

BBDD

```
-- Host: 127.0.0.1
-- Versión del servidor: 10.4.6-MariaDB - mariadb.org binary distribution
-- SO del servidor: Win64
-- HeidiSQL Versión: 10.2.0.5599
-----

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET NAMES utf8 */;
/*!50503 SET NAMES utf8mb4 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;

-- Volcando estructura de base de datos para bd_empleados_hibernate_2019_encrypt
DROP DATABASE IF EXISTS `bd_empleados_hibernate_2019_encrypt`;
CREATE DATABASE IF NOT EXISTS `bd_empleados_hibernate_2019_encrypt` /*!40100 DEFAULT CHARACTER SET utf8 */;
USE `bd_empleados_hibernate_2019_encrypt`;

-- Volcando estructura para tabla bd_empleados_hibernate_2019_encrypt.departamento
DROP TABLE IF EXISTS `departamento`;
CREATE TABLE IF NOT EXISTS `departamento` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `codigo` varchar(2) NOT NULL,
  `descripcion` varchar(50) NOT NULL,
  `localizacion` varchar(50) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `codigo` (`codigo`)
) ENGINE=InnoDB AUTO_INCREMENT=12 DEFAULT CHARSET=utf8;

-- Volcando datos para la tabla bd_empleados_hibernate_2019_encrypt.departamento: ~6 rows (aproximadamente)
DELETE FROM `departamento`;
/*!40000 ALTER TABLE `departamento` DISABLE KEYS */;
INSERT INTO `departamento` (`id`, `codigo`, `descripcion`, `localizacion`) VALUES
(1, '10', 'Ventas', 'Madrid'),
(2, '20', 'Contabilidad', 'Valencia'),
(3, '30', 'Logística', 'Albacete'),
(4, '40', 'Gerencia', 'Madrid'),
(8, '50', 'I&D', 'Alicante'),
(11, '60', 'Ventas', 'Granada');
/*!40000 ALTER TABLE `departamento` ENABLE KEYS */;

-- Volcando estructura para tabla bd_empleados_hibernate_2019_encrypt.empleado
DROP TABLE IF EXISTS `empleado`;
CREATE TABLE IF NOT EXISTS `empleado` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `codigo` varbinary(50) NOT NULL,
  `nif` varchar(9) NOT NULL,
  `password` varchar(200) NOT NULL,
  `apellidos` varchar(25) NOT NULL,
  `nombre` varchar(15) NOT NULL,
  `profesion` varchar(15) NOT NULL,
  `salario` double DEFAULT NULL,
  `fechaNac` date NOT NULL,
  `fechaIng` date NOT NULL,
  `idDepartamento` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `nif` (`nif`),
  UNIQUE KEY `codigo` (`codigo`),
  KEY `FK_idx` (`idDepartamento`),
  CONSTRAINT `FK` FOREIGN KEY (`idDepartamento`) REFERENCES `departamento` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=31 DEFAULT CHARSET=utf8;

-- Volcando datos para la tabla bd_empleados_hibernate_2019_encrypt.empleado: ~3 rows (aproximadamente)
DELETE FROM `empleado`;
/*!40000 ALTER TABLE `empleado` DISABLE KEYS */;
INSERT INTO `empleado` (`id`, `codigo`, `nif`, `password`, `apellidos`, `nombre`, `profesion`, `salario`, `fechaNac`, `fechaIng`, `idDepartamento`) VALUES
(28, _binary 0x2CEFFBFD7259EFBFBDEFBFBDEFBFD3D75160F4056EFBFD05EFBFD, '99987765D', '1b47e87d02781e58a171f3221a7e4c032e67f2d0', 'López Pérez', 'Diego', 'Vendedor', 27000, '1991-01-18', '2019-12-20', 11),
(29, _binary 0xEFFBFBDEFBFBDEFBFBDEFBFBDEFBFD6F48EFBFD343C3C4FCA9F7E2FEFBFD, '08861760T', 'e7097cce199f6eda0670bf5c2486f94d18406983', 'Antúnez Picazo', 'Luisa', 'Vendedor', 27000, '1996-05-29', '2019-12-20', 11),
(30, _binary 0xEFFBFBDEFBFBDEFBFBDEFBFBDEFBFD2B3A5B103BEFBFD5D3BEFBFBDEFBFBFD69, '22117719B', '61632e2a55f7e115822cc70c24cf22542a4720b5', 'Díaz Querol', 'Emilia', 'Vendedor', 27000, '1983-11-10', '2019-12-20', 11);
/*!40000 ALTER TABLE `empleado` ENABLE KEYS */;

-- Volcando estructura para vista bd_empleados_hibernate_2019_encrypt.emple_depart
DROP VIEW IF EXISTS `emple_depart`;
-- Creando tabla temporal para superar errores de dependencia de VIEW
CREATE TABLE `emple_depart` (
  `id` INT(11) NOT NULL,
  `nif` VARCHAR(9) NOT NULL COLLATE 'utf8_general_ci',
  `nombre` VARCHAR(15) NOT NULL COLLATE 'utf8_general_ci',
  `apellidos` VARCHAR(25) NOT NULL COLLATE 'utf8_general_ci',
  `descripcion` VARCHAR(50) NOT NULL COLLATE 'utf8_general_ci',
  `localizacion` VARCHAR(50) NOT NULL COLLATE 'utf8_general_ci'
) ENGINE=MyISAM;

-- Volcando estructura para vista bd_empleados_hibernate_2019_encrypt.emple_depart
DROP VIEW IF EXISTS `emple_depart`;
-- Eliminando tabla temporal y crear estructura final de VIEW
DROP TABLE IF EXISTS `emple_depart`;
CREATE ALGORITHM=UNDEFINED DEFINER='root'@'localhost' SQL SECURITY DEFINER VIEW `emple_depart` AS select `e`.`id` AS `id`,`e`.`nif` AS `nif`,`e`.`nombre` AS `nombre`,`e`.`apellidos` AS `apellidos`,`d`.`descripcion` AS `descripcion`,`d`.`localizacion` AS `localizacion` from `empleado` `e` join `departamento` `d` where (`e`.`idDepartamento` = `d`.`id`) order by `e`.`id` ;

/*!40101 SET SQL_MODE=IFNULL(@OLD_SQL_MODE, '') */;
/*!40014 SET FOREIGN_KEY_CHECKS=IF(@OLD_FOREIGN_KEY_CHECKS IS NULL, 1, @OLD_FOREIGN_KEY_CHECKS) */;
/*!40101 SET CHARACTER SET CLIENT=@OLD_CHARACTER_SET_CLIENT */;
```

MUY IMPORTANTE

- ✓ Los id de las tablas son AUTOINCREMENT, si no, no funciona.
- ✓ Password debe ser VARCHAR(20) para cifrar con HASH.
- ✓ Código utiliza VARBINARY porque cifra con AES.

HASH // AES

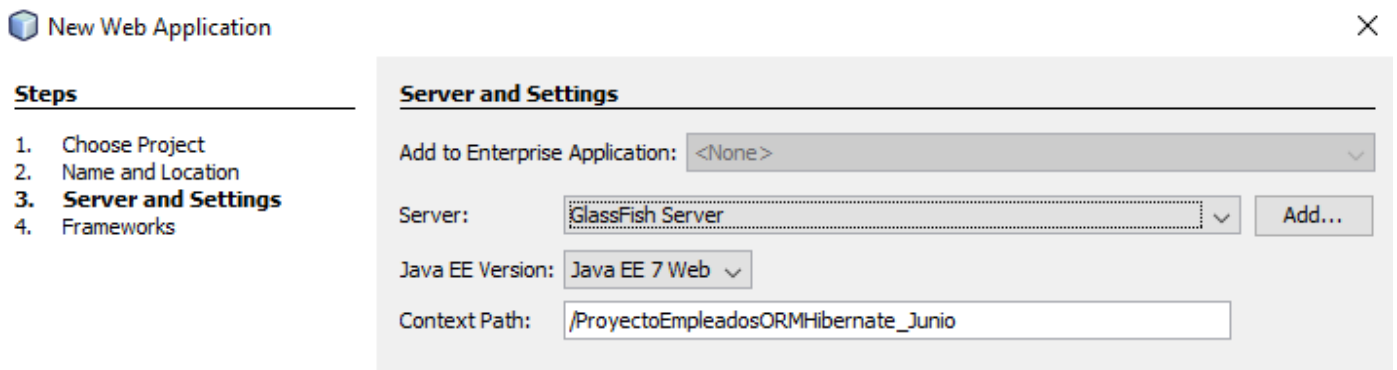
Utilizaremos HASH para encriptar en un solo sentido y AES para encriptar y desencriptar.

Además, la base de datos contiene una vista que genera el listado de empleados con la descripción del departamento, al que pertenecen.

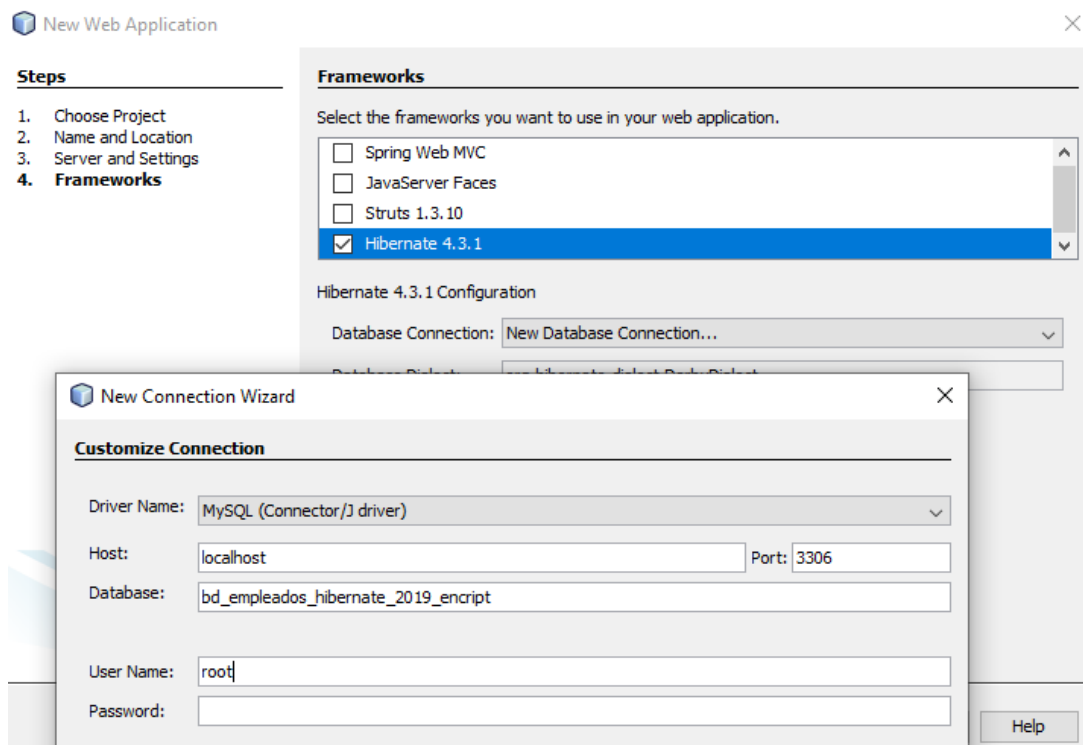
CREAR PROYECTO EN NETBEANS CON HIBERNATE

Abrimos NetBeans y comprobamos que tenemos el servidor GlassFish, que es el que vamos a utilizar.

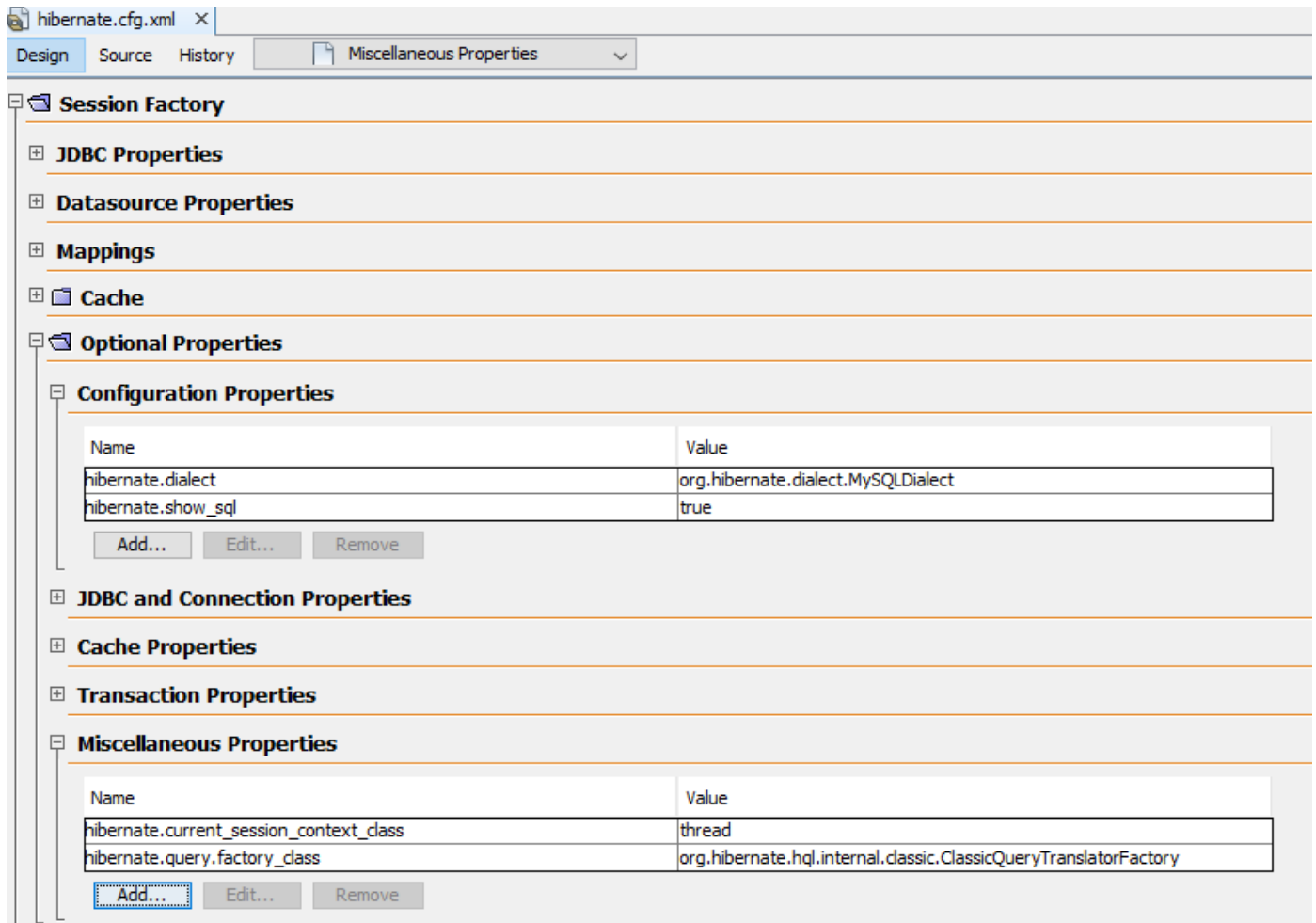
Creamos un proyecto JAVA WEB, le asignamos su nombre y conectamos el proyecto con GlassFish Server.



Después, en Frameworks encontraremos Hibernate que es donde vamos a conectar con la base de datos.



Si se nos ha creado correctamente, vamos a los archivos de configuración y modificamos los siguientes. El archivo de configuración es hibernate.cfg.xml, que se encuentra dentro de la carpeta por defecto de Source Packages. Se debe quedar así:



The screenshot shows the Eclipse IDE with the file `hibernate.cfg.xml` open. The 'Miscellaneous Properties' tab is selected. The 'Optional Properties' section is expanded, showing the following tables:

Name	Value
<code>hibernate.dialect</code>	<code>org.hibernate.dialect.MySQLDialect</code>
<code>hibernate.show_sql</code>	<code>true</code>

Below the table are buttons: `Add...`, `Edit...`, and `Remove`.

Name	Value
<code>hibernate.current_session_context_class</code>	<code>thread</code>
<code>hibernate.query.factory_class</code>	<code>org.hibernate.hql.internal.classic.ClassicQueryTranslatorFactory</code>

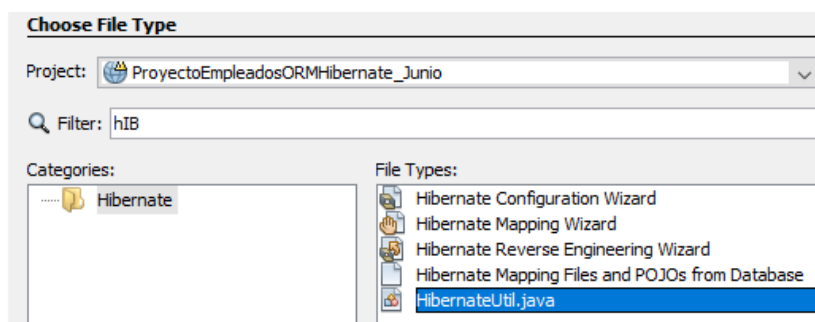
Below the table are buttons: `Add...`, `Edit...`, and `Remove`.

Volvemos a comprobar en la pestaña Source que los cambios se han hecho correctamente.

```
<hibernate-configuration>
<session-factory>
  <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
  <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
  <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/bd_empleados_hibernate_2019_encrypt?zeroDateTimeBehavior=convertToNull</property>
  <property name="hibernate.connection.username">root</property>
  <property name="hibernate.show_sql">true</property>
  <property name="hibernate.current_session_context_class">thread</property>
  <property name="hibernate.query.factory_class">org.hibernate.hql.internal.classic.ClassicQueryTranslatorFactory</property>
</session-factory>
</hibernate-configuration>
```

CREAR LA CAPA DAO DENTRO DE SOURCE PACKAGES

En esta capa vamos a tener la conexión a la base de datos, que será con el archivo `HibernateUtil.java`, por ahora. Dentro de `HibernateUtil`, vamos a tener objetos estáticos de la clase `SessionFactory`, que son las que se utilizan para cada conexión a la base de datos.

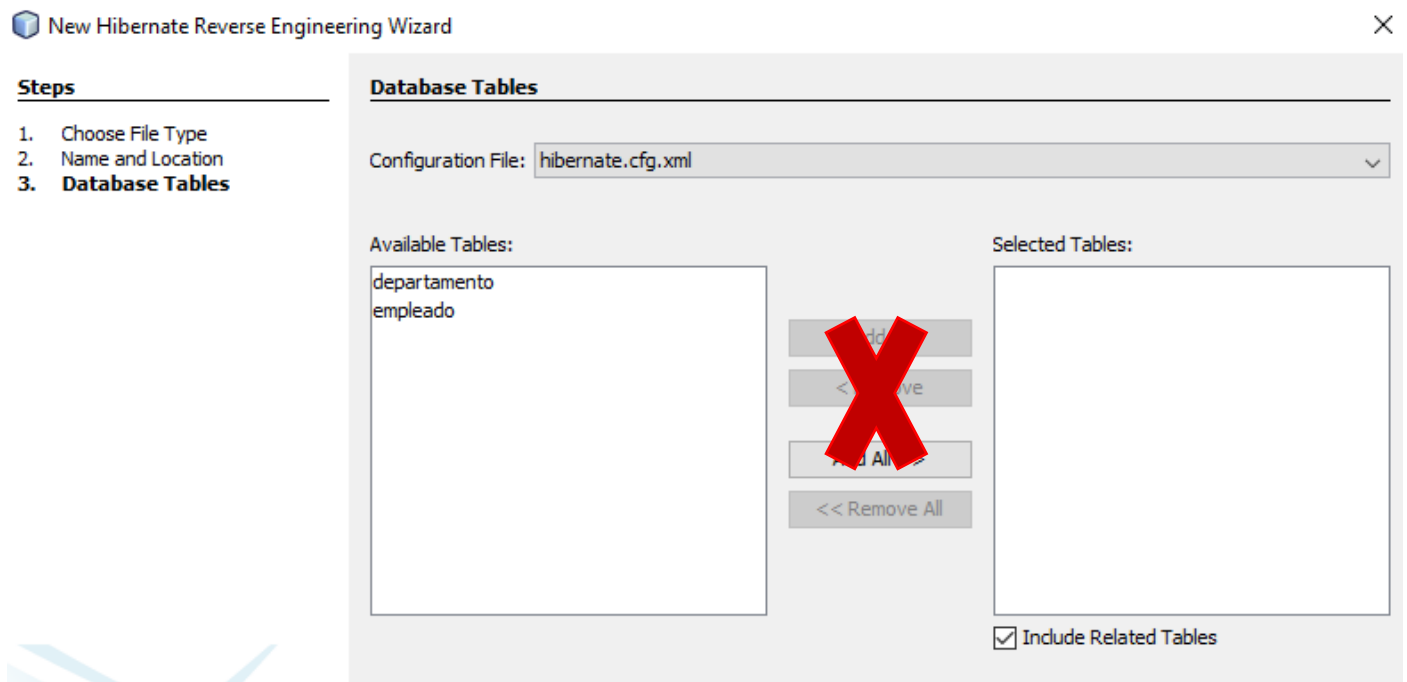


The screenshot shows the 'Choose File Type' dialog in Eclipse. The 'Project' is set to 'ProyectoEmpleadosORMHibernate_Junio'. The 'Filter' is set to 'hib'. The 'Categories' list shows 'Hibernate'. The 'File Types' list shows several options, with 'HibernateUtil.java' selected.

CREAMOS EL ARCHIVO HIBERNATE.REVENG.XML

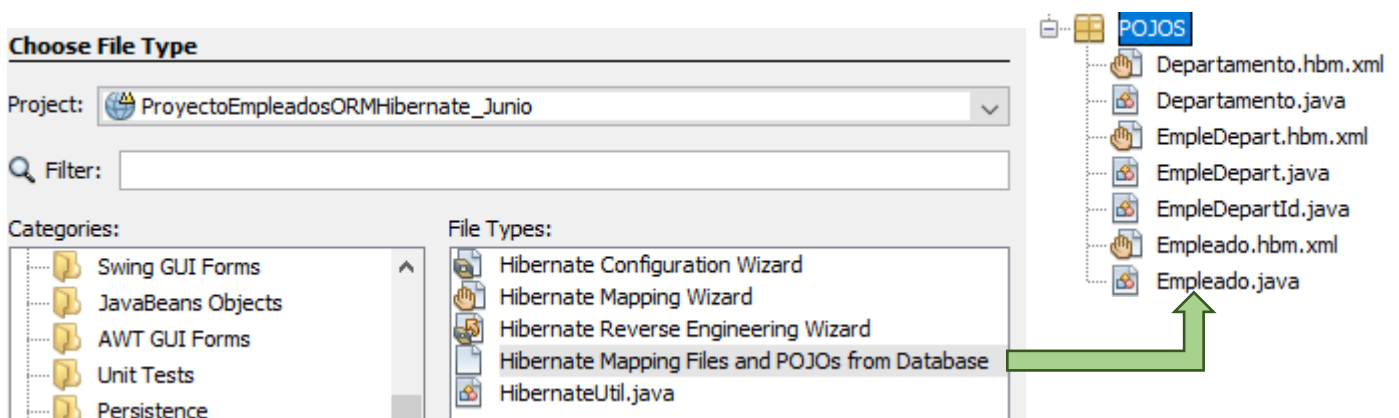
Este fichero es el de configuración del mapeo. Se relaciona con hibernate.cfg.xml y con las tablas a mapear.

Si hay vistas en la base de datos, no se debe incluir ninguna tabla, simplemente pulsamos NEXT. En este caso, debemos hacer esto porque contiene vistas.



CREAMOS LOS POJOS

Con esto creamos una clase Java por cada tabla y un fichero de configuración XML que relaciona la tabla con la clase.



Se añaden automáticamente el mapeo de la vista al hibernate.cfg.xml. Compruebo que se crean correctamente.

```
<mapping resource="POJOS/EmpleDepart.hbm.xml"/>
<mapping resource="POJOS/Empleado.hbm.xml"/>
<mapping resource="POJOS/Departamento.hbm.xml"/>
```

EJERCICIOS Y MODIFICACIONES

Antes de nada, hay que crear el paquete BLL, que va a contener todos los controladores que vamos a utilizar. Los controladores en esta ocasión son Servlet, los podemos encontrar cuando creamos un archivo dentro de la carpeta Web.

También, hay que crear una Java Class dentro del paquete DAO, que será donde tenemos nuestros métodos, la mía se va a llamar DAOoperaciones.

Después, la conexión se establece añadiendo dentro de cada controlador esto:

```
private SessionFactory SessionBuilder;

@Override
public void init() {
    SessionBuilder = HibernateUtil.getSessionFactory();
}
```

DEPARTAMENTO

ALTA SIN EMPLEADOS

Creamos el primer controlador, le añadimos la conexión lo primero y empezamos. Vamos a insertar un departamento sin empleados, vamos a hacer los datos manualmente.

Lo primero que vamos a hacer es crear un objeto de tipo departamento, y con el constructor que ya contiene la clase Departamento, añadimos manualmente los valores.

Después dentro de un try-catch, invocamos al método que se encuentra en el DAOOperaciones, que en este caso será insertarDepartamento. A este método le pasamos el objeto de departamento y la conexión. Si todo ha salido correctamente, nos llevará a una vista que nos dirá que está bien y si no, saltará el catch.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        Departamento ObjDepart = new Departamento("22", "Informatica", "Socuellamos");
        try {
            new DAOOperaciones().insertarDepartamento(ObjDepart, SessionBuilder);
            response.sendRedirect("../VISTAS/Correcto.jsp?codigo=departamento");
        } catch (HibernateException HE) {
            response.sendRedirect("../VISTAS/Error.jsp?codigo=departamento");
        }
    }
}
```

Dentro del DAOOperaciones, la mecánica es casi siempre la misma. En este caso, usamos session.save(), porque queremos insertar un departamento. No vamos a tener problemas, porque se pueden añadir departamentos sin empleados.

```
public void insertarDepartamento(Departamento _ObjDepar, SessionFactory _SessionBuilder) {
    Session session = _SessionBuilder.openSession();
    Transaction Tx = null;
    try {
        Tx = session.beginTransaction();
        session.save(_ObjDepar);
        Tx.commit();
    } catch (HibernateException HE) {
        HE.printStackTrace();
        if (Tx != null) {
            Tx.rollback();
        }
        throw HE;
    } finally {
        session.close();
    }
}
```

Por último, debemos comprobar en la base de datos que se ha realizado correctamente.

 id	 codigo	descripcion	localizacion
1	10	Ventas	Madrid
2	20	Contabilidad	Valencia
3	30	Logística	Albacete
4	40	Gerencia	Madrid
8	50	I&D	Alicante
11	60	Ventas	Granada
23	22	Informatica	Socuellamos

ALTA CON EMPLEADOS ASOCIADOS

En este ejercicio, vamos a dar de alta un departamento y a su misma vez asociarle un empleado. Creamos el controlador y le añadimos la conexión.

Empezamos creando un objeto Departamento y pasándole sus valores correspondientes. Hacemos lo mismo para crear un empleado, pero aquí le asignamos ya el departamento.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        //CREAMOS EL OBJETO DEPARTAMENTO CON SUS VALORES
        Departamento objDepartamento = new Departamento("73", "Moda", "Albacete");

        //CREAMOS EL OBJETO EMPLEADO CON SUS VALORES Y LE ASIGNAMOS EL DEPARTAMENTO
        Empleado objEmpleado1 = new Empleado(objDepartamento, ("123456879".getBytes(StandardCharsets.UTF_8)),
            "06290542F", Hash.sha1("Edu"), "Mondejar Sevillano", "Pepe", "Futbolista",
            2540.2, java.sql.Date.valueOf(LocalDate.of(1998, 12, 3)),
            java.sql.Date.valueOf(LocalDate.now()));

        HashSet departamento = new HashSet();
        departamento.add(objEmpleado1);
        objDepartamento.setEmpleados(departamento);

        try {
            new DAOOperaciones().insertarDepartamentoEmpleados(objDepartamento, SessionBuilder);
            out.println("SE HA INSERTADO EL EMPLEADO JUNTO A SU DEPARTAMENTO NUEVO");
        } catch (HibernateException he) {
            out.println("ERROR HIBERNATE INSERCIÓN DEPARTAMENTO CON EMPLEADOS");
        }
    }
}
```

Para asignar al departamento un empleado, lo hacemos con el setEmpleados, desde el método que contiene la clase Departamento.

Por otra parte, tenemos las fechas, que cuando se mapean se quedan como DATE, pero es muy enredoso, por lo que las convertimos en LOCALDATE, con la sentencia java.sql.Dateof(LocalDate.of(...)).

Tenemos que recordar que la contraseña del empleado está cifrada con HASH y para ello necesitamos una clase que nos proporciona el profesor para poder usar el método sha1 de HASH. Por lo que, creamos la clase dentro de un paquete llamado CLASES.

```
package CLASES;
public class Hash {
    public static String getHash(String txt, String hashType) {
        try {
            java.security.MessageDigest md
                = java.security.MessageDigest
                    .getInstance(hashType);
            byte[] array = md.digest(txt.getBytes());
            StringBuffer sb = new StringBuffer();
            for (int i = 0; i < array.length; ++i) {
                sb.append(Integer.toHexString((array[i] & 0xFF) | 0x100)
                    .substring(1, 3));
            }
            return sb.toString();
        } catch (java.security.NoSuchAlgorithmException e) {
            System.out.println(e.getMessage());
        }
        return null;
    }
    /* Retorna un hash MD5 a partir de un texto */

    public static String md5(String txt) {
        return Hash.getHash(txt, "MD5");
    }
    /* Retorna un hash SHA1 a partir de un texto */

    public static String sha1(String txt) {
        return Hash.getHash(txt, "SHA1");
    }
}
```

En cuanto al código del empleado, está cifrado por AES, tenemos que ir al archivo Empleado.xml, la siguiente sentencia que muestro a continuación. Con esto podemos encriptar y desencriptar lo que contenga AES.

```
<property name="codigo" type="binary">
    <column name="codigo" not-null="true" unique="true"
        read="AES_DECRYPT(codigo, 'anabel')" write="AES_ENCRYPT(?, 'anabel')"/>
</property>
```

Y, además, para insertar un departamento junto a sus empleados, tenemos que poner CASCADE="ALL" también.

```
<set name="empleados" table="empleado" cascade="all" inverse="true" lazy="true" fetch="select">
    <key>
        <column name="idDepartamento"/>
    </key>
    <one-to-many class="POJOS.Empleado"/>
</set>
```

Comprobamos en la base de datos, que se haya insertado correctamente.

id	codigo	descripcion	localizacion
1	10	Ventas	Madrid
2	20	Contabilidad	Valencia
3	30	Logística	Albacete
4	40	Gerencia	Madrid
8	50	I&D	Alicante
11	60	Ventas	Granada
23	22	Informatica	Socuellamos
28	73	Deportes	Casas Ibañez

id	codigo	nif	password	apellidos	nombre	profesion	salario	fechaNac	fechaIng	i...
28	0x2CEFBFBD7...	99987765D	1b47e87d027...	López Pérez	Diego	Vendedor	27.000	1991-01-18	2019-12-20	11
29	0xEFBBDEFB...	08861760T	e7097cce199f...	Antúñez Picazo	Luisa	Vendedor	27.000	1996-05-29	2019-12-20	11
30	0xEFBBDEFB...	22117719B	61632e2a55f7...	Diaz Querol	Emilia	Vendedor	27.000	1983-11-10	2019-12-20	11
31	0xC278A35A9...	06290542F	af49beb4802f...	Mondejar Sevillano	Eduardo	Futbolista	2.540,2	1998-12-03	2020-04-23	28

BAJA CON EMPLEADOS ADSCRITOS. SELECCIÓN POR CÓDIGO

En este apartado, vamos a dar de baja un departamento mediante su código y a su misma vez, se borrarán los empleados.

Como seguimos teniendo el CASCADE="ALL", en el archivo Departamento.xml, si borramos un departamento, borramos todo lo asociado a él.

Por lo que, creamos otro controlador, y esta vez no hace falta crear un objeto, simplemente le pasamos una variable de tipo String, que contenga el código del departamento que queremos.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        String codDepart = "73";
        try {
            new DAOOperaciones().borrarDepartamento(codDepart, SessionBuilder);
            response.sendRedirect("./VISTAS/Correcto.jsp?codigo=departamento-borrar-con-empleados");
        } catch (HibernateException HE) {
            response.sendRedirect("./VISTAS/Error.jsp?codigo=departamento-borrar-con-empleados");
        }
    }
}
```

En el método dentro del DAOOperaciones, utilizamos la sentencia `sesion.delete()`. En este caso, descubrimos que se pueden utilizar sentencias HQL. Le pasamos la variable y creamos un objeto, el cual vamos a borrar.

```
public void borrarDepartamento(String _codDepartamento, SessionFactory _SessionFactory) {
    Session session = _SessionFactory.openSession();
    Transaction Tx = null;
    try {
        Tx = session.beginTransaction();
        String hql = "from Departamento where codigo= :vcodDepartamento";
        Query q = session.createQuery(hql);
        q.setParameter("vcodDepartamento", _codDepartamento);
        Departamento objDepart = (Departamento) q.uniqueResult();
        session.delete(objDepart);
        Tx.commit();
    } catch (HibernateException HE) {
        HE.printStackTrace();
        if (Tx != null) {
            Tx.rollback();
        }
        throw HE;
    } finally {
        session.close();
    }
}
```

Comprobamos en la base de datos, que los datos ya no aparecen.

En el caso, de que no tuviéramos el CASCADE="ALL" en el archivo Departamento.xml, lo podemos hacer de otra forma.

Dentro de Heidi, podemos asignar el CASCADE="ALL" en la clave foránea entre Empleado y Departamento.

Básico		Opciones	Índices	Llaves foráneas	Particiones	Código CREATE	Código ALTER
Agregar	Nombre de la llave	Columnas	Tabla de refere...	C...	En UPDATE	En DELETE	
Borrar	FK	idDepartamento	departamento	id	NO ACTION	CASCADE	
Limpiar							

INSERCIÓN O ACTUALIZACIÓN DE DEPARTAMENTO.

En este ejercicio, vamos a modificar un departamento o si no existe, insertar el departamento. Por lo que empezamos creando otro controlador.

Creamos un objeto de departamento, y ponemos todos sus datos y uno de ellos diferente, para ver si lo actualiza o no.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        Departamento objDepartamento= new Departamento("22","Informatica","Socuellamos");

        try{
            new DAOOperaciones().modificarDepartamento(objDepartamento, SessionBuilder);
            response.sendRedirect("./VISTAS/Correcto.jsp?codigo=departamento-actualizar");
        } catch (HibernateException HE) {
            response.sendRedirect("./VISTAS/Error.jsp?codigo=departamento-actualizar");
        }
    }
}
```

Creamos el método modificarDepartamento y le pasamos el objeto. Como no me funciona el saveOrUpdate, no he podido adivinar porque no funciona, lo he hecho con la sentencia session.merge().

```
public void modificarDepartamento(Departamento _TransientObjDepartamento, SessionFactory _SessionFactory) {

    Session session = _SessionFactory.openSession();
    Transaction Tx = null;
    try {
        Tx = session.beginTransaction();
        String hql = "from Departamento where codigo= :vcodigo";
        Query q = session.createQuery(hql);
        q.setParameter("vcodigo", _TransientObjDepartamento.getCodigo());
        Departamento PersistentObjDepartamento = (Departamento) q.uniqueResult();

        _TransientObjDepartamento.setId(PersistentObjDepartamento.getId());
        Hibernate.initialize(PersistentObjDepartamento.getEmpleados());
        session.merge(_TransientObjDepartamento);

        Tx.commit();
    } catch (HibernateException HE) {
        HE.printStackTrace();
        if (Tx != null) {
            Tx.rollback();
        }
        throw HE;
    } finally {
        session.close();
    }
}
```

Comprobamos en la base de datos que se ha actualizado.

 id	 codigo	descripcion	localizacion
1	10	Ventas	Madrid
2	20	Contabilidad	Valencia
3	30	Logística	Albacete
4	40	Gerencia	Madrid
8	50	I&D	Alicante
11	60	Ventas	Granada
23	22	Informatica	Socuellamos

ALTA ASIGNADO A UN DEPARTAMENTO EXISTENTE. FACILITANDO CÓDIGO DEPARTAMENTO

Vamos a dar de alta un empleado nuevo y a su misma vez se lo vamos a adjuntar a un departamento que ya existe. El departamento lo vamos a pasar por su código.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        int IdDepart = 3;

        //CREAMOS UN CONSTRUCTOR SOLAMENTE CON EL ID, EN LA CLASE DEPARTAMENTO
        Departamento objDepartamento = new Departamento(IdDepart);

        //ENCRIPCIONES PARA AÑADIR AL EMPLEADO
        String PasswordEncriptado = Hash.sha1("Anabel");
        String codigo = "1234457";
        byte[] codigoBinario = codigo.getBytes(StandardCharsets.UTF_8);

        //CREAMOS EL OBJETO EMPLEADO Y LE AÑADIMOS LOS VALORES
        Empleado ObjEmpleado = new Empleado(objDepartamento, codigoBinario,
            "06290540V", PasswordEncriptado, "Morales Nuñez", "Anabel",
            "Estudiante", 21000.0, java.sql.Date.valueOf(LocalDate.of(1999, 7, 8)),
            java.sql.Date.valueOf(LocalDate.now()));

        try {
            new DAOOperaciones().insertarEmpleado(ObjEmpleado, SessionBuilder);
            response.sendRedirect("../VISTAS/Correcto.jsp?codigo=empleado-alta-con-departamento");
        } catch (HibernateException HE) {
            response.sendRedirect("../VISTAS/Error.jsp?codigo=empleado-alta-con-departamento");
        }
    }
}
```

Empezamos creando el constructor solamente con el id del departamento, en la clase Departamento.

```
public Departamento(Integer id) {
    this.id = id;
}
```

Despues, encriptamos con AES y HASH, la contraseña y el código, respectivamente. Ahora sí, creamos el objeto Empleado con todo y le añadimos el objDepartamento.

Creamos el método en el DAOOperaciones y usamos la sentencia session.save().

```
public void insertarEmpleado(Empleado _ObjEmpleado, SessionFactory _SessionFactory) {
    Session session = _SessionFactory.openSession();
    Transaction Tx = null;
    try {
        Tx = session.beginTransaction();
        session.save(_ObjEmpleado);
        Tx.commit();
    } catch (HibernateException HE) {
        HE.printStackTrace();
        if (Tx != null) {
            Tx.rollback();
        }
        throw HE;
    } finally {
        session.close();
    }
}
```

Comprobamos en la base de datos.

id	codigo	nif	password	apellidos	nombre	profesion	salario	fechaNac	fechaIng	idDepartamento
28	0x2CEFBFB...	99987765D	1b47e87d02...	López Pérez	Diego	Vendedor	27.000	1991-01-18	2019-12-20	11
29	0xEFBBDEF...	08861760T	e7097cce19...	Antúñez Picazo	Luisa	Vendedor	27.000	1996-05-29	2019-12-20	11
30	0xEFBBDEF...	22117719B	61632e2a55f...	Díaz Querol	Emilia	Vendedor	27.000	1983-11-10	2019-12-20	11
34	0x2AB75EFD...	06290540V	8f014f67684...	Morales Nuñez	Anabel	Estudiante	21.000	1999-07-08	2020-04-23	3

BAJA. SELECCIÓN POR NIF Y PASSWORD.

En este caso, vamos a dar de baja un empleado pasándole el nif.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        //COGEMOS EL NIF DE CUALQUIER EMPLEADO
        String nif = "06290542F";
        String password=Hash.sha1("Edu");
        try {
            new DAOOperaciones().borrarEmpleado(nif,password, SessionBuilder);
            response.sendRedirect("./VISTAS/Correcto.jsp?codigo=empleado-borrar");
        } catch (HibernateException HE) {
            response.sendRedirect("./VISTAS/Error.jsp?codigo=empleado-borrar");
        }
    }
}
```

Creamos el método borrarEmpleado, donde vamos a pasarle el nif.

```
public void borrarEmpleado(String _nifEmpleado, SessionFactory _SessionFactory) {
    Session session = _SessionFactory.openSession();
    Transaction Tx = null;
    try {
        Tx = session.beginTransaction();
        String hql = "from Empleado where nif=:vnifEmpleado";
        Query q = session.createQuery(hql);
        q.setParameter("vnifEmpleado", _nifEmpleado);
        Empleado objEmpleado = (Empleado) q.uniqueResult();
        session.delete(objEmpleado);
        Tx.commit();
    } catch (HibernateException HE) {
        HE.printStackTrace();
        if (Tx != null) {
            Tx.rollback();
        }
        throw HE;
    } finally {
        session.close();
    }
}
```

Comprobamos en la base de datos.

MODIFICACIÓN DE DATOS. CUALQUIERA MENOS NIF.

En esta parte, vamos a modificar los datos de un empleado. Creamos un controlador y dentro de él, el objeto de empleado que contenga todos los datos, pero cambiamos el que queramos

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {

        Empleado objEmpleado = new Empleado("123456870".getBytes(StandardCharsets.UTF_8),
            "06290540S", Hash.sha1("Daniel"), "Pardo", "Daniel", "Diseñador",
            2540.2, java.sql.Date.valueOf(LocalDate.of(1988, 11, 5)),
            java.sql.Date.valueOf(LocalDate.now()));

        Departamento objDepartamento = new Departamento(37);
        objEmpleado.setDepartamento(objDepartamento);

        try{
            new DAOOperaciones().modificarEmpleado(objEmpleado, SessionBuilder);
            response.sendRedirect("../VISTAS/Correcto.jsp?codigo=empleado-modificar");
        } catch (HibernateException HE) {
            response.sendRedirect("../VISTAS/Error.jsp?codigo=empleado-modificar");
        }
    }
}
```

Creamos un método para modificar al empleado y usamos la sentencia session.merge().

```
public void modificarEmpleado(Empleado _TransientObjEmpleado, SessionFactory _SessionFactory) {

    Session session = _SessionFactory.openSession();
    Transaction Tx = null;
    try {
        Tx = session.beginTransaction();
        String hql = "from Empleado where nif= :vnif";
        Query q = session.createQuery(hql);
        q.setParameter("vnif", _TransientObjEmpleado.getNif());
        Empleado PersistentObjEmpleado = (Empleado) q.uniqueResult();

        _TransientObjEmpleado.setId(PersistentObjEmpleado.getId());

        session.merge(_TransientObjEmpleado);

        Tx.commit();

    } catch (HibernateException HE) {
        HE.printStackTrace();
        if (Tx != null) {
            Tx.rollback();
        }
        throw HE;
    } finally {
        session.close();
    }
}
```

Comprobamos en la base de datos que está todo correcto.

id	codi...	nif	pass...	apellidos	nombre	profesion	salario	fechaNac
28	0x2CEF...	99987765D	1b47...	López Pérez	Diego	Vendedor	27.000	1991-01-18
29	0xEFBF...	08861760T	e709...	Antúñez Picazo	Luisa	Vendedor	27.000	1996-05-29
30	0xEFBF...	22117719B	6163...	Díaz Querol	Emilia	Vendedor	27.000	1983-11-10
35	0xC278...	06290542F	af49...	Mondejar Sevillano	Eduardo	Contable	2.540,2	1998-12-03

GENERAL DE EMPLEADOS

En este apartado, vamos a sacar un listado general de todos los empleados junto a sus departamentos.

Antes que nada, tenemos que recordar que los empleados estan unidos a los departamentos, por lo que Hibernate no nos dejará sacar los departamentos de los empleados sin inicializar los departamentos.

Empezamos creando el controlador, pero en vez de crear objetos como anteriormente, creamos un List que es muy parecido a ArrayList.

En este punto, vamos a crear también una sesión que será donde almacenemos lo obtenido en el método y así poder pintarlo después en una vista, que es donde sacaremos todos los datos.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        HttpSession session = request.getSession(true);
        List<Empleado> ListadoGeneralEmpleados = new DAOOperaciones().getListadoGeneralEmpleados(SessionBuilder);

        session.setAttribute("listadoEmpleados", ListadoGeneralEmpleados);
        response.sendRedirect("./VISTAS/ListadoEmpleados.jsp");
    }
}
```

Creamos el método que obtenga el listado de los empleados. Como he dicho al principio, hay que hacer un `Hibernate.initialize()`, en este caso de departamentos. Con un iterador vamos a recorrer el listado de empleados.

```
public List<Empleado> getListadoGeneralEmpleados(SessionFactory _SessionFactory) {
    Session session = _SessionFactory.openSession();
    Transaction Tx = null;
    Tx = session.beginTransaction();
    try {
        String hql = "from Empleado";
        Query q = session.createQuery(hql);
        List ListadoEmpleados = q.list();
        Iterator IterEmpleado = ListadoEmpleados.iterator();
        while (IterEmpleado.hasNext()) {
            Empleado objEmpleado = (Empleado) IterEmpleado.next();
            Hibernate.initialize(objEmpleado.getDepartamento());
        }

        return ListadoEmpleados;
    } catch (HibernateException HE) {
        HE.printStackTrace();
        if (Tx != null) {
            Tx.rollback();
        }
        throw HE;
    } finally {
        session.close();
    }
}
```

Después, si todo es correcto, nos llevara a la vista, donde vamos a pintar la información que hemos guardado en la sesión. Esta vez no hay que comprobar nada en la base de datos, simplemente sacar los datos por pantalla.

```
<%
    HttpSession session = request.getSession(true);
    List<Empleado> ListadoGeneralEmpleados = (ArrayList) session.getAttribute("listadoEmpleados");
    Iterator IterEmpleado = ListadoGeneralEmpleados.iterator();
    while (IterEmpleado.hasNext()) {
        Empleado objEmpleado = (Empleado) IterEmpleado.next();

        <td><%= objEmpleado.getNif() %></td>
        <td><%= objEmpleado.getNombre() %></td>
        <td><%= objEmpleado.getApellidos() %></td>
        <td><%= objEmpleado.getDepartamento().getDescripcion() %></td>
        <td><%= objEmpleado.getDepartamento().getLocalizacion() %></td>
    }
    </tr>
<%
}
%>
```

En vez de hacer `out.println()`, usamos el `=`.

Si no hubiéramos hecho el `Hibernate.initialize()`, nos saltaría un error 505 de que falta por inicializar y un error de lazy. Obtenemos los datos solicitados por pantalla.

LISTADO GENERAL DE EMPLEADOS CON SUS DEPARTAMENTOS

NIF	NOMBRE	APELLIDOS	DEPARTAMENTO	LOCALIZACIÓN
99987765D	Diego	López Pérez	Ventas	Granada
08861760T	Luisa	Antúnez Picazo	Ventas	Granada
22117719B	Emilia	Díaz Querol	Ventas	Granada

Volver atrás

DEPARTAMENTO Y EMPLEADOS, MEDIANTE CÓDIGO DEPARTAMENTO

En este caso vamos a hacerlo al revés, vamos a sacar los datos de un departamento y los empleados que tiene asociados a él. El mismo procedimiento que el anterior pero ahora en vez de un List de Empleados, lo hacemos de Departamento y creamos una sesión nueva.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        HttpSession session = request.getSession(true);
        String codigo="60";
        List<Departamento> ListaGeneralDepartamentos=new DAOOperaciones().listadoDepartamentoyEmpleados(codigo,Session);
        session.setAttribute("listadoDepartamentoyEmpleados", ListaGeneralDepartamentos);
        response.sendRedirect("../VISTAS/ListadoDepartamentoyEmpleados.jsp");
    }
}
```

Como tenemos que sacar los departamentos mediante el código de el mismo, lo hacemos así:

```
public List<Departamento> listadoDepartamentoyEmpleados(String _codigo, SessionFactory _SessionFactory) {
    Session session = _SessionFactory.openSession();
    Transaction Tx = null;
    Tx = session.beginTransaction();
    try {
        String hql = "from Departamento where codigo= :vcodigo";
        Query q = session.createQuery(hql);
        q.setParameter("vcodigo", _codigo);
        List<Departamento> ListadoDepartamento = q.list();
        Iterator IterDepartamento = ListadoDepartamento.iterator();
        while (IterDepartamento.hasNext()) {
            Departamento objDepartamento = (Departamento) IterDepartamento.next();
            Hibernate.initialize(objDepartamento.getEmpleados());
        }
        return ListadoDepartamento;
    } catch (HibernateException HE) {
        HE.printStackTrace();
        if (Tx != null) {
            Tx.rollback();
        }
        throw HE;
    } finally {
        session.close();
    }
}
```

Si todo es correcto, nos llevara a la vista que hemos creado nueva. Ahí es donde vamos a pintar la sesión y sacar los datos. Esta vez tendremos que empezar con departamentos, crear su iterador, su propio objeto y desde él, hacemos lo mismo con empleados.

```
<%
    HttpSession session = request.getSession(true);
    List<Departamento> ListadoDepartamentoyEmpleados = (ArrayList) session.getAttribute("listadoDepartamentoyEmpleados");
    Iterator IterDepartamento = ListadoDepartamentoyEmpleados.iterator();
    while (IterDepartamento.hasNext()) {
        Departamento objDepartamento = (Departamento) IterDepartamento.next();
        Iterator IterEmpleado = objDepartamento.getEmpleados().iterator();
    }
%>
<th colspan="4">DEPARTAMENTO DE <%= (objDepartamento.getDescripcion()).toUpperCase() %></th>
</tr>
</thead>
<tbody>
<tr>
<td>NIF</td>
<td>NOMBRE</td>
<td>APELLIDOS</td>
<td>PROFESION</td>
</tr>
<%
    while (IterEmpleado.hasNext()) {
        Empleado objEmpleado = (Empleado) IterEmpleado.next();
    }
%>
```

Ejecutamos el proyecto y comprobamos que funciona correctamente.

LISTADO DE UN DEPARTAMENTO Y SUS EMPLEADOS

DEPARTAMENTO DE VENTAS			
NIF	NOMBRE	APELLIDOS	PROFESION
08861760T	Luisa	Antúñez Picazo	Vendedor
99987765D	Diego	López Pérez	Vendedor
22117719B	Emilia	Díaz Querol	Vendedor

[Volver atrás](#)

DEPARTAMENTOS

En este caso, solamente vamos a sacar los datos de los departamentos que tenemos en estos momentos.

Empezamos con nuestro controlador, que es el mismo procedimiento que los dos anteriores.

Creamos una sesión nueva, un método para sacar un listado de los departamentos, se lo metemos en la sesión y lo pintamos en una vista.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        HttpSession session = request.getSession(true);
        List<Departamento> ListaGeneralDepartamentos=new DAOOperaciones().getListadoGeneralDepartamentos(SessionBu
        session.setAttribute("listadoDepartamentos", ListaGeneralDepartamentos);
        response.sendRedirect("../VISTAS/ListadoDepartamentos.jsp");
    }
}
```

En cuanto al método, podemos hacerlo de dos formas:

Esta manera, es sin el Hibernate.initialize(). Los datos los saca correctamente, ya que no utilizamos nada relacionado con empleados.

```
public List<Departamento> getListadoGeneralDepartamentos(SessionFactory _SessionFactory) {
    Session session = _SessionFactory.openSession();
    Transaction Tx = null;
    Tx = session.beginTransaction();
    try {
        String hql = "from Departamento";
        Query q = session.createQuery(hql);
        List<Departamento> ListadoDepartamento = q.list();
        return ListadoDepartamento;
    } catch (HibernateException HE) {
        HE.printStackTrace();
        if (Tx != null) {
            Tx.rollback();
        }
        throw HE;
    } finally {
        session.close();
    }
}
```

La segunda manera es utilizando el Hibernate.initalize(), simplemente por si en algún momento nos da un error o tenemos que implementar los empleados relacionados con los departamentos.

Pintamos en la vista los valores mediante la sesión.

```
<%
    HttpSession session = request.getSession(true);
    List<Empleado> ListadoGeneralEmpleados = (ArrayList) session.getAttribute("listadoEmpleados");
    Iterator IterEmpleado = ListadoGeneralEmpleados.iterator();
    while (IterEmpleado.hasNext()) {
        Empleado objEmpleado = (Empleado) IterEmpleado.next();
    }
%>
```

Ejecutamos el proyecto y comprobamos que todo lo muestra bien.

LISTADO GENERAL DE EMPLEADOS CON SUS DEPARTAMENTOS		
CÓDIGO	DESCRIPCIÓN	LOCALIZACIÓN
10	Ventas	Madrid
20	Contabilidad	Valencia
30	Logística	Albacete
40	Gerencia	Madrid
50	I&D	Alicante
60	Ventas	Granada
22	Informatica	Albacete