
M6.UF2.A4

**BASES DE DATOS EMBEBIDAS Y
ORIENTADAS A OBJETOS**

Eduard Lara

INDICE

1. Introducción a BBDO (ODMG, ODL, OQL)
2. BBOO NeoDatis
3. SQLite
4. Derby Apache
5. HSQLDB Y H2

1. INTRODUCCION BDOO

- **BDOO:** Bases de datos orientadas a Objetos
- Son aquellas bases de datos cuyo modelo de datos está orientado a objetos. Almacenan tanto métodos como datos
- Las BDOO simplifican la programación orientada a objetos (POO) almacenando directamente los objetos en la BD.
- **SGBDOO:** sistema gestor de bases de datos que almacena objetos.
- **ODMG:** (Object Database Management Group) es un grupo de fabricantes de bdd que tiene como objetivo definir estándares para los SGBDOO. Uno de dichos estándares (se llama ODMG, precisamente) especifica los elementos que debe contener el SGBDOO

1. EL ESTÁNDAR ODMG

- La última versión del estándar, ODMG 3.0 propone los siguientes componentes para un SGBDOO:
 - Modelo de objetos
 - Lenguaje de definición de objetos (**ODL**, Object Definition Language)
 - Lenguaje de consulta de objetos (**OQL**, Object Query Language)
 - Conexión con los lenguaje C++, Smalltalk y Java

1. LENGUAJE DEFINICION OBJETOS (ODL)

- El lenguaje ODL es el equivalente al lenguaje de definición de datos (DDL) de los SGDB tradicionales.
- Define atributos, relaciones y signatura de las operaciones
- La sintaxis de ODL extiende el lenguaje de definición de interfaces de CORBA (Common Object Broker Architecture).
- Algunas de las palabras reservadas para definir los objetos son:
 - **class** → declaración del objeto
 - **extent** → define la extensión, nombre para el actual conjunto de objetos de la clase.
 - **key[s]** → declara lista de claves
 - **attribute** → declara un atributo
 -

1. LENGUAJE CONSULTA OBJETOS (OQL)

- **OQL: Object Query Language** es el lenguaje estándar de consultas de BDOO. Las características son:
 - Es orientado a objetos y está basado en el modelo de objetos de la ODMG
 - Es un lenguaje declarativo del tipo de SQL. Su sintaxis básica es similar a SQL
 - No incluye operaciones de actualización, solo de consulta. Para modificar un objeto de la bbdd se utilizan los propios métodos del objeto.
 - Dispone de operadores sobre colecciones (max, min, count, etc.) y cuantificadores (for all, exists).
- La sintaxis básica es:

```
SELECT <lista de valores>  
FROM <lista de colecciones y miembros típicos>  
[WHERE <condición>]
```

1. LENGUAJE CONSULTA OBJETOS (OQL)

- En el FROM podemos indicar colecciones o expresiones que evalúan una colección. Se suele utilizar una variable iterador para ir recorriendo todos los elementos de la colección. Así, las siguientes expresiones son equivalentes:

```
FROM Clientes x  
FROM x IN Clientes  
FROM Clientes AS x
```

- Para acceder a los atributos y objetos relacionados se utilizan expresiones de camino. Veamos algunos ejemplos:

```
SELECT x.nombre.nombreper FROM x IN Clientes WHERE x.sexo="M"  
SELECT x.nombre.nombreper FROM Clientes x WHERE x.sexo="M"
```

1. LENGUAJE CONSULTA OBJETOS (OQL)

- Obtener el nombre de los clientes que son mujeres:
SELECT x.nombre.nombreper FROM x IN Clientes WHERE x.sexo="M";
- O, la misma consulta, pero utilizando una sintaxis todavía más parecida a SQL:
SELECT x.nombre.nombreper FROM Clientes x WHERE x.sexo="M";
- Obviamente, OQL admite operadores de comparación, funciones de comparación (IN, LIKE), funciones de agregación (SUM, AVG, MIN, MAX, COUNT) y todo un largo etc.

1. LENGUAJE CONSULTA OBJETOS (OQL)

- Los valores pueden ser comparados usando los operadores habituales (=, >, <, >=, <=, etc.)
- Para comparar cadenas de caracteres se pueden utilizar los operadores IN y LIKE de la misma manera que se utilizan en SQL
- Un manual lo podéis encontrar [aquí](#).
- En cuanto a ejemplos prácticos con NeoDatis tenéis [esto](#) que seguro que os resulta de gran utilidad.

2. BDOO NEODATIS

- A continuación veremos una base de datos sencilla que aporta una API simple. Se trata de NeoDatis Object Database
- Neodatis ODB es una base de datos orientada a objetos con licencia GNU que actualmente corre en los lenguajes Java, .Net, Google Android, Groovy y Scala.
- Con Neodatis podemos evitar la falta de impedancia entre los mundos OO y los relacionales, actuando como una capa de persistencia transparente para Java.
- Neodatis ODB soporta consultas nativas, es decir, podemos lanzar una consulta directamente desde Java, por ejemplo.
- Además es bastante simple e intuitivo. Los objetos pueden ser añadidos fácilmente a la base de datos, lo que requiere clases no repetitivas y que las clases ya existentes no puedan modificarse.
- También cuenta con un explorador ODB, una herramienta gráfica para navegar, consultar, actualizar y borrar objetos, así como la importación / exportación de bases de datos desde y hacía archivos XML

2. CONSULTAS BDOO NEODATIS

- Para realizar consultas en NeoDatis usamos la clase **CriteriaQuery** en donde especificaremos la clase y el criterio para realizar la consulta.
- La estructura básica es:

```
IQuery query = new CriteriaQuery(Jugadores.class,  
    Where.equal("deporte", "tenis"));  
query.orderByDesc("nombre,edad");  
Objects<Jugador> objects = odb.getObjects(query);
```

- Para obtener el primer objeto usamos el método `getFirst()`:
Jugador j = odb.getObjects(query).getFirst();
- Este método lanza la excepción **IndexOutOfBoundsException** si no localiza ningún objeto que cumpla con el criterio.

2. CONSULTAS BDOO NEODATIS

- Con CriteriaQuery se puede usar la interfaz ICriterion para construir el criterio de la consulta. Por ejemplo:

```
ICriterion criterio = Where.equal("edad",14);
```

```
CriteriaQuery query = new CriteriaQuery(Jugadores.class, criterio);
```

```
Objects<Jugadores> objects = odb.getObjects(query);
```

2. CONSULTAS BDOO NEODATIS

- Otros criterios de consulta:

// Jugadores que empiezan por M

ICriterion criterio2 = Where.like("nombre","M%");

// Jugadores que empiezan por M

ICriterion criterio3 = Where.gt("edad",14);

// Jugadores que no empiezan por M

ICriterion criterio4 = Where.not(Where.like("nombre","M%"));

// Jugadores de 15 años de Madrid

**ICriterion criterio = new And().add(Where.equal("ciudad",
"Madrid")).add(Where.equal("edad",15));**

// Jugadores >= de 15 años y de Madrid

**ICriterion criterio2 = new And().add(Where.equal("ciudad",
"Madrid")).add(Where.ge("edad",15));**

2. CONSULTAS BDOO NEODATIS

- Igual que en postgresQL cada objeto dispone de su identificador que se suele llamar OID. Podemos ver los datos del objeto haciendo, por ejemplo:

```
public static void main (String[] args) {  
    Jugadores jug1=new Jugadores();  
    ODB odb= ODBFactory.open("neodatis.test");  
    Jugadores j1 = new Jugadores("Luis","voleibol", "Madrid", 14);  
    OID oid= odb.store(j1);  
    jug1 = (Jugadores)odb.getObjectFromId(oid);  
    System.out.printf("%s,%s,%s, %d %n", jug1.getNombre(),  
    jug1.getDeporte(),jug1.getCiudad(),jug1.getEdad()); odb.close();  
}  
}
```

2. PRACTICA BDOO NEODATIS

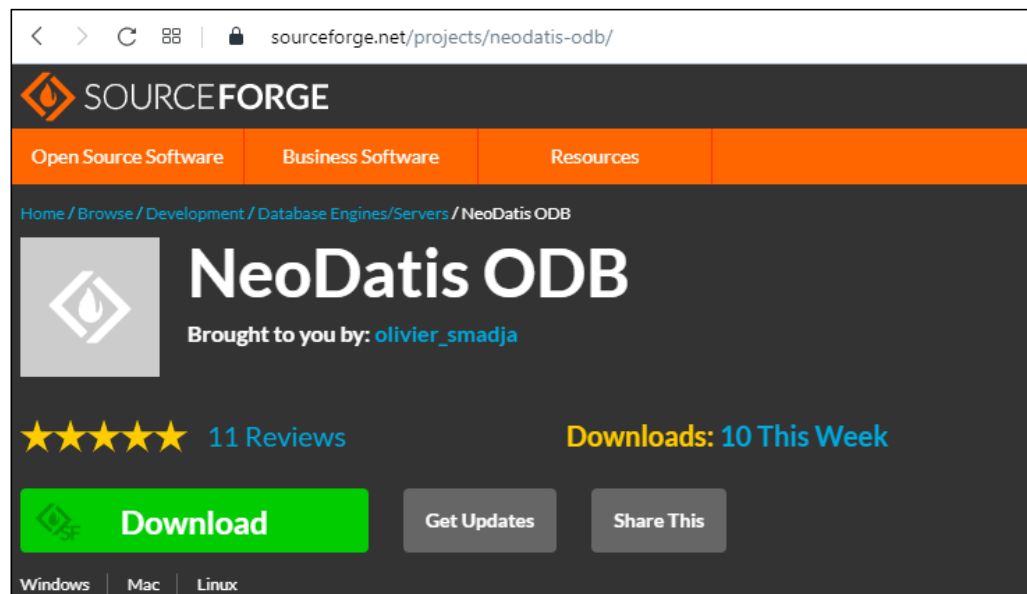
- Realizar un CRUD que almacene objetos de tipo Jugador en una base de datos Neodatis de nombre *neodatis.test*.
- Los elementos de la practica a realizar se encuentran en:
<http://www.programandoapasitos.com/2016/03/acceso-datos-neodatis.html>

```
//CRUD--Create, Read, Update, Delete
while (!fin) {
    System.out.println("1. Visualizar la lista de jugadores");
    System.out.println("2. Insertar un nuevo jugador");
    System.out.println("3. Borrar jugador");
    System.out.println("4. Modificar jugador");
    System.out.println("5. Consulta de jugadores por ciudad");
    System.out.println("6. Salir");
    System.out.print("Introduce que opcion quieres?");
    int opcion = reader.nextInt();
    switch(opcion) {
```

- Desde la web <http://neodatis.wikidot.com/> tendremos acceso a las librerías y a la documentación.

2. PRACTICA BDOO NEODATIS

Paso 1. Descarga Neodatis de <https://sourceforge.net/projects/neodatis-odb/>, y descomprime el archivo zip.

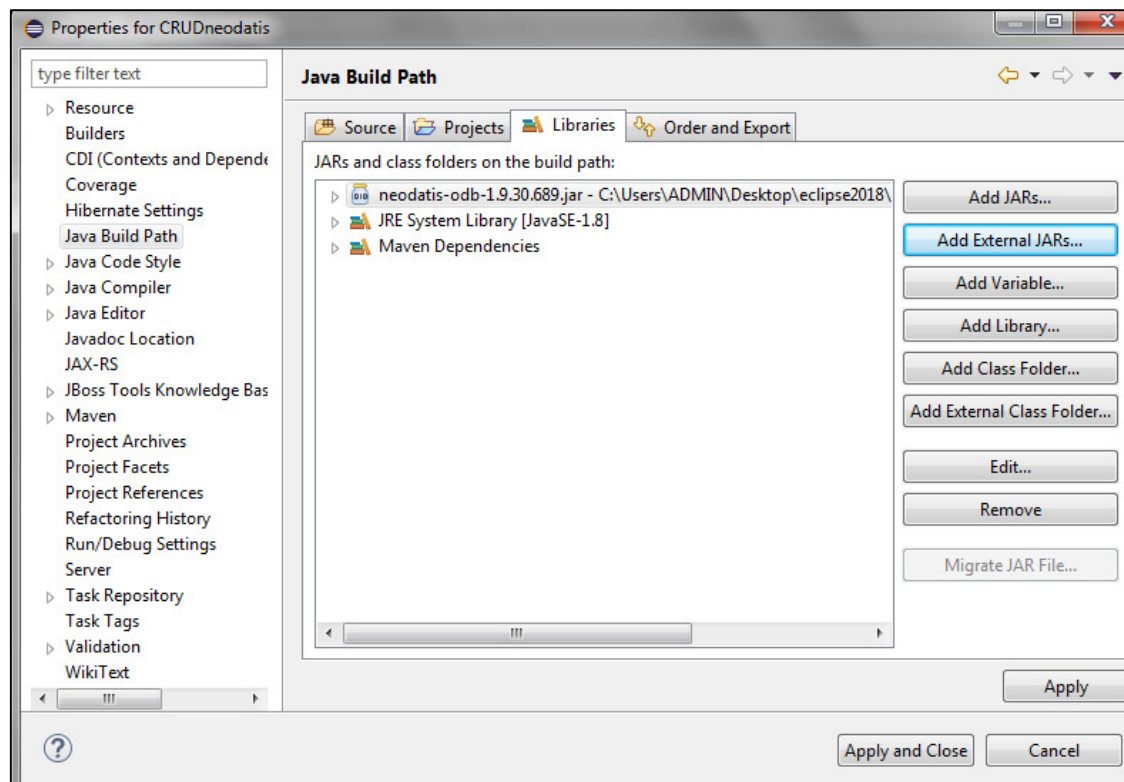


Nombre	Fecha de modifica...	Tipo	Tamaño
doc	09/01/2021 13:53	Carpeta de archivos	
lib	09/01/2021 13:53	Carpeta de archivos	
maven	09/01/2021 13:53	Carpeta de archivos	
src	09/01/2021 13:53	Carpeta de archivos	
LICENSE.ASM.txt	09/01/2021 13:53	Archivo TXT	2 KB
license.txt	09/01/2021 13:53	Archivo TXT	27 KB
neodatis-odb-1.9.30.689.jar	09/01/2021 13:53	Executable Jar File	754 KB
odb-explorer.bat	09/01/2021 13:53	Archivo por lotes ...	1 KB
odb-explorer.sh	09/01/2021 13:53	Shell Script	1 KB
readme.txt	09/01/2021 13:53	Archivo TXT	2 KB

De aquí nos interesan dos archivos. **Neodatis-odb-1.9.30.689.jar**, que será la librería para nuestro proyecto en Java, y **odb-explorer**, para usar el explorador de base de datos. .bat si usamos Windows, .sh para Linux.

2. PRACTICA BDOO NEODATIS

Paso 2. Crea un proyecto java y agrega el driver jar al proyecto, en Java Build Path. Si el proyecto es maven, bastaría con incluir la dependencia en pom.xml.



```
<dependencies>
  <dependency>
    <groupId>org.neodatis.odb</groupId>
    <artifactId>neodatis-odb</artifactId>
    <version>1.9.30.689</version>
  </dependency>
</dependencies>
```

2. PRACTICA BDOO NEODATIS

Paso 3. Crea la clase POJO Jugador, con sus getters y setters:

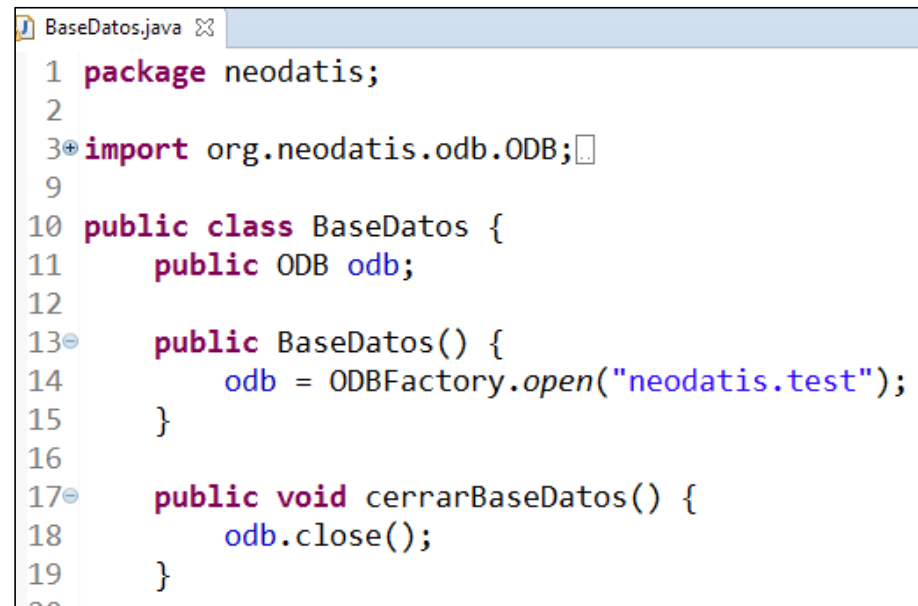
```
Jugador.java
1 package neodatis;
2
3 public class Jugador {
4
5     // Propiedades
6     private String nombre;
7     private String deporte;
8     private String ciudad;
9     private int edad;
10
11     // Constructores
12     public Jugador(){
13     }
14
15     public Jugador(String nombre, String deporte, String ciudad, int edad) {
16         this.nombre = nombre;
17         this.deporte = deporte;
18         this.ciudad = ciudad;
19         this.edad = edad;
20     }
21 }
```

2. PRACTICA BDOO NEODATIS

Paso 4. Para realizar el CRUD, usaremos el típico fichero BaseDatos.java, donde el objeto conexión de la base de datos, ahora es un objeto de tipo ODB.

El método **open()** de la clase **ODBFactory** abre una conexión a la base de datos y devuelve un objeto **ODB**.

Para validar los cambios en la bbdd, se usa el método **close()**. Es importante llamar al método close al salir del programa, sino la base de datos se puede corromper



```
BaseDatos.java
1 package neodatis;
2
3 import org.neodatis.odb.ODB;
9
10 public class BaseDatos {
11     public ODB odb;
12
13     public BaseDatos() {
14         odb = ODBFactory.open("neodatis.test");
15     }
16
17     public void cerrarBaseDatos() {
18         odb.close();
19     }
20 }
```

2. PRACTICA BDOO NEODATIS

Paso 5. Para almacenar los objetos se usa el método **store()** y para recuperarlos o listarlos el método **getObjects()**.

```
public void insertar(Jugador jug) {  
    odb.store(jug);  
}
```

```
public void listar() {  
    Objects<Jugador> objetos=odb.getObjects(Jugador.class);  
    System.out.println(objetos.size() + " jugadores:");  
    int i = 1;  
    while(objetos.hasNext()){  
        Jugador jug= objetos.next();  
        System.out.println((i++) + jug.toString());  
    }  
}
```

2. PRACTICA BDOO NEODATIS

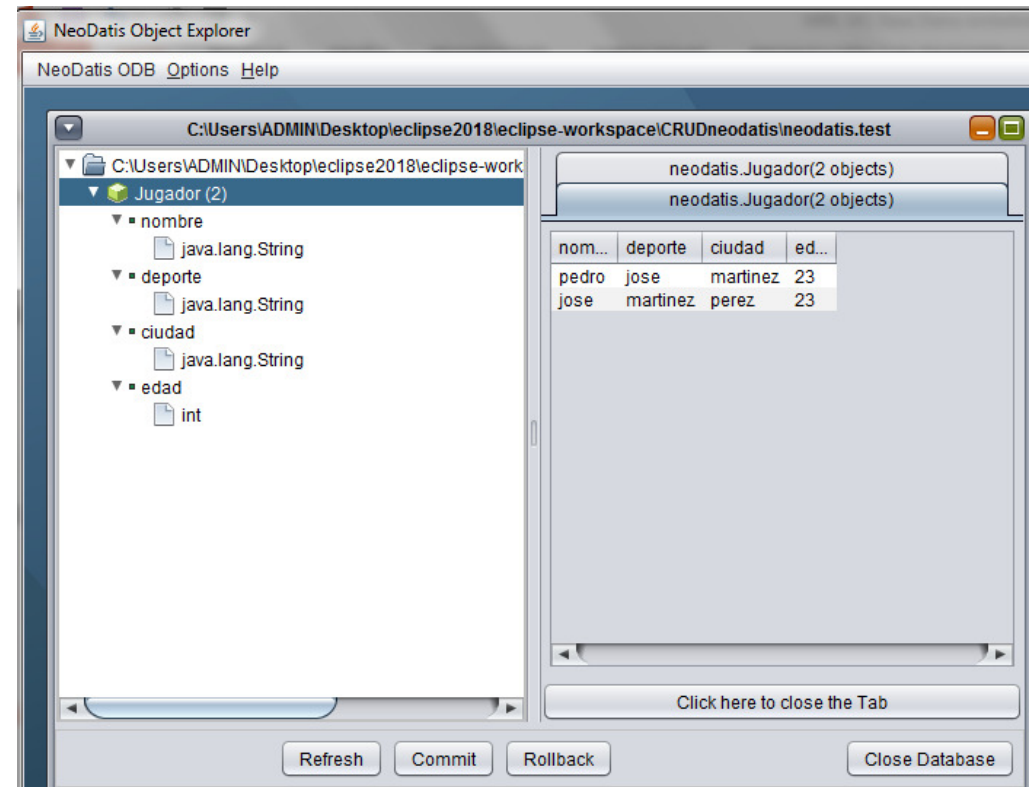
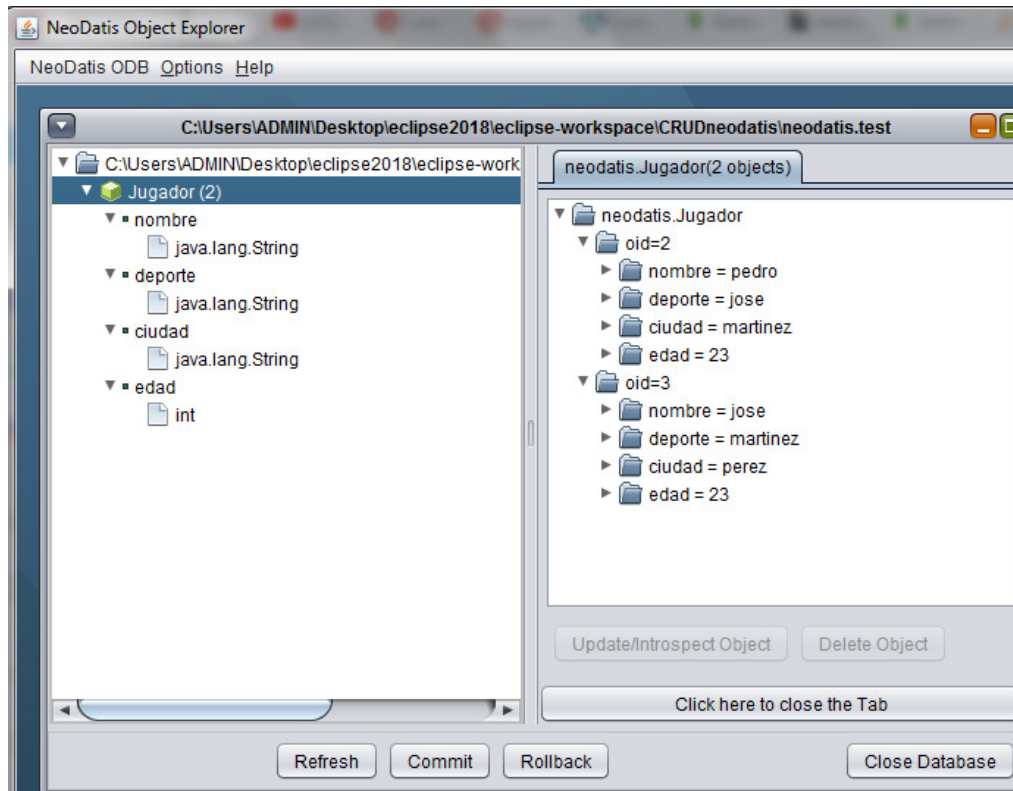
Paso 6. Para borrar un objeto crearemos un objeto **IQuery** para traernos a memoria los objetos a borrar. Nos posicionamos en el primero objeto y lo borramos usando el ODB. Con la modificación haremos un proceso similar:

```
public void borrar(Jugador jugador) {  
    IQuery query = new CriteriaQuery(Jugador.class,  
        Where.equal("nombre", jugador.getNombre()));  
    Objects<Jugador> objects = odb.getObjects(query);  
    Jugador jug=(Jugador) odb.getObjects(query).getFirst();  
    odb.delete(jug);  
}
```

```
public void actualizar(Jugador jugador) {  
    IQuery query = new CriteriaQuery(Jugador.class,  
        Where.equal("nombre", jugador.getNombre()));  
    Objects<Jugador> objects = odb.getObjects(query);  
    Jugador jug=(Jugador) odb.getObjects(query).getFirst();  
    jug.setDeporte(jugador.getDeporte());  
    jug.setCiudad(jugador.getCiudad());  
    jug.setEdad(jugador.getEdad());  
    odb.store(jug);  
}
```

2. PRACTICA BDOO NEODATIS

Paso 7. Una vez hemos creado unos objetos con la aplicación, y una vez la hemos cerrado correctamente , con la utilidad odb-explorer.bat podemos abrir el fichero de la base de datos y consultarlos.



3. BASES DE DATOS EMBEBIDAS

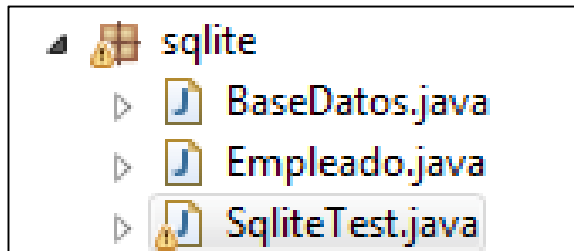
- Cuando desarrollamos pequeñas aplicaciones en las que no vamos a almacenar grandes cantidades de datos no es necesario que utilicemos un sistema gestor de base de datos como Oracle o MySql.
- En su lugar podemos utilizar una base de datos embebida donde el motor está incrustado en la aplicación y sea exclusivo para ella.
- La base de datos se inicia cuando se ejecuta la aplicación, y termina cuando se cierra la aplicación.
 - SQLite
 - Apache Derby

3. SQLITE

- SGBDR multiplataforma escrito en C que proporciona un motor muy ligero.
- Las bbdd se guardan en forma de ficheros por lo que es fácil trasladar la bbdd con la aplicación que la usa.
- Cuenta con una pequeña utilidad que nos permitirá ejecutar comandos SQL en modo consola.
- Es un proyecto de dominio público e implementa la mayor parte de SQL-92, incluyendo transacciones, consistencia (foreign keys), triggers, consultas complejas, etc.
- SQLite se puede utilizar desde C/C++, PHP, Visual Basic, Perl, Java, etc.
- La implementación más popular del driver JDBC para SQLite es la de Xerial <https://github.com/xerial/sqlite-jdbc>

3. PRACTICA SQLITE

Paso 1. Realiza un CRUD con la base de datos SQLITE, partiendo del CRUD realizado con MYSQL y JDBC. Adapta esta práctica para que funcione según el ejemplo con SQLITE de <https://github.com/xerial/sqlite-jdbc>.



```
while (true) {  
    System.out.println("1. Visualizar la lista de empleados");  
    System.out.println("2. Incrementar salario de empleado");  
    System.out.println("3. Insertar un nuevo empleado");  
    System.out.println("4. Borrar un nuevo empleado");  
    System.out.println("5. Salir");  
    System.out.print("Introduce que opcion quieres?");  
    int opcion = reader.nextInt();  
    switch(opcion) {
```

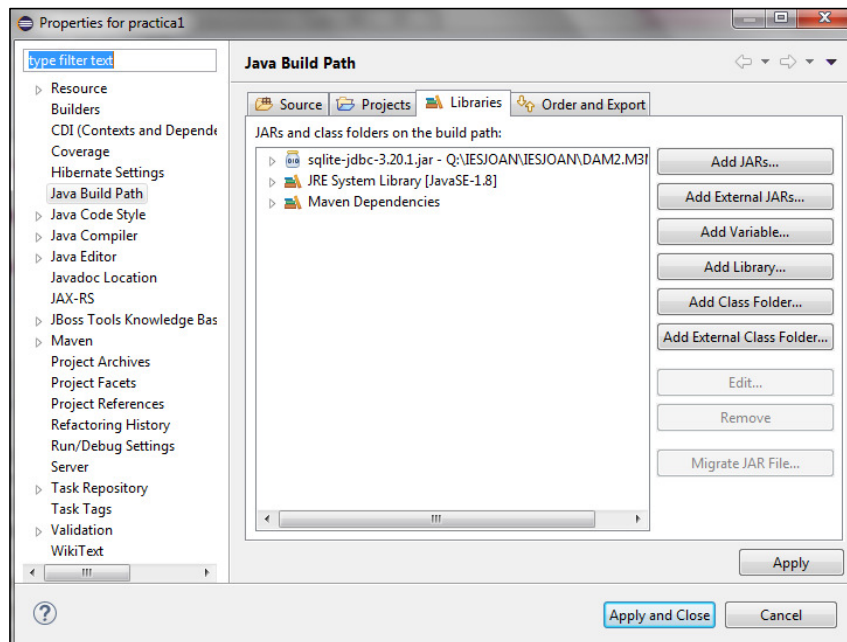
Sólo se debe de modificar el fichero BaseDatos.java y crear la base de datos del ejemplo para que se adapte a la clase Empleado

3. PRACTICA SQLITE

Paso 2. Agrega al proyecto el driver de SQLITE:

- Mediante maven, indicando la dependencia en el fichero pom.xml.
- O bien físicamente descargándolo de

<https://jar-download.com/artifacts/org.xerial/sqlite-jdbc/3.20.1/source-code>



```
<dependency>
  <groupId>org.xerial</groupId>
  <artifactId>sqlite-jdbc</artifactId>
  <version>3.20.1</version>
</dependency>
```

3. PRACTICA SQLITE

Paso 3. El constructor de BaseDatos.java debe de crear la base de datos y la tabla empleados de la siguiente manera:

```
public class BaseDatos {
    private Connection conexion;

    public BaseDatos() {
        try {
            this.conexion = DriverManager.getConnection("jdbc:sqlite:sample.db");

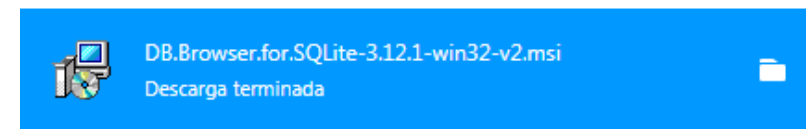
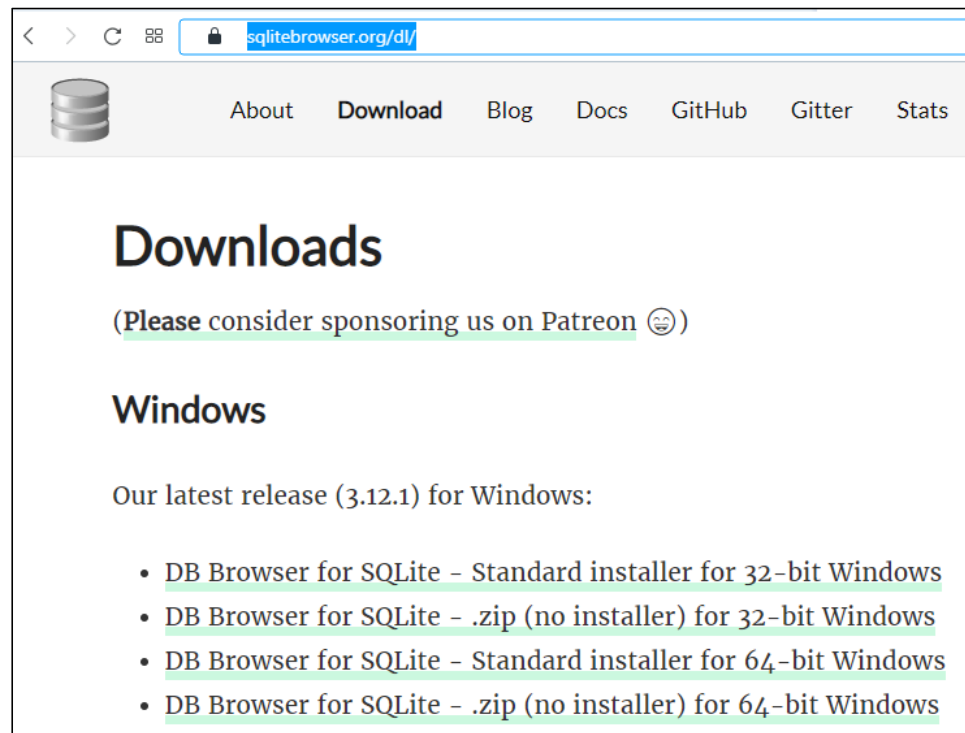
            Statement statement = conexion.createStatement();
            statement.setQueryTimeout(30); // set timeout to 30 sec.

            statement.executeUpdate("drop table if exists empleados");
            statement.executeUpdate("create table empleados (id integer primary key autoincrement, "
                                   + "name string, apellido string, salary int)");
            statement.executeUpdate("insert into empleados values(null, 'leo','messi',30000)");
            statement.executeUpdate("insert into empleados values(null, 'oso','yogui',20000)");

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

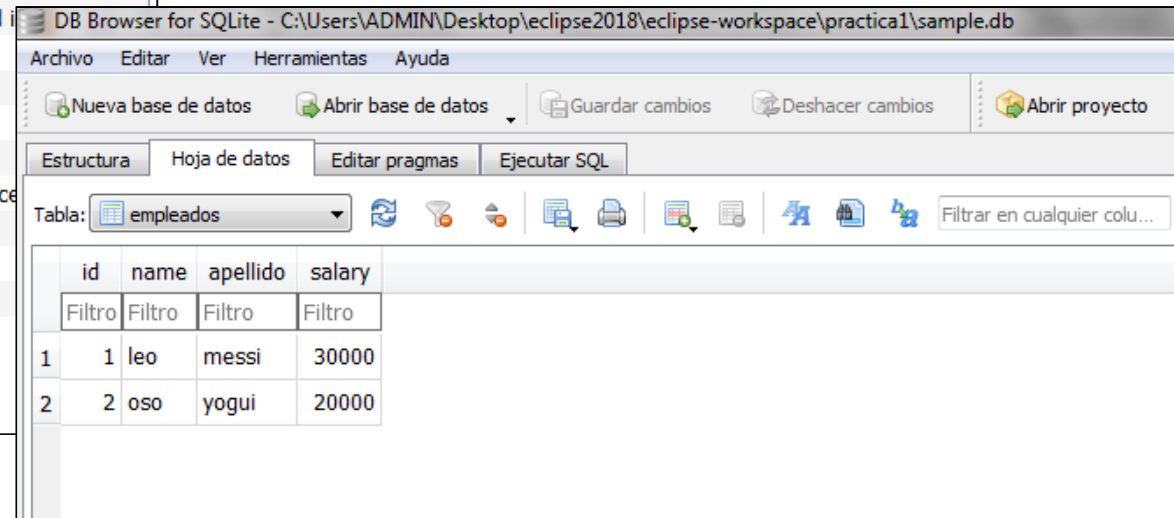
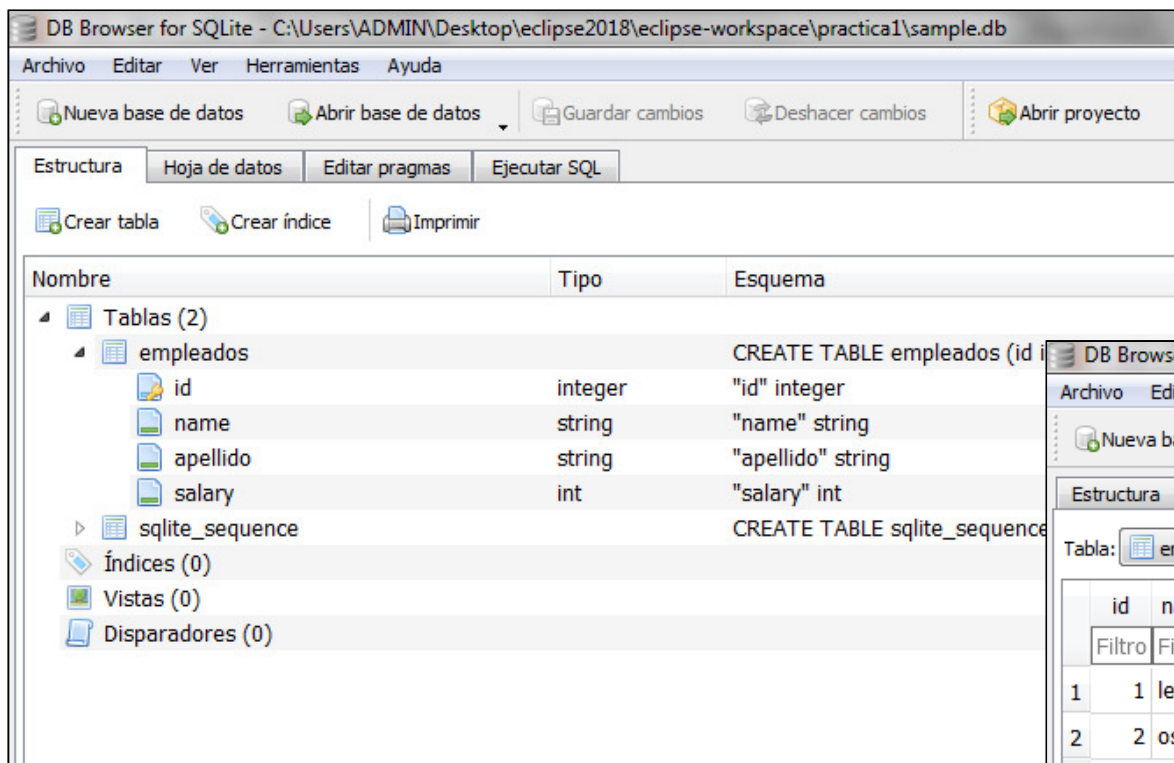
3. PRACTICA SQLITE

Paso 4. El programa DB Browser for SQLite es el visualizador de ficheros-base datos Sqlite más popular. Nos permite acceder a la base de datos creada por nuestra aplicación y comprobar su contenido. Se puede descargar de la pagina web: <https://sqlitebrowser.org/dl/>



3. PRACTICA SQLITE

Paso 5. Una vez instalado, DB Browser nos permite observar las tablas de la base de datos Sqlite creada con nuestra aplicación, y su contenido:



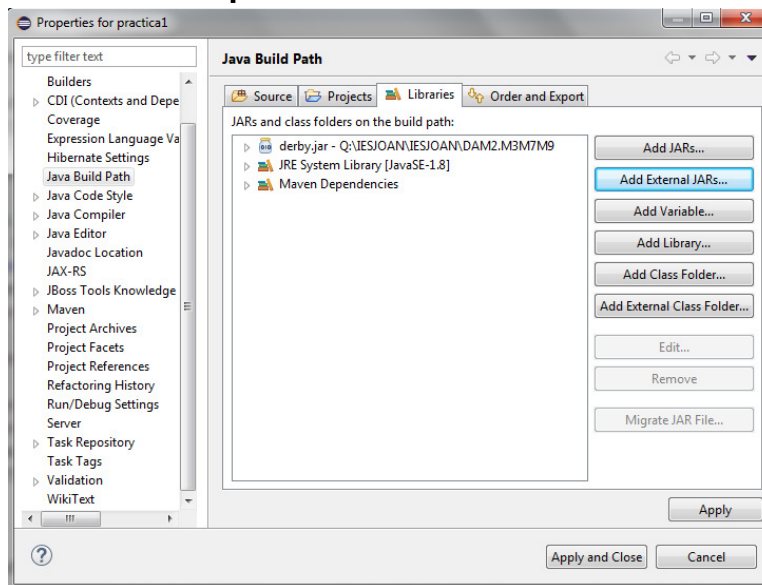
4. APACHE DERBY

- Es un SGBDR de código abierto implementado completamente en Java y forma parte del Apache DB Project.
- Ventajas: tamaño reducido, basada en Java, soporta estándares SQL, ofrece un controlador integrado JDBC y puede incrustarse fácilmente en cualquier solución Java.
- Soporta también el tradicional paradigma cliente-servidor utilizando el servidor de red Derby.

4. PRACTICA APACHE DERBY

Paso 1. Adapta la practica de SQLITE para que funcione ahora con DERBY. Las librerías derby se pueden agregar al proyecto de dos maneras:

- Descargando las librerías de: https://db.apache.org/derby/derby_downloads.html, y agregando en Java Build Path los siguientes jar: derby.jar y/o derbytools.jar y derbyshared.jar.
- Añadiendo las dependencias de Maven en pom.xml



```
<dependency>
  <groupId>org.apache.derby</groupId>
  <artifactId>derby</artifactId>
  <version>10.8.3.0</version>
</dependency>
```

4. PRACTICA APACHE DERBY

Paso 2. Sólo se debe de modificar el fichero BaseDatos.java.

```
public class BaseDatos {
    private Connection conexion;

    public BaseDatos() {
        try {
            //Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
            String dbURL1 = "jdbc:derby:empleadosDB;create=true";
            this.conexion = DriverManager.getConnection(dbURL1);
            Statement statement = conexion.createStatement();
            statement.executeUpdate("drop table empleados");    //IF EXISTS NO LO PERMITE
                                                                //AUTOINCREMENT NO LO PERMITE

            statement.executeUpdate("create table empleados (id integer primary key "
                                   + "GENERATED ALWAYS AS IDENTITY (Start with 1, Increment by 1), "
                                   + "name varchar(20), apellidos varchar(20), salary int)");
            statement.executeUpdate("insert into empleados(name, apellidos, salary) values('leo','messi', 30000)");
            statement.executeUpdate("insert into empleados(name, apellidos, salary) values('oso','yogui', 20000)");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```


4. PRACTICA APACHE DERBY

Paso 3. El lenguaje que soporta Apache Derby tiene algunas limitaciones respecto SQLite:

- `//Class.forName("org.apache.derby.jdbc.EmbeddedDriver");`
- `ConexString` → `"jdbc:derby:empleadosDB;create=true"`; para crear dd
- No admite "if exists" ni "autoincrement".
- En lugar de autoincrement usar id integer primary key GENERATED ALWAYS AS IDENTITY (Start with 1, Increment by 1)
- En los insert se debe de especificar el nombre de las columnas: `insert into empleados(name, apellidos, salary) values('leo','messi', 30000);`

4. PRACTICA APACHE DERBY

Paso 4. Conexión modo terminal apache Derby database

Para conectarnos a una base de datos Derby, desde el directorio lib hacemos:

```
C>java -jar derbyrun.jar ij
```

```
ij>CONNECT 'jdbc:derby:c:/usuario/.../ejemplodb';
```

- Donde:
 - **connect** → comando para establecer conexión
 - jdbc:derby → protocolo JDBC especificado por DERBY
 - ejemplodb → es el nombre de la bbdd que vamos a crear (se crea una carpeta con dicho nombre y dentro una serie de ficheros)
 - create=true → atributo usado para crear la base de datos
- Para salir de **ij** escribimos **exit**;
- Para obtener ayuda podemos escribir **help**;
- **ij** es una utilidad que nos permite crear nuestra bbdd desde consola

4. PRACTICA APACHE DERBY

Conexión modo terminal apache Derby database

```
C:\Windows\system32\cmd.exe - java -jar derbyrun.jar ij
C:\Users\ADMIN\Desktop\db-derby-10.14.2.0-bin\lib>java -jar derbyrun.jar ij
Versión de ij 10.14
ij> connect 'jdbc:derby:C:/Users/ADMIN/Desktop/eclipse2018/eclipse-workspace/practical/empleadosDB';
ij> show tables;
TABLE_SCHEM          TABLE_NAME          REMARKS
-----
SYS                   SYSALIASES
SYS                   SYSCHECKS
SYS                   SYSCOLPERMS
SYS                   SYSCOLUMNS
SYS                   SYSCONGLOMERATES
SYS                   SYSCONSTRAINTS
SYS                   SYSDEPENDS
SYS                   SYSFILES
SYS                   SYSFOREIGNKEYS
SYS                   SYSKEYS
SYS                   SYSPERMS
SYS                   SYSROLES
SYS                   SYSROUTINEPERMS
SYS                   SYSSCHEMAS
SYS                   SYSSEQUENCES
SYS                   SYSSTATEMENTS
SYS                   SYSSTATISTICS
SYS                   SYSTABLEPERMS
SYS                   SYSTABLES
SYS                   SYSTRIGGERS
SYS                   SYSUSERS
SYS                   SYSVIEWS
SYSIBM                SYSDDUMMY1
APP                   EMPLEADOS

24 filas seleccionadas
ij> select * from empleados
> ;
ID          NAME          APELLIDOS          SALARY
-----
1           leo           messi              130000
2           oso           yogui              120000

2 filas seleccionadas
ij>
```

5. HSQldb Y H2

HSQldb

- Hyperthreaded Structured Query Language Database es un SGBDR escrito en Java.
- Gestiona hasta 270 mil millones de filas de datos en una sola bbdd y tiene capacidad de copia en caliente.
- Implementa SQL:2011
- OpenOffice lo incluye para dar soporte a la aplicación Base

H2

- H2 es un SGBDR escrito en Java. Licencia pública de Mozilla
- Tiene varios modos de estar embebida: como servidor o completamente en memoria