ORM

Mapeo Objeto Relacional

1

Definiciones

- **ORM, O/RM, O/R *mapping*, *Object Relational Mapping*.** Es una técnica de programación para convertir datos entre el sistema de tipos utilizado por un lenguaje de programación orientado a objetos y el utilizado por una base de datos relacional, utilizando un motor de persistencia.
- Básicamente, las tablas se transformarán en clases y las filas en objetos.

2

Ventajas

- **Rapidez en el desarrollo.** Muchas de las herramientas permiten la creación del modelo por medio del esquema de la base de datos.
- **Abstracción de la base de datos.** Se separa el sistema utilizado, de la Base de datos. Un cambio del sistema de almacenamiento no tendría que afectar a la aplicación.
- **Reutilización.** Permite utilizar los métodos de un objeto de datos desde distintas zonas de la aplicación, o desde distintas aplicaciones.
- **Seguridad.** Los ORM suelen implementar sistemas para evitar tipos de ataques como pueden ser los *SQL injections*.
- **Mantenimiento del código.** Debido a la estabilidad de la capa de datos.
- **Lenguaje propio para realizar las consultas.** Se dejan de utilizar la sentencias SQL.

3

Inconvenientes

- **Tiempo utilizado en el aprendizaje.** Las herramientas son complejas en su utilización y además necesitan configuración en su instalación.
- **Aplicaciones algo mas lentas.** Debido al proceso de transformación de los datos: las consultas sobre la base de datos, primero se deben transformar al lenguaje propio de la herramienta, luego leer los registros y por último crear los objetos.

4

Algunas herramientas

- Propel (PHP)
- Doctrine (PHP)
- JPA (Java)
- ADO*db Active Record* (PHP)
- Hibernate (Java)
- LINQ (VB.Net y C#)
- NHibernate (C#)
- peewee (Python)
- Object (Python)
- Sequelize (NodeJs)
- SQLAlchemy (python)

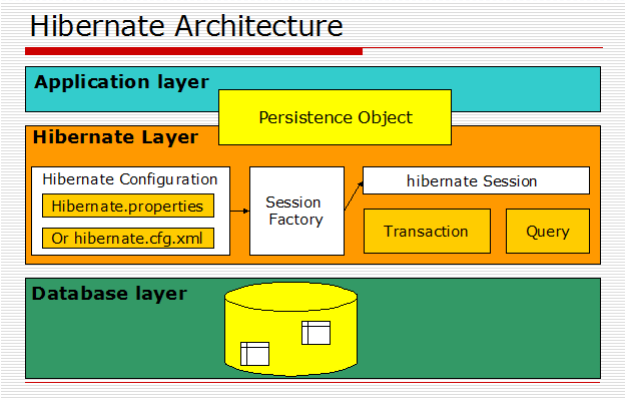
5

Hibernate

- Desarrollado por Red Hat.
- Multiplataforma, ya que se basa en la máquina virtual de Java.
- Software libre, distribuido bajo licencia GNU LGPL.
- Lenguaje de consultas propio, **HQL** (*Hibernate Query Language*).
- Permite independizar la capa de negocio del almacenamiento de la información.
- La capa de persistencia permite abstraer al programador de las particularidades de una determinada base de datos.

6

Arquitectura Hibernate (I)



La filosofía es mapear objetos java normales, conocidos como POJO's (*Plain Old Java Objects*). Para almacenar y recuperar estos objetos es necesario establecer una comunicación con el motor de Hibernate, mediante sesiones (clase ***Session***), similar a las conexiones JDBC.

Arquitectura Hibernate (II)

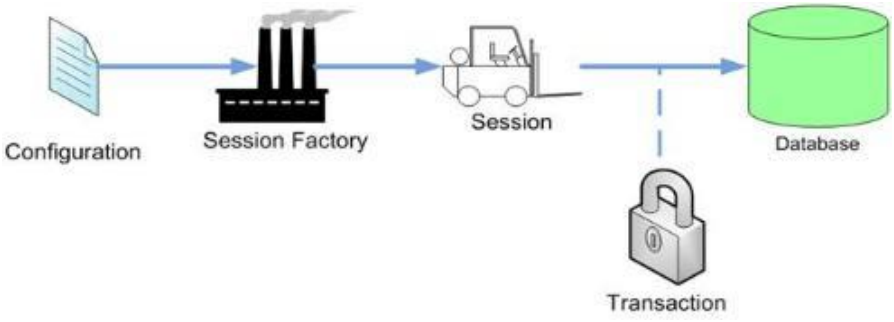
- La clase ***Session*** incorpora una serie de métodos que facilitan la interconexión con la base de datos. Algunos de ellos son:
  - save(Object obj)
  - createQuery(String cons)
  - beginTransaction()
  - close()
- Las instancias de *Session* apenas consumen memoria, por lo que continuamente se están creando y destruyendo.

Interfaces de Hibernate

- **SessionFactory** (*org.hibernate.SessionFactory*). Permite obtener instancias Session. Normalmente se comparte entre los diferentes hilos de ejecución. Suele haber una por aplicación, aunque puede haber más, si se utilizan varias bases de datos, una por cada una.
- **Configuration** (*org.hibernate.cfg.Configuration*). Se utiliza para especificar la ubicación de los documentos que indican el mapeado de los objetos y propiedades específicas de Hibernate.
- **Query** (*org.hibernate.Query*). Permite realizar y controlar las consultas a la base de datos. Pueden ser en HQL, o en el dialecto SQL nativo de la base de datos.
- **Transaction** (*org.hibernate.Transaction*). Permite encapsular un conjunto de instrucciones, de forma que si se produce un fallo en alguna de ellas, se producirá un fallo en la transacción.

9

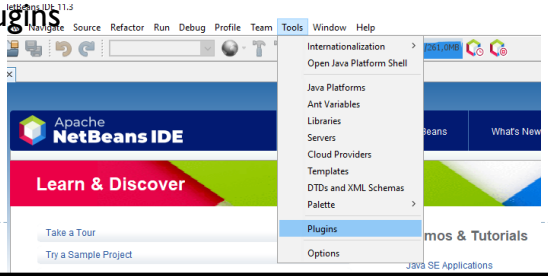
Aplicación con Hibernate



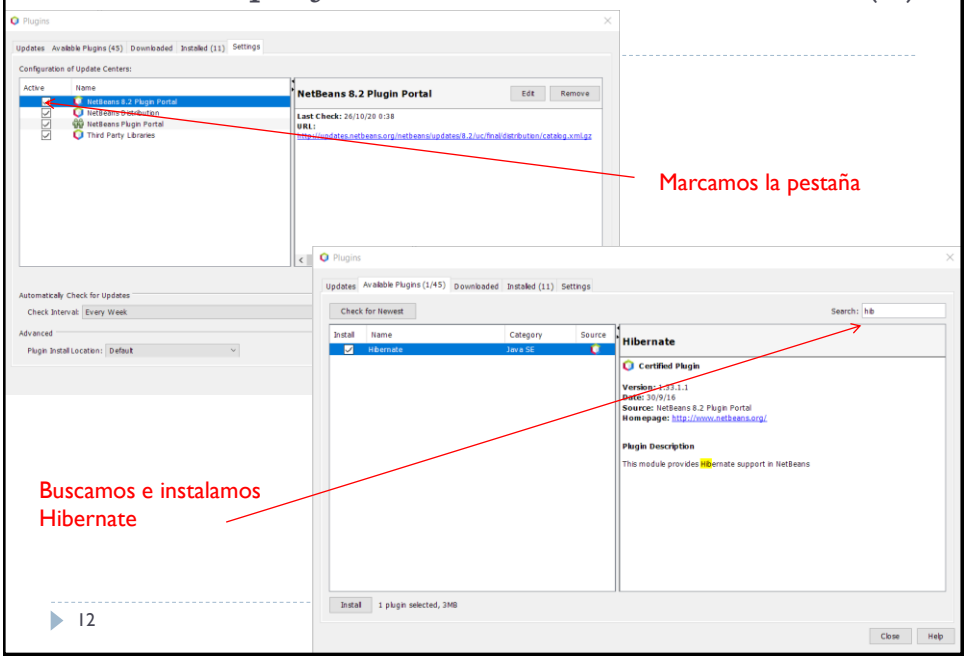
10

Creación de proyecto Hibernate en Netbeans (I)

- La última versión estable de Hibernate es la 6.1.6, existe la versión de desarrollo 6.2.
- Hibernate, de momento, es compatible con las versiones 11, 17 o 18 de Java.
- Por lo que vamos a trabajar con Netbeans 11 y con el jdk 11. Una vez instalados y comprobado que se puede crear un proyecto, es necesario instalar un plugin. Para ello se accede a Tools->Plugins

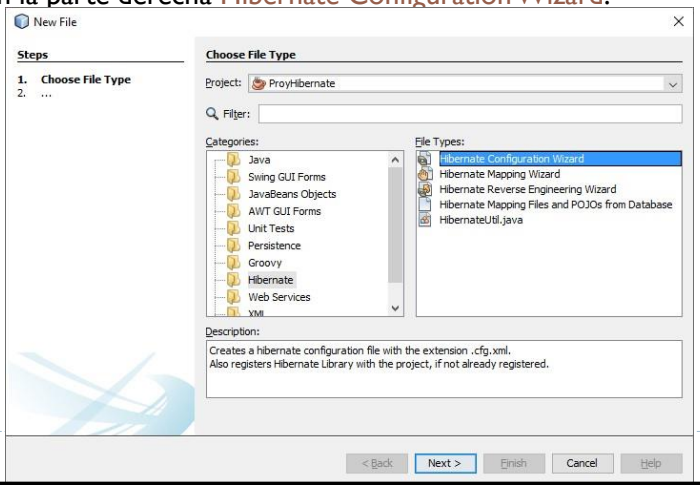


Creación de proyecto Hibernate en Netbeans (II)



Fichero de configuración de Hibernate (I)

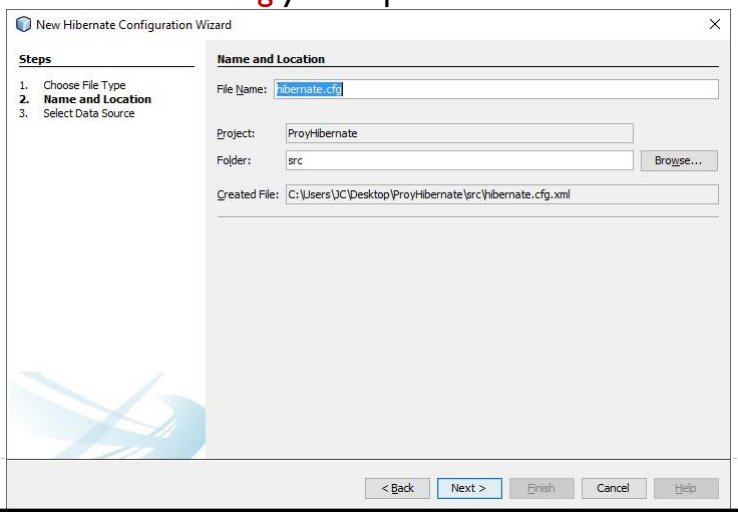
- Para crear el fichero de configuración se hace clic derecho sobre el proyecto, y se elige New->Other…, y en la nueva ventana que aparece, en la parte izquierda como Categories se elige Hibernate, y en la parte derecha Hibernate Configuration Wizard:



13

Fichero de configuración de Hibernate (II)

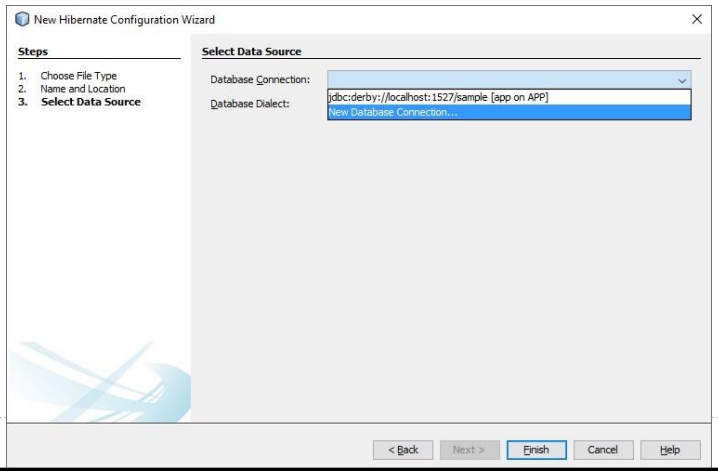
- Podemos dejar el nombre del fichero que nos sugiere por defecto hibernate.cfg y la carpeta src:



14

Fichero de configuración de Hibernate (III)

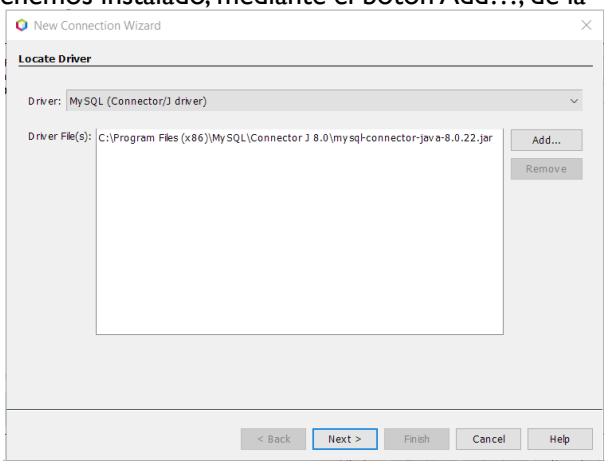
- Ahora debemos conectar con la base de datos, en este caso con la base de datos ejemplo de MySQL. Para ello en Database Connection, elegimos New Database Connection:



15

Fichero de configuración de Hibernate (IV)

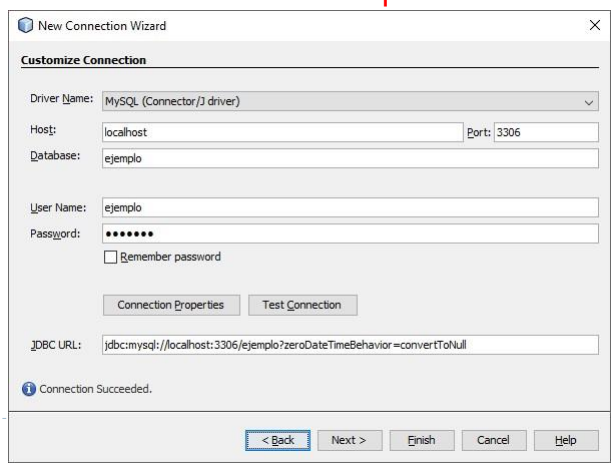
- Entre los *drivers* que nos aparecen estará el de MySQL (Connector/J driver), que será el que elegiremos, pero tenemos que especificar la ruta donde lo tenemos instalado, mediante el botón Add…, de la parte derecha:



16

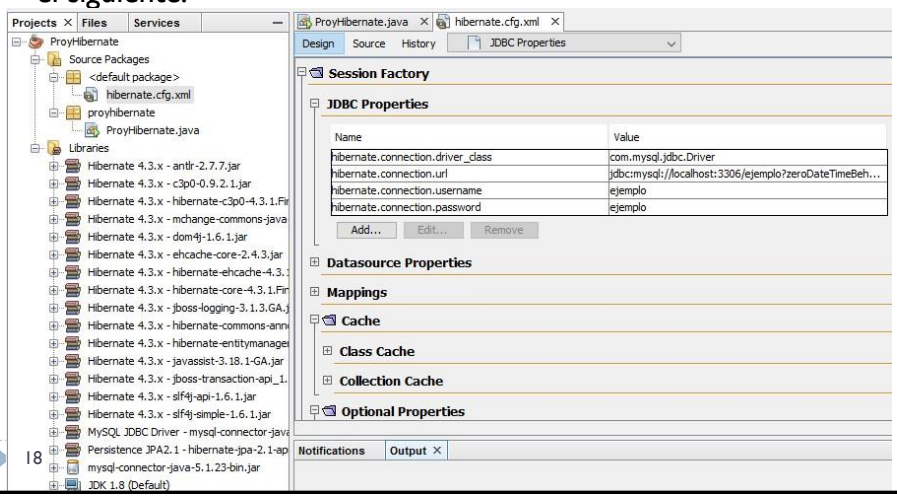
Fichero de configuración de Hibernate (V)

- A continuación tendremos que configurar los datos de la máquina a la que nos vamos a conectar y los de la base de datos, en nuestro caso será en nuestro equipo y la base de datos ejemplo, deberemos verificar la conexión hasta que el resultado sea favorable:



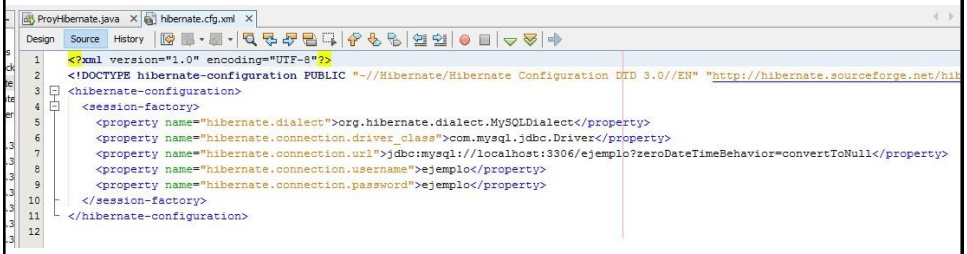
Fichero de configuración de Hibernate (VI)

- Finalizamos aceptando las pantallas siguientes y se nos generará el fichero de configuración. El aspecto de nuestro proyecto será el siguiente:



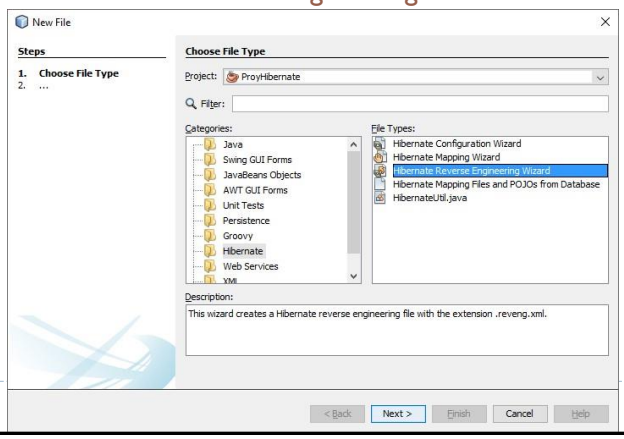
Fichero de configuración de Hibernate (VII)

- Si cambiamos de la vista *Design* a la vista *Source*, su aspecto será este:



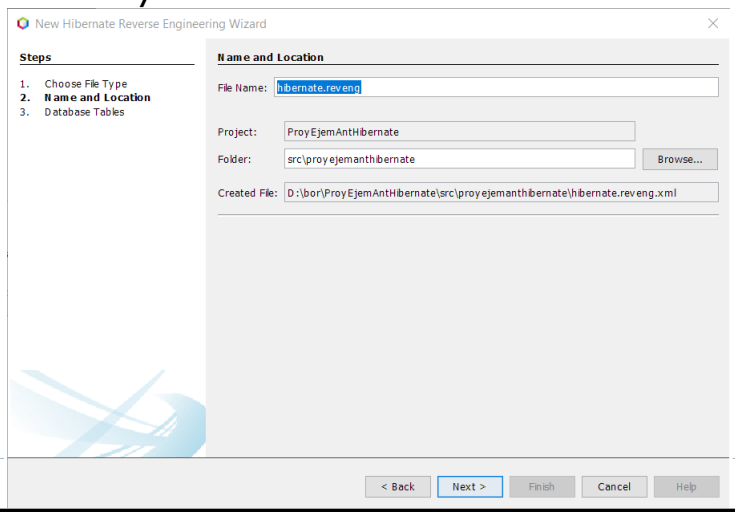
Fichero de Ingeniería inversa (I)

- Para crear el fichero XML Hibernate Reverse Engineering (.reveng.xml), que es el encargado de crear las clases de las tablas MySQL, hacemos clic de nuevo con el botón derecho sobre la carpeta del proyecto y New->Other… Sobre la nueva ventana que aparece, ahora elegimos Hibernate Reverse Engineering Wizard:



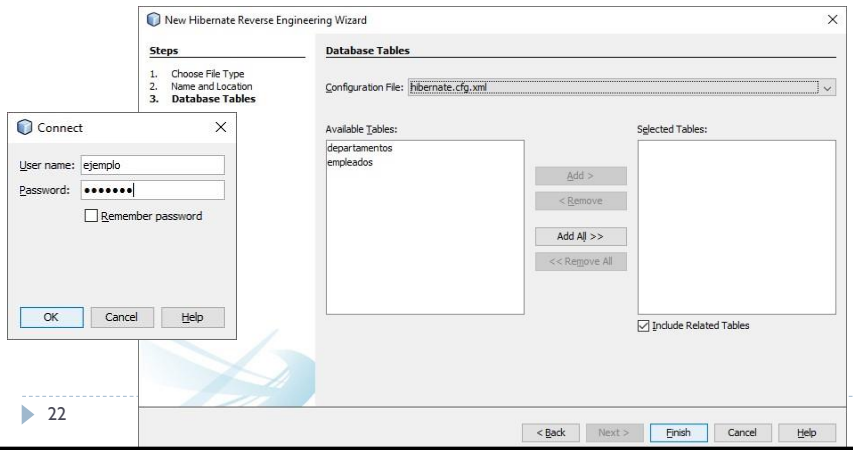
Fichero de Ingeniería inversa (II)

- En la siguiente ventana dejamos los nombres propuestos por defecto y la localización:



Fichero de Ingeniería inversa (III)

- Al pulsar en el botón siguiente intenta conectarse a la base de datos, si no lo ha hecho antes, y si no hemos guardado la contraseña nos la solicitará. En la ventana de tablas disponibles, deben aparecer departamentos y empleados, que serán las que añadiremos:



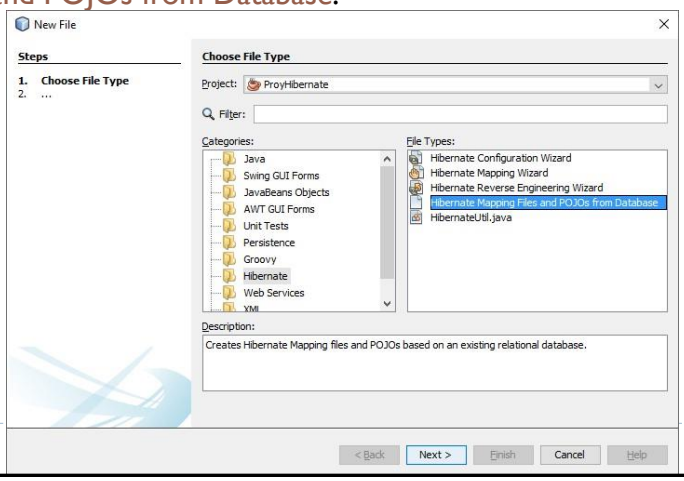
Fichero de Ingeniería inversa (IV)

- Al final nos aparece el fichero XML creado:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-reverse-engineering PUBLIC "-//Hibernate/Hibernate Reverse
Engineering DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-reverse-engineering-
3.0.dtd">
<hibernate-reverse-engineering>
  <schema-selection match-catalog="ejemplo"/>
  <table-filter match-name="empleados"/>
  <table-filter match-name="departamentos"/>
</hibernate-reverse-engineering>
```

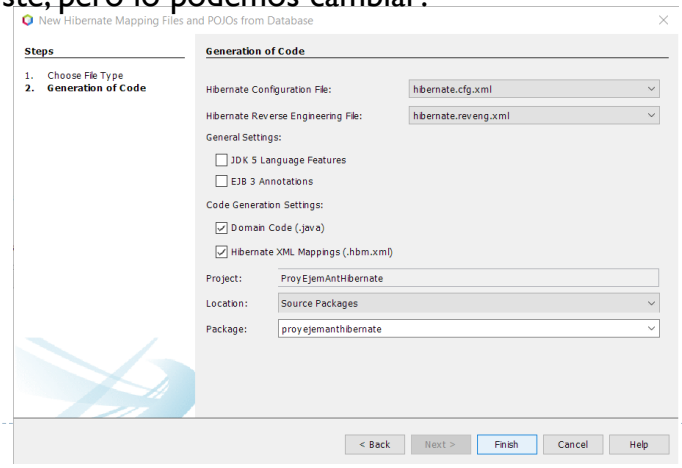

Generación de las clases de la base de datos (I)

- Para crear los archivos de mapeo y POJOs de las tablas, hacemos clic derecho en la carpeta de código del proyecto y elegimos New->Other … para escoger Hibernate Mapping Files and POJOs from Database:



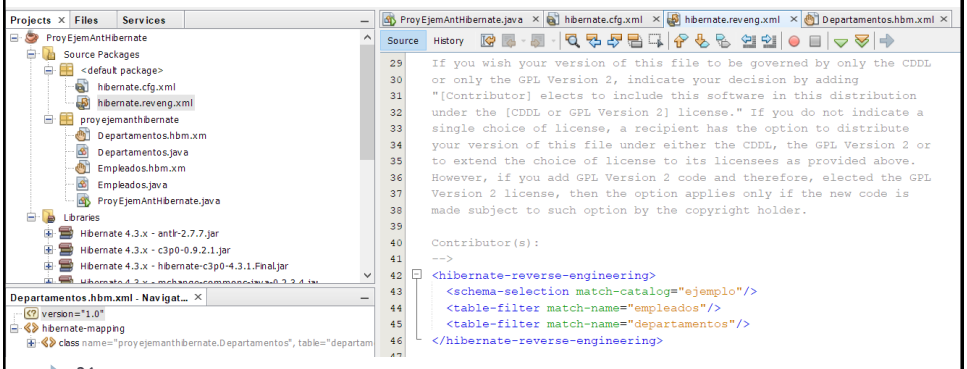
Generación de las clases de la base de datos (II)

- Nos permite especificar un nombre de paquete para colocar las clases que va a generar. En principio nos da el que existe, pero lo podemos cambiar:



Generación de las clases de la base de datos (III)

- Nos habrá creado el paquete y dentro tantas clases java como tablas hayamos seleccionado de la base de datos, y por cada clase un fichero XML.



Generación de las clases de la base de datos (IV)

- El fichero XML para departamentos es:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="primero.Departamentos" table="departamentos" catalog="ejemplo"
optimistic-lock="version">
      <id name="deptNo" type="int">
         <column name="dept_no" />
         <generator class="assigned" />
      </id>
      <property name="dnombre" type="string">
         <column name="dnombre" length="15" />
      </property>
      <property name="loc" type="string">
         <column name="loc" length="15" />
      </property>
   </class>
</hibernate-mapping>
```


Generación de las clases de la base de datos (V)

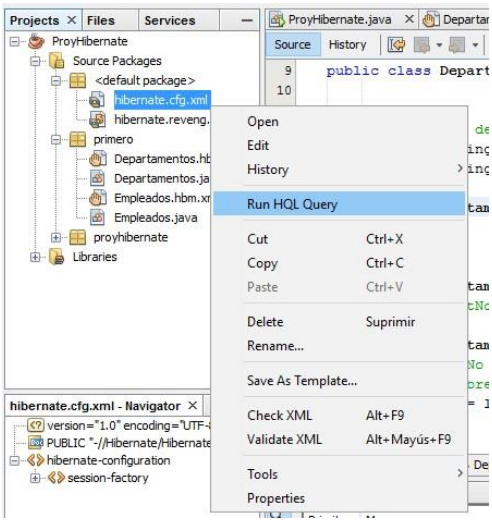
- La clase Java para departamentos contiene:

```
package proyejemanthibernate;
public class Departamentos  implements java.io.Serializable {
    private byte deptNo;
    private String dnombre;
    private String loc;
    private Set empleadoses = new HashSet(0);

   public Departamentos() {
   }
   public Departamentos(int deptNo) {
      this.deptNo = deptNo;
   }
   public Departamentos(byte deptNo, String dnombre, String loc) {
      this.deptNo = deptNo;
      this.dnombre = dnombre;
      this.loc = loc;
   }
   public Departamentos(byte deptNo, String dnombre, String loc, Set
empleadoses) {
      this.deptNo = deptNo;
      this.dnombre = dnombre;
      this.loc = loc;
      this.empleadoses = empleadoses;
   }
   public byte getDeptNo() {
      return this.deptNo;
   }
   public void setDeptNo(byte deptNo) {
      this.deptNo = deptNo;
   }
   public String getDnombre() {
      return this.dnombre;
   }
   public void setDnombre(String dnombre) {
      this.dnombre = dnombre;
   }
   public String getLoc() {
      return this.loc;
   }
   public void setLoc(String loc) {
      this.loc = loc;
   }
   public Set getEmpleadoses() {
      return this.empleadoses;
   }
   public void setEmpleadoses(Set empleadoses)
{
      this.empleadoses = empleadoses;
   }
}
```

Consultas con Hibernate (I)

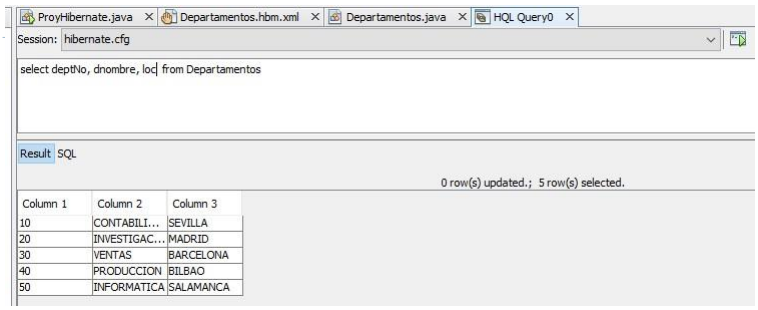
- Se puede seguir realizando consultas en **SQL**, o bien se pueden realizar en **HQL** (*Hibernate Query Language*).
- Para ejecutar una consulta se hace clic con el botón derecho sobre el fichero de configuración en el árbol del proyecto y se elige la opción Run HQL Query



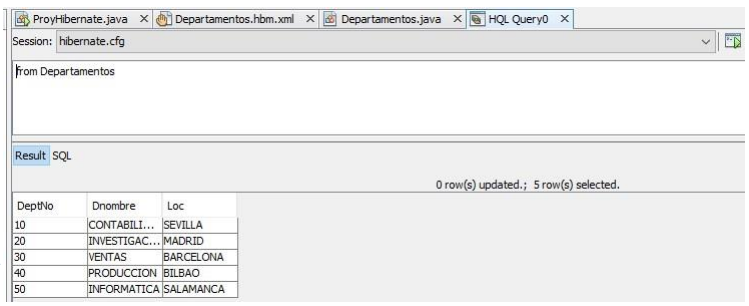
29

Consultas con Hibernate (II)

Consulta en SQL:



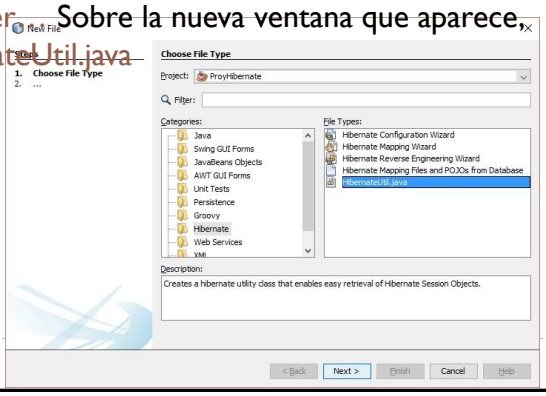
Consulta en HQL:



30

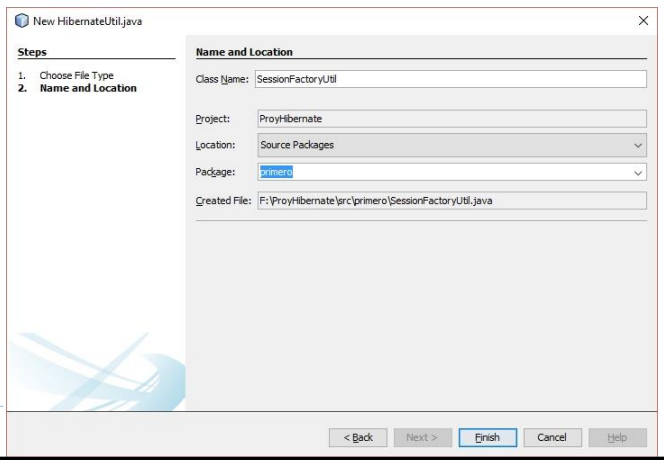
Programación en Hibernate (I)

- Una vez configurada la conexión y creadas nuestras clases, vamos a crear una aplicación Java que se conecte con la base de datos.
- Para la conexión se puede crear la clase directamente o se puede hacer generando el fichero con las mismas opciones utilizadas para los ficheros de configuración. Botón derecho sobre la carpeta del proyecto y New->Other... Sobre la nueva ventana que aparece, ahora elegimos HibernateUtil.java



Programación en Hibernate (II)

- Le podemos cambiar el nombre, le ponemos SessionFactoryUtil, y le decimos el paquete donde guardarlo que debería ser el mismo en el que están las clases.



Programación en Hibernate (III)

- Nos generará un código similar a este:

```
package primero;

import org.hibernate.cfg.AnnotationConfiguration;
import org.hibernate.SessionFactory;

public class SessionFactoryUtil {
    private static final SessionFactory sessionFactory;

    static {
        try {
            // Create the SessionFactory from standard (hibernate.cfg.xml)
            // config file.
            sessionFactory = new AnnotationConfiguration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            // Log the exception.
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    } 33
}
```

Programación en Hibernate (IV)

- Para crear una instancia de la base de datos se utiliza el patrón ***Singleton***, un patrón de diseño que restringe la creación de objetos y garantiza que sólo se puede crear una instancia de la clase y proporciona un punto de acceso global a ella.
- Esto se consigue creando en la clase un método que crea una instancia del objeto si todavía no existe. Para asegurar que la clase no puede ser instanciada nuevamente se hace que el constructor sea protegido o privado.
- Se recomienda sustituir el código generado anteriormente por este otro:

Programación en Hibernate (V)

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

public class SessionFactoryUtil {
    private static SessionFactory sessionFactory;
    private static ServiceRegistry serviceRegistry;

    private static SessionFactory configureSessionFactory() throws HibernateException {
        Configuration configuration = new Configuration();
        configuration.configure();
        serviceRegistry = new StandardServiceRegistryBuilder().applySettings(
                configuration.getProperties()).build();
        sessionFactory = configuration.buildSessionFactory(serviceRegistry);
        return sessionFactory;
    }
    public static SessionFactory getSessionFactory() {
        return configureSessionFactory();
    }
}
```


Programación en Hibernate (VI)

- El *singleton* se ha implementado como una clase de ayuda que accede a SessionFactory para obtener un objeto de tipo sesión sessionFactory.
- Antes de crear el objeto hay que tener en cuenta los datos del fichero de configuración por lo que se llama al método configure(), que se encarga de cargar el fichero hibernate.cfg.xml
- A continuación se construye la SessionFactory que va a ser la factoría que nos va a permitir crear las sesiones que necesitemos.

Programación en Hibernate (VII)

- Ahora creamos una clase main, que es la que vamos a ejecutar.
- Esta clase realizará las siguientes acciones:
  - Creará una sesión.
  - Iniciará una transacción.
  - Insertará una fila en la tabla departamentos, pidiendo los datos al usuario.
  - Terminará la transacción.
  - Terminará la sesión.

Programación en Hibernate (VIII)

```
public class ProyHibernate {
    public static void main(String[] args) {
            SessionFactory sesion = SessionFactoryUtil.getSessionFactory();
            Session session = sesion.openSession();
            Transaction tx = session.beginTransaction();
            System.out.println("Inserción de una fila en la tabla Departamentos.");
            Scanner scr = new Scanner(System.in);
            Departamentos depart = new Departamentos();
            System.out.println("Número departamento: ");
            depart.setDeptNo(Integer.parseInt(scr.nextLine()));
            System.out.println("Nombre departamento: ");
            depart.setDnombre(scr.nextLine());
            System.out.println("Localización: ");
            depart.setLoc(scr.nextLine());//Inserción de un departamento

            session.save(depart);
            tx.commit();
            session.close();
            }
    }
```

Inserción de un departamento

Programación en Hibernate (IX)

```
public class ProyHibernate {
    public static void main(String[] args) {
            SessionFactory sesion = SessionFactoryUtil.getSessionFactory();
            Session session = sesion.openSession();
            Transaction tx = session.beginTransaction();
            System.out.println("Inserción de un empleado en el departamento 10");
            Empleados em= new Empleados();
            em.SetEmpNo((short) 1234);
            em.SetApellido("Alonso");
            em.setDir((short) 4567);
            em.setOficio("Programador");
            em.setSalario(2500);
            em.setComision(15);
            Departamentos depart = new Departamentos();
            depart.setDeptNo((byte) 10);
            em.setDepartamentos(depart);
            java.util.Date hoy= new java.util.Date();
            java.sql.Date fecha= new java.sql.Date(hoy.getTime());
            em.setFechaAlt(fecha);
            session.save(em);
            tx.commit();
            session.close();
            }  }
```

Inserción de un empleado

Se producirá excepción si existen valores de clave duplicados, o no existe el departamento

Programación en Hibernate (X)

- Los posibles estados de los objetos Hibernate son:
  - **Transitorio**. Se ha instanciado con new y no está asociado a una *Session* de Hibernate. Aún no tiene una representación persistente en la base de datos y no tiene un valor identificador. Se puede pasar de transitorio a persistente asociándolo con una sesión (save()):

```
Departamentos dep= new Departamentos();
dep.setDepNo ((byte) 60);
dep.setDnombre("VENTAS");
dep.setLoc("345D");
session.save(dep);      //En este momento se hace persistente
```

  - **Persistente**. Tiene una representación en la base de datos y un valor identificador. Se encuentra en el ámbito de una *Session*. Hibernate detecta los cambios que se producen sobre el objeto y los sincroniza cuando se complete la unidad de trabajo.
  - **Separado**. Es un objeto que se ha hecho persistente, pero su *Session* se ha cerrado. La referencia al objeto es válida, por lo que el objeto puede ser modificado e incluso puede ser reunido a una nueva *Session*.

Programación en Hibernate (XI)

- La recuperación de un objeto persistente conociendo su clave se hace mediante el método load().
- Es necesario haber instanciado un objeto de la clase para poder recuperarlo sobre él, de esta forma:

```
Departamentos dep= new Departamentos();
dep= (Departamentos) session.load(Departamentos.class, (byte)20);
System.out.println("Depto.:"+dep.getDnombre());
System.out.println("Localización:"+dep.getLoc());
```

- Se produce excepción si no existe el valor de la clave. Por lo que se puede utilizar el método get(), que permite la consulta, pero sin producir excepción:

```
dep= (Departamentos) session.get(Departamentos.class, (byte)20);
if (dep==null)
    System.out.println("El departamento no existe");
else {
    System.out.println("Depto.:"+dep.getDnombre());
    System.out.println("Localización:"+dep.getLoc());
}
```


Programación en Hibernate (XII)

```
public static void emplDepa(SessionFactory sesion) {
        Session session = sesion.openSession();
        Transaction tx = session.beginTransaction();
        System.out.println("Empleados de un Departamento.");
        Departamentos dep = new Departamentos();
        Scanner scr = new Scanner(System.in, System.getProperty("os.name").contains("Windows") ? "iso-8859-1" : "UTF-8");
        System.out.println("Número departamento: ");
        byte depto = Byte.parseByte(scr.nextLine());
        dep = (Departamentos) session.get(Departamentos.class, depto);
        if (dep == null) {
                System.out.println("Ese departamento NO existe...");
        } else {
                Set<Empleados> listaempleados = dep.getEmpleadoses();
                Iterator<Empleados> it = listaempleados.iterator();
                while (it.hasNext()) {
                        Empleados empl = new Empleados();
                        empl = it.next();
                        System.out.println(empl);
                }
        }
        tx.commit();
        session.close();
```

Empleados de un departamento

Programación en Hibernate (XIII)

- **Inserción**. Se utiliza el método save(), ya visto. Hibernate se encarga de hacer un insert SQL en la base de datos.
- **Borrado**. Para realizar un borrado de un objeto primero tiene que ser cargado con load() para después borrarlo con el método delete(), asegurándonos de que no haya referencias de objetos. Ejemplo:

```
Empleados em= new Empleados();
em= (Empleados) sesión.load(Empleados.class, (short)1234);
session.delete(em);
```

- **Modificación**. Hay que cargarlo primero con load(), se modifican los valores con los métodos setter y se actualiza con update(). Ejemplo:

```
Empleados em= new Empleados();
em= (Empleados) sesión.load(Empleados.class, (short)1246);
System.out.println("Salario antiguo:"+ em.getSalario());
System.out.println("Comisión antigua:"+ em.getComision());
em.setSalario((float) 2000);
em.setComision((float) 30);
session.update(em);
```


Programación en Hibernate (XIV)

- **Consultas**.
- Las consultas se representan con una instancia de la interfaz org.hibernate.Query. Se utiliza el método createQuery("consulta"). Ejemplo:

```
Query q= session.createQuery("from Departamentos");
```

- Para recuperar los datos de la consulta se pueden utilizar los métodos uniqueResult(), list(), e iterate(). Ejemplos:

```
Departamentos dep =(Departamentos) q.uniqueResult();
…
List <Departamentos> lista= q.list();
…
Iterator iter= q.iterate();
```

Programación en Hibernate (XV)

- **Consultas**.
- El método uniqueResult() se utiliza cuando se sabe que el resultado sólo va a devolver un objeto.
- El método list() devuelve una colección con todos los resultados de la consulta en una única comunicación con la base de datos, que puede prolongarse si los datos son numerosos.
- El método iterate() devuelve un iterador Java, que permitirá ir recuperando datos de la base mediante iterator.next(), lo que supone mayor número de accesos pero menor necesidad de memoria para guardar los datos.
- Se dispone del método setFetchSize() para fijar la cantidad de resultados que se recuperan de la base de datos a la vez.

45

Programación en Hibernate (XVI)

- **Ejemplo1:**

```
Departamentos depar= new Departamentos();
Query q= session.createQuery("from Departamentos");
List <Departamentos> lista= q.list();
Iterator <Departamentos> iter= lista.iterator();
While (iter.hasNext()) {
        depar= (Departamentos) iter.next();
        System.out.println(depar);}
```

- **Ejemplo 2:**

```
Departamentos depar= new Departamentos();
Query q= session.createQuery("from Departamentos");
q.setFechSize(10);
Iterator iter= q.iterate();
While (iter.hasNext()) {
        depar= (Departamentos) iter.next();
        System.out.println(depar);}
```

46

Programación en Hibernate (XVII)

- **Parámetros en las consultas**.
- Se pueden enlazar valores a los parámetros con nombres, que son identificados mediante la notación :nombre, o se pueden enlazar valores a los parámetros de tipo ? de JDBC.
- Ejemplo 1:

```
Query q= session.createQuery("from Empleados emp where emp.deptNo= :ndep and emp.oficio= :ofi");
q.setInteger("ndep", 14);
q.setString("ofi","Administrador");
```

- Ejemplo 2:

```
Query q= session.createQuery("from Empleados emp where emp.deptNo= ? and emp.oficio= ?);
q.setInteger(0, 14);
q.setString(1,"Administrador");
```

47

Programación en Hibernate (XVIII)

- Consultas sobre clases no asociadas
  - Se utilizan cuando en la consulta intervienen varias tablas y los atributos que se devuelven no están asociados a ninguna clase. En estos casos se utiliza la clase Object para recogerlos. Un array de Objects donde el primer elemento del array se corresponde con la primera clase que va a la derecha del FROM, el segundo con la segunda, y así sucesivamente. Supongamos que empleados y departamentos no estuvieran asociada:

```
Query cons= session.createQuery("from Empleados e, Departamentos d where e.deptNo=d.deptNo order by e.apellido");
Iterator q= cons.iterate();
while (q.hasNext()){
    Object[] res= (Object[]) q.next();
    Empleados em= (Empleados) res[0];
    Departamentos de= (Departamentos) res[1];
    System.out.println(em.getApellido() + " - " + em.getSalario()+" - " + de.getDnombre() + " - " + de.getLoc());
    }
```

48

Programación en Hibernate (XIX)

- Consultas con funciones de grupo
  - Las funciones que realizan un cálculo y devuelven un único valor numérico, como pueden ser avg(), sum(), count, etc. pueden implementarse utilizando el método uniqueResult(). Ejemplo, cálculo del salario medio de los empleados:

```
Query cons= session.createQuery("select avg(em.salario) from Empleados as em");
Double suma= (Double) cons.uniqueResult();
System.out.println("Salario medio:" + suma);
```

  - En el caso de que intervengan varias funciones de grupo y devuelvan varias filas, se pueden utilizar objetos devueltos por las consultas.

Programación en Hibernate (XX)

- Consultas que devuelven objetos de clases no mapeadas
  - Si los resultados de una consulta no se corresponden con los objetos de las clases mapeadas, podemos añadir una nueva clase de forma que sus objetos se correspondan con el resultado de la consulta. Por ejemplo, supongamos que queremos obtener para cada Departamento su número y su nombre, el número de empleados y el salario medio de los Empleados. Construiremos una nueva clase con esos atributos y sus métodos getter y setter, que llamaremos totales

Programación en Hibernate (XXI)

- Consultas que devuelven objetos de clases no mapeadas

```
package primero;
public class Totales {
    private Object cuenta;
    private Object numero;
    private Object media;
    private Object nombre;
    public Totales(final Object numero, final Object cuenta, final Double media, final String
    nombre){
        this.cuenta= cuenta;
        this.media= media;
        this.nombre= nombre;
        this.numero= numero;
    }
    public Totales(){}

    public Object getCuenta() {return this.cuenta;}
    public void setCuenta(final Object cuenta){this.cuenta= cuenta;}
    …
}
```


Programación en Hibernate (XXII)

- Consultas que devuelven objetos de clases no mapeadas
  - El código de la consulta sería:

```
Query cons= session.createQuery("select new primero.Totales("
+ "de.deptNo, count(em.empNo)," +
"coalesce(avg(em.salario),0), de.dnombre )" +
"from Departamentos as de, Empleados as em" +
"where de.deptNo=em.deptNo"+
"group by de.deptNo, de.dnombre" );
Iterator q= cons.iterate();
while (q.hasNext()) {
        Totales tot= (Totales) q.next();
        System.out.println("Numero Dep: "+tot.getNumero()+"
Nombre:"+tot.getNombre()+" Salario medio:"+tot.getMedia()+"
N° empl: "+ tot.getCuenta());
}
```

Programación en Hibernate (XXIII)

- Consultas que devuelven objetos de clases no mapeadas
  - La misma consulta utilizando directamente Object:

```
Query cons= session.createQuery("select de.deptNo as dep,
count(em.empNo) as cuenta," +
"coalesce(avg(em.salario),0) as media, de.dnombre as nombre " +
"from Departamentos as de, Empleados as em" +
"where de.deptNo=em.deptNo"+ "group by de.deptNo,
de.dnombre" );
List<Object[]> filas= cons.list();
for (int i=0; i<filas.size(); i++) {
        Object[] filaActual= filas.get(i);
        System.out.println("Numero Dep: "+filaActual[0]+"
Nombre:"+filaActual[3]+" Salario medio:"+filaActual[2]+" Nº
empl: "+ filaActual[1]);
}
```


Programación en Hibernate (XXIV)

- **Update y Delete en HQL**
  - La sintaxis es: (UPDATE | DELETE) [FROM] NombreEntidad [WHERE condición]
  - donde los elementos entre corchetes son opcionales, sólo puede haber una entidad mencionada en la cláusual FROM y puede tener un alias, no se puede especificar asociaciones en una consulta masiva de HQL, aunque se puede utilizar subconsultas con WHERE, donde sí puede haber asociaciones.
  - Para ejecutar UPDATE o DELETE se utiliza el método **Query.executeUpdate()**, que devuelve un valor entero con el número de filas afectadas. Es necesario hacer commit; Ejemplo, modificación del salario de Alonso:

```
String hqlModif= "update Empleados e set e.salario= :nuevoSal where
e.apellido= :ape";
int filasModif= session.createQuery(hqlModif)
.setDouble("nuevoSal", 2455.60)
.setString("ape", "Alonso")
.executeUpdate();
System.out.println("Filas modificadas: "+filasModif);
tx.commit;
```

Programación en Hibernate (XXV)

- **Update y Delete en HQL**
  - Ejemplo, borrado de los empleados del departamento 20:

```
String hqlDel= "delete Empleados e where e.deptNo= ?";
int filasDel= session.createQuery(hqlDel)
.setInteger(0, 20)
.executeUpdate();
System.out.println("Filas borradas: "+filasDel);
tx.commit();
}
```

55

Programación en Hibernate (XXVI)

- **Insert en HQL**
  - La sintaxis es: INSERT INTO NombreEntidad (lista_propiedades) sentencia_select
  - Sólo pueden insertarse datos procedentes de otra tabla.
  - Ejemplo, inserción de los datos de otra tabla que tiene que estar mapeada:

```
String hqlInsert= "insert into Departamentos (deptNo, dnombre, loc) select n.deptNo, n.dnombre, n.loc from Nuevos n";
int filasIns= session.createQuery(hqlInsert)
.executeUpdate();
System.out.println("Filas insertadas: "+filasIns);
tx.commit;
```

56