ORM-Hibernate

Herramientas de mapeo objeto-relacional

ORM - ÍNDICE DE CONTENIDOS

1. Concepto de mapeo objeto-relacional (ORM).
2. Características de las herramientas ORM.
3. Instalación y configuración de una herramienta ORM.
4. Estructura de ficheros de Hibernate. Mapeo y clases persistentes.
5. Sesiones. Objeto para crearlas.
6. Carga, almacenamiento y modificación de objetos.
   - Otros conceptos
7. Consultas HQL (Hibernate Query Language).

OBJETIVOS

a) Instalar la herramienta ORM.

b) Configurar la herramienta ORM.

c) Definir los ficheros de mapeo.

d) Aplicar mecanismos de persistencia a los objetos.

e) Desarrollar aplicaciones que modifican y recuperan objetos persistentes.

f) Desarrollar aplicaciones que realizan consultas usando el lenguaje SQL.

g) Gestionar las transacciones.

1. CONCEPTO DE MAPEO OBJETORELACIONAL (ORM)

- El mapeo objeto-relacional (más conocido con sus siglas ORM) es una técnica de programación que permite convertir datos

entre el sistema de tipos utilizado en un lenguaje de programación y el utilizado en una base de datos relacional.

• Las ventajas principales del mapeo objeto-relacional son:

  – Rapidez en el desarrollo.
  – Abstracción de la base de datos.
  – Reutilización.
  – Mantenimiento del código.
  – Lenguaje propio para realizar las consultas.
  – El tiempo utilizado en el aprendizaje.

- Aplicaciones algo más lentas.

2. CARACTERÍSTICAS DE LAS HERRAMIENTAS ORM

• El modelo relacional trata con relaciones y conjuntos de datos.

• El paradigma OO trata con clases de objetos, objetos, atributos, métodos y asociaciones entre objetos.

• Un mapeo objeto-relacional (ORM) tiene como misión evitar estas diferencias y facilitar la tarea del programador.

• Teóricamente a partir de los objetos Java se puede hacer la persistencia en un sistema relacional ejecutando:

- ***orm.save*** (e*lemento_a_persistir*)
- y ***orm.load/get*** (*objeto_persistido*).
  - ✓ load – objeto "proxy" (fake)
  - ✓ get – objeto instancia proveniente de la BD

3. INSTALACIÓN Y CONFIGURACIÓN DE UNA HERRAMIENTA ORM

- En esta sección, para tener una herramienta de referencia para abordar el resto del capítulo, se utilizar Hibernate como ORM que permite el mapeo objeto-relacional para Java.

• Hibernate está diseñado para ser flexible en cuanto al esquema de tablas utilizado y al adaptarse a su uso sobre una base de datos ya existente.

3.1 INSTALACIÓN MANUAL

Para instalar Hibernate (versión 5.X) y poder ser utilizado, por ejemplo, en el IDE Neatbeans 12.X.X se pueden seguir los

siguientes pasos:

– Paso 1: Acceder al sitio de Hibernate.

– Paso 2: Descargar la última versión.

– Paso 3: Una vez descargado el fichero hibernate-release-5.X.X.Final.zip se descomprime.

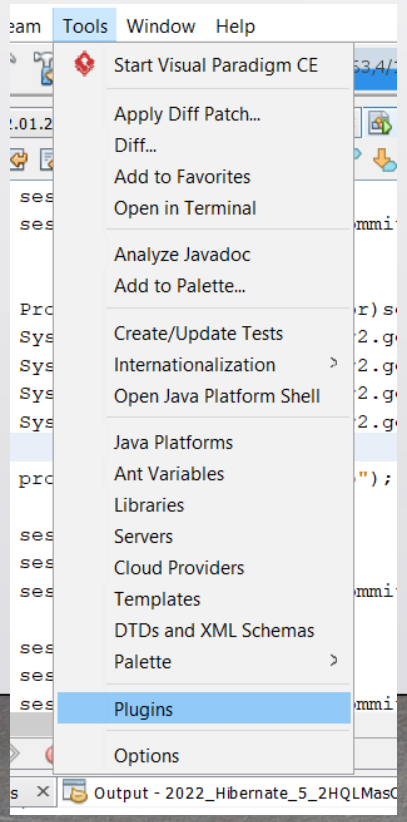
– Paso 4: En este paso se copian todos los ficheros jar que se encuentran en la carpeta **lib\required** en la carpeta lib de nuestro proyecto Java.

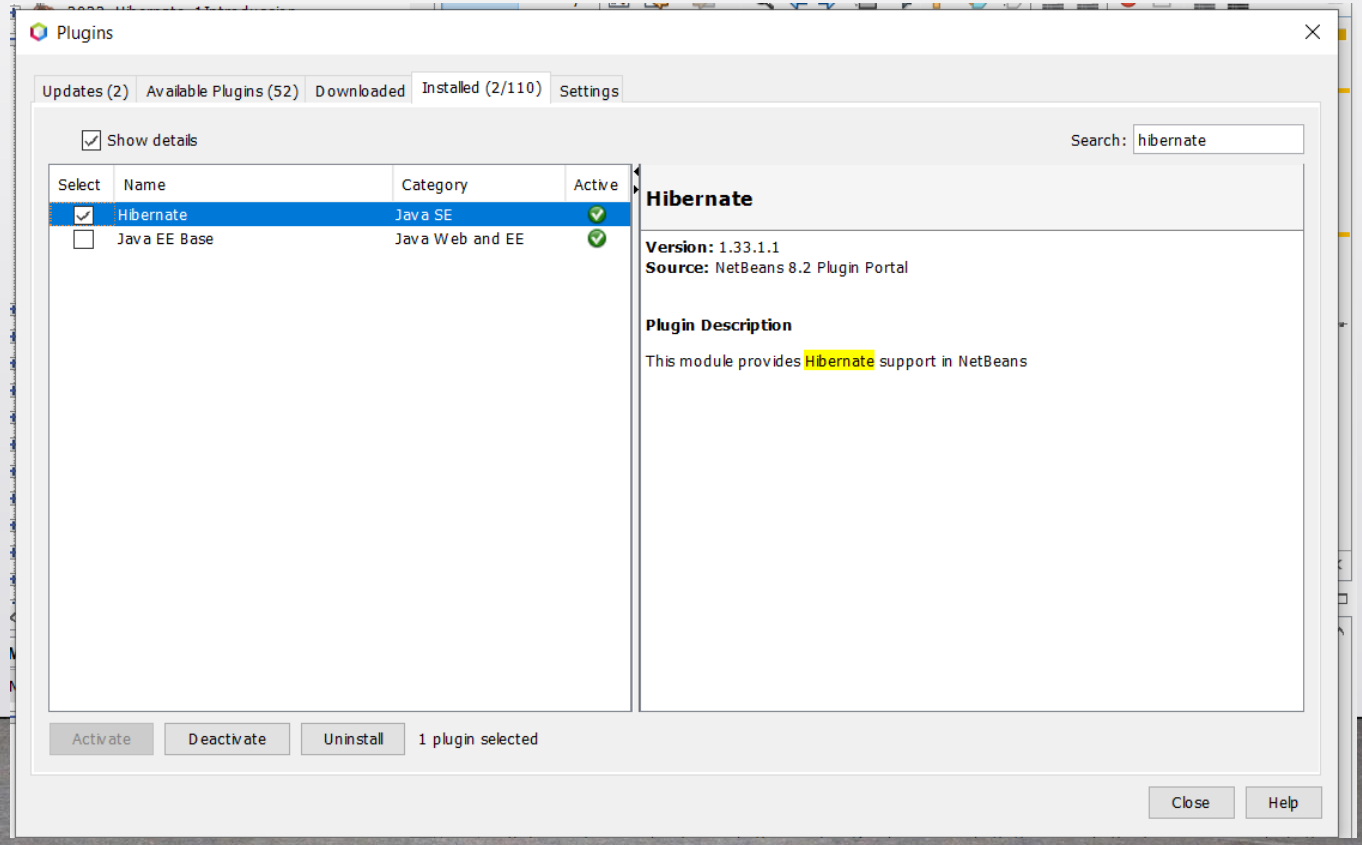
– Paso 5: Copiar el fichero hibernate-entitymanager-5.X.X.Final.jar de la carpeta **lib\jpa** también en la carpeta lib de nuestro proyecto Java.

– Paso 6: Es necesario indicar a NetBeans 12.X.X que se desean usar todas esas librerías, para ello botón derecho pulsar sobre el árbol en el nodo “Libraries” y **seleccionar** la opción de menú “Add Jar/Folder...”

3.2 USAR NETBEANS CON JEE

- Una solución más sencilla que la anterior es descargarse una versión de IDE NetBeans que contenga ya el paquete Hibernate (si fuese posible).
- Descargar para tener accesibles las HibernateTools:





4. ESTRUCTURA DE FICHEROS DE HIBERNATE. MAPEO Y CLASES PERSISTENTES

- Hibernate tiene dos clases importantes:

– Las clases Java (**.java**) que representan los objetos que tienen correspondencia con las tablas de la base de datos relacional.

– El fichero de mapeo (***.hbm.xml***) que indica el mapeo entre los atributos de una clase y los campos de la tabla relacional con la que está asociado.

4.1. CLASES JAVA PARA REPRESENTAR LOS OBJETOS (POJO)

- Las clases Java representan objetos en una aplicación que use Hibernate
- A estas clases Hibernate se refiere a ellas como POJO (Plain Old Java Objects)

```
public class Albumes  implements java.io.Serializable {
    private int id;
    private String titulo;
    private String autor;
    public Albumes() {}
    public Albumes(int id) {  this.id = id; }
    public Albumes(int id, String titulo, String autor) {
      this.id = id;
      this.titulo = titulo;
      this.autor = autor;
    }
    public int getId() {   return this.id;  }
    public void setId(int id) {     this.id = id;}
    public String getTitulo() { return this.titulo  }
    public void setTitulo(String titulo) {      this.titulo = titulo;  }
    public String getAutor() {   return this.autor;  }
    public void setAutor(String autor) { this.autor = autor; }
}
```

4.2. FICHERO DE MAPEO ‘’.hbm.xml‘’ (ó .xml)

- Cada clase que se requiere hacer persistente, se creará un fichero .xml con la información que permitirá mapear esa clase a una BDRel
- Este fichero .xml debería estar en el mismo paquete que la clase cuyos objetos se quieren hacer persistentes.

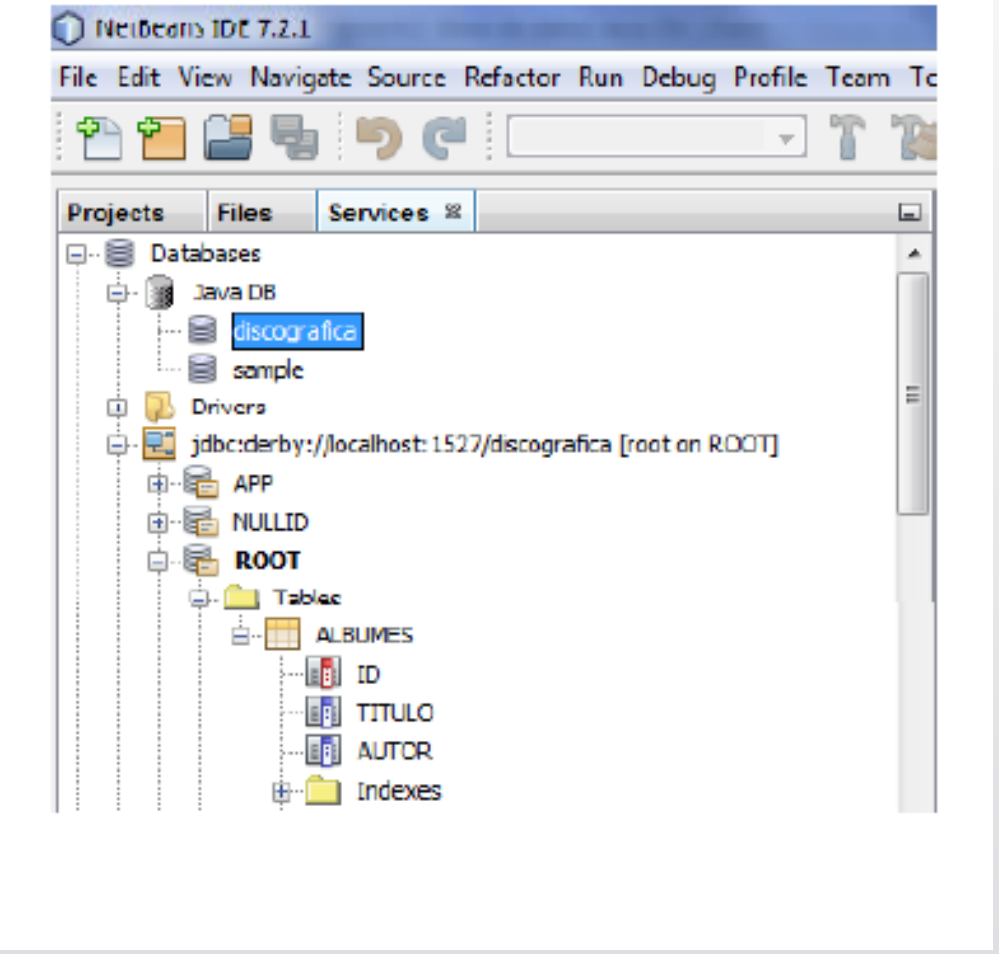
```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 13-ene-2013 10:31:54 by Hibernate Tools 3.2.1.GA -->
<hibernate-mapping>
   <class name="accesohibernate.Albumes" table="ALBUMES" schema="ROOT">
      <id name="id" type="int">
         <column name="ID" />
         <generator class="assigned" />
      </id>
      <property name="titulo" type="string">
         <column name="TITULO" length="30" />
      </property>
      <property name="autor" type="string">
         <column name="AUTOR" length="20" />
      </property>
   </class>
</hibernate-mapping>
```

4.3. CREAR FICHEROS DE MAPEO CON NETBEANS

- Antes de explicar los pasos a seguir es necesario crear el proyecto Java y una conexión a una bases de datos.

– Crear con NetBeans un proyecto tipo Aplicación Java (Java Application).

– Crear una base de datos (pej. en el propio entorno de IDE NetBeans (JavaDB), MySQL..).



4.3. CREAR FICHEROS DE MAPEO CON NETBEANS

➢ Fichero de configuración:

- Al proyecto Java se le añade un nuevo archivo (hibernate.cfg.xml). Este archivo contendrá la información necesaria para conectar a la BD sobre la que se hará la persistencia.
- Para crearlo, se pueden seguir los siguientes pasos:

1. File/New/(others..) Hibernate Configuration File
2. Siguiente
3. Nombre por defecto (hibernate.cfg.xml). Siguiente
4. Seleccionar en la lista desplegable la conexión a la BD (creada con anterioridad)
5. Una vez seleccionada, Terminar.

4.3. CREAR FICHEROS DE MAPEO CON NETBEANS

➢ Fichero de ingeniería inversa (prescindible)

- Este archivo indicará el esquema en que se encontrará la tabla a mapear
- Para crearlo, se siguen los siguientes pasos:

1. (1) seleccionando el paquete **default package** se pulsa con el botón derecho y se selecciona **Nuevo** e *Hibernate Reverse Engineering Wizard*.
2. Pulsar Siguiente.
3. El nombre por defecto del fichero será *hibernate.revenge* pulsar Siguiente.
4. Se selecciona la tabla que se quiere mapear
5. Una vez seleccionada se pulsa Terminar.

4.3 CREAR FICHEROS DE MAPEO CON NETBEANS

➢ Fichero POJO y de mapeo

- Fich. POJO-clase Java a partir tablas BD y ficheros mapeo (hbm.xml)
- Pasos:
  - *default package*-btn. Dercho/Nuevo/Otros/Hibernate mapping files
  - Sigte.
  - Indicar el nombre de paquete en que almacenar los nuevos fich. a crear.
  - Terminar.

4.3 CREAR FICHEROS DE MAPEO CON NETBEANS

➢ Ficheros **HibernateUtil.java**

- Fich. Gestión Conexiones BD – mapear Objs en las Tablas

- Creación:
  1. Paquete con los mapeos. Btn Drcho./Nuevo/Otros/HibernateUtil.java
  2. Sigte.
  3. Nombre (por defecto. NewHibernanteUtil)
  4. Terminar

```
public class HibernateUtil {
 public static SessionFactory buildSessionFactory() {
        try {
            return new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
  }
}
```

5. SESIONES. OBJETO PARA CREARLAS

- Session – org.hibernate.Session
- Métodos para leer, guardar o borrar entidades de la BD.
- Clase ***Session*** – métodos:
  - *openSession*()
  - get() – para recuperar objeto
  - load()
  - **beginTransaction**() – para hacer transacciones
  - save() – para hacer persistente objeto en BD
  - saveOrUpdate()
  - delete() – para eliminar datos de objeto
  - update() – para modificar objeto
  - **getTransaction().commit**()
  - *close*()
  - createQuery() – para consulta HQL (Hibernate Query Language) y ejecutar sobre BD

6. CARGA, ALMACENAMIENTO Y MODIFICACIÓN DE OBJETOS

- SessionFactory sf = NewHibernateUtil.buildSessionFactory();
  Session sesion = sf.openSession();
- Session sesion = HibernateUtil.getSessionFactory().openSession();
- sesion.beginTransaction() – para leer no hace falta

GUARDAR

- Sesion.save(objeto)

ACTUALIZAR

- Sesion.update(objeto)

LEER

- sesion.get(objeto.class,id)

BORRAR

- Sesion.delete(objeto)

- sesion.getTransaction().commit(); – para leer no hace falta
- sesion.close();

Un poco de historia

- Estándar de construcción de App Empresariales
  - J2EE vs JEE (ahora Oracle Corporation, inicialmnt Sun Microsystems)
- JPA (*javax.persistence*) -> Especificación en JavaSE(Standard) y JEE(Enterprise)
  - Entities (annotations o documento XML)
  - JPQL
- Hibernate –> Implementación (otras ObjectDB, EclipseLink, OpenJPA..)

- POJO, JavaBeans vs EnterpriseJavaBeans(EJB)
  - POJO (Plain Old)
  - JavaBeans vs EnterpriseJavaBeans(EJB)
  - EJB
    - De Entidad (Persistencia gestionada por Contenedor-CMP, P gestionada por v ean BMP)- (Reemplazados por JPA a partir EJB 3.0.. JEE 5.0)
    - De Sesión (Con Estado-statefull/ Sin Estado-stateless)
    - Dirigidos por Mensajes

JPA

- En la implementación Hibernate:
  - JPA 1.0: Hibernate ORM 3.2+.
  - JPA 2.0: Hibernate ORM 3.5+.
  - JPA 2.1: Hibernate ORM 4.3+.
  - JPA 2.2: Hibernate ORM 5.3+.

JavaBeans vs EnterpriseJavaBeans(EJB)

- JavaBeans (utilizarlos, reutillizarlos, sustituirlos y conectarlos)
  - Para que funcione (nomenclatura Métodos, construcción y comportamiento):
    - Debe tener un constructor sin argumentos.
    - Sus atributos de clase deben ser privados.
    - Sus propiedades deben ser accesibles mediante métodos get y set que siguen una convención de nomenclatura estándar.
    - Debe ser serializable.
  - Tres partes:
    - Propiedades: Los atributos que contiene.
    - Métodos: Se establecen los métodos get y set para acceder y modificar los atributos.
    - Eventos: Permiten comunicar con otros JavaBeans.
- EJB(Enterprise JavaBeans)
  - Tecnología componentes lado Servidor parte de JavaEE

JPQL (Java Persistence Query Language)

- lenguaje de consulta OO. Independiente Plataforma (parte de especificación (JPA)).
- Está inspirado en gran medida por SQL
- Recuperar objetos (consultas SELECT). Consultas de actualización (UPDATE) y borrado (DELETE)
- JPQL está basado en Hibernate Query Language (HQL),
  - Hibernate y HQL se crearon antes de la especificación JPA. Hasta la versión Hibernate 3, JPQL es un subconjunto de HQL.

7. HQL (Hibernate Query Language)

- Anterior a JPQL
- Incluido en la biblioteca mapeo objeto-relacional Hibernate
- Versión de sintaxis SQL, adaptada devolución objetos.
- Características:

– Los tipos de datos son los de Java.

– Las consultas son independientes del lenguaje de SQL específico de la base de datos.

– Las consultas son independientes del modelo de tablas de la base de datos. No se necesita conocer el modelo lo que hace de la independencia una ventaja.

– Es posible tratar con las colecciones de Java (*java.io.List*, por ejemplo).

– Es posible navegar entre los distintos objetos en la propia consulta.

7. HQL (Hibernate Query Language) en Java

- Ej.

```
Session sesion = sf.openSession();
//Anteriormente:
    // List<Ciclos> lista = sesion.createCriteria(Ciclos.class).list();

    String c= "select c from ClaseX c where comuna like %YYYY%";
    List< ClaseX > lista= sesion.createQuery(c).list();
```

Hibernate - Documentación

- https://docs.jboss.org/hibernate/orm/5.6/quickstart/html_single/

- https://docs.jboss.org/hibernate/orm/5.6/userguide/html_single/Hibernate_User_Guide.html

Hibernate y JPA

- Hibernate (implementación)
  - En 2001 Gavin King
- JPA (especificación)
  - Surgió posteriormente
  - podemos usar Hibernate siguiendo un estándar
  - por lo que podríamos cambiar de Hibernate a otra implementación sin problemas

Persistencia Obj-BD con Hibernate

- Métodos
  - Ficheros XML (hibernate.cfg.xml - configuración, .hbm.xml - mapeo)
    - ✓ Propias Hibernate
    - ✓ Hibernate funcionó durante años sin JPA
  - Anotaciones Código
    - ✓ Las del estándar JPA (@Entity…)
    - ✓ Posibilidad - Uso persistence.xml - configuración (en caso de utilizar anotaciones JPA)

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
             http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
             version="2.0" xmlns="http://java.sun.com/xml/ns/persistence">

  <persistence-unit name="PersistenciaMySQL">
  <!-- Representamos las clases. Para que se enlacen a la BD y se creen si noExisten-->
    <class>gal.mrnovoa.hibernate.modelo.Empleado</class>
    <class>gal.mrnovoa.hibernate.modelo.Autor</class>
    <class>gal.mrnovoa.hibernate.modelo.Libro</class>
    <class>gal.mrnovoa.hibernate.modelo.Publicacion</class>
    <class>gal.mrnovoa.hibernate.modelo.Comentario</class>
     <class>gal.mrnovoa.hibernate.modelo.Usuario</class>
```

persistence.xml

```
 <properties>

<!--H2-->

 <!-- <property name="javax.persistence.jdbc.driver" value="org.h2.Driver" />-->

        <!--Para que guarde en Memoria o en Disco -->

        <!--<property name="javax.persistence.jdbc.url" value="jdbc:h2:mem:test" />-->

        <!--Para que guarde en un archivo llamado Empresa ubicado en el escritorio(/Desktop/) con extensión
Empresa.mv.db o directamente en al carpeta del usuario-->

        <!-- <property name="javax.persistence.jdbc.url" value="jdbc:h2:~/Empresa" />-->

        <!--<property name="javax.persistence.jdbc.url" value="jdbc:h2:~/Autores" />-->


 <!--    <property name="javax.persistence.jdbc.user" value="sa" />

        <property name="javax.persistence.jdbc.password" value="" />


        <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect" />

        -->

        <property name="hibernate.hbm2ddl.auto" value="create-drop" />

    </properties>

  </persistence-unit>


</persistence>
```

```
 <!--MySQL  -->
     <property name="javax.persistence.jdbc.driver"
value="com.mysql.jdbc.Driver" />
     <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3307/bdMySQL_hb_eclipse" />
<property name="javax.persistence.jdbc.user" value="root" />
<property name="javax.persistence.jdbc.password" value="usbw" />
<property name="hibernate.show_sql" value="true" />
<property name="hibernate.format_sql" value="false" />
<property name="hibernate.dialect"
value="org.hibernate.dialect.MySQL5InnoDBDialect" />
<!--<property name="hibernate.hbm2ddl.auto" value="create" />-->
```

Hibernate configuración - opciones

HibernateUtil - SessionFactory

- Opción1 (sin HibernateUtil)

```
sessionFactory = new Configuration().configure().buildSessionFactory();
Session session=sessionFactory.openSession();
```

- Opción2
  - ```
    public class NewHibernateUtil {
    public static SessionFactory buildSessionFactory() {
      try {
        return new Configuration().configure().buildSessionFactory();
      } catch (Throwable ex) {
        System.err.println("Initial SessionFactory creation failed." + ex);
        throw new ExceptionInInitializerError(ex);
      }
    }
    ```
  - ```
    SessionFactory sf = NewHibernateUtil.buildSessionFactory();
    Session sesion = sf.openSession();
    ```

EntityManager - EntityManagerFactory

```
//Gestor de persistencia para operaciones CRUD
private static EntityManager manager;//static para poder verlo desde main
//OPCION1.Usando Java EE o EJBs--apps Web
// <persistence-unit name="Persistencia"> en persistence.xml
//@PersistenceContext(unitName = "Persistencia")
//USO con EJBs , pero como no estamos en apps Web->forma TRADICIONAL
//@PersistenceUnit(unitName ="Persistencia" )
//OPCION2. EntityManagerFactory
private static EntityManagerFactory emf = Persistence.createEntityManagerFactory("Persistencia");
public static void main(String[] args) {
      manager = emf.createEntityManager();
      manager.createQuery("FROM Empleado").getMaxResults();//Sin usar de momento JPQL
      manager.createQuery("FROM Empleado").getFirstResult();
      .....
```

Hibernate configuración - opciones

HibernateUtil - SessionFactory

```
Session sesion = sf.openSession();
sesion.beginTransaction();
Ciclos i = (Ciclos) sesion.get(Ciclos.class, id);
sesion.saveOrUpdate(obj);
sesion.save(obj);
sesion.delete(obj);


sesion.getTransaction().commit();
 sesion.close();
```

EntityManager - EntityManagerFactory

```
manager.getTransaction().begin();
Empleado e1 = manager.find(Empleado.class, 10L);
manager.persist(e);
manager.remove(e);


manager.getTransaction().commit();
manager.close();
```

Introduccion al ORM

- 2022_Hibernate_1Introducción

Hibernate

- XML
  - 2022_Hibernate_2_1_XML
- JPA
  - 2022_Hibernate_2_1_XML

Relaciones

- @OneToOne
  - XML
  - JPA
  - Bidireccionales de las 2
- @OneToMany o @ManyToOne
  - XML
  - JPA
  - Ordenado en las 2
- @ManyToMany

Claves Primarias y Tipos de datos

- XML
- JPA
- EjProfesor
  - XML
  - JPA
  - Componente XML y JPA
  - Enumerado XML y JPA

HQL

- Consultas Simples
- Más Consultas
- Colecciones
- Optimización
- Otros

Objetos y Validaciones

Arquitectura

Spring con Hibernate, OpenSessionInView y Spring MVC.