МИНИСТЕРСТВО ВОЗДУШНОГО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ

Кафедра вычислительных машин, комплексов, систем и сетей

Методические указания по курсовому проектированию по дисциплине “Технология программирования”

МОСКВА 2009

Содержание

1. Создание нового проекта

Запустите Visual Studio , выберите меню File\ New project
В появившемся на экране диалоговом окне New Project создайте новый проект –Windows-приложение, на языке Visual C# , как это показано на рис.1.
Присвойте имя проекту.

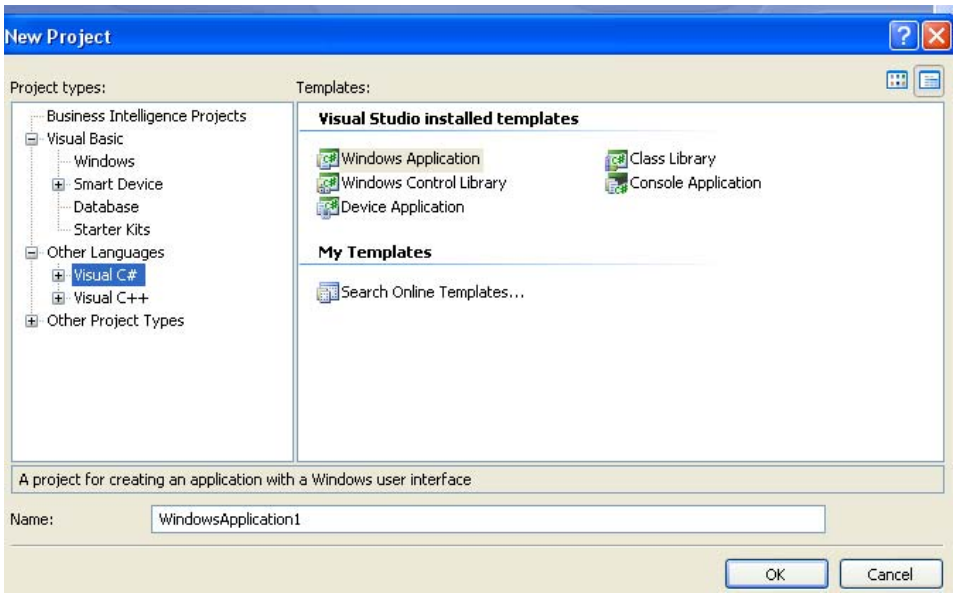


Рис.1

2. Создание базы данных на SQL сервере.

Базу данных на SQL сервере можно создавать двумя способами:

- В среде Visual Studio
- В среде самого SQL сервера (если к нему имеется доступ)

Создадим базу данных с помощью среды Visual Studio, для этого откройте окно **Server Explorer** (команда View\Server Explorer)

Окно **Server Explorer** – очень полезный инструмент, который, как видно из его названия, позволяет просматривать серверы. В нем имеются функции, специально предназначенные для взаимодействия с БД. С помощью этого окна можно получить доступ к любой схеме из БД – таблице, представлению или хранимой процедуре.

В иерархической структуре элементов окна Server Explorer щелкните правой кнопкой мыши меню **Data Connections** и выберете пункт **Create New SQL Server Database** (рис. 2)

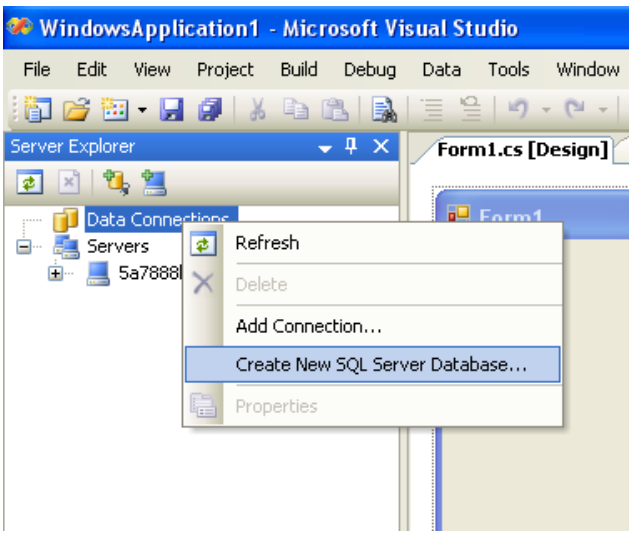


Рис. 2

В окне, представленном на рис. 3, необходимо задать имя сервера (*Server name*), параметры аутентификации пользователя при подключении к серверу и имя создаваемой базы данных (*New database name*)

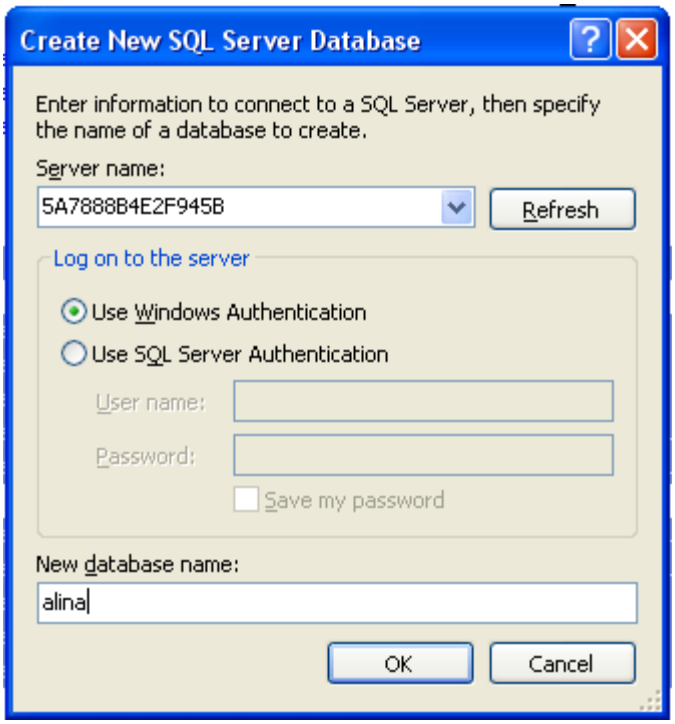


Рис. 3

После нажатия кнопки ОК и указания всех параметров создания новой БД в окне подключения к серверу Server Explorer появится имя созданной базы данных.
Из структуры объектов созданной БД щелкните правой кнопкой мыши **Tables** и выберете пункт меню **Add New table** (рис. 4)

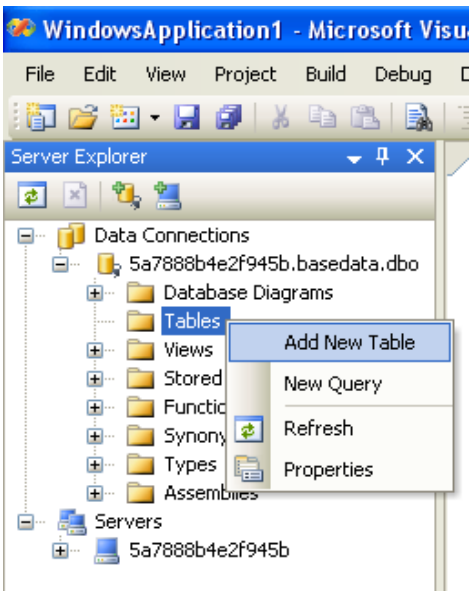


Рис. 4

Заполните структуру таблицы: имя столбца (Column name) , тип данных (data type), Allow Nulls (флаг, позволяющий задать полю значение *NULL*) – (рис. 5)
Сохраните таблицу (Save).
В окне на рис. 6 задайте имя таблицы, после чего она появится в списке Tables в окне Server Explorer (рис. 7)

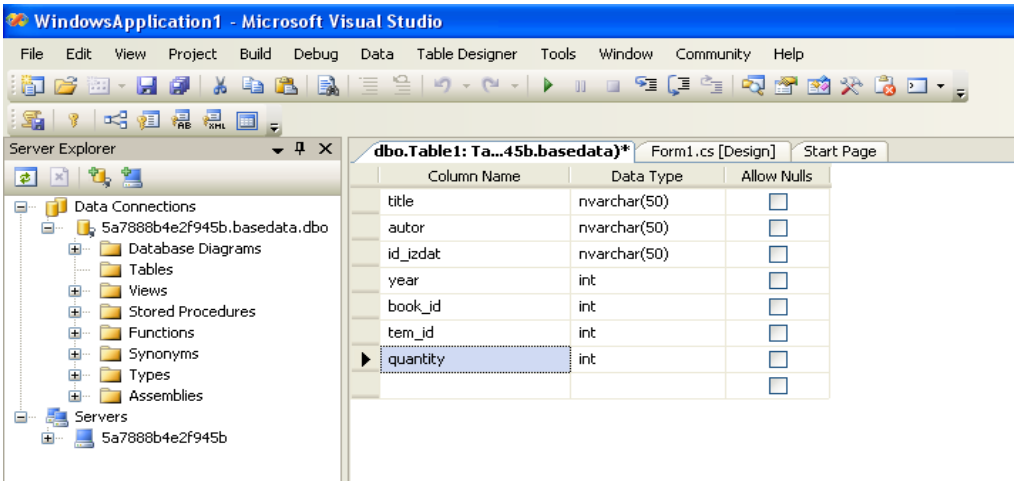


Рис. 5

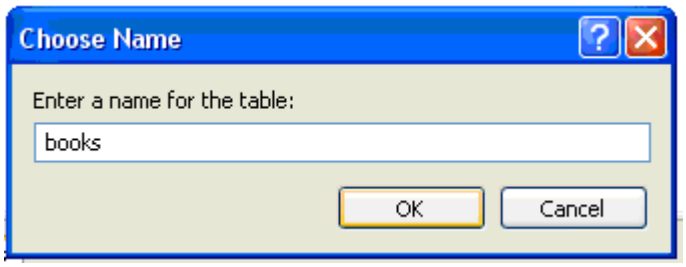


Рис. 6

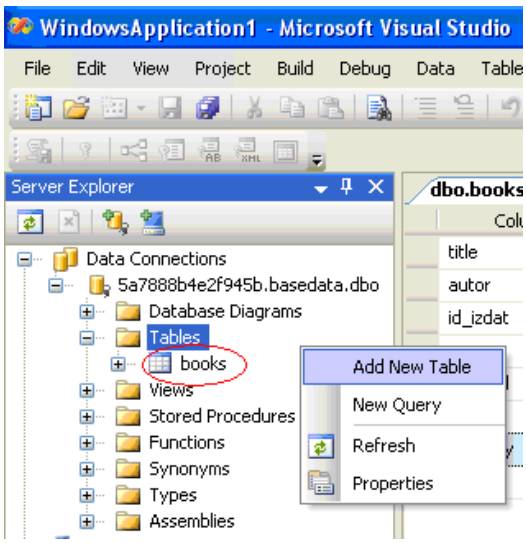


Рис. 7

Заполнить таблицу данными, а также просмотреть содержимое таблицы можно, щелкнув на имени созданной таблицы правой кнопкой мыши, и выбрать пункт меню **Show Table Data** (Рис. 8)

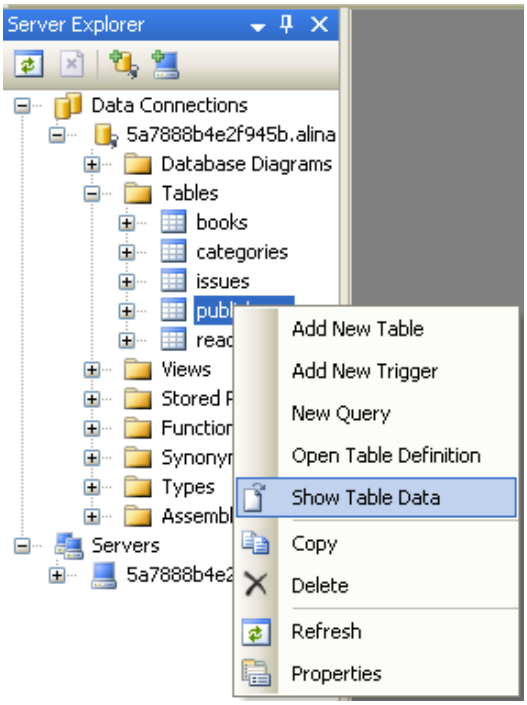


Рис. 8

3. Подключение к базе данных

После создания всех таблиц БД создадим строку подключения к ней *ConnectionString* и объект класса *DataSet* со строгим контролем типов.

Для этого выберем пункт меню **Data- Add New Data Source** (рис. 9)

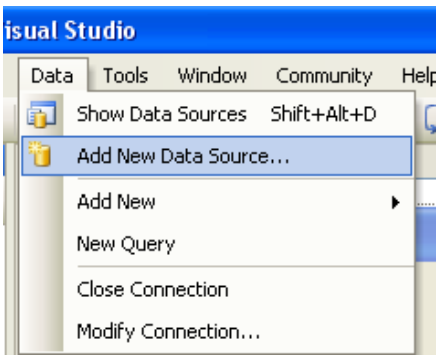


Рис. 9

В окне мастера **Data Source Configuration Wizard** выберем источник данных **Database**. Далее **Next** (рис. 10)

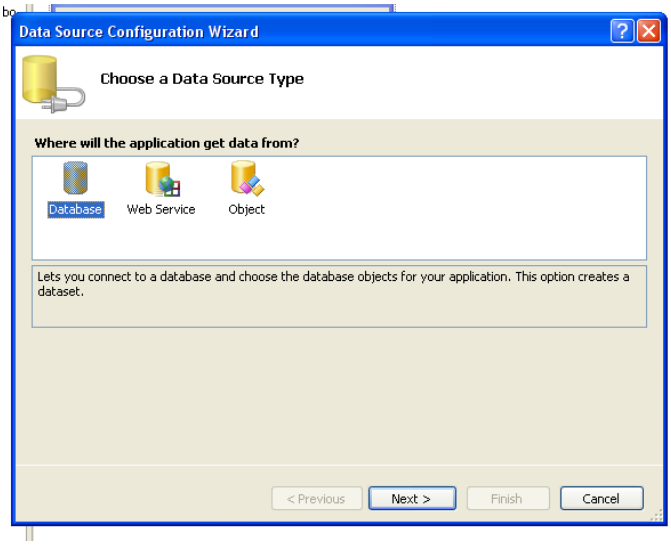


Рис 10

Окно мастера, представленное на рис. 11 приглашает ввести данные для своего соединения. Можно выбрать доступные в окне Server Explorer соединения из списка, либо создать новое соединение, щелкнув кнопку New Connection.

В нашем случае мы уже создали подключение к базе данных при ее создании на сервере, и оно отображается в списке Server Explorer, поэтому в раскрывающемся списке окна мастера на рис. 11 выбираем его.

В мастере есть возможность контролировать сохранение информации в строке подключения. Рядом с надписью Connection string можно щелкнуть кнопку +/-, чтобы вывести на экран строку подключения.

Для создания этой строки используются классы ADO.NET версии 2.0 *SqlConnection* и *SqlConnectionStringBilder*, которые входят в набор классов, разработанных специально для взаимодействия с SQL Server. Они расположены в пространстве имен *System.Data.SqlClient*, а указанный набор классов еще называют поставщиком данных SQL Client .NET Data Provider, или *SqlClient*.

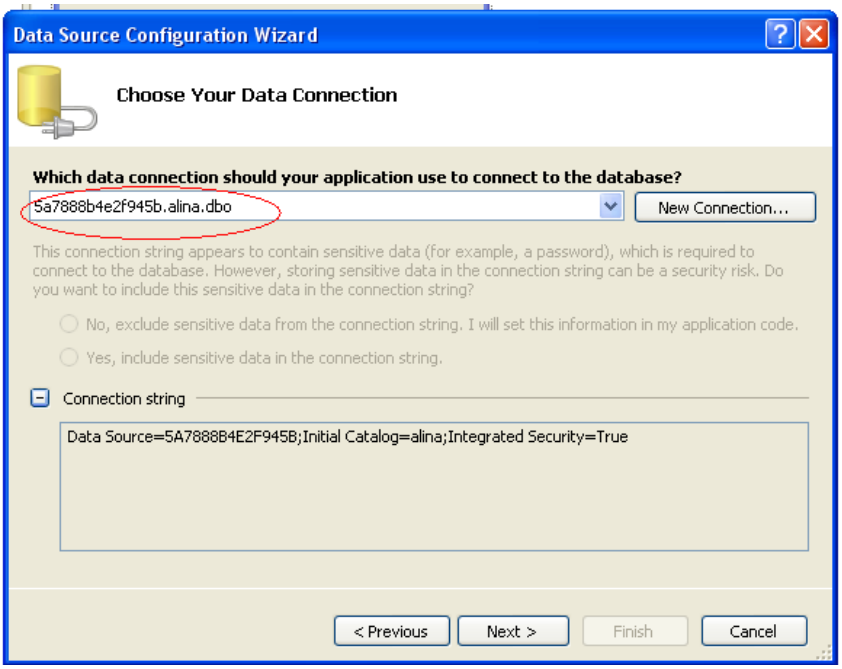


Рис. 11

4. Сохранение строки подключения

Щелкнув на кнопку Next, мастер Data Source Configuration отобразит диалоговое окно, показанное на рис. 12, в котором он поинтересуется, сохранять ли строку подключения в файле конфигурации приложения. И хотя в данном подходе есть много за и против, одно из ключевых его преимуществ состоит в том, что он представляет собой простой способ отделить информацию о подключении от остальной части кода.

Если в диалоговом окне установить флажок **Yes, save the connection string as,** то среда Visual Studio сохранит строку подключения и имя, которое вы введете, в файле конфигурации приложения для выполняемого проекта. Кроме того, она добавит логические операторы в код проекта, чтобы получать и использовать эту строку подключения для взаимодействия с БД.

Файлы конфигурации приложений — стандартное место хранения информации о строке подключения.

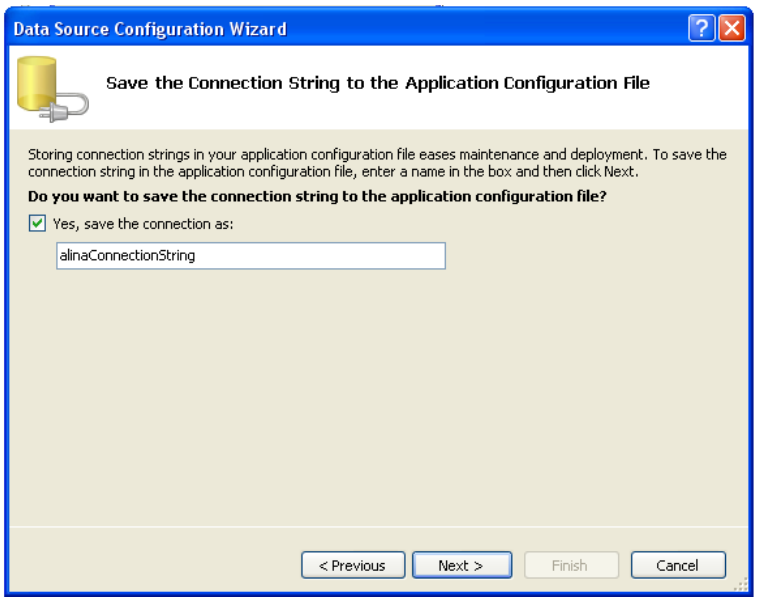


Рис. 12

5. Создание класса Dataset со строгим контролем типов.

После того как вы определите, нужно ли хранить информацию о строке подключения в файле конфигурации приложения для вашего проекта, мастер Data Source Configuration выведет в показанном на рис.13 диалоговом окне все доступные в БД таблицы, представления, хранимые процедуры и функции, и вы сможете указать, какие структуры хотели бы включить в новый источник данных (класс *DataSet*).
Можно выбирать также отдельные столбцы в таблицах и представления.
В нижней части диалогового окна имеется поле, которое позволяет указать имя для класса *DataSet.* По умолчанию Visual Studio использует имя подключенной в данный момент БД и добавляет к этому имени слово DataSet.

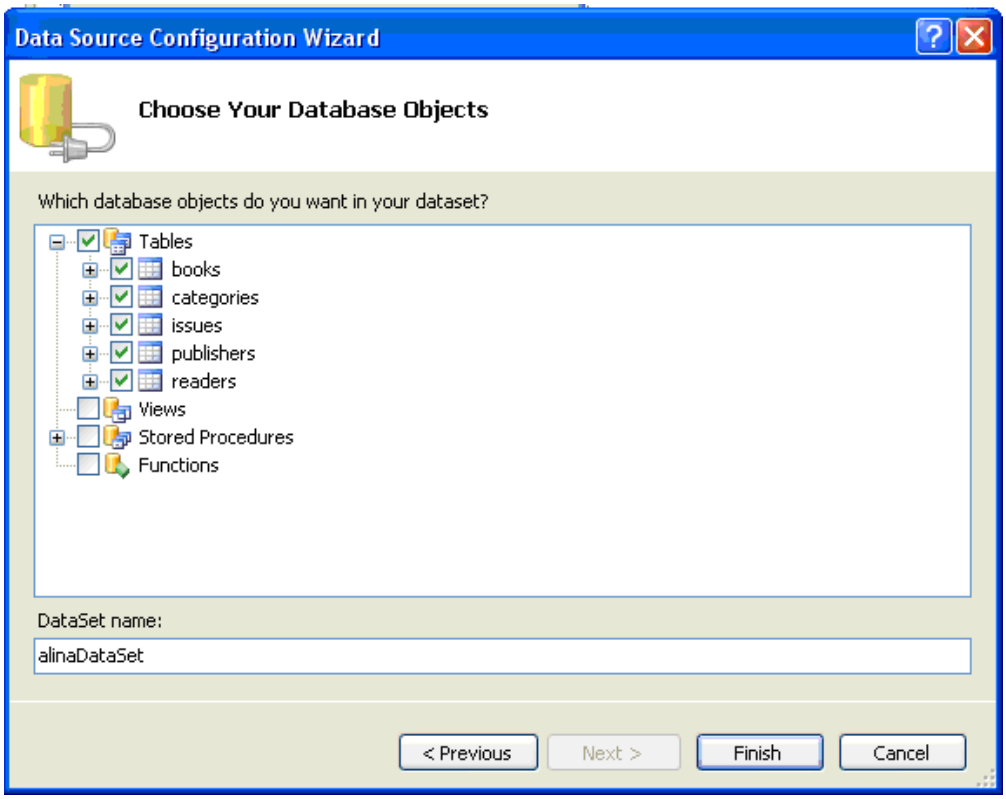


Рис. 13

После нажатия на кнопку Finish работа мастера Data Source Configuration будет завершена и в окне Solution Explorer (команда View- Solution Explorer) в проекте появится новая запись. Элемент нового проекта (рис. 14) имеет имя, указанное в диалоговом окне на рис. 13 и содержит определение класса, известного как DataSet со строгим контролем типов.

Объект *DataSet* — это набор данных, в него входят коллекции *DataTables* и *DataRelations*. Коллекция *DataTable* содержит наборы строк и столбцов с данными, класс *DataRelation* позволяет связывать данные элементов *DataTables.*

Данные в объекте *DataSet* отсоединены от БД. После собирания результатов запроса в объекте *DataSet* с помощью объекта *SqlDataAdapter* соединение между БД и объектом *DataSet* прекращается. Объект *SqlDataAdapter* предназначен для работы с отсоединенными данными. Возможно, наилучшее подтверждение такой его структуры — метод *Fill.* Для вызова этого метода не требуется «живое» подключение к БД. Когда вызван метод *Fill* объекта *SqlDataAdapter,* не имеющего открытого подключения к БД, объект открывает соединение, выполняет запрос к БД, выбирает и заносит результаты запроса в объект *DataSet* и затем закрывает соединение с БД.

Таким образом, между объектами *DataAdapter* и *DataSet* нет прямой связи.

Чтобы заполнить объект *DataTable* в объекте *DataSet,* последний передают в качестве параметра методу *Fill* объекта *SqlDataAdapter.*

Изменения содержимого объекта *DataSet* не сказываются на содержимом БД. Если другие пользователи изменят данные БД, соответствующие содержимому вашего объекта *DataSet,* вы не увидите этих изменений.
Чтобы в ADO.NET передать в БД изменения, хранящиеся в объекте *DataSet,* используется метод *Update* объекта *SqlDataAdapter.* Он принимает объект *DataSet* в качестве параметра. Объект *DataSet* может кэшировать изменения, но логика обновления находится именно в объекте *SqlDataAdapter.*

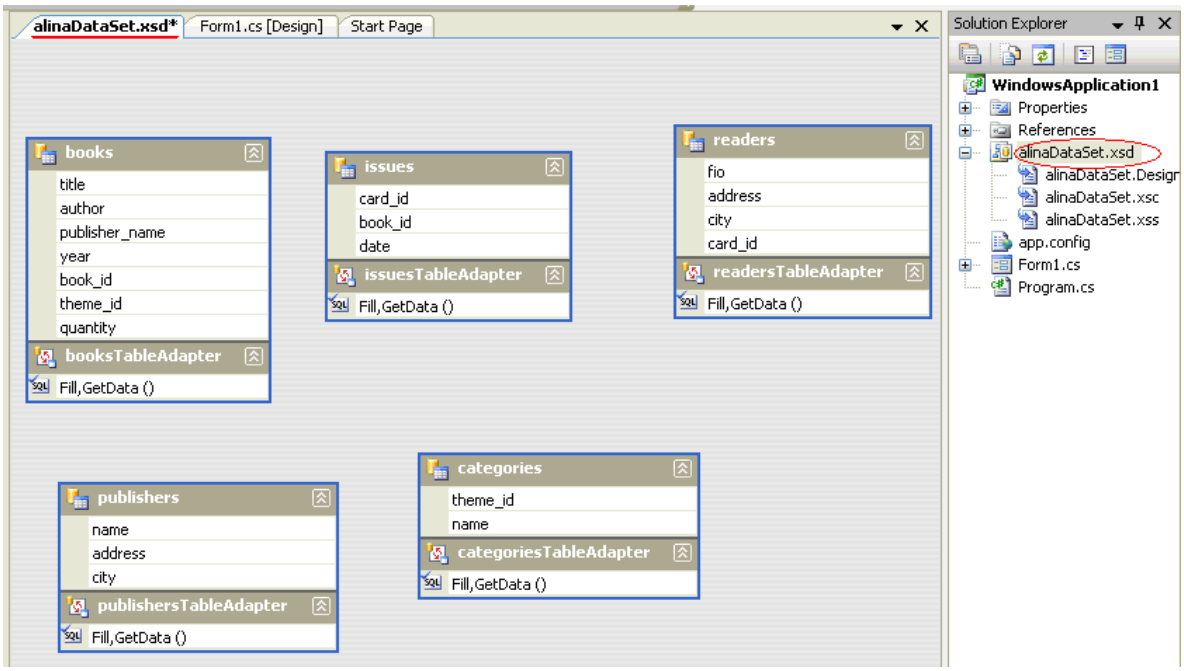


Рис 14

Преимущество периода разработки объектов *DataSet* со строгим контролем типов состоит в том, что благодаря технологии IntelliSense и автодополнению операторов в Visual Studio .NET писать код для доступа к содержимому объекта *DataSet* со строгим контролем типов гораздо проще, чем аналогичный код для обычного объекта *DataSet.*

Кроме того, это дает возможность уменьшить код, поскольку код инициализации класса *DataSet* со строгим контролем типов включает код для создания схемы и необходимых объектов *DataTable, DataColumn, DataRelation* и *Constraint.*

Во время создания Windows- или веб-приложения, в котором применяется связывание с данными посредством объекта *DataSet* со строгим контролем типов, гораздо проще связать элементы управления с данными во время выполнения. Такой объект содержит собственную информацию схемы, и Visual Studio .NET предоставит вам список таблиц и полей, с которыми можно связать элемент управления.

6. Добавление объектов DataRelation. Каскадирование изменений

После создания класса *Dataset* и его объектов *DataTable*, необходимо связать их между собой.

Чтобы установить для объекта *DataTable* свойство *PrimaryKey,* необходимо выбрать один или несколько объектов *DataColumn,* которые представляют собой первичный ключ. Затем щелкните правой кнопкой мыши выбранный объект *DataColumn* и выберите команду Set Primary Key контекстного меню (Рис. 15). Если впоследствии потребуется изменить или удалить первичный ключ, то это можно будет сделать с помощью контекстного меню.

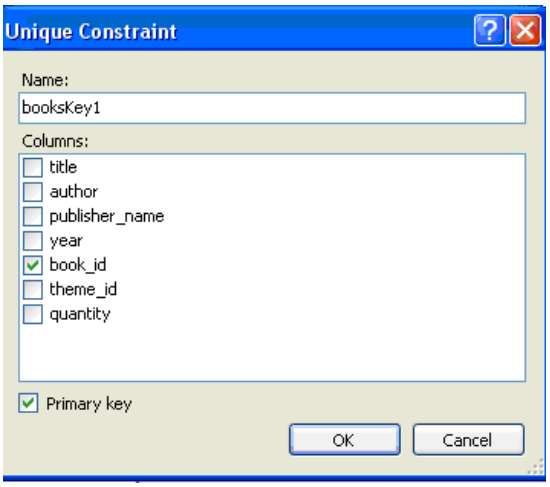


Рис. 15

Чтобы добавить объект *DataRelation,* щелкните правой кнопкой мыши любой элемент конструктора *DataSet* со строгим контролем типов (или откройте меню Data) и выберите команду Add and Relation. Это дает возможность открыть диалоговое окно, показанное на рис. 16.

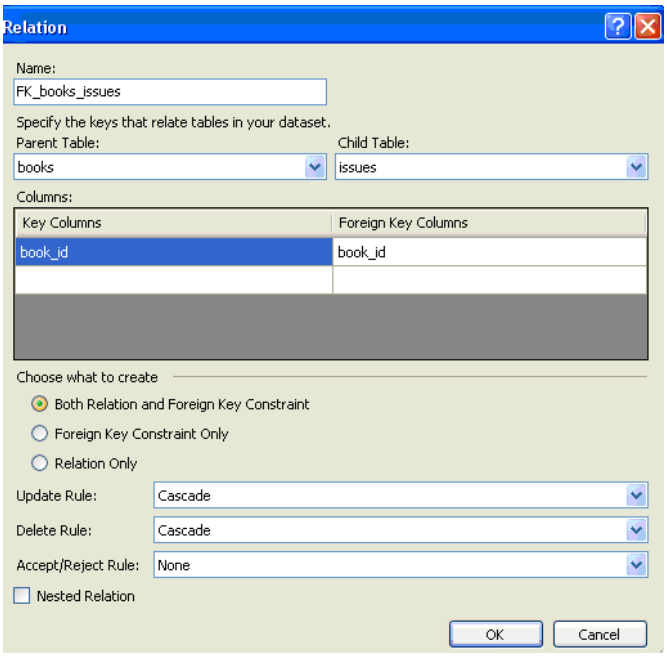


Рис. 16

Родительские и дочерние объекты *DataTable* можно выбрать с помощью списков в верхней части диалогового окна. Имя объекта *DataRelation* изменится, как только вы измените таблицы,

использованные для объекта *DataRelation.* Выбрав родительские и дочерние объекты *DataTable,* выберите в таблице прямо под списками объекты *DataColumn* для *DataRelation.*

Нижняя половина диалогового окна дает вам возможность определить, будет ли создан объект *DataRelation, ForeignKeyConstraint* или оба. По умолчанию создается объект *DataRelation. У*становки по умолчанию могут породить проблемы для многих разработчиков.

Если вы собираетесь разрешить внесение изменений в данные объекта *DataSet,* то объекты *ForeignKeyConstraint* должны быть связаны с объектами *Data-Relation*. Объект *ForeignKeyConstraint* каскадно вносит изменения в родительскую и дочернюю записи в *DataSet.*

Давайте рассмотрим два распространенных сценария, когда могут возникнуть проблемы из-за того, что объект *ForeignKeyConstraint* не связан с объектом *DataRelation* в *DataSet,* который позволяет вносить изменения.

Если в объекте *DataSet* удалить родительскую запись, то дочерние записи автоматически будут помечены как удаленные при условии (и только при условии), что объект *DataRelation* связан с объектом *ForeignKeyConstraint,* свойству *DeleteRule* которого присвоено значение *Cascade.* В противном случае дочерние записи *Data Row* не будут помечены как отложенные для удаления. При попытке внести отложенные удаления обратно в базу данных она может отменить удаление родительской записи из-за того, что дочерние записи все еще существуют.

Аналогично, изменение в ключевом значении родительской записи автоматически будет передано в соответствующие дочерние записи при условии (и только при условии), что объект *DataRelation* связан с объектом *ForeignKeyConstraint,* свойству *UpdateRule* которого присвоено значение *Cascade.*

Таким образом, необходимо выбрать переключатель Both Relation and Foreign Key Constraint, и свойствам *UpdateRule, DeleteRule* и *AcceptRejectRule* объекта *ForeignKeyConstraint* присвоить значения *Cascade, Cascade* и *None.*

После установки отношений связности, будет сформирована схема объекта DataSet. (рис. 17)

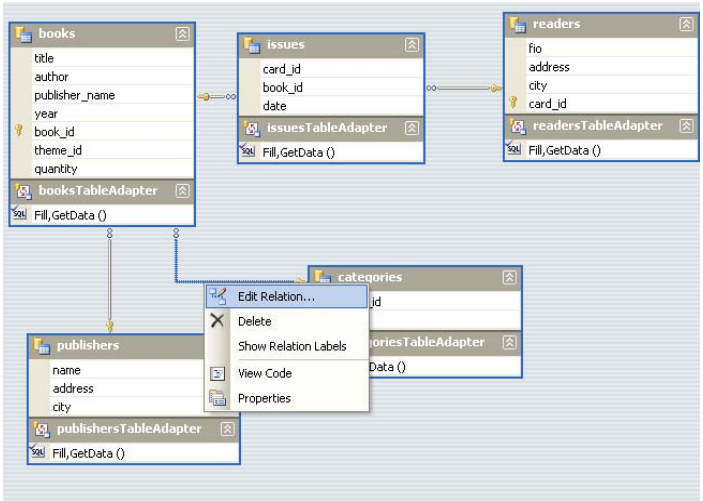


Рис. 17

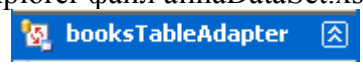
7. Генерирование логики обновления

Ранее говорилось, что объект *SqlDataAdapter* предназначен для работы именно с отсоединенными данными объекта DataSet.

Чтобы в ADO.NET передать в БД изменения, хранящиеся в объекте *DataSet,* используется метод *Update* объекта *SqlDataAdapter.* Он принимает объект *DataSet* в качестве параметра. Объект *DataSet* может кэшировать изменения, но логика обновления находится именно в объекте *SqlDataAdapter.*

Мастер TableAdapter Configuration Wizard генерирует способ передачи отложенных вставок, обновлений и удалений в базу данных. Логика этих действий содержится в методах *InsertCommand, UpdateCommand* и *DeleteCommand* объекта *SqlDataAdapter,* который хранится внутри объекта *TableAdapter.*

Сгенерируем логику обновления на примере объекта booksTableAdapter класса alinaDataSet.

1. Необходимо выбрать в окне Solution Explorer файл alinaDataSet.xsd
2. Щелкните правой кнопкой на элементе 

объекта books Data-Table.
3. Затем в окне Properties найдите свойство *UpdateCommand* данного объекта. Выберите свойство *CommandText* и щелкните кнопку с изображением многоточия (...), которая расположена справа от его значения.

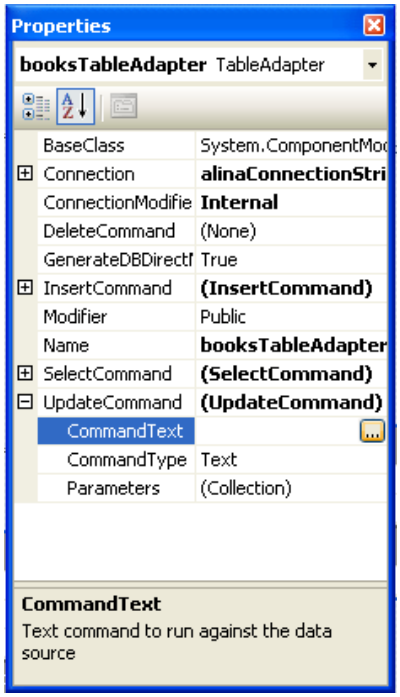


Рис.

4. В результате откроется окно Query Builder со свойством *CommandText* для свойства *UpdateCommand* объекта *TableAdapter* (рис.). В окне необходимо сгенерировать запрос UPDATE отложенных изменений

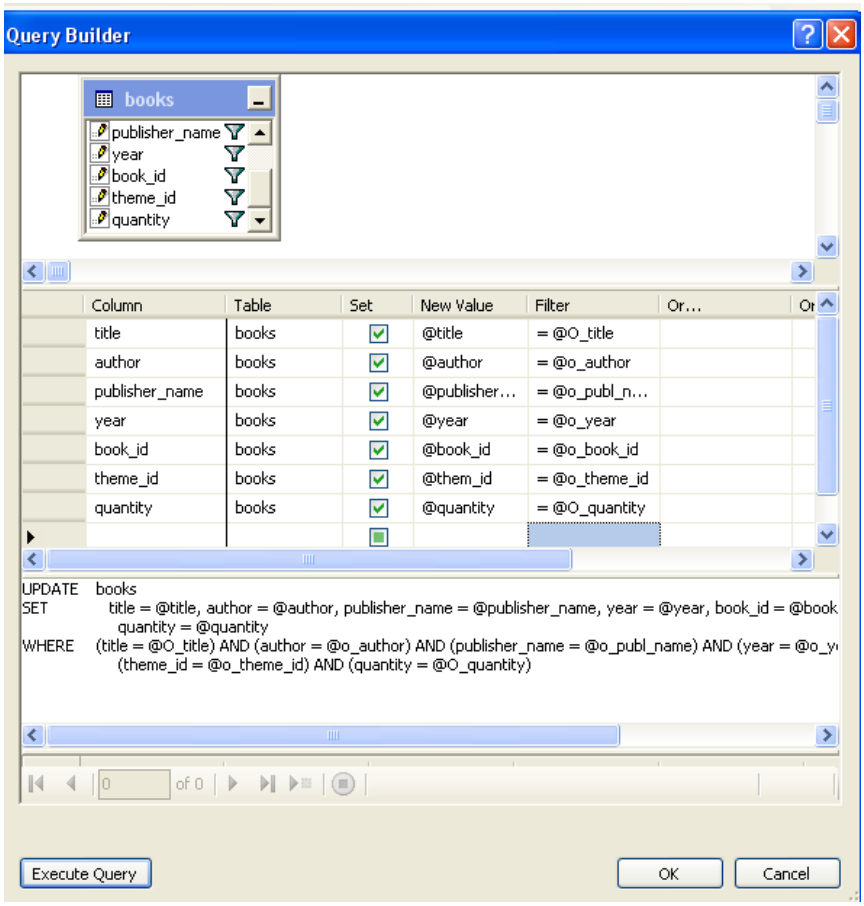


Рис.

5. Аналогично в окне свойства сгенерируем запрос *DeleteCommand* (рис.)

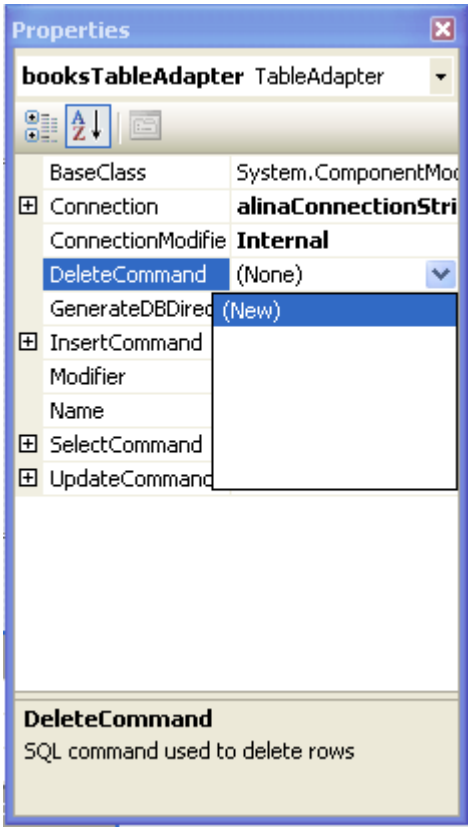


Рис.

6. В окне Query Builder необходимо сгенерировать запрос DELETE отложенных удалений (рис.)

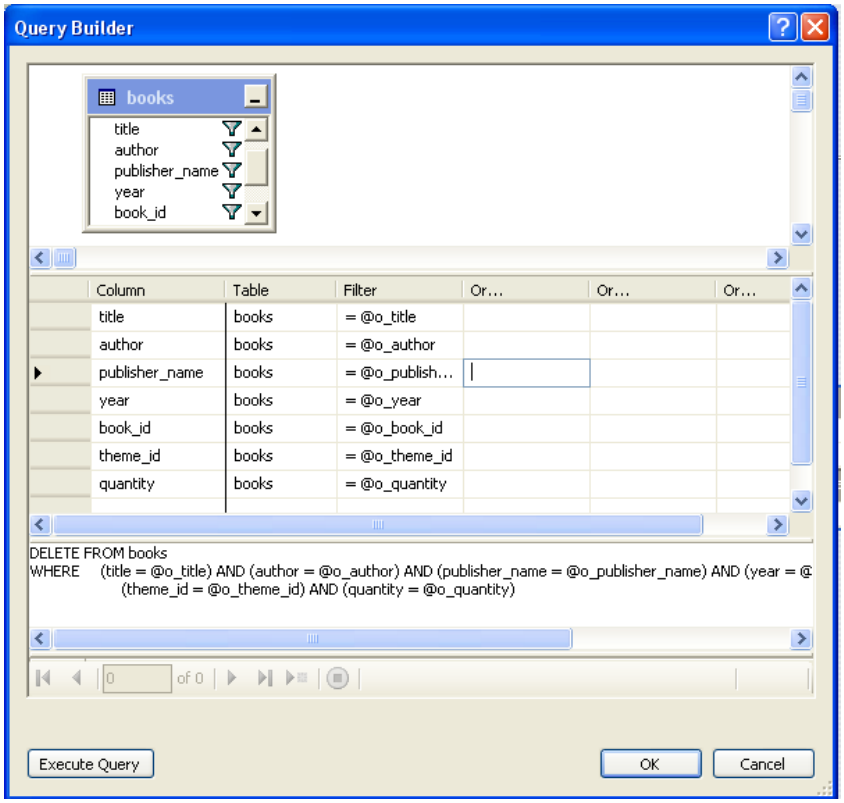


Рис.

Таким образом, у каждого объекта TableAdapter объекта DataSet должны быть сгенерированы методы логики обновления *InsertCommand, UpdateCommand, DeleteCommand и SelectCommand,* которые впоследствии будут использоваться при создании средств ввода и обновления данных в объектах Data-Table *(*рис.)

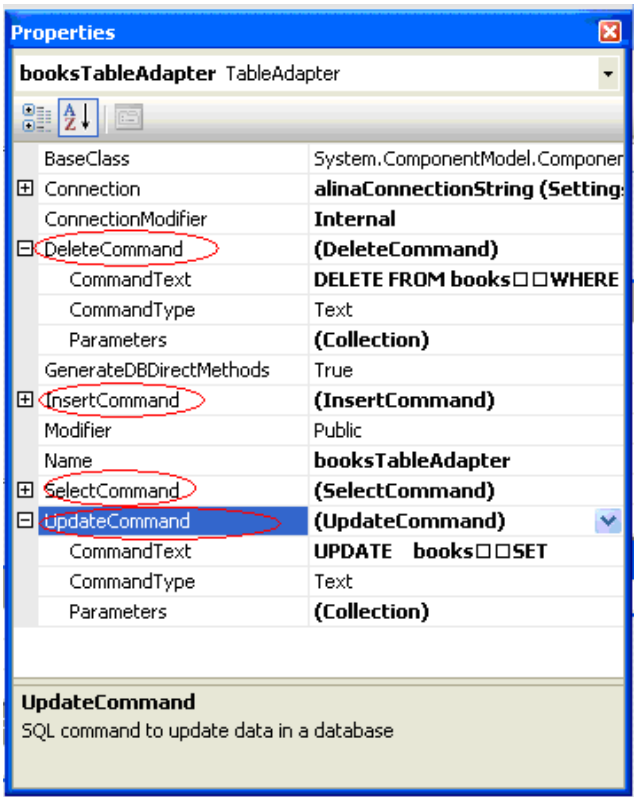


Рис.

8. Создание элементов пользовательского интерфейса системы

Для создания пользовательского интерфейса системы удобно использовать меню и вкладки.

8.1 Создание вкладок

1. Командой **Вид→Панель элементов (View-Toolbox)** или нажав клавиши **Ctrl+Alr+X** откройте **Панель элементов**.
2. В группе **Контейнеры** выберете 

и перетащите мышкой на форму. На форме появится компонент **`tabControl1`**. Путем растягивания **`tabControl1`** можно придать ему нужные размеры. Результат показан на рис. .

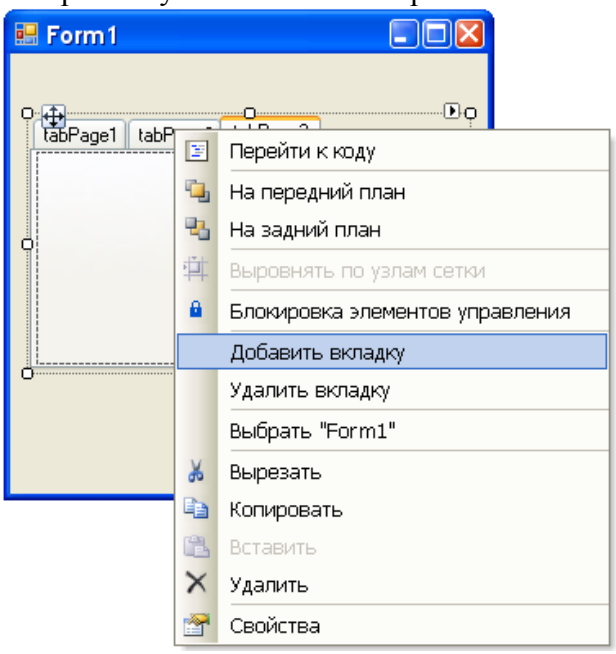


Рис.

3. Далее, щелкнув правой кнопкой мыши на вкладках, выберете **Добавить вкладку**. Добавьте необходимое количество вкладок.
4. Сделать нужную вкладку активной можно, щелкнув мышью по ее заголовку. Далее необходимо щелкнуть по полю вкладки. При этом в окне **Свойства** будут отображаться свойства данной вкладки, например, **`tabPage1`**:
    1. **`Text`** – заголовок вкладки
    2. **`Name`** – имя вкладки, используемое в программе
    3. и другие.

    Теперь на вкладках можно размещать элементы просмотра данных и управления.
5. При желании можно изменить цвет вкладок, шрифт, добавить фоновое изображение.

8.2 Создание меню

1. В **Панели элементов (View\Toolbox)** необходимо выбрать компонент 

. Перетащите его на форму, в результате чего на поле под формой появится объект **menuSrip1**, а на форме сверху строка меню.
2. Для ввода элементов меню необходимо щелкнуть по полосе меню вверху формы (рис.).

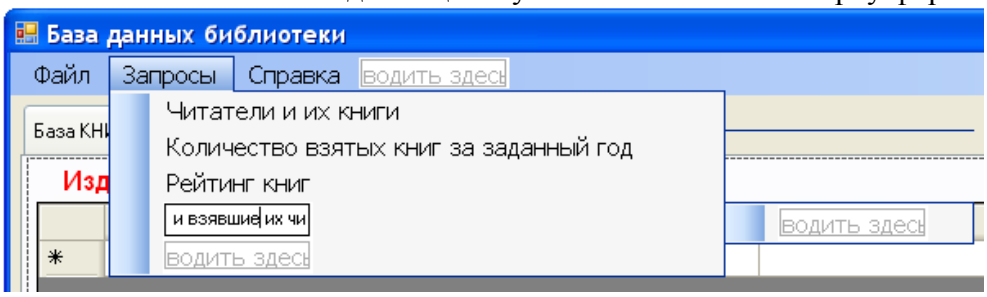
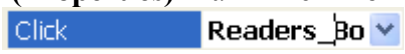


Рис.

3. После окончания ввода всех пунктов меню можно выделить пункт меню одним щелчком мыши и в окне Свойства изменить свойство Name, используемое в программе для работы с этим пунктом меню (по умолчанию имена пунктов меню формируются из введенного названия, в том числе из русских букв).
4. Далее в окне **Свойства (Properties)** нажмите кнопку  (события). Найдите в списке событий событие Click 

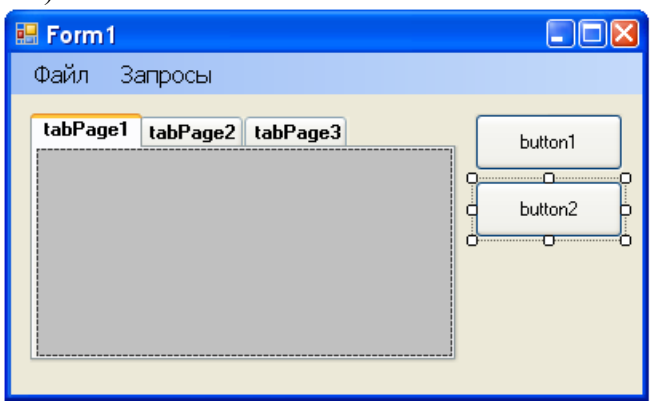
и дважды щелкните по нему. В результате откроется окно ввода кода, в котором будет автоматически вставлен обработчик события Click (нажатие мышью).

```
private void Readers_BooksToolStripMenuItem_Click(object sender, EventArgs e)
        {
            /*Здесь необходимо вставить код, выполняющийся при выборе
              данного пункта меню*/
        }
```

8.3 Создание кнопок

1. В **Панели элементов (View\Toolbox)** необходимо выбрать компонент 

. Перетащить его на форму, в результате чего на форме появится элемент **button1** (рис.).



2. Измените свойства кнопки (надпись **Text**, имя **Name** и др.)
3. Двойным щелчком по кнопке создайте обработчик нажатия кнопки. При этом будет сгенерирован следующий код:

```
private void button1_Click(object sender, EventArgs e)
        {
            /*Здесь необходимо вставить код, выполняющийся
              при нажатии кнопки*/
        }
```

8.4 Создание таблиц

1. Необходимо отобразить источник данных, который был создан с помощью мастера Data Source Configuration командой Data\Show Data Sources.
2. Открыть все узлы в иерархической структуре элементов. (рис. 18)

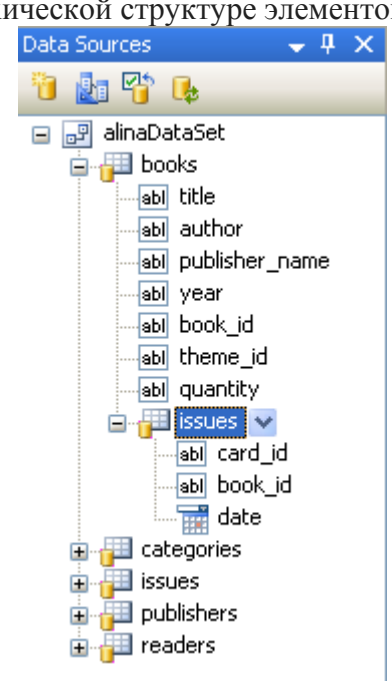
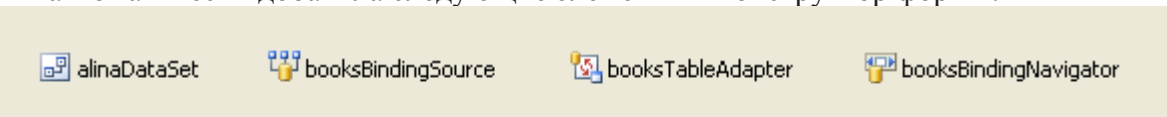


Рис 18

3. Перетащите необходимый узел на форму, например books, при этом среда Visual Studio автоматически добавила следующие элементы в конструктор формы:



Таким легким способом можно создать все таблицы DataGridView.
При этом будет автоматически генерироваться код заполнения таблиц данными при запуске формы методом Fill :

```
private void Form1_Load(object sender, EventArgs e)
        {

            this.categoriesTableAdapter.Fill(this.alinaDataSet.categories);
            this.publishersTableAdapter.Fill(this.alinaDataSet.publishers);
            this.readersTableAdapter.Fill(this.alinaDataSet.readers);
            this.booksTableAdapter.Fill(this.alinaDataSet.books);
            this.issuesTableAdapter.Fill(this.alinaDataSet.issues);
        }
```

Пример создания пользовательского интерфейса системы представлен на рис.

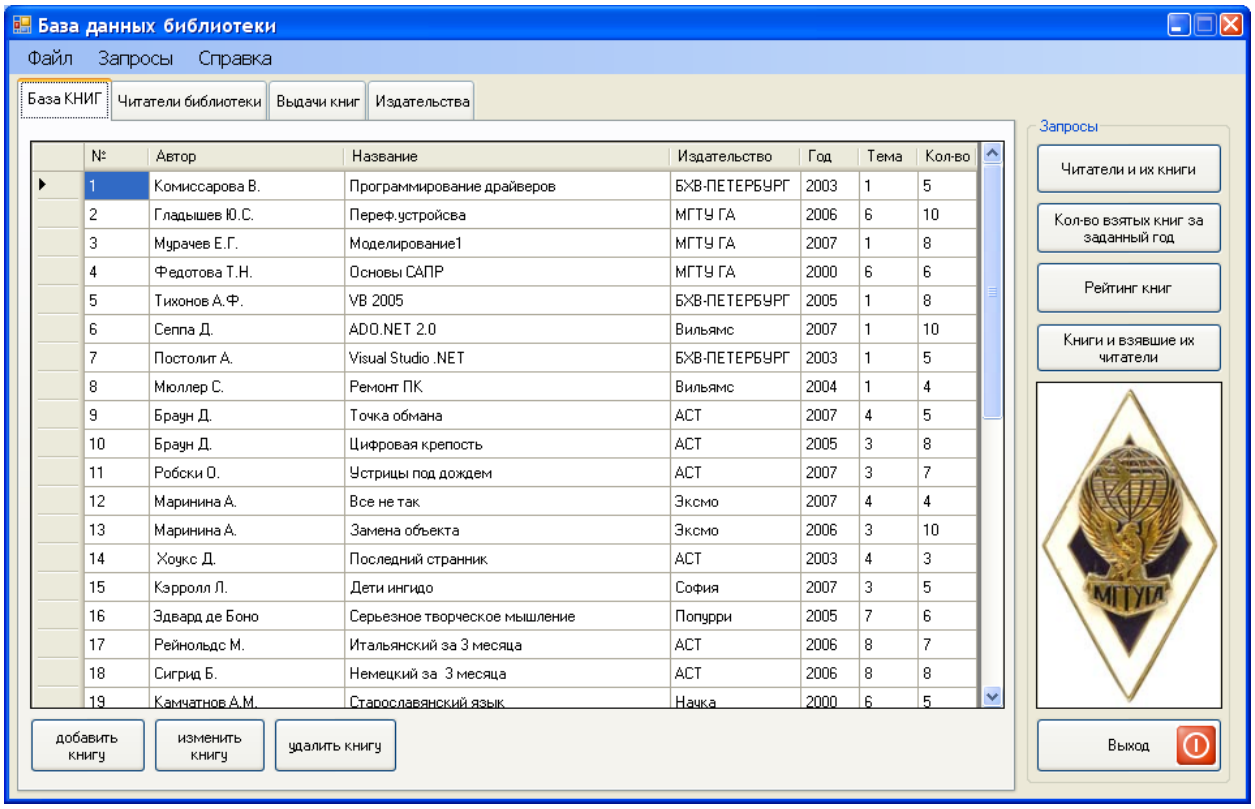
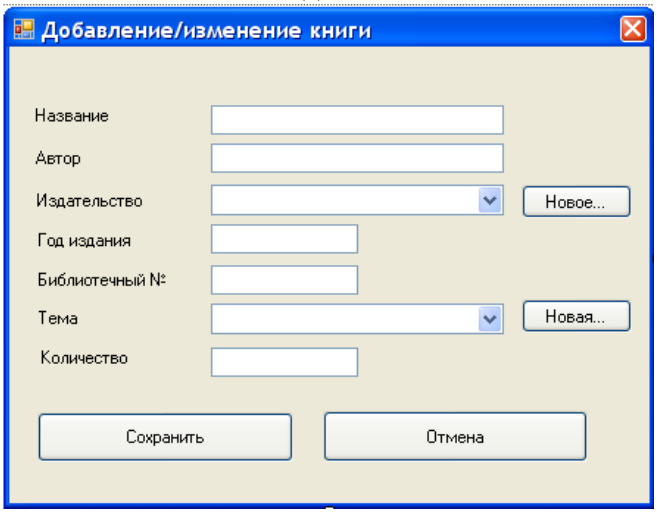


Рис.

9. Создание средств ввода и обновления данных

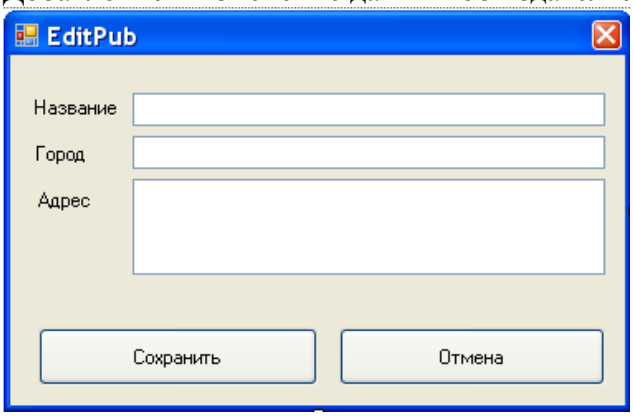
1. Создать на вкладке **База КНИГ** кнопки **Добавить книгу**, **Изменить книгу**, **Удалить книгу**.
2. Создать формы:

Добавление и изменение данных о книге **AddBook:**

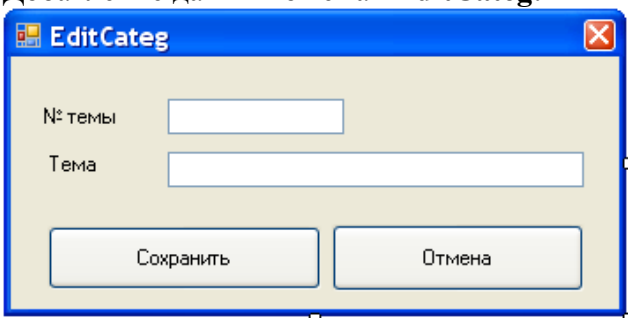


Кнопка **Отмена**: DialogResult=Cancel
Кнопка **Новое…**: DialogResult=Yes
Кнопка **Новая…**: DialogResult=No

Добавление и изменение данных об издательстве **EditPub:**



Добавление данных о темах **EditCateg:**



3. Написать процедуру связывания полей ввода с данными для формы AddBook:

```
public void Fill(BindingSource bsBooks, BindingSource bsCateg, BindingSource bsPub)
        {
            this.tbTitle.DataBindings.Add("Text", bsBooks, "title", false,
                                          DataSourceUpdateMode.OnValidation);
            this.tbAuthor.DataBindings.Add("Text", bsBooks, "author", false,
                                          DataSourceUpdateMode.OnValidation);
            this.tbYear.DataBindings.Add("Text", bsBooks, "year", false,
                                          DataSourceUpdateMode.OnValidation);
            this.tbID.DataBindings.Add("Text", bsBooks, "book_id", false,
                                          DataSourceUpdateMode.OnValidation);
            this.tbQuantity.DataBindings.Add("Text", bsBooks, "quantity", false,
                                          DataSourceUpdateMode.OnValidation);

            this.cbPub.DataSource = bsPub;
            this.cbPub.DisplayMember = "name";
            this.cbPub.ValueMember = "name";
            this.cbPub.DataBindings.Add("SelectedValue", bsBooks,
                                          "publisher_name");

            this.cbTheme.DataSource = bsCateg;
            this.cbTheme.DisplayMember = "name";
            this.cbTheme.ValueMember = "theme_id";
            this.cbTheme.DataBindings.Add("SelectedValue", bsBooks, "theme_id");
          }
```

Написать аналогичные функции для остальных форм.

4. В обработчике нажатия кнопки Сохранить формы AddBook написать код, проверяющий правильность ввода данных:

```
private void button1_Click(object sender, EventArgs e)
        {
            if (tbTitle.Text == "")
            {
```

```
            MessageBox.Show("Введите название книги",
                            "Добавление/изменение книги", MessageBoxButtons.OK,
                            MessageBoxIcon.Warning);
            return;
        }

        ..............

        /*Проверка остальных полей*/

        ..............

        if (tbQuantity.Text == "")
        {
            MessageBox.Show("Введите количество экземпляров",
                            "Добавление/изменение книги", MessageBoxButtons.OK,
                            MessageBoxIcon.Warning);
            return;
        }
        //DialogResult устанавливается равным OK для закрытия формы
        this.DialogResult = System.Windows.Forms.DialogResult.OK;

      }
```

5. В файле главной формы создать функцию:

```
private void EditBooks()
        {
            AddBook window = new AddBook();
            window.GetConn(sqlcConnect);
            window.Fill(this.booksBindingSource, this.categoriesBindingSource,
this.publishersBindingSource);

        m1: DialogResult drRes = window.ShowDialog();


            if (drRes == System.Windows.Forms.DialogResult.Yes)
            {
                this.AddPub();
                goto m1;
            }
            if (drRes == System.Windows.Forms.DialogResult.No)
            {
                this.AddCategories();
                goto m1;
            }
            if (drRes == System.Windows.Forms.DialogResult.OK)
                this.Save();
            else
                booksBindingSource.CancelEdit();
        }
```

Функция осуществляет вывод формы добавления книги и при необходимости форм добавления издательства и темы.

6. Создать функцию добавления издательства:

```
public void AddPub()
        {
            this.publishersBindingSource.AddNew();
            EditPub window = new EditPub();
            window.Fill(this.publishersBindingSource);

            if (window.ShowDialog() == System.Windows.Forms.DialogResult.OK)
                this.SavePub();
            else
                publishersBindingSource.CancelEdit();
```

```
            this.Validate();
            this.publishersBindingSource.EndEdit();
            this.publishersTableAdapter.Update(this.alinaDataSet.publishers);
        }
```

7. Создать функцию добавления темы:

```
private void AddCategories()
        {
            this.categoriesBindingSource.AddNew();

            EditCateg window = new EditCateg();
            window.Fill(this.categoriesBindingSource);

            DialogResult drRes = window.ShowDialog();

            if (drRes == System.Windows.Forms.DialogResult.OK)
                this.SaveCateg();
            else
                categoriesBindingSource.CancelEdit();
          }
```

8. Написать обработчики нажатия кнопок **Добавить книгу** и **Изменить книгу**.

```
private void add_Click_1(object sender, EventArgs e)
        {
            this.booksBindingSource.AddNew();
            this.EditBooks();
        }

private void edit_Click(object sender, EventArgs e)
        {
            this.EditBooks();
        }
```

9. Написать обработчик нажатия кнопки **Удалить книгу**, выводящий предупреждение перед удалением.

```
private void delete_Click_1(object sender, EventArgs e)
        {

            if (MessageBox.Show("Вы действительно хотите удалить эту запись?",
                                "Удаление книги", MessageBoxButtons.YesNo,
                                MessageBoxIcon.Question) ==
               System.Windows.Forms.DialogResult.Yes)
            {
                this.booksBindingSource.RemoveCurrent();
                this.Save();
            }
         }
```

Аналогичным образом создаются формы для ввода и изменения данных в других таблицах.

10. Создание формы окна подключения к серверу

Параметры подключения к серверу в проекте хранятся в файле **app.config**. Чтобы при установке программы на конкретный компьютер не приходилось ее перекомпилировать, удобно использовать окно подключения к серверу, в котором указываются параметры подключения.

1. Создать форму **`login`** (рис.)

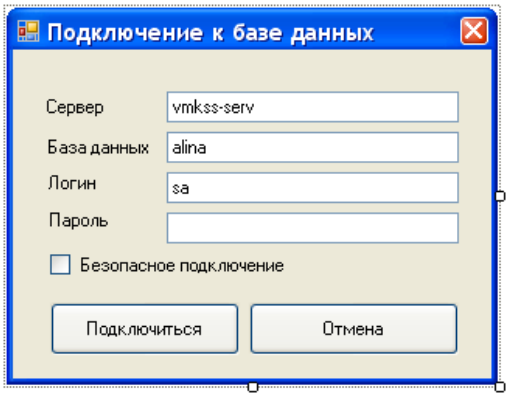


Рис.

2. Использовать пространство имен **`System.Data.SqlClient`**:

```
using System.Data.SqlClient;
```

3. В обработчике события Click кнопки Подключиться (button1) прописать следующий код:

```
private void button1_Click(object sender, EventArgs e)
{
  SqlConnectionStringBuilder bldr = new SqlConnectionStringBuilder();
  bldr.DataSource=this.tbDataSource.Text;           //сервер
  bldr.InitialCatalog = this.tbInitialCatalog.Text; //база данных
  if(this.tbUserID.Text!="")
      bldr.UserID = this.tbUserID.Text;             //логин
  if(this.tbPassword.Text!="")
      bldr.Password = this.tbPassword.Text;         //пароль
  bldr.IntegratedSecurity = this.chbSequrity.Checked;//безопасное подключение
  strCon = bldr.ConnectionString;
}
```

4. Свойству **`DialogResult`** кнопки **Отмена** (button2) присвоить значение **`Cancel`**.
5. Объявить глобальную переменную **`string strCon`** (строка подключения) и создать метод для доступа к ней:

```
string strCon;
public string RetStr()
{
    return strCon;
}
```

6. Изменить файл Program.cs в соответствии со следующим примером:

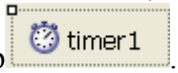
```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace lib2009
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]

        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);

            login window = new login();
conn_error: DialogResult drRes = window.ShowDialog();
            if (drRes == System.Windows.Forms.DialogResult.OK)
            {
                Form1 MainFrm = new Form1();
```

```
                SqlConnection cn = new SqlConnection(window.RetStr());
                try
                {
                    cn.Open();
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Ошибка при кодключении к серверу. База данных
недоступна.", "Ошибка",
                                    MessageBoxButtons.OK, MessageBoxIcon.Error);
                    goto conn_error;
                }

                logo logo_scr = new logo(); //запуск заставки
                logo_scr.ShowDialog();

                MainFrm.GetConn(cn);

                Application.Run(MainFrm);
            }


        }
    }
}
```

Здесь **Form1** – главная форма приложения. Она должна содержать метод доступа к переменной

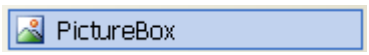
```
SqlConnection sqlcConnect:
SqlConnection sqlcConnect;
public void GetConn(SqlConnection cn)
        {
            sqlcConnect = cn;
        }
```

7. Создать форму заставки (**logo**)

8. Закрытие заставки осуществляется с помощью таймера
9. Разместить на форме logo таймер 

.
10. Установить следующие свойства таймера timer1:
    - Enabled=True
    - Interval=3000 (3000 мс = 3 с)
11. В обработчике события Tick написать следующий код:

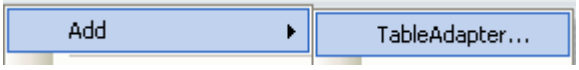
```
private void timer1_Tick(object sender, EventArgs e)
        {
            this.Close();
        }
```

12. Разместить на форме компонент 

и загрузить подходящую картинку.

11. Разработка запросов

11.1 Добавление объекта TableAdapter в объект Dataset

1. В окне **Data Sources (Источники данных)** щелкнуть правой кнопкой мыши на **alinaDataSet**. В появившемся меню выбрать пункт Изменить набор данных в конструкторе **(Edit DataSet with Designer)**.
2. В открывшемся окне щелкнуть в любом месте поля правой кнопкой мыши и из меню выбрать (**Add/TableAdapter**) : 

Откроется мастер **TableAdapter Configuration Wizard** (рис.). Первое окно мастера, показанное на рис. , приглашает вас ввести данные для своего соединения. Выбираем созданное нами соединение

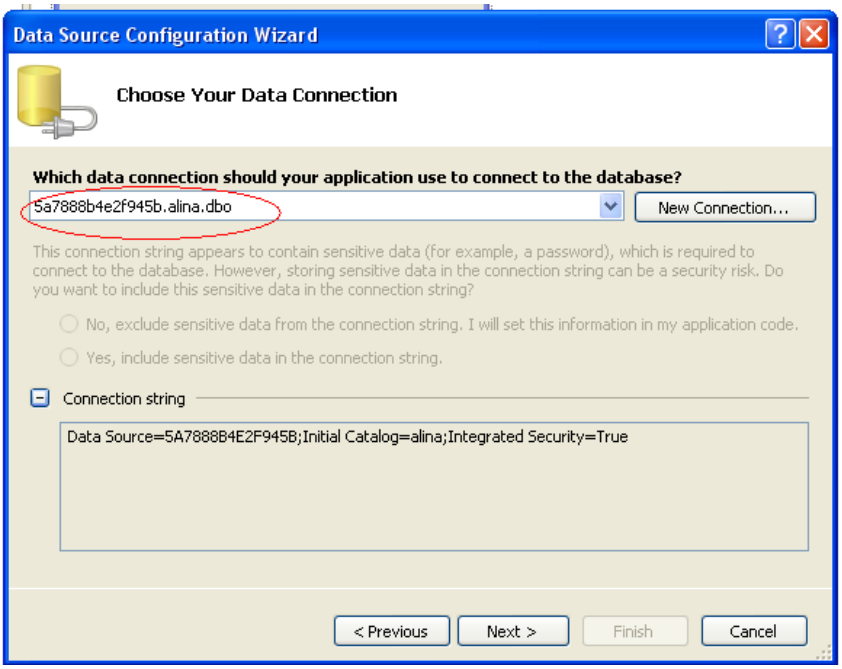


Рис.

3. После того как вами будет предоставлена информация о соединении, мастер запрашивает тип действия, которое будет выполнять объект *TableAdapter*. Для этого используются три переключателя. Как видно из рис. , вы можете указать SQL-выражение, создать новую основанную на запросе хранимую процедуру либо использовать уже существующую процедуру. Два последних переключателя доступны только в том случае, если вы работаете с базой данных SQL Server. Второй переключатель создает хранимые процедуры для получения, вставки, обновления и удаления записей данных, которые основываются на SQL-запросе. Третий переключатель дает возможность выбрать хранимую процедуру для получения, вставки, обновления и удаления записей данных.
   Выберем первый переключатель создания SQL выражения:

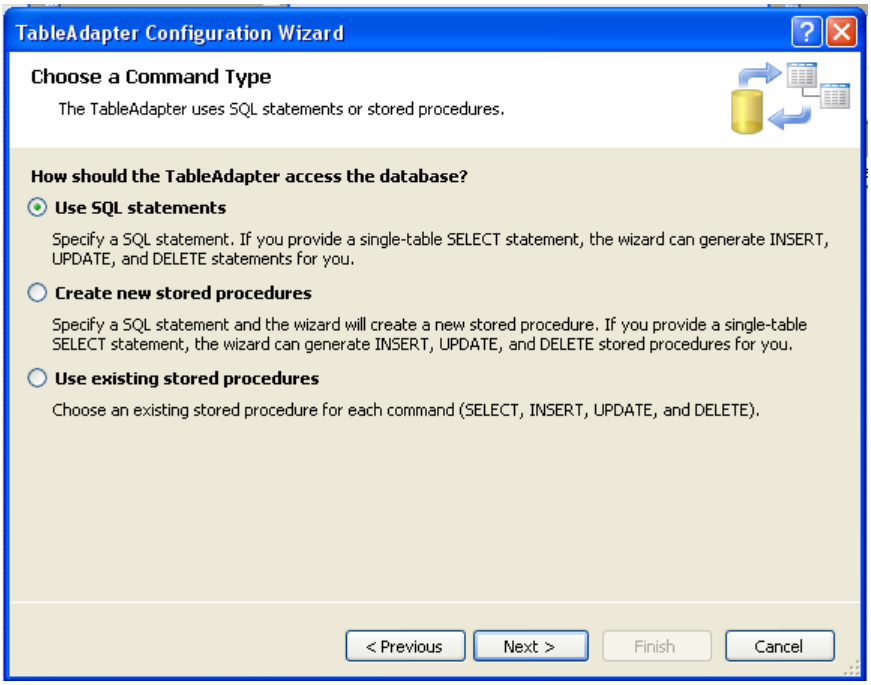
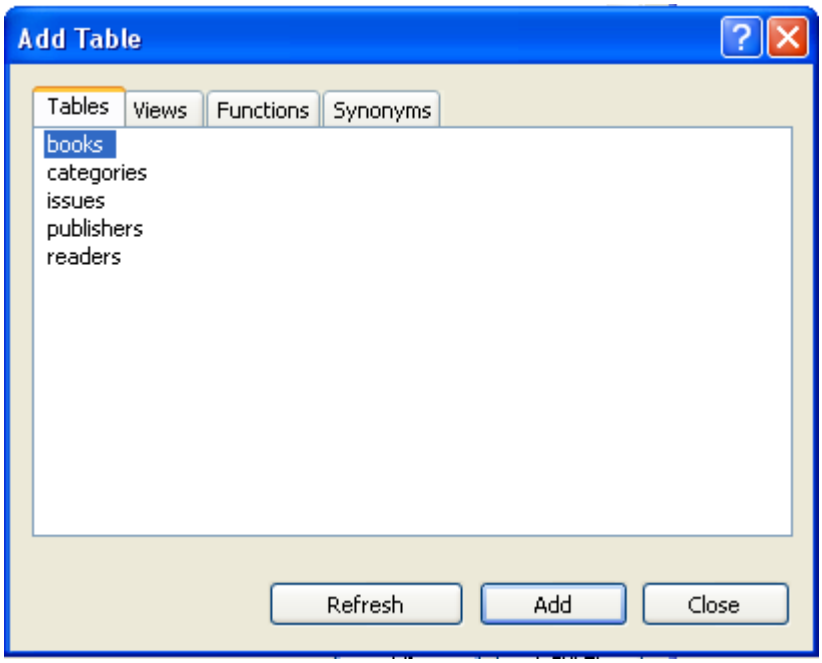


Рис.

4. На приглашение Мастера **Введите инструкцию SQL** ввести в окно текст запроса или воспользоваться построителем запросов, выберем Query Builder, нажав соответствующую кнопку. (рис.) Построитель запросов позволяет создать запрос, а также выполнить его отладку.

Рис

5. Мастер выведет следующее окно, позволяющее выбрать соответствующие таблицы для вывода данных запроса (рис.)

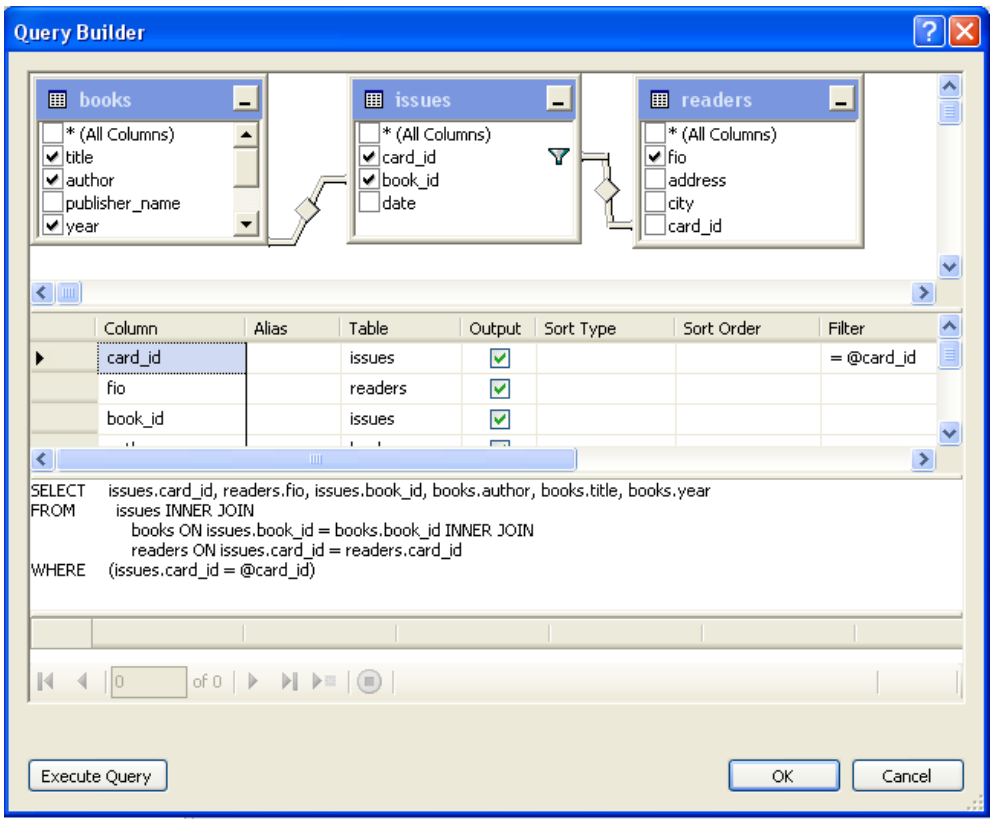


Рис

6. После выбора таблицы для своего SQL-утверждения, Query Builder даст возможность создать свой запрос графическим способом. Как видно из рис. , выбрать нужные поля нетрудно. Кроме того, можно добавлять порядок сортировки и критерии запроса. При этом не нужно набирать вручную соответствующий SQL-запрос. Query Builder выводит на экран SQL-код вашего запроса, и вы сможете выполнить его, щелкнув кнопку Execute Query. Если вы довольны запросом, щелкните кнопку ОК.

Сформированный текст запроса «Выдача справок по заданному читателю и его книгам»:

```
SELECT issues.card_id, readers.fio, issues.book_id, books.author, books.title,
books.year
FROM     issues INNER JOIN
         books ON issues.book_id = books.book_id INNER JOIN
         readers ON issues.card_id = readers.card_id
WHERE  (issues.card_id = @card_id)
```



Рис

7. Когда вы предоставите свое SQL-утверждение, мастер позволит управлять методами, доступными объекту *TableAdapter*. По умолчанию мастер создаст метод *Fill,* который внесет результаты запроса в соответствующий объект *DataTable* в объекте *DataSet* со строгим контролем типов. Он также добавит метод *GetData,* который возвратит новую копию объекта *DataTable* со строгим контролем типов, содержащую результаты запроса. С помощью мастера можно указать, нужны ли вам эти методы, а также управлять названиями методов, как показано на рис. .

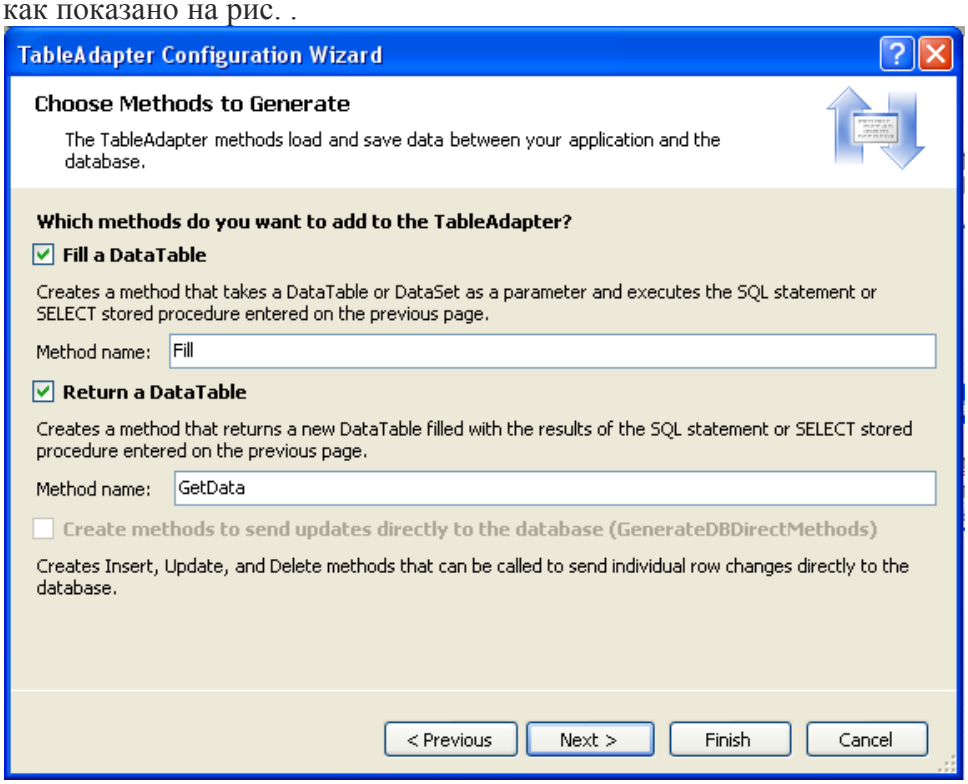
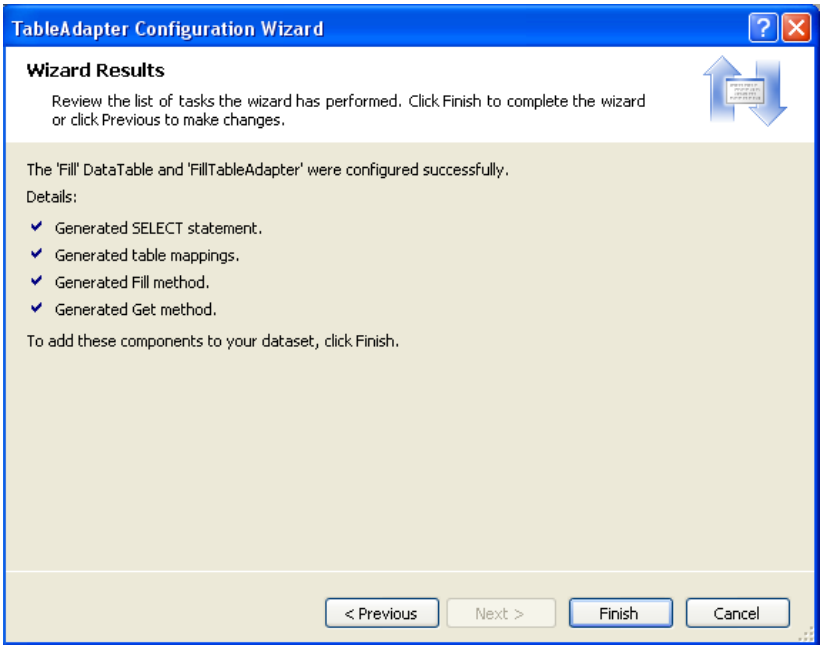
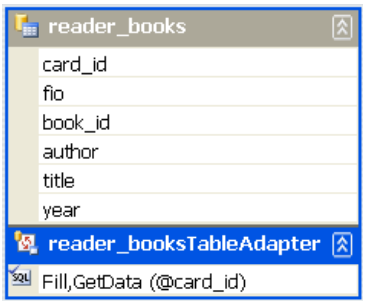


Рис.

8. И наконец, мастер TableAdapter Configuration Wizard выводит на экран результаты, как показано на рис. , объясняя, какие функциональные возможности мастер предоставил новому объекту *TableAdapter*

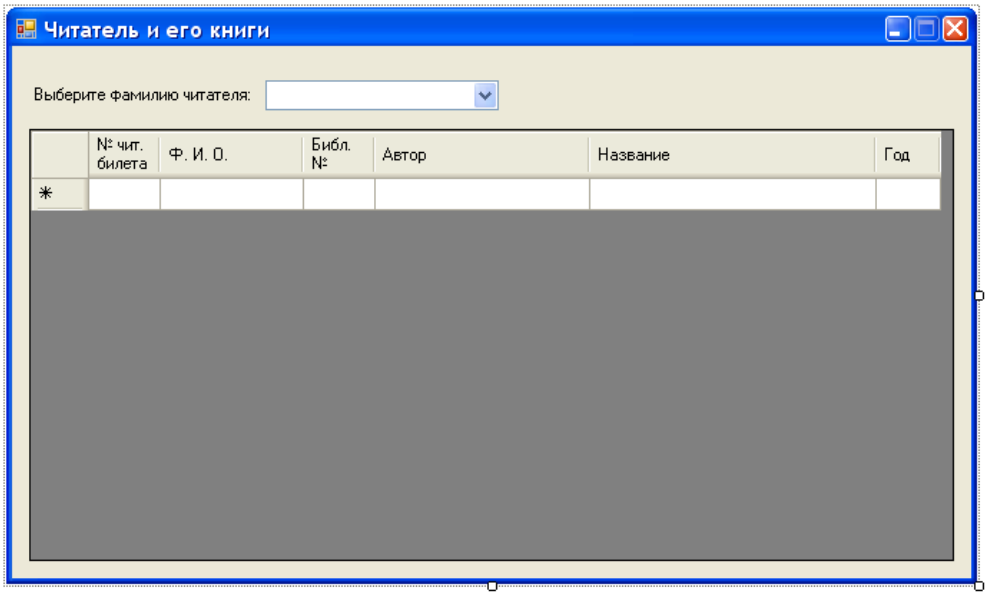


9. После завершения работы мастера в окне **alinaDataSet** появится следующая таблица, свойства которой можно изменять в окне **Свойства**:



11.2 Создание формы для выполнения запроса «Выдача справок по заданному читателю и его книгам»

1. Создать форму **`Reader_Books`** и расположить на ней компоненты **ComboBox**, **DataGridView** и **BindingSource**.
2. Установить следующие значения свойств **BindingSource**:
   - `Name=readerbooksBindingSource  //название`
   - `DataSource=alinaDataSet  //источник данных`
   - `DataMember=reader_books  //таблица`
3. Свойству **Name** ComboBox присвоить значение **cbReader**, а свойству **DataSource** – значение **readersBindingSource**.
4. Свойству **DataSource** dataGridView1 присвоить значение **readerbooksBindingSource**.



5. Для заполнения **dataGridView1** необходимо в обработчике события написать следующий код:

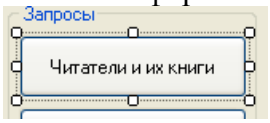
```
this.reader_booksTableAdapter.Fill(this.alinaDataSet.reader_books,
                                   (int)this.cbReader.SelectedValue);
```

Текст программы для выполнения запроса

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace lib2009
{
    public partial class Reader_Books : Form
    {
        SqlConnection sqlcConnect;

        public Reader_Books()
        {
            InitializeComponent();
        }
        public void GetConn(SqlConnection cn)
        {
            sqlcConnect = cn;
        }

        private void Reader_Books_Load(object sender, EventArgs e)
        {
            this.readersTableAdapter.Connection = sqlcConnect;
            this.reader_booksTableAdapter.Connection = sqlcConnect;
            // TODO: Данная строка кода позволяет загрузить данные в таблицу
"alinaDataSet.readers". При необходимости она может быть перемещена или удалена.
            this.readersTableAdapter.Fill(this.alinaDataSet.readers);
        }

        private void button1_Click(object sender, EventArgs e)
        {
```

```
            this.reader_booksTableAdapter.Fill(this.alinaDataSet.reader_books,
(int)this.cbReader.SelectedValue);
        }

private void cbReader_SelectedIndexChanged(object sender, EventArgs e)
        {
            if(cbReader.Text!="")
            this.reader_booksTableAdapter.Fill(this.alinaDataSet.reader_books,
                                               (int)this.cbReader.SelectedValue);
        }
    }
}
```

6. Вызов формы Читатель и его книги (Reader_Buuks) из главной формы




```
private void btnReadBooks_Click(object sender, EventArgs e)
        {
            Reader_Books window = new Reader_Books();
            window.GetConn(sqlcConnect);
            window.Show();
        }
```

7. Вызов из меню:

```
private void Readers_BooksToolStripMenuItem_Click(object sender, EventArgs e)
        {
            btnReadBooks.PerformClick();
        }
```

12. Обновление информации в окне программы

Для обновления информации в окне программы необходимо вызывать методы **`Fill()`** соответствующих адаптеров таблиц. У всех объектов **TableAdapter** есть свойство **ClearBeforeFill**, которому изначально присвоено значение **True**. Если это значение присвоено ему при вызове метода **Fill**, объект **TableAdapter** очистит записи, доступные в объекте **DataTable** со строгим контролем типов. Чтобы это не вызывало ошибки, необходимо перед вызовом метода **`Fill()`** очистить соответствующие таблицы. Для этого необходимо создать метод Clr, выполняющий следующий запрос (для таблицы issues):

```
SELECT  card_id, book_id, date
FROM     issues
WHERE  (card_id <> card_id)
```

Результат выполнения этого запроса всегда пуст.

Для обновления информации в окне используется следующий код:

```
private void tabPage1_Enter(object sender, EventArgs e)
        {
            this.issuesTableAdapter.Clr(this.alinaDataSet.issues);

            this.categoriesTableAdapter.Fill(this.alinaDataSet.categories);
            this.publishersTableAdapter.Fill(this.alinaDataSet.publishers);

            this.readersTableAdapter.Fill(this.alinaDataSet.readers);
            this.booksTableAdapter.Fill(this.alinaDataSet.books);
            this.issuesTableAdapter.Fill(this.alinaDataSet.issues);
        }
```

Этот код прописывают в обработчике события **Enter** вкладок.

Литература

1. Дэвид Сеппа Программирование на ADO.NET 2.0 .- СПб.: Питер, 2007
2. Дейтел Х. C# в подлиннике. Наиболее полное руководство. – СПб.: БХВ – Петербург, 2006