Acceso a datos

ORM – Mapeo objeto-relacional

JPA – Proveedores JPA – Anotaciones JPA fundamentales

IES Clara del Rey – Madrid

JPA

Jakarta Persistence API. Antes Java Persistence API.

Es una especificación, como JDBC, para definir un estándar para el acceso a bases de datos desde Java, pero a través de ORM.

JDBC es de bajo nivel de abstracción, trabaja con tablas, filas, columnas, procedimientos almacenados, etc.

JPA es de alto nivel, establece una serie de convenciones, anotaciones, y técnicas para definir el mapeo entre los objetos de la aplicación y las tablas, columnas, procedimientos, etc.

Aunque puede haber ORM que no se adhieran al estándar definido por JPA, no es habitual. A los fabricantes de ORM les interesa cumplir los estándares.

JPA – Proveedores

Los ORM compatibles con JPA (que implementan la especificación JPA) más utilizados son:

- Hibernate. Posiblemente el más utilizado, y el que se usará en los proyectos del curso.
- EclipseLink. Además, es implementación de JAXB (exportación XML).
- Apache OpenJPA.

Hay otras, y también hay ORM que no siguen la especificación JPA.

Hibernate, es anterior a JPA, pero una vez que se estableció el estándar, se adaptó para funcionar de dos formas, como proveedor JPA y con su propia configuración.

JPA – Entidades

Son clases del dominio de la aplicación, que se usan como representación de los datos (tablas) en la base de datos.

Son POJO (Plain Old Java Object). Una entidad debe cumplir lo siguiente:

- Tener al menos un constructor sin argumentos con acceso public o protected. Puede tener otros constructores.
- Ser una clase, no puede ser un enum o una interfaz.
- No debe ser final. Tampoco pueden serlo sus atributos, ni sus métodos.
- Suele usarse una entidad por cada tabla de la base de datos, aunque hay excepciones.
- Cada objeto de la clase será una fila de la tabla.

JPA – Entidades

Aparte de los requisitos, hay que tener en cuenta que:

- Pueden ser tanto clases abstractas como concretas.
- Pueden formar jerarquía de herencia y asociación / composición:
  - En la herencia, pueden heredar de otra clase que también sea una entidad o de una que no lo sea.
  - En la asociación y composición, estas relaciones suelen estar asociadas a claves ajenas en las tablas de la BD.
- Como con todas las clases Java, los atributos son privados, y se accederá a ellos con getters y setters

JPA – Entidades – Estado de entidades

Una entidad puede estar en uno de estos tres estados:

- Transitorio (transient): Objeto creado, pero que no se ha guardado. El ORM no es consciente de la existencia del objeto.
- Persistente (persistent): Ya se ha guardado le objeto en la BD. El ORM hace seguimiento del objeto.
- Desasociado (detached): Objeto que era persistente, pero tras cerrarse la conexión se pierde este estado, porque el ORM abandona el seguimiento del objeto. Para poder asociar un objeto desasociado, debe ser serializable.

Los estados no son muy relevantes cuando se trabaja con Spring Data JPA, pero hay que tenerlos más en cuenta cuando se trabaja con JPA "puro".

JPA – Anotaciones

En JPA se usan anotaciones para mapear las entidades (clases) Java a registros (tablas y filas) de la base de datos.

Al marcar una clase con anotaciones específicas, como @Entity, @Table, @Column, entre otras, se define la forma en que la clase y sus atributos se deben almacenar y recuperar en la base de datos.

Actúan como directrices para el proveedor de persistencia (como, por ejemplo, Hibernate) sobre cómo realizar el mapeo objeto-relacional.

Permiten configurar la persistencia de objetos Java en bases de datos de manera sencilla y declarativa, basada en convenciones.

Evitan escribir consultas SQL manualmente.

JPA – Anotaciones fundamentales

Aunque hay más, las más utilizadas son:

- @Entity: Define una clase como entidad persistente.
- @Table: Especifica detalles de la tabla para mapear la entidad en la BD.
- @Id: Marca un atributo como clave primaria de la entidad.
- @GeneratedValue: Configura la generación automática de valores para un campo de la BD.
- @Column: Define atributos de la columna de BD asociada a un atributo.
- @Basic: Configura opciones básicas de mapeo para un atributo.
- @Transient: Excluye un campo de ser persistido en la BD.
- @Temporal: Especifica el tipo temporal para un campo de fecha.

JPA – Anotaciones fundamentales

Las utilizadas para definir relaciones entre entidades son:

- @ManyToOne: Establece una relación de muchos a uno.
- @OneToMany: Define una relación de uno a muchos.
- @ManyToMany: Define una relación de muchos a muchos.
- @OneToOne: Configura una relación de uno a uno.

Estas anotaciones se complementan con:

- @JoinColumn: Personaliza la columna de unión para una relación.
- @JoinTable: Define una tabla de unión para relaciones many-to-many.

JPA – Anotaciones fundamentales

Otras anotaciones útiles:

- @Enumerated: Indica cómo almacenar enums en la BD.

Y mecanismos para definir entidades con claves compuestas (más de una columna), usando dos opciones:

- @IdClass / @Id
- @Embeddable y @EmbeddedId

JPA – Anotación @Entity

Se utiliza para marcar una clase como una entidad persistente.

```
@Entity
public class Usuario {
    // campos, constructores, métodos, etc.
}
```

Las clases con la anotación @Entity debe tener una clave primaria. Las claves deben tener también un atributo @Id para definir la clave, o definir una clave compuesta.

Una entidad en JPA representa (generalmente) una tabla o vista en la base de datos.

Cada instancia, cada objeto de la clase anotada con @Entity corresponde a una fila en la tabla de la base de datos.

JPA – Anotación @Table

Permite añadir detalles sobre la tabla de la BD que se va a asociar con la entidad. El atributo (parámetro) más importante de esta anotación es "name", que permite cambiar el nombre de la tabla de la BD.

```
@Entity
@Table(name = "usuarios")
public class Usuario {
    // campos, constructores, métodos, etc.
}
```

En este ejemplo las entidades de la clase "Usuario" se almacenan en la tabla de nombre "usuarios"

JPA – Anotación @Id

Se usa para marcar un atributo como la clave primaria de la entidad. Se coloca sobre el atributo que actuará como clave primaria.

Es obligatorio indicar la clave primaria en las entidades, así que siempre que usemos @Entity, tendremos que usar @Id en uno de sus atributos.

```java
@Entity
public class Producto {
    @Id
    private Long id;

    // Otros campos y métodos de la entidad
}
```

En el caso de claves compuestas (multicolumna), que no son las más habituales, pueden usarse las anotaciones @IdClass y @EmbeddedId.

JPA – Anotación @GeneratedValue

Se utiliza para especificar cómo se generan automáticamente los valores de un campo de la BD.

```java
@Entity
public class Producto {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // Otros campos y métodos de la entidad
}
```

Es muy habitual usarlo en las claves primarias de tipo numérico, que se suelen generar con secuencias o con columnas identity.

JPA – Anotación @GeneratedValue

El parámetro "strategy" define la estrategia de generación de valores:

- GenerationType.AUTO: Lo decide el proveedor de persistencia.
- GenerationType.IDENTITY: Usa columnas identity. No todos los SGBD ni todas las versiones de estos sistemas tienen columnas identity.
- GenerationType.SEQUENCE: Usa una secuencia. Requiere otras anotaciones / parámetros adicionales para definir la secuencia.
- GenerationType.TABLE: Usa una tabla en la que previamente se han cargado valores a utilizar como id (simula secuencias). También requiere otras anotaciones.

Cuando se usan secuencias o tablas se usa @SecuenceGenerator o @TableGenerator y el parámetro "generator" de @GeneratedValue.

JPA – Anotación @Column

Se utiliza para personalizar los atributos de una columna de la BD.

Algunos atributos/parámetros de @Column son:

- name: Define el nombre de la columna en la base de datos.
- nullable: Especifica si puede contener valores nulos (true por defecto).
- unique: Indica si los valores deben ser únicos (false por defecto).
- length: Define la longitud máxima de la columna (tipo String).
- precision / scale: Precisión y la escala en tipos flotantes (con capacidad para almacenar decimales)

JPA – Anotación @Column

```java
@Entity
public class Producto {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "nombre_producto", nullable = false, length = 100)
    private String nombre;

    @Column(name = "precio", precision = 10, scale = 2)
    private BigDecimal precio;

    // Otros campos y métodos de la entidad
}
```

JPA – Anotación @Basic

JPA realiza un mapeo básico por defecto de los atributos de una entidad.

Para personalizar este mapeo por defecto se puede usar @Basic.

Los parámetros más importantes de esta anotación son

- optional: Indica si el atributo puede ser nulo (true por defecto).
- fetch: Define la estrategia de recuperación de datos (EAGER o LAZY). Si se usa LAZY no se lee la columna de la base de datos hasta que se accede a ella. Esto es útil, por ejemplo, cuando se tienen campos muy grandes (BLOBS, por ejemplo) en una columna.
- cascade: Define operaciones de cascada que se deben aplicar a la entidad asociada.

JPA – Anotación @Basic

```java
@Entity
public class Usuario {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Basic(optional = false)
    private String nombre;

    @Basic(optional = false)
    private String apellidos;

    @Basic(fetch = FetchType.LAZY)
    private Blob imagen;

    // Constructores, getters, setters y otros métodos
}
```

JPA – Anotación @Transient

Se utiliza para marcar un atributo como no persistente. El atributo no se guardará en la BD, ni se recuperará un valor de la BD al leer objetos.

Útil para no guardar información temporal, calculada, o para no guardar información sensible, como contraseñas generadas.

```java
@Entity
public class Usuario {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // Otros campos, constructores, getters, setters, etc.

    @Transient
    private String password;
}
```

JPA – Anotación @Temporal

Se utiliza para indicar cómo se debe mapear el campo de fecha en la base de datos. Puede tener los siguientes valores:

- TemporalType.DATE: Campos de fecha sin información de hora.
- TemporalType.TIME: Campos de tiempo sin información de fecha.
- TemporalType.TIMESTAMP: Campos de fecha y hora.

```
@Temporal(TemporalType.DATE)
private Date fechaNacimientoDate;

@Temporal(TemporalType.TIME)
private Calendar horaNacimientoCalendar;

private LocalDateTime fechaUltimaModificacion;
```