Java Persistence API

- Sesión 3: Mapeo entidad-relación (tablas)

Índice

- Acceso al estado de la entidad
- Mapeo de entidades y columnas
- Mapeo de tipos
- Mapeo de la clave primaria
- Objetos embebidos
- Herencia

Acceso al estado de la entidad

- El proveedor de persistencia debe poder acceder al estado de la entidad para rellenarla con los datos de la BD o leerla para actualizar la BD
- Cuando definimos una podemos colocar la anotación `@Id` en los atributos (variables de instancias) o en las propiedades (getters y setters)
- Dependiendo de ello el proveedor de persistencia utilizará los atributos o las propiedades para acceder al estado de la entidad

Acceso por campo

- Los campos deben declararse private o protected
- El proveedor no utiliza los getters y setters, utiliza reflexión para modificar las instancias

```
@Entity
public class Empleado {
   @Id
   private int id;
   private long salario;

   public Empleado {}
   public Empleado(int id) { this.id = id; }

   public int getId() { return id; }
   public String getNombre() { return nombre; }
   public long getSalario() { return salario; }
   public void setSalario(long salario) { this.salario = salario; }
}
```

Acceso por propiedad

- La anotación se define en el método `getId()`
- El proveedor utiliza los getters y setters para actualizar la entidad y para definir el nombre de sus campos (y columnas en la BD)
- Notar que es distinto el nombre del atributo `sueldo` del de la propiedad `getSalario()`; el nombre de la columna en la BD sería `salario`

```
@Entity
public class Empleado {
   private int id;
   private long sueldo;

   public Empleado {}
   public Empleado(int id) { this.id = id; }

   @Id public int getId() { return id; }
   public String getNombre() { return nombre; }
   public long getSalario() { return salario; }
   public void setSalario(long salario) { this.sueldo = salario; }
}
```

Mapeo de entidades

- Por defecto una entidad se mapea con una tabla en la base de datos con el mismo nombre
- El esquema de la base de datos se define en la unidad de persistencia
- Es posible modificar estos valores por defecto con la anotación `@Table` y los elementos `name` y `schema`

```
@Entity
@Table(name="EMP", schema="IT")
public class Empleado { ... }
```

Mapeo de columnas

- Por defecto un atributo de una entidad se mapea en una columna con el mismo nombre
- Es posible modificar esto con la anotación `@Column` y el elemento `name`

```
@Entity
public class Empleado {
    @Id
    @Column(name="EMP_ID")
    private int id;
    private String nombre;
    @Column(name=SAL)
    private long salario;
    // ...
}
```

Mapeo de tipos

- JPA realiza un mapeo automático de los tipos Java de las entidades a tipos SQL
- El tipo SQL asociado con el tipo Java viene definido por el proveedor de persistencia, pudiendo variar de un gestor de base de datos a otro
- La especificación JPA define una lista de tipos que podemos usar en los atributos de las entidades

Posibles tipos Java

- **Tipos primitivos Java**: `byte`, `int`, `short`, `long`, `boolean`, `char`, `float`, `double`
- **Clases *wrapper* de los tipos primitivos**: `Byte`, `Integer`, `Short`, `Long`, `Boolean`, `Character`, `Float`, `Double`
- **Arrays de bytes y char**: `byte[]`, `Byte[]`, `char[]`, `Character[]`
- **Tipos numéricos largos**: `java.math.BigInteger`, `java.math.BigDecimal`
- **Strings**: `java.lang.String`
- **Tipos temporales de Java**: `java.util.Date`, `java.util.Calendar`
- **Tipos temporales de JDBC**: `java.sql.Date`, `java.sql.Time`, `java.sql.Timestamp`
- **Tipos enumerados**: cualquier tipo enumerado del sistema o definido por el usuario
- **Objetos serializables**: cualquier tipo serializable del sistema o definido por el usuario

LOBs

- Existen tipos especiales en las BD que permiten almacenar objetos de gran tamaño, se denominan LOBs (Large Objects).
- Dos tipos de LOBs
  - CLOBs: grandes objetos formados por caracteres (texto, por ejemplo el contenido de una página HTML)
  - BLOBs: grandes objetos binarios (por ejemplo, una imagen)
- En JPA debemos utilizar la anotación `@Lob` para indicar este tipo de dato. Lo podemos indicar junto a las siguientes clases:
  - `String`, `char[]` y `Character[]`: se mapea con un CLOB
  - `byte[]`, `byte[]` y `Serializable`: se mapea con un BLOB

```
@Entity
public class Empleado {
   @Id private int id;
   @Lob
   private byte[] foto;
   // ...
}
```

Tipos enumerados

- Es posible usar tipos enumerados y definir el modo de mapearlos en la BD
- Un ejemplo de tipo enumerado y su uso en una entidad
- Por defecto se mapea en una columna de enteros y se asocia cada valor de la enumeración a un entero
- Es posible hacer que en el mapeo se conviertan los valores en Strings, utilizando la anotación `@Enumerated(EnumType.STRING)` junto al tipo enumerado

Ejemplo de tipo enumerado

```
public enum TipoEmpleado {
    EMPLEADO_TIEMPO_PARCIAL,
    EMPLEADO_TIEMPO_COMPLETO,
    EMPLEADO_EXTERNO
}
```

```
@Entity
public class Empleado {
    @Id private int id;
    private TipoEmpleado tipo;
    // ...
}
```

```
@Entity
public class Empleado {
    @Id private int id;
    @Enumerated(EnumType.STRING)
    private TipoEmpleado tipo;
    // ...
}
```

Tipos temporales

- Es posible mapear los siguientes tipos temporales Java del paquete `java.sql`: `java.sql.Date`, `java.sql.Time` y `java.sql.Timestamp`
- Se mapean en los correspondientes tipos SQL
- También se pueden mapear los tipos: `java.util.Date` y `java.util.Calendar`
- Hay que indicar en qué tipo SQL se mapean con la anotación `@Temporal` y la especificación del tipo: `TemporalType.DATE`, `TemporalType.TIME` o `TemporalType.TIMESTAMP`

Ejemplo de tipos temporales

```
@Entity
public class Empleado {
   @Id private int id;
   @Temporal(TemporalType.DATE)
   private Calendar fechaNacimiento;
   @Temporal(TemporalType.DATE)
   @Column(name="S_DATE")
   private Date fechaInicio;
   // ...
}
```

Estado transitorio

- Una atributo se marca con el modificador transient cuando no se quiere mapear en la BD

```
@Entity
public class Empleado {
   @Id private int id;
   private String nombre;
   private long salario;
   transient private String traduccion;
   // ...

   public String toString() {
      if (traduccion == null) {
         traduccion =

ResourceBundle.getBundle("EmpResources").getString("Empleado");
      }
      return traduccion + ": " + id + " " + nombre;
   }
}
```

Mapeo del identificador

- El identificador de la entidad se mapea con la clave primaria de la tabla
- El atributo debe ser de uno de los siguientes tipos:
  - **Tipos Java primitivos**: `byte`, `int`, `short`, `long`, `char`
  - **Clases wrapper de tipos primitivos**: `Byte`, `Integer`, `Short`, `Long`, `Character`
  - **Arrays de tipos primitivos o de clases wrappers**
  - **Cadenas**: `java.lang.String`
  - **Tipos numéricos grandes**: `java.math.BigInteger`
  - **Tipos temporales**: `java.util.Date`, `java.sql.Date`

Generación automática del identificador

- Es posible descargar la responsabilidad de generar el identificador único de una entidad en JPA
- Se pueden definir cuatro estrategias: AUTO, TABLE, SEQUENCE o IDENTITY en la que se utilizan valores enumerados del tipo GenerationType
- La más sencilla es AUTO, que se basa en el sistema de generación de claves primarias de la BD

```
@Entity
public class Empleado {
   @Id @GeneratedValue(strategy=GenerationType.AUTO)
   private int id;
   // ...
}
```

Objetos embebidos

- Es posible definir atributos que se van mapear en más de una columna, utilizando objetos embebidos

```
@Embeddable
public class Direccion {
   private String calle;
   private String ciudad;
   private String provincia;
   @Column(name="COD_POSTAL")
   private String codigoPostal;
   // ...
}

@Entity
public class Empleado {
   @Id private int id;
   private String nombre;
   private long salario;
   @Embedded private Direccion direccion;
   // ...
}
```

Características de los objetos embebidos

- Los objetos embebidos no son entidades
- Es una forma muy cómoda de implementar con JPA una especie de tipos de datos definido por el usuario en SQL
- Las instancias de los objetos embebidos no pueden compartirse por más de una entidad (no son entidades)

Recuperación perezosa

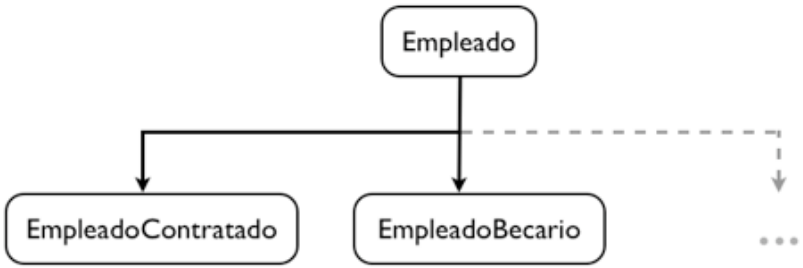
- El comportamiento por defecto de JPA cuando carga un objeto de la BD es cargar todos sus atributos (*eager fetching*, en inglés)
- Es posible marcar ciertos atributos que tienen gran tamaño con la característica de carga perezosa (*lazy fetching*) utilizando el elemento `fetch=FetchType.LAZY` de la anotación `@Basic`
- JPA sólo recupera el atributo de la BD cuando se accede a él

```
@Entity
public class Empleado {
    @Id private int id;
    @Basic(fetch=FetchType.LAZY)
    @Lob @Column(name="PIC")
    private byte[] foto;
    // ...
}
```

Herencia

- La herencia es una característica fundamental del modelo OO que no existe en el modelo relacional
- Formas de mapeo:
  - Tabla única
  - Tablas join
  - Una tabla por clase
- El mapeo más sencillo es el de tabla única: todos los registros de la jerarquía de clases van a una misma tabla. El tipo se de objeto se indica con una columna discriminante.

Ejemplo



ID	Nombre	Salario	Tipo	PlanPensiones	SeguroMedico
1	Antonio	2.300	Contrato	230	NULL
2	Juan	1.200	Beca	NULL	150
3	María	2.400	Contrato	240	NULL

Código (1)

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="Tipo", discriminatorType=DiscriminatorType.STRING)
public abstract class Empleado {
    ...
}
```

```
@Entity
@DiscriminatorValue(value="Contrato")
public class EmpleadoContratado extends Empleado {
   private Long planPensiones;

   public Long getPlanPensiones() {
      return planPensiones;
   }

   public void setPlanPensiones(Long planPensiones) {
      this.planPensiones = planPensiones;
   }
}
```

Código (2)

```
@Entity
@DiscriminatorValue(value="Beca")
public class EmpleadoBecario extends Empleado {
   private Long seguroMedico;

   public Long getSeguroMedico() {
      return seguroMedico;
   }

   public void setSeguroMedico(Long seguroMedico) {
      this.seguroMedico = seguroMedico;
   }
}
```

```
Empleado emp = new EmpleadoContratado();
emp.setId(id);
emp.setNombre(nombre);
emp.setSueldo(sueldo);
emp.setPlanPensiones(sueldo/10);
```

¿Preguntas?