02-2021

code
URJC

Desarrollo de Aplicaciones Web

Tema 3.2 – Bases de datos SQL en Spring

Universidad
Rey Juan Carlos

Micael Gallego
Correo: micael.gallego@urjc.es
Twitter: @micael_gallego

Bases de datos SQL en Spring

code URJC

- **Bases de datos SQL en Spring**
- Java Persistence API (JPA)
- Consultas
- Paginación
- Gestión del esquema
- Arquitectura
- Imágenes
- Accesos concurrentes

Bases de datos SQL en Spring

code URJC

- El proyecto **Spring Data** ofrece mecanismos para el acceso a **Bases de datos relacionales y no relacionales**
  - **Creación del esquema** partiendo de las clases del código Java (o viceversa)
  - **Conversión automática** entre objetos Java y el formato propio de la base de datos
  - **Creación de consultas** en base a métodos en interfaces

http://projects.spring.io/spring-data/
http://projects.spring.io/spring-data-jpa/

Bases de datos SQL en Spring

code URJC

ejem1

- **Objeto de la base de datos (entidad)**

Anotaciones usadas para generar el esquema y para hacer la conversión entre objetos y filas

El atributo anotado con **@Id** es la clave primaria de la tabla. Lo habitual es que se genere de forma automática

```
@Entity
public class Customer {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private long id;

  private String firstName;
  private String lastName;

  // Constructor necesario para la carga desde BBDD
  protected Customer() {}

  public Customer(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  // Getter, Setters and toString
  ...
}
```

Bases de datos SQL en Spring

code URJC

ejem1

- **Repositorio**

El interfaz padre **JpaRepository** dispone de métodos para **consultar, guardar, borrar** y **modificar** objetos de la base de datos.

```
public interface CustomerRepository extends JpaRepository<Customer, Long> {

   List<Customer> findByLastName(String lastName);

   List<Customer> findByFirstName(String firstName);

}
```

Métodos que se traducen automáticamente en **consultas a la BBDD** en base al nombre y los parámetros.

Si no se necesita ningún tipo de consulta especial, el **interfaz puede quedar vacío**

Clase de la **entidad** y clase de su identificador (generalmente **Integer** o **Long**)

Bases de datos SQL en Spring

code URJC

ejem1

- **Uso de la base de datos**
  - Un componente **inyecta el repositorio** con @Autowired
  - Métodos disponibles:
    - Consultar todos (**findAll**)
    - Consulta por id (**findById**)
    - Guardar un nuevo objeto o actualizarlo (**save**)
    - Borrar un objeto (**delete**)
    - Consultas por cualquier criterio (**métodos añadidos al interfaz**)

Bases de datos SQL en Spring

code URJC

ejem1

- Uso de la BBDD

```
@Controller
public class DataBaseUsage implements CommandLineRunner {

  @Autowired
  private CustomerRepository repository;

  @Override
  public void run(String... args) throws Exception {

     repository.save(new Customer("Jack", "Bauer"));
     repository.save(new Customer("Chloe", "O'Brian"));

     List<Customer> bauers = repository.findByLastName("Bauer");
     for (Customer bauer : bauers) {
          System.out.println(bauer);
     }

     repository.delete(bauers.get(0));
  }
}
```

Controlador especial que se ejecuta cuando se **inicia la aplicación**. Puede leer los parámetros de la **línea de comandos**

Código de ejemplo que usa el **repositorio** para guardar, consultar y borrar datos de la **BBDD**

Bases de datos SQL en Spring

code URJC

ejem1

- pom.xml

```
<groupId>es.codeurjc</groupId>
<artifactId>bbdd_ejem1</artifactId>
<version>0.1.0</version>

<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>3.0.0-RC2</version>
</parent>
...
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
 <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
  </dependency>
</dependencies>
```

Dependencia a **Spring Data** para BBDD **relacionales**

Dependencia a la **BBDD en memoria H2**. En las bases de datos en memoria, el esquema se genera de forma **automática** al iniciar la app

Bases de datos SQL en Spring

code URJC

ejem2

- Una aplicación web puede gestionar la información en una BBDD
  - Se añaden las dependencias de JPA y la BBDD en el **pom.xml**
  - En vez de usar una estructura en memoria se usa un **repositorio**
- Como ejemplo vamos a transformar la API REST de gestión de posts

Bases de datos SQL en Spring

code URJC

ejem2

- pom.xml

```
...
<dependencies>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
  </dependency>

</dependencies>
```

Dependencias **Web**

Dependencia a la **BBDD H2**

Bases de datos SQL en Spring

code URJC

ejem2

- Objeto de la base de datos (entidad)

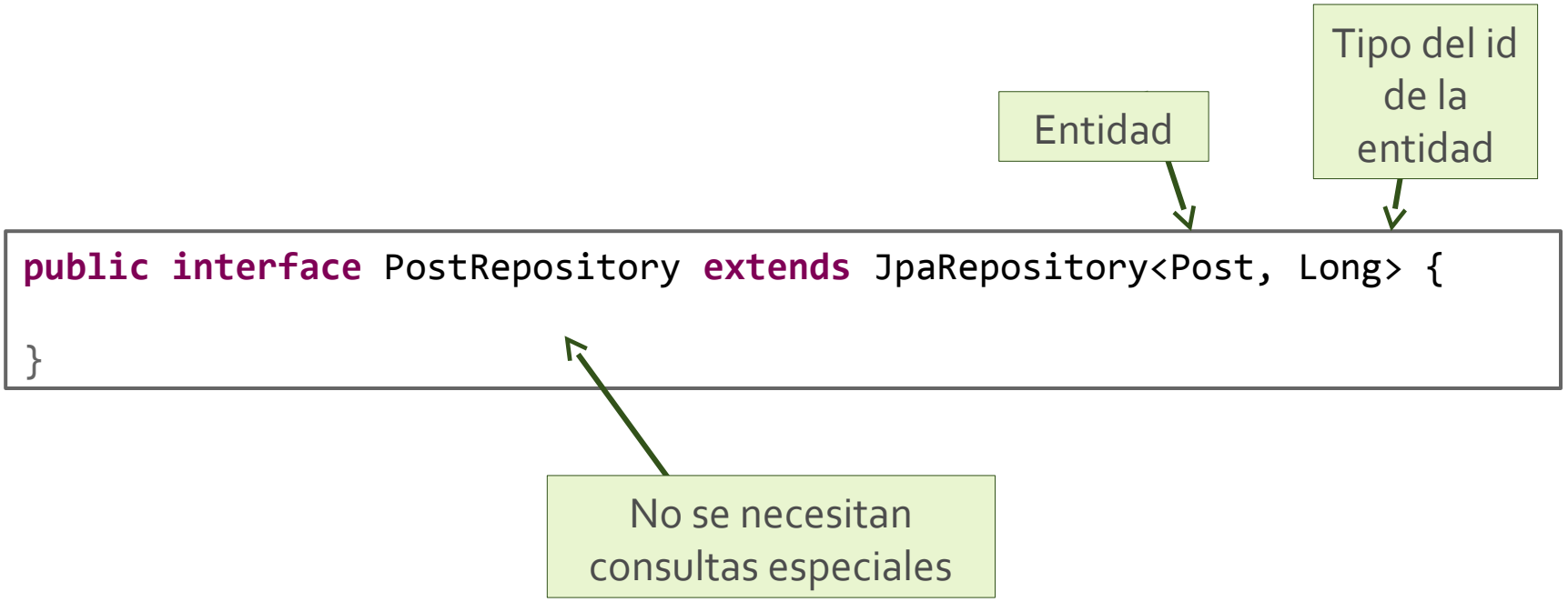
```
@Entity
public class Post {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long id;

	private String username;
	private String title;
	private String text;

	public Post() { }

	public Post(String username, String title, String text) {
		super();
		this.username = username;
		this.title = title;
		this.text = text;
	}
}
```

Bases de datos SQL en Spring

code URJC

ejem2

- Repositorio



Bases de datos SQL en Spring

code URJC

ejem2

- Controlador REST

Se pueden inyectar tantos repositorios como sea necesario

Un método **@PostConstruct** se ejecutará después de haber inyectado las dependencias

```
@RestController
@RequestMapping("/posts")
public class PostController {

  @Autowired
  private PostRepository posts;

  @PostConstruct
  public void init() {
    posts.save(new Post("Pepe", "Vendo moto", "Barata, barata"));
    posts.save(new Post("Juan", "Compro coche", "Pago bien"));
  }

  @GetMapping("/")
  public Collection<Post> getPosts() {
    return posts.findAll();
  }

...
```

Desde los métodos se usa el repositorio

Bases de datos SQL en Spring

code URJC

ejem2

- **Consulta de un recurso (GET)**
  - Gestión de *Optional* con *findById*

```
@GetMapping("/{id}")
public ResponseEntity<Post> getPost(@PathVariable long id) {

  Optional<Post> post = posts.findById(id);

  if (post.isPresent()) {
    return ResponseEntity.ok(post.get());
  } else {
    return ResponseEntity.notFound().build();
  }
}
```

Bases de datos SQL en Spring

code URJC

ejem2

- **Consulta de un recurso (GET)**
  - Gestión de *Optional* con *findById*

```
@GetMapping("/{id}")
public ResponseEntity<Post> getPost(@PathVariable long id) {

  Optional<Post> post = posts.findById(id);

  if (post.isPresent()) {
    return ResponseEntity.ok(post.get());
  } else {
    return ResponseEntity.notFound().build();
  }
}
```

Si hay valor (**isPresent**) lo obtenemos (**get**) para devolverlo

En un repositorio el método **findById** devuelve *Optional<Post>*

Bases de datos SQL en Spring

code URJC

ejem2

- Creación de recurso (POST)

```
@PostMapping("/")
public ResponseEntity<Post> createPost(@RequestBody Post post) {

  posts.save(post);

  URI location = fromCurrentRequest().path("/{id}")
    .buildAndExpand(post.getId()).toUri();

  return ResponseEntity.created(location).body(post);
}
```

Bases de datos SQL en Spring

code URJC

ejem2

- Reemplazo de recurso (PUT)

```
@PutMapping("/{id}")
public ResponseEntity<Post> replacePost(@PathVariable long id,
  @RequestBody Post newPost) {

  Optional<Post> post = posts.findById(id);

  if (post.isPresent()) {

    newPost.setId(id);
    posts.save(newPost);

    return ResponseEntity.ok(newPost);
  } else {
     return ResponseEntity.notFound().build();
  }
}
```

Bases de datos SQL en Spring

code URJC

ejem2

- Borrado de recurso (DELETE)

```
@DeleteMapping("/{id}")
public ResponseEntity<Post> deletePost(@PathVariable long id) {

  Optional<Post> post = posts.findById(id);

  if (post.isPresent()) {
    posts.deleteById(id);
    return ResponseEntity.ok(post.get());
  } else {
    return ResponseEntity.notFound().build();
  }
}
```

Bases de datos SQL en Spring

code URJC

ejem2

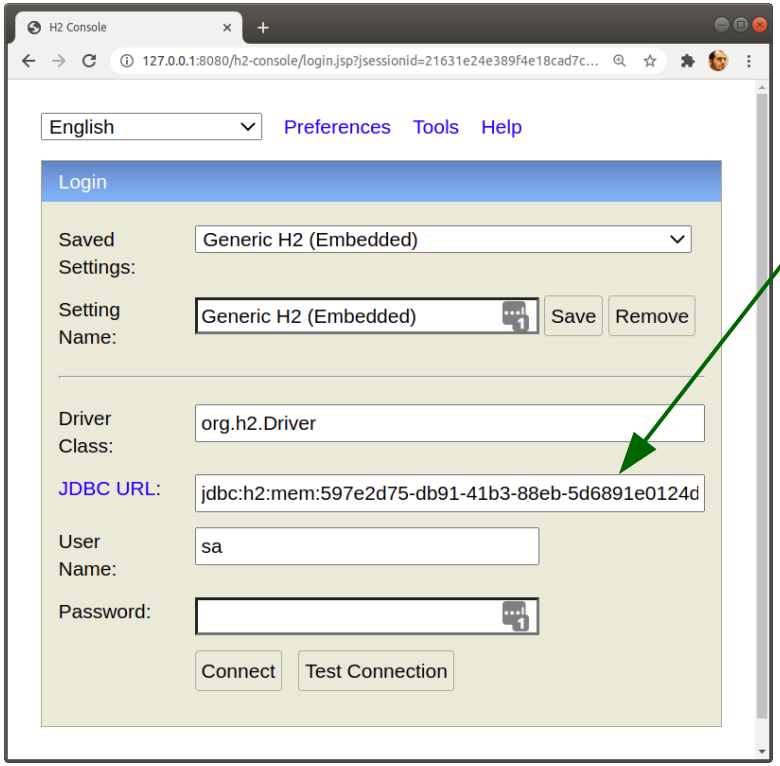
- **Consola web de base de datos H2**
  - Si desarrollamos una **aplicación web** que use una **base de datos H2** podemos acceder a una consola web de la propia BBDD
  - La consola es accesible en
    http://127.0.0.1:8080/h2-console
  - Para activar esa consola:
    - Usar las **devtools** en ese proyecto
    - O especificar en el fichero **application.properties**

```
spring.h2.console.enabled=true
```

Bases de datos SQL en Spring

ejem2

- Consola web de base de datos H2



Hay que poner como JDBC URL la ruta que aparece en el log

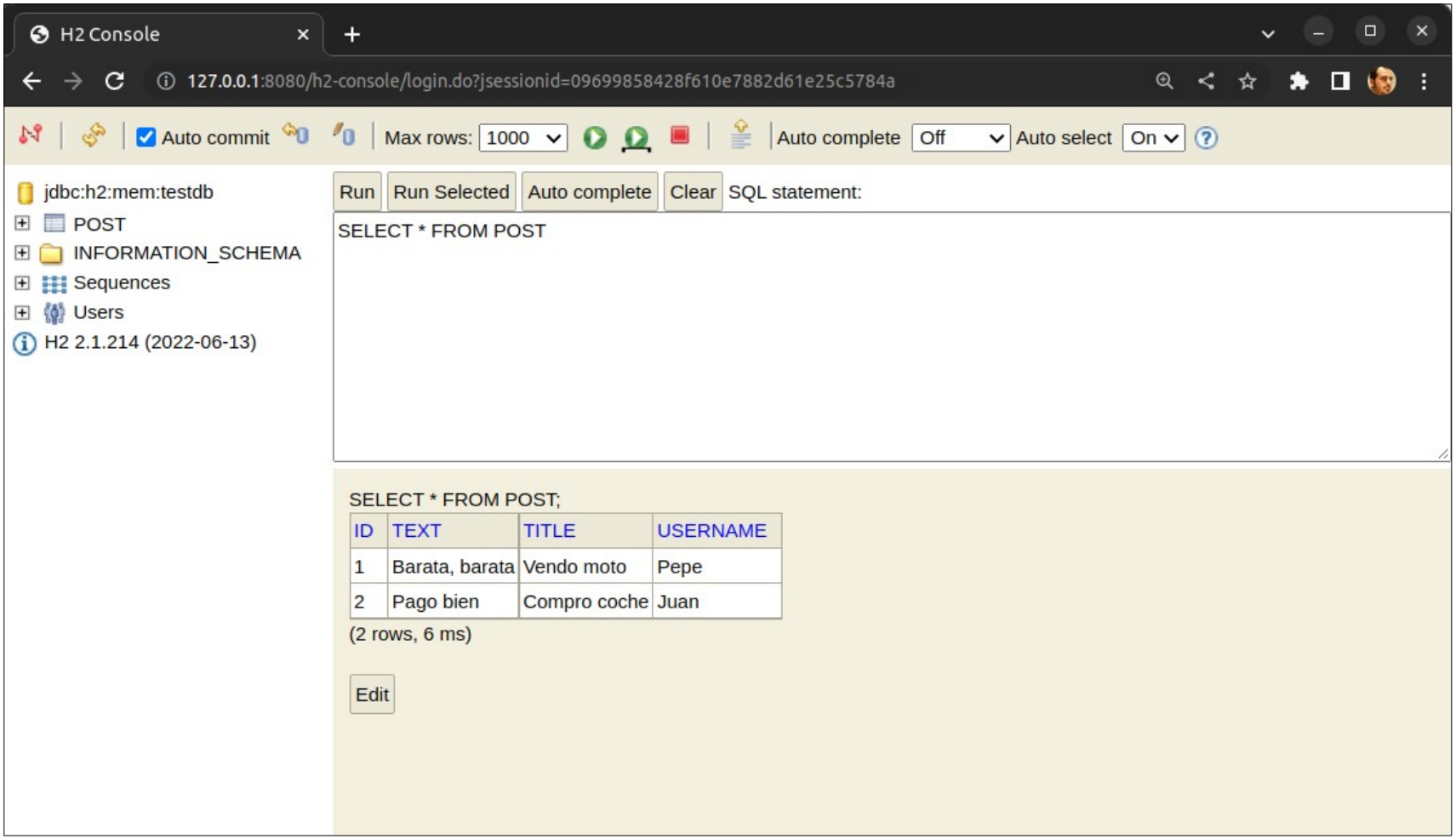
```
H2 console available at '/h2-
console'. Database available at
'jdbc:h2:mem:597e2d75-db91-41b3-
88eb-5d6891e0124d'
```

Para simplificar podemos configurar una URL fija:

```
spring.datasource.url=jdbc:h2:mem:testdb
```

Bases de datos SQL en Spring

ejem2



Bases de datos SQL en Spring

- **Manipulación de Optional**
  - En las primeras versiones de Java para indicar la **ausencia de valor** se devolvía **null**
  - Pero **null** es propenso a errores y puede provocar **NullPointerException**
  - **Optional** representa en el sistema de tipos que el valor es opcional y obliga al desarrollador a considerar cómo quiere gestionar la ausencia de valor

https://blogs.oracle.com/javamagazine/12-recipes-for-using-the-optional-class-as-its-meant-to-be-used

Bases de datos SQL en Spring

- **Manipulación de Optional**
  - SpringBoot ofrece dos estrategias para gestionar mejor el Optional en una API REST:
    - **Optional.orElseThrow():** Se eleva una excepción NoSuchElementException si el recurso no existe. Esa excepción se puede convertir en un 404
    - **ResponseEntity.of(Optional resource):** Devuelve 404 si el recurso no existe.

Bases de datos SQL en Spring

code URJC

ejem3

- **Manipulación de Optional**
  - Configuración de Spring para que las excepciones **NoSuchElementException** generadas en un controlador se conviertan en un 404

```
@ControllerAdvice
class NoSuchElementExceptionControllerAdvice {

  @ResponseStatus(HttpStatus.NOT_FOUND)
  @ExceptionHandler(NoSuchElementException.class)
  public void handleNoTFound() {
  }
}
```

Bases de datos SQL en Spring

code URJC

- ## Manipulación de Optional

ejem3

Elevando la excepción

```
@GetMapping("/{id}")
public Post getPost(@PathVariable long id) {

  return posts.findById(id).orElseThrow();
}
```

Si no hay post se eleva **NoSuchElementException**

ejem4

ResponseEntity.of(...)

```
@GetMapping("/{id}")
public ResponseEntity<Post> getPost(@PathVariable long id) {

  Optional<Post> post = posts.findById(id);

  return ResponseEntity.of(post);
}
```

Método **ResponseEntity.of()** devuelve 404 si el post no está presente

Bases de datos SQL en Spring

code URJC

- ## Manipulación de Optional

ejem3

Elevando la excepción

```
@DeleteMapping("/{id}")
public Post deletePost(@PathVariable long id) {

  Post post = posts.findById(id).orElseThrow();
  posts.deleteById(id);
  return post;
}
```

Si no hay post se eleva **NoSuchElementException**

ejem4

ResponseEntity.of(...)

```
@DeleteMapping("/{id}")
public ResponseEntity<Post> deletePost(@PathVariable long id) {

  Optional<Post> post = posts.findById(id);
  post.ifPresent(p -> posts.deleteById(id));
  return ResponseEntity.of(post);
}
```

Sólo se ejecuta la operación de borrado si hay post: **ifPresent()**

Bases de datos SQL en Spring

code URJC

- **Manipulación de Optional**

ejem3

Elevando la excepción

```
@PutMapping("/{id}")
public Post replacePost(@PathVariable long id,
  @RequestBody Post newPost) {

  posts.findById(id).orElseThrow();

  newPost.setId(id);
  posts.save(newPost);

  return newPost;
}
```

Bases de datos SQL en Spring

code URJC

- **Manipulación de Optional**

ejem4

ResponseEntity.of(...)

```
@PutMapping("/{id}")
public ResponseEntity<Post> replacePost(@PathVariable long id,
@RequestBody Post newPost) {

  Optional<Post> post = posts.findById(id);

  return ResponseEntity.of(post.map(p -> {

    newPost.setId(id);
    posts.save(newPost);

    return newPost;
  }));
}
```

El método **map(...)** permite guardar el newPost y devolverlo si hay post

Ejercicio 1

- Transforma el ejercicio de la **gestión de Items** para que utilice una base de datos en vez de guardar la información en memoria

Bases de datos SQL en Spring

code URJC

- Bases de datos SQL en Spring
- **Java Persistence API (JPA)**
- Consultas
- Paginación
- Gestión del esquema
- Arquitectura
- Imágenes
- Accesos concurrentes

Java Persistence API (JPA)

code URJC

- **Las apps de base de datos en Spring usan dos librerías a la vez:**
  - **JPA (*Java Persistence API*)**
    - Estándar de Jakarta EE para acceso a base de datos relacionales
  - **Spring Data**
    - Facilidades propias de Spring que facilitan el acceso a base de datos con JPA

Java Persistence API (JPA)

code URJC

- **Spring Data**
  - Se puede usar con **cualquier** base de datos (relacional o NoSQL)
  - Permite la creación de **consultas partiendo de métodos** en los repositorios
  - Los métodos pueden tener varios parámetros que forman **expresiones lógicas**: Y, O, No…
  - Los métodos se pueden anotar para definir la consulta de forma más precisa con **JPQL o SQL**

https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories

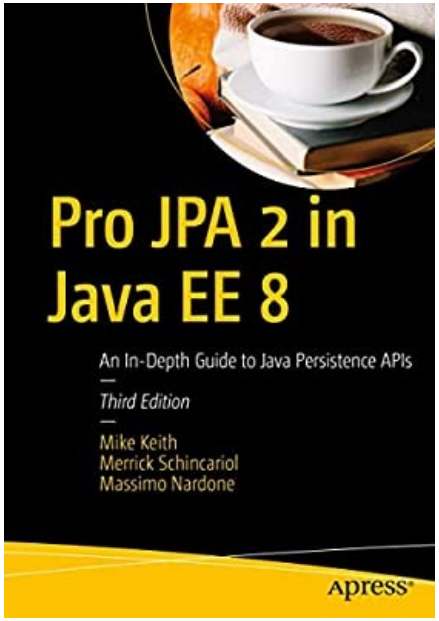
Java Persistence API (JPA)

- **JPA (*Java Persistence API*)**
  - Acceso a Bases de datos relacionales desde Java
  - **JPA** forma parte del estándar **Jakarta EE**
  - **Funcionalidades**
    - Generación del **esquema** de la base de datos partiendo de las entidades (clases Java)
    - **Conversión automática** entre objetos y filas de tablas
    - Lenguaje de consulta de alto nivel **JPQL**

https://jakarta.ee/specifications/persistence/3.1/

Java Persistence API (JPA)

- JPA (*Java Persistence API*)



- **Pro JPA 2 in Java EE 8**
- By Mike Keith , Merrick Schincariol, Massimo Nardone
- Apress (2018)

https://learning.oreilly.com/library/view/pro-jpa-2/9781484234204/

https://www.baeldung.com/learn-jpa-hibernate

Java Persistence API (JPA)

- **JPA (*Java Persistence API*)**
  - Para usar JPA es necesario usar una **librería** concreta que lo implemente
  - En Spring por defecto se usa la librería **Hibernate**

http://www.hibernate.org

Java Persistence API (JPA)

- **JPA (*Java Persistence API*)**
  - La técnica para convertir entre el modelo de objetos y el modelo relacional se conoce como "**mapeo objeto relacional**" (*object relational mapping*) u ORM
  - Un ORM realiza las **conversiones** pertinentes entre **objetos/clases** y **filas/tablas**
  - Modos de funcionamiento:
    - **Generación de tablas partiendo de clases***
    - Generación de clases partiendo de tablas

Generación de tablas desde entidades

code URJC

ejem4

- **Mostrar las sentencias SQL**
  - Para aprender JPA y depurar los posibles errores es muy útil poder ver qué sentencias se ejecutan en la BBDD

application.properties

```
spring.jpa.properties.hibernate.format_sql=true
logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
```

Generación de tablas desde entidades

code URJC

ejem4

- **Entidad con atributos básicos**
  - Se genera una tabla por cada entidad
  - Por cada atributo de la clase de un tipo simple (entero, float, String, boolean…), se crea un campo en la tabla

```
@Entity
public class Post {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private long id;

  private String username;
  private String title;
  private String text;
}
```

```
create table post (
  id bigint not null,
  text varchar(255),
  title varchar(255),
  username varchar(255),
  primary key (id)
)
```

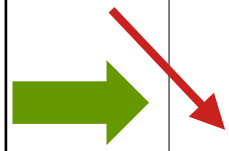
Generación de tablas desde entidades

code URJC

ejem4

- **Palabras reservadas SQL en entidades**
  - Si el nombre de una entidad o uno de sus atributos es una **palabra reservada en SQL** para esa base de datos, se obtendrá un **ERROR al ejecutar la sentencia**

```
@Entity
public class Post {

  @Id
  @GeneratedValue(...)
  private long id;

  private String user;
  private String title;
  private String text;
}
```



```
create table post (
  id bigint not null,
  text varchar(255),
  title varchar(255),
  user varchar(255),
  primary key (id)
)
```

ERROR!!!

Generación de tablas desde entidades

code URJC

ejem4

- **Palabras reservadas SQL en entidades**
  - **1)** Se puede cambiar el nombre de la tabla o atributo por uno que no sea palabra reservada con

```
@Table(name="nombreTabla")
```

```
@Column(name="nombreCampo")
```

2) Se puede escapar el nombre de la tabla o atributo

```
@Table(name="\"nombreTabla\"")
```

```
@Column(name="\"nombreCampo\"")
```

  - **3)** Se puede pedir a Hibernate que escape TODOS los nombres de tablas y campos en las sentencias SQL

```
spring.jpa.properties.hibernate.globally_quoted_identifiers=true
spring.jpa.properties.hibernate.globally_quoted_identifiers_skip_column_definitions=true
```

https://vladmihalcea.com/escape-sql-reserved-keywords-jpa-hibernate/

Generación de tablas desde entidades

- **Relaciones entre entidades**
  - Existe una relación cuando una entidad tiene como **atributo otra entidad**
  - En memoria, un **objeto** está asociado a uno o más **objetos**
  - También es posible tener una colección de entidades (**List, Set, Map…**)
  - Las relaciones pueden ser **1:1, 1:N, M:N**

Generación de tablas desde entidades

- **Relaciones entre entidades**
  - **1:1**
    - Unidireccional
    - Bidireccional
  - **1:N**
    - Unidireccional (1 → N y N → 1)
    - Bidireccional
  - **N:M**
    - Bidireccional

Generación de tablas desde entidades

code URJC

ejem5

- **Relación 1:1 unidireccional**
  - Una entidad tiene una referencia a otra entidad
  - El atributo que apunta a otro es anotado con **@OneToOne**
  - Cada objeto tiene que ser guardado en la base de datos de forma **independiente**
  - Al **cargar un objeto** de la BBDD podemos acceder al objeto **relacionado sin cargarlo explícitamente**

Generación de tablas desde entidades

code URJC

ejem5

- Relación 1:1 unidireccional

```
@Entity
public class Student {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int startYear;

  @OneToOne
  private Project project;

}
```

```
@Entity
public class Project {

  @Id
  @GeneratedValue(...)
  private long id;

  private String title;
  private int calification;
}
```

Generación de tablas desde entidades

code URJC

ejem5

- Relación 1:1 unidireccional

```
@Entity
public class Student {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int startYear;

  @OneToOne
  private Project project;

}
```

```
@Entity
public class Project {

  @Id
  @GeneratedValue(...)
  private long id;

  private String title;
  private int calification;
}
```

Relación 1:1 unidireccional

code URJC

ejem5

```java
@Entity
public class Student {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int startYear;

  @OneToOne
  private Project project;
}
```

```java
@Entity
public class Project {

  @Id
  @GeneratedValue(...)
  private long id;

  private String title;
  private int calification;
}
```

Generación de tablas

```
create table project (
  id bigint not null,
  calification integer not null,
  title varchar(255),
  primary key (id)
)

create table student (
  id bigint not null,
  name varchar(255),
  startYear integer not null,
  project_id bigint,
  primary key (id)
)

alter table student
  add constraint Fkr...
  foreign key (project_id)
  references project
```

Generación de tablas desde entidades

code URJC

ejem5

- Relación 1:1 unidireccional

```
Project p1 = new Project("TFG1", 8);
projectRepository.save(p1);

Student s1 = new Student("Pepe", 2010);
s1.setProject(p1);

Student s2 = new Student("Juan", 2011);

studentRepository.save(s1);
studentRepository.save(s2);
```

Cada objeto se tiene que guardar en su repositorio

Inicialización de datos

Generación de tablas desde entidades

code URJC

ejem5

- Relación 1:1 unidireccional

Consulta de datos

```
studentRepository.findAll();
```

```
[
 {
  "id" : 1,
  "name" : "Pepe",
  "year" : 2010,
  "project" : {
    "id" : 1,
    "title" : "TFG1",
    "calification" : 8
  }
 },
 {
  "id" : 2,
  "name" : "Juan",
  "startYear" : 2011
 }
]
```

Generación de tablas desde entidades

code URJC

ejem6

- **Operaciones en cascada**
  - Hay veces que un objeto **siempre está asociado** a otro objeto (relación de composición)
  - Por ejemplo, un proyecto siempre está asociado a un estudiante
  - Cuando se **crea el estudiante**, **se crea el proyecto**, cuando se borra el estudiante, se borra el proyecto

Generación de tablas desde entidades

code URJC

ejem6

- **Operaciones en cascada**
  - Si la anotación @OneToOne se configura con **cascade = CascadeType.ALL** entonces ambos objetos de la relación tienen el mismo ciclo de vida
  - Al **guardar** el objeto principal, se **guarda** el asociado
  - Al **borrar** el objeto principal, se **borra** el asociado

Generación de tablas desde entidades

code URJC

ejem6

- Operaciones en cascada

```
//Deleting a student delete her associated project
@DeleteMapping("/students/{id}")
public Student deleteStudent(@PathVariable Long id) {
    Student student = studentRepository.findById(id).get();
    studentRepository.deleteById(id);
    return student;
}

//A project only can be deleted if it has no associated student.
@DeleteMapping("/projects/{id}")
public Project deleteProject(@PathVariable Long id) {
    Project project = projectRepository.findById(id).get();
    projectRepository.deleteById(id);
    return project;
}
```

Generación de tablas desde entidades

code URJC

ejem6

- Operaciones en cascada

```
@Entity
public class Student {

    @Id
    @GeneratedValue(...)
    private long id;

    private String name;
    private int startYear;

    @OneToOne(cascade=CascadeType.ALL)
    private Project project;
}
```

Como la relación en **cascade=ALL**, no es necesario guardar el objeto project explícitamente.

El **project** se guarda cuando el **student** se guarda

```
Student s1 = new Student("Pepe", 2010);
s1.setProject(new Project("TFG1", 8));
studentRepository.save(s1);
```

Generación de tablas desde entidades

code URJC

ejem7

- **Relación 1:1 bidireccional**
  - Para facilitar el desarrollo, los **dos objetos de una relación** pueden tener un atributo apuntando al otro objeto
  - Ambas entidades tienen que anotarse con **@OneToOne**
  - Aunque en memoria ambos objetos se apunten entre sí, en la base de datos la relación se guarda en la **tabla de una entidad**

Generación de tablas desde entidades

code URJC

ejem7

- **Relación 1:1 bidireccional**
  - La entidad principal cuya tabla guarda la información anota el atributo con **@OneToOne**
  - El atributo debe apuntar al otro objeto de la otra entidad cuando se **guardar en la BBDD**
  - La otra entidad se anota con **@OneToOne(mappedby="attr")** indicando el nombre del atributo de la otra principal
  - Este atributo sólo se usa en lecturas

Generación de tablas desde entidades

ejem7

- Relación 1:1 bidireccional

```java
@Entity
public class Student {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int startYear;

  @OneToOne(cascade=CascadeType.ALL)
  private Project project;
}
```

```java
@Entity
public class Project {

  @Id
  @GeneratedValue(...)
  private long id;

  private String title;
  private int calification;

  @OneToOne(mappedBy="project")
  private Student student;

}
```

```java
Student s1 = new Student("Pepe", 2010);
s1.setProject(new Project("TFG1", 8));
studentRepository.save(s1);
```

Al guardar, la entidad principal debe apuntar a la otra entidad

Generación de tablas desde entidades

code URJC

ejem7

- Relación 1:1 bidireccional

```
create table project (
   id bigint not null,
   calification integer not null,
   title varchar(255),
   primary key (id)
)

create table student (
   id bigint not null,
   name varchar(255),
   startYear integer not null,
   project_id bigint,
   primary key (id)
)

alter table student
   add constraint FKr6av6arpoy7wpbqipg41id1mn
   foreign key (project_id)
   references project
```

Como la entidad principal sigue siendo Student, las tablas son las mismas que con relación unidireccional

Generación de tablas desde entidades

- **Relaciones entre entidades**
  - **1:1**
    - Unidireccional
    - Bidireccional
  - **1:N**
    - Unidireccional (1 → N y N → 1)
    - Bidireccional
  - **N:M**
    - Bidireccional

Generación de tablas desde entidades

code URJC

ejem8

- **Relación 1:N**
  - Cuando existe una relación **1:N** entre entidades se usan las anotaciones **@OneToMany** y **@ManyToOne**
  - Puede ser **unidireccional** o **bidireccional**
  - Cualquier sentido de la relación puede ser **la entidad principal** (la que se usa para guardar los datos)
  - También se puede usar **cascade**

Generación de tablas desde entidades

code URJC

ejem8

- Relación 1:N unidireccional

```
@Entity
public class Team {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int ranking;

  @OneToMany
  private List<Player> players;

}
```

```
@Entity
public class Player {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int goals;

}
```

Generación de tablas desde entidades

code URJC

ejem8

- Relación 1:N unidireccional

```
create table player (
  id bigint not null,
  goals integer not null,
  name varchar(255),
  primary key (id)
)

create table team (
  id bigint not null,
  name varchar(255),
  ranking integer not null,
  primary key (id)
)

create table team_players (
  team_id bigint not null,
  players_id bigint not null
)
```

Como es @OneToMany, un team sólo puede estar asociado a un equipo

```
alter table team_players
  add constraint UK... unique (players_id)

alter table team_players
  add constraint FK...
  foreign key (players_id)
  references player

alter table team_players
  add constraint FK...
  foreign key (team_id)
  references team
```

Tabla que relaciona Teams y Players

Generación de tablas desde entidades

code URJC

ejem8

- Relación 1:N unidireccional

```
Player p1 = new Player("Torres", 10);
Player p2 = new Player("Iniesta", 10);

playerRepository.save(p1);
playerRepository.save(p2);

Team team = new Team("Selección", 1);

team.getPlayers().add(p1);
team.getPlayers().add(p2);

teamRepository.save(team);
```

Generación de tablas desde entidades

code URJC

ejem8

- **Relación 1:N unidireccional**
  - Sin cascase los objetos tienen ciclos de vida independientes (Un player puede estar sin team)

```
//Deleting a team doesn't delete its associated players
@DeleteMapping("/teams/{id}")
public Team deleteTeam(@PathVariable Long id) {
  Team team = teamRepository.findById(id).get();
  //Force loading players from database to be returned as JSON
  Hibernate.initialize(team.getPlayers());
  teamRepository.deleteById(id);
  return team;
}
```

Generación de tablas desde entidades

code URJC

ejem9

- Relación 1:N unidireccional (cascade)

```
@Entity
public class Blog {

  @Id
  @GeneratedValue(...)
  private long id;

  private String title;
  private String text;

  @OneToMany(cascade=CascadeType.ALL)
  private List<Comment> comments;
}
```

```
@Entity
public class Comment {

  @Id
  @GeneratedValue(...)
  private long id;

  private String author;
  private String message;
}
```

```
Blog blog = new Blog("New", "My new product");
blog.getComments().add(new Comment("Cool", "Pepe"));
blog.getComments().add(new Comment("Very cool", "Juan"));
repository.save(blog);
```

Generación de tablas desde entidades

code URJC

ejem10

- Relación 1:N bidireccional

```
@Entity
public class Team {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int ranking;

  @OneToMany(mappedBy="team")
  private List<Player> players;

}
```

```
@Entity
public class Player {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int goals;

  @ManyToOne
  private Team team;
}
```

Generación de tablas desde entidades

code URJC

ejem10

- Relación 1:N bidireccional

```
create table player (
  id bigint not null,
  goals integer not null,
  name varchar(255),
  team_id bigint,
  primary key (id)
)

create table team (
  id bigint not null,
  name varchar(255),
  ranking integer not null,
  primary key (id)
)

alter table player
  add constraint FK...
  foreign key (team_id)
  references team
```

Ahora la entidad principal es **Player**, por eso tiene el atributo **team_id** y no hay tabla de relación

Generación de tablas desde entidades

code URJC

ejem10

- Relación 1:N bidireccional

```
Team team = new Team("Selección", 1);
teamRepository.save(team);


Player p1 = new Player("Torres", 10);
Player p2 = new Player("Iniesta", 10);

p1.setTeam(team);
p2.setTeam(team);

playerRepository.save(p1);
playerRepository.save(p2);
```

La **entidad principal** ahora es **Player**, por eso se configura el Team en el objeto Player antes de guardar (y no al revés)

Un Player puede estar sin asociar a un Team

Generación de tablas desde entidades

code URJC

ejem11

- **Relación 1:N bidireccional (cascade)**
  - Cuando se configura una relación de composición en la BBDD se suele configurar **cascade** y **orphanRemoval**
  - Se suele configurar **cascade=ALL** para que no haya que guardar los objetos de forma independiente
  - Se suele configurar **orphanRemoval=true** para que no pueda haber una parte sin si todo

https://vladmihalcea.com/orphanremoval-jpa-hibernate/

Generación de tablas desde entidades

code URJC

ejem11

- Relación 1:N bidireccional (cascade)

```
@Entity
public class Post {

      @Id
      @GeneratedValue(...)
      private long id;

      private String username;
      private String title;
      private String text;

      @OneToMany(mappedBy="post", cascade=CascadeType.ALL, orphanRemoval=true)
      private List<Comment> comments = new ArrayList<>();

      public void addComment(Comment comment) {
          comments.add(comment);
          comment.setPost(this);
      }

      public void removeComment(Comment comment) {
          comments.remove(comment);
          comment.setPost(null);
      }
}
```

Mantiene los atributos sincronizados en ambos extremos (Post y Comment)

Generación de tablas desde entidades

code URJC

ejem11

- Relación 1:N bidireccional (cascade)

```
@Entity
public class Comment {

    @Id
    @GeneratedValue(...)
    private long id;

    private String username;
    private String comment;

    @ManyToOne
    @JsonIgnore
    private Post post;

}
```

Generación de tablas desde entidades

code URJC

ejem11

- Relación 1:N bidireccional (cascade)

```
create table comment (
  id bigint not null,
  comment varchar(255),
  username varchar(255),
  post_id bigint,
  primary key (id)
)

create table post (
  id bigint not null,
  text varchar(255),
  title varchar(255),
  username varchar(255),
  primary key (id)
)

alter table comment
  add constraint FK..
  foreign key (post_id)
  references post
```

Generación de tablas desde entidades

code URJC

ejem11

- **Relación 1:N bidireccional (cascade)**
  - **API REST**
    - Al obtener un post individual o en una colección (GET) se devolverán los comentarios
    - Los comentarios de un post se podrán consultar, borrar, actualizar y añadir de forma independiente
      - http://server/posts/6/comments/3
    - Al reemplazar un post (PUT) no se verán afectados sus comentarios

Generación de tablas desde entidades

- **Relaciones entre entidades**
  - **1:1**
    - Unidireccional
    - Bidireccional
  - **1:N**
    - Unidireccional (1 → N y N → 1)
    - Bidireccional
  - **N:M**
    - Bidireccional

Generación de tablas desde entidades

code URJC

ejem12

- **Relación M:N**
  - Cuando existe una relación M:N entre entidades se usa la anotación **@ManyToMany**
  - Puede ser **unidireccional** o **bidireccional**
  - Cualquier sentido de la relación puede ser **la entidad principal** (la que se usa para guardar los datos)
  - También se puede usar **cascade**

Generación de tablas desde entidades

code URJC

ejem12

- Relación M:N bidireccional

```
@Entity
public class Team {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int ranking;

  @ManyToMany(mappedBy="team")
  private List<Player> players;

}
```

```
@Entity
public class Player {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int goals;

  @ManyToMany
  private List<Team> teams;

}
```

Generación de tablas desde entidades

ejem12

- Relación M:N bidireccional

```
create table player (
  id bigint not null,
  goals integer not null,
  name varchar(255),
  primary key (id)
)

create table team (
  id bigint not null,
  name varchar(255),
  ranking integer not null,
  primary key (id)
)

create table player_teams (
  players_id bigint not null,
  teams_id bigint not null
)
```

Como es @ManyToMany ya no está la restricción de que un player sólo pueda estar asociado a un team

```
alter table player_teams
  add constraint FK...
  foreign key (teams_id)
  references team

alter table player_teams
  add constraint FK...
  foreign key (players_id)
  references player
```

Generación de tablas desde entidades

code URJC

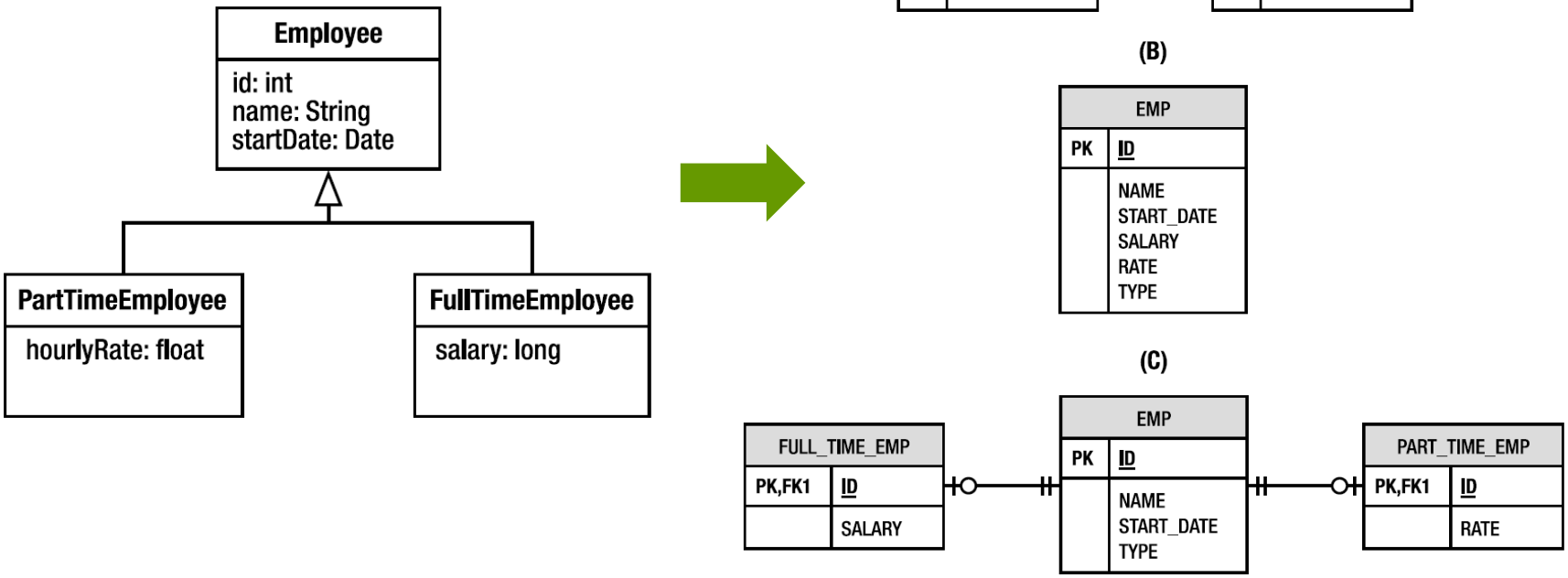
ejem12

- Relación M:N bidireccional

```
Team team1 = new Team("Selección", 1);
Team team2 = new Team("FC Barcelona", 1);
Team team3 = new Team("Atlético de Madrid", 2);

teamRepository.save(team1);
teamRepository.save(team2);
teamRepository.save(team3);

Player p1 = new Player("Torres", 10);
Player p2 = new Player("Iniesta", 10);

p1.getTeams().add(team1);
p1.getTeams().add(team3);

p2.getTeams().add(team1);
p2.getTeams().add(team2);

playerRepository.save(p1);
playerRepository.save(p2);
```

Generación de tablas desde entidades

- **Herencia entre clases**
  - Existen varias estrategias



Java Persistence API (JPA)

code URJC

- Spring Data
  - http://projects.spring.io/spring-data/
- Spring Data Commons
  - http://docs.spring.io/spring-data/commons/docs/current/reference/html/
- Spring Data JPA
  - http://docs.spring.io/spring-data/jpa/docs/current/reference/html/
- Bases de datos SQL en Spring Boot
  - http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-sql.html
- Inicialización de bases de datos SQL en Spring Boot
  - http://docs.spring.io/spring-boot/docs/current/reference/html/howto-database-initialization.html
- Tutorial Spring Data JPA
  - https://spring.io/guides/gs/accessing-data-jpa/

Bases de datos SQL en Spring

code URJC

- Bases de datos SQL en Spring
- Java Persistence API (JPA)
- **Consultas**
- Paginación
- Gestión del esquema
- Arquitectura
- Imágenes
- Accesos concurrentes

Consultas

code URJC

- Estrategias para especificación de consultas con SpringData:
  - **Métodos en los repositorios**
  - Java Persistence Query Language (JPQL)
  - QueryDSL

Consultas

code URJC

- **Métodos en los repositorios**
  - En base al nombre del método y el tipo de retorno se construye la consulta a la BBDD

ejem13

```
public interface PostRepository extends JpaRepository<Post, Long> {

    List<Post> findByUsername(String username);

    List<Post> findByTitle(String title);
}
```

Consultas con un único campo como criterio

http://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.query-methods.details

Consultas

- Métodos en los repositorios

```
public interface PersonRepository extends Repository<Person, Long> {

        List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);

        // Enables the distinct flag for the query
        List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);
        List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);

        // Enabling ignoring case for an individual property
        List<Person> findByLastnameIgnoreCase(String lastname);

        // Enabling ignoring case for all suitable properties
        List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);

        // Enabling static ORDER BY for a query
        List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
        List<Person> findByLastnameOrderByFirstnameDesc(String lastname);

}
```

Consultas combinando varios campos

Ordenación, IgnoreCase, Order...

Consultas

- **Métodos en los repositorios**
  - **Combinación lógica:** And, Or
  - **Comparadores:** Between, LessThan, GreatherThan
  - **Modificadores:** IgnoreCase
  - **Ordenación:** OrderBy…Asc / OrderBy…Desc

Consultas

code URJC

- **Métodos en los repositorios**
  - **Consulta**
    - List<Elem> find…By…(…)
    - List<Elem> read…By…(…)
    - List<Elem> query…By…(…)
    - List<Elem> get…By…(…)
  - **Número de elementos**
    - int count…By…(…)

Consultas

- **Métodos en los repositorios**
  - Propiedades de los objetos relacionados
    - No sólo podemos filtrar por una propiedad de la entidad, también podemos filtrar por un **atributo de otra entidad** con la que esté relacionada
      - **Person** con un atributo **address**
      - **Address** tiene un atributo **zipCode**

```
List<Person> findByAddressZipCode(ZipCode zipCode);
```

```
List<Person> findByAddress_ZipCode(ZipCode zipCode);
```

Consultas

- **Métodos en los repositorios**
  - Ordenación
    - Podemos pasar un parámetro Sort que controla la ordenación
    - Existe el método findAll(Sort sort)

```
repository.findAll(Sort.by("nombre")));
repository.findAll(Sort.by(Order.asc("nombre"))));
```

    - Podemos añadir métodos personalizados

```
List<User> findByLastname(String lastname, Sort sort);
```

Consultas

- **Métodos en los repositorios**
  - Ordenación
    - El objeto Sort se puede crear desde la URL

```
http://server/anuncios/?sort=nombre,desc
```

```
@GetMapping("/posts/")
public List<Post> getPosts(Sort sort) {

  return posts.findAll(sort);

}
```

Consultas

- **Métodos en los repositorios**
  - Limitar los resultados

```
User findFirstBy...();

User findTopBy...();

User findTopDistinctBy...();

List<User> queryFirst10By...();

List<User> findTop3By...();

List<User> findFirst10By...();
```

Consultas

- Estrategias para especificación de consultas con SpringData:
  - Métodos en los repositorios
  - **Java Persistence Query Language (JPQL)**
  - QueryDSL

Consultas

- **Java Persistence Query Language (JPQL)**
  - JPQL es un lenguaje similar a SQL pero referencia conceptos de las entidades JPA

ejem14

```
public interface TeamRepository extends JpaRepository<Team, Long> {

    @Query("select t from Team t where t.name = ?1")
    List<Team> findByName(String name);
}
```

NOTA: En este ejemplo la query JPQL no sería necesaria porque el nombre del método define justo esa consulta

Consultas

- **Java Persistence Query Language (JPQL)**
  - Estructura de sentencia SELECT

```
SELECT ... FROM ...
[WHERE ...]
[GROUP BY ... [HAVING ...]]
[ORDER BY ...]
```

  - Actualización y borrado

```
DELETE FROM ... [WHERE ...]

UPDATE ... SET ... [WHERE ...]
```

Consultas

- **Java Persistence Query Language (JPQL)**
  - Ejemplos

```
SELECT e from Employee e
WHERE e.salary BETWEEN 30000 AND 40000
```

```
SELECT e from Employee e WHERE e.name LIKE 'M%'
```

```
SELECT DISTINCT b FROM Blog b JOIN b.comments c
WHERE c.author=?1
```

Consultas

- Java Persistence Query Language (JPQL)

ejem15

```
public interface PostRepository extends JpaRepository<Post, Long> {

  @Query("SELECT DISTINCT p FROM Post p JOIN p.comments c WHERE c.username=?1")
  List<Post> findByCommentsUser(String user);

}
```

NOTA: La query JPQL no sería necesaria porque el método del repositorio ejecuta justo esa consulta

Consultas

- **Structured Query Language (SQL)**
  - Incluso podemos usar SQL nativo directamente

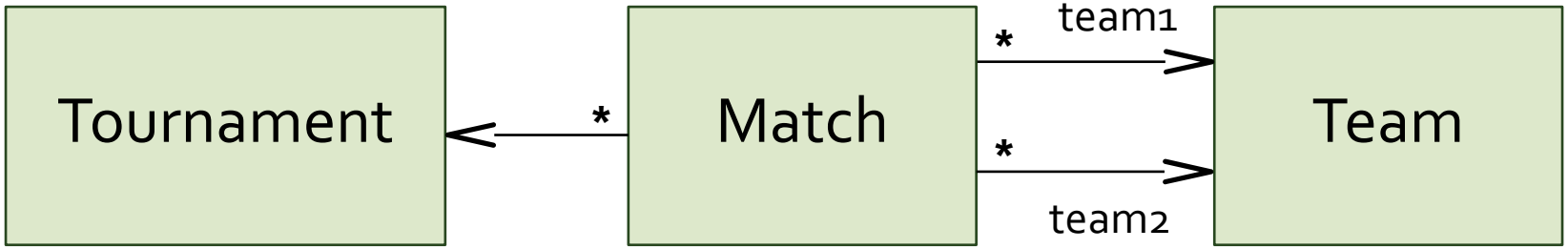
```
public interface UserRepository extends JpaRepository<User, Long> {

  @Query(
     value = "SELECT * FROM USERS WHERE EMAIL_ADDRESS = ?1",
     nativeQuery = true)
  User findByEmailAddress(String emailAddress);

}
```

Consultas

code URJC

ejem16

- Las consultas permiten relacionar varias entidades sin necesidad de tener atributos directos
  - **Modelo**



  - ¿Qué equipos juegan en un torneo?
  - ¿En qué torneos juega un equipo?

Consultas

code URJC

ejem16

```java
@Entity
public class Tournament {

  @Id
  @GeneratedValue(...)
  private long id;

  private String data;

}
```

```java
@Entity
public class Team {

  @Id
  @GeneratedValue(...)
  private long id;

  private String data;

}
```

```java
@Entity
public class Match {

  @Id
  @GeneratedValue(...)
  private long id;

  private String data;

  @ManyToOne
  private Team team1;

  @ManyToOne
  private Team team2;

  @ManyToOne
  Tournament tournament;
}
```

Consultas

code URJC

ejem16

```
public interface MatchRepository extends JpaRepository<Match, Long> {

  @Query("SELECT m FROM Match m WHERE m.tournament = :t")
  public List<Match> getMatches(Tournament t);

}
```

```
public interface TeamRepository extends JpaRepository<Team, Long> {

  @Query("SELECT distinct team FROM Match m, Team team "
    + "WHERE (m.team1 = team OR m.team2 = team) AND  m.tournament = :t")
  public List<Team> getTeams(Tournament t);

}
```

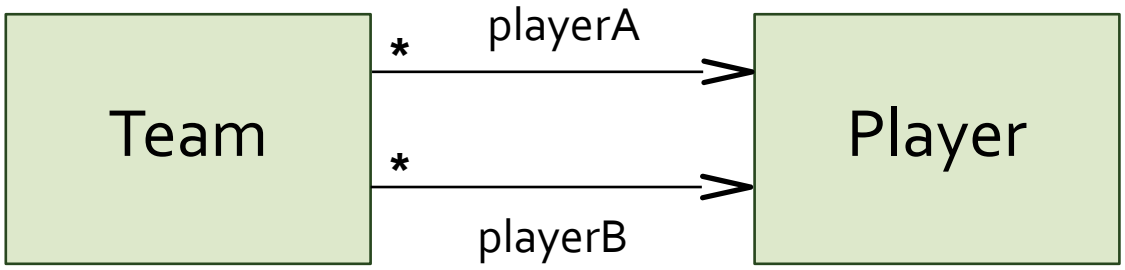
```
public interface TournamentRepository extends JpaRepository<Tournament, Long> {

  @Query("SELECT distinct t FROM Match m JOIN m.tournament t "
    + "WHERE m.team1 = :team OR m.team2 = :team")
  public List<Tournament> getTournaments(Team team);

}
```

Consultas

code URJC

ejem16b

- Ejemplo de consultas avanzadas
  - ¿Qué jugadores juegan conmigo?




```
public interface PlayerRepository extends JpaRepository<Player, Long> {

  @Query("""
    select distinct u from Player u, Team t
    where (t.playerA = u and t.playerB = :player) or (t.playerB = u and t.playerA = :player)
  """)
  List<Player> findPairsOf(Player player);
}
```

Consultas

code URJC

- Estrategias para especificación de consultas con SpringData:
  - Métodos en los repositorios
  - Java Persistence Query Language (JPQL)
  - **QueryDSL**

Consultas

code URJC

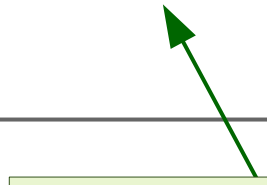
- **QueryDSL**
  - Permite implementar las consultas usando código Java
  - A diferencia de JPQL, el **compilador** avisa de **errores** en la consulta
  - Favorece la **refactorización** y el **autocompletado**
  - Se puede usar con **SpringData**

http://www.querydsl.com/

Consultas

- **QueryDSL**
  - Partiendo de las entidades se genera código que sirve para hacer las consultas
  - Uso en SpringData

```
public interface TodoRepository extends
    Repository<Todo, Long>, QuerydslPredicateExecutor<Todo> {

}
```



Añadimos otro interfaz padre al repositorio

Consultas

- **QueryDSL**
  - La consulta se define Partiendo de la clase "Q" generada automáticamente
  - Se define el predicado (filtro)
  - Se usa el método findAll(...) del repositorio

```
@GetMapping("/")
public Iterable<Todo> todos() {

    Predicate q = QTodo.todo.title.eq("Foo");

    return repository.findAll(page);

}
```

Consultas

code URJC

- QueryDSL

Con un static import de Qtodo.* las sentencias son más concisas

```
todo.title.eq("Foo").and(todo.description.eq("Bar"));
```

```
todo.title.eq("Foo").or(todo.title.eq("Bar"));
```

```
todo.title.eq("Foo").and(todo.description.eq("Bar").not());
```

```
todo.description.containsIgnoreCase(searchTerm)
    .or(todo.title.containsIgnoreCase(searchTerm));
```

Consultas

code URJC

- QueryDSL

ejem17

```
@GetMapping("/")
public Iterable<Post> getPosts(
  @RequestParam(required = false) String commentsUser) {

  if(commentsUser == null) {
    return posts.findAll();
  } else {
    return posts.findAll(post.comments.any().username.eq(commentsUser));
  }
}
```

Consultas

code URJC

- QueryDSL
  - Dependencias

ejem17

```
<dependencies>
  ...
  <dependency>
    <groupId>com.querydsl</groupId>
    <artifactId>querydsl-core</artifactId>
  </dependency>
  <dependency>
    <groupId>com.querydsl</groupId>
    <artifactId>querydsl-jpa</artifactId>
  </dependency>
</dependencies>
```

Consultas

- QueryDSL

ejem17

```
<plugin>
  <groupId>com.mysema.maven</groupId>
  <artifactId>apt-maven-plugin</artifactId>
  <version>1.1.3</version>
  <dependencies>
    <dependency>
      <groupId>com.querydsl</groupId>
      <artifactId>querydsl-apt</artifactId>
      <version>4.4.0</version>
    </dependency>
  </dependencies>
  <executions>
    <execution>
      <goals>
        <goal>process</goal>
      </goals>
      <configuration>
        <outputDirectory>target/generated-sources</outputDirectory>
        <processor>com.querydsl.apt.jpa.JPAAnnotationProcessor</processor>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Tarea para generar las clases partiendo de las entidades

Bases de datos SQL en Spring

code URJC

- Bases de datos SQL en Spring
- Java Persistence API (JPA)
- Consultas
- **Paginación**
- Gestión del esquema
- Arquitectura
- Imágenes
- Accesos concurrentes

Paginación

- Con SpringData es muy sencillo que los resultados de las consultas se devuelvan **paginados**
- Basta poner el parámetro **Pageable** en el método del controlador y pasar ese parámetro al método del repositorio
- Cambiar el result de **List**<Element> a **Page**<Element>
- Una Page contiene la información de una **página** (info, nº pág, nº total elementos)
- Se integra con Spring MVC para las **webs** y **APIs REST**

Paginación

ejem18

Devolvemos un objeto **Page<Anuncio>**

En nuestros métodos podemos añadir un parámetro **Pageable**

```
public interface PostRepository extends JpaRepository<Post, Long> {

    Page<Post> findByUsername(String username, Pageable page);

    Page<Post> findByTitle(String title, Pageable page);
}
```

```
Page<Post> a = repository.findByUsername("pepe", PageRequest.of(0, 50));

Page<Post> a = repository.findAll(PageRequest.of(3, 20));
```

El método findAll también tiene versión **Paginable**

Creamos un objeto **PageRequest** para indicar en num página y tamaño

Paginación

code URJC

ejem18

- En el controlador, podemos crear directamente el objeto Pageable como parámetro

```
@RestController
@RequestMapping("/posts")
public class PostController {

  @Autowired
  private PostRepository posts;

  @GetMapping("/")
  public Page<Post> getPosts(
    @RequestParam(required = false) String user, Pageable page) {

    if (user != null) {
      return posts.findByUsername(user, page);
    } else {
      return posts.findAll(page);
    }
  }
}
```

Paginación

code URJC

ejem18

- Con la URL habitual se devuelve la **primera página** con **20 elementos**

```
http://server/posts/
```

- En las peticiones se pueden incluir **parámetros** para solicitar cualquier página

```
http://server/posts/?page=1&size=3
```

- Válido para **webs** y para **APIs REST**

Paginación

- En las APIs REST, el JSON de respuesta incluye información de paginación

Paginación

- La serialización por defecto en SpringBoot > 1 no es muy adecuada, se puede implementar un conversión a JSON personalizada

```
{
    "content": [
        {
            "id": 1,
            "username": "Pepe",
            "title": "Hola caracola",
            "text": "XXXX"
        },
        {
            "id": 2,
            "username": "Juan",
            "title": "Hola caracola",
            "text": "XXXX"
        }
    ],
    "first": true,
    "last": true,
    "totalPages": 1,
    "totalElements": 2,
    "numberOfElements": 2,
    "size": 20,
    "number": 0,
    "sort": []
}
```

code URJC

ejem19

```
@JsonComponent
public class PageImplJacksonSerializer extends JsonSerializer<PageImpl<?>> {

	@SuppressWarnings("rawtypes")
	@Override
	public void serialize(PageImpl page, JsonGenerator jsonGenerator, SerializerProvider serializerProvider)
			throws IOException {

		jsonGenerator.writeStartObject();

		jsonGenerator.writeFieldName("content");
		serializerProvider.defaultSerializeValue(page.getContent(), jsonGenerator);

		jsonGenerator.writeBooleanField("first", page.isFirst());
		jsonGenerator.writeBooleanField("last", page.isLast());
		jsonGenerator.writeNumberField("totalPages", page.getTotalPages());
		jsonGenerator.writeNumberField("totalElements", page.getTotalElements());
		jsonGenerator.writeNumberField("numberOfElements", page.getNumberOfElements());

		jsonGenerator.writeNumberField("size", page.getSize());
		jsonGenerator.writeNumberField("number", page.getNumber());

		Sort sort = page.getSort();

		jsonGenerator.writeArrayFieldStart("sort");

		for (Sort.Order order : sort) {
			jsonGenerator.writeStartObject();
			jsonGenerator.writeStringField("property", order.getProperty());
			jsonGenerator.writeStringField("direction", order.getDirection().name());
			jsonGenerator.writeBooleanField("ignoreCase", order.isIgnoreCase());
			jsonGenerator.writeStringField("nullHandling", order.getNullHandling().name());
			jsonGenerator.writeEndObject();
		}

		jsonGenerator.writeEndArray();
		jsonGenerator.writeEndObject();
	}
}
```

https://www.wimdeblauwe.com/blog/2018/2018-06-10-pageimpl-json-serialization-with-spring-boot-2/

Consultas

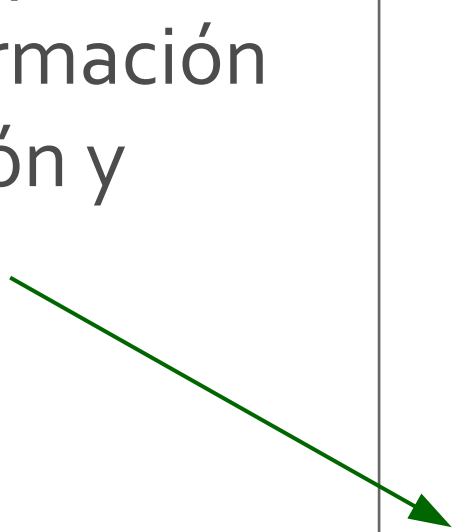
- **Métodos en los repositorios**
  - Ordenación
    - El objeto Pageable incluye la información de ordenación de la URL

```
http://server/posts/?page=1&size=3&sort=username,desc
```

```
@GetMapping("/posts/")
public Page<Post> getPost(Pageable page) {

  return posts.findAll(page);

}
```

Consultas

- En las APIs REST, el JSON de respuesta incluye información de paginación y ordenación



```
{
    "content": [
        {
            "id": 1,
            "username": "Pepe",
            "title": "Hola caracola",
            "text": "XXXX"
        },
        {
            "id": 2,
            "username": "Juan",
            "title": "Hola caracola",
            "text": "XXXX"
        }
    ],
    "first": true,
    "last": true,
    "totalPages": 1,
    "totalElements": 2,
    "numberOfElements": 2,
    "size": 3,
    "number": 0,
    "sort": [
        {
            "property": "username",
            "direction": "DESC",
            "ignoreCase": false,
            "nullHandling": "NATIVE"
        }
    ]
}
```

Ejercicio 2

- Añade paginación a la página web de gestión de **posts**, en la página principal

Bases de datos SQL en Spring

code URJC

- Bases de datos SQL en Spring
- Java Persistence API (JPA)
- Consultas
- Paginación
- **Gestión del esquema**
- Estructura de una aplicación con BBDD
- Imágenes
- Accesos concurrentes

Gestión del esquema

- En las **Bases de datos SQL** es necesario **crear el esquema** antes de insertar los datos (crear tablas, relaciones, etc…)
- En las bases de **datos en memoria** (H2, Derby, HSQL…), el esquema siempre se construye de forma **automática** al iniciar la aplicación
- Cuando se usa una base de datos en producción los datos tienen que guardarse en disco y gestionar adecuadamente la **creación y actualización del esquema**

MySQL

code URJC

ejem20

- Instalación en ubuntu
- Servidor

```
$ sudo apt-get install mysql-server
```

- Herramienta interactiva

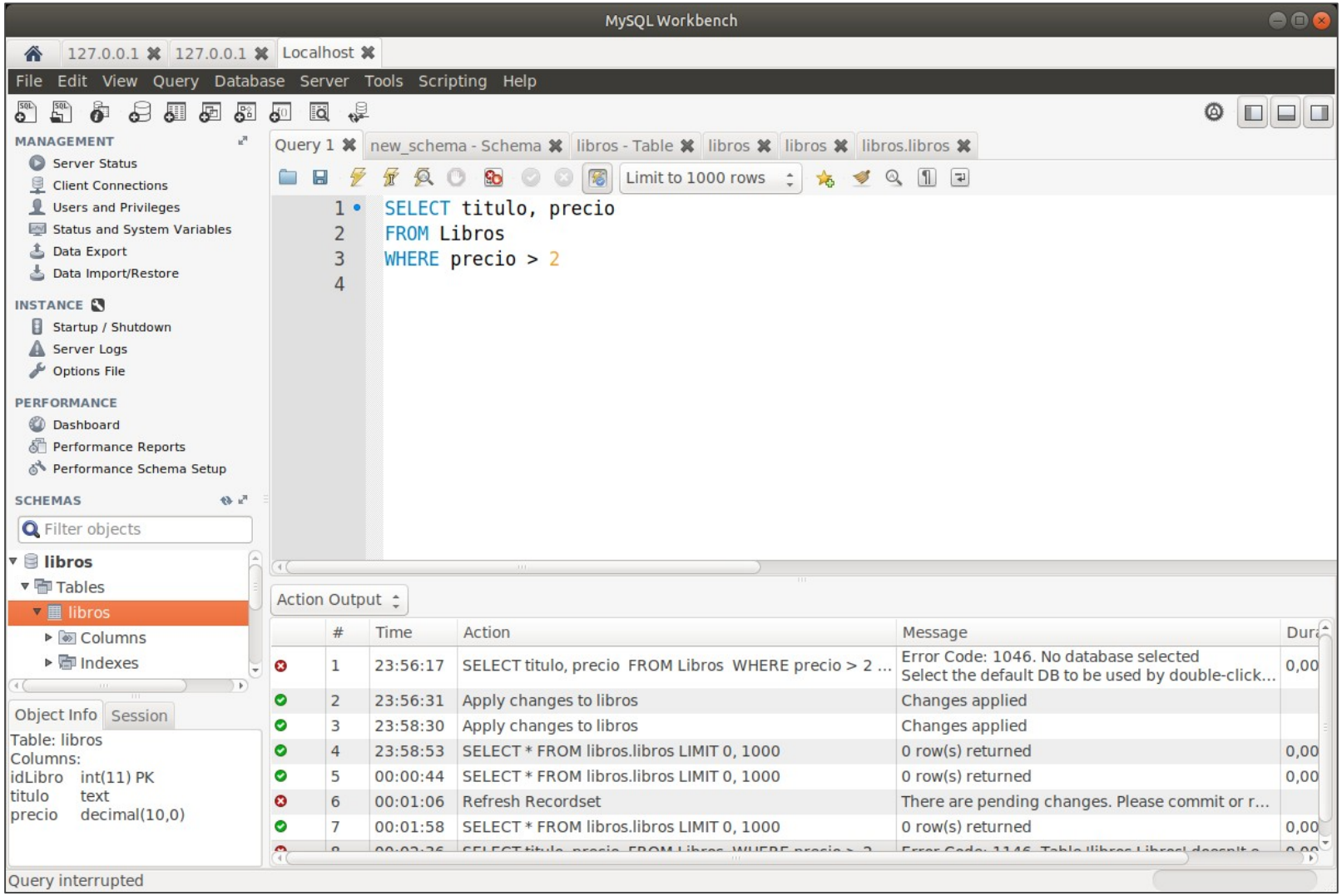
```
$ sudo apt-get install mysql-workbench
```

- Docker

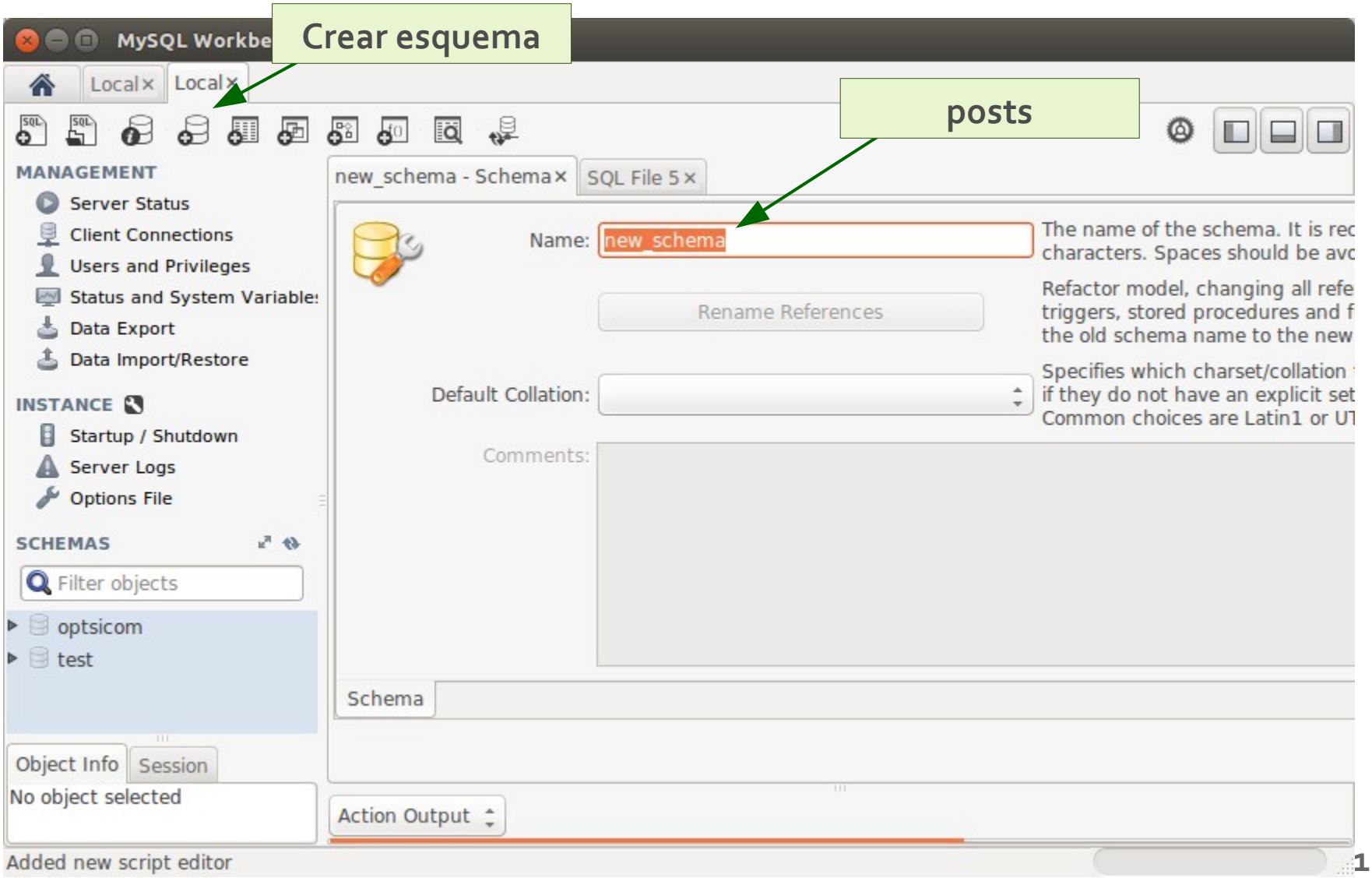
```
$ docker run --rm -e MYSQL_ROOT_PASSWORD=password \
  -e  MYSQL_DATABASE=posts -p 3306:3306 -d mysql:8.0.22
```

http://dev.mysql.com/downloads/

MySQL


MySQL Workbench
127.0.0.1
127.0.0.1
Localhost
File Edit View Query Database Server Tools Scripting Help
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
SCHEMAS
Filter objects
libros
Tables
libros
Columns
Indexes
Object Info
Session
Table: libros
Columns:
idLibro int(11) PK
titulo text
precio decimal(10,0)
Query 1
new_schema - Schema
libros - Table
libros
libros
libros.libros
Limit to 1000 rows
1 SELECT titulo, precio
2 FROM Libros
3 WHERE precio > 2
4
Action Output
Time Action Message
1 23:56:17 SELECT titulo, precio FROM Libros WHERE precio > 2 ... Error Code: 1046. No database selected
Select the default DB to be used by double-click...
2 23:56:31 Apply changes to libros Changes applied
3 23:58:30 Apply changes to libros Changes applied
4 23:58:53 SELECT * FROM libros.libros LIMIT 0, 1000 0 row(s) returned
5 00:00:44 SELECT * FROM libros.libros LIMIT 0, 1000 0 row(s) returned
6 00:01:06 Refresh Recordset There are pending changes. Please commit or r...
7 00:01:58 SELECT * FROM libros.libros LIMIT 0, 1000 0 row(s) returned
Query interrupted

MySQL


Crear esquema
posts
MySQL Workbe
Local ×
Local ×
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variable
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
SCHEMAS
Filter objects
optsicom
test
Object Info
Session
No object selected
new_schema - Schema ×
SQL File 5 ×
Name:
new_schema
Rename References
Default Collation:
Comments:
Schema
Action Output
Added new script editor

MySQL

ejem20

- pom.xml

```
<groupId>es.codeurjc</groupId>
<artifactId>bbdd_ejem20</artifactId>
<version>0.1.0</version>

...
<dependencies>
  ...
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
  </dependency>
</dependencies>
```

Dependencia a **Spring Data** para BBDD relacionales

Dependencia al driver **MySQL**

MySQL

ejem20

- Datos de conexión a la BBDD

Este fichero tiene que estar en el fichero **src/main/resources**

Host en el que está alojado el servidor MySQL

Nombre del **esquema**. Este esquema tiene que ser creado manualmente por el desarrollador usando herramientas de MySQL

application.properties

```
spring.datasource.url=jdbc:mysql://localhost/posts
spring.datasource.username=root
spring.datasource.password=pass
spring.jpa.hibernate.ddl-auto=...
```

Usuario y contraseña. En algunos SO estos datos se configuran al instalar MySQL

Gestión del esquema

code URJC

ejem20

- **Propiedad de generación del esquema**

```
spring.jpa.hibernate.ddl-auto=...
```

- **create-drop:** Crea el esquema al iniciar la aplicación y le borra al finalizar (ideal para programar como si la BBDD estuviera en memoria)
- **create:** Crea el esquema al iniciar la aplicación
- **update:** Añade al esquema actual las tablas y atributos necesarios para hacer el esquema compatible con las clases Java (no borra ningún elemento). Si el esquema está vacío, se genera completo.
- **validate:** Verifica que el esquema de la BBDD es compatible con las entidades de la aplicación y si no lo es genera un error.
- **none:** No hace nada con el esquema y asume que es correcto.

Gestión del esquema

- **Estrategia de desarrollo con BBDD**
  - **Fase 1:** Desarrollo
  - **Fase 2:** Despliegue
  - **Fase 3:** Ejecución
  - **Fase 4:** Actualización de versión

Gestión del esquema

- **Estrategia de desarrollo con BBDD**
  - **Fase 1: Desarrollo: ddl-auto=create-drop**
    - Cualquier cambio en las entidades se refleja de forma automática en la BBDD al arrancar la aplicación
    - Se suele trabajar con datos inciales de desarrollo para que reiniciar la aplicación no requiera la introducción manual de datos

Gestión del esquema

- **Estrategia de desarrollo con BBDD**
  - **Fase 2: Despliegue: ddl-auto=update**
    - Al iniciar la aplicación por primera vez se genera el esquema.
    - Al iniciar la aplicación por segunda vez el esquema no se modifica (porque la app no ha cambiado)
    - Si se quiere evitar el tiempo extra que requiere comprobar el esquema, se puede configurar la gestión del esquema a "none"

Gestión del esquema

- **Estrategia de desarrollo con BBDD**
  - **Fase 3: Ejecución: ddl-auto=none**
    - El esquema no debe modificarse en las siguientes ejecuciones de la aplicación

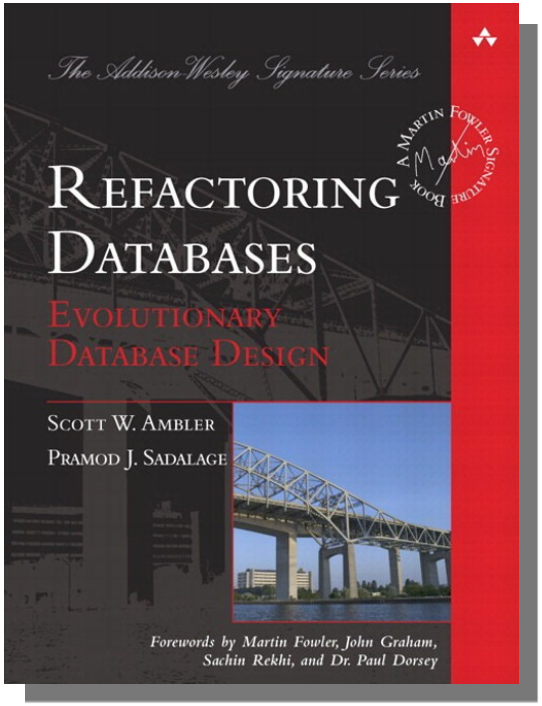
Gestión del esquema

- **Estrategia de desarrollo con BBDD**
  - **Fase 4: Actualización: Usar herramientas específicas**
    - Si una nueva versión de la aplicación requiere cambios en las tablas es necesario **aplicar los cambios pero sin alterar los datos (migración del esquema)** antes de ejecutar la nueva versión
    - Para ello se utilizan herramientas específicas

Gestión del esquema

- **Migración del esquema de la BBDD**
  - Cuando una aplicación con BBDD se usa en **producción** se **guardan datos** en la misma que se deben mantener
  - Es posible que la aplicación **evolucione**:
    - Incluyendo nuevas entidades
    - Modificando las entidades existentes
  - Para **actualizar** la aplicación en **producción**, antes hay que **migrar** la BBDD al nuevo esquema

Gestión del esquema



Refactoring Databases: Evolutionary Database Design

by Scott W. Ambler and Pramod J. Sadalage

Addison Wesley Professional (2006)

http://www.ambysoft.com/books/refactoringDatabases.html

Gestión del esquema

- Existen dos herramientas externas que facilitan la evolución del esquema:

http://flywaydb.org/

Increase reliability of deployments by versioning your database

http://www.liquibase.org/

Fast database change. Fluid delivery

Gestión del esquema

- **Flyway y Liquibase** mantienen una **lista de cambios** que hay que hacer en el esquema para **evolucionar/migrar** de una versión a otra superior

- Cuando la aplicación **Spring** se **inicia**, se pueden ejecutar de forma **automática estos cambios** para convertir el esquema en la versión deseada

https://www.baeldung.com/database-migrations-with-flyway

https://thorben-janssen.com/database-migration-with-liquibase-getting-started

Bases de datos SQL en Spring

code URJC

- Bases de datos SQL en Spring
- Java Persistence API (JPA)
- Consultas
- Paginación
- Gestión del esquema
- **Arquitectura**
- Imágenes
- Accesos concurrentes

Arquitectura

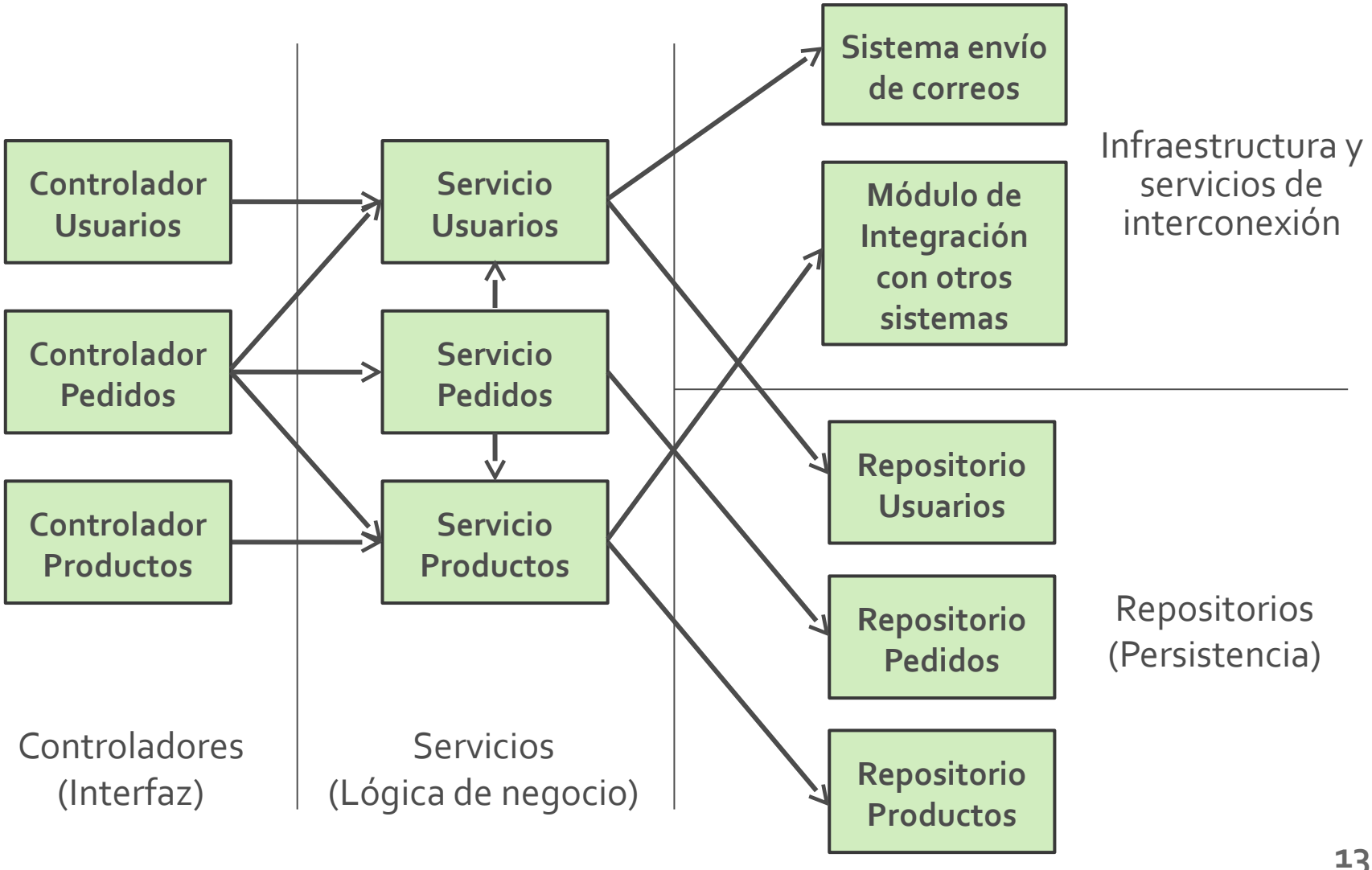
- La **arquitectura de software** define, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación entre ellos.
- Existen diferentes **estilos arquitectónicos** que pueden ser adecuados para diferentes tipos de aplicaciones

Arquitectura

code URJC

- Las aplicaciones Spring con BBDD suelen usar una arquitectura en la que las clases se agrupan por responsabilidades:
  - **Controladores:** Gestión de API REST
  - **Entidades:** Elementos básicos de información que se persisten en la BD. Modelo.
  - **Repositorios:** Guardar y consultar entidades
  - **Servicios:** Lógica de negocio que gestiona entidades
  - **Infraestructura:** Envío de mails
  - **Servicios de interconexión**

Arquitectura



Arquitectura

- **Implementación**
  - **Controladores:** @Controller o @RestController
  - **Entidades:** @Entity
  - **Repositorios:** implements JpaRepository
  - **Servicios:** @Service
  - **Infraestructura:** @Service
  - **Servicios de interconexión:** @Service

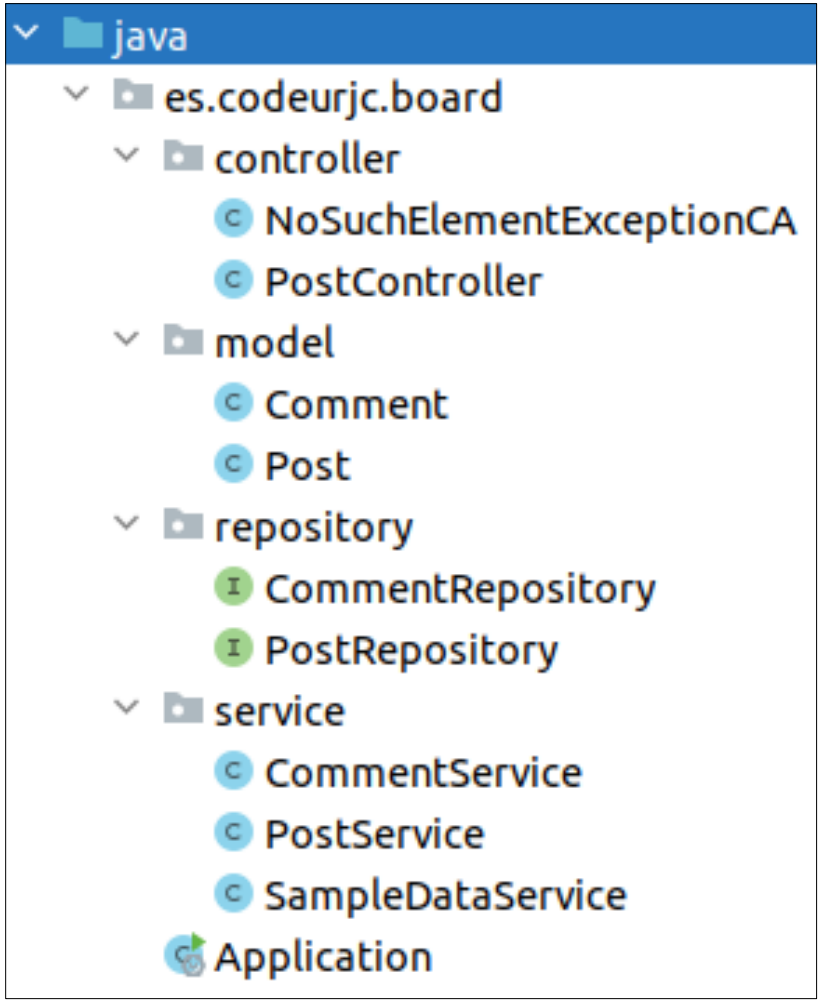
Arquitectura

- **Ventajas de esta arquitectura:**
  - Los **servicios** pueden testearse de forma unitaria (con dobles de repositorios y otros servicios)
  - Los **controladores** pueden testearse de forma unitaria (con doble del servidor web y de los servicios)
  - La **lógica de negocio** se puede reutilizar entre varios tipos de controladores (web, REST, línea de comandos…)

Arquitectura

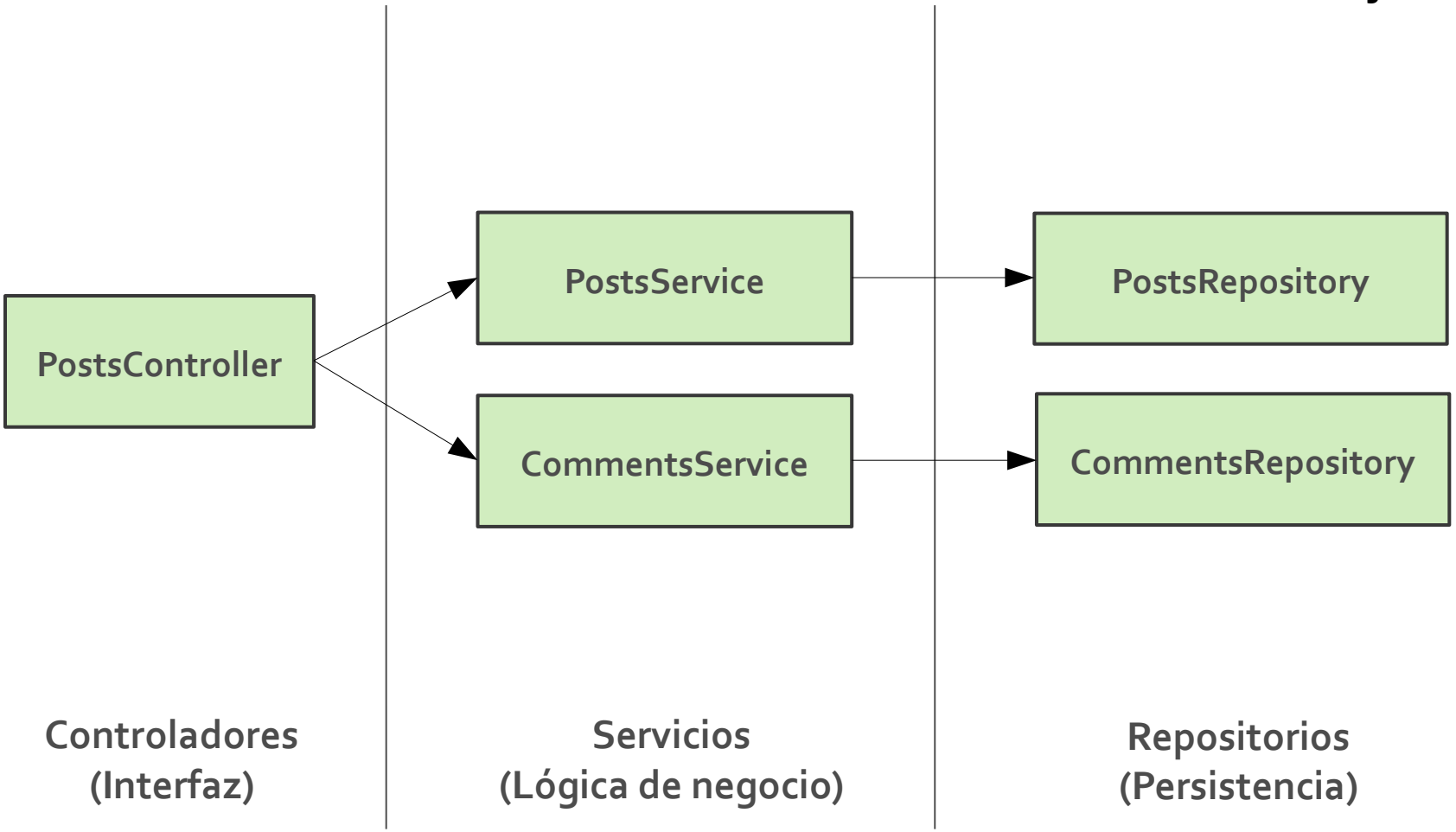
code URJC

ejem21



Arquitectura

ejem21



Bases de datos SQL en Spring

code URJC

- Bases de datos SQL en Spring
- Java Persistence API (JPA)
- Consultas
- Paginación
- Gestión del esquema
- Arquitectura
- **Imágenes**
- Accesos concurrentes

Imágenes

- En vez de guardar las **imágenes** en disco, se pueden **guardar en la BD**
- Eso favorece la **escalabilidad** porque las aplicaciones son ***stateless*** (no tienen estado) y se pueden replicar en **diferentes máquinas**
- Simplifica el despliegue en entornos en los que **no se tiene acceso al disco duro** de forma persistente (Heroku, Kubernetes…)
- Otra opción es usar **servicios de persistencia de ficheros** (S3, Minio…)

Imágenes

code URJC

ejem22

- Atributo Blob (*Binary Large Object*)

```
@Entity
public class Post {

    @Id
    @GeneratedValue(...)
    private long id;

    private String username;
    private String title;
    private String text;
    private String image;

    @Lob
    @JsonIgnore
    private Blob imageFile;


    ...
}
```

Imágenes

code URJC

ejem22

- Upload Image

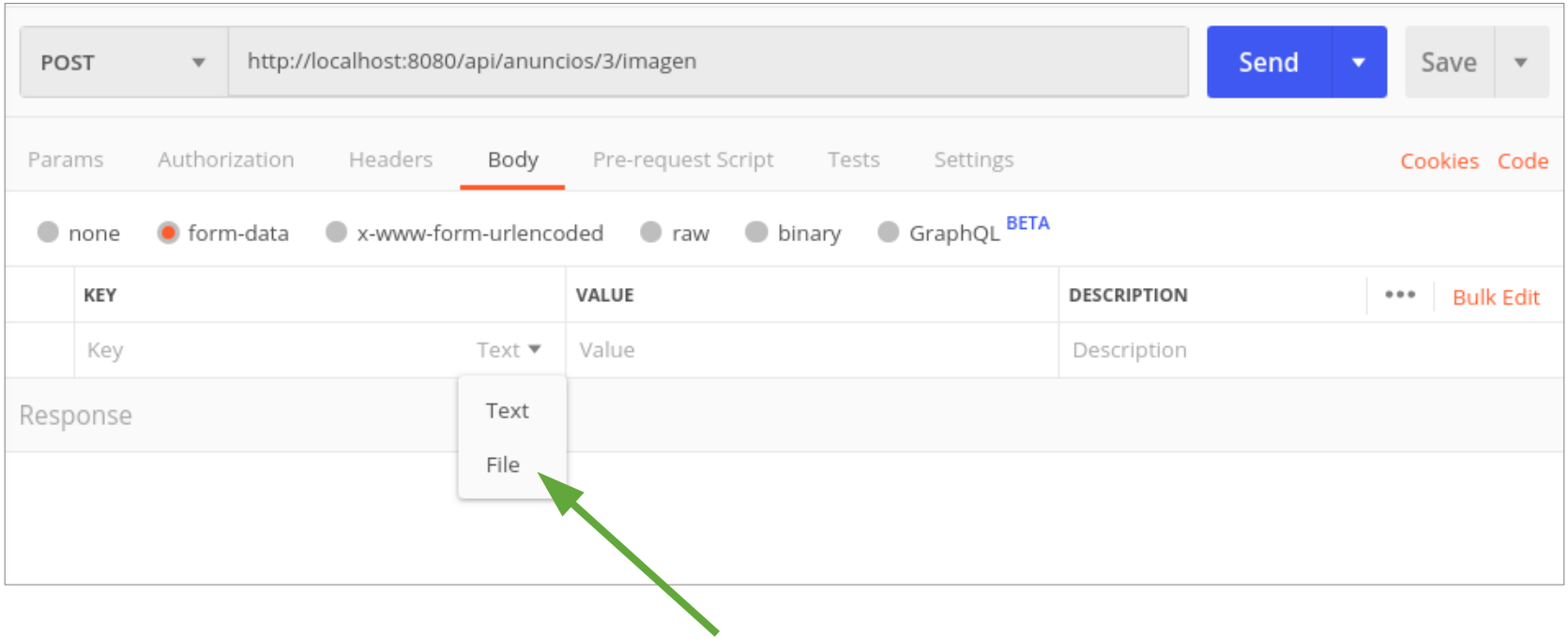
```
@PostMapping("/{id}/image")
public ResponseEntity<Object> uploadImage(@PathVariable long id,
  @RequestParam MultipartFile imageFile) throws IOException {

  Post post = posts.findById(id).orElseThrow();

  URI location = fromCurrentRequest().build().toUri();

  post.setImage(location.toString());

  post.setImageFile(BlobProxy.generateProxy(
    imageFile.getInputStream(), imageFile.getSize()));

  posts.save(post);

  return ResponseEntity.created(location).build();
}
```

Imágenes

code URJC

ejem22

- Subir un fichero con Postman

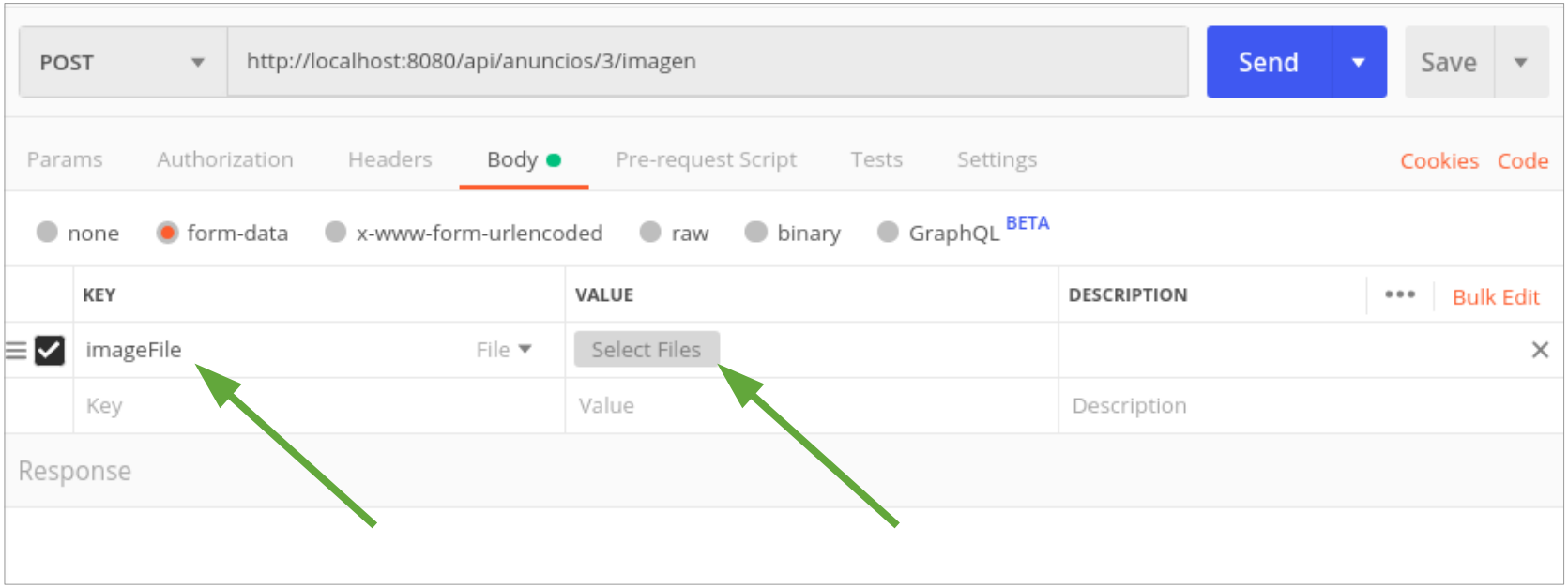


Imágenes

code URJC

ejem22

- Subir un fichero con Postman



Imágenes

code URJC

ejem22

- Download Image

```
@GetMapping("/{id}/image")
public ResponseEntity<Object> downloadImage(@PathVariable long id)
  throws SQLException {

  Post post = posts.findById(id).orElseThrow();

  if (post.getImageFile() != null) {

    Resource file = new InputStreamResource(
      post.getImageFile().getBinaryStream());

    return ResponseEntity.ok()
      .header(HttpHeaders.CONTENT_TYPE, "image/jpeg")
      .contentLength(post.getImageFile().length())
      .body(file);

  } else {
    return ResponseEntity.notFound().build();
  }
}
```

Imágenes

code URJC

ejem22

- Delete Image

```
@DeleteMapping("/{id}/image")
public ResponseEntity<Object> deleteImage(@PathVariable long id)
  throws IOException {

  Post post = posts.findById(id).orElseThrow();

  post.setImageFile(null);
  post.setImage(null);

  posts.save(post);

  return ResponseEntity.noContent().build();
}
```

Bases de datos SQL en Spring

code URJC

- Bases de datos SQL en Spring
- Java Persistence API (JPA)
- Consultas
- Paginación
- Gestión del esquema
- Arquitectura
- Imágenes
- **Accesos concurrentes**

Accesos concurrentes

- Es importante gestionar de forma adecuada qué ocurre cuando varios usuarios interactúan con una aplicación con base de datos **concurrentemente**
- Cuando los usuarios únicamente consultan información **no hay interferencias entre ellos**
- Cuando un usuario (o los dos) escriben, se pueden producir **interferencias no deseadas**

Accesos concurrentes

code URJC

- **Ejemplos de situaciones problemáticas:**
  - Dos usuarios consultan un recurso desde la API REST a la misma vez.
    - Uno de ellos modifica una propiedad y reemplaza el recurso.
    - El otro modifica otra propiedad y también reemplaza el recurso borrando los cambios de la modificación previa (*lost updates*)
  - Dos usuarios deciden comprar la última entrada de un concierto a la misma vez

Accesos concurrentes

- **Existen varias técnicas que nos ayudan a gestionar estas situaciones**
  - Transacciones de la base de datos
  - Cabeceras ETag e If-Match en la API REST
  - Sentencias SQL que verifican precondiciones

Accesos concurrentes

- **Transacciones de la base de datos**
  - Varias sentencias se ejecutan dentro de una unidad lógica
  - Los cambios de todas ellas se aplican o se descartan. No es posible aplicar cambios parciales

https://docs.oracle.com/javase/tutorial/jdbc/basics/transactions.html

https://docs.spring.io/spring-framework/docs/current/reference/html/data-access.html#transaction

Accesos concurrentes

code URJC

- **Transacciones de la base de datos**
  - El grado de aislamiendo de las transacciones es configurable (*Isolation level*)
  - Afecta a los datos que se pueden leer desde una transacción de otras transacciones que se ejecutan de forma concurrente.

Transacciones de las base de datos

- **Isolation level:**
  - **READ_UNCOMMITED:** Lectura de valores escritos por otras transacciones que todavía no han finalizado (y podrían abortarse finalmente) (*Dirty reads*)
  - **READ_COMMITED:** Si durante la transacción se lee la misma fila dos veces y esa fila es modificada por otra transacción entre medias se leen valores diferentes (*Non-repeatable reads*)
  - **REPEATABLE_READ:** Si se hace una consulta y se obtienen unas filas y otra transacción añade nuevas filas y se vuelve a hacer la misma consulta, aparecen filas nuevas (*Phantom reads)*
  - ***SERIALIZABLE:*** Las transacciones están aisladas entre sí. Su comportamiento es como si se ejecutaran secuencialmente.

Accesos concurrentes

- **Cabeceras ETag e If-Match en la API REST**
  - **Problemática (*lost updates*):**
    - Dos usuarios consultan un recurso desde la API REST a la misma vez.
    - Uno de ellos modifica una propiedad y reemplaza el recurso.
    - El otro modifica otra propiedad y también reemplaza el recurso (borrando los cambios de la modificacion previa)

https://www.baeldung.com/etags-for-rest-with-spring
https://dzone.com/articles/concurrency-control-in-rest-api-with-spring-framew

Accesos concurrentes

- **Cabeceras ETag e If-Match en la API REST**
  - **Solución: *Optimistic locking***
    - Cada recurso tiene asociada una versión
    - Cuando se lee un recurso se recibe la versión (**cabecera ETag**)
    - Cuando se quiere modificar el recurso (PUT o PATH) se tiene que enviar la versión recibida previamente (**cabecera If-Match**)
    - Si el recurso no ha sido modificado y sigue en esa versión, se acepta el cambio. Si no, error.

Accesos concurrentes

- **Sentencias SQL que verifican precondiciones**
  - **Problemática:**
    - Dos usuarios deciden comprar la última entrada de un concierto a la misma vez
    - Sólo uno de ellos debería poder comprar la entrada

Accesos concurrentes

- **Sentencias SQL que verifican precondiciones**
  - **Solución:**
    - Se podría usar un nivel de aislamiento de las transacciones SERIALIZABLE. Eso implica que la primera transacción consigue la entrada y la segunda detecta que ya no hay disponibles.
    - También se pueden ejecutar sentencias SQL atómicas que tengan la precondición de que queden plazas libres (más eficiente)

Accesos concurrentes

- Sentencias SQL que verifican precondiciones

```
@Transactional
@Modifying
@Query("UPDATE Event e SET e.tickets=e.tickets+1 " +
       "WHERE e.id = :id AND e.tickets+1 <= e.max_capacity")
public int reserveTicket(@Param("id") long id);
```

```
int rowsUpdated = eventRepository.reserveTicket(eventId);
if (rowsUpdated == 1) {
    //Ticket buy
} else {
    //Event full. No ticket
}
```