El concepto de Spring Data Projections es uno de los conceptos importantes a nivel de Spring Data . ¿Para que sirve una proyección ? . Una proyección sirve para generar un DTO rápido sobre una consulta de Spring Data . Vamos a ver un ejemplo sencillo con la clase Libro y Spring Data como Framework.[ihc-hide-content ihc_mb_type=”show” ihc_mb_who=”4″ ihc_mb_template=”1″]

```
package com.arquitecturajava.data1;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table (name="Libros")
public class Libro {
  @Id
  private String isbn;
  private String titulo;
  private String autor;
  private Date fecha;
  private double precio;
  public Libro(String isbn, String titulo, String autor, Date fecha,
double precio) {
    super();
```

```
  this.isbn = isbn;
  this.titulo = titulo;
  this.autor = autor;
  this.fecha = fecha;
  this.precio = precio;
}
public Libro() {
  super();
}


public Libro(String isbn) {
  super();
  this.isbn = isbn;
}


public String getIsbn() {
  return isbn;
}
public void setIsbn(String isbn) {
  this.isbn = isbn;
}
public String getTitulo() {
  return titulo;
}
public void setTitulo(String titulo) {
  this.titulo = titulo;
}
public String getAutor() {
  return autor;
```

```
  }
  public void setAutor(String autor) {
    this.autor = autor;
  }
  public Date getFecha() {
    return fecha;
  }
  public void setFecha(Date fecha) {
    this.fecha = fecha;
  }
  public double getPrecio() {
    return precio;
  }
  public void setPrecio(double precio) {
    this.precio = precio;
  }


  @Override
  public String toString() {
    return "Libro [isbn=" + isbn + ", titulo=" + titulo + ", autor=" +
autor + ", fecha=" + fecha + ", precio="
        + precio + "]";
  }


  @Override
  public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((isbn == null) ? 0 : isbn.hashCode());
```

```
    return result;
  }


  @Override
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    Libro other = (Libro) obj;
    if (isbn == null) {
      if (other.isbn != null)
        return false;
    } else if (!isbn.equals(other.isbn))
      return false;
    return true;
  }
}
```

Una vez tenemos la clase libro construida , el siguiente paso es generar un Repositorio de Spring Data que se encargue por ejemplo de buscar los libros por titulo . Un enfoque sencillo es usar un namedMethod que se apoya en convenciones.

```
package com.arquitecturajava.data1;

import java.util.List;

import org.springframework.data.repository.CrudRepository;
```

```
public interface LibroRepository extends CrudRepository<Libro, String>
{

  // nombres diseñados por convencion
  public List<Libro> findByTitulo(String titulo);
}
```

Con esto es suficiente disponemos de una clase de repositorio que nos busca los libros en la base de datos. Es momento de construir una prueba unitaria y ver si nuestro repositorio es capaz de seleccionar el libro adecuado por titulo . Esta prueba hará uso de scripts de carga de datos.

```
package com.arquitecturajava.data1;

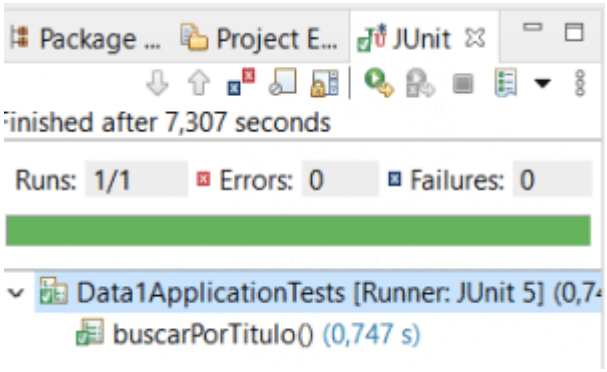
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.greaterThan;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.hamcrest.Matchers.equalTo;
import static org.hamcrest.Matchers.hasItem;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.util.List;
import java.util.Optional;

import org.assertj.core.util.Arrays;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.jdbc.Sql;

@SpringBootTest
```

```
@Sql({ "/schema.sql", "/data.sql" })
class Data1ApplicationTests {
  @Autowired
  LibroRepository repositorioLibro;
  @Test
  void buscarPorTitulo() {
    List<Libro> lista = repositorioLibro.findByTitulo("PHP");
    assertThat(lista,hasItem(new Libro("3B")));
  }
}
```

Si ejecutamos la prueba unitaria esta funciona sin problema



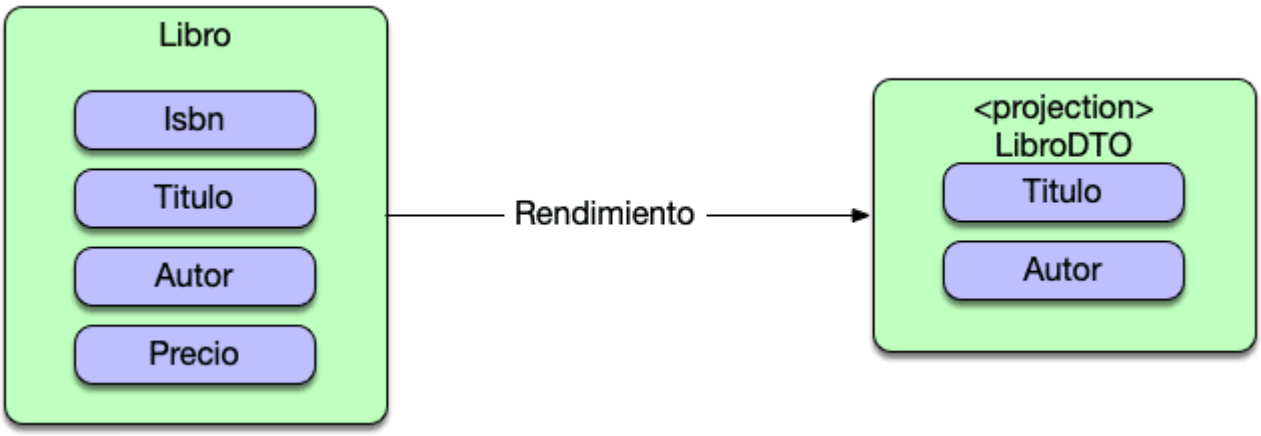
Spring Data Projections

¿Es esta una consulta correcta ? . En principio si no tenemos porque dudar de ello. Sin embargo existen situaciones en las que quizás alguien nos solicite una información mas compacta es decir no necesitamos devolver una lista con todos los campos de los libros que es lo que sucede ahora si revisamos la consulta SQL que el framework lanza por la consola.

```
select libro0_.isbn as isbn1_1_, libro0_.autor as autor2_1_,
libro0_.fecha as fecha3_1_, libro0_.precio as precio4_1_,
libro0_.titulo as titulo5_1_ from Libros libro0_ where
```

```
libro0_.titulo=?
```

¿Qué pasaría si solo quisiéramos un par de campos? . En este casuística en vez de devolver un objeto de negocio , tenemos que devolver un DTO o a nivel de Spring Data una sencilla Proyección que se construye apoyándonos en un interface.



Vamos a ver el código necesario para realizar este refactoring:

```
package com.arquitecturajava.data1;

public interface LibroDTO {

  String getTitulo();
  String getAutor();

}
```

El primer paso es definir el interface , el segundo paso es modificar la clase repositorio para que devuelva un Spring Projection que a fin y al cabo es un interface que hace las funciones de DTO sencillo.

```
package com.arquitecturajava.data1;
```

```
import java.util.List;

import org.springframework.data.repository.CrudRepository;


public interface LibroRepository extends CrudRepository<Libro, String>
{

  // nombres diseñados por convencion
  public List<LibroDTO> findByTitulo(String titulo);
}
```

Como se puede observar la lista ya no es una lista de Libros sino que es una lista de LibroDTO si revisamos la consulta generada por JPA nos encontraremos con que la consulta se ha reducido y selecciona los datos que realmente necesita.

```
select libro0_.titulo as col_0_0_, libro0_.autor as col_1_0_ from
Libros libro0_ where libro0_.titulo=?
```

Acabamos de construir un Spring Data Projection para mejorar el rendimiento de un repositorio en concreto

Otros artículos relacionados

1. Curso Spring Data y buenas prácticas
2. Spring Data Custom Query
3. JPA Orphan Removal y como usarlo
   [/ihc-hide-content]