Enlace de interés https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html

En Spring Data JPA, tanto JpaRepository como CrudRepository son interfaces que proporcionan métodos para realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre las entidades de la base de datos. Sin embargo, hay diferencias clave entre ellas en cuanto a la funcionalidad que ofrecen:

1. **CrudRepository**

CrudRepository es la interfaz más básica que ofrece Spring Data JPA para operaciones CRUD. Incluye métodos fundamentales para trabajar con la persistencia de datos.

Métodos principales que proporciona CrudRepository:

- save(S entity): Guarda o actualiza una entidad.
- findById(ID id): Encuentra una entidad por su ID.
- findAll(): Devuelve todas las entidades.
- deleteById(ID id): Elimina una entidad por su ID.
- deleteAll(): Elimina todas las entidades.
- count(): Devuelve el número total de entidades.
- existsById(ID id): Verifica si una entidad existe por su ID.

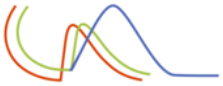
Este repositorio es ideal cuando solo necesitas las operaciones básicas de CRUD.

2. **2. JpaRepository**

JpaRepository extiende de CrudRepository y agrega más funcionalidad específica de JPA, además de proporcionar más métodos útiles para operaciones comunes en bases de datos. Si trabajas con JPA y necesitas características adicionales, JpaRepository es la opción adecuada.

Métodos adicionales que proporciona JpaRepository (además de los que hereda de CrudRepository):

- **Paginación y ordenación**:
    - findAll(Pageable pageable): Devuelve una lista de entidades paginadas.
    - findAll(Sort sort): Devuelve todas las entidades con un orden específico.



- **Operaciones masivas**:
    - saveAll(Iterable<S> entities): Guarda una lista de entidades.
    - deleteInBatch(Iterable<T> entities): Elimina un grupo de entidades en un solo lote.
    - deleteAllInBatch(): Elimina todas las entidades en un solo lote.
- **Sincronización y referencias**:
    - flush(): Sincroniza el estado de persistencia con la base de datos.
    - saveAndFlush(S entity): Guarda una entidad y luego realiza un flush.
    - getOne(ID id): Obtiene una referencia a una entidad sin cargarla completamente desde la base de datos.

3. **Diferencias clave:**
4. **Funcionalidad adicional en JpaRepository**: JpaRepository proporciona características avanzadas como paginación, ordenación, operaciones por lotes y sincronización de la sesión, lo que lo hace más adecuado para aplicaciones complejas que necesitan más que simples operaciones CRUD.
5. **Alcance de uso**: Si solo necesitas las operaciones básicas (CRUD), CrudRepository es suficiente. Sin embargo, si necesitas características más avanzadas como la paginación, ordenación y manejo masivo de datos, debes usar JpaRepository.
6. **Resumen:**

- **Usa CrudRepository** si solo necesitas las operaciones CRUD básicas.
- **Usa JpaRepository** si necesitas funcionalidades adicionales como paginación, ordenación y operaciones por lotes, o si prefieres trabajar con el modelo completo de JPA.

1. Métodos de Consultas en JPA

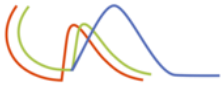
Estrategias de Búsqueda de Consultas

El módulo JPA de Spring Data admite la definición de consultas manualmente como una cadena de texto, o permite derivarlas del nombre del método.

Las consultas derivadas con los predicados como: **IsStartingWith**, **StartingWith**, **StartsWith**, **IsEndingWith**, **EndingWith**, **EndsWith**, **IsNotContaining**, **NotContaining**, **NotContains**, **IsContaining**, **Containing**, **Contains**, entre otros, sanitizan los argumentos. Esto significa que si los argumentos contienen caracteres reconocidos por LIKE como comodines, serán escapados para que coincidan solo como literales. El carácter de escape usado puede configurarse al establecer la propiedad escapeCharacter en la anotación @EnableJpaRepositories. Compara con el uso de **expresiones SpEL**.

Consultas Declaradas

Aunque obtener una consulta derivada del nombre del método es conveniente, en algunas situaciones el analizador de nombres de métodos puede no admitir la palabra clave que se desea usar, o el nombre del método se volvería innecesariamente largo.



En estos casos, se pueden usar **consultas nombradas de JPA** a través de una convención de nomenclatura (ver **Using JPA Named Queries** para más información) o anotar el método con @Query (ver **Using @Query** para detalles).

Creación de Consultas

Generalmente, el mecanismo de creación de consultas para JPA funciona como se describe en **Métodos de Consulta**. El siguiente ejemplo muestra a lo que se traduce un método de consulta JPA:

Ejemplo 1. Creación de consultas a partir de nombres de métodos

java

```
public interface UserRepository extends Repository<User, Long> {

 List<User> findByEmailAddressAndLastname(String emailAddress, String lastname);

}
```

Aquí, creamos una consulta usando la API de criterios de JPA, pero esencialmente, esto se traduce en la siguiente consulta:

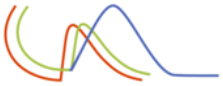
```
select u from User u where u.emailAddress = ?1 and u.lastname = ?2
```

Spring Data JPA realiza una verificación de propiedades y recorre propiedades anidadas, como se describe en **Expresiones de Propiedades**.

La siguiente tabla describe las palabras clave admitidas por JPA y lo que traduce un método que contiene esas palabras clave:

Palabra clave	**Ejemplo**	**Fragmento JPQL**
Distinct	findDistinctByLastnameAndFirstname	select distinct … where x.lastname = ?1 and x.firstname = ?2
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Is, Equals	findByFirstname, findByFirstnameIs, findByFirstnameEquals	… where x.firstname = ?1
Between	findByStartDateBetween	… where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… where x.age >= ?1

... (continúa con las otras palabras clave) ...



DISTINCT puede ser engañoso y no siempre produce los resultados esperados. Por ejemplo, select distinct u from User u producirá un resultado completamente diferente a select distinct u.lastname from User u. El primer caso incluye User.id, por lo que no habrá duplicados. El segundo ejemplo encontrará solo apellidos únicos.

Configuración Basada en Anotaciones

La configuración basada en anotaciones tiene la ventaja de no requerir otro archivo de configuración, reduciendo el esfuerzo de mantenimiento. Sin embargo, cada nueva declaración de consulta requiere recompilar la clase de dominio.

Ejemplo 2. Configuración de consulta nombrada basada en anotaciones

```
@Entity

@NamedQuery(name = "User.findByEmailAddress",

  query = "select u from User u where u.emailAddress = ?1")

public class User {

}
```

Usando Consultas Nativas

La anotación @Query permite ejecutar consultas nativas al establecer la propiedad nativeQuery en true.

Ejemplo 10. Declarar una consulta nativa en el método de consulta usando @Query

```
public interface UserRepository extends JpaRepository<User, Long> {

  @Query(value = "SELECT * FROM USERS WHERE EMAIL_ADDRESS = ?1", nativeQuery = true)

  User findByEmailAddress(String emailAddress);

}
```

Spring Data JPA no admite actualmente la ordenación dinámica para consultas nativas, porque no puede manipular de forma confiable la consulta SQL real. Sin embargo, se pueden usar consultas nativas para la paginación especificando la consulta de conteo por uno mismo.