**Hibernate interview questions**

[(by mohan\_\_oon)](mailto:mohanmaali144@gmail.com)

-------------------------------------------------------------------------------------------------

**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.

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

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.

**What are some of the important interfaces of Hibernate framework?**

Hibernate core interfaces are:

Configuration

SessionFactory

Session

Criteria

Query

Transaction

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.

**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.

**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.

**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.

**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.

There are different inheritance mapping strategies available:

Single Table Strategy

Table Per Class Strategy

Mapped Super Class Strategy

Joined Table Strategy

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

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.

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

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.

**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.

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();

**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.

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

List<InterviewBitEmployee> list=query.list();

System.out.println(list.get(0));

**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 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.

**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.

**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.

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.

@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;

**}**

**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. 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.

@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;

}

**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.

@NamedQueries(

{

@NamedQuery(

name = "findIBEmployeeByFullName",

query = "from InterviewBitEmployee e where e.fullName = :fullName"

)

}

)

Usage:

TypedQuery query = session.getNamedQuery("findIBEmployeeByFullName");

query.setParameter("fullName","Hibernate");

List<InterviewBitEmployee> ibEmployees = query.getResultList();

**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.

**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

**What is HQL?**

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

**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.

Name some of the important interfaces of Hibernate framework?

Hibernate interfaces are:

• SessionFactory (org.hibernate.SessionFactory)

• Session (org.hibernate.Session)

• Transaction (org.hibernate.Transaction)

**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.

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.

**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.

**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.

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

**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.

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.

**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.

**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.

**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).

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.

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.

**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

**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.

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.

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.

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.

**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.

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.

**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.

**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

**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().

**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.

**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.

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

2

3

4 <id name="ClassID" type="string" >

<column name= "columnID" length="10" >

<generator/>

</id>

**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.

**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:

**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.