2.1 INTRODUCCIÓN

En general el término de acceso a datos significa el proceso de recuperación o manipulación de datos extraídos de un origen de datos local o remoto. Los orígenes de datos no tienen por qué ser relacionales y pueden provenir de muchas fuentes distintas. Algunos de los orígenes de datos con los que podemos encontrarnos son: una base de datos relacional remota en un servidor, una base de datos relacional local, una hoja de cálculo, un fichero de texto en nuestro ordenador, un servicio de información online, etc.

En esta unidad nos centraremos en los orígenes de datos relacionales, aprenderemos a realizar programas Java para acceder a una base de datos relacional. Para ello necesitaremos los **conectores**, que no son más que el software que se necesita para realizar las conexiones desde nuestro programa Java con una base de datos relacional.

2.2 EL DESFASE OBJETO-RELACIONAL

Actualmente las bases de datos orientadas a objetos están ganando cada vez más aceptación frente a las bases de datos relacionales, ya que solucionan las necesidades de aplicaciones más sofisticadas que requieren el tratamiento de elementos más complejos. Un ejemplo de estas son las aplicaciones para diseño y fabricación en ingeniería, los 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 Programación Orientada a Objetos* (POO) cuyos elementos complejos (nombrados anteriormente) son los Objetos.

En este sentido, las bases de datos relacionales no están diseñadas para almacenar estos objetos ya que existe un desfase entre las construcciones típicas que proporciona el modelo de datos relacional y las proporcionadas por los ambientes de programación basados en objetos; es decir, al guardar los datos de un programa bajo el enfoque orientado a objetos se incrementa la complejidad del programa dando lugar a más código y más esfuerzo de programación debido a la diferencia de esquemas entre los elementos a almacenar (objetos) y las características del repositorio de la base de datos (tablas). A esto es a lo que se denomina **desfase objeto-relacional** (o *desajuste de la impedancia*) y se refiere a los problemas que ocurren debido a las diferencias entre el modelo de datos de la base de datos y el del lenguaje de programación orientado a objetos.

Sin embargo, el paradigma relacional y el paradigma orientado a objetos pueden ser "amigos". Cada vez que los objetos deben extraerse o almacenarse en una base de datos relacional se requiere un mapeo desde las estructuras provistas en el modelo de datos a las provistas por el entorno de programación. Este tema se trata más ampliamente en la Unidad 3.

2.3 BASES DE DATOS EMBEBIDAS

Cuando desarrollamos pequeñas aplicaciones en las que no vamos a almacenar grandes cantidades de información 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 esté 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.

Por lo general, este tipo de bases de datos vienen del movimiento Open Source, aunque también hay algunas de origen propietario. Veamos algunas de ellas.

2.3.1 SQLITE

SQLite es un sistema gestor de base de datos multiplataforma escrito en C que proporciona un motor muy ligero. Las bases de datos se guardan en forma de ficheros por lo que es fácil trasladar la base de datos con la aplicación que la usa. Cuenta con una utilidad que nos permitirá ejecutar comandos SQL contra una base de datos SQLite en modo consola. Es un proyecto de dominio público.

La biblioteca implementa la mayor parte del estándar SQL-92, incluyendo transacciones de base de datos atómicas, consistencia de base de datos, aislamiento y durabilidad, triggers (o disparadores) y la mayor parte de las consultas complejas. Los programas que utilizan la funcionalidad de SQLite lo hacen a través de llamadas simples a subrutinas y funciones. SQLite se puede utilizar desde programas en C/C++, PHP, Visual Basic, Perl, Delphi, Java, etc.

Su instalación es sencilla. Desde la página http://www.sqlite.org/download.html se puede descargar. Para sistemas Windows podemos descargar el fichero ZIP *sqlite-shell-win32-x86-3070900.zip*, al descomprimirlo obtenemos un único fichero ejecutable *(sqlite3.exe)*. Al ejecutarlo desde la línea de comandos escribimos el nombre del fichero que contendrá la base de datos, si el fichero no existe se creará, si existe cargará la base de datos. El siguiente ejemplo crea la base de datos *ejemplo.db* (en la carpeta *D:\db\SQlite*), todas las tablas que creemos en esta sesión se almacenarán en este fichero, para finalizar se escribe el comando *.quit*:

```
D:\>sqlite3 D:\db\SQlite\ejemplo.db
SQLite version 3.7.9 2011-11-01 00:52:41
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> BEGIN TRANSACTION;
sqlite> CREATE TABLE departamentos (
   ...>  dept_no  TINYINT(2) NOT NULL PRIMARY KEY,
   ...>  dnombre  VARCHAR(15),
   ...>  loc      VARCHAR(15)
   ...> );
sqlite>
sqlite>
sqlite> INSERT INTO departamentos VALUES (10,'CONTABILIDAD','SEVILLA');
sqlite> INSERT INTO departamentos VALUES (20,'INVESTIGACIÓN','MADRID');
sqlite> INSERT INTO departamentos VALUES (30,'VENTAS','BARCELONA');
sqlite> INSERT INTO departamentos VALUES (40,'PRODUCCIÓN','BILBAO');
sqlite> COMMIT;
sqlite> .quit
```

Para instalarlo desde Linux escribimos desde la línea de comandos: `$sudo apt-get install sqlite3`. Y para ejecutarlo escribimos lo siguiente para crear la base de datos en la carpeta */home/usuario/db/SQLite*:

```
$ sqlite3 /home/usuario/db/SQLite/ejemplo.db
```

Actividad 1: Crea las tablas EMPLEADOS y DEPARTAMENTOS en SQLite e inserta filas en ellas. La descripción de las tablas es la siguiente:

DEPARTAMENTOS:	EMPLEADOS:
DEPT_NO numérico, clave primaria DNOMBRE VARCHAR(15), LOC VARCHAR(15)	EMP_NO numérico, clave primaria APELLIDO VARCHAR(10), OFICIO VARCHAR(10), DIR numérico, FECHA_ALT DATE, SALARIO numérico, COMISION numérico, DEPT_NO numérico, clave ajena, referencia a DEPARTAMENTOS

2.3.2 APACHE DERBY

Apache Derby, es una base de datos relacional de código abierto, implementado en su totalidad en Java que forma parte del *Apache DB subproject* y está disponible bajo la licencia Apache, versión 2.0. Algunas ventajas de esta base de datos son: su tamaño reducido, está basada en Java y soporta los estándares SQL, ofrece un controlador integrado JDBC que permite incrustar Derby en cualquier solución basada en Java, soporta el tradicional paradigma cliente-servidor utilizando el *Derby Network Server*, es fácil de instalar, implementar y utilizar.

Para realizar la instalación descargamos la última versión desde la página web: http://db.apache.org/derby/derby_downloads.html. En Windows descargamos el fichero: *db-derby-10.8.2.2-bin.zip*, y lo descomprimimos por ejemplo en *D:\db-derby-10.8.2.2-bin*. A partir de ahora, para poder utilizar Derby en nuestros programas Java, solo será necesario tener accesible la librería *derby.jar* en el CLASSPATH de nuestro programa.

Ejemplos para configuar la variable CLASSPATH:
Windows:
```
C:\> SET DERBY_INSTALL=D:\db-derby-10.8.2.2-bin
C:\> SET CLASSPATH=DERBY_INSTALL\lib\derby.jar;DERBY_INSTALL\lib\derbytools.jar;.
```
Linux:
```
$ export DERBY_INSTALL=/opt/db-derby-10.8.2.2-bin
$ export CLASSPATH=$DERBY_INSTALL/lib/derby.jar:$DERBY_INSTALL/lib/derbytools.jar:.
```
Para visualizar la variable CLASSPATH:
Windows:
```
C:\> echo %CLASSPATH%
```
Linux:
```
$ $CLASSPATH
```


Apache Derby trae una serie de ficheros .BAT que nos permitirán ejecutar por consola órdenes para crear nuestras bases de datos y ejecutar sentencias DDL y DML. El fichero es **IJ.BAT** y se encuentra en la carpeta bin *(D:\db-derby-10.8.2.2-bin\bin)*. Desde la línea de comandos del DOS nos dirigimos a dicha carpeta y ejecutamos el fichero **IJ.BAT:**

```
D:\db-derby-10.8.2.2-bin\bin>ij
```

Para crear una base de datos de nombre *ejemplo* en la carpeta *D:\db\Derby* escribimos desde el indicador **ij>** la siguiente orden:

```
connect 'jdbc:derby:D:\db\Derby\ejemplo;create=true';
```

Donde:

- *connect*, es el comando para establecer la conexión.
- *jdbc:derby*, es el protocolo JDBC especificado por DERBY.
- *ejemplo* es el nombre de la base de datos que voy 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 la línea de comandos de IJ, escribimos `exit;`

El siguiente ejemplo muestra la creación de la base de datos *ejemplo*, la creación de la tabla DEPARTAMENTOS, la inserción de filas en la tabla y la ejecución del script *Empleados.sql* que crea la tabla EMPLEADOS e inserta filas en ella. Al final se ejecuta la orden *exit* para salir:

```
D:\db-derby-10.8.2.2-bin\bin>ij
Versión ij 10.8
ij> connect 'jdbc:derby:D:\db\Derby\ejemplo;create=true';
ij> CREATE TABLE departamentos (
> dept_no   INT NOT NULL PRIMARY KEY,
> dnombre   VARCHAR(15),
> loc       VARCHAR(15)
> );
0 filas insertadas/actualizadas/suprimidas
ij> INSERT INTO departamentos VALUES (10,'CONTABILIDAD','SEVILLA');
1 fila insertada/actualizada/suprimida
ij> INSERT INTO departamentos VALUES (20,'INVESTIGACIÓN','MADRID');
1 fila insertada/actualizada/suprimida
ij> INSERT INTO departamentos VALUES (30,'VENTAS','BARCELONA');
1 fila insertada/actualizada/suprimida
ij> INSERT INTO departamentos VALUES (40,'PRODUCCIÓN','BILBAO');
1 fila insertada/actualizada/suprimida
ij> commit;
ij> run 'Empleados.sql';
ij> CREATE TABLE empleados (
 emp_no     INT  NOT NULL PRIMARY KEY,
 apellido   VARCHAR(10),
 oficio     VARCHAR(10),
 dir        INT,
```

```
    fecha_alt DATE,
    salario   FLOAT,
    comision  FLOAT,
    dept_no   INT NOT NULL REFERENCES departamentos(dept_no)
);
0 filas insertadas/actualizadas/suprimidas
ij> INSERT INTO empleados VALUES (7369,'SANCHEZ','EMPLEADO',7902,'1990-12-
17',1040,NULL,20);
1 fila insertada/actualizada/suprimida
ij> INSERT INTO empleados VALUES (7499,'ARROYO','VENDEDOR',7698,'1990-02-20',
1500,390,30);
1 fila insertada/actualizada/suprimida
ij> exit;
```

Para volver a usar la base de datos escribiríamos la siguiente orden desde la línea de comandos de IJ: *connect 'jdbc:derby:D:\db\Derby\ejemplo';*.

Para realizar la instalación en Linux (Ubuntu) seguimos los siguientes pasos. En primer lugar es necesario instalar los paquetes JDK y JAVADB de Java (si no los tenemos instalados) ejecutando la siguiente orden desde la línea de comandos:

```
sudo apt-get install sun-java6-jdk sun-java6-plugin sun-java6-javadb
```

En algunas versiones de Linux (Ubuntu) puede no encontrar el paquete debido a un cambio de licencias de Oracle, en ese caso antes hay que añadir un nuevo repositorio y después actualizar el sistema:

```
sudo add-apt-repository ppa:ferramroberto/java
sudo apt-get update
```

A continuación descargamos la versión para Linux y la descomprimimos en la carpeta /opt, en este caso se ha descargado la versión: *db-derby-10.8.2.2-bin.tar.gz*. Se creará la carpeta */opt/db-derby-10.8.2.2-bin*. Configuramos la variable DERBY_INSTALL y DERBY_HOME con el nombre de carpeta donde se ha descargado, ejecutando desde la línea de comandos:

```
$ export DERBY_INSTALL=/opt/db-derby-10.8.2.2-bin
$ export DERBY_HOME=/opt/db-derby-10.8.2.2-bin
```

Para usar Derby necesitamos incluir en el CLASSPATH los ficheros JAR: *derby.jar* y *derbytools.jar*, escribimos las siguientes órdenes:

```
$ export CLASSPATH=$DERBY_INSTALL/lib/derby.jar:$DERBY_INSTALL/lib/derbytools.jar:.
```

A continuación nos dirigimos a la carpeta donde está instalado Derby y ejecutamos *setEmbeddedCP*:

```
$ cd $DERBY_INSTALL/bin
$ ./setEmbeddedCP
```


Desde aquí ya podemos utilizar la utilidad IJ para crear nuestra base de datos escribiendo desde la línea de comandos: `java org.apache.derby.tools.ij`. Lo primero que se visualiza es la versión. El siguiente ejemplo muestra la creación de la base de datos *ejemplo* en la carpeta */home/usuario/db/Derby/*, una vez creada finalizamos la conexión ejecutando la orden *exit*:

```
$ java org.apache.derby.tools.ij
Versión ij 10.8
ij> connect 'jdbc:derby:/home/usuario/db/Derby/ejemplo;create=true';
Mon Jan 02 21:37:24 CET 2012 Thread[main,5,main] java.io.FileNotFoundException: derby.
log (Permission denied)
------------------------------------------------------------
Mon Jan 02 21:37:25 CET 2012:
 Arrancando Derby versión The Apache Software Foundation - Apache Derby - 10.8.2.2 -
(1181258): instancia a816c00e-0134-a024-0bfb-000000be33d0
en el directorio de base de datos /home/usuario/db/Derby/ejemplo  con el cargador de
clases sun.misc.Launcher$AppClassLoader@7d772e

java.vendor=Sun Microsystems Inc.
java.runtime.version=1.6.0_21-b06
user.dir=/opt/db-derby-10.8.2.2-bin/bin
derby.system.home=null

Cargador de clases de base de datos iniciado - derby.database.classpath=''
ij> exit;
------------------------------------------------------------
Mon Jan 02 21:40:55 CET 2012: cerrando motor de Derby
------------------------------------------------------------
Mon Jan 02 21:40:57 CET 2012:
Cerrando la instancia a816c00e-0134-a024-0bfb-000000be33d0 en el directorio de
base de datos /home/usuario/db/Derby/ejemplo con cargador de clases sun.misc.
Launcher$AppClassLoader@7d772e
------------------------------------------------------------
$
```

Para volver a usar la base de datos escribiríamos la siguiente orden desde la línea de comandos de IJ: *connect 'jdbc:derby:/home/usuario/db/Derby/ejemplo';*. El resto de comandos SQL se usan igual que se usaron en el ejemplo para Windows.

Actividad 2: Crea las tablas EMPLEADOS y DEPARTAMENTOS en Apache Derby e inserta filas en ellas.

2.3.3 HSQLDB

HSQLDB (*Hyperthreaded Structured Query Language Database*) es un sistema gestor de bases de datos relacional escrito en Java. La suite ofimática OpenOffice lo incluye desde su versión 2.0 para dar soporte a la aplicación *Base*. Es compatible con SQL ANSI-92 y SQL: 2008. Puede mantener la base de datos en memoria o en ficheros en disco. Permite integridad referencial (claves foráneas),

procedimientos almacenados en Java, disparadores y tablas en disco de hasta 8GB. Se distribuye con licencia BSD (*Berkeley Software Distribution*) que es una licencia muy cercana al dominio público.

Desde la web http://sourceforge.net/projects/hsqldb/files/ podemos descargarnos la última versión. En Windows se ha descargado el fichero *hsqldb-2.2.6.zip*. Lo descomprimimos por ejemplo en la unidad D, se creará una carpeta con el nombre del fichero ZIP (*D:\hsqldb-2.2.6\hsqldb*), dentro de esa carpeta hay una con el nombre **hsqldb**, la llevamos a la unidad D para que nos quede instalado en *D:\hsqldb*. A continuación creamos una carpeta en *D:\hsqldb\data* para guardar los datos de la base de datos que vamos a crear, la llamamos *ejemplo*, nos debe quedar: *D:\hsqldb\data\ejemplo*.

Abrimos la línea de comandos del DOS y nos dirigimos a la carpeta *D:\hsqldb\bin*, ejecutamos el fichero BAT de nombre *runUtil* con el parámetro *DatabaseManager*, para conectarnos a la base de datos y que se ejecute la interfaz gráfica de este sistema gestor de base datos:

```
D:\hsqldb\bin>runUtil DatabaseManager
```

Se abre una ventana desde la que tenemos que configurar la conexión, escribimos en el campo *Setting Name* un nombre para la conexión, de la lista *Type* seleccionamos la opción *HSQL Database Engine Standalone*, para que la base de datos la tome de un fichero si existe y si no existe la cree; y 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 el botón *OK*, véase Figura 2.1.

Figura 2.1. Configurar conexión en HSQLDB.

A continuación se abre una nueva ventana desde la que podemos ejecutar comandos DDL y DML para crear y manipular objetos de nuestra base de datos, véase Figura 2.2. Para ejecutar una sentencia SQL pulsamos el botón *Execute*. Desde la opción de menú *View->Refresh Tree* podemos actualizar el árbol de objetos.

Figura 2.2. Ejecución de sentencias SQL en HSQLBD.

Para instalarlo en Ubuntu primero guardamos el fichero *hsqldb-2.2.6.zip* en la carpeta */opt*. Después desde la línea de comandos nos vamos a dicha carpeta y lo descomprimimos ejecutando la siguiente orden:

```
$ sudo unzip hsqldb-2.2.6.zip
```

Se creará en *opt* una carpeta con nombre *hsqldb-2.2.6*, dentro hay otra carpeta que se llama *hsqldb*, la movemos a */opt*. Al final nos debe quedar */opt/hsqldb*. A continuación escribo desde la línea de comandos las siguientes órdenes para establecer el PATH con las librerías de hsqldb en la variable CLASSPATH:

```
$ CLASSPATH="$CLASSPATH:/opt/hsqldb/lib/hsqldb.jar"
$ export CLASSPATH
```

Creamos en la carpeta *home/usuario/db/Hsqldb* la carpeta *ejemplo* (nos debe quedar */home/usuario/db/Hsqldb/ejemplo*) para almacenar todos los datos de la base de datos *ejemplo* que crearemos a continuación. Seguidamente ejecutamos desde la línea de comandos la siguiente orden:

```
$ java org.hsqldb.util.DatabaseManager
```

Se abre la ventana de conexión, similar a la mostrada en la Figura 2.1. Rellenamos los campos como se hizo anteriormente. En el campo URL escribimos la carpeta donde se almacenará la base de datos y su nombre, nos debe quedar: *jdbc:hsqldb:file:/home/usuario/db/Hsqldb/ejemplo*. El resto de operaciones son similares.

Actividad 3: Crea las tablas EMPLEADOS y DEPARTAMENTOS en HSQLDB e inserta filas en ellas.

2.3.4 H2

H2 es un sistema gestor de base de datos relacional programado íntegramente en Java. Está disponible como software de código libre bajo la *Licencia Pública de Mozilla* o la *Eclipse Public License*. Desde la web http://sourceforge.net/projects/hsqldb/files/ podemos descargarnos la última versión. Para probarla descargamos la versión ZIP para todas las plataformas *h2-2011-11-26.zip*. En Windows lo descomprimimos por ejemplo en la unidad D se creará una carpeta con el nombre *D:/h2*. Desde la línea de comandos de Windows nos dirigimos a la carpeta *D:\h2\bin* y ejecutamos el fichero **h2.bat** para arrancar la consola:

```
D:\h2\bin>h2
```

Se abre el navegador web con la consola de administración de H2, véase Figura 2.3. Escribimos un nombre para la configuración de la base de datos, en el campo URL JDBC escribimos la URL para la conexión a nuestra base de datos: *jdbc:h2:D:/db/H2/ejemplo/ejemplo* (si las carpetas no existen las crea), pulsamos el botón *Guardar* para que guarde la configuración y a continuación el botón

Conectar. Cada vez que queramos conectarnos a nuestra base de datos usaremos el nombre dado a la configuración. Podemos pulsar el botón *Probar la conexión* antes de conectar para ver si todo ha ido bien, entonces se mostrará el mensaje: *Prueba correcta*. Se creará la carpeta *ejemplo* en *H2* y dentro los ficheros de nuestra base de datos.

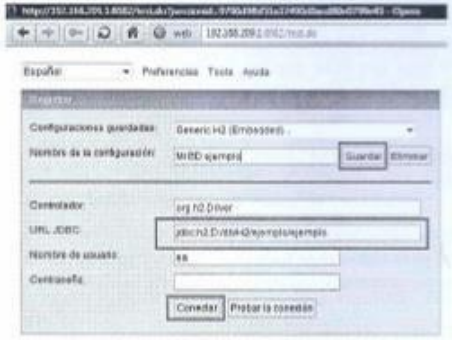

Figura 2.3. Establecer conexión en H2.

Una vez conectados se visualiza una nueva pantalla, Figura 2.4, desde la que podremos realizar las operaciones sobre la base de datos. Se muestran los comandos más importantes y un script de ejemplo. Desde la zona de instrucciones SQL podremos escribir las sentencias para crear tablas, insertar filas, etc., véase Figura 2.5.

Figura 2.4. Pantalla de manejo de la base de datos en H2.

Los botones Ejecutar (Ctrl+Enter) y O nos permitirán ejecutar las sentencia SQL que escribamos en el área de instrucción SQL. El botón *Desconectar* nos lleva a la pantalla inicial donde elegimos la conexión.

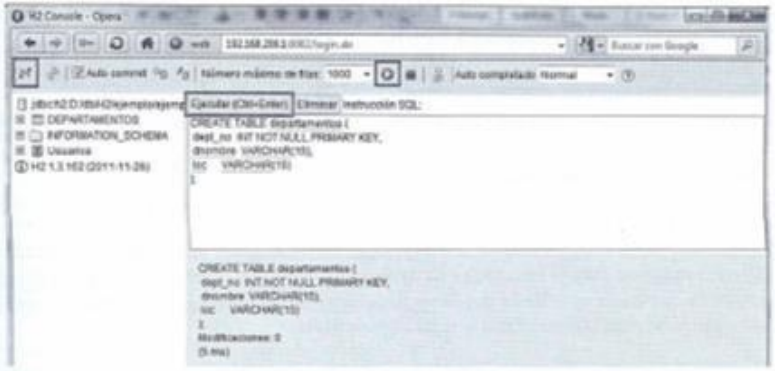

Figura 2.5. Ejecución de sentencias en H2.

Desde el enlace *Preferencias* de la ventana inicial se pueden configurar diversos aspectos como: los clientes permitidos (locales/remotos), conexión segura (uso de 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.

En Linux Ubuntu extraemos el fichero *h2-2011-11-26.zip* en la carpeta *opt* de esta manera tendremos */opt/h2*. Nos dirigimos a la carpeta */opt/h2/bin* y ejecutamos el fichero **h2.sh** para arrancar la consola (también se puede ejecutar desde el entorno gráfico), desde la línea de comandos escribiríamos:

```
/opt/h2/bin$ ./h2.sh
```

Si se visualiza el mensaje de permiso denegado ejecutamos las siguientes órdenes para dar permiso de ejecución y después para ejecutar el fichero **h2.sh**:

```
/opt/h2/bin$ sudo chmod +x h2.sh
/opt/h2/bin$ ./h2.sh
```

El resto de pasos son similares a los vistos anteriormente, salvo la escritura de la URL. En el campo URL JDBC podemos escribir lo siguiente *jdbc:h2:/home/usuario/db/H2/ejemplo/ejemplo*, para que nos guarde la base de datos en la carpeta */home/usuario/db/H2/ejemplo*, se crearían las carpetas, si no existen, y dentro los ficheros para la base de datos de nombre *ejemplo*.

Actividad 4: Crea las tablas EMPLEADOS y DEPARTAMENTOS en H2 e inserta filas en ellas.

2.3.5 DB4O

Db4o (*DataBase 4 (for) Objects*) es un motor de base de datos 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. Proporciona algunas características interesantes:

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

Para utilizar Db4o nos dirigimos a la web http://www.db4o.com/, descargamos la última versión y la descomprimimos. Para el ejemplo se ha descargado la versión *db4o-8.0-java.zip*. Al descomprimirla hay una carpeta de nombre *lib* donde se encuentra los JAR (*db4o-.jar*) que tenemos que agregar a nuestro proyecto para utilizar el motor de la base de datos.

Desde Eclipse podemos crear una librería con todos los JAR para ello seleccionamos nuestro proyecto, pulsamos el botón derecho del ratón y seleccionamos **Build Paths-> Add Libreries.** Se visualiza una ventana desde la que hemos de elegir la opción **User Library** y pulsar el botón *Next*. A continuación se visualiza una nueva ventana desde la que pulsamos el botón *User Libraries*, a continuación pulsamos el botón *New* para dar un nombre a la librería, por ejemplo *Db4oLib*. Seguidamente pulsamos el botón *Add JARs*, localizamos todos los *db4o-.jar* de la carpeta *lib* y pulsamos el botón *Abrir*, véase Figura 2.6. A continuación *OK* y para finalizar pulsamos *Finish*. Aunque no es necesario añadir todos los JAR, bastaría con añadir el JAR *db4o-8.0.224.15975-core-java5.jar*.

La librería quedará integrada en nuestro proyecto Eclipse. La instalación en Linux es similar. Si no usamos entorno gráfico para nuestros programas Java hemos de incluir en la variable CLASSPATH los JAR necesarios.

Figura 2.6. Añadir librerías Db4o en Eclipse.

El siguiente ejemplo muestra un programa Java que crea crea una base de datos y almacena objetos Persona en ella. Se ha definido la clase **Persona.java** formada por los atributos *nombre* y *ciudad* y los métodos *get* y *set* para obtener y almacenar los valores de un registro Persona:

```
public class Persona {
  private String nombre;
  private String ciudad;

  public Persona(String nombre,String ciudad)  {
  this.nombre=nombre;
  this.ciudad=ciudad;
  }
  public Persona() {
  this.nombre=null;
  this.ciudad=null;
  }
  public String getNombre(){return nombre;}
  public void setNombre(String nom){nombre=nom;}

  public String getCiudad(){return ciudad;     }
  public void setCiudad(String dir){ciudad=dir;}

}//fin Persona
```

Desde Eclipse para generar automáticamente los *getters* y los *setters* de los atributos pulsamos con el botón derecho del ratón en el código de la clase, seleccionamos la opción *Source* y a continuación *Generate Getters and Setters*, véase Figura 2.7. A continuación hemos de seleccionar los atributos y pulsar el botón *OK*.

Figura 2.7. Generar *Getters* y *Setters*.

El siguiente código muestra la clase **Main.java** que crea la base de datos (si no existe) e inserta objetos Persona en ella:

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

public class Main {
     final static String BDPer = "D:/eclipse/bd_db4o/DBPersonas.yap";

     public static void main(String[] args) {
            ObjectContainer db= Db4oEmbedded.openFile
                                (Db4oEmbedded.newConfiguration(),BDPer);

            // Creamos Personas
            Persona p1 = new Persona("Juan", "Guadalajara");
            Persona p2 = new Persona("Ana", "Madrid");
            Persona p3 = new Persona("Luis", "Granada");
            Persona p4 = new Persona("Pedro", "Asturias");

            //Almacenar objetos Persona en la base de datos
            db.store(p1);
            db.store(p2);
            db.store(p3);
            db.store(p4);

            db.close();          //cerrar base de datos

     }//fin de main
}//fin de la clase Main
```

Para realizar cualquier acción, ya sea insertar, modificar o realizar consultas debemos manipular una instancia de ***ObjectContainer*** donde se define el fichero de base de datos, en el ejemplo el fichero se llama *DBPersonas.yap* y se almacena en la variable *BDPer* donde será necesario incluir el trayecto donde se encuentra el fichero. Algunos de los métodos más importantes son:

- Para abrir la base de datos llamamos al método ***openFile( )***:
  `ObjectContainer db = Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(),BDPer);`
- Para cerrarla llamamos a ***close()***:
  `db.close();`
- Para almacenar objetos utilizamos ***store()***:
  `db.store(p1);`
- Para recuperar objetos podemos utilizar la interfaz Query-By-Example ***queryByExample()***. El siguiente ejemplo muestra todos los objetos Persona existentes en la base de datos, si no existe ninguno el método **size()** aplicado al objeto **ObjectSet** devuelve 0:

```
Persona per = new Persona(null,null);
ObjectSet<Persona> result = db.queryByExample(per);
if(result.size() == 0) System.out.println("No existen Registros de Personas..");
      else{
```



```
System.out.println("Número de registros: "+result.size());

while(result.hasNext()){
   Persona p = result.next();
   System.out.println("Nombre:" + p.getNombre()+
                      ", Ciudad:" + p.getCiudad());
}

db.close();    //cerrar base de datos
```

El siguiente ejemplo obtendría los objetos Persona cuyo nombre es Juan:

```
Persona per = new Persona("Juan",null);
ObjectSet<Persona> result = db.queryByExample(per);
```

El siguiente ejemplo obtendría los objetos Persona cuya ciudad es Guadalajara:

```
Persona per = new Persona (null,"Guadalajara");
ObjectSet<Persona> result = db.queryByExample(per);
```

Para modificar un objeto primero hay que localizarlo y después se modifica con ***store()***. El siguiente ejemplo modifica la ciudad de Juan a Toledo y luego visualiza sus datos:

```
ObjectSet<Persona> result = db.queryByExample(new Persona("Juan",null));
if(result.size() == 0) System.out.println("No existe Juan…");
else{
        Persona existe = (Persona) result.next();
        existe.setCiudad("Toledo");
        db.store(existe); //ciudad modificada
        //consultar los datos
        result = db.queryByExample(new Persona("Juan",null));
        existe = (Persona) result.next();
        System.out.println("Nombre:" +existe.getNombre()+
                           " Nueva Ciudad: "+ existe.getCiudad());
```

Para eliminar objetos utilizamos ***delete()***, antes será necesario localizar el objeto a eliminar. El siguiente ejemplo elimina todos los objetos cuyo nombre sea Juan:

```
ObjectSet<Persona> result = db.queryByExample(new Persona("Juan",null));
if(result.size() == 0) System.out.println("No existe Juan…");
else {
   Persona existe = (Persona) result.next();
   System.out.println("Registros a borrar: "+ result.size());
   if (result.size()>1) {  //varios objetos con nombre Juan
         while(result.hasNext()){
             Persona p = result.next();
             db.delete(p);
             System.out.println("Borrado...." );
         }
   } else     db.delete(existe);     //solo hay un objeto con nombre Juan
```

En Linux cambiaríamos la localización del fichero, por ejemplo:

```
final static String BDPer = "/home/usuario/Java/Db4o/DBPersonas.yap";
```

Si queremos ejecutar desde la línea de comandos los programas Java tendríamos que modificar la variable CLASSPATH. En el siguiente ejemplo los ficheros Java están en la carpeta */home/usuario/Java/Db4o*, dentro está la carpeta *lib* que contiene los JAR de Db4o, añadiremos al CLASSPATH el fichero *db4o-8.0.224.15975-core-java5.jar*, después se compilarían los ficheros y al final se ejecutaría Main:

```
/Java/Db4o$ export CLASSPATH=$CLASSPATH:/home/usuario/Java/Db4o/lib/db4o-8.0.224.15975-
core-java5.jar
/Java/Db4o$ javac Persona.java
/Java/Db4o$ javac Main.java
/Java/Db4o$ java Main
```

En la Unidad 4 se profundizará más acerca de las bases de datos orientadas a objetos.

Actividad 5: Crea una base de datos Db4o de nombre EMPLEDEP.YAP e inserta objetos EMPLEADOS y DEPARTAMENTOS en ella. Después obtén todos los objetos empleado de un departamento concreto. Visualiza también el nombre de dicho departamento.

2.3.6 OTRAS

Existen más sistemas de bases de datos embebidos tanto en software libre como en sistemas propietarios. Algunos ejemplos son:

- **Firebird** que se deriva del código fuente de *InterBase 6.0*, de *Borland*. Es un sistema gestor de bases de datos relacional de código abierto que no tiene licencias duales, por lo que es totalmente libre y se puede usar tanto en aplicaciones comerciales como de código abierto. Se presenta en tres versiones de servidor: *SuperServer*, *Classic* y *Embedded*. La edición embebida (*Embedded*) es un completo servidor Firebird empacado en unos cuantos ficheros. Es fácil distribuir aplicaciones, puesto que no requiere instalación, ideal para crear catálogos en CDROM, versiones mono usuario, de evaluación o portátiles de las aplicaciones. Algunas de sus características son:
  - Completo soporte para procedimientos almacenados y disparadores.
  - Integridad referencial.
  - Bajo consumo de recursos.
  - Lenguaje interno para procedimientos almacenados y disparadores (PSQL).
  - Soporte para funciones externas.
  - Poca o ninguna necesidad de administradores especializados.
  - Múltiples formas de acceder a la base de datos: nativo/API, drivers dbExpress, ODBC, OLEDB, proveedor .Net, driver JDBC nativo tipo 4, módulo Python, PHP, Perl, etc.
  - Etc.

- **SQL Server Mobile** (*Microsoft SQL Server 2005 Mobile Edition*): Es una base de datos compacta y con una gran variedad de funciones diseñada para admitir una amplia lista de dispositivos inteligentes y Tablet PC. Es un producto de Microsoft que incluye varias características de las bases de datos relacionales a la vez que ocupa poco espacio. Algunas características son:
  - Un motor de base de datos compacto y un sólido optimizador de consultas.
  - Compatibilidad con la réplica de mezcla y el acceso a datos remotos.
  - Integración con Microsoft SQL Server 2005.
  - Las herramientas de administración son Microsoft SQL Server Management Studio y SQL Server Management Studio Express.
  - Integración con Microsoft Visual Studio 2005.
  - Acceso a datos remotos y réplica de mezcla para sincronizar datos.
  - Microsoft Proveedor de datos .NET Framework y .NET Compact Framework para SQL Server Compact Edition (System.Data.SqlServerCe).
  - Compatibilidad con Microsoft ADO.NET y el proveedor de OLE DB para SQL Server Compact Edition.
  - Un subconjunto de sintaxis SQL.
  - Se implementa como una base de datos incrustada en equipos de escritorio, dispositivos móviles y Tablet PC.
  - Compatibilidad con la tecnología de implementación ClickOnce.
- **Oracle Embedded**: bajo este nombre Oracle ofrece un conjunto de soluciones al desarrollo software que aporta ciertas ventajas como la instalación "silenciosa" (se instala solo sin tener que hacer clic), la configuración automática o el funcionamiento desatendido y menores costes de implementación, licencia, hardware y administración. La tecnología Embedded de Oracle ofrece elementos especializados y una gran libertad de elección según el tipo de necesidades. La oferta de bases de datos permite elegir entre tres elementos:
  - ***Oracle Database 11g***: para aplicaciones empresariales.
  - ***Oracle TimesTen In-Memory Database***: para aplicaciones en tiempo real que necesitan un tiempo de respuesta de microsegundos. *TimesTen* es una base de datos relacional que se ejecuta en memoria y que gracias a ello ofrece unos altísimos tiempos de respuesta y una caché de datos en tiempo real. Está especialmente diseñada para entornos de misión crítica.
  - ***Oracle Berkeley DB:*** proporciona a los desarrolladores una base de datos embebida permitiendo varias opciones de almacenamiento: clave/valor, SQL, XML / XQuery o Java Object para su modelo de datos. Entre sus características cabe destacar que se puede ejecutar en disco o en memoria y proporciona un alto rendimiento. Algunas características de Oracle Berkeley DB son:
    - Rendimiento extraordinario, elimina los gastos de la comunicación interprocesos y SQL.
    - Administración cero, toda la administración se realiza a través de una API, se integra por completo en la aplicación y es invisible para los usuarios finales.
    - En dispositivos móviles, el tamaño de las librerías es de menos de 1MB y el tiempo de ejecución de los requisitos de memoria dinámica es de solo unos pocos KB.
    - Bajo coste total de propiedad, al ser embebida se reducen los gastos de implementación, licencias y hardware, y los costes de administración.

El siguiente ejemplo en Java muestra el uso de una base de datos Berkeley para almacenar pare clave/valor. Para poder ejecutarlo hemos de agregar el fichero *je-5.0.34.jar* a nuestro proyecto Eclipse a nuestro CLASSPATH, este se puede conseguir descargando la versión ZIP de *Berkeley DB Java Edition 5.0.34* (*je-5.0.34.zip*) desde la URL de Oracle: http://www.oracle.com/technetwork/database/berkeleydb/downloads/index.html. El fichero se encuentra en la carpeta *lib*. Primero será necesario crear un entorno de base de datos y a continuación la base de datos. En el ejemplo la base de datos se llama NUMEROS y se almacena en la carpeta *D:/je-5.0.34/ejemplo*:

```
import java.io.File;
import java.io.UnsupportedEncodingException;

import com.sleepycat.je.Cursor;
import com.sleepycat.je.Database;
import com.sleepycat.je.DatabaseConfig;
import com.sleepycat.je.DatabaseEntry;
import com.sleepycat.je.Environment;
import com.sleepycat.je.EnvironmentConfig;
import com.sleepycat.je.LockMode;
import com.sleepycat.je.OperationStatus;
import com.sleepycat.je.Transaction;

public class Main {
 public static void main(String[] args) throws UnsupportedEncodingException {

     String envDir = "D:/je-5.0.34/ejemplo";//carpeta donde se almacenará la BD

     // Creamos un entorno de base de datos transaccional
     EnvironmentConfig envConfig = new EnvironmentConfig();
     envConfig.setAllowCreate(true);
     envConfig.setTransactional(true);
     Environment env = new Environment(new File(envDir), envConfig);

    //Crear una base de datos transaccional para ese entorno
     Transaction tx = env.beginTransaction(null, null);
     DatabaseConfig dbConfig = new DatabaseConfig();
     dbConfig.setTransactional(true);
     dbConfig.setAllowCreate(true);
     dbConfig.setSortedDuplicates(true);
     Database myDb = env.openDatabase(null, "NUMEROS", dbConfig);

     // DatabaseEntry representa los pares clave valor de cada registro
     DatabaseEntry clave =new DatabaseEntry();
     DatabaseEntry valor =new DatabaseEntry();

     //introducimos pares clave/valor
     clave = new DatabaseEntry("1".getBytes("UTF-8"));
     valor = new DatabaseEntry("uno".getBytes("UTF-8"));
     myDb.put(tx, clave, valor);

     clave = new DatabaseEntry("2".getBytes("UTF-8"));
     valor = new DatabaseEntry("dos".getBytes("UTF-8"));
     myDb.put(tx, clave, valor);
```



```
     clave = new DatabaseEntry("3".getBytes("UTF-8"));
     valor = new DatabaseEntry("tres".getBytes("UTF-8"));
     myDb.put(tx, clave, valor);
     tx.commit();

     String laclave;
     String elvalor;
     //Cursor para recorrer todos los pares Clave/Valor
     Cursor cursor = myDb.openCursor(null, null);

     while (cursor.getNext(clave, valor, LockMode.DEFAULT) ==
               OperationStatus.SUCCESS) {
         laclave = new String(clave.getData(), "UTF-8");
         elvalor = new String(valor.getData(), "UTF-8");
         System.out.println("Clave = " + laclave +  " Valor = " + elvalor);
     }

     cursor.close();   //cerramos cursor
     myDb.close();     //cerramos BD
     env.close();      //cerramos entorno

    }//fin de main

}//fin de la clase
```

La ejecución del programa visualiza la siguiente información:

```
Clave = 1 Valor = uno
Clave = 2 Valor = dos
Clave = 3 Valor = tres
```

Para probarlo en Linux cambiaríamos la carpeta donde se almacenará la base de datos (que tiene que existir), por ejemplo:

```
String envDir = "/home/usuario/db/Berkeley/ejemplo";
```

Después desde la línea de comandos definimos en el CLASSPATH donde se encuentra el fichero .jar, por ejemplo, si está en la carpeta */home/usuario/Java/Berkeley* escribimos:

```
$ export CLASSPATH=$CLASSPATH:/home/usuario/Java/Berkeley/je-5.0.34.jar
```

Compilamos y ejecutamos (suponemos que nuestro programa Java está en la misma carpeta):

```
Java/Berkeley$ javac Main.java
Java/Berkeley$ java Main
Clave = 1 Valor = uno
Clave = 2 Valor = dos
Clave = 3 Valor = tres
Java/Berkeley$
```