

ACLARACIONES HIBERNATE

CLAVE SIMPLE AUTOGENERADA

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
</hibernate-mapping>
<class name="ejemplo03.Profesor">
  <id column="Id" name="id" type="integer">
    <generator class="increment"/>
  </id>
  <property name="nombre"/>
  <property name="ape1"/>
  <property name="ape2"/>
</class>
</hibernate-mapping>
```

```
public class Profesor {

    private int id;
    private String nombre;
    private String ape1;
    private String ape2;

    public Profesor() {
    }

    public Profesor(String nombre, String ape1, String ape2) {
        this.nombre = nombre;
        this.ape1 = ape1;
        this.ape2 = ape2;
    }

    public int getId() {
        return id;
    }

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

    /**
     * @return the nombre
     */
    public String getNombre() {
        return nombre;
    }
}
```

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
<hibernate-mapping>
  <class name="ejemplo03.Profesor">
    <id column="Id" name="id" type="integer">
      <generator class="increment"/>
    </id>
    <property name="nombre"/>
    <property name="ape1"/>
    <property name="ape2"/>
  </class>
</hibernate-mapping>

```

```

public class Profesor {

    private int id;
    private String nombre;
    private String ape1;
    private String ape2;

    public Profesor() {
    }

    public Profesor(String nombre, String ape1, String ape2) {
        this.nombre = nombre;
        this.ape1 = ape1;
        this.ape2 = ape2;
    }

    public int getId() {
        return id;
    }

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

    /**
     * @return the nombre
     */
    public String getNombre() {
        return nombre;
    }

```

CLAVE COMPUESTA

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge
]
</hibernate-mapping>
<class name="POJOS.Usos" table="usos" catalog="gimnasioshb">
  <composite-id>
    <key-many-to-one name="socio" class="POJOS.Socio" column="socio"/>
    <key-property name="fecha" type="date" column="fecha"/>
    <key-property name="hora" type="string" column="hora"/>
  </composite-id>
  //Dentro de composite-id solo puede ser key-many-to-one y no existe
  //key-one-to-many
</class>
</hibernate-mapping>

```

```

package POJOS;

import java.io.Serializable;
import java.sql.Date;

/**
 *
 * @author
 */
public class Usos implements Serializable{

    private String hora;
    private Date fecha;
    private POJOS.Socio socio;

    public Usos() {}

    public Usos(String hora, Date fecha) {
        this.hora=hora;
        this.fecha=fecha;
    }
}

```

RELACIONES

- ONE TO ONE

- Manera Sencilla One To One (Autor-Telefono)

```
<hibernate-mapping>
  <class name="POJOS.C_Telefono" table="telefonos">
    <id name="dni" column="dni">
      <generator class="foreign">
        <param name="property">cliente</param>
      </generator>
    </id>
    <property name="numero" column="numero"/>
    <property name="compañia" column="compañia"/>
    <one-to-one name="cliente" class="POJOS.C_Cliente" constrained="true" cascade="all"/>
  </class>
</hibernate-mapping>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.
] <hibernate-mapping>
]   <class name="clases.Telefono" table="telefonos">
      <id name="dni" column="dni"/>
      <property name="telf" column="telefono"/>
      <one-to-one name="autor" class="clases.Autor" cascade="save-update" constrained="true"/>
-   </class>
- </hibernate-mapping>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.
] <hibernate-mapping>
]   <class name="clases.Telefono" table="telefonos">
      <id name="dni" column="dni"/>
      <property name="telf" column="telefono"/>
      <one-to-one name="autor" class="clases.Autor" cascade="save-update" constrained="true"/>
-   </class>
- </hibernate-mapping>
```

- ONE TO MANY

- Ej. Empresa (Departamento - Empleado)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="pojos.Departamento" table="departamentos">
    <id name="nDep" column="nDep" type="integer"/>
    <property name="nombre" column="nombre" />
    <property name="localidad" column="localidad"/>
    <set name="empleados" table="empleados" inverse="true">
      <key>
        <column name="nDep" not-null="true"/>
      </key>
      <one-to-many class="pojos.Empleado"/>
    </set>
  </class>
</hibernate-mapping>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
] <hibernate-mapping>
]   <class name="pojos.Empleado" table="empleados">
      <id name="nuss" column="nuss" type="string"/>
      <property name="nombre" column="nombre" />
      <property name="oficio" column="oficio" />
      <property name="direccion" column="direccion" />
      <property name="fecha" column="fecha" />
      <property name="salario" column="salario" />
      <property name="comision" column="comision" />
]   <many-to-one name="departamento" class="pojos.Departamento">
      <column name="nDep" not-null="true"/>
-   </many-to-one>
-   </class>
- </hibernate-mapping>
```

```
public class Departamento implements Serializable {

    private int nDep;
    private String nombre;
    private String localidad;
    Set <Empleado> empleados;

    public Departamento() {
    }

    public Departamento(int nDep, String nombre, String localidad) {
        this.nDep = nDep;
        this.nombre = nombre;
        this.localidad = localidad;
        this.empleados = new HashSet<>();
    }

    public int getnDep() {
        return nDep;
    }

    public void setnDep(int nDep) {
        this.nDep = nDep;
    }

    public String getNombre() {
        return nombre;
    }

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

    public String getLocalidad() {
        return localidad;
    }

    public void setLocalidad(String localidad) {
        this.localidad = localidad;
    }

    public Set<Empleado> getEmpleados() {
        return empleados;
    }

    public void setEmpleados(Set<Empleado> empleados) {
        this.empleados = empleados;
    }
}
```

```
public class Empleado implements Serializable {

    private String nuss;
    private String nombre;
    private String oficio;
    private String direccion;
    private Date fecha;
    private float salario;
    private float comision;
    private Departamento departamento;

    public Empleado() {
    }

    public Empleado(String nuss, String nombre, String oficio, String
        this.nuss = nuss;
        this.nombre = nombre;
        this.oficio = oficio;
        this.direccion = direccion;
        this.fecha = fecha;
        this.salario = salario;
        this.comision = comision;
    }

    public String getNuss() {
        return nuss;
    }

    public void setNuss(String nuss) {
        this.nuss = nuss;
    }
}
```



```
public String getNombre() {  
    return nombre;  
}  
  
public void setNombre(String nombre) {  
    this.nombre = nombre;  
}  
  
public String getOficio() {  
    return oficio;  
}  
  
public void setOficio(String oficio) {  
    this.oficio = oficio;  
}  
  
public String getDireccion() {  
    return direccion;  
}  
  
public void setDireccion(String direccion) {  
    this.direccion = direccion;  
}  
  
public Date getFecha() {  
    return fecha;  
}  
  
public void setFecha(Date fecha) {  
    this.fecha = fecha;  
}  
  
public float getSalario() {  
    return salario;  
}
```

```

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

    public float getComision() {
        return comision;
    }

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

    public Departamento getDepartamento() {
        return departamento;
    }

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

```

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="pojos.Departamento" table="departamentos">
        <id name="nDep" column="nDep" type="integer"/>
        <property name="nombre" column="nombre" />
        <property name="localidad" column="localidad"/>

        <set name="empleados" table="empleados" inverse="true">
            <key>
                <column name="nDep" not-null="true"/>
            </key>
            <one-to-many class="pojos.Empleado"/>
        </set>
    </class>
</hibernate-mapping>

```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
] <hibernate-mapping>
]   <class name="pojos.Empleado" table="empleados">
      <id name="nuss" column="nuss" type="string"/>
      <property name="nombre" column="nombre" />
      <property name="oficio" column="oficio" />
      <property name="direccion" column="direccion" />
      <property name="fecha" column="fecha" />
      <property name="salario" column="salario" />
      <property name="comision" column="comision" />
]   <many-to-one name="departamento" class="pojos.Departamento">
      <column name="nDep" not-null="true"/>
-   </many-to-one>
-   </class>
- </hibernate-mapping>
```

```
public class Departamento implements Serializable {

    private int nDep;
    private String nombre;
    private String localidad;
    Set <Empleado> empleados;

    public Departamento() {
    }

    public Departamento(int nDep, String nombre, String localidad) {
        this.nDep = nDep;
        this.nombre = nombre;
        this.localidad = localidad;
        this.empleados = new HashSet<>();
    }

    public int getnDep() {
        return nDep;
    }

    public void setnDep(int nDep) {
        this.nDep = nDep;
    }

    public String getNombre() {
        return nombre;
    }

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

```
public String getLocalidad() {  
    return localidad;  
}  
  
public void setLocalidad(String localidad) {  
    this.localidad = localidad;  
}  
  
public Set<Empleado> getEmpleados() {  
    return empleados;  
}  
  
public void setEmpleados(Set<Empleado> empleados) {  
    this.empleados = empleados;  
}  
}
```

```
public class Empleado implements Serializable {

    private String nuss;
    private String nombre;
    private String oficio;
    private String direccion;
    private Date fecha;
    private float salario;
    private float comision;
    private Departamento departamento;

    public Empleado() {
    }

    public Empleado(String nuss, String nombre, String oficio, String
        this.nuss = nuss;
        this.nombre = nombre;
        this.oficio = oficio;
        this.direccion = direccion;
        this.fecha = fecha;
        this.salario = salario;
        this.comision = comision;
    }

    public String getNuss() {
        return nuss;
    }

    public void setNuss(String nuss) {
        this.nuss = nuss;
    }
}
```

```
public String getNombre() {  
    return nombre;  
}  
  
public void setNombre(String nombre) {  
    this.nombre = nombre;  
}  
  
public String getOficio() {  
    return oficio;  
}  
  
public void setOficio(String oficio) {  
    this.oficio = oficio;  
}  
  
public String getDireccion() {  
    return direccion;  
}  
  
public void setDireccion(String direccion) {  
    this.direccion = direccion;  
}  
  
public Date getFecha() {  
    return fecha;  
}  
  
public void setFecha(Date fecha) {  
    this.fecha = fecha;  
}  
  
public float getSalario() {  
    return salario;  
}
```

```

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

    public float getComision() {
        return comision;
    }

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

    public Departamento getDepartamento() {
        return departamento;
    }

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

```

- MANY TO MANY

- Ej. Librería (Libros - Autores)

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.org/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="hibernatelibreria.C_PojoLibro" table="libros">
        <id column="codigo" name="codigo" type="integer"/>
        <property column="titulo" name="titulo" type="string"/>
        <property column="precio" name="precio" type="float"/>
        <set name="autores" table="libros_autores" inverse="false" cascade="save-update">
            <key>
                <column name="libro" not-null="true"/>
            </key>
            <many-to-many entity-name="hibernatelibreria.C_PojoAutor">
                <column name="autor" not-null="true"/>
            </many-to-many>
        </set>
    </class>
</hibernate-mapping>

```



```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.org/hibernate-mapping-3.0.dtd" [
<hibernate-mapping>
    <class name="hibernatelibreria.C_PojoAutor" table="autores">
        <id column="id" name="id" type="integer"></id>
        <property name="nombre" column="nombre" type="string"/>
        <property name="nacionalidad" column="nacionalidad" type="string"/>
        <set name="libros" table="libros_autores" inverse="true" cascade="save-update">
            //se refiere al atributo libros dentro del set de la clase
            <key>
                <column name="autor" not-null="true"/>
                //se refiere a la columna autor dentro de la tabla Libros_Autores
            </key>
            <many-to-many entity-name="hibernatelibreria.C_PojoLibro">
                <column name="libro" not-null="true"/>
                //se refiere a la columna libro dentro de la tabla Libros_Autores
            </many-to-many>
        </set>
    </class>
</hibernate-mapping>

```

```

public class C_PojoAutor implements Serializable{
    private int id;
    private String nombre;
    private String nacionalidad;
    private Set<C_PojoLibro> libros ;

    public C_PojoAutor() {
    }

    public C_PojoAutor(int id, String nombre, String nacionalidad) {
        this.id = id;
        this.nombre = nombre;
        this.nacionalidad = nacionalidad;
        this.libros = new HashSet();
    }

    public int getId() {
        return id;
    }

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

    public String getNombre() {
        return nombre;
    }
}

```

```

public class C_PojoLibro implements Serializable{
    private int codigo;
    private String titulo;
    private float precio;
    private Set<C_PojoAutor> autores = new HashSet();

    public C_PojoLibro() {
    }

    public C_PojoLibro(int codigo, String titulo, float precio) {
        this.codigo = codigo;
        this.titulo = titulo;
        this.precio = precio;
    }

    public int getCodigo() {
        return codigo;
    }
}

```

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.org/hibernate-mapping-3.0.dtd" [
<hibernate-mapping>
    <class name="hibernatelibreria.C_PojoAutor" table="autores">
        <id column="id" name="id" type="integer"></id>
        <property name="nombre" column="nombre" type="string"/>
        <property name="nacionalidad" column="nacionalidad" type="string"/>
        <set name="libros" table="libros_autores" inverse="true" cascade="save-update">
            //se refiere al atributo libros dentro del set de la clase
            <key>
                <column name="autor" not-null="true"/>
                //se refiere a la columna autor dentro de la tabla Libros_Autores
            </key>
            <many-to-many entity-name="hibernatelibreria.C_PojoLibro">
                <column name="libro" not-null="true"/>
                //se refiere a la columna libro dentro de la tabla Libros_Autores
            </many-to-many>
        </set>
    </class>
</hibernate-mapping>
]

```

```

public class C_PojoAutor implements Serializable{
    private int id;
    private String nombre;
    private String nacionalidad;
    private Set<C_PojoLibro> libros ;

    public C_PojoAutor() {
    }

    public C_PojoAutor(int id, String nombre, String nacionalidad) {
        this.id = id;
        this.nombre = nombre;
        this.nacionalidad = nacionalidad;
        this.libros = new HashSet();
    }

    public int getId() {
        return id;
    }

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

    public String getNombre() {
        return nombre;
    }

```

```

public class C_PojoLibro implements Serializable{
    private int codigo;
    private String titulo;
    private float precio;
    private Set<C_PojoAutor> autores = new HashSet();

    public C_PojoLibro() {
    }

    public C_PojoLibro(int codigo, String titulo, float precio) {
        this.codigo = codigo;
        this.titulo = titulo;
        this.precio = precio;
    }

    public int getCodigo() {
        return codigo;
    }

```

TABLAS

```

public static void CrearTablas () {

    try{

        Connection conexion=DriverManager.getConnection("jdbc:mysql://localhost:3307/?user=root&password=usbw");
        Statement sentencia=conexion.createStatement();

        sentencia.execute("CREATE DATABASE IF NOT EXISTS LIBRERIA_HIBERNATE_ BD ");
        sentencia.execute("USE LIBRERIA_HIBERNATE_ BD ");

        sentencia.execute("CREATE TABLE IF NOT EXISTS AUTORES"+
            "(idAutor INT(4) UNSIGNED ZEROFILL NOT NULL AUTO_INCREMENT, "+
            "nombre VARCHAR(30) NOT NULL, "+
            "nacionalidad VARCHAR(30) NOT NULL, "+
            "PRIMARY KEY (idAutor)) "+
            "ENGINE INNODB;");

        sentencia.execute("CREATE TABLE IF NOT EXISTS LIBROS "+
            "(idLibro INT(4) UNSIGNED ZEROFILL NOT NULL AUTO_INCREMENT, "+
            "titulo VARCHAR(30) NOT NULL, "+
            "precio INT NOT NULL, "+
            "PRIMARY KEY (idLibro)) "+
            "ENGINE INNODB;");

        sentencia.execute("CREATE TABLE IF NOT EXISTS LIBROS_AUTORES "+
            "(autor INT(4) UNSIGNED NOT NULL, "+
            "libro INT(4) UNSIGNED NOT NULL, "+
            "PRIMARY KEY (autor, libro), "+
            "CONSTRAINT fkl_autor "+
            "FOREIGN KEY (autor) references AUTORES (idAutor) "+
            "ON DELETE CASCADE "+
            "ON UPDATE CASCADE , "+
            "CONSTRAINT fk2_libro "+
            "FOREIGN KEY (libro) references LIBROS (idLibro) "+
            "ON DELETE CASCADE "+
            "ON UPDATE CASCADE) "+
            "ENGINE INNODB;");

        conexion.close();

        System.out.println("-- BASE DE DATOS LISTA --");

    }catch (SQLException e) {

        System.out.println(e.getMessage());

    }

}

```

```

public static void CrearTablas () {

    try{

        Connection conexion=DriverManager.getConnection("jdbc:mysql://localhost:3307/?user=root&password=usbw");
        Statement sentencia=conexion.createStatement();

        sentencia.execute("CREATE DATABASE IF NOT EXISTS LIBRERIA_HIBERNATE_MIGUEL;");
        sentencia.execute("USE LIBRERIA_HIBERNATE_MIGUEL;");

        sentencia.execute("CREATE TABLE IF NOT EXISTS AUTORES"+
            "(idAutor INT(4) UNSIGNED ZEROFILL NOT NULL AUTO_INCREMENT,"+
            "nombre VARCHAR(30) NOT NULL,"+
            "nacionalidad VARCHAR(30) NOT NULL,"+
            "PRIMARY KEY (idAutor))"+
            "ENGINE INNODB;");

        sentencia.execute("CREATE TABLE IF NOT EXISTS LIBROS "+
            "(idLibro INT(4) UNSIGNED ZEROFILL NOT NULL AUTO_INCREMENT,"+
            "titulo VARCHAR(30) NOT NULL,"+
            "precio INT NOT NULL,"+
            "PRIMARY KEY (idLibro))"+
            "ENGINE INNODB;");

        sentencia.execute("CREATE TABLE IF NOT EXISTS LIBROS_AUTORES "+
            "(autor INT(4) UNSIGNED NOT NULL,"+
            "libro INT(4) UNSIGNED NOT NULL,"+
            "PRIMARY KEY (autor,libro), "+
            "CONSTRAINT fkl_autor"+
            "    FOREIGN KEY (autor) references AUTORES (idAutor)+"
            "    ON DELETE CASCADE "+
            "    ON UPDATE CASCADE ,"+
            "CONSTRAINT fk2_libro"+
            "    FOREIGN KEY (libro) references LIBROS (idLibro)+"
            "    ON DELETE CASCADE"+
            "    ON UPDATE CASCADE)"+
            "ENGINE INNODB;");

        conexion.close();

        System.out.println("-- BASE DE DATOS LISTA --");

    }catch (SQLException e) {

        System.out.println(e.getMessage());

    }

}

```

HERENCIA

- Empleado (EmpleadoFijo y EmpleadoTemporal)

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="herencia_Tarefa4.C_Empregado" table="empregado">
<id column="nss" name="nss" type="string"/>
<property column="nome" name="nome"/>
<property column="apelido1" name="apelido1"/>
<property column="apelido2" name="apelido2"/>
<property column="dataNascimento" name="dataNascimento"/>
<property column="sexo" name="sexo"/>
<property column="rua" name="rua"/>
<property column="codigoPostal" name="codigoPostal"/>
<property column="localidade" name="localidade"/>
<property column="provincia" name="provincia"/>

<!-- mapeo classe empregado temporal -->
<joined-subclass name="herencia_Tarefa4.C_EmpregadoTemporal" table="empregadoTemporal">
<key column="Nss"/>
<property column="dataInicio" name="dataInicio"/>
<property column="dataFin" name="dataFin"/>
<property column="costeHora" name="costeHora"/>
<property column="numHoras" name="numHoras"/>
</joined-subclass>

<!-- mapeo classe empregado fixo -->
<joined-subclass name="herencia_Tarefa4.C_EmpregadoFixo" table="empregadoFixo">
<key column="Nss"/>
<property column="salario" name="salario"/>
<property column="dataAlta" name="dataAlta"/>
<property column="categoria" name="categoria"/>
</joined-subclass>
</class>
</hibernate-mapping>

```

```

import java.sql.*;

public class creacionTablas {

    public static void crearTablas(Statement sentencia,Connection conexion) {
        try {
            sentencia.execute("CREATE TABLE IF NOT EXISTS empregado (nss VARCHAR(15) PRIMARY KEY, nome VARCHAR(25) NOT NULL, apelido1 VARCHAR(25) NOT NULL, apelido2 VARCHAR(25), dataNascimento DATE, sexo CHAR(1) NOT NULL, rua VARCHAR(20) NOT NULL, localidade VARCHAR(20) NOT NULL, codigoPostal INT);");
            sentencia.execute("CREATE TABLE IF NOT EXISTS empregadoTemporal (nss VARCHAR(15) PRIMARY KEY, dataInicio DATE NOT NULL, dataFin DATE NOT NULL, costeHora float NOT NULL, numHoras INT);");
            sentencia.execute("CREATE TABLE IF NOT EXISTS empregadoFixo (nss VARCHAR(15) PRIMARY KEY, salario float NOT NULL, dataAlta DATE NOT NULL, categoria VARCHAR(20));");
            sentencia.execute("ALTER TABLE empregadoTemporal ADD FOREIGN KEY (nss) REFERENCES empregado (nss) ON DELETE CASCADE ON UPDATE CASCADE;");
            sentencia.execute("ALTER TABLE empregadoFixo ADD FOREIGN KEY (nss) REFERENCES empregado (nss) ON DELETE CASCADE ON UPDATE CASCADE;");
        } catch (SQLException e) {
            System.out.println("Error: " + e.getMessage());
            System.exit(1);
        }
    }
}

```

```

public class C_Empregado implements Serializable{
    private String nome;
    private String apelido1;
    private String apelido2;
    private String nss;
    private Date dataNascimento;
    private char sexo;
    private String rua, provincia, localidade;
    private int codigoPostal;

    public C_Empregado() {
    }

    public C_Empregado(String nome, String apelido1, String apelido2, String nss, Date dataNascimento, char sexo, String rua, String provincia, String localidade, int codigoPostal) {
        this.nome = nome;
        this.apelido1 = apelido1;
        this.apelido2 = apelido2;
        this.nss = nss;
        this.dataNascimento = dataNascimento;
        this.sexo = sexo;
        this.rua = rua;
        this.provincia = provincia;
        this.localidade = localidade;
        this.codigoPostal = codigoPostal;
    }

    public String getApelido1() {
        return apelido1;
    }
}

```

```

import java.sql.Date;

public class C_EmpregadoFixo extends C_Empregado{
    private float salario;
    private Date dataAlta;
    private String categoria;

    public C_EmpregadoFixo() {
    }

    public C_EmpregadoFixo(float salario, Date dataAlta, String categoria, String nome, String apelido1, String apelido2, String nss, Date dataNascimento, char sexo, String rua, String provincia, String localidade, int codigoPostal) {
        super(nome, apelido1, apelido2, nss, dataNascimento, sexo, rua, provincia, localidade, codigoPostal);
        this.salario = salario;
        this.dataAlta = dataAlta;
        this.categoria = categoria;
    }
}

```

```

public class C_EmpregadoTemporal extends C_Empregado{
    private Date dataInicio;
    private Date dataFin;
    private Float costeHora;
    private Float numHoras;

    public C_EmpregadoTemporal() {
    }

    public C_EmpregadoTemporal(Date dataInicio, Date dataFin, Float costeHora, Float numHoras, String nome,
        super(nome, apelido1, apelido2, nss, dataNascimento, sexo, rua, provincia, localidade, codigoPostal)
        this.dataInicio = dataInicio;
        this.dataFin = dataFin;
        this.costeHora = costeHora;
        this.numHoras = numHoras;
    }
}

```

```

public class C_Empregado implements Serializable{
    private String nome;
    private String apelido1;
    private String apelido2;
    private String nss;
    private Date dataNascimento;
    private char sexo;
    private String rua, provincia, localidade;
    private int codigoPostal;

    public C_Empregado() {
    }

    public C_Empregado(String nome, String apelido1, String apelido2, String nss, Date dataNascimento, char sexo, String rua, String provincia, String
        this.nome = nome;
        this.apelido1 = apelido1;
        this.apelido2 = apelido2;
        this.nss = nss;
        this.dataNascimento = dataNascimento;
        this.sexo = sexo;
        this.rua = rua;
        this.provincia = provincia;
        this.localidade = localidade;
        this.codigoPostal = codigoPostal;
    }

    public String getApelido1() {
        return apelido1;
    }
}

```

```

import java.sql.Date;

public class C_EmpregadoFixo extends C_Empregado{
    private float salario;
    private Date dataAlta;
    private String categoria;

    public C_EmpregadoFixo() {
    }

    public C_EmpregadoFixo(float salario, Date dataAlta, String categoria, String nome, String apelido1, String apelido2, String nss, Date dataNascimento, char sexo, String rua, String provincia, String localidade, int codigoPostal)
        super(nome, apelido1, apelido2, nss, dataNascimento, sexo, rua, provincia, localidade, codigoPostal)
        this.salario = salario;
        this.dataAlta = dataAlta;
        this.categoria = categoria;
    }
}

```

```

public class C_EmpregadoTemporal extends C_Empregado{
    private Date dataInicio;
    private Date dataFin;
    private Float costeHora;
    private Float numHoras;

    public C_EmpregadoTemporal() {
    }

    public C_EmpregadoTemporal(Date dataInicio, Date dataFin, Float costeHora, Float numHoras, String nome,
        super(nome, apelido1, apelido2, nss, dataNascimento, sexo, rua, provincia, localidade, codigoPostal)
        this.dataInicio = dataInicio;
        this.dataFin = dataFin;
        this.costeHora = costeHora;
        this.numHoras = numHoras;
    }
}

```

ACLARACIONES JPA

CLAVE SIMPLE AUTOGENERADA

```
1  RELACIONES -JPA- @
2  @Entity(name="playa") //nombre a guardar en la BD
3  public class Player {
4      @Id
5      @GeneratedValue //id generado automáticamente
6      private long id;
7      private String nickName;
8      @ManyToOne //N-1
9      @JoinColumn(name="team_id")//FK-será la existente en BD
10         //Nexo de unión entre las 2 Entities
11         //no tiene sentido si se quiere permitir elementos descolgados-violacion 1FN
12         //más coherente usar JoinTable
13     private Team team;
14     ...
15 }
16
```

```
1  RELACIONES -JPA- @
2  @Entity(name="playa") //nombre a guardar en la BD
3  public class Player {
4      @Id
5      @GeneratedValue //id generado automáticamente
6      private long id;
7      private String nickName;
8      @ManyToOne //N-1
9      @JoinColumn(name="team_id")//FK-será la existente en BD
10         //Nexo de unión entre las 2 Entities
11         //no tiene sentido si se quiere permitir elementos descolgados-violacion 1FN
12         //más coherente usar JoinTable
13     private Team team;
14     ...
15 }
16
```

CLAVE COMPUESTA

@Entity

```
public class Fracture {
    Long bankId; Long userId; String bone;
```

@Id

```
public Long getBankId() {return bankId;}
```

@Id

```
public Long getPlayerId() {return userId;}
public void setBankId(Long bankId){this.bankId = bankId;}
public void setPlayerId(Long userId){this.userId = userId;}
public String getBone() {return bone;}
public void setBone(String bone) {this.bone = bone;}
```


RELACIONES

- ONE TO ONE

```
30 //-----
31 @Entity
32 // @NamedQueries
33 @Table(name= "DIRECCION")//la tabla se llama así
34 public class Direccion {
35     ...
36 }
37 @OneToOne(mappedBy = "direccion",
38             fetch = FetchType.LAZY)//nombre del campo en la clase Empleado
39 }
40
41 public class Empleado {
42
43     @Id
44     @Column(name = "COD_EMPLEADO")
45     private Long codigo;
46
47     @Column(name = "APELLIDOS")
48     private String apellidos;
49
50     @Column(name = "NOMBRE")
51     private String nombre;
52
53     @Column(name = "FECHA_NACIMIENTO")
54     //private Date fechaNacimiento;
55     private java.time.LocalDate fechaNacimiento;//java 8
56
57     @OneToOne(cascade = {CascadeType.ALL})//para que la actualización de dirección se propague en Insert/Delete..
58     @JoinColumn(name = "ID_DIRECCION")//columna en la TablaEmpleado
59     private Direccion direccion;
60 }
61
62 //-----
63
```

```
30 //-----
31 @Entity
32 // @NamedQueries
33 @Table(name= "DIRECCION")//la tabla se llama así
34 public class Direccion {
35     ...
36 }
37 @OneToOne(mappedBy = "direccion",
38             fetch = FetchType.LAZY)//nombre del campo en la clase Empleado
39 }
40
41 public class Empleado {
42
43     @Id
44     @Column(name = "COD_EMPLEADO")
45     private Long codigo;
46
47     @Column(name = "APELLIDOS")
48     private String apellidos;
49
50     @Column(name = "NOMBRE")
51     private String nombre;
52
53     @Column(name = "FECHA_NACIMIENTO")
54     //private Date fechaNacimiento;
55     private java.time.LocalDate fechaNacimiento;//java 8
56
57     @OneToOne(cascade = {CascadeType.ALL})//para que la actualización de dirección se propague en Insert/Delete..
58     @JoinColumn(name = "ID_DIRECCION")//columna en la TablaEmpleado
59     private Direccion direccion;
60 }
61
62 //-----
63
```

- ONE TO MANY

```

RELACIONES -JPA- @
@Entity(name="playa") //nombre a guardar en la BD
public class Player {
    @Id
    @GeneratedValue //id generado automáticamente
    private long id;
    private String nickName;
    @ManyToOne //N-1
    @JoinColumn(name="team_id")//FK-será la existente en BD
    //Nexo de unión entre las 2 Entities
    //no tiene sentido si se quiere permitir elementos descolgados-violacion 1FN
    //más coherente usar JoinTable

    private Team team;
    ...
}

@Entity
public class Team { //Nombre entidad a guardar en BD
    @Id
    @GeneratedValue
    private long id;
    private String name;
    @OneToMany(mappedBy="team", //1-N -> nombre de la Entidad relacionada
    //mappedBy - NombreCampo en la otra Entity Player
    //fetch - no será necesario, ya que por defecto en JPA es Lazy en @OneToOne y@OneToMany
    targetEntity=Player.class,
    fetch=FetchType.EAGER, //OJO.EAGER-que se cargue siempre.No recomendable...para solucionar que el
    lazy no cargue datos necesarios
    cascade = CascadeType.ALL)//cascadeType.ALL - pej. para que se guarden los libros automáticamente
    cuando se crea el autor
    private List<Player> players;
    //-----
}

RELACIONES -JPA- @
@Entity(name="playa") //nombre a guardar en la BD
public class Player {
    @Id
    @GeneratedValue //id generado automáticamente
    private long id;
    private String nickName;
    @ManyToOne //N-1
    @JoinColumn(name="team_id")//FK-será la existente en BD
    //Nexo de unión entre las 2 Entities
    //no tiene sentido si se quiere permitir elementos descolgados-violacion 1FN
    //más coherente usar JoinTable

    private Team team;
    ...
}

@Entity
public class Team { //Nombre entidad a guardar en BD
    @Id
    @GeneratedValue
    private long id;
    private String name;
    @OneToMany(mappedBy="team", //1-N -> nombre de la Entidad relacionada
    //mappedBy - NombreCampo en la otra Entity Player
    //fetch - no será necesario, ya que por defecto en JPA es Lazy en @OneToOne y@OneToMany
    targetEntity=Player.class,
    fetch=FetchType.EAGER, //OJO.EAGER-que se cargue siempre.No recomendable...para solucionar que el
    lazy no cargue datos necesarios
    cascade = CascadeType.ALL)//cascadeType.ALL - pej. para que se guarden los libros automáticamente
    cuando se crea el autor
    private List<Player> players;
    //-----
}

```

- MANY TO MANY

```

//
@Entity
public class LeftManyStudent {
    @Id
    @GeneratedValue
    long id;
    String studentName;

    @ManyToMany
    @JoinTable(name = "join_table",
        joinColumns = { @JoinColumn(name = "lmstudent_id") },
        inverseJoinColumns={@JoinColumn(name="rmcourse_id")}
    )
    List<RightManyCourse> courses = new ArrayList();

    public List<RightManyCourse> getCourses() {
        return courses;
    }
    public void setCourses(List<RightManyCourse> righties) {
        this.courses = righties;
    }

    @Entity
    public class RightManyCourse {

        @Id
        @GeneratedValue
        long id;
        String courseCode;

        @ManyToMany
        @JoinTable(name = "join_table",
            joinColumns={@JoinColumn(name="rmcourse_id")},
            inverseJoinColumns={@JoinColumn(name="lmstudent_id")}
        )
        List<LeftManyStudent> students = new Vector();

        public List<LeftManyStudent> getStudents() {
            return students;
        }
        public void setStudents(List<LeftManyStudent> lefties) {
            this.students = lefties;
        }
    }
}

```

```

//
@Entity
public class LeftManyStudent {
    @Id
    @GeneratedValue
    long id;
    String studentName;

    @ManyToMany
    @JoinTable(name = "join_table",
        joinColumns = { @JoinColumn(name = "lmstudent_id") },
        inverseJoinColumns={@JoinColumn(name="rmcourse_id")}
    )
    List<RightManyCourse> courses = new ArrayList();

    public List<RightManyCourse> getCourses() {
        return courses;
    }
    public void setCourses(List<RightManyCourse> righties) {
        this.courses = righties;
    }

    @Entity
    public class RightManyCourse {

        @Id
        @GeneratedValue
        long id;
        String courseCode;

        @ManyToMany
        @JoinTable(name = "join_table",
            joinColumns={@JoinColumn(name="rmcourse_id")},
            inverseJoinColumns={@JoinColumn(name="lmstudent_id")}
        )
        List<LeftManyStudent> students = new Vector();

        public List<LeftManyStudent> getStudents() {
            return students;
        }
        public void setStudents(List<LeftManyStudent> lefties) {
            this.students = lefties;
        }
    }
}

```

HERENCIA

Las Bases de Datos relacionales no tienen una manera de tener herencia entre tablas, para solucionar esto, JPA proporciona una serie de estrategias y nos encamina a una herencia con Hibernate.

Estrategias en Herencia con Hibernate

JPA nos proporciona diferentes estrategias para lograr la herencia:

- MappedSuperclass: La clase padre no puede ser una entidad
- Single Table – las entidades que son de diferentes clases y tienen un padre común se colocan en una sola tabla.
- Joined Table – cada clase tiene su propia tabla y realizar queries sobre subclases requiere join de tablas.
- Table-Per-Class – todas las propiedades se encuentran en la tabla, así que no se requiere ninguna join.

Aplicar herencia con Hibernate en estas entidades nos permite trabajar con polimorfismo entre clases. Cada una de las estrategias anteriores, dará como resultado una estructura en base de datos diferente.

Estrategia MappeSuperClass

Con esta estrategia de herencia, lo que buscamos es añadir en nuestra tabla de base de datos las columnas que heredamos. Y la clase padre tendrá que llevar la anotación @MappedSuperClass vamos a verlo con un ejemplo:

```
@MappedSuperclass
@Getter
@Setter
public class Car {

    @Id
    private Long id;
    private String color;

}
@Getter
@Setter
@Entity
public class Model extends Car {

    private String model;

}
```

Con esta última clase, lo que hemos hecho ha sido heredar de la Clase Car, con lo que tenemos sus atributos. En Base de Datos tendremos **una tabla Model con tres columnas**. Hay que tener en cuenta que la clase Padre no lleva @Entity.

Estrategia Joined Table

En esta estrategia, la herencia es definida en la clase abstracta padre para ello vamos a usar la anotación @Inheritance. Con esta estrategia y a diferencia de la anterior, **todas las clases hijas tendrán su propia tabla**.

Tanto en la tabla padre como en la que hereda se tendrá el id de la tabla padre, es decir, tendrá una columna haciendo referencia. Se puede utilizar la anotación `@PrimaryKeyJoinColumn` para customizar este identificador. El campo que sea la clave (key), se definirá únicamente en la clase padre.

```
147 @Entity
148 @Inheritance(strategy = InheritanceType.JOINED)
149 @Getter
150 @Setter
151 public class Vehicle {
152     @Id
153     private long vehicleId;
154     private String model;
155 }
156
157 En la clase anterior hemos definido el tipo de estrategia a utilizar, y las clases hijos extenderán de aquí.
158
159 @Entity
160 @Getter
161 @Setter
162 public class Car extends Vehicle {
163     private String name;
164 }
165
```

```
147 @Entity
148 @Inheritance(strategy = InheritanceType.JOINED)
149 @Getter
150 @Setter
151 public class Vehicle {
152     @Id
153     private long vehicleId;
154     private String model;
155 }
156
157 En la clase anterior hemos definido el tipo de estrategia a utilizar, y las clases hijos extenderán de aquí.
158
159 @Entity
160 @Getter
161 @Setter
162 public class Car extends Vehicle {
163     private String name;
164 }
165
```

Esta clase Car extiende de Vehicle, ambas clases crearán una tabla en Base de Datos. Ambas tablas tendrán la columna vehicleId, y la tabla Car tendrá una **foreign key constraint** a la tabla padre.

Tal y como hemos dicho un poco más arriba podemos utilizar la anotación, PrimaryKeyJoinColumn, para poder cambiar el nombre de la columna que se hereda.

```
170 @Entity
171 @Getter
172 @Setter
173 @PrimaryKeyJoinColumn(name = "carId")
174 public class Car extends Vehicle {
175 }
```

```
170 @Entity
171 @Getter
172 @Setter
173 @PrimaryKeyJoinColumn(name = "carId")
174 public class Car extends Vehicle {
175 }
```

La mayor desventaja de esta aproximación, y que va ligado con las bases de datos relacionales, es que se van a ir haciendo joins con las tablas hijas, y esto penaliza mucho en las búsquedas cuando se tienen muchos registros.

Estrategia Tabla por Clase

En esta estrategia, vamos a tener una tabla en Base de Datos por cada clase que se cree con su entidad, incluyendo las heredadas.

El esquema resultante es similar al que usa `@MappedSuperclass`, con la diferencia que la clase padre también es una tabla, repitiendo tantas veces como se herede los mismos atributos.

Para poder utilizar esta estrategia es necesario definirla así, `@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)`

```
185 @Entity
186 @Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
187 @Getter
188 @Setter
189 public class Vehicle {
190     @Id
191     private Long vehicleId;
192
193     private String model;
194 }
195
```

```
185 @Entity
186 @Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
187 @Getter
188 @Setter
189 public class Vehicle {
190     @Id
191     private Long vehicleId;
192
193     private String model;
194 }
195
```

Esta estrategia además usa SQL UNION. Cuando no se conoce la subclase concreta, el objeto relacionado podría estar en cualquiera de las tablas de la subclase, lo que hace que las uniones a través de la relación sean imposibles, por lo tanto, proporciona un soporte deficiente para las relaciones polimórficas.

```
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Ancestor {
    @Id
    @GeneratedValue
    private Long id;
    private String nationality;
    public Long getId() {return id;}
    public void setId(Long id) {this.id = id;}
    public String getNationality() {return nationality;}
    public void setNationality(String nationality) {
        this.nationality = nationality;
    }
}
```

```
@Entity
public class Parent extends Ancestor{
    private String lastName;
```

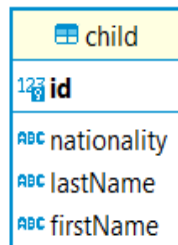
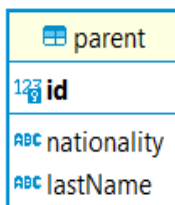
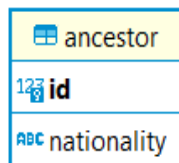
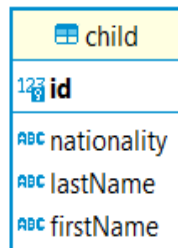
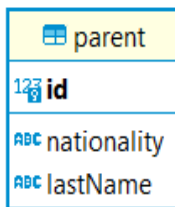
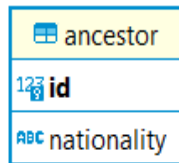


```

    public String getLastName() {return lastName;}
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}

@Entity
public class Child extends Parent {
    private String firstName;
    public String getFirstName() {return firstName;}
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
}

```



Estrategia Single Table

En este caso, se crea una tabla para cada jerarquía de las clases. Esta va a ser la estrategia que se

utiliza por defecto en JPA si no se especifica.

El identificador está definido en la clase Padre, y podemos extender y heredar de él.

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@Getter
@Setter
public class vehicle {
    @Id
    private Long vehicleId;
    private String name;
}

@Entity
public class Motorbike extends Vehicle {
    private String color;
}

@Entity
public class Car extends Vehicle {
    private String colorDoors;
}
```

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@Getter
@Setter
public class vehicle {
    @Id
    private Long vehicleId;
    private String name;
}

@Entity
public class Motorbike extends Vehicle {
    private String color;
}

@Entity
public class Car extends Vehicle {
    private String colorDoors;
}
```

Discriminator Values

Ya que todos los registros van a estar en las mismas tablas, le tenemos que especificar a Hibernate una manera de distinguirlos, y es aquí cuando entra en juego los Discriminator Values.

Por defecto, esta discriminación se suele realizar a través una columna llamada DTYPE, la cual tendrá el nombre de la entidad como valor. Pero por lo general vamos a querer customizar esta columna, por lo que vamos a usar la anotación @DiscriminatorColumn.

```
@Entity(name="vehicles")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="vehicle_type",
    discriminatorType = DiscriminatorType.INTEGER)
public class Vehicle {
    // ...
}
```

```

@Entity(name="vehicles")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="vehicle_type",
    discriminatorType = DiscriminatorType.INTEGER)
public class Vehicle {
    // ...
}

```

Con la definición de la clase anterior hemos decidido diferenciar por la columna vehicle_type en lugar de por DTYPE. A continuación, le decimos a Hibernate como hacer esa diferencia.

```

@Entity
@DiscriminatorValue("1")
public class Motorbike extends Vehicle {
    private String color;
}
@Entity
@DiscriminatorValue("2")
public class Car extends Vehicle {
    private String colorDoors;
}

```

```

@Entity
@DiscriminatorValue("1")
public class Motorbike extends Vehicle {
    private String color;
}
@Entity
@DiscriminatorValue("2")
public class Car extends Vehicle {
    private String colorDoors;
}

```

Con la definición del `@DiscriminatorValue` en las entidades anteriores hemos conseguido diferenciar que tipo de objeto será con la columna `vehicle_type`.

También se nos da la posibilidad de usar en lugar de `@DiscriminatorColumn` usar `@DiscriminatorFormula`, en donde en lugar de añadir valores se especifica una formula por ejemplo:

```

@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorFormula("case when color is not null then 1 else 2 end")
public class Vehicle {
    ...
}

```

```

@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorFormula("case when color is not null then 1 else 2 end")
public class Vehicle {
    ...
}

```

Con el objeto anterior hemos conseguido guardar un registro con una condición, es decir si el atributo `color` (de la clase `Motorbike`) no es `null`, se guardará con valor 1 la columna `vehicle_type`.

En esta estrategia vamos a tener la ventaja del rendimiento al hacer queries polimórficas, ya que solo se accede a una tabla.

CLAVES (PK)

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;
    // ...
}
```

@Id

private Long id;

Technical IDs are easier to manage and all involved systems, mainly the database and Hibernate, can index them very efficiently. This allows you to focus on the business logic of your application and avoids performance issues.

- 1 options to generate primary keys
 - 1.1 GenerationType.AUTO
 - 1.2 GenerationType.IDENTITY
 - 1.3 GenerationType.SEQUENCE
 - 1.4 GenerationType.TABLE
- 2 Summary

1.options to generate primary keys

The JPA specification supports 4 different primary key generation strategies which generate the primary key values programmatically or use database features, like auto-incremented columns or sequences. The only thing you have to do is to add the @GeneratedValue annotation to your primary key attribute and choose a generation strategy.

1.1 GenerationType.AUTO

```
@Id  
@GeneratedValue  
private Long id;  
GenerationType.AUTO
```

The GenerationType.AUTO is the default generation type and lets the persistence provider choose the generation strategy.

```
@Id  
@GeneratedValue(strategy = GenerationType.AUTO)  
private Long id;
```

If you use Hibernate as your persistence provider, it selects a generation strategy based on the database specific dialect. For most popular databases, it selects GenerationType.SEQUENCE

1.2 GenerationType.IDENTITY

GenerationType.IDENTITY

The GenerationType.IDENTITY is the easiest to use but not the best one from a performance point of view. It relies on an auto-incremented database column and lets the database generate a new value with each insert operation. From a database point of view, this is very efficient because the auto-increment columns are highly optimized, and it doesn't require any additional statements.

```
@Id  
@GeneratedValue(strategy = GenerationType.IDENTITY)  
private Long id;
```

This approach has a significant drawback if you use Hibernate. Hibernate requires a primary key value for each managed entity and therefore has to perform the insert statement immediately. This prevents it from using different optimization techniques like JDBC batching.

1.3 GenerationType.SEQUENCE

GenerationType.SEQUENCE

The GenerationType.SEQUENCE is my preferred way to generate primary key values and uses a database sequence to generate unique values.

It requires additional select statements to get the next value from a database sequence. But this has no performance impact for most applications. And if your application has to persist a huge number of new entities, you can use some Hibernate specific optimizations to reduce the number of statements.

```
@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE)
private Long id;
```

If you don't provide any additional information, Hibernate will request the next value from its default sequence. You can change that by referencing the name of a `@SequenceGenerator` in the generator attribute of the `@GeneratedValue` annotation. The `@SequenceGenerator` annotation lets you define the name of the generator, the name, and schema of the database sequence and the allocation size of the sequence.

```
@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "book_generator")
@SequenceGenerator(name="book_generator", sequenceName = "book_seq",
allocationSize=50)
private Long id;
```

`GenerationType.TABLE`

The `GenerationType.TABLE` gets only rarely used nowadays. It simulates a sequence by storing and updating its current value in a database table which requires the use of pessimistic locks which put all transactions into a sequential order. This slows down your application, and you should, therefore, prefer the `GenerationType.SEQUENCE`, if your database supports sequences, which most popular databases do.

1.4 `GenerationType.TABLE`

```
@Id
@GeneratedValue(strategy = GenerationType.TABLE)
private Long id;
```

You can use the `@TableGenerator` annotation in a similar way as the already explained `@SequenceGenerator` annotation to specify the database table which Hibernate shall use to simulate the sequence.

```
@Id
@GeneratedValue(strategy = GenerationType.TABLE, generator = "book_generator")
@TableGenerator(name="book_generator", table="id_generator", schema="bookstore")
private Long id;
```

Summary

As you've seen, JPA offers 4 different ways to generate primary key values:

AUTO: *Hibernate selects the generation strategy based on the used dialect,*

IDENTITY: *Hibernate relies on an auto-incremented database column to generate the primary key,*

SEQUENCE: *Hibernate requests the primary key value from a database sequence,*

TABLE: *Hibernate uses a database table to simulate a sequence.*

I prefer to use the `GenerationType.SEQUENCE` because it is very efficient and allows Hibernate to decide when to perform the insert statement. This provides the required flexibility to use other performance optimization techniques like JDBC batching.

When you like to learn more about performance tuning and how Hibernate can optimize the `GenerationType.SEQUENCE`, have a look at my [Hibernate Performance Tuning Online Training](#).

- **Una sola tabla** para guardar toda la jerarquía de clases.
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)

La tabla tiene el campo «**dtype**», este campo será el que se use como discriminador para saber a qué tipo concreto se refiere el registro. Por defecto en este campo se guardará el nombre de la clase (sin el nombre del paquete).

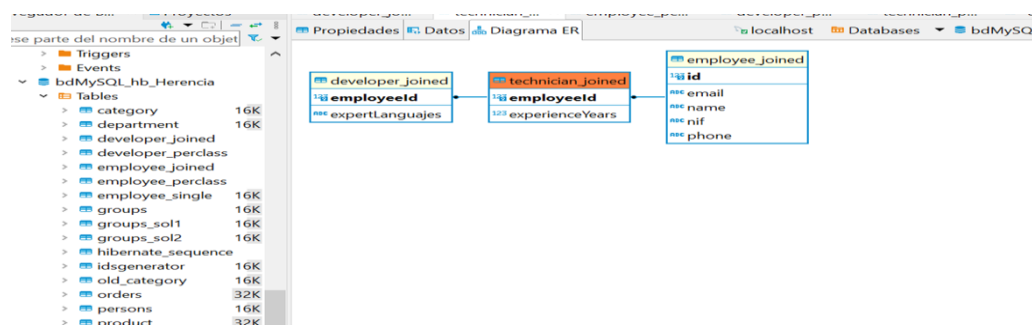
[illegible]

- Una **tabla para el padre** de la jerarquía, con las **cosas comunes**, y otra **tabla para cada clase hija** con las **cosas concretas**. («join» de tablas)
@Inheritance(strategy=InheritanceType.JOINED)

Es la opción más normalizada, y por lo tanto la más flexible (puede ser interesante si tenemos un modelo de clases muy cambiante), ya que para añadir nuevos tipos basta con añadir nuevas tablas y si queremos añadir nuevos atributos sólo hay que modificar la tabla correspondiente al tipo donde se está añadiendo el atributo.

Comparando con el ejemplo anterior, ambas clases lo único que hacen es indicar el nombre del campo que contiene la clave ajena (la FK) para hacer el join con el padre de la jerarquía.

Tiene la desventaja de que, para recuperar la información de una clase, hay que ir haciendo join con las tablas de las clases padre.



- Una **tabla independiente para cada tipo**. («union» de tablas)

@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)

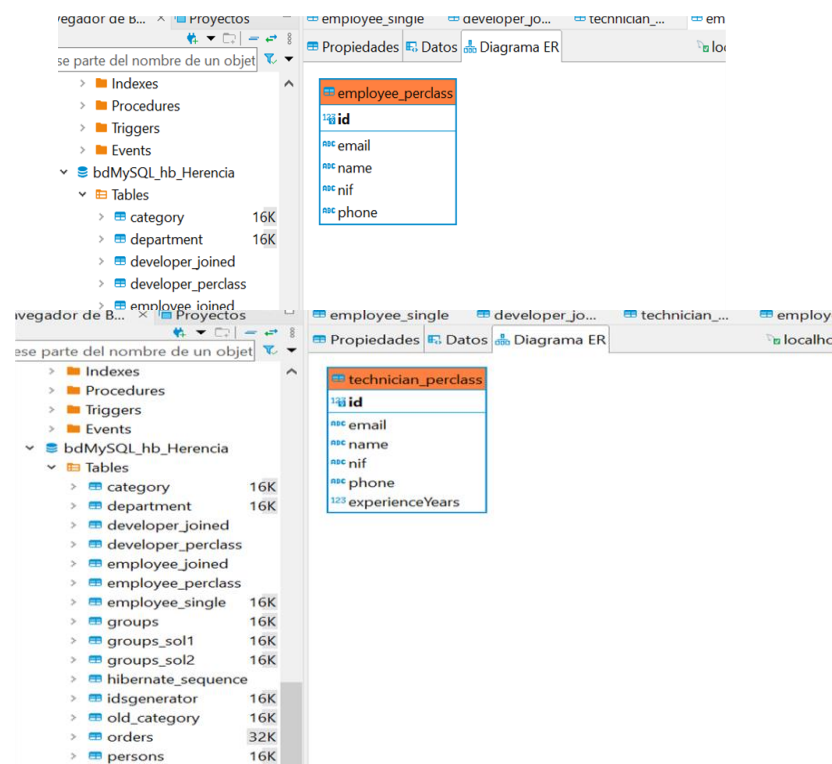
En este caso cada tabla es independiente, pero los atributos del padre (atributos comunes en los hijos), tienen que estar repetidos en cada tabla. También es destacable que los ids (las PK) no son definidos como auto incrementales. Esto es una limitación de esta

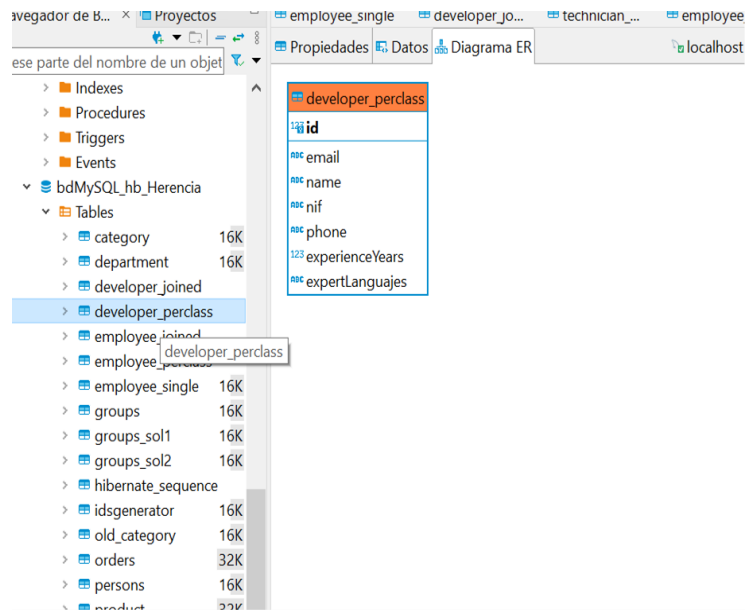
estrategia: los ids deben ser únicos para las tres tablas, así que no podemos marcar los ids como auto incrementales en cada tabla.

Es recomendable que la base de datos sea la encargada de generar las claves, y no la aplicación. Si queremos que siga siendo la base de datos la encargada de generar los ids, habrá que usar una secuencia u otra tabla (en nuestro ejemplo la tabla `IdsGenerator`) que se encargue de guardar los ids únicos para las tres tablas.

En principio puede tener serios problemas de rendimiento, si estamos trabajando con polimorfismo, por los SQL UNION que tiene que hacer para recuperar la información.

Sería la opción menos aconsejable. Tanto es así que en la versión 3.0 de EJBs, aunque está recogida en la especificación, no es obligatoria su implementación.





- Una tabla para las subclases pero ninguna para la superclase.

`@MappedSuperclass`

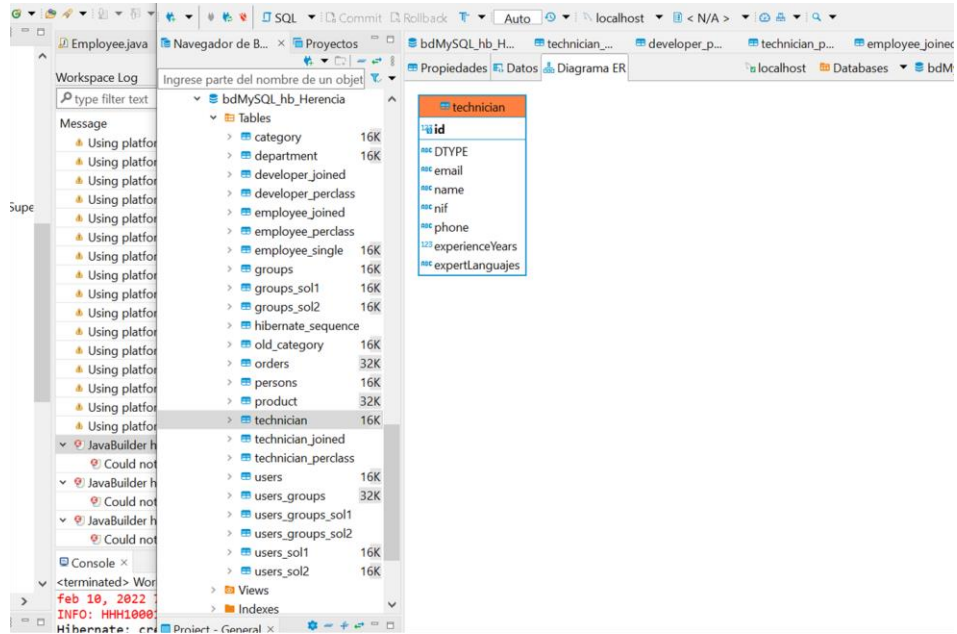
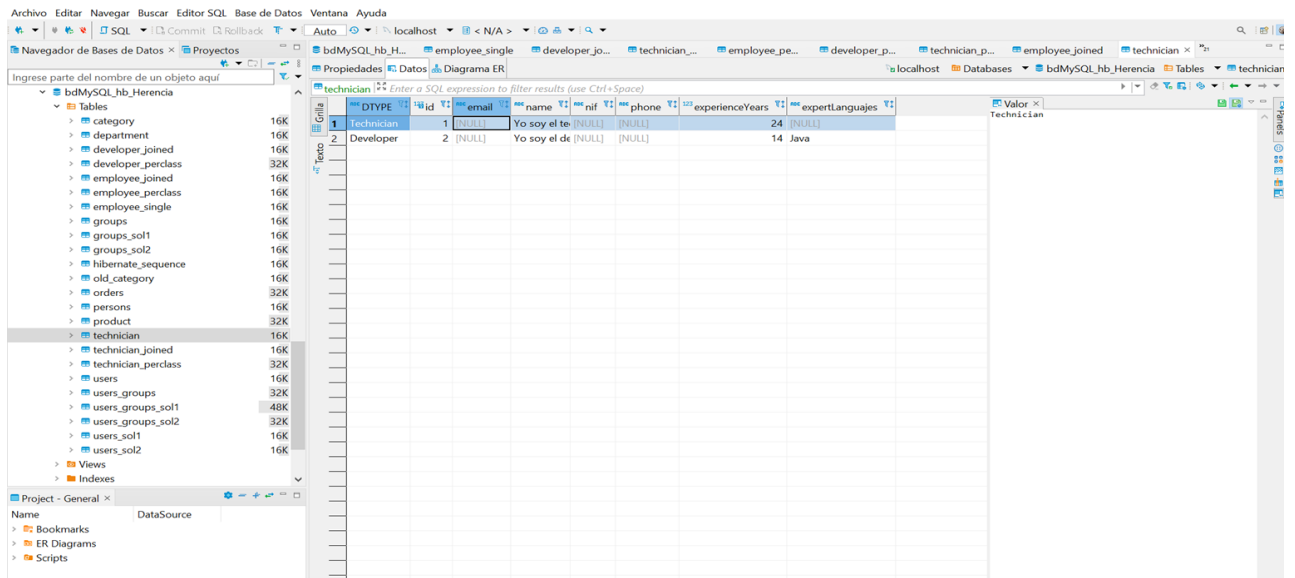
*** addEmployee ***

```
java.lang.IllegalArgumentException: Unknown entity:
teis.dual.hibernate.herencia.mappedSuperclass.Employee
    at org.hibernate.internal.SessionImpl.firePersist(SessionImpl.java:777)
    at org.hibernate.internal.SessionImpl.persist(SessionImpl.java:760)
    at
teis.dual.hibernate.herencia.mappedSuperclass.WorkingWithObjectsTest.addEmployee
(WorkingWithObjectsTest.java:91)
    at
teis.dual.hibernate.herencia.mappedSuperclass.WorkingWithObjectsTest.main(WorkingWithObjectsTest.java:40)
Hibernate: insert into Technician (email, name, nif, phone, experienceYears,
DTYPE) values (?, ?, ?, ?, ?, 'Technician')
La clave del nuevo objeto es: 1
Hibernate: insert into Technician (email, name, nif, phone, experienceYears,
expertLenguajes, DTYPE) values (?, ?, ?, ?, ?, ?, 'Developer')
La clave del nuevo objeto es: 2
```

la superclase debería ser abstract y no instanciarla, entonces pasaría:

*** addEmployee ***

```
Hibernate: insert into Technician (email, name, nif, phone, experienceYears,
DTYPE) values (?, ?, ?, ?, ?, 'Technician')
La clave del nuevo objeto es: 1
Hibernate: insert into Technician (email, name, nif, phone, experienceYears,
expertLenguajes, DTYPE) values (?, ?, ?, ?, ?, ?, 'Developer')
La clave del nuevo objeto es: 2
```



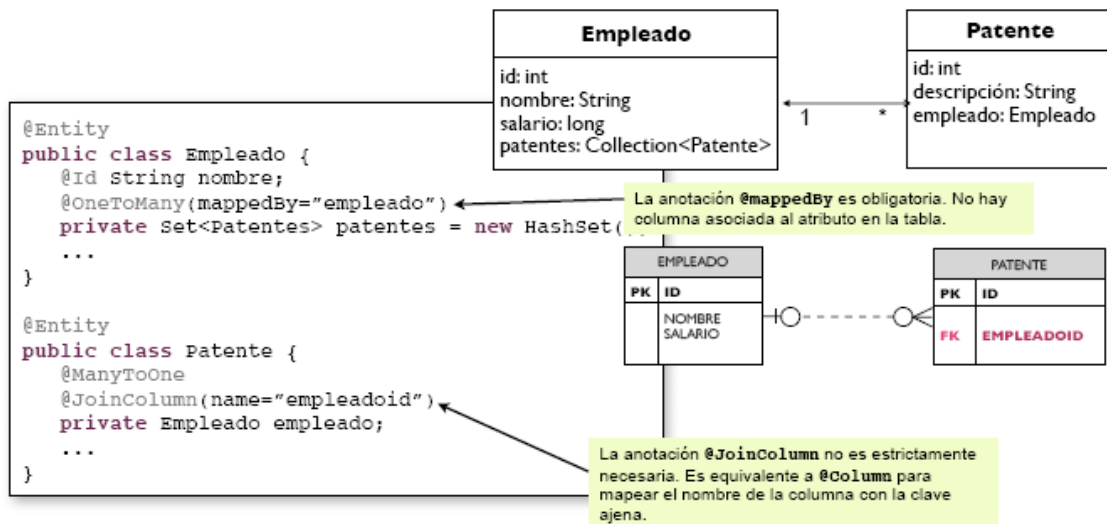
Actualización de las relaciones

- A efectos de la BD y de las posteriores queries, bastaría con actualizar la entidad propietaria de la relación, la que contiene la clave ajena (en el caso del ejemplo empleados y departamentos, el empleado)
- Problema en memoria: si el entity manager ya está gestionando la colección inversa (el conjunto de empleados del departamento) no va a volver a consultarla de la BD (a no ser que hagamos un refresh de la entidad)
- Solución: es conveniente siempre actualizar en memoria la colección

```
emp.setDepartamento(depto);  
depto.getEmpleados().add(emp);
```

Mapeo con clave ajena

- En JPA se indica la tabla que contiene la clave ajena (la entidad propietaria de la relación) con el elemento `mappedBy` en el otro lado de la relación



One-to-one unidireccional

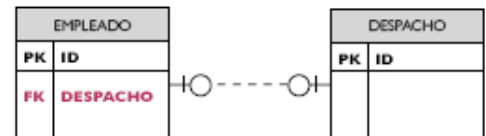
```
@Entity
public class Empleado {
    // ...
    @OneToOne
    private despacho Despacho;
    // ...

    public void setDespacho(Despacho despacho) {
        this.despacho = despacho;
    }
}

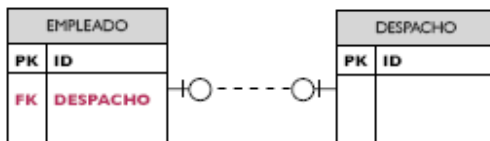
@Entity
public class Despacho {
    @Id
    String idDespacho;

    // ...
}
```

No se usa la anotación `@JoinColumn` porque el nombre de la columna coincide con el del atributo (`despacho`).



One-to-one bidireccional



```
@Empleado
public class Empleado {
    // ...
    @OneToOne
    private despacho Despacho;
    // ...

    public void setDespacho(Despacho despacho) {
        this.despacho = despacho;
    }
}

@Entity
public class Despacho {
    // ...
    @OneToOne(mappedBy="despacho")
    private empleado Empleado;
    // ...

    public Empleado getEmpleado() {
        return this.empleado;
    }
}
```

One-to-many (Many-to-one) bidireccional



Clave ajena

```
@Entity
public class Empleado {
    // ...
    @ManyToOne
    @JoinColumn(name="departamento")
    private departamento Departamento;
    // ...
}

@Entity
public class Departamento {
    // ...
    @OneToMany(mappedBy="departamento")
    private Set<Empleado> empleados = new HashSet();
    // ...
}
```


Many-to-many bidireccional (anotaciones)

```
@Entity
public class Empleado {
    @Id String nombre;
    @ManyToMany
    private Set<Proyecto> proyectos = new HashSet();

    public Set<Proyecto> getProyectos() {
        return this.proyectos;
    }

    public void setProyectos(Set<Proyecto> proyectos) {
        this.proyectos = proyectos;
    }

    // ...
}
```

```
@Entity
public class Proyecto {
    @Id String codigo;
    @ManyToMany(mappedBy="proyectos");
    private Set<Empleado> empleados = new HashSet();

    public Set<Empleado> getEmpleados() {
        return empleados;
    }

    // ...
}
```

Anotación JoinTable

- Si queremos especificar el nombre y características de la tabla join, podemos utilizar la anotación `@JoinTable`:

```
@Entity
public class Empleado {
    @Id String nombre;
    @ManyToMany
    @JoinTable(
        name = "EMPLEADO_PROYECTO",
        joinColumns = {@JoinColumn(name = "EMPLEADO_NOMBRE")},
        inverseJoinColumns = {@JoinColumn(name = "PROYECTO_CODIGO")}
    )
    private Set<Proyecto> proyectos = new HashSet<Proyecto>();
}
```

Actualización de las relaciones

- A efectos de la BD y de las posteriores queries, bastaría con actualizar la entidad propietaria de la relación, la que contiene la clave ajena (en el caso del ejemplo, el empleado)
- Problema en memoria: si el entity manager ya está gestionando la colección inversa (el conjunto de empleados del departamento) no va a volver a consultarla de la BD (a no ser que hagamos un refresh de la entidad)
- Solución: es conveniente siempre actualizar en memoria la colección

```
emp.setDepartamento(depto);  
depto.getEmpleados().add(emp);
```

Actualización de relaciones (1)

- Siempre se sincroniza con la base de datos la entidad que contiene la clave ajena (se le llama entidad propietaria de la relación)
- La otra entidad conviene actualizarla también para mantener la relación consistente en memoria
- Añadiendo un empleado a un proyecto:

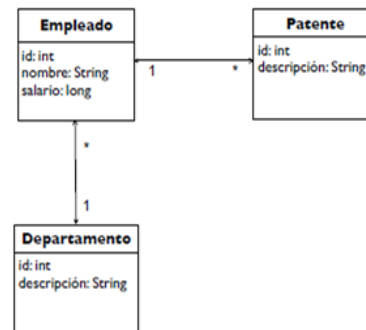
```
Proyecto proyecto = em.find(Proyecto.class, "P04");
Set<Proyectos> proyectos = empleado.getProyectos();
proyectos.add(proyecto);
```

- Cambiando un empleado de proyectos;

```
Empleado empleado = em.find(Empleado.class, "Miguel Garcia");
Proyecto proyectoBaja = em.find(Proyecto.class, "P04");
Proyecto proyectoAlta = em.find(Proyecto.class, "P01");
empleado.getProyectos().remove(proyectoBaja);
empleado.getProyectos().add(proyectoAlta);
```

Carga perezosa

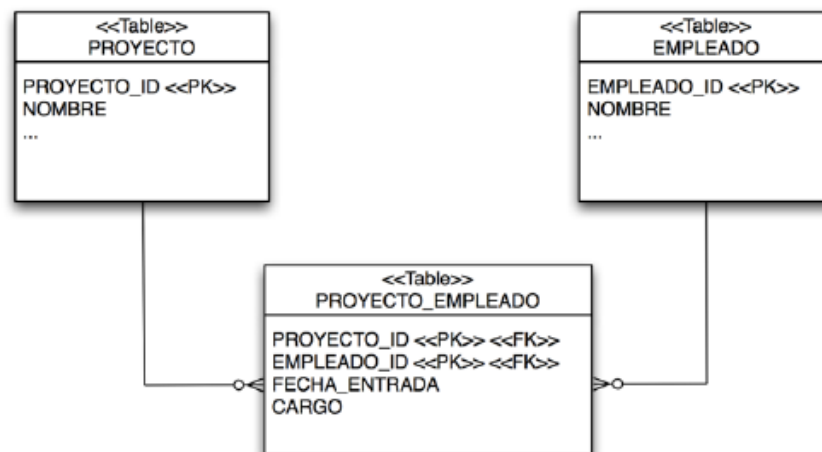
- Al recuperar un departamento sería muy costoso recuperar todos sus empleados
- En las relaciones “a muchos” JPA utiliza la carga perezosa por defecto
- Es posible desactivar este comportamiento con la opción `fetch=FetchType.EAGER` en el tipo de relación
- Un empleado no tiene muchas patentes



```
@Entity
public class Empleado {
    @Id String nombre;
    @OneToMany(fetch=FetchType.EAGER)
    private Set<Patentes> patentes = new HashSet();
    // ...
}
```

Columnas adicionales en la tabla join

- Es muy usual en esquemas heredados usar columnas adicionales en las tablas join

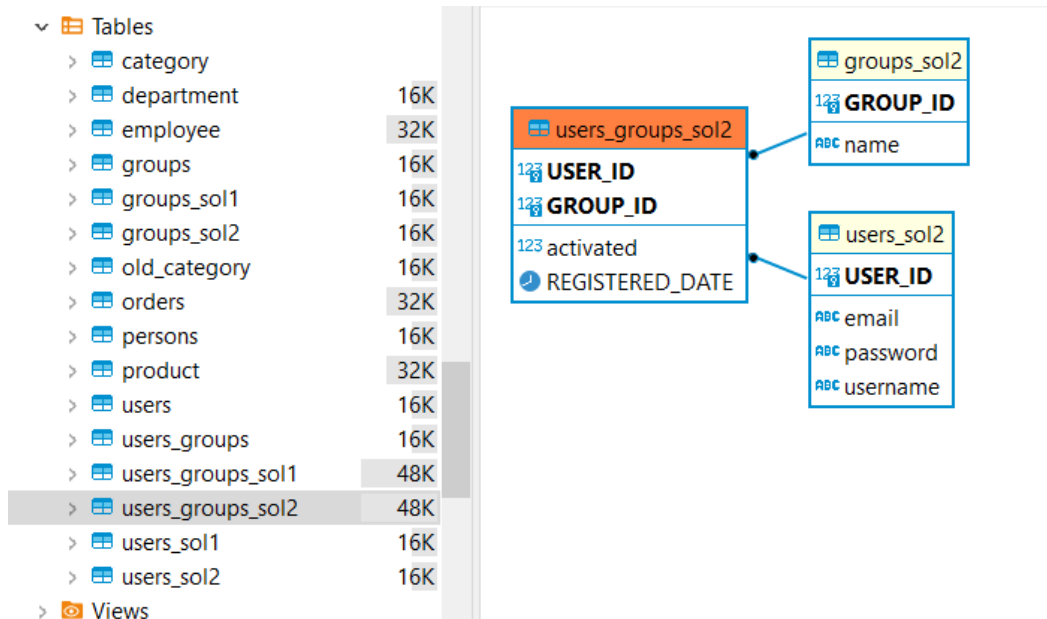
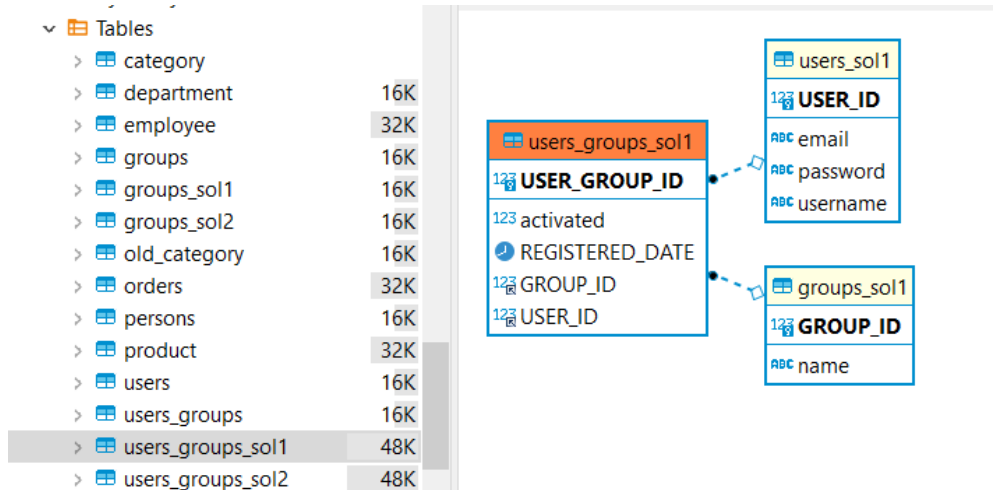


Mapeado

- Hay que crear una entidad adicional (ProyectoEmpleado) con una clave primaria compuesta y una relación uno-a-muchos con las otras entidades (Proyecto y Empleado)
- En esta entidad adicional se mapean los atributos de las claves primarias y los de las relaciones en las mismas columnas, con los atributos `insertable = false` y `updatable = false`

En el proyecto **Many2Many** implementado un ejemplo **Users_Groups** en los paquetes sol1 y sol2.





Archivos configuración

cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<!-- Version 8 MySQL hiberante-cfg.xml example for Hibernate 5 -->
<hibernate-configuration>
    <session-factory>
        <property
name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <!-- property
name="connection.driver_class">com.mysql.jdbc.Driver</property -->
        <property
name="connection.url">jdbc:mysql://localhost:3307/database</property>
        <property
name="dialect">org.hibernate.dialect.MySQL8Dialect</property>
        <property name="connection.username">root</property>
        <property name="connection.password">usbw</property>
        <property name="connection.pool_size">3</property>
        <!--property
name="dialect">org.hibernate.dialect.MySQLDialect</property-->
        <property name="current_session_context_class">thread</property>
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <!-- mapping class="com.mcnz.jpa.examples.Player" / -->
    </session-factory>
</hibernate-configuration>
```

- Con cfg.xml, creación de Hibernate SessionFactory (distinto desde versión Hibernate 5):

```
public static Session getCurrentSessionFromConfig() {
    // SessionFactory in Hibernate 5 example
    Configuration config = new Configuration();
    config.configure();
    // local SessionFactory bean created
    SessionFactory sessionFactory = config.buildSessionFactory();
    Session session = sessionFactory.getCurrentSession();
    return session;
}
```

- Sin cfg.xml, creación de Hibernate SessionFactory:

```
public static Session getCurrentSession() {
    // Hibernate 5.4 SessionFactory example without XML
    Map<String, String> settings = new HashMap<>();
    settings.put("connection.driver_class", "com.mysql.jdbc.Driver");
    settings.put("dialect", "org.hibernate.dialect.MySQL8Dialect");
    settings.put("hibernate.connection.url",
        "jdbc:mysql://localhost:3307/hibernate_examples");
    settings.put("hibernate.connection.username", "root");
    settings.put("hibernate.connection.password", "usbw");
    settings.put("hibernate.current_session_context_class", "thread");
    settings.put("hibernate.show_sql", "true");
    settings.put("hibernate.format_sql", "true");

    ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder()
        .applySettings(settings).build();

    MetadataSources metadataSources = new MetadataSources(serviceRegistry);
    // metadataSources.addAnnotatedClass(Player.class);
    Metadata metadata = metadataSources.buildMetadata();

    // here we build the SessionFactory (Hibernate 5.4)
    SessionFactory sessionFactory =
        metadata.getSessionFactoryBuilder().build();
    Session session = sessionFactory.getCurrentSession();
    return session;
}
```

persistence.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-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd"
  >
  <persistence-unit name="jpa-tutorial">
    <properties>
      <property name="javax.persistence.jdbc.driver"
        value="com.mysql.jdbc.Driver"/>
      <property name="javax.persistence.jdbc.url"
        value="jdbc:mysql://localhost/hibernate_examples"/>
      <property name="hibernate.dialect"
        value="org.hibernate.dialect.MySQL8Dialect" />
      <property name="javax.persistence.jdbc.user"
        value="root"/>
      <property name="javax.persistence.jdbc.password"
        value="password"/>
      <property name="hibernate.show_sql"
        value="true" />
      <property name="hibernate.format_sql"
        value="true" />
    </properties>
  </persistence-unit>
</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.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd"
  >
  <persistence-unit name="jpa-tutorial">
    <properties>
      <property name="javax.persistence.jdbc.driver"
        value="com.mysql.jdbc.Driver"/>
      <property name="javax.persistence.jdbc.url"
        value="jdbc:mysql://localhost/hibernate_examples"/>
      <property name="hibernate.dialect"
        value="org.hibernate.dialect.MySQL8Dialect" />
      <property name="javax.persistence.jdbc.user"
        value="root"/>
      <property name="javax.persistence.jdbc.password"
        value="password"/>
      <property name="hibernate.show_sql"
        value="true" />
      <property name="hibernate.format_sql"
        value="true" />
    </properties>
  </persistence-unit>
</persistence>
```


- Truco para obtener un SessionFactory bean con JPA's EntityManager:

```
public static SessionFactory getCurrentSessionFromJPA() {
    // JPA and Hibernate SessionFactory example
    EntityManagerFactory emf =
        Persistence.createEntityManagerFactory("jpa-tutorial");
    EntityManager entityManager = emf.createEntityManager();
    // Get the Hibernate Session from the EntityManager in JPA
    Session session = entityManager.unwrap(org.hibernate.Session.class);
    SessionFactory factory = session.getSessionFactory();
    return factory;
}
```

- Usando sólo JPA

```
@Entity
public class Player {

    @Id @GeneratedValue
    private Long id;
    private String name;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

```
public static void main(String args[]){
    // EntityManager persist example

    EntityManagerFactory emf =
        Persistence.createEntityManagerFactory("jpa-
tutorial");//persistence.xml
    EntityManager entityManager = emf.createEntityManager();
```

```

try {
    // Hibernate transaction management begins

    entityManager.getTransaction().begin();
    // Transactional hibernate and JPA code goes here

    Player p = new Player();
    p.setPassword("abc123");
    p.setName("Cameron");
    entityManager.persist(p);

    entityManager.getTransaction().commit();

} catch (Exception e) {
    entityManager.getTransaction().rollback();
}
}

```

Usando rollback

HQL

Explicación genérica para uso de HQL en Hibernate.

No se procederá exactamente igual de forma recomendada desde versión Hibernate5. Aunque los métodos estén deprecated, siguen operativos.

- **Genérico y sobre todo operativo en versiones anteriores a Hibernate 5**

Para realizar **consultas en hibernate**, podemos utilizar varios métodos:

1- **Método get()** :puede retornar un solo objeto ó bien una colección, aplicando el método iterator(),ej:

```

public static void libro(BufferedReader lee) throws IOException {
    Libro l = null; Session session;
    boolean existe = Ver.verLibros();
    if (existe) {
        do {
            System.out.println("Introduce el código del libro: ");
            int codigo = Integer.parseInt(lee.readLine());
            session = NewHibernateUtil.getSession();
            l = (Libro) session.get(Libro.class, codigo);

            if (l == null) {
                System.out.println("No hay ningún libro con ese código");
            }
            session.close();
        } while (l == null);

        Ver.verLibro(l);
    }
}

```

2-

- a) **CreateQuery()** retorna una colección con el método list().
- b) Puede **retornar un solo resultado** con método uniqueResult():

```

{
    System.out.println("----- Uso de list() con un único dato escalar -----");
    List<Object[]> listDatos = session.createQuery("SELECT p.nombre FROM Profesor p").list();

    for (Object datos : listDatos) {
        System.out.println(datos);
    }
}

{
    System.out.println("----- Uso de uniqueResult -----");
    Profesor profesor = (Profesor) session.createQuery("SELECT p FROM Profesor p WHERE id=101").uniqueResult();
    System.out.println("Profesor con Id 101=" + profesor.getNombre());
}

```

3-Método createCriteria(), al igual que el anterior puede retornar una colección ó un solo objeto:

En las nuevas implementaciones se usará directamente createQuery y se mapeará la lista resultante.

```
public static void listaTalleres(SessionFactory sf){

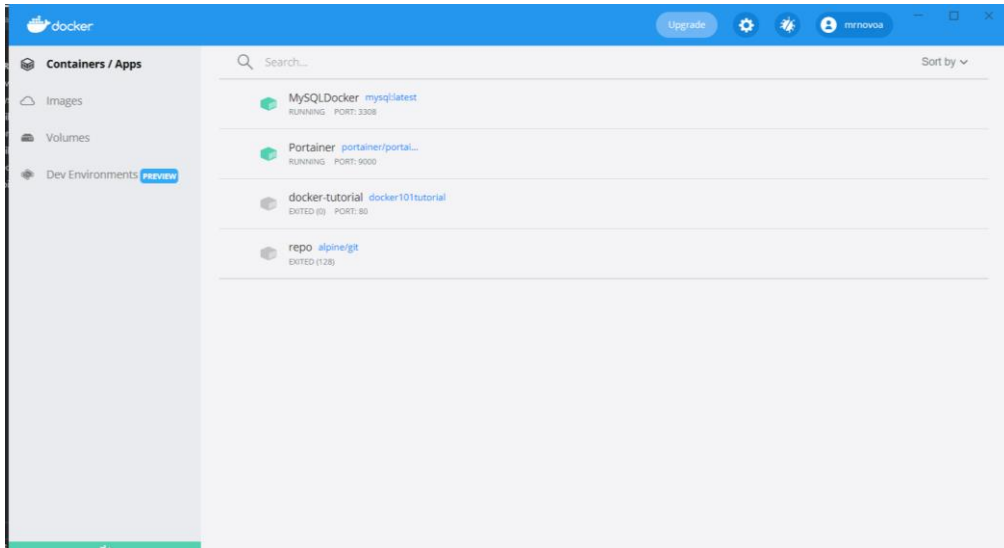
    Session session = sf.openSession();
    //List<Talleres> lista = session.createCriteria(Talleres.class).list();

    String c= "select t from Talleres t";
    List<Talleres> lista= session.createQuery(c).list();

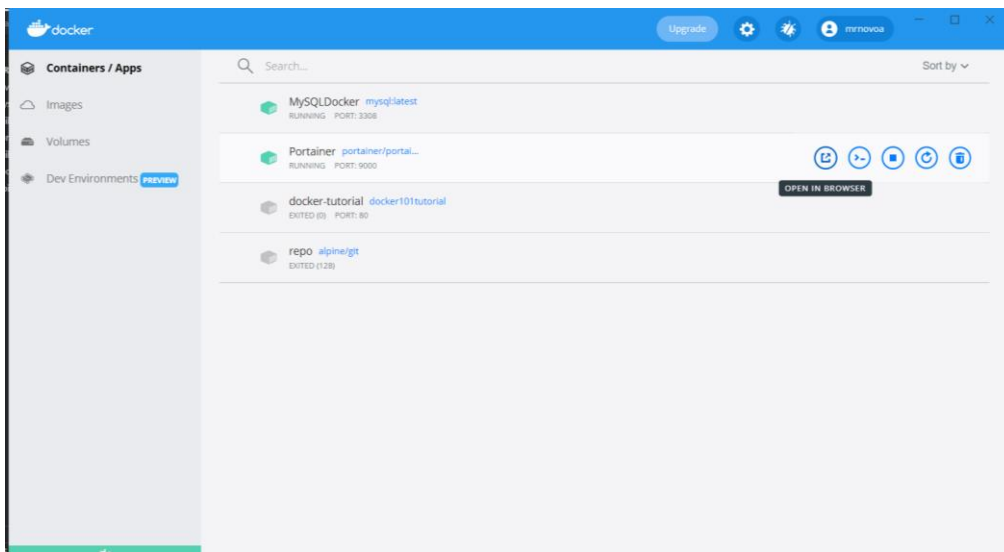
    System.out.println("-----LISTA DE TALLERES---");
    System.out.printf("%-4s %-30s %n", "ID", "NOMBRE");
    for(Talleres i: lista){
        System.out.printf("%-4s %-30s %n", i.getCodigo(), i.getNombre());
    }
}
```

Docker

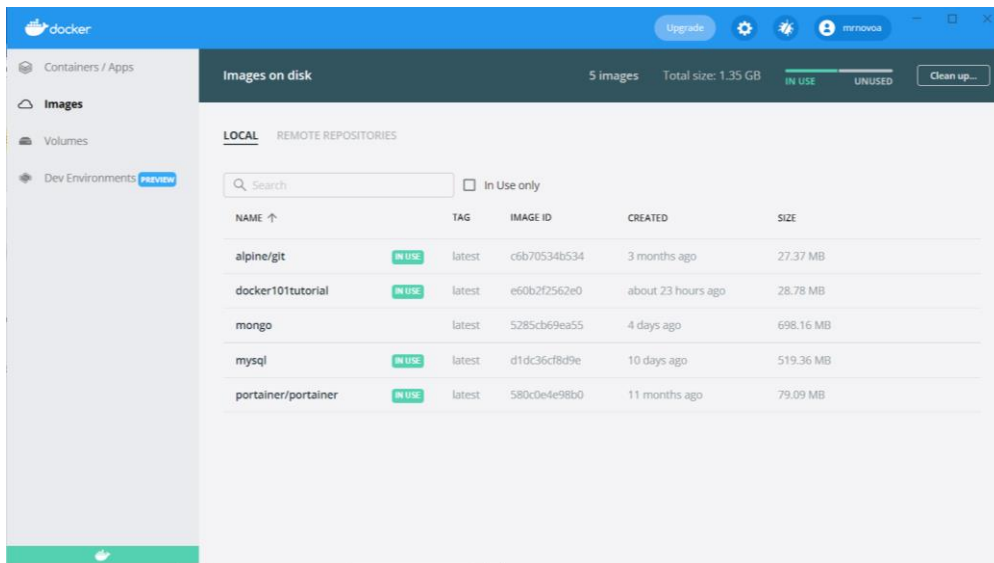
Desktop



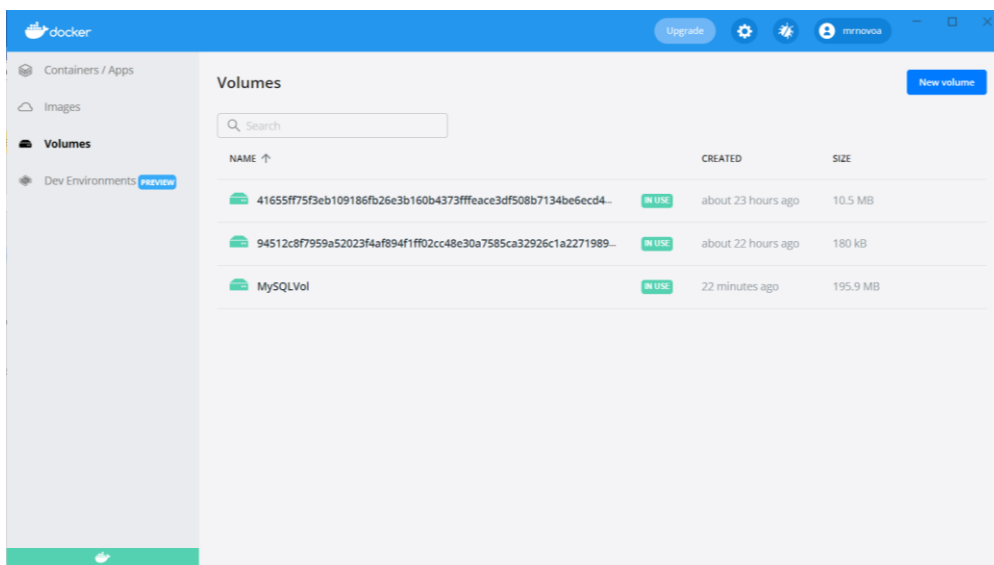
- Contenedores



- Imágenes



- Volúmenes



Portainer

En PWSHELL

```
docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock portainer/portainer
```

The top screenshot shows the Portainer web interface at localhost:9000/#/home. The sidebar menu includes options like Dashboard, App Templates, Stacks, Containers, Images, Networks, Volumes, Events, Host, SETTINGS, Extensions, Users, Endpoints, Registries, and Settings. The main content area displays 'Endpoints' with a 'Refresh' button and a search bar. A summary for the 'local' endpoint shows 0 stacks, 4 containers, 2 volumes, and 5 images. The bottom screenshot shows the Portainer web interface at localhost:9000/#/images. The sidebar menu is the same. The main content area displays 'Image list' with a 'Pull image' form and a table of images. The table has columns for Name, Tags, Size, and Created. The images listed are:

Name	Tags	Size	Created
alpa254: c6b70534b534597a0b938ba5374...	alpa254/gpt-intent	27.4 MB	2023-05-20 06:10:20
alpa254: e46b2f2862e0b79e6f00271f9db0...	alpa254/structured-qa	28.8 MB	2023-05-05 14:20:41
alpa254: 5285db49a5547dca0ba973d939a...	alpa254/intent	698.2 MB	2023-05-02 04:55:36
alpa254: d1d03cf9b7b0d092b0ba41d0c0eb...	alpa254/intent	519.4 MB	2023-05-27 09:58:36
alpa254: 590d0e4a98b066259796cf28a55721...	portainer/portainer-intent	79.1 MB	2023-05-18 20:53:42

localhost:9000/#/containers

portainer.io Containers

Container list

Containers

Start Stop Kill Restart Pause Resume Remove Add container

Search...

Name	State <small>id</small>	Quick actions	Stack	Image	Created	Published Ports	Ownership
MySQLDocker	running	Stop Kill Restart Pause Resume Remove	-	mysql:latest	2022-02-06 12:30:51	3308:3306	administrators
Portainer	running	Stop Kill Restart Pause Resume Remove	-	portainer/portainer	2022-02-05 15:17:45	9000:9000	administrators
docker-tutorial	stopped	Start Stop Kill Restart Pause Resume Remove	-	docker101/tutorial	2022-02-05 14:22:02	-	administrators
repo	stopped	Start Stop Kill Restart Pause Resume Remove	-	alpine/git	2022-02-05 14:19:29	-	administrators

Items per page 10

A new version is available

portainer.io 1.24.2

- Templates

localhost:9000/#/templates











portainer.io

+ Add template

Select a category

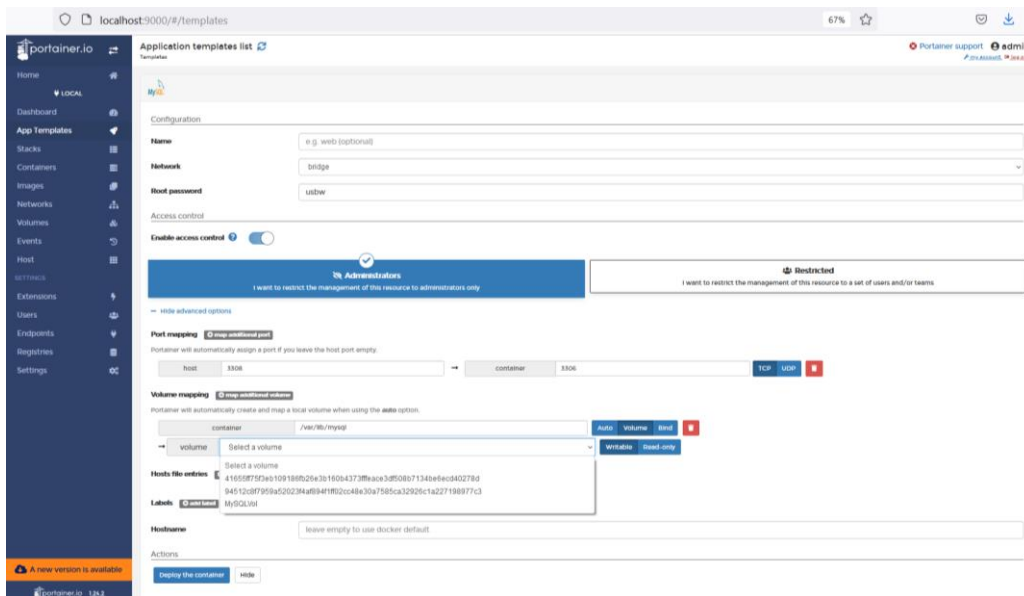
Show container templates ☐

Search...

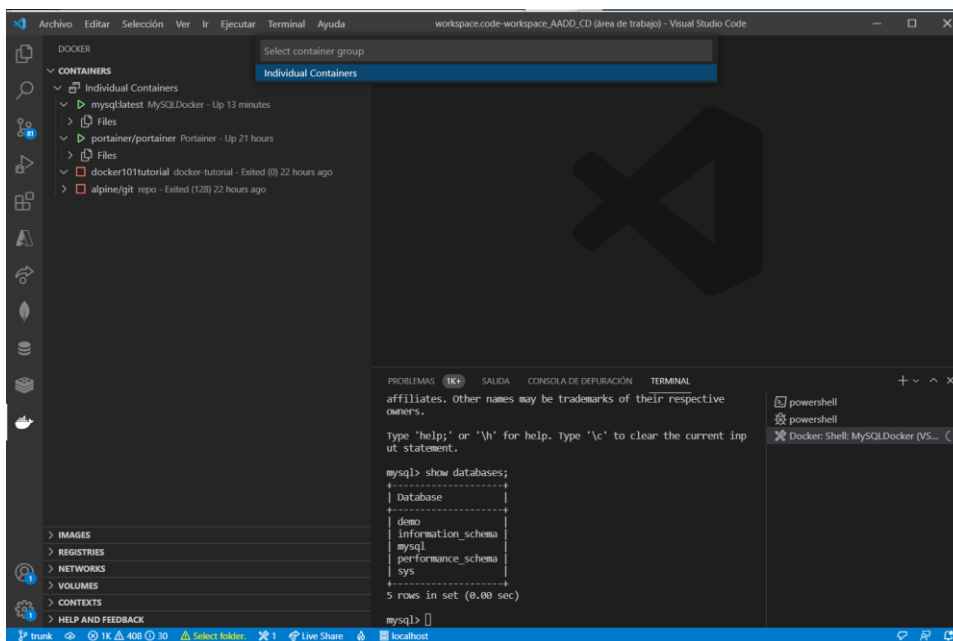
	Registry <small>container</small>	Update Delete	docker
	Nginx <small>container</small>	Update Delete	webserver
	Httpd <small>container</small>	Update Delete	webserver
	Caddy <small>container</small>	Update Delete	webserver
	MySQL <small>container</small>	Update Delete	database
	MariaDB <small>container</small>	Update Delete	database
	PostgreSQL <small>container</small>	Update Delete	database
	Mongo <small>container</small>	Update Delete	database
	CockroachDB <small>container</small>	Update Delete	database
	CrateDB <small>container</small>	Update Delete	database

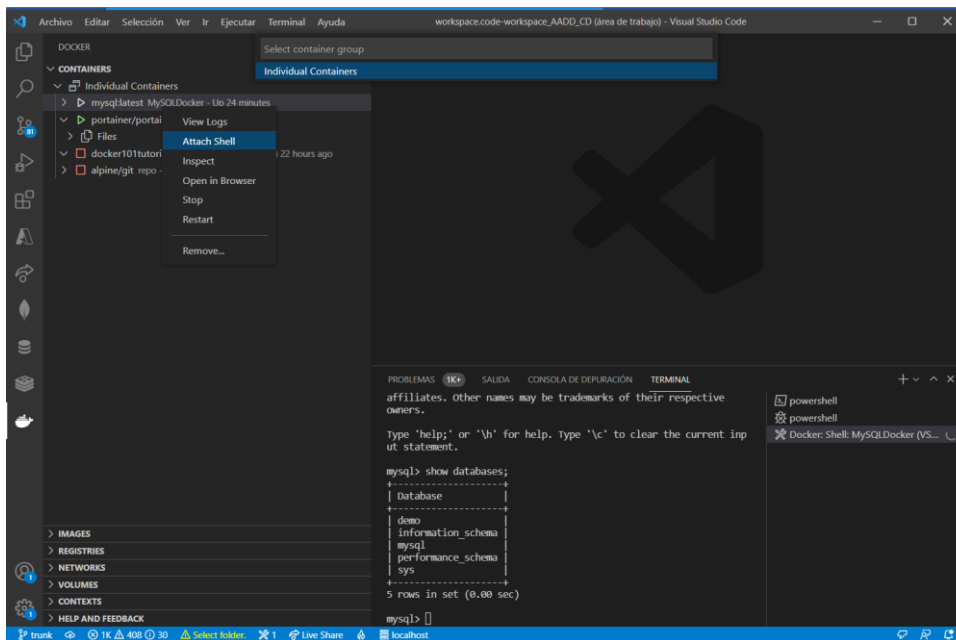
A new version is available

portainer.io 1.24.2



VisualStudioCode





- PowerShell

mysql -p

DBeaver

DBeaver 21.3.0 - <MySQL- localhost_usbWebServ> Script

Archivo Editar Navegar Buscar Editor SQL Base de Datos Ventana Ayuda

SQL Commit Rollback Auto MySQL- local ... t_usbWebServ ejemplo

Navegador de Bas... x Proyectos x <MySQL- loca... <MySQL- loca... <MySQL- loca... x 34

select now();

Ingrese parte del nom

- > (SQLite) - DB
- > Autores.mv.db
- > bdH2_hb_eclipse
- > Empresa.mv.db
- > librosApache
- > librosH2.mv.db
- > librosHSQLdb.m...
- > librosSQLite.db
- > MySQL- localhos
- > Databases
- > Users
- > Administer
- > System Info
- > MySQLDocker_lo

Ajustes de MySQL

Configuración de la conexión "MySQLDocker_localhost"

Editar driver 'MySQL'

Settings Librerías Propiedades del driver Parámetros avanzados

Name	Value
connectTimeout	20000
enabledTLSProtocols	TLSv1,TLSv1.1,TLSv1.2
rewriteBatchedStatements	true
useSSL	false
allowPublicKeyRetrieval	true

Reset to Defaults Aceptar Cancelar

You can use variables in connection parameters.

Driver name: MySQL Edit Driver Settings

Probar conexión ... Aceptar Cancelar

escribir texto de filtro

Descripción	Complemento	Fecha
Public Key Retrieval is not allowed	org.jkiss.dbeaver.m...	6/2/22 12:40
Public Key Retrieval is not allowed	org.jkiss.dbeaver.m...	6/2/22 12:40
Public Key Retrieval is not allowed	org.jkiss.dbeaver.m...	6/2/22 12:40
Public Key Retrieval is not allowed	org.jkiss.dbeaver.m...	6/2/22 12:40
Public Key Retrieval is not allowed	org.jkiss.dbeaver.m...	6/2/22 12:39

MySQLDocker_localhost CET es Editable Inserción inteligente 1