**Custom repository implementation pattern** in Spring Data JPA

@Repository  
public interface CustomServiceRuleRepository {

}

Vs

public interface CustomServiceRuleRepository extends CrudRepository {

}

In Spring Data JPA, there is a key difference between declaring a repository with just @Repository versus extending an interface like CrudRepository, JpaRepository, or another specific repository interface.

**1. Using @Repository without Extending CrudRepository or JpaRepository**

When you simply annotate an interface with @Repository, like this:

@Repository

public interface CustomServiceRuleRepository {

}

It behaves as a **marker interface** and does not inherit any predefined methods for interacting with the database. This means:

* **No built-in CRUD operations**: You don't have access to methods like save(), findById(), delete(), or findAll().
* **Custom implementation required**: You need to manually implement this repository by providing a custom repository class (usually using the EntityManager) if you want to define specific behavior.
* **Use Case**: This is often used for defining **custom behavior** beyond standard CRUD operations, or if you want to create a repository that works alongside another Spring Data JPA repository.

To provide custom functionality, you'd typically implement a class, such as:

@Repository

public class CustomServiceRuleRepositoryImpl implements CustomServiceRuleRepository {

@Autowired

private EntityManager entityManager;

// Custom methods using entityManager

public void customQueryMethod() {

// custom logic

}

}

This is part of the **custom repository implementation pattern** in Spring Data JPA.

**2. Extending CrudRepository**

When you extend CrudRepository, like this:

@Repository

public interface CustomServiceRuleRepository extends CrudRepository<ServiceRule, Long> {

}

You inherit all the basic **CRUD operations** (Create, Read, Update, Delete) out of the box. Specifically, CrudRepository provides:

* save(S entity) – Save an entity.
* findById(ID id) – Find an entity by its ID.
* findAll() – Find all entities.
* deleteById(ID id) – Delete an entity by its ID.
* count() – Return the number of entities.

**Key Points**:

* **No custom implementation required**: The repository is automatically managed by Spring Data JPA, so you don't need to write manual SQL or query logic for basic CRUD operations.
* **Custom queries still possible**: You can define additional custom query methods using @Query annotation or Spring Data JPA query derivation.
* **Use Case**: Ideal for **basic CRUD operations** where the repository needs to perform standard tasks like saving, updating, or deleting records.

**3. Extending JpaRepository**

Alternatively, if you need more features like pagination and sorting, you can extend JpaRepository, which is a more powerful extension of CrudRepository.

@Repository

public interface CustomServiceRuleRepository extends JpaRepository<ServiceRule, Long> {

}

* **JpaRepository provides**:
  + Everything from CrudRepository, plus:
  + findAll(Pageable pageable) – Pagination support.
  + findAll(Sort sort) – Sorting support.
  + flush() – Flush the persistence context.
* **Use Case**: When you need not just CRUD operations, but also **pagination and sorting** out of the box.

**When to Use Which:**

* **@Repository without extending CrudRepository**:
  + When you need to **fully customize** the repository behavior.
  + For example, when you want to manually define every method using EntityManager or custom SQL queries.
* **Extending CrudRepository**:
  + When you need **basic CRUD operations** and don't need pagination or sorting.
  + It's simple and lightweight but still gives you the flexibility to add custom queries if needed.
* **Extending JpaRepository**:
  + When you need **advanced features** like pagination and sorting in addition to the CRUD methods.
  + It's useful for **larger datasets** or cases where sorting and paginating results is important.

**Summary:**

| **Repository Type** | **Use Case** |
| --- | --- |
| **@Repository only** | For full control and when custom implementations are needed (you provide custom methods and logic manually). |
| **Extending CrudRepository** | For standard **CRUD operations** with minimal setup, used when basic data access is needed without extra features like pagination. |
| **Extending JpaRepository** | For more **advanced features** like pagination, sorting, and batch operations, making it suitable for more complex data access requirements. |

In most cases for new tech projects, extending JpaRepository is preferred as it provides the most flexibility and functionality. However, if you only need basic CRUD without any extras, CrudRepository is simpler. If you need very specific behavior, you might implement a custom repository with just @Repository.