#### **8. How can you integrate Spring Data JPA with a Spring Boot application?**

To integrate Spring Data JPA with a Spring Boot application, you need to follow these steps:

1. Add the Spring Data JPA and database driver dependencies in your Maven or Gradle build file.  
2. Configure the data source properties in the application.properties or application.yml file. This includes the URL, username, password, and Hibernate dialect for your specific database.  
3. Create an entity class that represents a table in your database. Use annotations like @Entity, @Id, etc., to map the class and its fields to the corresponding table and columns.  
4. Create a repository interface for each entity class. Extend it from JpaRepository or CrudRepository which provides basic CRUD operations.  
5. Autowire the repository in your service class to use it.

#### **11. How can you handle transactions in Spring Data JPA?**

Spring Data JPA handles transactions using the @Transactional annotation. This annotation can be applied at both class and method levels, providing flexibility in transaction management. When placed on a class, it applies to all methods within that class. If used on a method, it only affects that specific method.

The @Transactional annotation has several properties like propagation behavior, isolation level, timeout settings, read-only status, and rollback rules for exceptions. The default propagation setting is REQUIRED which means if a transactional method is executed inside a transaction context, it joins that transaction. If not, a new one starts.

Isolation level defines how data is isolated from other transactions during execution. Default is DEFAULT, meaning it uses the underlying datastore’s default isolation level. Timeout specifies the time after which the transaction must roll back. Read-only marks the transaction as read-only, optimizing resource usage. Rollback rules define when to roll back a transaction based on the exceptions thrown.

Or

#### **25. Can you describe how transactions work in Spring Data JPA and how they can be managed?**

Spring Data JPA manages transactions through the @Transactional annotation. This annotation can be applied at both class and method levels, providing flexibility in transaction scope. When a method annotated with @Transactional is invoked, Spring Data JPA starts a new transaction. If an existing transaction is already running, it will join that transaction.

The propagation behavior of transactions can be managed using the propagation attribute of the @Transactional annotation. For instance, Propagation.REQUIRED denotes that a current transaction must exist, otherwise a new one will be created. Conversely, Propagation.NEVER indicates that a transaction should not exist, executing outside any transaction context.

Rollbacks are automatically performed when runtime exceptions occur within the transactional method. However, this behavior can be customized by specifying the rollbackFor and noRollbackFor attributes of the @Transactional annotation.

#### **14. What is the difference between JPA and Hibernate?**

JPA (Java Persistence API) is a specification for object-relational mapping and data persistence. It’s not an implementation but provides guidelines that any ORM tool can implement. Hibernate, on the other hand, is an actual implementation of JPA. It was developed before JPA and has additional features beyond the standard scope of JPA. These include Criteria Queries, caching, and lazy loading. While JPA provides portability across different implementations, Hibernate offers more advanced functionalities at the cost of being vendor-specific.

#### **12. How do you map relationships between entities in JPA and what are the different types of relationships?**

In JPA, relationships between entities are mapped using annotations on fields representing the relationship. The four types of relationships are: OneToOne, ManyToOne, OneToMany, and ManyToMany.

OneToOne is used when one entity instance is related to exactly one other entity instance. For example, a User has one Profile.

ManyToOne is used when multiple instances of an entity are associated with a single instance of another entity. For example, many Orders can be linked to one Customer.

OneToMany is the inverse of ManyToOne, where one entity instance is associated with multiple instances of another entity. For example, one Author can write many Books.

ManyToMany represents a complex relationship where multiple instances of an entity are associated with multiple instances of another entity. For example, a Student can enroll in many Courses, and each Course can have many Students.

#### **15. How would you handle lazy and eager loading in Spring Data JPA?**

In Spring Data JPA, handling lazy and eager loading involves setting fetch type in entity relationships. By default, @OneToOne and @ManyToOne associations are eagerly loaded while @OneToMany and @ManyToMany are lazily loaded.

For eager loading, FetchType.EAGER is used which loads all related entities immediately. This can lead to performance issues if there are many related entities. For example:

@ManyToOne(fetch = FetchType.EAGER)

private User user;

Lazy loading, FetchType.LAZY, only loads related entities when they’re accessed for the first time, improving performance. However, it may cause exceptions if session is closed before accessing them. Example:

@OneToMany(fetch = FetchType.LAZY)

private List<Order> orders;

To handle these issues, use Open Session in View pattern or manually initialize collections.

#### **17. How can you handle exceptions in Spring Data JPA?**

In Spring Data JPA, exceptions are handled using the @ExceptionHandler annotation or by implementing a ControllerAdvice class. The @ExceptionHandler is used within a controller and handles exceptions thrown by request handling methods. It takes as argument the exception thrown which can be user-defined or one of many provided by Spring.

The ControllerAdvice class allows for global exception handling across all controllers. This class is annotated with @ControllerAdvice and contains methods annotated with @ExceptionHandler. Each method should correspond to an exception that needs to be handled.

For database related exceptions, Spring provides the DataAccessException hierarchy. These exceptions wrap the specific details of data access issues from different databases.

#### **18. What is the difference between CrudRepository and JpaRepository interfaces in Spring Data JPA?**

CrudRepository and JpaRepository are two interfaces provided by Spring Data JPA. CrudRepository provides basic CRUD operations, while JpaRepository extends PagingAndSortingRepository which in turn extends CrudRepository, thus inheriting its functionalities.

The key difference lies in the additional methods that JpaRepository offers over CrudRepository. These include flushing the persistence context to the database, deleting records in a batch, and other functionality related to JPA such as entity graph handling.

JpaRepository also integrates with Hibernate to provide some specific features like automatic flushes during queries, which can be beneficial for certain applications. However, if these extra features aren’t required, using CrudRepository may lead to cleaner, simpler code.