1. Instalación del plugin Hibernate.

Se recomienda la instalación del plugin sobre una versión Netbeans 8.2 ó Netbeans 11.3. También es posible realizar la instalación sobre otra versión de Netbeans, pero puede ser necesario realizar algún paso adicional, como sucede en la versión Netbeans 12.

Problema instalación Hibernate sobre Netbenas 12 => https://stackoverflow.com/questions/62464233/netbeans-12-0-lts-hibernate-plugin-require-freemarker-integration-implementati

You can fix this by

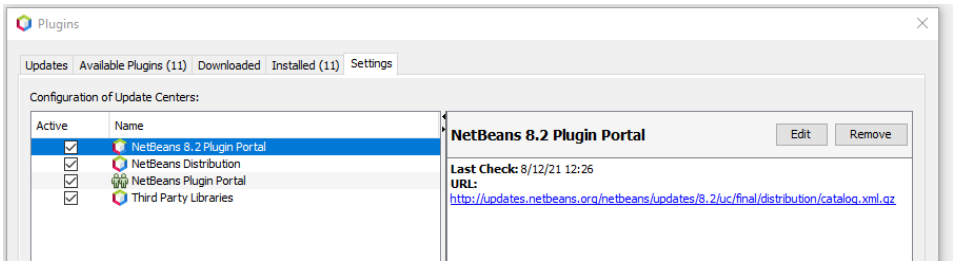
1. Close your running netbeans application
2. Download freemarker.jar with OpenIDE-Module-Implementation-Version: 238 org-netbeans-libs-freemarker-RELEASE81.jar and rename the downloaded library to `org-netbeans-libs-freemarker.jar`
3. Navigate to ide/modules subfolder in apache netbeans installation directory
4. You will find the freemarker library `org-netbeans-libs-freemarker.jar`, replace this with your downloaded library.
5. Restart the netbeans(or perhaps restarting the computer should work) and you should be able to download the hibernate plugin

ps- you can check the OpenIDE-Module-Implementation-Version by opening the jar library as an archive and inside the META-INF/MANIFEST.MF file

Also refer this for more information

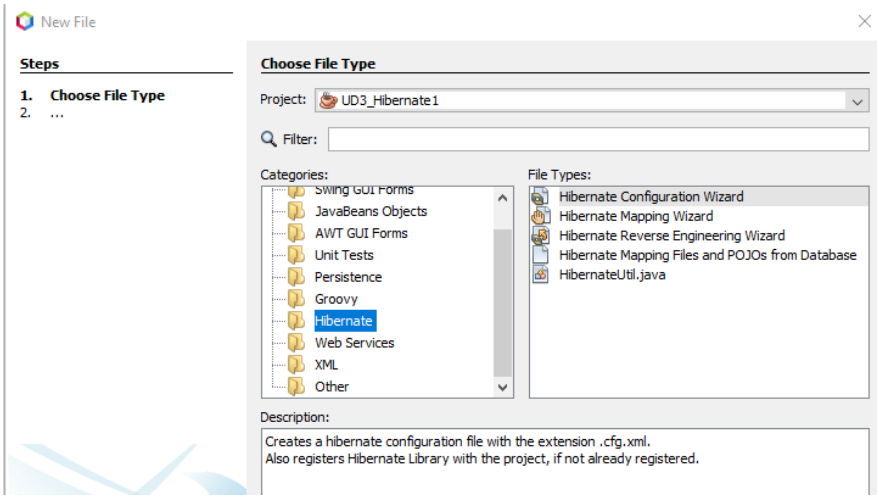
Instalación en Netbeans 11.3

Paso 1. Ir al menú Tools > Plugings y en la pestaña Settings selecciona casilla para que esté activo el centro de Netbenas 8.2 Plugin Portal, que por defecto viene desactivado



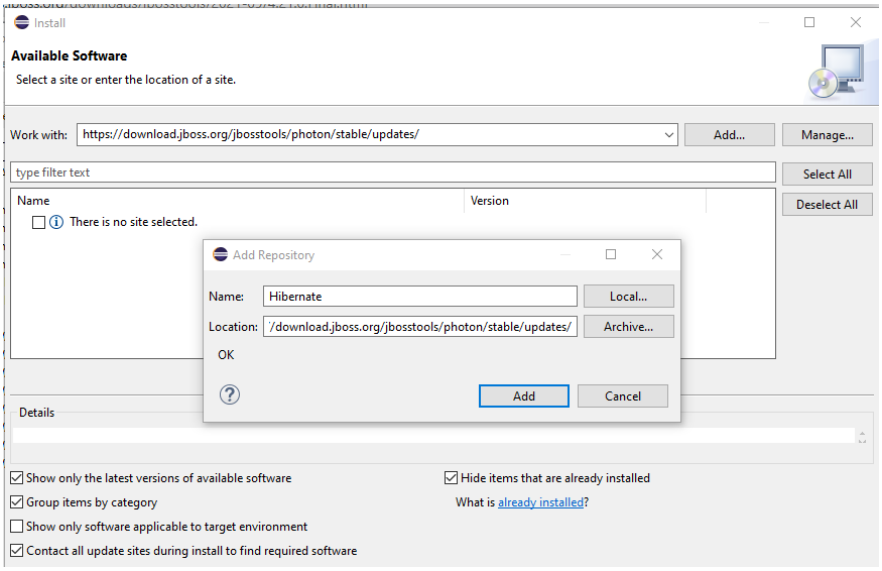
Paso 2. En la pestaña Available Plugins pulsamos el botón Check for newest y nos debe aparecer Hibernate 1.33.1.1. Lo seleccionamos y pulsamos Install. Al finalizar la instalación es necesario cerrar y avolver a abrir Netbeans.

Paso 3. Para comprobar que se ha realizado la instalación vamos al menú File > New File y entre las categorías de ficheros disponibles debe aparecer Hibernate y al seleccionarlo aparecen los distintos asistentes que nos ayudarán a crear los ficheros.

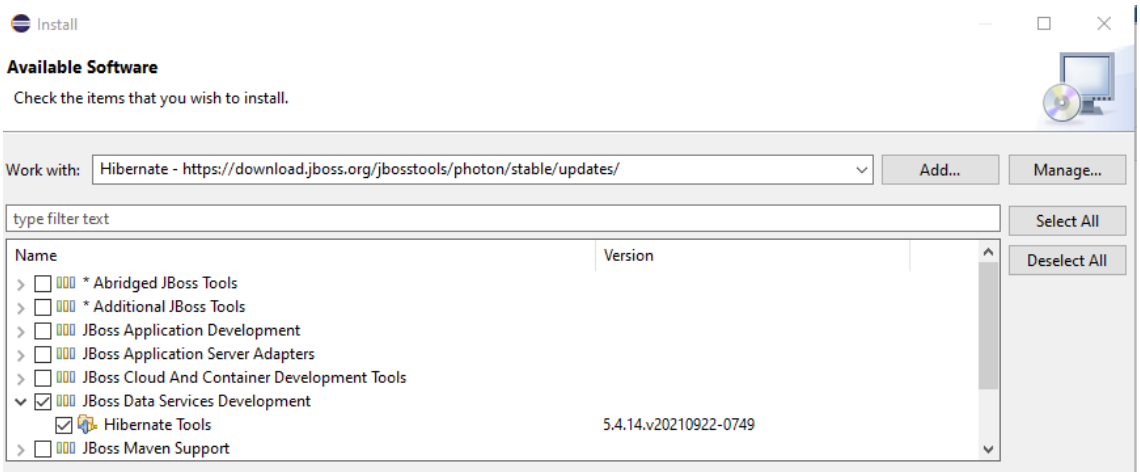


Instalación en Eclipse. También se puede realizar la instalación sobre otro IDE como Eclipse. En este caso los pasos serían:

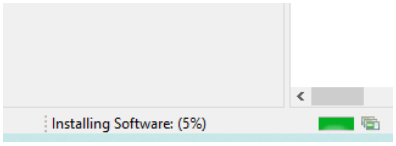
Paso 1. Ir al menú Help > Install new software. Poner la URL que aparece en la imagen en el campo Work With, pulsar Add e indicar Hibernate en la ventana Add Repository. La URL dependerá de la versión de Eclipse que se tiene instalada, que en este caso era la versión photon (junio 2021)



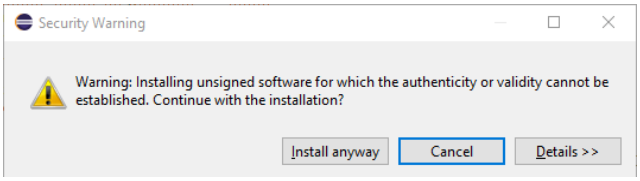
Paso 2. Seleccionar Hibernate Tools dentro de JBoss Data Services Development Tools

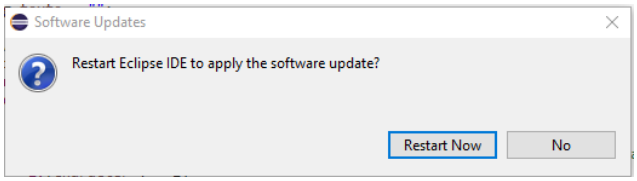


Paso 3. Pulsar Instalar y esperar a que termine. Para ver el avance de la instalación en la esquina inferior derecha de Eclipse está el indicador del avance de la instalación.

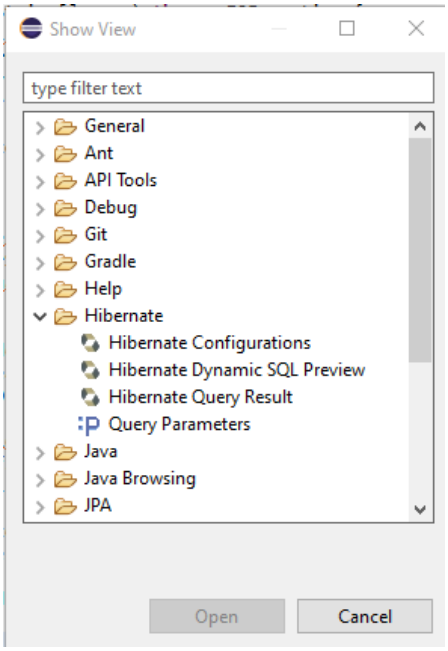


Cuando va por el 51% me avisa de que si quiero instalar software que no está firmado y le digo que instale de todos modos (Install anyway). Después me pide reiniciar Eclipse





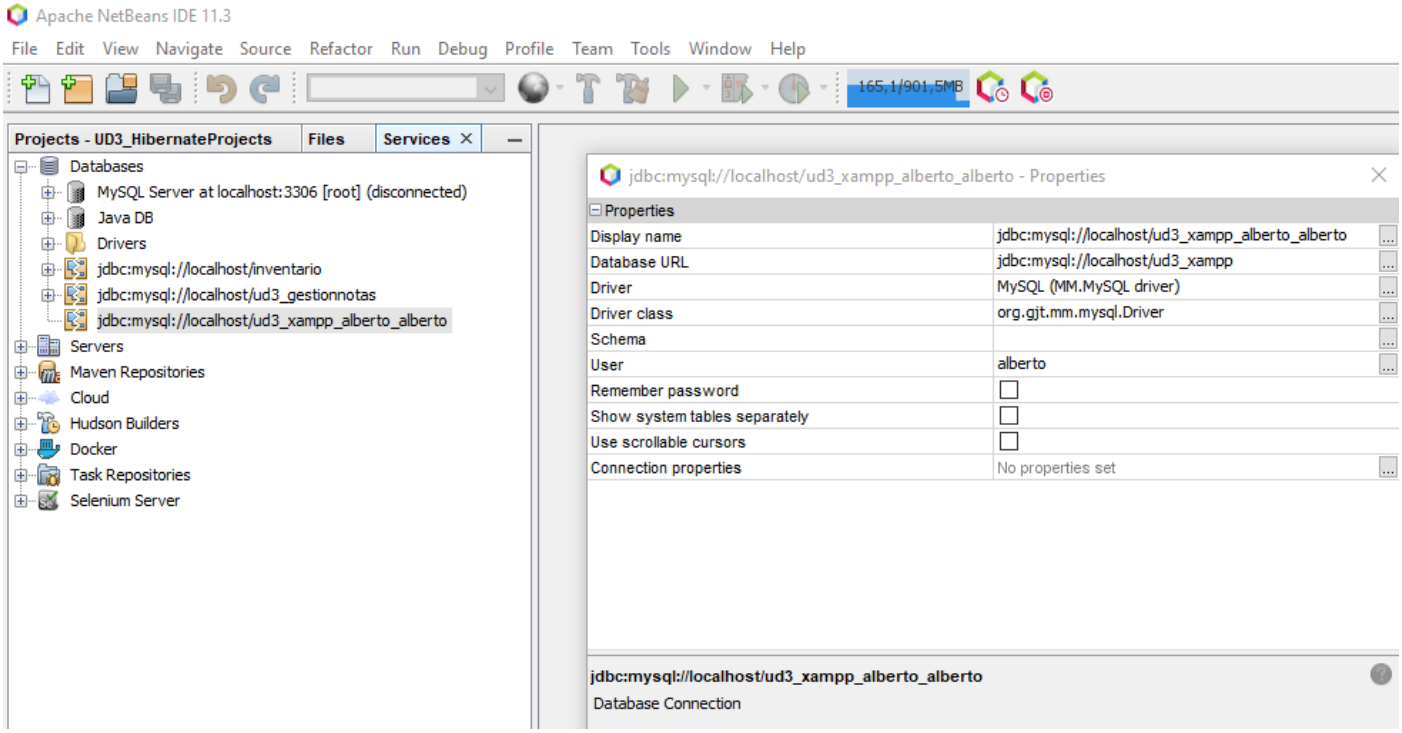
Paso 4. Para comprobar que se ha instalado voy al menú Windows > Show View > Other deben aparecer las opciones de Hibernate



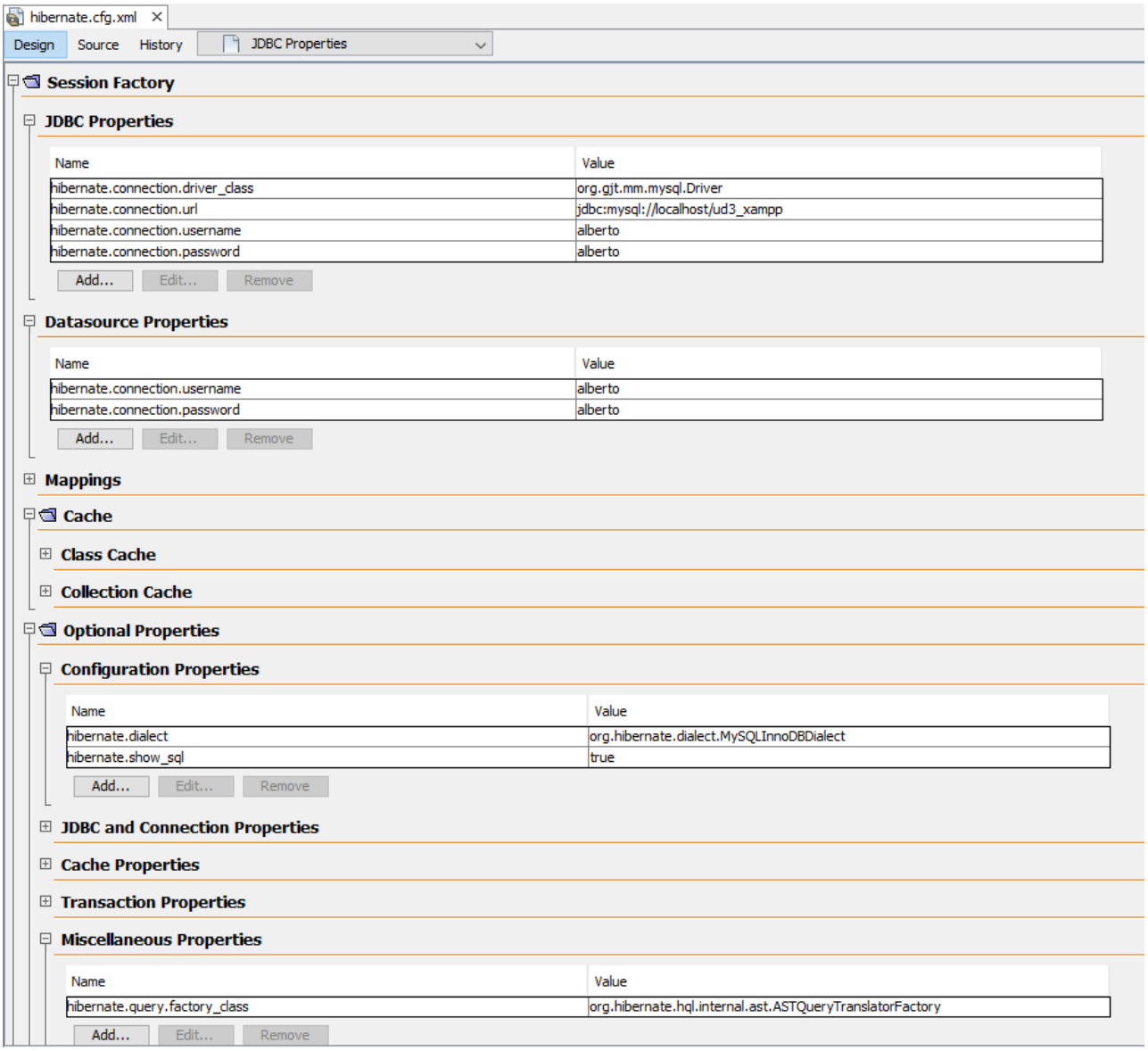
2. Crear un Proyecto con Hibernate.

Emplear Hibernate como herramienta para realizar las operaciones de acceso a los datos de una base de datos requiere de una configuración previa, que se lleva a cabo en varios pasos y queda almacenada en diferentes ficheros que forman parte del proyecto.

Paso 1. Establecer conexión con la BD. En Netbeans se realiza desde la pestaña Services. Se crea una nueva conexión con los parámetros adecuados a la BD e incorporando el driver jdbc correspondiente a la BD y versión de la misma.



Paso 2. Crear el fichero de configuración Hibernate.cfg.xml empleando el asistente Hibernate Configuration Wizard. El archivo se puede editar empleando la herramienta gráfica (pestaña Design).



Nota: La propiedad hibernate.show_sql = true hace que las sentencias sql que se ejecutan se visualicen en la consola

Los cambios efectuados se van registrando en el fichero y podemos ver su contenido al pulsar la pestaña Source.

```
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class">org.gjt.mm.mysql.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost/ud3_xampp</property>
    <property name="hibernate.connection.username">alberto</property>
    <property name="hibernate.connection.password">alberto</property>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLInnoDBDialect</property>
    <property name="hibernate.show_sql">true</property>
    <property name="hibernate.query.factory_class">org.hibernate.hql.internal.ast.ASTQueryTranslatorFactory</property>
    <mapping resource="ud3_hibernate1/Departamentos.hbm.xml"/>
    <mapping resource="ud3_hibernate1/Empleados.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Paso 3. Crear el fichero HibernateUtil que incluye el objeto SesionFactory que permite el acceso del resto de clases de la aplicación al objeto Session. empleando el asistente que directamente genera el fichero HibernateUtil.java. Una vez creado el fichero es conveniente actualizar las clases que se marcan como deprecated. El resultado final sería:

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

public class HibernateUtil {

    private static SessionFactory sessionFactory;

    public static SessionFactory getSessionFactory() {

        if (sessionFactory == null) {
            // loads configuration and mappings
            Configuration configuration = new Configuration().configure();
            ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder().applySettings(configuration.getProperties()).build();

            // builds a session factory from the service registry
            sessionFactory = configuration.buildSessionFactory(serviceRegistry);
        }
        return sessionFactory;
    }
}
```

Paso 4. Realizar la ingeniería inversa de la BD para determinar los objetos de la misma que queremos mapear creando el fichero hibernate.reveng.xml. Para ello se emplea el asistente Hibernate Reverse Engineering Wizard. El resultado de mapear las tablas 'departamentos' y 'empleados' de la base de datos 'ud3_xampp', sería el siguiente:

```xml
-->
<hibernate-reverse-engineering>
  <schema-selection match-catalog="ud3_xampp"/>
  <table-filter match-name="departamentos"/>
  <table-filter match-name="empleados"/>
</hibernate-reverse-engineering>
```

Paso 5. Crear los POJOs (clases que definen como objeto cada una de las tablas) y los archivos que mapean cada POJO con la tabla de la BD correspondiente. Para ello se emplea el asistente Hibernate Mapping Files and POJOs from Database. El asistente crea dos ficheros (NombreTabla.java y NombreTabla.hbm.xml) por cada una de las tablas que mapeamos en el paso anterior.

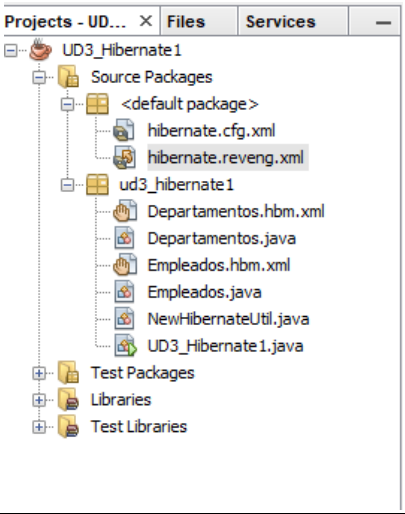
Conviene fijarse en que cuando hay claves foráneas en las tablas, en la tabla que está referenciada se crea un atributo o propiedad con una interfaz de tipo SET (conjunto) que es la que va a permitir devolver los datos en el caso de consultas que relacionen varias tablas. Ejemplo uso Interfaz Set =>
https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=610:interfaces-set-y-sortedset-del-api-java-clases-hashset-y-treeset-ordenado-ejemplo-diferencias-cu00924c&catid=58&Itemid=180

Por ejemplo, en nuestro caso, en la tabla Empleados el dept_no es clave foránea que referencia a la clave primaria de la tabla Departamentos y en los ficheros que crea Hibernate para la tabla Departamentos podemos ver esta definición.

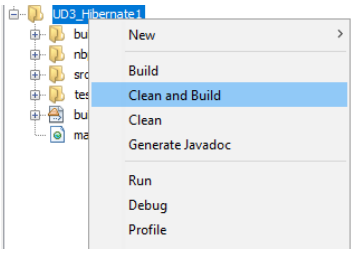
```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<!-- Generated 18-nov-2021 11:43:30 by Hibernate Tools 4.3.1 -->
<hibernate-mapping>
    <class name="ud3_hibernate1.Departamentos" table="departamentos" catalog="ud3_xampp" optimistic-lock="version">
        <id name="deptNo" type="byte">
            <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>
        <set name="empleadoses" table="empleados" inverse="true" lazy="true" fetch="select">
            <key>
                <column name="dept_no" not-null="true" />
            </key>
            <one-to-many class="ud3_hibernate1.Empleados" />
        </set>
    </class>
</hibernate-mapping>
```

```java
package ud3_hibernate1;
// Generated 18-nov-2021 11:43:29 by Hibernate Tools 4.3.1

import java.util.HashSet;
import java.util.Set;

/**
 * Departamentos generated by hbm2java
 */
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(byte deptNo) {
        this.deptNo = deptNo;
    }
    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;
    }
}
```

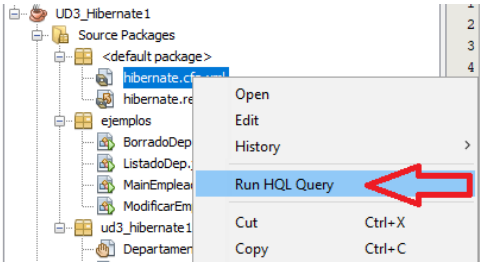
Tras realizar los diferentes pasos, tendremos una estructura de proyecto similar a la de la imagen

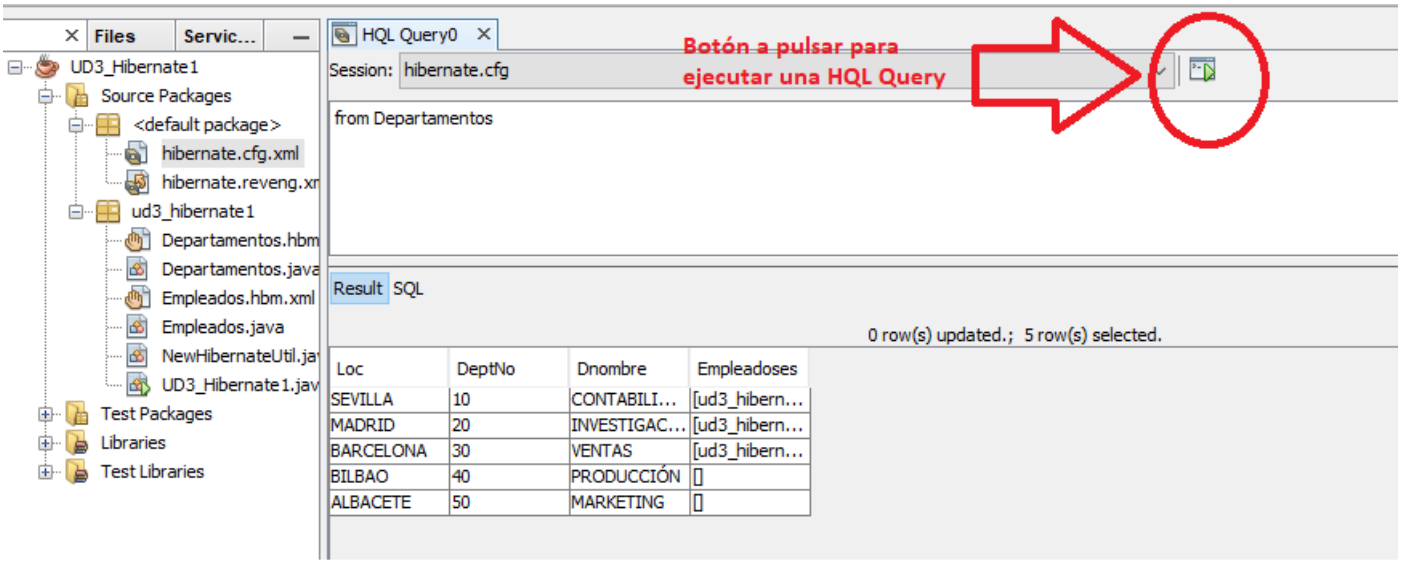


IMPORTANTE. Como hemos estado añadiendo clases, librerías, paquetes y otros archivos al proyecto es importante que en Netbeans selecciones el proyecto y la opción de menú Clean and Build. Entre otras cosas este proceso actualiza el archivo manifiesto del proyecto que dice donde está la clase main y el classpath a las diferentes librerías. De no hacerlo, puede que no funcione correctamente Hibernate al no reconocer las librerías o alguno de los ficheros creados.



Una vez configurado Hibernate y construido el proyecto (Clean and Build), ya podemos realizar consultas en la BD empleando el lenguaje de consultas específico de Hibernate (HQL). Para ello, nos posicionamos sobre el fichero hibernate.cfg.xml y con botón derecho se abre el menú para abrir una ventana de ejecución de consultas.





3. Resumen Lenguaje HQL.

Las consultas en HQL no son sensibles a mayúsculas a excepción de las clases y propiedades Java.

La cláusula más simple es *from* que obtienen todas las instancias de una clase, por ejemplo *from Empleados*

La cláusula *order by* ordena los resultados de la consulta y la cláusula *where* permite refinar las instancias que devuelve la consulta.

Ejemplos:

from Empleados where deptNo = 10 order by apellido
from Empleados as em where deptNo = 10 order by 1 desc
from Empleados where em.deptNo = 10

Podemos asignar alias a las clases usando la cláusula *as*, por ejemplo *from Empleados as em*, o sin usar dicha cláusula, es decir, poniendo *from Empleados em.*

Pueden aparecer varias clases a la derecha de *from*, y entonces se obtiene el producto cartesiano (cross join), como por ejemplo con *from Empleados as em, Departamentos as dep*

Para obtener el valor de determinados atributos o propiedades (columnas) usamos la cláusula select, como por ejemplo *select apellido, salario from Empleados* para obtener el apellido y salario de la clase Empleados

Las consultas pueden devolver múltiples objetos y/o atributos como un array de tipo Object[], una lista, una clase, etc.

Soporta las siguientes funciones de grupo, con una semántica similar a la de SQL:

- *avg(…), sum(…), min(…), max(…)*
- *count(*), count(…), count(distinct …), count(all …)*

Se puede utilizar alias para nombrar los atributos y expresiones, y también operadores aritméticos, de concatenación y funciones SQL reconocidas en la cláusula select.

Ejemplos de funciones de grupo:

select avg(salario) as med, count (empNo) as c from Empleados
select avg(salario), count (empNo) from Empleados

```
select avg(salario) + sum (salario), count(empNo) from Empleados
select apellido || '*' || oficio as campo from Empleados
select count(distinct deptNo) from Empleados
```

Ejemplos de expresiones en la cláusula where:

```
from Empleados where deptNo in (10,20)
from Empleados where deptNo not in (10,20)
from Empleados where salario between 2000 and 3000
from Empleados where salario not between 2000 and 3000
from Empleados where comisión is null
from Empleados where comisión is not null
select lower(apellido), coalesce(comisión, 0) from Empleados
select apellido from Empleados where apellido like 'A%'
```

Se pueden agrupar consultas usando *group by* y *having*. Las funciones SQL y las funciones de agregación están permitidas en las cláusulas *having* y *order by*, siempre que lo soporte la base de datos que se esté empleando. Las cláusulas *group by* y *order by* no pueden contener expresiones aritméticas.

Ejemplos:

```
select de.nombre, avg(em.salario) from Empleados em, Departamentos de
        where em.deptNo = de.deptNo
        group by de.nombre
        having avg(em.salario) > 2000
```

Cuando la base de datos permite subconsultas, Hibernate soporta subconsultas dentro de consultas. Una subconsulta se debe encerrar entre paréntesis. Incluso se permiten subconsultas correlacionadas, que son aquellas que se refieren a un alias en la consulta exterior.

Ejemplos:

```
from Empleados as em where em.salario >
        (select avg(em2.salario) from Empleados em2 where em2.deptNo = em.deptNo)

from Empleados as em where em.salario >
        (select avg(salario) from Empleados)
```

Asociaciones y uniones (joins)

Cuando se realiza el mapeo de tablas empleando el asistente de Hibernate las asociaciones de las claves ajenas que puedan tener las tablas se generan de forma automática, como podemos comprobar para la clase Empleado viendo el contenido del fichero Empleados.hbm.xml, en el que aparece la relación <many-to-one> y en el fichero Departamentos la relación < one-to-many>

```xml
<hibernate-mapping>
    <class name="ud3_hibernate1.Empleados" table="empleados" catalog="ud3_xampp" optimistic-lock="version">
        <id name="empNo" type="short">
            <column name="emp_no" />
            <generator class="assigned" />
        </id>
        <many-to-one name="departamentos" class="ud3_hibernate1.Departamentos" fetch="select">
            <column name="dept_no" not-null="true" />
        </many-to-one>
        <property name="apellido" type="string">
            <column name="apellido" length="10" />
        </property>
        <property name="oficio" type="string">
            <column name="oficio" length="10" />
        </property>
        <property name="dir" type="java.lang.Short">
            <column name="dir" />
        </property>
        <property name="fechaAlt" type="date">
            <column name="fecha_alt" length="10" />
        </property>
        <property name="salario" type="java.lang.Float">
            <column name="salario" precision="6" />
        </property>
        <property name="comision" type="java.lang.Float">
            <column name="comision" precision="6" />
        </property>
    </class>
</hibernate-mapping>
```

```xml
<hibernate-mapping>
    <class name="ud3_hibernate1.Departamentos" table="departamentos" catalog="ud3_xampp" optimistic-lock="version">
        <id name="deptNo" type="byte">
            <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>
        <set name="empleadoses" table="empleados" inverse="true" lazy="true" fetch="select">
            <key>
                <column name="dept_no" not-null="true" />
            </key>
            <one-to-many class="ud3_hibernate1.Empleados" />
        </set>
    </class>
</hibernate-mapping>
```

Sin embargo, estas asociaciones entre tablas también se pueden hacer de forma manual sobre tablas que no tienen clave ajena definida. Un ejemplo sería si queremos emplear el atributo *dir* de la tabla Empleados, que es también un número de empleado que se corresponde con el director de un empleado, para establecer una relación uno a muchos (one-to-many) entre el atributo *dir* y el atributo *emp_no* de esa misma tabla Empleados. Para ello, añado la relación al final del fichero Empleados.hbm.xml

```xml
        <!-- Añado mapeo relación entre atributo emp y emp-no -->
        <set name ="empleacargo" table="empleados" >
            <key>
                <column name="dir" not-null="true" />
            </key>
            <one-to-many class= "ud3_hibernate1.Empleados" />
        </set>
    </class>
</hibernate-mapping>
```

y en la clase Empleados.java añadimos una colección con el nombre que hemos dado a la relación (empleacargo) y los métodos get y set que permiten acceder a ella.

```java
private Float comision;
private Set<Empleados> empleacargo = new HashSet<Empleados>(0);      // Añado colección por la relación empleacargo
```

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

    // Añado get y set para relación empleacargo
    public Set<Empleados> getEmpleacargo(){
        return empleacargo;
    }

    public void setEmpleacargo(Set<Empleados> empleacargo){
        this.empleacargo = empleacargo;
    }

}
```

Hecho esto ya podré realizar consultas con joins usando la relación empleacargo.

HQL Query0

Session: hibernate.cfg

```
from Empleados as emp join emp.empleacargo
```

Result SQL — Set Max. Row Count: 100

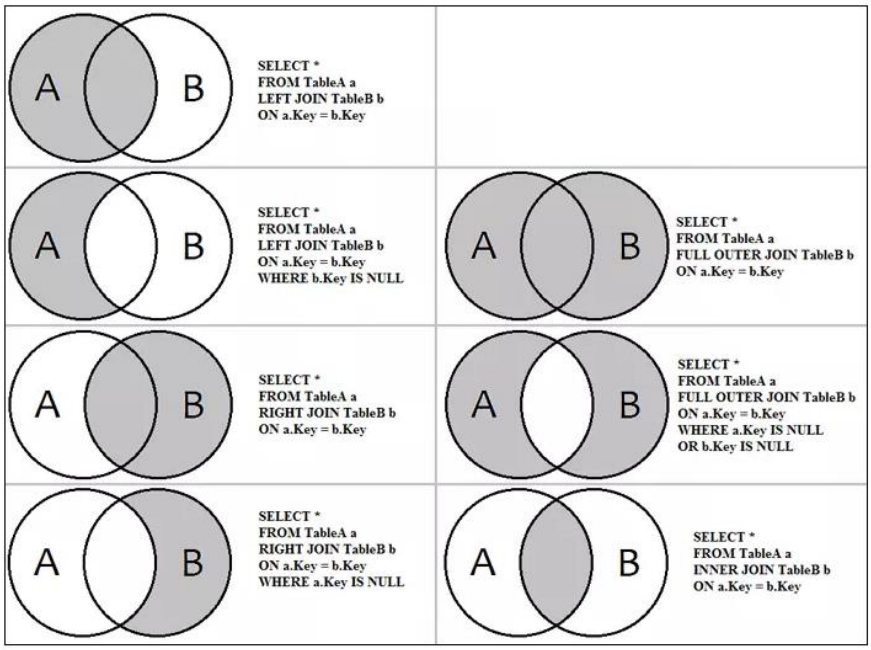
0 row(s) updated.; 13 row(s) selected.

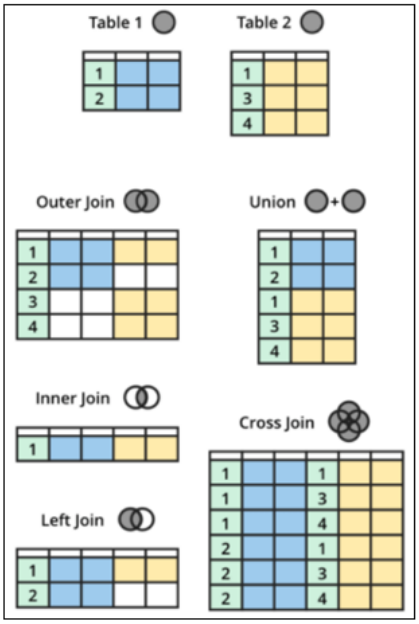
Dir	Comision	Salario	Apellido	FechaAlt	Empleacargo	Departame...	EmpNo	Oficio	Dir	Comision	Salario	Apellido	FechaAlt	Empleacargo	Departame...	EmpNo	Oficio
7566	NULL	3000.0	FERNÁNDEZ	1991-12-03	[ud3_hibernate1.Em...	ud3_hiberna...	7902	ANALISTA	7902	NULL	1040.0	SÁNCHEZ	1990-12-17	[]	ud3_hiberna...	7369	EMPLEADO
7839	NULL	3005.0	NEGRO	1991-05-01	[ud3_hibernate1.Em...	ud3_hiberna...	7698	DIRECTOR	7698	390.0	1500.0	ARROYO	1990-02-20	[]	ud3_hiberna...	7499	VENDEDOR
7839	NULL	3005.0	NEGRO	1991-05-01	[ud3_hibernate1.Em...	ud3_hiberna...	7698	DIRECTOR	7698	650.0	1625.0	SALA	1991-02-22	[]	ud3_hiberna...	7521	VENDEDOR
NULL	NULL	4100.0	REY	1991-11-17	[ud3_hibernate1.Em...	ud3_hiberna...	7839	PRESIDENTE	7839	NULL	2900.0	JIMÉNEZ	1991-04-02	[ud3_hibernate1.Empleados@776d0b9b, ud3_...	ud3_hiberna...	7566	DIRECTOR
7839	NULL	3005.0	NEGRO	1991-05-01	[ud3_hibernate1.Em...	ud3_hiberna...	7698	DIRECTOR	7698	1020.0	1600.0	MARTÍN	1991-09-29	[]	ud3_hiberna...	7654	VENDEDOR
NULL	NULL	4100.0	REY	1991-11-17	[ud3_hibernate1.Em...	ud3_hiberna...	7839	PRESIDENTE	7839	NULL	3005.0	NEGRO	1991-05-01	[ud3_hibernate1.Empleados@70e4fb80, ud3_h...	ud3_hiberna...	7698	DIRECTOR
NULL	NULL	4100.0	REY	1991-11-17	[ud3_hibernate1.Em...	ud3_hiberna...	7839	PRESIDENTE	7839	NULL	2885.0	CEREZO	1991-06-09	[ud3_hibernate1.Empleados@9c2e410]	ud3_hiberna...	7782	DIRECTOR
7839	NULL	2900.0	JIMÉNEZ	1991-04-02	[ud3_hibernate1.Em...	ud3_hiberna...	7566	DIRECTOR	7566	NULL	3000.0	GIL	1991-11-09	[ud3_hibernate1.Empleados@2f418ef8]	ud3_hiberna...	7788	ANALISTA
7839	NULL	3005.0	NEGRO	1991-05-01	[ud3_hibernate1.Em...	ud3_hiberna...	7698	DIRECTOR	7698	0.0	1350.0	TOVAR	1991-09-08	[]	ud3_hiberna...	7844	VENDEDOR
7566	NULL	3000.0	GIL	1991-11-09	[ud3_hibernate1.Em...	ud3_hiberna...	7788	ANALISTA	7788	NULL	1430.0	ALONSO	1991-09-23	[]	ud3_hiberna...	7876	EMPLEADO
7839	NULL	3005.0	NEGRO	1991-05-01	[ud3_hibernate1.Em...	ud3_hiberna...	7698	DIRECTOR	7698	NULL	1335.0	JIMENO	1991-12-03	[]	ud3_hiberna...	7900	EMPLEADO
7839	NULL	2900.0	JIMÉNEZ	1991-04-02	[ud3_hibernate1.Em...	ud3_hiberna...	7566	DIRECTOR	7566	NULL	3000.0	FERNÁNDEZ	1991-12-03	[ud3_hibernate1.Empleados@777fdb0c]	ud3_hiberna...	7902	ANALISTA
7839	NULL	2885.0	CEREZO	1991-06-09	[ud3_hibernate1.Em...	ud3_hiberna...	7782	DIRECTOR	7782	NULL	1690.0	MUÑOZ	1992-01-23	[]	ud3_hiberna...	7934	EMPLEADO

Recordamos los tipos de uniones entre tablas

SQL JOINS

- **(INNER) JOIN**: Devuelve registros que tienen valores coincidentes en ambas tablas
- **LEFT (OUTER) JOIN**: Devuelve todos los registros de la tabla izquierda y los registros coincidentes de la tabla derecha
- **RIGHT (OUTER) JOIN**: Devuelve todos los registros de la tabla derecha y los registros coincidentes de la tabla izquierda
- **FULL (OUTER) JOIN**: Devuelve todos los registros cuando hay una coincidencia en la tabla izquierda o derecha





La información completa sobre el lenguaje de consulta de Hibernate (HQL) se encuentra en el siguiente enlace => https://docs.jboss.org/hibernate/orm/3.5/reference/es-ES/html/queryhql.html

Actividad 3.2 Consultas básica HQL

Realiza consultas HQL con las tablas mapeadas. Prueba estas consultas:

```
from Empleados as e where e.departamentos.deptNo = 10
from Departamentos where deptNo=10
from Departamentos as d join d.empleadoses
from Departamentos as d left outer join d.empleadoses
```

HQL Query1

Session: hibernate.cfg

```
from Empleados as e where e.departamentos.deptNo = 10
```

Result SQL

0 row(s) updated.; 3 row(s) selected.

Dir	EmpNo	Departamentos	Apellido	FechaAlt	Salario	Oficio	Comision
7839	7782	ud3_hibernate1.Departamentos@4764c067	CEREZO	1991-06-09	2885.0	DIRECTOR	NULL
NULL	7839	ud3_hibernate1.Departamentos@4764c067	REY	1991-11-17	4100.0	PRESIDENTE	NULL
7782	7934	ud3_hibernate1.Departamentos@4764c067	MUÑOZ	1992-01-23	1690.0	EMPLEADO	NULL

HQL Query0

Session: hibernate.cfg

```
from Departamentos where deptNo=10
```

Result SQL

0 row(s) updated.; 1 row(s) selected.

Empleadoses	Loc	Dnombre	DeptNo
[ud3_hibernate1.Empleados@5d9bcc6f, ud3_hibernate1.Empleados@51d072f6, ud3_hibernate1.Empleados@68580e5f]	SEVILLA	CONTABILI...	10

Al escribir la consulta HQL en la pestaña SQL puedo ver la sentencia SQL que se está ejecutando

Session: hibernate.cfg

```
from Departamentos as d join d.empleadoses
```

Result SQL Set Max. Row Cou

```
select departamen0_.dept_no as col_0_0_, empleadose1_.emp_no as col_1_0_ from ud3_xampp.departamentos departamen0_ inner join
ud3_xampp.empleados empleadose1_ on departamen0_.dept_no=empleadose1_.dept_no
```

Y en la pestaña Result vemos el resultado de la consulta

Session: hibernate.cfg

```
from Departamentos as d join d.empleadoses
```

Result SQL — Set Max. Row Count: 100

0 row(s) updated.; 14 row(s) selected.

Dnombre	Empleadoses	Loc	DeptNo	Dir	Departame...	EmpNo	Apellido	Oficio	FechaAlt	Salario	Comision
INVESTIGAC...	[ud3_hibern...	MADRID	20	7902	ud3_hiberna...	7369	SÁNCHEZ	EMPLEADO	1990-12-17	1040.0	NULL
VENTAS	[ud3_hibern...	BARCELONA	30	7698	ud3_hiberna...	7499	ARROYO	VENDEDOR	1990-02-20	1500.0	390.0
VENTAS	[ud3_hibern...	BARCELONA	30	NULL	ud3_hiberna...	7521	SALA	VENDEDOR	1991-02-22	1625.0	650.0
INVESTIGAC...	[ud3_hibern...	MADRID	20	7839	ud3_hiberna...	7566	JIMÉNEZ	DIRECTOR	1991-04-02	2900.0	NULL
VENTAS	[ud3_hibern...	BARCELONA	30	7698	ud3_hiberna...	7654	MARTÍN	VENDEDOR	1991-09-29	1600.0	1020.0
VENTAS	[ud3_hibern...	BARCELONA	30	7839	ud3_hiberna...	7698	NEGRO	DIRECTOR	1991-05-01	3005.0	NULL
CONTABILI...	[ud3_hibern...	SEVILLA	10	7839	ud3_hiberna...	7782	CEREZO	DIRECTOR	1991-06-09	2885.0	NULL
INVESTIGAC...	[ud3_hibern...	MADRID	20	7566	ud3_hiberna...	7788	GIL	ANALISTA	1991-11-09	3000.0	NULL
CONTABILI...	[ud3_hibern...	SEVILLA	10	NULL	ud3_hiberna...	7839	REY	PRESIDENTE	1991-11-17	4100.0	NULL
VENTAS	[ud3_hibern...	BARCELONA	30	7698	ud3_hiberna...	7844	TOVAR	VENDEDOR	1991-09-08	1350.0	0.0
INVESTIGAC...	[ud3_hibern...	MADRID	20	7788	ud3_hiberna...	7876	ALONSO	EMPLEADO	1991-09-23	1430.0	NULL
VENTAS	[ud3_hibern...	BARCELONA	30	7698	ud3_hiberna...	7900	JIMENO	EMPLEADO	1991-12-03	1335.0	NULL
INVESTIGAC...	[ud3_hibern...	MADRID	20	7566	ud3_hiberna...	7902	FERNÁNDEZ	ANALISTA	1991-12-03	3000.0	NULL
CONTABILI...	[ud3_hibern...	SEVILLA	10	7782	ud3_hiberna...	7934	MUÑOZ	EMPLEADO	1992-01-23	1690.0	NULL

Session: hibernate.cfg

```
from Departamentos as d left outer join d.empleadoses
```

Result SQL — Set Max. Row Count: 100

```
select departamen0_.dept_no as col_0_0_, empleadose1_.emp_no as col_1_0_ from ud3_xampp.departamentos departamen0_ left outer join ud3_xampp.empleados empleadose1_ on departamen0_.dept_no=empleadose1_.dept_no
```

Session: hibernate.cfg

```
from Departamentos as d left outer join d.empleadoses
```

Result SQL — Set Max. Row Count: 100

0 row(s) updated.; 15 row(s) selected.

Dnombre	Empleadoses	Loc	DeptNo	Dir	Departame...	EmpNo	Apellido	Oficio	FechaAlt	Salario	Comision
INVESTIGAC...	[ud3_hibern...	MADRID	20	7902	ud3_hiberna...	7369	SÁNCHEZ	EMPLEADO	1990-12-17	1040.0	NULL
VENTAS	[ud3_hibern...	BARCELONA	30	7698	ud3_hiberna...	7499	ARROYO	VENDEDOR	1990-02-20	1500.0	390.0
VENTAS	[ud3_hibern...	BARCELONA	30	NULL	ud3_hiberna...	7521	SALA	VENDEDOR	1991-02-22	1625.0	650.0
INVESTIGAC...	[ud3_hibern...	MADRID	20	7839	ud3_hiberna...	7566	JIMÉNEZ	DIRECTOR	1991-04-02	2900.0	NULL
VENTAS	[ud3_hibern...	BARCELONA	30	7698	ud3_hiberna...	7654	MARTÍN	VENDEDOR	1991-09-29	1600.0	1020.0
VENTAS	[ud3_hibern...	BARCELONA	30	7839	ud3_hiberna...	7698	NEGRO	DIRECTOR	1991-05-01	3005.0	NULL
CONTABILI...	[ud3_hibern...	SEVILLA	10	7839	ud3_hiberna...	7782	CEREZO	DIRECTOR	1991-06-09	2885.0	NULL
INVESTIGAC...	[ud3_hibern...	MADRID	20	7566	ud3_hiberna...	7788	GIL	ANALISTA	1991-11-09	3000.0	NULL
CONTABILI...	[ud3_hibern...	SEVILLA	10	NULL	ud3_hiberna...	7839	REY	PRESIDENTE	1991-11-17	4100.0	NULL
VENTAS	[ud3_hibern...	BARCELONA	30	7698	ud3_hiberna...	7844	TOVAR	VENDEDOR	1991-09-08	1350.0	0.0
INVESTIGAC...	[ud3_hibern...	MADRID	20	7788	ud3_hiberna...	7876	ALONSO	EMPLEADO	1991-09-23	1430.0	NULL
VENTAS	[ud3_hibern...	BARCELONA	30	7698	ud3_hiberna...	7900	JIMENO	EMPLEADO	1991-12-03	1335.0	NULL
INVESTIGAC...	[ud3_hibern...	MADRID	20	7566	ud3_hiberna...	7902	FERNÁNDEZ	ANALISTA	1991-12-03	3000.0	NULL
CONTABILI...	[ud3_hibern...	SEVILLA	10	7782	ud3_hiberna...	7934	MUÑOZ	EMPLEADO	1992-01-23	1690.0	NULL

Sabiendo cómo funcionan las consultas en Hibernate, ahora lo que tendremos que aprender es a realizar las operaciones básicas que se realizan sobre los registros de una base de datos, es decir, MOSTRAR, INSERTAR, ACTUALIZAR y BORRAR registros.

4. Operaciones de carga, almacenamiento, modificación y borrado de datos-objetos.

Para las operaciones de manipulación de los datos-objetos de la base de datos se emplean métodos de la clase Session, que según el caso de almacenamiento, modificación y borrado se incluyen dentro de una instancia de Transaction. Los métodos se resumen en las siguientes tablas:

MÉTODO	DESCRIPCIÓN
<T> T load (Class<T> Clase, Serializable id)	Devuelve la instancia persistente de la clase indicada con el identificador dado. La instancia tiene que existir, si no existe el método lanza una excepción (ObjectNotFoundExcpetion).
Object load (String nombreClase, Serializable id)	Similar al método anterior pero en este caso indicamos en el primer parámetro el nombre de la clase en formato String
<T> T get (Class<T> Clase, Serializable id)	Devuelve la instancia persistente de la clase indicada con el identificador dado. La instancia tiene que existir, si no existe devuelve null
Object get (String nombreClase, Serializable id)	Similar al método anterior pero en este caso indicamos en el primer parámetro el nombre de la clase en formato String

MÉTODO	DESCRIPCIÓN
Serializable save (Object obj)	Guarda el objeto en la base de datos. Hace que la instancia transitoria del objeto sea persistente.
void update (Object objeto)	Actualiza en la base de datos el objeto que se pasa como argumento. El objeto a modificar debe ser cargado con el método load() o get()
void delete (Object objeto)	Elimina de la base de datos el objeto que se pasa como argumento. El objeto a eliminar debe ser cargado con el método load() o get().

Ejemplo MainSesion.

Indica las clases que debemos manejar para disponer de una Session para poder realizar operaciones con la BD.

```java
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;

public class MainSesion {

	public static void main(String[] args) {
		// Inicializa el entorno Hibernate
		Configuration cfg = new Configuration().configure();
		// Crea el ejemplar de sesion factory
		SessionFactory sessionFactory = cfg.buildSessionFactory(new StandardServiceRegistryBuilder().configure().build());
		// Obtiene un objeto session
		Session session = sessionFactory.openSession();

/* otra forma
Configuration configuration = new Configuration().configure();
StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder().applySettings(configuration.getProperties());
SessionFactory sessionFactory = configuration.buildSessionFactory(builder.build());

Session session = sessionFactory.openSession();
*/
		session.close();
		System.exit(0);
	}
}
```

Ejemplo CargaObjetos.

Carga el objeto Departamentos con un determinado identificador (PK), empleando el método load y muestra alguno de sus atributos en pantalla

```java
import org.hibernate.ObjectNotFoundException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.exception.ConstraintViolationException;

import ud3_hibernate1.Departamentos;
import ud3_hibernate1.HibernateUtil;

public class CargaObjetos {
	public static void main(String[] args) {
		SessionFactory sesion = HibernateUtil.getSessionFactory();
		Session session = sesion.openSession();
		// Visualizo los datos del departamento 20
		Departamentos dep = new Departamentos();
		try {
			//dep = (Departamentos) session.load(Departamentos.class, (byte) 28);
			dep = (Departamentos) session.load("ud3_hibernate1.Departamentos", (byte) 20);

			System.out.printf("Nombre Dep: %s%n", dep.getDnombre());
			System.out.printf("Localidad: %s%n", dep.getLoc());
		} catch (ObjectNotFoundException o) {
			System.out.println("NO EXISTE EL DEPARTAMENTO!!");
		}

		session.close();
		System.exit(0);

	}

}
```

Ejemplo CargaObjetosGet.

Carga el objeto Departamentos con un determinado identificador (PK), empleando el método get y muestra alguno de sus atributos en pantalla. Conviene fijarse que el método get devuelve null si el objeto-registro no existe.

```
import org.hibernate.ObjectNotFoundException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.exception.ConstraintViolationException;

import ud3_hibernate1.Departamentos;
import ud3_hibernate1.HibernateUtil;

public class CargaObjetosGet {

        public static void main(String[] args) {
                SessionFactory sesion = HibernateUtil.getSessionFactory();
                Session session = sesion.openSession();

                Departamentos dep = (Departamentos) session.get(Departamentos.class, (byte) 10);
                if (dep==null) {
                        System.out.println("El departamento no existe");
                }
                else
                {
                        System.out.printf("Nombre Dep: %s%n",dep.getDnombre());
                        System.out.printf("Localidad: %s%n",dep.getLoc());
                }
                session.close();
                System.exit(0);
        }
}
```

Actividad 3.3. Ejemplo MainEmpleado.java

Inserta un empleado en la tabla Empleados. Hay que fijarse en que el método setDepartamentos en la clase Empleados necesita que se le pase un objeto de la clase Departamentos. Por eso, lo que hace es crear una instancia de Departamentos e insertarle al menos el valor del departamento.

```
Departamentos d = new Departamentos(); // creo un objeto Departamentos
d.setDeptNo((byte) 10); // el n□mero de dep es 10
em.setDepartamentos(d);
```

Analizo las diferentes partes del código:

a) Importa clases y abre sesión y transacción

```
import ud3_hibernate1.*;            // Importo los POJOs para acceder a sus atributos

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.TransientPropertyValueException;
import org.hibernate.exception.ConstraintViolationException;

public class MainEmpleado {
        public static void main(String[] args) {
                // Abre sesión y transacción
                SessionFactory sesion = HibernateUtil.getSessionFactory();
                Session session = sesion.openSession();
                Transaction tx = session.beginTransaction();
```

b) Crea instancia de empleado y le asigna valores a sus atributos o características

```
System.out.println("Inserto un EMPLEADO EN EL DEPARTAMENTO 10.");
// Crea una instancia de empleado y le asigna valores
Float salario = new Float(1500);// inicializo el salario
Float comision = new Float(10); // inicializo la comisión
Empleados em = new Empleados(); // creo un objeto empleados
em.setEmpNo((short) 4457); // el numero de empleado es 4455
em.setApellido("PEPE");
em.setDir((short) 7499); // el director es el numero de empleado 7499
em.setOficio("VENDEDOR");
em.setSalario(salario);
em.setComision(comision);
Departamentos d = new Departamentos();  // creo un objeto Departamentos
d.setDeptNo((byte) 10); // el número de dep es 10
em.setDepartamentos(d);
// fecha de alta
java.util.Date hoy = new java.util.Date();
java.sql.Date fecha = new java.sql.Date(hoy.getTime());
em.setFechaAlt(fecha);
```

c) Inserta el registro (objeto), confirma la transacción y cierra la sesión, gestionando las diferentes excepciones que pueden aparecer.

```
try {
        session.save(em);
        try {
                tx.commit();
        } catch (ConstraintViolationException e) {
                System.out.println("EMPLEADO DUPLICADO");
                System.out.printf("MENSAJE: %s%n", e.getMessage());
                System.out.printf("COD ERROR: %d%n", e.getErrorCode());
                System.out.printf("ERROR SQL: %s%n", e.getSQLException().getMessage());
        }
} catch (TransientPropertyValueException e) {
        System.out.println("EL DEPARTAMENTO NO EXISTE");
        System.out.printf("MENSAJE: %s%n", e.getMessage());
        System.out.printf("Propiedad: %s%n", e.getPropertyName());
} catch (Exception e) {
        System.out.println("ERROR NO CONTROLADO....");
        e.printStackTrace();
}

session.close();
System.exit(0);
```

Inserta el objeto empleado en la tabla empleados usando el método save

Confirma la transacción

Cierra la sesión

Actividad 3.3. Ejemplo ListadoDep.java

Muestra por pantalla un listado con los datos del departamento 10 y sus empleados. Al disponer en la clase Departamentos de un atributo que enlaza con la tabla empleados con un método getEmpleados, puede llamar a este método sin tener que cargar datos desde la tabla empleados. Esto es una de las ventajas que ofrece Hibernate al crear métodos que facilitan la consulta de información entre tablas que están relacionadas por claves foráneas o ajenas (FK). Analizo las diferentes partes del código:

a) Abre sesión y carga los datos del departamento 10 en una instancia de objeto Departamento, empleando el método load.

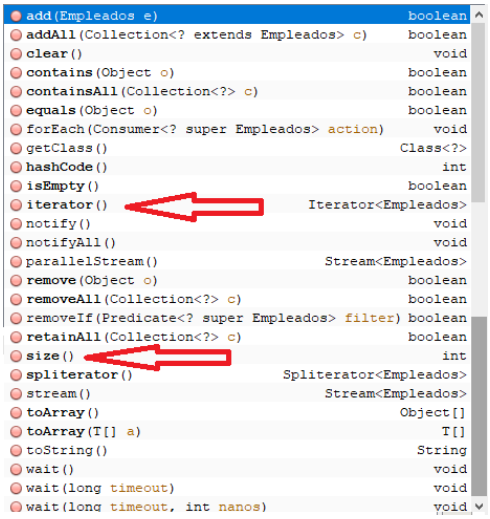
```
import java.util.Iterator;
import java.util.Set;

import ud3_hibernate1.*;

import org.hibernate.Session;
import org.hibernate.SessionFactory;

public class ListadoDep {
        public static void main(String[] args) {
                SessionFactory sesion = HibernateUtil.getSessionFactory();
                Session session = sesion.openSession();

                System.out.println("================================");
                System.out.println("DATOS DEL DEPARTAMENTO 10.");

                Departamentos dep = new Departamentos();
                dep = (Departamentos) session.load(Departamentos.class, (byte) 10);
                System.out.println("Nombre Dep:" + dep.getDnombre());
                System.out.println("Localidad:" + dep.getLoc());
```

Crea una instancia de objeto Departamentos y usa el método load para cargar en el objeto los datos del departamento 10

b) Emplea el método getEmpleadoses de la clase Departamentos para obtener los datos de los empleados que pertenecen al departamento que se ha cargado en memoria en el paso anterior (instancia dep).

```
                System.out.println("================================");
                System.out.println("EMPLEADOS DEL DEPARTAMENTO 10.");

                Set<Empleados> listaemple = dep.getEmpleadoses();// obtenemos empleados

                Iterator<Empleados> it = listaemple.iterator();

                System.out.printf("Número de empleados: %d %n", listaemple.size());
```

Los datos se devuelven en un conjunto (Set<Empleados>) para el que disponemos de varios métodos para trabajar con él, como por ejemplo el método size() para saber el número de elementos que incluye, o el método iterator() que devuelve un iterador para recorrer los elementos.



c) Emplea el iterador para recorrer los elementos del conjunto de Empleados imprimiendo algunos de sus atributos. Después cierra la sesión.

```
            Iterator<Empleados> it = listaemple.iterator();
            System.out.printf("Número de empleados: %d %n", listaemple.size());
            while (it.hasNext()) {
                    // Empleados emple = new Empleados();
                    Empleados emple = it.next();
                    System.out.printf("%s * %.2f %n", emple.getApellido(), emple.getSalario());
            }
            System.out.println("================================");
            session.close();
            System.exit(0);
        }
```

```
==============================
DATOS DEL DEPARTAMENTO 10.
Hibernate: select departamen0_.dept_no as dept_no1_0_0_, departamen0_.dnombre as dnombre2_0_0_, departamen0_.loc as loc3_0_0_ from ud2_xampp.departamentos departamen0_ where departamen0_.dept_no=?
Nombre Dep:CONTABILIDAD
Localidad:SEVILLA
==============================
EMPLEADOS DEL DEPARTAMENTO 10.
Hibernate: select empleadose0_.dept_no as dept_no2_0_0_, empleadose0_.emp_no as emp_no1_1_0_, empleadose0_.emp_no as emp_no1_1_1_, empleadose0_.dept_no as dept_no2_1_1_, empleadose0_.apellido as ap
Número de empleados: 3
MUÑOZ * 1690,00
REY * 4100,00
CEREZO * 2885,00
==============================
```

En este ejemplo debe añadirse código para controlar la excepción ObjectNotFoundException que salta si el método load detecta que la instancia que tiene que cargar no existe en la base de datos

MÉTODO	DESCRIPCIÓN
<T> T load (Class<T> Clase, Serializable id)	Devuelve la instancia persistente de la clase indicada con el identificador dado. La instancia tiene que existir, si no existe el método lanza una excepción (ObjectNotFoundExcpetion).

```
DATOS DEL DEPARTAMENTO 10.
Hibernate: select departamen0_.dept_no as dept_no1_0_0_, departamen0_.dnombre as dnombre2_0_0_, departamen0_.loc as loc3_0_0_ from ud2_xampp.departamentos departamen0_ where departamen0_.dept_no=?
Exception in thread "main" org.hibernate.ObjectNotFoundException: No row with the given identifier exists: [ud3_hibernate1.Departamentos#50]
        at org.hibernate.internal.SessionFactoryImpl$1$1.handleEntityNotFound(SessionFactoryImpl.java:252)
        at org.hibernate.proxy.AbstractLazyInitializer.checkTargetState(AbstractLazyInitializer.java:261)
        at org.hibernate.proxy.AbstractLazyInitializer.initialize(AbstractLazyInitializer.java:175)
        at org.hibernate.proxy.AbstractLazyInitializer.getImplementation(AbstractLazyInitializer.java:285)
        at org.hibernate.proxy.pojo.javassist.JavassistLazyInitializer.invoke(JavassistLazyInitializer.java:185)
        at ud3_hibernate1.Departamentos_$$_jvstf14_1.getDnombre(Departamentos_$$_jvstf14_1.java)
        at ListadoDep.main(ListadoDep.java:19)
```

```
    try{
        dep = (Departamentos) session.load(Departamentos.class, (byte) 10);
        System.out.println("Nombre Dep:" + dep.getDnombre());
        System.out.println("Localidad:" + dep.getLoc());

        System.out.println("============================");
        System.out.println("EMPLEADOS DEL DEPARTAMENTO 10.");

        Set<Empleados> listaemple = dep.getEmpleadoses();// obtenemos empleados

        Iterator<Empleados> it = listaemple.iterator();

        System.out.printf("NÚmero de empleados: %d %n", listaemple.size());
        while (it.hasNext()) {
                // Empleados emple = new Empleados();
                Empleados emple = it.next();
                System.out.printf("%s * %.2f %n", emple.getApellido(), emple.getSalario());
        }
        System.out.println("============================");

    } catch (ObjectNotFoundException o) {
                System.out.println("NO EXISTE EL DEPARTAMENTO...");
    }
        session.close();
        System.exit(0);
}
```

Actividad 3.3. Ejemplo Borrado.java

Elimina el departamento 10. Analizo las diferentes partes del código:

a) Abre sesión y carga los datos del departamento 10 en una instancia de objeto Departamento, empleando el método load.

```
import org.hibernate.ObjectNotFoundException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.exception.ConstraintViolationException;

import ud3_hibernate1.*;

public class BorradoDep {

        public static void main(String[] args) {
                SessionFactory sesion = HibernateUtil.getSessionFactory();
                Session session = sesion.openSession();
                Transaction tx = session.beginTransaction();

                Departamentos de = (Departamentos) session.load(Departamentos.class, (byte) 10);
```

b) Trata de eliminar el objeto, confirma la transacción y luego cierra la sesión.

```
                try {
                        session.delete(de); // elimina el objeto
                        tx.commit();
                        System.out.println("Departamento eliminado");

                } catch (ObjectNotFoundException o) {
                        System.out.println("NO EXISTE EL DEPARTAMENTO...");
                } catch (ConstraintViolationException c) {
                        System.out.println("NO SE PUEDE ELIMINAR, TIENE EMPLEADOS...");
                }catch (Exception e) {
                        System.out.println("ERROR NO CONTROLADO....");
                        e.printStackTrace();
                }
                session.close();
                System.exit(0);
        }
```

Emplea el método delete para eliminar el objeto en la base de datos

Confirma la transacción

No consigue eliminar el objeto y salta una excepción ConstraintViolationException porque la base de datos no le deja al ser clave foránea y tener empleados que tienen asignado ese departamento.

```
Hibernate: select departamen0_.dept_no as dept_no1_0_0_, departamen0_.dnombre as dnombre2_0_0_, departamen0_.loc as loc3_0_0_ from ud2_xampp.departamentos departamen0_ where departamen0_.dept_no=?
Hibernate: delete from ud2_xampp.departamentos where dept_no=?
NO SE PUEDE ELIMINAR, TIENE EMPLEADOS...
```

Actividad 3.3. Ejemplo ModificarEmpleado.java

Modifica el salario y el departamento del empleado 7369, le asigna el departamento 30 y le sube el salario 1000 €. Analizo las diferentes partes del código:

a) Abre sesión y carga los datos del empleado 7369 en una instancia de objeto Departamento, empleando el método load, controlando la excepción ObjectNotFoundException, por si no existe el empleado.

```
import org.hibernate.ObjectNotFoundException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.exception.ConstraintViolationException;

import ud3_hibernate1.*;

public class ModificarEmpleado {

        public static void main(String[] args) {
                SessionFactory sesion = HibernateUtil.getSessionFactory();
                Session session = sesion.openSession();
                Transaction tx = session.beginTransaction();

                Empleados em = new Empleados();
                try {
                        em = (Empleados) session.load(Empleados.class, (short) 7369);
                        System.out.printf("Modificación empleado: %d%n", em.getEmpNo());
                        System.out.printf("Salario antiguo: %.2f%n", em.getSalario());
                        System.out.printf("Departamento antiguo: %s%n", em.getDepartamentos().getDnombre());
```

b) Modifica el salario y lo asigna con el método setSalario de la clase Empleados. Después carga en memoria los datos del departamento 30, pero usando el método get de la sesion que, a diferencia del método load que lanza una excepción, simplemente devuelve null si el identificador de la clase (el departamento) no existe. Necesita cargar el objeto Departamento porque al método setDepartamentos de la clase Empleados debe pasarle como parámetro una instancia de la clase Departamentos. Para modificar el registro en la base de datos emplea el método update de la sesión.

```
        float NuevoSalario= em.getSalario()+1000;
        em.setSalario(NuevoSalario);

        Departamentos dep = (Departamentos) session.get(Departamentos.class, (byte) 30);
        if (dep == null) {
                System.out.println("El departamento NO existe");
        } else {
                em.setDepartamentos(dep);
                session.update(em); // modifica el objeto
                tx.commit();
                System.out.printf("Salario nuevo: %.2f%n",em.getSalario());
                System.out.printf("Departamento nuevo: %s %n", em.getDepartamentos().getDnombre());
        }
```

Actualiza el registro en la base de datos con el método update

Confirma la transacción

Imprime el nombre del nuevo departamento

c) Captura las posibles excepciones y cierra la sesión.

```
                } catch (ObjectNotFoundException o) {
                        System.out.println("NO EXISTE EL EMPLEADO...");
                } catch (ConstraintViolationException c) {
                        System.out.println("NO SE PUEDE ASIGNAR UN DEPARTAMENTO QUE NO EXISTE.....");
                } catch (Exception e) {
                        System.out.println("ERROR NO CONTROLADO....");
                        e.printStackTrace();
                }
                session.close();
                System.exit(0);
        }
```

Actividad 3.4. Ejercicio - **Modificar registro en la BD y mostrar información por pantalla**

Sube el salario a todos los empleados del departamento 10. Muestra el apellido y el salario del empleado antes y después de actualizar.

```
// Actividad 3.4. Sube el salario a todos los empleados de un departamento

import java.util.Iterator;
import java.util.Set;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import ud3_hibernate1.Departamentos;
import ud3_hibernate1.Empleados;
import ud3_hibernate1.HibernateUtil;

public class Actividad3_4 {

    static SessionFactory sesion;
    static Session session;

    public static void main(String[] args) {
        /*if (args.length != 1) {              // Control de si se ha pasado el valor como parámetro
            System.out.println("Hay que introducir 1 argumento");
            System.exit(0);
        }*/

        String subida= "200"; // valor de la subida
        float subir = 0;
        byte dpto = 10;       // valor del departamento

        try {
            subir =Float.parseFloat(subida);       // Convierte el valor al tipo float
        }catch (NumberFormatException n){
            System.out.println("Valor de la subida no numérico");
            System.exit(0);
        }
        // Apertura de sesión con la clase HibernateUtil y de transacción
        sesion = HibernateUtil.getSessionFactory();
        session = sesion.openSession();
        Transaction tx = session.beginTransaction();
        System.out.printf("SUBIMOS %.2f a los empleados del dep %d %n", subir, dpto);

        // Carga el objeto Departamento con el identificador que tenga la variable dpto
        Departamentos dep = new Departamentos();
        dep = (Departamentos) session.load(Departamentos.class, dpto);
        System.out.println("Nombre Dep:"+dep.getDnombre());
        System.out.println("Localidad:"+dep.getLoc());

        System.out.println("=============================================" );
        System.out.println("EMPLEADOS DEL DEPARTAMENTO " + dpto + " sin actualizar" );
        VerEmpleados(dpto);

        // Carga los empleados del departamento
        Set<Empleados> listaemple = dep.getEmpleadoses(); //obtenemos empleados
        Iterator <Empleados> it = listaemple.iterator();
        System.out.println("Número de empleados: " + listaemple.size() );
        while(it.hasNext()) { // Recorre los empleados subiendo el salario y almacena de nuevo el empleado en la BD
            Empleados emple=it.next();
            emple.setSalario(emple.getSalario() + subir);
            session.update(emple);
        }

        tx.commit();    // confirma la transacci□n

        System.out.println("=============================================" );
        System.out.println("EMPLEADOS DEL DEPARTAMENTO " + dpto + " actualizados" );
        VerEmpleados(dpto);

        session.close();
        System.exit(0);

    }  // fin main
```

Método VerEmpleados para mostrar por pantalla los empleados

```
    // Método que imprime el apellido y el salario de los empleados
    private static void VerEmpleados(byte dpto) {
        Departamentos dep = new Departamentos();
        dep = (Departamentos) session.load(Departamentos.class, dpto);
        Set<Empleados> listaemple = dep.getEmpleadoses(); //obtenemos empleados
        Iterator <Empleados> it = listaemple.iterator();

        while(it.hasNext()) {
            Empleados emple=it.next();
            System.out.printf("Empleado:%s salario: %.2f %n", emple.getApellido(), emple.getSalario());
        }
    }  // fin método VerEmpleados

}  // fin de la clase
```

5. Ejemplos de operaciones con consultas.

Podemos realizar consultas HQL dentro de nuestra aplicación que se representan con una instancia de org.hibernate.Query. Se obtiene una Query utilizando el objeto Session actual y la Query dispone de métodos útiles para tratar los datos que se recuperan en la consulta. Podemos encontrar información del interface Query en el siguiente enlace => https://docs.jboss.org/hibernate/orm/3.2/api/org/hibernate/Query.html

A continuación vemos algunos ejemplos del uso de la interface Query y sus métodos.

Actividad 3.5. Ejemplo 5. ListaDepartamentos

Realiza una consulta de todas las filas de la tabla departamentos y las recorre empleando el método list() disponible en la Query. El método list() devuelve en una colección todos los resultados de la consulta. En la colección se encuentran instanciadas todas las entidades que corresponden al resultado de la ejecución de la consulta Este método realiza una única comunicación con la base de datos en la que se traen todos los resultados, por lo que si los registros recuperados son muchos tardará bastante tiempo y necesitará memoria suficiente para almacenar todos los objetos.

```
Query q = session.createQuery("from Departamentos");
List <Departamentos> lista = q.list();
// Obtenemos un Iterador y recorremos la lista.
Iterator <Departamentos> iter = lista.iterator();
System.out.printf("NÚmero de registros: %d%n",lista.size());
while (iter.hasNext())
{
   //extraer el objeto
        Departamentos   depar = (Departamentos) iter.next();
   System.out.printf("%d, %s%n", depar.getDeptNo(), depar.getDnombre());
}
```

Actividad 3.5. Ejemplo 6. ListaDepartamentosIterator

Realiza una consulta de todas las filas de la tabla departamentos y las recorre empleando el método iterate() disponible en la Query. El método iterate() devuelve un iterador Java para recuperar los datos de la consulta. En la consulta se devuelve sólo los identificadores de los registros o entidades. Después, en cada llamada al método Iterator.next() se ejecuta una nueva consulta para recuperar todos los datos de la entidad. Esto supone que se realizan más accesos a la base de datos que con el método list(), lo que aumenta el tiempo total de procesamiento, pero necesita menos memoria porque se cargan menos datos. Para reducir el número de accesos a la base de datos, el método setFetchSize() permite indicar el número de filas o registros, que como máximo, se recuperan con cada llamada a Iterator.Next().

```
Query q = session.createQuery("from Departamentos");
q.setFetchSize(10);
Iterator iter = q.iterate();
while (iter.hasNext())
{
   //extraer el objeto
   Departamentos   depar = (Departamentos) iter.next();
   System.out.printf("%d, %s%n", depar.getDeptNo(), depar.getDnombre());
}
```

Actividad 3.5. Ejemplo 7. ListaEmpleados20

Realiza una consulta de seleccionando las filas de la tabla empleados que pertenecen a un determinado departamento y luego las recorre empleando el método lista() disponible en la Query

```
Query q = session.createQuery("from Empleados as e where e.departamentos.deptNo = 20");
List<Empleados> lista = q.list();
// Obtenemos un Iterador y recorremos la lista
Iterator<Empleados> iter = lista.iterator();
while (iter.hasNext()) {
        Empleados emp = (Empleados) iter.next();// extraer el objeto
        System.out.printf("%s, %.2f %n", emp.getApellido(), emp.getSalario());
}
```

Actividad 3.5. Ejemplo 8. EjemploUniqueResult

Emplea el método uniqueResult() de la Query que devuelve un único objeto o nulo si la consulta no devuelve nada. Este método es útil cuando sabemos que la consulta devuelve un único objeto y nos evita tener que recorrer una lista.

```
// Visualiza los datos del departamento 10
String hql = "from Departamentos as dep where dep.deptNo = 10";
Query q = session.createQuery(hql);
Departamentos dep = (Departamentos) q.uniqueResult();
System.out.printf("%d, %s, %s%n", dep.getDeptNo(), dep.getLoc(), dep.getDnombre());

// Visualiza los datos del departamento de nombre VENTAS
hql = "from Departamentos as dep where dep.dnombre = 'VENTAS' ";
q = session.createQuery(hql);
dep = (Departamentos) q.uniqueResult();
System.out.printf("%d, %s, %s%n", dep.getDeptNo(), dep.getLoc(), dep.getDnombre());
```

Actividad 3.5. Ejemplo 9. ConsultasParametros

Hibernate soporta parámetros con nombre y parámetros de estilo JDBC (?) en las consultas HQL. Los parámetros con nombre son identificadores de la forma :nombre en la cadena de la consulta. Hibernate **numera los parámetros desde cero**. El uso de los parámetros con nombre ofrece ventajas como que:

- Da igual el orden en que aparecen en la consulta
- Pueden aparecer más de una vez en la misma consulta
- Se autodocumentan, ya que el nombre del parámetro dice algo más que el símbolo de interrogación(?)

Para asignar nombre a los parámetros se emplean los métodos setXXX. La sintaxis más simple de utilizar es usando el método setParameter().

Query	**setInteger**(int position, int val)
Query	**setInteger**(String name, int val)
Query	**setString**(int position, String val)
Query	**setString**(String name, String val)

Query	**setParameter**(int position, Object val) Bind a value to a JDBC-style query parameter.
Query	**setParameter**(int position, Object val, Type type) Bind a value to a JDBC-style query parameter.
Query	**setParameter**(String name, Object val) Bind a value to a named query parameter.
Query	**setParameter**(String name, Object val, Type type) Bind a value to a named query parameter.

```
String hql = "from Empleados where empNo = :numemple";
Query q1 = session.createQuery(hql);
q1.setParameter("numemple", (short) 7369);
Empleados emple = (Empleados) q1.uniqueResult();
System.out.printf("%s, %s %n", emple.getApellido(), emple.getOficio());

Query q2 = session.createQuery("from Empleados emp where emp.departamentos.deptNo = :ndep and emp.oficio = :ofi");
q2.setParameter("ndep", (byte) 10);
q2.setParameter("ofi", "DIRECTOR");
List<Empleados> lista = q2.list();
Iterator<Empleados> iter = lista.iterator();
while (iter.hasNext()) {
        // extraer el objeto
        Empleados emp = (Empleados) iter.next();
        System.out.printf("%d, %s%n", emp.getEmpNo(), emp.getApellido());
}

String hql3 = "from Empleados emp where emp.departamentos.deptNo = ? and emp.oficio = ?";
Query q3 = session.createQuery(hql3);
q3.setParameter(0, (byte) 10);
q3.setParameter(1, "DIRECTOR");
List<Empleados> lista2 = q3.list();
Iterator<Empleados> iter2 = lista2.iterator();
while (iter2.hasNext()) {
        // extraer el objeto
        Empleados emp = (Empleados) iter2.next();
        System.out.printf("%d, %s%n", emp.getEmpNo(), emp.getApellido());
}
```

```
String hql4 = "from Empleados emp where emp.departamentos.deptNo = :ndep and emp.oficio = :ofi";
Query q4 = session.createQuery(hql4);
q4.setInteger("ndep", (byte) 10);
q4.setString("ofi", "DIRECTOR");
List<Empleados> lista3 = q4.list();
Iterator<Empleados> iter3 = lista3.iterator();
while (iter3.hasNext()) {
        // extraer el objeto
        Empleados emp = (Empleados) iter3.next();
        System.out.printf("%d, %s%n", emp.getEmpNo(), emp.getApellido());
}

// from Empleados emp where emp.departamentos.deptNo in (10,20)
System.out.println("Empleados con departamento 10, 20 ");
List<Byte> numeros = new ArrayList<Byte>();
numeros.add((byte) 10);
numeros.add((byte) 20);
String hql5 = "from Empleados emp where emp.departamentos.deptNo in (:listadep) "
            + "order by emp.departamentos.deptNo ";
Query q5 = session.createQuery(hql5);
q5.setParameterList("listadep", numeros);

List<Empleados> lista4 = q5.list();
Iterator<Empleados> iter4 = lista4.iterator();
while (iter4.hasNext()) {
        // extraer el objeto
        Empleados emp = (Empleados) iter4.next();
        System.out.printf("%d, %d, %s%n", emp.getDepartamentos().getDeptNo(), emp.getEmpNo(), emp.getApellido());
}
```

```
// Parámetro de tipo Date
SimpleDateFormat formatoDelTexto = new SimpleDateFormat("yyyy-MM-dd");
String strFecha = "1991-12-03";
Date fecha = null;
try {
        fecha = formatoDelTexto.parse(strFecha);
} catch (ParseException ex) {
        ex.printStackTrace();
}
String hql6 = "from Empleados where fechaAlt = :fechalta ";

Query q6 = session.createQuery(hql6);
q6.setDate("fechalta", fecha);
System.out.println("Empleados cuya fecha de alta es 1991-12-03");
List<Empleados> lista5 = q6.list();
Iterator<Empleados> iter5 = lista5.iterator();
while (iter5.hasNext()) {
        // extraer el objeto
        Empleados emp = (Empleados) iter5.next();
        System.out.printf("%d, %d, %s%n", emp.getDepartamentos().getDeptNo(), emp.getEmpNo(), emp.getApellido());
}
```

Actividad 3.5. Ejemplo 10. ClasesNoAsociadas

Consultas sobre clases no asociadas. Si queremos recuperar los datos de una consulta en la que intervienen varias tablas y no tenemos asociada a ninguna clase los atributos que devuelve esa consulta podemos utilizar la clase Object. Los resultados se devuelven en un array de objetos, donde el primer elemento del array se corresponde con la primera clase que ponemos a la derecha de FROM, el siguiente elemento con la siguiente clase y así sucesivamente. Este ejemplo realiza una consulta para obtener los datos de los empleados y de sus departamentos. El resultado de la consulta se recibe en un array de objetos donde el primer elemento del array pertenece a la clase Empleados y el segundo a la clase Departamentos.

```
{
        String hql = "from Empleados e, Departamentos d where e.departamentos.deptNo = d.deptNo order by apellido";
        Query cons = session.createQuery(hql);
        Iterator q = cons.iterate();
        while (q.hasNext()) {
                Object[] par = (Object[]) q.next();
                Empleados em = (Empleados) par[0];
                Departamentos de = (Departamentos) par[1];
                System.out.printf("%s, %.2f, %s, %s %n", em.getApellido(), em.getSalario(), de.getDnombre(),
                                de.getLoc());
        }
        System.out.println("=============================================");
}
{
        // MOSTRAR SALARIO MEDIO DE LOS EMPLEADOS
        String hql = "select avg(em.salario) from Empleados as em";
        Query cons = session.createQuery(hql);
        Double suma = (Double) cons.uniqueResult();
        System.out.printf("Salario medio: %.2f%n", suma);
}
```

```
ALONSO, 1430,00, INVESTIGACIÓN, MADRID
ARROYO, 1900,00, VENTAS, BARCELONA
CEREZO, 3485,00, CONTABILIDAD, SEVILLA
FERNÁNDEZ, 3000,00, INVESTIGACIÓN, MADRID
GIL, 3000,00, INVESTIGACIÓN, MADRID
JIMÉNEZ, 2900,00, INVESTIGACIÓN, MADRID
JIMENO, 1735,00, VENTAS, BARCELONA
MARTÍN, 2000,00, VENTAS, BARCELONA
MUÑOZ, 2290,00, CONTABILIDAD, SEVILLA
NEGRO, 3405,00, VENTAS, BARCELONA
REY, 4700,00, CONTABILIDAD, SEVILLA
SALA, 2025,00, VENTAS, BARCELONA
SÁNCHEZ, 8040,00, INVESTIGACIÓN, MADRID
TOVAR, 1750,00, VENTAS, BARCELONA
=============================================
Salario medio: 2975,71
```

```
// Mostrar el salario medio y el numero de empleados
System.out.println("================================================");
String hql = "select avg(salario), count(empNo) from Empleados ";
Query cons = session.createQuery(hql);
Object[] resultado = (Object[]) cons.uniqueResult();
System.out.printf("Salario medio: %.2f%n", resultado[0]);
System.out.printf("Número de empleados: %d%n", resultado[1]);

// Mostrar el salario medio y el numero de empleados por departamento
hql = "select e.departamentos.deptNo, avg(salario), " + " count(empNo) from Empleados e "
            + " group by e.departamentos.deptNo ";

cons = session.createQuery(hql);
Iterator iter = cons.iterate();
while (iter.hasNext()) {
        Object[] par = (Object[]) iter.next();
        Byte depar = (Byte) par[0];
        Double media = (Double) par[1];
        Long cuenta = (Long) par[2];
        System.out.printf("Dep: %d, Media: %.2f, N° emp: %d %n", depar, media, cuenta);
}
System.out.println("================================================");
```

```
================================================
Salario medio: 2975,71
Número de empleados: 14
Dep: 10, Media: 3491,67, N° emp: 3
Dep: 20, Media: 3674,00, N° emp: 5
Dep: 30, Media: 2135,83, N° emp: 6
================================================
```

Actividad 3.5. Ejemplo 11. UsoClaseTotales

Además de las clases mapeadas con tablas de la base de datos, podemos incluir en el proyecto otras clases que no están mapeadas con tablas, pero que representan datos que hemos obtenido de la base de datos a través de una consulta. Este sería el caso si a partir de las tablas empleados y departamentos quiero obtener una consulta en la que aparezcan en nombre del departamento, su número, el número de empleados y el salario medio.

Como los datos de esta consulta no están asociados a ninguna clase, puedo crear una y utilizarla sin necesidad de mapearla. Cada fila devolverá un objeto de esa clase. Por ejemplo, creamos la clase Totales con 4 atributos: número, cuenta, media y nombre, y los constructores getter y setter correspondientes. La clase Totales sería:

```
public class Totales {
private Long cuenta; //numero empleados
private Byte numero; //numero departamento
private Double media;//media salario
private String nombre;//nombre dep

public Totales( Byte numero,  Long cuenta,
                Double media,  String nombre ) {
        this.cuenta = cuenta;
        this.media = media;
        this.nombre = nombre;
        this.numero = numero;
}

public Totales(){}

public Long getCuenta() {return this.cuenta;}
public void setCuenta( Long cuenta) {this.cuenta = cuenta;}

public Byte getNumero() {return this.numero;}
public void setNumero( Byte numero) {this.numero = numero;}

public Double getMedia() {return this.media;}
public void setMedia(final Double media) {this.media = media;}

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

Y ejemplos de uso de esta clase en las consultas serían los siguientes:

```
String hql="select new ud3_hibernate1.Totales(" +
                    "de.deptNo,  count(em.empNo), " +
                    "coalesce(avg(em.salario),0), "
                    + "de.dnombre" +
                    ")" +
                    " from Departamentos as de, Empleados as em" +
                    " where de.deptNo=em.departamentos.deptNo" +
                    " group by de.deptNo,de.dnombre" ;

hql=("select new ud3_hibernate1.Totales(" +
                    "em.departamentos.deptNo, "
                    + "count(em.empNo)," +
                    "coalesce(avg(em.salario),0), "
                    + "em.departamentos.dnombre" +
                    ")" +
                    " from Empleados as em " +
                    " group by em.departamentos.deptNo,"
                    + "em.departamentos.dnombre" );

hql= "select new ud3_hibernate1.Totales(" +
                   " d.deptNo,  count(e.empNo), coalesce(avg(e.salario),0) , "+
                   " d.dnombre ) "+
                   " from Empleados as e right join  e.departamentos as d "+
                   " group by  d.deptNo, d.dnombre ";

        Query cons = session.createQuery(hql);
        Iterator q = cons.iterate();
        while (q.hasNext()) {
           Totales tot =(Totales) q.next();
           System.out.printf(
                   "Numero Dep: %d, Nombre: %s, Salario medio: %.2f, N emple: %d%n",
                   tot.getNumero(), tot.getNombre(),tot.getMedia(),tot.getCuenta());
        }
```

Que obtendrían el siguiente resultado:

```
Numero Dep: 10, Nombre: CONTABILIDAD, Salario medio: 3491,67, N emple: 3
Numero Dep: 20, Nombre: INVESTIGACIÓN, Salario medio: 3674,00, N emple: 5
Numero Dep: 30, Nombre: VENTAS, Salario medio: 2135,83, N emple: 6
Numero Dep: 40, Nombre: PRODUCCIÓN, Salario medio: 0,00, N emple: 0
```

Actividad 3.5. Ejemplo 12. UsoUpdate

Tanto el UPDATE como el DELETE se realizan con executeUpdate(). En este ejemplo se modifica el salario de GIL y eliminamos los empleados del departamento 20.

La sentencia tx.commit() valida la transacción y tx.rollback() deshace la transacción.

```
// Modificamos el salario de GIL
String hqlModif = "update Empleados set salario = :nuevoSal where apellido = :ape";
Query q1 = session.createQuery(hqlModif);
q1.setParameter("nuevoSal", (float) 2500.34);
q1.setString("ape", "GIL");
int filasModif = q1.executeUpdate();
System.out.printf("FILAS MODIFICADAS: %d%n", filasModif);

// Eliminamos los empleados del departamento 20
String hqlDel = "delete Empleados e where e.departamentos.deptNo = :dep";
Query q = session.createQuery(hqlDel);
q.setInteger("dep", 20);
int filasDel = q.executeUpdate();
System.out.printf("FILAS ELIMINADAS: %d%n", filasDel);

//tx.rollback(); // Deshace la transacci n
tx.commit(); // Valida la transacci n
```

Actividad 3.5. Ejemplo 13. UsoINSERT

En el ejemplo se insertan en la tabla departamentos los datos de nuevos departamentos que están en otra tabla. Para hacerlo, primero creamos la tabla en la BD usando el script SQL creaTablaNuevos.sql

```sql
CREATE TABLE nuevos (
    dept_no TINYINT(2) NOT NULL PRIMARY KEY,
    dnombre VARCHAR(15),
    loc VARCHAR(15)
) ENGINE = InnoDB;

INSERT INTO nuevos VALUES (51, 'PERSONAL', 'MADRID');
INSERT INTO nuevos VALUES (52, 'NOMINAS', 'TOLEDO');
INSERT INTO nuevos VALUES (53, 'OCIO', 'BARCELONA');
```

	dept_no	dnombre	loc
Editar Copiar Borrar	51	PERSONAL	MADRID
Editar Copiar Borrar	52	NOMINAS	TOLEDO
Editar Copiar Borrar	53	OCIO	BARCELONA

Después tengo que modificar el fichero hibernate.reveng.xml para mapear esta nueva tabla.

```
<hibernate-reverse-engineering>
  <schema-selection match-catalog="ud2_xampp"/>
  <table-filter match-name="departamentos"/>
  <table-filter match-name="empleados"/>
  <table-filter match-name="nuevos"/>
</hibernate-reverse-engineering>
```

Por último, hay que crear la clase nuevos y el fichero nuevos.hbm.xml utilizando los asistentes. Hecho esto, ya se puede ejecutar el ejemplo que inserta los nuevos departamentos.

```
public class EjemploUsoINSERT {
        public static void main(String[] args) {
                SessionFactory sesion = HibernateUtil.getSessionFactory();
                Session session = sesion.openSession();
                Transaction tx = session.beginTransaction();

                // Insertamos los departamentos de la tabla NUEVOS, la tabla tiene
                // que estar mapeada en nuestro proyecto
                String hqlInsert = "insert into Departamentos (deptNo, dnombre, loc) " +
                                   " select n.deptNo, n.dnombre, n.loc from Nuevos n";
                Query cons=session.createQuery( hqlInsert );
                int filascreadas = cons.executeUpdate();
                tx.commit(); // valida la transacción
                System.out.printf("FILAS INSERTADAS: %d%n",filascreadas);

                session.close();
                System.exit(0);
        }
}
```

Actividad 3.6. Ejercicio – Consulta con el método createQuery()

Realiza una consulta con createQuery() para obtener los datos del departamento 20 y visualiza también el apellido de sus empleados.

```
package ejemplo_hibernate1;

import java.util.Iterator;
import java.util.Set;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
// Importa las clases de mapean las entidades que están en otro paquete de la aplicación
import ud3_hibernate1.Departamentos;
import ud3_hibernate1.Empleados;
import ud3_hibernate1.HibernateUtil;

public class Actividad3_6 {
        public static void main(String[] args) {
                SessionFactory sesion = HibernateUtil.getSessionFactory();
                Session session = sesion.openSession();

                Departamentos depart = new Departamentos();
                String hql="from Departamentos as dep  where dep.deptNo = 20";
                Query q= session.createQuery(hql);
                depart=(Departamentos) q.uniqueResult();
                System.out.printf("%d, %s, %s %n",depart.getDeptNo(), depart.getLoc(), depart.getDnombre());

                Set<Empleados> listaemple = depart.getEmpleadoses();//obtenemos empleados
                Iterator <Empleados> it = listaemple.iterator();
                System.out.printf("Número de empleados: %d%n",  listaemple.size() );
                while(it.hasNext()) {
                        Empleados emple = it.next();
                        System.out.printf("%s, %.2f%n",emple.getApellido(), emple.getSalario());
                 }
                System.out.println("============================" );
                session.close();
                System.exit(0);
        }
}  // fin clase Actividad3_6
```