Tema: V.
Acceso a BBDD desde aplicaciones de escritorio: modelo de capas.

Herramientas Avanzadas para el Desarrollo de Aplicaciones

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante

Contenido

Preliminares

- En este tema vamos a ver como hacer uso de BBDD relacionales desde aplicaciones de escritorio.
- Haremos una breve introduccion al modelo de capas que será explicado en profundidad en la parte web.
- Pondremos en la práctica lo anterior haciendo uso de sqlite3 . Para ello veremos primero las características de *sqlite3* y posteriormente ejemplos de uso desde aplicaciones escritas en Vala.

Presentación de Sqlite3 (I)

- 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*).

Presentación de Sqlite3 (II)

Algunas otras características no menos importantes de sqlite son:

- Implementa gran parte de `SQL92`.
- Una bb.dd. completa se almacena en un solo fichero multiplataforma.
- Soporta bb.dd. 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.
- 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 bb.dd.: `sqlite3`.

Presentación de Sqlite3 (III)

Algunas aplicaciones que usan sqlite:

- Adobe Photoshop Elements utiliza SQLite como motor de base de datos en su última versión del producto (la 6.0) en sustitución del 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.

Sqlite y lenguajes de programación

Sqlite puede ser utilizado desde diversos lenguajes de programación, algunos de ellos son:

- C, C++, Vala, Java
- Pascal, Delphi
- Python, Perl
- PHP
- Desde .NET se puede acceder usando el proyecto de código abierto System.Data.SQLite

Presentación de Sqlite3 (IV)

Para completar esta introducción es interesante que consultes los siguientes enlaces:

- Proyectos, aplicaciones y empresas que usan sqlite .
- La sintáxis 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 bb.dd. Sqlite.
- Para ponerlo en marcha abrimos un terminal en modo texto y tecleamos la orden: `sqlite3`.

Ejemplo de uso de sqlite3

http://www.sqlite.org/sqlite.html

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

Por ejemplo, para crear una bb.dd. llamada '`test.db`' y que tenga una tabla llamada `tbl1` podríamos hacer lo siguiente:

```
$ 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`'.

Metadatos en sqlite

http://www.sqlite.org/sqlite.html

Los metadatos o el esquema de una bb.dd. 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>
```


Algunas órdenes útiles en sqlite3 (I)

http://www.sqlite.org/sqlite.html

- Una vez dentro del intérprete de órdenes de sqlite, éste reconoce -además de la sintáxis 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 bb.dd. disponibles.

Algunas órdenes útiles en sqlite3 (II)

http://www.sqlite.org/sqlite.html

.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 bb.dd.

.indices tabla Muestra los índices de la tabla '`tabla`'.

.schema Muestra las órdenes '`CREATE TABLE`' y '`CREATE INDEX`' que se usaron para crear la bb.dd. 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.

Algunas órdenes útiles en sqlite3 (III)

http://www.sqlite.org/sqlite.html

.dump tabla Vuelca el contenido de la bb.dd. 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:
```


Algunas órdenes útiles en sqlite3 (IV)

http://www.sqlite.org/sqlite.html

.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
```


Algunas órdenes útiles en sqlite3 (V)

http://www.sqlite.org/sqlite.html

.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
```

Uso no interactivo de Sqlite3 (I)

- Sqlite3 pude ser llamado con la opción '`--help`' y saber que 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:
  1. sentencias individuales de `SQL`.
  2. una serie de sentencias `SQL` guardadas en un archivo.

  Veamos unos ejemplos:

- 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);
```


Uso no interactivo de Sqlite3 (II)

- Ejecutando una sentencia '`SELECT`':

```
user@host:~$ sqlite3 -header -column test.db 'select * from
    test'

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

- Exportar una bb.dd.:

```
user@host:~$ sqlite3 test.db '.dump' > dbbackup
```

- Ejecutar sentencias '`SQL`' guardadas en un fichero:

```
user@host:~$ sqlite3 test.db < statements.sql
```

  O también así:

```
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

Uso de sqlite desde un lenguaje de programación

- Hemos visto como usar sqlite desde línea de órdenes y también con una aplicación con interfaz gráfico como es `sqlitebrowser`.
- Vamos a ver ahora como 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 Vala.

Sqlite en C (I)

- El código de las dos siguientes transparencias se guarda en un archivo llamado '`sqlite-example.c`'.
- Se compila con la orden: '*gcc sqlite-example.c -o sqlite-example -lsqlite3*'

```c
#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 en C (II)

```c
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 en C (III)

- Esta aplicación tiene dos argumentos, el primero es la bb.dd. con la que trabajar y el segundo, la orden `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 estas:

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

8   ids = goodbye
    value = 20
```


Sqlite en Vala (I)

- Veamos exactamente el mismo ejemplo en Vala.
- Lo compilaremos con la siguiente orden: '*valac –pkg sqlite3 sqlitesample.vala*'.

```
using Sqlite;

public class SqliteSample : GLib.Object {
  public static int callback (int n_columns,
                              string[] values,
                              string[] column_names)
  {
    for (int i = 0; i < n_columns; i++) {
      stdout.printf ("%s = %s\n", column_names[i], values[i]);
    }

    stdout.printf ("\n");
    return 0;
  }
}
```

Sqlite en Vala (II)

```
1   public class SqliteSample : GLib.Object {
      public static int main (string[] args) {
3       Database db;
        int rc;
5
        if (args.length != 3) {
7         stderr.printf ("Usage: %s DATABASE SQL-STATEMENT\n", args[0]);
          return 1;
9       }

11      if (!FileUtils.test (args[1], FileTest.IS_REGULAR)) {
          stderr.printf ("Database %s does not exist or is directory\n",
               args[1]);
13        return 1;
        }
15
        rc = Database.open (args[1], out db);
17      if (rc != Sqlite.OK) {
          stderr.printf ("Can't open database: %d, %s\n", rc, db.errmsg ()
               );
19        return 1;
        }
21
        rc = db.exec (args[2], callback, null);
23      if (rc != Sqlite.OK) {
          stderr.printf ("SQL error: %d, %s\n", rc, db.errmsg ());
25        return 1;
        }
27
        return 0;
29    }
    }
```


Arquitectura de capas (II)

Entidad de negocio (EN), Componente de acceso a datos (CAD).

Dividiremos el código en tres capas o componentes:

- Capa de interfaz de usuario.
- 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 puede almacenarse/modificarse/recuperarse... en la bb.dd. con la que trabajemos.
- Capa de persistencia o *Componente de Acceso a Datos* (**CAD**).
  - Los *CAD* implementan la lógica de comunicación con la bb.dd. la cual es bidireccional entre las *EN* y la bb.dd.
  - Las operaciones habituales que proporciona un *CAD* son las de **creación**, **lectura**, **actualización** y **borrado** de registros de la bb.dd.

Arquitectura de capas (I)

Entidad de negocio (EN), Componente de acceso a datos (CAD).

- 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[1] 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.

[1] Será ampliado posteriormente en la parte web.