Sathya Technologies, Ameerpet

Hibernate

By: Raghu

HIBERNATE FRAMEWORK

Framework: -

It is a combination of technology & Design Pattern which is used to develop application faster also called as "RAD [Rapid application development]".Ex:Hibernate, Spring, Struts, etc.

Technology:-

It is a concept used to develop one fixed type of process. It is a created using Core Java mainly in case of Java technology. Ex: Technology is JDBC, Servlet, JSP, EJB, JMS, Java Mail, JAXB, etc.

Design Pattern:-

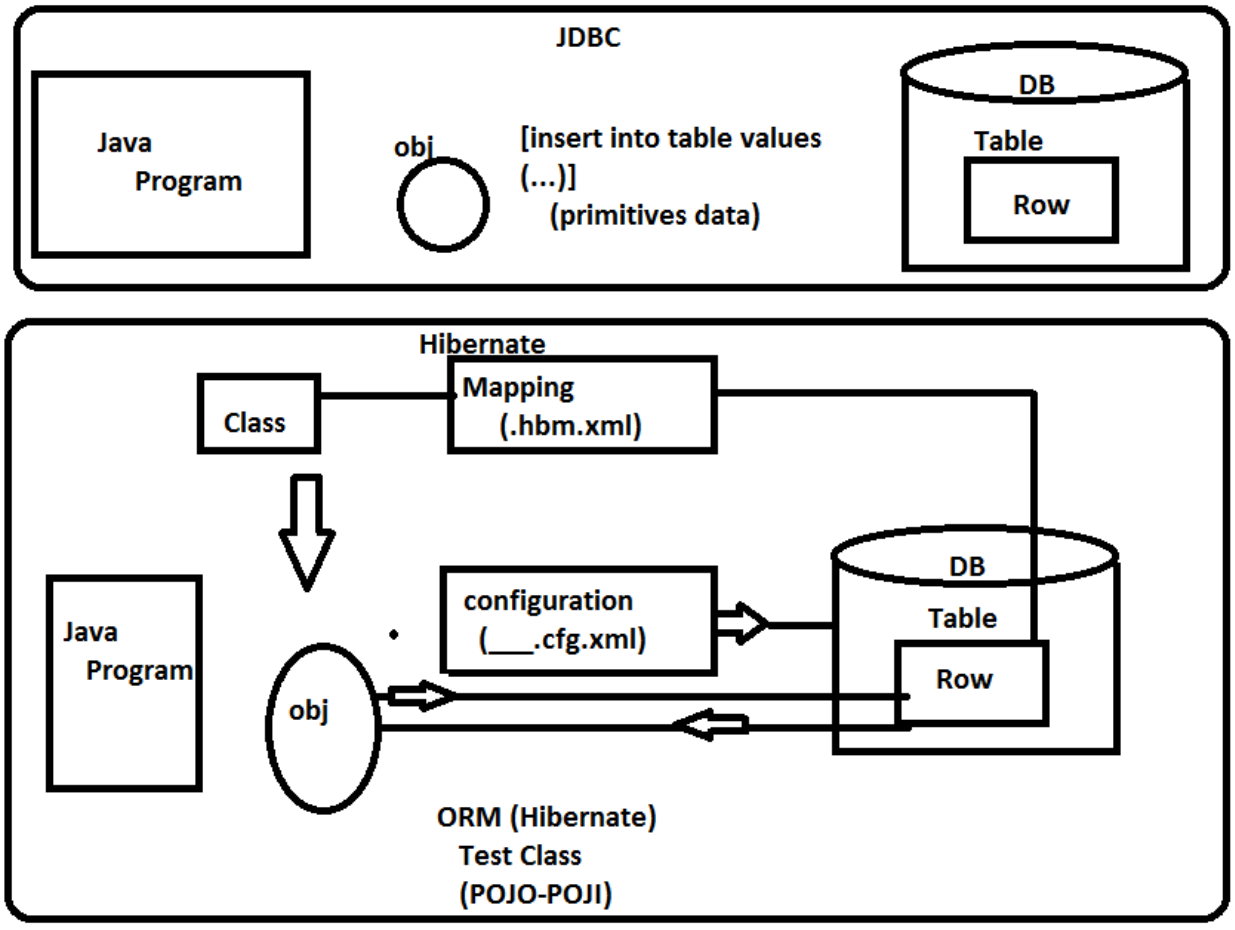
It is a design for coding (Or application development to write in better manner i.e. less no of lines of code and execution in less time, takes less memory) Ex. Singleton, Prototype, Factory, MVC, Template, Abstract, POJI-POJO.

Performance is inversely proportional to Code [execution time, Memory].

ORM (Object Relational Mapping): -

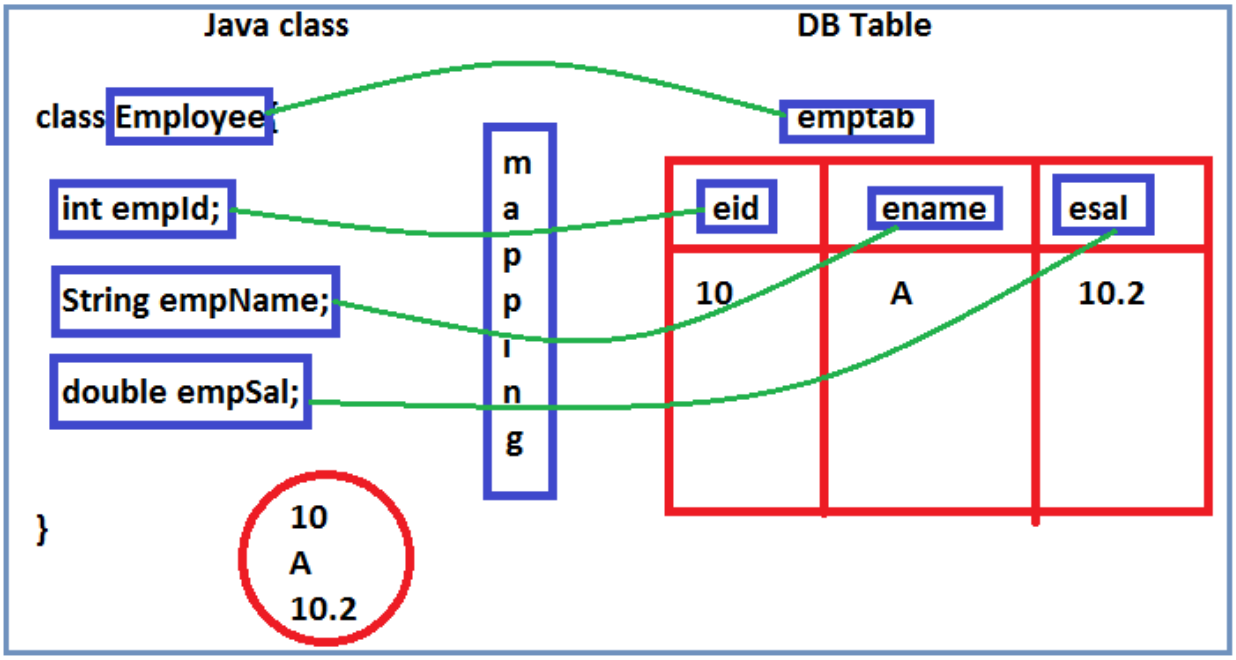
- It is theory concept followed by Hibernate. In case of JDBC it will perform operations in primitive's format even Java Application gives object.
- ORM will convert object to row and row to object, but to get this we [programmer] should follow mapping rule. I.e. class name-mapped with – Table Name, Variable-mapped with-column.

Diagram: -



- Ex: - Mapping can be done using XML/Annotations concept.
- Here Mapping Annotations are called as JPA (Java Persistence API Annotations.
- Persist = Link with D.

Diagram: -



- To write one Hibernate Application basic files are: -
- Model/Entity class
- Mapping file (XML/Annotations)
- Configuration file (cfg.xml)
- Test class.

Hibernate: -

It is a framework having below Java Technologies for DB Programing: -

1.**JDBC** 2.**JTA(Java Transaction API)** 3.**JNDI(Java Naming Directory Interface).**

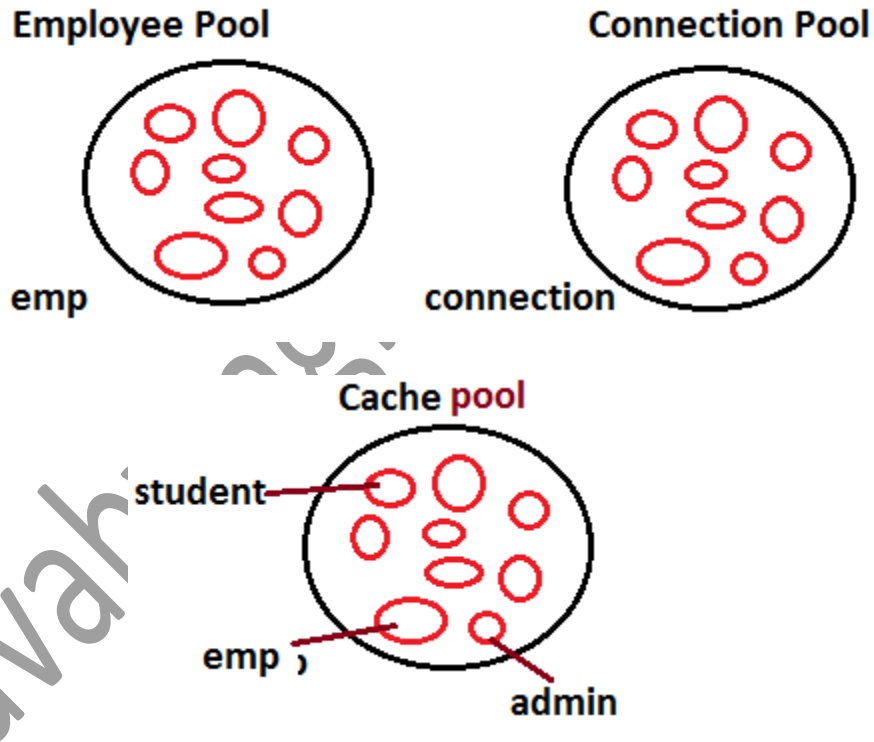
1. **JDBC: -** It is used to perform DB operations like insert/update/delete/select operations.
2. **JTA: -**It provides two operations commit(confirm changes) and rollback(cancel changes) on DB operation.
   If set of queries executed without any problem exception or error then commit operations is used.
   Else we can cancel all queries using Rollback operation.

3. **JNDI: -** This concept is used to maintain "Connection Pool" (Set of Connection Object) that provides faster services for DB operations.

Pool and Cache: -

Both are temporary memories.

1. **Pool: -** It is set of same type objects.
   Ex: Employee objects set = Employee Connection Pool
   String constant pool = Set of String Objects.
   Connection pool = Set of Connection objects.
2. **Cache: -** It is the Collection of different type of objects.
   Ex: employee, student, admin objects as one memory.



Hibernate Configuration File: -

This is first file in Hibernate coding follows below naming format: "hibernate.cfg.xml".

- It contains Properties in key=value format, both key and values are type String.

- **Connection Properties are 4:**
  1. driverClassName 2. url 3.userName 4.Password
- **Hibernate Properties are:**
  5. **dialect: -** It is a class, generates SQL queries when programmer performs operations based on DB details.
  Ex. OracleDialect, MySQLDialect, SybaseDialect, etc.
  Dialect makes Hibernate Application as DB independent.
  6**.show_sql: -** It is a Boolean property, default value is false. If it is true then Hibernate Framework prints Generated SQL's on console.
  7**.hbm2ddl.auto: -** Possible values are
  a) create b) update c) **validate d) create-drop
  a) create: - On every operation it (Hibernate) creates new table if Table exist then it will dropped and creates new Table.
  b) update: - If exists uses same else creates new table.
  c) validate: - It is only default value, In this case hibernate will not create/modify any table only programmer can perform DDL operation.
  d) create-drop :- It will creates a table , performs operations and drops the table.

Model Class/Entity Class:-

A class that connected to DB table is called as Entity Class.

Here Model means Data(Object).

We should follow below rules to write model Class: -

1. Class must have package statement.
2. Class must be public.
3. Class can have variables those must be type private.
4. Class must have default constructor.
5. Write setters/getters method for every variable in the class.
6. ** Model class can override 3 methods from Object(java.lang) class Those are toString(), hashCode() and equals() [non-static, non-final, no-private].

7. Model class can have annotations.
   - ➔ Core Annotations (java.lang package).
   - ➔ JPA Annotations (Java Persistency API).
8. Model class can inherit Hibernate API (classes & interface) and also one interface is allowed i.e. Serializable (java.io) (I).

Printing decimal number in other formats (Binary,octal,hex): -

Binary:- 1/0
Octal:- 0 2 3 4 5 6 7
Hex:- 0 1 2 3 4 5 6 7 8 9 10 A/a 11 B/b 12 C/c 13 D/d 14 E/e 15 F/f
Use Wrapper classes static methods to do number conversion.

Code: -

```
package org.sathyatech.test;

publicclass ClientApp {

	publicstaticvoid main(String[] args) {

		intsid = 10;

		// printing decimal number in other formats (Binary,octal,hex).
		// Binary:- 1/0
		// Octal:- 0 2 3 4 5 6 7
		// Hex:- 0 1 2 3 4 5 6 7 8 9 10 A/a 11 B/b 12 C/c 13 D/d 14 E/e 15 F/f
		// Use Wrapper classes static methods to do number conversion.

		String bin = Integer.toBinaryString(sid);
		System.out.println(bin);
		System.out.println("------------------");

		String oct = Integer.toOctalString(sid);
		System.out.println(oct);

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

```
        String hex = Integer.toHexString(sid);
        System.out.println(hex);
        System.out.println(hex.toUpperCase());
        System.out.println("------------------");

        intsid1 = 0xabcd; //hex
        intsid2 = 012;  //oct

        System.out.println(sid1);
        System.out.println("------------------");
        System.out.println(sid2);
}
}
```

toString: -

This method is from java.lang.Object class and it is non-final, non-static, and non-private also.

1. By default it will be called when we print reference variable (or object name) automatically. Ex: Employee emp = new Employee();
   Here emp is reference variable
   System.out.println(emp) is equal to
   System.out.println(emp.toString());
2. toString() method can be overridden in child (or any) class. If not overridden, it prints by default fullClassName@hashCode in HexaDecimal format.
3. ----------Ex: Code----------

```
Public class Test{
   Psvm(){
        TreeMap tm = new TreeMap();
        Tm.put(“10”,”A”);
        Sysout(tm.getClass().getName());
        Sysout(“@”);
        Int h = tm.hashCode();
```

```
            String hs = Integer.toHexString(h);
            Sysout(hs);
        }
    }
```

4. In Java API, almost every class has overridden toString(). Ex are String, ArrayList, Exception, HashMap etc.
5. In Hibernate Model Class override toString() method using Eclipse shortcut: - "Alt+Shift+S" then press "S" again, click "OK".

Model Class/ Entity Class: -

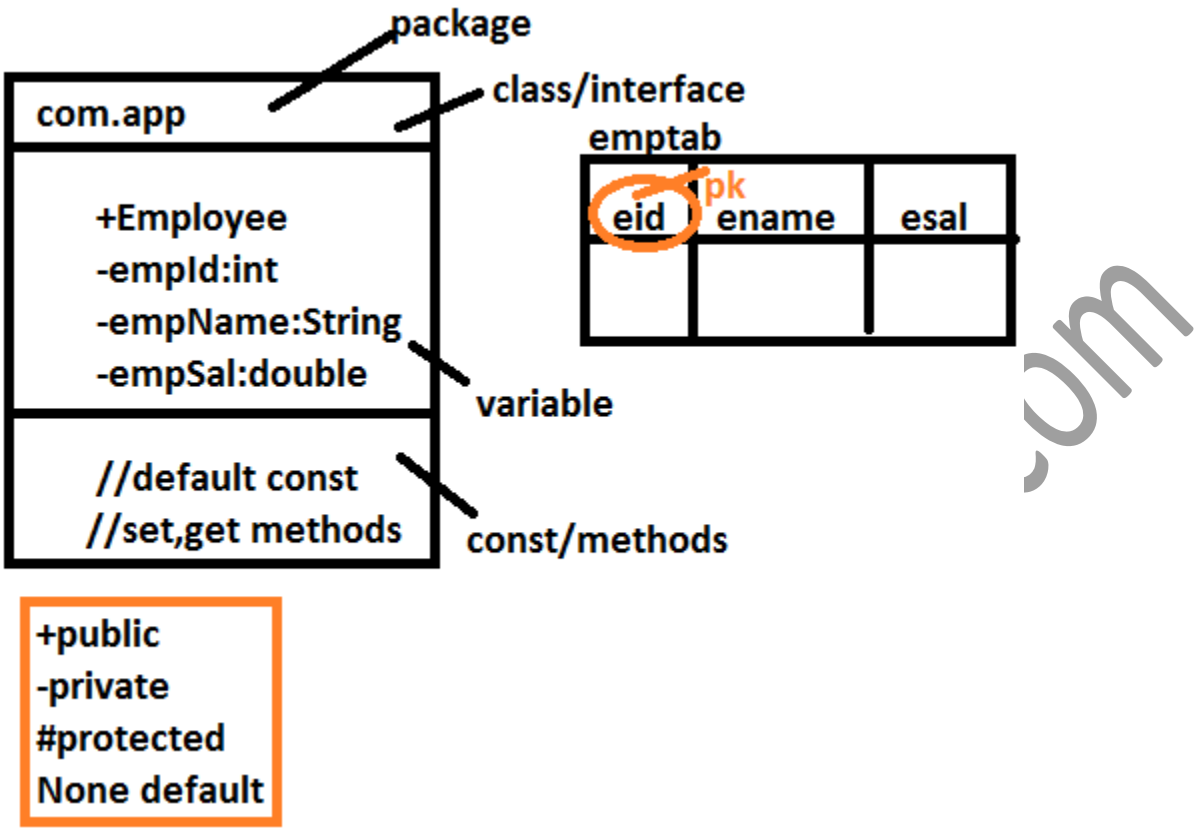
A class that represents DB table or indicates data is known as Model(Data) or Entity(Connected to table) class.

**Every Model class must be connected to one table.

**No. of columns in table is equal to No. of variables in Model class.

****Every table must contain Primary key as per Hibernate design.

UML Class Design: -



Example 1: -Define Employee Module Entity class using below design with Annotations Mapping code (@Entity, @Id, @Table, @Column)..

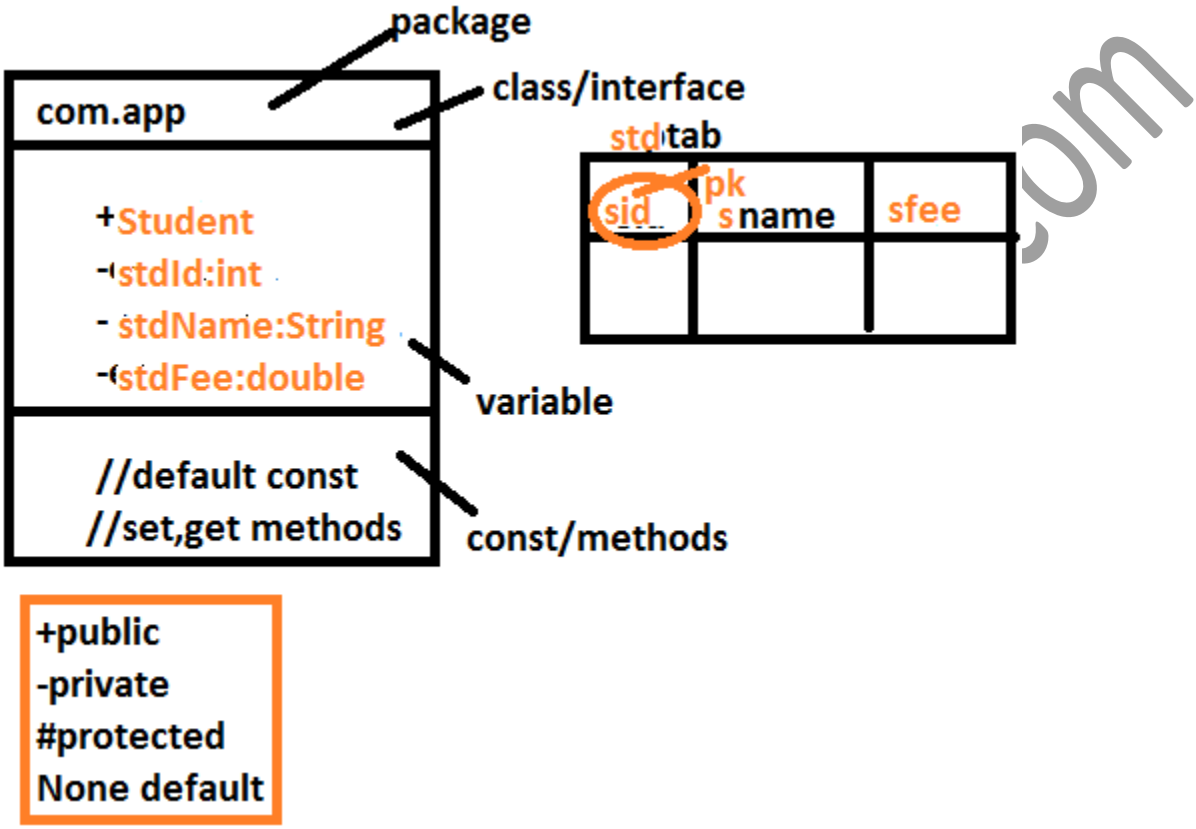
```
package org.sathyatech.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="emptab")
publicclass Employee {

        public Employee() {
         System.out.println("Employee::-0-param constructor");
```

```
	}

	@Id
	@Column(name="eid")
	privateintempNo;

	@Column(name="ename")
	private String empName;

	@Column(name="esal")
	privatedoubleempSal;

	publicint getEmpNo() {
		returnempNo;
	}
	publicvoid setEmpNo(intempNo) {
		this.empNo = empNo;
	}
	public String getEmpName() {
		returnempName;
	}
	publicvoid setEmpName(String empName) {
		this.empName = empName;
	}
	publicdouble getEmpSal() {
		returnempSal;
	}
	publicvoid setEmpSal(doubleempSal) {
		this.empSal = empSal;
	}
	@Override
	public String toString() {
		return"Employee [empNo=" + empNo + ", empName=" + empName
+ ", empSal=" + empSal + "]";
	}
}
```

While applying multiple Annotations at one level we can write in any order.

Example 2: -

UML Class Design: -




```
package org.sathyatech.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="stdtab")
publicclass Student {


        public Student() {
```

```
        System.out.println("Student::-o-param constructor");
    }

    @Id
    @Column(name="sid")
    privateintstdId;

    @Column(name="sname")
    private String stdName;

    @Column(name="sfee")
    privatedoublestdFee;

    publicint getStdId() {
        returnstdId;
    }

    publicvoid setStdId(intstdId) {
        this.stdId = stdId;
    }

    public String getStdName() {
        returnstdName;
    }

    publicvoid setStdName(String stdName) {
        this.stdName = stdName;
    }

    publicdouble getStdFee() {
        returnstdFee;
    }

    publicvoid setStdFee(doublestdFee) {
        this.stdFee = stdFee;
    }
```

```
	@Override
	public String toString() {
		return"Student [stdId=" + stdId + ", stdName=" + stdName + ",
stdFee=" + stdFee + "]";
	}

}
```

@Entity: - *It maps class with table and variable with columns.*

@Id: - *It indicates primary key.*

@Table: - *To provide table name, it is optional. If Table is not provided then className is taken as tableName.*

@Column: - *To provide column details. It is optional. If it is not provided then variable Name is columnName.*

All these annotations are from package **"javax.persistence".

Hibernate Configuration File: -

This is also called as setup file. It should be created with name "***hibernate.cfg.xml***" in application.It must contain 4 connection properties those are ***driver class name, url, username, password***.

And should also contain ***dialect (class generates SQL).***

show_sql and ***hbm2ddl.auto*** are optional.

Cfg file = property [key=value] + mapping class

Full keys details are: -

```
	<hibernate-configuration>
<session-factory>
<!-- Connection setting -->
<property name="hibernate.connection.driver_class"> _______</property>
	<propertyname="hibernate.connection.url"> _____</property>
```

```
        <propertyname="hibernate.connection.username">_____</property>
        <propertyname="hibernate.connection.password">_______</property>


                    <!-- hibernate setting -->
        <propertyname="hibernate.dialect"> _________</property>
        <propertyname="hibernate.show_sql">____</property><!—true/false -->
                <!-- It shows the hibernate fired sql query on console -->
        <propertyname="hibernate.hbm2ddl.auto">____</property>
                create/update/create-drop/validate<!-- It will create table if not in
DB -->

<!-- mapping resources/locations  -->
        <mappingresource="com/nj/emp_app/config/employee.hbm.xml"/>

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

-------*CFG with Oracle Database Example*-----

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

        <!-- Connection setting -->
        <propertyname="hibernate.connection.driver_class">oracle.jdbc.driver.Ora
cleDriver</property>
        <propertyname="hibernate.connection.url">jdbc:oracle:thin:@localhost:15
21/orcl</property>
        <propertyname="hibernate.connection.username">system</property>
        <propertyname="hibernate.connection.password">tiger</property>

<!-- hibernate setting -->
        <propertyname="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect
</property>
        <propertyname="hibernate.show_sql">true</property><!-- It shows the
hibernate fired sql query on console -->
        <propertyname="hibernate.hbm2ddl.auto">update</property><!-- It will
create table if not in DB -->
```

```
                <!-- mapping resources/locations  -->
        <mappingresource="com/nj/emp_app/config/employee.hbm.xml"/>

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

------*CFG with MySQL Database Example*------

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

        <!-- Connection setting -->
        <propertyname="hibernate.connection.driver_class">com.mysql.jdbc.Drive
r</property>
        <propertyname="hibernate.connection.url">jdbc:mysql://localhost:3306/h
bDB</property>
        <propertyname="hibernate.connection.username">root</property>
        <propertyname="hibernate.connection.password">root</property>

<!-- hibernate setting -->
        <propertyname="hibernate.dialect">org.hibernate.dialect.MySQLDialect</
property>
        <propertyname="hibernate.show_sql">true</property>
        <propertyname="hibernate.hbm2ddl.auto">update</property>

<!-- mapping resources/locations  -->
        <mappingresource="com/nj/emp_app/config/employee.hbm.xml"/>

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

Hibernate Test Class: -

This class is used to perform DB operation like save, update, delete and select.

- Here Operation are considered as 2 types:

1. Select Operation
2. Non-select Operations (insert, update, delete).

Transaction should be applied for non-select operations, begin-transaction then commit/rollback.

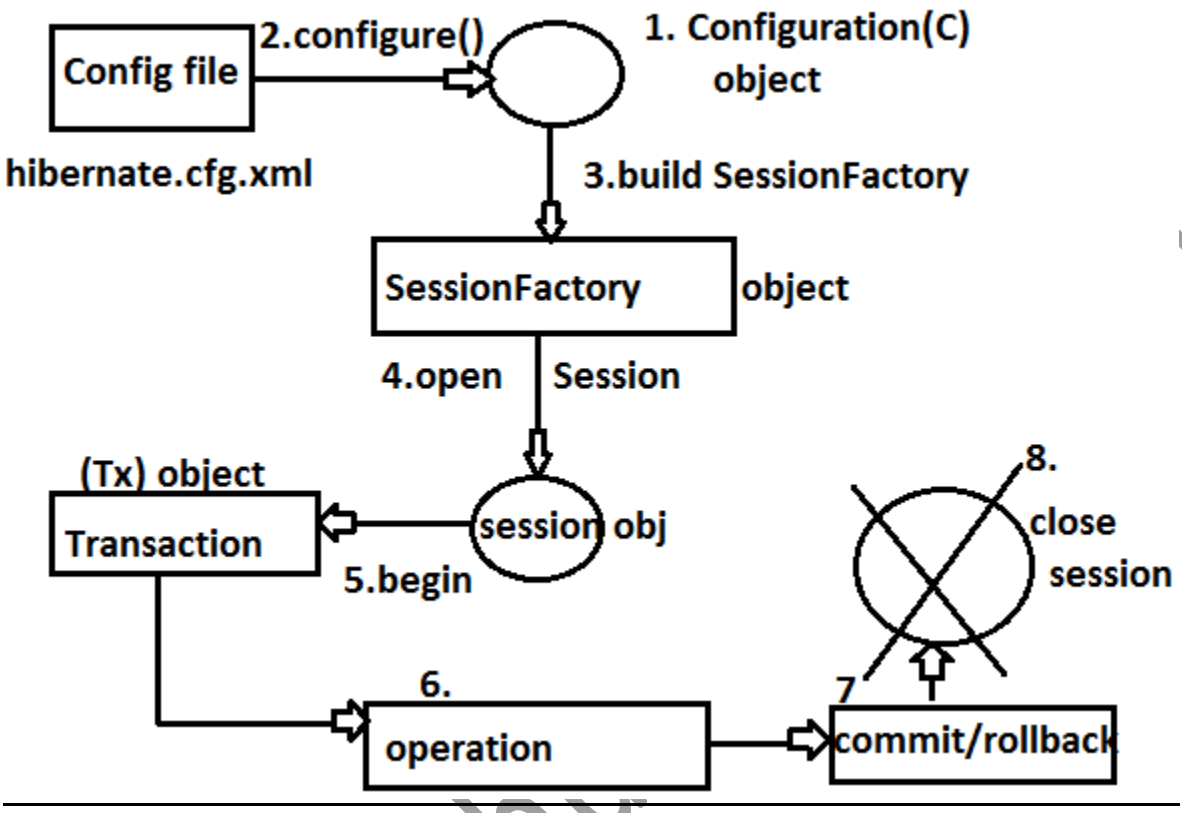
For select Operation Transaction is not required. Even applied by programmer, gives no meaning (no error/no exception).

Here SessionFactory(I) will perform:-

1. Loading driver class
2. Connection Management (create, maintain, close connection object)
3. Statement objects management.

Test Class Flow: -

1. Create Configuration class object.(org.hibernate.cfg)
   Ex: **Configuration cfg = new Configuration();**
2. Load Configuration XML File into cfg object using
   cfg.configure();
3. Build one SessionFactory(I) object using cfg object.
   SessionFactory sf = cfg.buildSessionFactory();
4. Open Session using SessionFactory sf obj.
   object Session ses = sf.openSession();
5. Begin Transaction using Session ses object
   Transaction tx =ses.beginTransaction();
6. Perform operation using session object.
   Ex: save(), update(), delete(), get(), load()….
7. Commit/Rollback Tx if created
   tx.commit() / tx.rollback()

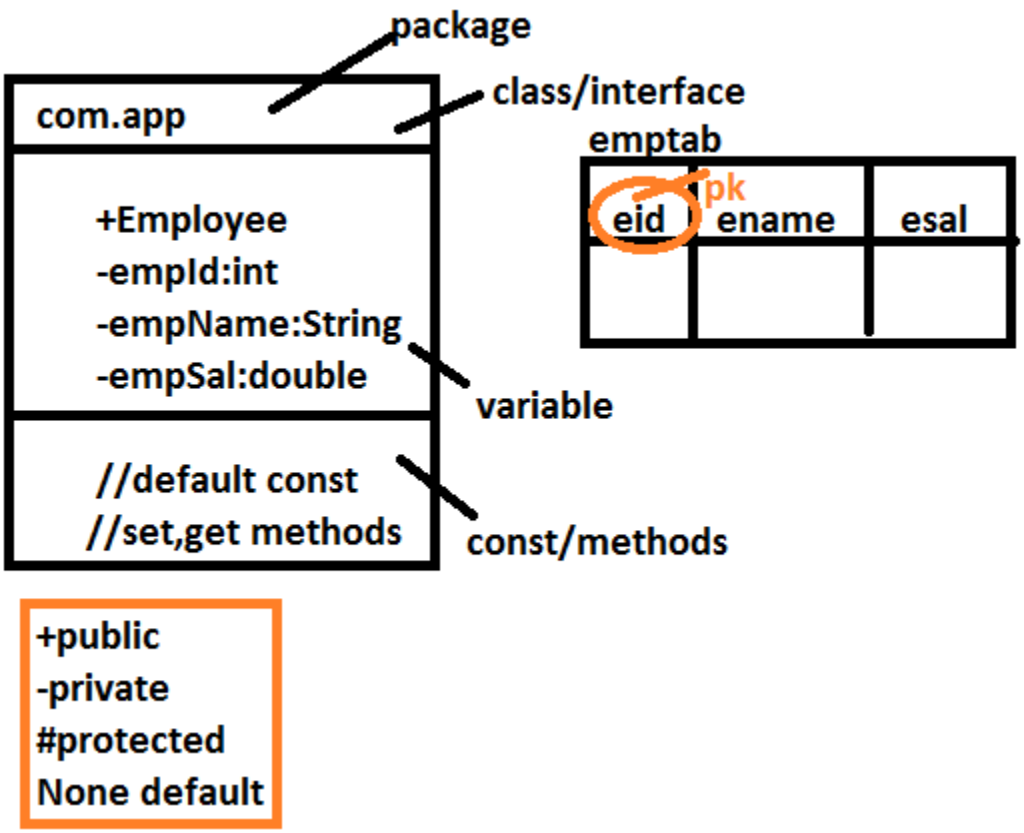


Example code with imports: -

```
package org.sathyatech.emp_app.test;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

publicclass ClientApp {

        publicstaticvoid main(String[] args) {

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

```
            SessionFactory sf = cfg.buildSessionFactory();
            Session ses = sf.openSession();
            Transaction tx = ses.beginTransaction();
            //operations
            tx.commit(); // tx.rollback();
            ses.close();
        }
}
```

Hibernate First App: -



Employee (Model) Class: -

```
@Entity
```

```
@Table(name="emptab")
publicclass Employee {
	@Id
	@Column(name="emp_Id")
	privateintempId;
	@Column(name="emp_Name")
	private String empName;
	@Column(name="emp_Sal")
	privatedoubleempSal;

	public Employee() {
	 System.out.println("Employee:: 0-param constructor");
  }

	publicint getEmpId() {
		returnempId;
	}
	publicvoid setEmpId(intempId) {
		this.empId = empId;
	}
	public String getEmpName() {
		returnempName;
	}
	publicvoid setEmpName(String empName) {
		this.empName = empName;
	}
	publicdouble getEmpSal() {
		returnempSal;
	}
	publicvoid setEmpSal(doubleempSal) {
		this.empSal = empSal;
	}
	@Override
	public String toString() {
		return"Employee [empId=" + empId + ", empName=" + empName +
", empSal=" + empSal + "]";
	}
```

```
}
```

Hibernate Configuration File: -

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

        <!-- Connection setting -->
<propertyname="
connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<propertyname="
connection.url">jdbc:oracle:thin:@localhost:1521/orcl</property>
<propertyname="hibernate.connection.username">system</property>
<propertyname="hibernate.connection.password">tiger</property>

<!-- hibernate setting -->
<propertyname="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>
<propertyname="hibernate.show_sql">true</property
<propertyname="hibernate.hbm2ddl.auto">update</property>

<mappingclass="com.sathyatech.model.Employee"/>

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

Test Class: -

```
package org.sathyatech.emp_app.test;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import org.sathyatech.emp_app.model.Employee;
```

```
public class ClientApp {

        public static void main(String[] args) {

                Configuration cfg = new Configuration();
                cfg.configure();
                SessionFactory sf = cfg.buildSessionFactory();
                Session ses = sf.openSession();
                Transaction tx = ses.beginTransaction();
                //operations
                Employee emp = new Employee();
                emp.setEmpNo(101);
                emp.setEmpName("Ram");
                emp.setEmpSal(12.36);
                tx.commit(); // tx.rollback();
                ses.close();
        }
}
```


Database operations in Hibernate using Session (I): -

Session(I) supports two types of operations in Hibernate. Those are **select** operations and **non-select** operations.

Non-select example operations are: -

1) save(obj) : serializable
2) update(obj) : void
3) saveOrupdate(obj) : void
4) delete(obj) : void

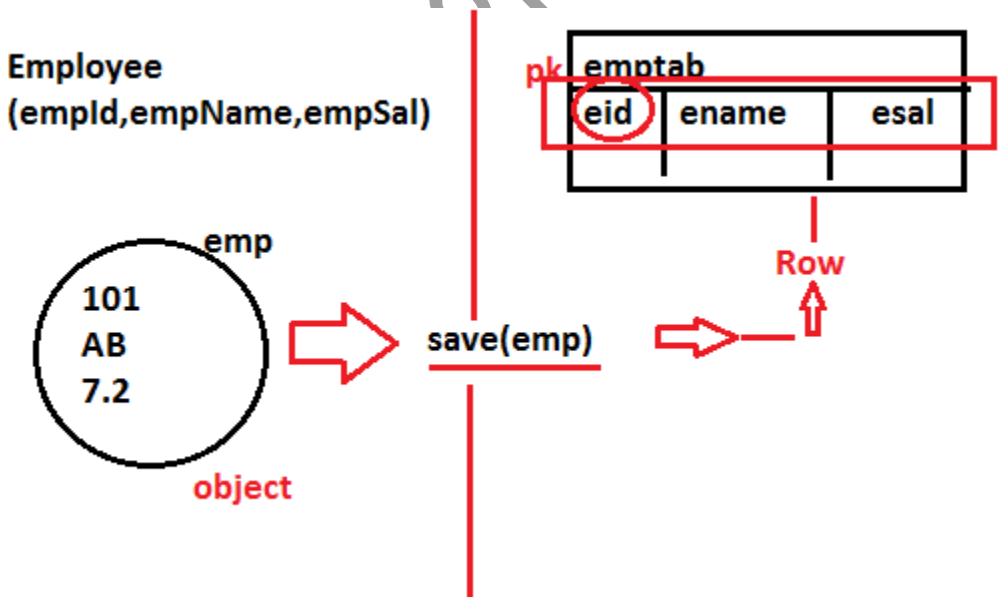
Select operations are:

1) get(T.class, Id): T object (T=Model className)
2) load(T.class, Id): T object (T=Model className)

1)save(obj):serializable

This method is used to object to row format. This object should be model class (Persistable class) object.

**When programmer write save() hibernate generates SQL looks as below: insert into <table> value(..). Example:



****save (obj)** may throw Exception if same PrimaryKey object (with different/same data) is given again.I.e**. ConstraintVoilationException:uniqueConstraintViolated**.

The **Class having table in Database** is called "**Persistable Class**".

Test Class:

```
package com.sathyatech.test;


import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;


import com.sathyatech.model.Employee;


public class BasicEmployeeTest {

	public static void main(String[] args) {

		// step 1: load the configuration file
		Configuration cfg = new Configuration();
		cfg.configure("com/sathyatech/config/hibernate.cfg.xml");


		// step 2: create a SessionFactory obj
		SessionFactory sf = cfg.buildSessionFactory();
```

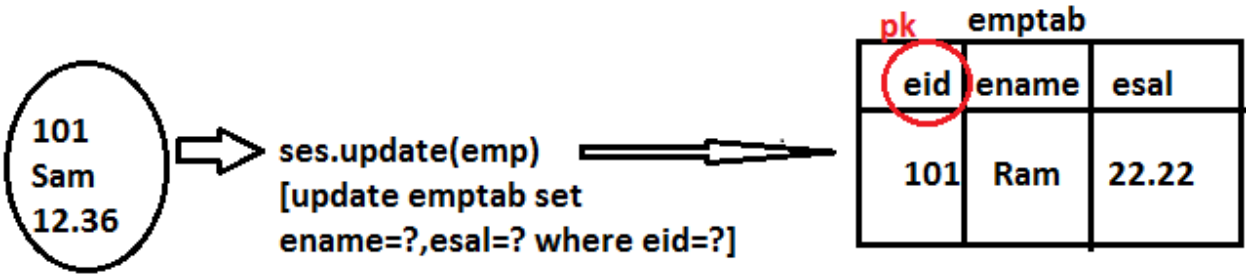
```
// step 3: create a Session obj
Session ses = sf.openSession();

// step 4:start a Transaction
Transaction tx = ses.beginTransaction();

// Creating object
Employee emp = new Employee();
emp.setEmpId(123);
emp.setEmpName("Sam");
emp.setEmpSal(12.36);

// step 5: execute operation
ses.save(emp);

// step 6: commit transaction
 tx.commit();

// step 7: close session
 ses.close();

// step 8: close SessionFactory
```

```
            sf.close();
        }
}
```

2)Update (obj): void: -

This method is used to update all columns values based on primary key column value.

**If Record(ROW) found based on primary key it update else does something (no exception/no error).




```
// Creating object
 Employee emp = new Employee();
emp.setEmpId(123);
emp.setEmpName("Ram");
emp.setEmpSal(44.22);

// step 5: execute operation
ses.update(emp);
```

Batch Test Example: - Test Class

```
// Creating object
            Employee emp = new Employee();
            emp.setEmpId(124);
            emp.setEmpName("Ram");
```

```
emp.setEmpSal(44.22);


Employee emp1 = new Employee();
emp1.setEmpId(125);
emp1.setEmpName("Sham");
emp1.setEmpSal(41.41);


Employee emp2 = new Employee();
emp2.setEmpId(126);
emp2.setEmpName("Nam");
emp2.setEmpSal(45.45);


// step 5: execute operation
ses.save(emp);
ses.save(emp1);
ses.save(emp2);
```


format_sql (boolean): -

It is a boolean property, default value is false. If it is true it will be display SQL query in clause by clause (part by part).

To enable this add below key in hibernate.cfg.xml.

<property name="hibernate.format_sql">true</property>

It works if show_sql is "true",else above line has meaningless.

cfg.Configure(): -

This method will search for "hibernate.cfg.xml" in "src" only.

If location is different (ex: com/app/hibernate.cfg.xml) or name is different

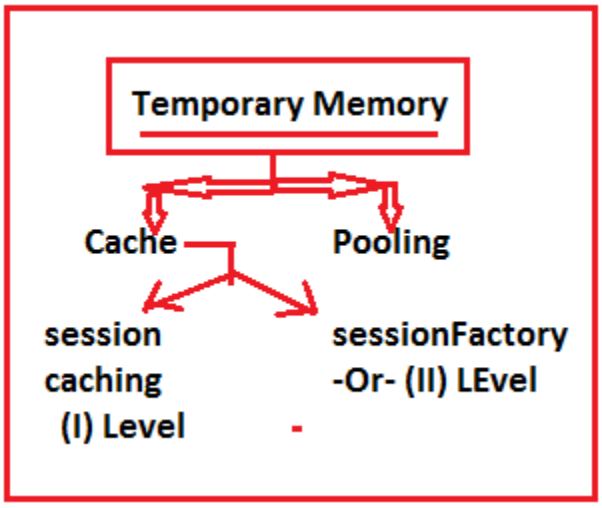
(ex: sample.cfg.xml) or both different (ex: com/app/sample.cfg.xml) then use **cfg.configure(String)** (overloaded) method.

```
// step 1: load the configuration file
 Configuration cfg = new Configuration();
cfg.configure("com/sathyatech/config/hibernate.cfg.xml");
//cfg.configure("sample.cfg.xml");
//cfg.configure("com/app/sample.cfg.xml");
```


Temoporary Memory In Hibernate: -

Hibernate supports 2 types of memories. Those are

1. Cache
2. Pool

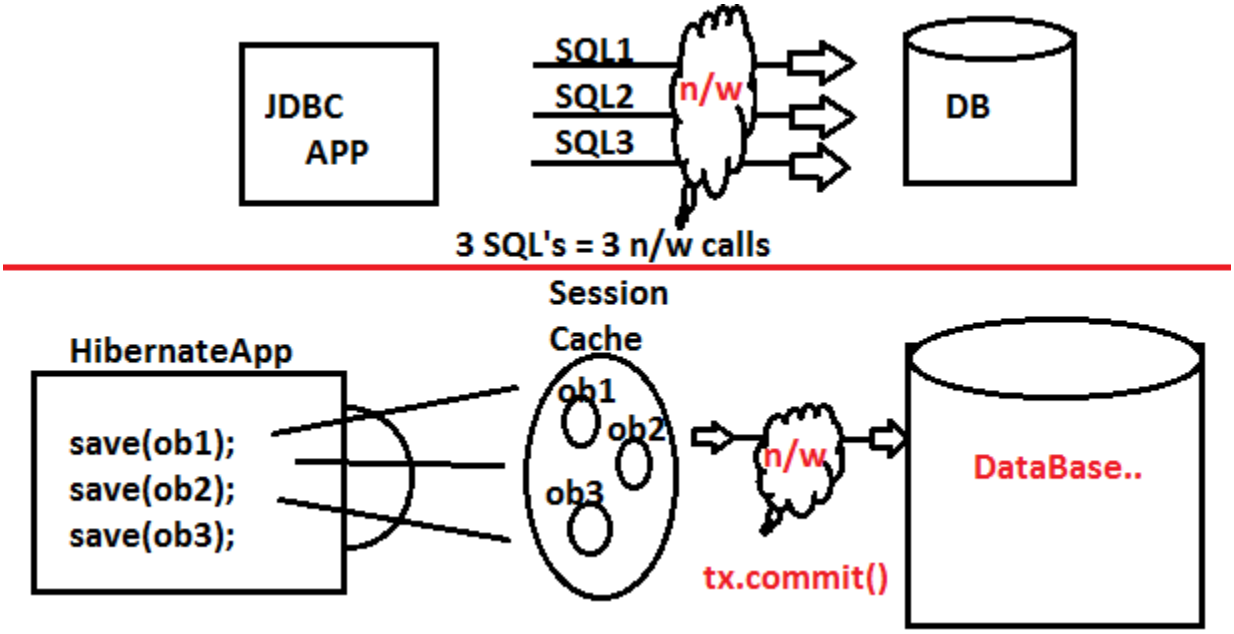


Both are used to improve the performance of application.

1. **Cache: -**It is used in Hibernate to reduce network calls between Hibernate app and Database. Hence that saves time and improve the performance.
   Here, 1 commit() = 1 Network Call
   Cache's are 2 types in Hibernate those are
   A) Session (I Level) Cache
   B) SessionFactory (II Level) Cache

A) **Session (I Level) Cache: -** This memory is handled by Hibernate only. Programmer cannot control this memory. On performing any session operation that goes to Session cache only on commit() operation, data will be send to Database.
It executes all or none.
By default every operation is Batch in Hibernate.
If one operation is failed in Batch then all are **cancelled**.

Example Design in Hibernate: -



Example Code: - Batch Test

```
// Creating object
Employee emp = new Employee();
emp.setEmpId(124);
emp.setEmpName("Ram");
emp.setEmpSal(44.22);

Employee emp1 = new Employee();
emp1.setEmpId(125);
emp1.setEmpName("Sham");
emp1.setEmpSal(41.41);

Employee emp2 = new Employee();
emp2.setEmpId(126);
emp2.setEmpName("Nam");
emp2.setEmpSal(45.45);

// step 5: execute operation
ses.save(emp);
ses.save(emp1);
ses.save(emp2);
```

```
// step 6: commit transaction
tx.commit();
```

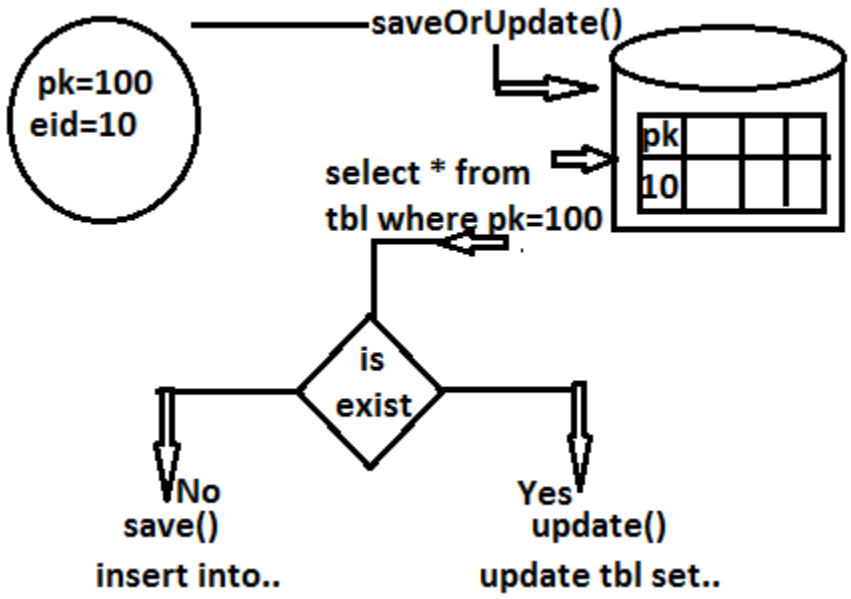
saveOrUpdate(): -

This operation executes select query first based on primary key. If row exist then it executes update query else insert query.

Example Code: -

```
// Creating object
Employee emp = new Employee();
emp.setEmpId(124);
emp.setEmpName("Ram");
emp.setEmpSal(44.22);

// step 5: execute operation
ses.saveOrUpdate(emp);
// step 6: commit transaction
tx.commit();
```

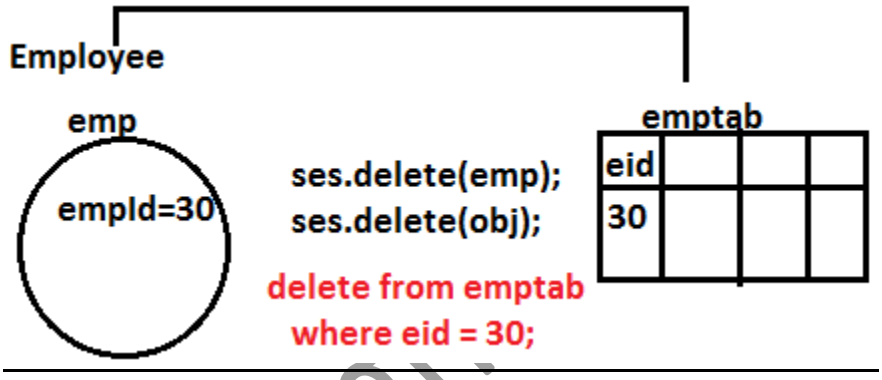
Example Design: -



delete(): -

This method is used to delete one row based on **"Primary key".** But input is **"model class object"** having only **primary variable** value.

Ex Code:
```
// Creating object
Employee emp = new Employee();
emp.setEmpId(124);
emp.setEmpName("Ram");
emp.setEmpSal(44.22);
// step 5: execute operation
ses.delete(emp);
// step 6: commit transaction
tx.commit();
```



selectOperation: -

Data select using get(): -

Session supports selecting one row based on Primary key column get() will select one row and that is converted to one object.

row> get() > Object

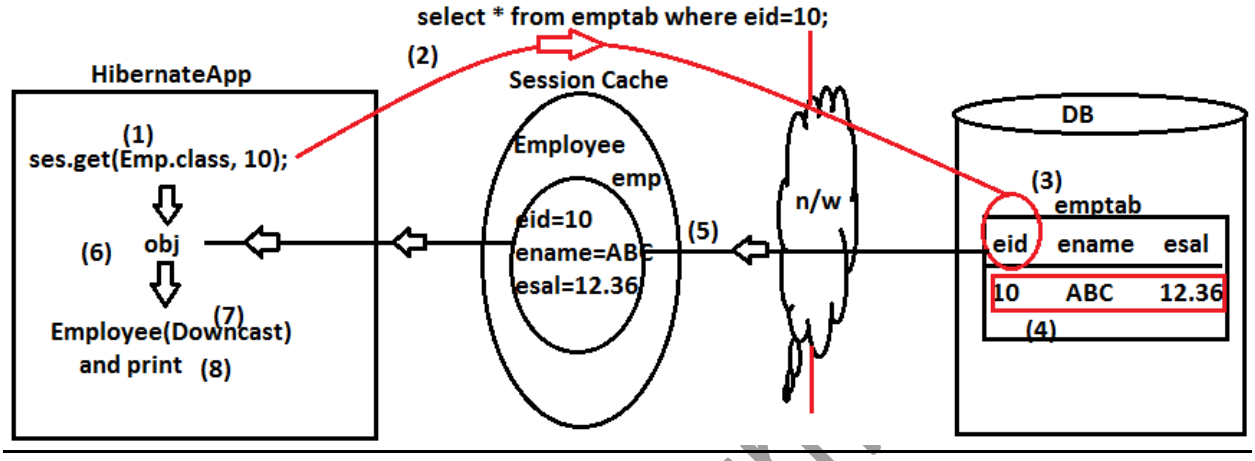
If row is not exist for given primary key then returns **"null"**data.

get() works only based on primary key, not other.

Syntax: **get(T.class, ID): Object**

Here, T = Model class Name, Type ID = Primary Key value. Object = row is converted to java.lang.Object.

get() Execution Flow: -



get() is a select Operation so,**"Tx"** is not required while writing this operation. get() always connected to DB to fetch data every time. It never goes to Cache directly.

Example Code: -

```
// Creating object
Employee emp = new Employee();
emp.setEmpId(124);
emp.setEmpName("Ram");
emp.setEmpSal(44.22);

// step 5: execute operation
Object obj = ses.get(Employee.class,124);
Employee e = (Employee)obj;
System.out.println(e);

Object obj1 = ses.get(Employee.class,300);
Employee e1 = (Employee)obj1;
System.out.println(e1); // NULL
```

Q) casting null data will throw any exception or not?

Ans: If reference has null performing operation like method call, property call returns NullPointerException(NPE).

Ex: Object ob = null;

Ob.toString(); //NPE

But casting (Down cast) never returns exception, null casted value is “null” only.

Ex: Object ob = null

Employee e = (Employee)ob;

Sysout(e); // Output is null only.

Data select using load(): -

This method creates one **proxy (dummy/fake object)** in Session cache and returns same to application. On performing any operation using object after casting then it goes to DB and fetch the data, fills to object variable.

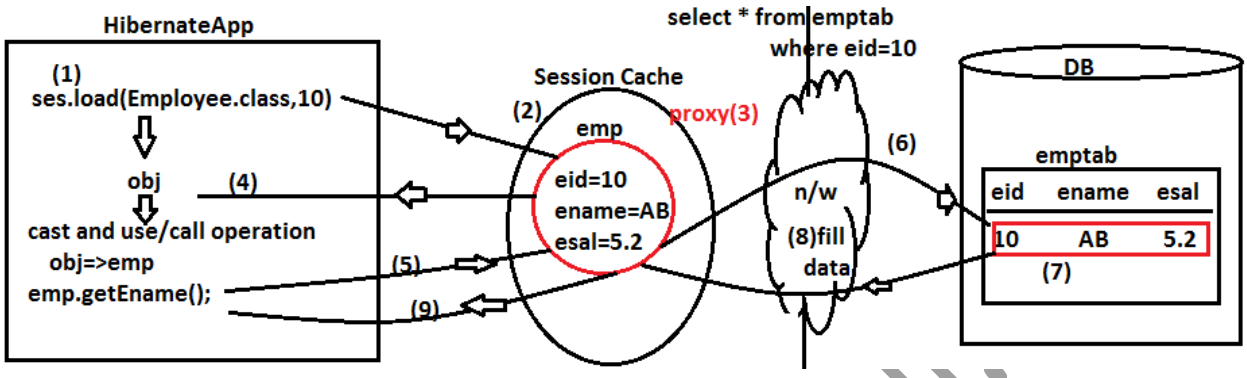
If row is not available in DB then it returns “ObjectNotFoundException”.

Example code: -

```
// Creating object
Employee emp = new Employee();
emp.setEmpId(124);
emp.setEmpName("Ram");
emp.setEmpSal(44.22);

// step 5: execute operation
Object obj = ses.load(Employee.class,124);
Employee e = (Employee)obj;
System.out.println(e);

Object obj1 = ses.load(Employee.class,300);
Employee e1 = (Employee)obj1;
```

System.*out*.println(e1); // Exception ObjectNotFoundException

load() Flow: -

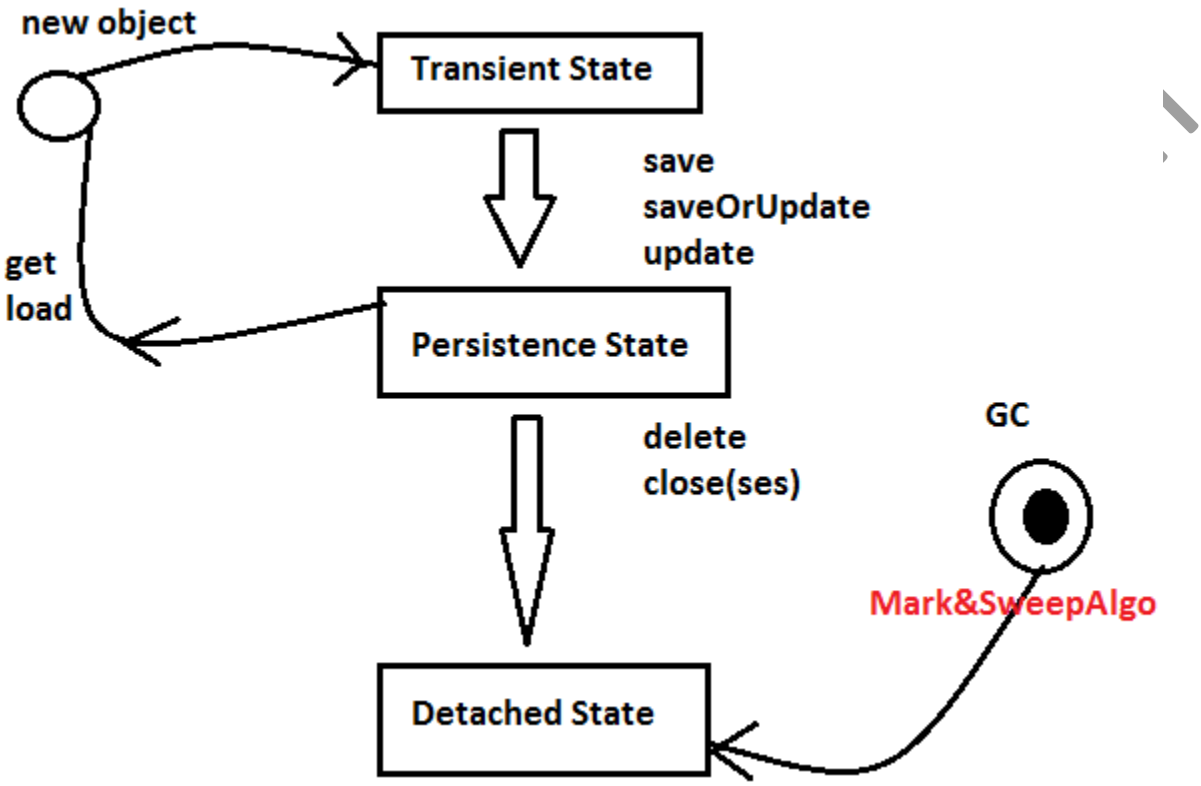


get()	load()
get hits DB always.	load hits cache always.
get returns null id record not exist.	load throws Exception if record not exist then (ObjectNotFoundException).
every time it makes network call.	only on reading data it makes network call.

Object State In Hibernate: -

Hibernate contains objects in different place based on place used our object storage, one state is assigned to that.

Object State are assigned based on operations given below:



Transient State: An object is in heap memory and not connected to any cache then Object State is Transient.

Ex: Employee emp = new Employee(); > Object created in Heap (JVM/RAM) But not in cache.

Persistence State: On Performing Session operations like save(), update(), saveOrupdate(), get(), load() then Object will be placed (Connected) to Cache, then Object is in Persistence State.

Ex: Employee e = ses.get(Emp.class,55);

Detached State: An Object removed from Cache, which is eligible for "garbage collection", is in Detached State.

Ex: ses.delete(emp);

GarbageCollector (GC) follows "**Mark&Sweep**" algorithm.

Annotations Concept#3: Date AND Time: -

In model class for Date (java.util.Date) we can specify DB storage format using Annotation [@Temporal] + Enum [TemporalType]-DATE, TIME, TIMESTAMP.

DATE: - Stores only Date like 11/02/2018

TIME: - Stores only Time like 09:35:25

TIMESTAMP: - Stores Date and Time like 11/02/2018 09:35:59

Model Class:

```
@Temporal(TemporalType.TIMESTAMP)
    @Column(name="dte_a")
    private Date dteA;

    @Temporal(TemporalType.TIME)
    @Column(name="dte_b")
    private Date dteB;

    @Temporal(TemporalType.DATE)
    @Column(name="dte_c")
    private Date dteC;
```

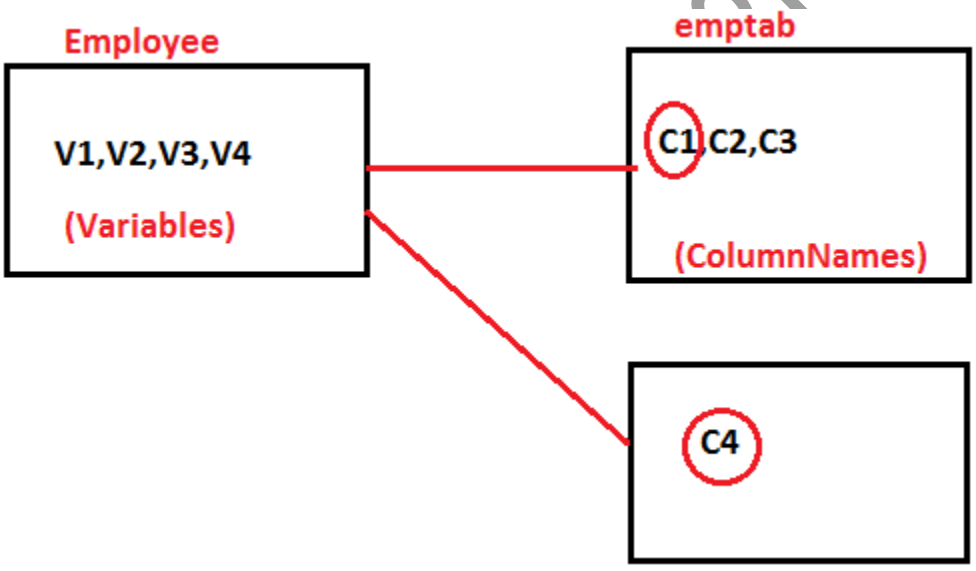
Secondary Table in Hibernate: -

By default one class can be connected to one table only. But Hibernate supports few variables in class can be stored with 2nd table also called as **"Secondary Table".**

Specially fields like sum, total, avg, final grades, export data, logical values are stored in another table instead using main table, for these concept we use Secondary Table.

@Entityand **@Table**provides mapping to **main table**also called as "**PrimaryTable".**

@SecondaryTableis used to provide mapping to 2nd, 3rd, 4th, 5th, …table also called as Secondary Table.



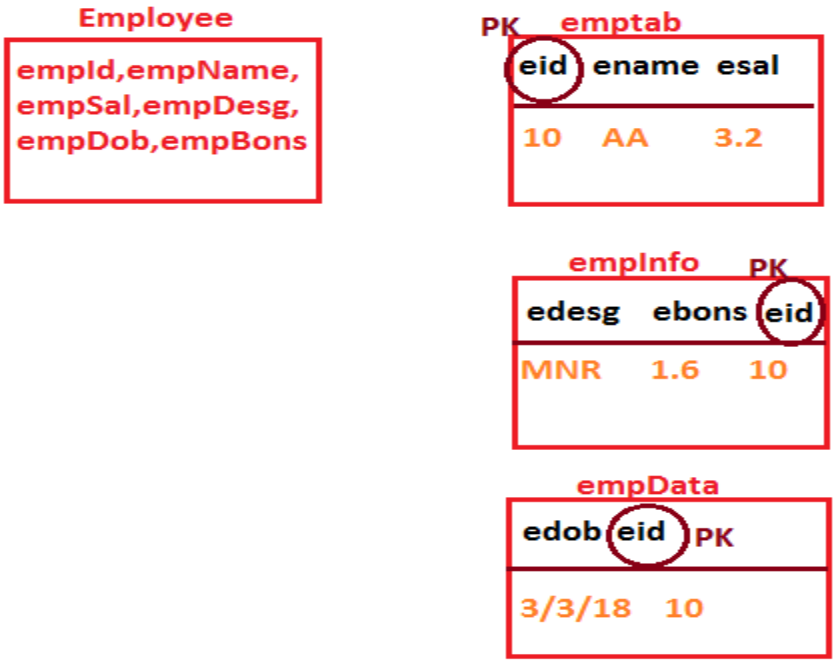
Every table in Hibernate must have **"PK" Column**, Even **SecondaryTable also**.

13. SECONDARY TABLE: Example

```
@Entity
@Table(name="emp_tab")
@SecondaryTables( {
@SecondaryTable(name = "emp_child",
pkJoinColumns=@PrimaryKeyJoinColumn(
```

name="eidFk",referencedColumnName="eid")
})
)

class Employee {
@Id
@Column(name="eid")
int empId;
@Column(name="ename",table="emp_child") -> Required (bydefault maps to PT)
String empName;
@Column(name="esal")
double empSal;
}
pkJoinColumns: - **Optional**…alias Name for Primary Key ColumnName
referencedColumnName: - **Optional**



Model Class:

package com.sathyatech.model;

```
import java.util.Date;


import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.SecondaryTable;
import javax.persistence.SecondaryTables;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;


@Entity
@Table(name="emptab")
@SecondaryTables({
	@SecondaryTable(name="empInfo"),
	@SecondaryTable(name="empData")
})
public class Employee {


	@Id
	@Column(name="emp_Id")
	private int empId;
```

```
@Column(name="emp_Name")
private String empName;

@Column(name="emp_Sal",table="empData")
private double empSal;


@Column(name="emp_Desg",table="empInfo")
private String empDesg;

@Column(name="emp_Bns",table="empData")
private double empBonus;


@Column(name="emp_Dob",table="empInfo")
@Temporal(TemporalType.DATE)
private Date empDab;

public Employee() {
        System.out.println("Employee::0-param constructor");
  }
```

```
public int getEmpId() {
        return empId;
}
public void setEmpId(int empId) {
        this.empId = empId;
}
public String getEmpName() {
        return empName;
}
public void setEmpName(String empName) {
        this.empName = empName;
}
public double getEmpSal() {
        return empSal;
}
public void setEmpSal(double empSal) {
        this.empSal = empSal;
}
public String getEmpDesg() {
        return empDesg;
}
```

```
	public void setEmpDesg(String empDesg) {
		this.empDesg = empDesg;
	}
	public Date getEmpDab() {
		return empDab;
	}
	public void setEmpDab(Date empDab) {
		this.empDab = empDab;
	}
	public double getEmpBonus() {
		return empBonus;
	}
	public void setEmpBonus(double empBonus) {
		this.empBonus = empBonus;
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName +
", empSal=" + empSal + ", empDesg=" + empDesg
				+ ", empDab=" + empDab + "]";
	}
}
```

7.COMPONENT MAPPING:

@Embeddable ----------→**Selecting class**[required]
public class Address{
@Column(name="hno")
private int hno;
@Column(name="loc")
private String loc;
}
@Entity
class Employee {
@Embedded ----------→**This is connecting with Employee (required)**
@AttributeOverrides({ --→**Provides New Column Names (optional)**
@AttributeOverride(name="hno",column=@Column(name="hno")),
@AttributeOverride(name="loc",column=@Column(name="location"))
})
private Address addr=new Address();
}

Mapping **multiple classes** to **one table** if Child class is Fully Dependent on Parent Class then use 2 Annotations: -

For Selection → @Embeddable

For Connection → @Embedded

Example:

Employee.java

```
package com.sathyatech.model;

import javax.persistence.AttributeOverride;

import javax.persistence.AttributeOverrides;

import javax.persistence.Column;
```

```
import javax.persistence.Embedded;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;



@Entity
@Table(name="emptab")
public class Employee {

	@Id
	@Column(name="emp_Id")
	private int empId;

	@Column(name="emp_Name")
	private String empName;

	@Column(name="emp_Sal")
	private double empSal;

	@Embedded
	@AttributeOverrides({
```

```
	@AttributeOverride(name="addrId",column=@Column(name="emp_AddrI
d")),

	@AttributeOverride(name="loc",column=@Column(name="emp_Loc"))
	})
	private Address addr;  //HAS-A

	public int getEmpId() {
		return empId;
	}
	public void setEmpId(int empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public double getEmpSal() {
		return empSal;
	}
```

```
	public void setEmpSal(double empSal) {
		this.empSal = empSal;
	}
	public Address getAddr() {
		return addr;
	}
	public void setAddr(Address addr) {
		this.addr = addr;
	}
	public Employee() {
		 System.out.println("Employee::0-param constructor");
	   }
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName +
", empSal=" + empSal + ", addr=" + addr + "]";
	}
}
```

Address.java

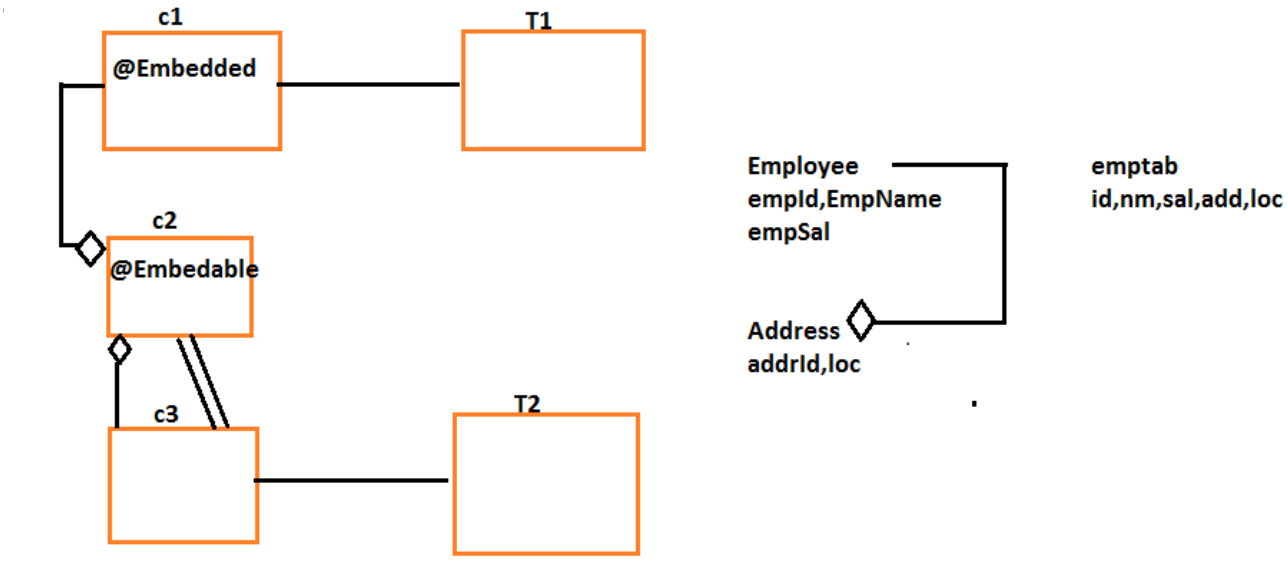
```
package com.sathyatech.model;
import javax.persistence.Column;
import javax.persistence.Embeddable;
```

```
@Embeddable
public class Address {

	@Column(name="aid")
	private int addrId;

	@Column(name="loc")
	private String loc;

	public int getAddrId() {
		return addrId;
	}
	public void setAddrId(int addrId) {
		this.addrId = addrId;
	}
	public String getLoc() {
		return loc;
	}
	public void setLoc(String loc) {
		this.loc = loc;
	}
```

```
	@Override
	public String toString() {
		return "Address [addrId=" + addrId + ", loc=" + loc + "]";
	}
}
```

Test.java

```
// Creating object
		Address a = new Address();
		a.setAddrId(1545);
		a.setLoc("HYD");

		Employee emp = new Employee();
		emp.setEmpId(7777);
		emp.setEmpName("Mahadeva");
		emp.setEmpSal(45.125);
		emp.setAddr(a);

		ses.save(emp);
```



4.BLOB and CLOB:

```
@Lob
private byte[] image;
@Lob
private char[] doc;
```

Here, LOB=Large Object indicates lengthy data to be stored in DB. These are given from Database and supported by Hibernate.

For **BLOB**, representation in Hibernate use **byte[]** with **@Lob** annotation and also for **CLOB**, representation in Hibernate use **char[]** with @Lob annotation.

BLOB → Binary Large OBject.

CLOB → Character Large Object.

@Lob is from javax.persistence package.

Model Class:

@Entity

@Table(name="emptab")

```
public class Employee {


	@Id
	@Column(name="emp_Id")
	private int empId;


	@Lob
	@Column(name="my_Details")
	private char[] myDetails;


	@Lob
	@Column(name="my_Image")
	private byte[] myImage;
```

Test Class:

```
//loading image from File System to byte[]
		File f = new File("C:/Users/Jarvis/Desktop/sexobeat.jpg");
		FileInputStream fis = new FileInputStream(f);
		byte[] arr = new byte[fis.available()];
		fis.read(arr);


		String data = "Hello, How are you!!";
```

```
// Creating object
Employee emp = new Employee();
emp.setEmpId(7174);
emp.setMyDetails(data.toCharArray());
emp.setMyImage(arr);

//save Object
ses.save(emp);
```

****) available():int**

This method is used to calculate size of loaded file into FileInputStream. This size is bytes returned as int byte. Ex: 350 byte (int).

Hibernate Collections Mapping: -

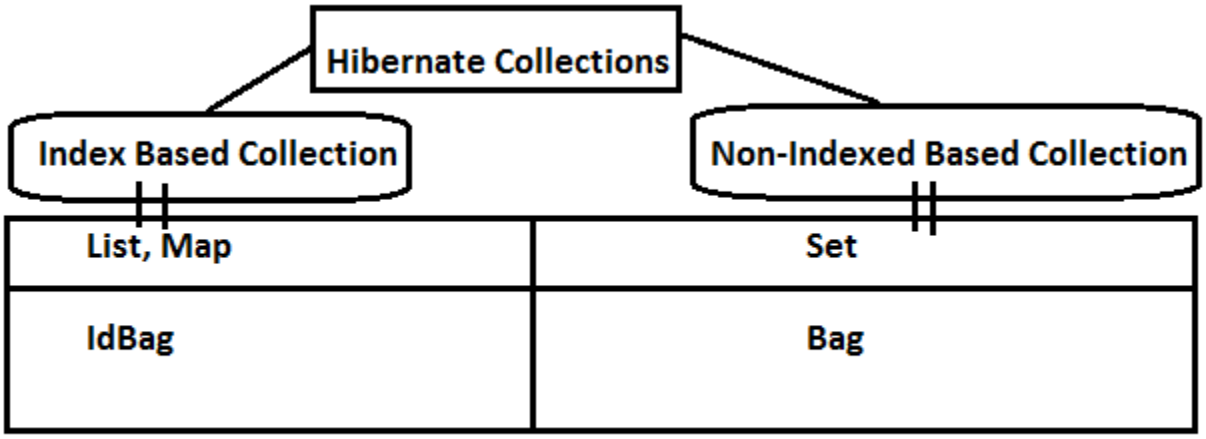
Hibernate supports basic and advanced collections collections mapping with tables. It is provided in 2 ways. Those are

1) Index Based Collections
2) Non-Indexed Based Collections

List, Map and **IdBag** comes under **Index Based** collection.

Set and **Bag** comes under **Non-Indexed Based** Collections.

For Every collection variable one child table will be created that is linked with parent table by using PK-FK columns.



** Bag and IdBag are type "List" Internally design is given from Hibernate F/w

10. BAG AND IDBAG

Bag:

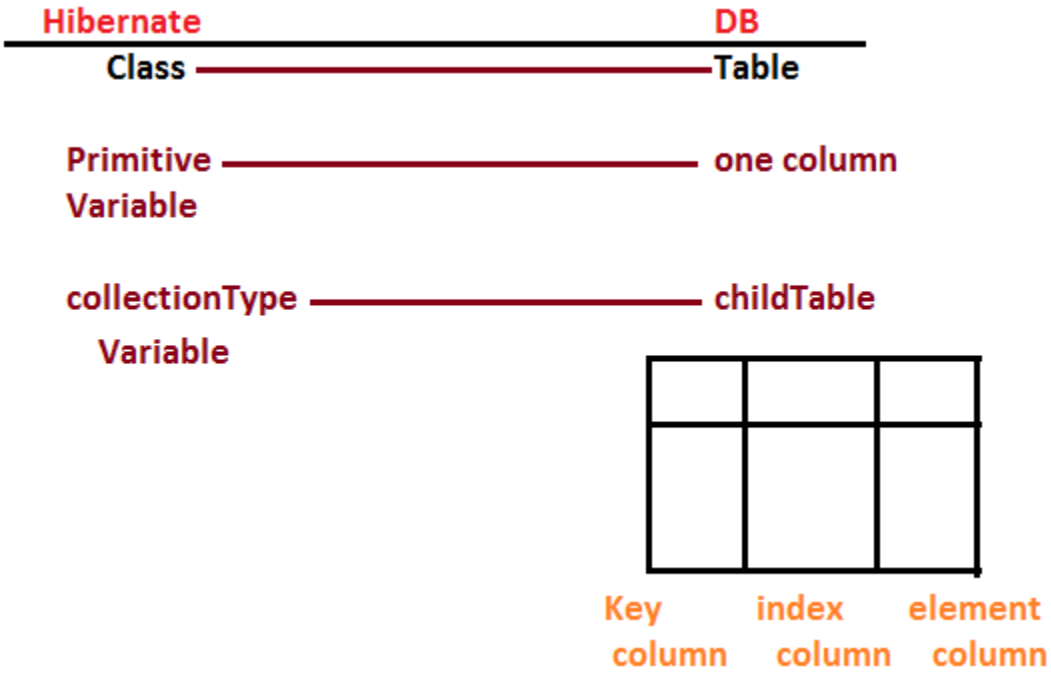
```
@ElementCollection
@CollectionTable(name="emp_data", //table
joinColumns=@JoinColumn(name="eidFk")) //key col
@Column(name="prjs") //element col
private List<String> data=new ArrayList<String>(0);
```

IdBag

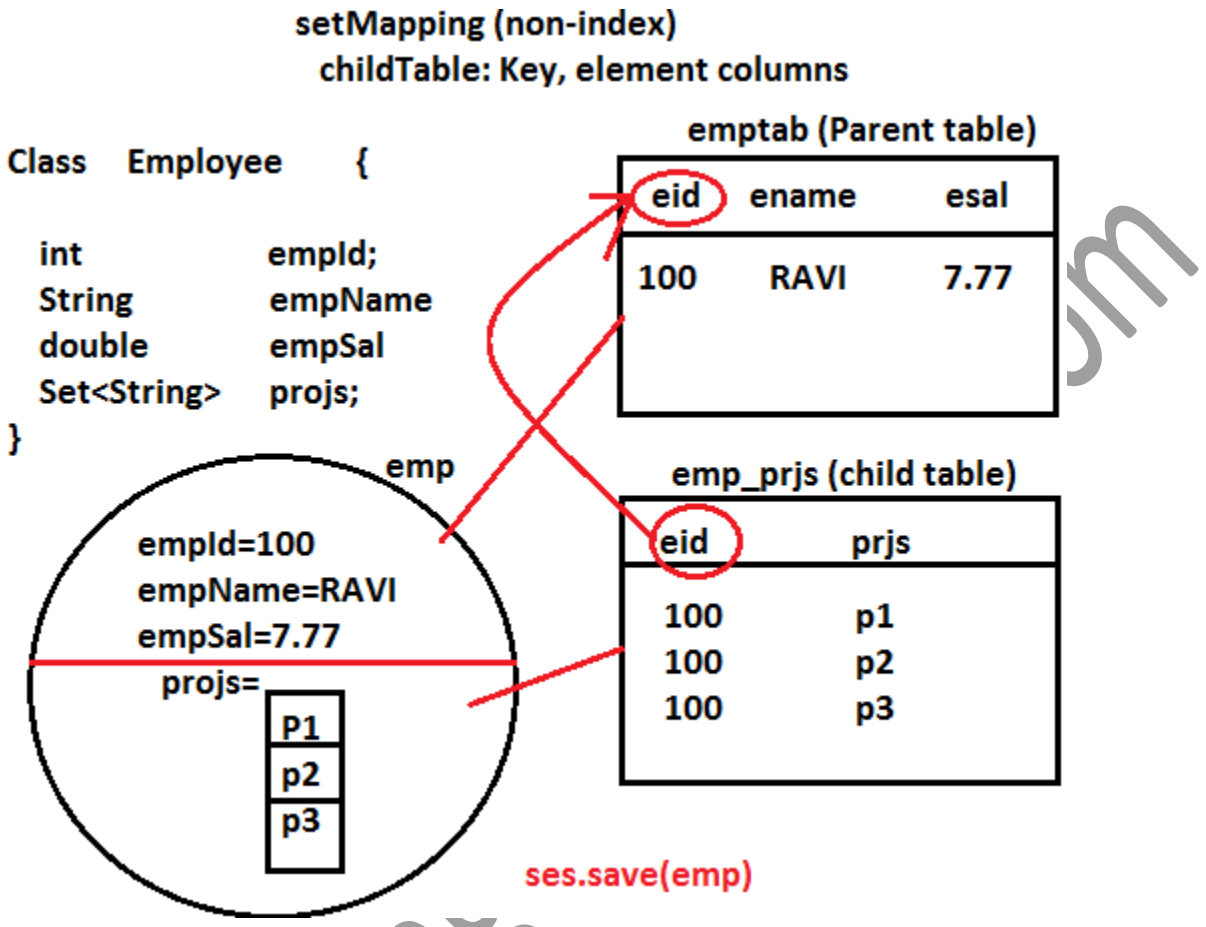
```
@GenericGenerator(name="sample",strategy="increment")
@Entity
@Table(name="emp_tab")
class Employee{
@ElementCollection
@CollectionTable(name="emp_data", //table
joinColumns=@JoinColumn(name="eidFk")) //key col
@CollectionId (columns=@Column(name="unqPos"),
generator = "sample",
type = @Type(type="long"))
@Column(name="prjs") //element col
private List<String> data=new ArrayList<String>(0);
}
```

1. LIST, SET AND MAP WITH PRIMITIVES:

```
@ElementCollection
@CollectionTable(name="emp_dtls", //table
joinColumns=@JoinColumn(name="eidFk")) //key col
@Column(name="lst_data") //element col
private Set<String> details=new HashSet<String>(0);

@ElementCollection
@CollectionTable(name="emp_data", //table
joinColumns=@JoinColumn(name="eidFk")) //key col
@OrderColumn(name="pos") //index col
@Column(name="prjs") //element col
private List<String> data=new ArrayList<String>(0);

@ElementCollection
@CollectionTable(name="emp_models", //table
joinColumns=@JoinColumn(name="eidFk")) //key col
@MapKeyColumn(name="pos") //index col
@Column(name="model_data") //element col
private Map<Integer,String> models=new HashMap<Integer, String>();
```



Example#1: -



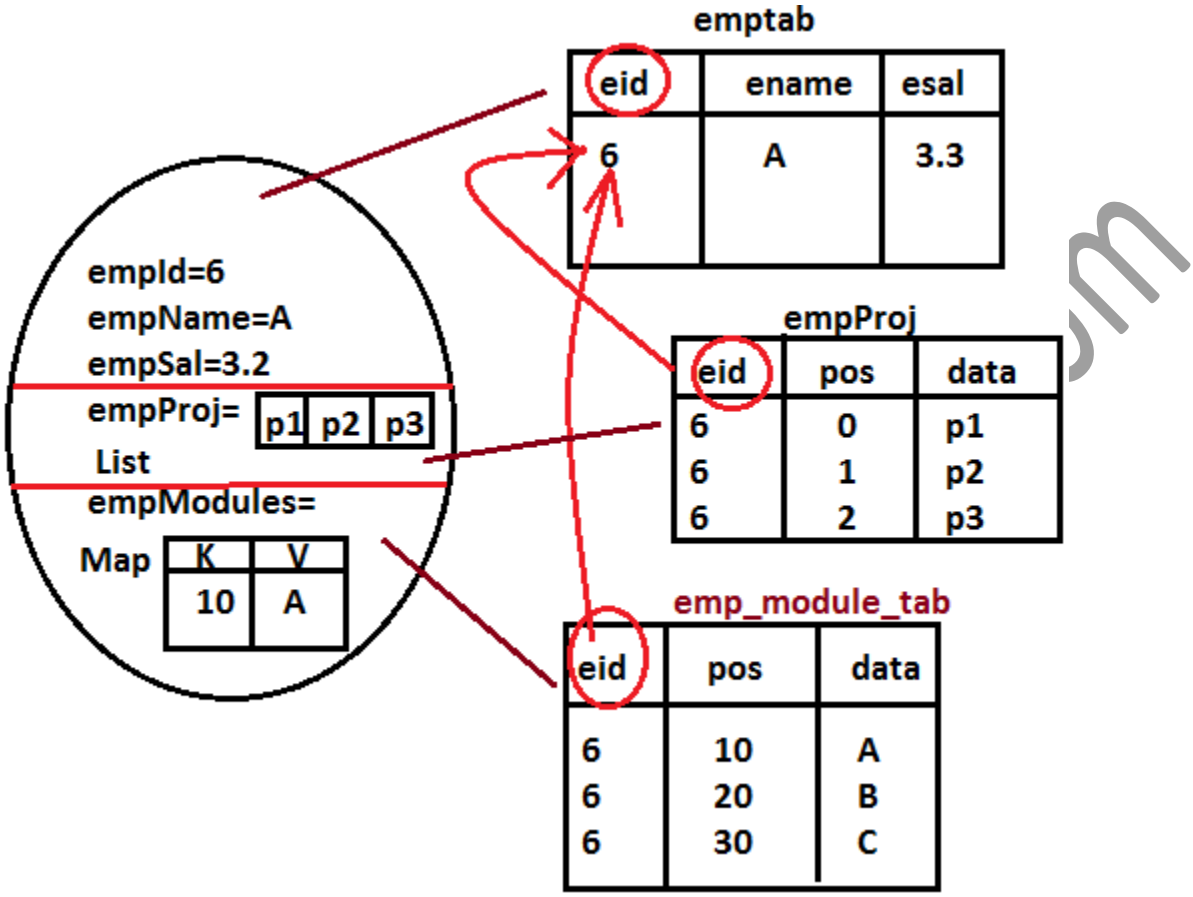
Model Class: Set Ex

```
@ElementCollection // child table-required
        @CollectionTable(   // optional
                name="emp_proj",
                joinColumns=@JoinColumn(name="emp_Id") // key-column - optional
                )
        @Column(name="proj") // element column-optional
        private Set<String> projects;
```

Test.java

```
// CreatingCollection obj
        Set<String> projects = new HashSet<String>();
        projects.add("p1");
        projects.add("p2");
        projects.add("p3");

        // Creating Employee object and adding Collection obj
        Employee emp = new Employee();
        emp.setEmpId(100);
        emp.setEmpName("RAVI");
        emp.setEmpSal(7.77);
        emp.setProjects(projects);

        //save Object
        ses.save(emp);
```

Example#2: -



Model Class: Collection Mapping ->List and Map

```
@ElementCollection // child table-required
        @CollectionTable(  // optional
                name="emp_proj",//table name
        joinColumns=@JoinColumn(name="emp_Id")// key-column
        )
        @OrderColumn(name="pos") //index-column
        @Column(name="data")    // element-column 1
        private List<String> projects;
```

```
@ElementCollection // child table-required
@CollectionTable(   // optional
                name="emp_models", //table name
                joinColumns=@JoinColumn(name="emp_Id") // key-column
                )
@MapKeyColumn(name="pos") //index-column
@Column(name="data")    // element-column
private Map<Integer,String> models;
```

Test.java

```
// Creating List Collection obj
            List<String> projects = new ArrayList<String>();
            projects.add("p1");
            projects.add("p2");
            projects.add("p3");

            //Creating Map Coll Obj
            Map<Integer,String> model = new LinkedHashMap<Integer,String>();
            model.put(10, "A");
            model.put(20, "B");
            model.put(30, "C");
```

```
// Creating Employee object and adding Collection obj
Employee emp = new Employee();
emp.setEmpId(100);
emp.setEmpName("RAVI");
emp.setEmpSal(7.77);
emp.setProjects(projects);
emp.setModels(model);

//save Object
ses.save(emp);
```

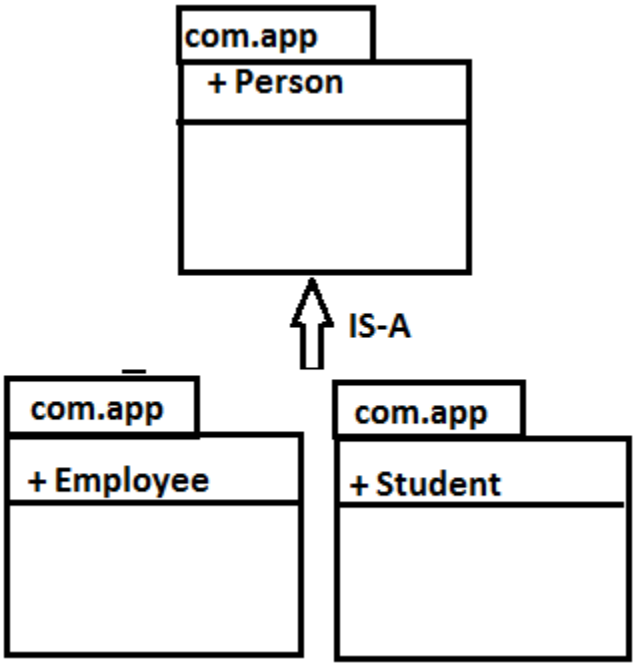
Relational Mappingsin Hibernate Between two models classes we can provide

IS-A Relation and

HAS-A Relation

IS-A Relation between model classes are handled in hibernate using "**Inheritance Mapping".**

HAS-A Relation between model classes are handled in Hibernate using "**Association Mapping".**



IS – A Relation [Inheritance Mapping]: -

It provides 3 designs to store Data in Tables format.

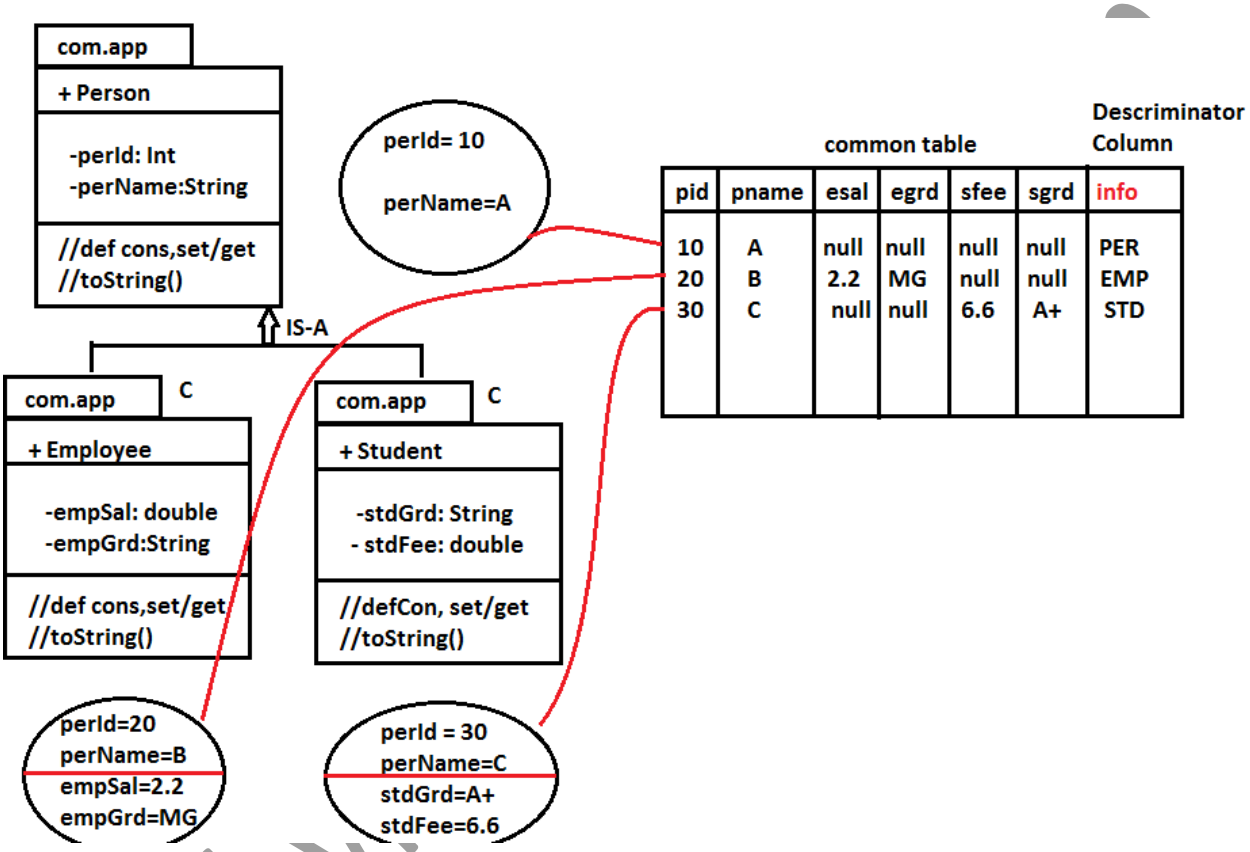
Those are

A. Table Per Class Hierarchy
B. Table Per Sub Class
C. Table Per Concrete Class

1.Table per class Hierarchy: -

It creates one table for all classes. Every class object will be stored as one row with its info column called as "**Discriminator Column".**

No of columns in table = no of variables in all classes + 1 extra column.



i. TABLE PER CLASS HIERARCHY

```
@Entity
@Table(name="empt_tab")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="ob_type",discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue("EMP")
class Employee{
@Id
@Column(name="eid");
```

```
private int empId;
@Column(name="ename");
private String empName;
}
@DiscriminatorValue("REG")
class RegEmployee extends Employee {
@Column(name="emp_prj");
private String projId;
@Column(name="emp_bouns");
private double yearlyBouns;
}

@DiscriminatorValue("CNT")
class ContractEmployee extends Employee {
@Column(name="emp_wrk_hrs");
private double workingHrs;
@Column(name="emp_shift_grade");
private String shiftGrade;
```

Persion.java

```
@Entity
@Table(name="common_tab")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="info",discriminatorType=DiscriminatorType.STRIN
G)
@DiscriminatorValue("PER")
public class Person {

    public Person() {
        System.out.println("Person::0-param constructor");
    }

    @Id
    @Column(name="pid")
    private int perId;

    @Column(name="pname")
    private String perName;
```

```
	public int getPerId() {
		return perId;
	}

	public void setPerId(int perId) {
		this.perId = perId;
	}

	public String getPerName() {
		return perName;
	}

	public void setPerName(String perName) {
		this.perName = perName;
	}

	@Override
	public String toString() {
		return "Person [perId=" + perId + ", perName=" + perName + "]";
	}
}
```

Student.java

```
@Entity
@DiscriminatorValue("STD")
public class Student extends Person {

	@Column(name="sfee")
	private double stdFee;

	@Column(name="sgrd")
	private String stdGrd;

	public double getStdFee() {
		return stdFee;
	}
```

```
	public void setStdFee(double stdFee) {
		this.stdFee = stdFee;
	}

	public String getStdGrd() {
		return stdGrd;
	}

	public void setStdGrd(String stdGrd) {
		this.stdGrd = stdGrd;
	}

	@Override
	public String toString() {
		return "Student [stdFee=" + stdFee + ", stdGrd=" + stdGrd + "]";
	}
}
```

Employee.java

```
@Entity
@DiscriminatorValue("EMP")
public class Employee extends Person {

	@Column(name="esal")
	private double empSal;

	@Column(name="egrd")
	private String empGrd;


	public Employee() {
		System.out.println("Employee::0-param constructor");
	  }


	public double getEmpSal() {
		return empSal;
```

```
	}


	public void setEmpSal(double empSal) {
		this.empSal = empSal;
	}


	public String getEmpGrd() {
		return empGrd;
	}


	public void setEmpGrd(String empGrd) {
		this.empGrd = empGrd;
	}


	@Override
	public String toString() {
		return "Employee [empSal=" + empSal + ", empGrd=" + empGrd + "]";
	}
}
```

Test.java

```
// Creating Employee,Student,Person object
		Person per = new Person();
		per.setPerId(10);
		per.setPerName("A");

		Employee emp = new Employee();
		emp.setPerId(20);
		emp.setPerName("B");
		emp.setEmpGrd("CEO");
		emp.setEmpSal(1255.55);

		Student std = new Student();
```

```
std.setPerId(30);
std.setPerName("C");
std.setStdGrd("A+");
std.setStdFee(7.6);

//save Object
ses.save(emp);
ses.save(per);
ses.save(std);
```

ii. TABLE PER SUB CLASS

In this design, hibernate creates one child table for one sub class, which is connected to parent table using Join column (key column) that behaves like Pk-FK.

Note:

1. No of classes = no. of tables with connected columns.
2. Every child table connected to parent table using key column (or PKJoinColumn).
3. If we save one object then parent data inserted into parent table and child data inserted into child table.

```
@Entity
@Table(name="emp")
@Inheritance(strategy=InheritanceType.JOINED)
class Employee{
@Id
@Column(name="eid");
private int empId;
@Column(name="ename");
private String empName;
}

@Entity
@Table(name="reg_emp")
```

```
@PrimaryKeyJoinColumn(name="eidFk")
class RegEmployee extends Employee {
@Column(name="emp_prj");
private String projId;
@Column(name="emp_bouns");
private double yearlyBouns;
}

@Entity
@Table(name="cnt_emp")
@PrimaryKeyJoinColumn(name="eidFk")
class ContractEmployee extends Employee {
@Column(name="emp_wrk_hrs");
private double workingHrs;
@Column(name="emp_shift_grade");
private String shiftGrade;
}
```

Person.java

```
@Entity

@Table(name="per_tab")

@Inheritance(strategy=InheritanceType.JOINED)

public class Person {


    public Person() {

        System.out.println("Person::0-param constructor");

    }


    @Id
```

```
@Column(name="pid")
private int perId;

@Column(name="pname")
private String perName;

public int getPerId() {
	return perId;
}

public void setPerId(int perId) {
	this.perId = perId;
}

public String getPerName() {
	return perName;
}

public void setPerName(String perName) {
	this.perName = perName;
}
```

```
	@Override
	public String toString() {
		return "Person [perId=" + perId + ", perName=" + perName + "]";
	}
}
```

Employee.java

```
@Entity
@Table(name="emptab")
@PrimaryKeyJoinColumn(name="pidFK")
public class Employee extends Person {

	@Column(name="esal")
	private double empSal;

	@Column(name="egrd")
	private String empGrd;

	public Employee() {
		System.out.println("Employee::0-param constructor");
	  }
```

```
public double getEmpSal() {
        return empSal;
}



public void setEmpSal(double empSal) {
        this.empSal = empSal;
}



public String getEmpGrd() {
        return empGrd;
}



public void setEmpGrd(String empGrd) {
        this.empGrd = empGrd;
}



@Override
```

```
	public String toString() {
		return "Employee [empSal=" + empSal + ", empGrd=" + empGrd + "]";
	}
}
```

Student.java

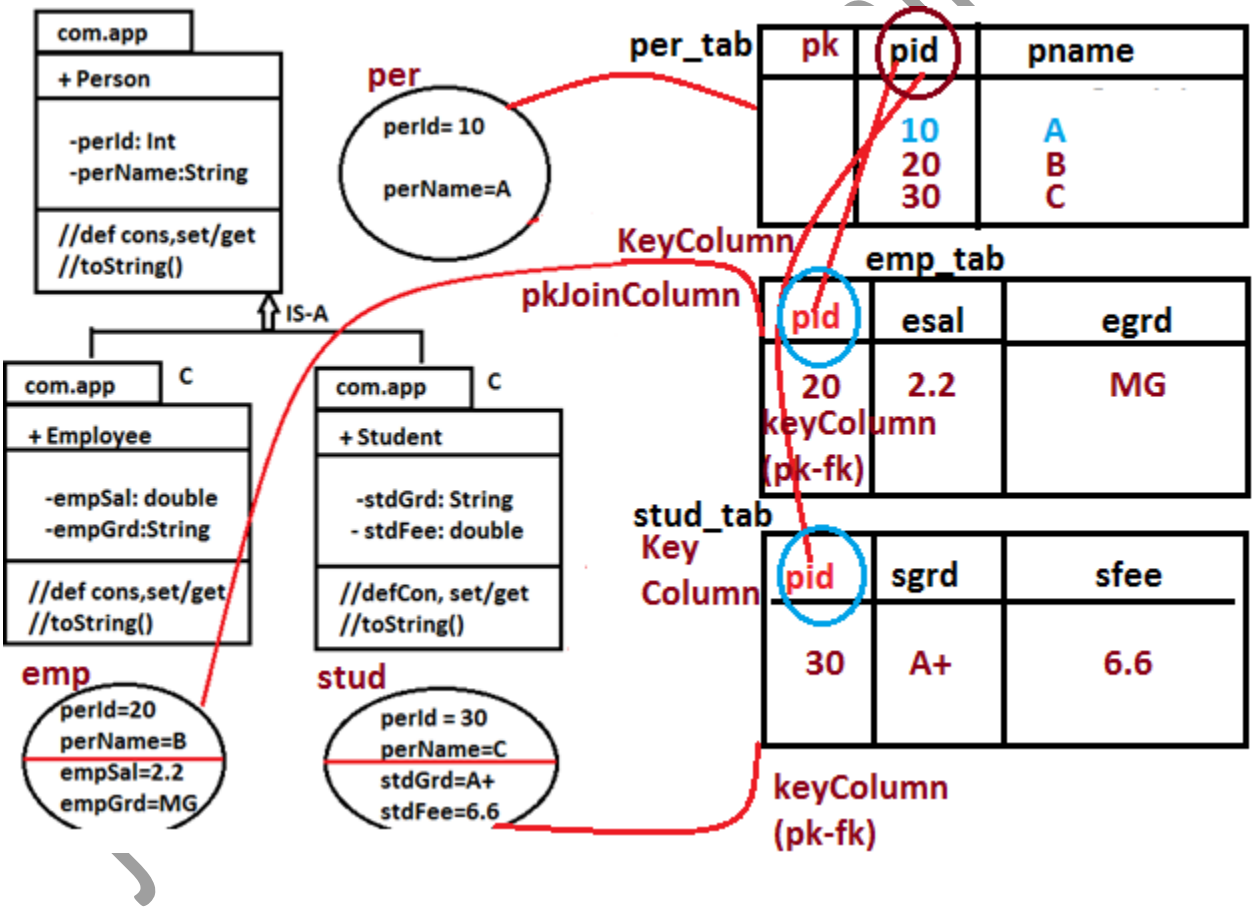
```
@Entity
@Table(name="studtab")
@PrimaryKeyJoinColumn(name="pidFK")
public class Student extends Person {

	@Column(name="sfee")
	private double stdFee;

	@Column(name="sgrd")
	private String stdGrd;

	public double getStdFee() {
		return stdFee;
	}

	public void setStdFee(double stdFee) {
		this.stdFee = stdFee;
```

```
	}


	public String getStdGrd() {
		return stdGrd;
	}


	public void setStdGrd(String stdGrd) {
		this.stdGrd = stdGrd;
	}


	@Override
	public String toString() {
		return "Student [stdFee=" + stdFee + ", stdGrd=" + stdGrd + "]";
	}
}
```

Test.java

```
// Creating Employee,Student,Person object
		Person per = new Person();
		per.setPerId(10);
		per.setPerName("A");

		Employee emp = new Employee();
		emp.setPerId(20);
		emp.setPerName("B");
		emp.setEmpGrd("CEO");
		emp.setEmpSal(1255.55);
```

```
Student std = new Student();
std.setPerId(30);
std.setPerName("C");
std.setStdGrd("A+");
std.setStdFee(7.6);

//save Object
ses.save(emp);
ses.save(per);
ses.save(std);
```

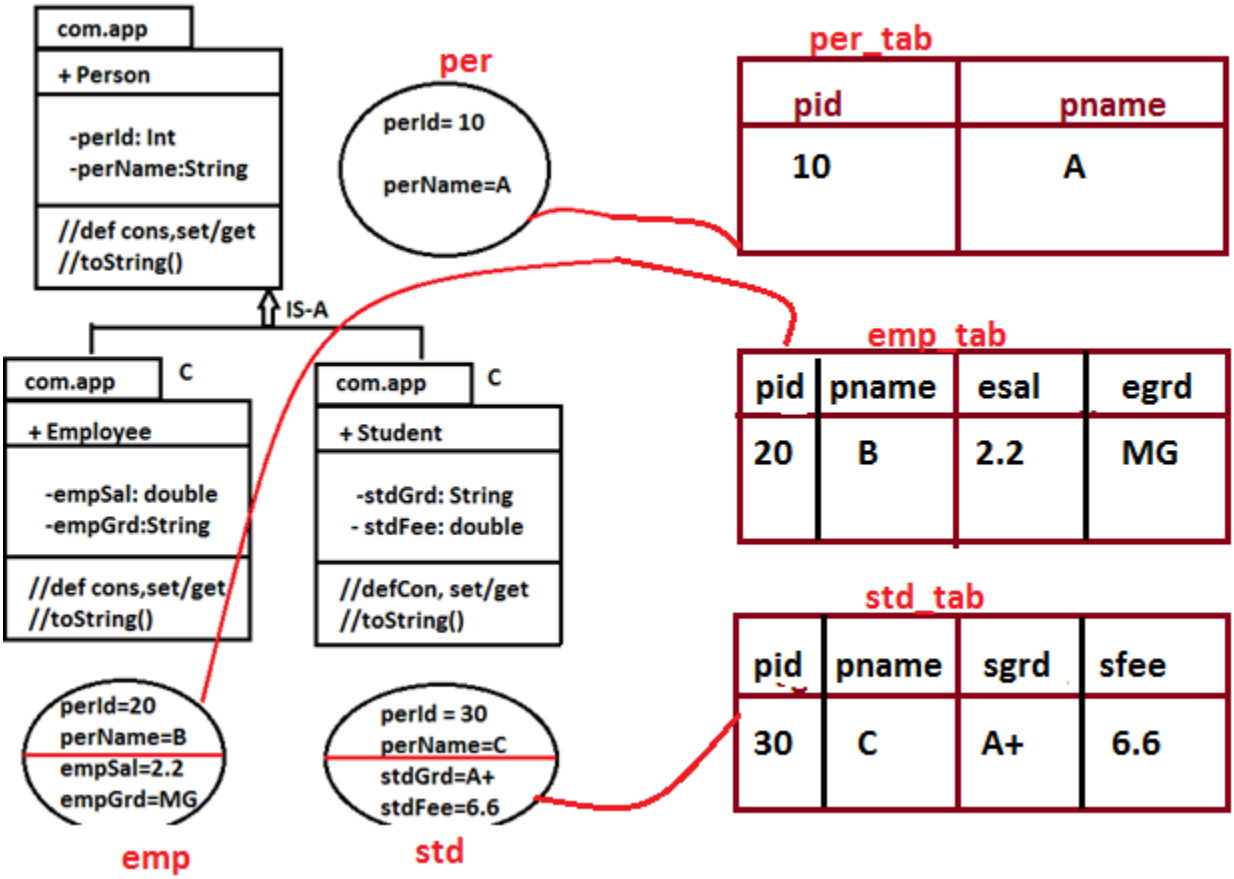


iii. TABLE PER CONCRETE CLASS

This design creates

1. No. of classes = no. of tables

2. All are independent tables, no connection like discriminator column or KeyColumn (no extra columns between tables).
3. Parent Object stored in parent table child object stored in child table.




```
@Entity
@Table(name="emp")
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
class Employee{
@Id
@Column(name="eid");
private int empId;
@Column(name="ename");
private String empName;
}

@Entity
@Table(name="reg_emp")
class RegEmployee exntends Employee {
```

```
@Column(name="emp_prj");
private String projId;
@Column(name="emp_bouns");
private double yearlyBouns;
}

@Entity
@Table(name="cnt_emp")
class ContractEmployee extends Employee {
@Column(name="emp_wrk_hrs");
private double workingHrs;
@Column(name="emp_shift_grade");
private String shiftGrade;
}
```

Person.java

```
@Entity

@Table(name="per_tab")

@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)

public class Person {


    public Person() {

        System.out.println("Person::0-param constructor");

    }


    @Id

    @Column(name="pid")
```

```
private int perId;


@Column(name="pname")
private String perName;


public int getPerId() {
	return perId;
}


public void setPerId(int perId) {
	this.perId = perId;
}


public String getPerName() {
	return perName;
}


public void setPerName(String perName) {
	this.perName = perName;
}


@Override
```

```
	public String toString() {
		return "Person [perId=" + perId + ", perName=" + perName + "]";
	}
}
```

Student.java

```
@Entity
@Table(name="studtab")
public class Student extends Person {

	@Column(name="sfee")
	private double stdFee;

	@Column(name="sgrd")
	private String stdGrd;

	public double getStdFee() {
		return stdFee;
	}

	public void setStdFee(double stdFee) {
		this.stdFee = stdFee;
	}
```

```
	public String getStdGrd() {
		return stdGrd;
	}

	public void setStdGrd(String stdGrd) {
		this.stdGrd = stdGrd;
	}

	@Override
	public String toString() {
		return "Student [stdFee=" + stdFee + ", stdGrd=" + stdGrd + "]";
	}
}
```

Employee.java

```
@Entity
@Table(name="emptab")
public class Employee extends Person {

	@Column(name="esal")
	private double empSal;
```

```
@Column(name="egrd")
private String empGrd;



public Employee() {
        System.out.println("Employee::0-param constructor");
   }



public double getEmpSal() {
        return empSal;
}



public void setEmpSal(double empSal) {
        this.empSal = empSal;
}



public String getEmpGrd() {
        return empGrd;
}
```

```
	public void setEmpGrd(String empGrd) {
		this.empGrd = empGrd;
	}



	@Override
	public String toString() {
		return "Employee [empSal=" + empSal + ", empGrd=" + empGrd + "]";
	}
}
```

Test.java

```
// Creating Employee,Student,Person object
		Person per = new Person();
		per.setPerId(10);
		per.setPerName("A");

		Employee emp = new Employee();
		emp.setPerId(20);
		emp.setPerName("B");
		emp.setEmpGrd("CEO");
```

```
emp.setEmpSal(1255.55);


Student std = new Student();
std.setPerId(30);
std.setPerName("C");
std.setStdGrd("A+");
std.setStdFee(7.6);


//save Object
ses.save(emp);
ses.save(per);
ses.save(std);
```

Multiplicity: - [Association Mapping]

Rows in one table (parent table) will be connected to Rows in another table (child table).

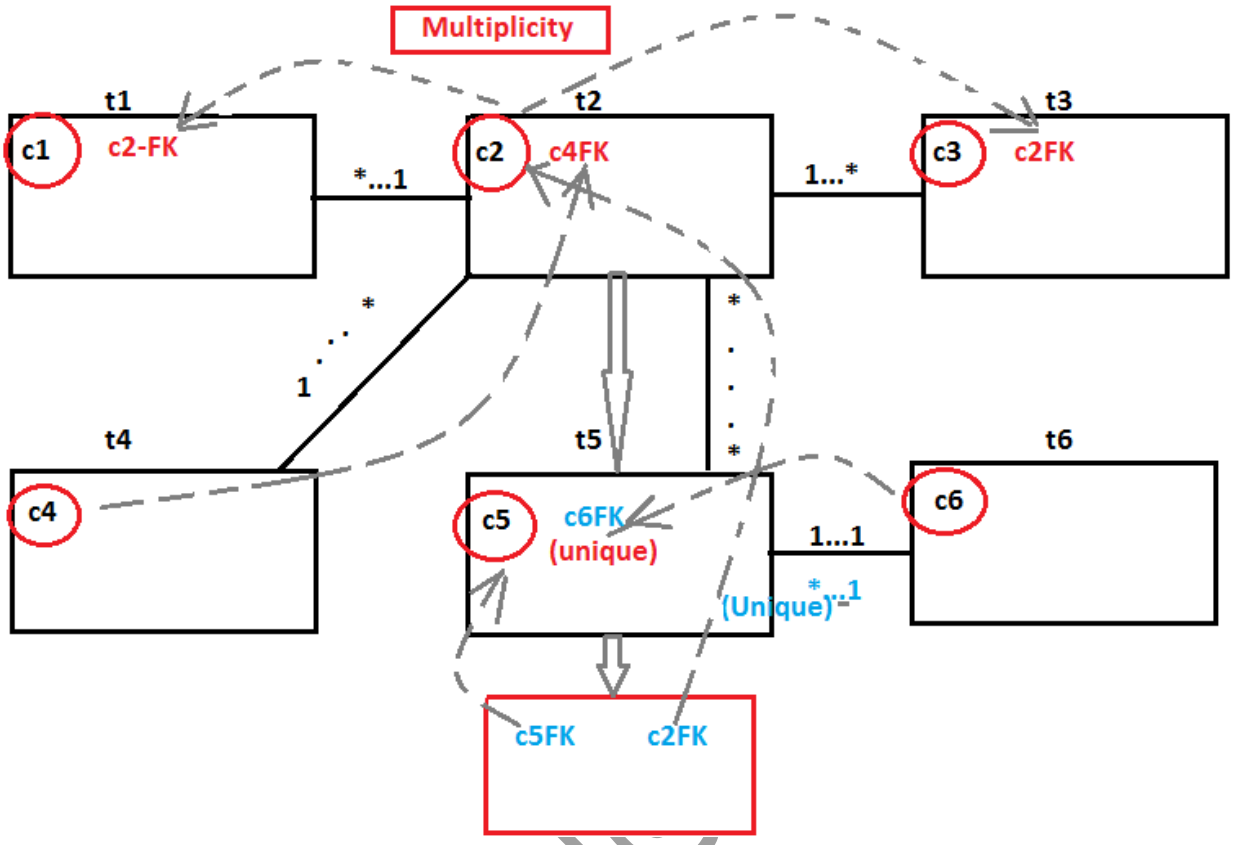
Database provided 4 Multiplicities,those are: -

1. One-to-one (1…1)
2. One-to-many (1…*)
3. Many-to-one (*…1)
4. Many-to-many (*…*)

In case of Multiplicity, one table PK will be given to another table FK.

Based on Multiplicity PK-FK columns are given to table.

- *(star) many side extra column (FK Column) will be created.

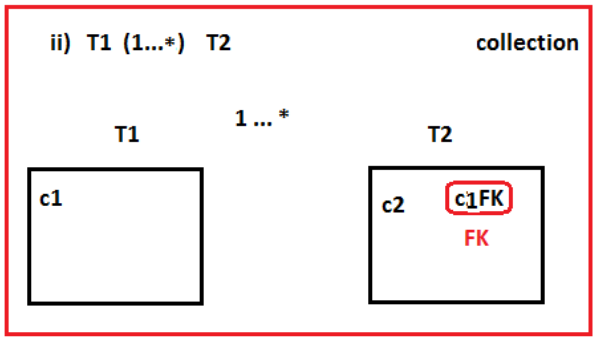


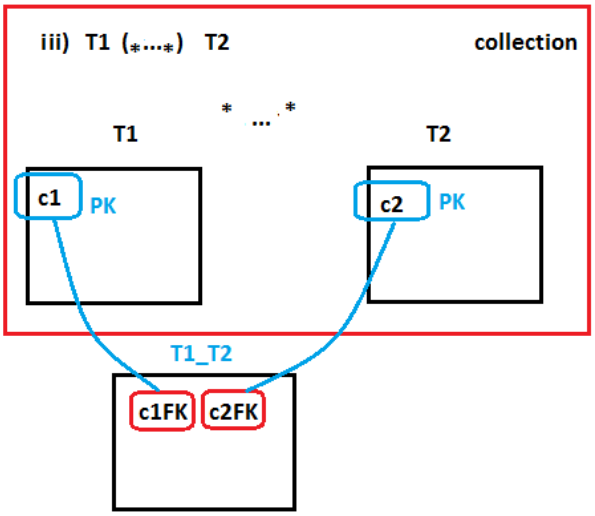
- Hibernate divided these multiplicities into 2 types.
  1. Non-Collection Based
  2. Collection Based

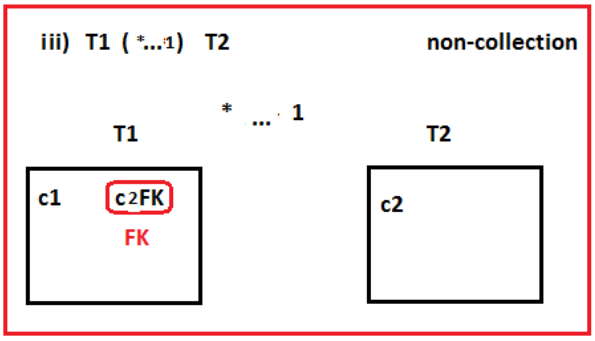
Non-Collection	Collection
1...1	1...*
*...1	*...*

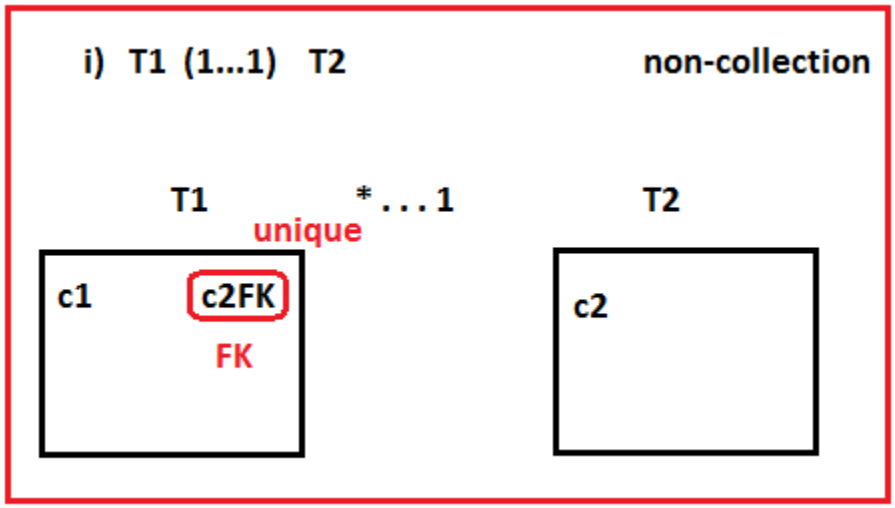
- In case of **1...1** use ***(unique) ...1** and
- For ***...*3rd table** will be created with **2FK Columns**.

Design of one-to-one, one-to-many, many-to-many, many-to-one: -









9. ASSOCIATION MAPPING In Hibernate (HAS-A Relational Mapping):

To handle multiplicity in Hibernate, we choose ASSOCIATION MAPPING concept. It supports easy configuration with Annotation, like @OneToMany, @ManyToMany, @ManyToMany etc.

To get Multiplicity coding steps are: -

1. Create Two Model Class
2. Apply HAS-A Relation between Models
3. Apply multiplicity Annotation.
   **In Case of CollectionType multiplicity we must make HAS-A Type variable one collection.[Set,List,Map]

- **1 . . . 1 & * . . . 1→** Non-Collection **1 . . . * & * . . . *** Collection[Set,List,Map]

Examples: -

1. Employee *. . . 1 ----------<> Dept (Has-A)

```
Class Employee{                          Class Dept{
  @ManyToOne
Dept deptobj;
}                                        }
```

2. Student 1 . . . * ----------<> Subjects (HAS-A)

```
Class Student{                           Class Subjects{

   @OneToMany
   Set<Subjects> sub;
}                                        }
```

3. Employee *…* ----------<>Projects (HAS-A)

```
Class Employee{                          Class Projects{

@ManyToMany
List<Projects> proj;
```

}
}

1 . . . 1

4. Admin----------→Role

HAS-A

Class Admin{
Class Role{

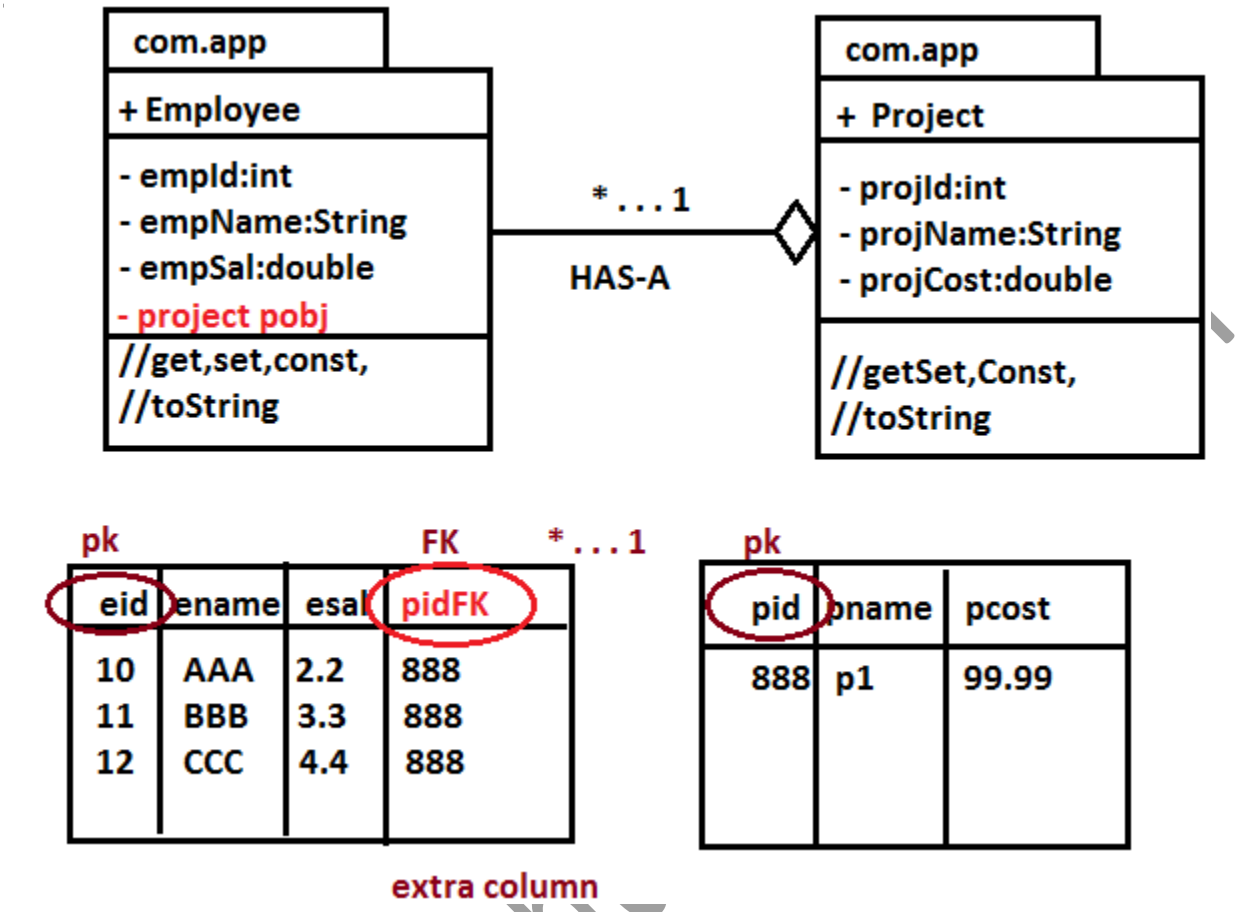
@ManyToOne
Role robj;
}
}

***** Select Alt+Shift+R Change Selected Thing..Ok**
***** Select Alt+Shift+/ For Comment**

Ex#1: ManyToOne (* . . . 1) | Non-Collection

Steps: -

1. UML Design
2. Table Design
3. Coding.



i. Many-To-One and One-To-One (Employee ----<> Address HAS-A)

Box Means : Bi-Directional (optional one)

```
@Entity
@Table(name="addrs_tab")
public class Address {
@Id
@Column(name="aid")
private int addrId;
@Column(name="loc")
private String loc;
@OneToMany(mappedBy="addr")
private List<Employee> emp=new ArrayList<Employee>(0);
}
```

```
@Entity
```

```
@Table(name="empt_tab")
class Employee{
@Id
@Column(name="eid");
private int empId;
@ManyToOne(fetch=FetchType.EAGER,cascade=CascadeType.ALL)
@JoinColumn(name="aidFk",unique=true/false)
private Address addr=new Address();
}
```

Many to One Example:

Employee.java

```
@Entity

@Table(name="emptab")

public class Employee {


        @Id

        @Column(name="eid")

        private int empId;


      @Column(name="ename")

        private String empName;


        @Column(name="eSal")

        private double empSal;
```

```
@ManyToOne
@JoinColumn(name="pidFK") // optional
private Project pObj; // HAS-A

public Employee() {
        System.out.println("Employee::0-param constructor");
   }

public int getEmpId() {
        return empId;
}

public void setEmpId(int empId) {
        this.empId = empId;
}

public String getEmpName() {
        return empName;
}

public void setEmpName(String empName) {
        this.empName = empName;
```

```
	}

	public double getEmpSal() {
		return empSal;
	}

	public void setEmpSal(double empSal) {
		this.empSal = empSal;
	}

	public Project getpObj() {
		return pObj;
	}

	public void setpObj(Project pObj) {
		this.pObj = pObj;
	}

	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName +
", empSal=" + empSal + ", pObj=" + pObj + "]";
```

```
	}
}
```

Project.java

```
@Entity
@Table(name="projtab")
public class Project {

	@Id
	@Column(name="pid")
	private int projId;

	@Column(name="pname")
	private String projName;

	@Column(name="pcost")
	private double projCost;

	public Project() {
		System.out.println("Project::0-param constructor");
	}
```

```
public int getProjId() {
        return projId;
}

public void setProjId(int projId) {
        this.projId = projId;
}

public String getProjName() {
        return projName;
}

public void setProjName(String projName) {
        this.projName = projName;
}

public double getProjCost() {
        return projCost;
}

public void setProjCost(double projCost) {
        this.projCost = projCost;
```

```
	}


	@Override
	public String toString() {
		return "Project [projId=" + projId + ", projName=" + projName + ",
projCost=" + projCost + "]";
	}
}
```

Test.java

```
// Creating Employee,Project object
		Project p = new Project();
		p.setProjId(888);
		p.setProjName("P1");
		p.setProjCost(99.99);

		Employee emp = new Employee();
		emp.setEmpId(10);
		emp.setEmpName("AAA");
		emp.setEmpSal(2.2);
		emp.setpObj(p);

		Employee emp1 = new Employee();
```

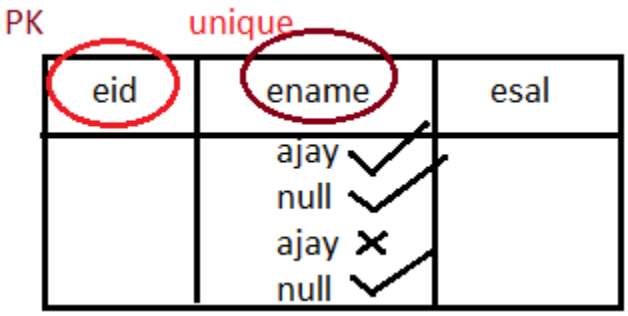
```
emp1.setEmpId(11);
emp1.setEmpName("BBB");
emp1.setEmpSal(3.3);
emp1.setpObj(p);

Employee emp2 = new Employee();
emp2.setEmpId(12);
emp2.setEmpName("CCC");
emp2.setEmpSal(4.4);
emp2.setpObj(p);

//save Object
ses.save(p);
ses.save(emp);
ses.save(emp1);
ses.save(emp2);
```


Unique Condition in Database: -

Unique condition can be applied to any column in Database. This column will accept “one value” only “one time”, i.e. no duplicate data is accepted.

***unique column can have multiple “null” value where null indicates “no data”.

Example:

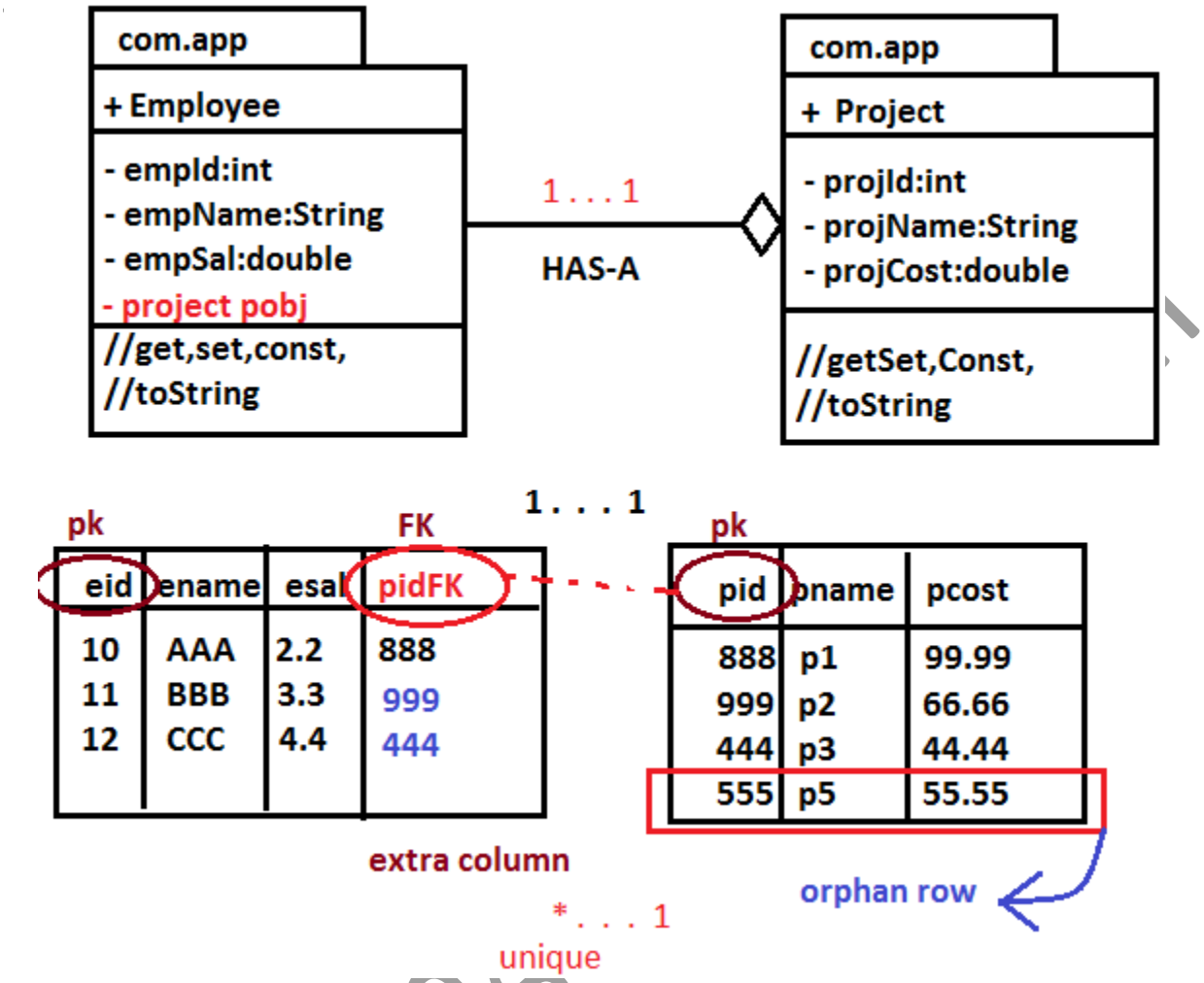


One-to-One (1 . . . 1)| Non-Collection: -

One Row in Parent table will be connected to one Row (at max) in child table.

Example: -

1) UML Design (Model Classes) and
2) Tables Design with example Rows



One To One Example:

Employee.java

```
@Entity
@Table(name="emptab")
public class Employee {

        @Id
        @Column(name="eid")
        private int empId;
```

```
@Column(name="ename")
private String empName;

@Column(name="eSal")
private double empSal;

@ManyToOne
@JoinColumn(name="pidFK",unique=true) // optional
private Project pObj; // HAS-A

public Employee() {
        System.out.println("Employee::0-param constructor");
   }

public int getEmpId() {
      return empId;
}

public void setEmpId(int empId) {
       this.empId = empId;
}
```

```
public String getEmpName() {
        return empName;
}

public void setEmpName(String empName) {
        this.empName = empName;
}

public double getEmpSal() {
        return empSal;
}

public void setEmpSal(double empSal) {
        this.empSal = empSal;
}

public Project getpObj() {
        return pObj;
}

public void setpObj(Project pObj) {
```

```
		this.pObj = pObj;
	}

	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName +
", empSal=" + empSal + ", pObj=" + pObj + "]";
	}
}
```

Project.java

```
@Entity
@Table(name="projtab")
public class Project {

	@Id
	@Column(name="pid")
	private int projId;

	@Column(name="pname")
	private String projName;

	@Column(name="pcost")
```

```
private double projCost;



public Project() {
        System.out.println("Project::0-param constructor");
}

public int getProjId() {
        return projId;
}

public void setProjId(int projId) {
        this.projId = projId;
}

public String getProjName() {
        return projName;
}

public void setProjName(String projName) {
        this.projName = projName;
}
```

```
	public double getProjCost() {
		return projCost;
	}

	public void setProjCost(double projCost) {
		this.projCost = projCost;
	}

	@Override
	public String toString() {
		return "Project [projId=" + projId + ", projName=" + projName + ",
projCost=" + projCost + "]";
	}
}
```

Test.java

```
// Creating Employee,Project object
		Project p1 = new Project();
		p1.setProjId(888);
		p1.setProjName("P1");
		p1.setProjCost(99.99);
```

```
Project p2 = new Project();
p2.setProjId(999);
p2.setProjName("P2");
p2.setProjCost(66.66);

Project p3 = new Project();
p3.setProjId(444);
p3.setProjName("P3");
p3.setProjCost(44.44);

Project p4 = new Project();
p4.setProjId(555);
p4.setProjName("P5");
p4.setProjCost(55.55);

Employee emp = new Employee();
emp.setEmpId(10);
emp.setEmpName("AAA");
emp.setEmpSal(2.2);
emp.setpObj(p1);

Employee emp1 = new Employee();
```

```
emp1.setEmpId(11);
emp1.setEmpName("BBB");
emp1.setEmpSal(3.3);
emp1.setpObj(p2);

Employee emp2 = new Employee();
emp2.setEmpId(12);
emp2.setEmpName("CCC");
emp2.setEmpSal(4.4);
emp2.setpObj(p3);

//save Object
ses.save(p1);
ses.save(p2);
ses.save(p3);
ses.save(p4);

ses.save(emp);
ses.save(emp1);
ses.save(emp2);
```

- **Oracle 10G XE**
  Localhost:8080/apex
- **Oracle 11G or Other**
  Oracle SQL Developer Tool

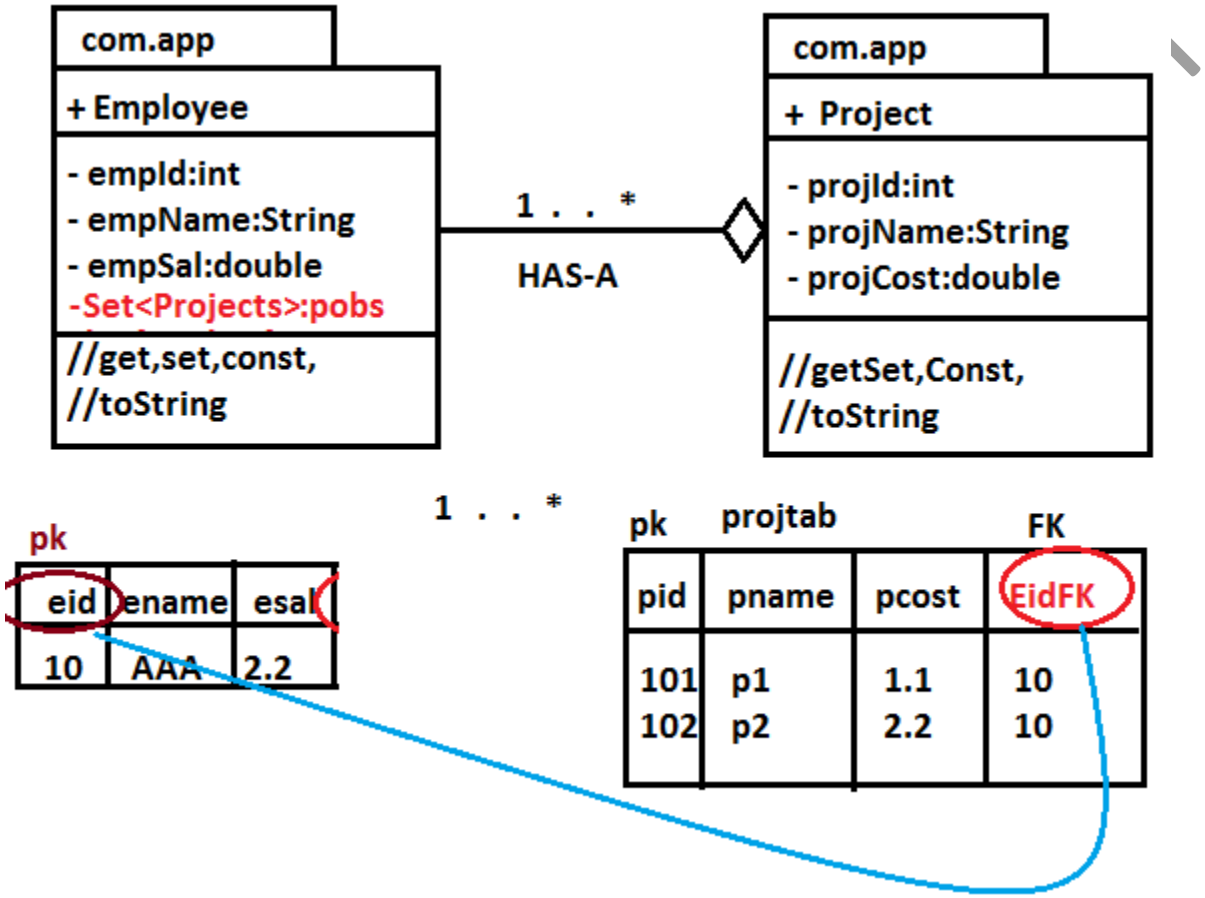
Orphan Row: - A child table row which has no connection with any parent table row is called as "**Orphan Row (Record)".[which is not mapped to any parent table row].**

OneToMany [1 . . *] Collection: - It is a collection based multiplicity. So, we must specify either List or Set type while applying HAS-A Relation between model classes.

1. UML For Models 2.Table Output

```
@Entity
@Table(name="addrs_tab")
public class Address {

@Id
@Column(name="aid")
@GeneratedValue
private int addrId;

@Column(name="loc")
private String loc;
@ManyToOne(mappedBy="addr")
private Employee emp;
}

@Entity
@Table(name="empt_tab")
class Employee{
@Id
```

```
@Column(name="eid");
private int empId;
@OneToMany(cascade=CascadeType.ALL,fetch=FetchType.EAGER)
@JoinColumn(name="eidFk")
private List<Address> addr=new ArrayList<Address>(0);
}
```



One To Many Example:

Employee.java

```
@Entity

@Table(name="emptab")

public class Employee {


        @Id
```

```
@Column(name="eid")
private int empId;

@Column(name="ename")
private String empName;

@Column(name="eSal")
private double empSal;

@OneToMany
@JoinColumn(name="eidFK") // optional
private Set<Project> pObj; // HAS-A

public Employee() {
        System.out.println("Employee::0-param constructor");
   }

public int getEmpId() {
        return empId;
}

public void setEmpId(int empId) {
```

```
        this.empId = empId;
    }

    public String getEmpName() {
        return empName;
    }

    public void setEmpName(String empName) {
        this.empName = empName;
    }

    public double getEmpSal() {
        return empSal;
    }

    public void setEmpSal(double empSal) {
        this.empSal = empSal;
    }

    public Set<Project> getpObj() {
        return pObj;
    }
```

```
	public void setpObj(Set<Project> pObj) {
		this.pObj = pObj;
	}

	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName +
", empSal=" + empSal + ", pObj=" + pObj + "]";
	}
}
```

Project.java

```
@Entity
@Table(name="projtab")
public class Project {

	@Id
	@Column(name="pid")
	private int projId;

	@Column(name="pname")
	private String projName;
```

```
@Column(name="pcost")
private double projCost;



public Project() {
	System.out.println("Project::0-param constructor");
}

public int getProjId() {
	return projId;
}

public void setProjId(int projId) {
	this.projId = projId;
}

public String getProjName() {
	return projName;
}

public void setProjName(String projName) {
```

```
		this.projName = projName;
	}

	public double getProjCost() {
		return projCost;
	}

	public void setProjCost(double projCost) {
		this.projCost = projCost;
	}

	@Override
	public String toString() {
		return "Project [projId=" + projId + ", projName=" + projName + ",
projCost=" + projCost + "]";
	}
}
```

Test.java

```
// Creating Employee,Project object
		Project p1 = new Project();
		p1.setProjId(101);
		p1.setProjName("P1");
```

```
p1.setProjCost(1.1);


Project p2 = new Project();
p2.setProjId(102);
p2.setProjName("P2");
p2.setProjCost(2.2);


Set<Project> pobs = new HashSet<Project>();
pobs.add(p1);
pobs.add(p2);


Employee emp = new Employee();
emp.setEmpId(10);
emp.setEmpName("AAA");
emp.setEmpSal(2.2);
emp.setpObj(pobs);


//save Object
ses.save(p1);
ses.save(p2);


ses.save(emp);
```

ManyToMany [* . . . *] Collection: -

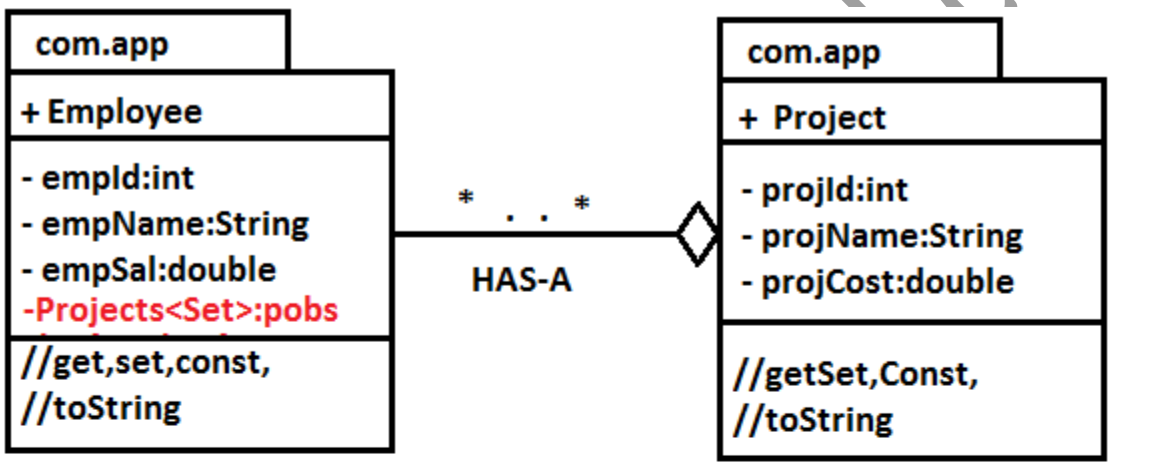
In this case extra table will be created with 2 FK Columns those are

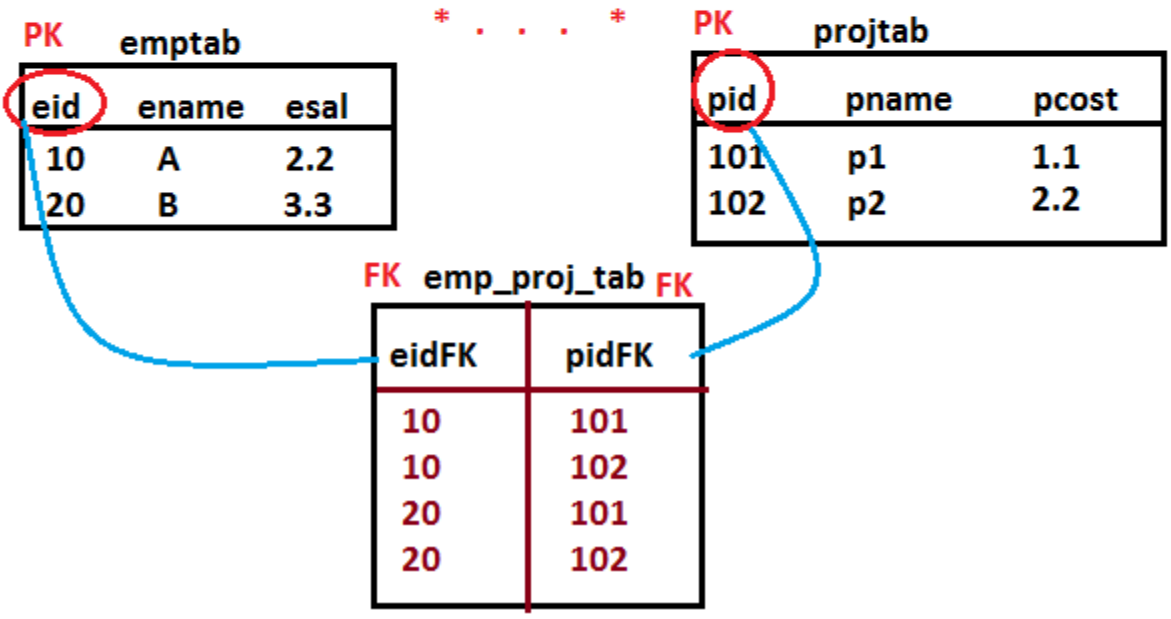
JoinColumn -> FK Column connected to parent table and

InverseJoinColumn ->FK Column connected to child table

1. UML Design for Models

2.Tables output





```
@Entity
@Table(name="addrs_tab")
public class Address {
@Id
@Column(name="aid")
@GeneratedValue
private int addrId;
@Column(name="loc")
private String loc;
@ManyToMany(mappedBy="addr")
private List<Employee> emp=new ArrayList<Employee>(0);
}
```

```
@Entity
@Table(name="empt_tab")
class Employee{
@Id
@Column(name="eid");
private int empId;
@ManyToMany(cascade=CascadeType.ALL,fetch=FetchType.EAGER)
@JoinTable(name="emp_addr",
joinColumns=@JoinColumn(name="eidFk"),
inverseJoinColumns=@JoinColumn(name="aidFk"))
private List<Address> addr=new ArrayList<Address>(0);
}
```

Example:

Employee.java

```
@Entity

@Table(name="emp_tab")

public class Employee {
```

```
@Id
@Column(name="eid")
private int empId;

@Column(name="ename")
private String empName;

@Column(name="eSal")
private double empSal;

@ManyToMany
@JoinTable(name="emp_prj_tab",
 joinColumns=@JoinColumn(name="eidFK"),
 inverseJoinColumns=@JoinColumn(name="pidFK")) // optional
private Set<Project> pObj; // HAS-A

public Employee() {
        System.out.println("Employee::0-param constructor");
   }

public int getEmpId() {
```

```
        return empId;
}

public void setEmpId(int empId) {
        this.empId = empId;
}

public String getEmpName() {
        return empName;
}

public void setEmpName(String empName) {
        this.empName = empName;
}

public double getEmpSal() {
        return empSal;
}

public void setEmpSal(double empSal) {
        this.empSal = empSal;
}
```

```
	public Set<Project> getpObj() {
		return pObj;
	}

	public void setpObj(Set<Project> pObj) {
		this.pObj = pObj;
	}

	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName +
", empSal=" + empSal + ", pObj=" + pObj + "]";
```

Project.java

```
@Entity
@Table(name="proj_tab")
public class Project {

	@Id
	@Column(name="pid")
	private int projId;
```

```
@Column(name="pname")
private String projName;

@Column(name="pcost")
private double projCost;



public Project() {
	System.out.println("Project::0-param constructor");
}

public int getProjId() {
	return projId;
}

public void setProjId(int projId) {
	this.projId = projId;
}

public String getProjName() {
	return projName;
}
```

```
	public void setProjName(String projName) {
		this.projName = projName;
	}

	public double getProjCost() {
		return projCost;
	}

	public void setProjCost(double projCost) {
		this.projCost = projCost;
	}

	@Override
	public String toString() {
		return "Project [projId=" + projId + ", projName=" + projName + ",
projCost=" + projCost + "]";
	}
}
```

Test.java

```
// Creating Employee,Project object
		Project p1 = new Project();
```

```
p1.setProjId(101);
p1.setProjName("P1");
p1.setProjCost(1.1);

Project p2 = new Project();
p2.setProjId(102);
p2.setProjName("P2");
p2.setProjCost(2.2);

Set<Project> pobs = new HashSet<Project>();
pobs.add(p1);
pobs.add(p2);

Employee emp1 = new Employee();
emp1.setEmpId(10);
emp1.setEmpName("A");
emp1.setEmpSal(2.2);
emp1.setpObj(pobs);

Employee emp2 = new Employee();
emp2.setEmpId(20);
emp2.setEmpName("B");
```

```
emp2.setEmpSal(3.3);

emp2.setpObj(pobs);


//save Object

ses.save(p1);

ses.save(p2);


ses.save(emp1);

ses.save(emp2);
```

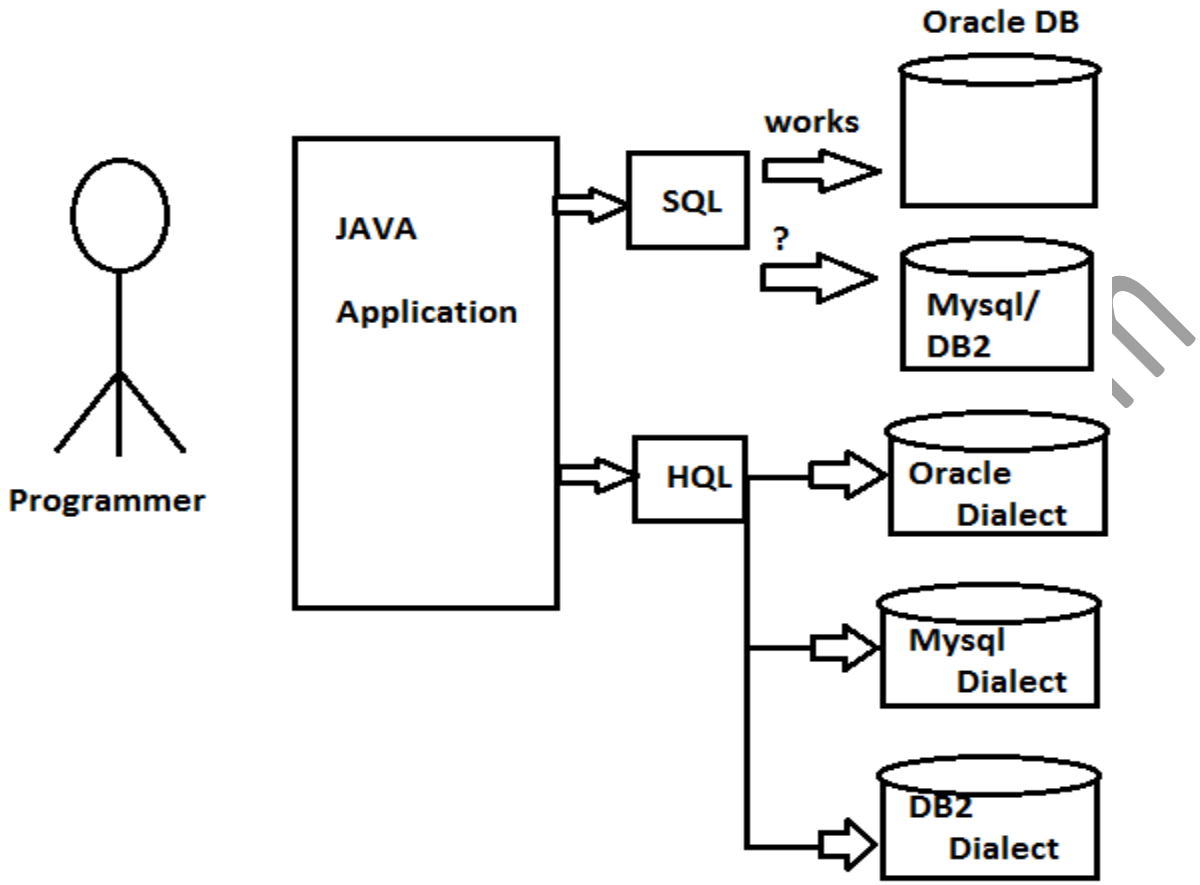
MultiRow Operations: -

By using Session operations like save(), update(), saveOrUpdate(), delete(), get(), and load() at a time we can work with **one Row/Object**.

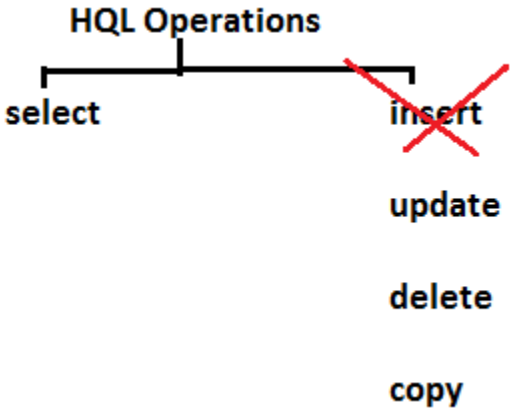
To work with **multiple Row/Object**environments , Hibernate has provided two special concepts. Those are

1. Query API [HQL]
2. Criteria API

Query API [HQL]: - HQL stands for Hibernate Query Operations. It is used to perform multi-row operations.Hibernate supports **both SQL and HQL** concepts.**SQL is Database dependent** i.e. SQL written for one DB may or may not work for another DB. Whereas **HQL**works for all types of Database i.e**. Database independent.**By using **Dialect**, HQL has become Database independent.**HQL supports** both **select and non-select** operations.



HQL supports **multi Row select and non-select** operations but it will **not supportsmulti Row insert operation**.



To convert SQL to HQL replace

Column Name ---→ Variable Name and

Table Name ---→ ClassName

Consider variable, class Name from Model class.

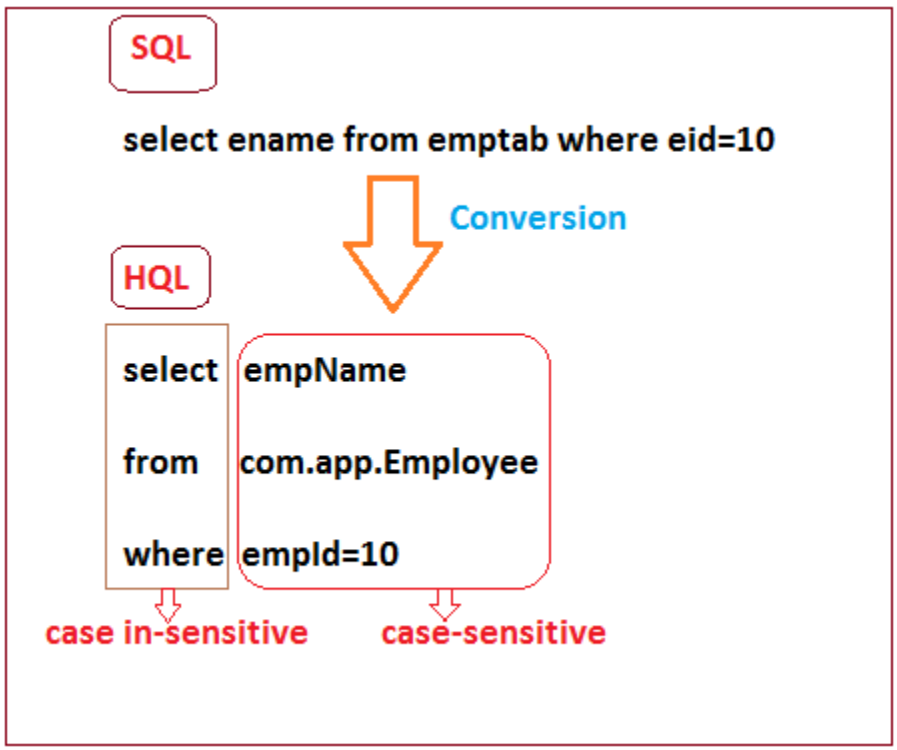
*) Ex:

SQL: ->select ename from emptab where eid=10

HQL: -> select empName from com.app.Employee where empId=10

*) SQL is case-insensitive i.e. we can write Query in any case (both upper and lower case)

But HQL is partially case-sensitive and insensitive. I.e. SQL related words are case insensitive (ex: select, from, where, group by, having, is not, etc.) and Java related words are case sensitive (ex: empId, com.app.Employee, etc.)



Examples: -

1] SQL: select max(esal) from emptab group by esal.

HQL: select max(empSal) from com.app.Employee group by empSal

2] *** (start with from clause in case of full loading/all columns select) ex:**

SQL: select * from emptab;

HQL: from com.app.Employee;

3] SQL: update emptab set esal=? Where eid=?

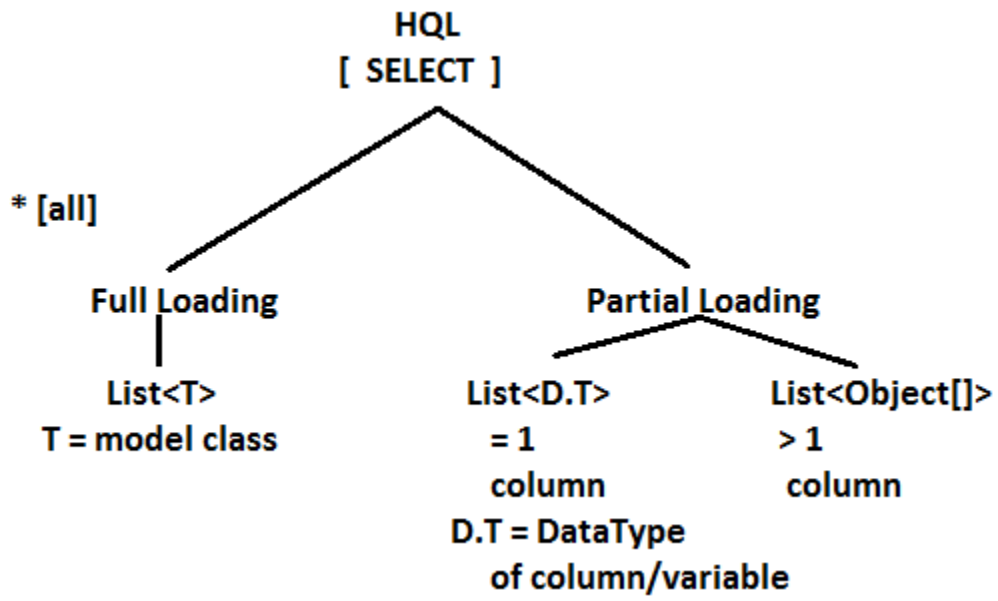
HQL: update com.app.Employee set empSal=? Where empId=?

4] SQL: delete from emptab where eid > ?

HQL: delete from com.app.Employee where empId > ?

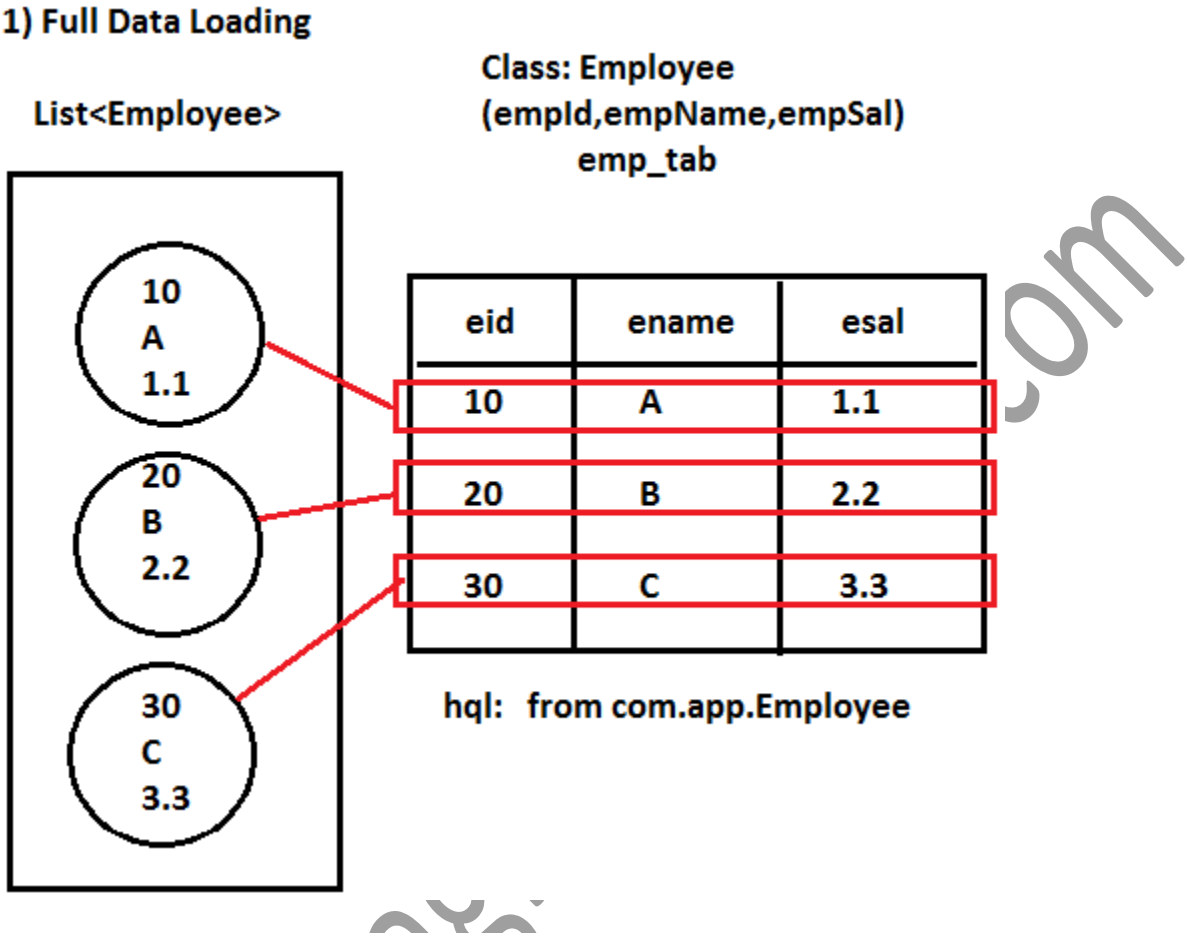
HQL select Operation: -

On executing **HQL select query** it returns data in **List** format, given below.

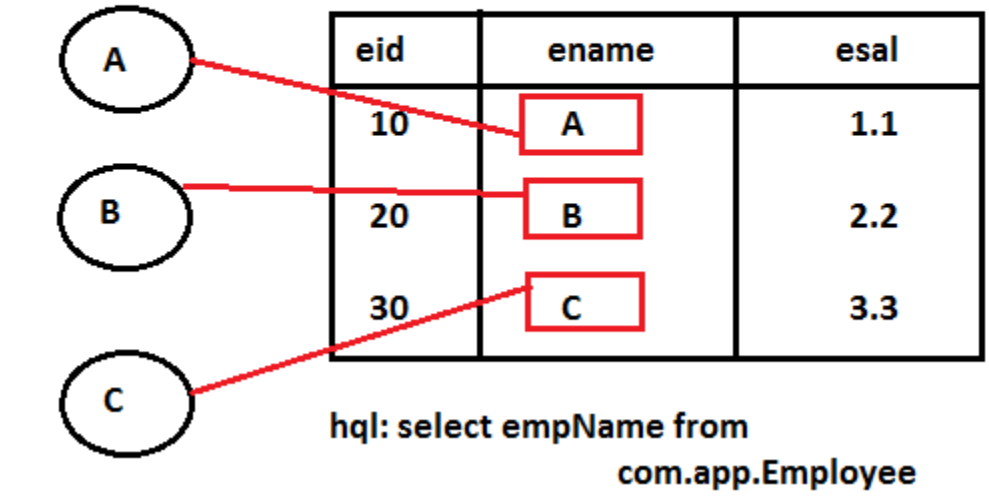


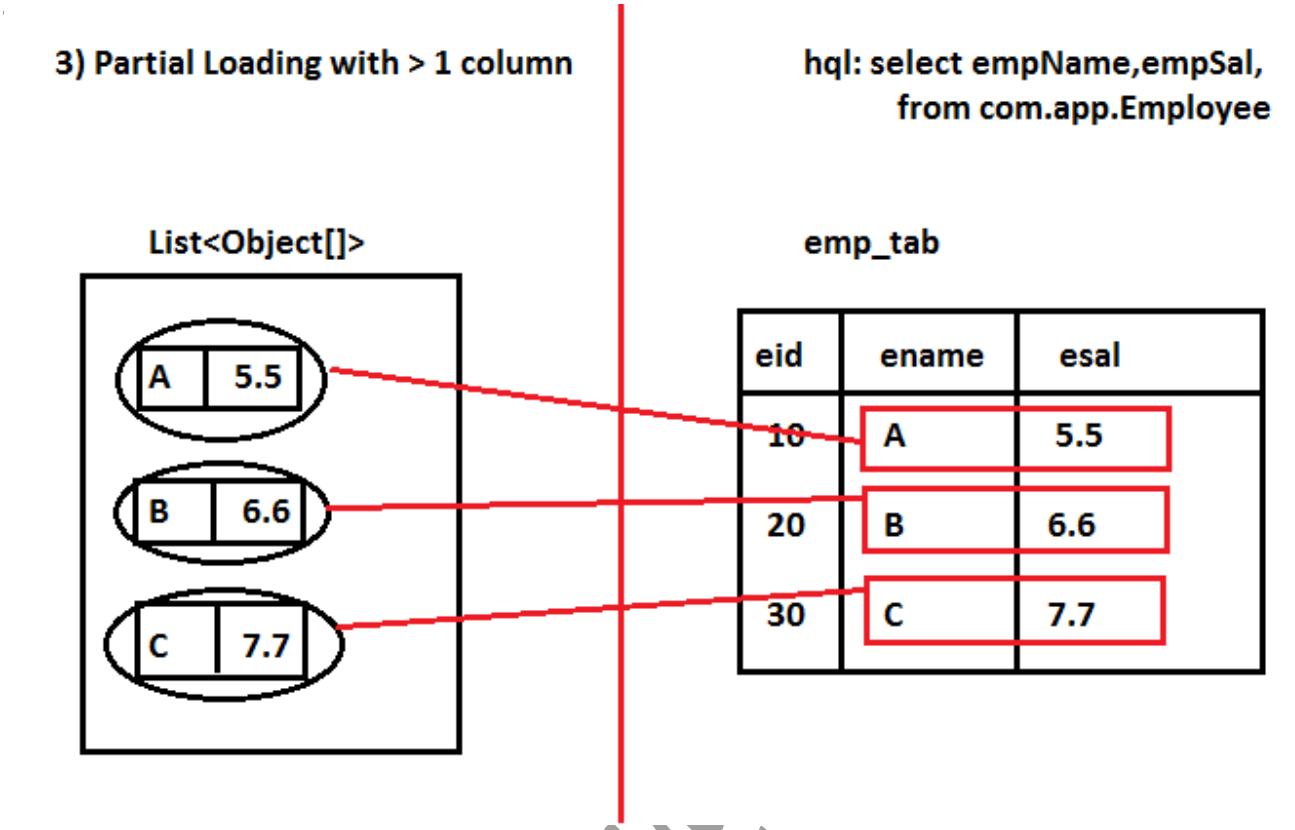
Example:

1) Full Data Loading



2) Partial Loading





Query(HQL) Programming Steps: -

In Test class, after Session Object, write code in below steps:

#1] Create HQL String

#2] Create Query object using createQuery(hql) with session

#3] call list() method on query that returns List<?>

EX: Test class:// cfg,sf,ses

String sql = “from com.app.Employee”;

Query q = ses.createQuery(hql);

List<Employee> el = q.list();

Sysout(el);

*****Syntax to get fully Qualified Name: "T.class.getName()"**

Here T = Model class Name

Example:

Employee.java

```
@Entity
@Table(name="emp_tab")
public class Employee {

    @Id
    @Column(name="eid")
    private int empId;

    @Column(name="ename")
    private String empName;

    @Column(name="eSal")
    private double empSal;

    public Employee() {
        System.out.println("Employee::0-param constructor");
    }
```

```
public int getEmpId() {
        return empId;
}

public void setEmpId(int empId) {
        this.empId = empId;
}

public String getEmpName() {
        return empName;
}

public void setEmpName(String empName) {
        this.empName = empName;
}

public double getEmpSal() {
        return empSal;
}

public void setEmpSal(double empSal) {
        this.empSal = empSal;
```

```
	}


	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName +
", empSal=" + empSal + "]";
	}
}
```

HQL Test.java

```
		Employee emp1 = new Employee();
		emp1.setEmpId(10);
		emp1.setEmpName("A");
		emp1.setEmpSal(2.2);

		Employee emp2 = new Employee();
		emp2.setEmpId(20);
		emp2.setEmpName("B");
		emp2.setEmpSal(3.3);

		Employee emp3 = new Employee();
		emp3.setEmpId(30);
```

```
emp3.setEmpName("C");
emp3.setEmpSal(4.4);

//save Object

ses.save(emp1);
ses.save(emp2);
ses.save(emp3);
```

HQL Test1.java

```
//Create HQL String
        String hql = "From com.sathyatech.model.Employee";

        //Create Query Object with Session
        Query q = ses.createQuery(hql);

        //Call List<>() method
        List<Employee> emps = q.list();
        for(Employee e:emps) {
                System.out.println(e);
        }
```

HQL Test2.java

```
//Create HQL String
                String hql = "From com.sathyatech.model.Employee";

                //Create Query Object with Session
                Query q = ses.createQuery(hql);

                //Call List<>() method
                List<Employee> emps = q.list();
                for(Employee e:emps) {
                        System.out.println(e);
                }
```

HQL Test3.java

```
//Create HQL String
        String hql = "Select empName from
com.sathyatech.model.Employee";

        //Create Query Object with Session
        Query q = ses.createQuery(hql);

        //Call List<>() method
        List<String> emps = q.list();
```

```
for(String e:emps) {
        System.out.println(e);
}
```

HQL Test4.java

```
//Create HQL String
        String hql = "Select empName,empSal from
com.sathyatech.model.Employee";

        //Create Query Object with Session
        Query q = ses.createQuery(hql);

        //Call List<>() method
        List<Object []> emps = q.list();
        for(Object [] e:emps) {
            System.out.println(e[0]);
            System.out.println(e[1]);
        //  System.out.println(e[2]);
        }
```

HQL Test5.java

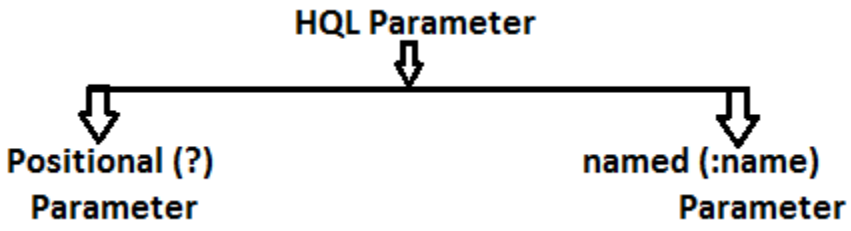
```
//Create HQL String
        String hql = "from "+Employee.class.getName();
```

```
//Create Query Object with Session
Query q = ses.createQuery(hql);


//Call List<>() method
List<Employee> emps = q.list();
for(Employee e:emps) {
        System.out.println(e);
}
```

HQL Parameters: -

To provide a value to HQL to where class only at runtime, hibernate provided parameters concept.

It supports 2 types of Parameters, Those are:



1] Positional Parameter: It is concept from JDBC, also supports in Hibernate.

It provides index number starts from **zero(0)** to every **?** symbol in HQL.Set data using **q.setParameter(index,data)** method

This method must be called after **query object** and before **list()** method.

```
//Create HQL String
String hql = "from com.sathyatech.test.Employee where empId>? or empName=? and empSal<?";

                    //Create Query Object with Session
                    Query q = ses.createQuery(hql);

                    //Set query to QueryParameters
                    q.setParameter(0,10);
                    q.setParameter(1,20);
                    q.setParameter(2,30);
```

Note: If where clause is modified then positions of parameters may get effected then we must re-check and re-write code for setParameters, if not it may leads to exception/error/wrong output.

So, Positional Parameters are not preferred in Hibernate.

From com.app.Employee where empId<? >>0

Or empName = ? >> 1

And empSal > ? >>2

**After modifications, HQL is from com.app.Employee where

empDsg = ? >> 0

empId < ? >> 0 -> 1

or empName = ? >> 1 ->2

and empSal > ? >> 2 -> 3

Example: Positional Parameters

Test.java

```
Employee emp1 = new Employee();
        emp1.setEmpId(10);
        emp1.setEmpName("A");
        emp1.setEmpSal(2.2);

        Employee emp2 = new Employee();
        emp2.setEmpId(20);
        emp2.setEmpName("B");
        emp2.setEmpSal(3.3);

        Employee emp3 = new Employee();
        emp3.setEmpId(30);
        emp3.setEmpName("B");
        emp3.setEmpSal(4.4);

        //save Object

        ses.save(emp1);
        ses.save(emp2);
        ses.save(emp3);
```

HQL Test.java

```
//Create HQL String

                String hql = "from com.sathyatech.model.Employee
where empId >? or empName =? and empSal <?";


                //Create Query Object with Session

                Query q = ses.createQuery(hql);


                //Set query to QueryParameters

                q.setParameter(0,10);

                q.setParameter(1,20);

                q.setParameter(2,30);


                //Call List<>() method

                List<Employee> emps = q.list();

                for(Employee e:emps) {

                        System.out.println(e);

                }
```

2]NamedParameter: -(:name)

These are used to provide a name to parameter that will be replaced with a value at runtime.

**Even where clause changed, names will not get affected.

So, it is better than Positional Parameter.

Use **"setParameter(name,data)" to set value at runtime to named Parameter.

Ex: Below HQL is written with Positional Parameters

from com.app.Employee where empId > ? and empName = ? or empSal < ?

? Replaced with:[Named Parameter]

from com.app.Employee where empId >**: a** and empName **: b** or empSal **< : c**

Example : HQL Test.java

```
//Create HQL String
                String hql = "from com.sathyatech.model.Employee
where empId >= :empId and empName = :empName";

                //Create Query Object with Session
                Query q = ses.createQuery(hql);

                //Set query to QueryParameters
                q.setParameter("empId", 11);
                q.setParameter("empName", "B");
```

```
//Call List<>() method
List<Employee> emps = q.list();
for(Employee e:emps) {
        System.out.println(e);
}
```

IN Clause In HQL: -

in clause is used to selected random rows (specific rows, not a range).

Ex: SQL: select * from emptab where eid in (3,5,7,10);

SQL: select * from emptab where ename in ("C","D","F");

**To handle in clause in HQL (Hibernate):

#1]create one List and add in clause values

#2]Write in clause in HQL

#3]pass data using **setParameterList(param,List)**

#4]execute Query

Q) Can we use both Positional and Named Parameter in one HQL?

→ Yes, But we must **start with Positional parameter one done**, then start using **Named Parameter**.

After Named Parameter, **no Positional Parameters** are accepted.

If we use Positional Parameter after Named Parameter in HQL, then Hibernate throws, **QuerySyntaxException:cannot define positional parameter after any named parameters have been defined.**

Examples Orders for Parameters:

#1 ? ? :a :b (Valid)

#2 ? :a ? ? (Invalid)

#3 :a :b :c :d (Valid)

#4 ? ? ? ? (Valid)

#5 :a ? :b ? (Invalid)

#6 ? :a ? :b (Invalid) #7 ? ? ? :a (Valid)

IN Clouse Example: HQL Test.java

```
//Create HQL String
                String hql = "from com.sathyatech.model.Employee
where empId in (:ab)";


                /*      List<Integer> data = new ArrayList<Integer>();
                data.add(11);
                data.add(12);
                data.add(13); */
```

```
List<Integer> list = Arrays.asList(10,20,30);



//Create Query Object with Session
Query q = ses.createQuery(hql);


//Set query to setParametersList
q.setParameterList("ab", list);


//Call List<>() method
List<Employee> emps = q.list();
for(Employee e:emps) {
        System.out.println(e);
    }
```

Named + Positional Parameter Example: HQL Test.java

```
//Create HQL String
        String hql = "from com.sathyatech.model.Employee
where empId =? or empName= :a and empSal= :b";


//Create Query Object with Session
Query q = ses.createQuery(hql);
```

```
//Set query to setParametersList
q.setParameter(0, 10);
q.setParameter("a", "A");
q.setParameter("b", 2.2);



//Call List<>() method
List<Employee> emps = q.list();
for(Employee e:emps) {
        System.out.println(e);
}
```

uniqueResult():Object -→

If query returns only [exactly]one value/row, then use uniqueResult() in place of list() method.

Ex: Query Returns only 1 value

#1 select count(empId) from Employee

#2 select avg(empSal) from Employee

#3 select sum(empSal) from Employee

#4 select min(empSal) from Employee

#5 select max(empSal) from Employee

** IF query returns more than one value, while using uniqueResult() method, Hiberate throws exception: **NonUniqueResultException:query did not return a unique result**

HQL Non-Select Operations: -

HQL supports 3 non-select Operations, those are

#1.update()

#2.delete()

#3.copy() [Copy from one table to another table]

Steps To Write Code: -

1] create HQL String

2] create Query Object

3] call exceuteUpdate(): int

(int = count = no.of rows effected)

Update SQL Syntax: -

Update <tableName> set <colName>=<value>,…where <condition>.

Update SQL Example: -

Update emptab set ename=?, esal=? Where eid=?

Equal HQL Query For Update: -

Update com.app.Employee set empName=?, empSal=? Where empId=?

uniqueResultExample: Test.java

```
//Create HQL String
                String hql = "from com.sathyatech.model.Employee
where empId=?";

                //Create Query Object with Session
                Query q = ses.createQuery(hql);

                //Set query to setParametersList
                q.setParameter(0, 10);

                //Call uniqueResult() method
                Object obj = q.uniqueResult();
```

```
//Downcasting
Employee e = (Employee)obj;

System.out.println(e);
```

Delete Operation in HQL: -

HQL Example:

delete from com.app.Employee where empId:empId;

Example: Test.java

```
//Create HQL String
        String hql = "delete from com.sathyatech.model.Employee where empId=:empId";

        //Create Query Object with Session
        Query q = ses.createQuery(hql);

        //Set query to setParametersList
        q.setParameter("empId", 20);

        //Call executeUpdate() method
        int count = q.executeUpdate();
```

```
System.out.println("Deleted Records are :: "+count);
```

Update Operation : HQL Update Test.java

```
//Create HQL String

String hql = "update com.sathyatech.model.Employee
set empName=:empName, empSal=:empSal where empId=:empId";


//Create Query Object with Session

Query q = ses.createQuery(hql);



//Set query to setParametersList

q.setParameter("empName", "ABC");

q.setParameter("empSal", 44.44);

q.setParameter("empId", 10);


//Call executeUpdate() method

int count = q.executeUpdate();


System.out.println("Updated Records are :: "+count);
```

Copy Data in HQL: -

This concept is used to copy rows from one table to another table (like source table to backup or destination table).

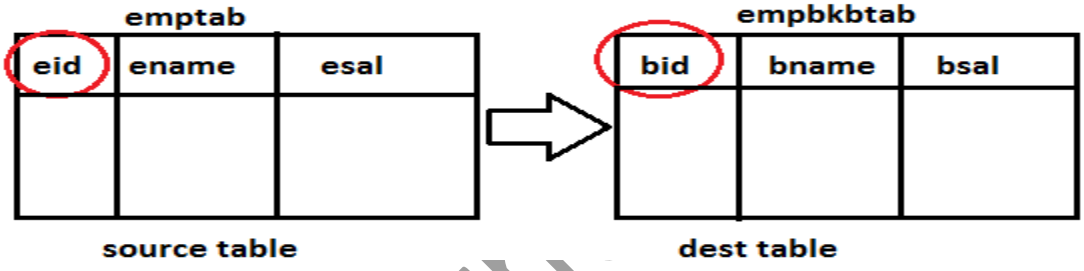
This concept is from SQL and also supported in Hibernate.

Syntax:

insert into … (source_tab)

select …. from .. (destination_tab)

Example: Consider below ex copy rows from emptab(source) to empbkptab(destination) having design



SQL:

insert into empbkbtab(bid,bname,bsal)

Select eid,ename,esal from emptab

HQL:

Insert into com.app.EmployeeBackUp(bid,bname,bsal)

Select empId,empName,empSal from com.app.Employee;

Example:

Employee.java

@Entity

@Table(name="emp_tab")

```
public class Employee {


	@Id
	@Column(name="eid")
	private int empId;


	@Column(name="ename")
	private String empName;


	@Column(name="eSal")
	private double empSal;
```

EmployeeBackUp.java

```
@Entity
@Table(name="empbkp_tab")
public class EmployeeBackUp {


	@Id
	@Column(name="bid")
	private int bkpId;
	@Column(name="bname")
	private String bkpName;
```

```
@Column(name="bsal")
private double bkpSal;
```

Test.java

```
//Create HQL String
                String hql = "insert into
com.sathyatech.model.EmployeeBackUp(bkpId,bkpName,bkpSal) select
empId,empName,empSal from com.sathyatech.model.Employee";

                //Create Query Object with Session
                Query q = ses.createQuery(hql);

                //Call executeUpdate() method
                int count = q.executeUpdate();

                System.out.println("Copied Records are :: "+count);
```

PAGINATION METHODS: -

To select rows in a range without using external where clause (by programer), Pagination methods are given from hibernate with Query and Criteria API.

Methods are:

setFirstResult(int): Starting index number

setMaxResult(int): max rows to be selected.

Example:

select rows from emptab, having eid range 92 to 30.

Test Class: //cfg,sf,ses,tx

String hql = “from com.app.Employee”;

Query q = ses.createQuery(hql);

q.setFirstResult(3);

q.setMaxResult(9);

List<Employee> emps = q.list();

for(Employee e:emps){Sysout(e)}

Question: if q.setFirstResult(6);

q.setMaxResult(4),

then which rows are selected? Write employee id’s of selected rows?

Ans: Employee id’s#-21,17,19,22

index in hb

index	eid	ename	esal
0	12	A	1.1
1	14	B	2.0
2	36	C	3.0
3	92	D	4.0
4	80	E	5.0
5	16	F	6.0
6	21	G	7.0
7	17	H	8.0
8	19	I	9.0
9	22	J	0.1
10	36	K	0.2
11	30	L	0.3
12	11	M	0.4
13	10	N	0.5
14	44	O	0.6
15	89	P	0.7

Criteria API: -

This concept is given by Hibernate to perform “Multi Row-Select” Operations.

It will not select Non-Select Operations.

No external query should be written by Programmer.

It is advanced concept of HQL (Query API).

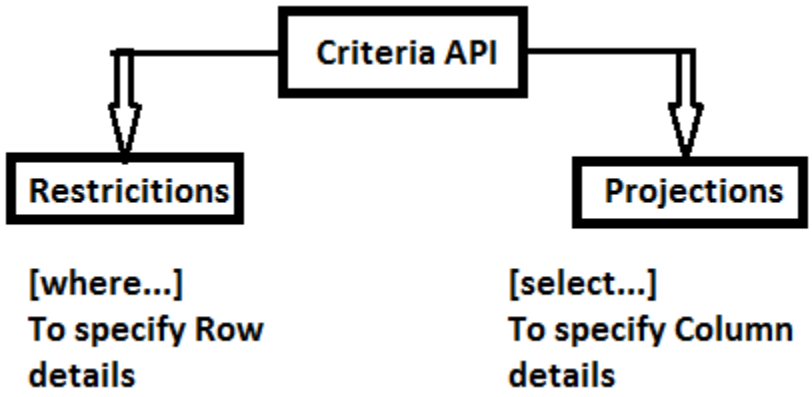
It supports Data loading from table to Application as List.

By default it will select “All Rows and All Columns”.

It supports two child API’s: -

#1] Restrictions (To specify where clause)

#2] Projections (To specify select clause).



#1.Create Criteria Object using Session.

Criteria c = ses.createCriteria(T.class);

#2. Call list() /uniqueResult for execution

List<T> data = c.list();

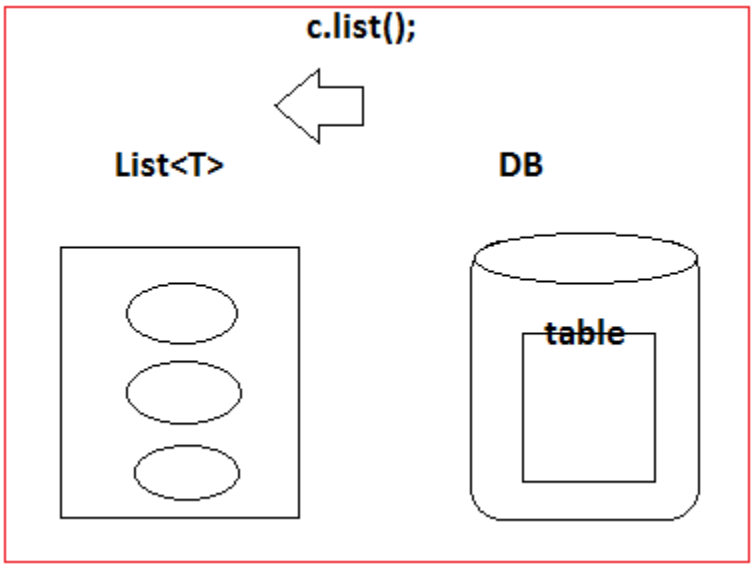
#3. Print data

Sysout(data);

By default “Criteria API” will execute query like “select * from <table>”.

It means,selecting all columns (full loading) and all rows.

Complete table rows are loaded into Session cache as a List<T>.



Syntax To Create Criteria Object: -

By using createCriteria() method which is in Session we can create Criteria object by passing

#1] .class Parameter

#2] Fully Qualified Class Name

Syntax#1

Criteria c = ses.createCriteria(T.class);

List <T> data = c.list();

Syntax#2

Criteria c = ses.createCriteria("com.app.Employee");

List<T> data = c.list();

Example: Test.java

//Create Criteria Object with Session

Criteria c = ses.createCriteria(Employee.class);

```
/* OR
 * Criteria c =
ses.createCriteria("com.sathyatech.model.Employee");
 */

//Call list()/uniqueResult() method
List<Employee> emps = c.list();
```

Criteria API With Projections: -

To select required columns from DB table use Projections API with Criteria Object.

Projections API is a child API of Criteria API. That means without Criteria, Projections will not work.

Using only Criteria (no projections) indicates full loading.

Criteria with Projections indicates partial loading.

--Steps to implement Projections—

#1 Create Criteria Object using session

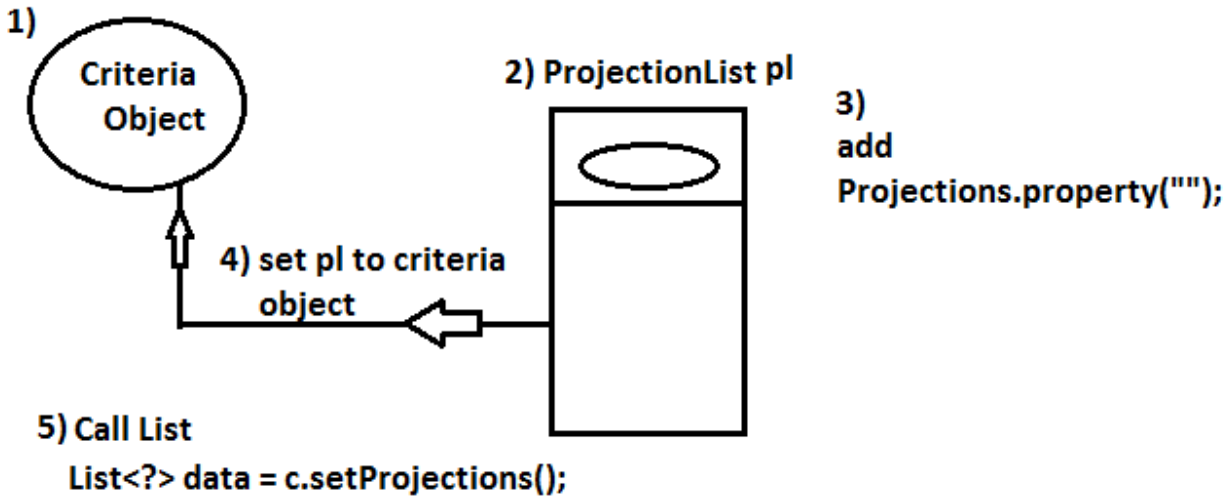
#2 Create empty “ProjectionsList”

#3 Add one by one “Projections Property to Projections List”

#4 Set PL to Criteria Object

#5 call list() method

--Flow Design--



Example: Criteria API Projection – One Or More Column ->

```
//Create Criteria Object with Session
                    Criteria c = ses.createCriteria(Employee.class);

                    /* OR
                     * Criteria c =
ses.createCriteria("com.sathyatech.model.Employee");
                    */

                    //Create empty ProjectionList obj

                    ProjectionList pl = Projections.projectionList();

                    //add Property to ProjectionList
```

```
pl.add(Projections.property("empName"));
pl.add(Projections.property("empSal"));



//Set ProjectionList to criteria Obj
c.setProjection(pl);


//Call list()/uniqueResult() method
List<Object[]> emps = c.list();


//Print Data
for(Object[] ob:emps) {
        System.out.println(ob[0]);
        System.out.println(ob[1]);
}
```

Restrictions: -

A Restrictions is a child API of Criteria. It is used to specify “where clause”, to select rows (not all).

To create where clause conditions, Restrictions class has provided methods for every symbol to indicate one condition.

EX:

SYMBOL	METHOD
>	gt
>=	ge
<	lt
<=	le
<> / !=	ne
=	eg
and	and
or	or
like	(like/ilike)
in	in
not in	not in

Steps To implement Restrictions: -

#1 Create Criteria Object

#2 Create Restrictions and add to Criteria Object ***

#3 Call list() / uniqueResult() method

1) Requirement: -

select * from emptab

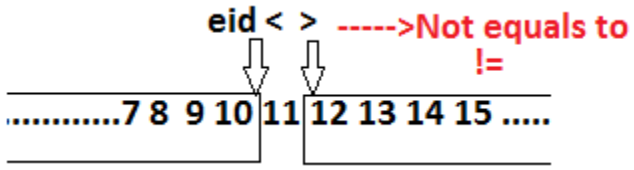
where **empId>11**

//select * from emptab

Criteria c = ses.createCriteria("Employee.class");

//where empId>=11

c.add(Restrictions .le("empId",11));



Restrictions API Examples: -

1)//eid>10

Restrictions.gt(“empId”,10);

2)//eid>=10

Restrictions.ge(“empId”,10);

3)//eid<10

Restrictions.lt(“empId”,10);

4)//eid<=10

Restrictions.le(“empId”,10);

5)//eid=10

Restrictions.eq(“empId”,10);

6)//eid<>10 / eid!=10

Restrictions.ne(“empId”,10);

7)//eid>=10 and eid<=20

Restrictions.between(“empId”,10,20);

8)//ename is null

Restrictions.isNull(“empName”);

9)//ename is not null

Restrictions.isNotNull(“empName”);

10)//ename exactly 3 chars

//ename like '_ _ _'

Restrictions.like("empName","___");

11)//ename like '_a%' or ename like '_A%'

//ignore case(upper & lower accepted)

Restrictions.ilike("empName","_A%");

12)//eid in (10,36,49)

Restrictions.in("empId",new Object[]{10,36,49});

13)//eid>10 or esal<=500

Restrictions.or(Restrictions.gt("empId",10),Restrictions.le("empSal",500.0));

14)//eid>10 and esal<=500

Restrictions.and(Restrictions.gt("empId",10),Restrictions.le("empSal",500.0));

Example: CriteriaAPI Restriction One Or More Row:

```
//Create Criteria Object with Session
                Criteria c = ses.createCriteria(Employee.class);

                /* OR
                 * Criteria c =
ses.createCriteria("com.sathyatech.model.Employee");
                 */
```

```
//Create Restrictions and add to Ctriteria Obj
c.add(Restrictions.le("empId", 11));
//Call list()/uniqueResult() method
List<Employee> emps = c.list();

//Print Data
for(Employee e:emps) {
        System.out.println(e);
}
```

Q) Write Criteria API code for below query?

SQL:- select ename,esal from emptab where (eid>10 or eid!=11) and (ename not null or esal >= 52.36)

```
Criteria c = ses.createCriteria(Employee.class);
c.add(Restrictions.and
(Restrictions.or(Restrictions.gt(“empId”,10),Restrictions.ne(“empId”,11)),
Restrictions.or(Restrictions.NotNull(“empName”),Restrictions.ge(“empSal”,52.36)
))
ProjectionList pl = Projections.projectionList();
```

```
pl.add(Projections.property("empName"));
pl.add(Projections.property("empSal"));
c.setProjection(pl);
List<Object[]> emps = c.list();
for(Object[] ob:emps){
Sysout(ob[0]);
Sysout(ob[1]);
}
```

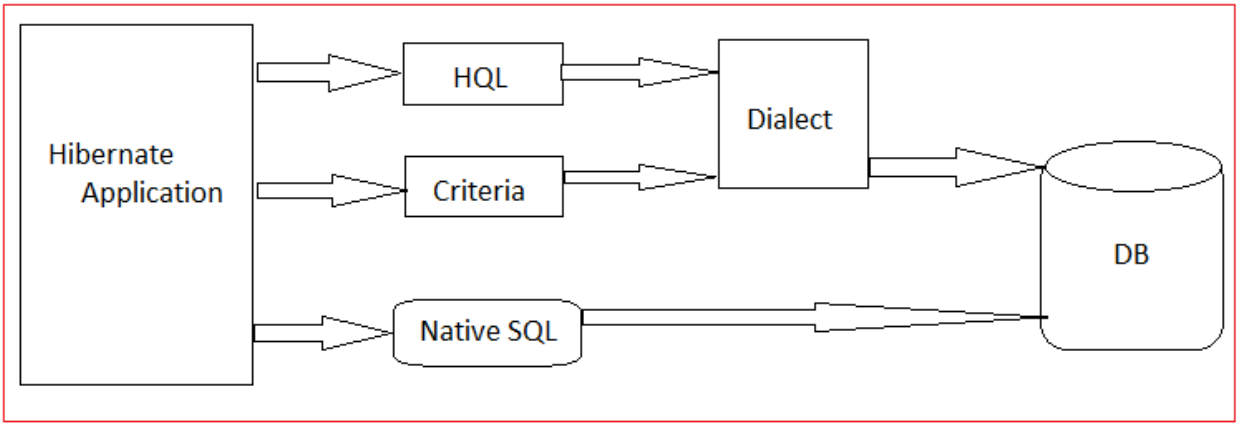
Native SQL: -

Hibernate supports writing SQL quieries in application, this is called as Native SQL.

In case of HQL/Criteria, those are given to Dialect to convert to SQL format.

** But native SQL has no conversion. So, SQL is faster in execution if compared with HQL/Criteria.

**SQL is DB Dependent, where HQL and Criteria is DB Independent.



**Native SQL supports both select and non-select operations.

*Native SQL supports Parameter Passing.

*Native SQL supports list(), uniqueResult(), and exceuteUpdate() methods.

*Native SQL supports both full loading and partial loading.

Steps to Native SQL Coding: -

#1 Create SQL String.

#2 Create SQLQuery object using Session

#3 Set Parameter if exist

#4 call list() /exceuteUpdate() method

#5 Print Result.

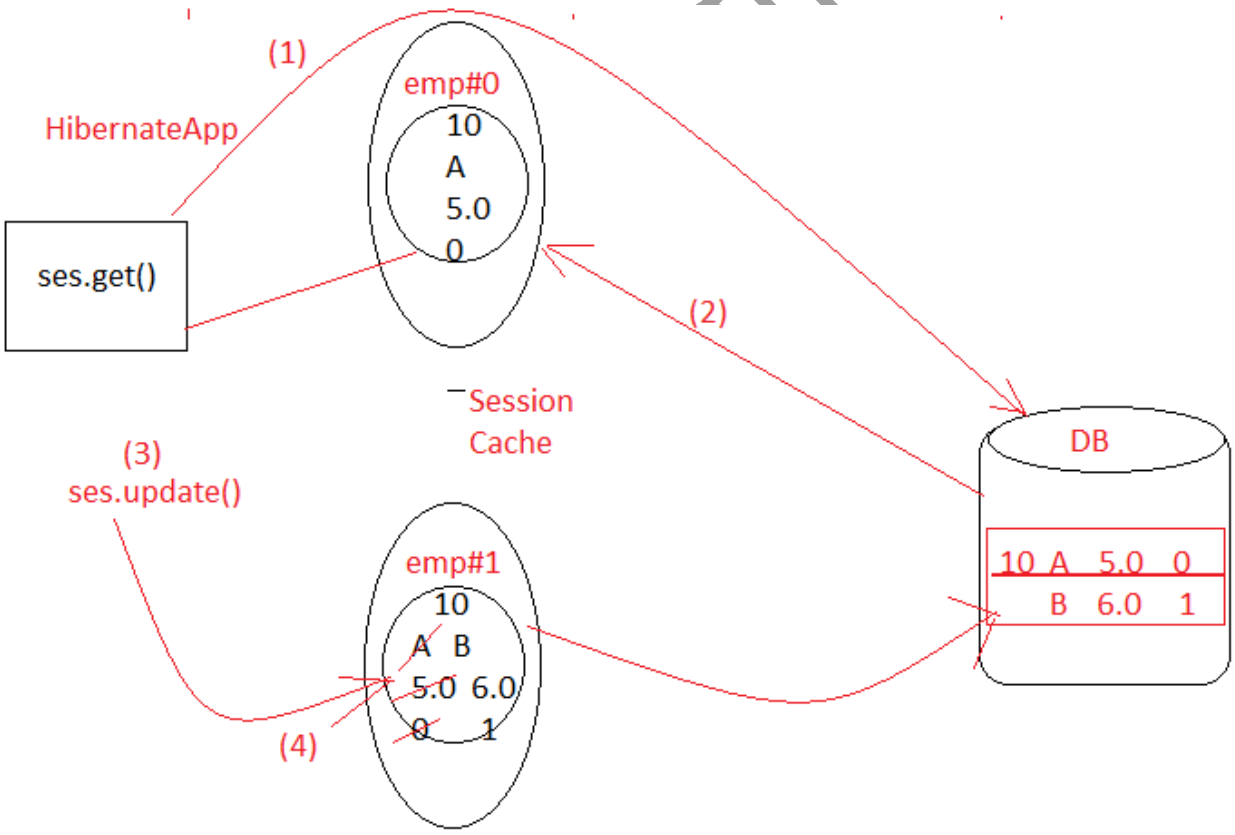
Version Column in Hibernate: -

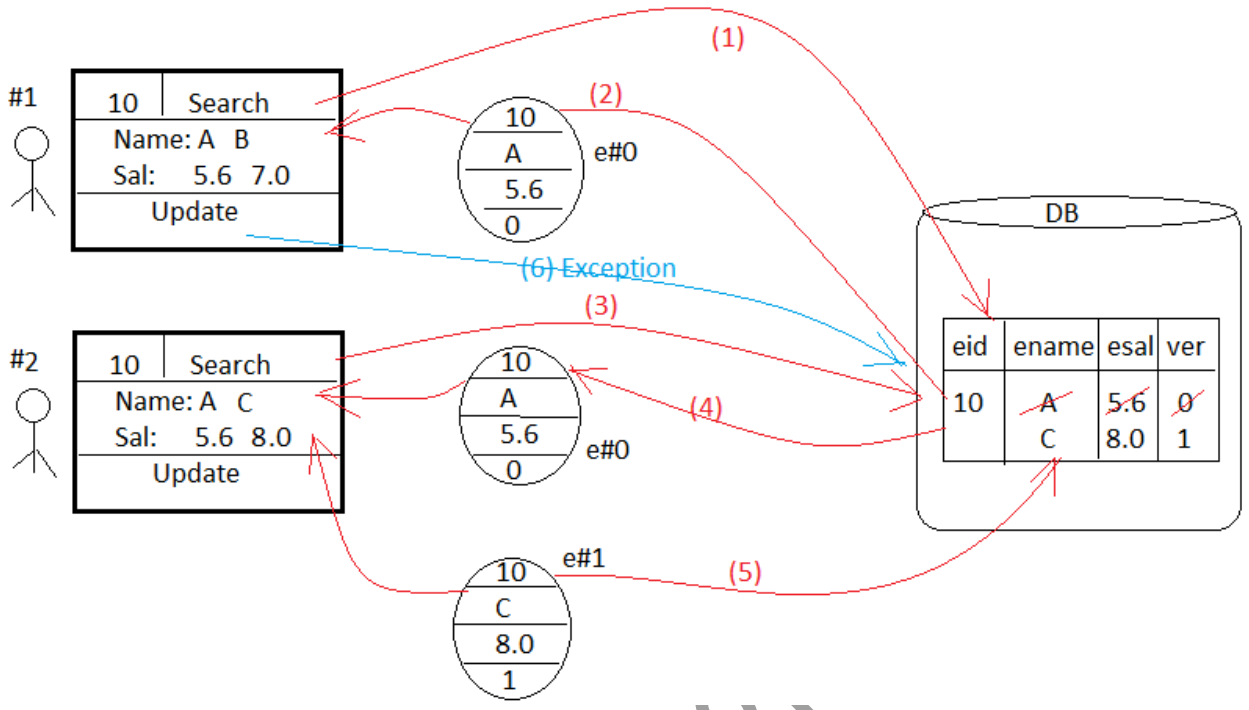
This column is handled by Hibernate only. It is used to count number of times row is updated.

It is also used to lock object in case of parallel access in multiple systems.

It is a type of object locking.

It updates rows always based on primary key column and version column.





Example:

Test1.java

```
Employee emp1 = new Employee();
        emp1.setEmpId(10);
        emp1.setEmpName("A");
        emp1.setEmpSal(2.2);

        Employee emp2 = new Employee();
        emp2.setEmpId(11);
        emp2.setEmpName("B");
        emp2.setEmpSal(3.3);
```

```
Employee emp3 = new Employee();
emp3.setEmpId(12);
emp3.setEmpName("B");
emp3.setEmpSal(4.4);

//save Object

ses.save(emp1);
ses.save(emp2);
ses.save(emp3);
```

Test2.java

```
String sql = "select ename,esal from emptab where eid>?";

//Create SQL Query
SQLQuery q = ses.createSQLQuery(sql);

q.setParameter(0, 10);

//Call list()/uniqueResult() method
List<Object[]> emps = q.list();
```

```
//Print Data
for(Object[] ob:emps) {
        System.out.println(ob[0]);
        System.out.println(ob[1]);
}
```

Test3.java

```
String sql = "delete from emptab where eid=?";

//Create SQL Query
SQLQuery q = ses.createSQLQuery(sql);

q.setParameter(0, 10);

//Call list()/uniqueResult() method
int count = q.executeUpdate();

System.out.println("Deleted Employees are:: "+count);
```

VersionColumn Example:

```
@Entity
@Table(name="emptab")
public class Employee {
```

```
@Id
@Column(name="emp_Id")
private int empId;

@Column(name="emp_Name")
private String empName;

@Column(name="emp_Sal")
private double empSal;

@Version
@Column(name="ver_Emp")
private int verEmp;
```

Serializable_Id Example:

```
// Creating object
        Employee emp = new Employee();
        emp.setEmpId(123);
        emp.setEmpName("Sam");
        emp.setEmpSal(12.36);

        // step 5: execute operation
```

```
Serializable id = ses.save(emp);


int eid = (Integer)id;//Downcasting and AutoUnboxing


/*
 * Integer nid = (Integer)id  //downcasting
 * int eid = nid; // auto un boxing
 */
System.out.println("Generated Emp Id::" +eid);
```

Hibernate Generators: -

Generators are classes in Hibernate.

These are used to generate one Primary Key value on save or insert operation.

Ex: Aadhar Card Id, Voter Id, Pan No, Driving Lic are primary keys which are generated on registration (save) operations.

Few more Ex: Amazon OrderId, Tracking Id, Mobile No, Recharge Ref No, bar code of Product.

These id's are generated when those records are created in application.

***Generators works only when save(insert) operation is done. Not application for other operations.

Types of Generators In Hibernate: -

1) Pre-defined generators

2) Custom Generators

1] **Pre-defined generators: -** These are already exist in Hibernate as classes.

Ex: sequence, increment, hilo, native, assigned, etc.

2] **Custom generators: -** We can define our own format of PrimaryKey.

Ex: 5chars + 4Numbers + 1Char = Pan Card Format.

CollegeCode + groupCode + number = OU18G1521500250 (PK)

AutoBoxing/AutoUnBoxing: -

It is a process of converting primitive to wrapper object without using externally constructors by Programmer.

Ex: AutoBoxing

int p=55; //Primitive

Integer e = p; //AutoBoxed (Object)

AutoUnBoxing: Converting wrapper object to Primitive type without using external methods of wrapper class.

Ex: Auto UnBoxing

Integer a = new Integer(5558); (Object)

int p = a;// Auto Unboxing

Upcasting and Downcasting: -

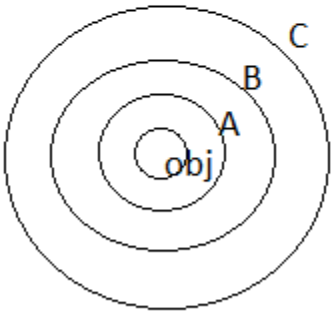
Upcasting: - A sub class object will be refered by it's one of super Type reference.

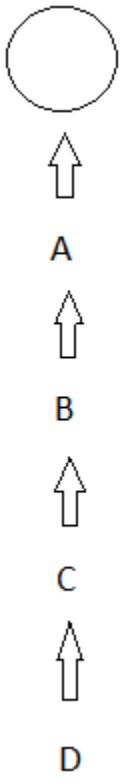
** IS-A (Inheritance) must exist to do upcasting.

** When object is created to one class memory is allocated to current class and it's super classes also.

Upcasting#

A a1 = new C();





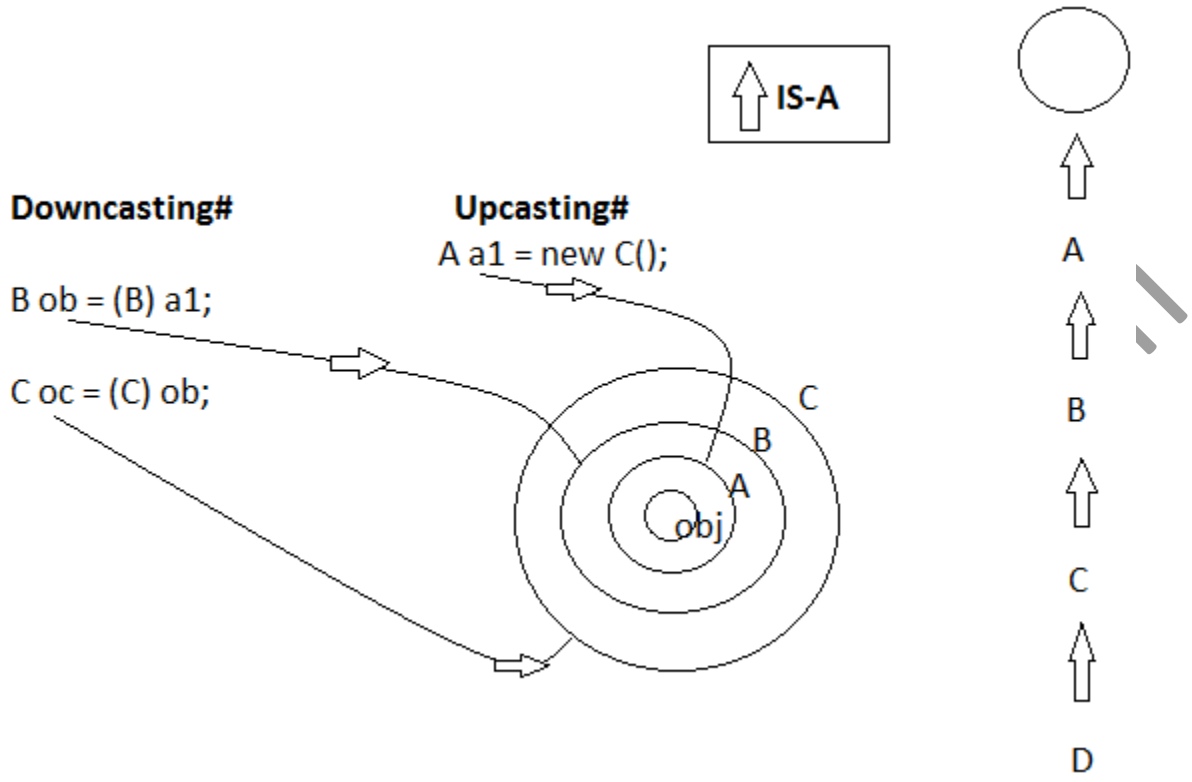
Downcasting: -

Only upcasted object can be downcasted.

To do Downcasting:

I] IS-A relation should exist between classes.

Ii] Already upcasted.



Q) Which Datatype can be used for Primary Key variable creation in Hibernate Model Class?

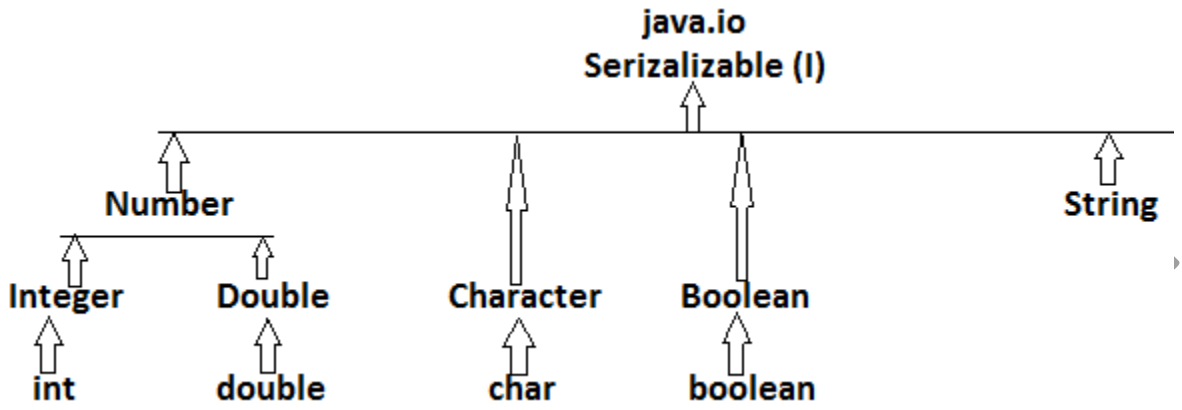
→ Any Data Type (Primitive or Class Type) which is directly or indirectly connected to java.io.Serializable(I).

Few examples are: byte,short,int,long,float,double,boolean, char and it's Wrappers, String are allowed to use as Primary Key Data type.

Ex: Serializable s = 200; is valid

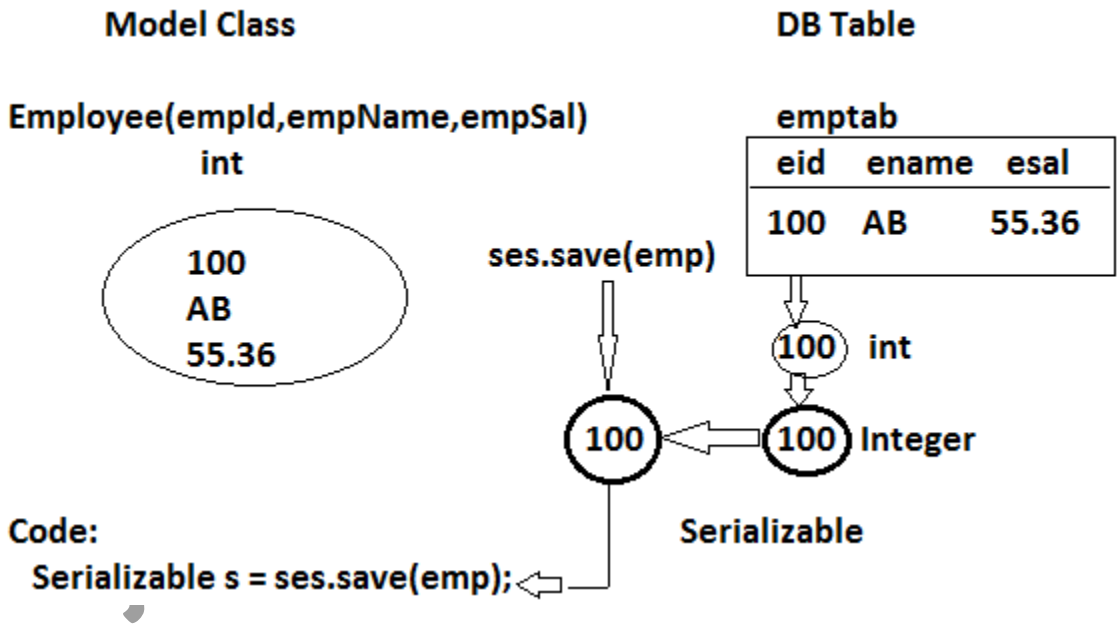
200 (int) -> Integer -> Number -> Serializable

-------(AutoBoxing)----(Upcasting)----------



save(obj):Serializable→

save() method is used to convert given object to DB table Row and returns Primary Key value as Serializable format by performing "autoboxing and upcasting".



Generators Example:

On performing save operation, hibernate generates one new value and set to primary key variable before save.

This value will be converted to Serializable and returned.

We should downcast and autoUnBoxing to see this generated value.

** use @GeneratedValue annotation in model class on top of primary key variable.

Example:

```
@Entity
@Table(name="emptab")
public class Employee {

    @Id
    @Column(name="emp_Id")
    //@GeneratedValue
    @GeneratedValue(generator="emp",strategy=GenerationType.SEQUENCE)
    @SequenceGenerator(name="emp",sequenceName="empseq",initialValue
=2,allocationSize=5)
    //@GeneratedValue(strategy=GenerationType.IDENTITY)
    //@GeneratedValue(strategy=GenerationType.TABLE)
    private int empId;
```

Generation Type: -

To generate primary key value on save operation, hibernate has provided different ways. These different ways are called as Generation Type.

Every type used one concept to generate number for primary key.

All possible values are given in enum i.e.

Bydefault **AUTO** is the value of Generation Type.

enum GenerationType{

AUTO, SEQUENCE, INDENTITY, TABLE

}

SEQUENCE Generator: -

It is a process of number generation in Database.

SEQUENCE concept is not supported by all databases.

Ex: Oracle supports SEQUENCE but not MySQL.

** In case of this generator, sequence created and called (executed) by hibernate only, on calling save operation.

Oracle DB Coding:

#1 Creating Sequence:

create sequence <seq_name> starts with <value> increament by <value>

#2 execute/call sequence: -

select <seq-name>.nextval from <table>

Ex:

create sequence sample starts with 5 increament by 2;

select sample.nextval from dual

** Hibernate by default creates one sequence with name “hibernate_sequence”.

*On save hibernate calls sql like select hibernate_sequence.nextval from dual to get Primary key value.

3]AUTO: It will choose one of below GeneratorType

IDENTITY	For mysql or similar
SEQUENCE	For oracle or similar
TABLE	for Others DBs

**It will detects automatically. It is only default GeneratorType.

4] TABLE: It uses **Hilo algorithm** to generate Primary Key value.

@GeneratedValue(strategy=GeneratorType.TABLE)

It generates values like:

2 pow 0 = 1 (Starting value)

2 pow 15 = 32768 (next) 2nd value

2 pow 15 + prev = 65536

Note: If we write code like @GeneratedValue which is equals to @GeneratedValue(strategy=GenerationType.AUTO)

It will check DB type and selects one of SEQUENCE, IDENTITY, or TABLE

EX: for Oracle: SEQUENCE

For MySQL: IDENTITY

Customized Generator: -

We can define one new generator which given number/String in our format.

Ex: OU-25-CS-20180330-STD0055

ST-EMP-558

ST-STD-658

Steps to Implments New Generator: -

#1 create new class with any name

#2 implement “IdentifierGenerator” interface given from hibernate (org.hibernate.id)

#3 override method “generate()” which returns “Serializable”

#4 define logic inside “generate()” method

#5 return format/value

#6 In model class use on Primary Key var
@GenericGenerator(strategy=”classname”)

Example:

Employee.java

```
@Entity
@Table(name="emptab")
public class Employee {

    @Id
    @Column(name="emp_Id")
```

```
	@GeneratedValue(generator="mygen")
	@GenericGenerator(name="mygen",strategy="com.sathyatech.model.MyGenerator")
	private String empId;
```

MyGenerator.java

```
public class MyGenerator implements IdentifierGenerator {

	@Override
	public Serializable generate(SessionImplementor arg0, Object arg1) throws HibernateException {

		String prefix = "ST-";
		String format = new SimpleDateFormat("yyyyMMdd").format(new Date());
		int random = Math.abs(new Random().nextInt());
		return prefix+format+"-"+random;
	}

}
```

Legacy Generators: -

Hibernate new versions still support Legecy (old) Generators. Those are:

increment = max+1

native = AUTO

hilo = TABLE

** These can be used in model class using @GenericGenerator annotation.

Example:

```
@Entity
@Table(name="emptab")
public class Employee {

    @Id
    @Column(name="emp_id")
    @GeneratedValue(generator="mygen")
    @GenericGenerator(name="mygen",strategy="increment")
    private int empId;
```

Bag And IdBag: -

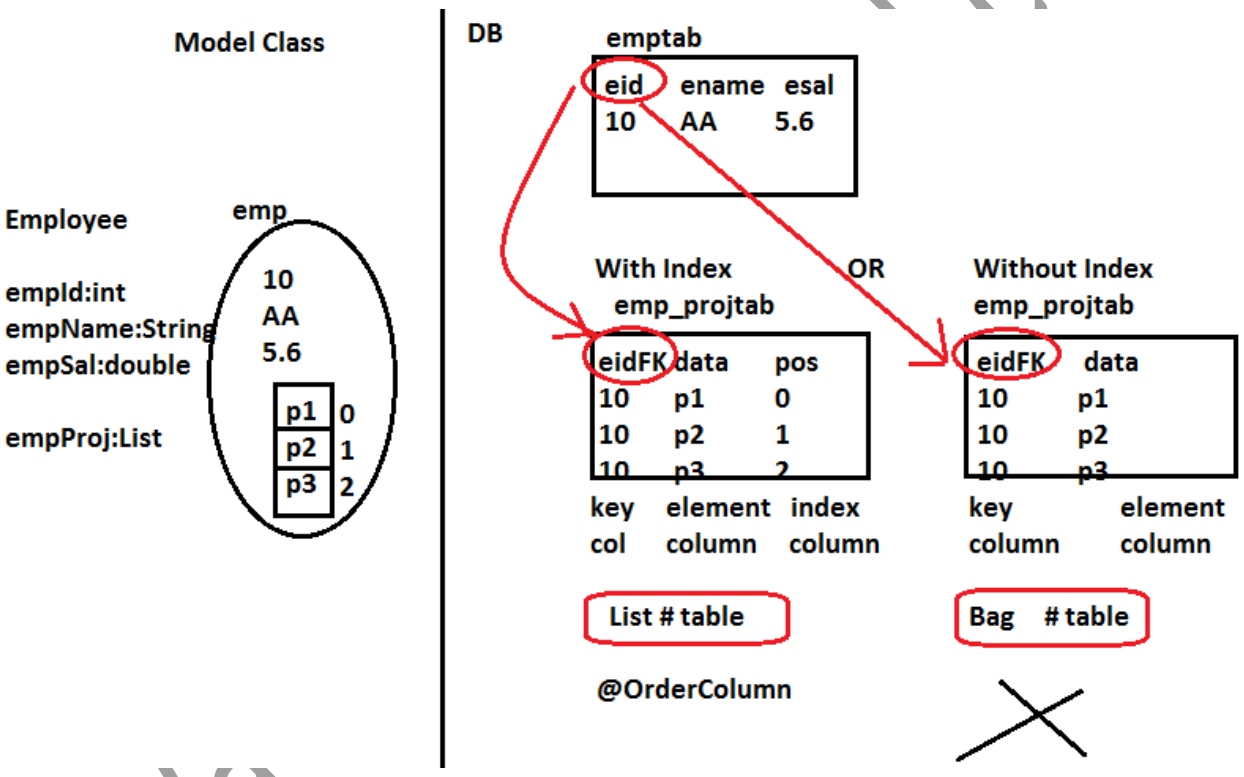
List is index based collection which store data as a child table with 3 columns.

Those are **Key Column, Element Column, Index Column.**

A **List table **without index** column is called as **Bag.**

** Here index means order (or position) of data in List.

** consider below example:



IdBag: -

It is a = List – index + ID Column

It maintains unique Identity Column in place of a simple index column, to identify every row uniquely.

These Id values are generated using one generator.

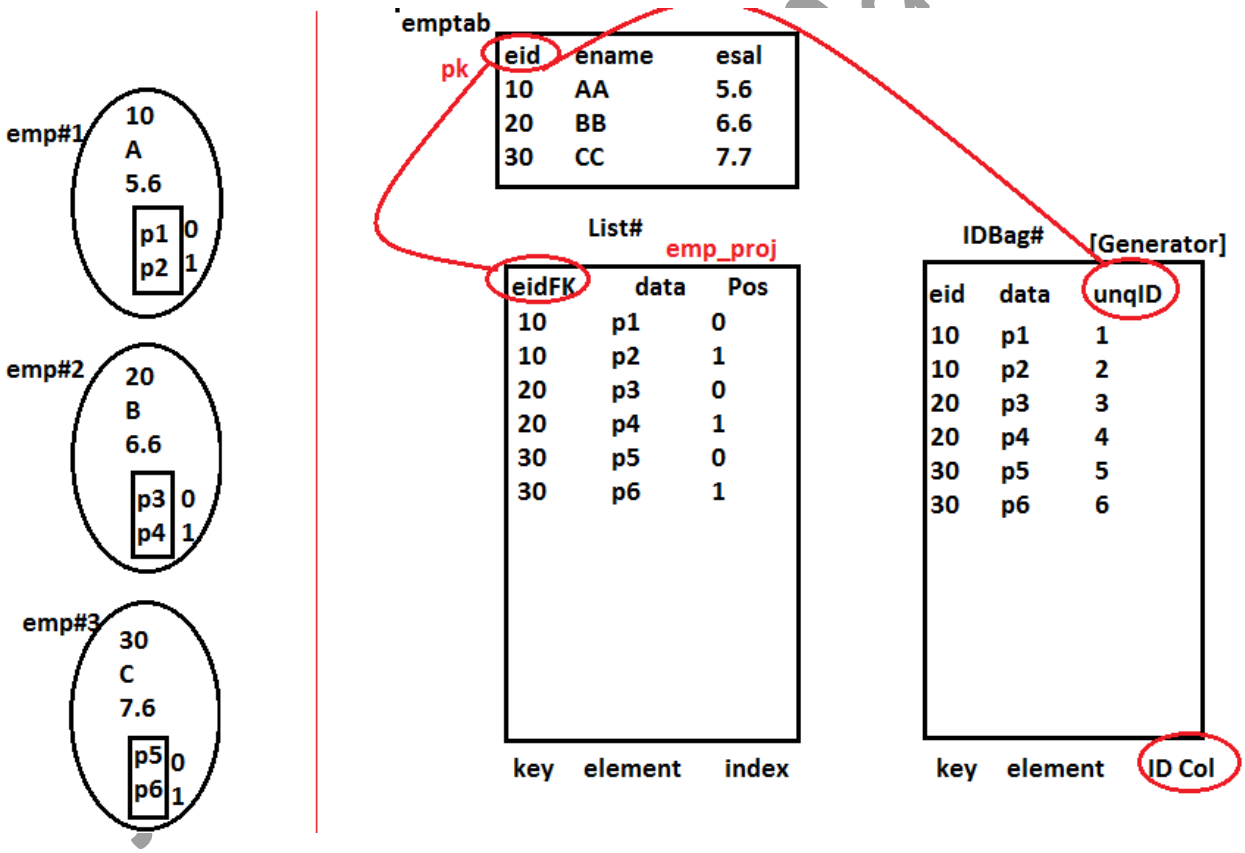
This extra column contains 3 params:

Name of the columns

Generator of the column

Data type of the column.

**Consider below example:



Example: ElementCollection: Bag

@Entity

```
@Table(name="emp_tab1")
public class Employee {

    @Id
    @Column(name="emp_Id")
    private int empId;

    @Column(name="emp_Name")
    private String empName;

    @Column(name="emp_Sal")
    private double empSal;

    @ElementCollection // child table-required
    @CollectionTable(  // optional
            name="emp_proj1",//table name
    /*Bag concept without @OrderColumn ie.Index Column.
    joinColumns=@JoinColumn(name="emp_Id")// key-column
    )
    //@OrderColumn(name="pos") //index-column
    @Column(name="data")    // element-column 1
    private List<String> projects;
```

```
/*@ElementCollection // child table-required
@CollectionTable(   // optional
                name="emp_models", //table name
                joinColumns=@JoinColumn(name="emp_Id") // key-column
                )
@MapKeyColumn(name="pos") //index-column
@Column(name="data")     // element-column
private Map<Integer,String> models;*/



public Employee() {
          System.out.println("Employee::0-param constructor");
   }


public int getEmpId() {
        return empId;
}
public void setEmpId(int empId) {
         this.empId = empId;
}
```

```
public String getEmpName() {
        return empName;
}

public void setEmpName(String empName) {
        this.empName = empName;
}

public double getEmpSal() {
        return empSal;
}

public void setEmpSal(double empSal) {
        this.empSal = empSal;
}

public List<String> getProjects() {
        return projects;
}

public void setProjects(List<String> projects) {
        this.projects = projects;
```

```
	}

	/*public Map<Integer, String> getModels() {
		return models;
	}

	public void setModels(Map<Integer, String> models) {
		this.models = models;
	}*/

	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName +
", empSal=" + empSal + ", projects=" + projects
				+ "]";
	}
}
```

Test.java

```
// Creating List Collection obj
		List<String> projects = new ArrayList<String>();
		projects.add("p1");
		projects.add("p2");
```

```
projects.add("p3");


//Creating Map Coll Obj

/*Map<Integer,String> model = new
LinkedHashMap<Integer,String>();

model.put(10, "A");

model.put(20, "B");

model.put(30, "C");

*/


// Creating Employee object and adding Collection obj

Employee emp = new Employee();

emp.setEmpId(100);

emp.setEmpName("RAVI");

emp.setEmpSal(7.77);

emp.setProjects(projects);

//emp.setModels(model);


//save Object

ses.save(emp);
```

Example: IdBag

Model Class

```
package com.sathyatech.model;


import java.util.List;

import java.util.Map;


import javax.persistence.CollectionTable;

import javax.persistence.Column;

import javax.persistence.ElementCollection;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.MapKeyColumn;

import javax.persistence.OrderColumn;

import javax.persistence.Table;


import org.hibernate.annotations.CollectionId;

import org.hibernate.annotations.GenericGenerator;

import org.hibernate.annotations.Type;



@Entity
```

```
@Table(name="emp_tab1")
@GenericGenerator(name="empGen",strategy="increment")
public class Employee {

    @Id
    @Column(name="emp_Id")
    private int empId;

    @Column(name="emp_Name")
    private String empName;

    @Column(name="emp_Sal")
    private double empSal;

    @ElementCollection // child table-required
    @CollectionTable(  // optional
            name="emp_proj1",//table name

    joinColumns=@JoinColumn(name="emp_Id")// key-column
    )
    @Column(name="data")    // element-column 1
    @CollectionId(columns=@Column(name="unqId"),
```

```
generator="empGen",
type=@Type(type="long")
)
private List<String> projects;

/*@ElementCollection // child table-required
@CollectionTable(   // optional
                name="emp_models", //table name
                joinColumns=@JoinColumn(name="emp_Id") // key-column
                )
@MapKeyColumn(name="pos") //index-column
@Column(name="data")    // element-column
private Map<Integer,String> models;*/


public Employee() {
        System.out.println("Employee::0-param constructor");
    }

public int getEmpId() {
        return empId;
}
```

```
public void setEmpId(int empId) {
        this.empId = empId;
}

public String getEmpName() {
        return empName;
}

public void setEmpName(String empName) {
        this.empName = empName;
}

public double getEmpSal() {
        return empSal;
}

public void setEmpSal(double empSal) {
        this.empSal = empSal;
}

public List<String> getProjects() {
        return projects;
```

```
	}

	public void setProjects(List<String> projects) {
		this.projects = projects;
	}

	/*public Map<Integer, String> getModels() {
		return models;
	}

	public void setModels(Map<Integer, String> models) {
		this.models = models;
	}*/

	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName +
", empSal=" + empSal + ", projects=" + projects
				+ "]";
	}
}
```

Test.java

```
// Creating List Collection obj
        List<String> projects = new ArrayList<String>();
        projects.add("p1");
        projects.add("p2");
        projects.add("p3");
        projects.add("p4");
        projects.add("p5");
        projects.add("p5");

        //Creating Map Coll Obj
        /*Map<Integer,String> model = new
LinkedHashMap<Integer,String>();
        model.put(10, "A");
        model.put(20, "B");
        model.put(30, "C");
        */

        // Creating Employee object and adding Collection obj
        Employee emp = new Employee();
        emp.setEmpId(100);
        emp.setEmpName("RAVI");
        emp.setEmpSal(7.77);
```

```
emp.setProjects(projects);


Employee emp1 = new Employee();
emp1.setEmpId(101);
emp1.setEmpName("Ram");
emp1.setEmpSal(8.77);
emp1.setProjects(projects);


Employee emp2 = new Employee();
emp2.setEmpId(102);
emp2.setEmpName("Sham");
emp2.setEmpSal(9.99);
emp2.setProjects(projects);
//emp.setModels(model);

//save Object
ses.save(emp);
ses.save(emp1);
ses.save(emp2);
```

Maven Tool:

It is used to maintain JAR's of the project.

JAR's are called as "dependencies" in Maven.

**If we write one maven application it contains "pom.xml" file.

POM->Project object model.

*pom file will be having details of dependencies(jars) with

groupId = Provider name

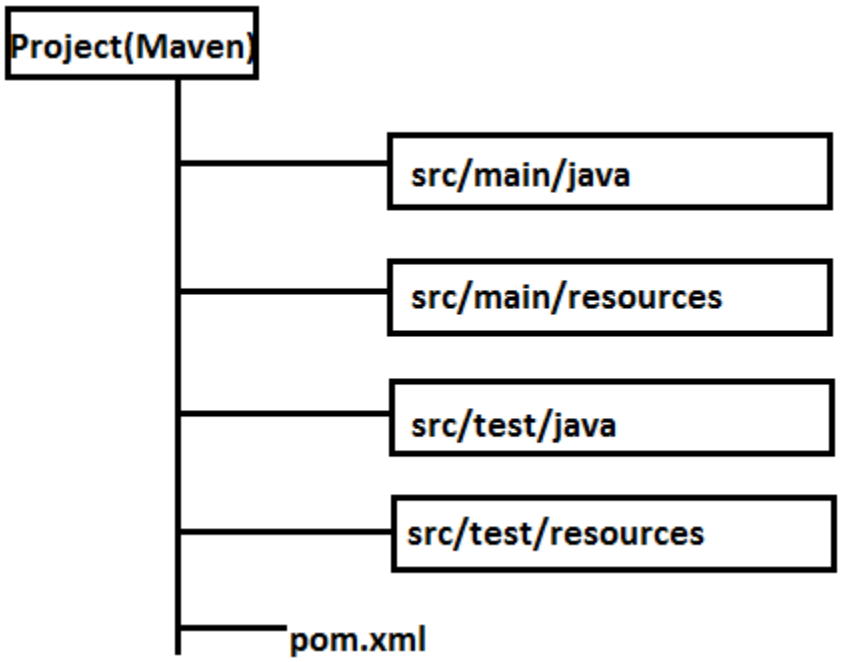
artifactId = jar name

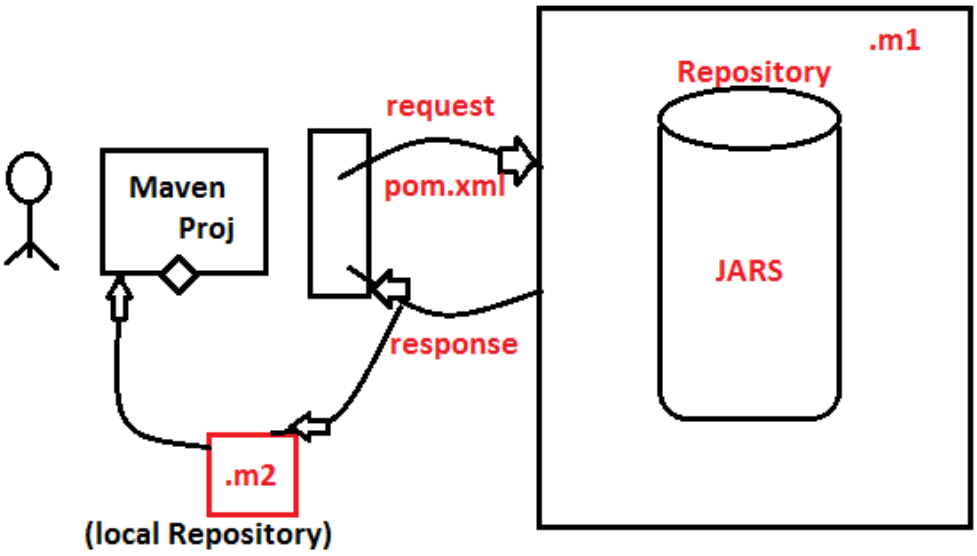
version = jar version.

*If pom.xml is updated then it is equals to making request to maven server.

*Maven server will check for dependencies in **maven Repository (.m1)** and returns jars as response.

These jars will be copied into **.m2** folder in our system then given to maven project. Here .m2 is called as "**local Repository**".





pom.xml

jar

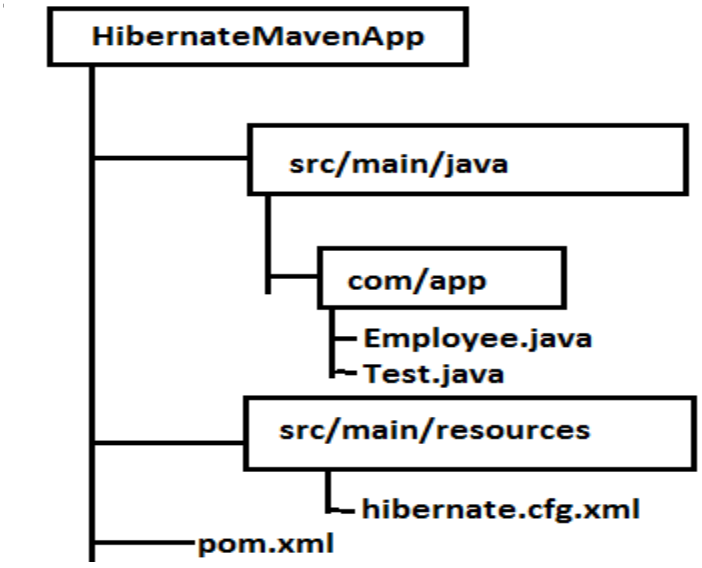
```
<dependency>
   <groupId></groupId>
   <artifactId></artifactId>
   <version></version>
</dependency>
```

groupId = Provider name

artifactId = jar name

version = jar version.

Hibernate Maven Project:



#2 Create Maven Simple project in Eclipse, (Hint: Download Eclipse Java EE)

>File > new > other > Search and select “Maven Project” > next

> click on checkbox “create simple project” > next >

Enter groupId : ramssoft

artifactId : HibernateApp

version : 1.0

>Finish.

#3 Update pom.xml code under <project>> double click on “pom.xml”

>open in pom.xml mode

> write below dependencies in pom.xml

--pom.xml—

<project>

…….

```
<dependencies>
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-core</artifactId>
<version>5.1.6</version>
</dependency>
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>5.1.6</version>
</dependency>
</dependencies>
```

#4 Write Code

1) Model and Test class in src/main/java
2) Hibernate.cfg.xml in src/main/resources

HibernateValidatorAPI: -

Validations are used to check data, before processing operation.

Ex: Is EmpName valid? It should not be null, should have 4-6 chars only.

*Validations can be done using:

1] Scripting (Java Script/JQuery/AJS)

2] Spring Form Validation

3] Bootstrap Validation API

In same way “Hibernate Validation API”.

This API will not be given as default in Hibernate CORE F/W.

*It must be taken as Add-On (extension)

---pom.xml—

```
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-validator</artifactId>
<version>4.3.2.Final</version>
</dependency>
```

String Type Validations In Hibernate: -

I] not null check

Ii] size check

Iii] pattern/format check

Validations can be applied on String DataType in Model class.

Ex:

1] not null check

```
@NotNull(message=”Enter Name..”)
@Column(name=”ename”)
private String empName;
```

2] size check (min/max)

@Size(min=5,max=8,message=”Name must be 5-8 chars only”)

@Column(name=”ename”)

private String empName;

3] Format(Pattern) check

@Pattern(regexp=”[a-z]{2,6}”),message=”Enter 2-6 chars lower chars only”)

@Column(name=”ename”)

Private String empName;

***These validations annotations are defined in package:

“javax.validation.contraints”.

*Here Assert means “expected”.

Ex: AssertTrue = expecting value is true.

**message property for every validation annotation is optional, it gives default message.

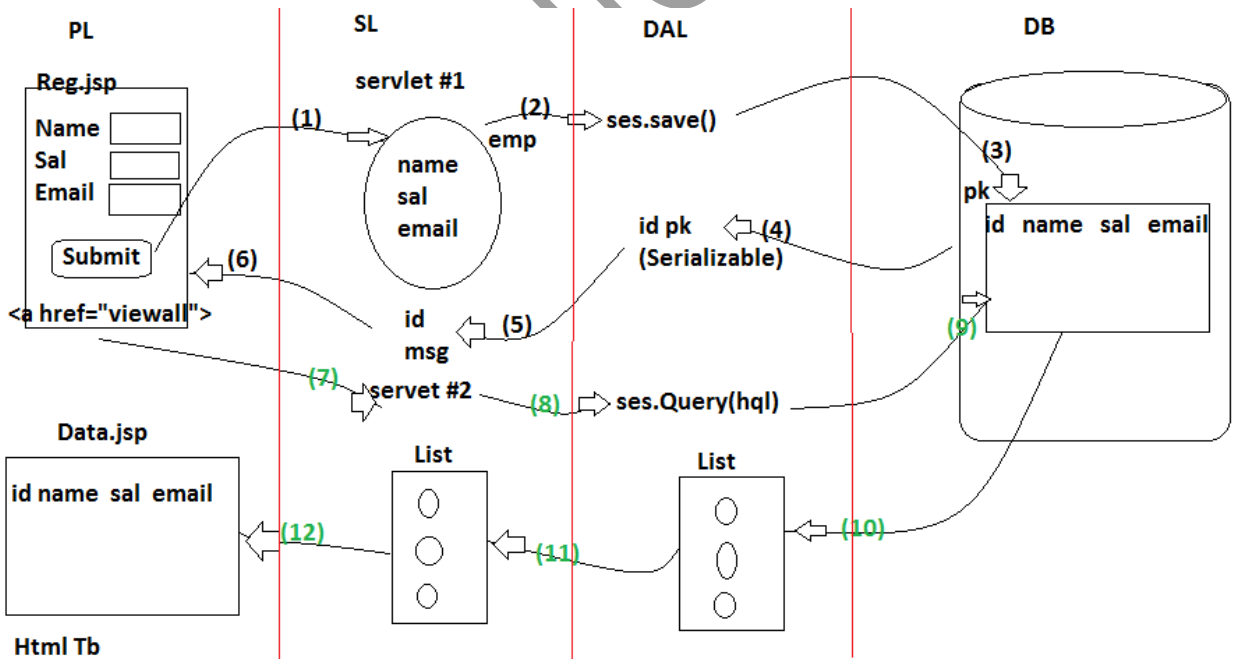
**If any validation is failed then hibernate throws,”ConstraitsVoilationException”.

Example:

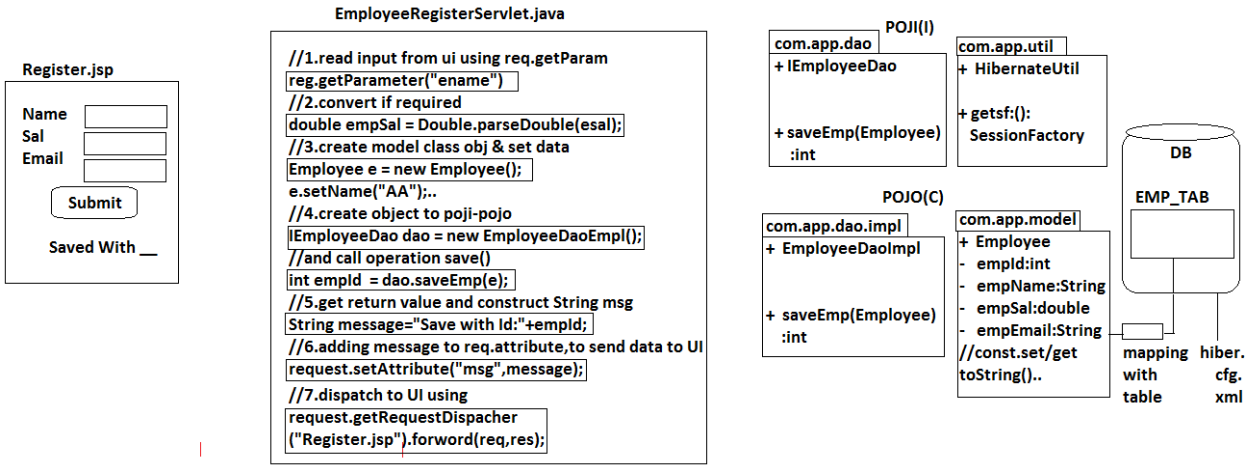
```
@Entity
@Table(name="emptab")
public class Employee {

    @Id
    @Column(name="emp_Id")
    private int empId;
```

```
@Column(name="emp_Name")
@Pattern(regexp="[a-z] {2,6}",message="Enter only lowe 2-6 chars only!")
private String empName;

@Column(name="emp_Sal")
@Min(value=6,message="Salary must be 6 or higher!")
private double empSal;

@AssertTrue(message="Employee must be activated before register")
@Column(name="status")
private boolean status;

@Past
@NotNull
@Column(name="dob")
private Date dob;
```

JSP SERVLET HIBERNATE INTEGRATION PROJECT WITH PAGINATION: -



Design#1 Register Employee:

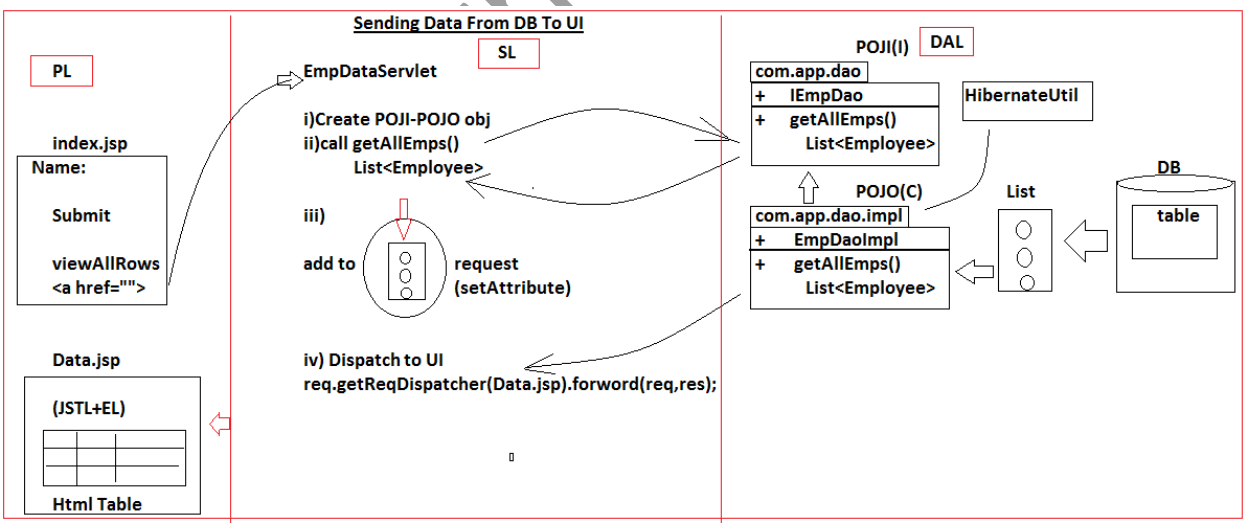


Coding Order:

1]Model class 2]hibernate.cfg.xml 3]HibernateUtil 4]POJI [IEmployeeDao] 5]POJO [EmployeeDaoImpl] 6] EmployeeRegisterServlet.java 7]index.jsp 8] EmployeeDataServlet.jsp 9]Data.jsp

** HibernateUtil creates singleton pattern for sessionFactory object.

Design #2:



JSTL Code For ForEach Loop:

Java Code:

List<Employee> emps = …

For(Employee e:emps){Sysout(e);}

##JSTL Code

<c:forEach items="${collName}" var="vn">

<c:out value="${value}">

</c:forEach>

##JSTL Equal code

<c:forEach items="${emps}" var="e">

<c:out value="${e}">

</c:forEach>

Pagination Code:

Basic info:

Int pageNumber(pn)=1;

Int totalRows(tr)=select count(empId) from Employee;

Int pageSize(ps)=3;

Then no of pages(np)

Int noOfPages(np)=tr/ps + (tr%ps>0?1:0)

--→In Dao use methods

q.setFirstResult((pn-1)*ps)

q.setMaxResult(ps);

-→In UI Pagination numbers code:

<%

Int count = (Integer) request.getAttribute("np");

```
For(int i=1;i<=count;i++){

Out.println("<a href='?Page="+i+"'> "+i+" </a>  ");

}

%>
```