Eén van de voordelen van Sqlite is dat je database als een eenvoudig bestand wordt bewaard. Je hebt dus geen sql server of ander database-systeem nodig. Sqlite is een volledig relationele database die je via de algekende sql-queries kan bewerken en bevragen.

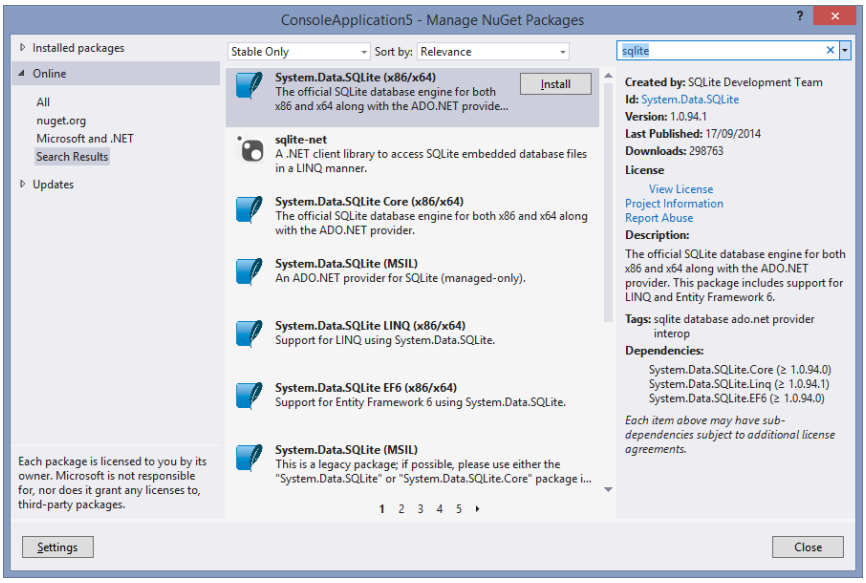
Inhoud

Benodigheden

Sqlite library

Sqlite maakt geen deel uit van de .NET familie. Je hebt dus een library nodig om een sqlite-bestand te kunnen benaderen.

Je kan via Nuget heel makkelijk de nodige dlls toevoegen. Rechterklik hiervoor in je solution explorer op References en kies voor “Manage Nuget Packages…”



Zoek vervolgens naar “sqlite” in de online nuget.org bibliotheek.

Je hebt de “System.Data.Sqlite (x86/x64)” bibliotheek nodig. Kies deze en klik op “install”.

Na een korte installatie zal je merken dat er bij je references enkele zaken werden toegevoegd.

Je kan nu in je C#-projecten de System.Data.SqQLite namespace toevoegen:

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

Vanaf nu kan je beginnen programmeren.

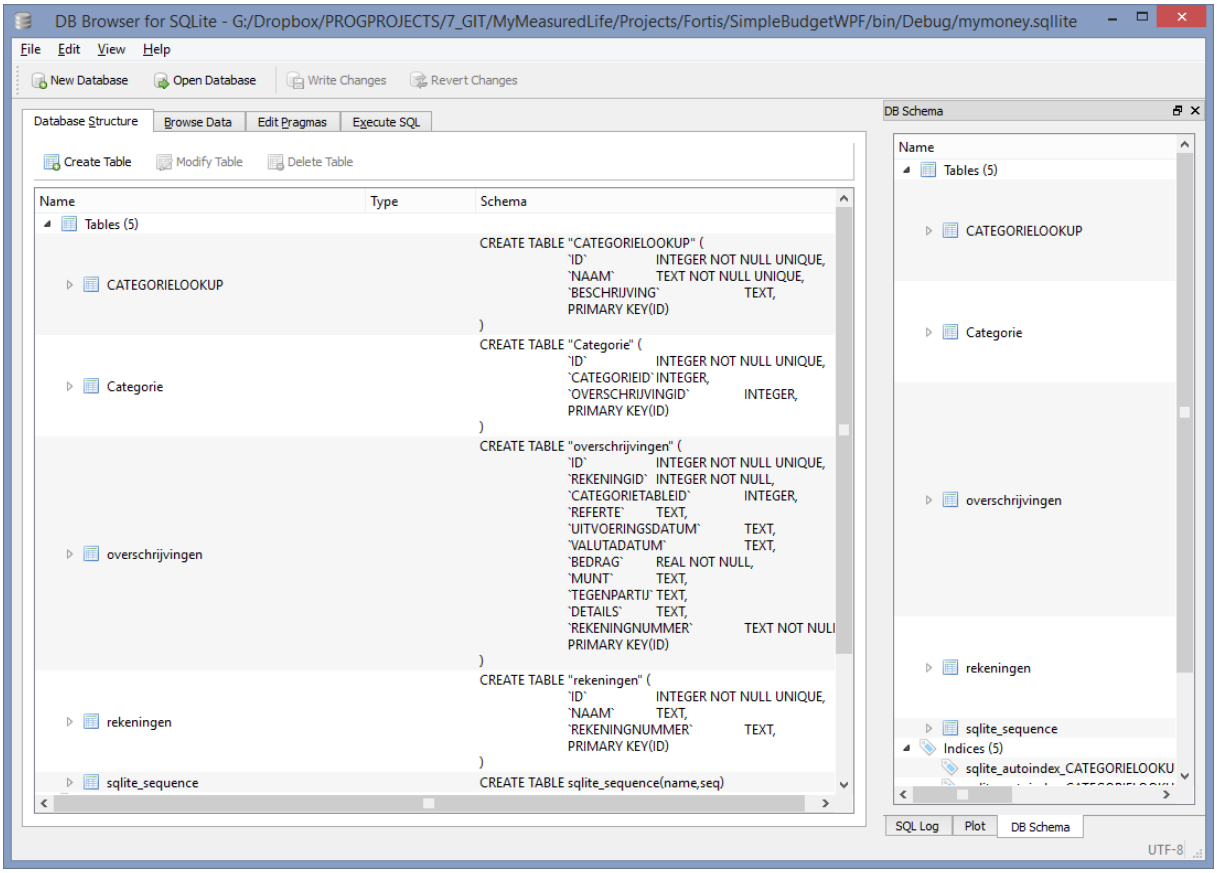
SqliteBrowser

SqliteBrowser is een krachtig, maar eenvoudig te gebruiken programma dat je toelaat om

- Een nieuwe sqlite database file aan te maken waarbij je alle tabellen,relaties en kollomen kan definieren
- Bestaande sqlite database files kunt openen om queries er op te testen
- Data en structuur van bestaande sqlite databases bekijken én aanpassen

Je kan de tool downloaden via:

http://sqlitebrowser.org/



De werking van deze tool is redelijk eenvoudig. Voor beginnende gebruikers is het aan te raden om van de “Edit Pragmas” tab weg te blijven.

Programmeren met sqlite

We bekijken nu de manieren die je kan hanteren om tegen een sqlite database te programmeren (voor een uitgebreide uitleg, kijk zeker naar http://blog.tigrangasparian.com/2012/02/09/getting-started-with-sqlite-in-c-part-one/)

Beginnen met een lege database?

Een van de eerste vragen die je je moet stellen is:

- moet m'n programma werken met een **bestaande** sqlite db,
- moet het programma zelf een lege sqlite db **genereren**
- maakt het programma gebruikt van een **lege, maar reeds bestaande** sqlite db

Bestaande

Indien wordt gewerkt met een bestaande database dan mag je ineens naar de stap "Verbindingen met een db maken" gaan.

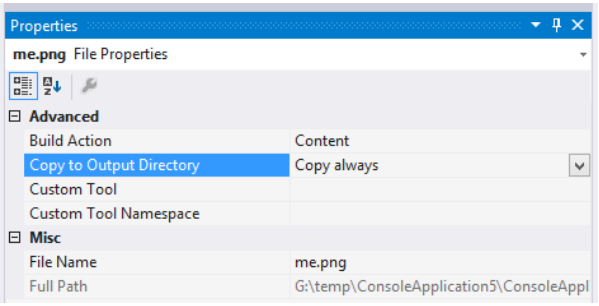
Lege

Indien je gebruik wenst te maken van een lege, reeds bestaande sqlite db , die je bijvoorbeeld reeds hebt gemaakt via de sqlitebrowser, dan dien je gewoon manueel deze file naar de juiste plaats te kopieren.

Voeg hiervoor eerst je aangemaakt database file toe aan je project. Rechterklik hiervoor op je project in de solution explorer en kies voor "Add…" , dan "Existing item…".

Vervolgens selecteer je je database bestand en klikt op ok.

Selecteer nu het bestand in je solution explorer en zorg ervoor dat bij de properties van dit bestand je de optie "Copy to output directory" kiest voor Copy Always:



Telkens je nu je project build zal dit bestand mee naar je bin/debug folder worden gekopieerd. Indien je het bestand in je project in een map plaatste dan zal de locatie van het bestand in die map onder bin/debug geplaatst worden. (handig voor je assets van een groot spel bijvoorbeeld)

Vervolgens kan je nu dit bestand kopieren naar de gewenste naam in je code (stel dat je de file "emptydatabase.sqlite" reeds hebt aangemaakt en in je project hebt toegevoegd:

```
if(File.Exists("mydatabase.sqlite") == false)
{
    File.Copy("emptydatabase.sqlite", "mydatabase.sqlite");
}
```

Genereren en connecteren

Het automatisch generen van een nieuwe database heeft uiteraard iets meer voeten in de aarde.

Eerst maak je een nieuwe sqlite –db aan als volgt:

```
SQLiteConnection.CreateFile("MyDatabase.sqlite");
```

Waarbij MyDatabase.sqlite je zelfgekozen naam voor de db is. De extensie ben je niet verplicht om sqlite te noemen.

Vervolgens maak je een verbinding met je database als volgt:

```
var m_dbConnection = new SQLiteConnection("Data Source=MyDatabase.sqlite;Version=3;");
```

Deze zogenaamde connection string kan extra details bevatten zoals een wachtwoord, readonly access e.d. Meer informatie vind je hier: http://www.connectionstrings.com/sqlite/ .

Rest je nu enkel nog het openen van een connectie

```
m_dbConnection.Open();
```

VERGEET NIET TE SLUITEN (via .Close()) OP HET EINDE.

Tabel genereren

Je kan nu de nodige tabellen generen door gewoon de nodig sql te schrijven. Als je dus bijvoorbeeld een tabel highscores wenst bestaande uit twee velden voor de namen (type varchar) en de scores (type int). Dan schrijven we de dit in een string:

```
string sql = "create table highscores (name varchar(20), score int)";
```

Vervolgens maak je een SqliteCommanda object waarin je de string plaatst alsook de connectie naar de database:

```
SQLiteCommand command = new SQLiteCommand(sql, m_dbConnection);
```

```
En dan voeren we de actie uit:
```

command.ExecuteNonQuery();

Tabel vullen

Een tabel vullen behelst dezelfde stappen volgen, alleen dien je nu in je string uiteraard een andere query te beschrijven, bijvoorbeeld

```
string sql = "insert into highscores (name, score) values ('Me', 9001)";
```

En dan weer

```
SQLiteCommand command = new SQLiteCommand(sql, m_dbConnection);
command.ExecuteNonQuery();
```

Verbindingen met een db maken

Indien je reeds een bestaande database hebt (of je hebt er een aangemaakt zoals in vorige stap), dan kan je dus verbinding maken als volgt:

```
var m_dbConnection = new SQLiteConnection("Data Source=MyDatabase.sqlite;Version=3;");
```

Waarbij MyDatabase.sqlite je zelfgekozen naam voor de db is. De extensie ben je niet verplicht om sqlite te noemen.

Deze zogenaamde connection string kan extra details bevatten zoals een wachtwoord, readonly access e.d. Meer informatie vind je hier: http://www.connectionstrings.com/sqlite/ .

Rest je nu enkel nog het openen van een connectie

```
m_dbConnection.Open();
```

VERGEET NIET TE SLUITEN (via .Close()) OP HET EINDE.

Queries op je db uitvoeren

We hebben reeds gezien dat tabellen aanmaken en data NAAR de database sturen (update, insert en delete) steeds gebruik maakt van het ExecuteNonQuery() commando.

Wanneer we echter ook data willen terugkrijgen dan zullen we gebruik moeten maken van een DataReader. Dat tonen we nu.

Uiteraard beginnen we weer eerst met het maken van string die onze query bevat

```
string sql = "select * from highscores order by score desc";
SQLiteCommand command = new SQLiteCommand(sql, m_dbConnection);
```

Vervolgens voeren we de query via het command object uit, maar gebruiken we d ExecuteReader – methode:

```
SQLiteDataReader reader = command.ExecuteReader();
```

Het teruggegeven reader-object gebruiken kunnen we nu gebruiken om lijn per lijn de data uit te lezen:

```
while (reader.Read())
    Console.WriteLine("Name: " + reader["name"] + "\tScore: " + reader["score"]);
```

Telkens je de Read() methode oproept op de reader zal deze een nieuwe lijn in z'n datavelden laden. Dit blijft werken tot alle lijnen (rows) zijn verwerkt en dan zal de reader "false" teruggeven)

Je kan na iedere read ieder veld uitlezen door de naam van de kolom (columns) als dictionary index mee te geven aan de reader, bv:

```
reader["name"]
```

Let er op dat je deze data steeds van het type object terugkrijgt. Wil je dus de data in de juiste vorm krijgen om er iets mee te doen dan moet je deze eerst nog casten. Stel dus dat je de score met eentje wil verhogen dan zou je kunnen schrijven:

```
int score = (int)reader["score"];
score++;
```