Unit: V.
DDBB access from desktop applications:layered model.

Herramientas Avanzadas para el Desarrollo de Aplicaciones

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante

lsi Department of Software and Computing Systems

Content

Preliminars

- In this unit we are going to see how to use relational DDBB from desktop applications.
- We will make a brief introduction to the layered model that will be explained in depth in the Web part.
- We will use sqlite3 . For this purpose we will first explain the *sqlite3* features and examples of use from applications written in Vala.

Introduction to Sqlite3 (I)

- Sqlite3 -or símply *Sqlite*- consists of a software library that implements a relational DDBB engine(SQL).
- As it is indicated in its web page , in a summarized way, its main features are:
  - It is self contained (*self-contained*).
  - It does not have a server process (*serverless*).
  - It does not need any special configuration to start working (*zero-configuration*).
  - It is transactional (*transactional*).

Introduction to Sqlite3 (II)

Some other features of sqlite are:

- It implements most of `SQL92`.
- A complete DDBB is stored in a single multiplatform file.
- It supports DDBB of terabytes and strings/blobs of gigabytes.
- It has reduced size in memory, for instance, completely configured it can ocuppy 400KiB.
- High speed.
- Very simple `API`, it could be used with different programming languages.
- It is written in `ANSI-C` in a single file '`.c`' and its corresponding '`.h`'.
- It comes with a text-mode application -`CLI`- that acts as DDBB administrator: `sqlite3`.

Introduction to Sqlite3 (III)

Some applications that use sqlite:

- Adobe Photoshop Elements uses SQLite as database engine in the last version of the product (6.0) instead of Microsoft Access.
- Mozilla Firefox uses SQLite for storing, among others, the cookies, favourites, history and valid network addresses.
- Several applications from Apple use SQLite, including Apple Mail and the RSS manager that is distributed with Mac OS X. The software Aperture of Apple stores the information of the images in a SQLite DDBB, using the API Core Data.
- The web browser Opera uses SQLite for managing WebSQL databases.
- Skype.

Sqlite and programming languages

Sqlite can be used with different programming languages, some of them are:

- C, C++, Vala, Java
- Pascal, Delphi
- Python, Perl
- PHP
- In .NET we can access using the open soure code project System.Data.SQLite

Introduction to Sqlite3 (IV)

To complete this introduction is interesting that you check the following links:

- Projects, applications and companies that use sqlite .
- The supported SQL syntax .
- Documentation in general.
- Books about sqlite.

`Recommended:` For a programmer of applications in general, is useful to check the way Sqlite is tested: How SQLite Is Tested .

sqlite3 - The command interpreter of Sqlite. (I)

- Sqlite includes a command interpreter called `sqlite3`.
- It allows introducing SQL commands and execute them directly in a Sqlite DDBB.
- To start it we open a text mode terminal and we type the command: `sqlite3`.

Example of use of 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 ";"
```

For example, to create a DDBB called 'test.db' and that has a table called tbl1 we could do as follows:

```
$ 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>
```

To exit from the sqlite3 interpreter we use the end of file character: Control-D or the command '.exit'.

Metadata en sqlite

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

The metadata or the DDBB schema in sqlite are stored in a special table called: `sqlite_master`. This table is used as any other table:

```
$ 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>
```

Some useful commands in sqlite3 (I)

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

- Once inside the sqlite command interpreter, this recognizes - besides the SQL syntax- a set of direct commands to perform certain actions.
- Some commands start by a command '**.**'.
- Let's see some of them:

  .help It shows a brief help of the recognized commands.

```
      sqlite> .help
2     .backup ?DB? FILE      -- Backup DB (default "main")
            to FILE
      .bail ON|OFF           -- Stop after hitting an
          error.  Default OFF
4     ...
```

.databases It shows the DDBB available.

Some useful commands in sqlite3 (II)

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

.mode list | line | column It changes the formatting of the output we get for instance with `select` sentences.

.output fichero-salida.txt Redirects the output to the file `fichero-salida.txt`.

.tables It shows the tables of the DDBB.

.indices tabla It shows the indexes of the table '`tabla`'.

.schema It shows the commands '`CREATE TABLE`' and '`CREATE INDEX`' that were used to create the current DDBB. If we pass as an argument the name of a table, then it is shown the command '`CREATE`' used to create this table and its indexes.

Some useful commands in sqlite3 (III)

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

.dump tabla It puts the content of the DDBB of this table in SQL format, for example:

```
sqlite> .output /tmp/test.sql
sqlite> .dump tabla
sqlite> .output stdout
```

.read fichero.sql It reads and executes the SQL code in the file '`fichero.sql`'.

.show It shows the value of several adjustments:

```
sqlite> .show
     echo: off
   explain: off
   headers: on
      mode: column
nullvalue: ""
    output: stdout
separator: "|"
     width:
```

Some useful commands in sqlite3 (IV)

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

.separator char It changes the separator of fields to the character '`char`':

```
sqlite> .separator ,
sqlite> .show
        echo: off
     explain: off
     headers: on
        mode: column
   nullvalue: ""
      output: stdout
   separator: ","
       width:
```

It allows us to import data from a '`CSV`' file, f.i., if we have a file with the next content:

```
5,value5
6,value6
7,value7
```

Some useful commands in sqlite3 (V)

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

.import fichero.csv tabla Imports the data of the file '`fichero.csv`' in the table '`tabla`' line by line:

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

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

Non-interactive use of Sqlite3 (I)

- Sqlite3 can be called with the option '`--help`' and see the different ways that it can be invoked.
- We can use *sqlite3* as a SQL interpreter...in this way we can execute from the command line:
  1. individual SQL sentences.
  2. a set of SQL sentences stored in a file.

  Let's see some examples:

- A single sentence:

```
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);
```

Non-interactive use of Sqlite3 (II)

- Executing the sentence 'SELECT':

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

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

- Export from a DDBB:

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

- Execute 'SQL' sentences stored in a file:

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

Or also:

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

SQLiteBrowser

- It is a graphic interface about sqlite.
- It is easy to use, and is portable in Windows/Mac/Linux.
- We can find it in its web

Use of sqlite from a programming language

- We have seen how to use sqlite from the command line and also with an application with graphic interface such as `sqlitebrowser`.
- We are going to see now how to use sqlite from an application written in a programming language.
- We will first see an example written in 'C' which is the 'original' language to work with sqlite...
- And then we will see a very simple example written in Vala.

Sqlite in C (I)

- The code of the two next slides is stored in a file called '`sqlite-example.c`'.
- It is compiled with the command: '*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 in C (II)

```
1   int main(int argc, char **argv) {
      sqlite3 *db;
3     char *zErrMsg = 0;
      int rc;
5
      if( argc!=3 ) {
7       fprintf(stderr, "Usage: %s DATABASE SQL-STATEMENT\n", argv[0]);
        return(1);
9     }

11    rc = sqlite3_open(argv[1], &db);
      if( rc ) {
13      fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
15      return(1);
      }
17
      rc = sqlite3_exec(db, argv[2], callback_fn, 0, &zErrMsg);
19    if( rc!=SQLITE_OK ) {
        fprintf(stderr, "SQL error: %s\n", zErrMsg);
21      sqlite3_free(zErrMsg);
      }
23
      sqlite3_close(db);
25    return 0;
    }
```

Sqlite in C (III)

- This application has two arguments, the first one is the DDBB to work with, and the second is the SQL command that we want to execute.
- Even if it does not seem so...we have created a simple version of the command line application *sqlite3*.
- We can execute things like these:

```
    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 in Vala (I)

- Let's see the same example using Vala.
- We will compile it with the next command: '*valac –pkg sqlite3 sqlitesample.vala*'.

```
1   using Sqlite;

3   public class SqliteSample : GLib.Object {
      public static int callback (int n_columns,
5                                 string[] values,
                                  string[] column_names)
7     {
        for (int i = 0; i < n_columns; i++) {
9         stdout.printf ("%s = %s\n", column_names[i], values[i]);
        }
11
        stdout.printf ("\n");
13      return 0;
      }
15  }
```

Sqlite in 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    }
    }
```

Layered architecture (I)

Bussiness entity (BE), Data access component (DAC).

- With the aim of getting a more legible and easily maintainable code for this type of applications we are going to see how to structure its source code.
- We propose to follow a layered pattern[1] for dividing the application code in logic '*divisions*' ...similar to what we saw in *MVC*.
- Each of these '*divisions*' is developed and maintained separately.

[1] It will be used and explained in more depth in the Web part.

Layered architecture (II)

Bussiness entity (BE), Data access component (DAC).

We will divide the code in three layers or components:

- User interface layer.
- Bussiness logic layer or *Bussiness entity* (**BE**).
  - It would be the equivalent to what in *MVC* we know as *Model* layer.
  - It has associated a *DAC* with which we can create/read/update/delete... in the DDBB we are working with.
- Persistence layer or *Data Access Component* (**DAC**).
  - The *DAC* implement the communication logic with the d.d.b.b. which is bidirectional between the *BE* and the d.d.b.b..
  - The usual operations provided by a *DAC* are **create**, **read**, **update** and **delete** of the registry of the d.d.b.b.