**Start: How to make relationship between two table using hibernate**

# **How to Implement Database Relationship in Hibernate**

The crux of the relational database management system is in the idea of establishing relationship among entities. We establish this relationship in a database with the help of DDL SQL join, foreign keys, etc. With Hibernate and EJB persistence specification we can easily model association between entities. In this article we shall use this ability to model association between entities from Java code, without bothering about the SQL.

## Relationship

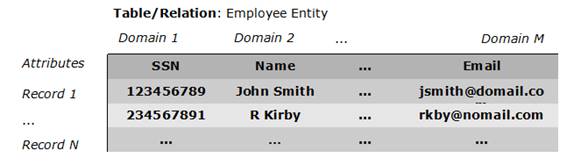
Database relationship has a very concrete mathematical foundation in classical database literature. To make it simple, we keep aside its mathematical counterpart and try to understand the same analogically.

Entities are the key players of any system - for example in a school, teachers, students, and classrooms are entities. In much the same way, in an organization employees, managers, projects, and departments are entities. These entities obviously have some relationship among them such as:

Employees [ works for ] a department

Employees [ works on ] a project

* Now, if Employee is an entity then 'John Smith', for example, is an instance of that entity. So Employeeis nothing but an abstraction of a real entity with some unique attributes. Each abstraction creates a relation/table in the database. And each table has a collection of such instances called records.



Now a collection of several such instances make a relation or table.

But while creating relationships among separate entities, we need to specify the number of times that an entity can participate in a relationship instance. This is called degree of relationship or cardinality ratios. There are three types of relationship degrees:

* One-to-One (1:1) – one employee works for one department.
* One-to-Many (1:N) or Many-to-One (N:1) depending upon from which perspective we are looking at the relationship. One employee works in many projects.
* Many-to-Many (N:M) – Many books are written by many authors.

## Mapping Model Class into Database Tables

In hibernate we can map a model object into a relation/table with the help of @Entity annotation. The member variables of the model object corresponds to the table attributes. The table attributes in this case are Java primitive types, which maps into corresponding database specific primitive types. @Iddefines a primary key into the database and @GeneratedValue specifies that this primary key would be auto generated by the database.

@Entity

public class Employee{

    @Id  @GeneratedValue(strategy=GenerationType.AUTO)

    private long empId;

    private String empName;

    private String address;

    private String sex;

    private float salary;

…

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EmpId (PK)** | **EmpName** | **...** | **Sex** | **salary** |
| 123456789 | John Smith | … | Male | 45000 |
| … | … | … | … | ... |

## Mapping Embedded Objects

In a situation where the member variable of a model object is another object itself, then this encapsulated object is merged into the embedded model class. The @Embedded and @Embeddableannotation are used to manage this type of relationship.

@Entity

public class Employee{

    @Id  @GeneratedValue(strategy=GenerationType.AUTO)

    private long empId;

    private String empName;

    @Embedded

    private Address address;

    private String sex;

    private float salary;

…

}

@Embeddable

public class Employee{

    private String street;

    private String province;

    private String country;

    private String zip;

…

}

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **EmpId (PK)** | **EmpName** | **…** | **street** | **...** | **zip** | **...** | **salary** |
| 123456789 | John Smith | ... | ... | ... | ... | ... | 45000 |
| … | … | ... | ... | ... | ... | ... | ... |

## One-to-One Association

To establish a one-to-one association between two separate object models we use @OneToOneannotation as follows. Here in this example we assume that each employee works in one department, so the cardinality ratio of the relationship is 1:1.

@Entity

public class Employee{

    @Id  @GeneratedValue(strategy=GenerationType.AUTO)

    private long empId;

…

   @OneToOne

   private Department department;

}

@Entity

public class Department{

    @Id  @GeneratedValue

    private long empId;

…

}

## One-to-Many or Many-to-One Association

One-to-Many or Many-to-One association are basically same, seen from alternate perspectives of owning and subordinate entities. The annotation used for this type of relationship is @OneToMany. In the example below we assume that each employee can work in more than one project, so the degree of relationship is 1:N.

@Entity

public class Employee{

    @Id  @GeneratedValue(strategy=GenerationType.AUTO)

    private long empId;

…

   @OneToMany

   private Collection<Project> projects=new ArrayList<>();

}

@Entity

public class Project{

    @Id  @GeneratedValue(strategy=GenerationType.AUTO)

    private long projectId;

…

}

In this case what Hibernate does is that it creates a separate table with a primary key denoted by @Id in model class and uses it to map the attributes in the database table.

**Employee**

|  |  |  |  |
| --- | --- | --- | --- |
| **EmpId (PK)** | **EmpName** | **…** | **salary** |
| 123456789 | John Smith | ... | 45000 |
| … | … | ... | ... |

**Project**

|  |  |  |
| --- | --- | --- |
| **projectId (PK)** | **…** | **startDate** |
| 123456789 | ... | 2/2/2012 |
| … | ... | ... |

**New Table: Employee\_Project**

|  |  |
| --- | --- |
| **EmpId** | **projectId** |
| ... | ... |

Further if we want, we can also make a reverse relationship from Project to Employee by making the following changes in the Project class.

@Entity

public class Project{

    @Id  @GeneratedValue(strategy=GenerationType.AUTO)

    private long projectId;

…

   @ManyToOne

   private Employee employee;

}

## Many-to-Many Association

When more than one entity instance is associated with multiple instances of a separate entity, The relation degree becomes N:M. This type of relationship is established in Hibernate with the help of @ManyToMany annotation. For example, the relationship between Book and Author; more than one author may write one or more books; alternatively, many books are written by one or more author.

@Entity

public class Book{

    @Id  @GeneratedValue(strategy=GenerationType.AUTO)

    private long isbn;

…

   @ManyToMany(mappedBy=”books”)

   private Collection<Author> authors=new ArrayList<>();

}

@Entity

public class Author{

    @Id  @GeneratedValue(strategy=GenerationType.AUTO)

    private long authorId;

…

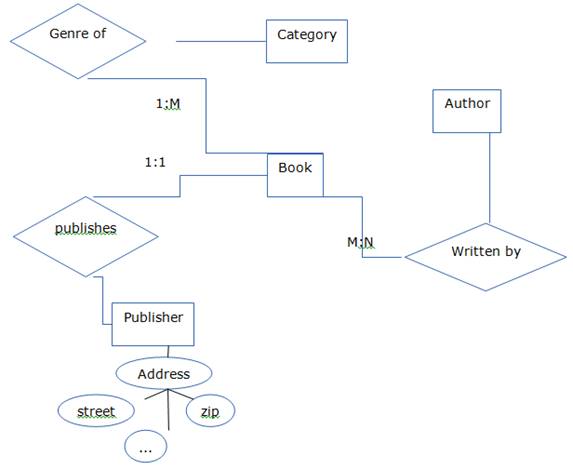
@ManyToMany

   private Collection<Book> books=new ArrayList<>();

}

## A Sample Application

The Java code below demonstrates all three types of relationships.



**Book.java**

//...import statements

@Entity

public class Book implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id

    @GeneratedValue(strategy = GenerationType.AUTO)

    private int id;

    private String isbn;

    private String title;

    @OneToOne

    private Publisher publisher;

    @OneToMany

    private Collection<Category> categories=new ArrayList<>();

    @ManyToMany

    private Collection<Author> authors=new ArrayList<>();

//...getters and setters

}

**Publisher.java**

//...import statements

@Entity

public class Publisher implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id

    @GeneratedValue(strategy = GenerationType.AUTO)

    private int id;

    private String publisherName;

    @Embedded

    private Address publisheAddress;

  //... getters and setters

}

**Address.java**

//...import statements

@Embeddable

public class Address implements Serializable {

    private String street;

    private String province;

    private String zip;

    private String email;

    private String phone;

 //... getters and setters

}

**Authors.java**

//...import statements

@Entity

public class Author implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id

    @GeneratedValue(strategy = GenerationType.AUTO)

    private int id;

    private String authorName;

    private String bio;

    @ManyToMany(mappedBy = "authors")

    private Collection<Book> books = new ArrayList<>();

   //... getters and setters

}

**Category.java**

//...import  statements

@Entity

public class Category implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id

    @GeneratedValue(strategy = GenerationType.AUTO)

    private int id;

    private String categoryName;

    private String description;

//... getters and setters

}

**HibernateUtil.java**

//...import statements

public class HibernateUtil {

     private static final  SessionFactory sessionFactory;

     private static final  ServiceRegistry serviceRegistry;

     static {

          try {

               Configuration config = getConfiguration();

               serviceRegistry = new ServiceRegistryBuilder().applySettings(

                         config.getProperties()).buildServiceRegistry();

               config.setSessionFactoryObserver(new SessionFactoryObserver() {

                    private static final long  serialVersionUID = 1L;

                    @Override

                    public void sessionFactoryCreated(SessionFactory factory) {

                    }

                    @Override

                    public void sessionFactoryClosed(SessionFactory factory) {

                         ServiceRegistryBuilder.destroy(serviceRegistry);

                    }

               });

               sessionFactory = config.buildSessionFactory(serviceRegistry);

          } catch (Throwable ex) {

               System.err.println("Initial SessionFactory creation failed." + ex);

               throw new ExceptionInInitializerError(ex);

          }

     }

     public static  Session openSession() {

          return sessionFactory.openSession();

     }

     private static  Configuration getConfiguration() {

          Configuration cfg = new Configuration();

          cfg.addAnnotatedClass(Book.class );

          cfg.addAnnotatedClass(Author.class );

          cfg.addAnnotatedClass(Category.class );

          cfg.addAnnotatedClass(Publisher.class );

          cfg.setProperty("hibernate.connection.driver\_class","com.mysql.jdbc.Driver");

          cfg.setProperty("hibernate.connection.url","jdbc:mysql://localhost:3306/mydatabase?zeroDateTimeBehavior=convertToNull");

          cfg.setProperty("hibernate.connection.username", "user1");

          cfg.setProperty("hibernate.connection.password", "secret");

          cfg.setProperty("hibernate.show\_sql", "true");

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

          cfg.setProperty("hibernate.hbm2ddl.auto", "create-drop");

          cfg.setProperty("hibernate.cache.provider\_class","org.hibernate.cache.NoCacheProvider");

          cfg.setProperty("hibernate.current\_session\_context\_class", "thread");

          return cfg;

     }

}

**MyLibrary.java**

//...import statements

public class MyLibrary {

    public static void main(String[] args) {

        Book book=new Book();

        book.setIsbn("81-7808-137-7");

        book.setTitle("Fundamentals of Database System");

        Publisher pub=new Publisher();

        pub.setPublisherName("Pearson");

        Address add=new Address();

        add.setStreet("123 ABC Street");

        add.setProvince("XYZ");

        add.setPhone("9282736446");

        add.setEmail("info@pearson.com");

        add.setZip("1010101");

        pub.setPublisheAddress(add);

        book.setPublisher(pub);

        Author a1=new Author();

        a1.setAuthorName("Elmasri");

        a1.setBio("bio not available");

        Author a2=new Author();

        a2.setAuthorName("Navathe");

        a2.setBio("bio not available");

        book.getAuthors().add(a1);

        book.getAuthors().add(a2);

        Category c1=new Category();

        c1.setCategoryName("Database");

        c1.setDescription("not Available");

        Category c2=new Category();

        c2.setCategoryName("Computer");

        c2.setDescription("not Available");

        book.getCategories().add(c1);

        book.getCategories().add(c2);

        Session session=HibernateUtil.openSession();

        session.beginTransaction();

        session.save(pub);

        session.save(a1);

        session.save(a2);

        session.save(c1);

        session.save(c2);

        session.save(book);

        session.getTransaction().commit();

        session.close();

    }

}

## Conclusion

Establishing relationship in a database often requires one to write complex DDL SQL statements. Obviously, we do not even get a hint of such complexity while working with Hibernate. Hibernate also allows one to create relationships using the inheritance mechanism. In a way it provides a solution where database programmers need to learn only one language – Java. This is the power of this excellent ORM tool.

**End**

1. **What is Hibernate Framework?**

**Object-relational mapping** or ORM is the programming technique to map application domain model objects to the relational database tables. Hibernate is java based ORM tool that provides framework for mapping application domain objects to the relational database tables and vice versa.

Hibernate provides reference implementation of Java Persistence API, that makes it a great choice as ORM tool with benefits of loose coupling. We can use Hibernate persistence API for CRUD operations. Hibernate framework provide option to map plain old java objects to traditional database tables with the use of JPA annotations as well as XML based configuration.

Similarly hibernate configurations are flexible and can be done from XML configuration file as well as programmatically. For a quick overview of hibernate framework usage, you can go through [Hibernate Beginners Tutorial](http://www.journaldev.com/2882/hibernate-tutorial-for-beginners-using-xml-annotations-and-property-configurations).

1. **What is Java Persistence API (JPA)?**

Java Persistence API (JPA) provides specification for managing the relational data in applications. Current JPA version 2.1 was started in July 2011 as JSR 338. JPA 2.1 was approved as final on 22 May 2013.

JPA specifications is defined with annotations in javax.persistence package. Using JPA annotation helps us in writing implementation independent code.

1. **What are the important benefits of using Hibernate Framework?**

Some of the important benefits of using hibernate framework are:

* 1. Hibernate eliminates all the boiler-plate code that comes with JDBC and takes care of managing resources, so we can focus on business logic.
  2. Hibernate framework provides support for XML as well as JPA annotations, that makes our code implementation independent.
  3. Hibernate provides a powerful query language (HQL) that is similar to SQL. However, HQL is fully object-oriented and understands concepts like inheritance, polymorphism and association.
  4. Hibernate is an open source project from Red Hat Community and used worldwide. This makes it a better choice than others because learning curve is small and there are tons of online documentations and help is easily available in forums.
  5. Hibernate is easy to integrate with other Java EE frameworks, it’s so popular that Spring Framework provides built-in support for integrating hibernate with Spring applications.
  6. Hibernate supports lazy initialization using proxy objects and perform actual database queries only when it’s required.
  7. Hibernate cache helps us in getting better performance.
  8. For database vendor specific feature, hibernate is suitable because we can also execute native sql queries.

Overall hibernate is the best choice in current market for ORM tool, it contains all the features that you will ever need in an ORM tool.

1. **What are the advantages of Hibernate over JDBC?**

Some of the important advantages of Hibernate framework over JDBC are:

* 1. Hibernate removes a lot of boiler-plate code that comes with JDBC API, the code looks cleaner and readable.
  2. Hibernate supports inheritance, associations and collections. These features are not present with JDBC API.
  3. Hibernate implicitly provides transaction management, in fact most of the queries can’t be executed outside transaction. In JDBC API, we need to write code for transaction management using commit and rollback. Read more at [JDBC Transaction Management](http://www.journaldev.com/2483/jdbc-transaction-management-and-savepoint-example-tutorial).
  4. JDBC API throws SQLException that is a checked exception, so we need to write a lot of try-catch block code. Most of the times it’s redundant in every JDBC call and used for transaction management. Hibernate wraps JDBC exceptions and throw JDBCException or HibernateException un-checked exception, so we don’t need to write code to handle it. Hibernate built-in transaction management removes the usage of try-catch blocks.
  5. Hibernate Query Language (HQL) is more object oriented and close to java programming language. For JDBC, we need to write native sql queries.
  6. Hibernate supports caching that is better for performance, JDBC queries are not cached hence performance is low.
  7. Hibernate provide option through which we can create database tables too, for JDBC tables must exist in the database.
  8. Hibernate configuration helps us in using JDBC like connection as well as JNDI DataSource for connection pool. This is very important feature in enterprise application and completely missing in JDBC API.
  9. Hibernate supports JPA annotations, so code is independent of implementation and easily replaceable with other ORM tools. JDBC code is very tightly coupled with the application.

1. **Name some important interfaces of Hibernate framework?**

Some of the important interfaces of Hibernate framework are:

* 1. **SessionFactory (org.hibernate.SessionFactory)**: SessionFactory is an immutable thread-safe cache of compiled mappings for a single database. We need to initialize SessionFactory once and then we can cache and reuse it. SessionFactory instance is used to get the Session objects for database operations.
  2. **Session (org.hibernate.Session)**: Session is a single-threaded, short-lived object representing a conversation between the application and the persistent store. It wraps JDBC java.sql.Connection and works as a factory for org.hibernate.Transaction. We should open session only when it’s required and close it as soon as we are done using it. Session object is the interface between java application code and hibernate framework and provide methods for CRUD operations.
  3. **Transaction (org.hibernate.Transaction)**: Transaction is a single-threaded, short-lived object used by the application to specify atomic units of work. It abstracts the application from the underlying JDBC or JTA transaction. A org.hibernate.Session might span multiple org.hibernate.Transaction in some cases.

1. **What is hibernate configuration file?**

Hibernate configuration file contains database specific configurations and used to initialize SessionFactory. We provide database credentials or JNDI resource information in the hibernate configuration xml file. Some other important parts of hibernate configuration file is Dialect information, so that hibernate knows the database type and mapping file or class details.

1. **What is hibernate mapping file?**

Hibernate mapping file is used to define the entity bean fields and database table column mappings. We know that JPA annotations can be used for mapping but sometimes XML mapping file comes handy when we are using third party classes and we can’t use annotations.

1. **Name some important annotations used for Hibernate mapping?**

Hibernate supports JPA annotations and it has some other annotations in org.hibernate.annotations package. Some of the important JPA and hibernate annotations used are:

* 1. **javax.persistence.Entity**: Used with model classes to specify that they are entity beans.
  2. **javax.persistence.Table**: Used with entity beans to define the corresponding table name in database.
  3. **javax.persistence.Access**: Used to define the access type, either field or property. Default value is field and if you want hibernate to use getter/setter methods then you need to set it to property.
  4. **javax.persistence.Id**: Used to define the primary key in the entity bean.
  5. **javax.persistence.EmbeddedId**: Used to define composite primary key in the entity bean.
  6. **javax.persistence.Column**: Used to define the column name in database table.
  7. **javax.persistence.GeneratedValue**: Used to define the strategy to be used for generation of primary key. Used in conjunction with javax.persistence.GenerationType enum.
  8. **javax.persistence.OneToOne**: Used to define the one-to-one mapping between two entity beans. We have other similar annotations as OneToMany, ManyToOne and ManyToMany
  9. **org.hibernate.annotations.Cascade**: Used to define the cascading between two entity beans, used with mappings. It works in conjunction with org.hibernate.annotations.CascadeType
  10. **javax.persistence.PrimaryKeyJoinColumn**: Used to define the property for foreign key. Used with org.hibernate.annotations.GenericGenerator and org.hibernate.annotations.Parameter

Here are two classes showing usage of these annotations.

package com.journaldev.hibernate.model;

import javax.persistence.Access;

import javax.persistence.AccessType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToOne;

import javax.persistence.Table;

import org.hibernate.annotations.Cascade;

@Entity

@Table(name = "EMPLOYEE")

@Access(value=AccessType.FIELD)

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "emp\_id")

private long id;

@Column(name = "emp\_name")

private String name;

@OneToOne(mappedBy = "employee")

@Cascade(value = org.hibernate.annotations.CascadeType.ALL)

private Address address;

//getter setter methods

}

package com.journaldev.hibernate.model;

import javax.persistence.Access;

import javax.persistence.AccessType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.OneToOne;

import javax.persistence.PrimaryKeyJoinColumn;

import javax.persistence.Table;

import org.hibernate.annotations.GenericGenerator;

import org.hibernate.annotations.Parameter;

@Entity

@Table(name = "ADDRESS")

@Access(value=AccessType.FIELD)

public class Address {

@Id

@Column(name = "emp\_id", unique = true, nullable = false)

@GeneratedValue(generator = "gen")

@GenericGenerator(name = "gen", strategy = "foreign", parameters = { @Parameter(name = "property", value = "employee") })

private long id;

@Column(name = "address\_line1")

private String addressLine1;

@OneToOne

@PrimaryKeyJoinColumn

private Employee employee;

//getter setter methods

}

1. **What is Hibernate SessionFactory and how to configure it?**

SessionFactory is the factory class used to get the Session objects. SessionFactory is responsible to read the hibernate configuration parameters and connect to the database and provide Session objects. Usually an application has a single SessionFactory instance and threads servicing client requests obtain Session instances from this factory.

The internal state of a SessionFactory is immutable. Once it is created this internal state is set. This internal state includes all of the metadata about Object/Relational Mapping.

SessionFactory also provide methods to get the Class metadata and Statistics instance to get the stats of query executions, second level cache details etc.

1. **Hibernate SessionFactory is thread safe?**

Internal state of SessionFactory is immutable, so it’s thread safe. Multiple threads can access it simultaneously to get Session instances.

1. **What is Hibernate Session and how to get it?**

Hibernate Session is the interface between java application layer and hibernate. This is the core interface used to perform database operations. Lifecycle of a session is bound by the beginning and end of a transaction.

Session provide methods to perform create, read, update and delete operations for a persistent object. We can execute HQL queries, SQL native queries and create criteria using Session object.

1. **Hibernate Session is thread safe?**

Hibernate Session object is not thread safe, every thread should get it’s own session instance and close it after it’s work is finished.

1. **What is difference between openSession and getCurrentSession?**

Hibernate SessionFactory getCurrentSession() method returns the session bound to the context. But for this to work, we need to configure it in hibernate configuration file. Since this session object belongs to the hibernate context, we don’t need to close it. Once the session factory is closed, this session object gets closed.

<property name="hibernate.current\_session\_context\_class">thread</property>

Hibernate SessionFactory openSession() method always opens a new session. We should close this session object once we are done with all the database operations. We should open a new session for each request in multi-threaded environment.

There is another method openStatelessSession() that returns stateless session, for more details with examples please read [Hibernate openSession vs getCurrentSession](http://www.journaldev.com/3522/hibernate-sessionfactory-opensession-vs-getcurrentsession-vs-openstatelesssession).

1. **What is difference between Hibernate Session get() and load() method?**

Hibernate session comes with different methods to load data from database. get and load are most used methods, at first look they seems similar but there are some differences between them.

* 1. get() loads the data as soon as it’s called whereas load() returns a proxy object and loads data only when it’s actually required, so load() is better because it support lazy loading.
  2. Since load() throws exception when data is not found, we should use it only when we know data exists.
  3. We should use get() when we want to make sure data exists in the database.

For clarification regarding the differences, please read [Hibernate get vs load](http://www.journaldev.com/3472/hibernate-session-get-vs-load-difference-with-examples).

1. **What is hibernate caching? Explain Hibernate first level cache?**

As the name suggests, hibernate caches query data to make our application faster. Hibernate Cache can be very useful in gaining fast application performance if used correctly. The idea behind cache is to reduce the number of database queries, hence reducing the throughput time of the application.

Hibernate first level cache is associated with the Session object. Hibernate first level cache is enabled by default and there is no way to disable it. However hibernate provides methods through which we can delete selected objects from the cache or clear the cache completely.  
Any object cached in a session will not be visible to other sessions and when the session is closed, all the cached objects will also be lost.

For better explanation, please read [Hibernate First Level Cache](http://www.journaldev.com/2969/hibernate-first-level-cache-example-with-explanation).

1. **How to configure Hibernate Second Level Cache using EHCache?**

One of the major benefit of using Hibernate in large application is it’s support for cache, hence reducing database queries and better performance. In earlier example, we looked into the [Hibernate First Level Cache](https://www.journaldev.com/2969/hibernate-caching-first-level-cache) and today we will look into Hibernate Second Level Cache using **Hibernate EHCache**implementation.

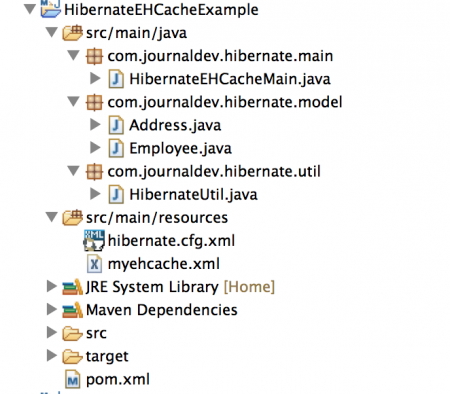
Hibernate Second Level cache providers include EHCache and Infinispan, but EHCache is more popular and we will use it for our example project. However before we move to our project, we should know different strategies for caching an object.

1. **Read Only**: This caching strategy should be used for persistent objects that will always read but never updated. It’s good for reading and caching application configuration and other static data that are never updated. This is the simplest strategy with best performance because there is no overload to check if the object is updated in database or not.
2. **Read Write**: It’s good for persistent objects that can be updated by the hibernate application. However if the data is updated either through backend or other applications, then there is no way hibernate will know about it and data might be stale. So while using this strategy, make sure you are using Hibernate API for updating the data.
3. **Nonrestricted Read Write**: If the application only occasionally needs to update data and strict transaction isolation is not required, a nonstrict-read-write cache might be appropriate.
4. **Transactional**: The transactional cache strategy provides support for fully transactional cache providers such as JBoss TreeCache. Such a cache can only be used in a JTA environment and you must specify hibernate.transaction.manager\_lookup\_class.

## Hibernate EHCache

Since EHCache supports all the above cache strategies, it’s the best choice when you are looking for second level cache in hibernate. I would not go into much detail about EHCache, my main focus will be to get it working for hibernate application.

Create a maven project in the Eclipse or your favorite IDE, final implementation will look like below image.

[](https://cdn.journaldev.com/wp-content/uploads/2014/06/Hibernate-EHCache-Example.png)

Let’s look into each component of the application one by one.

### Hibernate EHCache Maven Dependencies

For hibernate second level cache, we would need to add **ehcache-core** and **hibernate-ehcache**dependencies in our application. EHCache uses slf4j for logging, so I have also added **slf4j-simple** for logging purposes. I am using the latest versions of all these APIs, there is a slight chance that hibernate-ehcache APIs are not compatible with the ehcache-core API, in that case you need to check the pom.xml of hibernate-ehcache to find out the correct version to use. Our final pom.xml looks like below.

<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.journaldev.hibernate</groupId>

<artifactId>HibernateEHCacheExample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<description>Hibernate Secondary Level Cache Example using EHCache implementation</description>

<dependencies>

<!-- Hibernate Core API -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>4.3.5.Final</version>

</dependency>

<!-- MySQL Driver -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.0.5</version>

</dependency>

<!-- EHCache Core APIs -->

<dependency>

<groupId>net.sf.ehcache</groupId>

<artifactId>ehcache-core</artifactId>

<version>2.6.9</version>

</dependency>

<!-- Hibernate EHCache API -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-ehcache</artifactId>

<version>4.3.5.Final</version>

</dependency>

<!-- EHCache uses slf4j for logging -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-simple</artifactId>

<version>1.7.5</version>

</dependency>

</dependencies>

</project>

### Hibernate Second Level Cache – Hibernate EHCache Configuration

Hibernate Second level cache is disabled by default, so we would need to enable it and add some configurations to get it working. Our hibernate.cfg.xml file looks like below.

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-configuration SYSTEM "classpath://org/hibernate/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>

<property name="hibernate.connection.password">pankaj123</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost/TestDB</property>

<property name="hibernate.connection.username">pankaj</property>

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<property name="hibernate.current\_session\_context\_class">thread</property>

<property name="hibernate.show\_sql">true</property>

<property name="hibernate.cache.region.factory\_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>

<!-- For [singleton](https://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-examples) factory -->

<!-- <property name="hibernate.cache.region.factory\_class">org.hibernate.cache.ehcache.SingletonEhCacheRegionFactory</property>

-->

<!-- enable second level cache and query cache -->

<property name="hibernate.cache.use\_second\_level\_cache">true</property>

<property name="hibernate.cache.use\_query\_cache">true</property>

<property name="net.sf.ehcache.configurationResourceName">/myehcache.xml</property>

<mapping class="com.journaldev.hibernate.model.Employee" />

<mapping class="com.journaldev.hibernate.model.Address" />

</session-factory>

</hibernate-configuration>

Some important points about hibernate second level cache configurations are:

1. **hibernate.cache.region.factory\_class** is used to define the Factory class for Second level caching, I am using org.hibernate.cache.ehcache.EhCacheRegionFactory for this. If you want the factory class to be singleton, you should use org.hibernate.cache.ehcache.SingletonEhCacheRegionFactory class.

If you are using Hibernate 3, corresponding classes will be net.sf.ehcache.hibernate.EhCacheRegionFactory and net.sf.ehcache.hibernate.SingletonEhCacheRegionFactory.

1. **hibernate.cache.use\_second\_level\_cache** is used to enable the second level cache.
2. **hibernate.cache.use\_query\_cache** is used to enable the query cache, without it HQL queries results will not be cached.
3. **net.sf.ehcache.configurationResourceName** is used to define the EHCache configuration file location, it’s an optional parameter and if it’s not present EHCache will try to locate ehcache.xml file in the application classpath.

### Hibernate EHCache Configuration File

Our EHCache configuration file myehcache.xml looks like below.

<?xml version="1.0" encoding="UTF-8"?>

<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="ehcache.xsd" updateCheck="true"

monitoring="autodetect" dynamicConfig="true">

<diskStore path="java.io.tmpdir/ehcache" />

<defaultCache maxEntriesLocalHeap="10000" eternal="false"

timeToIdleSeconds="120" timeToLiveSeconds="120" diskSpoolBufferSizeMB="30"

maxEntriesLocalDisk="10000000" diskExpiryThreadIntervalSeconds="120"

memoryStoreEvictionPolicy="LRU" statistics="true">

<persistence strategy="localTempSwap" />

</defaultCache>

<cache name="employee" maxEntriesLocalHeap="10000" eternal="false"

timeToIdleSeconds="5" timeToLiveSeconds="10">

<persistence strategy="localTempSwap" />

</cache>

<cache name="org.hibernate.cache.internal.StandardQueryCache"

maxEntriesLocalHeap="5" eternal="false" timeToLiveSeconds="120">

<persistence strategy="localTempSwap" />

</cache>

<cache name="org.hibernate.cache.spi.UpdateTimestampsCache"

maxEntriesLocalHeap="5000" eternal="true">

<persistence strategy="localTempSwap" />

</cache>

</ehcache>

Hibernate EHCache provides a lot of options, I won’t go into much detail but some of the important configurations above are:

1. **diskStore**: EHCache stores data into memory but when it starts overflowing, it start writing data into file system. We use this property to define the location where EHCache will write the overflown data.
2. **defaultCache**: It’s a mandatory configuration, it is used when an Object need to be cached and there are no caching regions defined for that.
3. **cache name=”employee”**: We use cache element to define the region and it’s configurations. We can define multiple regions and their properties, while defining model beans cache properties, we can also define region with caching strategies. The cache properties are easy to understand and clear with the name.
4. Cache regions org.hibernate.cache.internal.StandardQueryCache and org.hibernate.cache.spi.UpdateTimestampsCache are defined because EHCache was giving warning to that.

### Hibernate Second Level Cache – Model Bean Caching Strategy

We use org.hibernate.annotations.Cache annotation to provide the caching configuration. org.hibernate.annotations.CacheConcurrencyStrategy is used to define the caching strategy and we can also define the cache region to use for the model beans.

package com.journaldev.hibernate.model;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.OneToOne;

import javax.persistence.PrimaryKeyJoinColumn;

import javax.persistence.Table;

import org.hibernate.annotations.Cache;

import org.hibernate.annotations.CacheConcurrencyStrategy;

import org.hibernate.annotations.GenericGenerator;

import org.hibernate.annotations.Parameter;

@Entity

@Table(name = "ADDRESS")

@Cache(usage=CacheConcurrencyStrategy.READ\_ONLY, region="employee")

public class Address {

@Id

@Column(name = "emp\_id", unique = true, nullable = false)

@GeneratedValue(generator = "gen")

@GenericGenerator(name = "gen", strategy = "foreign",

parameters = { @Parameter(name = "property", value = "employee") })

private long id;

@Column(name = "address\_line1")

private String addressLine1;

@Column(name = "zipcode")

private String zipcode;

@Column(name = "city")

private String city;

@OneToOne

@PrimaryKeyJoinColumn

private Employee employee;

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public String getAddressLine1() {

return addressLine1;

}

public void setAddressLine1(String addressLine1) {

this.addressLine1 = addressLine1;

}

public String getZipcode() {

return zipcode;

}

public void setZipcode(String zipcode) {

this.zipcode = zipcode;

}

public String getCity() {

return city;

}

public void setCity(String city) {

this.city = city;

}

public Employee getEmployee() {

return employee;

}

public void setEmployee(Employee employee) {

this.employee = employee;

}

}

package com.journaldev.hibernate.model;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToOne;

import javax.persistence.Table;

import org.hibernate.annotations.Cache;

import org.hibernate.annotations.CacheConcurrencyStrategy;

import org.hibernate.annotations.Cascade;

@Entity

@Table(name = "EMPLOYEE")

@Cache(usage=CacheConcurrencyStrategy.READ\_ONLY, region="employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "emp\_id")

private long id;

@Column(name = "emp\_name")

private String name;

@Column(name = "emp\_salary")

private double salary;

@OneToOne(mappedBy = "employee")

@Cascade(value = org.hibernate.annotations.CascadeType.ALL)

private Address address;

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public Address getAddress() {

return address;

}

public void setAddress(Address address) {

this.address = address;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

}

Note that I am using the same database setup as in [HQL example](https://www.journaldev.com/2954/hibernate-query-language-hql-example-tutorial), you might want to check that to create the database tables and load sample data.

### Hibernate SessionFactory Utility Class

We have a simple utility class to configure hibernate and get the SessionFactory singleton instance.

package com.journaldev.hibernate.util;

import org.hibernate.SessionFactory;

import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

import org.hibernate.cfg.Configuration;

import org.hibernate.service.ServiceRegistry;

public class HibernateUtil {

private static SessionFactory sessionFactory;

private static SessionFactory buildSessionFactory() {

try {

// Create the SessionFactory from hibernate.cfg.xml

Configuration configuration = new Configuration();

configuration.configure("hibernate.cfg.xml");

System.out.println("Hibernate Configuration loaded");

ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder().applySettings(configuration.getProperties()).build();

System.out.println("Hibernate serviceRegistry created");

SessionFactory sessionFactory = configuration.buildSessionFactory(serviceRegistry);

return sessionFactory;

}

catch (Throwable ex) {

System.err.println("Initial SessionFactory creation failed." + ex);

ex.printStackTrace();

throw new ExceptionInInitializerError(ex);

}

}

public static SessionFactory getSessionFactory() {

if(sessionFactory == null) sessionFactory = buildSessionFactory();

return sessionFactory;

}

}

Our hibernate second level cache project using Hibernate EHCache is ready, let’s write a simple program to test it.

### Hibernate EHCache Test Program

package com.journaldev.hibernate.main;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.stat.Statistics;

import com.journaldev.hibernate.model.Employee;

import com.journaldev.hibernate.util.HibernateUtil;

public class HibernateEHCacheMain {

public static void main(String[] args) {

System.out.println("Temp Dir:"+System.getProperty("java.io.tmpdir"));

//Initialize Sessions

SessionFactory sessionFactory = HibernateUtil.getSessionFactory();

Statistics stats = sessionFactory.getStatistics();

System.out.println("Stats enabled="+stats.isStatisticsEnabled());

stats.setStatisticsEnabled(true);

System.out.println("Stats enabled="+stats.isStatisticsEnabled());

Session session = sessionFactory.openSession();

Session otherSession = sessionFactory.openSession();

Transaction transaction = session.beginTransaction();

Transaction otherTransaction = otherSession.beginTransaction();

printStats(stats, 0);

Employee emp = (Employee) session.load(Employee.class, 1L);

printData(emp, stats, 1);

emp = (Employee) session.load(Employee.class, 1L);

printData(emp, stats, 2);

//clear first level cache, so that second level cache is used

session.evict(emp);

emp = (Employee) session.load(Employee.class, 1L);

printData(emp, stats, 3);

emp = (Employee) session.load(Employee.class, 3L);

printData(emp, stats, 4);

emp = (Employee) otherSession.load(Employee.class, 1L);

printData(emp, stats, 5);

//Release resources

transaction.commit();

otherTransaction.commit();

sessionFactory.close();

}

private static void printStats(Statistics stats, int i) {

System.out.println("\*\*\*\*\* " + i + " \*\*\*\*\*");

System.out.println("Fetch Count="

+ stats.getEntityFetchCount());

System.out.println("Second Level Hit Count="

+ stats.getSecondLevelCacheHitCount());

System.out

.println("Second Level Miss Count="

+ stats

.getSecondLevelCacheMissCount());

System.out.println("Second Level Put Count="

+ stats.getSecondLevelCachePutCount());

}

private static void printData(Employee emp, Statistics stats, int count) {

System.out.println(count+":: Name="+emp.getName()+", Zipcode="+emp.getAddress().getZipcode());

printStats(stats, count);

}

}

org.hibernate.stat.Statistics provides the statistics of [Hibernate SessionFactory](https://www.journaldev.com/3522/hibernate-sessionfactory), we are using it to print the fetch count and second level cache hit, miss and put count. Statistics are disabled by default for better performance, that’s why I am enabling it at the start of the program.

When we run above program, we get a lot of output generated by Hibernate and EHCache APIs, but we are interested in the data that we are printing. A sample run prints following output.

Temp Dir:/var/folders/h4/q73jjy0902g51wkw0w69c0600000gn/T/

Hibernate Configuration loaded

Hibernate serviceRegistry created

Stats enabled=false

Stats enabled=true

\*\*\*\*\* 0 \*\*\*\*\*

Fetch Count=0

Second Level Hit Count=0

Second Level Miss Count=0

Second Level Put Count=0

Hibernate: select employee0\_.emp\_id as emp\_id1\_1\_0\_, employee0\_.emp\_name as emp\_name2\_1\_0\_, employee0\_.emp\_salary as emp\_sala3\_1\_0\_, address1\_.emp\_id as emp\_id1\_0\_1\_, address1\_.address\_line1 as address\_2\_0\_1\_, address1\_.city as city3\_0\_1\_, address1\_.zipcode as zipcode4\_0\_1\_ from EMPLOYEE employee0\_ left outer join ADDRESS address1\_ on employee0\_.emp\_id=address1\_.emp\_id where employee0\_.emp\_id=?

1:: Name=Pankaj, Zipcode=95129

\*\*\*\*\* 1 \*\*\*\*\*

Fetch Count=1

Second Level Hit Count=0

Second Level Miss Count=1

Second Level Put Count=2

2:: Name=Pankaj, Zipcode=95129

\*\*\*\*\* 2 \*\*\*\*\*

Fetch Count=1

Second Level Hit Count=0

Second Level Miss Count=1

Second Level Put Count=2

3:: Name=Pankaj, Zipcode=95129

\*\*\*\*\* 3 \*\*\*\*\*

Fetch Count=1

Second Level Hit Count=2

Second Level Miss Count=1

Second Level Put Count=2

Hibernate: select employee0\_.emp\_id as emp\_id1\_1\_0\_, employee0\_.emp\_name as emp\_name2\_1\_0\_, employee0\_.emp\_salary as emp\_sala3\_1\_0\_, address1\_.emp\_id as emp\_id1\_0\_1\_, address1\_.address\_line1 as address\_2\_0\_1\_, address1\_.city as city3\_0\_1\_, address1\_.zipcode as zipcode4\_0\_1\_ from EMPLOYEE employee0\_ left outer join ADDRESS address1\_ on employee0\_.emp\_id=address1\_.emp\_id where employee0\_.emp\_id=?

4:: Name=Lisa, Zipcode=560100

\*\*\*\*\* 4 \*\*\*\*\*

Fetch Count=2

Second Level Hit Count=2

Second Level Miss Count=2

Second Level Put Count=4

5:: Name=Pankaj, Zipcode=95129

\*\*\*\*\* 5 \*\*\*\*\*

Fetch Count=2

Second Level Hit Count=4

Second Level Miss Count=2

Second Level Put Count=4

As you can see from output, statistics were disabled at first but we enabled it for checking our hibernate second level cache.

Step by step explanation of the output is as follows:

1. Before we load any data in our application, all the stats are 0 as expected.
2. When we are loading the Employee with id=1 for the first time, it’s first searched into first level cache and then second level cache. If not found in cache, database query is executed and hence fetch count becomes 1. Once the object is loaded, it’s saved into first level cache and second level cache both. So secondary level hit count remains 0 and miss count becomes 1. Notice that put count is 2, that is because Employee object consists of Address too, so both the objects are saved into second level cache and count is increased to 2.
3. Next, we are again loading the employee with id=1, this time it’s present in the first level cache. So you don’t see any database query and all other secondary level cache stats also remains same.
4. Next we are using evict() method to remove the employee object from the first level cache, now when we are trying to load it, hibernate finds it in the second level cache. That’s why no database query is fired and fetch count remains 1. Notice that hit count goes from 0 to 2 because both Employee and Address objects are read from the second level cache. Second level miss and put count remains at the earlier value.
5. Next we are loading an employee with id=3, database query is executed and fetch count increases to 2, miss count increases from 1 to 2 and put count increases from 2 to 4.
6. Next we are trying to load employee with id=1 in another session, Since hibernate second level cache is shared across sessions, it’s found in the second level cache and no database query is executed. Fetch count, miss count and put count remains same whereas hit count increases from 2 to 4.

So it’s clear that our Hibernate second level cache; Hibernate EHCache; is working fine. Hibernate statistics are helpful in finding the bottleneck in the system and optimize it to reduce the fetch count and load more data from the cache.

EHCache is the best choice for utilizing hibernate second level cache. Following steps are required to enable EHCache in hibernate application.

* 1. Add hibernate-ehcache dependency in your maven project, if it’s not maven then add corresponding jars.
  2. <dependency>
  3. <groupId>org.hibernate</groupId>
  4. <artifactId>hibernate-ehcache</artifactId>
  5. <version>4.3.5.Final</version>
  6. </dependency>
  7. Add below properties in hibernate configuration file.
  8. <property name="hibernate.cache.region.factory\_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>
  10. <!-- For singleton factory -->
  11. <!-- <property name="hibernate.cache.region.factory\_class">org.hibernate.cache.ehcache.SingletonEhCacheRegionFactory</property>
  12. -->
  14. <!-- enable second level cache and query cache -->
  15. <property name="hibernate.cache.use\_second\_level\_cache">true</property>
  16. <property name="hibernate.cache.use\_query\_cache">true</property>
  17. <property name="net.sf.ehcache.configurationResourceName">/myehcache.xml</property>
  18. Create EHCache configuration file, a sample file would look like below.
  19. <?xml version="1.0" encoding="UTF-8"?>
  20. <ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  21. xsi:noNamespaceSchemaLocation="ehcache.xsd" updateCheck="true"
  22. monitoring="autodetect" dynamicConfig="true">
  24. <diskStore path="java.io.tmpdir/ehcache" />
  26. <defaultCache maxEntriesLocalHeap="10000" eternal="false"
  27. timeToIdleSeconds="120" timeToLiveSeconds="120" diskSpoolBufferSizeMB="30"
  28. maxEntriesLocalDisk="10000000" diskExpiryThreadIntervalSeconds="120"
  29. memoryStoreEvictionPolicy="LRU" statistics="true">
  30. <persistence strategy="localTempSwap" />
  31. </defaultCache>
  33. <cache name="employee" maxEntriesLocalHeap="10000" eternal="false"
  34. timeToIdleSeconds="5" timeToLiveSeconds="10">
  35. <persistence strategy="localTempSwap" />
  36. </cache>
  38. <cache name="org.hibernate.cache.internal.StandardQueryCache"
  39. maxEntriesLocalHeap="5" eternal="false" timeToLiveSeconds="120">
  40. <persistence strategy="localTempSwap" />
  41. </cache>
  43. <cache name="org.hibernate.cache.spi.UpdateTimestampsCache"
  44. maxEntriesLocalHeap="5000" eternal="true">
  45. <persistence strategy="localTempSwap" />
  46. </cache>
  47. </ehcache>
  48. Annotate entity beans with @Cache annotation and caching strategy to use. For example,
  49. import org.hibernate.annotations.Cache;
  50. import org.hibernate.annotations.CacheConcurrencyStrategy;
  51. @Entity
  52. @Table(name = "ADDRESS")
  53. @Cache(usage=CacheConcurrencyStrategy.READ\_ONLY, region="employee")
  54. public class Address {
  55. }

That’s it, we are done. Hibernate will use the EHCache for second level caching, read [Hibernate EHCache Example](http://www.journaldev.com/2980/hibernate-ehcache-second-level-caching-example-tutorial) for a complete example with explanation.

1. **What are different states of an entity bean?**

## Hibernate Object States

There are three types of Hibernate object states.

## 1. Transient Object State:

An object which is not associated with hibernate session and does not represent a row in the database is considered as transient. It will be garbage collected if no other object refers to it.

An object that is created for the first time using the new() operator is in transient state. When the object is in transient sate then it will not contain any identifier (primary key value). You have to use save, persist or saveOrUpdate methods to persist the transient object.

* [Hibernate Tutorial](http://javawebtutor.com/articles/hibernate/hibernate_index.php)
* [Hibernate Basics](http://javawebtutor.com/articles/hibernate/hibernate-object-states.php)
* [Hibernate Inheritance Mapping](http://javawebtutor.com/articles/hibernate/hibernate-object-states.php)
* [Hibernate Query Language (HQL)](http://javawebtutor.com/articles/hibernate/hibernate-object-states.php)
* [Composite Primary Keys In Hibernate](http://javawebtutor.com/articles/hibernate/hibernate-composite-key.php)

[?](http://javawebtutor.com/articles/hibernate/hibernate-object-states.php)

|  |  |
| --- | --- |
| 1  2  3 | Employee emp = new Employee();  emp.setName("Ravi Raj");  // here emp object is in a transient state |

## 2. Persistent Object State:

An object that is associated with the hibernate session is called as Persistent object. When the object is in persistent state, then it represent one row of the database and consists of an identifier value.You can make a transient instance persistent by associating it with a Session.

[?](http://javawebtutor.com/articles/hibernate/hibernate-object-states.php)

|  |  |
| --- | --- |
| 1  2 | Long id = (Long) session.save(emp);  // emp object is now in a persistent state |

## 3. Detached Object State:

Object which is just removed from hibernate session is called as detached object.The sate of the detached object is called as detached state.

When the object is in detached sate then it contain identity but you can’t do persistence operation with that identity.

Any changes made to the detached objects are not saved to the database. The detached object can be reattached to the new session and save to the database using update, saveOrUpdate and merge methods.

[?](http://javawebtutor.com/articles/hibernate/hibernate-object-states.php)

|  |  |
| --- | --- |
| 1  2 | session.close();  //object in detached state |

## Example

[?](http://javawebtutor.com/articles/hibernate/hibernate-object-states.php)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | package com.jwt.hibernate.test;    import org.hibernate.Session;  import org.hibernate.Transaction;  import org.hibernate.cfg.AnnotationConfiguration;    import com.jwt.hibernate.Student;    public class ObjectStatesDemo {      public static void main(String[] args) {            // Transient object state          Student student = new Student();          student.setId(101);          student.setName("Mukesh");          student.setRoll("10");          student.setDegree("B.E");          student.setPhone("9999");          // Transient object state          Session session = new AnnotationConfiguration().configure()                  .buildSessionFactory().openSession();          Transaction t = session.beginTransaction();          // Persistent object state          session.save(student);          t.commit();          // Persistent object state          session.close();          // Detached object state        }  } |

## Output :

Hibernate:

insert

into

STUDENT

(degree, name, phone, roll, id)

values

(?, ?, ?, ?, ?)

## In The Database :

mysql> select \* from student;

+-----+--------+--------+-------+------+

| id | degree | name | phone | roll |

+-----+--------+--------+-------+------+

| 101 | B.E | Mukesh | 9999 | 10 |

+-----+--------+--------+-------+------+

1 row in set (0.05 sec)

An entity bean instance can exist is one of the three states.

* 1. **Transient**: When an object is never persisted or associated with any session, it’s in transient state. Transient instances may be made persistent by calling save(), persist() or saveOrUpdate(). Persistent instances may be made transient by calling delete().
  2. **Persistent**: When an object is associated with a unique session, it’s in persistent state. Any instance returned by a get() or load() method is persistent.
  3. **Detached**: When an object is previously persistent but not associated with any session, it’s in detached state. Detached instances may be made persistent by calling update(), saveOrUpdate(), lock() or replicate(). The state of a transient or detached instance may also be made persistent as a new persistent instance by calling merge().

1. **What is use of Hibernate Session merge() call?**

Hibernate merge can be used to update existing values, however this method create a copy from the passed entity object and return it. The returned object is part of persistent context and tracked for any changes, passed object is not tracked. For example program, read [Hibernate merge](http://www.journaldev.com/3481/hibernate-save-vs-saveorupdate-vs-persist-vs-merge-vs-update-explanation-with-examples).

1. **What is difference between Hibernate save(), saveOrUpdate() and persist() methods?**

Hibernate save can be used to save entity to database. Problem with save() is that it can be invoked without a transaction and if we have mapping entities, then only the primary object gets saved causing data inconsistencies. Also save returns the generated id immediately.

Hibernate persist is similar to save with transaction. I feel it’s better than save because we can’t use it outside the boundary of transaction, so all the object mappings are preserved. Also persist doesn’t return the generated id immediately, so data persistence happens when needed.

Hibernate saveOrUpdate results into insert or update queries based on the provided data. If the data is present in the database, update query is executed. We can use saveOrUpdate() without transaction also, but again you will face the issues with mapped objects not getting saved if session is not flushed. For example usage of these methods, read [Hibernate save vs persist](http://www.journaldev.com/3481/hibernate-save-vs-saveorupdate-vs-persist-vs-merge-vs-update-explanation-with-examples).

1. **What will happen if we don’t have no-args constructor in Entity bean?**

Hibernate uses [Reflection API](http://www.journaldev.com/1789/java-reflection-tutorial-for-classes-methods-fields-constructors-annotations-and-much-more) to create instance of Entity beans, usually when you call get() or load() methods. The method Class.newInstance() is used for this and it requires no-args constructor. So if you won’t have no-args constructor in entity beans, hibernate will fail to instantiate it and you will get HibernateException.

1. **What is difference between sorted collection and ordered collection, which one is better?**

When we use Collection API sorting algorithms to sort a collection, it’s called sorted list. For small collections, it’s not much of an overhead but for larger collections it can lead to slow performance and OutOfMemory errors. Also the entity beans should implement Comparable or Comparator interface for it to work, read more at [java object list sorting](http://www.journaldev.com/780/java-comparable-and-comparator-example-to-sort-objects).

If we are using Hibernate framework to load collection data from database, we can use it’s Criteria API to use “order by” clause to get ordered list. Below code snippet shows you how to get it.

List<Employee> empList = session.createCriteria(Employee.class)

.addOrder(Order.desc("id")).list();

Ordered list is better than sorted list because the actual sorting is done at database level, that is fast and doesn’t cause memory issues.

1. **What are the collection types in Hibernate?**

There are five collection types in hibernate used for one-to-many relationship mappings.

* 1. Bag
  2. Set
  3. List
  4. Array
  5. Map

1. **How to implement Joins in Hibernate?**

There are various ways to implement joins in hibernate.

* 1. Using associations such as one-to-one, one-to-many etc.
  2. Using JOIN in the HQL query. There is another form “join fetch” to load associated data simultaneously, no lazy loading.
  3. We can fire native sql query and use join keyword.

1. **Why we should not make Entity Class final?**

Hibernate use proxy classes for lazy loading of data, only when it’s needed. This is done by extending the entity bean, if the entity bean will be final then lazy loading will not be possible, hence low performance.

1. **What is HQL and what are it’s benefits?**

Hibernate Framework comes with a powerful object-oriented query language – Hibernate Query Language (HQL). It’s very similar to SQL except that we use Objects instead of table names, that makes it more close to object oriented programming.

Hibernate query language is case-insensitive except for java class and variable names. So SeLeCT is the same as sELEct is the same as SELECT, but com.journaldev.model.Employee is not same as com.journaldev.model.EMPLOYEE.

The HQL queries are cached but we should avoid it as much as possible, otherwise we will have to take care of associations. However it’s a better choice than native sql query because of Object-Oriented approach. Read more at[HQL Example](http://www.journaldev.com/2954/hibernate-query-language-hql-example-tutorial).

1. **What is Query Cache in Hibernate?**

Hibernate implements a cache region for queries resultset that integrates closely with the hibernate second-level cache.

This is an optional feature and requires additional steps in code. This is only useful for queries that are run frequently with the same parameters. First of all we need to configure below property in hibernate configuration file.

<property name="hibernate.cache.use\_query\_cache">true</property>

And in code, we need to use setCacheable(true) method of Query, quick example looks like below.

Query query = session.createQuery("from Employee");

query.setCacheable(true);

query.setCacheRegion("ALL\_EMP");

1. **Can we execute native sql query in hibernate?**

Hibernate provide option to execute native SQL queries through the use of SQLQuery object.

For normal scenarios, it is however not the recommended approach because we loose benefits related to hibernate association and hibernate first level caching. Read more at [Hibernate Native SQL Query Example](http://www.journaldev.com/3422/hibernate-native-sql-example-addscalar-addentity-addjoin-parameter-example).

1. **What is the benefit of native sql query support in hibernate?**

Native SQL Query comes handy when we want to execute database specific queries that are not supported by Hibernate API such as query hints or the CONNECT keyword in Oracle Database.

1. **What is Named SQL Query?**

Hibernate provides Named Query that we can define at a central location and use them anywhere in the code. We can created named queries for both HQL and Native SQL.

Hibernate Named Queries can be defined in Hibernate mapping files or through the use of JPA annotations @NamedQuery and @NamedNativeQuery.

1. **What are the benefits of Named SQL Query?**

Hibernate Named Query helps us in grouping queries at a central location rather than letting them scattered all over the code.  
Hibernate Named Query syntax is checked when the hibernate session factory is created, thus making the application fail fast in case of any error in the named queries.  
Hibernate Named Query is global, means once defined it can be used throughout the application.

However one of the major disadvantage of Named query is that it’s hard to debug, because we need to find out the location where it’s defined.

1. **What is the benefit of Hibernate Criteria API?**

Hibernate provides Criteria API that is more object oriented for querying the database and getting results. We can’t use Criteria to run update or delete queries or any DDL statements. It’s only used to fetch the results from the database using more object oriented approach.

Some of the common usage of Criteria API are:

* 1. Criteria API provides Projection that we can use for aggregate functions such as sum(), min(), max() etc.
  2. Criteria API can be used with ProjectionList to fetch selected columns only.
  3. Criteria API can be used for join queries by joining multiple tables, useful methods are createAlias(), setFetchMode() and setProjection()
  4. Criteria API can be used for fetching results with conditions, useful methods are add() where we can add Restrictions.
  5. Criteria API provides addOrder() method that we can use for ordering the results.

Learn some quick examples at [Hibernate Criteria Example](http://www.journaldev.com/2963/hibernate-criteria-example-tutorial).

## Chapter 15. Criteria Queries

[15.1. Creating a Criteria instance](https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/querycriteria.html#querycriteria-creating)

[15.2. Narrowing the result set](https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/querycriteria.html#querycriteria-narrowing)

[15.3. Ordering the results](https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/querycriteria.html#querycriteria-ordering)

[15.4. Associations](https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/querycriteria.html#querycriteria-associations)

[15.5. Dynamic association fetching](https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/querycriteria.html#querycriteria-dynamicfetching)

[15.6. Example queries](https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/querycriteria.html#querycriteria-examples)

[15.7. Projections, aggregation and grouping](https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/querycriteria.html#querycriteria-projection)

[15.8. Detached queries and subqueries](https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/querycriteria.html#querycriteria-detachedqueries)

[15.9. Queries by natural identifier](https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/querycriteria.html#query-criteria-naturalid)

Hibernate features an intuitive, extensible criteria query API.

## 15.1. Creating a Criteria instance

The interface org.hibernate.Criteria represents a query against a particular persistent class. The Session is a factory for Criteria instances.

Criteria crit = sess.createCriteria(Cat.class);

crit.setMaxResults(50);

List cats = crit.list();

## 15.2. Narrowing the result set

An individual query criterion is an instance of the interface org.hibernate.criterion.Criterion. The classorg.hibernate.criterion.Restrictions defines factory methods for obtaining certain built-in Criterion types.

List cats = sess.createCriteria(Cat.class)

.add( Restrictions.like("name", "Fritz%") )

.add( Restrictions.between("weight", minWeight, maxWeight) )

.list();

Restrictions can be grouped logically.

List cats = sess.createCriteria(Cat.class)

.add( Restrictions.like("name", "Fritz%") )

.add( Restrictions.or(

Restrictions.eq( "age", new Integer(0) ),

Restrictions.isNull("age")

) )

.list();

List cats = sess.createCriteria(Cat.class)

.add( Restrictions.in( "name", new String[] { "Fritz", "Izi", "Pk" } ) )

.add( Restrictions.disjunction()

.add( Restrictions.isNull("age") )

.add( Restrictions.eq("age", new Integer(0) ) )

.add( Restrictions.eq("age", new Integer(1) ) )

.add( Restrictions.eq("age", new Integer(2) ) )

) )

.list();

There are a range of built-in criterion types (Restrictions subclasses). One of the most useful allows you to specify SQL directly.

List cats = sess.createCriteria(Cat.class)

.add( Restrictions.sqlRestriction("lower({alias}.name) like lower(?)", "Fritz%", Hibernate.STRING) )

.list();

The {alias} placeholder with be replaced by the row alias of the queried entity.

You can also obtain a criterion from a Property instance. You can create a Property by calling Property.forName():

Property age = Property.forName("age");

List cats = sess.createCriteria(Cat.class)

.add( Restrictions.disjunction()

.add( age.isNull() )

.add( age.eq( new Integer(0) ) )

.add( age.eq( new Integer(1) ) )

.add( age.eq( new Integer(2) ) )

) )

.add( Property.forName("name").in( new String[] { "Fritz", "Izi", "Pk" } ) )

.list();

## 15.3. Ordering the results

You can order the results using org.hibernate.criterion.Order.

List cats = sess.createCriteria(Cat.class)

.add( Restrictions.like("name", "F%")

.addOrder( Order.asc("name") )

.addOrder( Order.desc("age") )

.setMaxResults(50)

.list();

List cats = sess.createCriteria(Cat.class)

.add( Property.forName("name").like("F%") )

.addOrder( Property.forName("name").asc() )

.addOrder( Property.forName("age").desc() )

.setMaxResults(50)

.list();

## 15.4. Associations

By navigating associations using createCriteria() you can specify constraints upon related entities:

List cats = sess.createCriteria(Cat.class)

.add( Restrictions.like("name", "F%") )

.createCriteria("kittens")

.add( Restrictions.like("name", "F%") )

.list();

The second createCriteria() returns a new instance of Criteria that refers to the elements of the kittenscollection.

There is also an alternate form that is useful in certain circumstances:

List cats = sess.createCriteria(Cat.class)

.createAlias("kittens", "kt")

.createAlias("mate", "mt")

.add( Restrictions.eqProperty("kt.name", "mt.name") )

.list();

(createAlias() does not create a new instance of Criteria.)

The kittens collections held by the Cat instances returned by the previous two queries are not pre-filtered by the criteria. If you want to retrieve just the kittens that match the criteria, you must use a ResultTransformer.

List cats = sess.createCriteria(Cat.class)

.createCriteria("kittens", "kt")

.add( Restrictions.eq("name", "F%") )

.setResultTransformer(Criteria.ALIAS\_TO\_ENTITY\_MAP)

.list();

Iterator iter = cats.iterator();

while ( iter.hasNext() ) {

Map map = (Map) iter.next();

Cat cat = (Cat) map.get(Criteria.ROOT\_ALIAS);

Cat kitten = (Cat) map.get("kt");

}

## 15.5. Dynamic association fetching

You can specify association fetching semantics at runtime using setFetchMode().

List cats = sess.createCriteria(Cat.class)

.add( Restrictions.like("name", "Fritz%") )

.setFetchMode("mate", FetchMode.EAGER)

.setFetchMode("kittens", FetchMode.EAGER)

.list();

This query will fetch both mate and kittens by outer join. See [Section 19.1, “Fetching strategies”](https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/performance.html#performance-fetching) for more information.

## 15.6. Example queries

The class org.hibernate.criterion.Example allows you to construct a query criterion from a given instance.

Cat cat = new Cat();

cat.setSex('F');

cat.setColor(Color.BLACK);

List results = session.createCriteria(Cat.class)

.add( Example.create(cat) )

.list();

Version properties, identifiers and associations are ignored. By default, null valued properties are excluded.

You can adjust how the Example is applied.

Example example = Example.create(cat)

.excludeZeroes() //exclude zero valued properties

.excludeProperty("color") //exclude the property named "color"

.ignoreCase() //perform case insensitive string comparisons

.enableLike(); //use like for string comparisons

List results = session.createCriteria(Cat.class)

.add(example)

.list();

You can even use examples to place criteria upon associated objects.

List results = session.createCriteria(Cat.class)

.add( Example.create(cat) )

.createCriteria("mate")

.add( Example.create( cat.getMate() ) )

.list();

## 15.7. Projections, aggregation and grouping

The class org.hibernate.criterion.Projections is a factory for Projection instances. You can apply a projection to a query by calling setProjection().

List results = session.createCriteria(Cat.class)

.setProjection( Projections.rowCount() )

.add( Restrictions.eq("color", Color.BLACK) )

.list();

List results = session.createCriteria(Cat.class)

.setProjection( Projections.projectionList()

.add( Projections.rowCount() )

.add( Projections.avg("weight") )

.add( Projections.max("weight") )

.add( Projections.groupProperty("color") )

)

.list();

There is no explicit "group by" necessary in a criteria query. Certain projection types are defined to be grouping projections, which also appear in the SQL group by clause.

An alias can be assigned to a projection so that the projected value can be referred to in restrictions or orderings. Here are two different ways to do this:

List results = session.createCriteria(Cat.class)

.setProjection( Projections.alias( Projections.groupProperty("color"), "colr" ) )

.addOrder( Order.asc("colr") )

.list();

List results = session.createCriteria(Cat.class)

.setProjection( Projections.groupProperty("color").as("colr") )

.addOrder( Order.asc("colr") )

.list();

The alias() and as() methods simply wrap a projection instance in another, aliased, instance of Projection. As a shortcut, you can assign an alias when you add the projection to a projection list:

List results = session.createCriteria(Cat.class)

.setProjection( Projections.projectionList()

.add( Projections.rowCount(), "catCountByColor" )

.add( Projections.avg("weight"), "avgWeight" )

.add( Projections.max("weight"), "maxWeight" )

.add( Projections.groupProperty("color"), "color" )

)

.addOrder( Order.desc("catCountByColor") )

.addOrder( Order.desc("avgWeight") )

.list();

List results = session.createCriteria(Domestic.class, "cat")

.createAlias("kittens", "kit")

.setProjection( Projections.projectionList()

.add( Projections.property("cat.name"), "catName" )

.add( Projections.property("kit.name"), "kitName" )

)

.addOrder( Order.asc("catName") )

.addOrder( Order.asc("kitName") )

.list();

You can also use Property.forName() to express projections:

List results = session.createCriteria(Cat.class)

.setProjection( Property.forName("name") )

.add( Property.forName("color").eq(Color.BLACK) )

.list();

List results = session.createCriteria(Cat.class)

.setProjection( Projections.projectionList()

.add( Projections.rowCount().as("catCountByColor") )

.add( Property.forName("weight").avg().as("avgWeight") )

.add( Property.forName("weight").max().as("maxWeight") )

.add( Property.forName("color").group().as("color" )

)

.addOrder( Order.desc("catCountByColor") )

.addOrder( Order.desc("avgWeight") )

.list();

## 15.8. Detached queries and subqueries

The DetachedCriteria class allows you to create a query outside the scope of a session and then execute it using an arbitrary Session.

DetachedCriteria query = DetachedCriteria.forClass(Cat.class)

.add( Property.forName("sex").eq('F') );

Session session = ....;

Transaction txn = session.beginTransaction();

List results = query.getExecutableCriteria(session).setMaxResults(100).list();

txn.commit();

session.close();

A DetachedCriteria can also be used to express a subquery. Criterion instances involving subqueries can be obtained via Subqueries or Property.

DetachedCriteria avgWeight = DetachedCriteria.forClass(Cat.class)

.setProjection( Property.forName("weight").avg() );

session.createCriteria(Cat.class)

.add( Property.forName("weight").gt(avgWeight) )

.list();

DetachedCriteria weights = DetachedCriteria.forClass(Cat.class)

.setProjection( Property.forName("weight") );

session.createCriteria(Cat.class)

.add( Subqueries.geAll("weight", weights) )

.list();

Correlated subqueries are also possible:

DetachedCriteria avgWeightForSex = DetachedCriteria.forClass(Cat.class, "cat2")

.setProjection( Property.forName("weight").avg() )

.add( Property.forName("cat2.sex").eqProperty("cat.sex") );

session.createCriteria(Cat.class, "cat")

.add( Property.forName("weight").gt(avgWeightForSex) )

.list();

## 15.9. Queries by natural identifier

For most queries, including criteria queries, the query cache is not efficient because query cache invalidation occurs too frequently. However, there is a special kind of query where you can optimize the cache invalidation algorithm: lookups by a constant natural key. In some applications, this kind of query occurs frequently. The criteria API provides special provision for this use case.

First, map the natural key of your entity using <natural-id> and enable use of the second-level cache.

<class name="User">

<cache usage="read-write"/>

<id name="id">

<generator class="increment"/>

</id>

<natural-id>

<property name="name"/>

<property name="org"/>

</natural-id>

<property name="password"/>

</class>

This functionality is not intended for use with entities with mutable natural keys.

Once you have enabled the Hibernate query cache, the Restrictions.naturalId() allows you to make use of the more efficient cache algorithm.

session.createCriteria(User.class)

.add( Restrictions.naturalId()

.set("name", "gavin")

.set("org", "hb")

).setCacheable(true)

.uniqueResult();

1. **How to log hibernate generated sql queries in log files?**

We can set below property for hibernate configuration to log SQL queries.

<property name="hibernate.show\_sql">true</property>

However we should use it only in Development or Testing environment and turn it off in production environment.

1. **What is Hibernate Proxy and how it helps in lazy loading?**

Hibernate uses proxy object to support lazy loading. Basically when you load data from tables, hibernate doesn’t load all the mapped objects. As soon as you reference a child or lookup object via getter methods, if the linked entity is not in the session cache, then the proxy code will go to the database and load the linked object. It uses javassist to effectively and dynamically generate sub-classed implementations of your entity objects.

1. **How to implement relationships in hibernate?**

We can easily implement one-to-one, one-to-many and many-to-many relationships in hibernate. It can be done using JPA annotations as well as XML based configurations. For better understanding, you should go through following tutorials.

* 1. [Hibernate One to One Mapping](http://www.journaldev.com/2916/hibernate-one-to-one-mapping-annotation-and-xml-configuration-example)
  2. [Hibernate One to Many Mapping](http://www.journaldev.com/2924/hibernate-one-to-many-mapping-annotation-and-xml-configuration-example-tutorial)
  3. [Hibernate Many to Many Mapping](http://www.journaldev.com/2934/hibernate-many-to-many-mapping-join-tables-annotation-and-xml-configuration-example)

1. **How transaction management works in Hibernate?**

Transaction management is very easy in hibernate because most of the operations are not permitted outside of a transaction. So after getting the session from SessionFactory, we can call session beginTransaction() to start the transaction. This method returns the Transaction reference that we can use later on to either commit or rollback the transaction.

Overall hibernate transaction management is better than JDBC transaction management because we don’t need to rely on exceptions for rollback. Any exception thrown by session methods automatically rollback the transaction.

1. **What is cascading and what are different types of cascading?**

When we have relationship between entities, then we need to define how the different operations will affect the other entity. This is done by cascading and there are different types of it.

Here is a simple example of applying cascading between primary and secondary entities.

import org.hibernate.annotations.Cascade;

@Entity

@Table(name = "EMPLOYEE")

public class Employee {

@OneToOne(mappedBy = "employee")

@Cascade(value = org.hibernate.annotations.CascadeType.ALL)

private Address address;

}

Note that Hibernate CascadeType enum constants are little bit different from JPA javax.persistence.CascadeType, so we need to use the Hibernate CascadeType and Cascade annotations for mappings, as shown in above example.  
Commonly used cascading types as defined in CascadeType enum are:

* 1. None: No Cascading, it’s not a type but when we don’t define any cascading then no operations in parent affects the child.
  2. ALL: Cascades save, delete, update, evict, lock, replicate, merge, persist. Basically everything
  3. SAVE\_UPDATE: Cascades save and update, available only in hibernate.
  4. DELETE: Corresponds to the Hibernate native DELETE action, only in hibernate.
  5. DETATCH, MERGE, PERSIST, REFRESH and REMOVE – for similar operations
  6. LOCK: Corresponds to the Hibernate native LOCK action.
  7. REPLICATE: Corresponds to the Hibernate native REPLICATE action.

1. **How to integrate log4j logging in hibernate application?**

Hibernate 4 uses JBoss logging rather than slf4j used in earlier versions. For log4j configuration, we need to follow below steps.

* 1. Add log4j dependencies for maven project, if not maven then add corresponding jar files.
  2. Create log4j.xml configuration file or log4j.properties file and keep it in the classpath. You can keep file name whatever you want because we will load it in next step.
  3. For standalone projects, use static block to configure log4j using DOMConfigurator or PropertyConfigurator. For web applications, you can use ServletContextListener to configure it.

That’s it, our setup is ready. Create org.apache.log4j.Logger instance in the java classes and start logging. For complete example code, you should go through [Hibernate log4j example](http://www.journaldev.com/2984/hibernate-4-log4j-configuration-example) and [Servlet log4j example](http://www.journaldev.com/1997/servlet-example-in-java-with-database-connection-and-log4j-integration).

1. **How to use application server JNDI DataSource with Hibernate framework?**

For web applications, it’s always best to allow servlet container to manage the connection pool. That’s why we define JNDI resource for DataSource and we can use it in the web application. It’s very easy to use in Hibernate, all we need is to remove all the database specific properties and use below property to provide the JNDI DataSource name.

<property name="hibernate.connection.datasource">java:comp/env/jdbc/MyLocalDB</property>

For a complete example, go through [Hibernate JNDI DataSource Example](http://www.journaldev.com/2905/hibernate-tomcat-jndi-datasource-example-tutorial).

1. **How to integrate Hibernate and Spring frameworks?**

Spring is one of the most used Java EE Framework and Hibernate is the most popular ORM framework. That’s why Spring Hibernate combination is used a lot in enterprise applications. The best part with using Spring is that it provides out-of-box integration support for Hibernate with **Spring ORM** module. Following steps are required to integrate Spring and Hibernate frameworks together.

* 1. Add hibernate-entitymanager, hibernate-core and spring-orm dependencies.
  2. Create Model classes and corresponding DAO implementations for database operations. Note that DAO classes will use SessionFactory that will be injected by Spring Bean configuration.
  3. If you are using Hibernate 3, you need to configure org.springframework.orm.hibernate3.LocalSessionFactoryBean ororg.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean in Spring Bean configuration file. For Hibernate 4, there is single classorg.springframework.orm.hibernate4.LocalSessionFactoryBean that should be configured.
  4. Note that we don’t need to use Hibernate Transaction Management, we can leave it to Spring declarative transaction management using @Transactional annotation.

For complete example go through [Spring Hibernate Integration](http://www.journaldev.com/3524/spring-hibernate-integration-example-tutorial-spring-4-hibernate-3-and-hibernate-4) and [Spring MVC Hibernate Integration](http://www.journaldev.com/3531/spring-mvc-hibernate-mysql-integration-crud-example-tutorial).

1. **What is HibernateTemplate class?**

When Spring and Hibernate integration started, Spring ORM provided two helper classes – HibernateDaoSupport and HibernateTemplate. The reason to use them was to get the Session from Hibernate and get the benefit of Spring transaction management. However from Hibernate 3.0.1, we can use SessionFactory *getCurrentSession()* method to get the current session and use it to get the spring transaction management benefits. If you go through above examples, you will see how easy it is and that’s why we should not use these classes anymore.

One other benefit of HibernateTemplate was exception translation but that can be achieved easily by using @Repository annotation with service classes, shown in above spring mvc example. This is a trick question to judge your knowledge and whether you are aware of recent developments or not.

1. **How to integrate Hibernate with Servlet or Struts2 web applications?**

Hibernate integration with Servlet or Struts2 needs to be done using ServletContextListener, a complete example can be found at [Hibernate Struts2 Integration Example](http://www.journaldev.com/3557/struts2-hibernate-integration-example-tutorial).

1. **Which design patterns are used in Hibernate framework?**

Some of the design patterns used in Hibernate Framework are:

* 1. Domain Model Pattern – An object model of the domain that incorporates both behavior and data.
  2. Data Mapper – A layer of Mappers that moves data between objects and a database while keeping them independent of each other and the mapper itself.
  3. [Proxy Pattern](http://www.journaldev.com/1572/proxy-design-pattern-in-java-example-tutorial) for lazy loading
  4. [Factory pattern](http://www.journaldev.com/1392/factory-design-pattern-in-java) in SessionFactory

1. **What are best practices to follow with Hibernate framework?**

Some of the best practices to follow in Hibernate are:

* 1. Always check the primary key field access, if it’s generated at the database layer then you should not have a setter for this.
  2. By default hibernate set the field values directly, without using setters. So if you want hibernate to use setters, then make sure proper access is defined as @Access(value=AccessType.PROPERTY).
  3. If access type is property, make sure annotations are used with getter methods and not setter methods. Avoid mixing of using annotations on both filed and getter methods.
  4. Use native sql query only when it can’t be done using HQL, such as using database specific feature.
  5. If you have to sort the collection, use ordered list rather than sorting it using Collection API.
  6. Use named queries wisely, keep it at a single place for easy debugging. Use them for commonly used queries only. For entity specific query, you can keep them in the entity bean itself.
  7. For web applications, always try to use JNDI DataSource rather than configuring to create connection in hibernate.
  8. Avoid Many-to-Many relationships, it can be easily implemented using bidirectional One-to-Many and Many-to-One relationships.
  9. For collections, try to use Lists, maps and sets. Avoid array because you don’t get benefit of lazy loading.
  10. Do not treat exceptions as recoverable, roll back the Transaction and close the Session. If you do not do this, Hibernate cannot guarantee that in-memory state accurately represents the persistent state.
  11. Prefer DAO pattern for exposing the different methods that can be used with entity bean
  12. Prefer lazy fetching for associations

1. **What is Hibernate Validator Framework?**

Data validation is integral part of any application. You will find data validation at presentation layer with the use of Javascript, then at the server side code before processing it. Also data validation occurs before persisting it, to make sure it follows the correct format.

Validation is a cross cutting task, so we should try to keep it apart from our business logic. That’s why JSR303 and JSR349 provides specification for validating a bean by using annotations. Hibernate Validator provides the reference implementation of both these bean validation specs. Read more at [Hibernate Validation Example](http://www.journaldev.com/3626/hibernate-validator-jsr303-example-tutorial).

1. **What is the benefit of Hibernate Tools Eclipse plugin?**

Hibernate Tools plugin helps us in writing hibernate configuration and mapping files easily. The major benefit is the content assist to help us with properties or xml tags to use. It also validates them against the Hibernate DTD files, so we know any mistakes before hand. Learn how to install and use at [Hibernate Tools Eclipse Plugin](http://www.journaldev.com/2940/hibernate-tools-eclipse-plugin-for-generating-hibernate-mapping-and-configuration-files).

That’s all for **Hibernate Interview Questions and Answers**, I hope it will help you for interview as a fresher or experienced person. Please let me know if I have missed any important question here, I will add that to the list.

**1.What is ORM ?**

ORM stands for object/relational mapping. ORM is the automated persistence of objects in a Java application to the tables in a relational database.

|  |
| --- |
|  |

**2.What does ORM consists of ?**

An ORM solution consists of the followig four pieces:

* API for performing basic CRUD operations
* API to express queries refering to classes
* Facilities to specify metadata
* Optimization facilities : dirty checking,lazy associations fetching

**3. What are the ORM levels?**

The ORM levels are:

* Pure relational (stored procedure.)
* Light objects mapping (JDBC)
* Medium object mapping
* Full object Mapping (composition,inheritance, polymorphism, persistence by reachability)

**4. What is Hibernate?**

Hibernate is a pure Java object-relational mapping (ORM) and persistence framework that allows you to map plain old Java objects to relational database tables using (XML) configuration files.Its purpose is to relieve the developer from a significant amount of relational data persistence-related programming tasks.

|  |
| --- |
|  |

**5. Why do you need ORM tools like hibernate?**

The main advantage of ORM like hibernate is that it shields developers from messy SQL. Apart from this, ORM provides following benefits:

* **Improved productivity**
  + High-level object-oriented API
  + Less Java code to write
  + No SQL to write
* **Improved performance**
  + Sophisticated caching
  + Lazy loading
  + Eager loading
* **Improved maintainability**
  + A lot less code to write
* **Improved portability**
  + ORM framework generates database-specific SQL for you

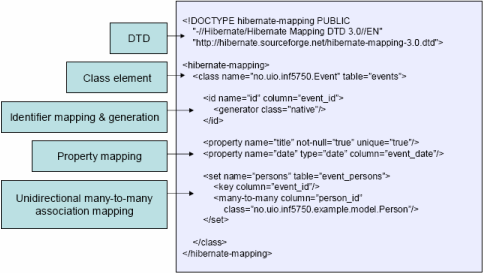
**6. What Does Hibernate Simplify?**

Hibernate simplifies:

* Saving and retrieving your domain objects
* Making database column and table name changes
* Centralizing pre save and post retrieve logic
* Complex joins for retrieving related items
* Schema creation from object model

**7. What is the need for Hibernate xml mapping file?**

Hibernate mapping file tells Hibernate which tables and columns to use to load and store objects. Typical mapping file look as follows:

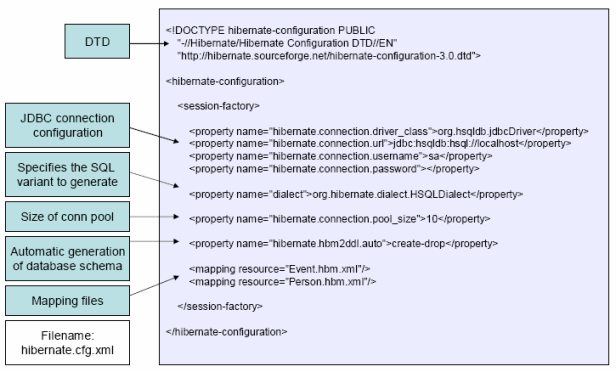
  
  
**8. What are the most common methods of Hibernate configuration?**

The most common methods of Hibernate configuration are:

* Programmatic configuration
* XML configuration (hibernate.cfg.xml)

**9. What are the important tags of hibernate.cfg.xml?**

Following are the important tags of hibernate.cfg.xml:

  
  
**10.What are the Core interfaces are of Hibernate framework?**

|  |
| --- |
|  |

The five core interfaces are used in just about every Hibernate application. Using these interfaces, you can store and retrieve persistent objects and control transactions.

* Session interface
* SessionFactory interface
* Configuration interface
* Transaction interface
* Query and Criteria interfaces

**11.What role does the Session interface play in Hibernate?**

The Session interface is the primary interface used by Hibernate applications. It is a single-threaded, short-lived object representing a conversation between the application and the persistent store. It allows you to create query objects to retrieve persistent objects.  
  
Session session = sessionFactory.openSession();

**Session interface role**:

* Wraps a JDBC connection
* Factory for Transaction
* Holds a mandatory (first-level) cache of persistent objects, used when navigating the object graph or looking up objects by identifier

**12. What role does the SessionFactory interface play in Hibernate?**

The application obtains Session instances from a SessionFactory. There is typically a single SessionFactory for the whole applicationå¹¼reated during application initialization. The SessionFactory caches generate SQL statements and other mapping metadata that Hibernate uses at runtime. It also holds cached data that has been read in one unit of work and may be reused in a future unit of work  
  
SessionFactory sessionFactory = configuration.buildSessionFactory();

**13. What is the general flow of Hibernate communication with RDBMS?**

The general flow of Hibernate communication with RDBMS is:

* Load the Hibernate configuration file and create configuration object. It will automatically load all hbm mapping files
* Create session factory from configuration object
* Get one session from this session factory
* Create HQL Query
* Execute query to get list containing Java objects

**14. What is Hibernate Query Language (HQL)?**

Hibernate offers a query language that embodies a very powerful and flexible mechanism to query, store, update, and retrieve objects from a database. This language, the Hibernate query Language (HQL), is an object-oriented extension to SQL.

**15. How do you map Java Objects with Database tables?**

* First we need to write Java domain objects (beans with setter and getter).
* Write hbm.xml, where we map java class to table and database columns to Java class variables.

**Example** :

<hibernate-mapping>  
  <class name="com.test.User"  table="user">  
   <property  column="USER\_NAME" length="255"   
      name="userName" not-null="true"  type="java.lang.String"/>  
   <property  column="USER\_PASSWORD" length="255"  
 name="userPassword" not-null="true"  type="java.lang.String"/>  
 </class>  
</hibernate-mapping>

**16.Whatï¿½s the difference between load() and get()?**

load() vs. get() :-

|  |  |
| --- | --- |
| **load()** | **get()** |
| Only use the load() method if you are sure that the object exists. | If you are not sure that the object exists, then use one of the get()methods. |
| load() method will throw an exception if the unique id is not found in the database. | get() method will return null if the unique id is not found in the database. |
| load() just returns a proxy by default and database wonï¿½t be hit until the proxy is first invoked. | get() will hit the database immediately. |

**17.What is the difference between and merge and update ?**

Use update() if you are sure that the session does not contain an already persistent instance with the same identifier, and merge() if you want to merge your modifications at any time without consideration of the state of the session.

**18.How do you define sequence generated primary key in hibernate?**

Using <generator> tag.  
**Example**:-

<id column="USER\_ID" name="id" type="java.lang.Long">   
 <generator class="sequence">   
 <param name="table">SEQUENCE\_NAME</param>  
 <generator>  
</id>

**19. Define cascade and inverse option in one-many mapping?**

cascade - enable operations to cascade to child entities.  
cascade="all|none|save-update|delete|all-delete-orphan"  
  
inverse - mark this collection as the "inverse" end of a bidirectional association.  
inverse="true|false"   
Essentially "inverse" indicates which end of a relationship should be ignored, so when persisting a parent who has a collection of children, should you ask the parent for its list of children, or ask the children who the parents are?

**20. What do you mean by Named ï¿½ SQL query?**

Named SQL queries are defined in the mapping xml document and called wherever required.  
**Example:**

<sql-query name = "empdetails">  
   <return alias="emp" class="com.test.Employee"/>  
      SELECT emp.EMP\_ID AS {emp.empid},  
                 emp.EMP\_ADDRESS AS {emp.address},  
                 emp.EMP\_NAME AS {emp.name}   
 FROM Employee EMP WHERE emp.NAME LIKE :name  
</sql-query>

Invoke Named Query :

List people = session.getNamedQuery("empdetails")  
 .setString("TomBrady", name)  
 .setMaxResults(50)  
 .list();

**21. How do you invoke Stored Procedures?**

<sql-query name="selectAllEmployees\_SP" callable="true">  
 <return alias="emp" class="employee">  
  <return-property name="empid" column="EMP\_ID"/>

<return-property name="name" column="EMP\_NAME"/>         
 <return-property name="address" column="EMP\_ADDRESS"/>  
    { ? = call selectAllEmployees() }  
 </return>  
</sql-query>

**22. Explain Criteria API**

Criteria is a simplified API for retrieving entities by composing Criterion objects. This is a very convenient approach for functionality like "search" screens where there is a variable number of conditions to be placed upon the result set.  
**Example**:

List employees = session.createCriteria(Employee.class)  
        .add(Restrictions.like("name", "a%") )  
          .add(Restrictions.like("address", "Boston"))  
 .addOrder(Order.asc("name") )  
 .list();

**23.Define HibernateTemplate?**

org.springframework.orm.hibernate.HibernateTemplate is a helper class which provides different methods for querying/retrieving data from the database. It also converts checked HibernateExceptions into unchecked DataAccessExceptions.

**24.What are the benefits does HibernateTemplate provide?**

The benefits of HibernateTemplate are :

* HibernateTemplate, a Spring Template class simplifies interactions with Hibernate Session.
* Common functions are simplified to single method calls.
* Sessions are automatically closed.
* Exceptions are automatically caught and converted to runtime exceptions.

**25.How do you switch between relational databases without code changes?**

Using Hibernate SQL Dialects , we can switch databases. Hibernate will generate appropriate hql queries based on the dialect defined.

**26.If you want to see the Hibernate generated SQL statements on console, what should we do?**

In Hibernate configuration file set as follows:   
<property name="show\_sql">true</property>

**27. What are derived properties?**

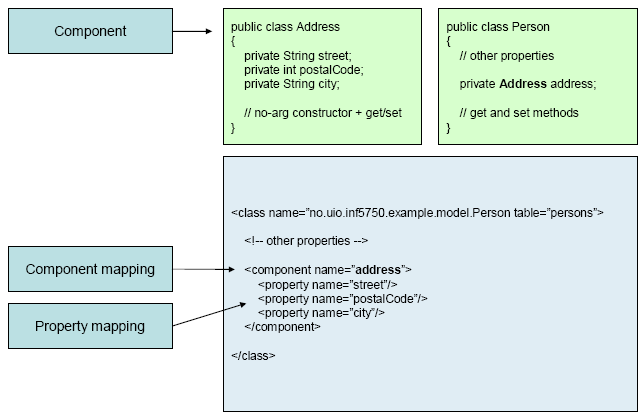
The properties that are not mapped to a column, but calculated at runtime by evaluation of an expression are called derived properties. The expression can be defined using the formula attribute of the element.

|  |
| --- |
|  |

**28. What is component mapping in Hibernate?**

* A component is an object saved as a value, not as a reference
* A component can be saved directly without needing to declare interfaces or identifier properties
* Required to define an empty constructor
* Shared references not supported

**Example**:

  
  
**29. What is the difference between sorted and ordered collection in hibernate?**

**Sorted collection vs. order collection**:-

|  |  |
| --- | --- |
| **sorted collection** | **order collection** |
| A sorted collection is sorting a collection by utilizing the sorting features provided by the Java collections framework. The sorting occurs in the memory of JVM which running Hibernate, after the data being read from database using java comparator. | Order collection is sorting a collection by specifying the order-by clause for sorting this collection when retrieval. |
| If your collection is not large, it will be more efficient way to sort it. | If your collection is very large, it will be more efficient way to sort it . |

**31. What is the advantage of Hibernate over jdbc?**

Hibernate Vs. JDBC:-

|  |  |
| --- | --- |
| **JDBC** | **Hibernate** |
| With JDBC, developer has to write code to map an object model's data representation to a relational data model and its corresponding database schema. | Hibernate is flexible and powerful ORM solution to map Java classes to database tables. Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this. |
| With JDBC, the automatic mapping of Java objects with database tables and vice versa conversion is to be taken care of by the developer manually with lines of code. | Hibernate provides transparent persistence and developer does not need to write code explicitly to map database tables tuples to application objects during interaction with RDBMS. |
| JDBC supports only native Structured Query Language (SQL). Developer has to find out the efficient way to access database, i.e. to select effective query from a number of queries to perform same task. | Hibernate provides a powerful query language Hibernate Query Language (independent from type of database) that is expressed in a familiar SQL like syntax and includes full support for polymorphic queries. Hibernate also supports native SQL statements. It also selects an effective way to perform a database manipulation task for an application. |
| Application using JDBC to handle persistent data (database tables) having database specific code in large amount. The code written to map table data to application objects and vice versa is actually to map table fields to object properties. As table changed or database changed then it’s essential to change object structure as well as to change code written to map table-to-object/object-to-table. | Hibernate provides this mapping itself. The actual mapping between tables and application objects is done in XML files. If there is change in Database or in any table then the only need to change XML file properties. |
| With JDBC, it is developer’s responsibility to handle JDBC result set and convert it to Java objects through code to use this persistent data in application. So with JDBC, mapping between Java objects and database tables is done manually. | Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects. It relieves programmer from manual handling of persistent data, hence reducing the development time and maintenance cost. |
| With JDBC, caching is maintained by hand-coding. | Hibernate, with Transparent Persistence, cache is set to application work space. Relational tuples are moved to this cache as a result of query. It improves performance if client application reads same data many times for same write. Automatic Transparent Persistence allows the developer to concentrate more on business logic rather than this application code. |
| In JDBC there is no check that always every user has updated data. This check has to be added by the developer. | Hibernate enables developer to define version type field to application, due to this defined field Hibernate updates version field of database table every time relational tuple is updated in form of Java class object to that table. So if two users retrieve same tuple and then modify it and one user save this modified tuple to database, version is automatically updated for this tuple by Hibernate. When other user tries to save updated tuple to database then it does not allow saving it because this user does not have updated data. |

|  |
| --- |
|  |

**32. What are the Collection types in Hibernate?**

* Bag
* Set
* List
* Array
* Map

**33. What are the ways to express joins in HQL?**

HQL provides four ways of expressing (inner and outer) joins:-

* An *implicit* association join
* An ordinary join in the FROM clause
* A fetch join in the FROM clause.
* A *theta-style* join in the WHERE clause.

**34. Define cascade and inverse option in one-many mapping?**

cascade - enable operations to cascade to child entities.  
cascade="all|none|save-update|delete|all-delete-orphan"  
  
inverse - mark this collection as the "inverse" end of a bidirectional association.  
inverse="true|false"   
Essentially "inverse" indicates which end of a relationship should be ignored, so when persisting a parent who has a collection of children, should you ask the parent for its list of children, or ask the children who the parents are?

**35. What is Hibernate proxy?**

The proxy attribute enables lazy initialization of persistent instances of the class. Hibernate will initially return CGLIB proxies which implement the named interface. The actual persistent object will be loaded when a method of the proxy is invoked.

|  |
| --- |
|  |

**36. How can Hibernate be configured to access an instance variable directly and not through a setter method?**

By mapping the property with access="field" in Hibernate metadata. This forces hibernate to bypass the setter method and access the instance variable directly while initializing a newly loaded object.

**37. How can a whole class be mapped as immutable?**

Mark the class as mutable="false" (Default is true),. This specifies that instances of the class are (not) mutable. Immutable classes, may not be updated or deleted by the application.

**38. What is the use of dynamic-insert and dynamic-update attributes in a class mapping?**

Criteria is a simplified API for retrieving entities by composing Criterion objects. This is a very convenient approach for functionality like "search" screens where there is a variable number of conditions to be placed upon the result set.

* dynamic-update (defaults to false): Specifies that UPDATE SQL should be generated at runtime and contain only those columns whose values have changed
* dynamic-insert (defaults to false): Specifies that INSERT SQL should be generated at runtime and contain only the columns whose values are not null.

**39. What do you mean by fetching strategy ?**

A *fetching strategy* is the strategy Hibernate will use for retrieving associated objects if the application needs to navigate the association. Fetch strategies may be declared in the O/R mapping metadata, or over-ridden by a particular HQL or Criteria query.

**40. What is automatic dirty checking?**

Automatic dirty checking is a feature that saves us the effort of explicitly asking Hibernate to update the database when we modify the state of an object inside a transaction.

**41. What is transactional write-behind?**

Hibernate uses a sophisticated algorithm to determine an efficient ordering that avoids database foreign key constraint violations but is still sufficiently predictable to the user. This feature is called transactional write-behind.

**42.What are Callback interfaces?**

Callback interfaces allow the application to receive a notification when something interesting happens to an object—for example, when an object is loaded, saved, or deleted. Hibernate applications don't need to implement these callbacks, but they're useful for implementing certain kinds of generic functionality.

**43.What are the types of Hibernate instance states ?**

Three types of instance states:

* Transient -The instance is not associated with any persistence context
* Persistent -The instance is associated with a persistence context
* Detached -The instance was associated with a persistence context which has been closed – currently not associated

**44.What are the differences between EJB 3.0 & Hibernate**

Hibernate Vs EJB 3.0 :-

|  |  |
| --- | --- |
| **Hibernate** | **EJB 3.0** |
| **Session**–Cache or collection of loaded objects relating to a single unit of work | **Persistence Context**-Set of entities that can be managed by a given EntityManager is defined by a persistence unit |
| **XDoclet Annotations** used to support Attribute Oriented Programming | **Java 5.0 Annotations** used to support Attribute Oriented Programming |
| **Defines HQL** for expressing queries to the database | **Defines EJB QL** for expressing queries |
| **Supports Entity Relationships** through mapping files and annotations in JavaDoc | **Support Entity Relationships** through Java 5.0 annotations |
| **Provides a Persistence Manager API** exposed via the Session, Query, Criteria, and Transaction API | **Provides and Entity Manager Interface** for managing CRUD operations for an Entity |
| **Provides callback support** through lifecycle, interceptor, and validatable interfaces | **Provides callback support** through Entity Listener and Callback methods |
| **Entity Relationships are unidirectional**. Bidirectional relationships are implemented by two unidirectional relationships | **Entity Relationships are bidirectional or unidirectional** |

**45. What are the types of inheritance models in Hibernate?**

There are three types of inheritance models in Hibernate:

* Table per class hierarchy
* Table per subclass
* Table per concrete class
* **1. What’s Hibernate?**
* Hibernate is a popular framework of Java which allows an efficient Object Relational mapping using configuration files in XML format. After java objects mapping to [database](http://career.guru99.com/category/database/) tables, database is used and handled using Java objects without writing complex database queries.
* **2. What is ORM?**  
  ORM (Object Relational Mapping) is the fundamental concept of Hibernate framework which maps database tables with Java Objects and then provides various API’s to perform different types of operations on the data tables.
* **3. How properties of a class are mapped to the columns of a database table in Hibernate?**
* Mappings between class properties and table columns are specified in XML file as in the below example:
* **4. What’s the usage of Configuration Interface in hibernate?**
* Configuration interface of hibernate framework is used to configure hibernate. It’s also used to bootstrap hibernate. Mapping documents of hibernate are located using this interface.
* **5. How can we use new custom interfaces to enhance functionality of built-in interfaces of hibernate?**
* We can use extension interfaces in order to add any required functionality which isn’t supported by built-in interfaces.
* **6. Should all the mapping files of hibernate have .hbm.xml extension to work properly?**
* No, having .hbm.xml extension is a convention and not a requirement for hibernate mapping file names. We can have any extension for these mapping files.
* **7. How do we create session factory in hibernate?**
* [](http://career.guru99.com/wp-content/uploads/2012/03/hibernate-interview-questions.png)  
  To create a session factory in hibernate, an object of configuration is created first which refers to the path of configuration file and then for that configuration, session factory is created as given in the example below:
* 

|  |  |
| --- | --- |
| 1  2  3  4 | Configuration config = new Configuration();  config.addResource(&amp;amp;quot;myinstance/configuration.hbm.xml&amp;amp;quot;);  config.setProperties( System.getProperties() );  SessionFactory sessions = config.buildSessionFactory(); |

* **8. What are POJOs and what’s their significance?**
* POJOs( Plain Old Java Objects) are java beans with proper getter and setter methods for each and every properties.  
  Use of POJOs instead of simple java classes results in an efficient and well constructed code.
* **9. What’s HQL?**  
  HQL is the query language used in Hibernate which is an extension of SQL. HQL is very efficient, simple and flexible query language to do various type of operations on relational database without writing complex database queries.
* **10. How can we invoke stored procedures in hibernate?**  
  In hibernate we can execute stored procedures using code as below:
* [xml]
* <sql-query name=”getStudents” callable=”true”>  
  <return alias=”st” class=”Student”>  
  <return-property name=”std\_id” column=”STD\_ID”/>  
  <return-property name=”s\_name” column=”STD\_NAME”/>  
  <return-property name=”s\_dept” column=”STD\_DEPARTMENT”/>  
  { ? = call selectStudents() }  
  </return>  
  </sql-query>
* [/xml]
* **11. What is criteria API?**
* Criteria is a simple yet powerful API of hibernate which is used to retrieve entities through criteria object composition.
* **12. What are the benefits of using Hibernate template?**  
  Following are some key benefits of using Hibernate template:  
  a. Session closing is automated.  
  b. Interaction with hibernate session is simplified.  
  c. Exception handling is automated.
* **13. How can we see hibernate generated SQL on console?**  
  We need to add following in hibernate configuration file to enable viewing SQL on the console for debugging purposes:
* [xml]
* <property name=”show\_sql”>true</property>
* [/xml]
* **14. What are the two types of collections in hibernate?**  
  Following are the two types of collections in hibernate:  
  a. Sorted Collection  
  b. Order Collection
* **15. What’s the difference between session.save() and session.saveOrUpdate() methods in hibernate?**  
  Sessionsave() method saves a record only if it’s unique with respect to its primary key and will fail to insert if primary key already exists in the table.  
  saveOrUpdate() method inserts a new record if primary key is unique and will update an existing record if primary key exists in the table already.
* **16. What the benefits are of hibernate over JDBC?**  
  a. Hibernate can be used seamlessly with any type of database as its database independent while in case of JDBC, developer has to write database specific queries.  
  b. Using hibernate, developer doesn’t need to be an expert of writing complex queries as HQL simplifies query writing process while in case of JDBC, its job of developer to write and tune queries.  
  c. In case of hibernate, there is no need to create connection pools as hibernate does all connection handling automatically while in case of JDBC, connection pools need to be created.
* **17. How can we get hibernate statistics?**  
  We can get hibernate statistics using getStatistics() method of SessionFactory class as shown below:  
  SessionFactory.getStatistics()
* **18. What is transient instance state in Hibernate?**  
  If an instance is not associated with any persistent context and also, it has never been associated with any persistent context, then it’s said to be in transient state.
* **19. How can we reduce database write action times in Hibernate?**  
  Hibernate provides dirty checking feature which can be used to reduce database write times. Dirty checking feature of hibernate updates only those fields which require a change while keeps others unchanged.
* **20. What’s the usage of callback interfaces in hibernate?**  
  Callback interfaces of hibernate are useful in receiving event notifications from objects. For example, when an object is loaded or deleted, an event is generated and notification is sent using callback interfaces.
* **21. When an instance goes in detached state in hibernate?**  
  When an instance was earlier associated with some persistent context (e.g. a table) and is no longer associated, it’s called to be in detached state.
* **22. What the four ORM levels are in hibernate?**  
  Following are the four ORM levels in hibernate:  
  a. Pure Relational  
  b. Light Object Mapping  
  c. Medium Object Mapping  
  d. Full Object Mapping
* **23. What’s transaction management in hibernate? How it works?**  
  Transaction management is the process of managing a set of statements or commands. In hibernate; transaction management is done by transaction interface as shown in below code:
* [java]  
  Session s = null;  
  Transaction tr = null;  
  try {  
  s = sessionFactory.openSession();  
  tr = s.beginTransaction();  
  doTheAction(s);  
  tr.commit();  
  } catch (RuntimeException exc) {  
  tr.rollback();  
  } finally {  
  s.close();  
  }
* [/java]
* **24. What the two methods are of hibernate configuration?**  
  We can use any of the following two methods of hibernate configuration:  
  a. XML based configuration ( using hibernate.cfg.xml file)  
  b. Programmatic configuration ( Using code logic)
* **25. What is the default cache service of hibernate?**  
  Hibernate supports multiple cache services like EHCache, OSCache, SWARMCache and TreeCache and default cache service of hibernate is EHCache.
* **26. What are the two mapping associations used in hibernate?**  
  In hibernate; we have following two types of mapping associations between entities:  
  a. One-to-One Association  
  b. Many-to-Many Association
* **27. What’s the usage of Hibernate QBC API?**  
  Hibernate Query By Criteria (QBC) API is used to create queries by manipulation of criteria objects at runtime.  
  **28. In how many ways, objects can be fetched from database in hibernate?**  
  Hibernate provides following four ways to fetch objects from database:  
  a. Using HQL  
  b. Using identifier  
  c. Using Criteria API  
  d. Using Standard SQL
* **29. How primary key is created by using hibernate?**  
  Database primary key is specified in the configuration file hbm.xml. Generator can also be used to specify how primary key is being created in the database.  
  In the below example, deptId acts as primary key:
* [xml]  
  <id name=”deptId” type=”string” >  
  <column name=”columnId” length=”30″/>  
  <generator/>  
  </id>  
  [/xml]
* **30. How can we reattach any detached objects in Hibernate?**
* Objects which have been detached and are no longer associated with any persistent entities can be reattached by calling session.merge() method of session class.  
  **31. What are different ways to disable hibernate second level cache?**
* Hibernate second level cache can be disabled using any of the following ways:  
  a. By setting use\_second\_level\_cache as false.  
  b. By using CACHEMODE.IGNORE  
  c. Using cache provider as org.hibernate.cache.NoCacheProvider
* **32. What is ORM metadata?**  
  All the mapping between classes and tables, properties and columns, Java types and SQL types etc is defined in ORM metadata.
* **33. Which one is the default transaction factory in hibernate?**  
  With hibernate 3.2, default transaction factory is JDBCTransactionFactory.
* **34. What’s the role of JMX in hibernate?**  
  Java Applications and components are managed in hibernate by a standard API called JMX API. JMX provides tools for development of efficient and robust distributed, web based solutions.  
  **35. How can we bind hibernate session factory to JNDI ?**  
  Hibernate session factory can be bound to JNDI by making configuration changes in hibernate.cfg file.
* **36. In how many ways objects can be identified in Hibernate?**  
  Object identification can be done in hibernate in following three ways:  
  a. Using Object Identity: Using == operator.  
  b. Using Object Equality: Using equals() method.  
  c. Using database identity: Relational database objects can be identified if they represent same row.
* **37. What different fetching strategies are of hibernate?**  
  Following fetching strategies are available in hibernate:  
  1. Join Fetching  
  2. Batch Fetching  
  3. Select Fetching  
  4. Sub-select Fetching  
  **38. How mapping of java objects is done with database tables?**  
  To map java objects with database tables, we need to have Java beans properties names same as column names of a database table. Then mapping is provided in hbm.xml file as given below:
* [xml]  
  <hibernate-mapping>  
  <class name=”Student”  table=”tbl\_student”>  
  <property  column=”studentname” length=”255″  
  name=”studentName” not-null=”true”  type=”java.lang.String”/>  
  <property  column=”studentDisciplne” length=”255″  
  name=”studentDiscipline” not-null=”true”  type=”java.lang.String”/>  
  </class>  
  </hibernate-mapping>  
  [/xml]
* **39. What are derived properties in hibernate?**  
  Derived properties are those properties which are not mapped to any columns of a database table. Such properties are calculated at runtime by evaluation of any expressions.
* **40. What is meant by a Named SQL Query in hibernate and how it’s used?**  
  Named SQL queries are those queries which are defined in mapping file and are called as required anywhere.  
  For example, we can write a SQL query in our XML mapping file as follows:
* [xml]
* <sql-query name = “studentdetails”>  
  <return alias=”std”/>  
  SELECT std.STUDENT\_ID AS {std.STUDENT\_ID},  
  std.STUDENT\_DISCIPLINE AS {std.discipline},
* FROM Student std WHERE std.NAME LIKE :name  
  </sql-query>
* [/xml]
* Then this query can be called as follows:
* [java]
* List students = session.getNamedQuery(&amp;quot;studentdetails&amp;quot;)  
  .setString(&amp;quot;TomBrady&amp;quot;, name)  
  .setMaxResults(50)  
  .list();
* [/java]
* **41. What’s the difference between load() and get() method in hibernate?**  
  Load() methods results in an exception if the required records isn’t found in the database while get() method returns null when records against the id isn’t found in the database.  
  So, ideally we should use Load() method only when we are sure about existence of records against an id.
* **42. What’s the use of version property in hibernate?**  
  Version property is used in hibernate to know whether an object is in transient state or in detached state.
* **43. What is attribute oriented**[**programming**](http://career.guru99.com/category/programming-2/)**?**  
  In Attribute oriented programming, a developer can add Meta data (attributes) in the java source code to add more significance in the code. For Java (hibernate), attribute oriented programming is enabled by an engine called XDoclet.
* **44. What’s the use of session.lock() in hibernate?**  
  session.lock() method of session class is used to reattach an object which has been detached earlier. This method of reattaching doesn’t check for any data synchronization in database while reattaching the object and hence may lead to lack of synchronization in data.
* **45. Does hibernate support polymorphism?**  
  Yes, hibernate fully supports polymorphism. Polymorphism queries and polymorphism associations are supported in all mapping strategies of hibernate.
* **46. What the three inheritance models are of hibernate?**  
  Hibernate has following three inheritance models:  
  a. Tables Per Concrete Class  
  b. Table per class hierarchy  
  c. Table per sub-class
* **47. How can we map the classes as immutable?**  
  If we don’t want an application to update or delete objects of a class in hibernate, we can make the class as immutable by setting mutable=false
* **48. What’s general hibernate flow using RDBMS?**  
  General hibernate flow involving RDBMS is as follows:  
  a. Load configuration file and create object of configuration class.  
  b. Using configuration object, create sessionFactory object.  
  c. From sessionFactory, get one session.  
  d. Create HQL query.  
  e. Execute HQL query and get the results. Results will be in the form of a list.
* **49. What is Light Object Mapping?**  
  Light Object Mapping is one of the levels of ORM quality in which all entities are represented as classes and they are mapped manually.
* **50. What’s difference between managed associations and hibernate associations?**  
  Managed associations relate to container management persistence and are bi-directional while hibernate associations are unidirectional.

## **Interview Question on Hibernate Framework**

Here is my list of Hibernate interview question, which I have collected from friends and colleagues. Hibernate is a popular Object Relational Mapping framework and good knowledge of advantages offered by Hibernate along with Hibernate Session API is key to do well in any Hibernate Interview.

**Difference between get and load in Hibernate?**

get vs load is one of the most frequently asked Hibernate Interview question, since correct understanding of both get() and load() is require to effectively using Hibernate. Main difference between get and load is that, get will hit the database if object is not found in the cache and returned completely initialized object, which may involve several database call while load() method can return proxy, if object is not found in cache and only hit database if any method other than getId() is called. This can save lot of performance in some cases. You can also see [difference between get and load in Hibernate](http://javarevisited.blogspot.com/2012/07/hibernate-get-and-load-difference-interview-question.html) for more differences and detailed discussion on this question.

**Difference between save, persist and saveOrUpdate methods in Hibernate?**

After get vs load, this is another Hibernate Interview question which appears quite often. All three methods i.e. save(), saveOrUpdate() and persist() is used to save objects into database, but has subtle differences e.g. save() can only INSERT records but saveOrUpdate() can either [INSERT or UPDATE records](http://javarevisited.blogspot.com/2013/01/jdbc-batch-insert-and-update-example-java-prepared-statement.html). Also, return type of save() is a Serializable object, while return type of persist() method is void. You can also check [save vs persist vs saveOrUpdate](http://javarevisited.blogspot.com/2012/09/difference-hibernate-save-vs-persist-and-saveOrUpdate.html) for complete differences between them in hibernate.

**What is named SQL query in Hibernate?**

This Hibernate Interview question is related to query functionality provided by Hibernate. Named queries are SQL queries which are defined in mapping document using <sql-query> tag and called using Session.getNamedQuery() method. Named query allows you to refer a particular query by the name you provided, by the way you can define named query in hibernate either by using annotations or xml mapping file, as I said above. @NameQuery is used to define single named query and @NameQueries is used to define multiple named query in hibernate.

**What is SessionFactory in Hibernate? is SessionFactory thread-safe?**

Another common Interview questions related to Hibernate framework. SessionFactory as name suggest is a factory to create hibernate Session objects. SessionFactory is often built during start-up and used by application code to get session object. It acts as single data store and its also [thread-safe](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html) so that multiple thread can use same SessionFactory. Usually a Java JEE application has just one SessionFactory, and individual threads, which are servicing client’s request obtain hibernate Session instances from this factory, that’s why any implementation of SessionFactory interface must be thread-safe. Also internal state of SessionFactory, which contains all meta data about Object/Relational mapping is [Immutable](http://javarevisited.blogspot.com/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html) and can not be changed once created.

**What is Session in Hibernate? Can we share single Session among multiple threads in Hibernate?**

This is usually asked as follow-up question of previous Hibernate Interview question. After SessionFactory its time for Session. Session represent a small unit of work in Hibernate, they maintain connection with database and they are **not thread-safe**, it means you can not share Hibernate Session between multiple threads. Though Session obtains database connection lazily it's good to close session as soon as you are done with it.

**What is difference between sorted and ordered collection in hibernate?**

This is one of the easy Hibernate interview question you ever face. A sorted collection is sorted in memory by using [Java Comparator](http://java67.blogspot.com/2012/10/how-to-sort-object-in-java-comparator-comparable-example.html), while a ordered collection uses database's order by clause for ordering. For large data set it's better to use ordered collection to avoid any [OutOfMemoryError in Java](http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html), by trying to sort them in memory.

**What is difference between transient, persistent and detached object in Hibernate?**

In Hibernate, Object can remain in three state transient, persistent or detached.  An object which is associated with Hibernate session is called persistent object. Any change in this object will reflect in database based upon your flush strategy i.e. automatic flush whenever any property of object change or explicit flushing by calling Session.flush() method. On the other hand if an object which is earlier associated withSession, but currently not associated with it are called detached object. You can reattach detached object to any other session by calling either update() or saveOrUpdate() method on that session. Transient objects are newly created instance of persistence class, which is never associated with any Hibernate Session. Similarly you can call persist() or save() methods to make transient object persistent. Just remember, here transient doesn’t represent [transient keyword in Java](http://javarevisited.blogspot.com/2012/03/difference-between-transient-and.html), which is altogether different thing.

**What does Session lock() method do in Hibernate?**

This one is one of the tricky Hibernate Interview question, because Session's lock() method reattach object without synchronizing or updating with database. So you need to be very careful while using lock() method. By the way you can always use Session's update() method to sync with database during reattachment. Some time this Hibernate question is also asked as *what is difference between Session's lock() and update() method*. You can use this key point to answer that question as well.

**What is Second level Cache in Hibernate?**

This is one of the first interview question related to caching in Hibernate, you can expect few more. Second level Cache is maintained at SessionFactory level and can improve performance by saving few [database round trip](http://javarevisited.blogspot.com/2012/01/improve-performance-java-database.html). Another worth noting point is that second level cache is available to whole application rather than any particular session.

**What is query cache in Hibernate ?**

This question, Some times asked as a follow-up of last Hibernate Interview question, QueryCache actually stores result of sql query for future calls. Query cache can be used along with second level cache for improved performance. Hibernate support various open source caching solution to implement Query cache e.g. EhCache.

**Why it's important to provide no argument constructor in Hibernate Entities?**  
Every Hibernate Entity class must contain a [no argument constructor](http://javarevisited.blogspot.com/2012/12/what-is-constructor-in-java-example-chainning-overloading.html), because Hibernate framework creates instance of them using Reflection API, by calling Class.newInstance() method. This method will throw InstantiationException if it doesn't found no argument constructor inside Entity class.  
  
**Can we make an Hibernate Entity Class final?**  
Yes, you can make an Hibernate Entity class final, but that's not a good practice. Since Hibernate uses proxy pattern for performance improvement in case of lazy association, by making an entity final, Hibernate will no longer be able to use proxy, because [Java doesn't allow extension of final class](http://javarevisited.blogspot.com/2011/12/final-variable-method-class-java.html), thus limiting your performance improvement options. Though, you can avoid this penalty, if your persistent class is an implementation of interface, which declares all public methods defined in Entity class.

**1. What is benefit of using ORM tools?**

The main advantage of ORM like hibernate is that it shields developers from messy SQL. Apart from this, ORM provides following benefits:

* Improved productivity
* High-level object-oriented API
* Less Java code to write
* No SQL to write
* Improved performance
* Sophisticated caching
* Lazy loading  / Eager loading
* Improved maintainability
* A lot less code to write
* Improved portability

ORM framework generates database-specific SQL for you.

**2. What are the most common methods of Hibernate configuration?**

The most common methods of Hibernate configuration are:

* Programmatic configuration
* XML configuration (hibernate.cfg.xml)

**3. What are the Core interfaces are of Hibernate framework?**

The five core interfaces are used in just about every Hibernate application. Using these interfaces, you can store and retrieve persistent objects and control transactions.

* Session interface
* SessionFactory interface
* Configuration interface
* Transaction interface
* Query and Criteria interface

**4. What role does the Session interface play in Hibernate?**

The Session interface is the primary interface used by Hibernate applications. It is a single-threaded, short-lived object representing a conversation between the application and the persistent store. Session **is not thread safe**so we should not share same session between multiple threads. It allows you to create query objects to retrieve persistent objects.

 Session session = sessionFactory.openSession();

 Session interface role:

* Wraps a JDBC connection
* Factory for Transaction
* Holds a mandatory (first-level) cache of persistent objects, used when navigating the object graph or looking up objects by identifier

**5. What role does the SessionFactory interface play in Hibernate?**

Application obtains Session instances from a SessionFactory. There is typically a **single SessionFactory**for the whole application created during application initialization. The SessionFactory caches generate SQL statements and other mapping metadata that Hibernate uses at runtime. It also holds cached data that has been read in one unit of work and may be reused in a future unit of work

   SessionFactory sessionFactory = configuration.buildSessionFactory();

**6. What is the general flow of Hibernate communication with RDBMS?**

The general flow of Hibernate communication with RDBMS is :

* Load the Hibernate configuration file and create configuration object. It will automatically load all hbm mapping files
* Create session factory from configuration object
* Get one session from this session factory
* Create HQL Query

Execute query to get list containing Java objects

**7. What is Hibernate Query Language (HQL)?**

Hibernate offers a query language that embodies a very powerful and flexible mechanism to query, store, update, and retrieve objects from a database. This language, the Hibernate query Language (HQL), is an object-oriented extension to SQL.

**8. What’s the difference between load() and get()?**

load() vs. get() :-

**load()**

* Only use the load() method if you are sure that the object exists.
* load() method will throw an exception if the unique id is not found in the database.
* load() just returns a proxy by default and database won’t be hit until the proxy is first invoked.

**get()**

* If you are not sure that the object exists, then use one of the get() methods.
* get() method will return null if the unique id is not found in the database.
* get() will hit the database immediately.

**9. What is the difference between and merge() and update() ?**

**update():** if you are sure that the session does not contain an already persistent instance with the same identifier.  
**merge()**: if you want to merge your modifications at any time without consideration of the state of the session.

**10. How do you define sequence generated primary key in hibernate?**

Using tag. Example:-

<id name="id" type="java.lang.Integer">

<column name="ID\_PRODUCT" />

<generator class="sequence-identity" >

<param name="sequence">PRODUCT\_ID\_SEQ</param>

</generator>

</id>

**Using Annotations:**

@SequenceGenerator(name="EL\_SEQ", sequenceName="EL\_SEQ",allocationSize=1)

**11. Define cascade and inverse option in one-many mapping?**

**cascade** - enable operations to cascade to child entities.

     cascade="all|none|save-update|delete|all-delete-orphan"

**inverse** - mark this collection as the "inverse" end of a bidirectional association.

     inverse="true|false"

Essentially "inverse" indicates which end of a relationship should be ignored, so when persisting a parent who has a collection of children, should you ask the parent for its list of children, or ask the children who the parents are?

**12. What do you mean by Named – SQL query?**

Named SQL queries are defined in the mapping xml document and called wherever required.

Example:

@NamedQueries({

@NamedQuery(

name = "findEmployeeById",

query = "from Employee s where s.Id= :Id"

)

})

@Entity

@Table(name = "Employee")

**public** **class** Employee **implements** java.io.Serializable {

Query query = session.getNamedQuery("findEmployeeById")

.setString("Id", "7277");

**13. How do you invoke Stored Procedures?**

{ ? = call selectAllEmployees() }

**14. Explain Criteria API ?**

Criteria is a simplified API for retrieving entities by composing Criterion objects. This is a very convenient approach for functionality like "search" screens where there is a variable number of conditions to be placed upon the result set.

Example :

List employees = session.createCriteria(Employee.class)

.add(Restrictions.like("name", "a%") )

.add(Restrictions.like("address", "Boston"))

.addOrder(Order.asc("name") )

.list();

**15. What are the benefits does HibernateTemplate provide?**  
org.springframework.orm.hibernate.HibernateTemplate is a helper class which provides different methods for querying/retrieving data from the database. It also converts checked HibernateExceptions into unchecked DataAccessExceptions.

The benefits of HibernateTemplate are :

 HibernateTemplate, a Spring Template class simplifies interactions with Hibernate Session.

 Common functions are simplified to single method calls.

 Sessions are automatically closed.

 Exceptions are automatically caught and converted to runtime exceptions.

**16. If you want to see the Hibernate generated SQL statements on console, what should we do?**

In Hibernate configuration file set as follows:  
  set**show\_sql** property to **true** in configuration file.

**17. What is the difference between sorted and ordered collection in hibernate?**

sorted collection vs. order collection :-

**Sorted collection**

* A sorted collection is sorting a collection by utilizing the sorting features provided by the Java collections framework. The sorting occurs in   the memory of JVM which running Hibernate, after the data being read from database using java **comparator**.
* If your collection is not large, it will be more efficient way to sort it.
* As it happens in jvm memory, it can throw Out of Memory error.

**Order collection**

* Order collection is sorting a collection by specifying the **order-by clause** in query for sorting this collection when retrieval.
* If your collection is very large, it will be more efficient way to sort it.

**18. What are the Collection types in Hibernate ?**

* Bag
* Set
* List
* Array
* Map

**19. What is Hibernate proxy?**

The proxy attribute enables l**azy initialization** of persistent instances of the class. Hibernate will initially return CGLIB proxies which implement the named interface. The actual persistent object will be loaded when a method of the proxy is invoked.

**20. How can Hibernate be configured to access an instance variable directly and not through a setter method ?**

By mapping the property with **access="field"** in Hibernate metadata. This forces hibernate to bypass the setter method and access the instance variable directly while initializing a newly loaded object.

**21. How can a whole class be mapped as immutable?**

Mark the class as **mutable="false"**(Default is true),. This specifies that instances of the class are (not) mutable. **Immutable classes, may not be updated or deleted by the application.**

**22. What are the types of Hibernate instance states ?**

Three types of instance states:

* **Transient** -The instance is not associated with any persistence context.
* **Persistent** -The instance is associated with a persistence context. You can update object in database by using session.flush() method.
* **Detached** -The instance was associated with a persistence context which has been closed – currently not associated. You can reattach detached object to any other session by calling either update() or saveOrUpdate() method on that session.

**23. What is automatic dirty checking?**

Automatic dirty checking is a feature that saves us the effort of explicitly asking Hibernate to update the database when we modify the state of an object inside a transaction.

**24. Sample of Hibernate transaction management?**

A org.hibernate.Session is designed to represent a single unit of work (a single atmoic piece of work to be performed.

Sample code of handling transaction from hibernate session.

Session session = HibernateUtil.getSessionFactory().getCurrentSession();

**session.beginTransaction();**

        Event theEvent = new Event();

        theEvent.setTitle(title);

        theEvent.setDate(theDate);

        session.save(theEvent);

**session.getTransaction().commit();**

**25. How to implement Optimistic locking in Database?**

You can implement optimistic locks in your DB table in this way (This is how optimistic locking is done in Hibernate):

- Add integer "version" column to your table.

- Increase value of this column with each update of corresponding row.

- To obtain lock, just read "version" value of row.

- Add "version = obtained\_version" condition to where clause of your update statement.

- Verify number of affected rows after update. If no rows were affected - someone has already modified your entry.

Your update should look like

*UPDATE mytable SET name = 'Andy', version = 3 WHERE id = 1 and version = 2;*

**26. What is Second level Cache and QueryCache in Hibernate?**

Second level Cache is maintained at SessionFactory level and It improves performance by saving few database round trip. Another worth noting point is that second level cache is available to whole application rather than any particular session.

     QueryCache actually stores result of sql query for future calls. Query cache can be used along with second level cache for improved performance. Hibernate support various open source caching solution to implement Query cache e.g. EhCache.

**1. What’s Hibernate?**

Hibernate is a popular framework of Java which allows an efficient Object Relational mapping using configuration files in XML format. After java objects mapping to [database](http://career.guru99.com/category/database/) tables, database is used and handled using Java objects without writing complex database queries.

**2. What is ORM?**  
ORM (Object Relational Mapping) is the fundamental concept of Hibernate framework which maps database tables with Java Objects and then provides various API’s to perform different types of operations on the data tables.

**3. How properties of a class are mapped to the columns of a database table in Hibernate?**

Mappings between class properties and table columns are specified in XML file as in the below example:

**4. What’s the usage of Configuration Interface in hibernate?**

Configuration interface of hibernate framework is used to configure hibernate. It’s also used to bootstrap hibernate. Mapping documents of hibernate are located using this interface.

**5. How can we use new custom interfaces to enhance functionality of built-in interfaces of hibernate?**

We can use extension interfaces in order to add any required functionality which isn’t supported by built-in interfaces.

**6. Should all the mapping files of hibernate have .hbm.xml extension to work properly?**

No, having .hbm.xml extension is a convention and not a requirement for hibernate mapping file names. We can have any extension for these mapping files.

**7. How do we create session factory in hibernate?**

[](http://career.guru99.com/wp-content/uploads/2012/03/hibernate-interview-questions.png)  
To create a session factory in hibernate, an object of configuration is created first which refers to the path of configuration file and then for that configuration, session factory is created as given in the example below:



|  |  |
| --- | --- |
| 1  2  3  4 | Configuration config = new Configuration();  config.addResource(&amp;amp;quot;myinstance/configuration.hbm.xml&amp;amp;quot;);  config.setProperties( System.getProperties() );  SessionFactory sessions = config.buildSessionFactory(); |

**8. What are POJOs and what’s their significance?**

POJOs( Plain Old Java Objects) are java beans with proper getter and setter methods for each and every properties.  
Use of POJOs instead of simple java classes results in an efficient and well constructed code.

**9. What’s HQL?**  
HQL is the query language used in Hibernate which is an extension of SQL. HQL is very efficient, simple and flexible query language to do various type of operations on relational database without writing complex database queries.

**10. How can we invoke stored procedures in hibernate?**  
In hibernate we can execute stored procedures using code as below:

[xml]

<sql-query name=”getStudents” callable=”true”>  
<return alias=”st” class=”Student”>  
<return-property name=”std\_id” column=”STD\_ID”/>  
<return-property name=”s\_name” column=”STD\_NAME”/>  
<return-property name=”s\_dept” column=”STD\_DEPARTMENT”/>  
{ ? = call selectStudents() }  
</return>  
</sql-query>

[/xml]

**11. What is criteria API?**

Criteria is a simple yet powerful API of hibernate which is used to retrieve entities through criteria object composition.

**12. What are the benefits of using Hibernate template?**  
Following are some key benefits of using Hibernate template:  
a. Session closing is automated.  
b. Interaction with hibernate session is simplified.  
c. Exception handling is automated.

**13. How can we see hibernate generated SQL on console?**  
We need to add following in hibernate configuration file to enable viewing SQL on the console for debugging purposes:

[xml]

<property name=”show\_sql”>true</property>

[/xml]

**14. What are the two types of collections in hibernate?**  
Following are the two types of collections in hibernate:  
a. Sorted Collection  
b. Order Collection

**15. What’s the difference between session.save() and session.saveOrUpdate() methods in hibernate?**  
Sessionsave() method saves a record only if it’s unique with respect to its primary key and will fail to insert if primary key already exists in the table.  
saveOrUpdate() method inserts a new record if primary key is unique and will update an existing record if primary key exists in the table already.

**16. What the benefits are of hibernate over JDBC?**  
a. Hibernate can be used seamlessly with any type of database as its database independent while in case of JDBC, developer has to write database specific queries.  
b. Using hibernate, developer doesn’t need to be an expert of writing complex queries as HQL simplifies query writing process while in case of JDBC, its job of developer to write and tune queries.  
c. In case of hibernate, there is no need to create connection pools as hibernate does all connection handling automatically while in case of JDBC, connection pools need to be created.

**17. How can we get hibernate statistics?**  
We can get hibernate statistics using getStatistics() method of SessionFactory class as shown below:  
SessionFactory.getStatistics()

**18. What is transient instance state in Hibernate?**  
If an instance is not associated with any persistent context and also, it has never been associated with any persistent context, then it’s said to be in transient state.

**19. How can we reduce database write action times in Hibernate?**  
Hibernate provides dirty checking feature which can be used to reduce database write times. Dirty checking feature of hibernate updates only those fields which require a change while keeps others unchanged.

**20. What’s the usage of callback interfaces in hibernate?**  
Callback interfaces of hibernate are useful in receiving event notifications from objects. For example, when an object is loaded or deleted, an event is generated and notification is sent using callback interfaces.

**21. When an instance goes in detached state in hibernate?**  
When an instance was earlier associated with some persistent context (e.g. a table) and is no longer associated, it’s called to be in detached state.

**22. What the four ORM levels are in hibernate?**  
Following are the four ORM levels in hibernate:  
a. Pure Relational  
b. Light Object Mapping  
c. Medium Object Mapping  
d. Full Object Mapping

**23. What’s transaction management in hibernate? How it works?**  
Transaction management is the process of managing a set of statements or commands. In hibernate; transaction management is done by transaction interface as shown in below code:

[java]  
Session s = null;  
Transaction tr = null;  
try {  
s = sessionFactory.openSession();  
tr = s.beginTransaction();  
doTheAction(s);  
tr.commit();  
} catch (RuntimeException exc) {  
tr.rollback();  
} finally {  
s.close();  
}

[/java]

**24. What the two methods are of hibernate configuration?**  
We can use any of the following two methods of hibernate configuration:  
a. XML based configuration ( using hibernate.cfg.xml file)  
b. Programmatic configuration ( Using code logic)

**25. What is the default cache service of hibernate?**  
Hibernate supports multiple cache services like EHCache, OSCache, SWARMCache and TreeCache and default cache service of hibernate is EHCache.

**26. What are the two mapping associations used in hibernate?**  
In hibernate; we have following two types of mapping associations between entities:  
a. One-to-One Association  
b. Many-to-Many Association

**27. What’s the usage of Hibernate QBC API?**  
Hibernate Query By Criteria (QBC) API is used to create queries by manipulation of criteria objects at runtime.  
**28. In how many ways, objects can be fetched from database in hibernate?**  
Hibernate provides following four ways to fetch objects from database:  
a. Using HQL  
b. Using identifier  
c. Using Criteria API  
d. Using Standard SQL

**29. How primary key is created by using hibernate?**  
Database primary key is specified in the configuration file hbm.xml. Generator can also be used to specify how primary key is being created in the database.  
In the below example, deptId acts as primary key:

[xml]  
<id name=”deptId” type=”string” >  
<column name=”columnId” length=”30″/>  
<generator/>  
</id>  
[/xml]

**30. How can we reattach any detached objects in Hibernate?**

Objects which have been detached and are no longer associated with any persistent entities can be reattached by calling session.merge() method of session class.  
**31. What are different ways to disable hibernate second level cache?**

Hibernate second level cache can be disabled using any of the following ways:  
a. By setting use\_second\_level\_cache as false.  
b. By using CACHEMODE.IGNORE  
c. Using cache provider as org.hibernate.cache.NoCacheProvider

**32. What is ORM metadata?**  
All the mapping between classes and tables, properties and columns, Java types and SQL types etc is defined in ORM metadata.

**33. Which one is the default transaction factory in hibernate?**  
With hibernate 3.2, default transaction factory is JDBCTransactionFactory.

**34. What’s the role of JMX in hibernate?**  
Java Applications and components are managed in hibernate by a standard API called JMX API. JMX provides tools for development of efficient and robust distributed, web based solutions.  
**35. How can we bind hibernate session factory to JNDI ?**  
Hibernate session factory can be bound to JNDI by making configuration changes in hibernate.cfg file.

**36. In how many ways objects can be identified in Hibernate?**  
Object identification can be done in hibernate in following three ways:  
a. Using Object Identity: Using == operator.  
b. Using Object Equality: Using equals() method.  
c. Using database identity: Relational database objects can be identified if they represent same row.

**37. What different fetching strategies are of hibernate?**  
Following fetching strategies are available in hibernate:  
1. Join Fetching  
2. Batch Fetching  
3. Select Fetching  
4. Sub-select Fetching  
**38. How mapping of java objects is done with database tables?**  
To map java objects with database tables, we need to have Java beans properties names same as column names of a database table. Then mapping is provided in hbm.xml file as given below:

[xml]  
<hibernate-mapping>  
<class name=”Student”  table=”tbl\_student”>  
<property  column=”studentname” length=”255″  
name=”studentName” not-null=”true”  type=”java.lang.String”/>  
<property  column=”studentDisciplne” length=”255″  
name=”studentDiscipline” not-null=”true”  type=”java.lang.String”/>  
</class>  
</hibernate-mapping>  
[/xml]

**39. What are derived properties in hibernate?**  
Derived properties are those properties which are not mapped to any columns of a database table. Such properties are calculated at runtime by evaluation of any expressions.

**40. What is meant by a Named SQL Query in hibernate and how it’s used?**  
Named SQL queries are those queries which are defined in mapping file and are called as required anywhere.  
For example, we can write a SQL query in our XML mapping file as follows:

[xml]

<sql-query name = “studentdetails”>  
<return alias=”std”/>  
SELECT std.STUDENT\_ID AS {std.STUDENT\_ID},  
std.STUDENT\_DISCIPLINE AS {std.discipline},

FROM Student std WHERE std.NAME LIKE :name  
</sql-query>

[/xml]

Then this query can be called as follows:

[java]

List students = session.getNamedQuery(&amp;quot;studentdetails&amp;quot;)  
.setString(&amp;quot;TomBrady&amp;quot;, name)  
.setMaxResults(50)  
.list();

[/java]

**41. What’s the difference between load() and get() method in hibernate?**  
Load() methods results in an exception if the required records isn’t found in the database while get() method returns null when records against the id isn’t found in the database.  
So, ideally we should use Load() method only when we are sure about existence of records against an id.

**42. What’s the use of version property in hibernate?**  
Version property is used in hibernate to know whether an object is in transient state or in detached state.

**43. What is attribute oriented**[**programming**](http://career.guru99.com/category/programming-2/)**?**  
In Attribute oriented programming, a developer can add Meta data (attributes) in the java source code to add more significance in the code. For Java (hibernate), attribute oriented programming is enabled by an engine called XDoclet.

**44. What’s the use of session.lock() in hibernate?**  
session.lock() method of session class is used to reattach an object which has been detached earlier. This method of reattaching doesn’t check for any data synchronization in database while reattaching the object and hence may lead to lack of synchronization in data.

**45. Does hibernate support polymorphism?**  
Yes, hibernate fully supports polymorphism. Polymorphism queries and polymorphism associations are supported in all mapping strategies of hibernate.

**46. What the three inheritance models are of hibernate?**  
Hibernate has following three inheritance models:  
a. Tables Per Concrete Class  
b. Table per class hierarchy  
c. Table per sub-class

**47. How can we map the classes as immutable?**  
If we don’t want an application to update or delete objects of a class in hibernate, we can make the class as immutable by setting mutable=false

**48. What’s general hibernate flow using RDBMS?**  
General hibernate flow involving RDBMS is as follows:  
a. Load configuration file and create object of configuration class.  
b. Using configuration object, create sessionFactory object.  
c. From sessionFactory, get one session.  
d. Create HQL query.  
e. Execute HQL query and get the results. Results will be in the form of a list.

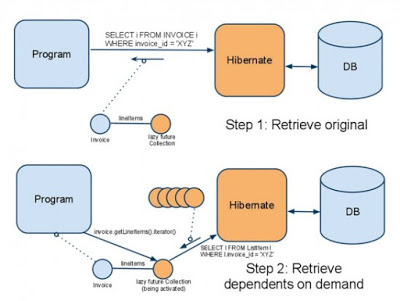
**49. What is Light Object Mapping?**  
Light Object Mapping is one of the levels of ORM quality in which all entities are represented as classes and they are mapped manually.

**50. What’s difference between managed associations and hibernate associations?**  
Managed associations relate to container management persistence and are bi-directional while hibernate associations are unidirectional.

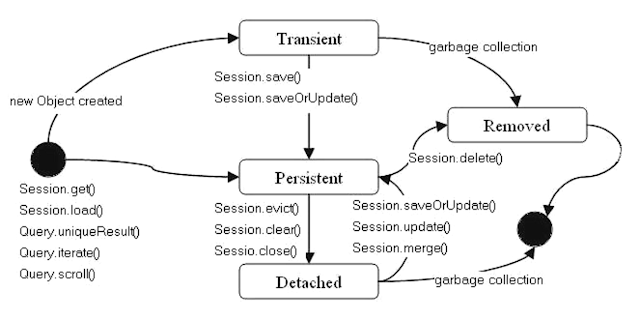
Important Questions and Anseres

## 20 Hibernate Questions from Java JEE Interviews

Here is my selected list of 20 Hibernate based questions for Java developers. It contains questions from Hibernate fundamentals, one-to-one, and one-to-many mappings, caching, Hibernate vs JDBC comparison, pros and cons of Hibernate, known problems with Hibernate, and Performance improvement.  
  
  
**What is Hibernate?**  
Hibernate is an ORM (Object-relational Mapping) framework, which allows the developer to concentrate on business logic by taking care of persistence of data by itself. Java developer can write code using object and Hibernate can take care of creating those object from data loaded from the database and saving update back to the database.  
  
  
**What are the advantages of Hibernate over JDBC?**(detailed answer)  
Apart from Persistence i.e. saving and loading data from Database, Hibernate also provides following benefits  
1) Caching  
2) Lazy Loading  
3) Relationship management and provides code for mapping an object to the data  
4) The developer is free from writing code to load/store data into the database.  
  
  
**Difference between get() vs load() method in Hibernate?** ([detailed answer](http://javarevisited.blogspot.com/2012/07/hibernate-get-and-load-difference-interview-question.html))  
This is one of the most frequently asked Hibernate interview question, I have seen it several times. The key difference between get() and load() method is that load() will throw an exception if an object with id passed to them is not found, but get() will return [null](http://javarevisited.blogspot.com/2014/12/9-things-about-null-in-java.html). Another important difference is that load can return proxy without hitting the database unless required (when you access any attribute other than id) but get() always go to the database, so sometimes using load() can be faster than the get() method. It makes sense to use the load() method if you know the object exists but get() method if you are not sure about object's existence.

[](https://pluralsight.pxf.io/c/1193463/424552/7490?u=https://www.pluralsight.com/courses/spring-jpa-hibernate)

**What is N+1 SELECT problem in Hibernate?**(detailed answer)  
The N+1 SELECT problem is a result of lazy loading and load on demand fetching strategy. In this case, Hibernate ends up executing N+1 SQL queries to populate a collection of N elements. For example, if you have a List of N Items where each Item has a dependency on a collection of Bid object. Now if you want to find the highest bid for each item then Hibernate will fire 1 query to load all items and N subsequent queries to load Bid for each item. So in order to find the highest bid for each item your application end up firing N+1 queries.  It's one of the important Hibernate interview questions and I suggest to read chapter 13 of [Java Persistence with Hibernate](http://www.amazon.com/Java-Persistence-Hibernate-Christian-Bauer/dp/1932394885?tag=javamysqlanta-20) to understand this problem in more details.  
  
  
**What are some strategies to solve the N+1 SELECT problem in Hibernate?**(detailed answer)  
This is the follow-up question of previous Hibernate interview question. If you answer the last query correctly then you would be most likely asked this one. Here are some strategies to solve the N+1 problem:  
1) pre-fetching in batches, this will reduce N+1 problem to N/K + 1 problem where  K is size of batch  
2) subselect fetching strategy  
3) disabling lazy loading  
  
  
**What is the difference between save() and persist() method in Hibernate?**([detailed answer](http://java67.blogspot.com/2016/01/difference-between-save-saveorupdate-and-persist-in-Hibernate.html))  
Main difference between save() and persist() method is that, save returns a Serializable object while return type of persist() method is void, so it doesn't return anything. Here is a nice diagram which explains the state transition in Hibernate:

[](https://click.linksynergy.com/fs-bin/click?id=JVFxdTr9V80&subid=0&offerid=323058.1&type=10&tmpid=14538&RD_PARM1=https://www.udemy.com/hibernate-and-java-persistence-api-jpa-fundamentals/)

**What is the requirement for a Java object to become Hibernate entity object?**([detailed answer](http://javarevisited.blogspot.com/2016/01/why-jpa-entity-or-hibernate-persistence-should-not-be-final-in-java.html))  
It should not be final and must provide a default, no-argument constructor. See the detailed answer to learn more about the special requirement for a Java object to become Hibernate Entity.  
  
  
**What are different types of caches available in Hibernate?**(detailed answer)  
This is another common Hibernate interview question. Hibernate provides the out-of-box caching solution but there are many caches e.g. first level cache, second level cache and query cache. First level cache is maintained at Session level and cannot be disabled but the second level cache is required to be configured with external cache provider like EhCache.  
  
  
**What is the difference between first and second level cache in Hibernate?**(detailed answer)  
This is again follow-up of previous Hibernate interview question. The first level cache is maintained at Session level while the second level cache is maintained at SessionFactory level and shared by all sessions. You can read [these books](http://javarevisited.blogspot.com/2014/01/top-5-hibernate-books-for-java-programmers-learning.html) to learn more about caching in Hibernate.  
  
  
**Does Hibernate Session interface is thread-safe in Java?**(detailed answer)  
No, Session object is not thread-safe in Hibernate and intended to be used with-in single thread in the application.  
  
  
**Does SessionFactory is thread-safe in Hibernate?**([detailed answer](http://javarevisited.blogspot.com/2013/05/10-hibernate-interview-questions-answers-java-j2ee-senior.html))  
SessionFactory is both Immutable and thread-safe and it has just one single instance in Hibernate application. It is used to create Session object and it also provide caching by storing SQL queries stored by multiple session. The second level cache is maintained at SessionFactory level. This can be a difficult and tricky question for less experienced Java developers who are not familiar with thread-safety and Immutability.  
  
  
**What is different between Session and Sessionfactory in Hibernate?**(detailed answer)  
This is another popular Hibernate interview question, mostly at a telephonic round of interviews. The main difference between Session and SessionFactory is that former is a single-threaded, short-lived object while later is Immutable and shared by all Session. It also lives until the Hibernate is running. Another difference between Session and SessionFactory is that former provides first level cache while SessionFactory provides the Second level cache.  
  
  
**What is criterion query in hibernate?**(detailed answer)  
Criteria is a simplified API for retrieving entities by composing Criterion objects also known as Criterion query. This is a very convenient approach for functionality like "search" screens where you can filter data on multiple conditions as shown in the following example:

List books = session.createCriteria(Book.class)

.add(Restrictions.like("name", "java%") )

.add(Restrictions.like("published\_year", "2015"))

.addOrder(Order.asc("name") )

.list();

This can be a tough question if you are not using Hibernate on a daily basis, I have interviewed several Java developers who have used Hibernate but doesn't know about Criterion query or API.  
  
  
**What are other ORM frameworks? Any alternative of Hibernate?**  
This is a general question, sometimes asked to start the conversation and other times to finish the interview. EJB and TopLink from Oracle are two of the most popular alternative to Hibernate framework.  
  
  
**What is the difference between save() and saveOrUpdate() method of Hibernate?**([detailed answer](http://javarevisited.blogspot.com/2012/09/difference-hibernate-save-vs-persist-and-saveOrUpdate.html))  
Though both save() and saveOrUpdate() method is used to store object into Database, the key difference between them is that save can only INSERT records but saveOrUpdate() can either INSERT or UPDATE records.  
  
  
**What is difference between getCurrentSession() and openSession() in Hibernate?**(detailed answer)  
An interesting Hibernate interview question as you might have used both getCurrentSession() and openSession() to obtain an instance of Session object. I have left this question unanswered for you to answer or find an answer based on your experience.  
  
  
**What is Hibernate Query Language (HQL)?**([detailed answer](http://javarevisited.blogspot.com/2014/03/hibernate-nullpointerexception-due-to-space-in-HQL-Query.html))  
Hibernate query language, HQL is an object-oriented extension to SQL. It allows you to query, store, update, and retrieve objects from a database without using SQL. This question is also similar to the earlier question about Criterion query, Java developers who have not used Hibernate extensively will not know much about features like HQL and Criterion.  
  
  
**When do you use merge() and update() in Hibernate?**(detailed answer)  
This is one of the tricky Hibernate interview questions. You should use update() if you are sure that the Hibernate session does not contain an already persistent instance with the same id and use merge() if you want to merge your modifications at any time without considering the state of the session. See [Java Persistence with Hibernate](http://www.amazon.com/Java-Persistence-Hibernate-Christian-Bauer/dp/1932394885?tag=javamysqlanta-20) for more details.  
  
**The difference between sorted and ordered collection in Hibernate?**(detailed answer)  
The main difference between sorted and ordered collection is that sorted collection sort the data in JVM's heap memory using Java's collection framework sorting methods while ordered collection is sorted using order by clause in the database itself. A sorted collection is more suited for small dataset but for a large dataset, it's better to use ordered collection to avoid [OutOfMemoryError in Java](http://java67.blogspot.com/2013/08/guide-of-javalangoutofmemoryerror-java-heap-space-tomcat-eclipse-minecraft-jboss.html) application.  
  
  
**How do you log SQL queries issued by the Hibernate framework in Java application?**  
You can use the show\_sql property to log SQL queries issued by the Hibernate framework, Just add the following line in your Hibernate configuration file:

<property name=”show\_sql”> true </property>

**What are the three states of a Hibernate Persistence object can be?**(detailed answer)  
The Hibernate persistent or entity object can live in following three states:  
1) transient  
2) persistent  
3) detached  
  
  
**What is the difference between the transient, persistent and detached state in Hibernate?**(detailed answer)  
New objects created in Java program but not associated with any hibernate Session are said to be in the transient state. On the other hand, an object which is associated with a Hibernate session is called Persistent object. While an object which was earlier associated with Hibernate session but currently it's not associate is known as a detached object. You can call save() or persist() method to store those object into the database and bring them into the Persistent state. Similarly, you can re-attach a detached object to hibernate sessions by calling either update() or saveOrUpdate() method.  
  
  
**Which cache is used by Session Object in Hibernate? First level or second level cache?**(detailed answer)  
A Session object uses the first-level cache. As I told before the second level cache is used at SessionFactory level. This is a good question to check if Candidate has been working in hibernate or not. If he has not worked in Hibernate from a long time then he would get confused in this question.  
  
  
That's all in this list of **Hibernate Interview question for Java  and JEE developers**. In this article, we have covered a lot of frequently asked Hibernate question for both beginners and experienced Java developers from all important topics of Hibernate framework e.g. Hibernate fundamentals, caching, collection mapping, performance tuning, common issues and Hibernate vs JDBC.  
  
Read more: <http://www.java67.com/2016/02/top-20-hibernate-interview-questions.html#ixzz5WEsTICTU>