UT5 - Herramientas de mapeo objeto relacional (ORM)

Ejemplo_1: Primer proyecto

1-Add dependencies:

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13.2</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>5.6.5.Final</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.27</version>
    </dependency>
</dependencies>
```

OJO poner la version 4.13.2 del JUnit ya 5.6.5.Final del org.hibernate

2-Add persistence:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.or
    <persistence-unit name="PrimerProyecto" transaction-type="RESOURCE_LOCAL">
        <class>es.hectorsanchez.Ejemplo1.Alumno</class>
        <properties>
            <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
            <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3310/ejemplo"/>
            <property name="javax.persistence.jdbc.user" value="root"/>
            <property name="javax.persistence.jdbc.password" value="root"/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL8Dialect"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
            <property name="hibernate.hbm2ddl.auto" value="create"/>
        </properties>
    </persistence-unit>
</persistence>
```

3-Clase Alumno:

```java
package es.hectorsanchez.Ejemplo1;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Alumno
{
    //Atributos
    @Id
    private int id;

    private String nombre;

    private float nota;

    public int getId()
    {
        return id;
    }
    //Getter_&_Setter...
    public void setId(int id)
    {
        this.id = id;
    }

    public String getNombre()
    {
        return nombre;
    }
```

4-Main:

```java
package es.hectorsanchez.Ejemplo1;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class App
{
    public static void main( String[] args )
    {
        //Configuramos el EMF a través de la unidad de persistencia
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("PrimerProyecto");

        //Generamos un EntityManager
        EntityManager em = emf.createEntityManager();

        //Iniciamos una transacción
        em.getTransaction().begin();

        // Construimos un objeto Alumno
        Alumno alumno1 = new Alumno();
        alumno1.setId(1);
        alumno1.setNombre("Recu");
        alumno1.setNota(5);

        // Construimos otro objeto Alumno
        Alumno alumno2 = new Alumno();
        alumno2.setId(2);
        alumno2.setNombre("María");
        alumno2.setNota(9);

        //Persistimos los objetos
        em.persist(alumno1);
        em.persist(alumno2);

        //Commiteamos la transacción
        em.getTransaction().commit();

        //Cerramos el EntityManager
        em.close();
    }
}
```

Ejemplo_2: Mapeo de entidades

```java
package es.hectorsanchez.Ejemplo2;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="alumnos")
//AddNombre a la tabla
public class Alumno
{
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    //Autoincrementable
    private int id;

    @Column(nullable = false, length = 20)
    //Nombre no puede ser nulo y max 20 char
    private String nombre;

    private float nota;

    //Getter & Setter...
    public int getId()
    {
        return id;
    }

    public void setId(int id)
    {
        this.id = id;
    }
```

```java
package es.hectorsanchez.Ejemplo2;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class App
{
    public static void main( String[] args )
    {
        //Configuramos el EMF a través de la unidad de persistencia
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("PrimerProyecto");

        //Generamos un EntityManager
        EntityManager em = emf.createEntityManager();

        //Iniciamos una transacción
        em.getTransaction().begin();

        // Construimos un objeto Alumno
        Alumno alumno1 = new Alumno();
        //alumno1.setId(1); No hace falta porque es auto-incrementado
        alumno1.setNombre("Pepe");
        alumno1.setNota(5);

        // Construimos otro objeto Alumno
        Alumno alumno2 = new Alumno();
        //alumno2.setId(2); No hace falta porque es auto-incrementado
        alumno2.setNombre("María");
        alumno2.setNota(9);

        //Persistimos los objetos
        em.persist(alumno1);
        em.persist(alumno2);

        //Commiteamos la transacción
        em.getTransaction().commit();

        //Cerramos el EntityManager
        em.close();
    }
}
```

Existen diferentes estrategias de asignación:

- **GenerationType.AUTO**: se escoge la mejor estrategia en función del dialecto SQL configurado (es decir, dependiendo del RDBMS).
- **GenerationType.SEQUENCE**: espera usar una secuencia SQL para generar los valores.
- **GenerationType.IDENTITY**: se utiliza una columna especial, autonumérica.
- **GenerationType.TABLE**: se usa una tabla extra en nuestra base de datos. Tiene una fila por cada tipo de entidad diferente, y almacena el siguiente valor a utilizar.

Ejemplo_3: Tipos embebidos

Únicamente tendremos **una única tabla** en la base de datos con los campos id, nombre, fechaNacimiento, calle y numero.

Si quisiésemos que la clase Alumno tuviese **dos direcciones**, Hibernate lanzaría una excepción indicando que hay columnas repetidas.

Para solucionarlo, sobreescribiremos los atributos de la clase embebida para que tengan otro nombre usando la anotación **@AttributeOverrides**

```
@Embeddable
public class Direccion {
    @Column
    private String calle;
    @Column
    private int numero;

    //Getter_&_Setter
    public String getCalle() {
        return calle;
    }
    public void setCalle(String calle) {
        this.calle = calle;
    }
    public int getNumero() {
        return numero;
    }
    public void setNumero(int numero) {
        this.numero = numero;
    }
}
```

```
@Entity
@Table(name="alumnos")
public class Alumno
{
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int id;

    @Column(nullable = false, length = 20)
    private String nombre;

    private float nota;

    @Embedded
    private Direccion direccion;

    @Embedded
    @AttributeOverrides({
        @AttributeOverride(name = "calle", column = @Column(name="calle_facturacion")),
        @AttributeOverride(name = "numero", column = @Column(name="numero_facturacion"))
    })
    private Direccion direccionFacturacion;
    //Getter_&_Setter...
```

En el main solo hay que añadir los campos a los objetos Alumno.

```
//Construimos un objeto Alumno---------------------------------
Alumno alumno1 = new Alumno();
//alumno1.setId(1); No hace falta porque es auto-incrementado
alumno1.setNombre("Recu");
alumno1.setNota(5);
//Add objeto Direccion al objeto Alumno
Direccion direccion1 = new Direccion();
direccion1.setCalle("PuenteSanMiguel");
direccion1.setNumero(77);
alumno1.setDireccion(direccion1);
//Add objeto Direccion (direccionFacturacion) a Alumno
Direccion direccionFacturacion1 = new Direccion();
direccionFacturacion1.setCalle("Reocin");
direccionFacturacion1.setNumero(1);
alumno1.setDireccionFacturacion(direccionFacturacion1);

//Construimos otro objeto Alumno-------------------------------
Alumno alumno2 = new Alumno();
//alumno2.setId(2); No hace falta porque es auto-incrementado
alumno2.setNombre("María");
alumno2.setNota(9);
//Add objeto Direccion al objeto Alumno
Direccion direccion2 = new Direccion();
direccion2.setCalle("Villapresente");
direccion2.setNumero(1);
alumno2.setDireccion(direccion2);
//Add objeto Direccion (direccionFacturacion) a Alumno
Direccion direccionFacturacion2 = new Direccion();
direccionFacturacion2.setCalle("Reocin");
direccionFacturacion2.setNumero(1);
alumno2.setDireccionFacturacion(direccionFacturacion2);
```

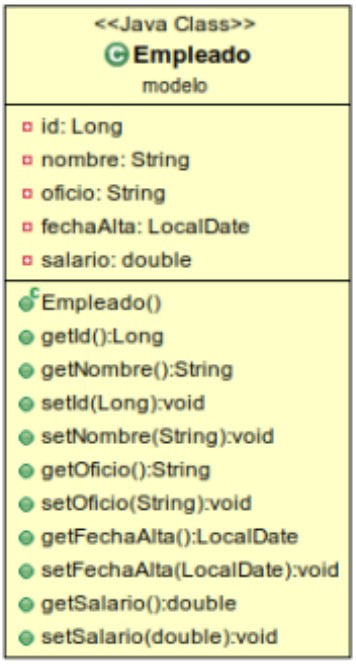
Hoja_1

EJERCICIOS

1.- Crea un proyecto Maven y configura las dependencias JPA y MySQL. Realiza todos los pasos necesarios para convertirlo en un proyecto que use JPA.

Crea una base de datos llamada **empresa** y edita el fichero persistence.xml para configurar la conexión

Crea la entidad **Empleado** dentro del paquete **modelo**:



En la clase **App** crea un EntityManager. Después crea dos empleados y haz que se guarden en la base de datos.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    <persistence-unit name="PrimerProyecto" transaction-type="RESOURCE_LOCAL">
        <class>es.hectorsanchez.hoja05_orm_01.modelo.Empleado</class>
        <properties>
            <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
            <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3310/empresa"/>
            <property name="javax.persistence.jdbc.user" value="root"/>
            <property name="javax.persistence.jdbc.password" value="root"/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL8Dialect"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
            <property name="hibernate.hbm2ddl.auto" value="create"/>
        </properties>
    </persistence-unit>
</persistence>
```

==========

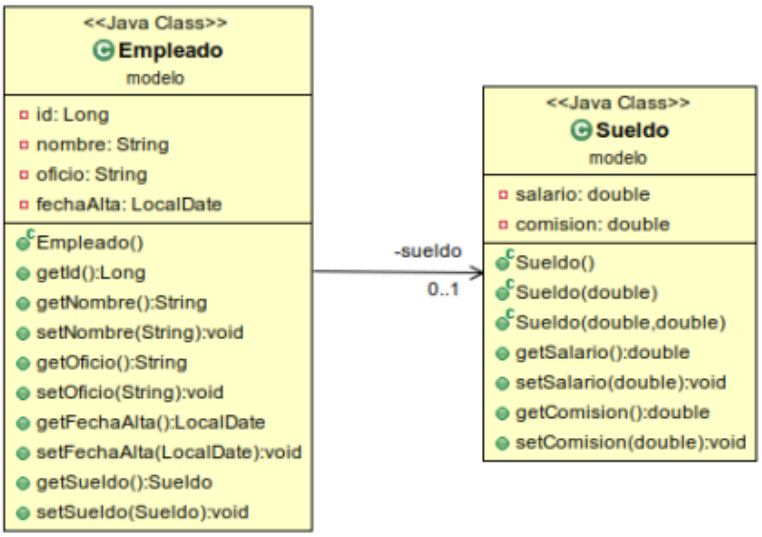
```java
package es.hectorsanchez.hoja05_orm_01;

import java.time.LocalDate;

public class App {
    public static void main(String[] args) {
        // Configuramos el EMF a través de la unidad de persistencia
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("PrimerProyecto");
        // Generamos un EntityManager
        EntityManager em = emf.createEntityManager();
        // Inicaimos una transaccion
        em.getTransaction().begin();
        //Construimos un empleado
        Empleado empleado1 = new Empleado();
        empleado1.setId((long) 1);
        empleado1.setNombre("Recu");
        empleado1.setOficio("Atleta");
        empleado1.setSalario(3000);
        empleado1.setFechaAlta(LocalDate.now());
        //Construimos otro empleado
        Empleado empleado2 = new Empleado();
        empleado2.setId((long) 2);
        empleado2.setNombre("Leras");
        empleado2.setOficio("Piloto");
        empleado2.setSalario(2000);
        empleado2.setFechaAlta(LocalDate.now());
        //Persistimos los objetos
        em.persist(empleado1);
        em.persist(empleado2);

        //Commiteamos la transacción
        em.getTransaction().commit();

        //Cerramos el EntityManager
        em.close();
    }
}
```

```java
package es.hectorsanchez.hoja05_orm_01.modelo;

import java.time.LocalDate;
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Empleado {
    @Id
    private Long id;
    private String nombre;
    private String oficio;
    private LocalDate fechaAlta;
    private double salario;

    public Empleado() {
        super();
    }

    //GETTRT_&_SETTR...
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }


      ...
```

Hoja_2

1.- Copia el proyecto anterior a otro llamado **Hoja05_ORM_02** y realiza las siguientes operaciones:

- Haz que la tabla que se genere en la base de datos se llame **empleados**
- Haz que la clave primaria sea autonumérica y que escoja la mejor estrategia. Además, elimina el método **setId** ya que ya no hará falta
- El nombre **NO** puede ser **nulo**
- La fecha de alta debe guardarse en la base de datos como **fecha_alta**
- La **longitud** del oficio debe ser 50
- Crea un tipo embebido que se llame **Sueldo**. Tendrá los atributos salario y comision (ambos de tipo double). El empleado tendrá un sueldo.
    - El salario **NO** puede ser **nulo**



Modifica el método main de la clase **App** para que se sigan guardando dos empleados.

```java
@Entity
@Table(name = "empleados")
public class Empleado {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Column(nullable = false)
    private String nombre;
    @Column(length = 50)
    private String oficio;
    @Column(name = "fecha_alta")
    private LocalDate fechaAlta;
    @Embedded
    @AttributeOverrides({
        @AttributeOverride(name = "salario", column = @Column(name = "salario", nullable = false)),
        @AttributeOverride(name = "comision", column = @Column(name = "comision")) })
    private Sueldo sueldo;

    // GETTER_&_SETTER...

@Embeddable
public class Sueldo {
    @Column
    private double salario;

    @Column
    private double comision;

    //GETTER_&_SETTER...
```

```java
public class App {
    public static void main(String[] args) {
        // Configuramos el EMF a través de la unidad de persistencia
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("Hoja_1");

        // Generamos un EntityManager
        EntityManager em = emf.createEntityManager();

        // Iniciamos una transacción
        em.getTransaction().begin();

        // Construimos un objeto Empleado
        Empleado empleado1 = new Empleado();
        empleado1.setNombre("Recu");
        empleado1.setOficio("Atleta");
        empleado1.setFechaAlta(LocalDate.now());
        Sueldo sueldo1 = new Sueldo();
        sueldo1.setSalario(30000);
        sueldo1.setComision(200);
        empleado1.setSueldo(sueldo1);

        // Construimos otro objeto Alumno
        Empleado empleado2 = new Empleado();
        empleado2.setNombre("Leras");
        empleado2.setOficio("Ingeniero");
        empleado2.setFechaAlta(LocalDate.now());
        Sueldo sueldo2 = new Sueldo();
        sueldo2.setSalario(2000);
        sueldo2.setComision(200);
        empleado2.setSueldo(sueldo2);

        // Persistimos los objetos
        em.persist(empleado1);
        em.persist(empleado2);

        // Commiteamos la transacción
        em.getTransaction().commit();

        // Cerramos el EntityManager
        em.close();
    }
}
```

empleados ×

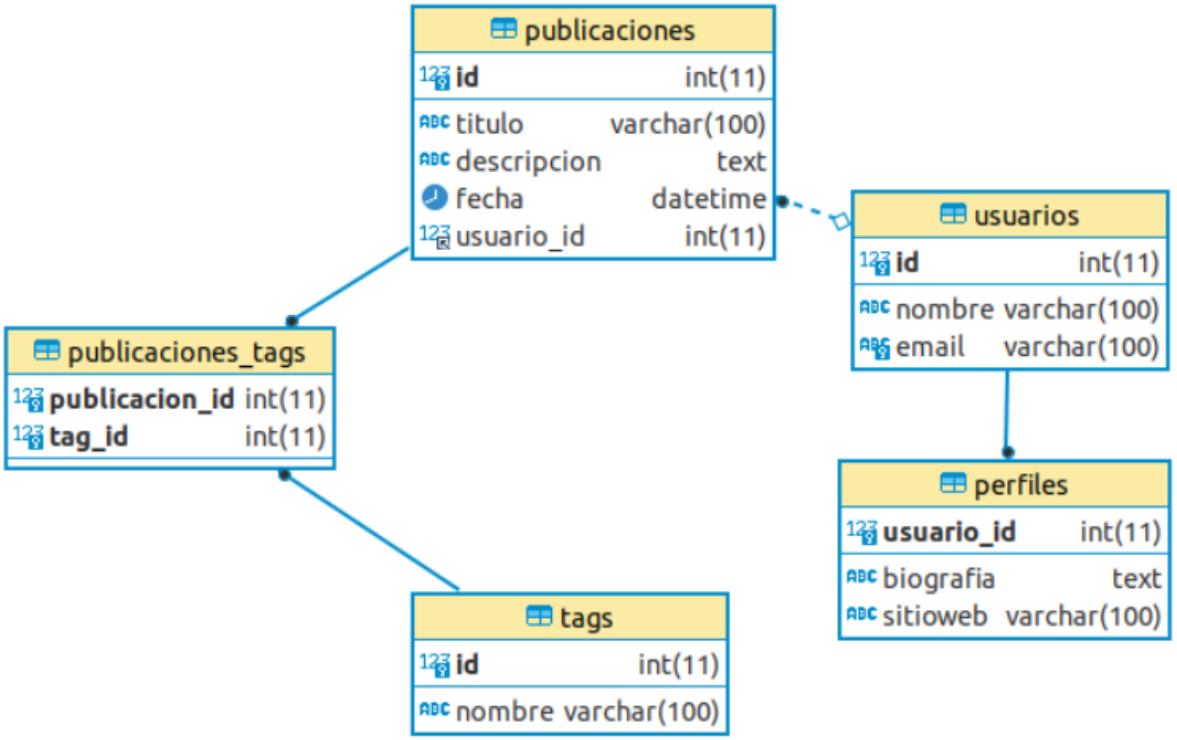
Propiedades | Datos | Diagrama ER

empleados | Enter a SQL expression to filter results (use Ctrl+Space)

Grilla | Texto

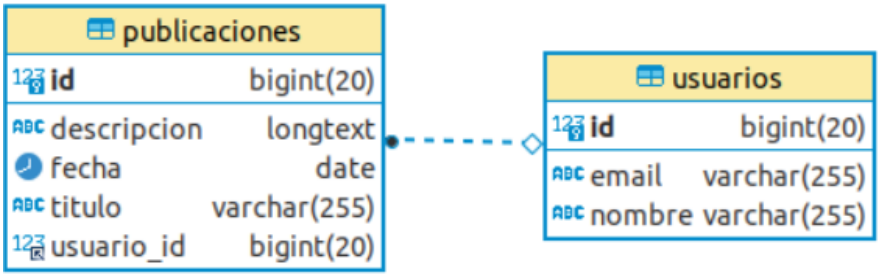
	id	fecha_alta	nombre	oficio	comision	salario
1	1	2022-02-15	Recu	Atleta	200	30.000
2	2	2022-02-15	Leras	Ingeniero	200	2.000

Ejemplo_4: Asociaciones



Ejemplo_4.1: Many-to-One Unidireccional

- Sólo se representa la asociación en el lado muchos.
- Se usa la anotación **@ManyToOne**
- Podemos añadir **@JoinColumn** para indicar el nombre de la columna que será la clave externa.
- También podemos añadir **@ForeignKey** para indicar el nombre de la restricción que se creará a nivel de base de datos.

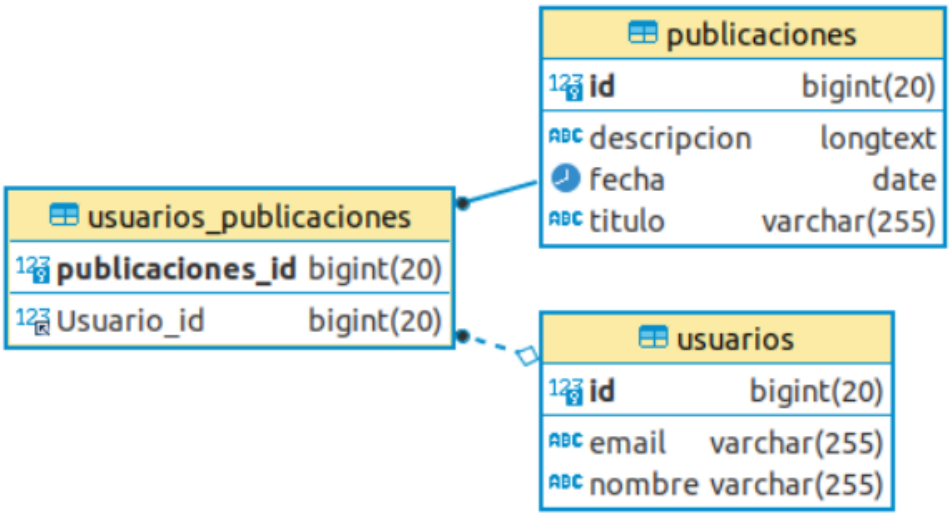


```java
package es.hectorsanchez.Ejemplo4;

import javax.persistence.Entity;

@Entity
@Table(name = "usuarios")
public class Usuario {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String email;

    private String nombre;

    //GETTER_&_SETTER...
```

```java
package es.hectorsanchez.Ejemplo4;

import java.time.LocalDate;

import javax.persistence.Entity;
import javax.persistence.ForeignKey;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.Lob;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name = "publicaciones")
public class Publicacion {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Lob
    private String descripcion;

    private LocalDate fecha;

    private String titulo;

    @ManyToOne
    @JoinColumn(name = "usuario_id",foreignKey = @ForeignKey(name = "USUARIO_ID_FK"))
    private Usuario usuario;

    //GETTER_&_SETTER...
```

Ejemplo_4.2: One-to-Many Unidireccional

- Representa la asociación en el lado uno.
- Por eso, en la clase debemos colocar una **colección** de elementos.
- Si la asociación **@OneToMany** no tiene la correspondiente asociación **@ManyToOne** será **unidireccional**. En caso de que sí exista será **bidireccional**.




```java
package es.hectorsanchez.Ejemplo4;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name = "usuarios")
public class Usuario {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String email;

    private String nombre;

    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
    //@JoinColumn(name = "usuario_id")
    private List<Publicacion> publicaciones = new ArrayList<>();

    //GETTER_&_SETTER...
```

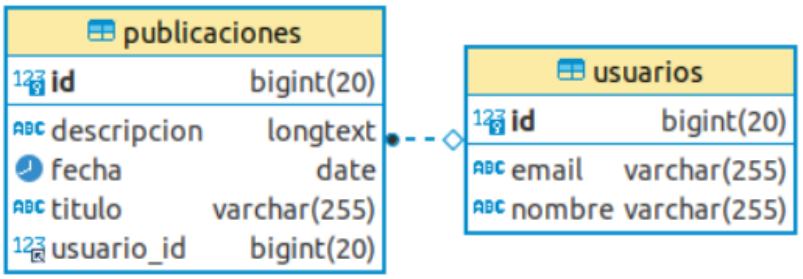
- **cascade** = **CascadeType.ALL** propagará (en cascada) todas las operaciones a las entidades relacionadas
- **orphanRemoval** = **true** indica que la entidad hija será borrada cuando se borre la padre

```java
package es.hectorsanchez.Ejemplo4;

import java.time.LocalDate;

import javax.persistence.Entity;
import javax.persistence.ForeignKey;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.Lob;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name = "publicaciones")
public class Publicacion {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Lob
    private String descripcion;

    private LocalDate fecha;

    private String titulo;
```

- Hibernate creará **3 tablas**: una tabla para cada entidad, y otra tabla para asociar ambas, añadiendo una clave única al identificador del lado muchos (publicaciones_id)
- También podemos añadir la anotación **@JoinColumn** para decirle a JPA que existe una clave foránea usuario_id en la tabla publicaciones y no crear una tabla para asociar ambas (sólo **2 tablas**)

Ejemplo_4.3: One-to-Many Bidireccional

- La asociación @OneToMany bidireccional necesita de una asociación @ManyToOne en el lado hijo.




```java
package es.hectorsanchez.Ejemplo4;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name = "usuarios")
public class Usuario {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String email;

    private String nombre;

    @OneToMany(mappedBy = "usuario", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Publicacion> publicaciones = new ArrayList<>();
```

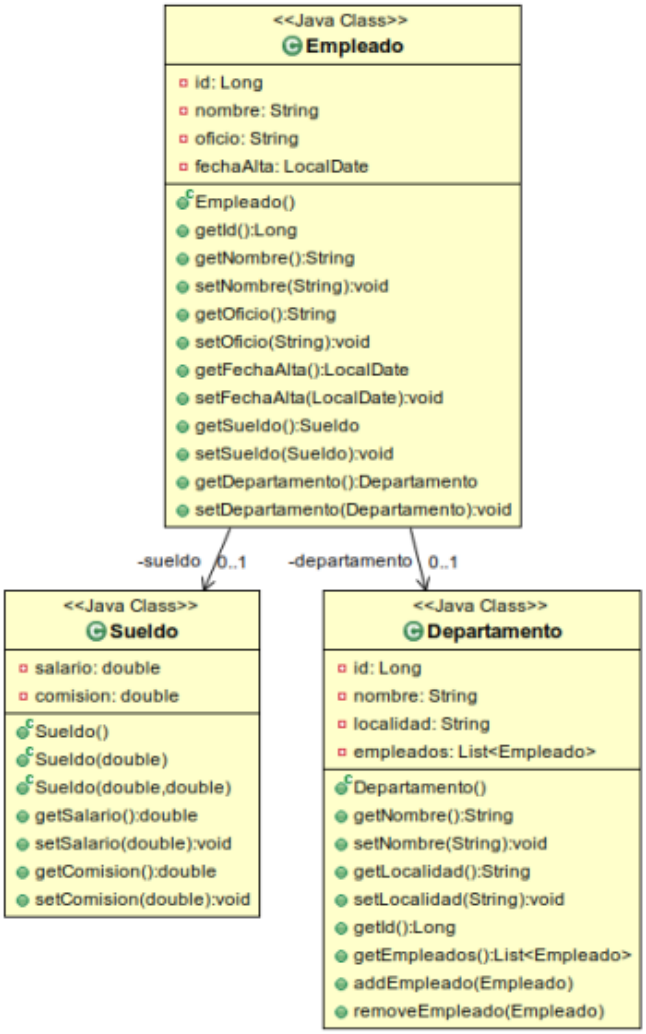
```java
    //GETTER_&_SETTER...

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public List<Publicacion> getPublicaciones() {
        return publicaciones;
    }

    public void setPublicaciones(List<Publicacion> publicaciones) {
        this.publicaciones = publicaciones;
    }

    public Publicacion addPublicacion(Publicacion publicacion)
    {
        getPublicaciones().add(publicacion);
        publicacion.setUsuario(this);

        return publicacion;
    }

    public Publicacion removePublicacion(Publicacion publicacion)
    {
        getPublicaciones().remove(publicacion);
        publicacion.setUsuario(null);

        return publicacion;
    }
}
```

- @OneToMany referenciará al otro lado mediante el atributo **mappedBy**.
- Fijarse en los métodos **addPublicacion** y **removePublicacion** ya que además de añadir o borrar una publicación establecerán cuál es su usuario.

```java
package es.hectorsanchez.Ejemplo4;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.ForeignKey;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.Lob;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name = "publicaciones")
public class Publicacion {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Lob
    private String descripcion;

    private LocalDate fecha;

    private String titulo;


    @ManyToOne
    private Usuario usuario;


    //GETTER_&_SETTER...
```

Hoja_3

1.- Copia el proyecto anterior a otro llamado **Hoja05_ORM_03** y realiza las tareas necesarias para satisfacer el siguiente diagrama de clases:



Hay que tener en cuenta que existe una asociación **one-to-many bidireccional**.

Desde la clase App habrá que crear dos departamentos y cuatro empleados (dos empleados por departamento)

```java
@Entity
@Table(name = "empleados")
public class Empleado {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Column(nullable = false)
    private String nombre;
    @Column(length = 50)
    private String oficio;
    @Column(name = "fecha_alta")
    private LocalDate fechaAlta;
    @Embedded
    @AttributeOverrides({ @AttributeOverride(name = "salario", column = @Column(name = "salario", nullable = false)),
            @AttributeOverride(name = "comision", column = @Column(name = "comision")) })
    private Sueldo sueldo;
    @ManyToOne
    private Departamento departamento;

    // GETTER_&_SETTER...
    public Departamento getDepartamento() {
        return departamento;
    }

    public void setDepartamento(Departamento departamento) {
        this.departamento = departamento;
    }

    //...
```

```java
@Entity
@Table(name = "departamentos")
public class Departamento {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String nombre;
    private String localidad;
    @OneToMany(mappedBy = "departamento", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Empleado> empelados = new ArrayList<>();

    // GETTER_&_SETTER...

    public void addEmpleado(Empleado empleado) {
        this.empelados.add(empleado);
    }

    public List<Empleado> getEmpelados() {
        return empelados;
    }

    public void setEmpelados(List<Empleado> empelados) {
        this.empelados = empelados;
    }

    // ...
```

```java
import javax.persistence.Column;

//Preguntar si esta es Embedida o ManyToOne???
@Embeddable
public class Sueldo {
    @Column
    private double salario;

    @Column
    private double comision;

    //GETTER_&_SETTER...

    public double getSalario() {
        return salario;
    }

    public void setSalario(double salario) {
        this.salario = salario;
    }

    public double getComision() {
        return comision;
    }

    public void setComision(double comision) {
        this.comision = comision;
    }
}
```

```java
public class App {
    public static void main(String[] args) {
        // Configuramos el EMF a través de la unidad de persistencia
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("Hoja_1");

        // Generamos un EntityManager
        EntityManager em = emf.createEntityManager();

        // Iniciamos una transacción
        em.getTransaction().begin();

        // Construimos dos objetos Departamento
        Departamento dep1 = new Departamento();
        dep1.setNombre("RecuEnterprise");
        dep1.setLocalidad("Oreña");

        Departamento dep2 = new Departamento();
        dep2.setNombre("IngenierosPorElMundo");
        dep2.setLocalidad("Santander");
```

```java
// Construimos un objeto Empleado
Empleado empleado1 = new Empleado();
empleado1.setNombre("Recu");
empleado1.setOficio("Atleta");
empleado1.setFechaAlta(LocalDate.now());
Sueldo sueldo1 = new Sueldo();
sueldo1.setSalario(30000);
sueldo1.setComision(200);
empleado1.setSueldo(sueldo1);
empleado1.setDepartamento(dep1);

// Construimos otro objeto Empleado
Empleado empleado2 = new Empleado();
empleado2.setNombre("Leras");
empleado2.setOficio("Ingeniero");
empleado2.setFechaAlta(LocalDate.now());
Sueldo sueldo2 = new Sueldo();
sueldo2.setSalario(2000);
sueldo2.setComision(200);
empleado2.setSueldo(sueldo2);
empleado2.setDepartamento(dep2);
```

```java
// Construimos otro objeto Empleado
Empleado empleado3 = new Empleado();
empleado3.setNombre("Maria");
empleado3.setOficio("Enfermera");
empleado3.setFechaAlta(LocalDate.now());
Sueldo sueldo3 = new Sueldo();
sueldo3.setSalario(2500);
sueldo3.setComision(400);
empleado3.setSueldo(sueldo3);
empleado3.setDepartamento(dep1);

// Construimos otro objeto Empleado
Empleado empleado4 = new Empleado();
empleado4.setNombre("BroAlextintor");
empleado4.setOficio("Ingeniero");
empleado4.setFechaAlta(LocalDate.now());
Sueldo sueldo4 = new Sueldo();
sueldo4.setSalario(2000);
sueldo4.setComision(400);
empleado4.setSueldo(sueldo4);
empleado4.setDepartamento(dep2);
```

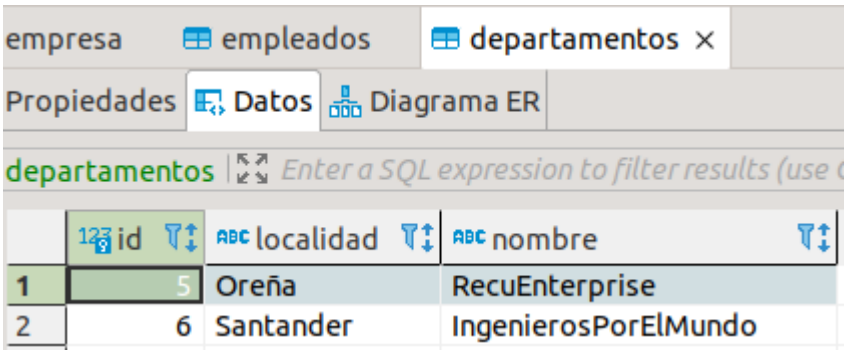
```java
        // Relleno los departamentos
        dep1.addEmpleado(empleado1);
        dep1.addEmpleado(empleado3);
        dep2.addEmpleado(empleado2);
        dep2.addEmpleado(empleado4);

        // Persistimos los objetos
        em.persist(empleado1);
        em.persist(empleado2);
        em.persist(empleado3);
        em.persist(empleado4);
        em.persist(dep1);
        em.persist(dep2);

        // Commiteamos la transacción
        em.getTransaction().commit();

        // Cerramos el EntityManager
        em.close();
    }
}
```

empresa | empleados × | departamentos

Propiedades | Datos | Diagrama ER

empleados | Enter a SQL expression to filter results (use Ctrl+Space)

	id	fecha_alta	nombre	oficio	comision	salario	departamento_id
1	1	2022-02-17	Recu	Atleta	200	30.000	5
2	2	2022-02-17	Leras	Ingeniero	200	2.000	6
3	3	2022-02-17	Maria	Enfermera	400	2.500	5
4	4	2022-02-17	BroAlextintor	Ingeniero	400	2.000	6



//
New conexion
Seleccionar base de datos
Editar el que hay, seleccionar los drivers
Drivers: usuario -> .m2 -> Repository -> org -> Postgresql -> ultima version -> .jar
postgresql - Pass!WD10 (+-)
Probar
Renombrar
==========
New JPA Entity from tables.
Seleccionar tablas.
Comprobar relaciones.
    Si no estan todas hay que generarlas a mano.
    Si no estan correctas hay que corregirlas.
Claves auto.
Corregir si no estan bien las claves primarias.
Finalizar.

A raíz de esto nos genera una las tablas en la base de datos.