Лабораторная работа 2

Типы данных SQLite. Использование встроенных в Visual Studio провайдеров SQLite. Работа с ошибками в базе данных.

Работа с Blob типами, хранение изображений

Теоретическая часть

ЧАСТЬ 1.

Наиболее удобно посмотреть типы данных **SQLite** в менеджере **SQLite Expert Professional**. Для этого нажмем кнопку "New Table", введем имя таблицы и нажмем кнопку "Add", в окне "New Field" в выпадающем списке можем посмотреть все возможные на данный момент типы данных.

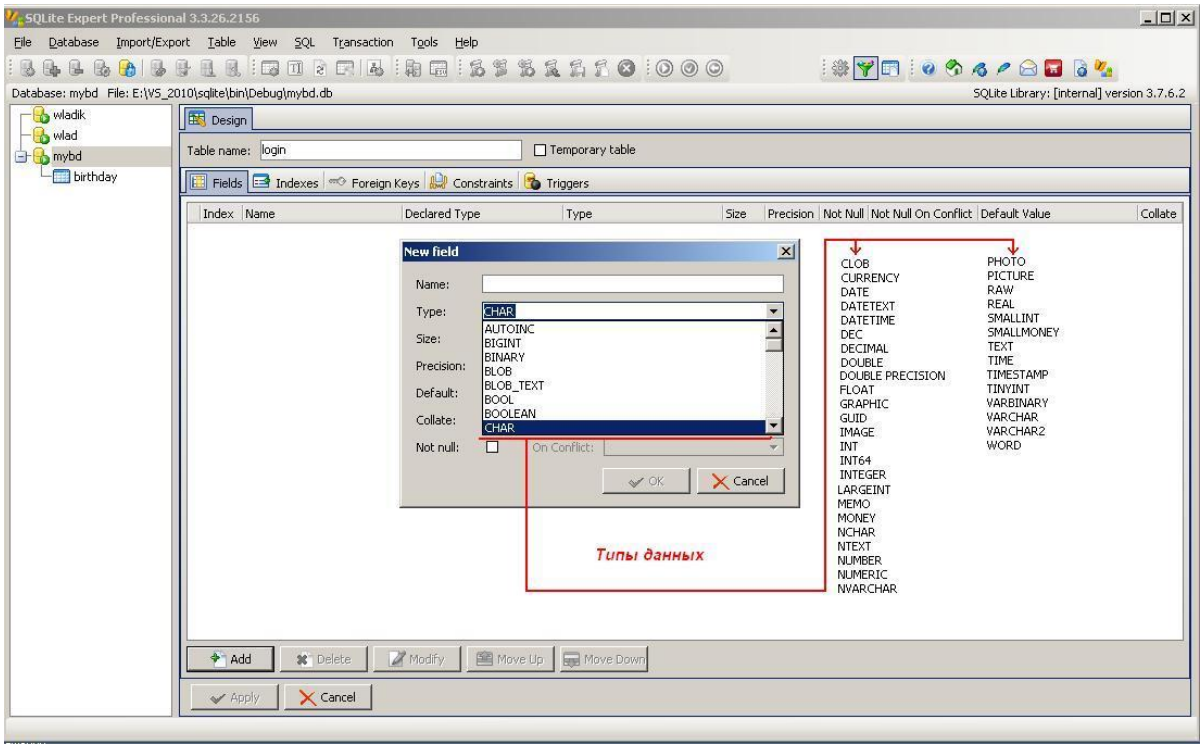


Рис.Типы данных SQLite

Действительно, при создании таблицы можно задать показанные выше типы, но это, в большинстве своем, так называемые **аффинированные типы**, или типы, которые провайдер приведет к одному из следующих типов хранения:

- NULL. Пустое значение в таблице базы.
- INTEGER. Целочисленное значение, хранящееся в 1, 2, 3, 4, 6 или 8 байтах, в зависимости от величины самого значения.
- REAL. Числовое значение с плавающей точкой. Хранится в формате 8-байтного числа IEEE с плавающей точкой.
- TEXT. Значение строки текста. Хранится с использованием кодировки базы данных (UTF-8, UTF-16BE или UTF-16LE).

- BLOB. Значение бинарных данных, хранящихся точно в том же виде, в каком были введены.

Ниже приведены правила **аффинированности** столбца по объявлению типа столбца и таблица преобразования объявленных данных в аффинированные типы в соответствии с пятью правилами:

1. Если объявление типа содержит строку "INT", то столбец ассоциируется с аффинированным INTEGER.

2. Если объявление типа столбца содержит любую из строк "CHAR", "CLOB", или "TEXT", то аффинированность определяется как TEXT. Обратите внимание, что тип VARCHAR содержит подстроку "CHAR", и поэтому ему тоже сопоставляется аффинированный TEXT.

3. Если объявление типа столбца содержит строку "BLOB" или если тип не указан, то столбец аффинируется с NONE.

4. Если объявление типа столбца содержит любую из строк "REAL", "FLOA" или "DOUB", аффинированность определяется как REAL.

5. В остальных случаях столбцу сопоставляется аффинированный NUMERIC.

Примеры названий типов из запросов `CREATE TABLE` или выражения `CAST`	**Результирующая аффинированность**	**Правило, использованное для определения аффинированности**
INT INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8	INTEGER	1
CHARACTER(20) VARCHAR(255) VARYING CHARACTER(255) NCHAR(55) NATIVE CHARACTER(70) NVARCHAR(100) TEXT CLOB	TEXT	2
BLOB нет указания типа данных	NONE	3
REAL DOUBLE DOUBLE PRECISION FLOATNONE	REAL	4
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME	NUMERIC	5

Использование встроенных в Visual Studio провайдеров SQLite

В Visual Studio имеется бесплатный провайдер " dotConnect for SQLite " (можно найти на сайте http://www.devart.com).

Скачаем его и установим, предварительно закрыв все Visual Studio Net.

В аннотации к провайдеру сказано:

Note: This package works with .NET Framework 2.0, 3.0, 3.5, and 4.0

Убедимся в этом. Откроем проект, который мы создали, удалим все ссылки в "Solution Explorer" в закладке "References" на SQLite, а также пространство имен в "sqliteclass.cs":

```
using Finisar.SQLite;
```

Выберем Toolbox (меню "View", "Toolbox") и увидим новую закладку "SQLiteData" (Рис.).

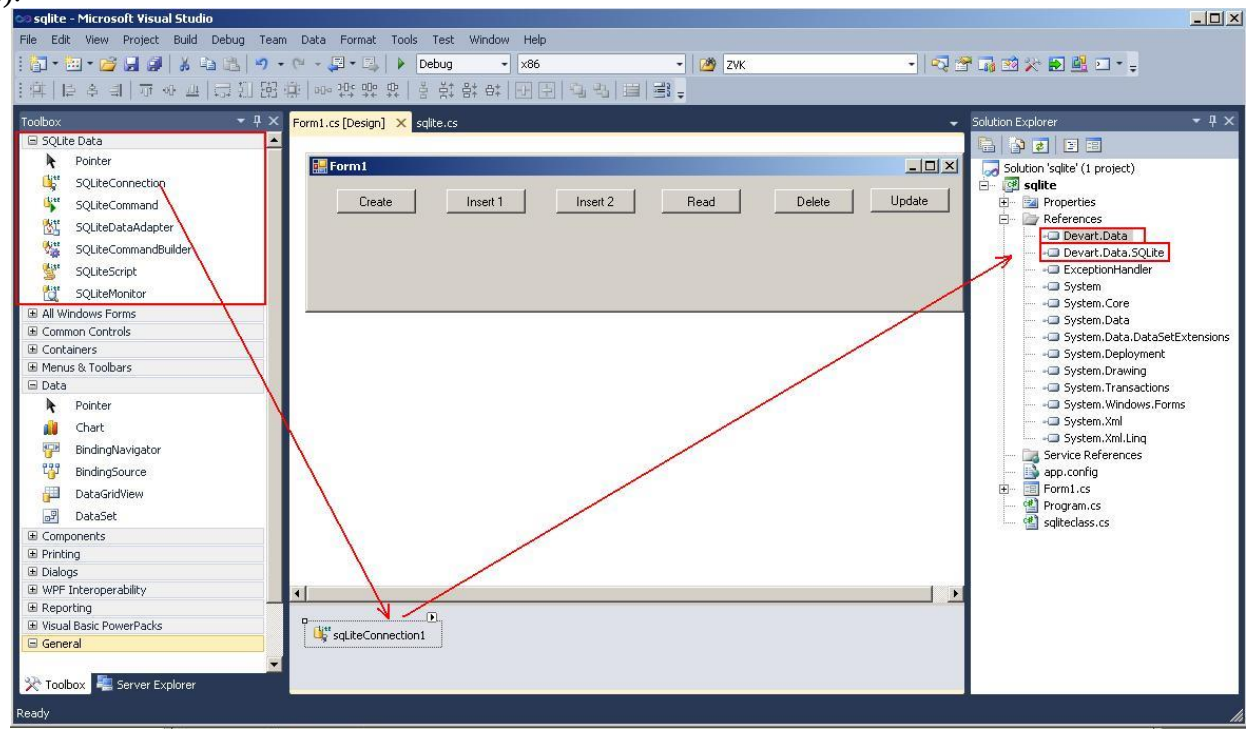


Рис. Встроенный провайдер SQLite

Перенесем в проект любой из контролов со вкладки "SQLiteData", например контрол "SQLiteConnection". Убедимся, что в "Solution Explorer", в закладке "References", появились новые библиотеки (Рис.):

```
Devart.Data
Devart.Data.SQLite
```

Добавим в класс "sqlclass.cs" пространства имен:

```
using Devart.Data;
using Devart.Data.SQLite;
```

Определимся на данном этапе, будем мы использовать стандартные контролы или нет. Но без стандартных контролов можно построить более гибкие приложения.

Удалим из проекта контрол "`sqLiteConnection1`" (на закладке "References" в Solution Explorer, при внесении контрола в проект библиотеки были добавлены, при удалении - они остаются). Значит, мы можем продолжить работу над проектом.

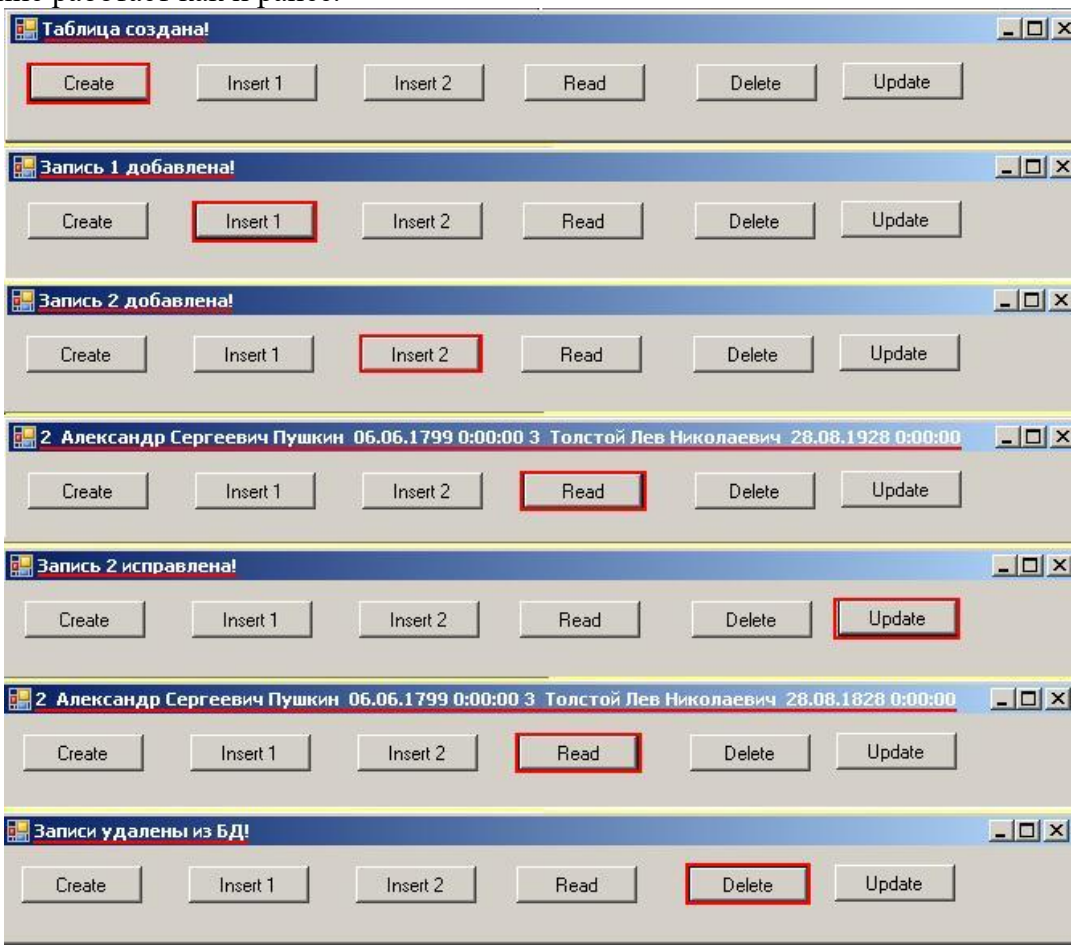
Изменим немного наш класс "sqliteclass.cs":

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlTypes;
using System.Data;
using Devart.Data;
using Devart.Data.SQLite;
```

```
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)
     {
       SQLiteConnection.CreateFile(FileData);
       con.ConnectionString = @"Data Source=" + FileData;
     }
     else
     {
       con.ConnectionString = @"Data Source=" + FileData;
     }

     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;

    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
 }
}
```

Вот и все изменения в проекте решения. Выполним решение и убедимся, что приложение работает как и ранее.



Работа с ошибками

Особых средств для обработки ошибок провайдеры SQLite не содержат.

Единственное, что можно, это объявить в классе "`sqlliteclass`" переменные и функции возврата ошибок:

```
private string sLasterror = string.Empty;
```

```
private string sErrors = string.Empty;
public  string sGetErrors()
{
 return sErrors;
}
public  string sGeLastError()
{
 return sLasterror;
}
```

И соответственно в функциях класса добавить код:

```
try
{
 .......
}catch(Excaption ex)
{
 sLasterror =ex.Message;
 sErrors+=ex.Message+" ";
}
```

Анализ ошибки выполнять уже в коде обращения к классу, после вызова данных функций.

Можно в `try-catch` предусмотреть и ведение журнала ошибок, добавив, например, такую функцию:

```
File.AppendAllText("logerrors.txt", DateTime.Now.ToString() + " " +
                   Path.GetFileName(FileData) + " " + sSql + " " +
                   ex.Message+" "+Environment.NewLine,
System.Text.Encoding.UTF8);
```

ЧАСТЬ 2.

Работа с Blob типами (хранение изображений)

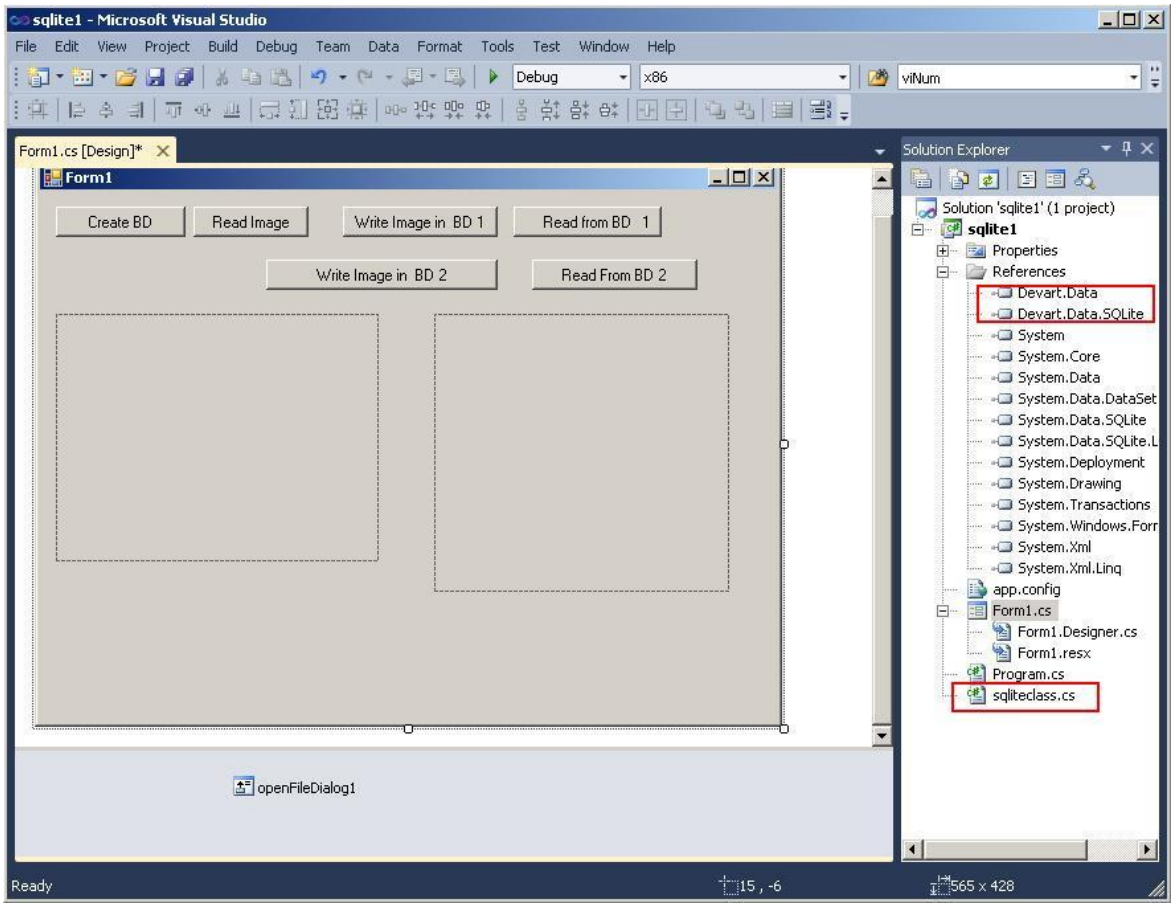
Решение приложения

Создадим новый проект решения с именем "sqlite1". Будем использовать шаги создания проекта, по методике, показанной выше.

Поместим на форму 6 контролов `Button`, 2 контрола `PictureBox` и один контрол `OpenFileDialog`. Также создадим класс "`sqliteclass`" (см. рис. ниже.). Класс можно взять из предыдущего проекта, скопировав его в наш проект, добавив в решение (`Solution Explorer`, узел "`sqlite1`", контекстное меню узла, "`Add`", "`Existing Item`") и переименовав пространство имен:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Devart.Data;
using Devart.Data.SQLite;
using System.IO;
using System.Data.SqlTypes;
using System.Data;

namespace sqlite1
{
    class sqliteclass
    {
```

Обратите внимание, что для работы с SQLite используется встроенный в Visual Studio провайдер и подход, описанный ранее. Но это нет принципиально.



Проект решения

Поставим задачу: необходимо хранить файлы фотографий в некоторой базе SQLite, причем, ничего, кроме фотографий, в этой базе храниться не будет (только номер фотографии и само фото).

Кнопка 1 - создание базы данных.

```
//Необходимые пространства имен в классе формы
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SQLite;
using System.Data.Common;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Drawing.Imaging;

namespace sqlite1
{
 public partial class Form1 : Form
 {
  //Необходимые переменные для класса формы
  private sqliteclass mydb = null;
  private string sCurDir = string.Empty;
  private string sPath = string.Empty;
  private string sSql = string.Empty;
```

```
  private int viNum = 0;
  public byte[] data = null;
  //Обработчик нажатия кнопки 1 - создание базы данных:
  private void button1_Click(object sender, EventArgs e)
  {
    File.Delete(sPath);
    mydb = new sqliteclass();
    sSql = "create table myphoto (id INTEGER PRIMARY KEY NOT NULL,photos
blob)";
    //Используем функцию класса, рассмотренную выше
    mydb.iExecuteNonQuery(sPath, sSql, 0);
    mydb = null;
    //Проверку работоспособности мы делали, повторять не будем
  }
```

Функция iExecuteNonQuery используется из класса "sqliteclass" прошлого решения. Можно скопировать все функции в класс "sqliteclass" из прошлого решения, если не воспользовались копированием класса.

При создании таблицы базы данных, не была включена опция "AUTOINCREMENT", так как метод записи, который будет рассмотрен далее некорректно работает с этим предложением.

Выбираем фотографию, которую будем записывать в БД и помещаем ее в pictureBox1.

```
private void button2_Click(object sender, EventArgs e)
{
 openFileDialog1.InitialDirectory = @"C:\";
 openFileDialog1.Filter = "All Embroidery
Files|*.bmp;*.gif;*.jpeg;*.jpg;" +
      "*.fif;*.fiff;*.png;*.wmf;*.emf" +
      "|Windows Bitmap (*.bmp)|*.bmp" +
      "|JPEG File Interchange Format (*.jpg)|*.jpg;*.jpeg" +
      "|Graphics Interchange Format (*.gif)|*.gif" +
      "|Portable Network Graphics (*.png)|*.png" +
      "|Tag Embroidery File Format (*.tif)|*.tif;*.tiff";
      //openFileDialog1.Filter += "|All Files (*.*)|*.*";
  if (openFileDialog1.ShowDialog() == DialogResult.OK)
  {
    Image image = Image.FromFile(openFileDialog1.FileName);
    MemoryStream memoryStream = new MemoryStream();
    //Здесь можно выбрать формат хранения
    image.Save(memoryStream, ImageFormat.Jpeg);
    data = memoryStream.ToArray();
    pictureBox1.Image = image;

 }
}
```

Помимо отображения записываемой фотографии байты файла фото были помещены в байтовый массив "`data`", используя `MemoryStream` и его метод `ToArray()`.

Возможности использования SQLiteDataAdapter для записи BLOB данных

Введем в наш класс две функции.

Задача первой будет "изучить" схему таблицы базы данных и возвратить ее в DataTable.

```
#region fReagSchema - Чтение схемы БД
public DataTable dtReagSchema(string FileData, string sSql)
{
 bool f = true;
 DataTable mydatatable =  new DataTable();
 SQLiteConnection con = new SQLiteConnection();
 try
 {
   con.ConnectionString = @"Data Source=" + FileData;
   con.Open();
   myDataAdapter = new SQLiteDataAdapter(sSql, con);
   mycommandbuilder = new SQLiteCommandBuilder(myDataAdapter);
   myDataAdapter.FillSchema(mydatatable, SchemaType.Source);
  }
  catch (Exception ex)
  {
    mydatatable = null;
  }
  if (myDataAdapter == null || mycommandbuilder == null)
  {
   mydatatable = null;
  }
  return mydatatable;
}
#endregion
Задача второй функции будет непосредственно внесение изменения в базу
данных:
#region Изменение базы данных
public int iWriteBlob(string FileData, DataTable mydatatable)
{
 int n = 0;
 using (SQLiteConnection con = new SQLiteConnection())
 {
  try
  {
      n = myDataAdapter.Update(mydatatable);
  }
  catch (Exception ex)
  {
      n = 0;
  }
 }
 return n;
}
#endregion
```

Эти две функции используются комплексно. При обращении к первой функции получают объект `DataTable`, содержащий схему базы данных.

Можно добавить строку таблицы базы данных в объект, как будто мы ее добавили в таблицу базы данных. Далее, после добавления строки, вызывается вторая функция, которая используя объекты `SQLiteDataAdapter` и `SQLiteCommandBuilder`, проводит изменение в таблице базы данных.

На стороне клиента заполним строку `DataTable`:

```
private void button3_Click(object sender, EventArgs e)
{
 if (data == null)
 {
     Text = "Не выбрано фото для записи в БД";
     return;
 }

 int id = 0;
 mydb = new sqliteclass();
 sSql = "Select max(id) from myphoto";
 object obj = mydb.oExecuteScalar(sPath, sSql);
 if (!string.IsNullOrEmpty(obj.ToString()))
 {
     id = Convert.ToInt32(obj) + 1;
 }
 else
 {
  id = 1;
 }
 //SQL предложение для получения схемы таблицы
 sSql = "Select * from myphoto";
 //Чтение схемы
 DataTable dt = mydb.dtReagSchema(sPath, sSql);
 //Добавление строки с данными
 DataRow datarow = dt.NewRow();
 datarow["id"] = id;
 datarow["photos"] = data;
 dt.Rows.Add(datarow);
 //Сохранение изменения
 int n = mydb.iWriteBlob(sPath, dt);
 if (n == 0)
 {
  Text = "Фото в БД не записано!";
 }
 else
 {
  Text = "Фото записано в БД!";
 }
 mydb = null;
}
```

Возможности использования SQL предложений с параметрами для записи BLOB данных

Приведенная ниже функция, которая была добавлена в класс позволяет записать в поле `Blob` байты изображения.

Это делается через параметры, а не через преобразование массива потому, что байты изображения могут содержать нулевой символ, что воспринимается как конец строки и единственная возможность непосредственно указать адаптеру о необходимости воспринимать то, что передается в массиве байт, как сплошной поток бит - "`SQLiteType.Blob`".

```
 #region ExecuteNonQuery
 public int iExecuteNonQueryBlob(string FileData, string sSql, string s0,
                                                 Int32 number, string s1,
byte[] data)
```

```
{
 int n = 0;
 try
 {
  using (SQLiteConnection con = new SQLiteConnection())
  {
   con.ConnectionString = @"Data Source=" + FileData;
   con.Open();
   using (SQLiteCommand sqlCommand = con.CreateCommand())
   {
    sqlCommand.CommandText = sSql;
    sqlCommand.Parameters.Add("@" + s0, SQLiteType.Int32).Value = number;
    sqlCommand.Parameters.Add("@" + s1, SQLiteType.Blob,
data.Length).Value = data;
    n = sqlCommand.ExecuteNonQuery();
   }
   con.Close();
  }
 }
 catch (Exception ex)
 {
  n = 0;
 }
 return n;
}
#endregion
```

Соответственно, использование данной функции, будет выглядеть примерно так:

```
private void button5_Click(object sender, EventArgs e)
{
    if (data == null)
    {
        Text = "Не выбрано фото для записи в БД";
        return;
    }

    int id = 0;
    mydb = new sqliteclass();
    sSql = "Select max(id) from myphoto";
    object obj = mydb.oExecuteScalar(sPath, sSql);
    if (!string.IsNullOrEmpty(obj.ToString()))
    {
        id = Convert.ToInt32(obj) + 1;
    }
    else
    {
        id = 1;
    }
    Text = id.ToString();
    sSql="Insert into myphoto (id,photos) values(@id,@photos)";
    int n = mydb.iExecuteNonQueryBlob(sPath, sSql, "id", id, "photos",
data);
    if (n == 0)
    {
        Text = "Фото не записано в БД";
    }
    else
    {
        Text = "Фото внесено в БД";
        data = null;
    }
    mydb = null;

}
```

Чтение Blob данных с использованием SQLiteDataAdapter

Рассмотрим два способа чтения `Blob` поля из базы данных, с использованием `SQLiteDataAdapter` и `SQLiteDataReader`

Это наиболее простой способ, основанный на том, что прочитанные в `DataTable` столбцы таблицы мы можем интерпретировать как любой тип данных, используя средства языка C#:

```
#region Чтение 1 поля BLOB
public byte[] rgbtReadBlob(string FileData, string sSql, string field)
{
 byte[] buffer = null;
 DataTable datatable = new DataTable();
 using (SQLiteConnection con = new SQLiteConnection())
 {
  try
  {
   con.ConnectionString = @"Data Source=" + FileData;
   con.Open();
   SQLiteDataAdapter myDataAdapter = new SQLiteDataAdapter(sSql, con);
   SQLiteCommandBuilder commandbuilder = new
SQLiteCommandBuilder(myDataAdapter);
   myDataAdapter.Fill(datatable);
   DataRow[] datarows = datatable.Select();
   DataRow datarow = datarows[0];
   //Получаем быйты фото из БД в массив
   buffer = (byte[])datarow[field];
  }
  catch (Exception ex)
  {
   buffer = null;
  }
 }
 return buffer;
}
#endregion
```

Чтение данных с использованием SQLiteDataReader

Метод основан на возможности `DataReader` читать данные столбца как массив бит. Функция `GetBytes` последовательно переписывает биты в массив байт, что позволяет получить байтовый образ сохраненного в памяти изображения.

```
#region Execute SQLiteDataReader
 public byte[] rgbytedreaderExecute(string FileData, string sSql)
 {
     byte[] data = null;
     SQLiteDataReader dr = null;
     try
```

```
     {
         using (SQLiteConnection con = new SQLiteConnection())
         {
             con.ConnectionString = @"Data Source=" + FileData;
             con.Open();
             using (SQLiteCommand sqlCommand = con.CreateCommand())
             {
                 sqlCommand.CommandText = sSql;
                 dr = sqlCommand.ExecuteReader();
             }
             dr.Read();
             data = GetBytes(dr);


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

         data = null;
     }
     return data;

 }
 static byte[] GetBytes(SQLiteDataReader reader)
 {
     //const int CHUNK_SIZE = 2 * 1024;
     byte[] bDate = new byte[1024];
     long lRead = 0;
     long lOffset = 0;
     using (MemoryStream memorystream = new MemoryStream())
     {
         while ((lRead = reader.GetBytes(0, lOffset, bDate, 0,
bDate.Length)) > 0)
         {
             byte[] bRead = new byte[lRead];
             Buffer.BlockCopy(bDate, 0, bRead, 0, (int)lRead);
             memorystream.Write(bRead, 0, bRead.Length); lOffset +=
lRead;
         }
         return memorystream.ToArray(); } }
#endregion
```

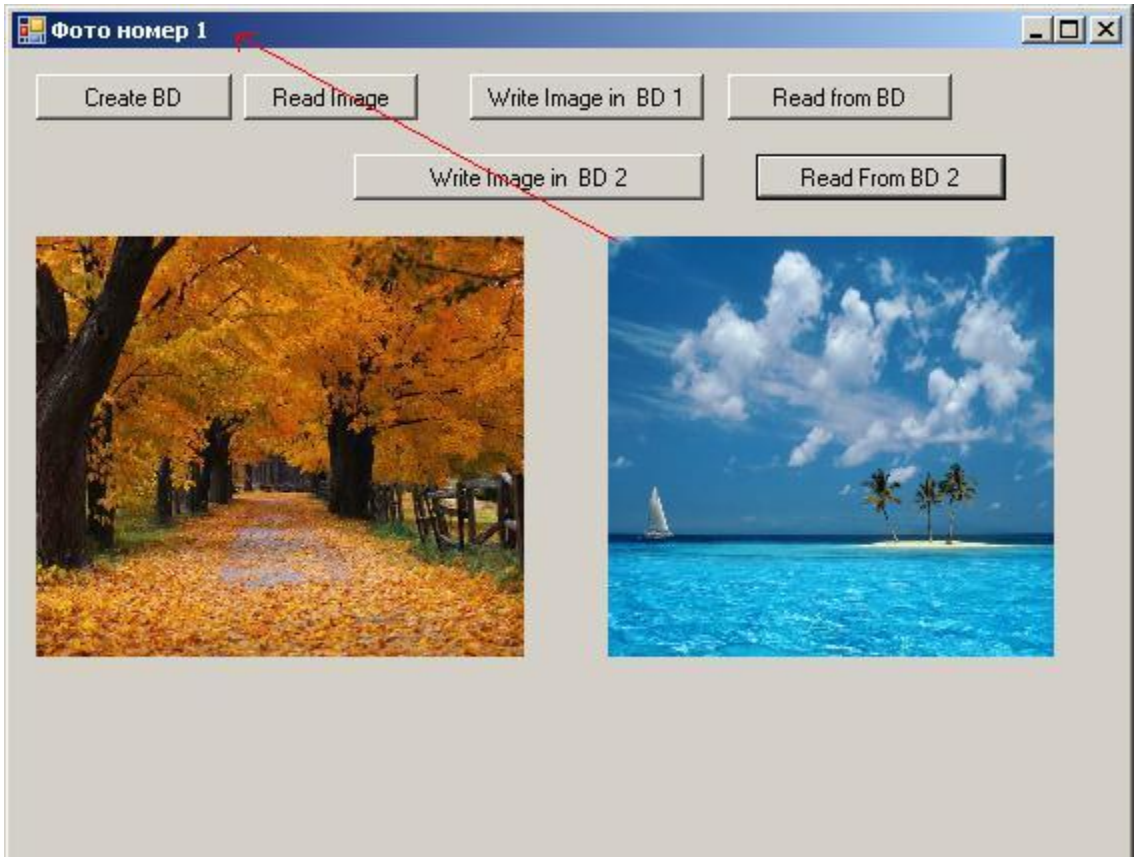
Функция ExecuteScalar

Функция oExecuteScalar, возвращает единичное значение столбца или агрегат из БД.

```
#region ExecuteScalar
public object oExecuteScalar(string FileData, string sSql)
{
 object obj = null;
 try
 {
  using (SQLiteConnection con = new SQLiteConnection())
  {
    con.ConnectionString = @"Data Source=" + FileData;
    con.Open();
    using (SQLiteCommand sqlCommand = con.CreateCommand())
    {
```

```
     sqlCommand.CommandText = sSql;
     obj = sqlCommand.ExecuteScalar();
    }
    con.Close();
  }
 }
 catch (Exception ex)
 {

  obj = null;
 }
 return obj;
}
#endregion
```

Выполнение решения показано на Рис. ниже.



Работа с Blob данными в SQLite

Задание на лабораторную работу

1. Реализовать приложение, позволяющее добавлять, извлекать и удалять изображения из базы данных SQLite. Для хранения изображений необходимо использовать тип BLOB.
2. Модифицировать приложение и БД таким образом, чтобы в ней хранились данные типов : INTEGER, REAL, TEXT и BLOB. При этом также реализовать хранение даты и значений да/нет.
3. В приложении реализовать Обработчик ошибок, который обрабатывает, например, ошибку подключения к БД. Реализовать логирование ошибок.
4. Подключение/отключение от СУБД реализовать в отдельном классе.