DANIEL GARCIA

Personal Website / Bio

6 Posts | 30 Comments

Computer science engineer. Developer. Traveller. Curious mind. contacto (at) danigarcia (dot) org

NEWS

POST CATEGORIES

- C#
- circular buffer
- Heap
- persistent dictionary
- stream heap
- file stream heap
- StreamHeap
- FileStreamHeap
- CachingDictionary
- caching dictionary
- PersistentDictionary
- Reactive Extensions
- rx
- Observable
- GCD
- greatest common divisor
- rational
- fraction
- Assert
- Generic
- array
- ArrayPool
- DateTime
- Testing
- TimeoutDictionary
- .Net
- Monads
- Network
- Socket
- Select
- TcpClient
- TcpListener
- Array Indexing
- java
- linq
- enumerators
- MVVM
- TreeView
- WPF
- JavaScript
- Property
- Properties
- get
- set
- getters

Daniel Garcia

Let's code something up!

<< Hello, world! | Home | Portable databases (II): using SQLite with Entity Framework >>

Portable databases: using SQLite with .NET

Comments (16) | Share

Most of applications we develop on .NET in a professional way use to imply the existence of a database, usually SQL Server, Oracle, DB2 or MySQL. Nevertheless, sometimes, even we need database support, it's not necessary to mantain a database manager, because its portability, licensing, data volume and complexity...

In Android, each application uses a single local SQLite database. Why not applying the same philosophy to a .NET application? Well, it's possible to encapsulate the database in a .db file inside a local directory on the same way Android applications do. Let's see now how to install SQLite for .NET and programming a little application which makes *CRUD* (Create, Retrieve, Update, Delete) operations over an entity. From there, it will be extrapolated to more complex entitites.

SQLite installation in .NET

First of all: downloading the SQLite library adapted to .NET. We can download it from this website.

Setups for 32-bit Windows (.NET Framework 4.5)

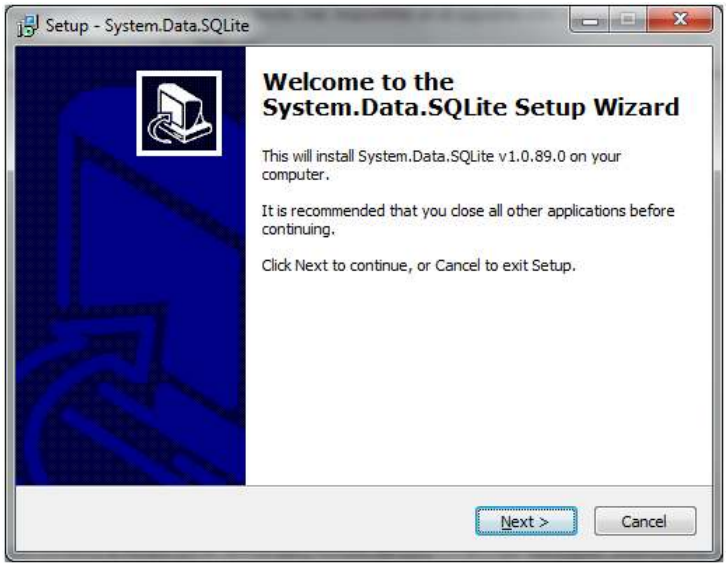
sqlite-netFx45-setup-bundle-x86-2012-1.0.89.0.exe (8.12 MiB)	This setup package features the mixed-mode assembly and will install all the necessary runtime components and dependencies for the x86 version of the System.Data.SQLite 1.0.89.0 (3.8.1) package. The Visual C++ 2012 Update 3 runtime for x86 is included. The .NET Framework 4.5 is required. **This is the only setup package that is capable of installing the design-time components for Visual Studio 2012.** (sha1: d8eb780a3554684d1ffc3fca767867dfa22fc407)
sqlite-netFx45-setup-x86-2012-1.0.89.0.exe (8.15 MiB)	This setup package will install all the necessary runtime components and dependencies for the x86 version of the System.Data.SQLite 1.0.89.0 (3.8.1) package. The Visual C++ 2012 Update 3 runtime for x86 is included. The .NET Framework 4.5 is required. (sha1: a7ff3e421ceb8b911d38481d5e5dcb799b992e9b)

We must to choose the version best adapted to our application: framework (2.0, 3.5, 4, 4.5...) such architecture (32 or 64 bits). In our case, we use Visual Studio 2012 over a x82 (32 bits) machine, so we look for the link displayed above. From the, we'll choose the version that is able to install the Visual Studio 2012 components (the upper one). Next, we'll begin the installation process.

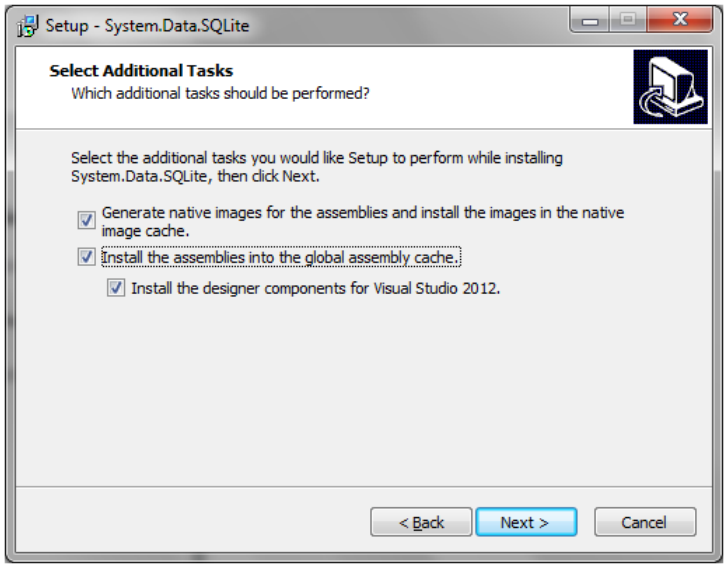
• setters

ARCHIVES

• January 2014 (3)
• December 2013 (3)

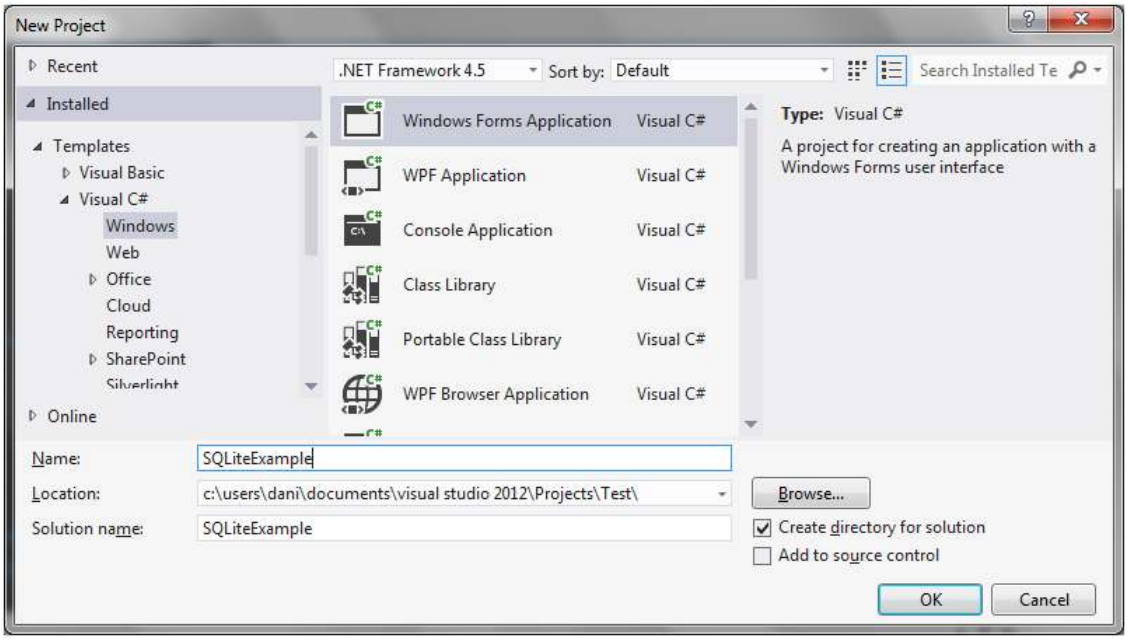


We'll select then the SQLite elements we want to install. In our case, we'll check all the options.

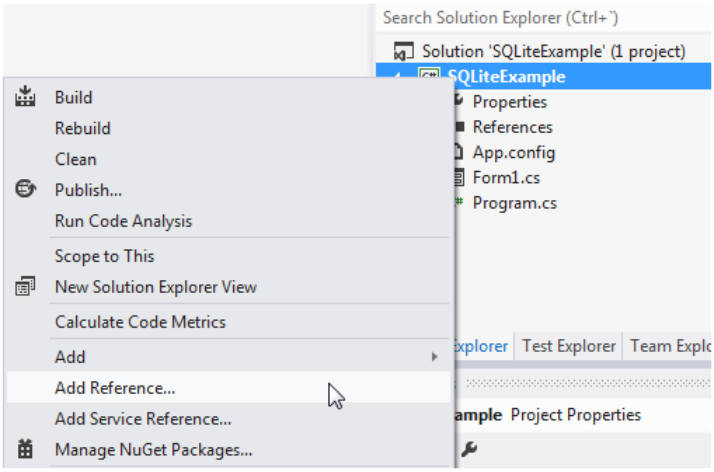


Creating the project

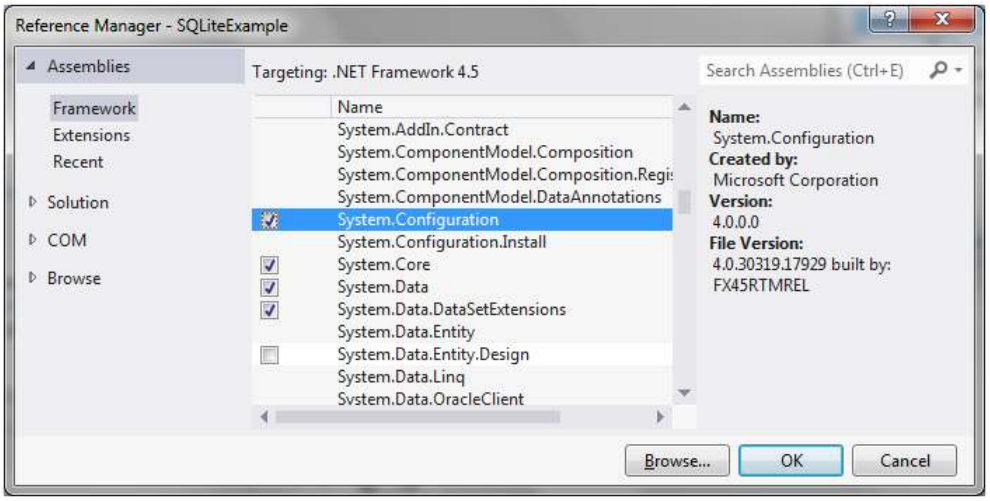
Our test application will be simple: it will consist in a desktop application where we'll code all the data access logic. Therefor, we'll select File > New Project... and we'll choose the option "Windows Forms Application."



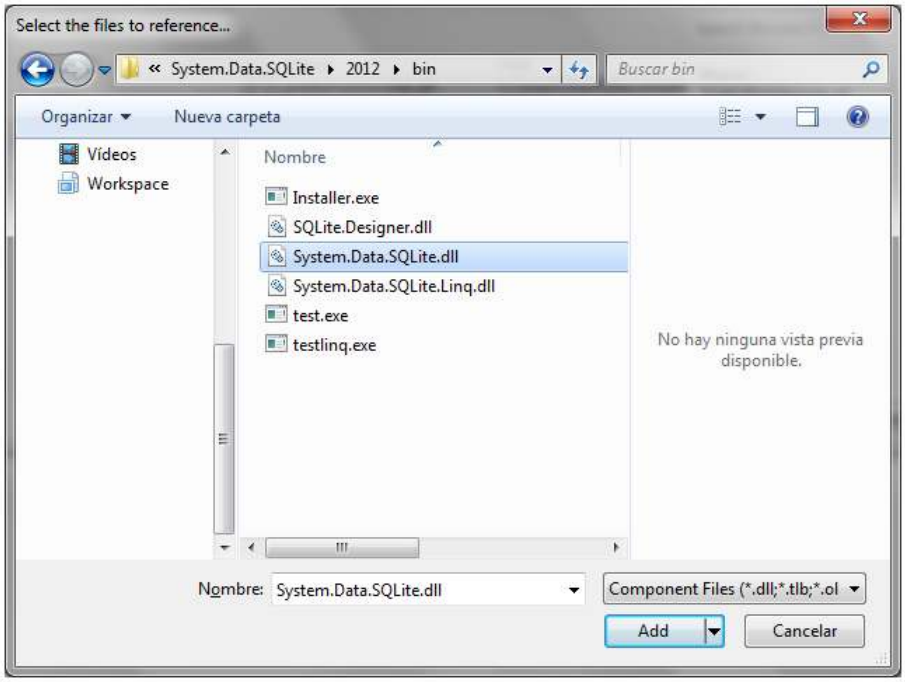
We need to add two references: one of them will be the component *System.Configuration* so we can access to the connection string, and the another one will be the DLL which implements the SQLite access. We'll do right-click over our project and we'll select *Add Reference*….



We can find the first element in the section *Framework*. We search the assembly and we'll check it.



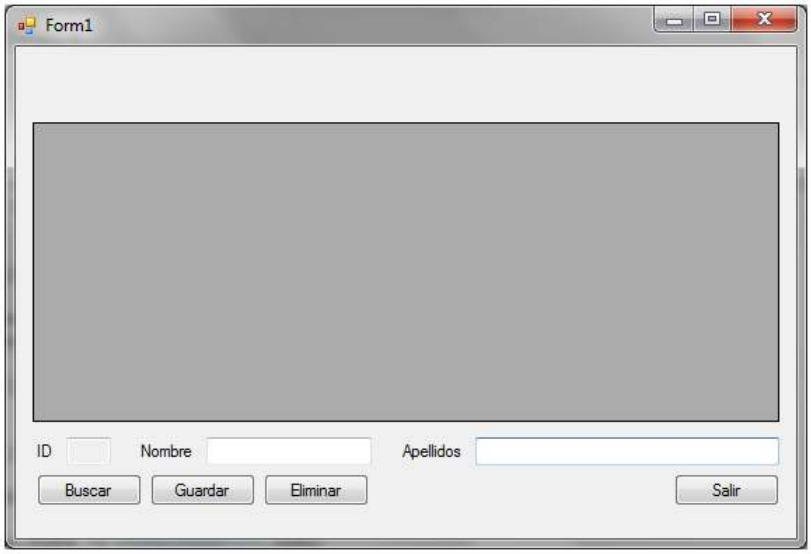
The second element must be selected through the *Browse* option. We must look out inside the directory we previously installed SQLite (C:\Program Files\System.Data.SQLite\2012\bin by default) and selecting the assembly *System.Data.SQLite.dll*.



Creating the UI

Let's begin the house from the rooftop, adding some controls to our application. We'll show a DataGridView, three labels, three textboxes and four buttons:

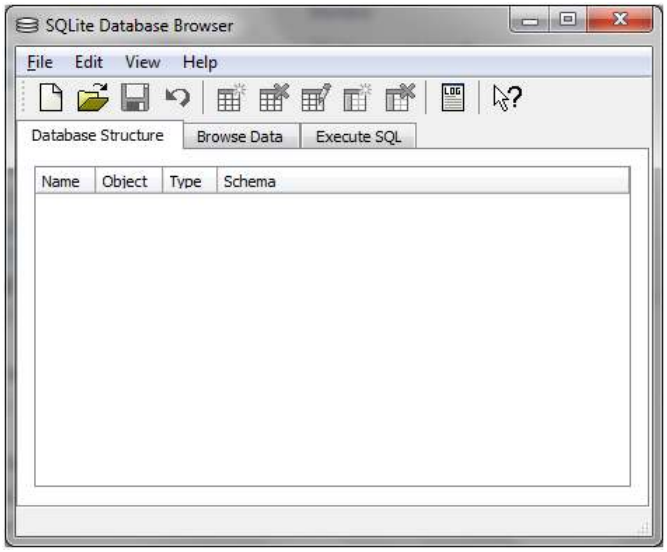
- The DataGridView will show the list with the retrieved data.
- TextBoxes will obtain data from a single registry for insertion, updating or deleting.
- Buttons will trigger the insertions, updates and deletes (and, of course, exiting the application).



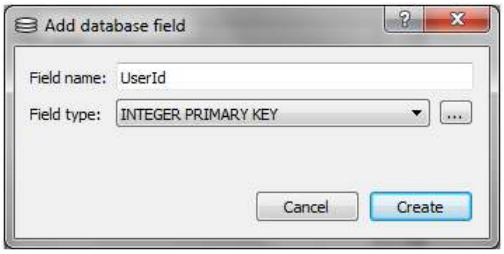
Creating the database

Hold on! we have the connector and the UI, but... we haven't a database yet! How can we create it? Well, an option is using the tool *SQLiteBrowser*, a little manager which will allow us to create and work with SQLite databases. We can download it from this link.

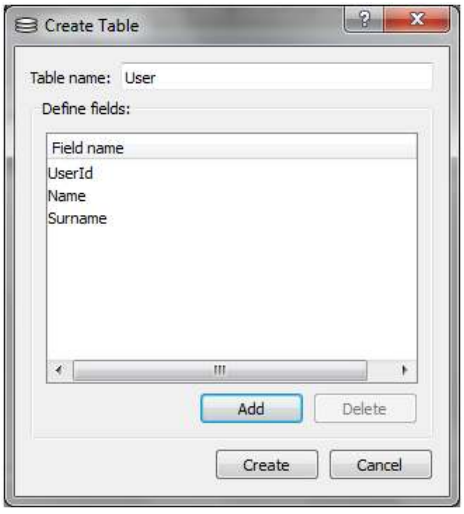
Once the application has been executed, we can see that the interface is pretty intuitive. If you want to create a new database, just push over the first icon from the left, which represent a blank document.



Then we must to create the tables with their respective fields. Needless to say that SQLite is a very limited database, much mor than SQL Server or Oracle, so that besides providing less funcionality, it contains a set of restrictions which must be satisfied, as the existence of an integer primary key in each table. Thus, we'll create a table with an *UserId* field, which type will be *INTEGER PRIMARY KEY*.



Let's add the rest of the fields we want to create. In example, *Name* and *Surname*. Once created these fields, we'll push *Create* button and we'll save the database, that won't be more than a file with '.db' extension.



Configuring the Connection String

As happens with other databases, it's possible to use a connection string to connect to SQLite. To do so, we must to indicate the following fields in our app.config file:

- Data Source: path of the file we just created.
- Version: SQLite Version. At present is version 3.
- New: Indicates if database exists or not.
- Compress: Using or not using data compression.

The file will be shown as follows:

```
1
2  <?xml version="1.0" encoding="utf-8" ?>
3  <configuration>
```

```
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>

  <connectionStrings>
    <add name="db" connectionString="Data Source=clients.db;Version=3;New=False;Compress
  </connectionStrings>

</configuration>
```

Coding the application

First of all, we must add the *using* clause to import SQLite elementos and the creation of the private attributes we're going to make use to.

```
using System.Data.SQLite;
```

Elements we're going to create are: connection string, connection itself and the strings which will implement the four operations we can perform over the table. Notice that we'll launch the queries directrly over the database through ADO.NET. SQLite allows also the use of Entity Framework, but that's something we'll see another time.

```
private String connectionString;
private SQLiteConnection connection;

private String SQLInsert = "INSERT INTO User(Name, Surname) VALUES(?, ?)";
private String SQLUpdate = "UPDATE User SET Name = ?, Surname = ? where UserId = ?";
private String SQLSelect = "SELECT * FROM User";
private String SQLDelete = "DELETE FROM User WHERE UserId = ?";
```

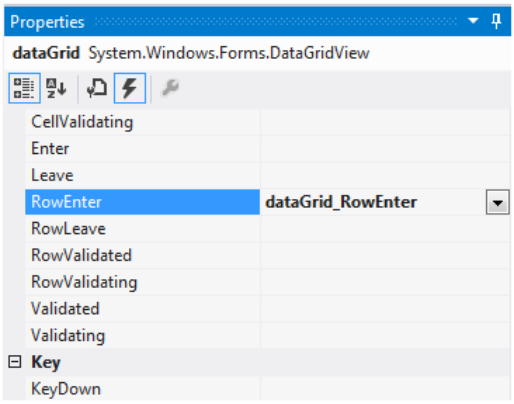
As we can check, parameter entries are coded through the question mark symbol, which will be replaced on runtime for the corresponding SQLiteParameter. We'll begin the application extracting the connection string from the app.config file and using it to create a new connection.

```
public Form1()
{
    InitializeComponent();
    connectionString = System.Configuration.ConfigurationManager.ConnectionStrings["db"].
    connection = new SQLiteConnection(connectionString);
}
```

We'll make double click over the "Exit" button and we'll apply the following code to the button, in which we'll close the connection if it's still open and finish the program.

```
private void btnExit_Click(object sender, EventArgs e)
{
    if (connection.State == ConnectionState.Open)
        connection.Close();

    Application.Exit();
}
```

Next, we must control that, when a row of the DataGrid is selected, its data should be loaded inside the text boxes. To do so, we select the DataGrid and, in its properties, we'll perform double click over the *RowEnter* event.



After that, we'll add the neccesary code to pass the row's data to the text boxes.

```
private void dataGrid_RowEnter(object sender, DataGridViewCellEventArgs e)
{
    // Retrive ID, name and surname from the row
    int id = int.Parse(dataGrid.Rows[e.RowIndex].Cells[0].Value.ToString());
    String name = (String)dataGrid.Rows[e.RowIndex].Cells[1].Value;
    String surname = (String)dataGrid.Rows[e.RowIndex].Cells[2].Value;
```

```
    // Assign values to the textboxes
    txtId.Text = id.ToString();
    txtName.Text = name;
    txtSurname.Text = surname;
}
```

The following step will be code the search process. We will create a *search* method which creates a SQLiteCommand from the string with the SELECT clause. Then, we will fulfill a DataTable using a DataAdapter, everything from the connection.

Despite of the grid load forces the first row to be selected by default, we remove the event handler before loading the DataGrid, restoring it again after the load is complete.

```
private void search()
{
    // Remove the handler of the RowEnter event to avoid that it should be triggered
    // when the search is performed
    dataGrid.RowEnter -= dataGrid_RowEnter;

    // Open the connection
    if (connection.State != ConnectionState.Open)
        connection.Open();

    // Create the SQLiteCommand and assign the query string
    SQLiteCommand command = connection.CreateCommand();
    command.CommandText = SQLSelect;

    // Create a new DataTable and a DataAdapter from the SELECT.
    // Then fill the DataTable with the DataAdpater
    DataTable dt = new DataTable();
    SQLiteDataAdapter da = new SQLiteDataAdapter(command);
    da.Fill(dt);

    // Associate the DataTable to the DataGrid and close the connection
    dataGrid.DataSource = dt;
    connection.Close();

    // Assign again the handler of the event
    dataGrid.RowEnter += dataGrid_RowEnter;
}
```

The method associated to the search button will perform a simple invocation of this procedure.

```
private void btnSelect_Click(object sender, EventArgs e)
{
    search();
}
```

Another auxiliary method will be a cleaning procedure, which will reset the content of the textboxes and it will be called every time an insertion, update or deletion is performed.

```
private void clean()
{
    txtId.Text = String.Empty;
    txtName.Text = String.Empty;
    txtSurname.Text = String.Empty;
}
```

Now, let's code the behavior of the 'Save' button. It will perform two different operations:

- If Id field is empty, it will create a new record with the data inserted in the Name and Surname fields.
- Else, it will update the record whose ID matches the value of the ID textfield..

In this case, we'll make use of parameters as follows:

```
private void btnSave_Click(object sender, EventArgs e)
{
    // If the textbox is empty, it will be an insertion
    if (String.IsNullOrEmpty(txtId.Text))
    {
        if (connection.State != ConnectionState.Open)
            connection.Open();

        // Create a command with the string of the INSERT sentence.
        SQLiteCommand command = connection.CreateCommand();
        command.CommandText = SQLInsert;

        // Add the Name y Surname parameters
        command.Parameters.AddWithValue("Name", txtName.Text);
        command.Parameters.AddWithValue("Surname", txtSurname.Text);

        // Execute the INSERT statement and close the connection
        command.ExecuteNonQuery();
        connection.Close();
```

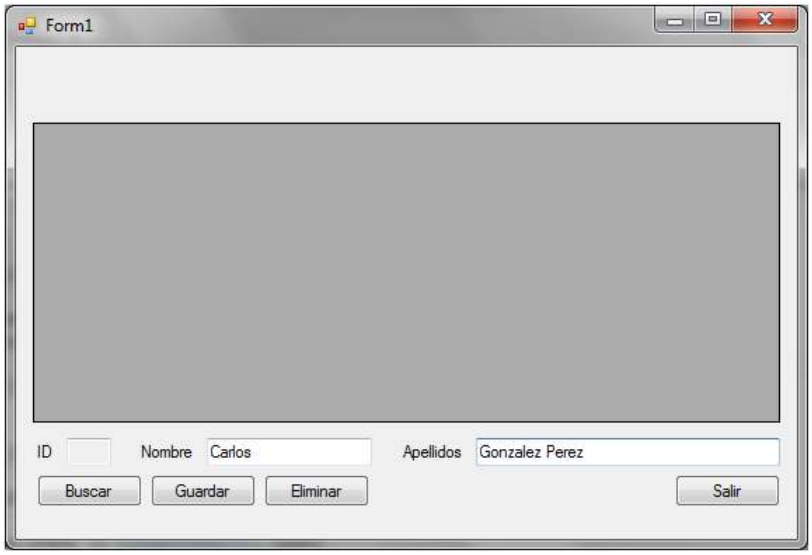
```
    }
    else
    {
        if (connection.State != ConnectionState.Open)
            connection.Open();

        // Create a command with the UPDATE statement string.
        SQLiteCommand command = connection.CreateCommand();
        command.CommandText = SQLUpdate;

        // Add parameters Name, Surname and UserId
        command.Parameters.AddWithValue("Name", txtName.Text);
        command.Parameters.AddWithValue("Surname", txtSurname.Text);
        command.Parameters.AddWithValue("UserId", int.Parse(txtId.Text));

        // Execute the update statement and close the connection
        command.ExecuteNonQuery();
        connection.Close();

    }
    clean();
    search();
}
```

At last, we'll code the delete process, similar to the previous one, passing a parameter with the ID to delete (checking before that the ID field is properly fulfilled)

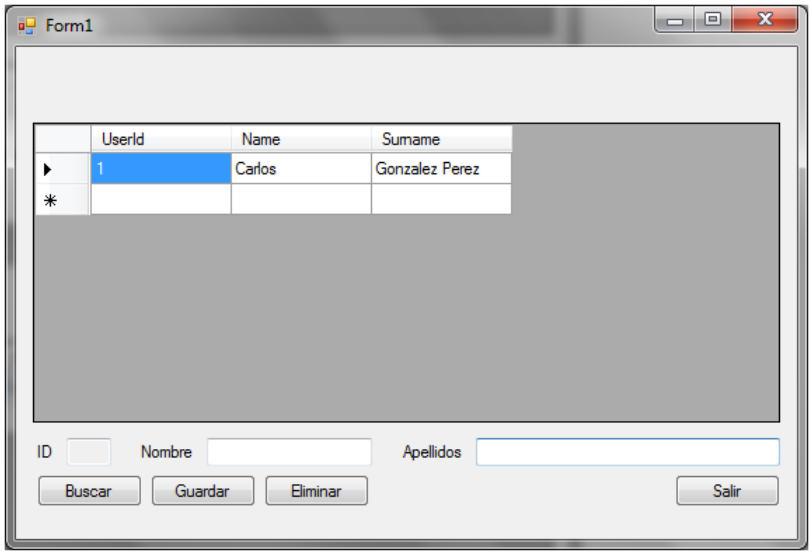
```

private void btnDelete_Click(object sender, EventArgs e)
{
    // Check the existence of a selected ID
    if (!String.IsNullOrEmpty(txtId.Text))
    {
        if (connection.State != ConnectionState.Open)
            connection.Open();

        // Create the command from the DELETE statement
        SQLiteCommand command = connection.CreateCommand();
        command.CommandText = SQLDelete;

        // Add the ID as parameter
        command.Parameters.AddWithValue("UserId", int.Parse(txtId.Text));

        // Execute the SQL statement
        command.ExecuteNonQuery();
        connection.Close();

        clean();
        search();
    }
}
```

Testing the program

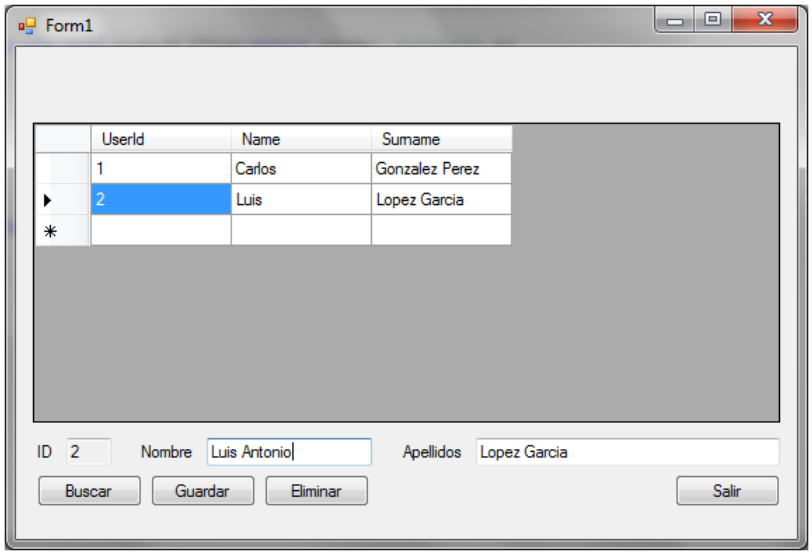
Let's push F5 key and launch the program. Then, we'll fill the Name and Surname fields and then, we'll push *Save*. This will insert a new record on the database, and it must be shown in the GridView, cleaning the textboxes.



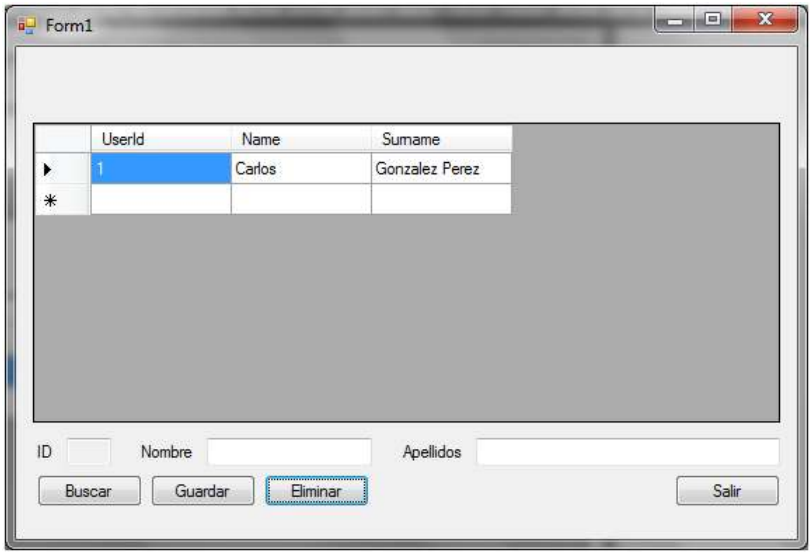
This operation also can be performed pushing the 'Search' button. The result will be as follows:



If we add a second record and then we select it, the ID field will take the value of the record identifier. If we modify its values in the textfields and we push *Save*, the record will be updated in the SQLite database.



Finally, to delete a record, it will be enough with select the column and push the *Delete* button.



This covers the four basic operations over a database, providing a portable application and independent from a complex database engine.

You can download the source code of this example from here.

You can find the Spanish version of this post here.

Posted on Sunday, December 22, 2013 3:18 AM .net , sqlite , example , c# , database , sql , ado.net | Back to top

Comments on this post: Portable databases: using SQLite with .NET

re: Portable databases: using SQLite with .NET

Thanks man Great tutorial well Explained.
I didn't find any tutorial helpful than yours.
Left by ashwin on Feb 27, 2014 1:33 PM

Perfect++

Great topic.That is the definite thing that I've been searching for days.
So useful.So so thank you for this amazing tutorial :)
Left by okhan on Mar 12, 2014 11:11 PM

re: Portable databases: using SQLite with .NET

I'm glad to hear that. Thanks for reading :)
Left by Daniel Garcia on Mar 13, 2014 10:59 PM

re: Portable databases: using SQLite with .NET

Thank's very much, you saved my day, I don't know why people can't make things as easy as your tutorial :)
Left by séddik on Mar 30, 2014 3:50 PM

re: Portable databases: using SQLite with .NET

Thank you so much! :)
Left by Damoeñ Garcoa on Mar 30, 2014 9:43 PM

re: Portable databases: using SQLite with .NET

This link contains step by step insert update delete data using Sqlite with Csharp with Source code available to download
http://geeksprogrammings.blogspot.com/2014/08/using-sqlite-database-csharp.html
Left by Rakib on Aug 04, 2014 3:32 PM

re: Portable databases: using SQLite with .NET

Marvellous explanation of a very useful topic. Spent a whole morning looking for the answer to this!
Thank you very much for the effort employed!

Left by DotNetDumbo on Nov 22, 2014 3:21 PM

re: Portable databases: using SQLite with .NET

This is my first seen tutorial on SQLite,it helps. Thanks
Left by Shequeri on Dec 03, 2014 8:44 PM

re: Portable databases: using SQLite with .NET

I do have a question. I am able to get an un-encrypted sqlite db to work but not an encrypted one. When I do the wizard and click "test connection" it is successful, however after that I age file encrypted or not a database file.

Does any one know of a solution?

Thank you very much in advance!
Left by Bill on Jan 11, 2015 6:18 AM

re: Portable databases: using SQLite with .NET

vry nice article!
this may also help u :
http://www.mindstick.com/Articles/cfb7a11c-5acf-4332-8f91-12e931df34d9
/Crud%20Operation%20in%20ASP%20NET%20Using%20SQLite%20Database
Left by jyotsana on Jan 22, 2015 8:58 AM

re: Portable databases: using SQLite with .NET

Could you please send me VB code for this? I use VB .net and VC C++.
Left by Ash on Apr 03, 2015 8:37 AM

re: Portable databases: using SQLite with .NET

Anybody know where I can find a complete description and program, like this article, except for Visual Studio 2010? We are still using Visual Studio 2010. Everything was moving along find until this article got to modifying the app.config file. This file does not exist in VS 2010. Where do I put the connection strings, etc.?
Left by Joel on May 22, 2015 11:01 PM

Restore an SQLite database using c#

Hi dear Sir/Miss,
I'm developping an application using Visual c# and SQLite as database,
In fact, i have an issue with the SQLite database,i want to create a button that restore an existing SQLite database which has '.db' as extension.
i'm stuck ;-((
Please can you guide me to do so,i'll be very grateful..
i'm using Microsoft Visual Studio 2010 'c#'
Sincerely yours
Hicham
Left by Hicham on May 16, 2016 11:49 PM

Problem with DataGridView

Hi,

I have a problem when I click the star(*) to create a new row. I receive an error System.NullReferenceException. I googled around but didn't found any answer. The closer I got (but don't know if it's causing the error) is that I need to define the defaults values for my DataGrid, but it didn't solved.

```
private void dgTable_DefaultValuesNeeded(object sender, DataGridViewRowEventArgs e) {
e.Row.Cells[0].Value = dgTable.RowCount;
```

```
for(int i = 1; i < dgTable.RowCount; i++) { e.Row.Cells[i].Value = string.Empty; }
}
```

Any suggestions?
Left by Jorge on Aug 30, 2016 9:52 PM

re: Portable databases: using SQLite with .NET

So very much useful information in such a very short article. Thank you!
Left by Vince on Feb 10, 2017 12:43 AM

Portable databases: using SQLite with .NET

"Imports System.Data.Sqlite" is the visual basic version of "using System.Data.SQLite;" which you've used for c; but the imports errors for me. Any idea?
(Otherwise looks great.)
Left by Michael on Apr 15, 2017 6:05 AM

Your comment:

Title:

re: Portable databases: using SQLite with .NET

Name:

Email: (never displayed)

(will show your gravatar)

Comment: *Allowed tags: blockquote, a, strong, em, p, u, strike, super, sub, code*

Verification:

Type the text

Privacy & Terms

Post Comment

POPULAR POSTS ON GEEKS WITH BLOGS

So you want to go Causal Neo4j in Azure? Sure we can do that

BizTalk Server best articles

Redirect Standard Error to Output using PowerShell

MS Dynamics CRM Adapter now available on Enzo Unified

GEEKS WITH BLOGS CONTENT CATEGORIES

ASP.Net SQL Server Apple Google SharePoint Windows Visual Studio Team Foundation Server Agile Office Design Patterns Web Azure

BRAND NEW POSTS ON GEEKS WITH BLOGS

So you want to go Causal Neo4j in Azure? Sure we can do that

MS Dynamics CRM Adapter now available on Enzo Unified

Redirect Standard Error to Output using PowerShell

BizTalk Server best articles

Nested Enumerable::Any in C++/CLI (difficult++)