Spring Data JPA

Wrapper for JPA

- Remplaza capa repository (si no se sobreescribe nada)

Elimina boiler plate (por ejemplo merge y persist)

Puede ser extendido para funcionalidad adicional

Se agrega la lib spring-jpa
Se agrega jpa:repositories → como el scan

} En JPA Context

Delete impl de repository

Interfaz Intigue extendemos JpaRepository

anotamos la interfaz de la misma manera de como estaba nuestra implementacion que borramos

Esto va nos da todos los metodos.

save
update
find

~~No soporta projection~~

solo Query

Si tenemos una busqueda a BD customizada podemos dejar ese metodo en la interfaz que extendimos de JpaRepository y lo marcamos con @Query y pasamos lo que estaba como named Query aqui.

Spring Data Course (lo que no existe Antonio) ①

Mejor spring con hibernate

DSL (Domain Specific Language)

Reduce Data Access Layer → simplificar

Generic Repositories

Audita Entidades

Tipos data access layer

JDBC
Spring JDBC → Simple DB
Native Sql
Reports

High number of inserts → Spring batch
hadoop

Data Graphs or relaciones excesivas → ORM's
Spring Data JPA
JPA - hibernate

Topic o Relacional
Things → No sql
Spring Data Mongo

Ayuda dependencia

spring-data-jpa

JPA Repositories donde estan las interfaces para repos

@ EnableJPARepositories

Cada repositorio representa una entidad

Cuando usamos jpa repos tenemos lo siguiente

- Query DSL
- CRUD Operation
- Paging Sorting
- Helpers
  - count
  - exist
  - Flush
  - deleteInBatch(Iterable Items

Jerarquia

Repository
↑
CRUD Repository
↑
PagingAndSorting Repository
↑
JPA Repository

Cuando auditamos columnas como por auditoria, similar a como lo haciamos en funcion publica, en donde cada tabla tenia las columnas creadoPor, modificadoPor, etc., podemos crear una superclase que tenga todos esos atributos y que todas las entidades hereden de esta superclase

---

Del lado de la capa repositorio, si tenemos metodos comunes, deberiamos de crear una interfaz que extienda de los repositorios de JPA y de ahi implementarla en nuestra JPA Repo

Tip

Si tenemos muchos objetos y no queremos modificarlos, podemos usar un proxy

---

Flush → toma todos los cambios que tenga y los persiste antes de que la transaccion termine

Esto proxy

Se pone en el repositorio

Tips

repository → nombre package

sufijo JpaRepository

marcar JPARepository con @Repository

Poner todos los repositorios en un solo paquete es bueno porque JPARepositories donde especificamos donde escanea por nuestro repositorio, solo deberiamos tener uno.

Query DSL

DSL → crea queries en tiempo de inicio de la aplicacion y no en tiempo de ejecucion

Logica basica el query parse

find	read	get
findBy	readBy	getBy
query	count	
queryBy	countBy	

criterios por medio de attributos names

Multiple criteria con And o Or

Como en JDA Repository este atado a una entidad, son los tipos de datos que podemos recibir definidos en la entidad, estos pueden ser

objetos
listas
primitivos

And Or

findByStateOrCountry(String state, String country)

findByStateAndCountry(String state, String country)

De esta manera
tienen default equals
es como si dijeramos
findByStateEquals where state = ? and country = ?

Equals
Is
Not

findByStateIs("CA") — a = b
findByStateEquals("CA") a = b
findByStateNot("CA") a <> b

no es igual a:

Like NotLike

findByStateLike("%A") > NOSOTROS SOMOS RESPONSABLES DE PONER EL COMODIN
("CA%")

StartingWith, EndingWith, Containin

Es igual que like pero no tenemos que especificar el comodin %, seria como lo siguiente

StartingWith = al%
EndingWith → %.state
containing = %.al%.

LessThan, GreatoThen

< >

Son para numeros

LessTha Equals

por ejemplo para un rango

findByAgeGreatoTheAndAgeLessTha(18, 21)

Befor, Afto, Between

Befor fecha menor que
Afto fecha despues o mayor que
Between Entre rango de fecha

* No son incluyentes de ningun lado, un workaround seria que pusieramos una fecha con un dia antes.

True False

cuando buscamos valores booleanos.
findActiveTrue
FindActiveFalse

IsNull
NotNull > similares y producen la misma salida.

IsNotNull

In Not In

Cuando necesitamos validar si una columna
un valor de una columna es parte de
una colección o un set de valores

findStateIn(Coleccion)
findStateNotIn(Coleccion)

IgnoreCase → reemplaza Upper

Ej. findByStateIgnoreCase("ca")
puede causar overhead hacer esto si tenemos un
dataset muy grande

OrderBy : Asc
Desc

Al final del query

findByStateOrderByStateAsc
Desc

First, Top y Distinct

findFirstByStateLike("%E") → limit 1
findTop5ByStateLike("A") → limit 5
findDistinctManufacturerByStateLike("A")
↓
distinct

distinct remueve duplicados
no es distinto a lo que le

@Query annotation

Tenemos que anotar el metodo con @Query
y a partir de ahi el Query, pueden ser
parametros nombrados

Ej: Select m from Model m where m.name
= :name

Los parametros deben matchear con los tipo
y numeros de parametros definidos.

Ej
@Query
Select m.name from Model m where
m.name = :name and
m.age = :edad"

public void queryByModelAndAge (String nombre,
int edad

En el metodo tenemos
que pasarlo por medio de
@Param("lowest") String nom
@Param("highest") Int age

Si deben matchear
con los parametros
del query

No deben
llamarse igual
que los parametros

Esta es alternativa si no queremos usar JPA
DSL que los nombres serian muy largos,
podemos usar esto, al final queda algo como
esto.
@Query (Select m from Model where m.name=:name"
public void queryBySomething (@Param("name") String nombre
{
...
}

Además del método anterior que tiene parámetros nombrados, podemos también parámetros basados en índices

?1 ?2 ...

Por qué usar @Query a lugar de DSL

- Por completitud
- Para tener control sobre fetch
- Para reutilizar si es que ya tenemos una capa de data repository y solo reutilizar y reutilizar los queries que ya funcionan

---

@Query Options

En un query podemos poner el % en un like dentro del query, de la siguiente manera

@Query("select m from model m where m.name like %?1")
List<Model> queryByName(String name);

- Por razones de Performance podemos utilizar queries nativos, especificando nativeQuery = true como sigue

~~@Query("select *~~

@Query(value="select * from Model", nativeQuery = true)

Al hacer esto nos estamos casando a la base de datos específica pero podemos optimizar nuestros queries

@Modifying
@Query (update Model m set m.name = ?1"

↓

Extiende las capacidades de crud operations y nos permite modificar especificamente por @Query

---

JPA Named Queries

stet up not inline

↓ Entity

@NamedQueries (name = "Model . someName",
query = "Select m .... ") m.name

↓

property name Model entity

Lo mismo podemos hacer con @Query por medio de su parametro name

---

Native Queries tambien los podemos hacer por medio de

@NamedNativeQuery (name "Entity.name)

Generamos un Named Query para el nombre de la entidad

Entidad.nombreNamed Query y esto en la entidad misma y se referencia con el nombre largo, se puede pasar al repositorio y como está ligado a una entidad y no tendríamos que pasar el nombre largo

---

Precedencia de JpaRepository

1 @Query → tiene la mayor precedencia
2 Métodos que matchen un named query por "nombre" o un native query por "nombre"
3 Métodos que sigan las reglas DSL

Paging y Sorting

Return type debe ser Page
como parámetro agregar Pageable y sort si se quiere hacer sort

* El pasar los valores, es responsabilidad del UI

Custom Repository

Creamos un interfaz que extiend de JPA Rep. la implementamos debe llamarse igual q la interfaz padre pero con Suffix impl

MyJPARepository ⟶ MyJPARepositoryCust

MyJPARepositoryImpl

Debe de ser igual, lo nombre, solo cambia lo ultimo y se agrega Impl

Si queremos cambiar este comportamiento default por medio de xml sería algo como esto

```
JPA:repository base-package="com.repositorios"
                repository-impl-postfix="CustomCha
```

Auditing

@CreatedBy
@CreatedDate
@LastModifiedBy
@LastModifiedDate

← Se ponen en la clase entidad

Auto set

JPA: auditing → en xml

@EnableJpaAuditing

Auditable<Autor <User>>

↓

interfaz

Locking

① Version

Optimistic → Si el número de versión no hace match vanza una excepción OptimisticLockException.

Pesimistic → Locks de larga duración, bloquea la información por la duración de la transacción previniendo que otra acceda a la información hasta que la transacción haya commit

② Lock(LockModeType.PESSIMISTIC_WRITE)
List<Matri> findBy MatriTypeNombre(
List<String> tipos