Hada T4:
Acceso a BBDD

Uso de SG de BBDD desde aplicaciones de escritorio.

Objetivos del tema

- Aprender a usar un SGBDD desde aplicaciones de escritorio.
- Conocer SQLite y aprender a usarlo desde código **C#**.
- El modelo de capas y el acceso a BBDD, entidades de negocio y componentes de acceso a datos.

Sqlite3

- Sqlite3 -o símplemente *Sqlite*- consiste en una biblioteca software que implementa un motor de BBDD relacional (SQL).
- Tal y como indica en su página web, de manera muy resumida, sus características principales son:
  - Es autocontenida (*self-contained*).
  - No tiene un proceso servidor (*serverless*).
  - No necesita ninguna configuración especial para comenzar a funcionar (*zero-configuration*).
  - Es transaccional (*transactional*).

Sqlite3: Características importantes

- Implementa gran parte de SQL92.
- Una bbdd completa se almacena en un solo fichero multiplataforma.
- Soporta bbdd. de terabytes y cadenas/blobs de gigabytes.
- Reducido tamaño en memoria, por ejemplo, completamente configurada puede ocupar unos `400KiB`.
- Muy rápida.
- API muy sencillo, pudiendo ser empleada desde diversos lenguajes de programación. Aquí dispones de diversos tutoriales.
- Escrita en `ANSI-C` en un solo fichero '**.c**' y el correspondiente '**.h**'.
- Viene con una aplicación en modo texto -CLI- que hace las veces de administrador de bbdd e intérprete de SQL: `sqlite3`.

Sqlite3: Aplicaciones comerciales que usan sqlite

- Adobe Photoshop Elements utiliza SQLite como motor de base de datos en sustitución de Microsoft Access.
- Mozilla Firefox usa SQLite para almacenar, entre otros, las cookies, los favoritos, el historial y las direcciones de red válidas.
- Varias aplicaciones de Apple utilizan SQLite, incluyendo Apple Mail y el gestor de RSS que se distribuye con Mac OS X. El software Aperture de Apple guarda la información de las imágenes en una base de datos SQLite, utilizando la API Core Data.
- El navegador web Opera usa SQLite para la gestión de bases de datos WebSQL.
- Skype.

Sqlite3: Y Lenguajes de programación

- De la familia de **C**: C, C++, C#, Java.
- De la familia de Pascal: Pascal, Delphi, FreePascal.
- Python, Perl.
- PHP.
- Desde .NET se puede acceder usando un proveedor ADO.NET de código abierto: System.Data.SQLite.

Sqlite3: Enlaces de interés

- Proyectos, aplicaciones y empresas que usan sqlite.
- La sintaxis de SQL soportada.
- Documentación en general.
- Libros sobre sqlite.
- Recomendable: Como programador de aplicaciones en general, también te puede ser muy útil consultar la manera en la cual se *testea* Sqlite: How SQLite Is Tested.

Sqlite3: El intérprete de órdenes de Sqlite I

- Sqlite incluye un intérprete de órdenes llamado `sqlite3`.
- Permite introducir órdenes SQL y ejecutarlas directamente contra una bbdd Sqlite que se le pasa como parámetro.
- Para ponerlo en marcha abrimos un terminal en modo texto y tecleamos la orden: `sqlite3`, p.e.:

```
$ sqlite3
    SQLite version 3.7.15.2 2013-01-09 11:53:05
    Enter ".help" for instructions
    Enter SQL statements terminated with a ";"
```

Sqlite3: El intérprete de órdenes de Sqlite II

```
$ sqlite3 test.db
    SQLite version 3.6.11
    Enter ".help" for instructions
    Enter SQL statements terminated with a ";"
    sqlite> create table tbl1(one varchar(10), two smallint);
    sqlite> insert into tbl1 values('hello!',10);
    sqlite> insert into tbl1 values('goodbye', 20);
    sqlite> select * from tbl1;
    hello!|10
    goodbye|20
    sqlite>
```

- Para salir del intérprete `sqlite3` usamos el carácter de fin de fichero: `Control-D` o la orden "`.exit`".

Sqlite3: Metadatos en Sqlite

- Los metadatos o el esquema de una bbdd en sqlite se almacenan en una tabla especial llamada: `sqlite_master`.
- Esta tabla se puede usar como cualquier otra tabla:

```
$ sqlite3 test.db
   SQLite version 3.6.11
   Enter ".help" for instructions
   sqlite> select * from sqlite_master;
     type = table
     name = tbl1
 tbl_name = tbl1
 rootpage = 3
      sql = create table tbl1(one varchar(10), two smallint)
   sqlite>
```

Sqlite3: Órdenes útiles I

- Una vez dentro del intérprete de órdenes de sqlite, éste reconoce -además de la sintaxis de SQL- una serie de órdenes directas para llevar a cabo ciertas acciones.
- Estas órdenes comienzan por un carácter “**.**”.
- Veamos algunas de ellas:

- **.help** : Muestra una breve ayuda de las órdenes reconocidas.
  - ```
    sqlite> .help
    .backup ?DB? FILE      -- Backup DB (default "main") to FILE
    .bail ON|OFF           -- Stop after hitting an error.  Default OFF
    ```
- **.databases** : Muestra las bbdd disponibles.
- **.mode** list | line | column : Cambia el formato de la salida producida por ejemplo por sentencias *select*.
- **.output** fichero-salida.txt : Redirige la salida al fichero “fichero-salida.txt”.
- **.tables** : Muestra las tablas de la bbdd.
- **.indices** tabla : Muestra los índices de la tabla “tabla”.
    ```

Sqlite3: Órdenes útiles II

- **.schema** : Muestra las órdenes “CREATE TABLE” y “CREATE INDEX” que se usaron para crear la bbdd actual. Si le pasamos como argumento el nombre de una tabla, entonces nos muestra la orden “CREATE” usada para crear esa tabla y sus índices.
- **.dump** tabla : Vuelca el contenido de la bbdd de esta tabla en formato SQL, por ejemplo:
  - ```
    sqlite> .output /tmp/test.sql
    sqlite> .dump tabla
    sqlite> .output stdout
    ```
- **.read** fichero.sql : Lee y ejecuta el código SQL contenido en “fichero.sql”.
- **.show** : Muestra el valor de diversos ajustes:
  - ```
    sqlite> .show
    echo: off
    explain: off
    headers: on
    mode: column
    nullvalue: ""
    output: stdout
    separator: "|"
    width:
    ```

Sqlite3: Órdenes útiles III

- **.separator** char : Cambia el separador de campos al carácter “char”:
  - sqlite> .separator ,
    sqlite> .show

    echo: off
    explain: off
    headers: on
    mode: column
    nullvalue: ""
    output: stdout
    separator: ","
    width:
- Esto nos permite importar datos de un fichero “CSV”, p.e., si tenemos un fichero con este contenido:

  5,value5
  6,value6
  7,value7

Sqlite3: Órdenes útiles IV

- **.import** fichero.csv tabla : Importa los datos del archivo “fichero.csv” en la tabla “tabla” línea a línea:

```
sqlite> .import fichero.csv tabla
sqlite> select * from tabla;

ids         value
----------  ----------
1           value1
2           value2
3           value3
```

Sqlite3: Uso no interactivo I

- Sqlite3 pude ser llamado con la opción “--help” y saber qué distintas formas de ser invocado tiene.
- Se puede emplear sqlite3 como un intérprete de SQL...de manera que podamos ejecutar desde la línea de órdenes:
  - sentencias individuales de SQL.
  - una serie de sentencias SQL guardadas en un archivo.
- Veamos unos ejemplos:

Sqlite3: Uso no interactivo II

- Una sola sentencia:
  - ```
    user@host:~$ sqlite3 -header -column test.db '.schema'

    CREATE TABLE test (ids integer primary key, value text);
    CREATE VIEW testview AS select * from test;
    CREATE INDEX testindex on test (value);
    ```
  - ```
    user@host:~$ sqlite3 -header -column test.db 'select * from test'

    ids         value
    ----------  ----------
    1           value1
    2           value2
    3           value3
    ```

Sqlite3: Uso no interactivo III

- Exportar una bbdd:
  - `user@host:~$ sqlite3 test.db '.dump' > dbbackup`
- Ejecutar sentencias SQL guardadas en un fichero:
  - `user@host:~$ sqlite3 test.db < statements.sql`
  - `user@host:~$ cat statements.sql | sqlite3 test.db`

SqliteBrowser

- Se trata de un interfaz gráfico sobre sqlite.
- Es sencillo de usar, además de portable entre Windows/Mac/Linux.
- Lo puedes encontrar en su web

Sqlite: Uso desde un lenguaje de programación I

- Hemos visto como usar sqlite desde línea de órdenes y también con una aplicación con interfaz gráfico como es `sqlitebrowser`.
- Vamos a ver ahora cómo podemos hacer uso de sqlite desde una aplicación escrita en un lenguaje de programación.
- Veremos primero un ejemplo en **C** dado que sería el lenguaje original para trabajar con sqlite...
- Y luego veremos un ejemplo también muy sencillo en **C#**.

Sqlite: Uso desde C I

```
#include <stdio.h>
#include <sqlite3.h>

static int
callback_fn(void *NotUsed, int argc, char **argv, char **azColName)
{
  int i;

  for(i=0; i<argc; i++) {
    printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
  }
  printf("\n");

  return 0;
}
```

Sqlite: Uso desde C II

```
int main(int argc, char **argv) {
  sqlite3* db;
  char*    zErrMsg = 0;
  int      rc;

  if (argc!=3) {
    fprintf(stderr,
            "Usage: %s DATABASE SQL-STATEMENT\n",
            argv[0]);
    return(1);
  }

  rc = sqlite3_open(argv[1], &db);
  if (rc) {
    fprintf(stderr,
            "Can't open database: %s\n",
            sqlite3_errmsg(db));
    sqlite3_close(db);
    return(1);
  }
```

```
 rc = sqlite3_exec(db, argv[2],
                   callback_fn, 0, &zErrMsg);
 if (rc != SQLITE_OK) {
    fprintf(stderr, "SQL error: %s\n", zErrMsg);
    sqlite3_free(zErrMsg);
 }

 sqlite3_close(db);
 return 0;
}
```

Sqlite: Uso desde C III

- Esta aplicación debe ser llamada con dos argumentos, el primero es la bbdd con la que trabajar y el segundo, la sentencia SQL que le damos.
- Aunque no lo parezca...*hemos creado una versión sencilla de la aplicación de línea de órdenes* `sqlite3`.
- Podemos ejecutar “cosas” como éstas:

```
sqlite-example test.db “create table test (ids integer primary key , value text )”;
sqlite-example test.db “insert into test values('hola', 10);”
sqlite-example test.db “insert into test values('adios',20);”
sqlite-example test.db “select * from test;”
ids = hello
value = 10

ids = goodbye
value = 20
```

Sqlite: Uso desde C# I

```
// Compile:
// mcs sqlite3.cs -r:Mono.Data.Sqlite.dll -r:System.Data
using System;
using Mono.Data.Sqlite;

public class Example { // class
 static void Main() { // Main
   string cs = "URI=file:test.db";
   using ( SqliteConnection con = new SqliteConnection(cs))
   { // using-1
       con.Open();
       using (SqliteCommand cmd = new SqliteCommand(con))
       { // using-2
          cmd.CommandText = "DROP TABLE IF EXISTS Cars";
          cmd.ExecuteNonQuery();
          cmd.CommandText = @"CREATE TABLE Cars(
                      Id INTEGER PRIMARY KEY,
                             Name TEXT, Price INT)";
          cmd.ExecuteNonQuery();
```

```
          cmd.CommandText = "INSERT INTO Cars
                             VALUES(1,'Audi',52642)";
          cmd.ExecuteNonQuery();
          cmd.CommandText = "INSERT INTO Cars
                             VALUES(2,'Mercedes',57127)";
          cmd.ExecuteNonQuery();
          cmd.CommandText = "INSERT INTO Cars
                             VALUES(3,'Skoda',9000)";
          cmd.ExecuteNonQuery();
          cmd.CommandText = "INSERT INTO Cars
                             VALUES(4,'Volvo',29000)";
          cmd.ExecuteNonQuery();
          cmd.CommandText = "INSERT INTO Cars
                             VALUES(5,'Bentley',350000)";
          cmd.ExecuteNonQuery();
          cmd.CommandText = "INSERT INTO Cars
                             VALUES(6,'Citroen',21000)";
          cmd.ExecuteNonQuery();
          cmd.CommandText = "INSERT INTO Cars
                             VALUES(7,'Hummer',41400)";
```

Sqlite: Uso desde C# II

```
          cmd.ExecuteNonQuery();
          cmd.CommandText = "INSERT INTO Cars
                            VALUES(8,'Volkswagen',21600)";
          cmd.ExecuteNonQuery();
       } // using-2
       con.Close();
    } // using-1
  } // Main
} // class
```

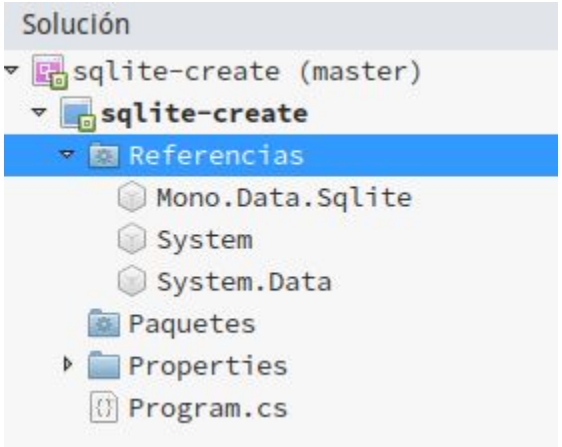
Las dos sentencias *using* de este ejemplo tienen que ver con la implementación del patrón RAII en C#. Puedes ver más información aquí.

Sqlite: Uso desde MonoDevelop

Las opciones ‘-r’ a la hora de compilar desde la línea de órdenes:

```
// mcs sqlite3.cs -r:Mono.Data.Sqlite.dll -r:System.Data
```

Son equivalentes a añadir esas *referencias* a un proyecto en MonoDevelop:



Arquitectura de capas y acceso a BBDD I

- Con el fin de obtener un código más legible y fácilmente mantenible para este tipo de aplicaciones vamos a ver cómo estructurar su código fuente.
- Proponemos para ello seguir un patrón de capas -será ampliado posteriormente en la parte web- para dividir el código de la aplicación según "*divisiones*" lógicas...similar a como vimos con **MVC**.
- Cada una de estas "*divisiones*" se desarrolla y mantiene por separado.

Arquitectura de capas y acceso a BBDD II

- Concretamente dividiremos el código en tres capas o componentes:
  a. Capa de interfaz de usuario.
  b. Capa de lógica de negocio o *Entidad de Negocio* (**EN**).
     - Sería el equivalente a lo que en *MVC* conocemos como la Capa del Modelo.
     - Se le asocia un **CAD** mediante el cual esta **EN** puede almacenarse/modificarse/recuperarse... en la bbdd con la que trabajemos.
  c. Capa de persistencia o *Componente de Acceso a Datos* (**CAD**).
     - Los *CAD* implementan la lógica de comunicación con la bbdd, la cual es bidireccional entre las **EN** y la bbdd.
     - Las operaciones habituales que proporciona un **CAD** son las de *creación*, *lectura*, *actualización* y *borrado* de registros de la bbdd.