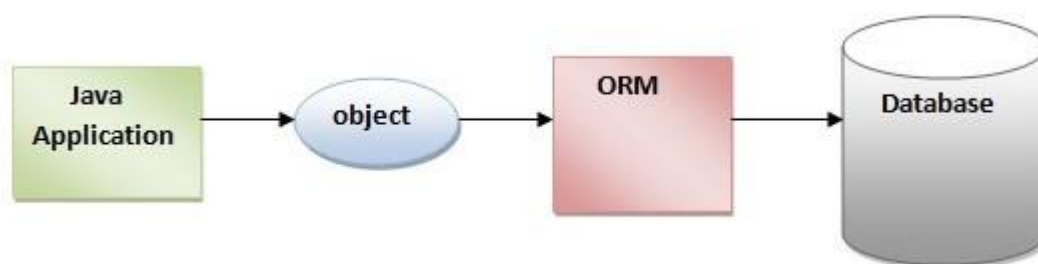


# Hibernate Framework

Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.

## ORM Tool

An ORM tool simplifies the data creation, data manipulation and data access. It is a programming technique that maps the object to the data stored in the database.



The ORM tool internally uses the JDBC API to interact with the database.

## What is JPA?

Java Persistence API (JPA) is a Java specification that provides certain functionality and standard to ORM tools. The **javax.persistence** package contains the JPA classes and interfaces.

## Advantages of Hibernate Framework

Following are the advantages of hibernate framework:

### 1) Open Source and Lightweight

Hibernate framework is open source under the LGPL license and lightweight.

### 2) Fast Performance

The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled by default.

### 3) Database Independent Query

HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, if database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.

#### 4) Automatic Table Creation

Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.

#### 5) Simplifies Complex Join

Fetching data from multiple tables is easy in hibernate framework.

#### 6) Provides Query Statistics and Database Status

Hibernate supports Query cache and provide statistics about query and database status.

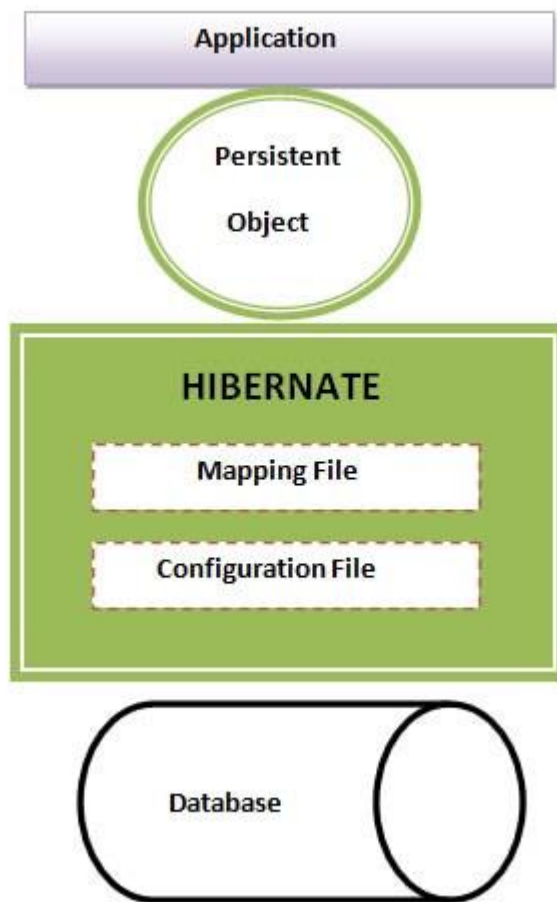
## Hibernate Architecture

The Hibernate architecture includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.

The Hibernate architecture is categorized in four layers.

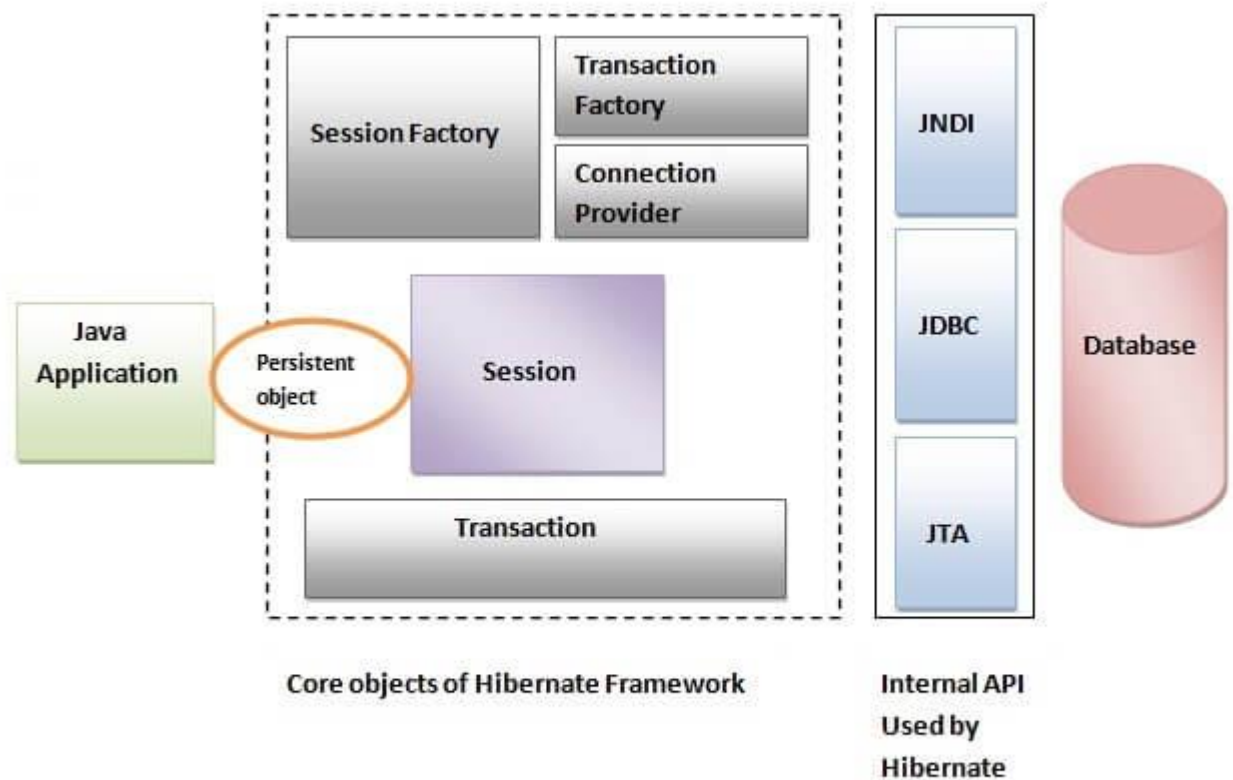
- Java application layer
- Hibernate framework layer
- Backhand api layer
- Database layer

Let's see the diagram of hibernate architecture:



This is the high level architecture of Hibernate with mapping file and configuration file.

---



Hibernate framework uses many objects such as session factory, session, transaction etc. alongwith existing Java API such as JDBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface).

## Elements of Hibernate Architecture

For creating the first hibernate application, we must know the elements of Hibernate architecture. They follows:

### *SessionFactory*

The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data. The `org.hibernate.SessionFactory` interface provides factory method to get the object of Session.

### *Session*

The session object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data. The

org.hibernate.Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

### *Transaction*

The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.

### *ConnectionProvider*

It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.

### *TransactionFactory*

It is a factory of Transaction. It is optional.

## First Hibernate Example

Here, we are going to create the first hibernate application without IDE. For creating the first hibernate application, we need to follow the following steps:

1. Create the Persistent class
2. Create the mapping file for Persistent class
3. Create the Configuration file
4. Create the class that retrieves or stores the persistent object
5. Load the jar file
6. Run the first hibernate application by using command prompt

### 1) Create the Persistent class

A simple Persistent class should follow some rules:

- **A no-arg constructor:** It is recommended that you have a default constructor at least package visibility so that hibernate can create the instance of the Persistent class by newInstance() method.
- **Provide an identifier property:** It is better to assign an attribute as id. This attribute behaves as a primary key in database.
- **Declare getter and setter methods:** The Hibernate recognizes the method by getter and setter method names by default.

- **Prefer non-final class:** Hibernate uses the concept of proxies, that depends on the persistent class. The application programmer will not be able to use proxies for lazy association fetching.

Pom.xml

```
-----

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com</groupId>
  <artifactId>hiber</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>hiber</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>

    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>5.6.5.Final</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.29</version>
    </dependency>

    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Let's create the simple Persistent class:

### *Employee.java*

```
package com.java.mypackage;
```

```
public class Employee {
```

```
  private int id;
```

```
  private String firstName,lastName;
```

```
  public int getId() {
```

```
    return id;
```

```
  }
```

```
  public void setId(int id) {
```

```
    this.id = id;
```

```
  }
```

```
  public String getFirstName() {
```

```
    return firstName;
```

```
  }
```

```
  public void setFirstName(String firstName) {
```

```
    this.firstName = firstName;
```

```
  }
```

```
  public String getLastName() {
```

```
    return lastName;
```

```
  }
```

```
  public void setLastName(String lastName) {
```

```
    this.lastName = lastName;
```

```
  }
```

```
}
```

---

## 2) Create the mapping file for Persistent class

The mapping file name conventionally, should be class\_name.hbm.xml. There are many elements of the mapping file.

- **hibernate-mapping** : It is the root element in the mapping file that contains all the mapping elements.

- **class** : It is the sub-element of the hibernate-mapping element. It specifies the Persistent class.
- **id** : It is the subelement of class. It specifies the primary key attribute in the class.
- **generator** : It is the sub-element of id. It is used to generate the primary key. There are many generator classes such as assigned, increment, hilo, sequence, native etc. We will learn all the generator classes later.
- **property** : It is the sub-element of class that specifies the property name of the Persistent class.

Let's see the mapping file for the Employee class:

### *employee.hbm.xml*

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 5.3//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
  <class name="com.java.mypackage.Employee" table="emp1000">
    <id name="id">
      <generator class="assigned"> </generator>
    </id>

    <property name="firstName"> </property>
    <property name="lastName"> </property>

  </class>

</hibernate-mapping>
```

---

## 3) Create the Configuration file

The configuration file contains information about the database and mapping file. Conventionally, its name should be hibernate.cfg.xml .

### *hibernate.cfg.xml*



```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping resource="employee.hbm.xml"/>
    </session-factory>

</hibernate-configuration>
```

---

## 4) Create the class that retrieves or stores the object

In this class, we are simply storing the employee object to the database.

```
package com.java.mypackage;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
```

```

import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreData {
    public static void main(String[] args) {

        //Create typesafe ServiceRegistry object
        StandardServiceRegistry ssr = new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

        Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

        SessionFactory factory = meta.getSessionFactoryBuilder().build();
        Session session = factory.openSession();
        Transaction t = session.beginTransaction();

        Employee e1=new Employee();
        e1.setId(101);
        e1.setFirstName("Gaurav");
        e1.setLastName("Chawla");

        session.save(e1);
        t.commit();
        System.out.println("successfully saved");
        factory.close();
        session.close();

    }
}

```

## Hibernate Example using Annotation in Eclipse

The hibernate application can be created with annotation. There are many annotations that can be used to create hibernate application such as @Entity, @Id, @Table etc.

Hibernate Annotations are based on the JPA 2 specification and supports all the features.

All the JPA annotations are defined in the **javax.persistence** package. Hibernate EntityManager implements the interfaces and life cycle defined by the JPA specification.

The core advantage of using hibernate annotation is that you don't need to create mapping (hbm) file. Here, hibernate annotations are used to provide the meta data.

## Create the Persistence class.

Here, we are creating the same persistent class which we have created in the previous topic. But here, we are using annotation.

**@Entity** annotation marks this class as an entity.

**@Table** annotation specifies the table name where data of this entity is to be persisted. If you don't use @Table annotation, hibernate will use the class name as the table name by default.

**@Id** annotation marks the identifier for this entity.

**@Column** annotation specifies the details of the column for this property or field. If @Column annotation is not specified, property name will be used as the column name by default.

To create the Persistence class, right click on **src/main/java - New - Class** - specify the class name with package - **finish**.

### Employee.java

```
package com.java.mypackage;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.Id;
```

```
import javax.persistence.Table;
```

```
@Entity
```

```
@Table(name= "emp500")
```

```
public class Employee {
```

```
@Id
```

```
private int id;
```

```
private String firstName,lastName;
```

```
public int getId() {
```

```
    return id;
```

```

}
public void setId(int id) {
    this.id = id;
}
public String getFirstName() {
    return firstName;
}
public void setFirstName(String firstName) {
    this.firstName = firstName;
}
public String getLastName() {
    return lastName;
}
public void setLastName(String lastName) {
    this.lastName = lastName;
}
}
}

```

## 4) Create the Configuration file

To create the configuration file, right click on **src/main/java - new - file** - specify the file name (e.g. hibernate.cfg.xml) - **Finish**.

### hibernate.cfg.xml

```

<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property name="connection.url">jdbc:mysql://localhost:3306/mphasis</property>
        <property name="connection.username">root</property>
        <property name="connection.password">password</property>
        <property name="show_sql">>true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>
        <mapping class="com.Employee"/>
    </session-factory>
</hibernate-configuration>

```

## 5) Create the class that retrieves or stores the persistent object.

### StoreData.java

```
package com.java.mypackage;
```

```
import org.hibernate.Session;
```

```
import org.hibernate.SessionFactory;
```

```
import org.hibernate.Transaction;
```

```
import org.hibernate.boot.Metadata;
```

```
import org.hibernate.boot.MetadataSources;
```

```
import org.hibernate.boot.registry.StandardServiceRegistry;
```

```
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
```

```
public class StoreData {
```

```
public static void main(String[] args) {
```

```
    StandardServiceRegistry ssr = new StandardServiceRegistryBuilder().configure("hib  
ernate.cfg.xml").build();
```

```
    Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();
```

```
    SessionFactory factory = meta.getSessionFactoryBuilder().build();
```

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

```
    Transaction t = session.beginTransaction();
```

```
        Employee e1=new Employee();
```

```
        e1.setId(101);
```

```
        e1.setFirstName("Gaurav");
```

```
        e1.setLastName("Chawla");
```

```
        session.save(e1);
```

```
        t.commit();
```

```
        System.out.println("successfully saved");
```

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

```
        session.close();
```

```
    }
```

```
}
```

# Web Application with Hibernate (using XML)

Here, we are going to create a web application with hibernate. For creating the web application, we are using JSP for presentation logic, Bean class for representing data and DAO class for database codes.

As we create the simple application in hibernate, we don't need to perform any extra operations in hibernate for creating web application. In such case, we are getting the value from the user using the JSP file.

---

## Example to create web application using hibernate

In this example, we are going to insert the record of the user in the database. It is simply a registration form.

---

### *index.jsp*

This page gets input from the user and sends it to the register.jsp file using post method.

```
<form action="register.jsp" method="post">
Name:<input type="text" name="name"/><br><br>
Password:<input type="password" name="password"/><br><br>
Email ID:<input type="text" name="email"/><br><br>
<input type="submit" value="register"/>

</form>
```

---

### *register.jsp*

This file gets all request parameters and stores this information into an object of User class. Further, it calls the register method of UserDao class passing the User class object.

```
<%@page import="com.java.mypack.UserDao"%>
<jsp:useBean id="obj" class="com.java.mypack.User">
</jsp:useBean>
<jsp:setProperty property="*" name="obj"/>
```

```
<%  
int i=UserDao.register(obj);  
if(i>0)  
out.print("You are successfully registered");  
  
%>
```

### *User.java*

It is the simple bean class representing the Persistent class in hibernate.

```
package com.java.mypack;  
  
public class User {  
private int id;  
private String name,password,email;  
  
//getters and setters  
  
}
```

---

### *user.hbm.xml*

It maps the User class with the table of the database.

```
<?xml version='1.0' encoding='UTF-8'?>  
<!DOCTYPE hibernate-mapping PUBLIC  
"-//Hibernate/Hibernate Mapping DTD 5.3//EN"  
"http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">  
  
<hibernate-mapping>  
<class name="com.java.mypack.User" table="u400">  
<id name="id">  
<generator class="increment"></generator>  
</id>
```

```
<property name="name"></property>
<property name="password"></property>
<property name="email"></property>
</class>

</hibernate-mapping>
```

---

### *UserDao.java*

A Dao class, containing method to store the instance of User class.

```
package com.java.mypack;
```

```
import org.hibernate.Session;
```

```
import org.hibernate.SessionFactory;
```

```
import org.hibernate.Transaction;
```

```
import org.hibernate.boot.Metadata;
```

```
import org.hibernate.boot.MetadataSources;
```

```
import org.hibernate.boot.registry.StandardServiceRegistry;
```

```
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
```

```
public class UserDao {
```

```
public static int register(User u){
```

```
int i=0;
```

```
StandardServiceRegistry ssr = new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
```

```
Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();
```

```
SessionFactory factory = meta.getSessionFactoryBuilder().build();
```

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

```
Transaction t = session.beginTransaction();
```

```
i=(Integer)session.save(u);
```

```
t.commit();
```



```
session.close();
```

```
return i;
```

```
}  
}
```

---

### *hibernate.cfg.xml*

It is a configuration file, containing informations about the database and mapping file.

1. `<?xml version='1.0' encoding='UTF-8'?>`
2. `<!DOCTYPE hibernate-configuration PUBLIC`
3. `"-//Hibernate/Hibernate Configuration DTD 5.3//EN"`
4. `"http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">`
- 5.
6. `<hibernate-configuration>`
- 7.
8. `<session-factory>`
9. `<property name="hbm2ddl.auto">create</property>`
10. `<property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>`
11. `<property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>`
12. `<property name="connection.username">system</property>`
13. `<property name="connection.password">jtp</property>`
14. `<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>`
- 15.
16. `<mapping resource="user.hbm.xml"/>`
17. `</session-factory>`
- 18.
19. `</hibernate-configuration>`

## Generator classes in Hibernate

1. [Hibernate Architecture](#)
2. [Hibernate Framework](#)
3. [Advantages of Hibernate Framework](#)

The <generator> class is a sub-element of id. It is used to generate the unique identifier for the objects of persistent class. There are many generator classes defined in the Hibernate Framework.

All the generator classes implements the **org.hibernate.id.IdentifierGenerator interface**. The application programmer may create one's own generator classes by implementing the IdentifierGenerator interface. Hibernate framework provides many built-in generator classes:

1. assigned
2. increment
3. sequence
4. hilo
5. native
6. identity
7. seqhilo
8. uuid
9. guid
10. select
11. foreign
12. sequence-identity

## Hibernate Query Language (HQL)

Hibernate Query Language (HQL) is same as SQL (Structured Query Language) but it doesn't depends on the table of the database. Instead of table name, we use class name in HQL. So it is database independent query language.

### Advantage of HQL

There are many advantages of HQL. They are as follows:

- database independent
- supports polymorphic queries
- easy to learn for Java Programmer

---

## Query Interface

It is an object oriented representation of Hibernate Query. The object of Query can be obtained by calling the `createQuery()` method Session interface.

The query interface provides many methods. There is given commonly used methods:

1. **`public int executeUpdate()`** is used to execute the update or delete query.
2. **`public List list()`** returns the result of the relation as a list.
3. **`public Query setFirstResult(int rowno)`** specifies the row number from where record will be retrieved.
4. **`public Query setMaxResult(int rowno)`** specifies the no. of records to be retrieved from the relation (table).
5. **`public Query setParameter(int position, Object value)`** it sets the value to the JDBC style query parameter.
6. **`public Query setParameter(String name, Object value)`** it sets the value to a named query parameter.

## Example of HQL to get all the records

```
Query query=session.createQuery("from Emp");//here persistent class name is Emp
List list=query.list();
```

---

## Example of HQL to get records with pagination

```
Query query=session.createQuery("from Emp");
query.setFirstResult(5);
query.setMaxResult(10);
List list=query.list();//will return the records from 5 to 10th number
```

---

## Example of HQL update query

```
Transaction tx=session.beginTransaction();
Query q=session.createQuery("update User set name=:n where id=:i");
```

```
q.setParameter("n","Udit Kumar");  
q.setParameter("i",111);
```

```
int status=q.executeUpdate();  
System.out.println(status);  
tx.commit();
```

---

## Example of HQL delete query

```
Query query=session.createQuery("delete from Emp where id=100");  
//specifying class name (Emp) not tablename  
query.executeUpdate();
```

## HQL with Aggregate functions

You may call avg(), min(), max() etc. aggregate functions by HQL. Let's see some common examples:

### Example to get total salary of all the employees

```
Query q=session.createQuery("select sum(salary) from Emp");  
List<Integer> list=q.list();  
System.out.println(list.get(0));
```

---

### Example to get maximum salary of employee

```
Query q=session.createQuery("select max(salary) from Emp");
```

---

### Example to get minimum salary of employee

```
Query q=session.createQuery("select min(salary) from Emp");
```

---

### Example to count total number of employee ID

```
Query q=session.createQuery("select count(id) from Emp");
```

---

## Example to get average salary of each employees

Query q=session.createQuery("select avg(salary) from Emp");

## HCQL (Hibernate Criteria Query Language)

The Hibernate Criteria Query Language (HCQL) is used to fetch the records based on the specific criteria. The Criteria interface provides methods to apply criteria such as retrieving all the records of table whose salary is greater than 50000 etc.

### Advantage of HCQL

The HCQL provides methods to add criteria, so it is **easy** for the java programmer to add criteria. The java programmer is able to add many criteria on a query.

### Criteria Interface

The Criteria interface provides many methods to specify criteria. The object of Criteria can be obtained by calling the **createCriteria()** method of Session interface.

#### *Syntax of createCriteria() method of Session interface*

1. **public** Criteria createCriteria(Class c)

The commonly used methods of Criteria interface are as follows:

1. **public Criteria add(Criterion c)** is used to add restrictions.
2. **public Criteria addOrder(Order o)** specifies ordering.
3. **public Criteria setFirstResult(int firstResult)** specifies the first number of record to be retrieved.
4. **public Criteria setMaxResult(int totalResult)** specifies the total number of records to be retrieved.
5. **public List list()** returns list containing object.
6. **public Criteria setProjection(Projection projection)** specifies the projection.

## Example of HCQL to get all the records

Criteria c=session.createCriteria(Emp.class);//passing Class class argument

List list=c.list();

---

## Example of HCQL to get the 10th to 20th record

```
Criteria c=session.createCriteria(Emp.class);  
c.setFirstResult(10);  
c.setMaxResult(20);  
List list=c.list();
```

---

## Example of HCQL to get the records whose salary is greater than 10000

```
Criteria c=session.createCriteria(Emp.class);  
c.add(Restrictions.gt("salary",10000));  
List list=c.list();
```

---

## Example of HCQL to get the records in ascending order on the basis of salary

```
Criteria c=session.createCriteria(Emp.class);  
c.addOrder(Order.asc("salary"));  
List list=c.list();
```

## Hibernate Named Query

The hibernate named query is way to use any query by some meaningful name. It is like using alias names. The Hibernate framework provides the concept of named queries so that application programmer need not to scatter queries to all the java code.

There are two ways to define the named query in hibernate:

- by annotation
  - by mapping file.
- 

## Hibernate Named Query by annotation

If you want to use named query in hibernate, you need to have knowledge of @NamedQueries and @NamedQuery annotations.

**@NameQueries** annotation is used to define the multiple named queries.

**@NamedQuery** annotation is used to define the single named query.

Let's see the example of using the named queries:

```
@NamedQueries(  
    {  
        @NamedQuery(  
            name = "findEmployeeByName",  
            query = "from Employee e where e.name = :name"  
        )  
    }  
)
```

## Example of Hibernate Named Query by annotation

In this example, we are using annotations to defined the named query in the persistent class. There are three files only:

- Employee.java
- hibernate.cfg.xml
- FetchDemo

In this example, we are assuming that there is em table in the database containing 4 columns id, name, job and salary and there are some records in this table.

### *Employee.java*

It is a persistent class that uses annotations to define named query and marks this class as entity.

```
package com.java;
```

```
import javax.persistence.*;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;
```

```
import javax.persistence.Id;
```

```
@NamedQueries(  
    {  
        @NamedQuery(  

```

```

        name = "findEmployeeByName",
        query = "from Employee e where e.name = :name"
    )
}
)

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

    public String toString(){return id+" "+name+" "+salary+" "+job;}

    int id;
    String name;
    int salary;
    String job;
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)

    //getters and setters
}

```

---

### *hibernate.cfg.xml*

It is a configuration file that stores the informations about database such as driver class, url, username, password and mapping class etc.

1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-configuration PUBLIC
3.     "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
4.     "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">
- 5.
6. <hibernate-configuration>
- 7.
8.     <session-factory>
9.         <property name="hbm2ddl.auto">update</property>
10.        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>



```
11. <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
12. <property name="connection.username">system</property>
13. <property name="connection.password">jtp</property>
14. <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
15. <mapping class="com.java.Employee"/>
16. </session-factory>
17.
18. </hibernate-configuration>
```

---

### *FetchData.java*

It is a java class that uses the named query and prints the informations based on the query. The **getNamedQuery** method uses the named query and returns the instance of Query.

```
package com.java;
```

```
import java.util.*;
```

```
import javax.persistence.*;
```

```
import org.hibernate.*;
```

```
import org.hibernate.boot.Metadata;
```

```
import org.hibernate.boot.MetadataSources;
```

```
import org.hibernate.boot.registry.StandardServiceRegistry;
```

```
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
```

```
public class Fetch {
```

```
public static void main(String[] args) {
```

```
StandardServiceRegistry ssr=new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
```

```
Metadata meta=new MetadataSources(ssr).getMetadataBuilder().build();
```

```
SessionFactory factory=meta.getSessionFactoryBuilder().build();
```

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

```
//Hibernate Named Query
```

```
TypedQuery query = session.getNamedQuery("findEmployeeByName");
```

```
query.setParameter("name", "amit");
```

```
List<Employee> employees=query.getResultList();

Iterator<Employee> itr=employees.iterator();
while(itr.hasNext()){
Employee e=itr.next();
System.out.println(e);
}
session.close();
}
}
```

## Hibernate Inheritance Mapping Tutorial

We can map the inheritance hierarchy classes with the table of the database. There are three inheritance mapping strategies defined in the hibernate:

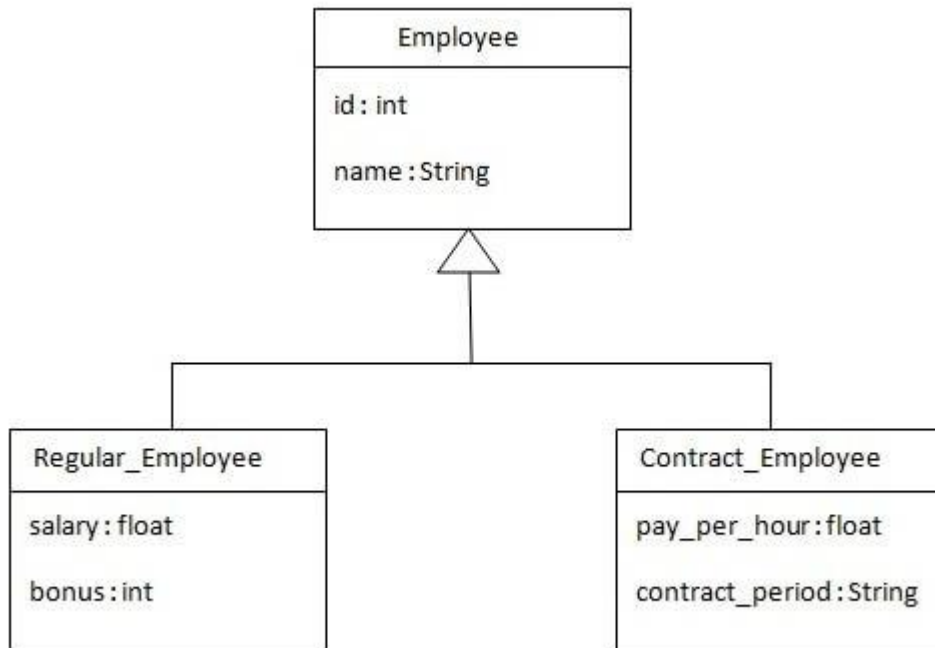
1. Table Per Hierarchy
2. Table Per Concrete class
3. Table Per Subclass

### Hibernate Table Per Hierarchy using Annotation

In the previous page, we have mapped the inheritance hierarchy with one table using xml file. Here, we are going to perform this task using annotation. You need to use `@Inheritance(strategy=InheritanceType.SINGLE_TABLE)`, `@DiscriminatorColumn` and `@DiscriminatorValue` annotations for mapping table per hierarchy strategy.

In case of table per hierarchy, only one table is required to map the inheritance hierarchy. Here, an extra column (also known as **discriminator column**) is created in the table to identify the class.

Let's see the inheritance hierarchy:



There are three classes in this hierarchy. Employee is the super class for Regular\_Employee and Contract\_Employee classes.

The table structure for this hierarchy is as shown below:

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
TYPE	VARCHAR2(255)	No	-	-
NAME	VARCHAR2(255)	Yes	-	-
SALARY	FLOAT	Yes	-	-
BONUS	NUMBER(10,0)	Yes	-	-
PAY_PER_HOUR	FLOAT	Yes	-	-
CONTRACT_DURATION	VARCHAR2(255)	Yes	-	-
				1 - 7

## Example of Hibernate Table Per Hierarchy using Annotation

You need to follow the following steps to create simple example:

- Create the persistent classes
- Edit pom.xml file
- Create the configuration file

- Create the class to store the fetch the data

## 1) Create the Persistent classes

You need to create the persistent classes representing the inheritance. Let's create the three classes for the above hierarchy:

*File: Employee.java*

```
package com.java.mypackage;
import javax.persistence.*;

@Entity
@Table(name = "employee101")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="type",discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue(value="employee")

public class Employee {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)

    @Column(name = "id")
    private int id;

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

    //setters and getters
}
File: Regular_Employee.java
```

```
package com.java.mypackage;

import javax.persistence.*;

@Entity
@DiscriminatorValue("regularemployee")
public class Regular_Employee extends Employee{

    @Column(name="salary")
    private float salary;

    @Column(name="bonus")
    private int bonus;

    //setters and getters
}
File: Contract_Employee.java
```

```
package com.java.mypackage;

import javax.persistence.Column;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@DiscriminatorValue("contractemployee")
public class Contract_Employee extends Employee{
```

```

@Column(name="pay_per_hour")
private float pay_per_hour;

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

//setters and getters
}

```

---

## Add the persistent classes in configuration file

Open the hibernate.cfg.xml file, and add entries of entity classes like this:

1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE hibernate-configuration PUBLIC
3.     ["-//Hibernate/Hibernate Configuration DTD 5.3//EN"](#)
4.     ["http://www.hibernate.org/dtd/hibernate-configuration-5.3.dtd"](http://www.hibernate.org/dtd/hibernate-configuration-5.3.dtd)>
5. <hibernate-configuration>
6.     <session-factory>
- 7.
8.     <property name="hbm2ddl.auto">update</property>
9.     <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
10.    <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
11.    <property name="connection.username">system</property>
12.    <property name="connection.password">jtp</property>
13.    <property name="connection.driver\_class">oracle.jdbc.driver.OracleDriver</property>
- 14.
15.    <mapping class="com.java.mypackage.Employee"/>
16.    <mapping class="com.java.mypackage.Regular\_Employee"/>
17.    <mapping class="com.java.mypackage.Contract\_Employee"/>

- 18.
19. </session-factory>
20. </hibernate-configuration>

The hbm2ddl.auto property is defined for creating automatic table in the database.

---

#### 4) Create the class that stores the persistent object

In this class, we are simply storing the employee objects in the database.

*File: StoreTest.java*

```
package com.java.mypackage;
```

```
import org.hibernate.Session;
```

```
import org.hibernate.SessionFactory;
```

```
import org.hibernate.Transaction;
```

```
import org.hibernate.boot.Metadata;
```

```
import org.hibernate.boot.MetadataSources;
```

```
import org.hibernate.boot.registry.StandardServiceRegistry;
```

```
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
```

```
public class StoreTest {
```

```
    public static void main(String args[])
```

```
    {
```

```
        StandardServiceRegistry ssr = new StandardServiceRegistryBuilder().configure("hibernat  
e.cfg.xml").build();
```

```
        Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();
```

```
        SessionFactory factory=meta.getSessionFactoryBuilder().build();
```

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

```
Transaction t=session.beginTransaction();

Employee e1=new Employee();
e1.setName("Gaurav Chawla");

Regular_Employee e2=new Regular_Employee();
e2.setName("Vivek Kumar");
e2.setSalary(50000);
e2.setBonus(5);

Contract_Employee e3=new Contract_Employee();
e3.setName("Arjun Kumar");
e3.setPay_per_hour(1000);
e3.setContract_duration("15 hours");

session.persist(e1);
session.persist(e2);
session.persist(e3);

t.commit();
session.close();
System.out.println("success");
}
}
```

---

## Table Per Concrete class using Annotation

In case of Table Per Concrete class, tables are created per class. So there are no nullable values in the table. Disadvantage of this approach is that duplicate columns are created in the subclass tables.

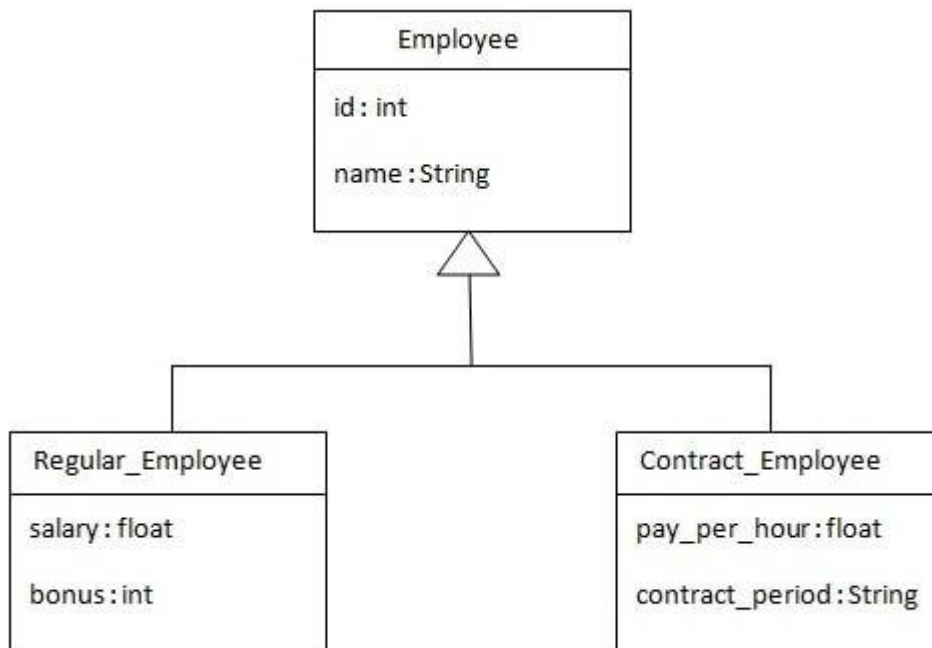
Here, we need to use `@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)` annotation in the parent class and `@AttributeOverrides` annotation in the subclasses.

**@Inheritance(strategy = InheritanceType.TABLE\_PER\_CLASS)** specifies that we are using table per concrete class strategy. It should be specified in the parent class only.



**@AttributeOverrides** defines that parent class attributes will be overridden in this class. In table structure, parent class table columns will be added in the subclass table.

The class hierarchy is given below:



The table structure for each table will be as follows:

#### *Table structure for Employee class*

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
				1 - 2

### Table structure for Regular\_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
SALARY	FLOAT	Yes	-	-
BONUS	NUMBER(10,0)	Yes	-	-
				1 - 4

### Table structure for Contract\_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
PAY_PER_HOUR	FLOAT	Yes	-	-
CONTRACT_DURATION	VARCHAR2(255)	Yes	-	-
				1 - 4

---

## Example of Table per concrete class

In this example we are creating the three classes and provide mapping of these classes in the employee.hbm.xml file.

### 1) Create the Persistent classes

You need to create the persistent classes representing the inheritance. Let's create the three classes for the above hierarchy:

File: Employee.java

```
package com.java.mypackage;
```

```
import javax.persistence.*;
```

```
@Entity
```

```
@Table(name = "employee102")
```

```
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
```

```
public class Employee {
```

```
    @Id
```

```
@GeneratedValue(strategy=GenerationType.AUTO)
```

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

```
private int id;
```

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

```
private String name;
```

```
//setters and getters
```

```
}
```

```
File: Regular_Employee.java
```

```
package com.java.mypackage;
```

```
import javax.persistence.*;
```

```
@Entity
```

```
@Table(name="regularemployee102")
```

```
@AttributeOverrides({
```

```
    @AttributeOverride(name="id", column=@Column(name="id")),
```

```
    @AttributeOverride(name="name", column=@Column(name="name"))
```

```
})
```

```
public class Regular_Employee extends Employee{
```

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

```
private float salary;
```

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

```
private int bonus;
```

```
//setters and getters
```

```
}
```

```
File: Contract_Employee.java
```

```
package com.java.mypackage;
```

```
import javax.persistence.*;
```

```
@Entity
```

```
@Table(name="contractemployee102")
```

```
@AttributeOverrides({
```

```
    @AttributeOverride(name="id", column=@Column(name="id")),
```

```

    @AttributeOverride(name="name", column=@Column(name="name"))
})
public class Contract_Employee extends Employee{

    @Column(name="pay_per_hour")
    private float pay_per_hour;

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

    public float getPay_per_hour() {
        return pay_per_hour;
    }
    public void setPay_per_hour(float payPerHour) {
        pay_per_hour = payPerHour;
    }
    public String getContract_duration() {
        return contract_duration;
    }
    public void setContract_duration(String contractDuration) {
        contract_duration = contractDuration;
    }
}

```

### Add mapping of hbm file in configuration file

*File: hibernate.cfg.xml*

1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-configuration PUBLIC
3.     "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
4.     "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">
- 5.
6. <!-- Generated by MyEclipse Hibernate Tools.                      -->
7. <hibernate-configuration>
8.     <session-factory>
9.         <property name="hbm2ddl.auto">update</property>
10.        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>

```

11.    <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</proper
      ty>
12.    <property name="connection.username">system</property>
13.    <property name="connection.password">jtp</property>
14. <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property
      >
15.
16.    <mapping class="com.java.mypackage.Employee"/>
17.    <mapping class="com.java.mypackage.Contract_Employee"/>
18.    <mapping class="com.java.mypackage.Regular_Employee"/>
19.  </session-factory>
20. </hibernate-configuration>

```

The hbm2ddl.auto property is defined for creating automatic table in the database.

---

#### 4) Create the class that stores the persistent object

In this class, we are simply storing the employee objects in the database.

*File: StoreData.java*

```

package com.java.mypackage;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreData {

    public static void main(String[] args) {

        StandardServiceRegistry ssr=new StandardServiceRegistryBuilder().configure("hi
        bernate.cfg.xml").build();
        Metadata meta=new MetadataSources(ssr).getMetadataBuilder().build();
    }
}

```

```
SessionFactory factory=meta.getSessionFactoryBuilder().build();
Session session=factory.openSession();

Transaction t=session.beginTransaction();

Employee e1=new Employee();
e1.setName("Gaurav Chawla");

Regular_Employee e2=new Regular_Employee();
e2.setName("Vivek Kumar");
e2.setSalary(50000);
e2.setBonus(5);

Contract_Employee e3=new Contract_Employee();
e3.setName("Arjun Kumar");
e3.setPay_per_hour(1000);
e3.setContract_duration("15 hours");

session.persist(e1);
session.persist(e2);
session.persist(e3);

t.commit();
session.close();
System.out.println("success");
}
}
```

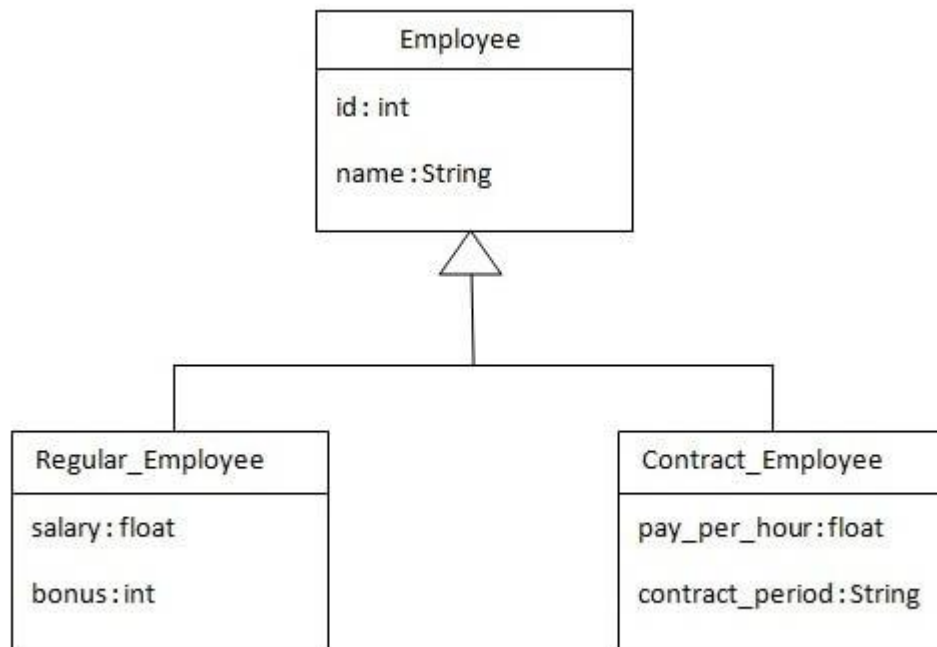
---

## Table Per Subclass using Annotation

As we have specified earlier, in case of table per subclass strategy, tables are created as per persistent classes but they are treated using primary and foreign key. So there will not be any duplicate column in the relation.

We need to specify **@Inheritance(strategy=InheritanceType.JOINED)** in the parent class and **@PrimaryKeyJoinColumn** annotation in the subclasses.

Let's see the hierarchy of classes that we are going to map.



The table structure for each table will be as follows:

#### *Table structure for Employee class*

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
				1 - 2

### Table structure for Regular\_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
EID	NUMBER(10,0)	No	-	1
SALARY	FLOAT	Yes	-	-
BONUS	NUMBER(10,0)	Yes	-	-
				1 - 3

### Table structure for Contract\_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
EID	NUMBER(10,0)	No	-	1
PAY_PER_HOUR	FLOAT	Yes	-	-
CONTRACT_DURATION	VARCHAR2(255)	Yes	-	-
				1 - 3

## Example of Table per subclass class using Annotation

In this example we are creating the three classes and provide mapping of these classes in the employee.hbm.xml file.

### 1) Create the Persistent classes

You need to create the persistent classes representing the inheritance. Let's create the three classes for the above hierarchy:

File: Employee.java

```
package com.java.mypackage;
```

```
import javax.persistence.*;
```

```
@Entity
```

```
@Table(name = "employee103")
```

```
@Inheritance(strategy=InheritanceType.JOINED)
```



```
public class Employee {  
    @Id  
    @GeneratedValue(strategy=GenerationType.AUTO)  
  
    @Column(name = "id")  
    private int id;  
  
    @Column(name = "name")  
    private String name;  
  
    //setters and getters  
}  
File: Regular_Employee.java
```

```
package com.java.mypackage;  
  
import javax.persistence.*;  
@Entity  
@Table(name="regularemployee103")  
@PrimaryKeyJoinColumn(name="ID")  
public class Regular_Employee extends Employee{  
  
    @Column(name="salary")  
    private float salary;  
  
    @Column(name="bonus")  
    private int bonus;  
  
    //setters and getters  
}  
File: Contract_Employee.java
```

```
package com.java.mypackage;

import javax.persistence.*;

@Entity
@Table(name="contractemployee103")
@PrimaryKeyJoinColumn(name="ID")
public class Contract_Employee extends Employee{

    @Column(name="pay_per_hour")
    private float pay_per_hour;

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

    //setters and getters
}
```

---

## 2) Add project information and configuration in pom.xml file.

Open pom.xml file and click source. Now, add the below dependencies between <dependencies>....</dependencies> tag.

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
```

```
<version>5.3.1.Final</version>
</dependency>

<dependency>
  <groupId>com.oracle</groupId>
  <artifactId>ojdbc14</artifactId>
  <version>10.2.0.4.0</version>
</dependency>
```

---

### 3)Create the configuration file

Open the hibernate.cfg.xml file, and add an entry of mapping resource like this:

```
<mapping class="com.java.mypackage.Employee"/>
<mapping class="com.java.mypackage.Contract_Employee"/>
<mapping class="com.java.mypackage.Regular_Employee"/>
```

Now the configuration file will look like this:

*File: hibernate.cfg.xml*

1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-configuration PUBLIC
3.     "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
4.     "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">
- 5.
6. <!-- Generated by MyEclipse Hibernate Tools.                   -->

```
7. <hibernate-configuration>
8.
9.   <session-factory>
10.     <property name="hbm2ddl.auto">update</property>
11.     <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
12.     <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
13.     <property name="connection.username">system</property>
14.     <property name="connection.password">jtp</property>
15. <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
16.
17.   <mapping class="com.java.mypackage.Employee"/>
18.   <mapping class="com.java.mypackage.Contract_Employee"/>
19.   <mapping class="com.java.mypackage.Regular_Employee"/>
20. </session-factory>
21.
22. </hibernate-configuration>
```

The hbm2ddl.auto property is defined for creating automatic table in the database.

---

### 3) Create the class that stores the persistent object

In this class, we are simply storing the employee objects in the database.

*File: StoreData.java*

```
package com.java.mypackage;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
```

```
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
```

```
public class StoreData {
```

```
    public static void main(String args[])
    {
        StandardServiceRegistry ssr = new StandardServiceRegistryBuilder().configure("hibernat
e.cfg.xml").build();
        Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

        SessionFactory factory=meta.getSessionFactoryBuilder().build();
        Session session=factory.openSession();

        Transaction t=session.beginTransaction();

        Employee e1=new Employee();
        e1.setName("Gaurav Chawla");

        Regular_Employee e2=new Regular_Employee();
        e2.setName("Vivek Kumar");
        e2.setSalary(50000);
        e2.setBonus(5);

        Contract_Employee e3=new Contract_Employee();
        e3.setName("Arjun Kumar");
        e3.setPay_per_hour(1000);
        e3.setContract_duration("15 hours");
        session.persist(e1);
        session.persist(e2);
        session.persist(e3);

        t.commit();
        session.close();
        System.out.println("success");
    }
}
```

# Collection Mapping in Hibernate

We can map collection elements of Persistent class in Hibernate. You need to declare the type of collection in Persistent class from one of the following types:

- java.util.List
- java.util.Set
- java.util.SortedSet
- java.util.Map
- java.util.SortedMap
- java.util.Collection
- or write the implementation of org.hibernate.usertype.UserCollectionType

The persistent class should be defined like this for collection element.

```
package com.java;
```

```
import java.util.List;
```

```
public class Question {
```

```
    private int id;
```

```
    private String qname;
```

```
    private List<String> answers;//List can be of any type
```

```
    //getters and setters
```

```
}
```

---

## Mapping collection in mapping file

There are many subelements of **<class>** elements to map the collection. They are **<list>**, **<bag>**, **<set>** and **<map>**. Let's see how we implement the list for the above class:

```
<class name="com.java.Question" table="q100">  
    <id name="id">  
        <generator class="increment"></generator>
```

```

</id>
<property name="qname"></property>

<list name="answers" table="ans100">
<key column="qid"></key>
<index column="type"></index>
<element column="answer" type="string"></element>
</list>

</class>

```

There are three subelements used in the list:

- **<key>** element is used to define the foreign key in this table based on the Question class identifier.
- **<index>** element is used to identify the type. List and Map are indexed collection.
- **<element>** is used to define the element of the collection.

This is the mapping of collection if collection stores string objects. But if collection stores entity reference (another class objects), we need to define **<one-to-many>** or **<many-to-many>** element. Now the Persistent class will look like:

```
package com.java;
```

```
import java.util.List;
```

```
public class Question {
```

```
private int id;
```

```
private String qname;
```

```
private List<Answer> answers;//Here, List stores the objects of Answer class
```

```
//getters and setters
```

```
}
```

```
package com.java;
```

```
import java.util.List;
```

```
public class Answer {
```

```
private int id;
```

```
private String answer;
private String posterName;
//getters and setters
}
```

Now the mapping file will be:

```
<class name="com.java.Question" table="q100">
    <id name="id">
        <generator class="increment"></generator>
    </id>
    <property name="qname"></property>

    <list name="answers" >
        <key column="qid"></key>
        <index column="type"></index>
        <one-to-many class="com.java.Answer" />
    </list>

</class>
```

Here, List is mapped by one-to-many relation. In this scenario, there can be many answers for one question.

---

## Understanding key element

The key element is used to define the foreign key in the joined table based on the original identity. The foreign key element is nullable by default. So for non-nullable foreign key, we need to specify not-null attribute such as:

```
<key column="qid" not-null="true" ></key>
```

The attributes of the key element are column, on-delete, property-ref, not-null, update and unique.

```
<key
column="columnname"
on-delete="noaction|cascade"
```



```
not-null="true|false"  
property-ref="propertyName"  
update="true|false"  
unique="true|false"  
</>
```

---

## Indexed collections

The collection elements can be categorized in two forms:

- **indexed** ,and
- **non-indexed**

The List and Map collection are indexed whereas set and bag collections are non-indexed. Here, indexed collection means List and Map requires an additional element **<index>**.

---

## Collection Elements

The collection elements can have value or entity reference (another class object). We can use one of the 4 elements

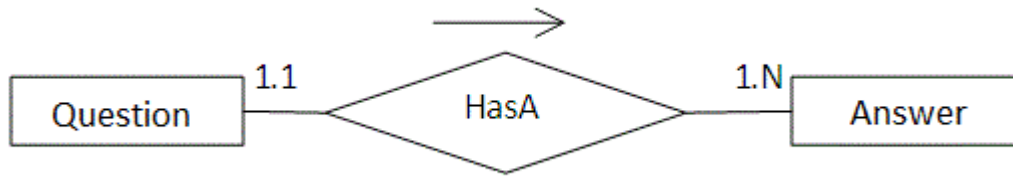
- **element**
- **component-element**
- **one-to-many**, or
- **many-to-many**

The element and component-element are used for normal value such as string, int etc. whereas one-to-many and many-to-many are used to map entity reference.

## Mapping List in Collection Mapping (using xml file)

If our persistent class has List object, we can map the List easily either by **<list>** element of class in mapping file or by annotation.

Here, we are using the scenario of Forum where one question has multiple answers.



Let's see how we can implement the list in the mapping file:

```
<class name="com.java.Question" table="q100">
...
  <list name="answers" table="ans100">
    <key column="qid"> </key>
    <index column="type"> </index>
    <element column="answer" type="string"> </element>
  </list>
...
</class>
```

*List and Map are index based collection, so an extra column will be created in the table for indexing.*

## Example of mapping list in collection mapping

In this example, we are going to see full example of collection mapping by list. This is the example of List that stores string value not entity reference that is why we are going to use **element** instead of **one-to-many** within the list element.

### 1) Create the Persistent class

This persistent class defines properties of the class including List.

```
package com.java;
```

```
import java.util.List;
```

```
public class Question {
  private int id;
  private String qname;
```

```
private List<String> answers;
```

```
//getters and setters
```

```
}
```

---

## 2) Create the Mapping file for the persistent class

Here, we have created the question.hbm.xml file for defining the list.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
  <class name="com.java.Question" table="q100">
    <id name="id">
      <generator class="increment"></generator>
    </id>
    <property name="qname"></property>

    <list name="answers" table="ans100">
      <key column="qid"></key>
      <index column="type"></index>
      <element column="answer" type="string"></element>
    </list>

  </class>

</hibernate-mapping>
```

---

## 3) Create the configuration file

This file contains information about the database and mapping file.

```
<?xml version='1.0' encoding='UTF-8'?>
```

```
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping resource="question.hbm.xml"/>
    </session-factory>

</hibernate-configuration>
```

---

## 4) Create the class to store the data

In this class we are storing the data of the question class.

```
package com.java;
```

```
import java.util.ArrayList;
```

```
import org.hibernate.*;
```

```
import org.hibernate.boot.Metadata;
```

```
import org.hibernate.boot.MetadataSources;
```

```
import org.hibernate.boot.registry.StandardServiceRegistry;
```

```
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
```

```
public class StoreData {
```

```
    public static void main(String[] args) {
```

```
        StandardServiceRegistry ssr=new StandardServiceRegistryBuilder().configure("hibernate.cf
g.xml").build();
```

```
Metadata meta=new MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory=meta.getSessionFactoryBuilder().build();
Session session=factory.openSession();

Transaction t=session.beginTransaction();

ArrayList<String> list1=new ArrayList<String>();
list1.add("Java is a programming language");
list1.add("Java is a platform");

ArrayList<String> list2=new ArrayList<String>();
list2.add("Servlet is an Interface");
list2.add("Servlet is an API");

Question question1=new Question();
question1.setQname("What is Java?");
question1.setAnswers(list1);

Question question2=new Question();
question2.setQname("What is Servlet?");
question2.setAnswers(list2);

session.persist(question1);
session.persist(question2);

t.commit();
session.close();
System.out.println("success");
}
}
```

## Output

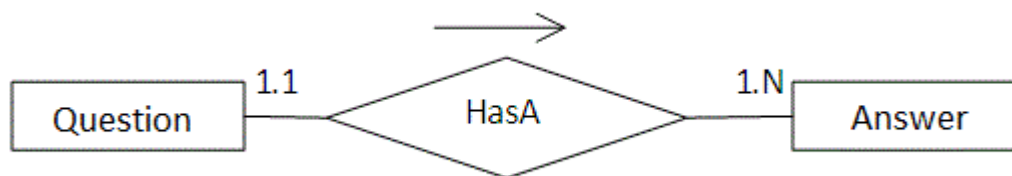
ID	QNAME
1	What is Java?
2	What is Servlet?

QID	TYPE	ANSWER
1	0	Java is a programming language
1	1	Java is a platform
2	0	Servlet is an Interface
2	1	Servlet is an API

## Mapping Bag in Collection Mapping (using xml file)

If our persistent class has List object, we can map the List by list or bag element in the mapping file. The bag is just like List but it doesn't require index element.

Here, we are using the scenario of Forum where one question has multiple answers.



Let's see how we can implement the bag in the mapping file:

```

<class name="com.java.Question" table="q100">
  ...
  <bag name="answers" table="ans100">
    <key column="qid"> </key>
    <element column="answer" type="string"> </element>
  </bag>
  ...
</class>

```

## Example of mapping bag in collection mapping

In this example, we are going to see full example of collection mapping by bag. This is the example of bag if it stores value not entity reference that is why are going to

use **element** instead of **one-to-many**. If you have seen the example of mapping list, it is same in all cases instead mapping file where we are using bag instead of list.

## 1) Create the Persistent class

This persistent class defines properties of the class including List.

```
package com.java;

import java.util.List;

public class Question {
    private int id;
    private String qname;
    private List<String> answers;

    //getters and setters

}
```

---

## 2) Create the Mapping file for the persistent class

Here, we have created the question.hbm.xml file for defining the list.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
<class name="com.java.Question" table="q101">
    <id name="id">
        <generator class="increment"> </generator>
    </id>
    <property name="qname"> </property>

    <bag name="answers" table="ans101">
        <key column="qid"> </key>
```

```
<element column="answer" type="string"></element>
</bag>
```

```
</class>
```

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

---

### 3) Create the configuration file

This file contains information about the database and mapping file.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<!-- Generated by MyEclipse Hibernate Tools.           -->
<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping resource="question.hbm.xml"/>
    </session-factory>

</hibernate-configuration>
```

---

### 4) Create the class to store the data

In this class we are storing the data of the question class.

```
package com.java;
```



```
import java.util.ArrayList;
```

```
import org.hibernate.Session;
```

```
import org.hibernate.SessionFactory;
```

```
import org.hibernate.Transaction;
```

```
import org.hibernate.boot.Metadata;
```

```
import org.hibernate.boot.MetadataSources;
```

```
import org.hibernate.boot.registry.StandardServiceRegistry;
```

```
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
```

```
public class StoreData {
```

```
    public static void main(String[] args) {
```

```
        StandardServiceRegistry ssr=new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
```

```
        Metadata meta=new MetadataSources(ssr).getMetadataBuilder().build();
```

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

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

```
        Transaction t=session.beginTransaction();
```

```
        ArrayList<String> list1=new ArrayList<String>();
```

```
        list1.add("Java is a programming language");
```

```
        list1.add("Java is a platform");
```

```
        ArrayList<String> list2=new ArrayList<String>();
```

```
        list2.add("Servlet is an Interface");
```

```
        list2.add("Servlet is an API");
```

```
        Question question1=new Question();
```

```
        question1.setQname("What is Java?");
```

```
        question1.setAnswers(list1);
```

```
        Question question2=new Question();
```

```
        question2.setQname("What is Servlet?");
```

```
        question2.setAnswers(list2);
```

```

session.persist(question1);
session.persist(question2);

t.commit();
session.close();
System.out.println("success");
}
}

```

## Output

ID	QNAME
1	What is Java?
2	What is Servlet?

QID	ANSWER
1	Java is a programming language
1	Java is a platform
2	Servlet is an Interface
2	Servlet is an API

## How to fetch the data

Here, we have used HQL to fetch all the records of Question class including answers. In such case, it fetches the data from two tables that are functional dependent.

```
package com.java;
```

```

import javax.persistence.TypedQuery;
import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

```

```

public class FetchData {
    public static void main(String[] args) {

```

```

        StandardServiceRegistry ssr=new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
        Metadata meta=new MetadataSources(ssr).getMetadataBuilder().build();

```

```

SessionFactory factory=meta.buildSessionFactory();
Session session=factory.openSession();

TypedQuery query=session.createQuery("from Question");
List<Question> list=query.getResultList();

Iterator<Question> itr=list.iterator();
while(itr.hasNext()){
    Question q=itr.next();
    System.out.println("Question Name: "+q.getQname());

    //printing answers
    List<String> list2=q.getAnswers();
    Iterator<String> itr2=list2.iterator();
    while(itr2.hasNext()){
        System.out.println(itr2.next());
    }

}
session.close();
System.out.println("success");

}
}

```

## Hibernate Mapping Set using XML

If our persistent class has Set object, we can map the Set by set element in the mapping file. The set element doesn't require index element. The one difference between List and Set is that, it stores only unique values.

Let's see how we can implement the set in the mapping file:

```
<class name="com.java.Question" table="q1002">
```

```
...
    <set name="answers" table="ans1002">
    <key column="qid"> </key>
    <element column="answer" type="string"> </element>
    </set>

...
</class>
```

---

## Example of mapping set in collection mapping

In this example, we are going to see full example of collection mapping by set. This is the example of set that stores value not entity reference that is why are going to use element instead of one-to-many.

### 1) Create the Persistent class

This persistent class defines properties of the class including Set.

```
package com.java;

import java.util.Set;

public class Question {
    private int id;
    private String qname;
    private Set<String> answers;

    //getters and setters

}
```

---

### 2) Create the Mapping file for the persistent class

Here, we have created the question.hbm.xml file for defining the list.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
```

```
"-//Hibernate/Hibernate Mapping DTD 5.3//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">
```

```
<hibernate-mapping>
<class name="com.java.Question" table="q1002">
  <id name="id">
    <generator class="increment"></generator>
  </id>
  <property name="qname"></property>

  <set name="answers" table="ans1002">
    <key column="qid"></key>
    <element column="answer" type="string"></element>
  </set>

</class>

</hibernate-mapping>
```

---

### 3) create the configuration file

This file contains information about the database and mapping file.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
  "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

  <session-factory>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
    <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
    <property name="connection.username">system</property>
    <property name="connection.password">jtp</property>
    <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
```

```
<mapping resource="question.hbm.xml"/>
</session-factory>
```

```
</hibernate-configuration>
```

---

## 4) Create the class to store the data

In this class we are storing the data of the question class.

```
package com.java;
```

```
import java.util.HashSet;
```

```
import org.hibernate.*;
```

```
import org.hibernate.boot.Metadata;
```

```
import org.hibernate.boot.MetadataSources;
```

```
import org.hibernate.boot.registry.StandardServiceRegistry;
```

```
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
```

```
public class StoreData {
```

```
    public static void main(String[] args) {
```

```
        StandardServiceRegistry ssr=new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
```

```
        Metadata meta=new MetadataSources(ssr).getMetadataBuilder().build();
```

```
        SessionFactory factory=meta.getSessionFactoryBuilder().build();
```

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

```
        Transaction t=session.beginTransaction();
```

```
        HashSet<String> set1=new HashSet<String>();
```

```
        set1.add("Java is a programming language");
```

```
        set1.add("Java is a platform");
```

```
        HashSet<String> set2=new HashSet<String>();
```

```

set2.add("Servlet is an Interface");
set2.add("Servlet is an API");

Question question1=new Question();
question1.setQname("What is Java?");
question1.setAnswers(set1);

Question question2=new Question();
question2.setQname("What is Servlet?");
question2.setAnswers(set2);

session.persist(question1);
session.persist(question2);

t.commit();
session.close();
System.out.println("success");
}
}

```

## Output

ID	QNAME
1	What is Java?
2	What is Servlet?

QID	ANSWER
1	Java is a programming language
1	Java is a platform
2	Servlet is an Interface
2	Servlet is an API

## How to fetch the data of Set

Here, we have used HQL to fetch all the records of Question class including answers. In such case, it fetches the data from two tables that are functional dependent.

```
package com.java;
```

```
import java.util.*;
```

```
import javax.persistence.TypedQuery;
```

```
import org.hibernate.*;
```

```

import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class FetchData {
    public static void main(String[] args) {

        StandardServiceRegistry ssr=new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
        Metadata meta=new MetadataSources(ssr).getMetadataBuilder().build();

        SessionFactory factory=meta.getSessionFactoryBuilder().build();
        Session session=factory.openSession();

        Transaction t=session.beginTransaction();

        TypedQuery query=session.createQuery("from Question");
        List<Question> list=query.getResultList();

        Iterator<Question> itr=list.iterator();
        while(itr.hasNext()){
            Question q=itr.next();
            System.out.println("Question Name: "+q.getQname());

            //printing answers
            Set<String> set=q.getAnswers();
            Iterator<String> itr2=set.iterator();
            while(itr2.hasNext()){
                System.out.println(itr2.next());
            }
        }
        session.close();
        System.out.println("success");

    }
}

```



# Hibernate Mapping Map using xml file

Hibernate allows you to map Map elements with the RDBMS. As we know, list and map are index-based collections. In case of map, index column works as the key and element column works as the value.

## Example of Mapping Map in collection mapping using xml file

You need to create following pages for mapping map elements.

- Question.java
- question.hbm.xml
- hibernate.cfg.xml
- StoreTest.java
- FetchTest.java

---

### Question.java

```
package com.java;
```

```
import java.util.Map;
```

```
public class Question {
```

```
    private int id;
```

```
    private String name,username;
```

```
    private Map<String,String> answers;
```

```
    public Question() {}
```

```
    public Question(String name, String username, Map<String, String> answers) {
```

```
        super();
```

```
        this.name = name;
```

```
        this.username = username;
```

```
        this.answers = answers;
```

```

}
public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public String getUsername() {
    return username;
}
public void setUsername(String username) {
    this.username = username;
}
public Map<String, String> getAnswers() {
    return answers;
}
public void setAnswers(Map<String, String> answers) {
    this.answers = answers;
}
}

```

---

### question.hbm.xml

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>

<class name="com.java.Question" table="question736">

```

```

<id name="id">
<generator class="native"></generator>
</id>
<property name="name"></property>
<property name="username"></property>

<map name="answers" table="answer736" cascade="all">
<key column="questionid"></key>
<index column="answer" type="string"></index>
<element column="username" type="string"></element>
</map>
</class>

```

```

</hibernate-mapping>
hibernate.cfg.xml

```

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

```

```

<hibernate-configuration>

```

```

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

```

```

    <mapping resource="question.hbm.xml"/>
    </session-factory>

```

```

</hibernate-configuration>

```

**StoreTest.java**

**package** com.java;

```
import java.util.HashMap;
import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
```

```
public class StoreTest {
    public static void main(String[] args) {
```

```
        StandardServiceRegistry ssr=new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
```

```
        Metadata meta=new MetadataSources(ssr).getMetadataBuilder().build();
```

```
        SessionFactory factory=meta.getSessionFactoryBuilder().build();
```

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

```
        Transaction t=session.beginTransaction();
```

```
        HashMap<String,String> map1=new HashMap<String,String>();
```

```
        map1.put("Java is a programming language","John Milton");
```

```
        map1.put("Java is a platform","Ashok Kumar");
```

```
        HashMap<String,String> map2=new HashMap<String,String>();
```

```
        map2.put("Servlet technology is a server side programming","John Milton");
```

```
        map2.put("Servlet is an Interface","Ashok Kumar");
```

```
        map2.put("Servlet is a package","Rahul Kumar");
```

```
        Question question1=new Question("What is Java?","Alok",map1);
```

```
        Question question2=new Question("What is Servlet?","Jai Dixit",map2);
```

```
        session.persist(question1);
```

```
        session.persist(question2);
```

```
        t.commit();
```

```

session.close();
System.out.println("successfully stored");
}
}

```

## Output

ID	NAME	USERNAME
1	What is Java?	Alok
2	What is Servlet?	Jai Dixit

QUESTIONID	ANSWER	USERNAME
1	Java is a programming language	John Milton
1	Java is a platform	Ashok Kumar
2	Servlet is a package	Rahul Kumar
2	Servlet technology is a server side programming	John Milton
2	Servlet is an Interface	Ashok Kumar

## FetchTest.java

```
package com.java;
```

```
import java.util.*;
```

```
import javax.persistence.TypedQuery;
```

```
import org.hibernate.*;
```

```
import org.hibernate.boot.Metadata;
```

```
import org.hibernate.boot.MetadataSources;
```

```
import org.hibernate.boot.registry.StandardServiceRegistry;
```

```
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
```

```
public class FetchTest {
```

```
public static void main(String[] args) {
```

```
    StandardServiceRegistry ssr=new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
```

```
    Metadata meta=new MetadataSources(ssr).getMetadataBuilder().build();
```

```
    SessionFactory factory=meta.getSessionFactoryBuilder().build();
```

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

```

TypedQuery query=session.createQuery("from Question ");
List<Question> list=query.getResultList();

Iterator<Question> iterator=list.iterator();
while(iterator.hasNext()){
    Question question=iterator.next();
    System.out.println("question id:"+question.getId());
    System.out.println("question name:"+question.getName());
    System.out.println("question posted by:"+question.getUsername());
    System.out.println("answers.....");
    Map<String,String> map=question.getAnswers();
    Set<Map.Entry<String,String>> set=map.entrySet();

    Iterator<Map.Entry<String,String>> iteratoranswer=set.iterator();
    while(iteratoranswer.hasNext()){
        Map.Entry<String,String> entry=(Map.Entry<String,String>)iteratoranswer.next();
        System.out.println("answer name:"+entry.getKey());
        System.out.println("answer posted by:"+entry.getValue());
    }
}
session.close();
}
}

```

## Hibernate One to One Example using Annotation

Here, we are going to perform one to one mapping by one-to-one element using annotation. In such case, no foreign key is created in the primary table.

In this example, one employee can have one address and one address belongs to one employee only. Here, we are using bidirectional association. Let's look at the persistent classes.

### 1) Persistent classes for one to one mapping

There are two persistent classes Employee.java and Address.java. Employee class contains Address class reference and vice versa.

## Employee.java

```
package com.java;
import javax.persistence.*;

@Entity
@Table(name="emp220")
public class Employee {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    @PrimaryKeyJoinColumn
    private int employeeId;
    private String name,email;
    @OneToOne(targetEntity=Address.class,cascade=CascadeType.ALL)
    private Address address;
    public int getEmployeeId() {
        return employeeId;
    }
    public void setEmployeeId(int employeeId) {
        this.employeeId = employeeId;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public Address getAddress() {
```

```

        return address;
    }
    public void setAddress(Address address) {
        this.address = address;
    }
}

```

## Address.java

```

package com.java;
import javax.persistence.*;

@Entity
@Table(name="address220")
public class Address {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int addressId;
    private String addressLine1,city,state,country;
    private int pincode;

    @OneToOne(targetEntity=Employee.class)
    private Employee employee;
    public int getAddressId() {
        return addressId;
    }
    public void setAddressId(int addressId) {
        this.addressId = addressId;
    }
    public String getAddressLine1() {
        return addressLine1;
    }
    public void setAddressLine1(String addressLine1) {
        this.addressLine1 = addressLine1;
    }
}

```



```
public String getCity() {  
    return city;  
}  
public void setCity(String city) {  
    this.city = city;  
}  
public String getState() {  
    return state;  
}  
public void setState(String state) {  
    this.state = state;  
}  
public String getCountry() {  
    return country;  
}  
public void setCountry(String country) {  
    this.country = country;  
}  
public int getPincode() {  
    return pincode;  
}  
public void setPincode(int pincode) {  
    this.pincode = pincode;  
}  
public Employee getEmployee() {  
    return employee;  
}  
public void setEmployee(Employee employee) {  
    this.employee = employee;  
}  
}
```

## 2) Add project information and configuration in pom.xml file.

Open pom.xml file and click source. Now, add the below dependencies between <dependencies>....</dependencies> tag. These dependencies are used to add the jar files in Maven project.

```

<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.3.1.Final</version>
</dependency>
<dependency>
  <groupId>com.oracle</groupId>
  <artifactId>ojdbc14</artifactId>
  <version>10.2.0.4.0</version>
</dependency>

```

### 3) Configuration file

This file contains information about the database and mapping file.

#### hibernate.cfg.xml

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

  <session-factory>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
    <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
    <property name="connection.username">system</property>
    <property name="connection.password">jtp</property>
    <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

    <mapping class="com.java.Address"/>
    <mapping class="com.java.Employee"/>
  </session-factory>

</hibernate-configuration>

```

## 4) User classes to store and fetch the data

### Store.java

```
package com.java;
```

```
import org.hibernate.*;
```

```
import org.hibernate.boot.Metadata;
```

```
import org.hibernate.boot.MetadataSources;
```

```
import org.hibernate.boot.registry.StandardServiceRegistry;
```

```
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
```

```
public class Store {
```

```
public static void main(String[] args) {
```

```
    StandardServiceRegistry ssr=new StandardServiceRegistryBuilder().configure("hibernate.cf  
g.xml").build();
```

```
    Metadata meta=new MetadataSources(ssr).getMetadataBuilder().build();
```

```
    SessionFactory factory=meta.getSessionFactoryBuilder().build();
```

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

```
    Transaction t=session.beginTransaction();
```

```
    Employee e1=new Employee();
```

```
    e1.setName("Ravi Malik");
```

```
    e1.setEmail("ravi@gmail.com");
```

```
    Address address1=new Address();
```

```
    address1.setAddressLine1("G-21,Lohia nagar");
```

```
    address1.setCity("Ghaziabad");
```

```
    address1.setState("UP");
```

```
    address1.setCountry("India");
```

```
    address1.setPincode(201301);
```

```
    e1.setAddress(address1);
```

```
    address1.setEmployee(e1);
```

```

    session.persist(e1);
    t.commit();

    session.close();
    System.out.println("success");
}
}

```

## Output

EMPLOYEEID	EMAIL	NAME	ADDRESS
1	ravi@gmail.com	Ravi Malik	2

ADDRESSID	ADDRESSLINE1	CITY	COUNTRY	PINCODE	STATE
2	G-21, Lohia nagar	Ghaziabad	India	201301	UP

## Fetch.java

```

package com.java;
import java.util.Iterator;
import java.util.List;

import javax.persistence.TypedQuery;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class Fetch {
    public static void main(String[] args) {
        StandardServiceRegistry ssl=new StandardServiceRegistryBuilder().configure("hibernate.cf
g.xml").build();
        Metadata meta=new MetadataSources(ssl).getMetadataBuilder().build();

        SessionFactory factory=meta.getSessionFactoryBuilder().build();
        Session session=factory.openSession();
    }
}

```

```

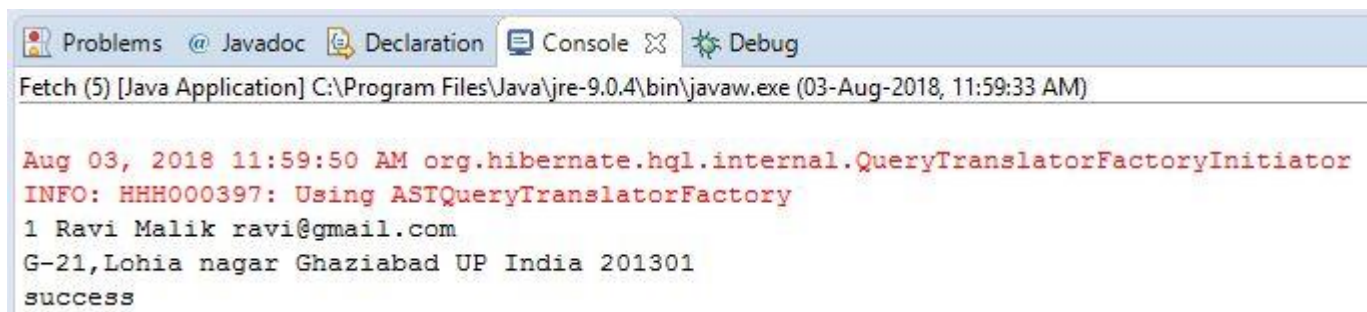
TypedQuery query=session.createQuery("from Employee");
List<Employee> list=query.getResultList();

Iterator<Employee> itr=list.iterator();
while(itr.hasNext()){
    Employee emp=itr.next();
    System.out.println(emp.getEmployeeId()+" "+emp.getName()+" "+emp.getEmail());
    Address address=emp.getAddress();
    System.out.println(address.getAddressLine1()+" "+address.getCity()+" "+
        address.getState()+" "+address.getCountry()+" "+address.getPincode());
}

session.close();
System.out.println("success");
}
}

```

## Output



```

Fetch (5) [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (03-Aug-2018, 11:59:33 AM)

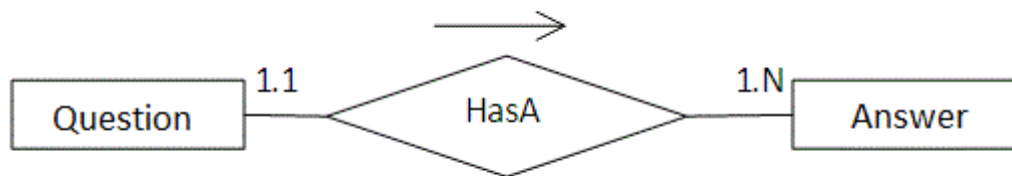
Aug 03, 2018 11:59:50 AM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator
INFO: HHH000397: Using ASTQueryTranslatorFactory
1 Ravi Malik ravi@gmail.com
G-21,Lohia nagar Ghaziabad UP India 201301
success

```

## Hibernate One to Many Example using Annotation

In this section, we will perform one-to-many association to map the list object of persistent class using annotation.

Here, we are using the scenario of Forum where one question has multiple answers.



In such case, there can be many answers for a question and each answer may have its own information that is why we have used list in the persistent class (containing the reference of Answer class) to represent a collection of answers.

## Example of One to Many mapping using annotation

### 1) Create the Persistent class

This persistent class defines properties of the class including List.

#### Question.java

```
package com.java;
import javax.persistence.*;
import java.util.List;

@Entity
@Table(name="q5991")
public class Question {

    @Id
    @GeneratedValue(strategy=GenerationType.TABLE)
    private int id;
    private String qname;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name="qid")
    @OrderColumn(name="type")
    private List<Answer> answers;
```

```

public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public String getQname() {
    return qname;
}
public void setQname(String qname) {
    this.qname = qname;
}
public List<Answer> getAnswers() {
    return answers;
}
public void setAnswers(List<Answer> answers) {
    this.answers = answers;
}
}

```

## Answer.java

```

package com.java;

import javax.persistence.*;

@Entity
@Table(name="ans5991")
public class Answer {
    @Id
    @GeneratedValue(strategy=GenerationType.TABLE)

    private int id;
    private String answername;
    private String postedBy;
    public int getId() {
        return id;
    }
}

```

```

}
public void setId(int id) {
    this.id = id;
}
public String getAnswername() {
    return answername;
}
public void setAnswername(String answername) {
    this.answername = answername;
}
public String getPostedBy() {
    return postedBy;
}
public void setPostedBy(String postedBy) {
    this.postedBy = postedBy;
}
}

```

## 2) Add project information and configuration in pom.xml file.

Open pom.xml file and click source. Now, add the below dependencies between <dependencies> .... </dependencies> tag.

```

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.3.1.Final</version>
</dependency>

```

```

<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc14</artifactId>
    <version>10.2.0.4.0</version>
</dependency>

```

## 3) Create the configuration file

This file contains information about the database and mapping file.



```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

        <mapping class="com.java.Question"/>
    </session-factory>

</hibernate-configuration>

```

## 4) Create the class to store the data

In this class we are storing the data of the question class.

```
package com.java;
```

```
import java.util.ArrayList;
```

```
import org.hibernate.Session;
```

```
import org.hibernate.SessionFactory;
```

```
import org.hibernate.Transaction;
```

```
import org.hibernate.boot.Metadata;
```

```
import org.hibernate.boot.MetadataSources;
```

```
import org.hibernate.boot.registry.StandardServiceRegistry;
```

```
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
```

```
public class StoreData {
```

```
public static void main(String[] args) {
```

```
StandardServiceRegistry ssr=new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
```

```
Metadata meta=new MetadataSources(ssr).getMetadataBuilder().build();
```

```
SessionFactory factory=meta.getSessionFactoryBuilder().build();
```

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

```
Transaction t=session.beginTransaction();
```

```
Answer ans1=new Answer();
```

```
ans1.setAnswername("Java is a programming language");
```

```
ans1.setPostedBy("Ravi Malik");
```

```
Answer ans2=new Answer();
```

```
ans2.setAnswername("Java is a platform");
```

```
ans2.setPostedBy("Sudhir Kumar");
```

```
Answer ans3=new Answer();
```

```
ans3.setAnswername("Servlet is an Interface");
```

```
ans3.setPostedBy("Jai Kumar");
```

```
Answer ans4=new Answer();
```

```
ans4.setAnswername("Servlet is an API");
```

```
ans4.setPostedBy("Arun");
```

```
ArrayList<Answer> list1=new ArrayList<Answer>();
```

```
list1.add(ans1);
```

```
list1.add(ans2);
```

```
ArrayList<Answer> list2=new ArrayList<Answer>();
```

```
list2.add(ans3);
```

```
list2.add(ans4);
```

```
Question question1=new Question();
```

```
question1.setQname("What is Java?");
```

```
question1.setAnswers(list1);
```

```

    Question question2=new Question();
    question2.setQname("What is Servlet?");
    question2.setAnswers(list2);

    session.persist(question1);
    session.persist(question2);

    t.commit();
    session.close();
    System.out.println("success");
}
}

```

**Note** - Using these annotations in a similar way, we can also perform one-to-many association for set, map and bag objects.

### Output

ID	QNAME
1	What is Java?
4	What is Servlet?

ID	ANSWERNAME	POSTEDBY	QID	TYPE
2	Java is a programming language	Ravi Malik	1	0
3	Java is a platform	Sudhir Kumar	1	1
5	Servlet is an Interface	Jai Kumar	4	0
6	Servlet is an API	Arun	4	1

## How to fetch the data of List

Here, we have used HQL to fetch all the records of Question class including answers. In such case, it fetches the data from two tables that are functional dependent.

```

package com.java;
import java.util.*;
import javax.persistence.TypedQuery;
import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class FetchData {
    public static void main(String[] args) {

```

```
StandardServiceRegistry ssr=new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
```

```
Metadata meta=new MetadataSources(ssr).getMetadataBuilder().build();
```

```
SessionFactory factory=meta.getSessionFactoryBuilder().build();
```

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

```
TypedQuery query=session.createQuery("from Question");
```

```
List<Question> list=query.getResultList();
```

```
Iterator<Question> itr=list.iterator();
```

```
while(itr.hasNext()){
```

```
    Question q=itr.next();
```

```
    System.out.println("Question Name: "+q.getQname());
```

```
    //printing answers
```

```
    List<Answer> list2=q.getAnswers();
```

```
    Iterator<Answer> itr2=list2.iterator();
```

```
    while(itr2.hasNext())
```

```
    {
```

```
        Answer a=itr2.next();
```

```
        System.out.println(a.getAnswername()+":"+a.getPostedBy());
```

```
    }
```

```
}
```

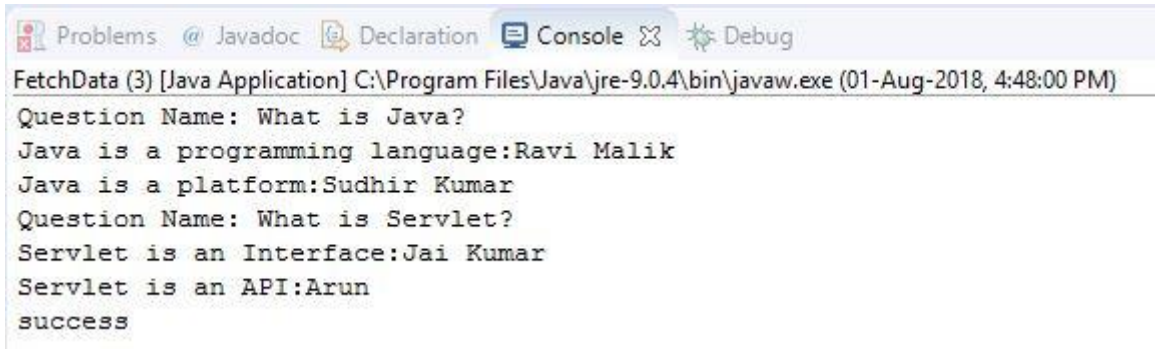
```
session.close();
```

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

```
}
```

```
}
```

## Output



```
Problems @ Javadoc Declaration Console Debug
FetchData (3) [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (01-Aug-2018, 4:48:00 PM)
Question Name: What is Java?
Java is a programming language:Ravi Malik
Java is a platform:Sudhir Kumar
Question Name: What is Servlet?
Servlet is an Interface:Jai Kumar
Servlet is an API:Arun
success
```

## Hibernate Many to One Mapping using Annotation

In many to one mapping, various attributes can be referred to one attribute only.

In this example, every employee has one company address only and one address belongs to many employees. Here, we are going to perform many to one mapping using annotation.

Let's look at the persistent classes

### 1) Persistent classes for one to one mapping

There are two persistent classes Employee.java and Address.java. Employee class contains Address class reference and vice versa.

#### Employee.java

```
package com.java;
import javax.persistence.*;

@Entity
@Table(name="emp107")
public class Employee {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int employeeId;
    private String name,email;
```

```

@ManyToOne(cascade=CascadeType.ALL)
private Address address;
public int getEmployeeId() {
    return employeeId;
}
public void setEmployeeId(int employeeId) {
    this.employeeId = employeeId;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public String getEmail() {
    return email;
}
public void setEmail(String email) {
    this.email = email;
}
public Address getAddress() {
    return address;
}
public void setAddress(Address address) {
    this.address = address;
}
}

```

## Address.java

```
package com.java;
```

```
import javax.persistence.*;
```

```
@Entity
```

```
@Table(name="address107")
```

```
public class Address {
```

```

@Id
@GeneratedValue(strategy=GenerationType.AUTO)
private int addressId;
private String addressLine1,city,state,country;
private int pincode;
@OneToOne(cascade=CascadeType.ALL)
private Employee employee;
public int getAddressId() {
    return addressId;
}
public void setAddressId(int addressId) {
    this.addressId = addressId;
}
public String getAddressLine1() {
    return addressLine1;
}
public void setAddressLine1(String addressLine1) {
    this.addressLine1 = addressLine1;
}
public String getCity() {
    return city;
}
public void setCity(String city) {
    this.city = city;
}
public String getState() {
    return state;
}
public void setState(String state) {
    this.state = state;
}
public String getCountry() {
    return country;
}
public void setCountry(String country) {
    this.country = country;
}

```

```

public int getPincode() {
    return pincode;
}

public void setPincode(int pincode) {
    this.pincode = pincode;
}

public Employee getEmployee() {
    return employee;
}

public void setEmployee(Employee employee) {
    this.employee = employee;
}
}

```

## 2) Add project information and configuration in pom.xml file.

Open pom.xml file and click source. Now, add the below dependencies between <dependencies>....</dependencies> tag. These dependencies are used to add the jar files in Maven project.

```

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.3.1.Final</version>
</dependency>
<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc14</artifactId>
    <version>10.2.0.4.0</version>
</dependency>

```

## 3) Configuration file

This file contains information about the database and mapping file.

### hibernate.cfg.xml

```
<?xml version='1.0' encoding='UTF-8'?>
```



```

<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">create</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

        <mapping class="com.java.Address"/>
        <mapping class="com.java.Employee"/>
    </session-factory>

</hibernate-configuration>

```

## 4) User classes to store and fetch the data

### Store.java

```

package com.java;

import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class Store {
    public static void main(String[] args) {

        StandardServiceRegistry ssr=new StandardServiceRegistryBuilder().configure("hibernate.cf
g.xml").build();

```

```
Metadata meta=new MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory=meta.getSessionFactoryBuilder().build();
Session session=factory.openSession();

Transaction t=session.beginTransaction();

Employee e1=new Employee();
e1.setName("Ravi Malik");
e1.setEmail("ravi@gmail.com");

Employee e2=new Employee();
e2.setName("Anuj Verma");
e2.setEmail("anuj@gmail.com");

Address address1=new Address();
address1.setAddressLine1("G-13,Sector 3");
address1.setCity("Noida");
address1.setState("UP");
address1.setCountry("India");
address1.setPincode(201301);

e1.setAddress(address1);
e2.setAddress(address1);

session.persist(e1);
session.persist(e2);
t.commit();

session.close();
System.out.println("success");
}
}
```

## OUTPUT

EMPLOYEEID	EMAIL	NAME	ADDRESS
1	ravi@gmail.com	Ravi Malik	2
3	anuj@gmail.com	Anuj Verma	2

ADDRESSID	ADDRESSLINE1	CITY	COUNTRY	PINCODE	STATE
2	G-13,Sector 3	Noida	India	201301	UP

## Fetch.java

**package** com.java;

**import** java.util.Iterator;

**import** java.util.List;

**import** javax.persistence.TypedQuery;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.boot.Metadata;

**import** org.hibernate.boot.MetadataSources;

**import** org.hibernate.boot.registry.StandardServiceRegistry;

**import** org.hibernate.boot.registry.StandardServiceRegistryBuilder;

**public class** Fetch {

**public static void** main(String[] args) {

StandardServiceRegistry ssr=**new** StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();

Metadata meta=**new** MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory=meta.getSessionFactoryBuilder().build();

Session session=factory.openSession();

TypedQuery query=session.createQuery("from Employee e");

List<Employee> list=query.getResultList();

Iterator<Employee> itr=list.iterator();

**while**(itr.hasNext()){

Employee emp=itr.next();

System.out.println(emp.getEmployeeid()+" "+emp.getName()+" "+emp.getEmail());

Address address=emp.getAddress();

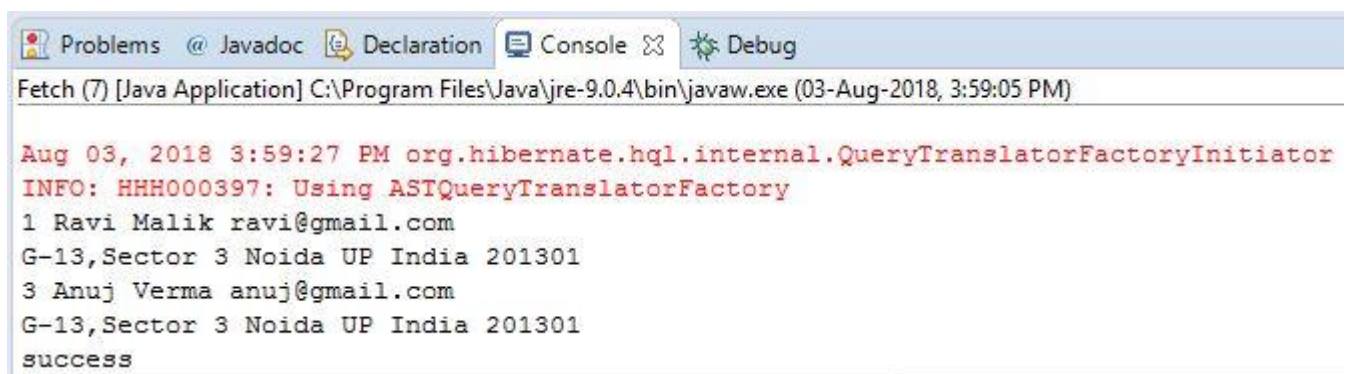
```

        System.out.println(address.getAddressLine1()+" "+address.getCity()+" "+
            address.getState()+" "+address.getCountry()+" "+address.getPincode());
    }

    session.close();
    System.out.println("success");
}
}

```

## Output



```

Fetch (7) [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (03-Aug-2018, 3:59:05 PM)

Aug 03, 2018 3:59:27 PM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator
INFO: HHH000397: Using ASTQueryTranslatorFactory
1 Ravi Malik ravi@gmail.com
G-13,Sector 3 Noida UP India 201301
3 Anuj Verma anuj@gmail.com
G-13,Sector 3 Noida UP India 201301
success

```

# Hibernate Many to Many Example using Annotation

In the previous section, we have performed many to many mapping using XML file. Here, we are going to perform this task using annotation.

We can map many to many relation either using list, set, bag, map etc. Here, we are going to use list for many-to-many mapping. In such case, three tables will be created.

## Example of Many to Many Mapping

In this example, we will generate a many to many relation between questions and answers by list.

### 1) Create the Persistent class

#### Question.java

```
package com.java;
```

```

import java.util.List;
import javax.persistence.*;

@Entity
@Table(name="ques1123")
public class Question {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int id;
    private String qname;

    @ManyToMany(targetEntity = Answer.class, cascade = { CascadeType.ALL })
    @JoinTable(name = "q_ans1123",
        joinColumns = { @JoinColumn(name = "q_id" ) },
        inverseJoinColumns = { @JoinColumn(name = "ans_id" ) })
    private List<Answer> answers;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getQname() {
        return qname;
    }
    public void setQname(String qname) {
        this.qname = qname;
    }
    public List<Answer> getAnswers() {
        return answers;
    }
    public void setAnswers(List<Answer> answers) {
        this.answers = answers;
    }
}

```

## Answer.java

```
package com.java;

import javax.persistence.*;

@Entity
@Table(name="ans1123")
public class Answer {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int id;
    private String answername;
    private String postedBy;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getAnswername() {
        return answername;
    }
    public void setAnswername(String answername) {
        this.answername = answername;
    }
    public String getPostedBy() {
        return postedBy;
    }
    public void setPostedBy(String postedBy) {
        this.postedBy = postedBy;
    }

}
```

2) Add project information and configuration in pom.xml file.

Open pom.xml file and click source. Now, add the below dependencies between <dependencies>....</dependencies> tag.

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.3.1.Final</version>
</dependency>
```

```
<dependency>
  <groupId>com.oracle</groupId>
  <artifactId>ojdbc14</artifactId>
  <version>10.2.0.4.0</version>
</dependency>
```

### 3) Create the configuration file

This file contains information about the database and mapping file.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
  "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

  <session-factory>
    <property name="hbm2ddl.auto">create</property>
    <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
    <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
    <property name="connection.username">system</property>
    <property name="connection.password">jtp</property>
    <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

    <mapping class="com.java.Question"/>
    <mapping class="com.java.Answer"/>
  </session-factory>
```

</hibernate-configuration>

## 4) Create the class to store the data

### StoreData.java

```
package com.java;
import java.util.ArrayList;
import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreData {
    public static void main(String[] args) {

        StandardServiceRegistry ssr=new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
        Metadata meta=new MetadataSources(ssr).getMetadataBuilder().build();
        SessionFactory factory=meta.getSessionFactoryBuilder().build();
        Session session=factory.openSession();
        Transaction t=session.beginTransaction();

        Answer an1=new Answer();
        an1.setAnswername("Java is a programming language");
        an1.setPostedBy("Ravi Malik");

        Answer an2=new Answer();
        an2.setAnswername("Java is a platform");
        an2.setPostedBy("Sudhir Kumar");

        Question q1=new Question();
        q1.setQname("What is Java?");
        ArrayList<Answer> l1=new ArrayList<Answer>();
        l1.add(an1);
        l1.add(an2);
        q1.setAnswers(l1);
```



```

Answer ans3=new Answer();
ans3.setAnswername("Servlet is an Interface");
ans3.setPostedBy("Jai Kumar");

```

```

Answer ans4=new Answer();
ans4.setAnswername("Servlet is an API");
ans4.setPostedBy("Arun");

```

```

Question q2=new Question();
q2.setQname("What is Servlet?");
ArrayList<Answer> l2=new ArrayList<Answer>();
l2.add(ans3);
l2.add(ans4);
q2.setAnswers(l2);

```

```

session.persist(q1);
session.persist(q2);

```

```

t.commit();
session.close();
System.out.println("success");

```

```

}
}

```

## Output

ID	QNAME
1	What is Java?
4	What is Servlet?

ID	ANSWERNAME	POSTEDBY
2	Java is a programming language	Ravi Malik
3	Java is a platform	Sudhir Kumar
5	Servlet is an Interface	Jai Kumar
6	Servlet is an API	Arun