ACCESO A DATOS

UNIDAD 04 – CORRESPONDENCIA OBJETO-RELACIONAL

INDICE

4.1 – Correspondencia Objeto-Relacional

4.1 – CORRESPONDENCIA OBJETO-RELACIONAL

- Los lenguajes OO muy extendidos desde los 90.
- **Java** muy **popular** en APPS de gestión empresarial y **uso de BBDD relacionales.**
- El uso de POO no implica usar las características OO.

4.1 – CORRESPONDENCIA OBJETO-RELACIONAL

- Cada vez más se planteó la necesidad de almacenar Objetos en BBDD.
- Este almacenamiento planteaba una serie de problemas.
- Surgieron varios planteamientos.
  - BBDD de objetos.
  - BBDD Objeto-Relacionales
  - ORM o correspondencia Objeto-Relacional

4.1 – CORRESPONDENCIA OBJETO-RELACIONAL

- **BBDD de objetos**
  - Almacenar directamente objetos.
  - Inicialmente la más lógica.
  - Presenta varios problemas:
    - Falta de modelo formal ampliamente aceptado.
    - Falta de estándares aceptados (a pesar del ODMG)

4.1 – CORRESPONDENCIA OBJETO-RELACIONAL

- **BBDD Objeto-Relacionales**
  - BBDD relacionales con capacidad para gestionar objetos.
  - En SQL:99 se introduce tipos definidos por usuario entre ellos para objetos.
  - Solución de compromiso.
  - Resuelven solo parcialmente el problema.

4.1 – CORRESPONDENCIA OBJETO-RELACIONAL

- **ORM o correspondencia Objeto-Relacional**
  - Establecer correspondencia entre clases POO y tablas de BBDD relacional.
  - Uso de mecanismos para registrar los cambios en los objetos en la BBDD y viceversa.
  - Permite la persistencia en BBDD puramente relacionales.
  - Hibernate es la más importante y hace uso de JDBC en Java.

4.2 – ORM con Hibernate

4.2 – ORM CON HIBERNATE

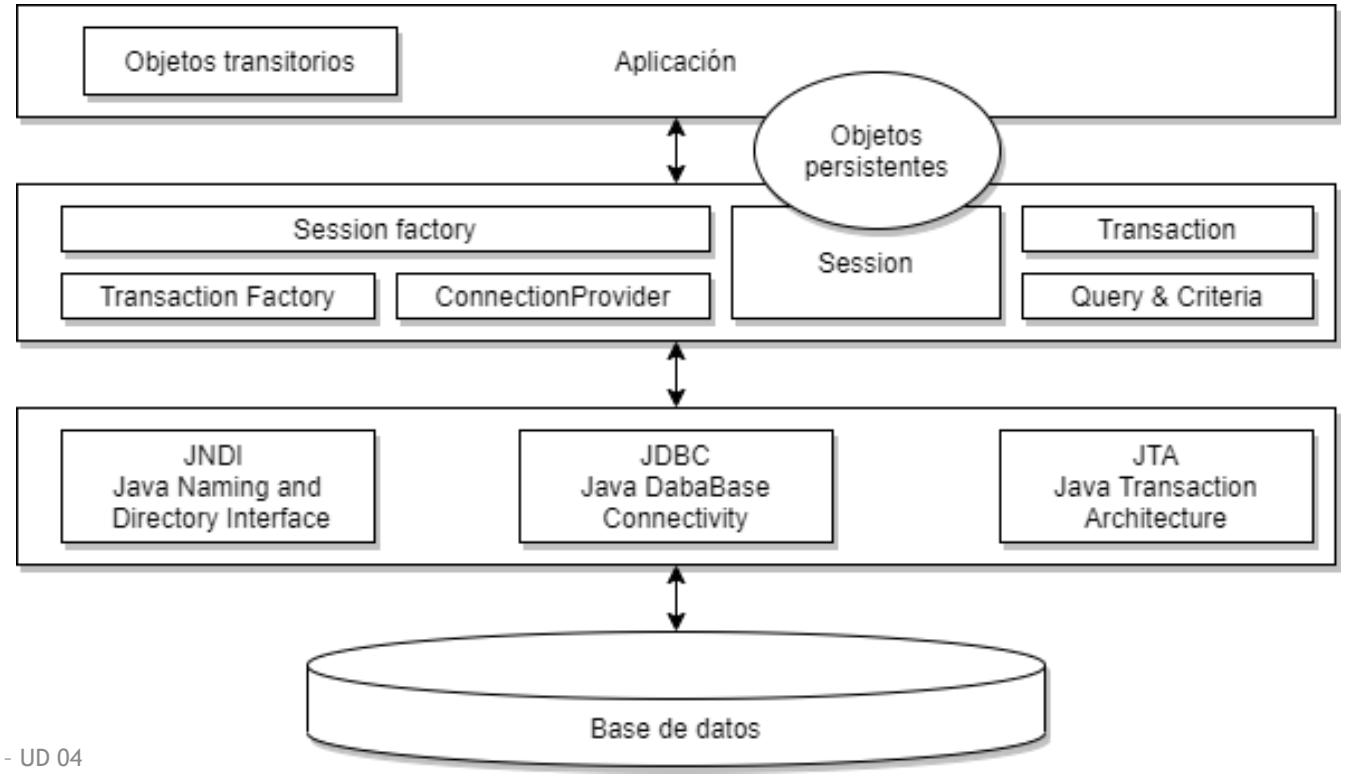
- Hibernate es un Framework de ORM para Java distribuido bajo licencia LGPL.
- Tiene soporte para *transacciones* y usa técnicas para mejorar el rendimiento (pool de conexiones, caching, batching).
- Hibernate 3 se convirtió en implementación certificada de JPA (API estándar de Java para persistencia de objetos).
- Hibernate proporciona 2 API, la suya y JPA.

4.2 – ORM CON HIBERNATE

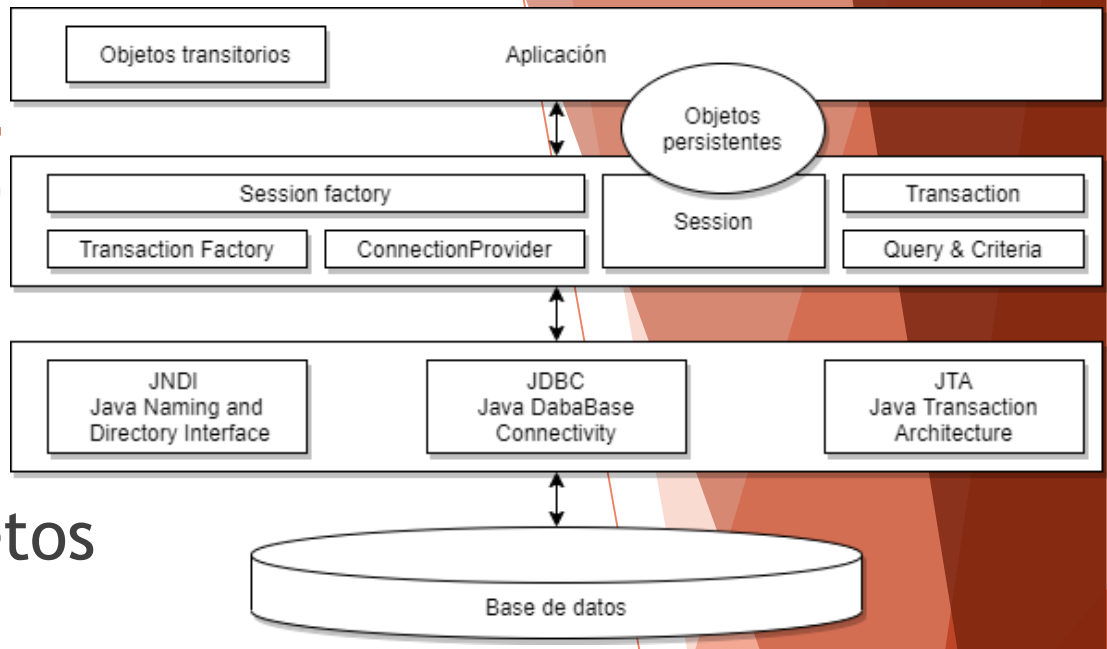
- Hibernate dispone de lenguaje propio para el manejo de objetos persistentes: **HQL**.
  - SQL opera sobre filas y columnas de tablas, mientras que HQL sobre conjuntos de objetos persistentes y sus atributos.
- El lenguaje estándar de JPA para manejo de objetos persistentes, JPQL, es un subconjunto de HQL.

4.2 – ORM CON HIBERNATE

- Arquitectura de Hibernate.

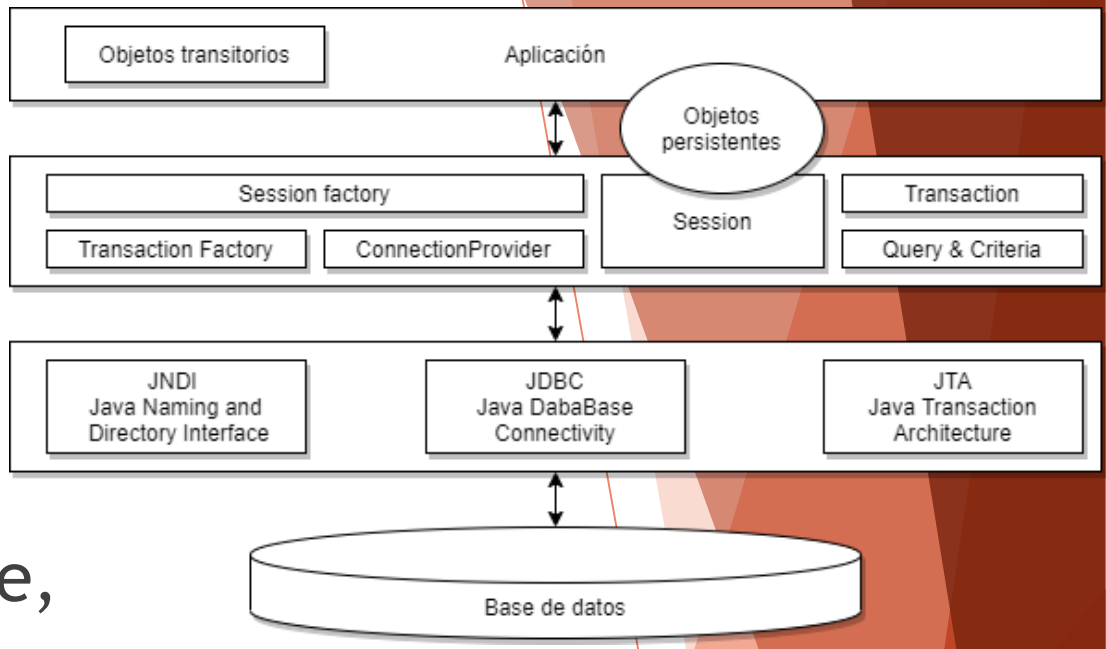


4.2 – ORM CON HIBERNATE



- Una aplicación con Hibernate usa objetos transitorios y persistentes.
- Los **persistentes** siempre asociados a una sesión (**Session**) creada por una SessionFactory.
- Se pueden crear objetos transitorios y convertirlos en persistentes para almacenarlos en BD.

4.2 – ORM CON HIBERNATE



- Se puede recuperar objetos persistente, realizar cambios y almacenarlos.
- Se puede agrupar operaciones en transacciones (Transaction).
- Puede utilizar sentencias HQL con la clase Query.

4.2 – ORM CON HIBERNATE

HIBERNATE

- La correspondencia entre objetos java y esquema E-R es compleja si se tienen en cuenta todos los posibles casos.
- Para empezar de forma sencilla:
  - Cada clase = una tabla.

Clase Java (Empleado)
Nombre
apellido
edad
salario

HIBERNATE

Base de Datos (Tabla Empleado)

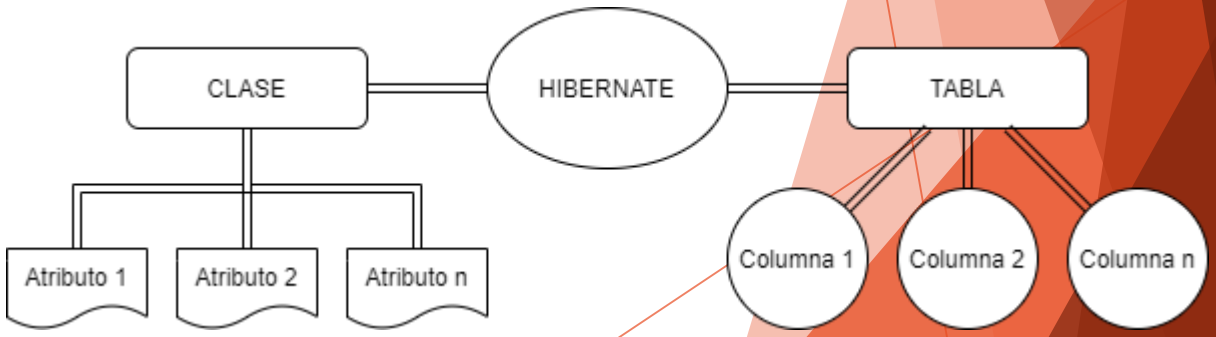
nombre	apellido	edad	salario
Juan	Pérez	18	800.00
Pedro	Guardia	35	100.80
María	López	21	650.00

4.2 – ORM CON HIBERNATE

- Hibernate permite ORM cuando se parte de una BD ya existente o para clases ya creadas (no solo si creamos la BD partiendo de cero).
- Hibernate impone condiciones:
  - Tablas deben tener clave primaria.(PK)
  - La PK no puede cambiar.
  - Las clases deben implementar la interfaz Serializable(*implements Serializable*).
  - *Las clases tienen unas convenciones a cumplir (Getters* y *Setters…*)

4.2 – ORM CON HIBERNATE

- La correspondencia mediante hbm o JPA.
- Hbm (Hibernate mapping):
  - Archivo xml relacionando clases-atributos con tablas-columnas.
- JPA (Java Persistence API):
  - Mediante anotaciones en las clases.



4.2 – ORM CON HIBERNATE

- Correspondencia a partir de tablas.
- Veremos un ejemplo de programa que hace uso de ORM con Hibernate.
- Los nombres de las tablas en singular.

4.2 – ORM CON HIBERNATE

- Pasos en nuestro ejemplo.
  - 1) Creación de la BD.
  - 2) Conexión a la BD desde nuestro IDE (NetBeans)
  - 3) Creación del proyecto Java.
  - 4) Configuración hibernate (hibernate.cfg.xml)
  - 5) Fichero ingeniería inversa (hibernate.reveng.xml)
  - 6) Ficheros POJO
  - 7) HibernateUtil.java

4.2 – ORM CON HIBERNATE

- 1) Creación de la BD.

Creación del E-R en MySQL	
create database proyecto_orm;, **use** proyecto_orm; **create table** sede(id_sede integer auto_increment not null, nom_sede char(20) not null, *primary key*(id_sede));	**create table** departamento(id_depto integer auto_increment not null, nom_depto char(32) not null, id_sede integer not null, *primary key*(id_depto), *foreign key* fk_depto_sede(id_sede) references sede(id_sede));
create table empleado(dni char(9) not null, nom_emp char(40) not null, id_depto integer not null, *primary key*(dni), *foreign key* fk_empleado_depto(id_depto) references departamento(id_depto));	**create table** empleado_datos_prof(dni char(9) not null, categoria char(2) not null, sueldo_bruto_anual decimal(8,2), *primary key*(dni), *foreign key* fk_empleado_datosprof_empl(dni) references empleado(dni));

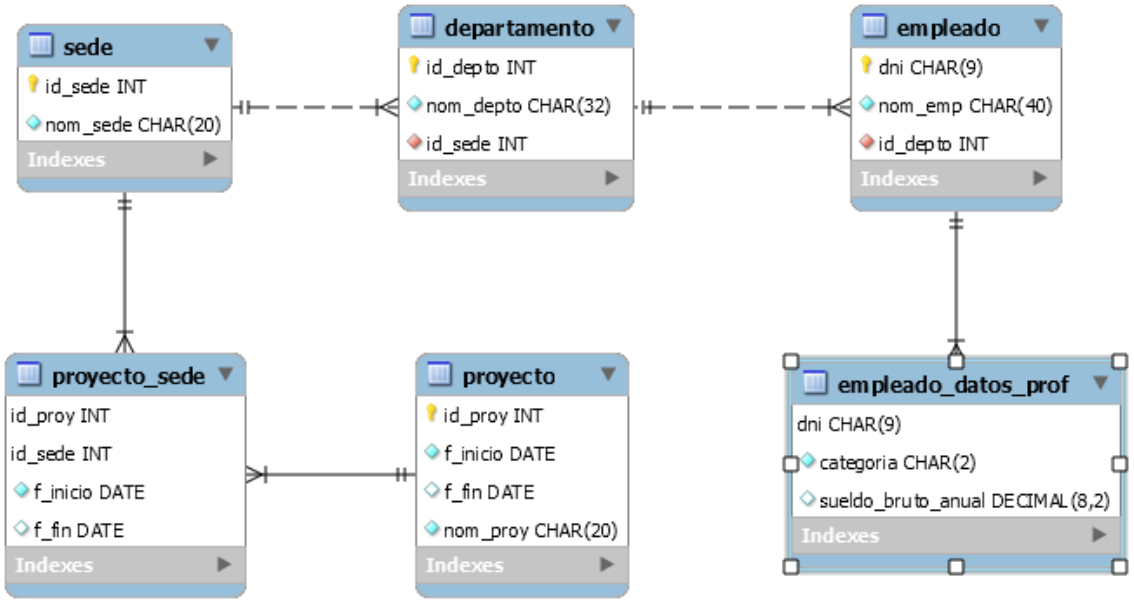
4.2 – ORM CON HIBERNATE

- 1) Creación de la BD.

Creación del E-R en MySQL (II)	
create table proyecto (id_proy integer auto_increment not null, f_inicio date not null, f_fin date,nom_proy char(20) not null, *primary key*(id_proy));	**create table** proyecto_sede (id_proy integer not null, id_sede integer not null, f_inicio date not null, f_fin date, *primary key*(id_proy, id_sede), *foreign key* fk_proysede_proy (id_proy) references proyecto(id_proy), *foreign key* fk_proysede_sede (id_sede) references sede(id_sede));
CREATE USER '**userORM_ad**'@'localhost' IDENTIFIED BY '**userORM_ad**'; **GRANT** SELECT,INSERT,UPDATE,DELETE,CREATE,DROP,EXECUTEON proyecto_orm.* TO '**userORM_ad**'@'localhost';	

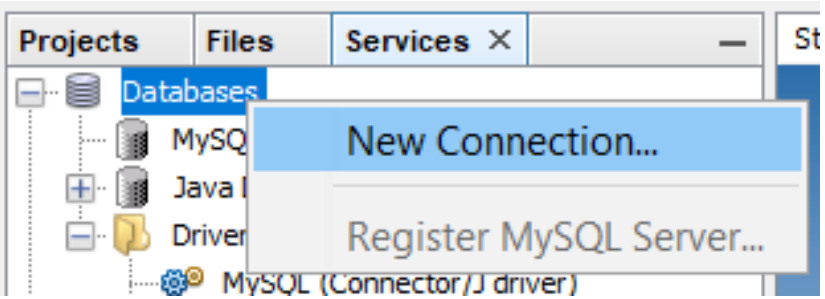
4.2 – ORM CON HIBERNATE

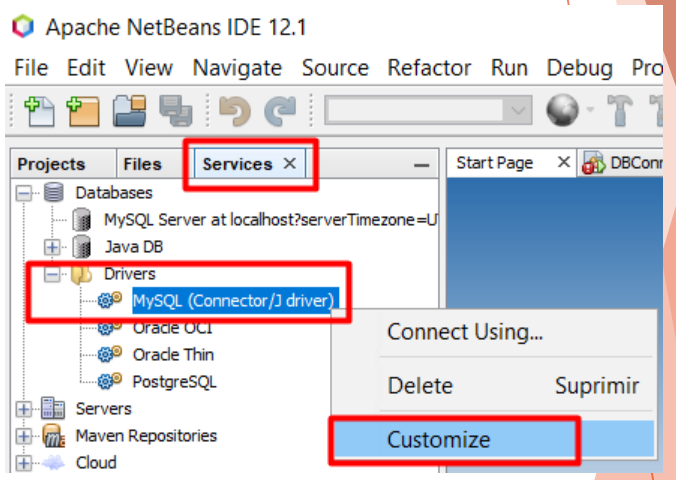
- 1) Creación de la BD. (database > Reverse Engineer)



4.2 – ORM CON HIBERNATE

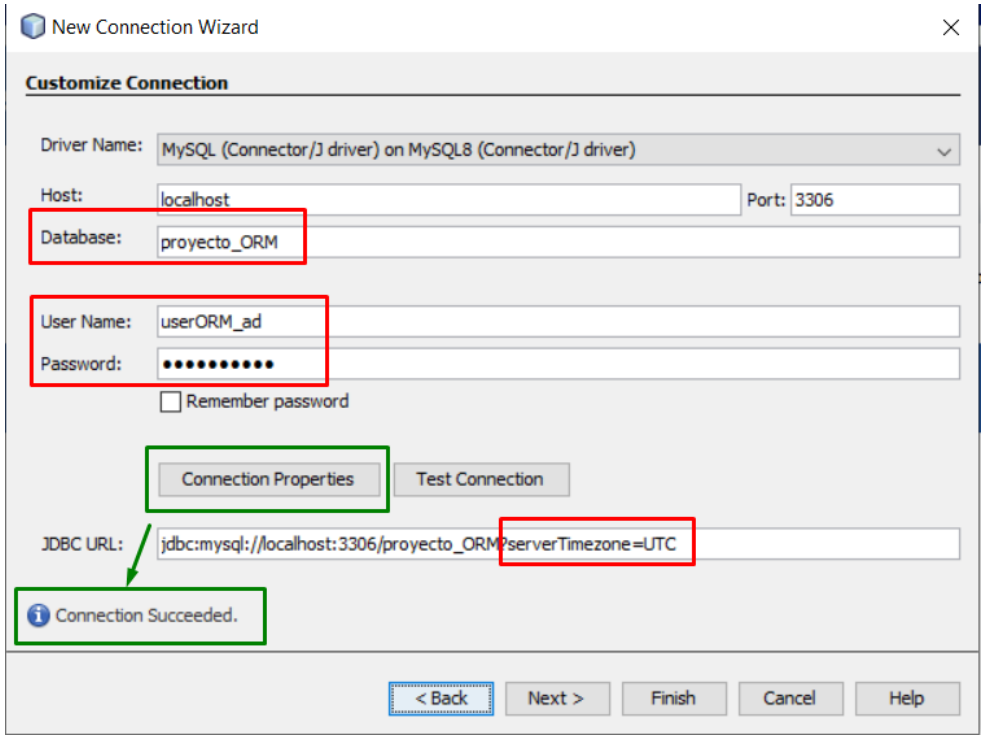
- 2) Conexión a la BD desde nuestro IDE
  - Confirmar driver instalado en IDE o instalar nuevo
  - Configurar conexión a la BD **proyecto_orm**





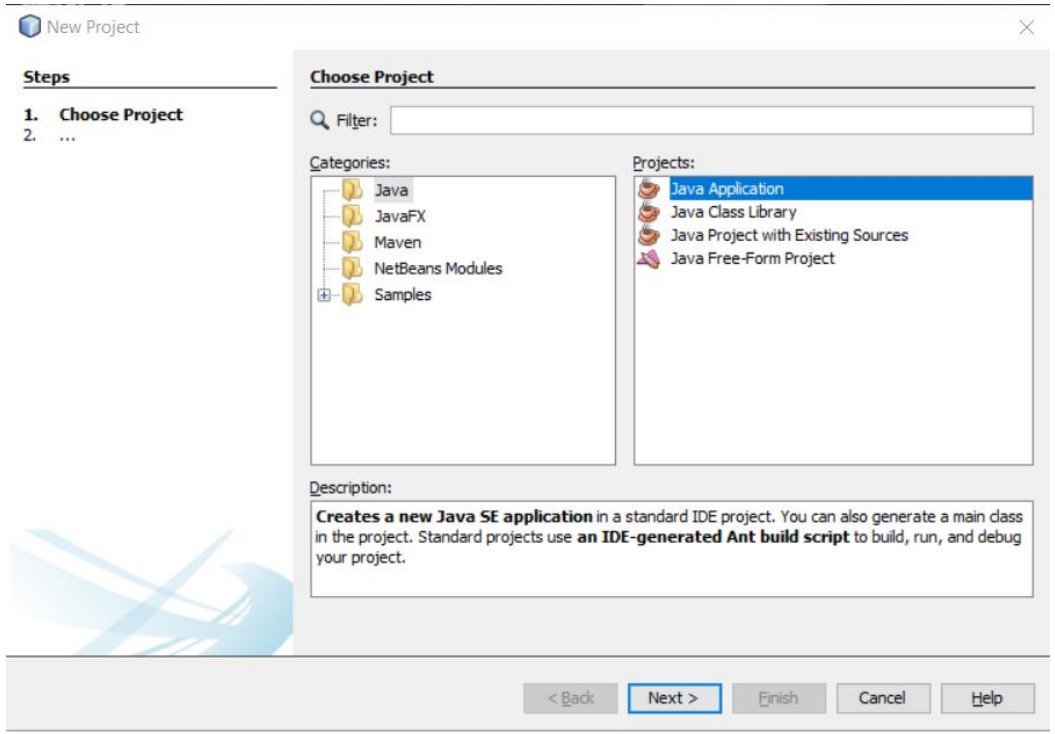
4.2 – ORM CON HIBERNATE

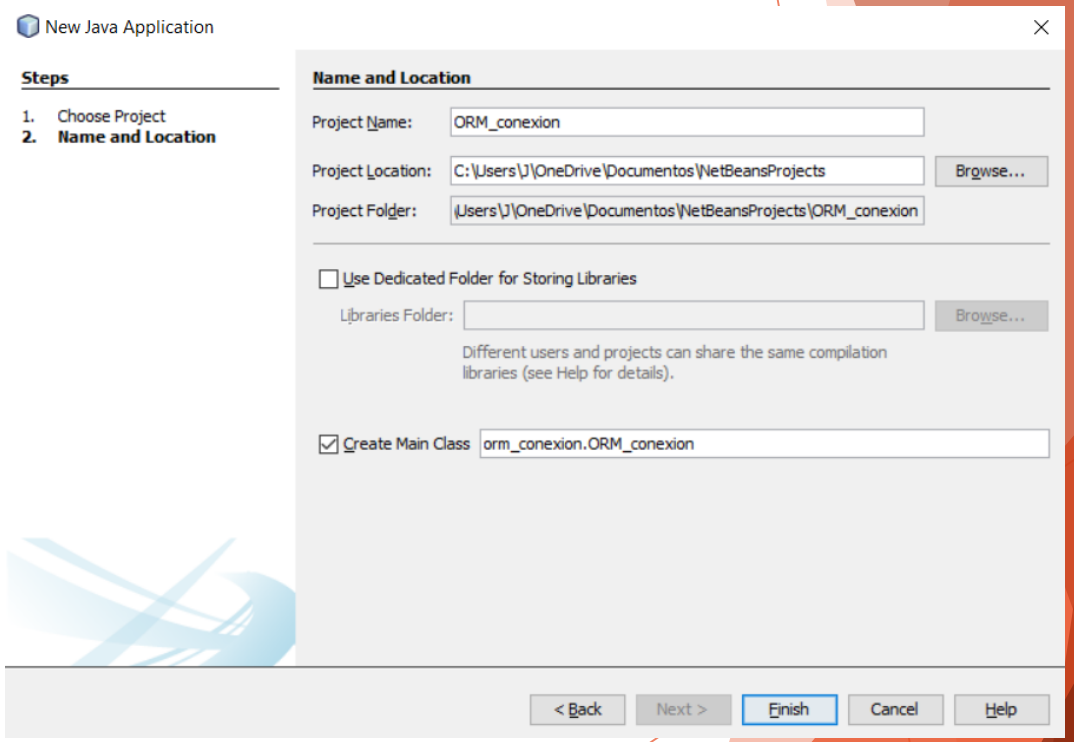
- 2) Conexión a la BD desde nuestro IDE
  - Confirmar conexión a la BD **proyecto_orm**



4.2 – ORM CON HIBERNATE

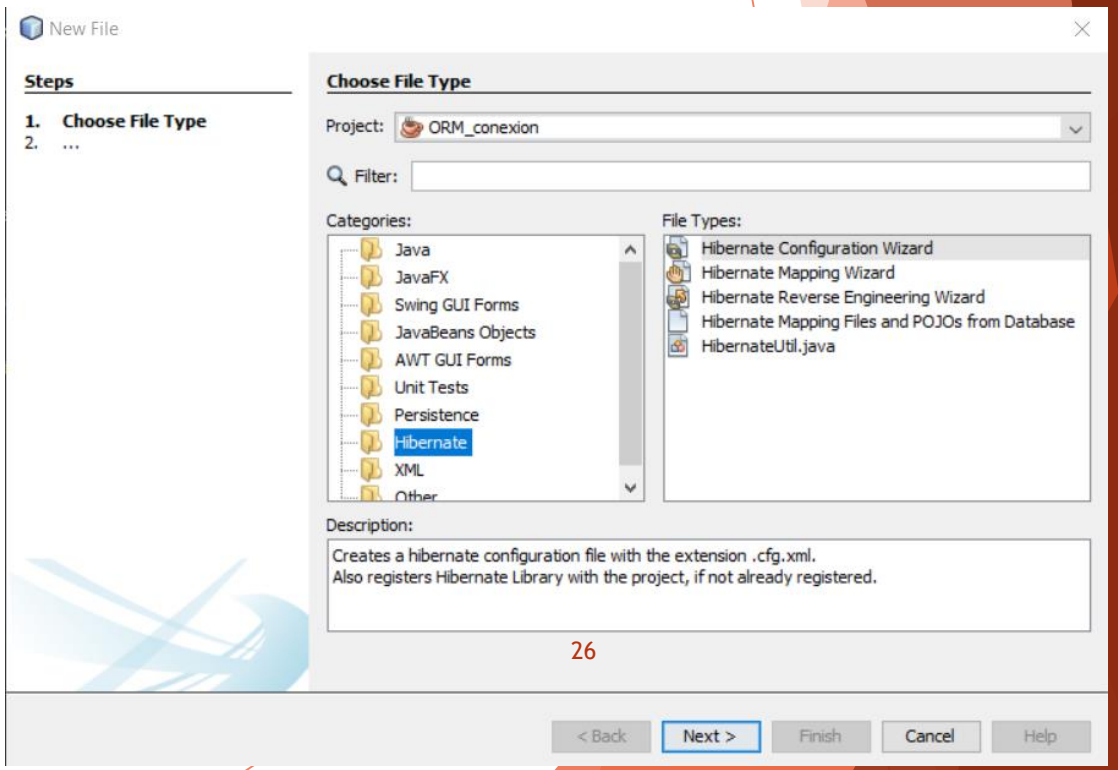
- 3) Creación del proyecto Java





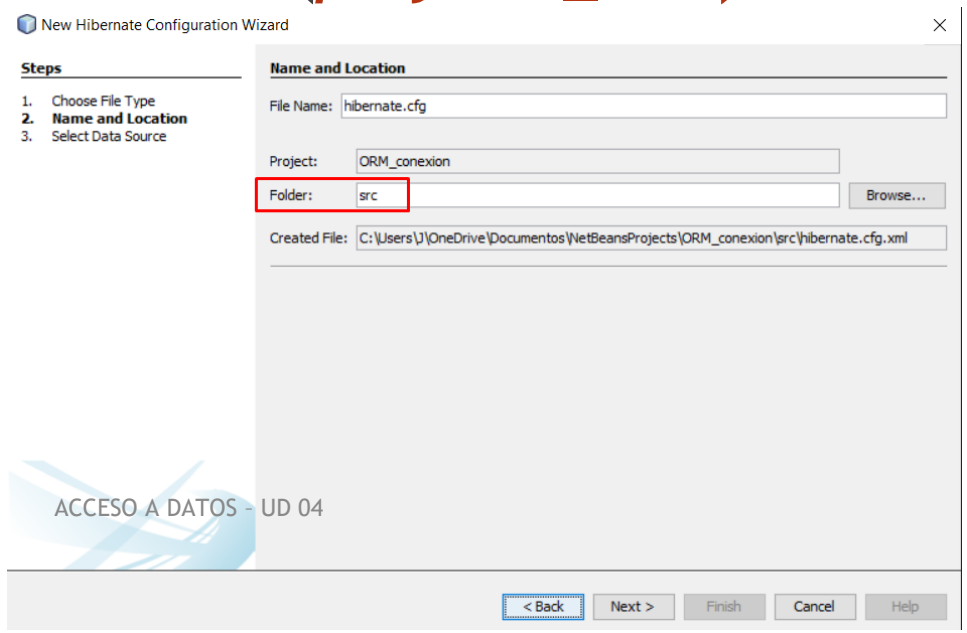
4.2 – ORM CON HIBERNATE

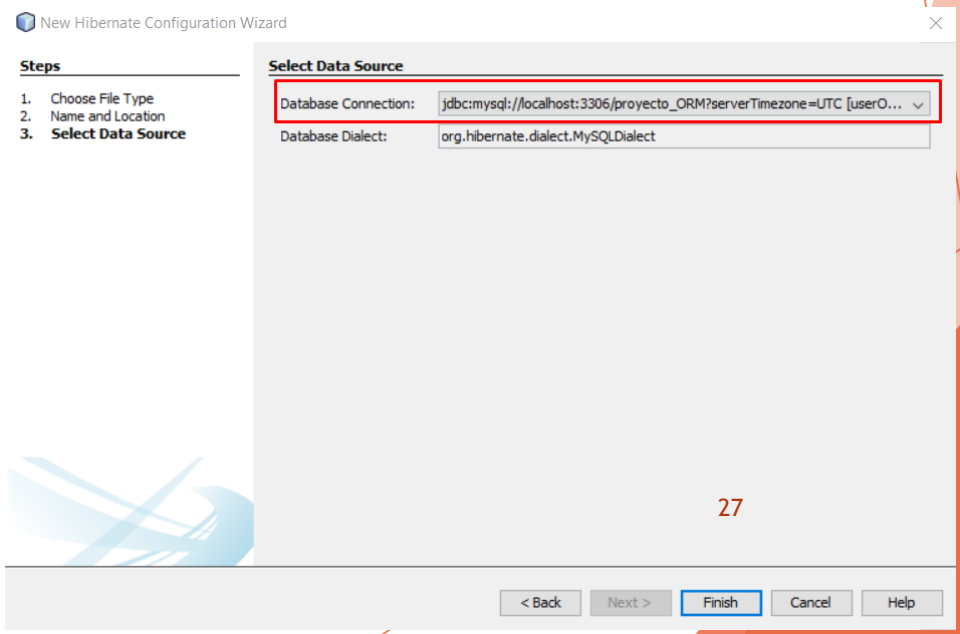
- 4) Configuración Hibernate (hibernate.cfg.xml)
  - Se trata de crear el fichero hibernate.cfg.xml con la información de conexión a la BD.
  - Sobre proyecto, botón derecho:
    - "New…">"Other…">"Hibernate"
  - **Hibernate Configuration Wizard**



4.2 – ORM CON HIBERNATE

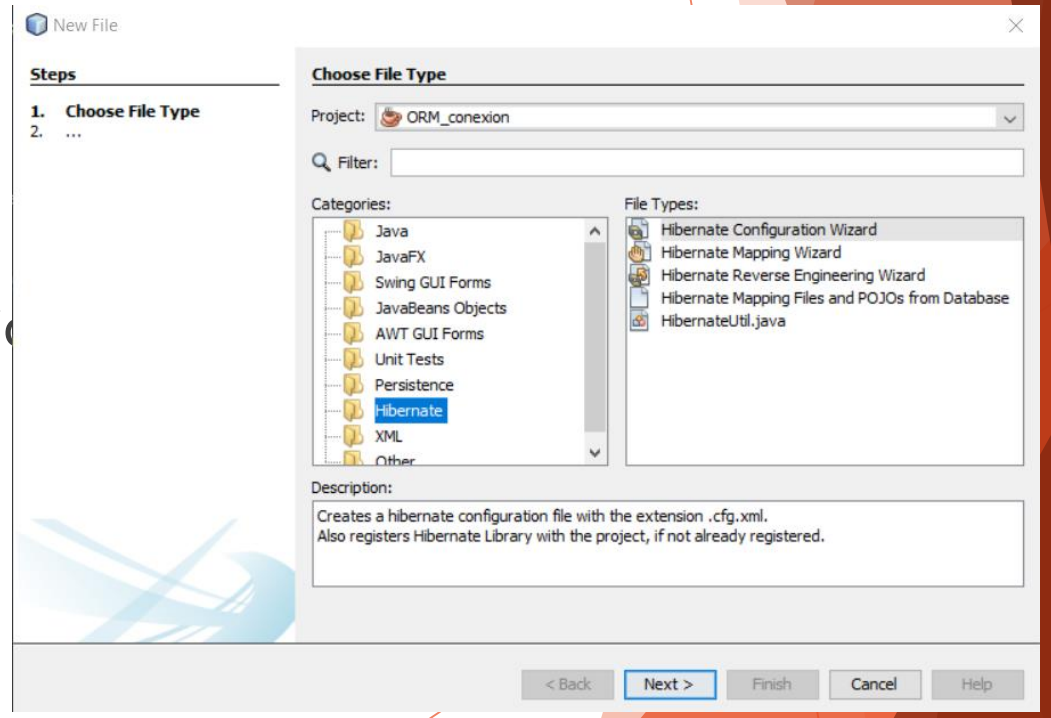
- 4) Configuración Hibernate (hibernate.cfg.xml)
  - **Name and Location:** ubicación del fichero (***src)***
  - **Select Data Source:** conexión que se usará para la BD (***proyecto_ORM)***





4.2 – ORM CON HIBERNATE

- 4) Configuración Hibernate (hibernate.cfg.xml)
  - Sobre proyecto, botón derecho:
    - "New…">"Other…">"Hibernate"
  - **Hibernate Configuration Wizard**
    - **Name and Location:** ubicación del fi
    - **Select Data Source:** ***proyecto_ORM***



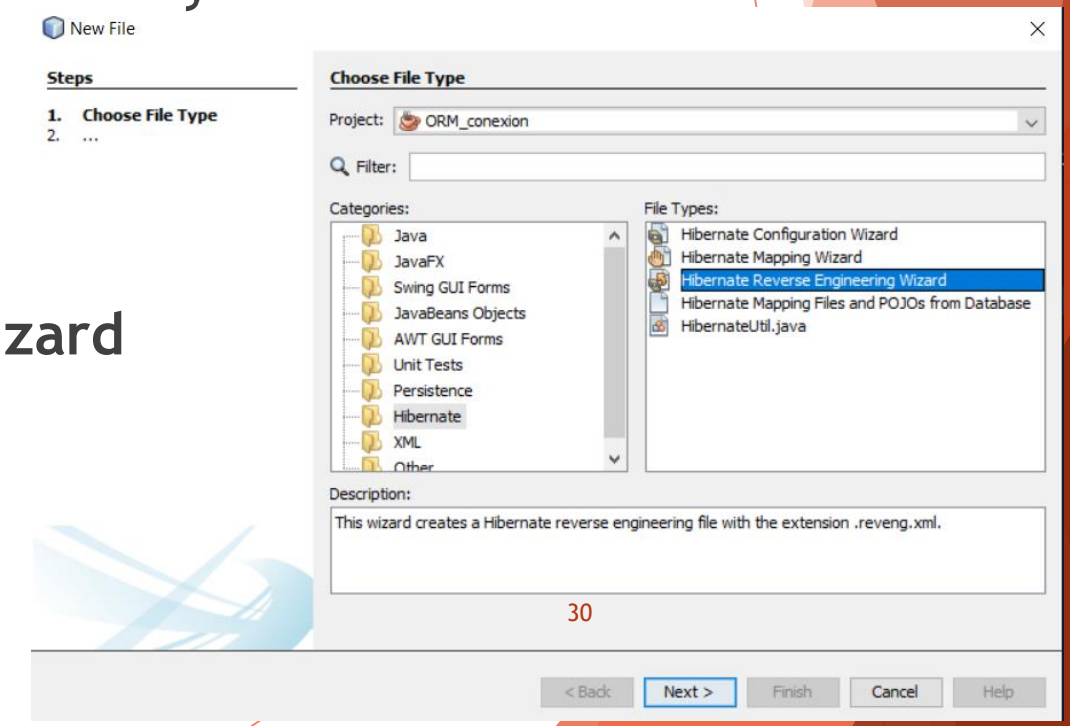
4.2 – ORM CON HIBERNATE

- 4) Configuración Hibernate (hibernate.cfg.xml)
  - Puede ser necesario editar el contenido del fichero para que quede como se muestra a continuación:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/proyecto_ORM?serverTimezone=UTC</property>
    <property name="hibernate.connection.username">userORM_ad</property>
    <property name="hibernate.connection.password">userORM_ad</property>
  </session-factory>
</hibernate-configuration>
```

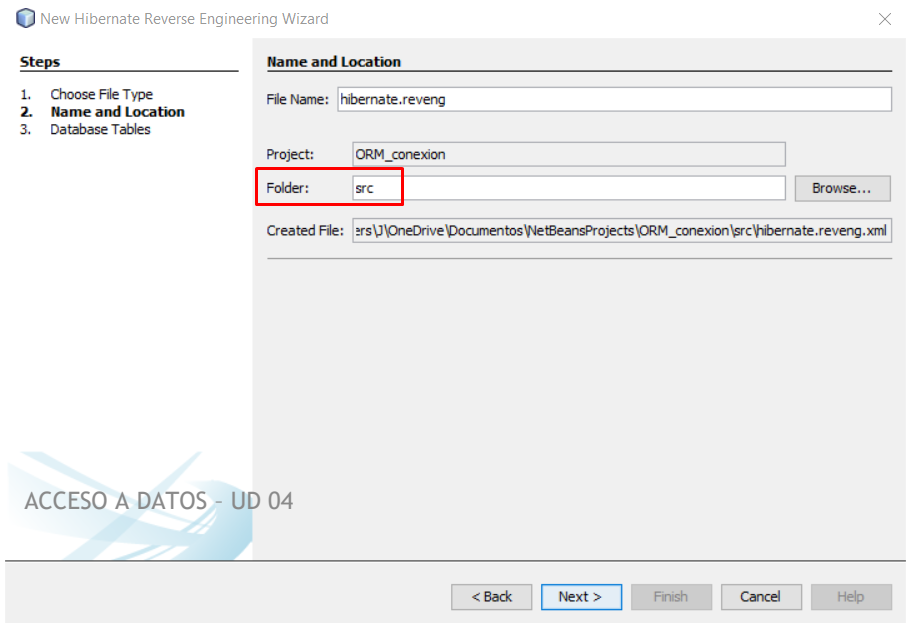
4.2 – ORM CON HIBERNATE

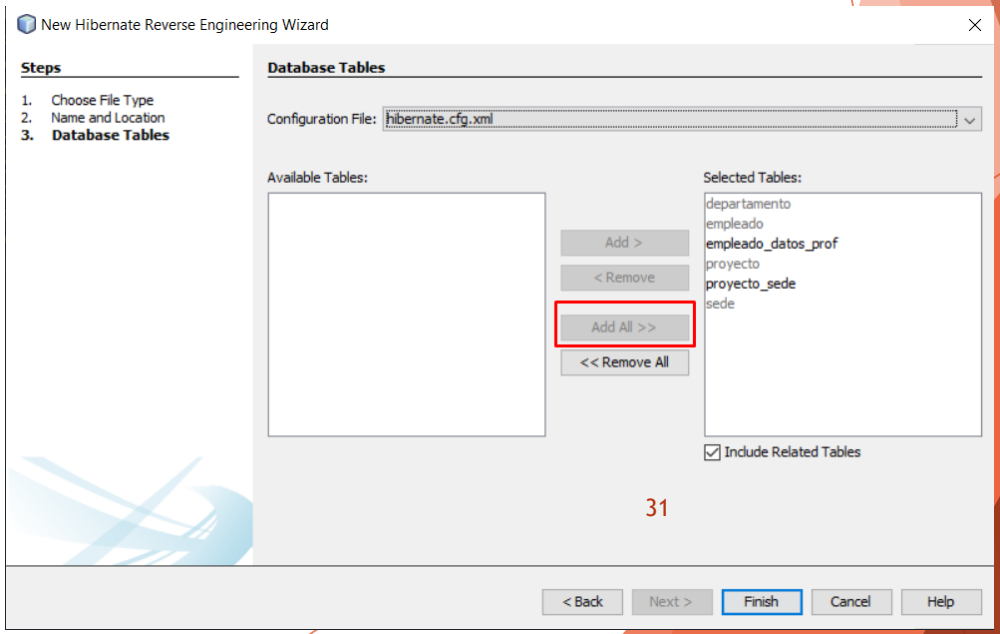
- 5) Fichero ingeniería inversa (hibernate.reveng.xml)
  - Se trata de crear el fichero hibernate.reveng.xml con la información específica de para qué clases y tablas se va a establecer la correspondencia.
  - Sobre proyecto, botón derecho:
    - "New…">"Other…">"Hibernate"
  - **Hibernate Reverse Engineering Wizard**



4.2 – ORM CON HIBERNATE

- 5) Fichero ingeniería inversa (hibernate.reveng.xml)
  - **Name and Location:** ubicación del fichero (***src)***
  - **Select Data Source:** conexión que se usará para la BD (***proyecto_ORM)***





4.2 – ORM CON HIBERNATE

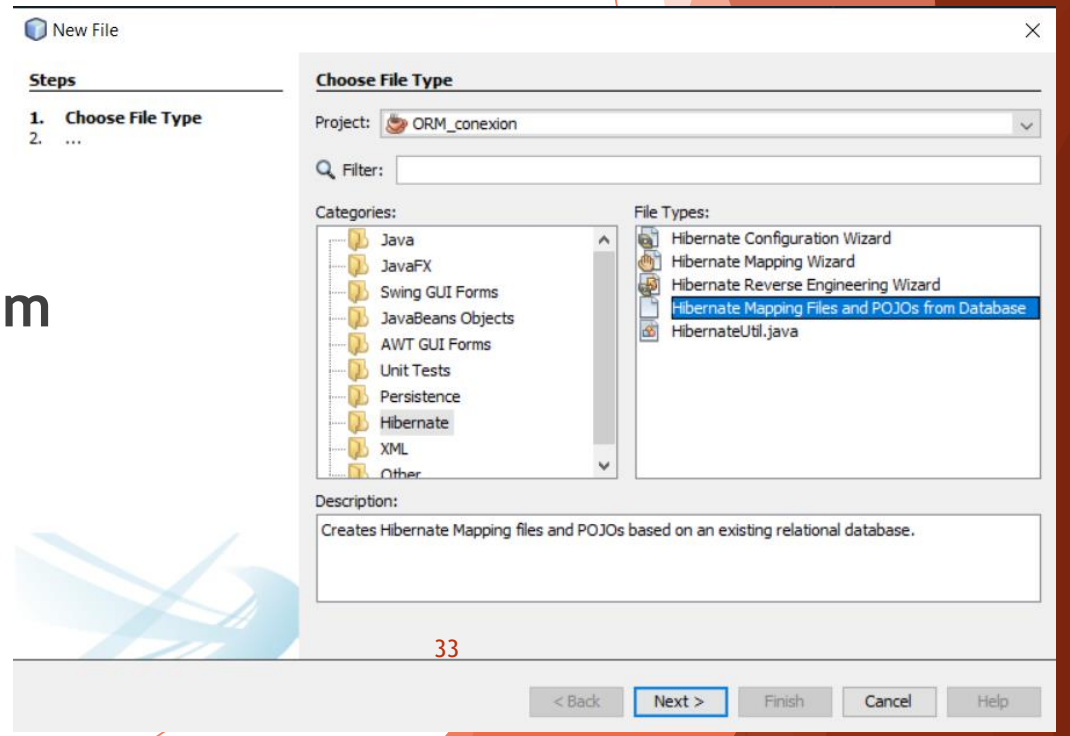
- 5) Fichero ingeniería inversa (hibernate.reveng.xml)
  - Puede ser necesario editar el contenido del fichero para que quede como se muestra a continuación:

```
<?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="proyecto_ORM"/>
  <table-filter match-name="empleado"/>
  <table-filter match-name="empleado_datos_prof"/>
  <table-filter match-name="departamento"/>
  <table-filter match-name="sede"/>
  <table-filter match-name="proyecto_sede"/>
  <table-filter match-name="proyecto"/>
</hibernate-reverse-engineering>
```

4.2 – ORM CON HIBERNATE

HIBERNATE

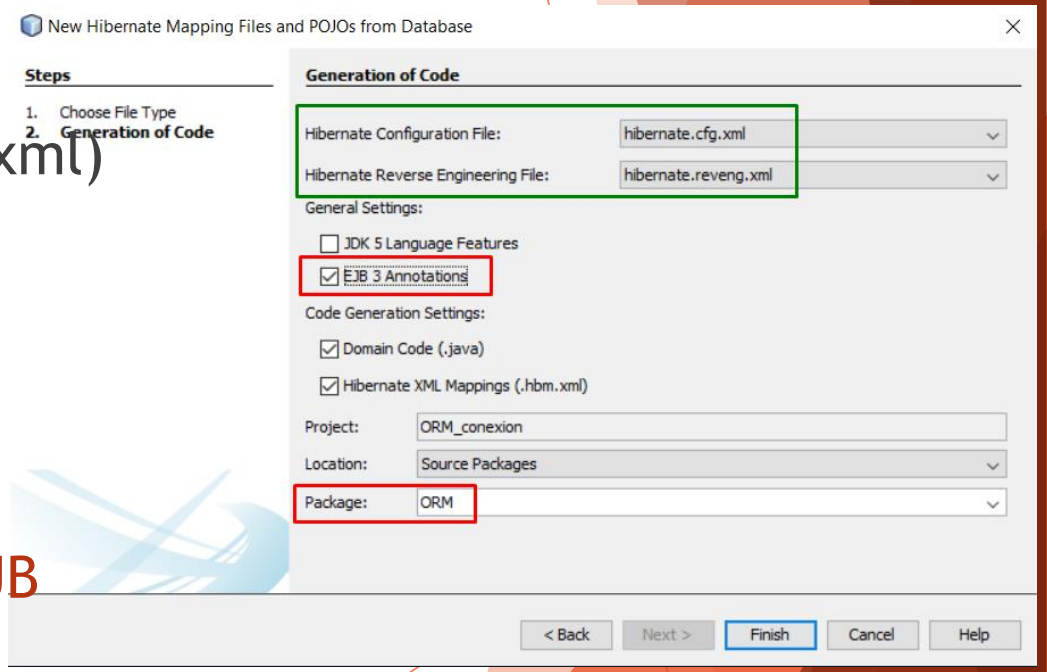
- 6) Ficheros POJO
- Clases y ficheros de correspondencia.
  - Con asistente: Sobre proyecto, botón derecho:
    - “New…”>”Other…”>”Hibernate”
  - **Hibernate Mapping Files and POJOs from Database**



4.2 – ORM CON HIBERNATE

HIBERNATE

- 6) Ficheros POJO
  - Hay que indicar:
  - **Fichero de configuración** (hibernate.cfg.xml)
  - **Fichero de ingeniería inversa** (hibernate.reveng.xml)
  - **Nombre paquete** (*ORM*)
  - Incluimos la opción de anotaciones JPA (EJB Annotations) aunque no se usarán.



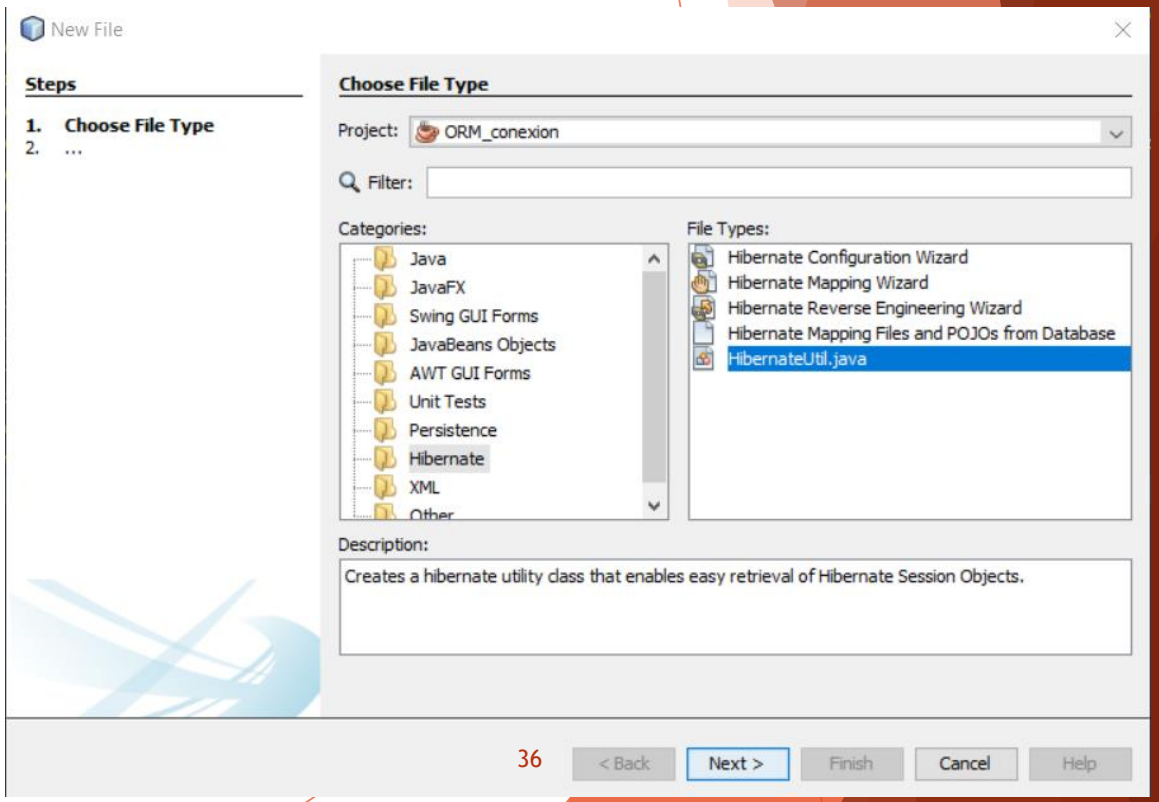
4.2 – ORM CON HIBERNATE

- 6) Ficheros POJO
  - Además de crear las clases (POJO) y ficheros de correspondencia, añade al fichero ***hibernate.cfg.xml*** las siguientes líneas para establecer la correspondencia mediante los correspondientes *ficheros hbm*.
  - Si para alguna clase se quisiera utilizar anotaciones (JPA) en lugar de ficheros hbm, se eliminaría del fichero la configuración correspondiente.

```
<mapping resource="orm_conexion/EmpleadoDatosProf.hbm.xml"/>
<mapping resource="orm_conexion/Departamento.hbm.xml"/>
<mapping resource="orm_conexion/Proyecto.hbm.xml"/>
<mapping resource="orm_conexion/ProyectoSede.hbm.xml"/>
<mapping resource="orm_conexion/Sede.hbm.xml"/>
<mapping resource="orm_conexion/Empleado.hbm.xml"/>
```

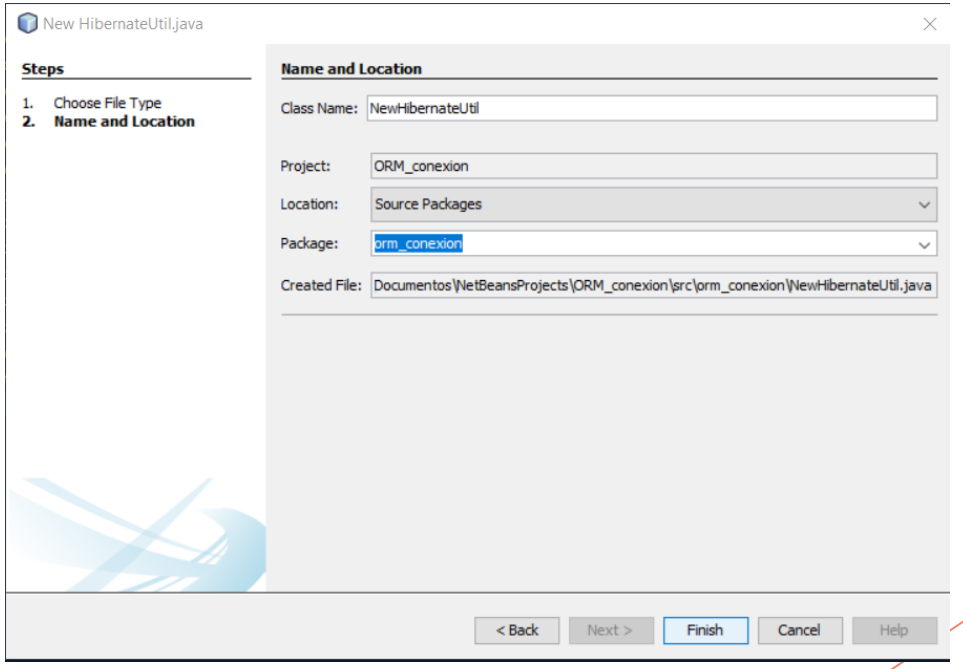
4.2 – ORM CON HIBERNATE

- 7) HibernateUtil.java
- Se trata de crear la clase ***HibernateUtil.java***, la cual facilita enormemente la inicialización de Hibernate.
  - Con asistente: Sobre proyecto, botón derecho:
    - "New…">"Other…">"Hibernate"
  - **HibernateUtil.java**



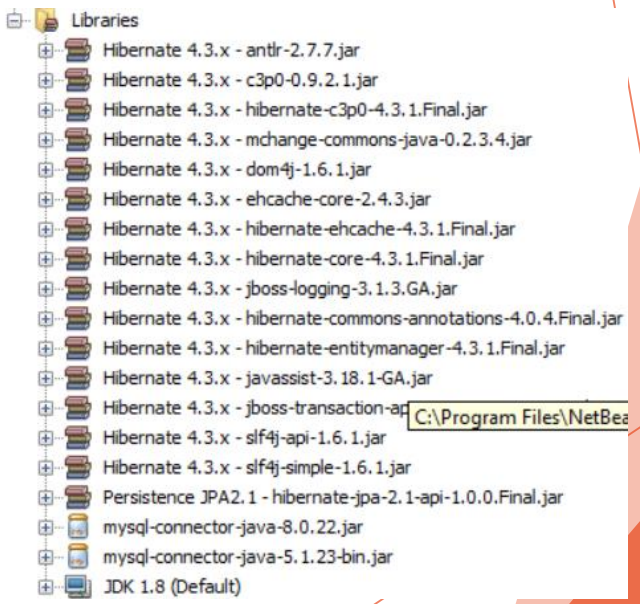
4.2 – ORM CON HIBERNATE

- 7) HibernateUtil.java
  - Solo necesita nombre del fichero y paquete en el que se quiere crear.



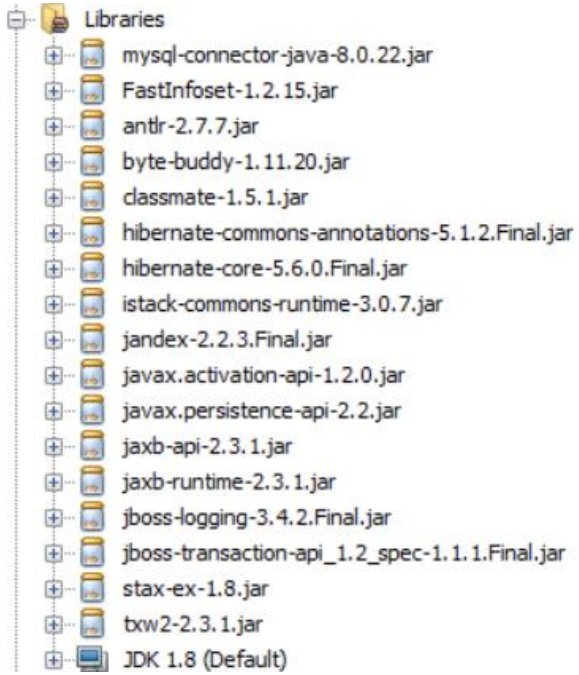
4.2 – ORM CON HIBERNATE

- A partir de ahora solo faltaría escribir el programa en el fichero ORM_conexión.java.
- Podemos comprobar los ficheros de *hibernate* en la carpeta *Libraries*.



4.2 – ORM CON HIBERNATE

- Para actualizar la versión de Hibernate
  - https://hibernate.org/orm/releases/
  - Importante revisar prerrequisitos.
  - Descargamos versión (5.6)
  - Descomprimimos
  - Actualizamos proyecto con contenido en carpeta lib*required*.



4.2 – ORM CON HIBERNATE

- Tras actualizar la versión de Hibernate, HibernateUtil.java no compila.
  - Conviene revisar los javadocs:
    - http://docs.jboss.org/hibernate/orm/5.3/javadocs
    - https://docs.jboss.org/hibernate/orm/4.3/javadocs/

```
Compiling 9 source files to C:\Users\J\OneDrive\Documentos\NetBeansProjects\ORM_conexion\build\classes
C:\Users\J\OneDrive\Documentos\NetBeansProjects\ORM_conexion\src\orm_conexion\HibernateUtil.java:8: error: cannot find symbol
import org.hibernate.cfg.AnnotationConfiguration;
  symbol:   class AnnotationConfiguration
  location: package org.hibernate.cfg
C:\Users\J\OneDrive\Documentos\NetBeansProjects\ORM_conexion\src\orm_conexion\HibernateUtil.java:25: error: cannot find symbol
            sessionFactory = new AnnotationConfiguration().configure().buildSessionFactory();
  symbol:   class AnnotationConfiguration
  location: class HibernateUtil
2 errors
C:\Users\J\OneDrive\Documentos\NetBeansProjects\ORM_conexion\nbproject\build-impl.xml:930: The following error occurred while executing this line:
C:\Users\J\OneDrive\Documentos\NetBeansProjects\ORM_conexion\nbproject\build-impl.xml:270: Compile failed; see the compiler error output for details.
BUILD FAILED (total time: 1 second)
```

4.2 – ORM CON HIBERNATE

- En la versión 4.3 indica que *AnnotationConfiguration* está *deprecated* y que en las nuevas versiones se sustituye por ***Configuration***.
- Vamos a ver un ejemplo de programa que creará:
  - 1 sede, 1 departamento de esa sede y un empleado de ese departamento.
  - Todo dentro de una sesión y dentro de una transacción.

4.2 – ORM CON HIBERNATE

- En la versión 4.3 indica que *AnnotationConfiguration* está *deprecated* y que en las nuevas versiones se sustituye por ***Configuration***.
- Vamos a ver un ejemplo de programa que creará:
  - 1 sede, 1 departamento de esa sede y un empleado de ese departamento.
  - Todo dentro de una sesión y dentro de una transacción.

4.2 – ORM CON HIBERNATE

- Ejemplo de programa:
  - Usamos la clase *HibernateUtil* para abrir una *Session*.
  - Las clases POJO están en el paquete ORM, con lo que les antecede al nombre y se usa el constructor sin parámetros.
  - Para guardar objetos en la BD se usa el método *save()*.

```
6   package orm_conexion;
7
8   import org.hibernate.Session;
9   import org.hibernate.Transaction;
10
11  public class ORM_conexion {
12
13      public static void main(String[] args) {
14          Transaction t = null;
15          try (Session s = HibernateUtil.getSessionFactory().openSession()){
16              t = s.beginTransaction();
17
18              ORM.Sede sede = new ORM.Sede();
19              sede.setNomSede("ALICANTE");
20              s.save(sede);
21
22              ORM.Departamento dpto = new ORM.Departamento();
23              dpto.setNomDepto("INVESTIGACIÓN Y DESARROLLO");
24              dpto.setSede(sede);
25              s.save(dpto);
26
27              ORM.Empleado emp = new ORM.Empleado();
28              emp.setDni("01234567A");
29              emp.setNomEmp("MAJERE");
30              emp.setDepartamento(dpto);
31              s.save(emp);
32
33              t.commit();
34          } catch (Exception e) {
35              e.printStackTrace(System.err);
36              if (t != null) {
37                  t.rollback();
38                  }
39          }
40      }
41  }
```

4.2 – ORM CON HIBERNATE

a) Haz un programa que cree una nueva sede, dos departamentos para esta nueva sede y dos empleados para cada uno de estos departamentos. Verifica que los datos se crean correctamente comprobando el contenido de las tablas.

4.2 – ORM CON HIBERNATE

b) Ejecuta otra ve el programa de ejemplo, y verifica si se produce alguna excepción. Localiza el tipo de excepción. Cambia el programa para que este tipo de excepción se gestione de manera separada y se proporcione una información más concisa pero suficiente, en lugar de la muy prolija proporcionada por *printStackTrace()*. No se puede capturar directamente una excepción del tipo *ConstraintViolationException*. Hay que utilizar repetidamente el método *getCause()* de *Exception* y verificar el tipo de excepción con *instanceof ConstraintViolationException*.

Haz que no puedan existir dos sedes distintas con idéntico nombre, y que no puedan existir dos departamentos distintos con idéntico nombre en una misma sede. La manera más sencilla es con índice únicos (sentencia *CREATE UNIQUE INDEX* de SQL). Verifica tu solución utilizando el programa de ejemplo inicial o pequeñas variaciones de él. Hay que introducir esta restricción en la propia base de datos, y hay que verificar que (en el caso en que se intente crear una nueva sede con el mismo nombre que una ya existente, y en el caso en que se intente crear un departamento ya existente en esa sede) se produce una excepción y el programa la gestiona adecuadamente.

4.2 – ORM CON HIBERNATE

- Ficheros hbm o de correspondencia.
  - Definen la correspondencia entre clases de Java y tablas de la BD.
- Hibernate requiere la opción *implements java.io.Serializable*.
- Atributos declarados como privados.
- Métodos *getter* y *setter*.
- Relación 1:n departamento con empleados se refleja en *Set empleados*.
- Relación 1:n sede con departamentos se refleja en atributo *Sede sede*.

4.2 – ORM CON HIBERNATE

- Definición de la tabla Departamento en MySQL.

Tabla Departamento en MySQL

```
create table departamento(
id_depto integer auto_increment not null,
nom_depto char(32) not null,
id_sede integer not null,
primary key(id_depto),
foreign key fk_depto_sede(id_sede) references sede(id_sede));
```

4.2 – ORM CON HIBERNATE

POJO Departamento.java

```java
@Entity
@Table(name="departamento"
    ,catalog="proyecto_orm"
)
public class Departamento  implements java.io.Serializable {


     private Integer idDepto;
     private Sede sede;
     private String nomDepto;
     private Set empleados = new HashSet(0);

    public Departamento() {
    }


    public Departamento(Sede sede, String nomDepto) {
        this.sede = sede;
        this.nomDepto = nomDepto;
    }
    public Departamento(Sede sede, String nomDepto, Set empleados) {
       this.sede = sede;
       this.nomDepto = nomDepto;
       this.empleados = empleados;
    }

     @Id @GeneratedValue(strategy=IDENTITY)


    @Column(name="id_depto", unique=true, nullable=false)
    public Integer getIdDepto() {
        return this.idDepto;
    }

    public void setIdDepto(Integer idDepto) {
        this.idDepto = idDepto;
    }
```

```java
    @ManyToOne(fetch=FetchType.LAZY)
        @JoinColumn(name="id_sede", nullable=false)
        public Sede getSede() {
            return this.sede;
        }

        public void setSede(Sede sede) {
            this.sede = sede;
        }


        @Column(name="nom_depto", nullable=false, length=32)
        public String getNomDepto() {
            return this.nomDepto;
        }

        public void setNomDepto(String nomDepto) {
            this.nomDepto = nomDepto;
        }

    @OneToMany(fetch=FetchType.LAZY, mappedBy="departamento")
        public Set getEmpleados() {
            return this.empleados;
        }

        public void setEmpleados(Set empleados) {
            this.empleados = empleados;
        }
```

4.2 – ORM CON HIBERNATE

- Fichero de correspondencia *departamento.hbm.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 07-nov-2021 22:53:49 by Hibernate Tools 4.3.1 -->
<hibernate-mapping>
    <class name="ORM.Departamento" table="departamento" catalog="proyecto_orm" optimistic-lock="version">
        <id name="idDepto" type="java.lang.Integer">
            <column name="id_depto" />
            <generator class="identity" />
        </id>
        <many-to-one name="sede" class="ORM.Sede" fetch="select">
            <column name="id_sede" not-null="true" />
        </many-to-one>
        <property name="nomDepto" type="string">
            <column name="nom_depto" length="32" not-null="true" />
        </property>
        <set name="empleados" table="empleado" inverse="true" lazy="true" fetch="select">
            <key>
                <column name="id_depto" not-null="true" />
            </key>
            <one-to-many class="ORM.Empleado" />
        </set>
    </class>
</hibernate-mapping>
```

4.2 – ORM CON HIBERNATE

- Fichero de correspondencia *departamento.hbm.xml*
- Elemento *<class>* indica clase *(ORM.Departamento)*, tabla *(departamento)* y BD *(proyecto_orm)*.
- Correspondencia de atributos
  - Name: nombre del atributo de la clase
  - Type: Tipo del atributo Java que corresponde con tipo SQL.
- El atributo de la Tabla se indica con <column> el cual puede tener atributo con información adicional (length).

4.2 – ORM CON HIBERNATE

- Tipos de elemento
  - **<id>** - atributo que es clave primaria en la tabla.

Departamento.java	CREATE TABLE departamento
Private Integer **idDepto**;	id_depto integer auto_increment not null, **primary key**(id_depto)

Departamento.hbm.xml

```
<id name="idDepto" type="java.lang.Integer">
      <column name="id_depto" />
      <generator class="identity" />
</id>
```

4.2 – ORM CON HIBERNATE

- <generator …/>
  - Generar valores para los campos de clave primaria.
  - Opciones:
    - **identity**: clave autogenerada. La BD genera el valor.
    - **assigned**: la aplicación proporciona el valor. (dni en empleado)
    - **sequence**: secuencia de la BD. En Oracle genera PK.
    - **native**: identity, sequence, hilo según la BD.
    - **foreign** : para relaciones de uno a uno.

4.2 – ORM CON HIBERNATE

- Tipos de elemento: **<property>** - asociar atributo de la clase con uno en la tabla.

Departamento.java	CREATE TABLE departamento
private String **nomDepto**;	nom_depto char(32) not null

Departamento.hbm.xml

```
<property name="nomDepto" type="String">
      <column name="nom_depto" length="32" not-null="true" />
</property >
```

RESUMEN

- ✓ La persistencia de objetos en bases de datos relacionales plantea un conjunto de problemas que se conocen en conjunto como ”desfase objeto relacional”.
- ✓ ORM (object-relational mapping), o correspondencia objeto relacional, consiste en el establecimiento de una correspondencia entre clases definidas en un lenguaje POO y tablas de una BD Relacional, y en el uso de mecanismos para que las modificaciones sobre los objetos se registren en la BD y a la inversa.
- ✓ La correspondencia objeto-relacional hace posible la persistencia de objetos en bases de datos puramente relacionales.

RESUMEN

- ✓ Hibernate es un Framework para ORM en lenguaje Java distribuido bajo licencia LGPL de GNU. Fue la primera herramienta de ORM y hoy día sigue teniendo gran aceptación.
- ✓ Además de tener su propia API nativa, Hibernate es una implementación certificada de JPA, el estándar de Java para ORM, que es parte de la especificación Java EE.
- ✓ Con Hibernate, la correspondencia se puede establecer mediante ficheros hbm (Hibernate mapping files o “ficheros de correspondencia de Hibernate”) o mediante anotaciones de JPA.
- ✓ Una clase para la que se ha definido la correspondencia O-R es una clase persistente.

Acceso a Datos

FIN DE LA PARTE I

GRACIAS