УДК 004.8

ИМПОРТ ДАННЫХ ИЗ DBF ФАЙЛА В SQL SERVER

Саакян И.Э., Губанов А.И.

Несмотря на то, что формат хранения данных ***dbf*** давно считается устаревшим, конвертации данных из такого типа файлов остается насущной задачей судя по количеству вопросов в Интернете [1,2]. В частности, сталкиваешся с такой задачей при необходимости импорта в SQL базу данных распространенной систесы КЛАДР (Классификатор адресов России, введен в действие с 01.12.2005 приказом ФНС России от 17.11.2005 № САЭ-3-13/594@). В давние времена ***Visual FoxPro 6*** и ***SQL Server 7.0*** это не составляло проблемы, но с тех пор многое изменилось. С выходом ***SQL Server 2005*** в ***MSDN*** появилась информация, что мастер импорта и экспорта в ***SQL Server*** не поддерживает импорт и экспорт ***dBASE***-файлов и других ***DBF***-файлов. В качестве решения рекомендовано использовать ***SQL Server Integration Services*** или промежуточный импорт в ***Access*** или ***Excel***. Такая же ситуация формально сохраняется по сей день, включая ***SQL Server 2012***. Это не всегда удобно, потому что, помимо ***SQL Server***, требует дополнительной установки ***MS Office***, а средства разработки ***ETL***-пакетов не входят в состав бесплатной редакции ***SQL Server Express***.

Для решения такой задачи можно, например, читать данные из исходной таблицы построчно и вставлять командой ***INSERT***. Решение в лоб, далеко не самое эффективное. Можно преобразовать их в ***XML*** и передать на сервер в таком виде целиком. Такое решение потребует, кроме всего прочего, написания хранимой процедуры.

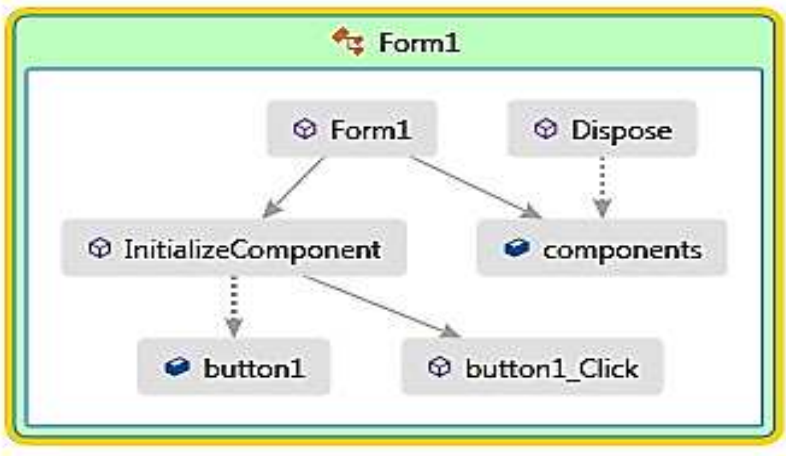

Рисунок 1 - Карта кода основной формы

С другой стороны, такая задача весьма распространена и было бы странным, если бы уже не было готовых решений. Не будем изобретать велосипед и воспользуемся

стандартным классом ***SqlBulkCopy***. Он как раз и предназначен для массовой загрузки на ***SQL Server*** данных из различных источников.

Проект был осуществлен при помощи Visual Studio 2012 (язык C#). Исходный код для основной формы:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using DbfHelper;

namespace Dbf
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            SqlConnection connection = new SqlConnection("Data
Source=...;Initial Catalog=KLADR;Integrated Security=True;Connection
Timeout=420000");
            using (OpenFileDialog dlg = new OpenFileDialog())
            {
                if (dlg.ShowDialog() == DialogResult.OK)
                {
                    BaseLocalSql localSql = new
BaseLocalSql(dlg.FileName);
                    BaseDbSql dbSql = new BaseDbSql();
                    var dt = localSql.SelectTable();
                    if(dt != null)

dbSql.Insert(connection.ConnectionString,dt);
                    MessageBox.Show("OK");}
            }

        }

    }
}
```

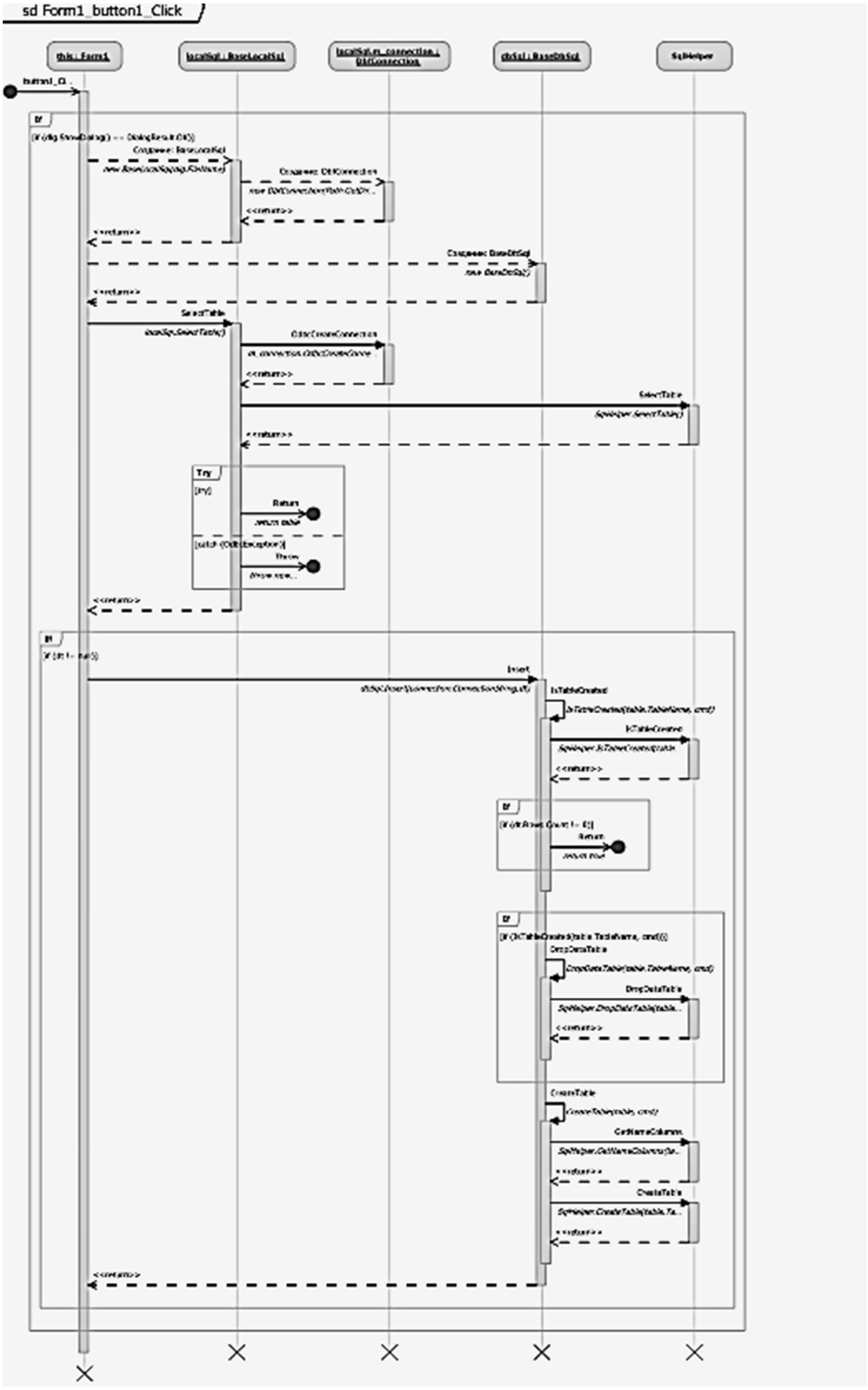

Рисунок 2 - Схема последовательностей для `BaseLocalSql`

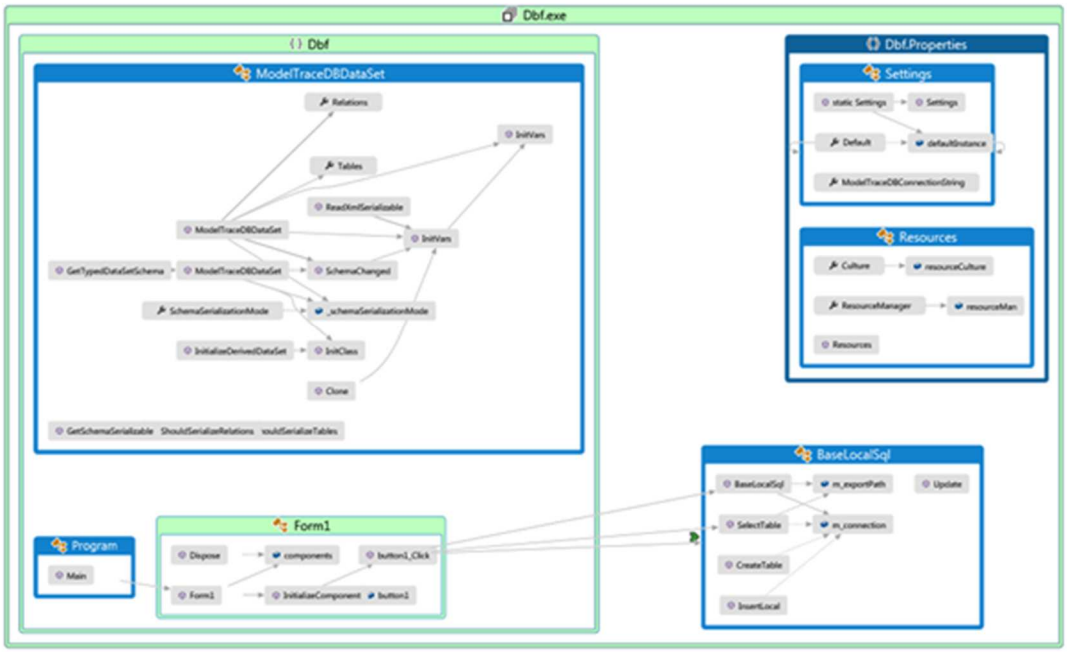

Рисунок 3 - Карта кода всех модулей

```
using System;
using System.Data.Odbc;
using System.Data.SqlClient;
using System.IO;
using DbfHelper.Interfaces;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Data;
using System.Runtime.InteropServices;
using System.Data.OleDb;

namespace DbfHelper
{
    public class BaseLocalSql : IBaseLocalSql
    {
        private string m_exportPath;
        private readonly DbfConnection m_connection;
        public BaseLocalSql(string exportPath)
        {
            m_exportPath = exportPath;
            m_connection = new
DbfConnection(Path.GetDirectoryName(m_exportPath));
        }

        public void CreateTable(string tableName, DbfData[] fields)
        {
```

```
            using (var connection =
m_connection.OdbcCreateConnection())
            {
                connection.Open();
                using (var comCreate = new
OdbcCommand(SqlHelper.CreateLocalTable(tableName, fields),
connection))
                {
                    try
                    {
                        comCreate.ExecuteNonQuery();
                    }
                    catch (OdbcException ex)
                    {
                        throw new Exception(ex.Message);
                    }
                }
                connection.Close();
            }
        }

        public DataTable SelectTable()
        {
            DataTable table = new DataTable();
            using (var connection =
m_connection.OdbcCreateConnection())
            {
                connection.Open();
                using (var comCreate = new
OdbcCommand(SqlHelper.SelectTable() + m_exportPath, connection))
                {
                    try
                    {
                        OdbcDataAdapter myReader =  new
OdbcDataAdapter(comCreate);
                        myReader.Fill(table);
                        table.TableName =
Path.GetFileNameWithoutExtension(m_exportPath);
                        connection.Close();
                        return table;
                    }
                    catch (OdbcException ex)
                    {
                        throw new Exception(ex.Message);
                    }
                }

            }
        }
```

```
        public void InsertLocal(string tableName, params DbfData[]
dates)
        {
            bool success = false;
            try
            {
                var connection =
m_connection.OdbcCreateConnection();
                connection.Open();

                var comInsert = new
OdbcCommand(SqlHelper.InsertLocal(tableName, dates), connection);
                comInsert.ExecuteNonQuery();
                comInsert.Dispose();
                connection.Close();
            }
            catch (OdbcException ex)
            {
                throw new Exception(ex.Message);
            }
        }

    }
}
```

Сначала создаются и открываются подключения к файловой базе и ***SQL Server***.

```
SqlBulkCopy bulkCopy =
    new SqlBulkCopy
       (connection,
        SqlBulkCopyOptions.TableLock |
        SqlBulkCopyOptions.FireTriggers |
        SqlBulkCopyOptions.UseInternalTransaction,
        null);
```

создаем обьект, передав ему подключение к серверу.

Однако в этом методе есть пара неприятных моментов.

SqlBulkCopy работает как транзакция, либо полностью выполняя копирование, либо, если возникла какая-то ошибка, полностью его не выполняя. При этом в случае возникновения ошибок он не генерирует никаких исключений. Т.е. разобраться в чем проблема может оказаться затруднительно. Но можно. Основная масса ошибок (если не все) будет возникать при вставке в таблицу, в том случае если на данные в столбцах наложены ограничения. Например, требования уникальности ключа. Тут 2 варианта борьбы:

а) простейший - дать возможность писать в таблицу все подряд. А с корректностью разбираться после вставки данных;

б) готовить данные перед подачей их на копирование. Тут сложнее, поскольку у самого ***SqlBulkCopy*** никаких возможностей проверки данных нет. Для реализации этого варианта придется либо перебирать данные, загруженные в ***DataRow*** и ***DataTable***, либо писать собственный класс, наследующий ***IDataReader*** и в нем выполнять проверку и исправление ошибочных значений.

Неопределенности при копировании больших объёмов. Обойти это можно установив количество строк в пакете, который будет отсылаться для вставки на сервер:

bulkCopy.BatchSize = 100000;

По умолчанию все данные из исходной таблицы передаются одним пакетом. Задавая размер нужно помнить, что чем меньше пакеты, тем дольше копирование. Например, таблица с тремя миллионами записей при размере пакета в 100 000 строк примерно загружается около минуты, а при размере пакета в 10 записей - значительно больше часа.

Помнить о памяти, для буферизации 300-т мегабайтной таблицы процесс забирает себе около 1,5 ГБайт памяти, можно получить ошибки системы «out of memory». Чтобы этого не было, придется отказаться от буферизации. В этом случае используем ридер. Все предельно просто, изменения минимальны:

```
OleDbDataReader read = comm.ExecuteReader();
bulkCopy.WriteToServer(read);
```

И не забывать о времени подключения к серверу параметр ***Connection Timeout*** в строке подключения.

Список информационных источников

[1] Саакян И.Э., Шушанский К.П. Учет и верификация мобильных пользователей корпоративных сетей // Автоматизация и управление в технических системах. – 2012. – № 2; URL: auts.esrae.ru/2-38 (дата обращения: 21.10.2013).

[2] Остроух, А.В. Информационные технологии в научной и производственной деятельности / [ред. А.В. Остроух] - М: ООО "Техполиграфцентр", 2011. - 240 с. - ISBN 978-5-94385-056-1.