SQLite Helper (C#)

adriancs

29 Mar 2014 Public Domain

Simplify the usage between C# and SQLite

Download SQLiteHelper (Class Only) V1.2.zip

Download SQLiteHelper V1.2 Demo App

Introduction

SQLite is an open source, embed, cross platform (Windows, IOS, Android, Linux) database engine. It requires no installation and zero configuration in order to work at client's computer.

I have written a small class, **SQLiteHelper** which aims to simplify the usage of SQLite in C#.

Prerequisite

This small class (**SQLiteHelper.cs**) is built on top of **System.Data.SQLite.DLL**. A reference of this DLL must be added into your projects.

Download: https://system.data.sqlite.org

Change Log

27 March 2014 - V1.2

- Added parameter support for "Select", "Execute" and "ExecuteScalar" methods.

22 March 2014 - V1.1

- Add: Parameterized SQL Execution Support
- New Method: Update Table's Structure
- New Method: LastInsertRowId
- New Method: GetTableList
- New Method: ShowDatabase
- New Method: AttachDatabase, DetachDatabase
- Modify: ExecuteScalar applies generics.
- Demo App Updated

List of Simplified Functions

1. GetTableStatus
2. GetTableList
3. GetColumnStatus
4. CreateTable
5. UpdateTableStructure
6. BeginTransaction, Commit, Rollback
7. Select
8. Execute
9. ExecuteScalar
10. Escape
11. Insert
12. Update
13. LastInsertRowId
14. RenameTable
15. CopyAllData
16. DropTable
17. ShowDatabase
18. AttachDatabase, DetachDatabase

Getting Start

Add this using statement at the top of your class:

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

`SQLiteConnection` and `SQLiteCommand` have to be initialized before using **`SQLiteHelper`**:

Example:

```
using (SQLiteConnection conn = new SQLiteConnection("data source=C:\\data"))
{
    using (SQLiteCommand cmd = new SQLiteCommand())
    {
        cmd.Connection = conn;
        conn.Open();

        SQLiteHelper sh = new SQLiteHelper(cmd);

        // do something...

        conn.Close();
    }
}
```

1. GetTableStatus

Get all information of tables in the database.

```
DataTable dt = sh.GetTableStatus();
```

Sample result:

type	name	tbl_name	rootpage	sql
table	sqlite_sequence	sqlite_sequence	3	CREATE TABLE sqlite_sequence(name,seq)

type	name	tbl_name	rootpage	sql
table	person2	person2	5	CREATE TABLE "person2"(id integer primary key autoincrement, name text, tel text, email text, job text, remarks text)
table	player	player	4	CREATE TABLE `player`(id integer primary key autoincrement, lvl integer, weaponid integer, teamid integer, location text, team_name text, remarks text)
table	product	product	6	CREATE TABLE "product"(id integer primary key autoincrement, name text, qty integer)

2. GetTableList

Get a list of tables in database.

```
DataTable dt = sh.GetTableList();
```

3. GetColumnStatus

Get all information of columns in specific table.

```
// Get column's information from table "person"
DataTable dt = sh.GetColumnStatus("person");
```

Sample Result:

cid	name	type	notnull	dflt_value	pk
0	id	integer	0		1
1	lvl	integer	0		0
2	weaponid	integer	0		0
3	teamid	integer	0		0
4	location	text	0		0
5	team_name	text	0		0
6	remarks	text	0		0

4. CreateTable

Create table.

Example table structure: **Person**

Column Name	Data Type	Primary Key	Auto Increment	Not Null	Default Value
id	int	true	true		
name	text				
membershipid	int				
level	decimal				5.5

```
SQLiteTable tb = new SQLiteTable("person");

tb.Columns.Add(new SQLiteColumn("id", true));
tb.Columns.Add(new SQLiteColumn("name"));
tb.Columns.Add(new SQLiteColumn("membershipid", ColType.Integer));
tb.Columns.Add(new SQLiteColumn("level", ColType.Decimal, false, false, "5.5"));

sh.CreateTable(tb);
```

5. UpdateTableStructure

As the name said, it is used to update a table's structure. Maybe you have added new columns, or drop/deleted some columns. This method helps you to update it.

The process at code behind:

- Assume that the old table is named: **person**
- The class creates a temporary table (named: **person_temp**) with your new defined structure.
- Copy all rows from **person** to **person_temp**.
- Drop/delete table of **person**.
- Rename table of **person_temp** to **person**

Code example:

```
SQLiteTable tb = new SQLiteTable();
tb.Columns.Add(new SQLiteColumn("id", true));
tb.Columns.Add(new SQLiteColumn("name"));
tb.Columns.Add(new SQLiteColumn("sku"));
tb.Columns.Add(new SQLiteColumn("code"));
tb.Columns.Add(new SQLiteColumn("category"));
tb.Columns.Add(new SQLiteColumn("remarks"));

sh.UpdateTableStructure("person", tb);
```

6. BeginTransaction, Commit, Rollback

What is transaction?

By default, every SQL query that is sent to SQLite database engine happens in a transaction. The engine automatically **BEGIN** a transaction and **COMMIT** it at the end. **COMMIT** is something like "Make it take effect".

If we send 3 SQL queries (INSERT, UPDATE, DELETE, etc...), 3 transactions are taken place. According to [SQLite official documentation - Frequently Asked Questions]:

> "*...A transaction normally requires two complete rotations of the disk platter, which on a 7200RPM disk drive limits you to about 60 transactions per second...*"

Which means, with a 7200RPM hard disk, the best that we can do is 60 INSERTs (or UPDATE, DELETE, etc) per second.

But, If we manually issue a **BEGIN TRANSACTION**, all the queries will be wrapped in **single** transaction, then SQLite can execute huge amount of queries per second. Somebody said he can execute 10 million per second at [stackoverflow.com], but this is also depends on the speed of hard disk that you are using.

Code example with SQLiteHelper:

```
sh.BeginTransaction();

try
{
    // INSERT.....
    // INSERT.....
    // UPDATE....
    // ... skip for another 50,000 queries....
    // DELETE....
    // UPDATE...
    // INSERT.....

    sh.Commit();
}
catch
{
    sh.Rollback();
}
```

ROLLBACK, in the above example means **Cancel Transaction**. All queries that have sent to SQLite database within that specific transaction are dismissed.

7. Select

Return the query result in **DataTable** format.

- `Select(string sql)`
- `Select(string sql, Dictionary<string, object> dicParameters = null)`
- `Select(string sql, IEnumerable<SQLiteParameter> parameters = null)`

Example 1:

```
DataTable dt = sh.Select("select * from person order by id;");
```

Example 2 (With parameters support):

```
var dic = new Dictionarystring, object();
dic["@aaa"] = 1;
dic["@bbb"] = 1;
DataTable dt = sh.Select("select * from member where membershipid = @aaa and locationid = @bbb;",
dic);
```

Example 3 (With parameters support):

```
DataTable dt = sh.Select("select * from member where membershipid = @aaa and locationid = @bbb;",
    new SQLiteParameter[] {
        new SQLiteParameter("@aaa", 1),
        new SQLiteParameter("@bbb", 1)
    });
```

8. Execute

Execute single SQL query.

- `Execute(string sql)`
- `Execute(string sql, Dictionary<string, object> dicParameters = null)`
- `Execute(string sql, IEnumerable<SQLiteParameter> parameters = null)`

Example:

```
sh.Execute("insert into person(name)values('hello');");
```

9. ExecuteScalar

Return the result of first row first column in specific data type.

- `ExecuteScalar(string sql)`
- `ExecuteScalar(string sql, Dictionary<string, object> dicParameters = null)`
- `ExecuteScalar(string sql, IEnumerable<SQLiteParameter> parameters = null)`
- `ExecuteScalar<datatype>(string sql)`
- `ExecuteScalar<datatype>(string sql, Dictionary<string, object> dicParameters = null)`
- `ExecuteScalar<datatype>(string sql, IEnumerable<SQLiteParameter> parameters = null)`

Example:

```
string a = sh.ExecuteScalar<string>("select 'Hello!';");

int b = sh.ExecuteScalar<int>("select 1000;");

decimal c = sh.ExecuteScalar<decimal>("select 4.4;");

DateTime d = sh.ExecuteScalar<DateTime>("select date('now');");

byte[] e = sh.ExecuteScalar<byte[]>("select randomblob(16);");
```

10. Escape

Escape string sequence for text value to avoid SQL injection or invalid SQL syntax to be constructed.

```
sh.Execute("insert into person(name) values('" + Escape(input) + "');");
```

11. Insert

Insert new row of data. All data will be added as parameters at code behind. This support blob (byte[]) value too.

```
var dic = new Dictionary<string, object>();
dic["name"] = "John";
dic["membershipid"] = 1;
dic["level"] = 6.8;

sh.Insert("person", dic);
```

12. Update

Update row. All data will be added as parameters at code behind. This support blob (byte[]) value too.

Example 1: Update with single condition (where id = 1)

```
var dicData = new Dictionary<string, object>();
dicData["name"] = "no name";
dicData["membershipid"] = 0;
dicData["level"] = 5.5;

sh.Update("person", dicData, "id", 1);
```

Example 2: Update with multiple condition (where membership = 1 and level = 5.5 and teamid = 1)

```
var dicData = new Dictionary<string, object>();
dicData["name"] = "no name";
dicData["status"] = 0;
dicData["money"] = 100;
dicData["dateregister"] = DateTime.MinValue;

var dicCondition = new Dictionary<string, object>();
dicCondition["membershipid"] = 1;
dicCondition["level"] = 5.5;
dicCondition["teamid"] = 1;

sh.Update("person", dicData, dicCondition);
```

13. LastInsertRowId

Get the last issued id (Auto-Increment)

```
sh.Insert("person", dicData);
long id = sh.LastInsertRowId();
```

14. RenameTable

Rename a table.

```
sh.RenameTable("person", "person_backup");
```

15. CopyAllData

Copy all data from one table to another.

```
sh.CopyAllData("person", "person_new");
```

Before copying, `SQLiteHelper` will scan the two tables for match columns. Only columns that exist in both tables will be copied.

16. DropTable

Drop table, delete a table

```
sh.DropTable("person");
```

17. ShowDatabase

Display attached databases.

```
DataTable dt = sh.ShowDatabase();
```

18. AttachDatabase, DetachDatabase

Attach or detach a database

```
sh.AttachDatabase("C:\\data2013.sq3", "lastyeardb");
sb.DetachDatabase("lastyeardb");
```

That's it, guys/girls. Comments are welcome.

Happy coding

Alternative

Lastly, I shall introduce other tools which you might consider when developing C#, VB.NET apps with SQLite:

1. SQLite.NET

> *SQLite.NET is designed to make working with sqlite very easy in a .NET environment. It is an open source, minimal library to allow .NET and Mono applications to store data in [http://www.sqlite.org SQLite 3 databases]. It is written in C# and is meant to be simply compiled in with your projects. It was first designed to work with MonoTouch on the iPhone, but has grown up to work on all the platforms (Mono for Android, .NET, Silverlight, WP7, WinRT, Azure, etc.).*

2. System.Data.SQLite.EF6

History

- 27 Mar 2014 - Release of V1.2
- 22 Mar 2014 - Release of V1.1
- 19 Mar 2014 - Initial work

License

About the Author

adriancs
Software Developer
Malaysia

Programming is an art.

Comments and Discussions

42 messages have been posted for this article Visit **https://www.codeproject.com/Articles/746191/SQLite-Helper-Csharp** to post and view comments on this article, or click **here** to get a print view with messages.