
DATUM

# El día de ayer vimos:

- Lectura de properties
- PathParam
- QueryParam
- Método Get

# Agenda del día

- Método Post y Put
- Conexión a base de datos
- Entidades con JPA
- Repository del Proyecto
- Realización de consultas nativas (Update y Delete)

# Método Post

## Patrón Post

```
.post("/post")
                .type(ModeloBody.class)
                .outType(RespuestaServicio.class)
                .param().name("modelo").type(RestParamType.body).description("Objecto del modelo").dataType("ModeloBody").endParam()
                .to("direct:saludar");
```

# Método Put

## Patrón Put

```
.put("/put")
                .type(ModeloBody.class)
                .outType(RespuestaServicio.class)
                .param().name("modelo").type(RestParamType.body).description("Objecto del modelo de empleado").dataType("ModeloBody").endParam()
                .to("direct:saludar");
```

# Script Base de datos

```
-- Crear Columna adicional en la tabla Empleados

ALTER TABLE Empleados

ADD COLUMN direccion VARCHAR(255);
```

# Conexión Base de datos

## Variables Globales:

```
spring.datasource.url=${DATASOURCE_URL}
spring.datasource.username=${DATASOURCE_USER}
spring.datasource.password=${DATASOURCE_PWD}
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.max-active=3
spring.datasource.min-active=1
spring.datasource.max-idle=3
spring.datasource.min-idle=1
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.show-sql=false
```

# Dependencias

## Spring Boot Starter Data JPA

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

## MySQL Connector/J

```
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
</dependency>
```

# Entidad

Una entidad representa una tabla almacenada en una base de datos. Cada instancia de una entidad representa una fila en la tabla. Las entidades persistentes se identifican mediante la anotación @Entity.

# Table

Esta anotación nos permite decidir contra que tabla de la base de datos nuestra entidad se va a mapear . En muchas situaciones la entidad no tiene el mismo nombre exacto de la base de datos y nos viene bien usar esta anotación.

# Column

Nos permitirá definir aspectos muy importantes sobre las columnas de la base de datos de la base de datos como lo es el nombre. En caso de no definir esta anotación en los atributos, JPA determinara el nombre de la columna de forma automática mediante el nombre del atributo.

# Objeto de Acceso de Datos

Es una interfaz que proporciona métodos abstractos para acceder y manipular datos en una fuente de datos, como una base de datos. Un DAO generalmente incluye sentencias nativas SQL y, a veces, operaciones adicionales relacionadas con la persistencia de datos.

# Repositorio

**CrudRepository**: Se trata de un repositorio genérico. Esto significa que contiene métodos aplicables a cualquier clase del dominio, pues esos métodos tienen un tipado genérico, el mismo que Repository.

# Repositorio

**JpaRepository**: extiende de CrudRepository e incluye algunas características adicionales específicas de JPA (Java Persistence API), que es una especificación estándar de Java para la gestión de la persistencia de datos.

# Query

Hay situaciones en las que necesitamos métodos muy específicos y nos puede venir bien usar la anotación de @Query que nos permite diseñar el método a medida a través de JPA.

```
@Query(value="UPDATE EMPLEADOS SET DIRECCION = :direccion WHERE IDENTIFICADOR = :id ", nativeQuery=true)
int updateEmpleado(@Param("id") Long id, @Param("direccion") String direccion);
```

# Param

La vinculación entre marcador y parámetro es configurable con la anotación @Param, en Spring Boot si obvias @Param cuando marcador y parámetro compartan el nombre, puede producir el siguiente error: org.springframework.dao.InvalidDataAccessApiUsageException

```
@Query(value="UPDATE EMPLEADOS SET DIRECCION = :direccion WHERE IDENTIFICADOR = :id ", nativeQuery=true)
int updateEmpleado(@Param("id") Long id, @Param("direccion") String direccion);
```

# Query

Al igual que utilizamos la anotación Query para realizar consultas SQL nativas, está anotación nos brinda la facilidad por medio de JPA de realizar consultas utilizando la entidad y sus atributos como valores para la consulta.

```
@Query(value="UPDATE Empleados emp SET emp.direccion = :direccion WHERE emp.identificador = :id ")
int updateEmpleado(@Param("id") Long id, @Param("direccion") String direccion);
```

# Log

El patrón log en Camel se implementa a través del uso de SLF4J (Simple Logging Facade for Java), que es una interfaz de registro común para Java. SLF4J no realiza directamente la escritura de registros, sino que proporciona una interfaz para diferentes sistemas de registro, como Log4j, Logback y Java Util Logging, entre otros.

DATUM

# Gracias por la Atención