POJO: [plain old java object] A pojo class is a class which should not extend any predefined class belongs to specific API and it should not implement any predefined interface belongs to specific api.

Following are pojo classes:

```
class C1
{
}
class C2 extends C1
{
}
class C3 extends java.lang.Thread
{
}
```

Note: Thread is a predefined class but its not confined for development of any particular kind of application. In another words Thread class can be used for any kind of java application. So our class C3 is a pojo class.

```
class C4 implements java.lang.Runnable
{
}
```

Note: Runnable is an interface which can be used to create a thread. So it can be used for development of any kind of java application.

```
class C5 implements java.io.Serializable
{
}
```

Note: Serializable interface implementation class objects will become serializable objects. So that we can transfer from one location to another location over the network.

Following are not POJO classes:

```
class C6 extends GenericServlet
{
}
```

Note: GenericServlet class is used only for the purpose of web application development. So our class is not a pojo class.

```
class C7 implements Servlet
{
}
```

Note: Our class is implementing Servlet interface which is confined for development of web applications.

```
class C8 implements java.rmi.Remote
{
}
```

Note: Remote interface is used only for development of Distributed Applications. So the above class is not a pojo class.

POJI Interfaces: The interfaces that are not API dependent are called as POJI [Plain Old Java Interface]

Composite Primary Key: If more than one member variable of POJO class is configured as primary key field then it is called as Composite Primary Key. To configure this we use <composite-id> tag in mapping file.

Note: Whether the database table have primary key or not still primary key field configuration on POJO class member variables hibernate mapping file is mandatory.

In a database table if primary key is not specified then we can prefer unique key field to configure as a primary key in hibernate mapping file.

If database team has given a table without constraints then it is recommended to configure multiple member variables of POJO class as a primary key filed (Composite primary key field)

States of Persistent class object [POJO class object]: There are 3 states which are as follows:

1. **Transient state:**
   a. Object is created by programmer with data. But it doesn't represent any table row.
   b. This object doesn't contain primary key value.
   c. The object which is created for POJO class and which is not under control of hibernate application resides in transient state.
2. **Persistent state:**
   a. The object that represents table row with primary key and managed under the control of hibernate software is called as persistent object.
   b. This object will be in synchronization with table row.
   c. Hibernate application developer's uses this kind of object in persistent logic development.
3. **Detached state:**
   a. When session is closed automatically persistent context will be destroyed and all the objects which are in persistent state will be thrown to detached state.
   b. When table row of persistence state object is deleted then object becomes detached object.

Hibernate application:

- It uses hibernate setup to interact with database software and this application is a client to database software.
- It uses two important objects to represent hibernate software. They are:
  - SessionFactory
  - Session

Note: Hibernate **Session** object is no way related with HttpSession object of Servlet API.

SessionFactory:

- It's an object of a class that implements org.hibernate.SessionFactory interface.
- It represents connection pool.
- Using this we can get session objects.
- Its a thread safe object.

Session:

- Each session object of SessionFactory pool is constructed by encapsulating JDBC connection and statement objects.
- Programmer uses Session object to interact with database software.
- Session object is not a thread safe object.
- Java Hibernate application can have multiple Hibernate Session objects.

Step3: Develop POJO class/persistent class

Employee.java

```
public class Employee implements java.io.Serializable
{
	int no;
	String fname, lname, email;

	public void setNo(int no)
	{
		this.no = no;
	}
	public int getNo()
	{
		return no;
	}
	public void setFname(String fname)
	{
		this.fname = fname;
	}
	public String getFname()
	{
		return fname;
	}
	public void setLname(String lname)
	{
		this.lname = lname;
	}
	public String getLname()
	{
		return lname;
	}
	public void setEmail(String email)
	{
		this.email = email;
	}
	public String getEmail()
	{
		return email;
	}
}
```

```
        jt1.setJob("programmer");
        jt1.setSalary(12000);
        jt1.setDepartment(101);

        Person p1= new Person();

        p1.setPname("Sai");
        p1.setPjob(jt1);

        ses.save(p1);
        tx.commit();
    System.out.println("records  are inserted successfully");
}//try
catch(HibernateException he)
{
        he.printStackTrace();
 }
```

To read record:

```
        Person p = (Person) ses.get(Person.class, new Integer(41));

        JobType jt = p.getPjob();

        System.out.println(p.getPid()+"  "+p.getPname()+"  "+jt.getJob()+"  "+jt.getSalary()+"
"+jt.getDepartment());
```


Algorithms:

1. In our programs till now we have given primary key values manually for our pojo class objects. But in real time we will make primary key values to be generated automatically. So that in order to generate primary key values automatically we are using algorithms.
2. To work with algorithms we need to do changes in mapping file and also in a client program.
3. Hibernate supplies lot of predefined algorithms as identity value generators for pojo class objects.
4. These algorithms are pre defined classes supplied by hibernate API implementing org.hibernate.Id.IdentifierGenerator.
5. To specify these algorithms use <generator> tag which is a sub tag of <id> tag in hibernate mapping file.
6. All these algorithm classes also have nick names or short names to use in mapping file. We have many algorithms which are as follows:

Short form	**Algorithm Class name**
• assigned (default)	org.hibernate.id.Assigned
• increment	org.hibernate.id.IncrementGenerator
• identity	org.hibernate.id.IdentityGenerator

Client.java

```
FileInputStream fis = new FileInputStream("data.properties");
Properties p = new Properties();
p.load(fis);
//activate hibernate software
Configuration cfg = new Configuration();
cfg.setProperties(p);

cfg.addFile("student.hbm.xml");
```

Note: In these both styles we have drawbacks. We cannot configure additional properties like **show_sql.**

Immutable SessionFactory object: After creating SessionFactory object if we modify hibernate configuration properties dynamically at run time of the client program then still the SessionFactory will contain old values and modified values will not be reflected in it.

When ever we modify hibernate-configuration file then we need to save & close it. Then we need to restart the application if the application is already running. Then itself application can create a new SessionFactory object after reading the modified configuration file.

Tools: By using tools we can create table, update table and we can also create and drop a table at run time of program based up on details given in mapping file but command should be given in configuration file.

1. **create:**
   a. It can be used to create table
   b. If table already exists with specified name then it will be dropped and a new table will be created with same name.

Steps:

a. Add the following tag in configuration file

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

b. Mapping file

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">




<hibernate-mapping>
    <class name="StudentBean" table="students">
        <id name="sid" length = "10" type = "int" />
        <property name="sname" length = "20"  type = "string"/>
```

Ex4:
Sql>delete from employee where firstname in ('Sai', 'Kanakadhar');
Hql>delete from EmpBean as eb where eb.firstname in ('Sai', 'Kanakadhar')

- To execute select queries of hql use list() or iterate() on query object.
- To select non-select statement hql queries call executeUpdate() on query object.
- Execution of hql query is nothing but converting hql query to the underlying database s/w specific sql query & sending that sql query to database s/w for execution.
- Hibernate 3.x software internally uses **AST query translator** factory to convert hql queries into database specific s/w equivalent sql queries.

list() vs iterate():

list() generates results by selecting all the records through the execution of a single select sql query [no lazy loading]

iterate() selects the records from database table by executing a separate query to read each record and a separate query to read all id values. [Lazy loading]

Eg: If we read 10 records from table then list() generates one query where as iterate() generates 11 queries [one for id values & 10 queries for 10 records].

Note:
1. We can see difference between list() and iterate() only when hql select queries are selecting all the columns of a table.
2. When iterate() is used to select specific columns values then lazy loading doesn't takes place.

Sql select queries of jdbc *vs* hql select queries of hibernate:
Jdbc code based select queries execution gives ResultSet object which is not serializable object. So we cannot send ResultSet object over the network.

Hibernate based select hql queries execution gives results in the form of collection framework **List** data structure. Since this **List** data structure is serializable object by default we can send it through network.

Note: All collection frame work data structures are serializable by default.
Note: In order to make pojo class object as serializable object, pojo class should implement java.io.Serializable interface.

Input values to hql queries:
To make hql queries flexible by setting input values of query from outside the query and to set query input values without bothering about database s/w, we can pass parameters to hql queries in java style.

Parameters in HQL queries:
1. Positional parameters (?)
2. Named Parameters (**:<name>**) [Recommended to use]

Example: Insertion of records from one table to another table.

Note: Here we need two pojo classes. One for source table and another for destination table. Here StuBean is for destination and StudentBean is for source. Data types of corresponding columns of both the tables should be same.

```
String qry = "insert into StuBean (sid1, sname1, tot_m1) select st.sid, st.sname, st.tot_m
from StudentBean as st where st.tot_m >= :tm";
Query q1 = ses.createQuery(qry);
//setting values for marks
q1.setFloat("tm", 98.0f);
int count = q1.executeUpdate();
System.out.println("No of records inserted into stu table are: "+count);
```


Criteria API: Using criteria api we can read multiple records at a time with a single method call. Here mainly we can work with two objects. They are:

1. **Criteria:** Using its object we can configure two things:
   a. Bean class name [by using which we have to retrieve records]
   b. Adding criterion objects.
2. **Criterion:** Criterion object can be created in many ways. We can work with static methods of following classes:
   a. **Restrictions:** From this class we can call static methods as follows:
      i. ge() [greater than or equal to]
      ii. ne() [not equal to]
      iii. lt() [less than]
      iv. gt() [greater than]
      v. le() [less than or equal to]
      vi. like() [here we can use wild card characters like "%" and "_" [under score]
      vii. and() [to retrieve records which satisfies more than one condition]
      viii. or()[to retrieve records which satisfies any one of two conditions]

Note: The above Restruictions class methods works based up on pojo class properties to specify conditions.

In order to centralize certain persistence logic or business logic for multiple modules of a project or multiple projects of a company then place that logic in database s/w as pl/sql procedure or function. Hibernate persistence logic can call pl/sql procedure or function of database s/w from client application only when they are developed with compatibility to hibernate.
These compatibility rules of developing stored procedures or functions differ from one to other database.
A cursor in pl/sql programming of oracle is a data type whose variable can store zero or any no. of selected records from database tables.

Rules & Limitations:

1. Pagination is not possible on results generated by queries of hibernate specific pl/sql procedures that means we cannot use setMaxResults() & setFirstResults() in this environment.

We must follow the following syntax to call the procedure:

{call procedure-name(<parameters>)}

We must follow the following syntax to call pl/sql function:

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

Note: pl/sql procedure does not return a value where as pl/sql function returns a value.

Rules specific to oracle to design stored procedure or function compatible with hibernate:

1. pl/sql procedures first parameter must be an out parameter returning resultset (cursor having records).
2. pl/sql function must return a result set [cursor having records]

Note: ***sys_refcursor*** is cursor type given by pl/sql programming of oracle having the ability to store selected records (one or more) like result set object of jdbc programming.

Note: For rules specific to Sybase, ms-sql server, database software's refer chapter 16 of pdf file.

Note: Hibernate application uses native sql programming to call pl/sql procedures or functions. Calling pl/sql procedure or function which is not compatible wit hibernate is possible indirectly from hibernate persistence logic by injecting jdbc code.

Limitations of hibernate s/w:

1. Since hibernate is not a distributed technology, hibernate persistence logic is not distributed persistence logic. So this persistence logic can't be used from remote clients directly.
2. Calling pl/sql procedures or functions from hibernate persistence logic is quite complex. More over these pl/sql function or procedure must be written in a compatible form of hibernate.
3. Pagination is not possible on result set generated by pl/sql procedure or function.
4. hibernate can't be used to interact with **non conventional** databases like text file, ms-excel etc.., [jdbc can do this]
5. Hibernate can't interact with few **conventional** databases like ms. Access.

Examples:

1. **Eg: Entity native sql query or select query that selects all columns values.**

```
2638
2639                q1.addScalar("sname1", Hibernate.STRING);
2640
2641                //Execution of native sql query
2642
2643                List l = q1.list();
2644
2645                System.out.println("Records are: ");
2646
2647                for( int i = 0; i < l.size() ; i++) //for each row
2648                {
2649                        Object row[] = (Object[]) l.get(i);
2650                        for( int j = 0; j < row.length ; j++) //for each column
2651                        {
2652                                System.out.print(row[j].toString()+" ");
2653                        }
2654
2655                        System.out.println();
2656                }
2657
```

6. Named Queries

a. Named scalar

In mapping file

```
2662 <hibernate-mapping>
2663         <class name="StudentBean" table="student">
2664                 <id name="sid" column = "sid"/>
2665                 <property name="sname" column="sname"/>
2666                 <property name="tot_m" column = "tot_m"/>
2667         </class>
2668         <sql-query name = "qry">
2669                 <return-scalar column = "sname" type = "string"/>
2670                 select sname from student where sname like ?
2671         </sql-query>
2672 </hibernate-mapping>
```

In client program:

```
2674                Query q1 = ses.getNamedQuery("qry");
2675
2676                q1.setString(0, "K%");
2677
2678                //Execution of native sql query
2679                List l = q1.list();
2680
2681                System.out.println("Records are: "+l.toString());
2682
2683
2684
```

b. Named Entity

In mapping file:

```
2687         <sql-query name = "qry">
```

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

Main xml file:

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
        <session-factory>
                <property
name="hibernate.connection.providerclass">org.hibernate.connection.ProxoolConnectionProvider</pr
operty>
                <property name="hibernate.proxool.pool_alias">ourpool</property>

                <property name="hibernate.proxool.xml">ourfile.xml</property>
                <property name="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</property>
                <property name="show_sql">true</property>
                <mapping resource="student.hbm.xml"/>
        </session-factory>
</hibernate-configuration>
```

Inheritance:

Hibernate persistence classes are pojo classes. So it can participate in inheritance to give the advantage of reusability and extensibility.

When classes hibernate is there in inheritance the database tables related to these persistence classes will also maintain the data with relationship. We need to configure relationships in mapping file based up on inheritance mapping concepts.

Inheritance between two servlet components is possible but inheritance between two ejb components is not possible.

Inheritance types: Hibernate s/w gives 3 approaches of performing inheritance mapping to get reusability, extensibility advantages of inheritance in different ways. They are:

1. Table for class hierarchy

Note: While working with multiple tables it is recommended not to use reverse engineering process of my-eclipse ide.

Table for concrete class hierarchy:
In this inheritance mapping even though pojo classes are there in inheritance their tables will not participate in relationship and every pojo class of inheritance hierarchy will use one separate table without having relationship with other table.

These pojo classes will be configured in hibernate mapping file as individual, independent, concrete classes without showing inheritance and without showing reusability of properties configuration.

Inheritance is mandatory. Through this retrieval of records from tables works properly. This is not industrial standard.

Examples:

1. **Table for class hierarchy**

Oracle table creation:

```
create table persons
(
        id number(5) primary key,
        name varchar2(20),
        company  varchar2(20),
        salary number(8,2),
        department  number(4),
        address varchar2(30),
        person_type  varchar2(5)
);
```

Mapping file:

```
<hibernate-mapping>
        <class name="Person"  table="persons"  discriminator-value="per" >
                 <id  name="id" />
                 <discriminator  column="person_type" type="string"  length="5" />
                  <property  name="name" />
                  <property  name="company" />
                <subclass  name="Employee"  discriminator-value="emp">
                          <property  name="salary" />
                          <property  name="department" />
                </subclass>
                 <subclass name="Customer" discriminator-value="cus">
                        <property  name="address" />
                </subclass>
        </class>
</hibernate-mapping>
```

Base pojo classs:

```
public class Person
{     private  int id;
```

```
3087            private String name,company;
3088
3089            public void setId(int n)
3090            {
3091                    id=n;
3092            }
3093            public int getId()
3094            {
3095                    return id;
3096            }
3097            public void setName(String s)
3098            {
3099                    name=s;
3100            }
3101            public String getName()
3102            {
3103                    return name;
3104            }
3105            public void setCompany(String s)
3106            {
3107                    company=s;
3108            }
3109            public String getCompany()
3110            {
3111                    return company;
3112            }
3113    }
```

Derived POJO class1:

```
3114    public class Employee  extends Person
3115    {
3116            private double salary;
3117            private int department;
3118
3119            public void setSalary(double d)
3120            {
3121                    salary=d;
3122            }
3123            public double getSalary()
3124            {
3125                    return salary;
3126            }
3127            public void setDepartment(int n)
3128            {
3129                    department=n;
3130            }
3131
3132
3133            public int getDepartment()
3134            {
3135                    return department;
3136
```

The process of combining a set of related operations into single unit and executing them by applying logic of do every thing or do nothing principle is called as transaction management.

While executing sensitive business logic and persistence logic we must deal with transaction management.

E.g.: Transfer money operation is composed with two operations.

1. Withdraw amount from source a/c.
2. Deposit amount in destination account.

So we need to execute these two operations by applying do every thing or nothing principle through transaction management.

Transaction management is applied on the code applies ACID properties support on database software.

A – Atomicity
C – Consistency
I – Isolation
D – Durability

Atomicity: The process of combining related sub operations into single unit is called atomicity.

Consistency: The process of getting guarantee that the rules kept on database s/w like balance must not be negative are not violated by end of the transaction is called as consistency.

Isolation: The process of getting concurrent operations from database s/w from multiple users and applications by applying logs is called as isolation.

Durability: The ability of bringing database s/w back to normal state by using log files and backup files when database is crashed and using database data for long time is called as durability.

Architecture Transaction management:

The application or component on which transaction management is enabled is called as transactional application or component.

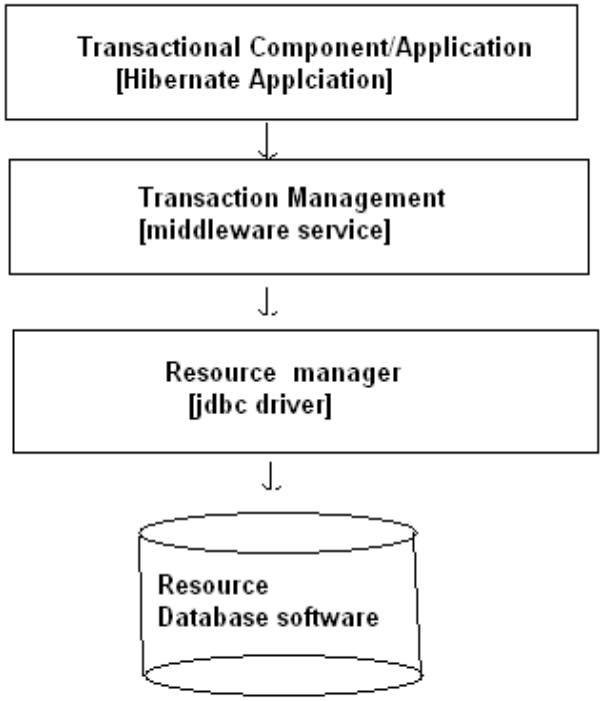
Transaction manager is responsible to begin transaction and to commit or rollback the transaction on the application code.

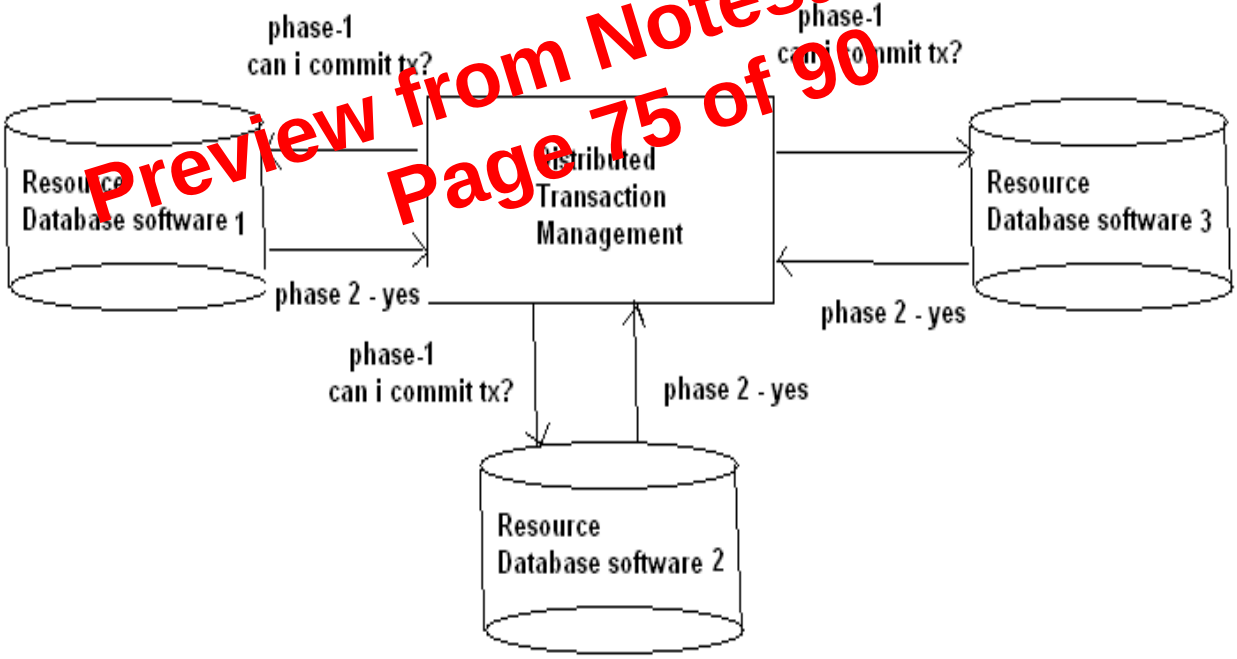
Based on no. of resources (database software's) that are involved there are two types of transactions.

1. **Local transaction:** All the operations of application code on which transaction management is enabled will deal with single resource [database s/w]
   Eg: Transfer money operation between two accounts of same bank.
2. **Distributed Transaction:** If multiple resources or database software's are involved for various operations of application code on which transaction management is called as distributed transaction.
   Eg: Transfer money operation between two accounts of two different banks.

Distributed transaction runs based on 2pc [two phase commit] protocol.

Fig: Architecture of transaction management





According to two pc (2 phase commit) protocol in phase 1 the distributed transaction manager asks all database software's permission to commit the transaction.

In phase 2 if all database software's gives permission to commit the transaction then it will be committed otherwise the distributed transaction will rollback.

Hibernate supports local transaction management but does not supports distributed transaction management.

OSCache (Open Symphony Cache) (org.hibernate.cache.OSCacheProvider)

1. It is a powerful .
2. flexible package
3. supports read-only and read/write caching.
4. Supports memory- based and disk-based caching.
5. Provides basic support for clustering via either JavaGroups or JMS.

Level 1 Cache: will be default in every hibernate program. When ever we get a session object from session-factory immediately it creates a persistent context and maintains all records. When ever we try to retrieve the same record more than one time then only once it gives a request to database server and gives you the same record for every request. So we can reduce no of database hits.

Note: When ever we need to remove any persistent state object explicitly from our level 1 cache then we can call evict(_) method belongs to session object

Example:

```
StudentBean st1;
st1 = (StudentBean) ses1.get(StudentBean.class, new Integer(105));
System.out.println(st1.getSid()+" "+st1.getSname()+" "+st1.getTot_m());

st1 = (StudentBean) ses1.get(StudentBean.class, new Integer(105));
System.out.println("Record Values r: ");
System.out.println(st1.getSid()+" "+st1.getSname()+" "+st1.getTot_m());
```

Note: Here we are reading the same record i.e. 105 for two times. But only one query will be displayed. It means internally it is fetching data from level 1 cache.

sess.evict():

```
StudentBean st1;
st1 = (StudentBean) ses1.get(StudentBean.class, new Integer(105));
System.out.println(st1.getSid()+" "+st1.getSname()+" "+st1.getTot_m());

ses1.evict(st1);

st1 = (StudentBean) ses1.get(StudentBean.class, new Integer(105));
System.out.println("Record Values r: ");
System.out.println(st1.getSid()+" "+st1.getSname()+" "+st1.getTot_m());
```

Note: In the above code snippet as we have called evict() method student object "st1" will be removed from level 1 cache. So that this time it will generate two queries to retrieve the same record.