Day-24 Interview Questions

1. What is HQL, and how does it differ from SQL in Hibernate?

HQL (Hibernate Query Language) is an object-oriented query language provided by Hibernate to query and manipulate data from a relational database. It operates on Hibernate entities and their properties rather than database tables and columns. HQL is database-agnostic, and Hibernate translates HQL queries into SQL queries at runtime. It allows developers to work with domain objects directly, making it more abstract and powerful than SQL.

2. Explain the basic syntax of an HQL query?

HQL queries follow a SQL-like syntax with some differences. A basic HQL query looks like this:
String hql = "FROM EntityName WHERE condition";

3. What are the advantages of using HQL over SQL in Hibernate?

- HQL is database-agnostic, so it allows for more portable code.
- HQL operates on entities and their properties, making it more object-oriented.
- HQL supports inheritance and polymorphism.
- HQL queries are automatically translated to SQL, reducing the need for manual SQL coding.
- HQL promotes cleaner and more maintainable code.

4. What are named queries in Hibernate, and why are they useful?

Named queries in Hibernate are pre-defined HQL or SQL queries associated with an entity class and given a name. They are stored in the mapping file or annotations, making them easy to reuse. Named queries improve code readability, maintainability, and promote the separation of query logic from the application code.

5. How do you define a named query in Hibernate?

Named queries can be defined in Hibernate using either XML mapping files or annotations.
For example, in XML:
<query name="findUserById">FROM User WHERE id = :userId</query>

6. How do you execute a named query in Hibernate?

To execute a named query, you can use the getNamedQuery() method on a Session or EntityManager object, passing the query name as a parameter.

For example, in Java:

```
Query query = session.getNamedQuery("findUserById");
query.setParameter("userId", 123);
List<User> users = query.list();
```

7. Explain the different types of inheritance strategies in Hibernate?

Hibernate supports three types of inheritance strategies:

- Table Per Hierarchy (Single Table): All subclasses are mapped to a single table, and a discriminator column is used to differentiate between them.
- Table Per Subclass (Joined Subclass): Each subclass is mapped to a separate table. A join operation is used to fetch related data.
- Table Per Concrete Class (Table Per Class): Each subclass is mapped to its own table without any shared superclass table.

8.What is the default inheritance strategy in Hibernate?

By default, Hibernate uses the "Table Per Hierarchy" (Single Table) strategy. This means that all subclasses share a single table, and a discriminator column is used to determine the actual subclass of each record.

9.How do you specify an inheritance strategy for an entity in Hibernate?

In Hibernate, you can specify the inheritance strategy using the @Inheritance annotation for annotation-based configuration or the <discriminator> element in XML mapping.

For example, using annotations:

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class BaseEntity { ... }
```