Spring framework 5 - Jpa y Spring data

ORM

La traducción del modelo relacional a objetos Java se le conoce como **ORM**. A través de **Jpa** (**Java Persistence Api**) un desarrollador puede guardar, obtener, actualizar y mapear información en una base de datos a través de objetos de Java.

Java Persistence Api

Jpa es una especificación que contiene múltiples implementaciones, las más populares son:

- Hibernate
- EclipseLink
- TopLink

Entity

Una entidad es una clase cuyos objetos serán persistidos en una base de datos, se creará una entidad para representar una tabla, cada instancia de la clase será un registro de la base de datos, a continuación algunas anotaciones importantes:

- **@Entity**: Se aplica en clases e indica que la clase será una entidad
- **@Table**: Se aplica en clases e indica el nombre de la tabla que se va a mapear
- **@Id**: Se aplica a atributos e indica que el campo será la llave primaria
- **@GeneratedValue**: Se aplica en llaves primarias e indica la forma en la que se generarán
- **@Column**: Se aplica en atributos e indica el nombre de la columna que representará

Relaciones

Jpa permite realizar las siguientes relaciones entre entidades:

- **@OneToOne**
- **@ManyToOne**
- **@ManyToMany**

@OneToOne

Ejemplo de **@OneToOne** :

```
@Entity
@Table(name = "profile")
public class Profile {
	@Id
	@GeneratedValue(strategy =
GenerationType.IDENTITY)
	@Column(name = "id")
	private Integer id;

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

	@Column(name = "lastname")
	private String lastName;
	//Getters y setters
}
@Entity
@Table(name = "user")
public class User {
	@Id
	@GeneratedValue(strategy =
GenerationType.IDENTITY)
	@Column(name = "id")
	private Integer id;

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

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


}
```

@ManyToOne

Ejemplo de **@ManyToOne**:

```
@Entity
@Table(name = "profile")
public class Profile {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "id")
	private Integer id;

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

	@Column(name = "lastname")
	private String lastName;

	@Column(name = "birthdate")
	@Temporal(TemporalType.DATE)
	private Date birthDate;

	@OneToOne
	 @JoinColumn(name ="user_id",
	referencedColumnName ="id")
	 private User user;
		//Getters y setters
}
@Entity
@Table(name = "address")
public class Address {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "id")
	private Integer id;

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

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

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

	@ManyToOne
	private Profile profile;
	//Getters y setters
}
```

@ManyToMany

Ejemplo de **@ManyToMany**:

```
@Entity
@Table(name = "role")
public class Role {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "id")
	private Integer id;

	@Column(name = "name")
	private String name;
	//Getters y setters

}
@Entity
@Table(name = "user")
public class User {
	//Mismo código al ejemplo anterior
}
@Entity
public class UserInRole {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Integer id;

	@ManyToOne
	@JoinColumn(name = "user_id")
	private User user;

	@ManyToOne
	@JoinColumn(name = "role_id")
	private Role role;
	//Getters y setters
}
```

Configuración

Para poder utilizar **Spring data** en un proyecto de **Spring boot** se debe incluir la siguiente dependencia:

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

A demás de la dependencia de **Spring data** se debe incluir la dependencia del driver jdbc, para este ejemplo se utilizará **H2**:

```
<dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>
</dependency>
```

application.properties

Para definir las configuraciones de la base de datos se deben agregar las siguientes líneas al archivo **application**.properties:

```
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.url=jdbc:h2:mem:testdb
```

Spring data

Spring Data permite tomar el patron de diseño **Dao(Data Access Object)** y simplificarlo hasta el punto de remover sus implementaciones completamente.

Al heredar de una interfaz definida por **Spring**, se obtendrá la funcionalidad básica para trabajar con una base de datos.

Tipos de repositorios

En **Spring data** el principal componente es el **Repository** y las sus principales interfaces son:

- **CrudRepository**
- **PagingAndSortingRepository**
- **JpaRepository**

Al crear una interfaz que herede de ellas se tendrá una implementación de un **DAO** (**Data Access Object**) con los métodos básicos para implementar **CRUD**(Create, read, update y delete)