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.