### Spring Data JPA

**Spring Data JPA** provides repository support for Java Persistence API (JPA).

It eases development of application that needs to access JPA data source.

Spring Data JPA is not JPA provider.

It is a library/framework that adds extra layer of abstraction on the top of JPA provider.

**The Goal of Spring Data Repository abstraction** **is to significantly reduce the amount of boiler code required to implement data access layer for various persistence stores**.

* Spring Data JPA provides built in methods for performing basic CRUD operations and you don’t need to write any query method.
* Spring Data JPA provides JPA specific repository.
* Instead of writing boilerplate code for a generic DAO class (as we would normally do with Hibernate/JPA without Spring Data JPA), we just declare a simple Spring Data Repository interface

**Repository**

* It is the central interface in the spring data repository abstraction. This is a marker interface.
* **If you are extending this interface**, **you have to declare your own methods and the implementations will be provided by the spring run-time.**

**CrudRepository**

* CrudRepository provides methods for the CRUD operations. you don’t need to write any query method.
* CrudRepository is an interface and extends Spring data Repository interface.
* To use CrudRepository we have to create our interface and extend CrudRepository. **We need not to implement our interface, its implementation will be created automatically at runtime**.
* When you define CrudRepository, you have to pass the two parameters: type of the entity and type of the entity’s id field.

CrudRepository<T, ID extends Serializable>

**This interface has the following methods:**

<S extends T> S save(S entity): Saves and updates the current entity and returns that entity.  
Optional<T> findById(ID primaryKey): Returns the entity for the given id.  
Iterable<T> findAll(): Returns all entities.  
long count(): Returns the count.  
void delete(T entity): Deletes the given entity.  
boolean existsById(ID primaryKey): Checks if the entity for the given id exists or not.

**PagingAndSortingRepository**

* This is extension of CrudRepository. It is specialized version for the **paging operations**.
* It provide **additional methods to retrieve entities** **using the pagination and sorting abstraction**.

**JpaRepository**

* JpaRepository is interface that extends from PagingAndSortingRepository interface.
* JpaRepository **provides methods that are specific to JPA**.

**This interface has the following methods:**

List<T> findAll()

<S extends T> List<S> findAll(Example<S> example)

<S extends T> List<S> findAll(Example<S> example, Sort sort)

List<T> findAll(Sort sort)

List<T> findAllById(Iterable<ID> ids)

T getOne(ID id) Returns a reference to the entity with the given identifier.

**Difference between CrudRepository, JpaRepository**

When we look at both of the interfaces, the clear differences are:

* **CrudRepository** is part of Spring Data Commons project and declared under the package org.springframework.data.repository.

Whereas **JpaRepository** is part of **JPA data** **store specific implementation** and declared under the package org.springframework.data.jpa.repository.

* **CrudRepository** extends from Repository interface.

Whereas **JpaRepository** extends from PagingAndSortingRepository which in turn extends from the CrudRepository.

* **JpaRepository** returns List type of entities and **CrudRepository** returns Iterable type of entities.

## **Query Methods or Custom Repository Methods**

Using query methods are most convenient way to write our own queries to fetch data from the database. Spring data JPA allows more powerful query formation using the methods names and parameter.

CrudRepository provides methods for generic CRUD operation and if we want to add custom methods in our interface that has extended CrudRepository, we can add in following ways.  
  
a. We can start our query method names with find...By, read...By, query...By, count...By, and get...By. Before By we can add expression such as Distinct . After By we need to add property names of our entity.  
b. To get data on the basis of more than one property we can concatenate property names using And and Or while creating method names.  
c. If we want to use completely custom name for our method, we can use @Query annotation to write query.  
  
**Find the code snippet that is using** the sample method name for the above scenarios.

public interface ArticleRepository extends CrudRepository<Article, Long> {

List<Article> findByTitle(String title);

List<Article> findDistinctByCategory(String category);

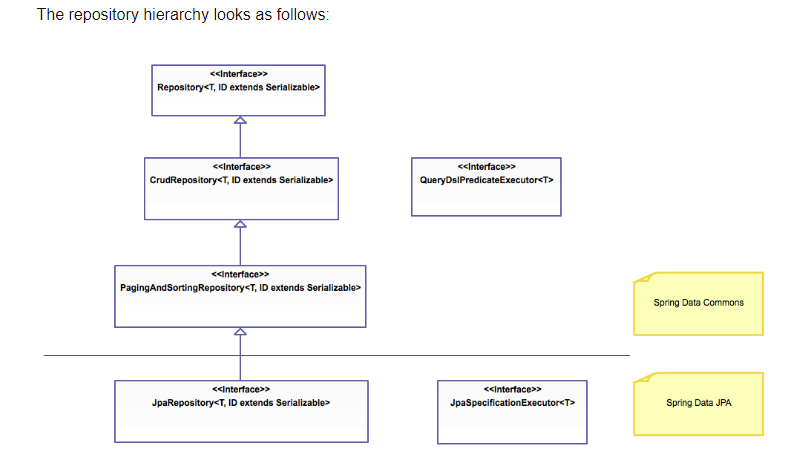
List<Article> findByTitleAndCategory(String title, String category);

@Query("SELECT a FROM Article a WHERE a.title=:title and a.category=:category")

List<Article> fetchArticles(@Param("title") String title, @Param("category") String category);

}

The implementation class of the above methods will be created by Spring at runtime automatically.



**Configure JPA repository**

To setup spring to create proxy instance for those interfaces either with Java Config or XML Configuration.

When a repository class is found, Spring will generate an appropriate proxy class at runtime to provide implementation details. So, the @EnableJpaRepositories annotation is required to enable Spring Data JPA in a Spring application.

|  |  |
| --- | --- |
| **Java Configuration**  @Configuration  @EnableJpaRepositories  Class AppConfig { } | **XML Configuration**  <jpa:repositories base-package=”com.repository”> |