

Hibernate Interview Questions

- 1.What is Hibernate?**
- 2.What is ORM in Hibernate?**
- 3.What is Java Persistence API (JPA)?**
- 4.Difference Between Persist And Merge**
- 5.What are the advantages of Hibernate over JDBC?**
- 6.What are some of the important interfaces of Hibernate framework?**
- 6. What is a Session in Hibernate?**
- 7. What is a SessionFactory?**
- 8. What do you think about the statement - “session being a thread-safe object”?**
- 9. What is the difference between first level cache and second level cache?**
- 10. What can you tell about Hibernate Configuration File?**
- 11.How do you create an immutable class in hibernate?**
- 12. Can you explain the concept behind Hibernate Inheritance Mapping?**
- 13. Is hibernate prone to SQL injection attack?**
- 14.Explain hibernate mapping file**
- 15.What are the most commonly used annotations available to support hibernate mapping?**
- 16.Explain Hibernate architecture**
- 17. Can you tell the difference between getCurrentSession and openSession methods?**
- 18. Differentiate between save() and saveOrUpdate() methods in hibernate session.**
- 19. Differentiate between get() and load() in Hibernate session**
- 20.What is criteria API in hibernate?**
- 21.What is HQL?**
- 22.Can you tell something about one to many associations and how can we use them in Hibernate?**

- 23. What are Many to Many associations?**
- 24. What does session.lock() method in hibernate do?**
- 25. What is hibernate caching?**
- 26. When is merge() method of the hibernate session useful?**
- 27. Collection mapping can be done using One-to-One and Many-to-One Associations. What do you think?**
- 28. Can you tell the difference between setMaxResults() and setFetchSize() of Query?**
- 29. Does Hibernate support Native SQL Queries?**
- 30. What happens when the no-args constructor is absent in the Entity bean?**
- 31. Can we declare the Entity class final?**
- 32. What are the states of a persistent entity?**
- 33. Explain Query Cache**
- 34. Can you tell something about the N+1 SELECT problem in Hibernate?**
- 35. How to solve N+1 SELECT problem in Hibernate?**
- 36. What are the concurrency strategies available in hibernate?**
- 37. What is Single Table Strategy?**
- 38. Can you tell something about Table Per Class Strategy.**
- 39. Can you tell something about Named SQL Query**
- 40. What are the benefits of NamedQuery?**
- 41. What are Callback interfaces?**
- 42. Define HibernateTemplate?**
- 43. If you want to see the Hibernate generated SQL statements on console, what should we do?**
- 44. What does ORM consists of?**
- 45. What are the ORM level?**
- 46. List some of the databases supported by Hibernate.**

- 47. How can we add criteria to a SQL Query?**
- 48. What is the difference between merge and update?**
- 49. How to create a HQL query?**
- 50. Explain native Query ?**
- 51. What is “dirty checking”?**
- 52. What is Hibernate tuning?**
- 53. Which are the design patterns that are used in Hibernate framework?**
- 54. . How can the primary key be created by using Hibernate?**
- 55. Explain about Hibernate Proxy and how it helps in Lazy loading?**
- 56. Difference between managed associations and Hibernate associations?**

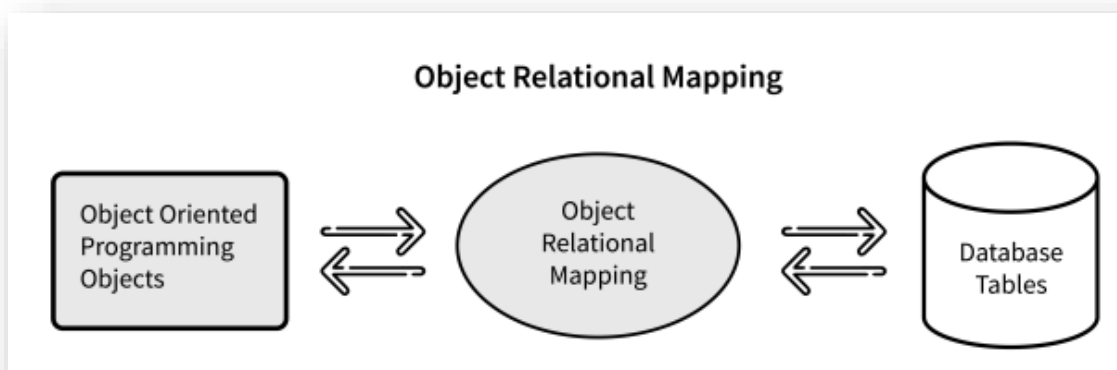
Hibernate Interview Questions And Answers

1.What is Hibernate?

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

2.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

3.What is Java Persistence API (JPA)?

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

ENTITY MANAGER FACTORY

- EntityManagerFactory provides instances of EntityManager for connecting to same database. All the instances are configured to use the same setting as defined by the default implementation. Several entity manager factories can be prepared for connecting to different data base.
- The EntityManagerFactory interface present in java.persistence package is used to provide an entity manager.

- Persistence - The Persistence is a bootstrap class which is used to obtain an EntityManagerFactory interface.

Syntax :

```
EntityManagerFactory emf=Persistence.createEntityManagerFactory("String");
```

string=persistence unit name

ENTITY MANAGER

• JPA EntityManager is used to access a database in a particular application. It is used to manage persistent entity instances, to find entities by their primary key identity, and to query over all entities.

IMPORTANT METHODS :

- persist – Make an instance managed and persistent.
- merge – Merge the state of the given entity into the current persistence context.
- remove – Remove the entity instance.
- find – Find by primary key. Search for an entity of the specified class and primary key. If the entity instance is contained in the persistence context, it is returned from there.

Syntax :

```
EntityManagerFactory emf = Persistence. createEntityManagerFactory ("String");
```

//string = persistence unit name

```
EntityManager em1 = EntityManagerFactory.createEntityManager();
```

```
EntityManager em2 = EntityManagerFactory.createEntityManager();
```

```
EntityManager em3 = EntityManagerFactory.createEntityManager();
```

NOTE : we can create multiple entity managers for one entity manager factory.

ENTITY TRANSACTION

- Interface used to control transactions on resource-local entity managers.

IMPORTANT METHODS :

- begin() method - This method is used to start the transaction.
- commit() method - This method is used to commit the transaction.

Syntax :

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("String");
```

//string = persistence unit name

```
EntityManager em1 = EntityManagerFactory.createEntityManager();
```

```
EntityTransaction et = EntityManager.getTransaction();
```

4.Difference Between Persist And Merge

persist	merge
<ul style="list-style-type: none">➤ Persist should be called only on new entities.➤ It will persist the entity object in database.➤ if you pass the object with duplicate primary key it will throw exception	<ul style="list-style-type: none">➤ merge is meant to reattach detached entities.➤ It will update the object in the database for duplicate key.➤ If the primary key is not matched it will insert the object as new record in table

5.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 implicitly 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.

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

Hibernate core interfaces are:

- Configuration
- SessionFactory
- Session
- Criteria
- Query
- Transaction

6. 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.

7. 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.

8. 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.

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.

9. 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.

10. 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.

11.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 marking mutable=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.

12. 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.

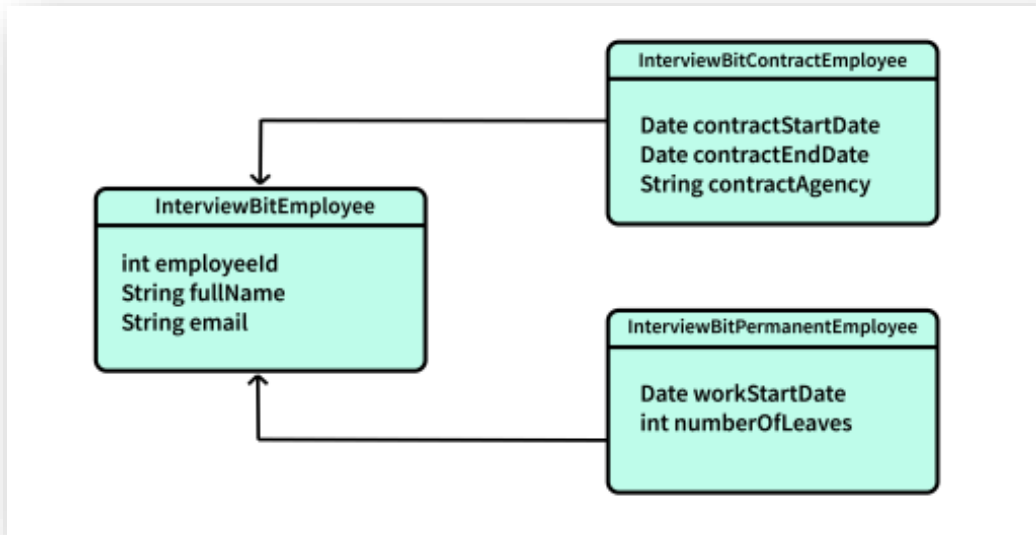
Hibernate's Inheritance Mapping strategies deal with solving how to hibernate being an ORM tries to map this problem between the inheritance of Java and flat structure of Databases.

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:



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

13. 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.

14.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>
```

15. 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;

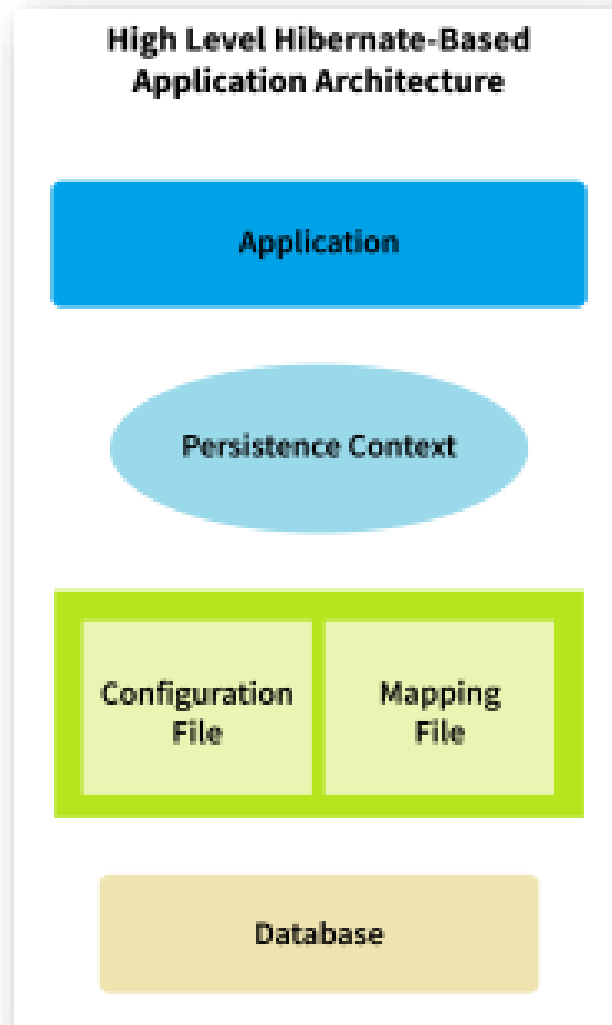
    @OneToOne(mappedBy = "employee")
    @Cascade(value = org.hibernate.annotations.CascadeType.ALL)
    private Address address;

    //getters and setters methods
}
```

16.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:

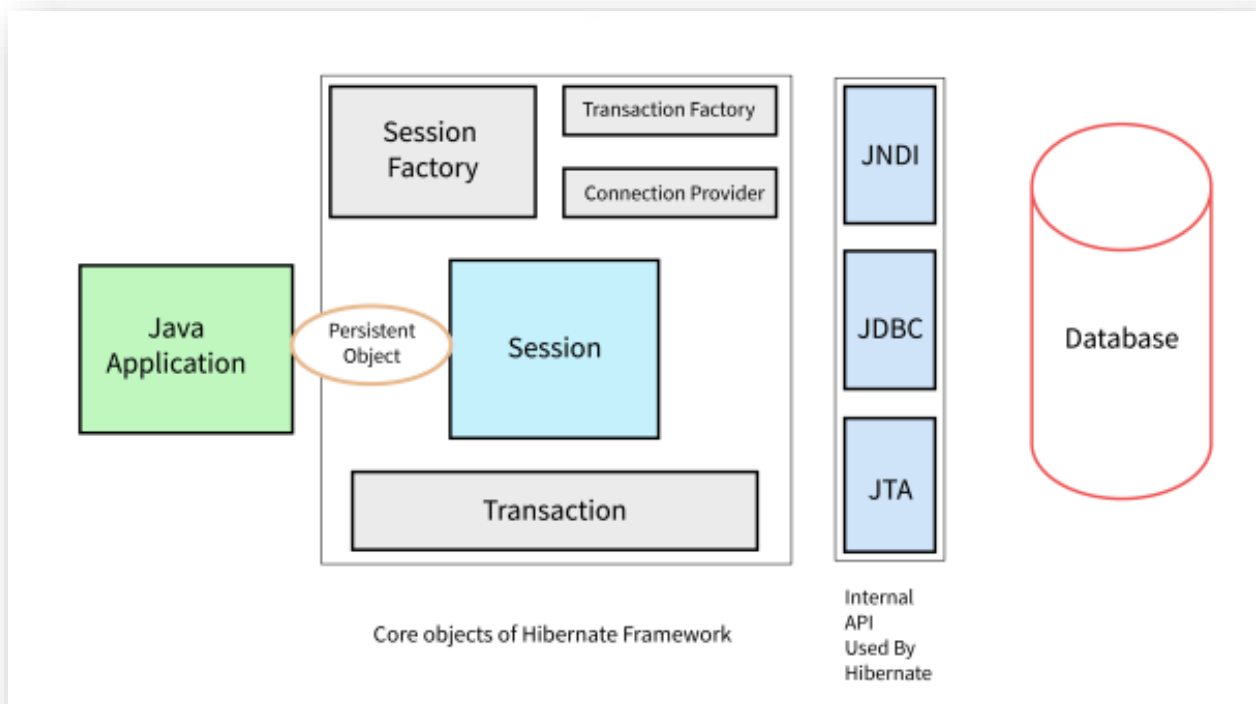
- Java Application
- Hibernate framework - Configuration and Mapping Files
- Internal API –
 - JDBC (Java Database Connectivity)
 - JTA (Java Transaction API)
 - JNDI (Java Naming Directory Interface).
- Database - MySQL, PostgreSQL, Oracle, etc



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

17. 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:

<code>getCurrentSession()</code>	<code>openSession()</code>
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 </property> . If not added, then using the method would throw an <code>HibernateException</code> .	A new session object has to be created for each request in a multi-threaded environment. Hence, you 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 <code>openSession()</code> .	In single threaded environment, it is slower than <code>getCurrentSession()</code> single threaded

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

18. 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:

<code>save()</code>	<code>saveOrUpdate()</code>
<code>save()</code> generates a new identifier and INSERT record into a database	<code>Session.saveOrUpdate()</code> 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 <code>Serializable</code> which is the newly generated identifier id value as a <code>Serializable</code> object.	The return type of the <code>saveOrUpdate()</code> 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.

19. 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 <code>ObjectNotFoundException</code> 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.

20. 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 retrieve 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 sensitiveness) 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 type Criteria 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.

21.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 rowNum): This method accepts the row number as the parameter using which the record of that row number would be retrieved.
- public Query setMaxResult(int rowCount): 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));
```

22.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() {}

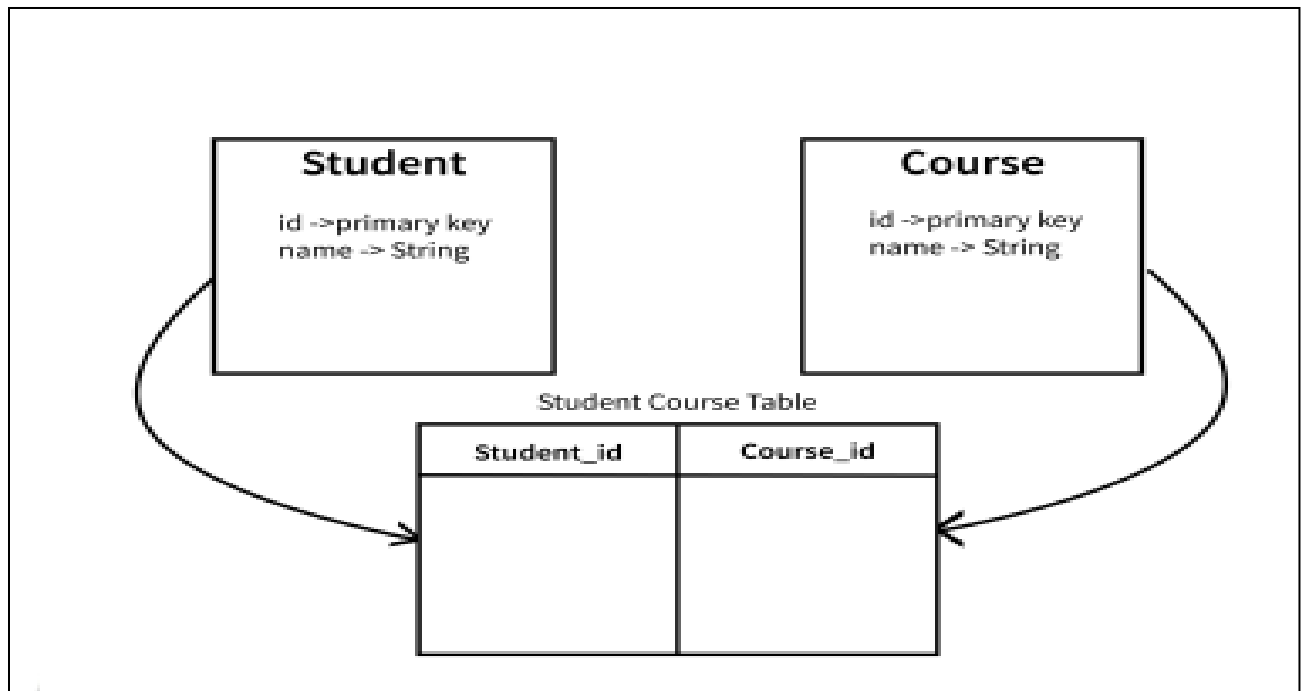
    // getters and setters
}

```

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

23. 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.

24. 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.

25. What is hibernate caching?

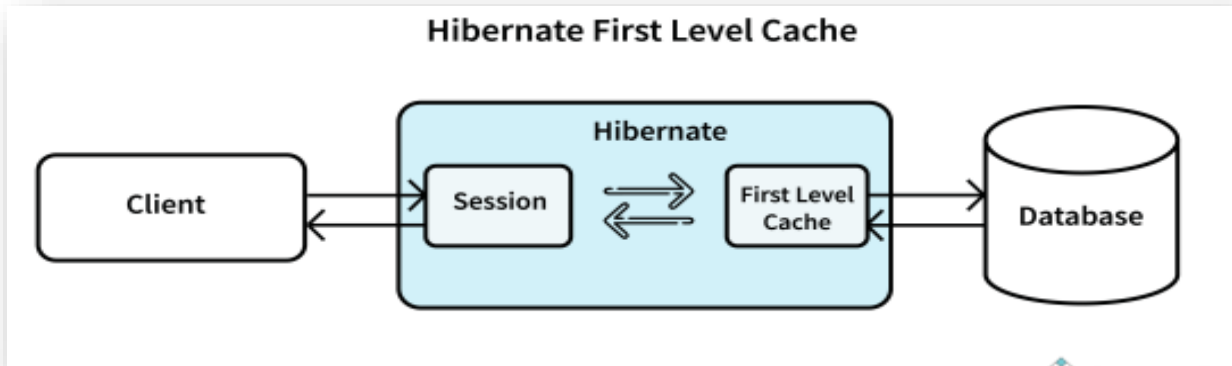
Hibernate caching is the strategy for improving the application performance by pooling objects in the cache so that the queries are executed faster. Hibernate caching is particularly useful when fetching the same data that is executed multiple times. Rather than hitting the database, we can just access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

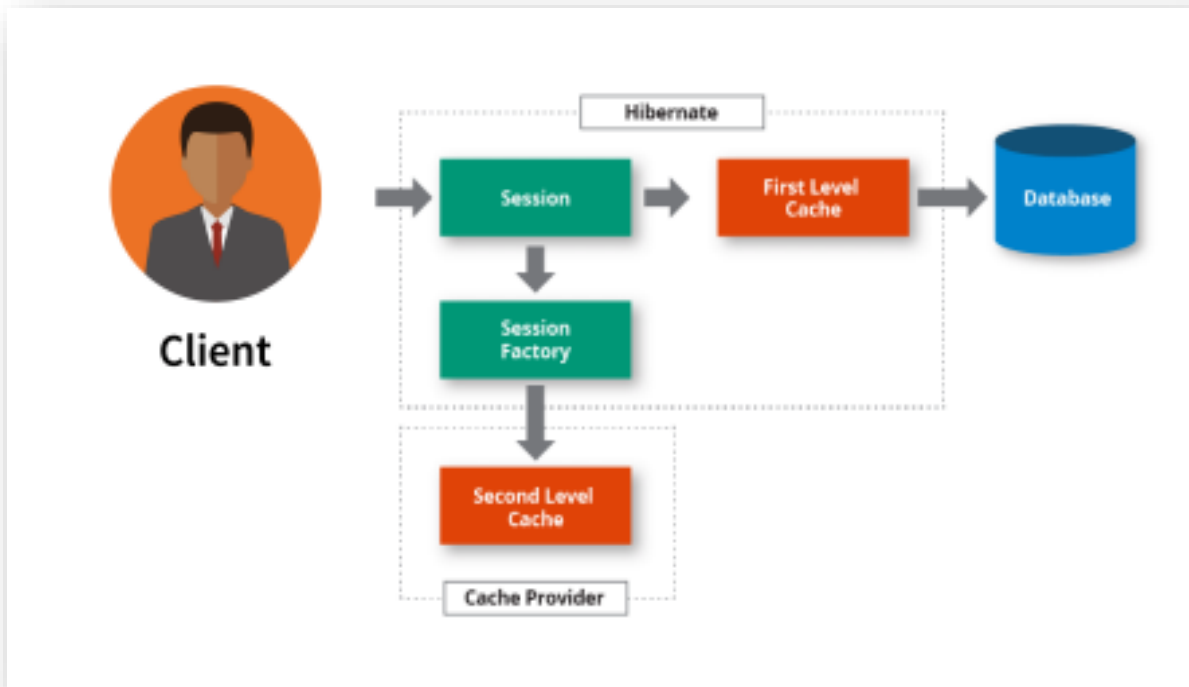
- Since it belongs to the session object, the scope of the data stored here will not be available to the entire application as an application can make use of multiple session objects.



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

26. 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.

27. 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.

28. 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.

29. 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 where it
    .addEntity(InterviewBitEmployee.class)
    .setParameter("fullName", "Hibernate"); //named parameters
List result = query.list();
```

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

30. 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`.

31. 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 can't be extended and hence lazy loading can not be supported.

32. 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 transi
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:

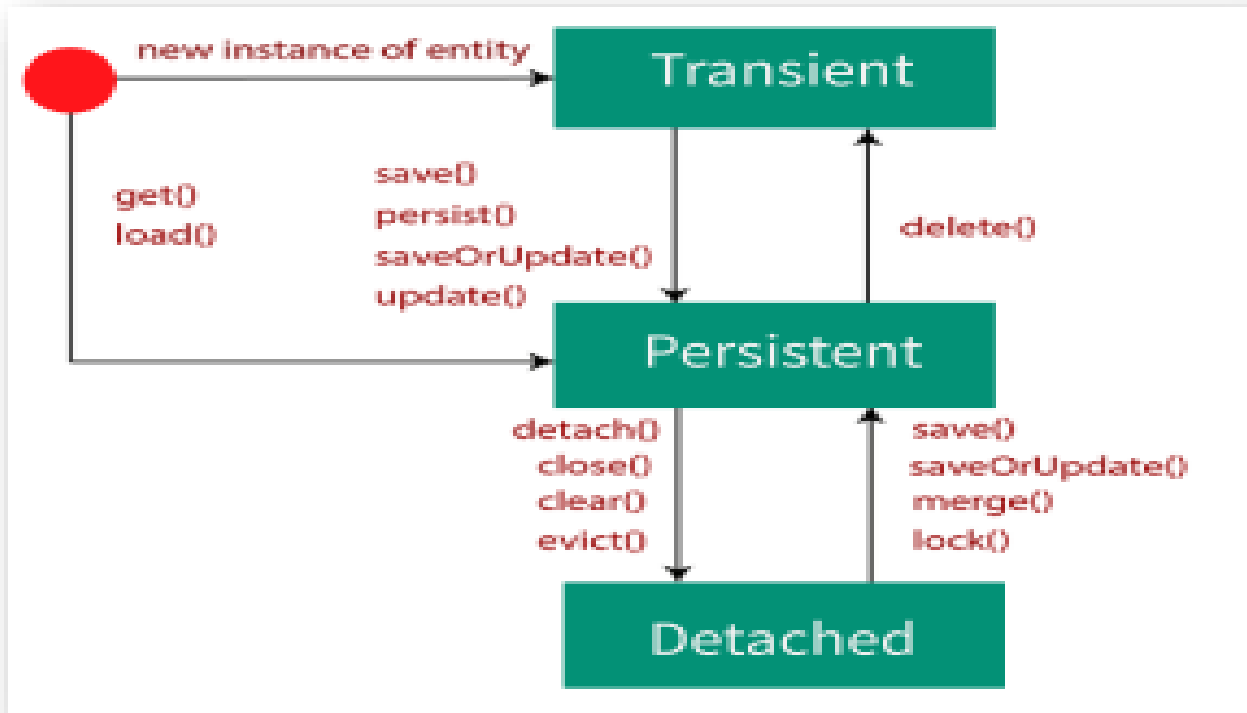
```
session.save(record);
session.persist(record);
session.update(record);
session.saveOrUpdate(record);
session.lock(record);
session.merge(record);
```

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

33. 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");
```

34.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.

35.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.

36.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.

37.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;
}
```

38.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.

39.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 @NamedQuery annotations.

- @NameQueries annotation is used for defining multiple named queries.
- @NamedQuery 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.

40.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.

41.What are Callback interfaces?

These interfaces are used in the application to receive a notification when some object events occur. Like when an object is loaded, saved or deleted. There is no need to implement callbacks in hibernate applications, but they're useful for implementing certain kinds of generic functionality.

42.Define HibernateTemplate?

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

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

In Hibernate configuration file set as follows:

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

44.What does ORM consists of?

An ORM solution consists of the following four pieces:

- API for performing basic CRUD operations
- API to express queries referring to classes

- Facilities to specify metadata
- Optimization facilities : dirty checking, lazy associations fetching

45. What are the ORM level?

The ORM levels are:

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

46. List some of the databases supported by Hibernate.

- MySQL.
- PostgreSQL.
- SQLServer.
- Sybase SQL Server.
- Oracle.
- FrontBase.
- Informix Dynamic Server.
- DB2/NT.

47. How can we add criteria to a SQL Query?

Apply criteria to a query

- Open your query in Design view.
- In the query design grid, click the Criteria row of the field where you want to add the criterion.
- Add the criteria and press ENTER. ...
- Click Run to see the results in Datasheet view.

48. What is the difference between merge and update?

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

49. How to create a HQL query?

```
String hql = "SELECT E. firstName FROM Employee E";  
Query query = session. createQuery(hql);  
List results = query. list();
```

```
String queryStr = "SELECT firstName From User";  
Query query = sessionObj.createQuery(queryStr);  
List<User> users = query.list();  
  
System.out.println(users.get(0));
```

OR

```
String queryStr = "SELECT user.firstName From User user";  
Query query = sessionObj.createQuery(queryStr);  
List<User> users = query.list();  
  
System.out.println(users.get(0));
```

50. Explain native Query ?

Software Developer | IOT developer (2015–present) · 1y. A native query is a query written in the specific database management system's native SQL language, instead of a general-purpose language like HQL or JPQL. The purpose is to take advantage of specific database functionality or to use a database-specific feature.

51. What is “dirty checking”?

Dirty Checking is a mechanism used by Hibernate to determine whether any value of an entity has changed since it was retrieved from the database. This helps Hibernate optimize database queries so that only the fields that have changed are updated.

52. What is Hibernate tuning?

Hibernate performance best practices for indexing Tables or Columns. If you're running queries against a table, the first step is to index it. These Hibernate performance tuning best practices will help improve query performance, especially if your application is fetching data from the database for display on the web.

53. 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 behaviors 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.

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

Hibernate also allows defining primary keys made up of @ManyToOne associations combined with @Id annotation. In this case, the entity class should also fulfil the conditions of a primary key class. However, the disadvantage of this method is that there's no separation between the entity object and the identifier.

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

Hibernate applies lazy loading approach on entities and associations by providing a proxy implementation of classes. Hibernate intercepts calls to an entity by substituting it with a proxy derived from an entity's class.

56. Difference between managed associations and Hibernate associations?

Managed Associations

- **Definition:** Managed associations refer to relationships between entities that are managed by an ORM framework, such as Java Persistence API (JPA). These frameworks handle the mapping between the object model and the database schema.
- **Lifecycle Management:** The ORM framework manages the lifecycle of the entities, including creation, retrieval, updating, and deletion. This includes handling the associations between entities.
- **Declarative Configuration:** Associations are often defined declaratively using annotations or XML configurations. For example, in JPA, you use annotations like @OneToMany, @ManyToOne, @OneToOne, and @ManyToMany.
- **Automatic Updates:** Changes to associations are automatically tracked and persisted by the ORM framework. If you update a collection or a relationship, the ORM ensures that the changes are reflected in the database.
- **Cascading Operations:** Managed associations support cascading operations, meaning that operations like persist, merge, remove, etc., can be propagated from one entity to its associated entities.

Hibernate Associations

- **Definition:** Hibernate associations refer to the specific implementation of entity relationships provided by the Hibernate ORM framework. Hibernate is a popular ORM framework in the Java ecosystem.
- **Specific Features:** Hibernate offers specific features and configurations for managing associations, such as @Fetch, @LazyCollection, and @Cascade annotations, which provide more fine-grained control over how associations are handled.
- **Mapping Configurations:** Hibernate allows you to define associations using annotations or XML mappings. It supports the same types of associations as JPA (@OneToMany, @ManyToOne, @OneToOne, @ManyToMany), but with additional Hibernate-specific configurations.

- **Lazy vs Eager Loading:** Hibernate provides control over lazy and eager loading of associations. Lazy loading delays the fetching of associated entities until they are actually accessed, while eager loading fetches them immediately.
- **Advanced Features:** Hibernate supports advanced features like second-level caching, custom types, and query capabilities (HQL, Criteria API). These features can be used to optimize the performance and flexibility of managing associations.