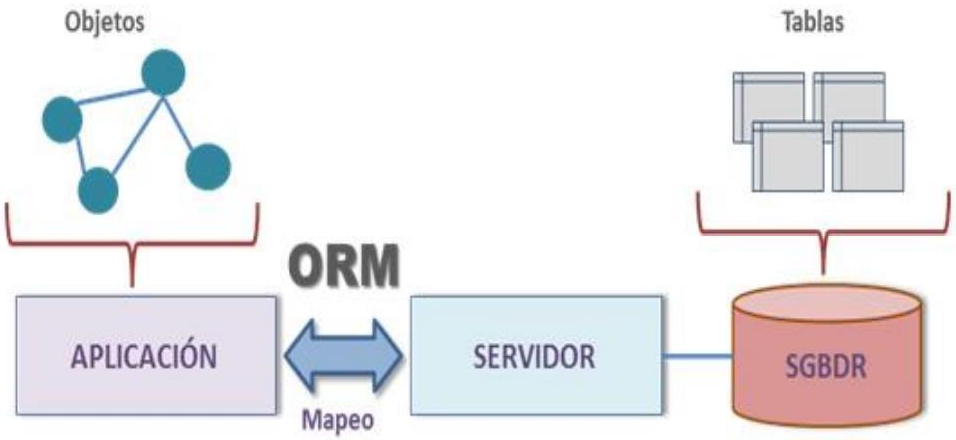
HERRAMIENTAS DE MAPEO OBJETO RELACIONAL (ORM)

4. INTRODUCCIÓN

En muchas ocasiones se requiere efectuar la conversión de los objetos con los que se trabaja a nivel de programación a modelo relacional de BD. En estos casos se requiere efectuar lo que se conoce como **mapeo** que consiste en una **conversión de las propiedades o atributos de los objetos a tablas, columnas y tuplas** del modelo relacional. Esta transformación se realiza mediante **herramientas de mapeo ORM (Object Relational Mapping).**

El mapeo objeto-relacional **(Object-Relational Mapping, ORM)** es una técnica de programación que permite convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional **utilizando un motor de persistencia**.



En la práctica este mapeo crea una **BDOO virtual** sobre la BDR que posibilita implementar características propias de la OO.

Actualmente**, las bases de datos relacionales solo pueden guardar datos primitivos**, por lo que no podemos guardar objetos que vayamos creando en nuestra aplicación, sino que lo que hacemos es **convertir los datos del objeto en datos primitivos** que si podremos almacenar en las tablas correspondientes de nuestras bases de datos. Si luego necesitamos ese objeto en alguna parte de nuestra aplicación, debemos de recuperar los datos primitivos de la base de datos y volver a construir el objeto.

El mapeo objeto-relacional nos ayuda a efectuar dicha conversión de objetos a la BD almacenándolos en las tablas, de modo que cuando se desean recuperar los datos del

sistema relacional lo que se hace es obtener los datos primitivos de la BD y volver a construir el objeto de forma transparente.

La utilización de ORM nos ofrece ciertas ventajas e inconvenientes:

✓ Ventajas

- **Rapidez en el desarrollo.** La mayoría de las herramientas actuales permiten la creación del modelo por medio del esquema de la base de datos, leyendo el esquema, nos crea el modelo adecuado.
- **Abstracción de la base de datos.** Al utilizar un sistema ORM, lo que conseguimos es separarnos totalmente del sistema de Base de datos que utilicemos de forma que si en un futuro debemos de cambiar de motor de bases de datos este cambio no afectará a nuestro sistema.
- **Reutilización.** Permite utilizar los métodos de un objeto de datos desde distintas zonas de la aplicación e incluso desde aplicaciones distintas.
- **Seguridad.** Los ORM suelen implementar sistemas para evitar tipos de ataques.
- **Mantenimiento del código.** Facilita el mantenimiento del código debido a la correcta ordenación de la capa de datos, haciendo que el mantenimiento del código sea mucho más sencillo.
- **Lenguaje propio para realizar las consultas.** Los mecanismos de mapeo tienen su propio lenguaje para hacer las consultas, lo que hace que los usuarios dejen de utilizar las sentencias SQL para que pasen a utilizar el lenguaje propio de cada herramienta.

✓ Desventajas

- **Tiempo utilizado en el aprendizaje.** Este tipo de herramientas suelen ser complejas por lo que su correcta utilización lleva un tiempo que hay que emplear en ver el funcionamiento correcto y ver todo el partido que se le puede sacar.
- **Aplicaciones algo más lentas.** Esto es debido a que todas las consultas que se hagan sobre la base de datos, el sistema primero deberá de transformarlas al lenguaje propio de la herramienta, luego leer los registros y por último crear los objetos.

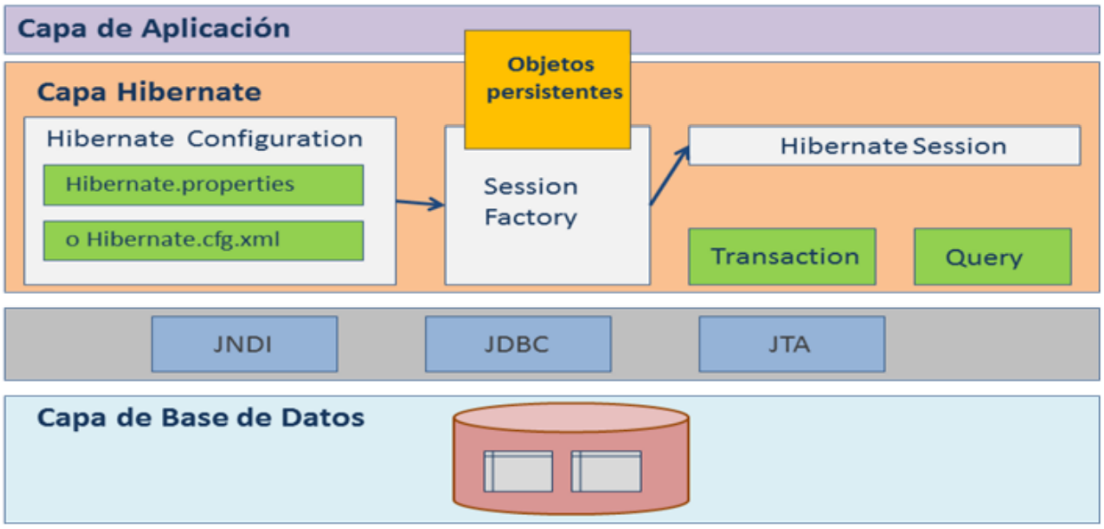
Actualmente hay muchos tipos de frameworks que permiten efectuar el mapeo objeto-relacional según el lenguaje de programación utilizado: Doctrine, Propel (PHP), ADO.NET (Microsoft.net) y la herramienta que se va a utilizar en esta unidad que es Hibernate.

Hibernate es una herramienta ORM para la tecnología JAVA y disponible también para la tecnología .NET con el nombre de NHibernate. Es software libre bajo la licencia GNU/LGPL.

4.1 ARQUITECTURA HIBERNATE

En esta unidad vamos a trabajar con la herramienta ORM Hibernate para realizar el mapeo objeto relacional en Java. Hibernate es software libre distribuido bajo licencia GNU LGPL ampliamente reconocido y utilizado para tratar objetos de Java Planos denominados **(POJO´s – Plain Old Java Objects).**

Para almacenar y recuperar estos objetos de la BD el desarrollador ha de mantener una conversación con el motor de Hibernate mediante el objeto sesión **(clase Session)** que es similar a la conexión JDBC.



La clase Session **(org.hibernate.Session)** se utiliza para obtener la conexión física con la BD. Es un objeto ligero que **está diseñado para ejecutarse cada vez que se necesita una interacción con las BD** de modo que los objetos persistentes se guardan y recuperan mediante el objeto Session. Los objetos Session no deben mantenerse abiertos mucho tiempo ya que no garantizan la seguridad de los subprocesos por lo que deben ser creados y destruidos a medida que se necesiten. La instanciación de la Session no consume mucha memoria y se crea y destruye de forma muy simple.

Tiene los métodos **save (Object objeto), load(), createQuery(String consulta), beginTransaction(), close(), etc**, que permiten interactuar con la BD. Hibernate utiliza varios API de Java existentes, como son **JDBC** (cualquier BD con un driver JDBC pueda utilizarse con Hibernate), Java Transacción API (**JTA**, establece una serie de interfaces java entre el manejador de transacciones y las partes involucradas en el sistema de transacciones distribuidas: el servidor de aplicaciones, el manejador de recursos y las aplicaciones trasaccionales) y Java Naming and Directory Interface (**JNDI**) (permiten

que Hibernate pueda ser integrado con servidores de aplicaciones JEE). Las interfaces que ofrece Hibernate son las siguientes:

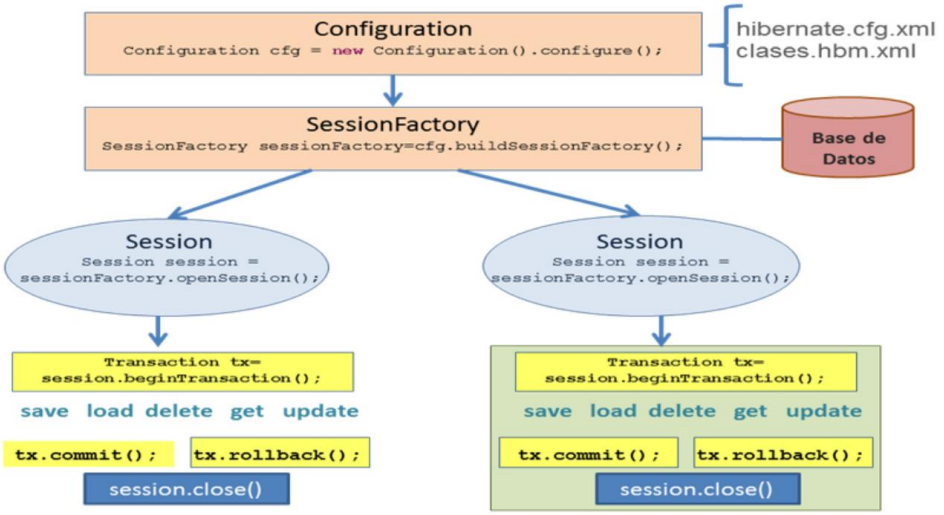
• **Interfaz SessionFactory (org.Hibernate.SessionFactory)** permite obtener instancias Session. **Esta interfaz debe compartirse entre muchos hilos de ejecución.** Normalmente hay **una única sesión SessionFactory** para toda la aplicación, creada durante la inicialización de la misma, y se utiliza para crear todas las sesiones relacionadas con un contexto dado. Si la aplicación accede a varias BD se necesitará **una SessionFactory por cada BD.**

• **Interfaz Configuration (org.hibernate.cfg.Configuration):** se utiliza para configurar Hibernate y es el primer objeto que hibernate instancia. Generalmente se hace una sola vez durante la inicialización de la aplicación y se utiliza para especificar la ubicación de los documentos que indican el mapeado de los objetos y propiedades específicas de Hibernate y a continuación poder crear la instancia del objeto SessionFactory. El objeto de configuración proporciona dos componentes claves:

- **Conexión de base de datos:** que se maneja a través de uno o más archivos de configuración soportadas por Hibernate. Estos archivos son **hibernate.properties y hibernate.cfg.xml**.
- **Configuración de Mapeo:** es un componente que crea la conexión entre las clases de Java y tablas de la base de datos.

• Query (org.hibernate.Query): Hibernate Query Language (HQL) permite realizar las consultas a la BD, controla la ejecución de estas y permite crear objetos. Una instancia de consulta se utiliza para enlazar los parámetros de consulta, limitar el número de resultados devueltos por la consulta, y, finalmente, para ejecutar la consulta.

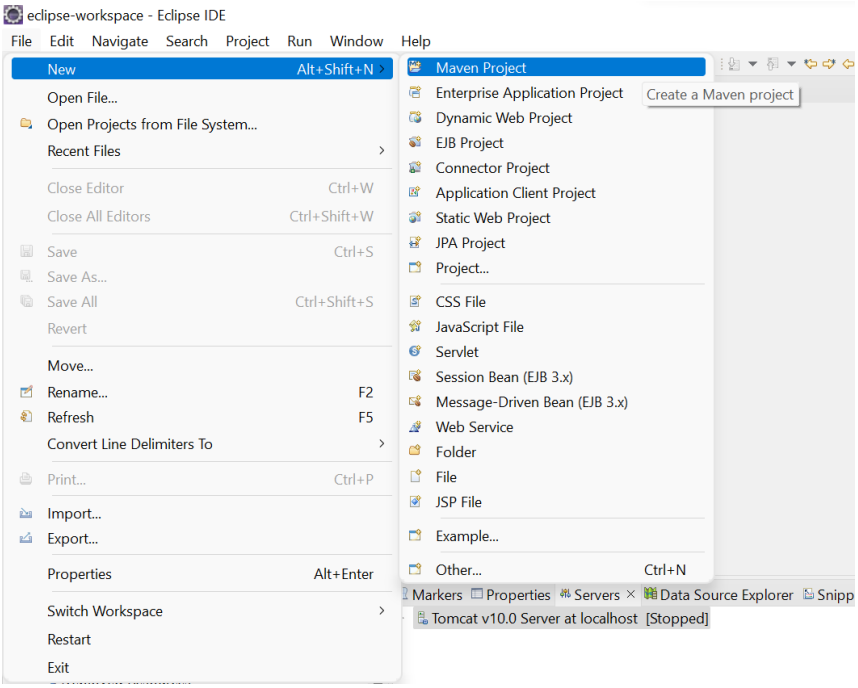
• **Transaction (org.hibernate.Transaction):** permite manejar las transacciones mediante un administrador de base de transacciones de forma que si se produce un error durante el período de vida de la transacción pueda ser reconocido. Es un objeto opcional de forma que se pueden gestionar las transacciones desde la propia aplicación. Esta figura muestra cómo es una aplicación con Hibernate:

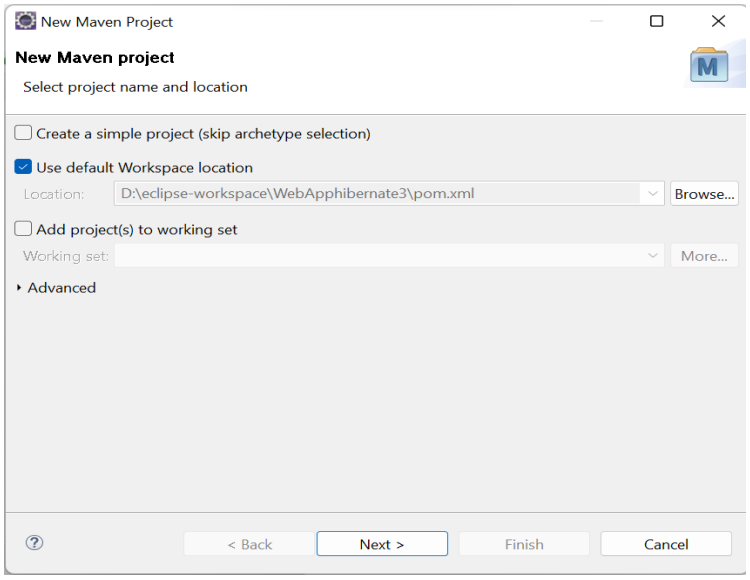

Configuration
Configuration cfg = new Configuration().configure();
hibernate.cfg.xml
clases.hbm.xml
SessionFactory
SessionFactory sessionFactory=cfg.buildSessionFactory();
Base de Datos
Session
Session session = sessionFactory.openSession();
Session
Session session = sessionFactory.openSession();
Transaction tx= session.beginTransaction();
save load delete get update
tx.commit();
tx.rollback();
session.close()
Transaction tx= session.beginTransaction();
save load delete get update
tx.commit();
tx.rollback();
session.close()

4.2 INSTALACIÓN Y CONFIGURACIÓN DE HIBERNATE EN MAVEN

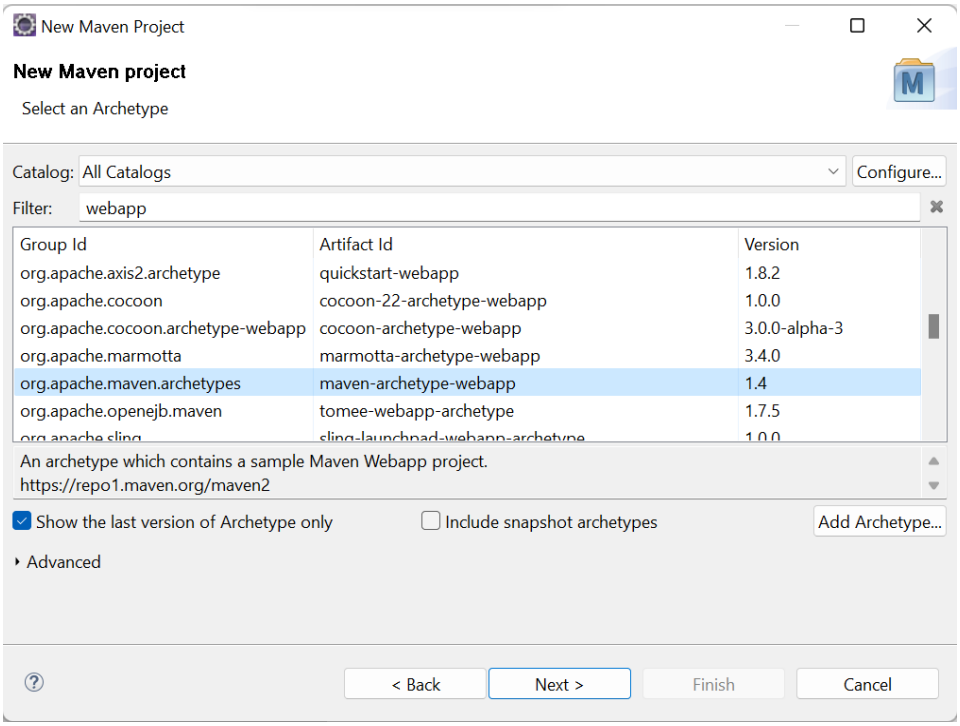
Para instalar Hibernate y poder utilizarlo con el IDE de Eclipse seguiremos los siguientes pasos:

✓ Creación un proyecto Maven desde Eclipse

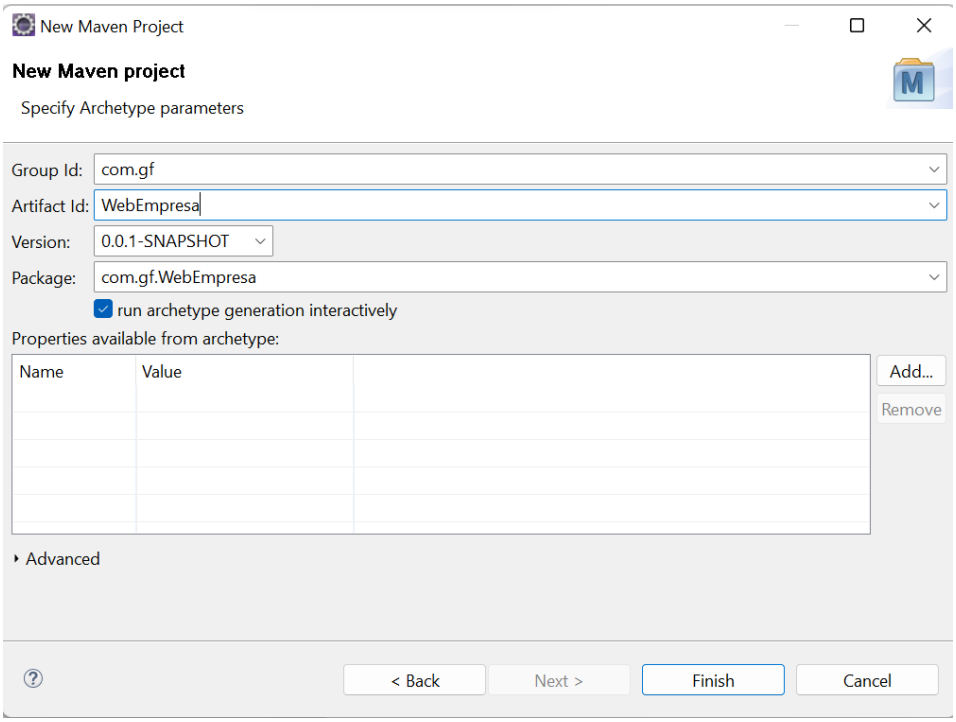




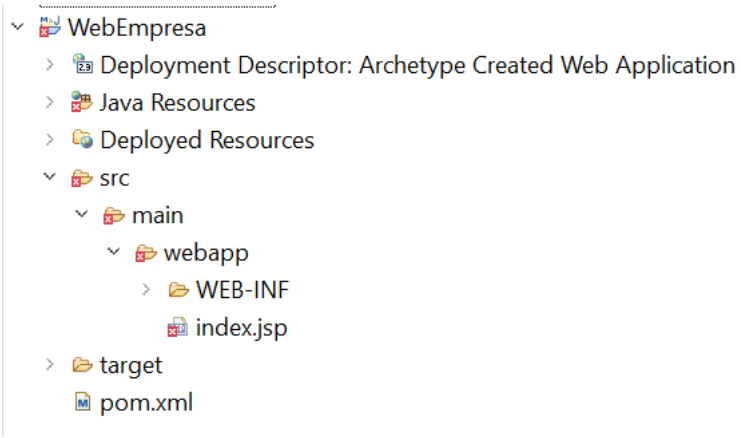
✓ Se selecciona el siguiente arquetipo para web (maven-archetype-webapp):



✓ Se introduce el group id y el artefact id y se termina la creación del proyecto.



✓ Se obtiene la siguiente estructura del proyecto

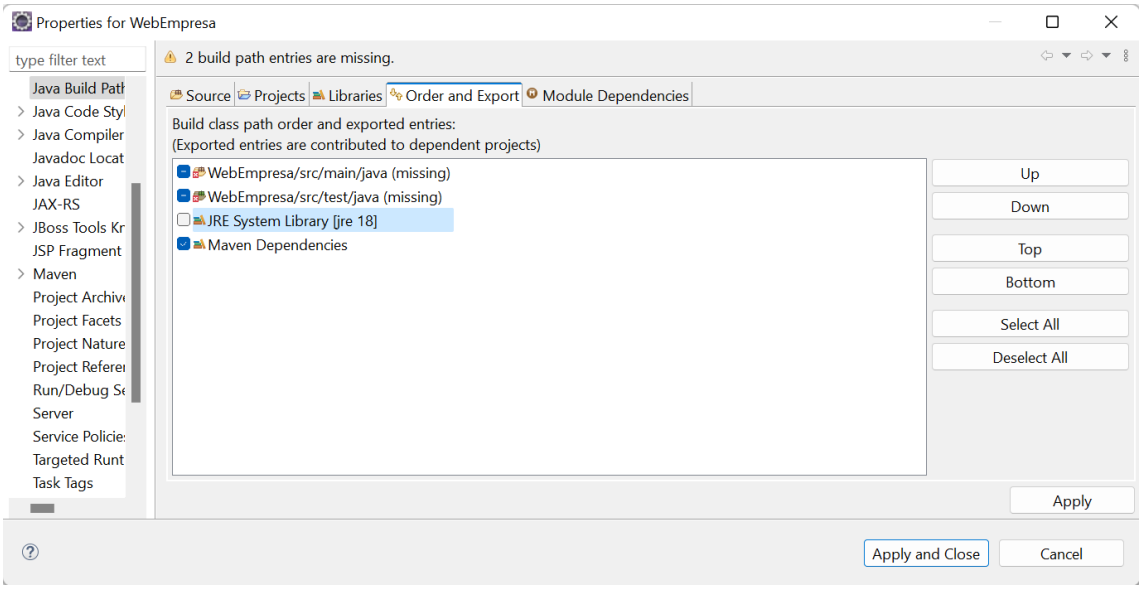


✓ Se accede al pom.xml y se actualiza la versión del plugin de compilación de Maven de Java si fuera necesario.

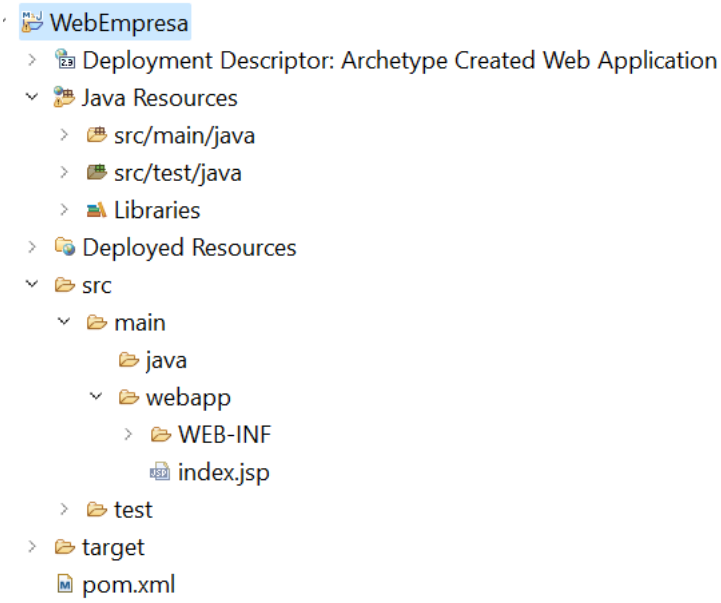
✓ Se añaden las dependencias que se necesitan para trabajar con JSP y Servlets si es necesario. También se añaden las dependencias asociadas a la última versión de Hibernate.

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.6.15.Final</version>
</dependency>
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.31</version>
</dependency>
<!-- https://mvnrepository.com/artifact/jakarta.servlet/jakarta.servlet-api --
<dependency>
    <groupId>jakarta.servlet</groupId>
    <artifactId>jakarta.servlet-api</artifactId>
    <version>5.0.0</version>
    <scope>provided</scope>
</dependency>
```

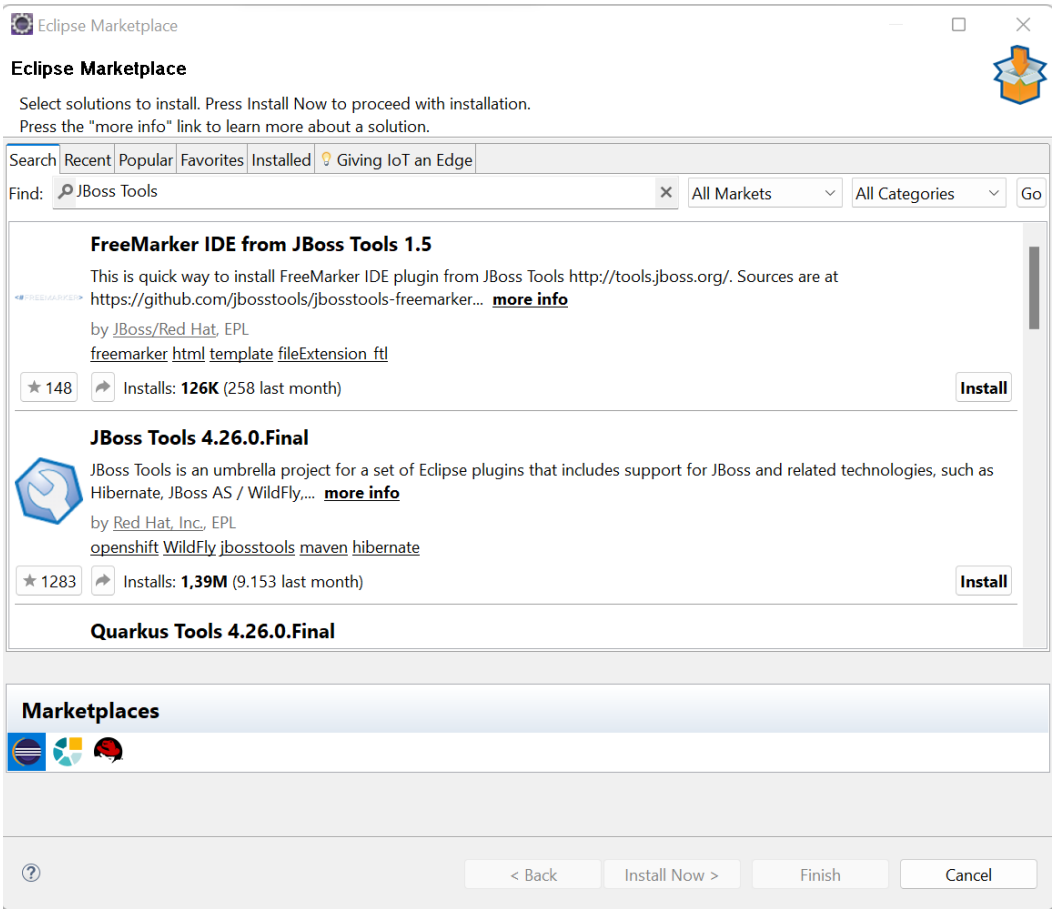
- ✓ Mostrar los directorios ocultos. Clic derecho al proyecto y seleccionar la opción **Build Path/Configure Build Path...** En la ventana que se abre se marca la casilla **Maven Dependencies** y se hace clic en Aplicar.



✓ Ahora la estructura del proyecto web está lista para trabajar.

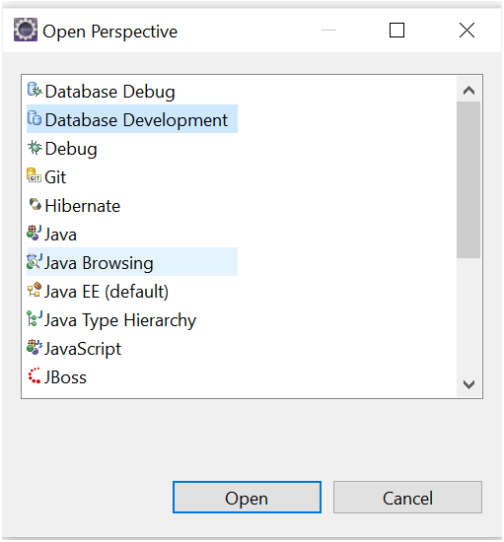


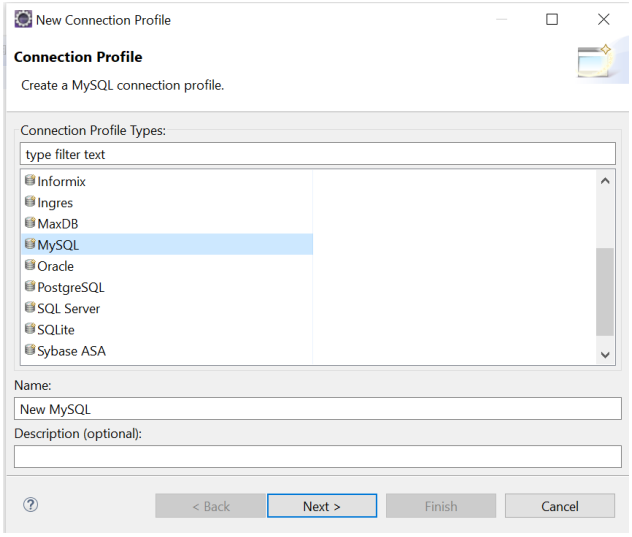
✓ A continuación, se configura Hibernate en Eclipse en sus entornos Java y Java EE usando el **JBoss Tools.** Para instalar JBoss Tools se puede hacer directamente desde el **Eclipse Marketplace** o se puede hacer una instalación más manual, desde Eclipse: **Help -> Install New Software.**

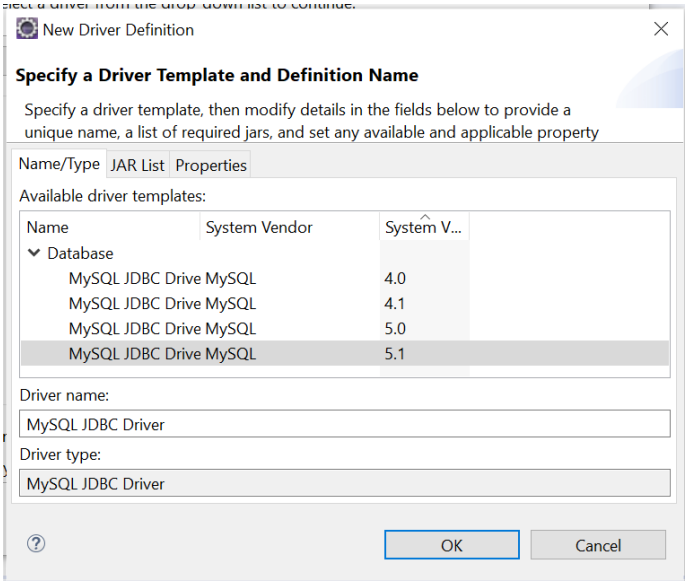


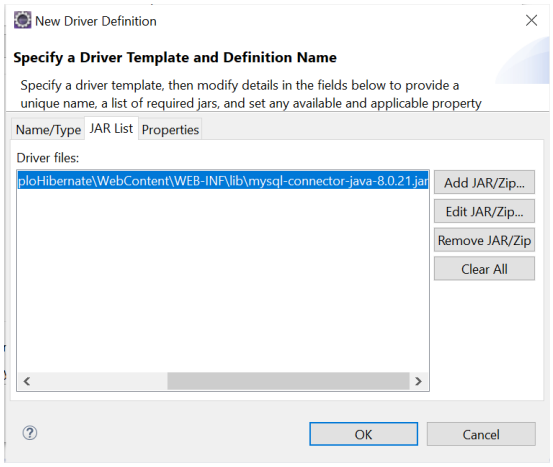
5.2 CONFIGURACIÓN DEL DRIVER EN EL SGBDR (MYSQL)

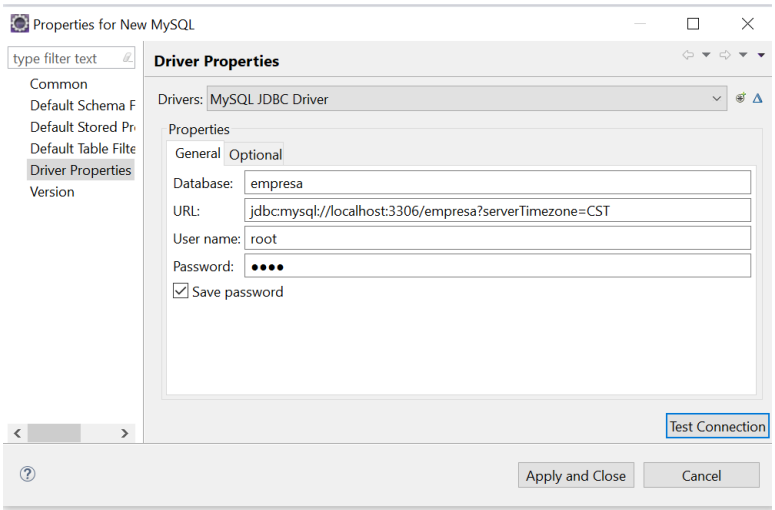
El siguiente paso va a ser crear la conexión a la base de datos desde **Windows - > Perspective -> Open Perspective -> Database Development,** se crea una conexión nueva y se configura adecuadamente todos los parámetros.

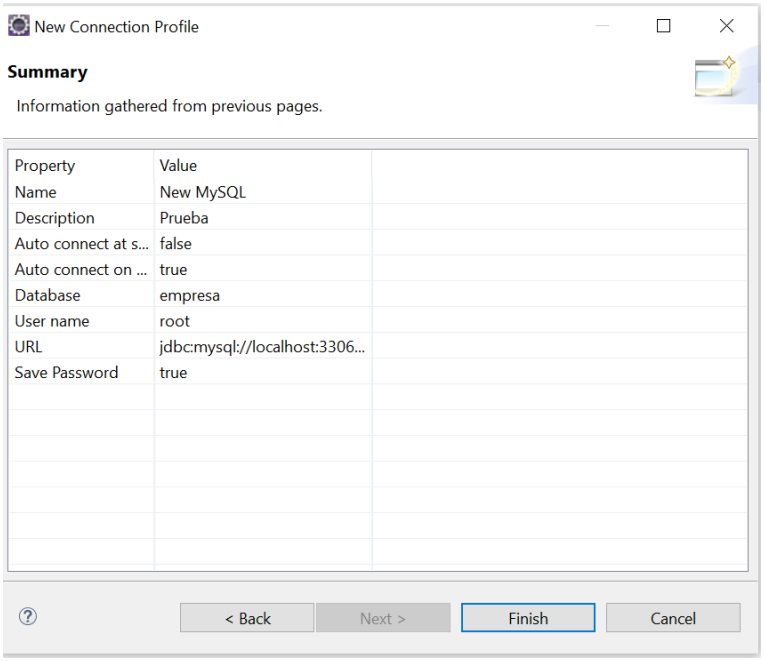





New Driver Definition
Specify a Driver Template and Definition Name
Specify a driver template, then modify details in the fields below to provide a unique name, a list of required jars, and set any available and applicable property
Name/Type
JAR List
Properties
Available driver templates:
Name
System Vendor
System V...
Database
MySQL JDBC Drive MySQL 4.0
MySQL JDBC Drive MySQL 4.1
MySQL JDBC Drive MySQL 5.0
MySQL JDBC Drive MySQL 5.1
Driver name:
MySQL JDBC Driver
Driver type:
MySQL JDBC Driver
OK
Cancel

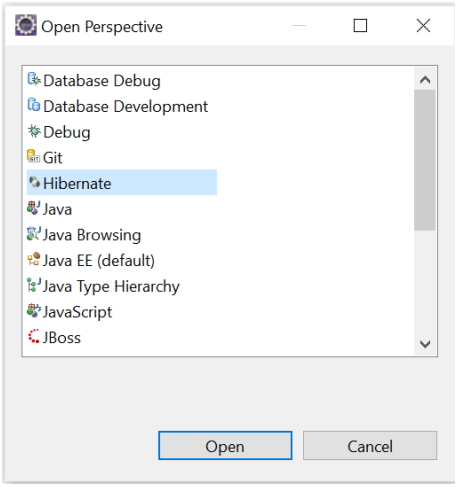

New Driver Definition
Specify a Driver Template and Definition Name
Specify a driver template, then modify details in the fields below to provide a unique name, a list of required jars, and set any available and applicable property
Name/Type
JAR List
Properties
Driver files:
ploHibernate\WebContent\WEB-INF\lib\mysql-connector-java-8.0.21.jar
Add JAR/Zip...
Edit JAR/Zip...
Remove JAR/Zip
Clear All
OK
Cancel


Properties for New MySQL
type filter text
Common
Default Schema F
Default Stored Pr
Default Table Filte
Driver Properties
Version
Driver Properties
Drivers: MySQL JDBC Driver
Properties
General
Optional
Database: empresa
URL: jdbc:mysql://localhost:3306/empresa?serverTimezone=CST
User name: root
Password: ••••
Save password
Test Connection
Apply and Close
Cancel

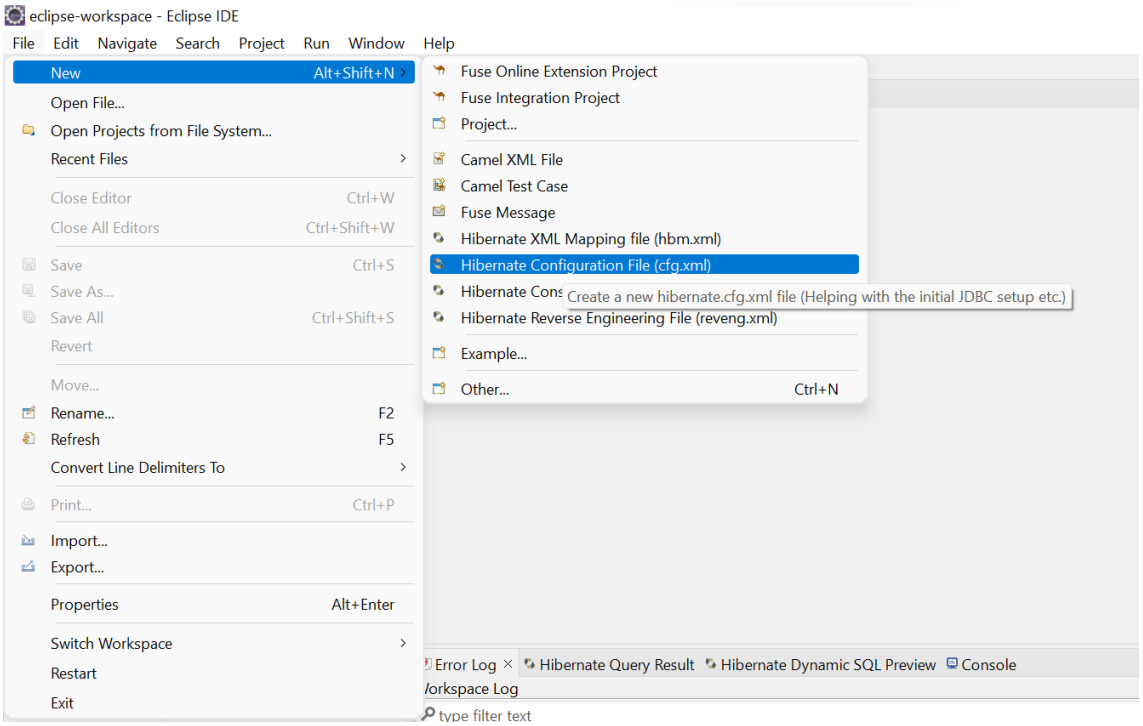


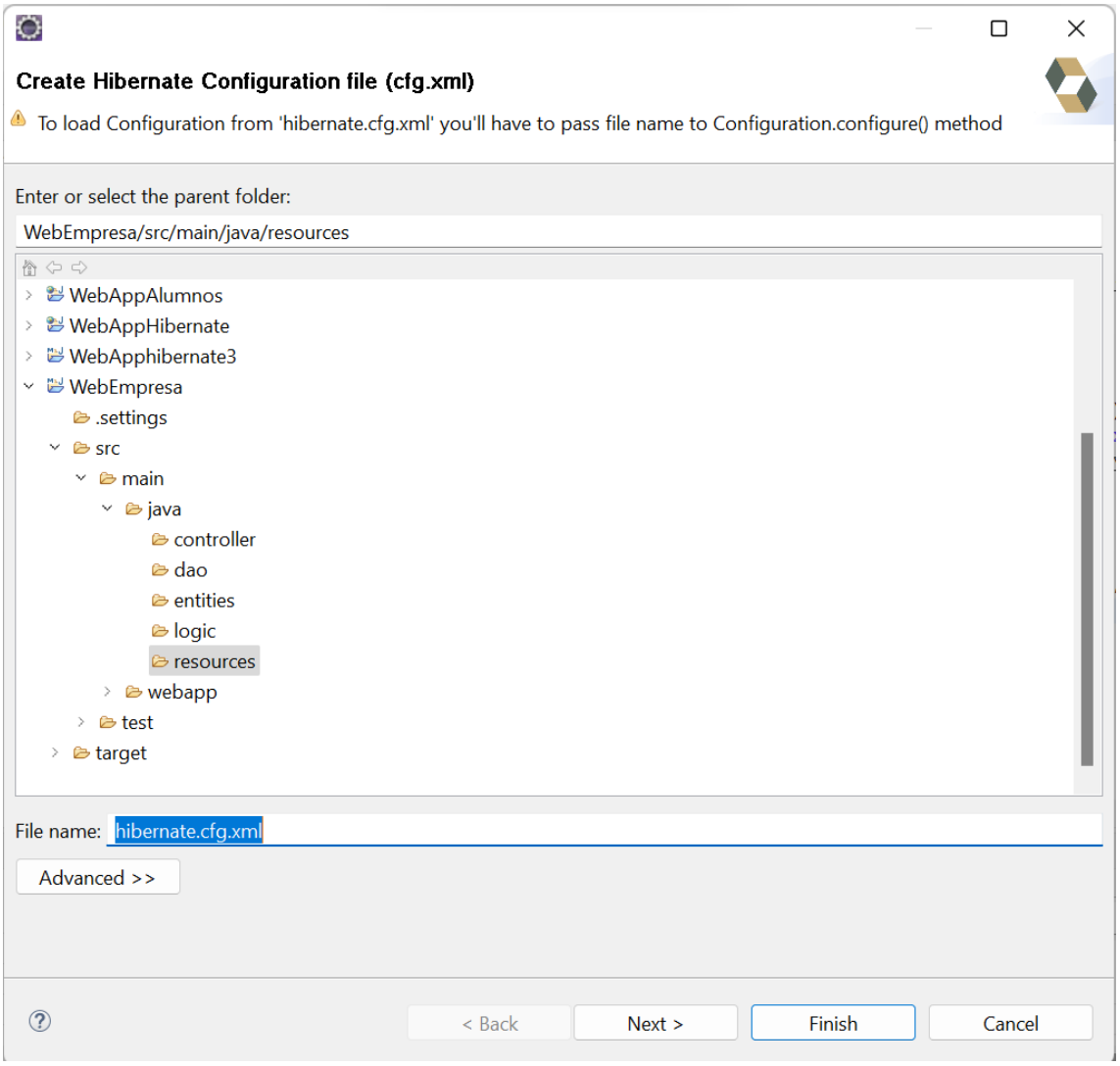
4.3 UTILIZACIÓN HIBERNATE TOOLS

A partir de ahora, se puede empezar a trabajar con **Hibernate Tools**, para esto se abre la **perspectiva Hibernate**, y se pueden crear distintos archivos de configuración.

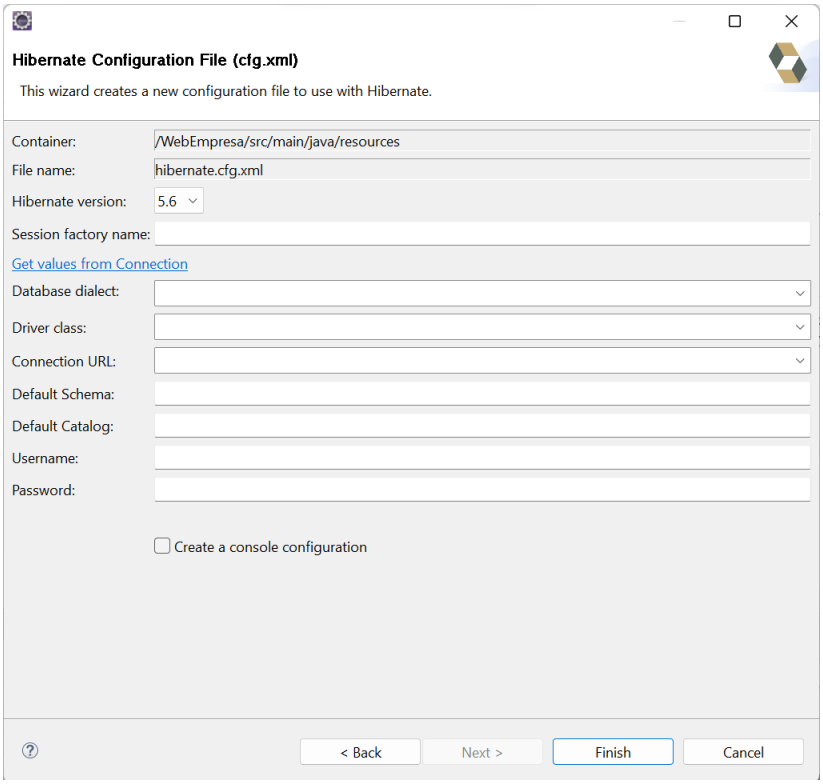


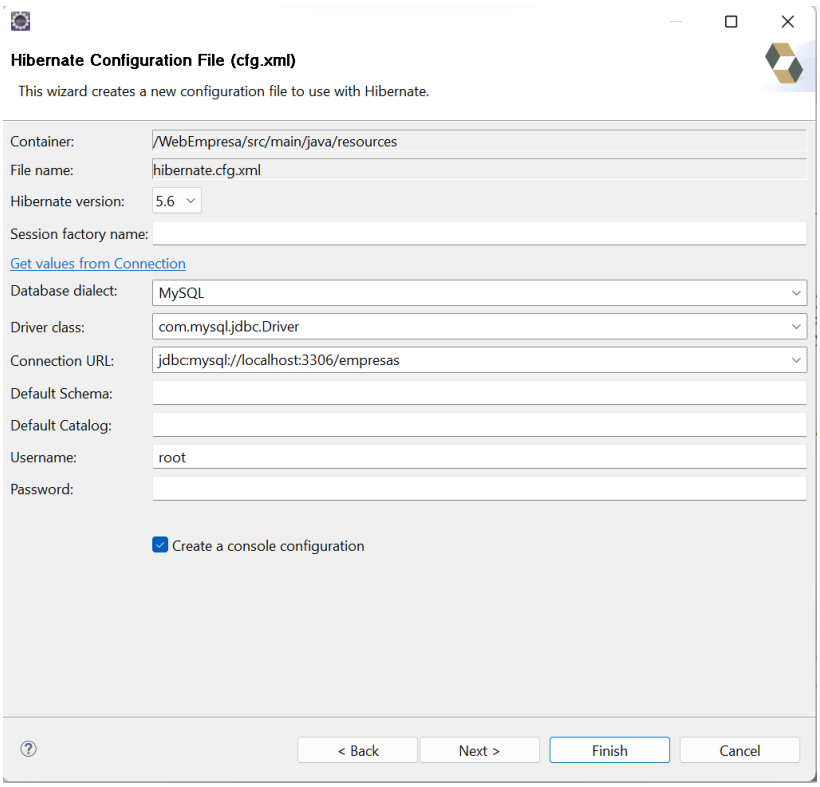
- ✓ Desde la perspectiva de Hibernate se crea una nueva configuración, en **New/Hibernate Configuration File (cfg.xml)** con el nombre de **hibernate.cfg.xml** (dentro de src).

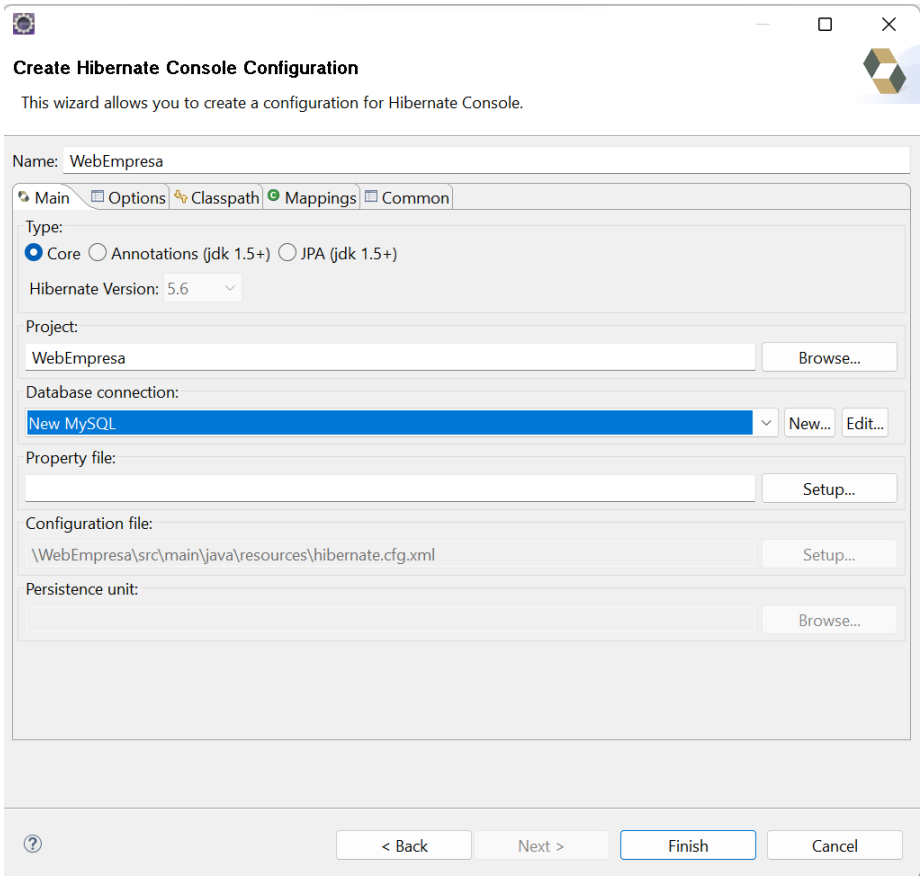

eclipse-workspace - Eclipse IDE
File Edit Navigate Search Project Run Window Help
New Alt+Shift+N
Open File...
Open Projects from File System...
Recent Files
Close Editor Ctrl+W
Close All Editors Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Import...
Export...
Properties Alt+Enter
Switch Workspace
Restart
Exit
Fuse Online Extension Project
Fuse Integration Project
Project...
Camel XML File
Camel Test Case
Fuse Message
Hibernate XML Mapping file (hbm.xml)
Hibernate Configuration File (cfg.xml)
Hibernate Cons
Create a new hibernate.cfg.xml file (Helping with the initial JDBC setup etc.)
Hibernate Reverse Engineering File (reveng.xml)
Example...
Other... Ctrl+N
Error Log
Hibernate Query Result
Hibernate Dynamic SQL Preview
Console
/orkspace Log
type filter text


Create Hibernate Configuration file (cfg.xml)
To load Configuration from 'hibernate.cfg.xml' you'll have to pass file name to Configuration.configure() method
Enter or select the parent folder:
WebEmpresa/src/main/java/resources
WebAppAlumnos
WebAppHibernate
WebApphibernate3
WebEmpresa
.settings
src
main
java
controller
dao
entities
logic
resources
webapp
test
target
File name: hibernate.cfg.xml
Advanced >>
< Back
Next >
Finish
Cancel

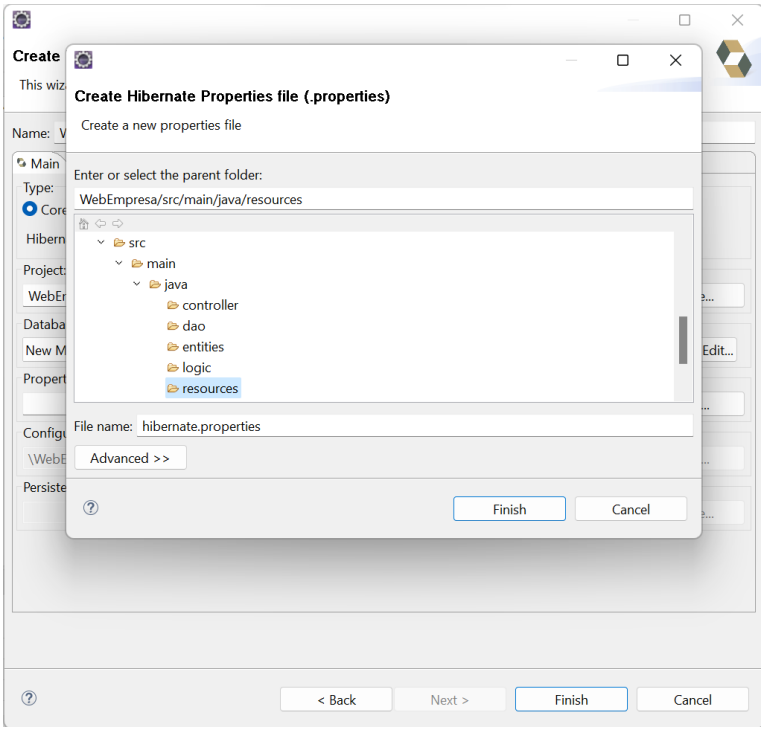
- ✓ Se selecciona el proyecto en el que se está trabajando, y la conexión de base de datos que se acaba de crear. El tipo depende de cómo se trabajará los mapeos y la persistencia, de momento se usa Core, ya que voy a trabajar con Hibernate en nativo.

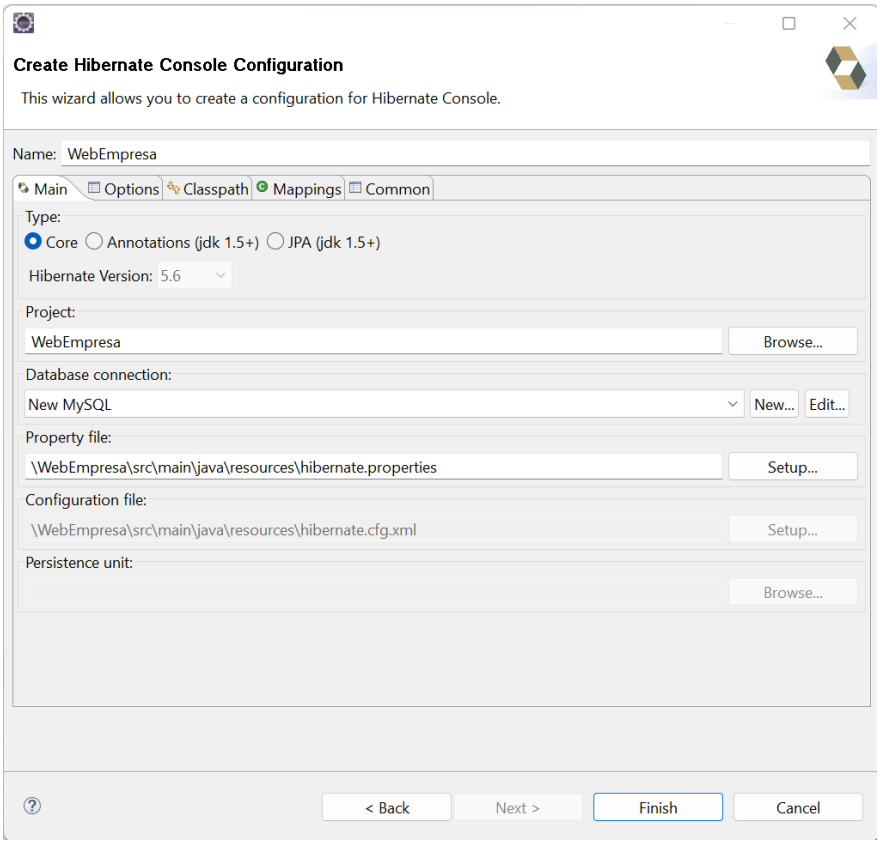


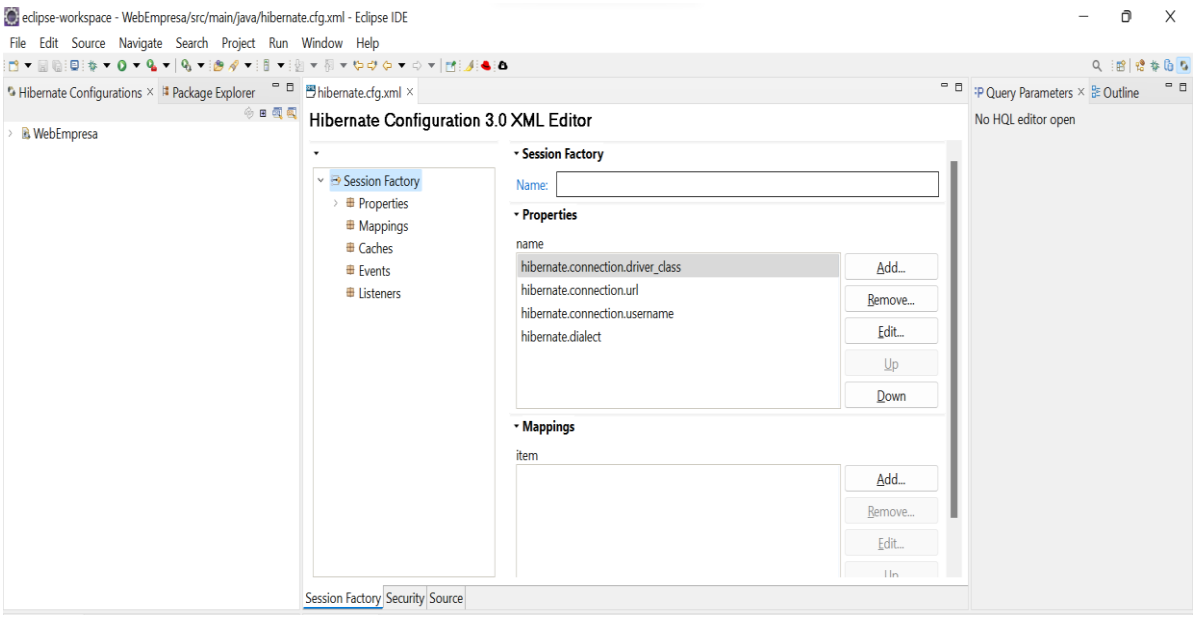




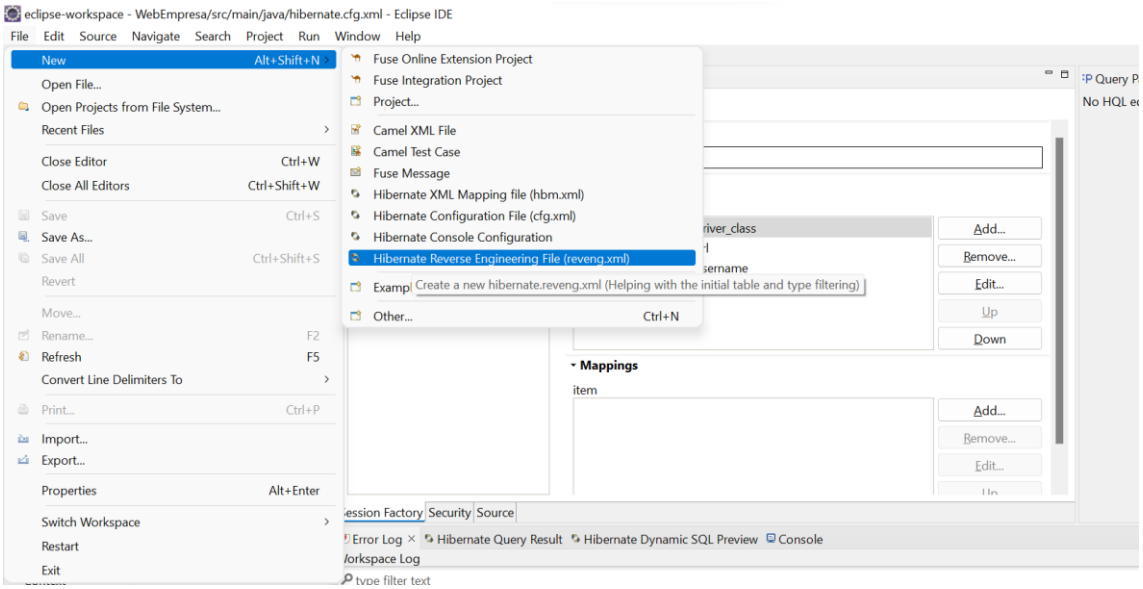
✓ En la parte de Property File, clickeamos en Setup y se crea uno nuevo archivo de propiedades de hibernate, con el nombre de **hibernate.properties** (dentro de src).

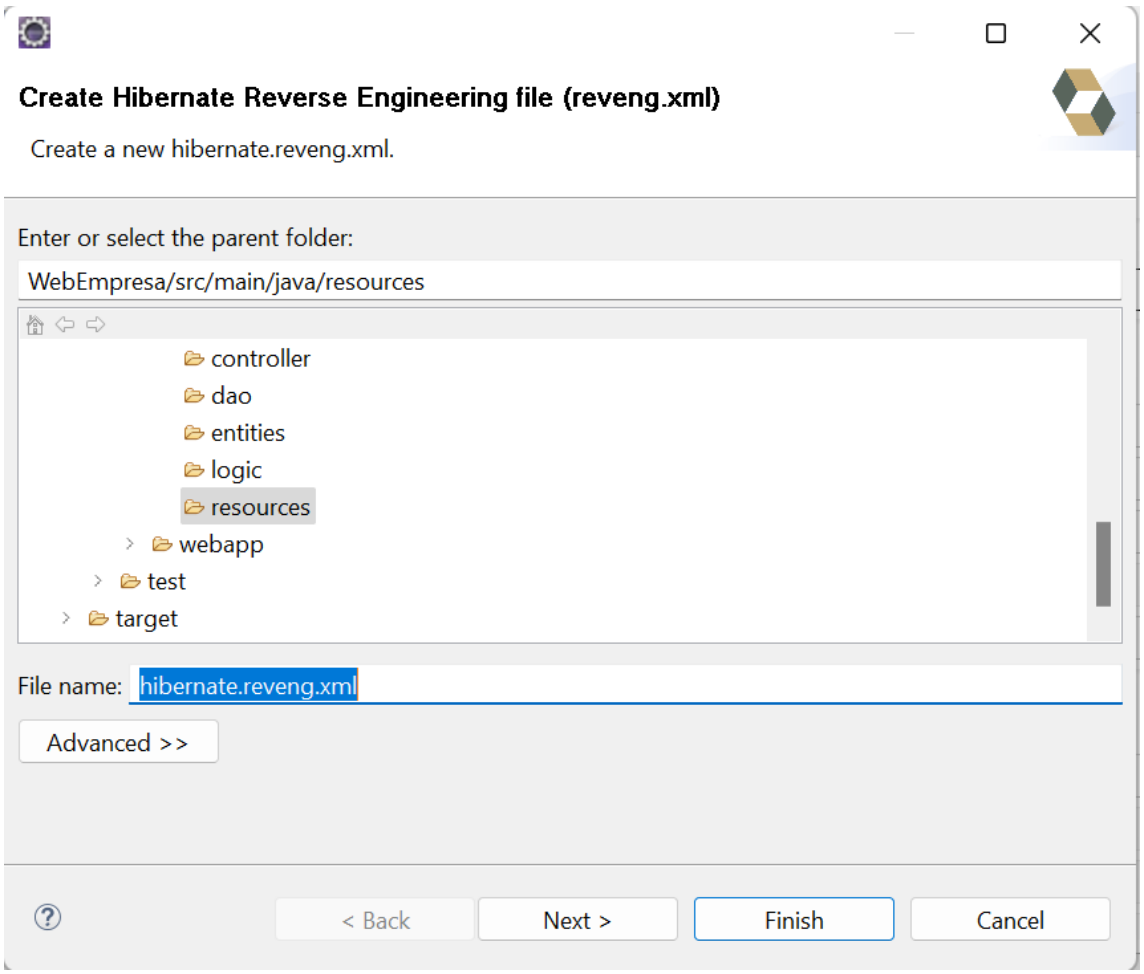

Create
Create Hibernate Properties file (.properties)
Create a new properties file
Enter or select the parent folder:
WebEmpresa/src/main/java/resources
src
main
java
controller
dao
entities
logic
resources
File name: hibernate.properties
Advanced >>
Finish
Cancel
< Back
Next >
Finish
Cancel

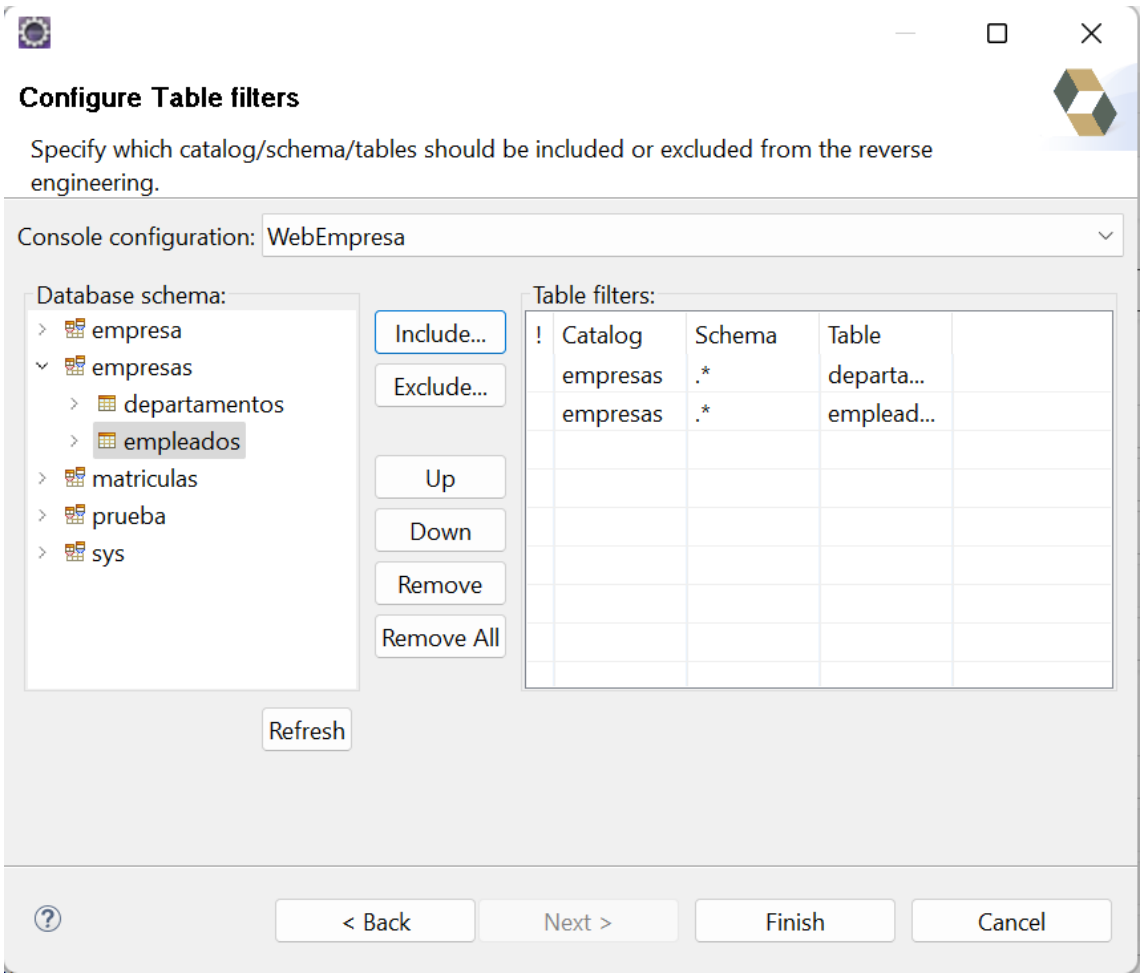

Create Hibernate Console Configuration
This wizard allows you to create a configuration for Hibernate Console.
Name: WebEmpresa
Main
Options
Classpath
Mappings
Common
Type:
Core
Annotations (jdk 1.5+)
JPA (jdk 1.5+)
Hibernate Version: 5.6
Project:
WebEmpresa
Browse...
Database connection:
New MySQL
New...
Edit...
Property file:
\WebEmpresa\src\main\java\resources\hibernate.properties
Setup...
Configuration file:
\WebEmpresa\src\main\java\resources\hibernate.cfg.xml
Setup...
Persistence unit:
Browse...
< Back
Next >
Finish
Cancel



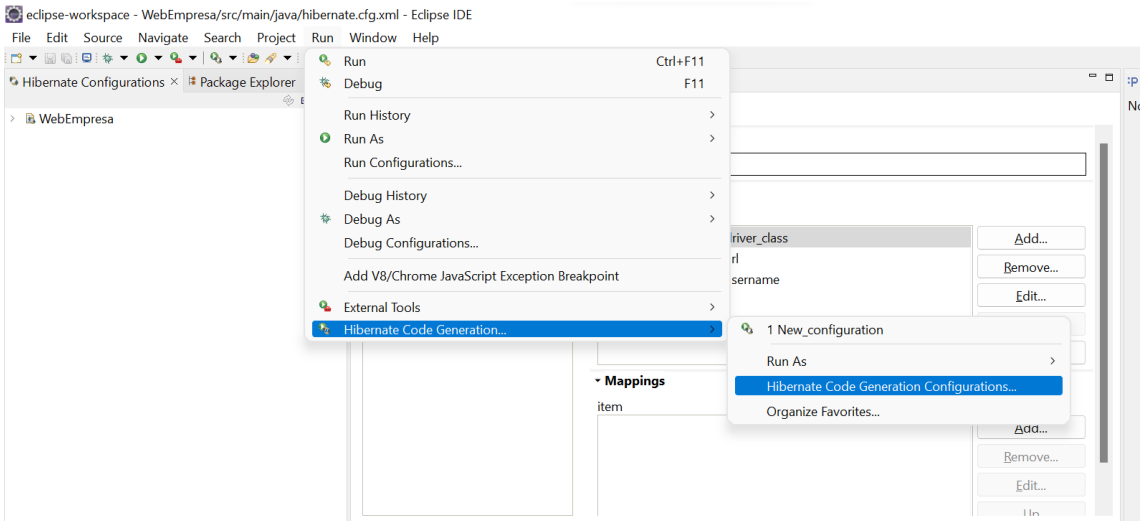
✓ Se crea una nueva configuración **Reverse engineer File (reveng.xml)**, ésta característica permitirá que Hibernate mapee automáticamente las tablas de la base de datos a clases. Permite proporcionar un grado más preciso de control del proceso de ingeniería inversa. En este archivo puede especificar asignaciones de tipos y filtrado de tablas.

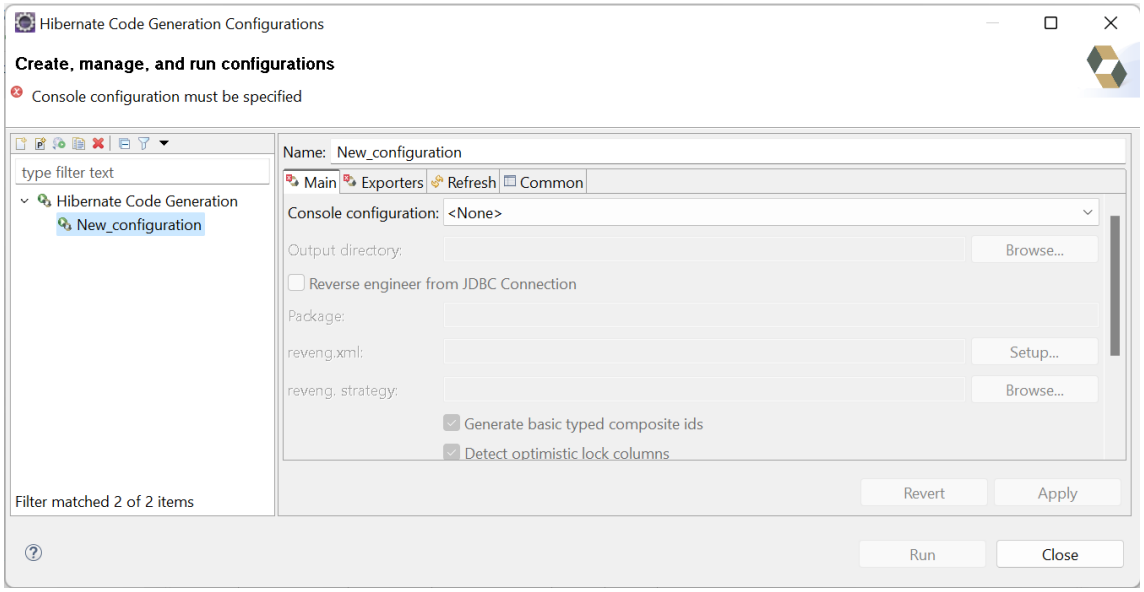



Create Hibernate Reverse Engineering file (reveng.xml)
Create a new hibernate.reveng.xml.
Enter or select the parent folder:
WebEmpresa/src/main/java/resources
controller
dao
entities
logic
resources
webapp
test
target
File name:
hibernate.reveng.xml
Advanced >>
< Back
Next >
Finish
Cancel

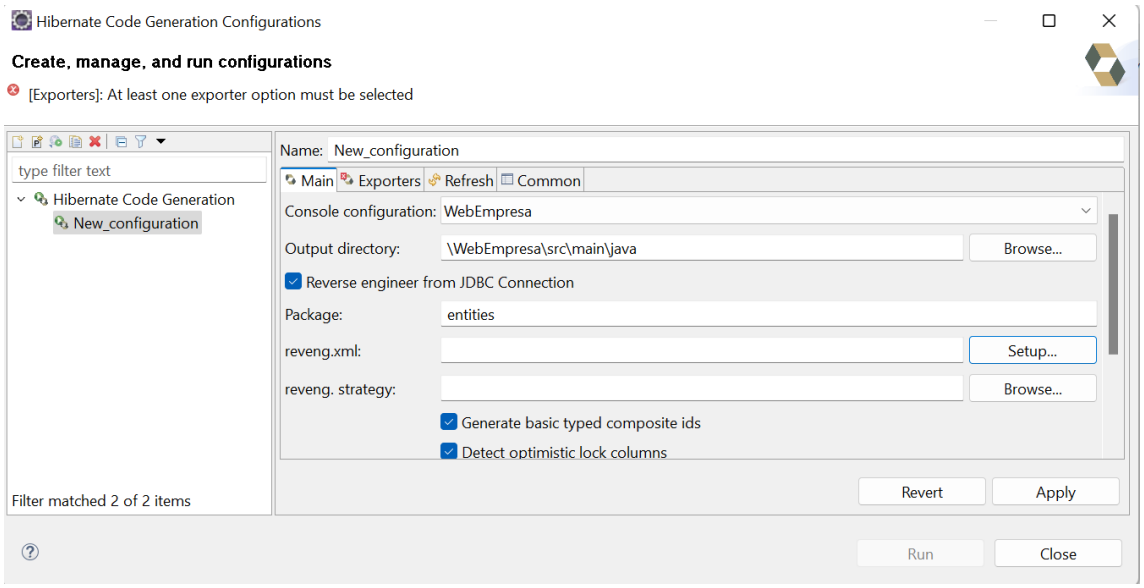


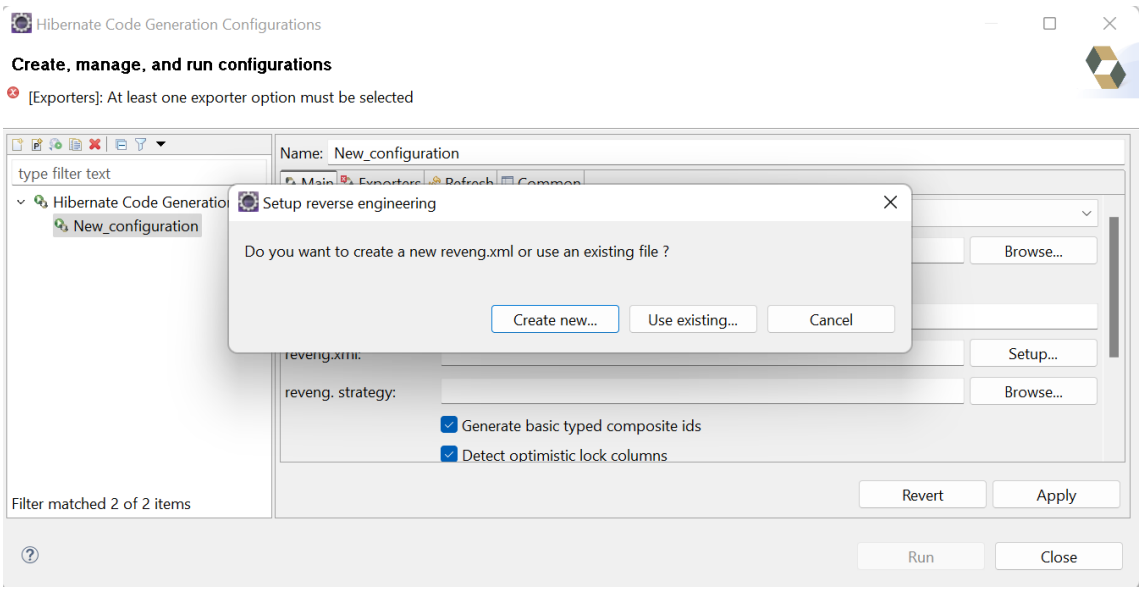
✓ Establecida la nueva configuración de Hibernate, se hará uso de una de las funcionalidades más potentes de Hibernate Tools que es la generación de código. Para esto, se accede a **Run -> Hibernate Code Generation -> Hibernate Code Generation Configuration**

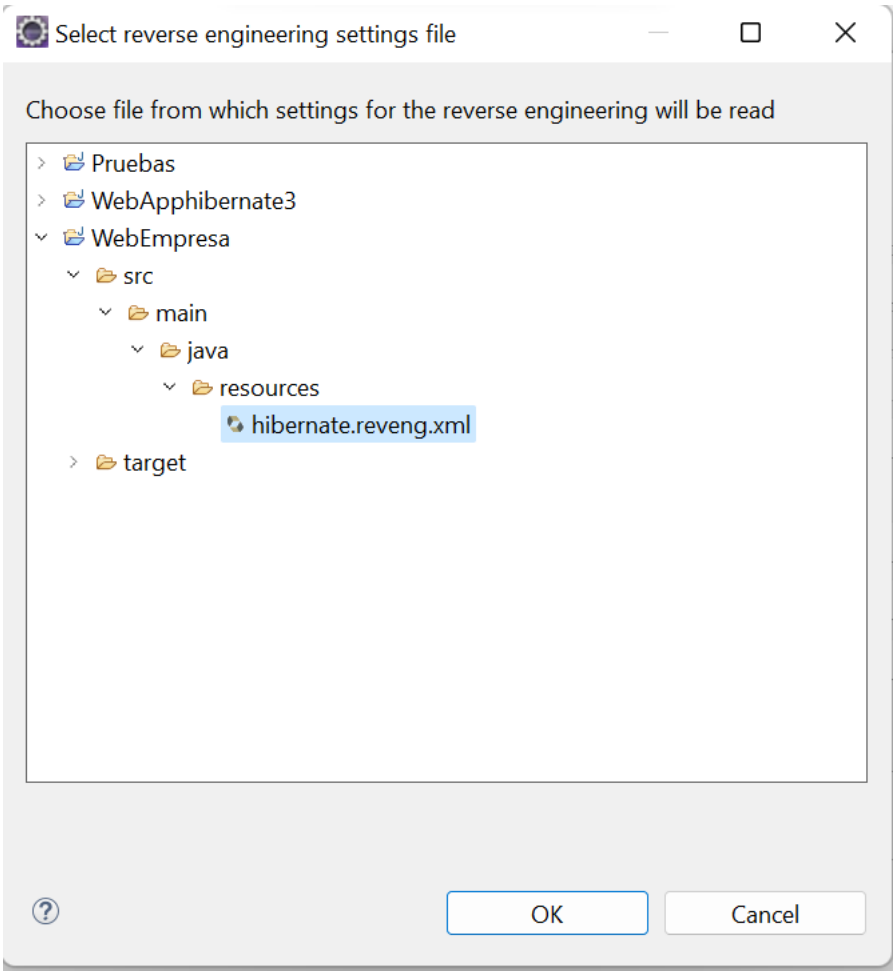


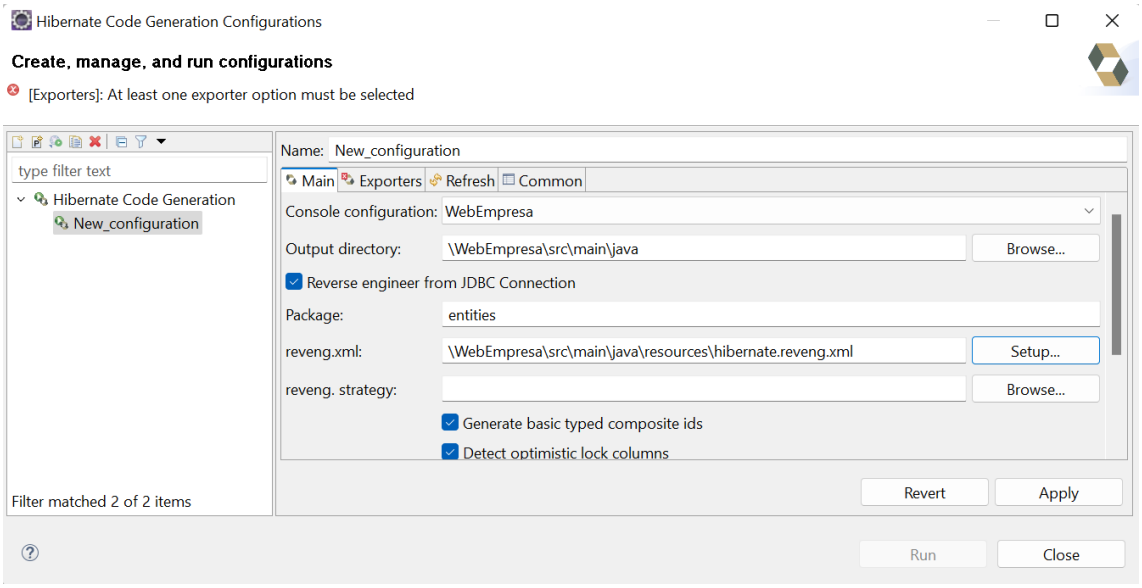


✓ Se ha creado una nueva configuración, y se tilda **Reverse engineer from JDBC Connection**, ésta característica permitirá que Hibernate mapee automáticamente las tablas de nuestra base de datos a clases como ya he comentado. Se selecciona el archivo **hibernate.reveng.xml** creado anteriormente.

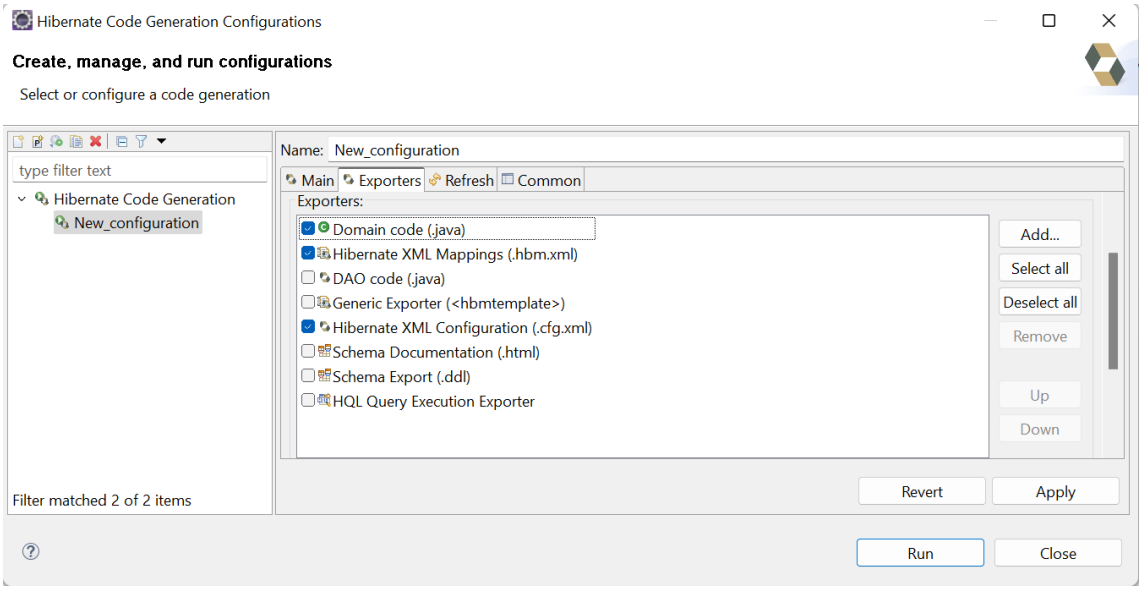



Hibernate Code Generation Configurations
Create, manage, and run configurations
[Exporters]: At least one exporter option must be selected
type filter text
Hibernate Code Generation
New_configuration
Filter matched 2 of 2 items
Name: New_configuration
Setup reverse engineering
Do you want to create a new reveng.xml or use an existing file ?
Create new...
Use existing...
Cancel
Browse...
Setup...
reveng. strategy:
Browse...
Generate basic typed composite ids
Detect optimistic lock columns
Revert
Apply
Run
Close


Select reverse engineering settings file
Choose file from which settings for the reverse engineering will be read
Pruebas
WebApphibernate3
WebEmpresa
src
main
java
resources
hibernate.reveng.xml
target
OK
Cancel



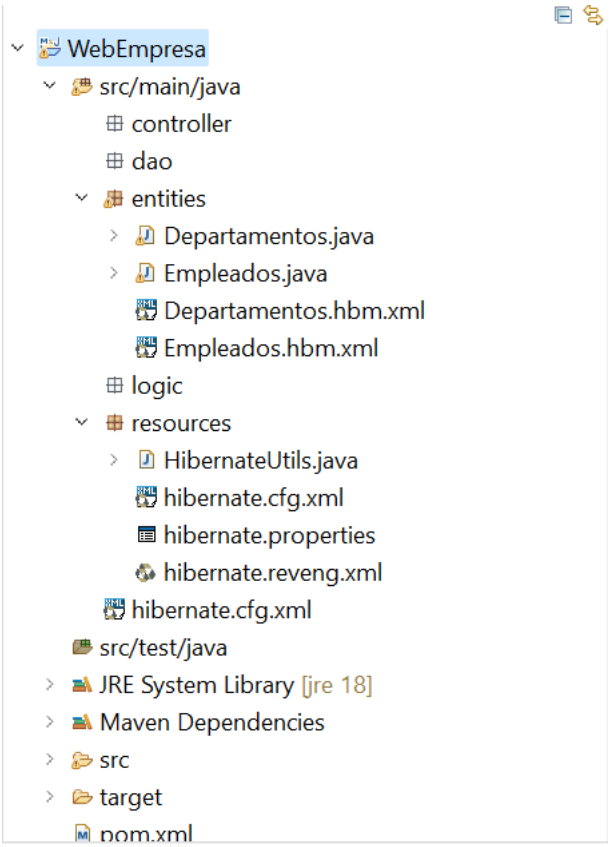
✓ En la pestaña **Exporters** se puede seleccionar que es lo que va a producir ese mapeo. Lo mínimo que se debería seleccionar son: los archivos .java (obviamente), los archivos .hbm.xml, que son los que establecen el mapeo en sí, y el archivo .cfg.xml, que es el archivo que creamos anteriormente, y se modificará para referenciar los .hbm.xml.



✓ Si se ejecuta Run se generarán las clases correspondientes a las tablas que existen en la base de datos. Hibernate utiliza los ficheros de mapeo para relacionar las tablas

de la BD con los objetos Java. Estos ficheros están en formato XML, tienen extensión **.hbm.xml** y se almacenan en el mismo directorio que las clases de Java.

✓ Se debe de tener en cuenta que ante cualquier cambio en el esquema de la base de datos, se debe volver a realizar la generación de código para actualizar los mapeos.



Finalmente, solo queda hacer la configuración de inicio (bootstrapping) y probarlo. Para el boostrapping se crea una clase llamada **HibernateUtils**.

✓ **HibernateUtils.java**

```
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;

public class HibernateUtils {
    private static final SessionFactory sessionFactory = buildSessionFactory();
    private static SessionFactory buildSessionFactory() {
        try {
            // Se configura Hibernate a partir del archivo.cfg.xml
            Configuration config = new Configuration();
            //Añadimos la ruta a los hbm.xml
            config.configure().addResource("\\entidades\\Empleados.hbm.xml");
            config.configure().addResource("\\entidades\\Departamentos.hbm.xml");
            // Se construye la SessionFactory
            ServiceRegistry sr = new StandardServiceRegistryBuilder().applySettings(config.getProperties()).build();
            return config.buildSessionFactory(sr);

        } catch (Throwable ex) {
            System.err.println("No se pudo crear la SessionFactory:" + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

✓ **PruebaHibernate**

```
/**
 * Servlet implementation class ServletHibernate
 */
@WebServlet("/ServletHibernate")
public class ServletHibernate extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public ServletHibernate() {
        super();
        // TODO Auto-generated constructor stub
    }
```

4.5 SESIONES Y OBJETOS HIBERNATE

✓ SESIONES

Para poder utilizar los mecanismos de persistencia Hibernate se debe inicializar el entorno Hibernate y obtener un objeto Session utilizando la clase SessionFactory. Esta clase llama al método configure() que carga el fichero de configuración hibernate.cfg.xml a partir del cual se construye la SessionFactory para obtener el objeto de sesión.

//Apertura de una session Hibernate Session sesion=HibernateUtil.getSessionFactory().openSession(); //Cierre de una session Hibernate sesion.close();

La clase HibernateUtil.java que hemos visto en el ejemplo anterior obtiene información del fichero de configuración hibernate.cfg.xml y nos permite obtener la sesión actual desde cualquier punto de la aplicación.

La clase Session tiene varios métodos como son:

- **beginTransaction():** método para establecer puntos de comienzo de las transacciones para poder hacer commit o rollback
- **save():** método para crear un objeto persistente en la BD
- **delete():** método para eliminar los datos de un objeto de la BD
- **update():** método para actualizar un objeto de la BD
- **get():** método para recuperar un objeto
- **createQuery():** método para crear una consulta en HQL (Hibernate Query Language)

Para poder utilizar los mecanismos de persistencia de Hibernate se debe inicializar el entorno Hibernate y obtener un objeto Session utilizando la clase SessioFactory, del siguiente modo:

```
//Carga el fichero de configuración .cfg.xml e inicializa el entorno Hibernate Configuration
conf=new Configuration().configure();

//Se instancia la SessionFactory para todas las sesiones relacionadas
SessionFactory sesionFact= conf.buildSessionFactory();
Session Session session=sessionFactory.openSession();
```

✓ **TRANSACCIONES**

Un objeto Session de Hibernate representa una única unidad de trabajo para un almacén de datos y lo abre una instancia de SessionFactory. También es necesario efectuar el control de las transacciones indicando su comienzo y su validación (commit) o vuelta atrás (rollback). Por ello hay que efectuar los siguientes pasos:

```
// Abrimos la sesion
Session sesion=sesionFact.openSession();
// Iniciamos una transacción
Transaction tr= sesion.beginTransaction();
//Código
// Validación de la transacción
 tr.commit(); tr.rollback();
```

//Cierre de la sesión

sesion.close();

➢ Crear base de datos llamada empresa formada por dos tablas: Departamentos y Empleados.

depNo	dnombre	dloc
1	RRHH	PALENCIA
2	PROGRAMACION	VALLADOLID

empNo	nombre	apellido	oficio	salario	depNo
1	PEDRO	FERRERO	JEFE RRHH	2500	1
2	JUAN	TROYANO	JEFE RRHH	2000	1
3	SONIA	CRESPO	JUNIOR	1000	2
4	RUTH	FERNANDEZ	SENIOR	800	2

➢ Crear una nueva configuración de Hibernate para la generación del código (POJOS y ficheros XML).

➢ Probar desde una aplicación la inserción, actualización, borrado y consulta sobre la base de datos de empresa.

✓ CARGA DE OBJETOS

Los métodos load() de Session proporciona una forma de recuperar una instancia persistente conociendo el identificador del registro.

```
// Visualizar los datos del departamento 1
Departamentos d=new Departamentos();
d=(Departamentos) sesion.load(Departamentos.class,(int) 1);
try{
   System.out.println("Los datos del departamento 1 son "
       +d.getDnombre()+"  "+d.getLoc());
}catch(ObjectNotFoundException oe){
       oe.printStackTrace();
       System.out.println("El departamento 1 no existe");
 }
```

✓ ALMACENAMIENTO, MODIFICACIÓN Y BORRADO DE OBJETOS

- Para almacenar objetos en la BD se utiliza el método save() de Session.

```
Departamentos d = new Departamentos();
//d.setDepNo(3);
d.setDloc("Palencia");
d.setDmonbre("Jefe de Proyecto");
session.save(d);
```

- Para borrar objetos se utiliza el método delete () de Session.

```
Departamentos d=new Departamentos();
d=(Departamentos) sesión.load(Departamentos.class, (int) 3);
sesion.delete(d);
```

- Para modificar un objeto se utiliza el método update() de Session.

```
Departamentos d=new Departamentos();
d=(Departamentos) sesión.load(Departamentos.class, (int) 2);
d.setDnombre("Programador");
d.setLoc("PALENCIA");
// Se modifican los datos
 sesion.update(d);
```

✓ CONSULTAS HQL (HIBERNATE QUERY LANGUAGE)

Hibernate aportar un lenguaje de consulta propio para recuperar como objetos los datos de la base de datos relacional. Ese lenguaje suele ser SQL o una aproximación a SQL.

Para el caso de Hibernate el lenguaje utilizado se llama HQL (Hibernate Query Language). Este lenguaje es una versión de la sintaxis de SQL, adaptada para devolver objetos.

La principal particularidad de HQL es que las consultas se realizan sobre los objetos Java creados en la aplicación, es decir las entidades que se hacen persistentes en Hibernate (o POJOs).

Además de estas características, para trabajar con HQL es necesario tener en cuenta una serie de consideraciones.

- **Mayúsculas:** el lenguaje no es sensible a mayúsculas y minúsculas en lo que corresponde con las palabras reservadas del lenguaje. Sin embargo, sí es sensible para el caso de los nombres de las clases y de sus atributos.
- **Filtrado:** como en SQL se pueden hacer criterios de selección con la cláusula WHERE. Sin embargo, no hay que confundir que las propiedades y nombre de

los objetos hacen referencia a los POJO y no a las tablas de la base de datos (como sí ocurre con SQL).

- **En las cláusulas WHERE** los literales van entre comas simple(‘) y los decimales se expresan con punto (.). Además, se pueden usar operadores =,,<=,>=,!=(distinto) y like para cadenas. Los criterios se pueden concatenar con operadores lógicos AND, OR y NOT.
- **En la cláusula SELECT** se suele poner una referencia a un objeto. Pero también se pueden usar funciones de agregación sobre atributos de las clases. Algunas son:AVG (media aritmética), SUM (suma de valores), COUNT (contar elementos). En realidad permite la gran mayoría de las que permite SQL.

✓ EJECUTAR HQL DESDE JAVA

Desde un proyecto Java con Hibernate se pueden ejecutar consultas HQL utilizando la clase Query (org.hibernate.Query Query). Esta clase representa una consulta HQL y la ejecuta para devolver el resultado como objetos Java. Las consultas Query son creadas con el método createQuery() de la clase Session.

Uno de los métodos más usados de la clase Query es list(). Este método devuelve un java.io.List (Colección) con los objetos devueltos por una consulta HQL.
Otro de los métodos utilizados es iterate() que devuelve un iterador Java. La consulta se realiza obteniendo el id de las entidades y en cada llamada se devuelve la entidad completa.

Ejemplo Método List()

```
// Generamos la consulta
Query consulta= sesion.createQuery("from Departamentos");
// Visualizamos los datos de todos los departamentos
List <Departamentos> listaDep=consulta.list();
// Obtenemos los datos sobre un iterador
Iterator it=listaDep.iterator();
System.out.println("Numero de departamentos "+ listaDep.size());
while(it.hasNext()){
      Departamentos d=new Departamentos();
      d=(Departamentos) it.next();
      System.out.println("Departamento             "+d.getDeptNo()+"
"+d.getDnombre()
           +"  "+d.getLoc());
}
```

Ejemplo método Iterate()

```
Query consulta= sesion.createQuery("from Departamentos");
// Visualizamos los datos de todos los departamentos
System.out.println("Datos de los departamentos");
consulta.setFetchSize(10);
// Obtenemos los datos sobre un iterador
Iterator it=consulta.iterate();
Departamentos d=new Departamentos();
while(it.hasNext()){
      d=(Departamentos) it.next();
      System.out.println("Departamento"+d.getDeptNo()+"
"+d.getDnombre()+"  "+d.getLoc());
}
```

Para los casos en que se deban tener acceso simultáneo a todas las entidades resultantes de un Query, el método Query.list() cumplirá con los requisitos, teniendo en cuenta que todos los resultados esperados ocuparán espacio en memoria.

Si lo que se necesita es realizar una operación con cada entidad y luego esa entidad no se utilizará más, el método Query.iterate() se ajusta mejor ya que permite limitar (con Query.setFetchSize()) la cantidad de resultados que se tienen en memoria y no consumir recursos innecesariamente, haciendo que la aplicación sea más eficiente y menos propensa a fallos por falta de memoria en casos potenciales de muchos resultados en un query.

Con createQuery() se pueden enlazar valores a los parámetros con nombre que son identificadores de la forma :nombre en la cadena de consulta o a los parámetros ? de JDBC. Hibernate numera los parámetros desde 0.

```
// Empleados cuyo número de departamento sea 1 y tengan por oficio
Programador
Query consulta=sesion.createQuery("from Empleados emp "
   + " where emp.departamentos.deptNo=? and emp.oficio=?";
consulta.setInteger(0,1);
consulta.setString(1, "Programador");
```

Para la versión Hibernate 5 o superior Query (org.hibernate.Query) se encuentra deprecated. En su lugar utilizaremos para hacer consultas:
CriteriaQuery(javax.persistence.criteria.CriteriaQuery)

Se crea una instancia de CriteriaBuilder:

```
CriteriaBuilder builder = session.getCriteriaBuilder();
```

Se crea una instancia de CriteriaQuery:

```
CriteriaQuery<Empleados> consulta =
sesion.getCriteriaBuilder().createQuery(Empleados.class);
```

Se define en la llamada a from la entidad a definir en la consulta:

```
consulta.from(Empleados.class);
```

Ejecuta la Query llamando a getResultList() y obteniendo como resultado un List parametrizado de Empleados.

```
List<Empleados> empleados = sesion.createQuery(consulta).getResultList();
Con un Iterator se puede recorrer la lista:
Iterator it = list.iterator();
Empleados empl = new Empleados();
while (it.hasNext()) {
        empl = (Empleados) it.next();
}
```

Si queremos realizar una Query como la siguiente por ejemplo:

```
Select nombre
From Empleados
Where EmpNo = 1
```

Se debe construir el siguiente código:

```
CriteriaBuilder criteriaBuilder = session.getCriteriaBuilder();
CriteriaQuery<Empleados> criteriaQuery =
criteriaBuilder.createQuery(Empleados.class);
//Indicamos el SELECT Y EL WHERE
Root<Empleados> root = criteriaQuery.from(Empleados.class);
criteriaQuery.select(root.get("nombre");
criteriaQuery.where(criteriaBuilder.equal(root.get("EmpNo"), 1));
```

```
Query<Empleados> query = session.createQuery(criteriaQuery);
List<Empleados>results = query.getResultList();
for(String nombre : results){
    System.out.println("Nombre: " + nombre);
}
```