Spring JPA Interview Questions and Answers

1) What is Spring Data JPA?

Spring Data JPA is part of the Spring Data project, which aims to simplify data access in Spring-based applications. It provides a layer of abstraction on top of JPA (Java Persistence API) to reduce boilerplate code and simplify database operations, allowing developers to focus more on business logic rather than database interaction details.

2) Explain features of Spring Data JPA?

Spring Data JPA offers features such as automatic repository creation, query method generation, pagination support, and support for custom queries. It provides a set of powerful CRUD methods out-of-the-box, simplifies the implementation of JPA repositories, and supports integration with other Spring projects like Spring Boot and Spring MVC.

3) How to create a custom Repository class in Spring JPA?

To create a custom repository class in Spring JPA, you can define an interface that extends the JpaRepository interface and add custom query methods. For example:

```
public interface CustomRepository<T, ID> extends JpaRepository<T, ID> {
    // Add custom query methods here
}
```

4) Difference between CRUDRepository and JPARepository.

CrudRepository provides basic CRUD operations, while JpaRepository provides JPA-specific methods like flushing changes to the database, deleting records in a batch, and more. JpaRepository extends CrudRepository, so it inherits all its methods and adds JPA-specific ones.

5) Write a custom query in Spring JPA?

We can write custom queries using the @Query annotation. For example:

```
@Query("SELECT u FROM User u WHERE u.firstName = :firstName")
List<User> findByFirstName(@Param("firstName") String firstName);
```

6) What is the purpose of save() method in CrudRepository?

The save() method in CrudRepository is used to save or update an entity. If the entity has a primary key, Spring Data JPA will determine whether to perform an insert or an update operation based on whether the entity already exists in the database.

7) What is the use of @Modifying annotation?

The @Modifying annotation is used in conjunction with query methods to indicate that the query modifies the state of the database. It is typically used with update or delete queries to inform the persistence provider that the query should be executed as a write operation, ensuring that the changes are propagated to the database.

8) Difference between findById() and getOne().

findById() returns an Optional containing the entity with the given ID, fetching it from the database immediately. getOne() returns a proxy for the entity with the given ID, allowing lazy loading of its state. If the entity is not found, getOne() throws an EntityNotFoundException.

9) Use of @Temporal annotation.

The @Temporal annotation is used to specify the type of temporal data (date, time, or timestamp) to be stored in a database column. It is typically applied to fields of type java.util.Date or java.util.Calendar to specify whether they should be treated as DATE, TIME, or TIMESTAMP.

10) Write a query method for sorting in Spring Data JPA.

We can specify sorting in query methods by adding the OrderBy keyword followed by the entity attribute and the sorting direction (ASC or DESC). For example:

List<User> findByOrderByLastNameAsc();

11) Explain @Transactional annotation in Spring.

The @Transactional annotation is used to mark a method, class, or interface as transactional. It ensures that the annotated method runs within a transaction context, allowing multiple database operations to be treated as a single atomic unit. If an exception occurs, the transaction will be rolled back, reverting all changes made within the transaction.

12) What is the difference between FetchType.Eager and FetchType.Lazy?

FetchType.Eager specifies that the related entities should be fetched eagerly along with the main entity, potentially leading to performance issues due to loading unnecessary data. FetchType.Lazy specifies that the related entities should be fetched lazily on demand, improving performance by loading them only when needed.

13) Use of @Id annotation.

The @Id annotation is used to specify the primary key of an entity. It marks a field or property as the unique identifier for the entity, allowing the persistence provider to recognize and manage entity instances.

14) How will you create a composite primary key in Spring JPA.

To create a composite primary key in Spring JPA, we can define a separate class to represent the composite key and annotate it with @Embeddable. Then, in the entity class, use @EmbeddedId to reference the composite key class.

15) What is the use of @EnableJpaRepositories method?

The @EnableJpaRepositories annotation is used to enable JPA repositories in a Spring application. It specifies the base package(s) where Spring should look for repository interfaces and configures the necessary beans to enable Spring Data JPA functionality.

16) What are the rules to follow to declare custom methods in Repository.

Custom methods in a repository interface must follow a specific naming convention to be automatically implemented by Spring Data JPA. The method name should start with a prefix such as findBy, deleteBy, or countBy, followed by the property names of the entity and optional keywords like And, Or, OrderBy, etc.

17) Explain QueryByExample in spring data jpa.

Query By Example (QBE) is a feature in Spring Data JPA that allows you to create dynamic queries based on the example entity provided. It generates a query using the non-null properties of the example entity as search criteria, making it easy to perform flexible and dynamic searches without writing custom query methods.

18) What is pagination and how to implement pagination in spring data?

Pagination is a technique used to divide large result sets into smaller, manageable chunks called pages. In Spring Data, pagination can be implemented using Pageable as a method parameter in repository query methods. Spring Data automatically handles the pagination details, allowing you to specify the page number, page size, sorting, etc.

19) Explain few CrudRepository methods.

Some commonly used methods in CrudRepository include save() to save or update entities, findById() to find entities by their primary key, deleteById() to delete entities by their primary key, findAll() to retrieve all entities, and count() to count the number of entities.

20) Difference between delete() and deleteInBatch() methods.

delete() method deletes a single entity from the database, while deleteInBatch() method deletes all entities passed as a collection in a single batch operation. The latter is more efficient for deleting multiple entities at once, as it reduces the number of database round trips.

21) You need to execute a complex query that involves multiple tables and conditional logic. How do you implement this in Spring JPA?

In Spring JPA, for complex queries involving multiple tables and conditions, I use the @Query annotation to define JPQL or native SQL queries directly on the repository methods. This allows for flexible and powerful querying capabilities beyond the standard CRUD methods provided by Spring Data JPA.

22) Your application requires the insertion of thousands of records into the database at once. How do you optimize this batch process using Spring JPA?

To optimize batch processing in Spring JPA, I enable batch inserts and updates by configuring spring.jpa.properties.hibernate.jdbc.batch_size in application.properties. This setting allows Hibernate to group SQL statements together, reducing database round trips and improving performance significantly.

23) You have entities with bidirectional relationships. How do you ensure these are correctly managed in Spring JPA to avoid common issues like infinite recursion?

In Spring JPA, when dealing with bidirectional relationships, I manage them by correctly setting up the @ManyToOne, @OneToMany, or @ManyToMany annotations with appropriate mappedBy attributes. To prevent issues like infinite recursion during serialization, I use @JsonManagedReference and @JsonBackReference annotations or DTOs to control JSON output.

24) How do you handle schema migration in a project using Spring JPA when the schema changes due to business requirements?

For schema migrations in Spring JPA projects, I integrate tools like Liquibase or Flyway. These tools are configured in Spring Boot applications to automatically apply database schema changes as part of the deployment process, ensuring the database schema is always in sync with the application's requirements.

25) You are experiencing performance issues with certain frequently accessed data. How can you implement caching in Spring JPA to improve performance?

To implement caching in Spring JPA, I use the Spring Cache abstraction with a cache provider like EHCache or Redis. I annotate frequently accessed data retrieval methods in the repository with

@Cacheable. This stores the result in the cache for subsequent requests, reducing the need to query the database repeatedly and thus improving performance.

Hibernate Most Asked Interview Questions (Optional)

Q1. What is Hibernate?

Hibernate is an open-source, lightweight, ORM (Object-Relational Mapping) tool in Java which is used to map Java classes to database tables and to convert Java data types to SQL data types.

Q2. What are the core components of Hibernate?

Core components of Hibernate include SessionFactory, Session, Transaction, ConnectionProvider, and TransactionFactory. These components are fundamental in performing database operations through Hibernate framework.

Q3. Explain the role of the SessionFactory in Hibernate.

SessionFactory is a factory class used to create Session objects. It is a heavyweight object meant to be created once per datasource or per database. It is used to open new sessions for interacting with the database.

Q4. What is a Session in Hibernate?

A Session in Hibernate is a single-threaded, short-lived object representing a conversation between the application and the database. It acts as a staging area for changes to be persisted in the database.

Q5. How does Hibernate manage transactions?

Hibernate manages transactions via its Transaction interface. Transactions in Hibernate are handled through a combination of the Java Transaction API (JTA) and JDBC. Hibernate integrates with the transaction management mechanism of the underlying platform.

Q6. What is HQL (Hibernate Query Language)?

HQL stands for Hibernate Query Language, a portable, database-independent query language defined by Hibernate. It is object-oriented, understanding notions like inheritance, polymorphism, and association.

Q7. What is the Criteria API in Hibernate?

The Criteria API is a programmable, object-oriented API in Hibernate used to define complex queries against database entities. It is used to build up a criteria query object programmatically where you can apply filtration rules and logical conditions.

Q8. Explain the concept of Object States in Hibernate.

In Hibernate, objects can exist in one of three states: transient (not associated with any session), persistent (associated with a session), and detached (was once associated with a session but then got detached).

Q9. What is the purpose of the Configuration class in Hibernate?

The Configuration class in Hibernate is used to configure settings from hibernate.cfg.xml file. It bootstraps the Hibernate and allows the application to specify properties and mapping documents to be used when creating a SessionFactory.

Q10. Describe the Second Level Cache in Hibernate.

The Second Level Cache in Hibernate is an optional cache that can store data across sessions. It is used to enhance performance by storing entities in cache memory, reducing database access.

Q11. What are the differences between get() and load() methods in Hibernate?

The get() method in Hibernate retrieves the object if it exists in the database; otherwise, it returns null. The load() method also retrieves the object, but if it doesn't exist, it throws an ObjectNotFoundException. load() can use a proxy to fetch the data lazily.

Q12. How does Hibernate ensure data integrity?

Hibernate ensures data integrity by managing database transactions, providing isolation levels, and supporting concurrency strategies. It also integrates with database constraints and can enforce application-level integrity using validators.

Q13. What is the N+1 SELECT problem in Hibernate? How can it be prevented?

The N+1 SELECT problem in Hibernate occurs when an application makes one query to retrieve N parent records and then makes N additional queries to retrieve related child objects. It can be prevented using strategies like join fetching, batch fetching, or subselect fetching to minimize the number of queries executed.

Q14. Explain the role of the @Entity annotation in Hibernate.

The @Entity annotation in Hibernate is used to mark a class as an entity, which means it is a mapped object and its instance can be persisted to the database.

Q15. What is cascading in Hibernate?

Cascading in Hibernate is the ability to propagate the operations from a parent entity to its associated child entities. It is used to manage the state transitions of associated objects automatically. CascadeType can be used to specify which operations are cascaded.

Q16. What is a Composite Key in Hibernate?

A Composite Key in Hibernate is a primary key made up of multiple columns. In Hibernate, a composite key can be represented using a separate class annotated with @Embeddable or @EmbeddedId to represent this composite key.

Q17. How does Hibernate handle SQL Injection?

Hibernate handles SQL Injection by using prepared statements that automatically escape SQL syntax. Additionally, using HQL or Criteria API also protects against SQL injection as they translate a query from HQL into SQL in a way that uses parameterized queries.

Q18. What is Lazy Loading in Hibernate?

Lazy Loading in Hibernate is a concept where an entity or collection of entities is not loaded until it is accessed for the first time. This is a performance optimization technique to defer the loading of objects until they are needed.

Q19. How can you achieve concurrency in Hibernate?

Concurrency in Hibernate can be achieved using versioning and locking mechanisms. Hibernate supports optimistic and pessimistic locking strategies to handle concurrent modifications of data effectively.

Q20. What is an optimistic locking in Hibernate?

Optimistic locking in Hibernate is a technique to ensure that a record is not updated by more than one transaction at the same time by using a version field in the database table. It checks the version of a record at the time of fetching and before committing an update to ensure consistency.

Q21. You have noticed that your Hibernate application is running slowly when fetching data from a database with many relationships. What strategy could you use to improve performance?

To optimize query performance in Hibernate, I would consider using lazy loading for entity relationships. This means Hibernate will only fetch related entities when they are explicitly accessed, not at the time of fetching the parent entity. Additionally, I might use batch fetching and adjust the fetch sizes in the configuration to reduce the number of database queries.

Q22. How do you handle a Hibernate session in a web application to ensure that it is properly closed, avoiding memory leaks?

In our web application, we manage Hibernate sessions by binding a session to the current thread using the CurrentSessionContext interface. We typically configure session opening and closing in a servlet filter or interceptors, ensuring that each request opens a session and ends by closing the session, thus preventing memory leaks.

Q23. During a transaction, an error occurs after several database operations have been successfully executed. How does Hibernate ensure data integrity in this situation?

Hibernate ensures data integrity by using transactions. If an error occurs during the transaction, hibernate rolls back all operations to the state before the transaction began, using either database transactions or the Java Transaction API (JTA). This rollback mechanism prevents partial data modifications that could lead to data inconsistency.

Q24. You need to add auditing features to track changes in entity data. What Hibernate feature would you use to achieve this?

To implement auditing in Hibernate, I would use Hibernate Envers. It's a Hibernate module that allows for versioning of entity classes. By simply annotating our entity classes with @Audited, we can keep track of changes to their state, automatically storing revisions in separate tables.

Q25. You are working with a legacy database where the table and column names do not follow your standard naming conventions. How can you map these tables without modifying the existing database schema?

In Hibernate, I handle legacy databases by customizing the ORM mapping. I use the @Table and @Column annotations to map entity classes to the specific table names and column names defined in the legacy database. This allows us to map the entities accurately to the database schema without any changes to the database itself.