Лабораторная работа №4
Работа с СУБД SQLITE

Введение

Когда создаются программы, использующее одну или несколько таблиц баз данных и (или) таблицы базы данных требуются только одному приложению (ограниченному числу приложений), то достаточно неудобно при сопровождении приложения отслеживать установки клиентского программного обеспечения для работы с базами данных, выполнять соответствующие настройки, решать вопросы безопасности и т.д., и т.п.

С точки зрения пользователя - SQLite, это одиночный файл на диске. Этот файл не требует для своего использования запуска сторонних процессов, наличия СУБД и дополнительных средств администрирования.

Протокол обмена обеспечивают вызовы функций (API) библиотек SQLite (библиотеки занимают чуть более 500кб. памяти), что не требует от программиста дополнительных знаний структуры БД и особенностей организации взаимодействия. Движок SQLite и интерфейс к ней реализованы в одной библиотеке.

От программиста на C#, работающего в Microsoft Visual Studio, не требуется дополнительных знаний по обеспечению обмена данными, обмен обеспечивается с использованием тех же компонент и функций языка.

Нет проблем с развертыванием и переносом базы данных с одного компьютера на другой. Сама база данных может поставляться вместе с приложением и копироваться другими пользователями в аналогичные программы. Она реализует основные возможности стандарта SQL 92 (включая триггеры и транзакции).

SQLite обеспечивает поддержку объема данных до 2-х терабайт, а размер записи (при использовании полей BLOB) ограничен только памятью компьютера.

Поддерживается небольшое, но вполне достаточное для большинства задач, количество типов данных.

Однако простота накладывает и ограничения. Приходится жертвовать отдельными характеристиками, характерными для больших баз данных, и, прежде всего, параллелизмом процессов, богатством встроенных библиотечных функций, все же небольшим ограничением стандарта языка SQL и некоторыми другими характеристиками. Но это не мешает все более широкому распространению SQLite для прикладных программ с небольшими базами данных, приложений, не требующих администрирования баз данных, для создания временной базы данных в процессе работы приложений, в качестве базы данных доступа и паролей на небольших Web порталах и для многих других целей.

Основы использования SQLite

Для использования SQLite в программах на C#, можно использовать два подхода:

1. Скачать соответствующие файлы библиотек провайдера, разместить их на своем компьютере и при формировании решения непосредственно использовать ссылки на данные библиотеки.

2. Скачать и установить соответствующий провайдер, интегрируемый в Visual Studio и использовать контролы провайдера, как и все другие визуальные контролы среды.

Есть и платные, и бесплатные решения (см. сайт sqlite.org).

Managed Only - для приложений, которые должны работать в Windows и в Linux-based операционных системах.

Compact Framework - для работы в среде .NET CF для мобильных устройств под управлением Windows CE.

Itanium - для работы на процессорах Intel Itanium.

x64 - версия для платформ с архитектурой x64.

x86 - версия для платформ с архитектурой x86.

Менеджеры для работы с БД

Есть большое количество платных и бесплатных менеджеров БД SQLite, но при использовании встраиваемых в Visual Studio провайдеров, появляется возможность доступа к БД из Server Explorer студии, и, в зависимости от провайдера, некоторые функции по манипулированию таблицами баз данных и составления запросов.

Подготовка демонстрационного решения

Создадим простейшее Windows Application решение.

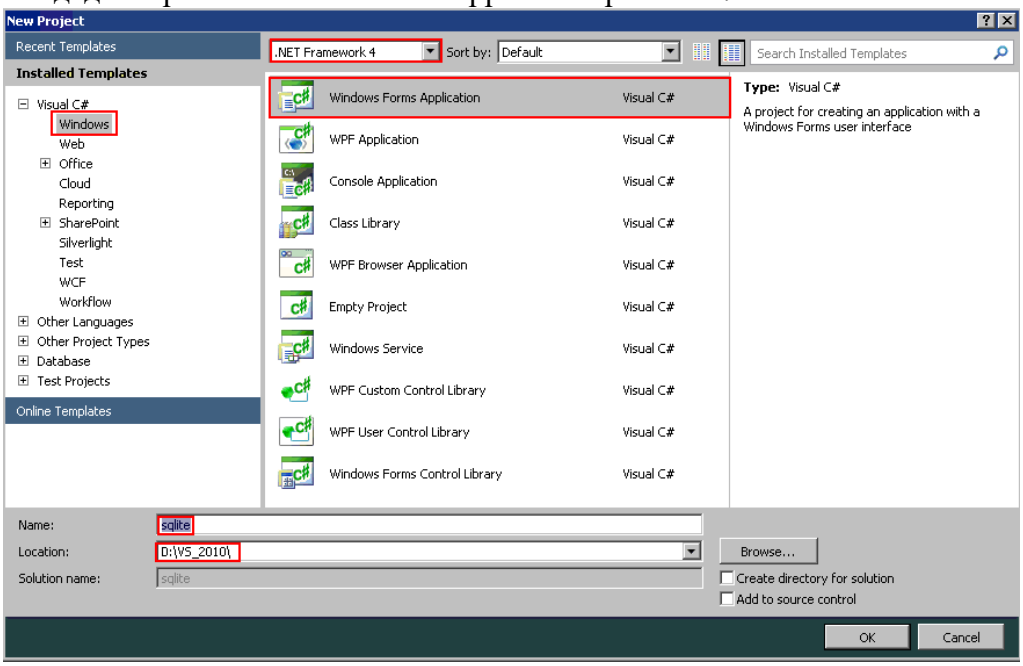

Рис.1 Создание решения приложения

На форму приложения поместим 6 кнопок, как показано на Рис.2.

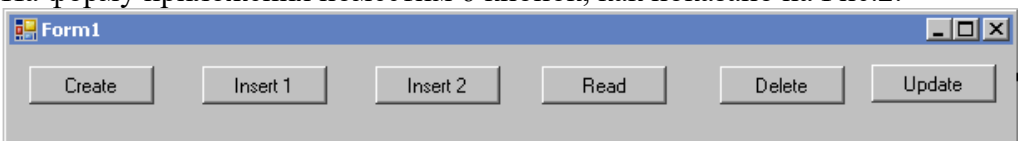

Рис.2 Решение приложения

Функции для работы с SqLite будут оформляться в классе "sqliteclass", для чего создадим этот класс, как показано на Рис.3., кликнув в Solution Explore на узле "sqlite" правой кнопкой мышки и выбрав пункты контекстного меню "Add" и "New Item".

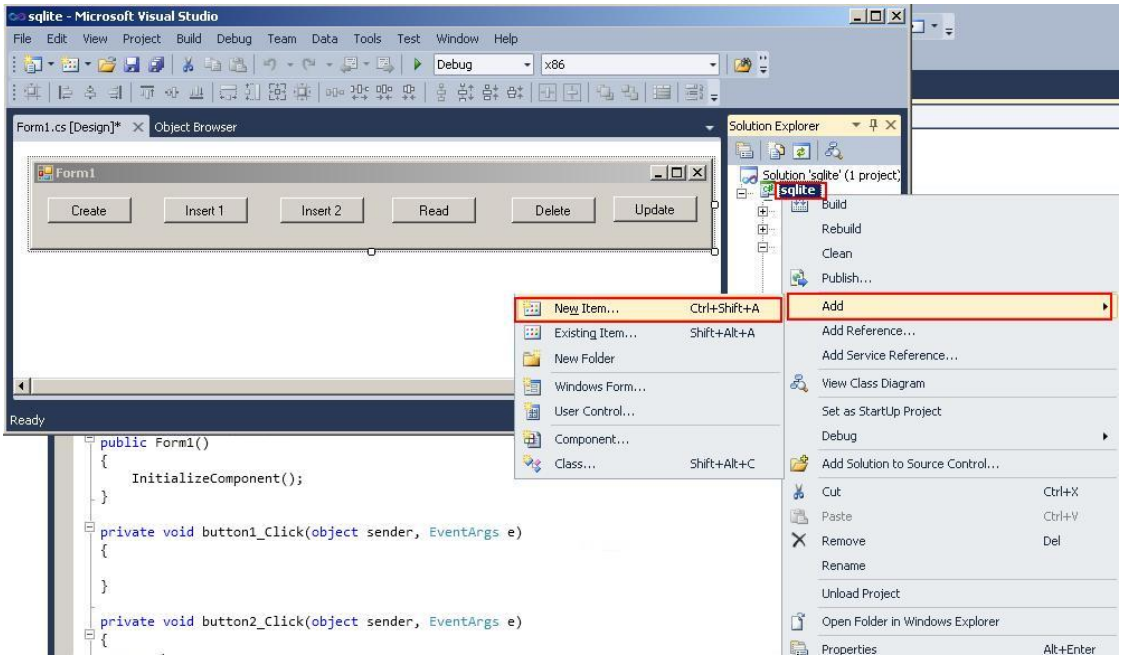

Рис.3 Добавление класса для работы с SQLite

В появившемся окне коллекции темплат "Add New Item" выберем "Class", в поле "Name" введем имя класса - "sqliteclass.cs" и нажмем кнопочку "Add". Класс создан - Рис.4.

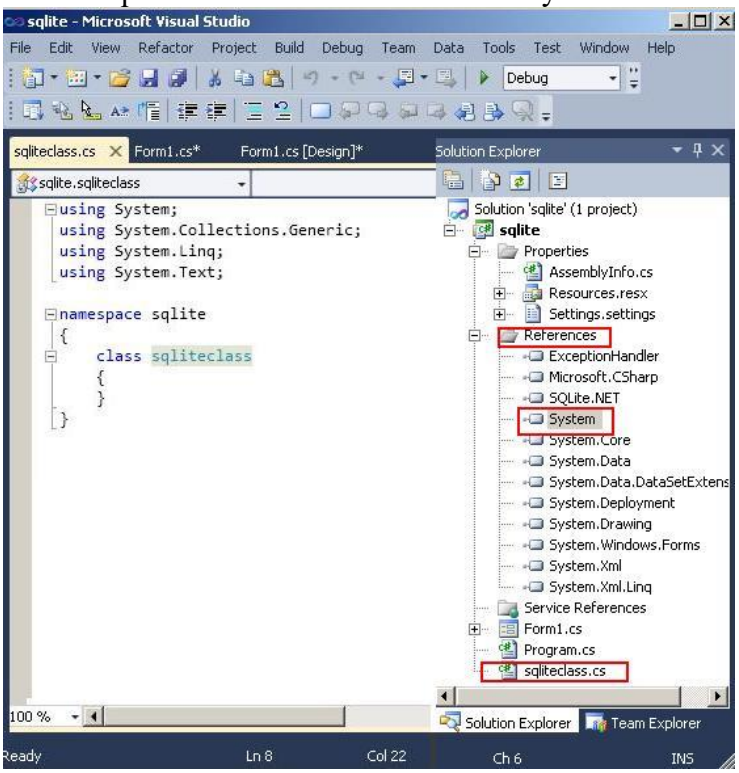

Рис.4 Класса для размещения функций для работы с SQLite

Теперь нужно включить в проект библиотеки провайдера SQLite.

Кликаем правой кнопкой мышки на закладке References (Рис.4.), выбираем пункт контекстного меню "AddReference...", в окне "AddReference" выбираем закладку "Browse", находим папку, куда были распакованы библиотеки (SQLite3NET) и последовательно выбираем в проект библиотеки "SQLite.Net.dll" и "ExceptionHandler.dll" (Рис.5.).

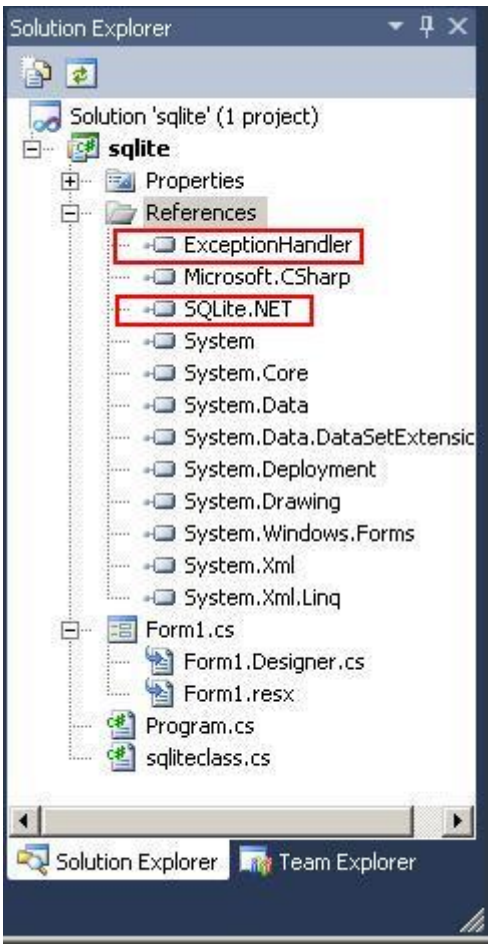

Рис.5 Добавление библиотек провайдера SQLite в решение

Переходим на закладку sqliteclass решения, и добавляем пространство имен "Finisar.SQLite" и "System.Data" - они необходимы для работы с базай данных.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO
using System.Data;
using Finisar.SQLite;
namespace sqlite
{
    class sqliteclass
    {
    }
}
```

На этом этапе наше решение готово для создания и организации взаимодействия с БД SQLite.

Организация взаимодействия с SQLite

Переходим на вкладку "Form1.cs" и добавляем переменные, которые нам понадобятся для работы, и, прежде всего, экземпляр класса "sqliteclass" - "mydb".

```
namespace sqlite
{
 public partial class Form1 : Form
 {
  private sqliteclass mydb = null;
  private string sPath = string.Empty;
  private string sSql = string.Empty;
.....
```

Базу данных будем создавать в директории приложения. Ее имя определим "mybd.db" (желательные расширения файла базы данных - это "db" или "db3", по крайней мере, менеджеры баз данных предлагают выбрать нам именно файл баз данных с этими расширениям, но расширение, в принципе, не играет роли). Кроме того, нам понадобится путь, где находится "exe" файл приложения и полный путь к файлу базы данных - переменная "sPath". Перейдем на вкладку "Form1.cs.[Design]" и дважды кликнем по заголовку окна решения. Будет создан обработчик загрузки формы приложения "Form1_Load", в нем и определим "sPath" (в C# есть множество способов определения того, где находится "exe" файл, это один из многих).

```
private void Form1_Load(object sender, EventArgs e)
{
 sPath = Path.Combine(Application.StartupPath, "mybd.db");
 //Чтобы не плодить MessageBox все сообщения выводим в заголовок формы
 Text = sPath;
}
```

В начало

3.2. Класс для работы с базой данных

Мы знаем, что SQL операторы можно подразделить на два основных типа:

- Не возвращающие значения из базы данных (в лучшем случае возвращающие число записей, которые затронуло их выполнение - "Insert", "Delete", "Update").
- Возвращающие запрошенную информацию из таблиц баз данных.

Поэтому в нашем классе создадим две функции для этих случаев, соответственно `iExecuteNonQuery` и `drExecute`. Первая будет возвращать число затронутых ею записей, вторая массив выбранных строк `DataRow`.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Finisar.SQLite;
using System.Data;

namespace sqlite
{
 class sqliteclass
 {
   //Конструктор
   public sqliteclass()
   {

   }
   #region iExecuteNonQuery
   public int iExecuteNonQuery(string FileData, string sSql, int where)
   {
    int n = 0;
    try
    {
     using (SQLiteConnection con = new SQLiteConnection())
     {
```

```
      if (where == 0)
      {
       con.ConnectionString    =    @"Data    Source="    +    FileData    +
";New=True;Version=3";
      }
      else
      {
        con.ConnectionString = @"Data Source=" + FileData +
";New=False;Version=3";
      }
      con.Open();
      using (SQLiteCommand sqlCommand = con.CreateCommand())
      {
       sqlCommand.CommandText = sSql;
       n = sqlCommand.ExecuteNonQuery();
      }
      con.Close();
     }
    }
    catch (Exception ex)
    {
     n = 0;
    }
    return n;
   }
   #endregion
   #region drExecute
   public DataRow[] drExecute(string FileData, string sSql)
   {
    DataRow[] datarows = null;
    SQLiteDataAdapter dataadapter = null;
    DataSet dataset = new DataSet();
    DataTable datatable = new DataTable();
    try
    {
       using (SQLiteConnection con = new SQLiteConnection())
      {
       con.ConnectionString    =    @"Data    Source="    +    FileData    +
";New=False;Version=3";
       con.Open();
       using (SQLiteCommand sqlCommand = con.CreateCommand())
       {
        dataadapter = new SQLiteDataAdapter(sSql, con);
        dataset.Reset();
        dataadapter.Fill(dataset);
        datatable = dataset.Tables[0];
        datarows = datatable.Select();

       }
      con.Close();
     }
    }
    catch (Exception ex)
    {

     datarows = null;
    }
    return datarows;

   }
   #endregion
 }
}
```

Отличие от работы с другими базами данных - это наличие нескольких собственных невизуальных контролов. Первый, это свой коннектор - "SQLiteConnection", второй - собственный провайдер команд - "SQLiteCommand", и третий - свой адаптер данных, "SQLiteDataAdapter". Других контролов, отличных от MS SQL и Oracle не требуется.

"ConnectionString" требует указания полного пути к БД. Сами процессы связи с БД также стандартные. Обратим внимание на параметр "New", равный "True" или "False", который, как ясно по контексту, при значении "True", создает новою базу данных, а при значении "False" - нет. Причем, его значение приоритетнее "if not exists" в SQL операторе "CREATE TABLE if not exists..."

Создание базы данных

Будем создавать базу данных "mybd.db", путь к файлу которой мы определили в переменной "sPath". В базе данных создадим таблицу "birthday" со значениями полей "ФИО", "Даты рождения" и "отметки о поздравлении в данном году":

```
private void button1_Click(object sender, EventArgs e)
{
 //Создаем экземпляр нашего класса
 mydb = new sqliteclass();
 sSql = @"CREATE TABLE birthday]([id] INTEGER PRIMARY KEY AUTOINCREMENT";
 sSql +=",[FIO] TEXT NOT NULL,[bdate] datetime NOT NULL,[gretinyear] INTEGER
DEFAULT 0);";
 //Пытаемся создать базу данных и таблицу одновременно, так как создание
 //базы данных есть всего лишь процесс создания пустого файла при организации
 //connection, или указание на файл, если БД (файл) существует
 //Третий параметр функции говорит, что БД создается вновь
 mydb.iExecuteNonQuery(sPath, sSql, 0);
```

Если мы запустим на данном этапе решение на выполнение, то получим необрабатываемое исключение. Причина в том, что для работы с SQLite ***все файлы библиотек провайдера должны лежать где и "exe" файл приложения - в сборке***.

Те ссылки, библиотеки которых необходимы в сборке приложения, можно копировать в сборку автоматически. Для этогов Solutation Explorer надо выбрать окно "Properties" ссылки и установить свойство "Copy Local" в "True".

После запуска приложения на выполнение можно убедиться, что база данных создана (mybd.db).

Ни при создании базы, ни при создании таблицы, sqlCommand.ExecuteNonQuery ничего не возвращает, вернее возвращает "0". Поэтому надо как-то проверить то, что мы создали. Можно сделать это так (продолжаем код, приведенный выше):

```
 //Проверка работы
 sSql = @"insert into birthday (FIO,bdate,gretinyear)";
 sSql +=" values('Александр Сергеевич Пушкин','1799-06-06',0);";
 if (mydb.iExecuteNonQuery(sPath, sSql, 1) == 0)
 {
   Text = "Ошибка проверки таблицы на запись,";
   Text +=" таблица или не создана или не прошла запись тестовой строки!";
   mydb = null;
   return;
 }
 sSql = "select * from  birthday";
 DataRow[] datarows = mydb.drExecute(sPath, sSql);
```

```
 if (datarows == null)
 {
  Text = "Ошибка проверки таблицы на чтение!";
  mydb = null;
  return;
 }
 Text = "";
 foreach (DataRow dr in datarows)
 {
  Text += dr["id"].ToString().Trim() + dr["FIO"].ToString().Trim() +
         dr["bdate"].ToString().Trim() + " ";
 }
 sSql = "delete from  birthday";
 if (mydb.iExecuteNonQuery(sPath, sSql, 1) == 0)
 {
  Text = "Ошибка проверки таблицы на удаление записи!";
  mydb = null;
  return;
 }

 Text = "Таблица создана!";
 mydb = null;
return;
}//private void button1_Click(object sender, EventArgs e)
```

Работа с базой данных

Работа с базой данных заключается в использовании функций класса для чтения и записи.

```
private void button2_Click(object sender, EventArgs e)
{
 mydb = new sqliteclass();
 sSql = @"insert into birthday (FIO,bdate,gretinyear)";
  sSql +=" values('Александр Сергеевич Пушкин','1799-06-06',0);";
 //Проверка работы
 if (mydb.iExecuteNonQuery(sPath, sSql, 1) == 0)
 {
  Text = "Ошибка записи!";
 }
 mydb = null;
 Text = "Запись 1 добавлена!";
 return;
}
private void button3_Click(object sender, EventArgs e)
{
 mydb = new sqliteclass();
 //Ошибка в дате специально для последующего исправления
 sSql = @"insert into birthday (FIO,bdate,gretinyear)";
 sSql +=  values('Толстой Лев Николаевич','1928-08-28',0);";
 //Проверка работы
 if (mydb.iExecuteNonQuery(sPath, sSql, 1) == 0)
 {
  Text = "Ошибка записи!";
 }
 mydb = null;
 Text = "Запись 2 добавлена!";
}

private void button4_Click(object sender, EventArgs e)
{
 mydb = new sqliteclass();
 sSql = "select * from  birthday";
 DataRow[] datarows = mydb.drExecute(sPath, sSql);
```

```
 if (datarows == null)
 {
  Text = "Ошибка чтения!";
  mydb = null;
  return;
 }
 Text = "";
 foreach (DataRow dr in datarows)
 {
  Text += dr["id"].ToString().Trim() + "  " + dr["FIO"].ToString().Trim()
        + "  " + dr["bdate"].ToString().Trim() + " ";
 }
 mydb = null;
}

private void button5_Click(object sender, EventArgs e)
{
 mydb = new sqliteclass();
 sSql = "delete from  birthday";
 if (mydb.iExecuteNonQuery(sPath, sSql, 1) == 0)
 {
  Text = "Ошибка удаления записи!";
   mydb = null;
   return;
 }
 mydb = null;
 Text = "Записи удалены из БД!";
}

private void button6_Click(object sender, EventArgs e)
{
 mydb = new sqliteclass();
 sSql = @"Update birthday set bdate='1828-08-28' where FIO like('%Толстой%');";
 if (mydb.iExecuteNonQuery(sPath, sSql, 1) == 0)
 {
  Text = "Ошибка обновления записи!";
  mydb = null;
  return;
 }
 mydb = null;
 Text = "Запись 2 исправлена!";
}
```

Работа с базой данных показана на Рис.6.

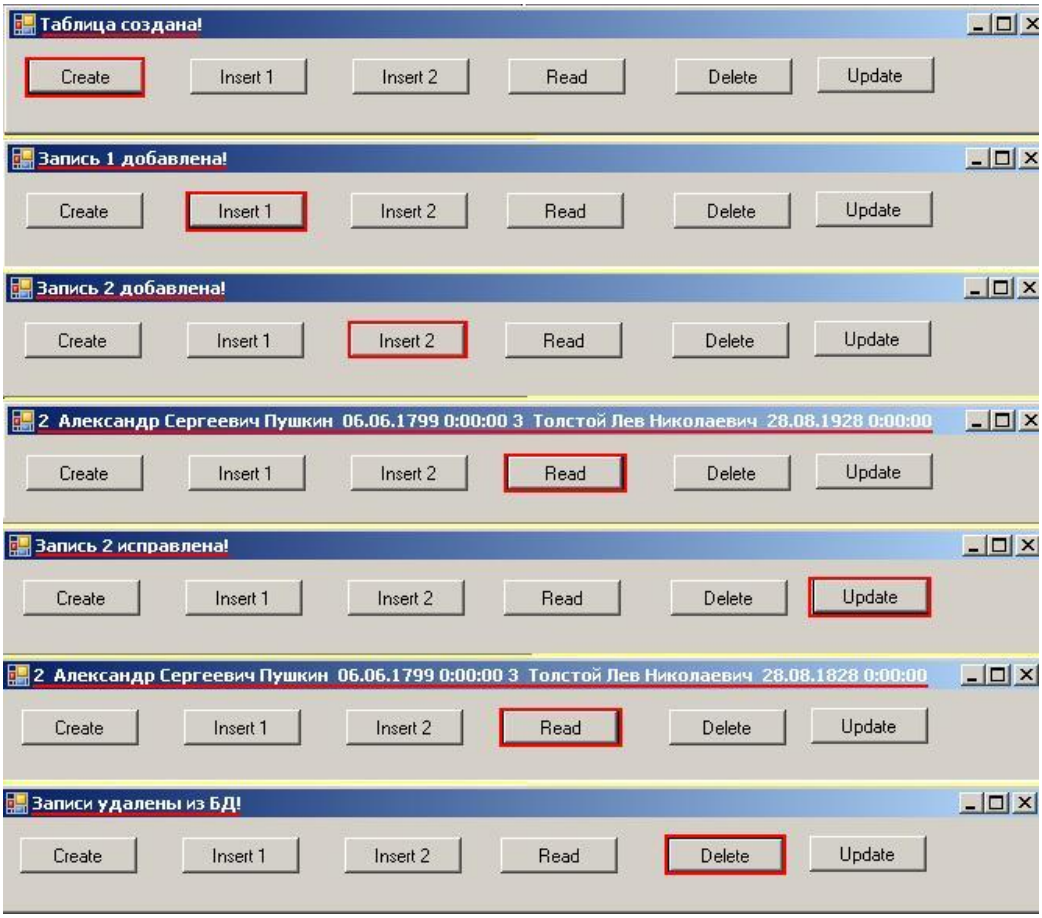

Рис.6 Работа с базой данных SQLite