
Manejo de Conectores I

(Introducción a BBDD)



Contenidos

Introducción

Desfase objeto-relacional

Bases de datos embebidas

Introducción

Desfase objeto-relacional

Bases de datos embebidas

Introducción

Acceso a datos significa **el proceso de recuperación o manipulación de datos extraídos de un origen de datos local o remoto**

Algunos orígenes de datos que podemos encontrar son

- Bases de datos relacionales en un servidor remoto

- Bases de datos relacional local

- Un fichero de texto

- Un servicio de información online

- etc

En la UF2 nos centraremos en los **orígenes de datos relacionales** a través de **Java**

Necesitaremos **conectores**, software que nos permite realizar conexiones desde Java a una BD Relacional

Introducción

Desfase objeto-relacional

Bases de datos embebidas

Desfase objeto-relacional

Hoy en día las **BD orientadas a objetos** están obteniendo más aceptación frente a las **BD relacionales**

Las BD orientadas a objetos **solucionan necesidades de aplicaciones más sofisticadas** que necesitan tratar **elementos más complejos**

- Sistemas de información geográfica (GIS)
 - Experimentos científicos
 - Aplicaciones multimedia
 - etc.
- Dentro de estas nuevas aplicaciones se definen las **orientadas a objetos (OO)** y en general el Paradigma de **POO** cuyos elementos complejos son los **Objetos**

Desfase objeto-relacional

Las **Bases de datos relacionales** no están diseñadas para almacenar **objetos**

Al guardar datos de un programa **POO** se incrementa la complejidad del programa dando lugar a más código y esfuerzo de programación debido a la diferencia de esquemas entre los objetos y las tablas

El **desfase objeto-relacional** se refiere a los problemas que ocurren debido a las diferencias entre el modelo de datos de la BD y el lenguaje **POO**

Sin embargo el paradigma relacional y orientado a objetos pueden trabajar juntos realizando un **mapeo** de las estructuras del modelo de datos a las del entorno de programación, esto lo trataremos en la UF3

Introducción

Desfase objeto-relacional

Bases de datos embebidas

Bases de datos embebidas

Para aplicaciones móviles no necesitamos guardar mucha información por lo que no es necesario utilizar un SGBD como Oracle o MySQL

Podemos utilizar una **BD embebida** donde el motor esté **incrustado en la aplicación** y sea **exclusivo de ella**.

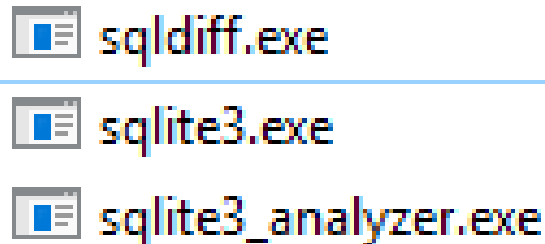
La base de datos se iniciará cuando **se ejecute la aplicación** y terminará cuando se **cierre**.

Muchas de estas BD vienen del movimiento **Open Source** aunque algunas también son **propietarias**

A continuación veremos algunas de ellas

Es un **SGBD multiplataforma de dominio público** escrito en **C** que proporciona un motor muy ligero
Las BD se guardan en forma de **ficheros** por lo que es fácil mover la BD junto a la aplicación
Cuenta con una utilidad que permite ejecutar **comandos SQL** contra una BD SQLite en modo **consola**
La biblioteca implementa la mayor parte del **estándar SQL-92**, incluyendo transacciones de BD atómicas, consistencia de BD, aislamiento y durabilidad, triggers y la mayor parte de consultas complejas
SQLite se utiliza mediante llamadas simples a subrutinas y funciones en **C++, Java, Visual Basic**, etc

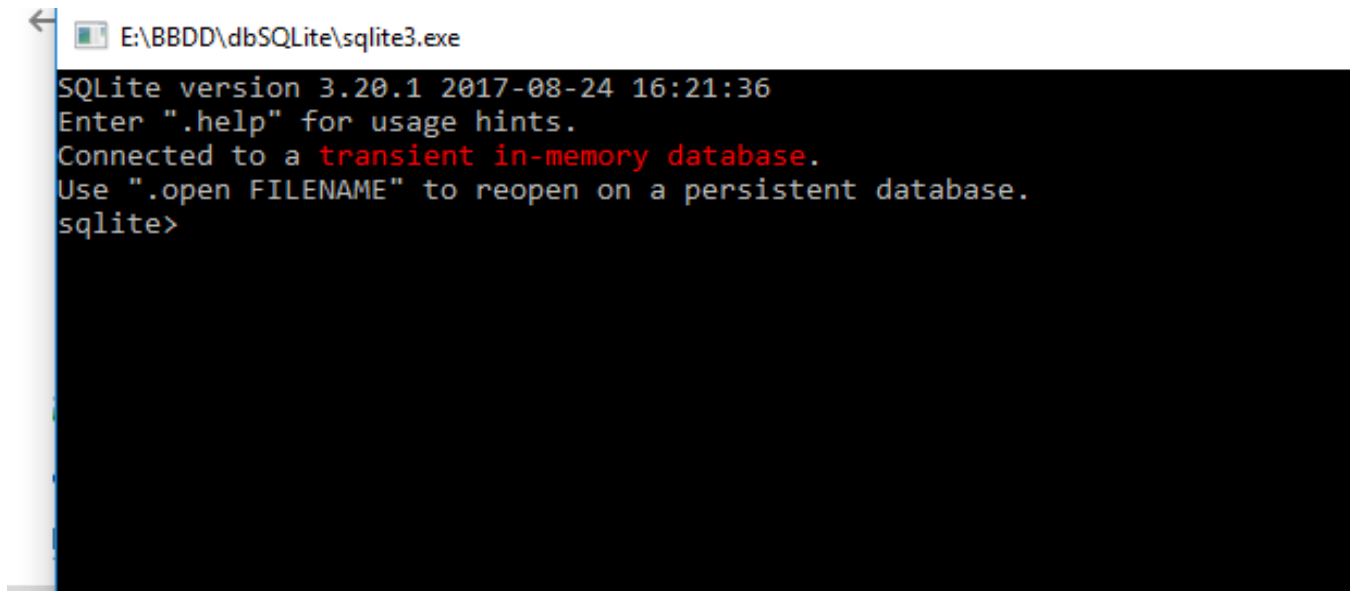
Es **muy sencilla**, desde la página <http://sqlite.org/download.html> se puede descargar
Para Windows podemos descargar [sqlite-tools-win32-x86-3200100.zip](#)
Descomprimos el fichero y obtenemos tres ejecutables

A list of three executable files, each preceded by a small icon of a document with a blue header bar. The files are: sqldiff.exe, sqlite3.exe, and sqlite3_analyzer.exe.

- sqldiff.exe
- sqlite3.exe
- sqlite3_analyzer.exe



- Creamos una carpeta donde se situará nuestra base de datos. Por ejemplo: C:/dbSQLite
- Copiamos el fichero sqlite3.exe dentro de dicha carpeta y lo ejecutamos.

A screenshot of a Windows file explorer window showing the path E:\BBDD\dbSQLite\sqlite3.exe. Below the file explorer, a terminal window is open, displaying the output of running the sqlite3.exe command. The terminal text is as follows:

```
SQLite version 3.20.1 2017-08-24 16:21:36
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```



- Para la creación, inserción, consultas, etc. de tablas se utiliza SQL.
- Los comandos reservados se escriben con un '.' delante (.table, .quit, .help)
- Si tecleamos .help obtendremos los comandos disponibles

```
E:\BBDD\dbSQLite\sqlite3.exe
sqlite> .help
.auth ON|OFF          Show authorizer callbacks
.backup ?DB? FILE      Backup DB (default "main") to FILE
.bail on|off           Stop after hitting an error.  Default OFF
.binary on|off         Turn binary output on or off.  Default OFF
.cd DIRECTORY          Change the working directory to DIRECTORY
.changes on|off        Show number of rows changed by SQL
.check GLOB            Fail if output since .testcase does not match
.clone NEWDB           Clone data into NEWDB from the existing database
.databases             List names and files of attached databases
.dbinfo ?DB?          Show status information about the database
.dump ?TABLE? ...     Dump the database in an SQL text format
                      If TABLE specified, only dump tables matching
```



Ejemplo de uso de comandos:

```
sqlite> .open empresa.db
```

```
sqlite>
```

```
sqlite> .read confSQLite.sql
```

```
sqlite>
```

```
sqlite> CREATE TABLE DEPARTAMENTO(  
    ID            INTEGER            PRIMARY KEY    NOT NULL,  
    NOMBRE        TEXT              NOT NULL,  
    LOCALIZACION TEXT              NOT NULL);
```

```
sqlite>
```

```
sqlite> INSERT INTO DEPARTAMENTO (ID, NOMBRE, LOCALIZACION)  
VALUES ( 10, 'Contabilidad', 'Madrid' );
```

```
sqlite>
```

```
sqlite> INSERT INTO DEPARTAMENTO (ID, NOMBRE, LOCALIZACION)  
VALUES ( 20, 'Marketing', 'Sevilla' );
```

Bases de datos embebidas

Manejando SQLite



```
sqlite> CREATE TABLE EMPLEADO (  
    ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,  
    APELLIDO      TEXT  NOT NULL,  
    NOMBRE        TEXT  NOT NULL,  
    ROL           TEXT,  
    SALARIO       REAL,  
    COMISION      REAL,  
    ID_DEPT       INTEGER,  
    FOREIGN KEY(ID_DEPT) REFERENCES DEPARTAMENTO(ID));  
sqlite>  
sqlite> .output empresaDDL.sql  
sqlite>  
sqlite> .schema  
sqlite>  
sqlite> .mode insert DEPARTAMENTO  
sqlite> select * from DEPARTAMENTO;
```



Es una **BD relacional** de **código abierto** implementada en Java disponible bajo la licencia Apache 2.0

Sus principales ventajas son su tamaño reducido, está basada en **Java** y soporta los **estándares SQL**

Ofrece un controlador integrado **JDBC** que permite incrustar Derby en cualquier programa basado en Java

Soporta el paradigma **cliente-servidor** utilizando **Derby Network Server**

Es una BD **fácil de instalar, implementar y utilizar.**

Bases de datos embebidas

Instalación Apache Derby



Para realizar la instalación descargamos la última versión de la página

http://db.apache.org/derby/derby_downloads.html

Para windows descargamos db-derby-x.x.x.x-bin.zip y lo descomprimos.

Para utilizar Derby en los programas Java, será necesario tener accesible la librería **derby.jar** añadiéndola a nuestro proyecto en Eclipse.



Apache Derby trae una serie de **ficheros .BAT** que nos permitirán ejecutar órdenes para **crear nuestras bases de datos** y **ejecutar sentencias DDL y DML**.

El fichero **IJ.BAT** se encuentra en la carpeta **bin**. Para crear y acceder a las bases de datos ejecutaremos el fichero **IJ.BAT**

Para crear una base de datos utilizaremos el siguiente comando en la línea de comandos IJ:

```
ij> connect 'jdbc:derby:<PATH>\<NOMBD>;create=true';
```

Para salir de la línea de comandos IJ teclearemos `exit;`

http://db.apache.org/derby/papers/DerbyTut/ij_intro.html

Es un **SGBD relacional** escrito en **Java** distribuida bajo **licencia BSD**

La Suite ofimática **OpenOffice** lo incluye en su versión 2.0 para dar soporte a la aplicación **Base**

Es compatible con **SQL ANSI-92** y **SQL:2008**

Puede mantener los datos en **memoria** o en **ficheros en disco**

Permite integridad referencial, procedimientos almacenados en Java, triggers y tablas en disco de hasta 8GB

Desde la web <http://sourceforge.net/projects/hsqldb/files/> podemos descargarnos la última versión [hsqldb-x.x.x.zip](#)

Lo descomprimos y dentro de la carpeta hsqldb-x.x.x movemos la carpeta hsqldb donde almacenemos nuestras BBDD embebidas.

A continuación creamos una **carpeta** llamada ejemplo en la carpeta hsqldb.

Accedemos a la carpeta bin y ejecutamos **runManagerSwing**

Se abrirá una ventana desde la que **configuraremos** la **conexión**

Escribirnos en el campo **Setting Name** un nombre para la conexión

En la lista **Type** seleccionamos la opción **HSQL Database Engine Standalone**, para que la BD la tome de un fichero si existe y si no existe la cree

En la casilla de **URL** escribimos el nombre de la carpeta donde se almacenará la base de datos y el de la base de datos: ejemplo/ejemplo. Pulsamos OK

A continuación se abre una nueva ventana desde la que podemos **ejecutar comandos DDL y DML** para crear manipular objetos de nuestra BD

Para ejecutar una **sentencia SQL** pulsamos el botón **Execute**

Desde la opción de menú **View -> Refresh Tree** podemos actualizar el árbol de objetos

```
create table TABLA(num int, texto varchar(40));  
insert into TABLA values (1234,'ABCD');  
insert into TABLA values (5678,'EFGH');  
select * from TABLA;
```

Es un **SGBD relacional** programado en **Java**

Está disponible bajo la **licencia pública de mozilla** o la **eclipse public licenses**.

Desde la web

<http://www.h2database.com/html/download.html>

podemos descargarnos la última versión.

Descargamos la última versión estable zip para todas las plataformas h2-yyyy-mm-dd.zip

Descomprimos por ejemplo en C:\ y creará una carpeta con el nombre c:\h2

Ejecutaremos el fichero **c:\h2\bin\h2.bat** para arrancar la consola

Se abrirá el **navegador web** con la **consola de administración de H2**

Escribimos un nombre para la configuración de la BD:
jdbc:h2:C:/db/H2/ejemplo/ejemplo (si las carpetas no existieran las crea)

Pulsar el botón **guardar** y después el botón **conectar**

Para conectarnos a la BD utilizaremos el nombre que utilizamos en la configuración

Comprobamos la configuración pulsando el botón **probar la conexión**

Se creará la carpeta ejemplo en H2 y dentro los ficheros de nuestra BD

Una vez **conectados** se visualiza una **nueva pantalla**
Desde esta podremos **realizar operaciones sobre la BD**
Se muestran los **comandos más importantes**
Desde la zona de instrucciones SQL podremos escribir
las sentencias para crear tablas, insertar filas, etc.
Los botones **Ejecutar**(CTRL+ENTER) y **|>** nos permitirán
ejecutar las sentencias SQL que escribamos en el área
de instrucción SQL.
El botón **Desconectar** nos lleva a la pantalla inicial donde
elegimos la conexión

En el enlace **Preferencias** de la ventana inicial se pueden configurar diversos aspectos como: los clientes permitidos (locales/remotos), conexión segura (SSL), puerto del servidor Web o notificar las sesiones activas. La opción **Tools** presenta una serie de herramientas que se pueden utilizar sobre la base de datos: backup, restaurar base de datos, ejecutar scripts, convertir la base de datos en un script, encriptación, etc.

Es un motor de **BD orientado a objetos**. Se puede utilizar de forma **embebida** o en aplicaciones **cliente-servidor**

Está disponible para entornos **Java** y **.Net**

Dispone de **licencia dual GPL/comercial**

Algunas características interesantes son:

- Evita el problema del **desfase objeto-relacional**
- **No existe un lenguaje SQL** para la manipulación de datos, en lugar existen métodos delegados
- Se instala añadiendo un único fichero de librería (**JAR** para Java o **DLL** para .NET)
- Se crea un **único fichero .YAP con la BD**

Bases de datos embebidas

DB4O (DataBase for Objects)



Encontrarás el jar de db4o en Canvas

db4o-8.0.184.15484.jar

Descárgalo en un lugar reconocible e inclúyelo como jar en el próximo proyecto Java.

Una base de datos db4o es un único fichero con una extensión .yap o .db4o

Para realizar cualquier acción en la BD debemos manipular una instancia de **ObjectContainer** donde se define el fichero de BD

```
ObjectContainer db =  
Db4oEmbedded.openFile(Db4oEmbedded.newConfiguratio  
n(), "nombreDB.yap");
```

Después de operar con la BD debemos cerrarla

```
db.close();
```

Para almacenar objetos utilizaremos el método **db.store(object)**.

Para eliminar objetos utilizaremos el método **db.delete(object)**.

Para hacer una consulta simple utilizaremos el método **db.queryByExample(object)** que retornará un **ObjectSet**. El ObjectSet podremos recorrerlo de la siguiente manera:

```
while (os.hasNext()) {  
    System.out.println(os.next());  
}
```

En una consulta simple el objeto que le pasamos como parámetro al método **db.queryByExample(object)** puede ser un objeto vacío en cuyo caso el **ObjectSet** contendrá todos los objetos almacenados.

También podemos obtener todos los objetos almacenados pasándole como parámetro al método **db.queryByExample** la clase correspondiente al tipo de los objetos:

```
db.queryByExample(Object.class);
```

Para realizar una consulta por uno de los campos del objeto debemos crear un objeto vacío y parametrizar el campo por el que queremos filtrar.

Para realizar una modificación de un objeto primero debemos obtener el objeto a modificar utilizando una consulta simple:

```
ObjectSet result = db.queryByExample(object);  
Object found = (Object) result.next();  
found.setAttribute(valor);  
db.store(found);
```


La API de consulta SODA nos ofrece la posibilidad de realizar consultas de bajo nivel en db4o, permitiendo el acceso directo a los nodos de los gráficos de la consulta. Usando combinaciones de palabras clave de restricción y consulta SODA, se puede crear lo que se llama un "gráfico" de consulta. Un gráfico es una red de objetos que representan un segmento de datos.

La clase Query proporciona la ubicación del nodo para restringir o seleccionar.

La clase Constraint nos permite restringir los resultados de la consulta con el nodo IQuery actual

```
Query query = db.query();  
query.constrain(Objeto.class);  
Constraint const =  
query.descend("atributo").constraint("valor");
```

SODA Query Keywords

descend()	Move from one node to another
orderAscending()	Order the result ascending according to the current node.
orderDescending()	Order the result descending according to the current node.
execute()	Execute the query graph and return the objects at the current node.

SODA Constraint Keywords

and(Constraint)	Performs an AND (&&) comparison between 2 constraints.
contains()	For collection nodes, matches will contain the specified value. For string values, behaves as Like().
endsWith(bool)	For strings, matches will end with the supplied value.
equal()	Combine with Smaller and Greater to include the specified value. (e.g. >= or <=)
greater()	Matching values will be greater than or larger than the supplied value.
identity()	Matching values will be the same object instance as the supplied value. (referential equality).

like()	For strings, matching values will contain the supplied value anywhere within the match.
not()	Performs a negation comparison. Matching values will NOT equal the supplied value. Added to any other constraint keyword, this will reverse the result.
or(Constraint)	Performs an OR () comparison between 2 constraints.
smaller()	Matching values will be smaller or less than the specified value.
startsWith(bool)	For strings, matches will start with the supplied value.

Vamos a crear una clase **Person** para crear una BD de **objetos Person**

```
public class Person {  
    private String name;  
    private String city;  
  
    public Person(String name, String city) {  
        this.name = name;  
        this.city = city;  
    }  
  
    public Person() {  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String getCity() {  
        return city;  
    }  
  
    public void setCity(String city) {  
        this.city = city;  
    }  
}
```

El siguiente código muestra la clase **ExampleDb4o.java** que crea la BD (si no existe) y añade objetos Person en ella

```
import com.db4o.Db4oEmbedded;
import com.db4o.ObjectContainer;

public class ExampleDb4o {

    final static String BDPPer = "../db4o-8.0/DBPeople.yap";

    public static void main(String[] args) {

        ObjectContainer db = Db4oEmbedded.openFile(
            Db4oEmbedded.newConfiguration(), BDPPer);
        // Create people
        Person p1 = new Person("Juan", "Guadalajara");
        Person p2 = new Person("Ana", "Madrid");
        Person p3 = new Person("Luis", "Granada");
        Person p4 = new Person("Pedro", "Asturias");
        // Store objects Person in the data
        db.store(p1);
        db.store(p2);
        db.store(p3);
        db.store(p4);

        db.close();

    } //end main
} // end ExampleDb4o
```

En nuestro proyecto de Eclipse crearemos una carpeta **db4o-8.0**

Bases de datos embebidas



Leer registros de la base de datos DB4O

```
Person person = new Person(null, null);

ObjectContainer db = Db4oEmbedded.openFile(
    Db4oEmbedded.newConfiguration(), BDPer);

// run through the database

ObjectSet<Person> result = db.queryByExample(person);
if (result.size() == 0)
    System.out.println("there aren't any register of person...");
else {
    System.out.println("Number of registers: " + result.size());
    while (result.hasNext()) {
        Person p = result.next();
        System.out.println("Name: " + p.getName() + " city: "
            + p.getCity());
    }
}
```


Bases de datos embebidas



Buscar un registro de la base de datos DB4O

```
// Obtain objects person with name Juan
Person p1 = new Person("Juan", null);
ObjectSet<Person> result1 = db.queryByExample(p1);

if (result1.size() != 0) {
    person = (Person) result1.next();
    System.out.println("Name: " + person.getName());
} else
    System.out.println("No existe Juan...");

// Obtain objects Person with citiy Guadalajara
Person p2 = new Person(null, "Guadalajara");
ObjectSet<Person> result2 = db.queryByExample(p2);

if (result2.size() != 0) {
    person = (Person) result2.next();
    System.out.println(" city: " + person.getCity());
} else
    System.out.println("No existe Guadalajara...");
```

```
// Modify an object
// Look for the object

ObjectSet<Person> result3 = db.queryByExample(new Person("Juan", null));
if (result3.size() == 0)
    System.out.println("No existe Juan...");
else { // modify the city
    person = (Person) result3.next();
    person.setCity("Toledo");
    db.store(person); // modified city
    // show data
    result3 = db.queryByExample(new Person("Juan", null));
    person = (Person) result3.next();
    System.out.println("Name: " + person.getName() + "New city: "
        + person.getCity());
}
```

```
// Delete an object
ObjectSet<Person> result4 = db.queryByExample(new Person("Juan", null));
if (result4.size() == 0)
    System.out.println("No existe Juan...");
else { // modify the city
    person = (Person) result4.next();
    System.out.println("Registers to delete: " + person.getName()
        + " " + person.getCity());
    if (result4.size() > 1) { // many Juan registers
        while (result4.hasNext()) {
            person = result4.next();
            db.delete(person);
            System.out.println("Borrado");
        }
    } else db.delete(person);
}
```

```
Query query = db.query();
query.constrain(Person.class);
Constraint whereCiudad = query.descend("city").constrain("León");
query.descend("name").constrain("Ana").or(whereCiudad);
```

```
ObjectSet<Person> result = query.execute();
while (result.hasNext()) {
    System.out.println(result.next());
}
```

Otros ejemplos:

```
query.descend("name").constrain("Ana").not();
```

```
query.descend("name").orderAscending();
```

```
query.descend("points").constrain(new Integer(99)).greater();
```

Crea una BBDD db4o de nombre productos.yap e inserta objetos de tipo PRODUCTOS (nombre, precio, cantidad, peso, tamaño, EAN, rendimiento_de_ventas) en ella. Usa los métodos consultar, modificar, eliminar y queries con greater.

