

## Módulo XI

# SPRING DATA JPA III - QUERY METHODS



**MASTER DE SPRING FRAMEWORK Y  
SPRING BOOT**

Impartido por Rafael Álvarez Martínez

18 Abril 2022

# CONTENIDOS

- ¿Qué son los query methods?
- Consultas Declaradas @QUERY.
- JOIN
- Consultas Derivadas.

# ¿QUÉ SON LOS QUERY METHODS?

- Son métodos que permiten obtener información (SELECT) desde la BD.
- Los métodos SÓLO se declaran en la interfaz del repositorio y Spring Data JPA realiza su implementación.
- Existe dos tipos de consultas:
  - Declaradas: el método lleva la anotación @Query que permiten indicar la consulta SQL de manera explícita.
  - Derivadas: no requieren ninguna anotación ya que el nombre del método hace referencia a la consulta a realizar.

# ¿QUÉ SON LOS QUERY METHODS?

- **Declarada:**

- La consulta SQL se debe incluir dentro de la anotación @Query
- **:arg** pasa argumentos desde el método a la consulta.

```
public interface PersonaRepository extends JpaRepository<Persona, Integer> {  
  
    @Query(" select p from com.aepi.entity.Persona p where p.nombre=:nombre")  
    public List<Persona> getPersonasPorNombre(String nombre);
```

- **?pos** pasa el argumento desde el método según la posición.

```
// Parametros posicionales (?1,?2,?3.....)  
// Al pasar los datos de esta manera no hacen falta los dos puntos  
@Query(" select p from com.aepi.entity.Persona p where p.id=?1 or p.salario>?2 ")  
public List<Persona> getDatosPosicionales(Integer id, Integer salario);
```



# ¿QUÉ SON LOS QUERY METHODS?

- **Derivada:**

- El nombre del método debe tener una palabra clave seguida del nombre del atributo de la clase y del parámetro del método.
- Por ejemplo, para recuperar todas las **Vacantes** que tenga un **estatus** concreto.

```
public interface VacantesRepository extends JpaRepository<Vacante, Integer> {  
  
    // select * from Vacantes where estatus = ?  
    List<Vacante> findByEstatus(String estatus);  
  
}
```

KEYWORD

ATRIBUTO DE LA CLASE

PARÁMETRO

# CONSULTAS DECLARADAS - @QUERY

- **Ejemplo 1:** EJEMPLOS/EjemploQuerysMethods

- Para obtener de la tabla personas todas las personas que hay.

```
@Entity
@Table(name = "personas")
public class Persona {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "nombre")
    private String nombre;

    @Column(name = "apellidos")
    private String apellidos;

    @Column(name = "edad")
    private Integer edad;

    @Column(name = "activo")
    private Boolean activo;

    @Column(name = "salario")
    private Integer salario;
```

```
@Query(" select p from com.aepi.entity.Persona p")
public List<Persona> getAllPersonas();
```



# CONSULTAS DECLARADAS - @QUERY

- **Ejemplo 1:** EJEMPLOS/EjemploQuerysMethods
  - Si el método tiene varios argumentos se puede indicar el orden en la consulta SQL mediante la posición que ocupan en la signature del método.

```
// Parametros posicionales (?1,?2,?3.....)  
// Al pasar los datos de esta manera no hacen falta los dos puntos  
@Query(" select p from com.aepi.entity.Persona p where p.id=?1 or p.salario>?2 ")  
public List<Persona> getDatosPosicionales(Integer id, Integer salario);
```



# CONSULTAS DECLARADAS - @QUERY

- **Tarea 1:** TAREAS/TareaUno

- Crea un proyecto con las dependencias de Spring Web, Thymeleaf, Validation, MySql Driver, Spring Boot Dev Tools, Spring Data JPA.
- Crea la BD **libros**.
- Crea la clase **Libro** (Integer id, String titulo, String autor, int paginas).
  - Añade las anotaciones para la persistencia como Entity.



# CONSULTAS DECLARADAS - @QUERY

- **Tarea 1:** TAREAS/TareaUno
  - Crea la interfaz **ILibrosRepo** que extienda a JpaRepository con las consultas declaradas para **obtener** los libros:
    - 1 que tengan el **id** que se pasa.
    - 2 ordenados ascendentemente por **título**.
    - 3 ordenados descentemente por **página**.
    - 4 con un número de **páginas** mayor a Z.
    - 5 que su **autor** sea igual a X.
    - 6 que su **autor** sea X y tengan **más** de Z páginas.



# CONSULTAS DECLARADAS - @QUERY

- **Tarea 1:** TAREAS/TareaUno
  - Crea el controlador **LibroController** con los endpoints:
    - **/index** muestra la vista **index.html**
      - Un botón para **Nuevo** libro que muestra el formulario.
      - Un botón **Filtrar** para mostrar un formulario que ejecuta la consulta elegida en un SELECT con una opción por cada una de las consultas declaradas en el repositorio.
      - Una tabla de BootStrap que muestra los **libros** de la BD con un botón de **Eliminar, Editar**.
      - **Editar** carga el libro en el formulario para cambiar sus datos.

# CONSULTAS DECLARADAS - @QUERY

- **Tarea 1:** TAREAS/TareaUno
  - Crea el controlador **LibroController** con los endpoints:
    - **/index** muestra todos los libros
    - **/form** muestra el formulario para editar y crear uno nuevo.
    - **/insert** insertar un libro recibido por POST desde index .
    - **/delete/{id}** elimina el libro con el id recibido por la ruta.
    - **/update/{id}** muestra el formulario con los datos del libro con ese id
    - **/filter** que ejecuta la consulta según la opción elegida en el select con el número que tiene en la lista de las consultas declaradas.

# CONSULTAS DECLARADAS - @QUERY


- **Tarea 1:** TAREAS/TareaUno

localhost:8080/index

## Libros

Todos Nuevo Filtrar

Id	Título	Autor	Paginas		
1	El Quijote	Cervantes	500	Editar	Eliminar
7	El péndulo de Foucault	Eco	450	Editar	Eliminar
8	A veces llueve en agosto	Rain Man	123	Editar	Eliminar



localhost:8080/form

## Libros

Todos Nuevo Filtrar

### Formulario Libro

Id

Título

Autor

Paginas

0

Enviar

localhost:8080/update/1

## Libros

Todos Nuevo Filtrar

### Formulario Libro

Id

1

Título

El Quijote

Autor

Cervantes

Paginas

500

Enviar

localhost:8080/filter

## Libros

Todos Nuevo Filtrar

### Formulario filtrado

que tengan el id que se pasa

Id

Título

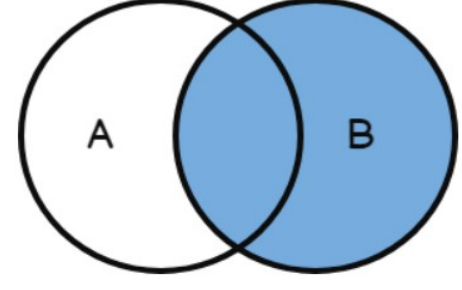
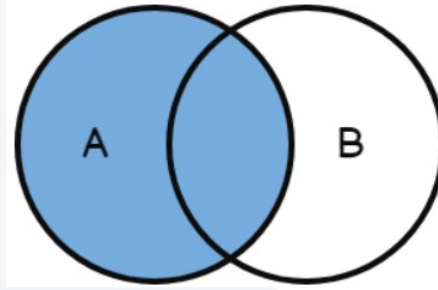
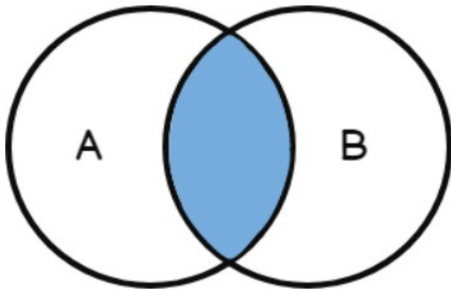
Autor

Paginas

0

Filtrar

- **Join** es el proceso de tomar datos de varias entidades y mostrarlos en una vista conjunta. Los tipos que existen son:
- **Inner** Join sólo se mostrarán los datos que estén en las dos tablas.
- **Left** Join los que estén en la tabla A.
- **Rigth** Join los que estén en la tabla B. (¡¡Cuidado!! Devuelve B en A)



# Inner Join - @ManyToOne

- Un **Usuario** tiene varios **Telefonos** y un **Telefono** solo es de un **Usuario**.
- Esta relación creará una columna en la tabla Telefono

```
@Entity
@Table(name="usuarios")
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Integer id;

    @OneToMany(mappedBy="usuario")
    Set<Telefono> telefonos = new HashSet<Telefono>();
}
```

```
@Entity
@Table(name = "telefonos")
public class Telefono {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Integer id;

    @ManyToOne
    @JoinColumn(name = "usuario_id")
    Usuario usuario;
}
```



# Inner Join - @ManyToOne

- Para obtener los usuarios que tengan un telefono con una consulta declarada en el Repositorio sería esta:

```
@Query("SELECT u FROM com.aepi.model.Usuario u JOIN u.telefonos")  
public List<Usuario> getAllTieneTelefono();
```

- Para obtener los telefonos que sean de un Usuario:

```
@Query("SELECT u FROM com.aepi.model.Telefono u JOIN u.usuario")  
public List<Telefono> getAllTieneUsuario();
```



# Inner Join - @ManyToMany

- Un **Grupo** tiene varios **Usuarios** y un **Usuario** está en varios **Grupos**.
- Esta relación creará una nueva tabla llamada grupo\_usuario

```
@ManyToMany(cascade = { CascadeType.PERSIST, CascadeType.MERGE })  
@JoinTable(name = "grupo_usuario",  
           joinColumns = @JoinColumn(name = "grupo_id"),  
           inverseJoinColumns = @JoinColumn(name = "usuario_id"))  
private Set<Usuario> usuarios = new HashSet<Usuario>();
```

```
@ManyToMany(mappedBy = "usuarios")  
private Set<Grupo> grupos = new HashSet<>();
```





# Inner Join - @ManyToMany

- Para obtener los **Grupos** que tengan **Usuarios** con una consulta declarada en el Repositorio sería esta:

```
@Query("SELECT g FROM com.aepi.model.Grupo g JOIN g.usuarios")  
public List<Grupo> getAllTienenUsuarios();
```

- Para obtener los **Usuarios** que estén en algún **Grupo**:

```
@Query("SELECT u FROM com.aepi.model.Usuario u JOIN u.grupos")  
public List<Usuario> getAllEstanGrupo();
```



# CONSULTAS DERIVADAS

- Existen multitud de query methods disponibles. Más en [Query methods](#)

Keyword	Sample	JPQL snippet <i>Java Persistence Query Language</i>
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
LessThanEqual	findByAgeLessThanEqual	... where x.age <= ?1
GreaterThan	findByAgeGreaterThan	... where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	... where x.age >= ?1
After	findByStartDateAfter	... where x.startDate > ?1
Before	findByStartDateBefore	... where x.startDate < ?1
IsNull	findByAgeIsNull	... where x.age is null
IsNotNull, NotNull	findByAge (Is) NotNull	... where x.age not null
Like	findByFirstnameLike	... where x.firstname like ?1
NotLike	findByFirstnameNotLike	... where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	... where x.firstname like ?1 (parameter bound with appended %) → where x.firstname like '%%'
EndingWith	findByFirstnameEndingWith	... where x.firstname like ?1 (parameter bound with prepended %) → where x.firstname like '%%'
Containing	findByFirstnameContaining	... where x.firstname like ?1 (parameter bound wrapped in %) → where x.firstname like '%%'
OrderBy	findByAgeOrderByLastNameDesc	... where x.age = ?1 order by x.lastname desc
Not	findByLastNameNot	... where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	... where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> age)	... where x.age not in ?1
True	findByActiveTrue()	... where x.active = true
False	findByActiveFalse()	... where x.active = false
IgnoreCase	findByFirstnameIgnoreCase	... where UPPER(x.firstname) = UPPER(?1)
First	findFirstBy	... limit 1
Top	findTopBy	... limit 1
First	findFirst15By	... limit 15
Top	findTop15By	... limit 15
Top	findTop20ByOrderByAgeDesc	... order by x.age desc limit 20

# CONSULTAS DERIVADAS

- El nombre del método debe seguir una serie de reglas para que de él se pueda derivar la consulta.
- La consulta debe comenzar por:
  - find...By puede no devolver ningún resultado.
  - get...By siempre debe devolver algo o dará Excepción.
  - read...By
  - query...By
  - count...By
- **By** es el delimitador para definir los criterios.



# CONSULTAS DERIVADAS

- **Ejemplo 2:** EJEMPLO/EjemploQuerysMethodsDerivadas
  - La clase Producto tiene los atributos id, idCategoria y nombre.
  - Para obtener de la tabla productos todos los productos deberíamos crear el método siguiente:

```
List<Producto> findAll();
```

- SQL:

```
SELECT * FROM productos
```

# CONSULTAS DERIVADAS

- **Ejemplo 2:** EJEMPLO/EjemploQuerysMethodsDerivadas
  - La clase Producto tiene los atributos id, idCategoria y nombre.
  - Para obtener de la tabla productos aquel que tengan un id concreto deberíamos crear el método siguiente:

Producto `findById(int id);`

- SQL:

`SELECT * FROM productos where id = ?`

# CONSULTAS DERIVADAS

- **Ejemplo 2:** EJEMPLO/EjemploQuerysMethodsDerivadas

- La clase Producto tiene los atributos id, idCategoria y nombre.
- Para obtener de la tabla productos todos los que pertenezcan a una categoria y los resultados se ordenen de forma ascendente por el nombre del producto.

```
List<Producto> findByIdCategoriaOrderByNombreAsc(int idCategoria);
```

- SQL:

```
SELECT * FROM productos where idCategoria = ? ORDER By nombre ASC;
```

# CONSULTAS DERIVADAS

- **Ejemplo 2:** EJEMPLO/EjemploQuerysMethodsDerivadas
  - La clase Producto tiene los atributos id, idCategoria y nombre.
  - Para obtener de la tabla productos todos los que pertenezcan a una categoria y su idCategoria sea mayor a un número.

```
List<Producto> findByIdCategoriaGreaterThan(int idCategoria);
```

- SQL:

```
SELECT * FROM productos where idCategoria > ?;
```

# CONSULTAS DERIVADAS

- **Ejemplo 2:** EJEMPLO/EjemploQuerysMethodsDerivadas
  - La clase Producto tiene los atributos id, idCategoria y nombre.
  - Para obtener de la tabla productos si existe un producto que tenga un nombre concreto.

boolean **existsByNombre**(String **nombre**);

- SQL:

```
SELECT * FROM productos where EXISTS (SELECT * FROM productos  
WHERE nombre = ?);
```



# CONSULTAS DERIVADAS

- **Ejemplo 2:** EJEMPLO/EjemploQuerysMethodsDerivadas
  - La clase Producto tiene los atributos id, idCategoria y nombre.
  - Para obtener de la tabla productos la cantidad de productos que pertenecen a una categoría.

```
long countByIdCategoria(int idCategoria);
```

- SQL:

```
SELECT count(*) FROM productos where idCategoria = ?;
```

# CONSULTAS DERIVADAS

- **Tarea 1 Bis:** TAREAS/TareaUnoBis

- A partir de la Tarea 1 modifica ILibroRepo para que las consultas también se puedan hacer mediante consultas derivadas.
  - 1 que tengan el id que se pasa.
  - 2 ordenados ascendentemente por título.
  - 3 ordenados descentemente por página.
  - 4 con un número de páginas mayor a Z.
  - 5 que su autor sea igual a X.
  - 6 que su autor sea X y tengan más de Z páginas.
- index.html añada un checkbox para elegir si la consulta será derivada.

# CONSULTAS DERIVADAS

- **Tarea 2:** TAREAS/TareaDos

- Crea un proyecto con las dependencias de Spring Web, Thymeleaf, Validation, MySql Driver, Spring Boot Dev Tools, Spring Data JPA.
- Crea la BD **socios**.
- Crea la clase **Socio** (Integer id, String nombre, String apellidos, int edad, double cuota, boolean activo).
- Añade las anotaciones para la persistencia como Entity.

# CONSULTAS DERIVADAS

- **Tarea 2:** TAREAS/TareaDos
  - Crea la interfaz **ISociosRepo** que extienda a JpaRepository con las consultas derivadas para:
    - 1 **obtener todos** los socios.
    - 2 **obtener** un socio por el id que se pasa.
    - 3 **obtener todos** los socios **ordenados** por el **nombre asc.**
    - 4 **obtener todos** los socios que estén **activos.**
    - 5 **obtener todos** los que su **apellido** empiece por una **letra.**
    - 6 **contar** cuantos socios activos hay.

# CONSULTAS DERIVADAS

- **Tarea 2:** TAREAS/TareaDos
  - Crea el controlador **SocioController** con los endpoints:
    - **/index** muestra la vista **index.html**
      - Una tabla de BootStrap que muestra los **socios** de la BD.
      - Un formulario con un SELECT con todas las opciones de consulta declaradas en el Repositorio.
    - **/filter** ejecuta la consulta según la opción elegida en el select y muestra el resultado en **index.html**
  - Inserta en la tabla socios al menos 5 registros al iniciar.

# CONSULTAS DERIVADAS

- **Tarea 2:** TAREAS/TareaDos

localhost:8080/index

## Formulario filtrado

todos

Id

Nombre

Apellidos

Filtrar

## Socios

Id	Nombre	Apellidos	Edad	Cuota	Activo
1	Pedro	Perez	14	13.0	true
2	Maria	Alvarez	44	23.0	false
3	Alberto	Ruiz	22	35.0	true
4	Alicia	Velazquez	11	45.0	false
5	Benito	Del Carmen	23	23.0	true

Socios - Hay 3 socios activos

- **Tarea 3:** TAREAS/TareaTres

- A partir del proyecto de la Tarea 2 del módulo 10 añade la vista **busqueda.html** con un formulario para filtrar las canciones con consultas derivadas.
  - 1 Todos.
  - 2 Ordenados ascendentemente por el titulo.
  - 3 Ordenados ascendentemente por la duracion.
  - 4 Su titulo contenga unas letras.
  - 5 Su duracion sea mayor a la pasada.
- Inserta al menos 5 registros de cada clase en la BD al iniciar.

- **Tarea 4:** TAREAS/TareaCuatro

- A partir del proyecto de la Tarea 4 del módulo 10 añade la vista **busquedas.html** con un formulario para filtrar los trenes con consultas declaradas.
  - 1 Todos.
  - 2 Ordenados por la matricula ascendentemente.
  - 3 Los que tienen vagones.
  - 4 Los que tienen viajeros. (JOIN FETCH una tabla generada)