Copied

- **Hibernate Interview Questions For Freshers**
  - 1.What is ORM in Hibernate?
  - 2.What are the advantages of Hibernate over JDBC?
  - 3.What are some of the important interfaces of Hibernate framework?
  - 4.What is a Session in Hibernate?
  - 5.What is a SessionFactory?
  - 6.What do you think about the statement - "session being a thread-safe object"?
  - 7.Can you explain what is lazy loading in hibernate?
  - 8.What is the difference between first level cache and second level cache?
  - 9.What can you tell about Hibernate Configuration File?
  - 10.How do you create an immutable class in hibernate?
  - 11.Can you explain the concept behind Hibernate Inheritance Mapping?
  - 12.Is hibernate prone to SQL injection attack?
- **Intermediate Interview Questions**
  - 13.Explain hibernate mapping file
  - 14.What are the most commonly used annotations available to support hibernate mapping?
  - 15.Explain Hibernate architecture
  - 16.Can you tell the difference between getCurrentSession and openSession methods?
  - 17.Differentiate between save() and saveOrUpdate() methods in hibernate session.
  - 18.Differentiate between get() and load() in Hibernate session
  - 19.What is criteria API in hibernate?
  - 20.What is HQL?
  - 21.Can you tell something about one to many associations and how can we use them in Hibernate?
  - 22.What are Many to Many associations?
  - 23.What does session.lock() method in hibernate do?
  - 24.What is hibernate caching?
  - 25.When is merge() method of the hibernate session useful?
  - 26.Collection mapping can be done using One-to-One and Many-to-One Associations. What do you think?
  - 27.Can you tell the difference between setMaxResults() and setFetchSize() of Query?
  - 28.Does Hibernate support Native SQL Queries?
- **Hibernate Interview Questions For Experienced**
  - 29.What happens when the no-args constructor is absent in the Entity bean?
  - 30.Can we declare the Entity class final?
  - 31.What are the states of a persistent entity?
  - 32.Explain Query Cache
  - 33.Can you tell something about the N+1 SELECT problem in Hibernate?
  - 34.How to solve N+1 SELECT problem in Hibernate?

- 35.What are the concurrency strategies available in hibernate?
- 36.What is Single Table Strategy?
- 37.Can you tell something about Table Per Class Strategy.
- 38.Can you tell something about Named SQL Query
- 39.What are the benefits of NamedQuery?
- **Hibernate MCQs**

Hibernate is a **Java-based persistence framework** and an **object-relational mapping (ORM)** framework that basically allows a developer to map POJO - plain old Java objects - to relational database tables.

The aim of hibernate framework is to free the developer from the common data persistence-related complex configurations and tasks. It does so by mapping the POJO objects with the database tables efficiently and most importantly in an abstract manner.

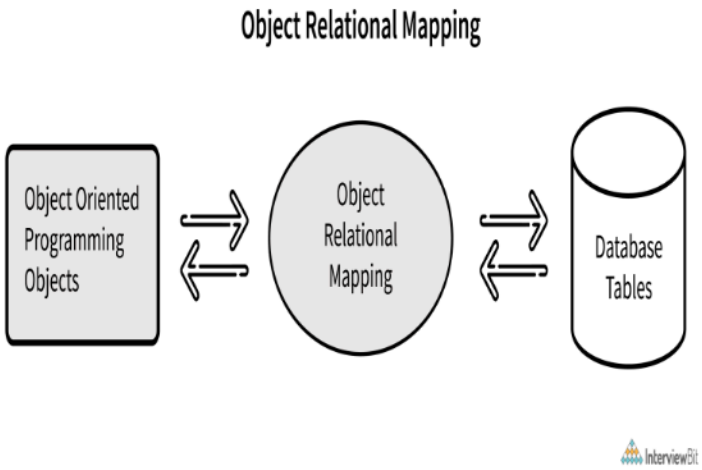
The developer need not know the underlying complications involved. Along with abstraction, the queries can be executed in a very efficient manner. All these helps developers to save a lot of time involved in development.

Top we will walk you through the top questions to get you ready for a Hibernate interview. This article would cover basic, intermediate, and advanced questions.

Hibernate Interview Questions For Freshers

1. What is ORM in Hibernate?

Hibernate ORM stands for **Object Relational Mapping**. This is a mapping tool pattern mainly used for converting data stored in a relational database to an object used in object-oriented programming constructs. This tool also helps greatly in simplifying data retrieval, creation, and manipulation.



Object Relational Mapping

2. What are the advantages of Hibernate over JDBC?

- The advantages of Hibernate over JDBC are listed below:

- **Clean Readable Code:** Using hibernate, helps in eliminating a lot of JDBC API-based boiler-plate codes, thereby making the code look cleaner and readable.
- **HQL (Hibernate Query Language):** Hibernate provides HQL which is closer to Java and is object-oriented in nature. This helps in reducing the burden on developers for writing database independent queries. In JDBC, this is not the case. A developer has to know the database-specific codes.
- **Transaction Management:** JDBC doesn't support implicit transaction management. It is upon the developer to write transaction management code using commit and rollback methods. Whereas, Hibernate implicity provides this feature.
- **Exception Handling:** Hibernate wraps the JDBC exceptions and throws unchecked exceptions like JDBCException or HibernateException. This along with the built-in transaction management system helps developers to avoid writing multiple try-catch blocks to handle exceptions. In the case of JDBC, it throws a checked exception called SQLException thereby mandating the developer to write try-catch blocks to handle this exception at compile time.
- **Special Features:** Hibernate supports OOPs features like inheritance, associations and also supports collections. These are not available in JDBC.

3. What are some of the important interfaces of Hibernate framework?

Hibernate core interfaces are:

- Configuration
- SessionFactory
- Session

- Criteria
- Query
- Transaction

4. What is a Session in Hibernate?

A session is an object that maintains the connection between Java object application and database. Session also has methods for storing, retrieving, modifying or deleting data from database using methods like persist(), load(), get(), update(), delete(), etc. Additionally, It has factory methods to return Query, Criteria, and Transaction objects.

5. What is a SessionFactory?

SessionFactory provides an instance of **Session**. It is a factory class that gives the Session **objects** based on the configuration parameters in order to establish the connection to the database.
As a good practice, the application generally has a single instance of SessionFactory. The internal state of a SessionFactory which includes metadata about ORM is immutable, i.e once the instance is created, it cannot be changed.

This also provides the facility to get information like statistics and metadata related to a class, query executions, etc. It also holds second-level cache data if enabled.

6. What do you think about the statement - "session being a thread-safe object"?

No, Session is not a thread-safe object which means that any number of threads can access data from it simultaneously.

7. Can you explain what is lazy loading in hibernate?

Lazy loading is mainly used for improving the application performance by helping to load the child objects on demand.

It is to be noted that, since Hibernate 3 version, this feature has been enabled by default. This signifies that child objects are not loaded until the parent gets loaded.

8. What is the difference between first level cache and second level cache?

Hibernate has 2 cache types. First level and second level cache for which the difference is given below:

First Level Cache	Second Level Cache
This is local to the Session object and cannot be shared between multiple sessions.	This cache is maintained at the SessionFactory level and shared among all sessions in Hibernate.
This cache is enabled by default and there is no way to disable it.	This is disabled by default, but we can enable it through configuration.
The first level cache is available only until the session is open, once the session is closed, the first level cache is destroyed.	The second-level cache is available through the application's life cycle, it is only destroyed and recreated when an application is restarted.

If an entity or object is loaded by calling the get() method then Hibernate first checked the first level cache, if it doesn't find the object then it goes to the second level cache if configured. If the object is not found then it finally goes to the database and returns the object, if there is no corresponding row in the table then it returns null.

9. What can you tell about Hibernate Configuration File?

Hibernate Configuration File or **hibernate.cfg.xml** is one of the most required configuration files in Hibernate. By default, this file is placed under the src/main/resource folder.
The file contains database related configurations and session-related configurations. Hibernate facilitates providing the configuration either in an XML file (like hibernate.cfg.xml) or a properties file (like hibernate.properties).

This file is used to define the below information:

- Database connection details: Driver class, URL, username, and password.
- There must be one configuration file for each database used in the application, suppose if we want to connect with 2 databases, then we must create 2 configuration files with different names.
- Hibernate properties: Dialect, show_sql, second_level_cache, and mapping file names.

10. How do you create an immutable class in hibernate?

Immutable class in hibernate creation could be in the following way. If we are using the XML form of configuration, then a class can be made immutable by markingmutable=false. The default value is true there which indicating that the class was not created by default.

In the case of using annotations, immutable classes in hibernate can also be created by using @Immutable annotation.

11. Can you explain the concept behind Hibernate Inheritance Mapping?

Java is an Object-Oriented Programming Language and Inheritance is one of the most important pillars of object-oriented principles. To represent any models in Java, inheritance is most commonly used to simplify and simplify the relationship. But, there is a catch. Relational databases do not support inheritance. They have a flat structure.

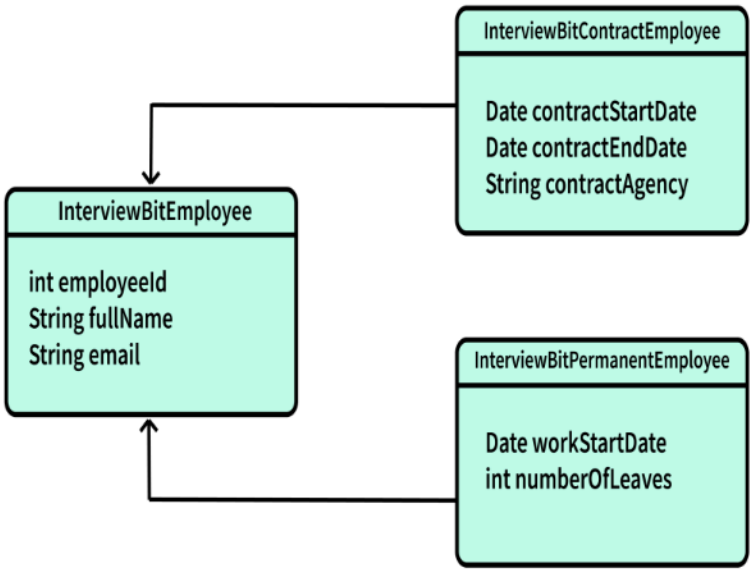
Hibernate's Inheritance Mapping strategies deal with solving how to hibernate being an ORM tries to map this problem between the inheritance of Java and flat structure of Databases.

Consider the example where we have to divide InterviewBitEmployee into Contract and Permanent Employees represented by IBContractEmployee and IBPermanentEmployee classes respectively. Now the task of hibernate is to represent these 2 employee types by considering the below restrictions:

The general employee details are defined in the parent InterviewBitEmployee class.

Contract and Permanent employee-specific details are stored in IBContractEmployee and IBPermanentEmployee classes respectively

The class diagram of this system is as shown below:



InterviewBit

Hibernate's Inheritance Mapping

There are different inheritance mapping strategies available:

- Single Table Strategy
- Table Per Class Strategy
- Mapped Super Class Strategy
- Joined Table Strategy

12. Is hibernate prone to SQL injection attack?

SQL injection attack is a serious vulnerability in terms of web security wherein an attacker can interfere with the queries made by an application/website to its database thereby allowing the attacker to view sensitive data which are generally irretrievable. It can also give the attacker to modify/ remove the data resulting in damages to the application behavior.

Hibernate does not provide immunity to SQL Injection. However, following good practices avoids SQL injection attacks. It is always advisable to follow any of the below options:

- Incorporate Prepared Statements that use Parameterized Queries.
- Use Stored Procedures.
- Ensure data sanity by doing input validation.

Intermediate Interview Questions

13. Explain hibernate mapping file

Hibernate mapping file is an XML file that is used for defining the entity bean fields and corresponding database column mappings.
These files are useful when the project uses third-party classes where JPA annotations provided by hibernating cannot be used.

In the previous example, we have defined the mapping resource as "InterviewBitEmployee.hbm.xml" in the config file. Let us see what that sample hbm.xml file looks like:

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <!-- What class is mapped to what database table-->
  <class name = "InterviewBitEmployee" table = "InterviewBitEmployee">
```

```
    <meta attribute = "class-description">
       This class contains the details of employees of InterviewBit.
    </meta>

    <id name = "id" type = "int" column = "employee_id">
       <generator class="native"/>
    </id>

    <property name = "fullName" column = "full_name" type = "string"/>
    <property name = "email" column = "email" type = "string"/>

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

14. What are the most commonly used annotations available to support hibernate mapping?

Hibernate framework provides support to JPA annotations and other useful annotations in the org.hibernate.annotations package. Some of them are as follows:

- javax.persistence.Entity: This annotation is used on the model classes by using "@Entity" and tells that the classes are entity beans.
- javax.persistence.Table: This annotation is used on the model classes by using "@Table" and tells that the class maps to the table name in the database.
- javax.persistence.Access: This is used as "@Access" and is used for defining the access type of either field or property. When nothing is specified, the default value taken is "field".
- javax.persistence.Id: This is used as "@Id" and is used on the attribute in a class to indicate that attribute is the primary key in the bean entity.
- javax.persistence.EmbeddedId: Used as "@EmbeddedId" upon the attribute and indicates it is a composite primary key of the bean entity.
- javax.persistence.Column: "@Column" is used for defining the column name in the database table.
- javax.persistence.GeneratedValue: "@GeneratedValue" is used for defining the strategy used for primary key generation. This annotation is used along with javax.persistence.GenerationType enum.
- javax.persistence.OneToOne: "@OneToOne" is used for defining the one-to-one mapping between two bean entities. Similarly, hibernate provides OneToMany, ManyToOne and ManyToMany annotations for defining different mapping types. org.hibernate.annotations.Cascade: "@Cascade" annotation is used for defining the cascading action between two bean entities. It is used with org.hibernate.annotations.CascadeType enum to define the type of cascading. Following is a sample class where we have used the above listed annotations:

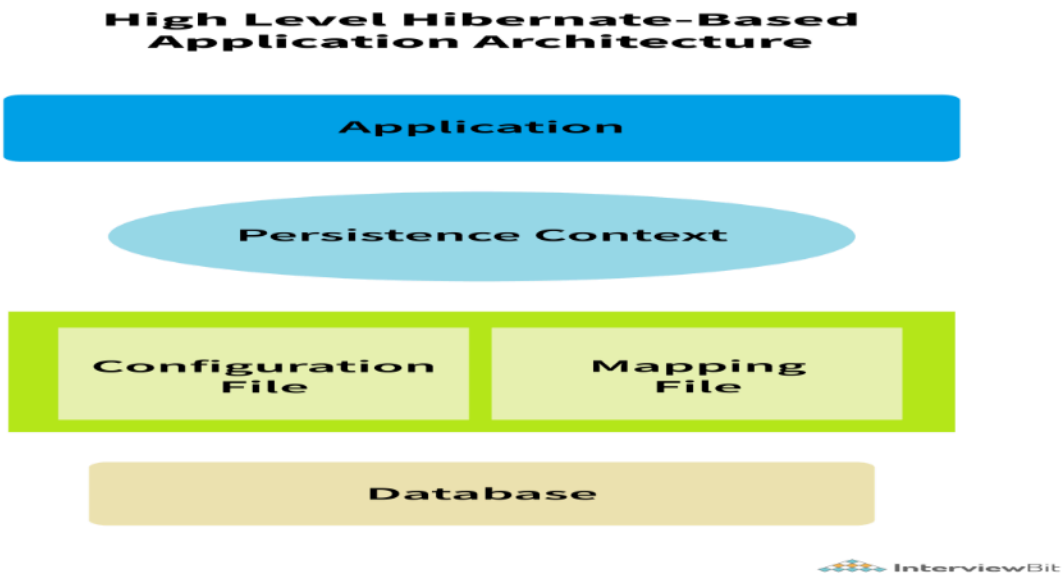
```
package com.dev.interviewbit.model;
import javax.persistence.Access;
import javax.persistence.AccessType;
import javax.persistence.Column;
```

```
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 = "InterviewBitEmployee")
@Access(value=AccessType.FIELD)
public class InterviewBitEmployee {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "employee_id")
   private long id;

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

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

   @OneToOne(mappedBy = "employee")
   @Cascade(value = org.hibernate.annotations.CascadeType.ALL)
   private Address address;

   //getters and setters methods
}
```

15. Explain Hibernate architecture

The Hibernate architecture consists of many objects such as a persistent object, session factory, session, query, transaction, etc. Applications developed using Hibernate is mainly categorized into 4 parts:

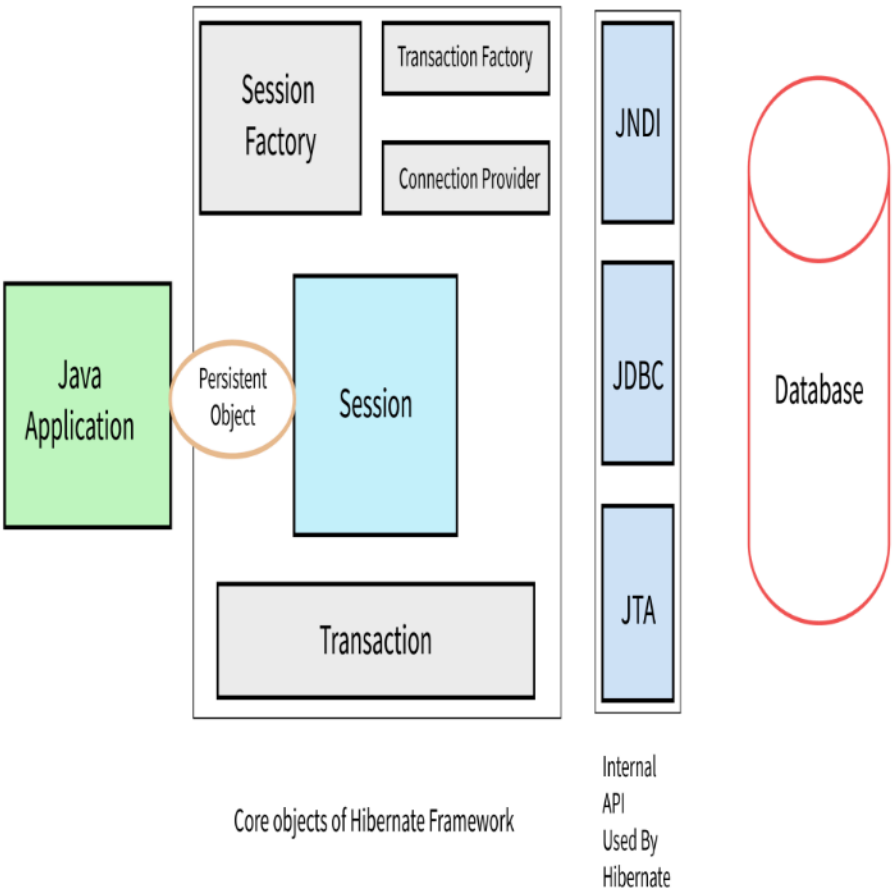
- Java Application
- Hibernate framework - Configuration and Mapping Files
- Internal API -
  - JDBC (Java Database Connectivity)
  - JTA (Java Transaction API)
  - JNDI (Java Naming Directory Interface).
- Database - MySQL, PostGreSQL, Oracle, etc



Hibernate Architecture

The main elements of Hibernate framework are:

- SessionFactory: This provides a factory method to get session objects and clients of ConnectionProvider. It holds a second-level cache (optional) of data.
- Session: This is a short-lived object that acts as an interface between the java application objects and database data.
  - The session can be used to generate transaction, query, and criteria objects.
  - It also has a mandatory first-level cache of data.
- Transaction: This object specifies the atomic unit of work and has methods useful for transaction management. This is optional.
- ConnectionProvider: This is a factory of JDBC connection objects and it provides an abstraction to the application from the DriverManager. This is optional.
- TransactionFactory: This is a factory of Transaction objects. It is optional.



Hibernate Objects

16. Can you tell the difference between getCurrentSession and openSession methods?

Both the methods are provided by the Session Factory. The main differences are given below:

getCurrentSession()	openSession()
This method returns the session bound to the context.	This method always opens a new session.
This session object scope belongs to the hibernate context and to make this work hibernate configuration file has to be modified by adding **<property name = "hibernate.current_session_context_class"> thread**	A new session object has to be created for each request in a multi-threaded environment. Hence, you

getCurrentSession()	openSession()
</property>. If not added, then using the method would throw an HibernateException.	need not configure any property to call this method.
This session object gets closed once the session factory is closed.	It's the developer's responsibility to close this object once all the database operations are done.
In a single-threaded environment, this method is faster than openSession().	In single threaded environment, it is slower than getCurrentSession()single-threadeda

Apart from these two methods, there is another method openStatelessSession() and this method returns a stateless session object.

17. Differentiate between save() and saveOrUpdate() methods in hibernate session.

Both the methods save records to the table in the database in case there are no records with the primary key in the table. However, the main differences between these two are listed below:

save()	saveOrUpdate()
save() generates a new identifier and INSERT record into a database	Session.saveOrUpdate() can either INSERT or UPDATE based upon existence of a record.
The insertion fails if the primary key already exists in the table.	In case the primary key already exists, then the record is updated.
The return type is Serializable which is the newly generated identifier id value as a Serializable object.	The return type of the saveOrUpdate() method is void.
This method is used to bring only a transient object to a persistent state.	This method can bring both transient (new) and detached (existing) objects into a persistent state. It is often used to re-attach a detached object into a Session

Clearly, saveOrUpdate() is more flexible in terms of use but it involves extra processing to find out whether a record already exists in the table or not.

18. Differentiate between get() and load() in Hibernate session

These are the methods to get data from the database. The primary differences between get and load in Hibernate are given below:

get()	load()
This method gets the data from the database as soon as it is called.	This method returns a proxy object and loads the data only when it is required.
The database is hit every time the method is called.	The database is hit only when it is really needed and this is called Lazy Loading which makes the method better.
The method returns null if the object is not found.	The method throws ObjectNotFoundException if the object is not found.
This method should be used if we are unsure about the existence of data in the database.	This method is to be used when we know for sure that the data is present in the database.

19. What is criteria API in hibernate?

Criteria API in Hibernate helps developers to build **dynamic criteria queries** on the persistence database. Criteria API is a more powerful and flexible alternative to HQL (Hibernate Query Language) queries for creating dynamic queries.

This API allows to programmatically development criteria query objects. The org.hibernate.Criteria interface is used for these purposes. The Session interface of hibernate framework has createCriteria() method that takes the persistent object's class or its entity name as the parameters and returns persistence object instance the criteria query is executed.

It also makes it very easy to incorporate restrictions to selectively retrieve data from the database. It can be achieved by using the add() method which accepts the org.hibernate.criterion.Criterion object representing individual restriction.

Usage examples:

To return all the data of InterviewBitEmployee entity class.

```
Criteria criteria = session.createCriteria(InterviewBitEmployee.class);
List<InterviewBitEmployee> results = criteria.list();
```

To retrive objects whose property has value equal to the restriction, we use Restrictions.eq() method. For example, to fetch all records with name 'Hibernate':

```
Criteria criteria= session.createCriteria(InterviewBitEmployee.class);
```

```
criteria.add(Restrictions.eq("fullName","Hibernate"));
List<InterviewBitEmployee> results = criteria.list();
```

To get objects whose property has the value "not equal to" the restriction, we use Restrictions.ne() method. For example, to fetch all the records whose employee's name is not Hibernate:

```
Criteria criteria= session.createCriteria(InterviewBitEmployee.class);
criteria.add(Restrictions.ne("fullName","Hibernate"));
List<Employee> results = criteria.list()
```

To retrieve all objects whose property matches a given pattern, we use Restrictions.like() (for case sensitivenes) and Restrictions.ilike()(for case insensitiveness)

```
Criteria criteria= session.createCriteria(InterviewBitEmployee.class);
criteria.add(Restrictions.like("fullName","Hib%",MatchMode.ANYWHERE));
List<InterviewBitEmployee> results = criteria.list();
```

Similarly, it also has other methods like isNull(), isNotNull(), gt(), ge(), lt(), le() etc for adding more varieties of restrictions. It has to be noted that for Hibernate 5 onwards, the functions returning an object of typeCriteria are deprecated. Hibernate 5 version has provided interfaces like CriteriaBuilder and CriteriaQuery to serve the purpose:

```
javax.persistence.criteria.CriteriaBuilder
javax.persistence.criteria.CriteriaQuery

// Create CriteriaBuilder
CriteriaBuilder builder = session.getCriteriaBuilder();

// Create CriteriaQuery
CriteriaQuery<YourClass> criteria = builder.createQuery(YourClass.class);
```

For introducing restrictions in CriteriaQuery, we can use the CriteriaQuery.where method which is analogous to using the WHERE clause in a JPQL query.

20. What is HQL?

Hibernate Query Language (HQL) is used as an extension of **SQL**. It is very simple, efficient, and very flexible for performing complex operations on relational databases without writing complicated queries. HQL is the object-oriented representation of query language, i.e instead of using table name, we make use of the class name which makes this language independent of any database.

This makes use of the Query interface provided by Hibernate. The Query object is obtained by calling the createQuery() method of the hibernate Session interface.

Following are the most commonly used methods of query interface:

- public int executeUpdate() : This method is used to run the update/delete query.
- public List list(): This method returns the result as a list.
- public Query setFirstResult(int rowNumber): This method accepts the row number as the parameter using which the record of that row number would be retrieved.
- public Query setMaxResult(int rowsCount): This method returns a maximum up to the specified rowCount while retrieving from the database.
- public Query setParameter(int position, Object value): This method sets the value to the attribute/column at a particular position. This method follows the JDBC style of the query parameter.
- public Query setParameter(String name, Object value): This method sets the value to a named query parameter.

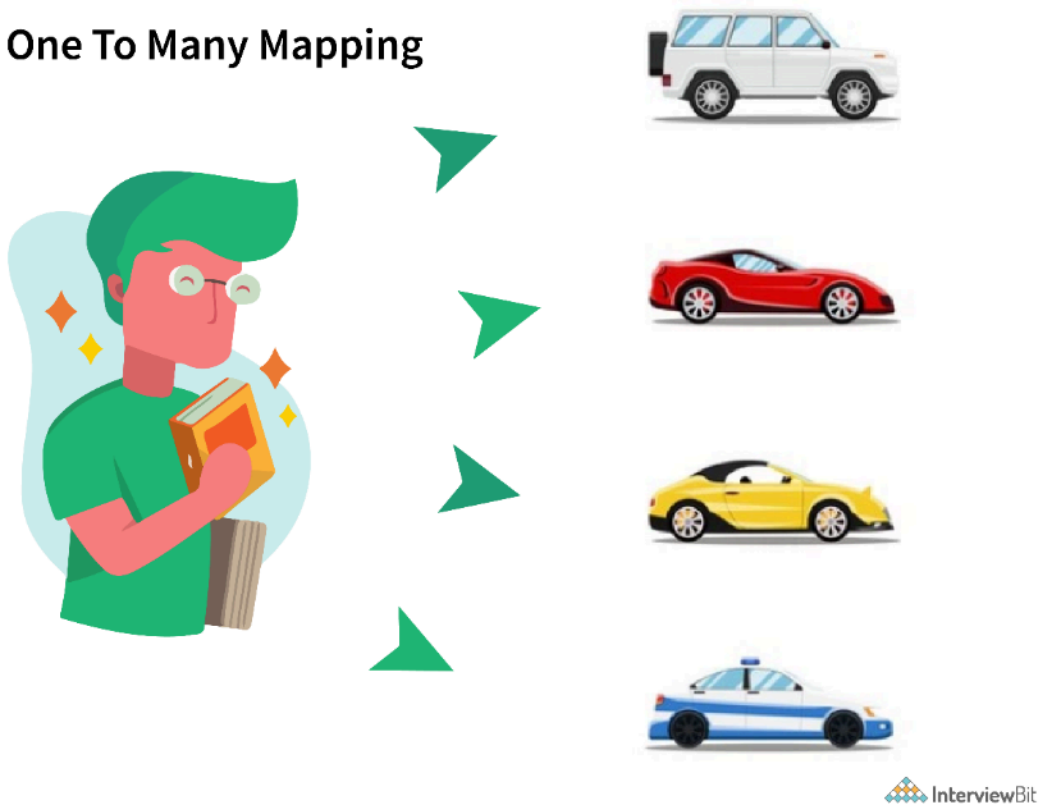
Example: To get a list of all records from InterviewBitEmployee Table:

```
Query query=session.createQuery("from InterviewBitEmployee");
List<InterviewBitEmployee> list=query.list();
System.out.println(list.get(0));
```

21. Can you tell something about one to many associations and how can we use them in Hibernate?

The **one-to-many association** is the most commonly used which indicates that one object is linked/associated with multiple objects.

For example, one person can own multiple cars.



Hibernate One To Many Mapping

In Hibernate, we can achieve this by using @OnetoMany of JPA annotations in the model classes. Consider the above example of a person having multiple cars as shown below:

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

   //...

   @OneToMany(mappedBy="owner")
   private Set<Car> cars;

   // getters and setters
}
```

In the Person class, we have defined the car's property to have @OneToMany association. The Car class would have owned property that is used by the mappedBy variable in the Person class. The Car class is as shown below:

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

   // Other Properties
```

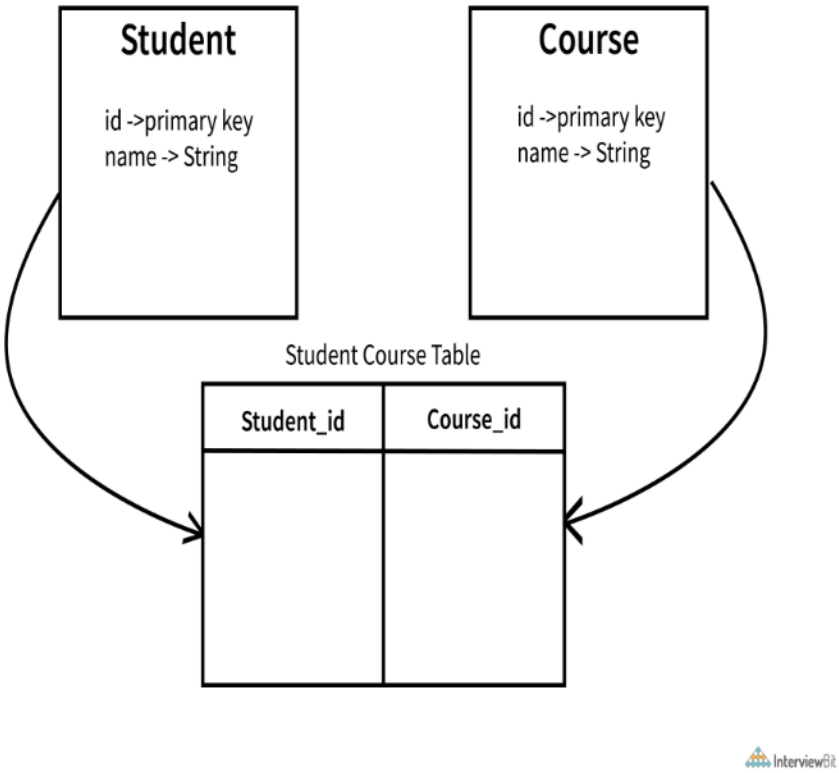
```
    @ManyToOne
    @JoinColumn(name="person_id", nullable=false)
    private Person owner;

    public Car() {}

    // getters and setters
}
```

@ManyToOne annotation indicates that many instances of an entity are mapped to one instance of another entity – many cars of one person.

22. What are Many to Many associations?

Many-to-many association indicates that there are multiple relations between the instances of two entities. We could take the example of multiple students taking part in multiple courses and vice versa.

Since both the student and course entities refer to each other by means of foreign keys, we represent this relationship technically by creating a separate table to hold these foreign keys.



Many To Many Associations

Here, Student-Course Table is called the Join Table where the student_id and course_id would form the composite primary key.

23. What does session.lock() method in hibernate do?

session.lock() method is used to reattach a detached object to the session. **session.lock()** method does not check for any data synchronization between the database and the object in the persistence context and hence this reattachment might lead to loss of data synchronization.

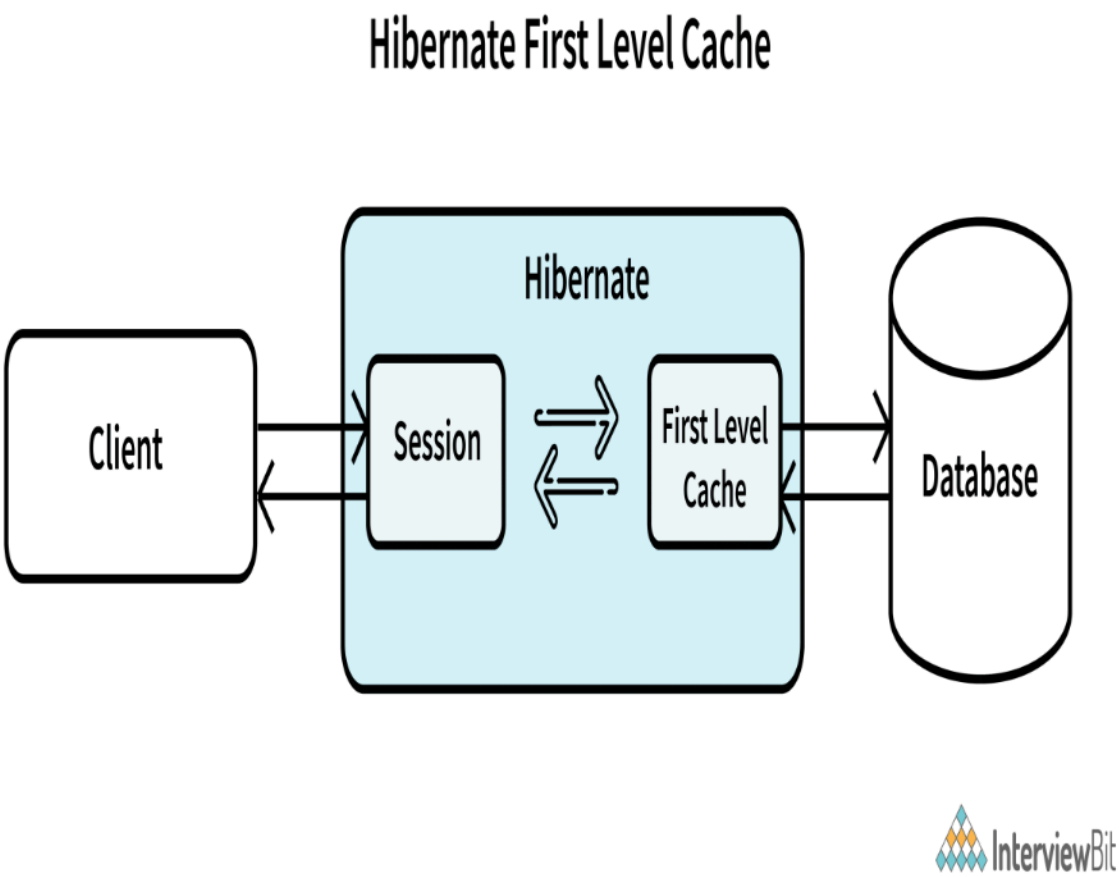
24. What is hibernate caching?

Hibernate caching is the strategy for improving the application performance by pooling objects in the cache so that the queries are executed faster. Hibernate caching is particularly useful when fetching the same data that is executed multiple times. Rather than hitting the database, we can just access the data from the cache. This results in reduced throughput time of the application.

Types of Hibernate Caching

First Level Cache:

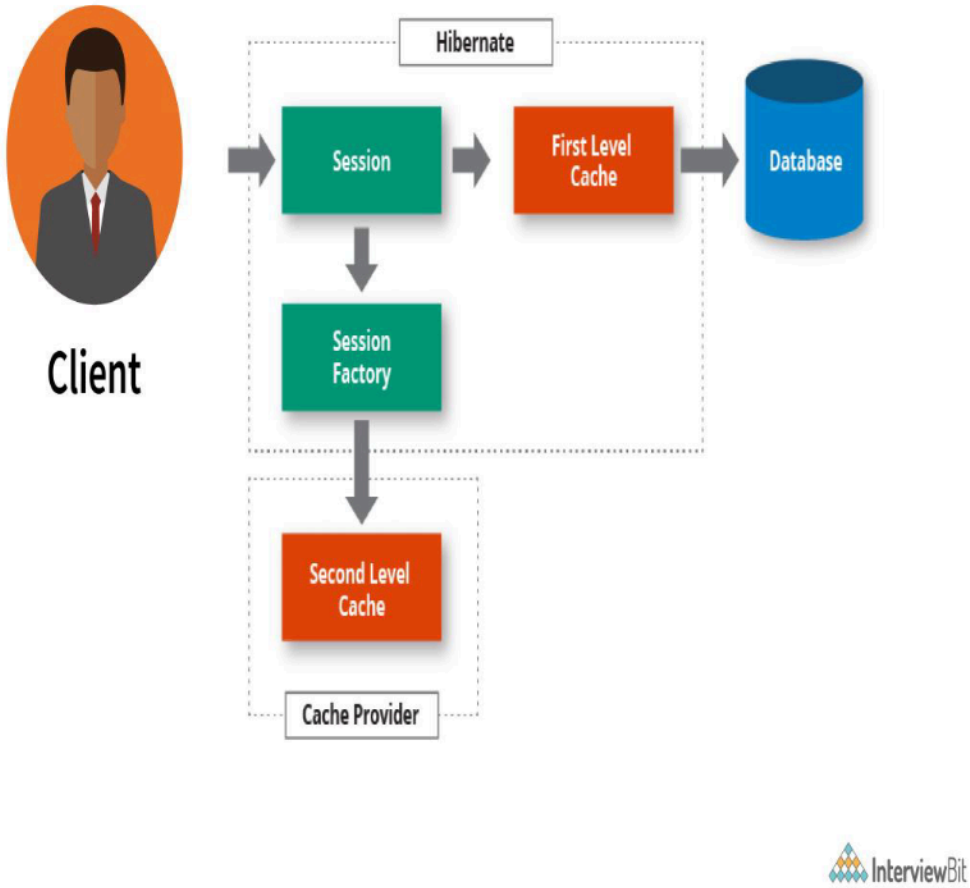
- This level is enabled by default.
- The first level cache resides in the hibernate session object.
- Since it belongs to the session object, the scope of the data stored here will not be available to the entire application as an application can make use of multiple session objects.



First Level Caching

Second Level Cache:

- Second level cache resides in the SessionFactory object and due to this, the data is accessible by the entire application.
- This is not available by default. It has to be enabled explicitly.
- EH (Easy Hibernate) Cache, Swarm Cache, OS Cache, JBoss Cache are some example cache providers.



Second Level Caching

25. When is merge() method of the hibernate session useful?

Merge() method can be used for updating existing values. The specialty of this method is, once the existing values are updated, the method creates a copy from the entity object and returns it. This result object goes into the persistent context and is then tracked for any changes. The object that was initially used is not tracked.

26. Collection mapping can be done using One-to-One and Many-to-One Associations. What do you think?

False, collection mapping is possible only with **One-to-Many** and **Many-to-Many associations.**

27. Can you tell the difference between setMaxResults() and setFetchSize() of Query?

setMaxResults() the function works similar to LIMIT in SQL. Here, we set the maximum number of rows that we want to be returned. This method is implemented by all database drivers.

setFetchSize() works for optimizing how Hibernate sends the result to the caller for example: are the results buffered, are they sent in different size chunks, etc. This method is not implemented by all the database drivers.

28. Does Hibernate support Native SQL Queries?

Yes, it does. Hibernate provides the createSQLQuery() method to let a developer call the native SQL statement directly and returns a Query object.

Consider the example where you want to get employee data with the full name "Hibernate". We don't want to use HQL-based features, instead, we want to write our own SQL queries. In this case, the code would be:

```
Query query = session.createSQLQuery( "select * from interviewbit_employee ibe whe
re ibe.fullName = :fullName")
                   .addEntity(InterviewBitEmployee.class)
                   .setParameter("fullName", "Hibernate"); //named parameters
List result = query.list();
```

Alternatively, native queries can also be supported when using NamedQueries.

Hibernate Interview Questions For Experienced

29. What happens when the no-args constructor is absent in the Entity bean?

Hibernate framework internally uses Reflection API for creating entity bean instances when get() or load() methods are called. The method Class.newInstance() is used which requires a no-args constructor to be present. When we don't have this constructor in the entity beans, then hibernate fails to instantiate the bean and hence it throws HibernateException.

30. Can we declare the Entity class final?

No, we should not define the entity class final because hibernate uses proxy classes and objects for lazy loading of data and hits the database only when it is absolutely needed. This is achieved by extending the entity bean. If the entity class (or bean) is made final, then it cant be extended and hence lazy loading can not be supported.

31. What are the states of a persistent entity?

A persistent entity can exist in any of the following states:

Transient:

- This state is the initial state of any entity object.
- Once the instance of the entity class is created, then the object is said to have entered a transient state. These objects exist in heap memory.
- In this state, the object is not linked to any session. Hence, it is not related to any database due to which any changes in the data object don't affect the data in the database.

```
InterviewBitEmployee employee=new InterviewBitEmployee(); //The object is in the t
ransient state.
   employee.setId(101);
   employee.setFullName("Hibernate");
   employee.setEmail("hibernate@interviewbit.com");
```

Persistent:

- This state is entered whenever the object is linked or associated with the session.
- An object is said to be in a persistence state whenever we save or persist an object in the database. Each object corresponds to the row in the database table. Any modifications to the data in this state cause changes in the record in the database.

Following methods can be used upon the persistence object:

```
session.save(record);
session.persist(record);
session.update(record);
session.saveOrUpdate(record);
session.lock(record);
session.merge(record);
```

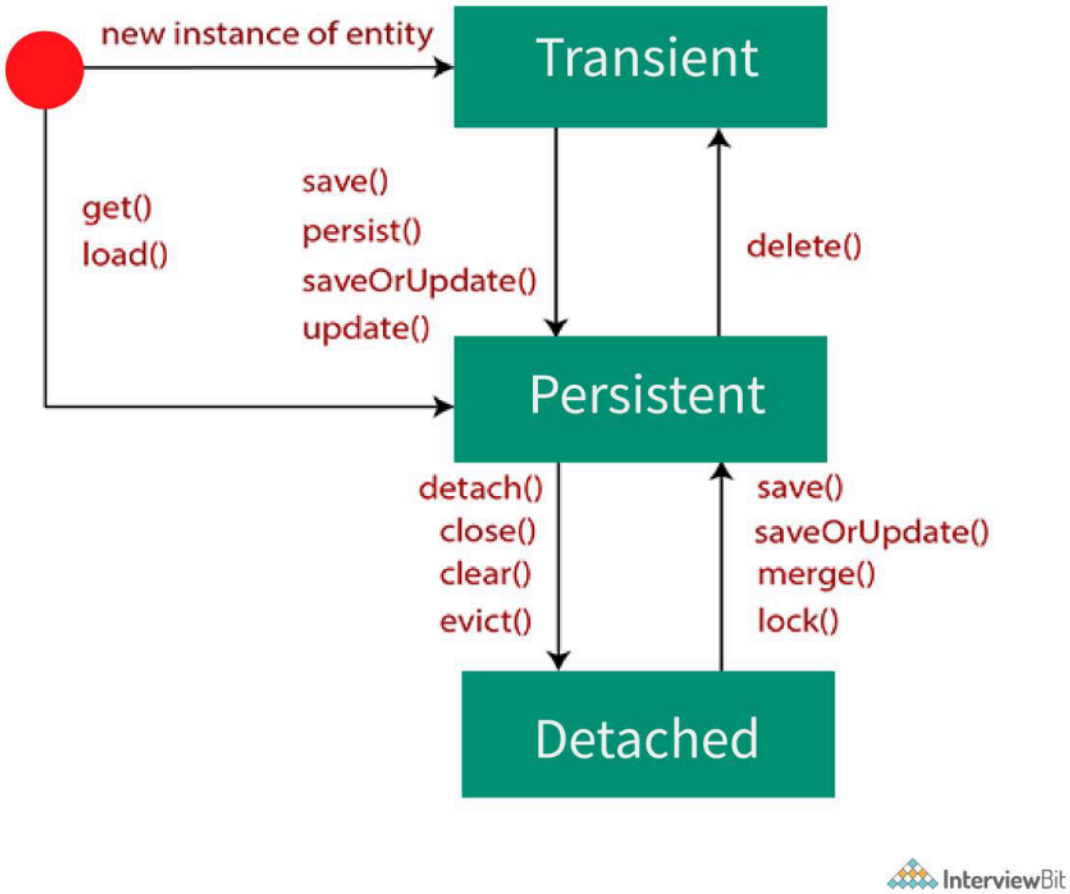
Detached:

- The object enters this state whenever the session is closed or the cache is cleared.
- Due to the object being no longer part of the session, any changes in the object will not reflect in the corresponding row of the database. However, it would still have its representation in the database.
- In case the developer wants to persist changes of this object, it has to be reattached to the hibernate session.
- In order to achieve the reattachment, we can use the methods load(), merge(), refresh(), update(), or save() methods on a new session by using the reference of the detached object.

The object enters this state whenever any of the following methods are called:

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

```
session.detach(record);
session.evict(record);
```



Persistent Entity

32. Explain Query Cache

Hibernate framework provides an optional feature called cache region for the queries' resultset. Additional configurations have to be done in code in order to enable this. The query cache is useful for those queries which are most frequently called with the same parameters. This increases the speed of the data retrieval and greatly improves performance for commonly repetitive queries.

This does not cache the state of actual entities in the result set but it only stores the identifier values and results of the value type. Hence, query cache should be always used in association with second-level cache.

Configuration:

In the hibernate configuration XML file, set the use_query_cache property to true as shown below:

```
<property name="hibernate.cache.use_query_cache">true</property>

In the code, we need to do the below changes for the query object:
Query query = session.createQuery("from InterviewBitEmployee");
query.setCacheable(true);
query.setCacheRegion("IB_EMP");
```

33. Can you tell something about the N+1 SELECT problem in Hibernate?

N+1 SELECT problem is due to the result of using lazy loading and on-demand fetching strategy. Let's take an example. If you have an N items list and each item from the list has a dependency on a collection of another object, say bid. In order to find the highest bid for each item while using the lazy loading strategy, hibernate has to first fire 1 query to load all items and then subsequently fire N queries to load big of each item. Hence, hibernate actually ends up executing N+1 queries.

34. How to solve N+1 SELECT problem in Hibernate?

Some of the strategies followed for solving the N+1 SELECT problem are:

- Pre-fetch the records in batches which helps us to reduce the problem of N+1 to (N/K) + 1 where K refers to the size of the batch.
- Subselect the fetching strategy
- As last resort, try to avoid or disable lazy loading altogether.

35. What are the concurrency strategies available in hibernate?

Concurrency strategies are the mediators responsible for storing and retrieving items from the cache. While enabling second-level cache, it is the responsibility of the developer to provide what strategy is to be implemented to decide for each persistent class and collection.

Following are the concurrency strategies that are used:

- **Transactional:** This is used in cases of updating data that most likely causes stale data and this prevention is most critical to the application.
- **Read-Only:** This is used when we don't want the data to be modified and can be used for reference data only.
- **Read-Write:** Here, data is mostly read and is used when the prevention of stale data is of critical importance.
- **Non-strict-Read-Write:** Using this strategy will ensure that there wouldn't be any consistency between the database and cache. This strategy can be used when the data can be modified and stale data is not of critical concern.

36. What is Single Table Strategy?

Single Table Strategy is a hibernate's strategy for performing inheritance mapping. This strategy is considered to be the best among all the other existing ones. Here, the inheritance data hierarchy is stored in the single table by making use of a discriminator column which determines to what class the record belongs.

For the example defined in the Hibernate Inheritance Mapping question above, if we follow this single table strategy, then all the permanent and contract employees' details are stored in only one table called InterviewBitEmployee in the database and the employees would be differentiated by making use of discriminator column named employee_type.

Hibernate provides @Inheritance annotation which takes strategy as the parameter. This is used for defining what strategy we would be using. By giving them value, InheritanceType.SINGLE_TABLE signifies that we are using a single table strategy for mapping.

- @DiscriminatorColumn is used for specifying what is the discriminator column of the table in the database corresponding to the entity.
- @DiscriminatorValue is used for specifying what value differentiates the records of two types.

The code snippet would be like this:

InterviewBitEmployee class:

```
@Entity
@Table(name = "InterviewBitEmployee")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "employee_type")
@NoArgsConstructor
@AllArgsConstructor
public class InterviewBitEmployee {
   @Id
   @Column(name = "employee_id")
   private String employeeId;
   private String fullName;
   private String email;
}
```

InterviewBitContractEmployee class:

```
@Entity
@DiscriminatorValue("contract")
@NoArgsConstructor
@AllArgsConstructor
public class InterviewBitContractEmployee extends InterviewBitEmployee {
   private LocalDate contractStartDate;
   private LocalDate contractEndDate;
```

```
   private String agencyName;
}
```

InterviewBitPermanentEmployee class:

```
@Entity
@DiscriminatorValue("permanent")
@NoArgsConstructor
@AllArgsConstructor
public class InterviewBitPermanentEmployee extends InterviewBitEmployee {
   private LocalDate workStartDate;
   private int numberOfLeaves;
}
```

37. Can you tell something about Table Per Class Strategy.

Table Per Class Strategy is another type of inheritance mapping strategy where each class in the hierarchy has a corresponding mapping database table. For example, the InterviewBitContractEmployee class details are stored in the interviewbit_contract_employee table and InterviewBitPermanentEmployee class details are stored in interviewbit_permanent_employee tables respectively. As the data is stored in different tables, there will be no need for a discriminator column as done in a single table strategy.

Hibernate provides @Inheritance annotation which takes strategy as the parameter. This is used for defining what strategy we would be using. By giving them value, InheritanceType.TABLE_PER_CLASS, it signifies that we are using a table per class strategy for mapping.

The code snippet will be as shown below:

InterviewBitEmployee class:

```
@Entity(name = "interviewbit_employee")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
@NoArgsConstructor
@AllArgsConstructor
public class InterviewBitEmployee {
   @Id
   @Column(name = "employee_id")
   private String employeeId;
   private String fullName;
   private String email;
}
```

InterviewBitContractEmployee class:

```
@Entity(name = "interviewbit_contract_employee")
@Table(name = "interviewbit_contract_employee")
@NoArgsConstructor
@AllArgsConstructor
```

```
public class InterviewBitContractEmployee extends InterviewBitEmployee {
   private LocalDate contractStartDate;
   private LocalDate contractEndDate;
   private String agencyName;
}
```

InterviewBitPermanentEmployee class:

```
@Entity(name = "interviewbit_permanent_employee")
@Table(name = "interviewbit_permanent_employee")
@NoArgsConstructor
@AllArgsConstructor
public class InterviewBitPermanentEmployee extends InterviewBitEmployee {
   private LocalDate workStartDate;
   private int numberOfLeaves;
}
```

Disadvantages:

- This type of strategy offers less performance due to the need for additional joins to get the data.
- This strategy is not supported by all JPA providers.
- Ordering is tricky in some cases since it is done based on a class and later by the ordering criteria.

38. Can you tell something about Named SQL Query

A named SQL query is an expression represented in the form of a table. Here, SQL expressions to select/retrieve rows and columns from one or more tables in one or more databases can be specified. This is like using aliases to the queries.

In hibernate, we can make use of @NameQueries and @NameQuery annotations.

- @NameQueries annotation is used for defining multiple named queries.
- @NameQuery annotation is used for defining a single named query.

Code Snippet: We can define Named Query as shown below

```
@NamedQueries(
   {
       @NamedQuery(
       name = "findIBEmployeeByFullName",
       query = "from InterviewBitEmployee e where e.fullName = :fullName"
       )
   }
)
```

:fullName refers to the parameter that is programmer defined and can be set using the query.setParameter method while using the named query.

Usage:

```
TypedQuery query = session.getNamedQuery("findIBEmployeeByFullName");
query.setParameter("fullName","Hibernate");
List<InterviewBitEmployee> ibEmployees = query.getResultList();
```

The getNamedQuery method takes the name of the named query and returns the query instance.

39. What are the benefits of NamedQuery?

In order to understand the benefits of NamedQuery, let's first understand the disadvantage of HQL and SQL. The main disadvantage of having HQL and SQL scattered across data access objects is that it makes the code unreadable. Hence, as good practice, it is recommended to group all HQL and SQL codes in one place and use only their reference in the actual data access code. In order to achieve this, Hibernate gives us named queries.

A named query is a statically defined query with a predefined unchangeable query string. They are validated when the session factory is created, thus making the application fail fast in case of an error.

Conclusion

Hibernate is the most powerful open source ORM tool that is used for mapping java objects with the database structures at run time. It has become more popular among software developers due to its nature of abstraction allowing developers to continue application development by being database independent and focus just on the application business logic.

Additional Resources

Practice Coding

Java Tutorials

Java Interview Questions

Hibernate MCQs

1.

Which among the below do not belong to the core interface of Hibernate?

- SessionManagement

- Configuration
- Session
- Criteria

2.

Which method returns the proxy object?

- getDatabase()
- get()
- loadDatabase()
- load()

3.

Which of the following method is used inside sessions only?

- end()
- merge()
- update()
- kill()

4.

Hibernate uses PersisterClassProvider by default.

- True
- False
- There is no such class named PersisterClassProvider
- Nothing is set by default

5.

Which among the below is held by the Session object?

- First Level Cache
- Second Level Cache
- Both First and Second Level Cache
- None of the above

6.

What are the ways to map tables in the case of Table per Concrete class where there would be three tables in the database that have no relation to each other?

- By self-creating the table for each class
- By union-subclass element
- Both a & b
- None of the above

7.

What holds the second level cache as per the Hibernate architecture?

- Session
- SessionFactory
- Connection
- Transaction

8.

Which of the following is true in the case of Hibernate annotations?

- By using annotations, we ensure that all metadata is clubbed into java file thereby helping user to understand the table and POJO structure simultaneously
- They are the powerful means to provide the metadata of the Object and Relational Table mapping.
- Both of the above
- None of the above

9.

Which method among the following options hits the database always?

- getDatabase()
- load()
- get()
- loadDatabase()

10.

Which of the following statements is FALSE about Session?

- Sessions can be shared between threads
- It is lightweight and a non-threadsafe object
- It is the primary interface for the persistence service

- **It represents a single unit-of-work with the database**

Q1. What is Hibernate?

HIBERNATE *Hibernate* is one of the most popular *Java frameworks* that simplify the development of Java application to interact with the database. It is an Object-relational mapping (ORM) tool. Hibernate also provides a reference implementation of Java API.

It is referred as a framework which comes with an abstraction layer and also handles the implementations internally. The implementations include tasks like writing a query for *CRUD*operations or establishing a connection with the databases, etc.

Hibernate develops persistence logic, which stores and processes the data for longer use. It is a lightweight tool and most importantly **open-sourced** which gives it an edge over other frameworks.

Q2. What are the major advantages of Hibernate Framework?

- It is open-sourced and lightweight.
- Performance of Hibernate is very fast.
- Helps in generating database independant queries.
- Provides facilities to automatically create a table.
- It provides query statistics and database status.

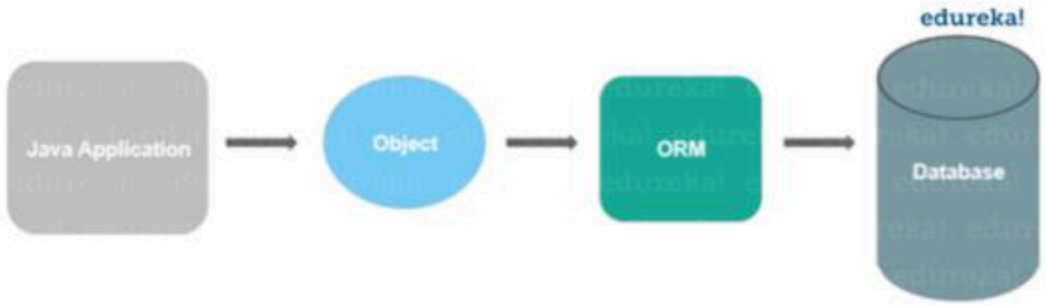
Q3. What are the advantages of using Hibernate over JDBC?

Major advantages of using Hibernate over JDBC are:

1. Hibernate eliminates a lot of boiler-plate code that comes with *JDBC API*, the code looks cleaner and readable.
2. This Java framework supports *inheritance*, associations, and collections. These features are actually not present in JDBC.
3. HQL (Hibernate Query Language) is more object-oriented and close to Java. But for JDBC, you need to write native SQL queries.
4. Hibernate implicitly provides transaction management whereas, in JDBC API, you need to write code for transaction management using *commit* and *rollback*.
5. JDBC throws SQLException that is a checked exception, so you have to write a lot of try-catch block code. Hibernate wraps JDBC exceptions and throw *JDBCException* or *HibernateException*which are the unchecked exceptions, so you don't have to write code to handle it has built-in transaction management which helps in removing the usage of try-catch blocks.

Q4. What is an ORM tool?

It is basically a technique that maps the object that is stored in the database. An ORM tool helps in simplifying data creation, manipulation, and access. It internally uses the Java API to interact with the databases.



Q5. Why use Hibernate Framework?

Hibernate overcomes the shortcomings of other technologies like *JDBC*.

- It overcomes the database dependency faced in the JDBC.
- Changing of the databases cost a lot working on JDBC, hibernate overcomes this problem with flying colors.
- Code portability is not an option while working on JDBC. This is easily handled by Hibernate.
- Hibernate strengthens the object level relationship.
- It overcomes the *exception-handling* part which is mandatory while working on JDBC.
- It reduces the length of code with increased readability by overcoming the boilerplate problem.

Q6. What are the different functionalities supported by Hibernate?

- Hibernate is an ORM tool.
- Hibernate uses Hibernate Query Language(HQL) which makes it database-independent.
- It supports auto DDL operations.
- This Java framework also has an *Auto Primary Key Generation* support.
- Supports cache memory.
- Exception handling is not mandatory in the case of Hibernate.

Q7. What are the technologies that are supported by Hibernate?

Hibernate supports a variety of technologies, like:

- XDoclet Spring
- *Maven*
- Eclipse Plug-ins
- *J2EE*

Q8. What is HQL?

HQL is the acronym of Hibernate Query Language. It is an Object-Oriented Query Language and is independent of the database.

Q9. How to achieve mapping in Hibernate?

Association mappings are one of the key features of Hibernate. It supports the same associations as the relational database model. They are:

- One-to-One associations
- Many-to-One associations
- Many-to-Many associations

You can map each of them as a uni- or bidirectional association.

Q10. Name some of the important interfaces of Hibernate framework?

Hibernate interfaces are:

- **SessionFactory** (org.hibernate.SessionFactory)
- **Session** (org.hibernate.Session)
- **Transaction** (org.hibernate.Transaction)

Q11. What is One-to-One association in Hibernate?

In this type of mapping, you only need to model the system for the entity for which you want to navigate the relationship in your query or domain model. You need an entity attribute that represents the association, so annotate it with an @*OneToOne* annotation.

Q12. What is One-to-Many association in Hibernate?

In this type of association, one object can be associated with multiple/different objects. Talking about the mapping, the One-to-Many mapping is implemented using a *Set Java* collection that does not have any redundant element. This *One-to-Many* element of the set indicates the relation of one object to multiple objects.

Q13. What is Many-to-Many association in Hibernate?

Many-to-Many mapping requires an entity attribute and a @*ManyToMany* annotation. It can either be unidirectional and bidirectional. In **Unidirectional**, the attributes model the association and you can use it to navigate it in your domain model or JPQL queries. The annotation tells Hibernate to map a Many-to-Many association. The **bidirectional** relationship, mapping allows you to navigate the association in both directions.

Q14. How to integrate Hibernate and Spring?

Spring is also one of the most commonly used Java frameworks in the market today. Spring is a JavaEE Framework and Hibernate is the most popular ORM framework. This is why Spring Hibernate combination is used in a lot of enterprise applications.

Following are the steps you should follow to integrate Spring and Hibernate.

1. Add Hibernate-entity manager, Hibernate-core and Spring-ORM dependencies.
2. Create Model classes and corresponding DAO implementations for database operations. The DAO classes will use *SessionFactory* that will be injected by the Spring Bean configuration.
3. Note that you don't need to use Hibernate Transaction Management, as you can leave it to the Spring declarative transaction management using *@Transactional annotation.*

Q15. What do you mean by Hibernate Configuration File?

Hibernate Configuration File mainly contains database-specific configurations and are used to initialize *SessionFactory*. Some important parts of the Hibernate Configuration File are Dialect information, so that hibernate knows the database type and mapping file or class details.

Hibernate Interview Questions for intermediate

Q16. Mention some important annotations used for Hibernate mapping?

Hibernate supports JPA annotations. Some of the major annotations are:

1. **javax.persistence.Entity:** This is used with model classes to specify they are entity beans.
2. **javax.persistence.Table:** It is used with entity beans to define the corresponding table name in the database.
3. **javax.persistence.Access:** Used to define the access type, field or property. The default value is field and if you want Hibernate to use the getter/setter methods then you need to set it to a property.
4. **javax.persistence.Id:** Defines the primary key in the entity bean.
5. **javax.persistence.EmbeddedId:** It defines a composite primary key in the entity bean.
6. **javax.persistence.Column:** Helps in defining the column name in the database table.
7. **javax.persistence.GeneratedValue:** It defines the strategy to be used for the generation of the primary key. It is also used in conjunction with *javax.persistence.GenerationType* enum.

Q17. What is Session in Hibernate and how to get it?

Hibernate Session is the interface between Java application layer and Hibernate. It is used to get a physical connection with the database. The *Session* object created is lightweight and designed to be instantiated each time an interaction is needed with the database. This *Session* provides methods to create, read, update and delete operations for a constant object. To get the Session, you can execute HQL queries, SQL native queries using the *Session* object.

Q18. What is Hibernate SessionFactory?

SessionFactory is the factory class that is used to get the Session objects. The SessionFactory is a heavyweight object so usually, it is created during application startup and kept for later use. This *SessionFactory* is a thread-safe object which is

used by all the threads of an application. If you are using multiple databases then you would have to create multiple *SessionFactory* objects.

Q19. What is the difference between openSession and getCurrentSession?

This *getCurrentSession()* method returns the session bound to the context and for this to work, you need to configure it in Hibernate configuration file. Since this session object belongs to the context of Hibernate, it is okay if you don't close it. Once the *SessionFactory* is closed, this session object gets closed.

openSession() method helps in opening a new session. You should close this session object once you are done with all the database operations. And also, you should open a new session for each request in a multi-threaded environment.

Q20. What do you mean by Hibernate configuration file?

The following steps help in configuring Hibernate file:

1. First, identify the POJOs (Plain Old Java Objects) that have a database representation.
2. Identify which properties of POJOs need to be continued.
3. Annotate each of the POJOs in order to map the Java objects to columns in a database table.
4. Create a database schema using the schema export tool which uses an existing database, or you can create your own database schema.
5. Add Hibernate Java libraries to the application's classpath.
6. Create a Hibernate *XML configuration file* that points to the database and the mapped classes.
7. In the Java application, you can create a Hibernate Configuration object that refers to your XML configuration file.
8. Also, build a Hibernate SessionFactory object from the Configuration object.

9. Retrieve the Hibernate Session objects from the SessionFactory and write down the data access logic for your application (create, retrieve, update, and delete).

Q21. What are the key components of a Hibernate configuration object?

The configuration provides 2 key components, namely:

- *Database Connection:* This is handled by one or more configuration files.
- *Class Mapping setup*: It helps in creating the connection between Java classes and database tables.

Q22. Discuss the Collections in Hibernate

Hibernate provides the facility to persist the Collections. A *Collection* basically can be a List, Set, Map, Collection, Sorted Set, Sorted Map. java.util.List, java.util.Set, java.util.Collection, etc, are some of the real interface types to declared the persistent collection-value fields. Hibernate injects persistent Collections based on the type of interface. The collection instances generally behave like the types of value behavior.

Q23. What are the collection types in Hibernate?

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

- Bag
- Set
- List
- Array
- Map

Q24. What is a Hibernate Template class?

When you integrate Spring and Hibernate, Spring ORM provides two helper classes – HibernateDaoSupport and HibernateTemplate. The main reason to use them was to get two things, the Session from Hibernate and Spring Transaction Management. However, from Hibernate 3.0.1, you can use the SessionFactory getCurrentSession() method to get the current session. The major advantage of using this Template class is the **exception translation** but that can be achieved easily by using *@Repository* annotation with service classes.

Q25. What are the benefits of using Hibernate template?

The following are the benefits of using this Hibernate template class:

- Automated Session closing ability.
- The interaction with the Hibernate Session is simplified.
- Exception handling is automated.

Q26. Which are the design patterns that are used in Hibernate framework?

There are a few design patterns used in Hibernate Framework, namely:

- *Domain Model Pattern:* An object model of the domain that incorporates both behavior as well as data.
- *Data Mapper:* A layer of the map that moves data between objects and a database while keeping it independent of each other and the map itself.
- *Proxy Pattern:* It is used for lazy loading.
- *Factory Pattern:* Used in SessionFactory.

Q27. Define Hibernate Validator Framework

Data validation is considered as an integral part of any application. Also, data validation is used in the presentation layer with the use of Javascript and the server-side code before processing. It occurs before persisting it in order to make sure it follows the correct format. Validation is a cross-cutting task, so we should try to keep it apart from the business logic. This Hibernate Validator provides the reference implementation of bean validation specs.

Q28. What is Dirty Checking in Hibernate?

Hibernate incorporates *Dirty Checking* feature that permits developers and users to avoid time-consuming write actions. This Dirty Checking feature changes or updates fields that need to be changed or updated, while keeping the remaining fields untouched and unchanged.

Q29. How can you share your views on mapping description files?

- Mapping description files are used by the Hibernate to configure functions.
- These files have the ***.hbm** extension, which facilitates the mapping between database tables and Java class.
- Whether to use mapping description files or not this entirely depends on business entities.

Q30. What is meant by Light Object Mapping?

The means that the syntax is hidden from the business logic using specific design patterns. This is one of the valuable levels of ORM quality and this Light Object Mapping approach can be successful in case of applications where there are very fewer entities, or for applications having data models that are metadata-driven.

Hibernate Interview Questions for experienced

Q31. What is meant by Hibernate tuning?

Optimizing the performance of Hibernate applications is known as Hibernate tuning.

The performance tuning strategies for Hibernate are:

1. SQL Optimization
2. Session Management
3. Data Caching

Q32. What is Transaction Management in Hibernate? How does it work?

Transaction Management is a property which is present in the Spring framework. Now, what role does it play in Hibernate?

Transaction Management is a process of managing a set of commands or statements. In hibernate, Transaction Management is done by transaction interface. It maintains abstraction from the transaction implementation (JTA, JDBC). A transaction is associated with Session and is instantiated by calling *session.beginTransaction()*.

Q33. How do you integrate Hibernate with Struts2 or Servlet web applications?

You can integrate any Struts application with Hibernate. There are no extra efforts required.

1. Register a custom *ServletContextListener*.
2. In the *ServletContextListener* class, first, initialize the Hibernate Session, store it in the servlet context.
3. Action class helps in getting the Hibernate Session from the servlet context, and perform other Hibernate task as normal.

Q34. What are the different states of a persistent entity?

It may exist in one of the following 3 states:

- Transient: This is not associated with the Session and has no representation in the database.
- Persistent: You can make a transient instance persistent by associating it with a Session.
- Detached: If you close the Hibernate Session, the persistent instance will become a detached instance.

Q35. How can the primary key be created by using Hibernate?

A Primary key is a special relational database table column designated to uniquely identify all table records. It is specified in the configuration file *hbm.xml*. The generator can also be used to specify how a Primary key can be created in the database.

```
1   <id name="ClassID" type="string" >
2   <column name= "columnID" length="10" >
3   <generator/>
4   </id>
```

Q36. Explain about Hibernate Proxy and how it helps in Lazy loading?

- Hibernate uses a proxy object in order to support *Lazy loading*.
- When you try loading data from tables, Hibernate doesn't load all the mapped objects.
- After you reference a child object through getter methods, if the linked entity is not present in the session cache, then the proxy code will be entered to the database and load the linked object.
- It uses Java assist to effectively and dynamically generate sub-classed implementations of your entity objects.

Q37. How can we see Hibernate generated SQL on console?

In order to view the SQL on a console, you need to add following in Hibernate configuration file to enable viewing SQL on the console for debugging purposes:

```
<property name="show_sql">true</property>
```

Q38. What is Query Cache in Hibernate?

Hibernate implements a separate cache region for queries resultset that integrates with the Hibernate second-level cache. This is also an optional feature and requires a few more steps in code.

Programming & Frameworks Training

Full Stack Web Development Internship Program

Reviews

5(2005)

Java Certification Training Course

Reviews

4(47134)

Python Scripting Certification Training

Reviews

5(10077)

Python Django Training and Certification

Reviews

5(5655)

Spring Framework Certification Training

Reviews

5(8658)

Node.js Certification Training

Reviews

5(6535)

PHP & MySQL with MVC Frameworks Certification Training

Reviews

5(3591)

Mastering Perl Scripting Certification Training

Reviews

5(4358)

Advanced Java Certification Training

Reviews

5(3851)

Next

Note: *This is only useful for queries that are run frequently with the same parameters.*

Q39. What is the benefit of Native SQL query support in Hibernate?

Hibernate provides an option to execute Native SQL queries through the use of the *SQLQuery* object. For normal scenarios, it is however not the recommended approach because you might lose other benefits like Association and Hibernate first-level caching.

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

Q40. What is Named SQL Query?

Hibernate provides another important feature called Named Query using which you can define at a central location and use them anywhere in the code.

You can create named queries for both HQL as well as for Native SQL. These Named Queries can be defined in Hibernate mapping files with the help of JPA annotations @NamedQuery and @NamedNativeQuery.

Q41. When do you use merge() and update() in Hibernate?

This is one of the tricky Hibernate Interview Questions asked.

update(): If you are sure that the Hibernate Session does not contain an already persistent instance with the same *id* .
merge(): Helps in merging your modifications at any time without considering the state of the Session.

Q42. Difference between get() vs load() method in Hibernate?

This is one of the most frequently asked Hibernate Interview Questions. The key difference between the get() and load() method is:

load(): It will throw an exception if an object with an ID passed to them is not found.
get(): Will return null.

load(): It can return proxy without hitting the database unless required.
get(): It always goes to the database.

So sometimes using *load()* can be faster than the *get()* method.

Q43. Difference between the first and second level cache in Hibernate?

The *first-level cache* is maintained at Session level while the *second level cache* is maintained at a SessionFactory level and is shared by all sessions.

Q44. Difference between Session and SessionFactory in Hibernate?

This is yet another popular Hibernate Interview Question asked.

- *A Session* is a single-threaded, short-lived object. It provides the first-level cache.
- SessionFactory is immutable and shared by all Session. It also lives until the Hibernate is running. It also provides the second-level cache.

Q45. Difference between save() and saveOrUpdate() method of Hibernate?

Even though *save()* and *saveOrUpdate()* method is used to store an object into Database, the key difference between them is that *save()* can only **Insert** records but *saveOrUpdate()* can either Insert or Update records.

Q46. Difference between sorted and ordered collection in Hibernate?

sorted collection sort the data in JVM's heap memory using Java's collection framework sorting methods. *The ordered collection* is sorted using order by clause in the database itself.

Note: *A sorted collection is more suited for small dataset but for a large dataset, it's better to use ordered collection to avoid*

Q47. Difference between the transient, persistent and detached state in Hibernate?

***Transient* state:** New objects are created in the Java program but are not associated with any Hibernate Session.

Persistent state: 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.

See Batch Details

Detached state: You can re-attach a detached object to Hibernate sessions by calling either update() or saveOrUpdate() method.

Q48. Difference between managed associations and Hibernate associations?

Managed associations: Relate to container management persistence and are bi-directional.

Hibernate Associations: These associations are unidirectional.

Q49. What are the best practices that Hibernate recommends for persistent classes?

- All Java classes that will be persisted need a default constructor.
- All classes should contain an ID in order to allow easy identification of your objects within Hibernate and the database. This property maps to the primary key column of a database table.
- All attributes that will be persisted should be declared private and have **getXXX** and **setXXX** methods defined in the JavaBean style.
- A central feature of Hibernate, proxies, depends upon the persistent class being either non-final, or the implementation of an interface that declares all public methods.
- All classes that do not extend or implement some specialized classes and interfaces required by the EJB framework.

Q50. What are the best practices to follow with Hibernate framework?

- Always check the primary key field access, if it's generated at the database layer then you should not have a setter for this.
- 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).
- 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.
- Use native sql query only when it can't be done using HQL, such as using the database-specific feature.
- If you have to sort the collection, use ordered list rather than sorting it using Collection API.
- 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.
- For web applications, always try to use JNDI DataSource rather than configuring to create a connection in hibernate.
- Avoid Many-to-Many relationships, it can be easily implemented using bidirectional One-to-Many and Many-to-One relationships.
- For collections, try to use Lists, maps and sets. Avoid array because you don't get benefit of lazy loading.
- Do not treat exceptions as recoverable, roll back the Transaction and close the Session. If you do not do this, Hibernate cannot guarantee that the in-memory state accurately represents the persistent state.

- Prefer DAO pattern for exposing the different methods that can be used with entity bean
- Prefer lazy fetching for associations

ibernate Interview Questions

1. What is Hibernate Framework?
2. What is Java Persistence API (JPA)?
3. What are the important benefits of using Hibernate Framework?
4. What are the advantages of Hibernate over JDBC?
5. Name some important interfaces of Hibernate framework?
6. What is hibernate configuration file?
7. What is hibernate mapping file?
8. Name some important annotations used for Hibernate mapping?
9. What is Hibernate SessionFactory and how to configure it?
10. Hibernate SessionFactory is thread safe?
11. What is Hibernate Session and how to get it?
12. Hibernate Session is thread safe?
13. What is difference between openSession and getCurrentSession?
14. What is difference between Hibernate Session get() and load() method?
15. What is hibernate caching? Explain Hibernate first level cache?
16. How to configure Hibernate Second Level Cache using EHCache?
17. What are different states of an entity bean?
18. What is use of Hibernate Session merge() call?
19. What is difference between Hibernate save(), saveOrUpdate() and persist() methods?
20. What will happen if we don't have no-args constructor in Entity bean?
21. What is difference between sorted collection and ordered collection, which one is better?
22. What are the collection types in Hibernate?
23. How to implement Joins in Hibernate?
24. Why we should not make Entity Class final?
25. What is HQL and what are it's benefits?
26. What is Query Cache in Hibernate?
27. Can we execute native sql query in hibernate?
28. What is the benefit of native sql query support in hibernate?
29. What is Named SQL Query?
30. What are the benefits of Named SQL Query?
31. What is the benefit of Hibernate Criteria API?
32. How to log hibernate generated sql queries in log files?
33. What is Hibernate Proxy and how it helps in lazy loading?
34. How to implement relationships in hibernate?
35. How transaction management works in Hibernate?
36. What is cascading and what are different types of cascading?
37. How to integrate log4j logging in hibernate application?
38. How to use application server JNDI DataSource with Hibernate framework?

39. How to integrate Hibernate and Spring frameworks?
40. What is HibernateTemplate class?
41. How to integrate Hibernate with Servlet or Struts2 web applications?
42. Which design patterns are used in Hibernate framework?
43. What are best practices to follow with Hibernate framework?
44. What is Hibernate Validator Framework?
45. What is the benefit of Hibernate Tools Eclipse plugin?

Hibernate Interview Questions and Answers

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.

2. 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.

3. 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.

4. 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 more 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.
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.

5. 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.

6. 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.

7. **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.

8. **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;
```

```java
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
}
```

9. 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.

10. 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.

11. 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.

12. 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.

13. 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</proper
ty>
```

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].

14. 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.

15. 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.

16. How to configure Hibernate Second Level Cache using EHCache?

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

- Add hibernate-ehcache dependency in your maven project, if it's not maven then add corresponding jars.
-
- `<dependency>`

```
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-ehcache</artifactId>
        <version>4.3.5.Final</version>
</dependency>
```

- Add below properties in hibernate configuration file.

```
<property name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>

<!-- For singleton 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>
```

- Create EHCache configuration file, a sample file myehcache.xml would look like below.

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="https://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>
```

- Annotate entity beans with @Cache annotation and caching strategy to use. For example,

```
import org.hibernate.annotations.Cache;
import org.hibernate.annotations.CacheConcurrencyStrategy;

@Entity
@Table(name = "ADDRESS")
@Cache(usage=CacheConcurrencyStrategy.READ_ONLY, region="employee")
public class Address {

}
```

That's it, we are done. Hibernate will use the EHCache for second level caching, read Hibernate EHCache Example for a complete example with explanation.

17. What are different states of an entity bean?

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

0. **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().
1. **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.
2. **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().

18. 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.

19. 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.

20. What will happen if we don't have no-args constructor in Entity bean?

Hibernate uses Reflection API 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`.

21. 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].

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.

22. What are the collection types in Hibernate?

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

0. Bag
1. Set
2. List
3. Array
4. Map

23. How to implement Joins in Hibernate?

There are various ways to implement joins in hibernate.

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

24. 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.

25. 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.

26. 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");
```

27. 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.

28. 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.

29. 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.

30. 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.

31. 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:

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

Learn some quick examples at [Hibernate Criteria Example](#).

32. 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.

33. 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.

34. 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.

0. Hibernate One to One Mapping
1. Hibernate One to Many Mapping
2. Hibernate Many to Many Mapping

35. 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.

36. 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:

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

37. 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.

- Add log4j dependencies for maven project, if not maven then add corresponding jar files.
- 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.
- 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 and Servlet log4j example.

38. 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/MyL
ocalDB</property>
```

For a complete example, go through Hibernate JNDI DataSource Example.

39. 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.

0. Add hibernate-entitymanager, hibernate-core and spring-orm dependencies.
1. 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.
2. If you are using Hibernate 3, you need to configure `org.springframework.orm.hibernate3.LocalSessionFactoryBean` or `org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean` in Spring Bean configuration file. For Hibernate 4, there is single class `org.springframework.orm.hibernate4.LocalSessionFactoryBean` that should be configured.
3. 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 and Spring MVC Hibernate Integration.

40. 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.

41. 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.

42. Which design patterns are used in Hibernate framework?

Some of the design patterns used in Hibernate Framework are:

- Domain Model Pattern – An object model of the domain that incorporates both behavior and data.
- 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.
- Proxy Pattern for lazy loading
- Factory pattern in SessionFactory

43. What are best practices to follow with Hibernate framework?

Some of the best practices to follow in Hibernate are:

- Always check the primary key field access, if it's generated at the database layer then you should not have a setter for this.
- 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)`.
- 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.
- Use native sql query only when it can't be done using HQL, such as using database specific feature.
- If you have to sort the collection, use ordered list rather than sorting it using Collection API.
- 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.
- For web applications, always try to use JNDI DataSource rather than configuring to create connection in hibernate.
- Avoid Many-to-Many relationships, it can be easily implemented using bidirectional One-to-Many and Many-to-One relationships.
- For collections, try to use Lists, maps and sets. Avoid array because you don't get benefit of lazy loading.
- 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.
- Prefer DAO pattern for exposing the different methods that can be used with entity bean
- Prefer lazy fetching for associations

44. 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.

45. 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.