold one parent object
(useful to make multiple child objects to hold one parent class)

@JoinColumn : To specify fk column

mappedBy param

⇒ In Bi-directional association we need to specify FK column at parent class and also at child class by using special annotation like @JoinColumn. If we want to specify that special annotation only at 1 side to reflect that to other side we need to use "mappedBy" param specifying the other side special property.

⇒ This parameter is available only in @OneToOne, @OneToMany, @ManyToMany annotation

Example

User.java (parent)

```
@OneToMany(targetEntity = PhoneNumber.class, cascade = CascadeType.ALL,
           fetch = FetchType.LAZY, mappedBy = "user")
private Set<PhoneNumber> phones;
```

PhoneNumber.java (child)

```
@ManyToOne(targetEntity = User.class, cascade = CascadeType.ALL, fetch = FetchType.LAZY)
@JoinColumn(name = "unid", referencedColumnName = "user_id")
private User user;
```

DB Tables

User_table (parent)
- user_id (PK)
- first_name (varchar2)

Phone_numbers (child)
- phone (PK)
- number_type
- unid (fk)

Annotations based Many To Many Associations

Annotations

@ManyToMany : To build Many To Many Association

@JoinColumn : To specify owning side FK column and reverse side FK column

@JoinTable : To specify 3rd table cfgs

Programmers (table1)

↦ pid (pk)

↦ pname

↦ salary

Projects (table2)

↦ projId (pk)

↦ projname

Programmers_projects (table3)

↦ programmer_id (fk)

↦ project_id (fk)

Programmer.java (parent class)

```
@Entity
@Table(name="programmers")
public class Programmer
{
   @Id
   private int id;
   private String pname;
   private long salary;
@ManyToMany(targetEntity=Project.class, cascade=CascadeType.ALL, fetch=FetchType.LAZY)
@JoinTable(name="programmers_projects", joinColumns=@JoinColumn(name="
 programmer_id", referencedColumnName="pid"),
 inverseJoinColumns=@JoinColumn(name="project_id", referencedColumnName="projid"))
//setters and getters
---
---
}
```

Project.java

```
@Entity
@Table(name="projects")
public class Project
{
   @Id
   private int projId;
   private String progname;
  @ManyToMany(targetEntity = Programmer.class, cascade=CascadeType.ALL,
          mappedBy="projects")
   private set<Programmer> programmers = new HashSet<Programmer>();
   //setters and getters
   ---
   ---
}
```

Working with @CollectionId (Alternate to <idbag> tag)

⇒ While working with ManyToMany association the 3rd table / join table / cross table does not contain uniqueness because fk columns contains duplicate values to build the association. To get uniqueness in 3rd table we can add 1 special index col and we can cfg Id generator to fill the values in the index column (other than assigned)

⇒ To cfg this special index columns through xml mapping we can use <idbag> tag, similarly we can use Hibernate supplied annotation @CollectionId for the same.

Example

Programmer.java

```
@ManyToMany(--)
@JoinTable(---)
@GenericGenerator(name="gen1", strategy="increment")
@CollectionId(columns=@Column(name="proj-prgmr_index"), type=@Type(type="int"),
    generator="gen1")
private List<Project> projects =new ArrayList<Project>();
```

Programmers_projects (join table)

Programmer_id	project_id	proj_prgmr_index
101	1001	1
101	1002	2
102	1002	3

Annotations based One To One Association

⇒ It can be implemented in 2 ways

a) As One To One FK

b) As One To One PK

1) One To One FK

Annotations

a) @ManyToOne with unique="true", not-null="true" (To build one to one fk association)

b) @JoinColumn (To specify FK column)

Example (One To One FK uni-directional)

Person (parent)

↦ person_id (PK)

↦ first Name

↦ last Name

↦ age

License (child)

↦ license_id (PK)

↦ type

↦ valid_from (date)

↦ valid_to (date)

↦ license_holder (fk)

Person.java

```
@Entity
@Table
public class Person
{
    @Id
    @Column (name ="person_id")
    private int id;
    private string firstName;
    private String lastName;
    private byte age;
   //setters and getters
  - - -
  - - -
}
```

License.java (child)

```
@Entity
@Table
public class License
{
  @Id
   @column (name = "license_id")
   private int lid;
   private String type;
   @Column (name = "valid_from")
   private Date datefrom;
  @ column(name="valid_to")
  private Date dateto;
@ManyToOne (targetEntity = Person.class, unique="true", not-null="true")
@JoinColumn(name="license_holder", referenced ColumnName="person_id")
private Person licenseholder;
//setters and getters
- -
}
```


⇒ If we use @OneToOne with @JoinColumn then it becomes OneToOneFK association.
If we use @OneToOne with @PrimaryKeyJoinColumn then it becomes OneToOnePK association.

2) OneToOne PK

Annotation

@OneToOne : To build OneToOne Association

@PrimaryKeyJoinColumn : Uses the PK column value of parent table as PK column value of child table with out having any FK constraint support.

Example

OneToOne PK Bi-directional

⇒ Here FK column is not required but "foreign" generator cfg is required to copy the identity value of parent object as the identity value of child object.

DB Table

student
- ⟼ sid (PK)
- ⟼ name
- ⟼ address

Lib_Membership
- ⟼ lid (PK)
- ⟼ doj (date)

student.java

```
@Entity
@Table
public class student
{
  @Id
  @Column(name="sid")
  @GenericGenerator(name="gen1", strategy="increment")
  @GeneratedValue(generator="gen1")
  private int id;
  private string name, address;
  @OneToOne(targetEntity=LibraryMembership.class, cascade=CascadeType.ALL,
                                  mappedBy="studentDetails")
  private LibraryMembership libraryDetails;

  //setters and getters
  ---
}
```

LibraryMembership.java

```
@Entity
@Table (name="lib_membership")
public class LibraryMembership
{
    @Id
    @Column(name="lid")
    @GenericGenerator( name="gen1", strategy="foreign",
        parameters= {@Parameter(name="property", value="studentDetails")})
    @GeneratedValue( generator="gen1")
    private int id;
    @Column(name="doj")
    private Date joiningDate;
    @OneToOne (targetEntity=Student.class, cascade=CascadeType.ALL)
    @PrimaryKeyJoinColumn(name="lid", referencedColumnName="sid")
    Private Student studentDetails;
    // setters and getters
    ---
}
```

Q What is the difference b/w FetchType and FetchMode in annotations ?

A FetchType specifies weather object should be loaded lazily/eagerly whereas FetchMode specifies the no. of select SQL queries that should be generated while loading the objects. Use the HB supplied @fetch annotations to specify FetchMode & use fetch attribute of various association annotations to specify "FetchType".

Example

In User.java

```
@OneToMany(targetEntity=PhoneNumber.class, fetch=FetchType.LAZY/EAGER,
                cascade=CascadeType.ALL)  +---? //fetchType
@JoinColumn(---)
@Fetch(FetchMode.SELECT/SUBSELECT/JOIN) +---? //fetchMode
Private Set<PhoneNumber> phones;
```

NOTE : We must use Criteria API logic in order to take the benefit of FetchMode.

Second Level Caching Hibernate | L2 | Global | Level2 | SessionFactory cache

⇒ Hibernate supports two levels of caching to reduce nlw round trips between client & server.

(1) Level1 | L1 | First Level | Local Machine

(i) It is built in cache associated with session object

(ii) It is 1 per session object

(2). Level2 | L2 | Second Level | Global cache

(i) It is configure cache associated with SessionFactory object

(ii) It is 1 per SessionFactory object

⇒ When HB persistence logic loads the object

a) Searches for that in L1 cache if available takes that object, if not available

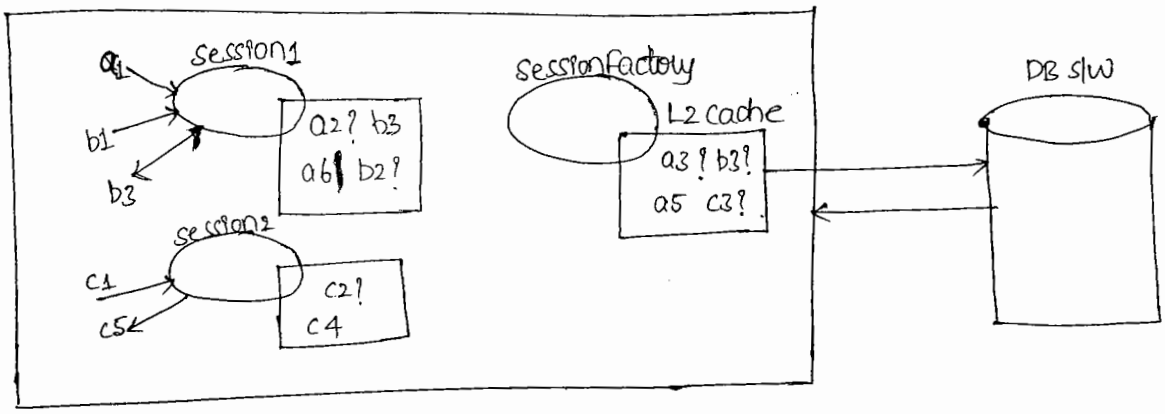
b) Searches for that in L2 cache if available takes that object, if not available

c) Gets the object by selecting record from DB table

d) Keeps that object in L2 cache

e) Keeps that object in L1 cache

f) Gives that object back to clientApp



NOTE

All the 3 requests are same requests.

⇒ Second Level cache is configurable caches. The following second level cache slwls are available. They are

a) Ehcache (Easy HB)
b) Swarm cache
c) Jboss Tree Cache
d) OS Cache
e) Tangosol coherence cache (commercial)

⇒ We need to specify Concurrency strategy towards placing objects & retrieving objects from L2 cache They are

a) Read-only
b) Read-write
c) Non strict read-write
d) Transactional

⇒ For more info various second level caches & concurrency strategy for ref page no: 190 to 192.

⇒ To cfg second level cache we must know the cache provider class name

EhCache → EhCacheProvider (Old) | EhCacheRegionFactory (new version) 4.x

OsCache → OsCacheProvider

and refer page no: 191

⇒ To control second level cache factory.close() → closes SessionFactory & also closes L2 cache

factory.evict(classname): → Removes the object of class from L2 cache. |*|

factory.evictCollection(): → Removes collection object from L2 cache.

*| → Deprecated.

⇒ We can also use Cache.evict() and Cache.evictCollection() for the same.

⇒ We can get cache object by using factory.getCache() method.

Example to cfg SecondLevel (EhCache) in HiberateApp

1) Keep any HBApp ready (FirstApp)
2) Add the following additional jar files

HB 4.3 home | lib | optional | EhCache all (jars) (3 jar files)

3) Add the following entries in HB cfg file to enable L2 cache

In Hibernate.cfg.xml

```
<property name="cache.use-second-level-cache">true</property>
<property name="cache.region.factory_class">org.hibernate.cache.EhCacheRegionFactory</property>
<property name="net.sf.ehcache.configurationResourceName">/com/nt/cfgs/ehcache.xml</property>
```

4) Develop ehcache.xml file in com.nt.cfgs pkg

Ehcache.xml

```
<ehcache>
  <diskStore path="java.io.tmpdir"/>
  <defaultCache maxElementsInMemory="1000" maxEternal="false"
    timeToIdleSeconds="10" timeToLiveSeconds="20" overflowToDisk="true" />
</ehcache>
```

5) Specify cache strategy in mapping file

Before <id> tag in Employee.hbm.xml

```
<cache usage="read-only"/>
```

6) Write following code in the Client Application

```
Session ses1 = HibernateUtil.getSession();
Session ses2 = HibernateUtil.getSession();
//Load object
EmpDetails ed = (EmpDetails) ses1.load(EmpDetails.class, 1001);
  //gets from DB & keeps in L2, L1 (ses1) caches
  S.O.P(ed);
EmpDetails ed1 = (EmpDetails) ses1.load(----);
  //gets from L1 cache
  S.O.P(ed1);
  ses1.clear(); //removes from L1 cache (ses1)
EmpDetails ed2 = (EmpDetails) ses2.load(----);
  //gets from L2 cache & keeps in L1 (ses2) cache
  S.O.P(ed2);
Thread.sleep(12000); //Removes from L2 cache
  ses.clear(); // Removes from L1 (ses2) cache
EmpDetails ed3 = (EmpDetails) ses2.load(----);
  //gets from DB & keeps in L2, L1 cache
  S.O.P(ed3);
```

We can specify caching strategy through annotation on the top of the domain class by using '@Cache' annotation

```
@Cache( usage = CacheConcurrencyStrategy.READ_ONLY)
@Entity
@Table (name="Employee")
public class EmpDetails
 {
   ---
   ---
 }
```

⇒ Query cache is part of second level cache & it maintains HQL Query results across the multiple execution with same arguments (parameter values)

⇒ Second level cache & Query Cache are introduced to HB from 3.x version

Example

1) keeps previous example that deals with second level cache ready

2) Add following property in HB cfg file

```
<property name="cache.use_query_cache"> true </property>
```

3) write following code in the client Application

```
Query query = ses.createQuery("from EmpDetails");
      query.setCacheable(true);
      query.setCacheRegion("region1");
List<EmpDetails> list = query.list();
 //gets from DB & keeps query cache
 for(EmpDetails ed:list)
 {
   S.O.P(ed);
 }
 S.O.P("---");
List<EmpDetails> ----
 for(---)
 {
 }
//HibernateUtil.factory.evictQueries("region1"); ment
        (or)
HibernateUtil.factory.getCache().evictQueryRegion("region1");
//removes from query cache
List<EmpDetails> list2 = query.list();
//gets from DB & keeps in query cache
```

⇒ If you don't specify query cache region name then the HQL Query results will be saved in default query cache region.

Transaction Management (Tx Mgt)

⇒ The process of combining related operations into a single unit and executing them by applying do-everything or nothing principles is called Transaction Management.

⇒ We execute sensitive logics like ticket booking, transfer money & etc.. logics by enabling Transaction Management support.

⇒ TransferMoney operation is the combination of 2 operations

a) Withdraw amount from source amount

b) Deposit amount into another account

NOTE If one operations fails another operation should ~~do~~ be failed. This demands Transaction Management support

⇒ While working with Transaction Management 3 operations are important

a) Begin Transaction

b) Continue transaction

--

c) Commit or rollback transaction

⇒ If only resource (DB s/w) is involved in Transaction Management operation then it is called Local Transaction

Ex: Transfer Money operation b/w two accounts of same bank

⇒ If more than resource (DB s/w) is involved in Transaction Management operation then it is called Global/Distributed Transaction.

Ex: HB doesn't supports Global Transaction but spring, EJB supports Global Transactions

All technologies, f/w supports Local Transactions (jdbc, hibernate, spring, ejb, --).

Sample Code in HB.

```
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    With draw operation;
    Deposit operation;
    tx.commit();
}
catch(Exception e){
    tx.rollback();
}
```

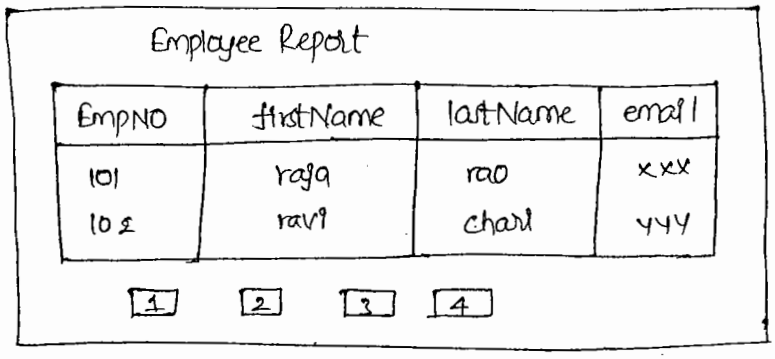
Pagination (using servlet with HB)

⇒ The process of displaying huge amount of records part by part in multiple pages is called "Pagination". This is very useful in Report generation.

⇒ we generally do Report Generation by using Criteria API with the support of setFirstResult, setMaxResults Method as shown below.

```
Criteria ct = ses.createCriteria(EmpDetails.class);
         ct.setFirstResult(0);
         ct.setMaxResults(3);
        // execute QBC
List<EmpDetails> list = ct.list(); //gives 1st 3 records
  // process the results
  for(EmpDetails ed: list)
   {
      s.o.p(ed);
   }
```

WebPage



HB Proj60 (Pagination)(ECDW)

```
|→java resources
     |→com.s.h.cfgs
          |→hibernate.cfg.xml
     |→com.s.h.domain |→EmpDetails.java
     |→com.s.h.dao |→EmpDao.java, EmpDaoFactory.java, EmpDaoImpl.java
     |→com.s.h.utility |→HBUtil.java
     |→com.s.h.test |→ClientApp.java
|→webcontent
     |→index.html
     |→WEB-INF |→web.xml
```

Locking in Hibernate

⇒ If multiple client apps, Threads access the records of DB Table simultaneously or concurrently then it is possibility of getting data corruption.

⇒ To solve this concurrency problem we need to use locking concepts of Hibernate.

a) Optimistic Locking

⇒ Use versioning feature of hibernate for this locking

⇒ This locking throws exception if the version of the objects i.e., these with client App is not matching with the version of record i.e., these with DB table

⇒ That exception will be raised is StaleObjectStateException / DirtyStateException i.e., state of the object i.e, these client App is not matching with the state of the record i.e, these is DB Table.

b) Pessimistic Locking

1) In this if one client access the record then lock will be applied on the record. So other clients can not access the records simultaneously until that lock is released.

2) For this we need to use ses.get() by specifying the LockMode as UPGRADE_NOWAIT

3) Pessimistic lock is recommended to use

4) Here No versioning is required

⇒ If we execute any application by enabling versioning features the optimistic lock will be applied automatically. If any version clash between client Application & DB table record StaleObjectStateException will be raised (refer APP21 of booklet).

Execution Procedure

a) Client1

Access the 101 record where version is 0 and goes to sleep state

b) Client2

Access the same 101 record and updates the record due to this version will be changed to 1 in DB table.

c) Client1

Completes the sleep state & come back to manipulate the record & notices versions are not matching due to this exception (StaleStateException) will be raised

Ex: App on Pessimistic Locking refer App22 of Main Booklet.

Execution

client 1

Access the 101 record and gets into sleep state

(Record is accessed through ses.get(Product.class, 101, LockMode.UPGRADE_NOWAIT)

client 2

During sleep period second client wants to access the 101 record. Due to this Error will be generated with message "Resource Busy".

Q Explain Different ORM Levels ? or ORM Quality Levels ?

Ref page no : 302, 303 of Main Booklet

1) Pure Relational (Stored Procedure)
2) Light Object Mapping (JDBC)
3) Medium Object Mapping (EJB)
4) Full Object Mapping (Composition, Inheritance..)

Q Mismatches b/w RDBMS DB s/w & Java Programming and how HB solves those mismatches?

Inheritance (subtype Problems)

Java classes can be there in the inheritance, But table can't be placed in the inheritance. For this use Inheritance Mapping concepts.

Problem of Granularity

Java classes can be designed through composition but DB Tables cannot be placed in composition. HB gives component Mapping to solve this problem.

Identity problem

In Java we can check the quality either through "==" operators (or) .equals() method. In DB s/w record is identified with its PK column value. HB solves the problem by identifying the object through identity field cfg <id> (or) @Id.

Navigation Problem

In db tables we can navigate from one table record to another table record through FK colum. But navigation b/w Java objects is possible through reference variables (parent class should have child class reference & vice versa)

HB solves this problem through Association Mapping.