МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Невинномысский технологический институт (филиал)

ЛАБОРАТОРНЫЙ ПРАКТИКУМ
по дисциплине
«Автоматизированное управление промышленными предприятиями»
Методические указания к выполнению лабораторных работ для студентов направления 15.04.04 – «Автоматизация технологических процессов и производств»

Невинномысск 2013

Методические указания предназначены для студентов направления 15.04.04 – «Автоматизация технологических процессов и производств». Они содержат основы теории, порядок проведения лабораторных работ и обработки экспериментальных данных, перечень контрольных вопросов для самоподготовки и список рекомендуемой литературы. Работы подобраны и расположены в соответствии с методикой изучения дисциплины. Объем и последовательность выполнения работ определяются преподавателем в зависимости от количества часов, предусмотренных учебным планом дисциплины, как для очной, так и для заочной форм обучения.

Методические указания разработаны в соответствии с требованиями ФГОС ВПО в части содержания и уровня подготовки выпускников направления 15.04.04 – «Автоматизация технологических процессов и производств».

№ п/п	Содержание компетенции	Шифр
Профессиональные компетенции		***ПК-№***
1.	Способностью разрабатывать эскизные, технические и рабочие проекты автоматизированных и автоматических производств различного технологического и отраслевого назначения, технических средств и систем автоматизацииуправления, контроля, диагностики и испытаний, систем управления жизненным циклом продукции и ее качеством с использованием современных средств автоматизации проектирования, отечественного и зарубежного опыта разработки конкурентоспособной продукции, проводить технические расчеты по проектам, технико-экономический и функционально-стоимостной анализ эффективности проектов, оценивать их инновационный потенциал и риски	ПК-4
2.	Способностью проводить математическое моделирование процессов, оборудования, средств и систем автоматизации, контроля, диагностики, испытаний и управления с использованием современных технологий научных исследований, разрабатывать алгоритмическое и программное обеспечение средств и систем автоматизации и управления	ПК-16

Составитель канд. техн. наук А.А. Евдокимов

Ответственный редактор канд. техн. наук Д.В. Болдырев

Содержание

Вариант задания на лабораторную работу

В соответствии с номером в журнале студент должен выбрать один из следующих тематических вариантов для выполнения лабораторных работ:

1. управления технологическим процессом;
2. управления распределенным хранением данных;
3. управления шлюзом локальной сети;
4. управления тестированием сотрудников предприятия;
5. управления финансовыми операциями;
6. управления контентом сайта;
7. управления базами конфиденциальных данных;
8. управления активным оборудованием сети;
9. поддержки принятия решения;
10. обработки данных;
11. управления доступом к сменным носителям;
12. управления обновлением ПО, включая антивирусное ПО;
13. автоматизации офиса;
14. диагностики;
15. peer-to-peer сети;
16. мониторинга состояния энергетического оборудования;
17. VPN сети;
18. IP-телефонии;
19. видеоконференции;
20. управления вычислительным кластером;
21. управления автономными роботами;
22. навигации (GPS, ГЛОНАС);
23. электронного документооборота;
24. видеонаблюдения и обнаружения движения;
25. CSRP.

Для выполнения лабораторных работ необходимо ознакомиться с тематикой и сформировать соответствующую базу данных в MS SQL Server.

Лабораторная работа №1

Соединение с источником данных

Цель работы: ознакомиться с ADO.NET и источниками данных. LINQ. Entity Framework 4.5.

Краткие сведения из теории

Для перемещения данных между их постоянным хранилищем и приложением в первую очередь необходимо создать соединение с источником данных (Connection). В арсенале ADO.NET для этих целей имеется ряд объектов:

SQLConnection – объект, позволяющий создать соединение с базами данных MS SQL Server;

OleDbConnection – объект, позволяющий создать соединение с любым источником данных (простые текстовые файлы, электронные таблицы, базы данных) через OLE DB;

OdbcConnection – объект, позволяющий создать соединение с ODBC-источниками данных.

Жизненный цикл объекта *Connection* состоит из таких этапов как: объявление объекта соединения; создание объекта соединения; определение строки соединения; использование соединения, например, для создания команды; открытие соединения; выполнение команды; закрытие соединения; обработка полученных данных; изменение команды; повторное открытие соединения; выполнение команды; закрытие соединения.

Пример объявления соединений показан в листинге 1.1.

Листинг 1.1.

```
public class Form1 : System.Windows.Forms.Form
{
    private System.Data.SqlClient.SqlConnection sqlConnection1;
    private System.Data.OleDb.OleDbConnection oleDbConnection1;
    private System.Data.Odbc.OdbcConnection odbcConnection1;
    private System.Data.Odbc.OdbcConnection odbcConnection2;
    ...
```

}

Операторы создания объектов соединения помещаются в блок инициализации, как показано в листинге 1.2.

Листинг 1.2.

```
private void InitializeComponent()
{
    this.sqlConnection1 = new System.Data.SqlClient.SqlConnection();
    this.oleDbConnection1 = new System.Data.OleDb.OleDbConnection();
    this.odbcConnection1 = new System.Data.Odbc.OdbcConnection();
    this.odbcConnection2 = new System.Data.Odbc.OdbcConnection();
…
}
```

Первое свойство объекта соединения, которое необходимо определить в блоке инициализации для установления связи с базой данных – это строка соединения *ConnectionString*. В строке соединения управляемых поставщиков необходимо, как минимум, указать местоположение базы данных и требуемую аутентификационную информацию. Помимо этого, каждый поставщик данных определяет дополнительные параметры соединения. Если в строке соединения не указаны значения всех возможных параметров, они считаются установленными по умолчанию.

Строки соединения управляемого поставщика SQL Server

Строки соединения управляемого поставщика SQL Server содержат множество параметров, однако наиболее часто используются только некоторые из них. Самыми распространенными из них являются:

Data Source – имя сервера баз данных;

Initial Catalog – база данных, находящаяся на сервере;

User ID – идентификатор пользователя, который должен применяться для аутентификации пользователя на сервере баз данных;

PWD- пароль, который должен применяться для аутентификации пользователя на сервере баз данных;

Например, строка соединения с базой данных “basa_user”, расположенной на MS SQL Server с именем “ITS-SERVER” для пользователя с именем “UserA” и паролем “123” будет выглядеть следующим образом:

this.sqlConnection1.ConnectionString = "user id=usera;

data source=\"ITS-SERVER\";initial catalog=basa_user;pwd=123";

Строки соединения управляемого поставщика OLE DB

Строки соединения управляемого поставщика OLE DB похожи на строки соединения SQL Server. Все параметры строки соединения, за исключением параметра Provider (Поставщик), определяются специфическим поставщиком OLE DB. В качестве значений параметра Provider могут быть использованы такие значения как: *SQLOLEDB.1*- для SQL Server; *Microsoft.Jet.OLEDB.4.0* – для Microsoft Access; *PostgreSQL* – для базы данных PostgreSQL. Приведем пример строки соединения для MS SQL Server:

this.oleDbConnection1.ConnectionString = @"User ID=usera;Data Source=""ITS-SERVER"";Initial Catalog=basa_user; Provider= ""SQLOLEDB.1"";pwd=123";

Строки соединения управляемого поставщика ODBC

Строки соединения управляемого поставщика ODBC немного отличаются от строк соединения SQL Server или OLE DB. Управляемый поставщик ODBC поддерживает два различных метода создания строки соединения:

- создание строки соединения на основе имени источника данных (Data Source Name DSN);
- использование динамических строк соединения.

Недостаток использования DSN заключается в том, что каждый компьютер должен либо быть специально настроенным, либо иметь доступ к DSN-файлам. В частности, это бывает проблематичным в таких системах, как кластер Web-серверов. С другой стороны, использование DSN позволяет сохранить определенный контроль над строками соединения. Так, если меняется местоположение сервера или аутентификационная информация, разработчику придется изменить всего лишь атрибуты DSN, а не программный код. Атрибуты DSN можно использовать также для динамического генерирования информации о соединении. В этом случае параметры

строки соединения, такие как DRIVER и SERVER, можно указать непосредственно в строке соединения, а не прятать их в атрибуте DSN.

Например, строка соединения на основе имени источника данных может выглядеть так:

this.odbcConnection1.ConnectionString = "UID=usera;DSN=stud;PWD=123";

а динамическая строка соединения с тем же источником данных выглядит следующим образом:

this.odbcConnection2.ConnectionString = "UID=usera;DRIVER=SQL Server;
PWD=123;SERVER=ITS-SERVER";

Объекты *Connection* имеют два базовых метода для открытия и закрытия соединения (*Open* и *Close*). Пример использования данных методов приведен в листинге 1.3.

Листинг 1.3.

```
private void Form1_Load(object sender, System.EventArgs e)
{
    try
    {
        this.sqlConnection1.Open();
        MessageBox.Show("Успешное SQL соединение");
        this.sqlConnection1.Close();
    }
    catch(Exception ex)
    {
        MessageBox.Show("Нет SQL соединения"+ex.Message);
    }
        try
    {
        this.oleDbConnection1.Open();
        MessageBox.Show("Успешное OLE DB соединение");
        this.oleDbConnection1.Close();
    }
```

```
        catch(Exception ex)
        {
            MessageBox.Show("Нет OLE DB соединения"+ex.Message);
        }
            try
        {
            this.odbcConnection1.Open();
            MessageBox.Show("Успешное ODBC1 соединение");
            this.odbcConnection1.Close();
        }
        catch(Exception ex)
        {
            MessageBox.Show("Нет ODBC1 соединения"+ex.Message);
        }
            try
        {
            this.odbcConnection2.Open();
            MessageBox.Show("Успешное ODBC2 соединение");
            this.odbcConnection2.Close();
        }
        catch(Exception ex)
        {
            MessageBox.Show("Нет ODBC2 соединения"+ex.Message);
        }
}
```

Задание на лабораторную работу

Создать три базы данных, состоящих каждая из двух таблиц: родительской и дочерней. Для создания баз использовать среды: MS SQL Server и MS Access. Таблицы во всех базах должны иметь одинаковую структуру. На Windows-форме, созданной в среде Microsoft Visual C#.NET, расположить четыре кнопки: две для соединения с БД MS SQL Server (SQLConnection, LINQtoSQL), для соединения с БД

MS Access (OdbcConnection с использованием DSN и с использованием динамической строки соединения). Предусмотреть вывод сообщения об успешном и неуспешном соединении с предоставлением информации об ошибке.

Содержание отчета

В отчете указать цель работы, привести листинг разработанной программы и формы с результатами работы.

Контрольные вопросы

1. Что такое ADO.NET?
2. Что такое провайдер в ADO.NET?
3. Как происходит взаимодействие клиента и провайдера в ADO.NET?
4. Что является источником данных?
5. В чем особенность использования DSN и динамической строки соединения?
6. Технология LINQ.

Лабораторная работа №2

Прямой доступ к данным

Цель работы: получить навык построения приложений, взаимодействующих с БД. LINQ. Entity Framework 4.5.

Краткие сведения из теории

Для выполнения основных задач, связанных с взаимодействием с базами данных, можно использовать объекты команд. Команда данных содержит ссылку на SQL-запрос или хранимую процедуру, которые собственно и реализуют конкретные действия. Команда данных – это экземпляр класса *System.Data.Odbc.OdbcCommand* или *System.Data.OleDb. OleDbCommand* или *System.Data.SqlClient.SqlCommand*.

С использованием объекта *DataCommand* в приложении можно выполнять следующие действия:

- Исполнять команды SELECT, которые возвращают набор записей. Причем результат выборки можно обрабатывать непосредственно, без его загрузки в набор данных *DataSet*. Для чтения результатов используется объект *DataReader*, который работает в режиме «только чтение, только вперед» и может быть связан с компонентом визуального отображения данных. Эти объекты полезно использовать в тех

случаях, когда имеет место ограничение на ресурсы памяти или требуется высокая скорость загрузки данных.

- Выполнять команды, обеспечивающие создание, редактирование, удаление объектов базы данных (например, таблиц, хранимых процедур и т.п.).
- Выполнять команды, обеспечивающие получение информации из каталогов баз данных.
- Выполнять динамические SQL-команды, позволяющие модифицировать, вставлять или удалять записи непосредственно в базе данных, вместо того, чтобы редактировать таблицы набора данных *DataSet*, а затем копировать эти изменения в базу данных.
- Выполнять команды, которые возвращают скалярное, то есть единственное значение.
- Выполнять команды, которые возвращают данные из базы данных SQL Server в формате XML. Эта возможность используется в Интернет- приложениях. Например, когда нужно выполнить запрос и получить данные в формате XML, чтобы преобразовать данные к HTML-формату и затем отправить их браузеру.

Существует два основных способа создания объекта *DataCommand.* Во-первых, можно воспользоваться стандартным синтаксисом создания объекта команды, как показано в листинге 2.1.

Листинг 2.1.

```
System.Data.Odbc.OdbcConnection con1;// соединение
System.Data.Odbc.OdbcCommand cmd1;//команда
…
cmd1=new System.Data.Odbc.OdbcCommand();
```

Во-вторых, объект команды можно создать на основе объекта *Connection* (листинг 2.2).

Листинг 2.2.

```
System.Data.Odbc.OdbcConnection con1;// соединение
System.Data.Odbc.OdbcCommand cmd2; //команда
…
cmd2=con1.CreateCommand();
```

Команда – мощный инструмент, позволяющий проводить сложные операции с базой данных. В ADO.NET существует три типа команд:

- Text – текстовая команда состоит из SQL-инструкций, указывающих управляемому поставщику на необходимость выполнения определенных действий на уровне базы данных. Текстовые команды передаются в базу данных без предварительной обработки, за исключением случаев передачи параметров;
- StoredProcedure – хранимая процедура; эта команда вызывает процедуру, которая хранится в самой базе данных;
- TableDirect – команда такого типа предназначена для извлечения из базы данных полной таблицы.

Тип команды устанавливается в свойстве *CommandType*, которое по умолчанию имеет значение *Text*, а сам текст команды прописывается в свойстве *CommandText*. Например:

Листинг 2.3.

```
cmd1=new System.Data.Odbc.OdbcCommand();
cmd1.Connection=con1;
cmd1.CommandText="DELETE FROM студент
                  WHERE номер_студента=377";
```

Выполнение команд

За подготовкой команды следует ее выполнение. В ADO.NET существует несколько способов выполнения команд, которые отличаются лишь информацией, возвращаемой из базы данных. Ниже приведены методы выполнения команд, поддерживаемые всеми управляемыми поставщиками:

- *ExecuteNonQuery()* – этот метод применяется для выполнения команд, которые не должны возвращать результирующий набор данных. Так как при вызове данного метода возвращается число строк, добавленных, измененных или удаленных в результате выполнения команды, он может использоваться в качестве индикатора успешного выполнения команды;
- *ExecuteScalar()*- этот метод выполняет команду и возвращает первый столбец первой строки первого результирующего набора данных. Данный метод может быть

полезен для извлечения из базы данных итоговой информации; количества, максимального или минимального значения, итоговой суммы или среднего значения;

- *ExecuteReader()* – этот метод выполняет команду и возвращает объект *DataReader*, представляющий собой однонаправленный поток записей базы данных.

Использование метода *ExecuteNonQuery()* показано в листинге 2.4.

Листинг 2.4.

```
System.Data.Odbc.OdbcConnection con1;// соединение
System.Data.Odbc.OdbcCommand cmd1;//команда
System.Data.Odbc.OdbcCommand cmd2; //команда
…
cmd1=new System.Data.Odbc.OdbcCommand();
cmd1.Connection=con1;
cmd1.CommandText="DELETE FROM студент
                    WHERE номер_студента=377";
try
{
     cmd1.ExecuteNonQuery();
     MessageBox.Show("Успешное удаление!");
}
catch(Exception ex)
{
     MessageBox.Show("Удаление не удалось!"+ex.Message);
}

cmd1.CommandText="UPDATE студент SET стипендия=2000
                    WHERE номер_студента=375";
try
{
     cmd1.ExecuteNonQuery();
     MessageBox.Show("Успешное изменение!");
}
```

```
catch(Exception ex)
{
    MessageBox.Show("Изменение не удалось!"+ex.Message);
}
cmd1.CommandText="INSERT INTO студент
                    VALUES(2,'Николаев', 'ИС-24',4000,'6.01.2005')";
try
{
    cmd1.ExecuteNonQuery();
    MessageBox.Show("Успешная вставка!");
}
catch(Exception ex)
{
    MessageBox.Show("Вставка не удалась!"+ex.Message);
}
cmd1.Dispose();
```

Использование метода *ExecuteScalar()* показано в листинге 2.5.

Листинг 2.5.

```
System.Data.Odbc.OdbcCommand cmd=
new System.Data.Odbc.OdbcCommand("SELECT MAX(стипендия)
                                FROM студент",con1);…
int result= (int)cmd.ExecuteScalar();
```

Использование параметров

Если в приложении используется объект *DataCommand*, который работает непосредственно с элементами базы данных, то выполняемые SQL-запросы и хранимые процедуры обычно требуют параметров. Перед выполнением таких запросов необходимо определить значения параметров. Объект *DataCommand* поддерживает коллекцию *Parameters*. Прежде чем выполнить команду, необходимо установить значение для каждого параметра команды, как показано в листинге 2.6.

Листинг 2.6.

```
    cmd2.CommandText="DELETE FROM студент
```

```
                    WHERE  номер_студента=?";
cmd2.Parameters.Add("p1",System.Data.Odbc.OdbcType.Int);
cmd2.Parameters["p1"].Value=337;
    try
    {
        cmd2.ExecuteNonQuery();
        MessageBox.Show("Успешное удаление!");
    }
    catch(Exception ex)
    {
        MessageBox.Show("Удаление не удалось!"+ex.Message);
    }
```

Следует отметить, что формат параметризованных запросов отличается для разных управляемых поставщиков. Если в синтаксисе параметризованного запроса в качестве параметра в основном применяется знак вопроса «?», то для управляемого поставщика SQL Server используется @имя_параметра, как показано в листинге 2.7.

Листинг 2.7.

```
cmd2.CommandText="DELETE FROM студент
                    WHERE  номер_студента=@p1";
cmd2.Parameters.Add("@p1",System.Data.Odbc.OdbcType.Int);
cmd2.Parameters["@p1"].Value=337;
```

Альтернатива использованию объектов параметров

Процесс создания команды может быть сведен к созданию обычных строк, как показано в листинге 2.8.

Листинг 2.8.

```
int t=370;
cmd.CommandText=string.Format("DELETE FROM студент
                    WHERE номер_студента={0}",t);
cmd.ExecuteNonQuery();
```

Объект DataReader

Функциональные возможности объекта *DataReader* включают режим однонаправленного курсора. Механизм чтения данных объекта *DataReader* является устойчивым к ошибкам, позволяя за один шаг считать текущую запись в память и проверить наличие индикатора конца записи. Наиболее эффективно объект *DataReader* применяется при получении результатов выполнения запроса и их передаче на дальнейшую обработку.

В отличие от большинства других интерфейсов доступа к данным, в которых для считывания и изменения данных используется один и тот объект, в ADO.NET операции чтения и изменения данных разделены. Исключительно для чтения данных предназначен основной объект ADO.NET *DataReader.*

Объект *DataReader* обеспечивает доступ к данным, извлеченным в результате выполнения SQL-запроса, в режиме только для чтения. Объект DataReader представлен соответствующим классом: *System.Data. Odbc.OdbcDataReader, System.Data.OleDb.OleDbDataReader* или *System. Data.SqlClient.SqlDataReader*.

На всем протяжении существования объекта *DataReader* при чтении или при просмотре строк соединение должно оставаться открытым. Так как при работе с объектом *DataReader* соединение остается открытым, он может не извлекать за один раз все данные, полученные в результате выполнения запроса.

Создание объекта *DataReader*

Объект *DataReader* не может быть создан напрямую из клиентского кода – он создается объектом команды во время ее выполнения с помощью метода *ExecuteReader().* Пример создания объекта Datareader показан в листинге 2.9.

Листинг 2.9.

```
System.Data.Odbc.OdbcDataReader rdr;
…
cmd1.CommandText="SELECT * FROM студент";
try
{
    rdr=cmd1.ExecuteReader();
    MessageBox.Show("Успешная выборка!");
```

```
}
catch(Exception ex)
{
    MessageBox.Show("Выборка не удалась!"+ex.Message);
}
```

Для перемещения по результирующему набору данных предназначен метод *Read()*. Метод *Read()* обязательно должен быть вызван один раз еще до считывания первой записи. Благодаря этому объект *DataReader* более устойчив к ошибкам, так как перемещение строкового курсора и проверка того, достигнут ли конец результирующего набора данных, производится в одном операторе. По достижению конца результирующего набора данных метод *Read()* возвратит *false*. Объект *DataReader* позволяет осуществлять доступ к отдельным столбцам, представляя результирующий набор данных в виде массива. Посредством соответствующего индексатора ([]) указывается либо порядковый номер столбца, либо его имя, например, как показано в листинге 2.10.

Листинг 2.10.

```
cmd=con.CreateCommand();
cmd.CommandText="SELECT номер_студента, фамилия, группа,
                    стипендия, дата FROM студент";
rdr=cmd.ExecuteReader();
rdr.Read();
textBox1.Text=rdr[0].ToString();
textBox2.Text=rdr[1].ToString();
textBox3.Text=rdr["группа"].ToString();
textBox4.Text=rdr[3].ToString();
textBox5.Text=rdr[4].ToString();
```

Поскольку индексатор возвращает объект, его можно привести к типу столбца или вызвать для этого один из *get*-методов объекта *DataReader*, обеспечивающих безопасность типов, как показано в листинге 2.11.

Листинг 2.11.

```
cmd=con.CreateCommand();
```

```
cmd.CommandText="SELECT номер_студента, фамилия, группа,
                    стипендия, дата FROM студент";
rdr=cmd.ExecuteReader();
rdr.Read();
int v0=rdr.GetInt32(0);
textBox1.Text=v0.ToString();
string v1=rdr.GetString(1);
textBox2.Text=v1;
string v2=rdr.GetString(2);
textBox3.Text=v2;
int v3=rdr.GetInt32(3);
textBox4.Text=v3.ToString();
DateTime  v4=rdr.GetDateTime(4);
textBox5.Text=v4.ToString();
```

Обеспечивающие безопасность типов get-методы не производят никаких преобразований – они просто осуществляют привидение типа. Все get-методы, обеспечивающие безопасность типов, требуют передачи в качестве параметра порядкового номера столбца, что, естественно, менее удобно, чем использование его имени.

В листинге 2.11 показана также привязка данных объекта *DataReader* к элементам управления Windows-формы типа TextBox - текстовое поле. В листинге 2.12 показана привязка данных объекта *DataReader* к списку.

Листинг 2.12.

```
while(rdr.Read())
{
    this.listBox1.Items.Add(rdr[1].ToString());
}
```

Задание на лабораторную работу

Создать объекты DataCommand для вставки, удаления и изменения данных в БД с параметрами. Выполнить команды модификации данных с выводом сообщения об удачном или неудачном выполнении. Создать объект DataCommand для выборки

данных и объект DataReader для получения результата. Вывести результат с помощью TextBox.

Содержание отчета

В отчете указать цель работы, привести листинг разработанной программы и формы с результатами работы.

Контрольные вопросы

1. Способы доступа к данным.
2. Методы доступа к данным.
3. Команды доступа, назначение и реализация.
4. Объект DataCommand.
5. Объект DataReader.

Лабораторная работа №3

Доступ к данным в Windows-формах

Цель работы: получить навык построения WindowsForm-приложения с возможностью доступа к данным. LINQ. Entity Framework 4.5.

Краткие сведения из теории

Механизм привязки данных в ADO.NET должен поддерживать управляемые данными настольные (Windows) приложения и Web-приложения. С этой целью механизм привязки данных позволяет связать с источником данных любое свойство элемента управления, предназначенное для записи.

Windows-формы поддерживают два типа привязки данных. Для элементов управления, содержащих единственное значение (таких, как TextBox, CheckBox и т.п.), Windows–формы поддерживают простую привязку данных, а для элементов управления, содержащих несколько значений (таких, как ListBox, ComboBox, DataGrid и т.п.) – сложную привязку данных.

Простая привязка данных

Свойство *DataBinding* описывает привязку данных к определенному свойству элементов управления. Например, значение свойства *Text* может определяться данными из одной таблицы, а цвет текста элемента управления – данными из другой таблицы. Для того, чтобы создать объект привязки данных, необходимо выполнить следующие действия:

- Указать свойство элемента управления, к которому необходимо осуществить привязку;
- Указать источник данных;
- Указать при необходимости часть источника данных.

В приложениях ADO.NET свойство *DataSource* обычно указывает на объект *DataTable* или *DataView,* а свойство *DataMember* – на столбец, данные которого необходимо извлечь. Создание объекта привязки приобретает следующий формат:

{Элемент_управления}.DataBindings.Add(“{Свойство}“,

{DataSource},”{DataMember}”);

Привязка текстового поля осуществляется следующим образом:

textBox1.DataBindings.Add(“Text”,ds1.Tables[“Таб_студ”],”фамилия”);

Здесь в тестовое поле будет помещено значение поля *«фамилия»* первой записи таблицы *«Таб_студ»* набора данных *ds1*.

Для привязки данных к списку необходимо определить значения трех свойств объекта ListBox:

DataSource – экземпляр класса, реализующего интерфейс List (например, класс *DataTable*);

DisplayMember –свойство объекта – источника (*DataSource*), которое будет отображаться в элементе управления;

ValueMember – идентификатор, хранящийся в элементе управления и определяющий строку объекта – источника, к которой происходит обращение.

Аналогично формируется привязка данных к объекту ComboBox.

Значение поля, указанного в свойстве элемента управления *ValueMember*, соответствующее выбранному пользователем элементу, хранится в свойстве *SelectValue* элемента управления ListBox или ComboBox. Оно отличается от значения свойства *SelectItem*, которое хранит значение поля, указанного в свойстве *DisplayMember*. В листингах 3.1 и 3.2 приведены примеры привязки данных к списку и раскрывающемуся списку соответственно.

Листинг 3.1.

```
this.listBox1.DataSource=ds1.Tables[“Таб_студ”];
this.listBox1.DisplayMember=”фамилия”;
```

this.listBox1.ValueMember=”номер_студента”;

Листинг 3.2.

this.comboBox1.DataSource=ds1.Tables[“Таб_гр”];

this.comboBox1.DisplayMember=”группа”;

this.comboBox1.ValueMember=”группа”;

Привязка данных к элементу управления DataGrid отличается от привязки данных к остальным элементам управления Windows-формы. При простой привязке данных элементу управления ставится в соответствие атомарное значение, при сложной привязке данных – список значений. Особенность привязки данных к элементу управления DataGrid заключается в том, что разработчик имеет дело с более обширным множеством данных. Элемент управления DataGrid позволяет установить значение свойства *DataSource* равным фактическому источнику данных (такому, как объект *DataTable, DataSet, DataView* и т.п.). Если изменить значение свойства *DataSource* на этапе выполнения, оно вступит в силу только после перезагрузки элемента управления DataGrid, осуществляемой с помощью метода *DataDrid.SetDataBinding().* В качестве параметров метод *SetDataBinding()* принимает новые значения свойств *DataSoiurce* и *DataMember*. Значение свойства *DataMember* – это именованная сущность, с которой будет связан элемент управления DataGrid. Привязка данных к элементу управления DataGrid осуществляется с помощью следующего оператора:

this.dataGrid1.SetDataBinding(ds1,”Таб_студ”);

Создадим Windows-приложение, в котором реализованы следующие функции:

- По значению группы, выбранному в раскрывающемся списке, производится заполнение таблицы списком студентов;
- Список студентов можно редактировать: изменять значения полей, удалять и вносить новые записи;
- Необходимо иметь возможность сохранить в базе данных внесенные изменения или отменить изменения;
- Об успешном или неудачном сохранении изменений в базе данных пользователь должен быть уведомлен с помощью диалогового окна.

Внешний вид Windows-приложения может быть таким, как показано на рис.3.1. Программный код представлен в листинге 3.3.

Листинг 3.3.

```
public class Form1 : System.Windows.Forms.Form
{
    // объявление объектов
      System.Data.Odbc.OdbcConnection con1;
      System.Data.Odbc.OdbcDataAdapter da1;
      System.Data.Odbc.OdbcDataAdapter da2;
      System.Data.Odbc.OdbcCommand selCmd;
```

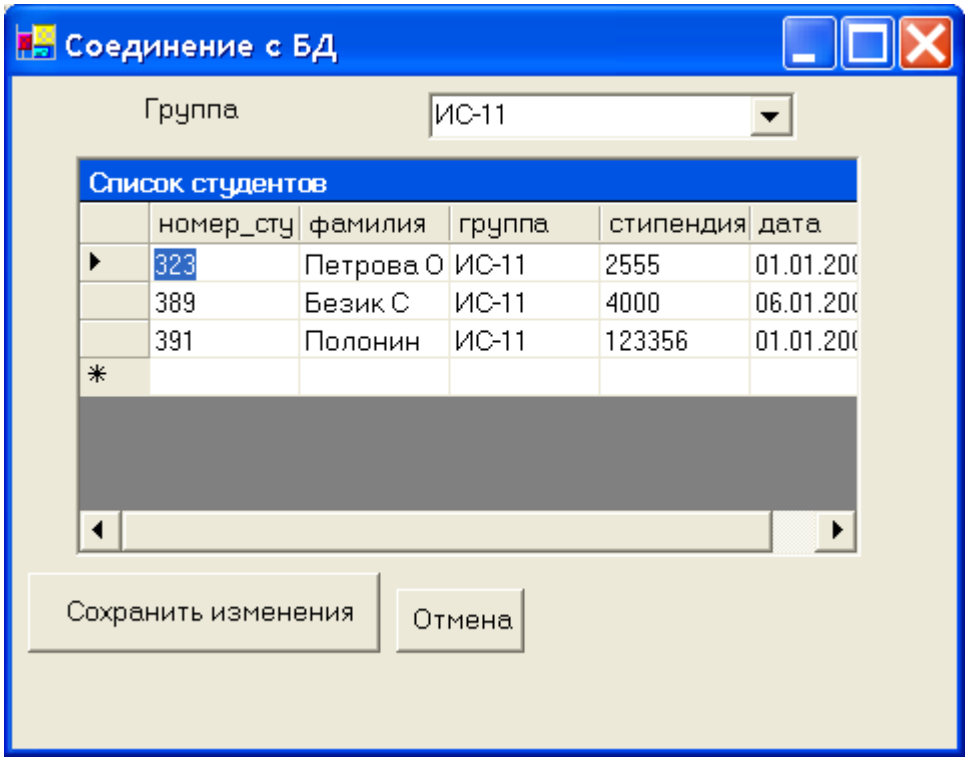

Рисунок 3.1 – Windows-форма приложения

```
      System.Data.Odbc.OdbcCommandBuilder b;
      System.Data.DataSet ds1;
      string selQry;
      …
private void Form1_Load(object sender, System.EventArgs e)
{
      // создание объектов
      con1=new System.Data.Odbc.OdbcConnection();
```

```
        da1= new System.Data.Odbc.OdbcDataAdapter();
        da2=new System.Data.Odbc.OdbcDataAdapter("SELECT группа
                                                FROM группа",con1);
        ds1=new DataSet();
        b=new System.Data.Odbc.OdbcCommandBuilder(da1);
        // команда выборки
        selCmd=new System.Data.Odbc.OdbcCommand();
        selQry="SELECT * FROM студент WHERE группа=?";
        selCmd.Parameters.Add("p0",System.Data.Odbc.OdbcType.Char,50, "группа");
        selCmd.Connection=con1;
        selCmd.CommandText=selQry;
        da1.SelectCommand=selCmd;

try
{
        con1.ConnectionString= "PWD=stud;DSN=stud";                    Messa-
geBox.Show("Успешное соединение ODBC!");
        da2.Fill(ds1,"Таб_гр");

        // заполнение раскрывающегося списка
        this.comboBox1.DataSource=ds1.Tables[«Таб_гр»];
        this.comboBox1.DisplayMember="группа";
        this.comboBox1.ValueMember=»группа»;
        // определение параметра
    selCmd.Parameters["p0"].Value=this.comboBox1.SelectedValue.ToString();
        da1.Fill(ds1,"Таб_студ");
        // заполнение таблицы
        this.dataGrid1.SetDataBinding(ds1,"Таб_студ");
}
catch(Exception ex)
{
```

```
        MessageBox.Show("Соединения ODBC нет!"+ex.Message);
        this.Close();
}
}
private void button1_Click(object sender, System.EventArgs e)
{
// кнопка «Сохранить изменения»
try
{
        // обновление базы данных
        da1.Update(ds1.Tables[«Таб_студ»]);
        MessageBox.Show(«Данные успешно сохранены!»);
        // обновление элемента управления в Windows-форме
        ds1.Tables["Таб_студ"].RejectChanges();
}
catch(Exception ex)
{
        MessageBox.Show(«Данные сохранить не удалось!»+ex.Message);
        ds1.Tables["Таб_студ"].Clear();
        da1.Fill(ds1,"Таб_студ");
}
}
private void comboBox1_SelectedIndexChanged(object sender,
                                                System.EventArgs e)
{
        // выбор значения в раскрывающемся списке
        selCmd.Parameters["p0"].Value=comboBox1.SelectedValue.ToString();
        ds1.Tables["Таб_студ"].RejectChanges();
}
private void button2_Click_1(object sender, System.EventArgs e)
{
```

```
// кнопка «Отменить изменения»
        ds1.Tables[«Таб_студ»].Clear();
        da1.Fill(ds1,"Таб_студ");
}
}
```

Задание на лабораторную работу

Изменить пользовательский интерфейс приложения, разработанное в лабораторной работе №2, заменив TextBox на ListBox, ComboBox, DataGrid.

Содержание отчета

В отчете указать цель работы, привести листинг разработанной программы и формы с результатами работы.

Контрольные вопросы

1. Способы вывода данных в Windows форме.
2. Отличия между ListBox, ComboBox, DataGrid.
3. Особенность привязки данных к элементу управления DataGrid.
4. Типы привязки данных Windows формы.

Лабораторная работа №4

Отсоединенный набор данных Dataset

Цель работы: получить навык использования отсоединенного набора данных Dataset

Краткие сведения из теории

Объект *DataSet* – это:

- Набор информации, извлеченной из базы данных; доступ к этому набору осуществляется в отсоединенном режиме;
- База данных, расположенная в памяти;
- Сложная реляционная структура данных со встроенной поддержкой XML-сериализации.

Роль объекта *DataSet* в ADO.NET заключается в предоставлении отсоединенного хранилища информации, извлеченной из базы данных, и в обеспечении для .NET возможностей базы данных, хранимой в памяти. Объект *DataSet* – это коллек-

ция структур данных, использующихся для организации отсоединенного хранилища информации.

Объект *DataSet* состоит из нескольких связанных друг с другом структур данных. Концептуально он представляет собой полный набор реляционной информации. Внутри объекта *Dataset* могут храниться пять объектов:

DataTable - набор данных, организованный в столбцы и строки;

DataRow – коллекция данных, которая представляет собой одну строку таблицы *DataTable*, объект *DataRow* является фактическим хранилищем данных;

DataColumn – коллекция правил, описывающая данные, которые можно хранить в объектах *DataRow*;

Constraint – данный объект используется для определения бизнес – правил объекта *DataTable* и задает допустимость хранения определенных данных в объекте *DataTable*;

DataRelation –описание связей между объектами *DataTable*.

Так как объект *DataSet* не связан с базой данных, его можно воспринимать как реляционное хранилище данных.

Данные, которые хранятся внутри объекта *DataSet*, содержат не только информацию, необходимую для поддержки отсоединенного кэша базы данных, но также предоставляют возможность перемещаться по нему как по некоторой иерархической структуре.

Основным предназначением объекта *DataSet* является хранение и изменение данных. Объекты *DataRow* являются основным хранилищем данных внутри объекта *DataSet*. Объект *DataRow* содержит массив значений, представляющих собой строку объекта *DataTable*. Объекты *DataRow* доступны из объекта *DataTable* через свойство *Rows*.

Заполнение объекта DataSet

В отличие от предыдущих уровней доступа к данным, объект *DataSet* не имеет никакого внутреннего представления об источнике данных. Таким образом, объект *DataSet* оказывается отсоединенным не только от данных, но и от информации об их происхождении. Следовательно, необходимо иметь объект, позволяющий соединить

объект *DataSet* и источник данных. Для заполнения объекта *DataSet* данными существует три метода:

- С помощью объекта *DataAdapter* (при этом информация, как правило, извлекается из базы данных);
- На основе XML-документа;
- Программным путем.

Выборка строки

Пусть в наборе данных с именем *ds1* существует две таблицы с именами *"Таб_студ"* и *"Таб_гр"*. В таблицы *"Таб_студ"* и *"Таб_гр"* с помощью объекта *DataAdapter* внесены записи таблиц *Студент* и *Группа*, расположенных на сервере базы данных. Пример выборки данных из объекта *DataSet* по номеру строки в текстовые поля представлен в листинге 4.1.

Листинг 4.1.

```
int n=10;
DataRow r=this.ds1.Tables["Таб_студ"].Rows[n];
textBox1.Text=r["номер_студента"].ToString();
textBox2.Text=r["фамилия"].ToString();
textBox3.Text=r["группа"].ToString();
textBox4.Text=r["стипендия"].ToString();
textBox5.Text=r["дата"].ToString();
```

Добавление строки

Для создания новой строки можно использовать соответствующие методы (*NewRow()* и *Add()*) объекта *DataTable*. Следует отметить, что метод *NewRow()* сам по себе не добавляет строку в объект *DataTable*. Для этого необходимо вызвать метод *Add()*, передав ему в качестве параметра объект строки. Пример добавления строки в объект DataSet приведен в листинге 4.2.

Листинг 4.2.

```
// вставить запись
DataRow r=ds1.Tables["Таб_студ"].NewRow();
    r["номер_студента"]=1;
    r["фамилия"]="Семенов";
```

```
    r["группа"]="ИС-41";
    r["стипендия"]=1500;
    r["дата"]="6.10.05";
ds1.Tables["Таб_студ"].Rows.Add(r);
```

Удаление строки

При использовании отсоединенных данных к удалению строки из коллекции предъявляется особое требование: строка должна продолжать существовать до тех пор, пока хранилище данных не будет обновлено с помощью объекта *DataSet*. Удаление строки может быть осуществлено, например, с помощью метода *Delete()* объекта *DataRow*. В этом случае строка удаляет себя сама. Пример удаления строки представлен в листинге 4.3.

Листинг 4.3.

```
int n=10;
DataRow r=this.ds1.Tables["Таб_студ"].Rows[n];
r.Delete();
```

При удалении строки можно воспользоваться диалоговым окном для подтверждения удаления, как показано в листинге 4.4.

Листинг 4.4.

```
int n=10;
DataRow r=this.ds1.Tables["Таб_студ"].Rows[n];
string s=MessageBox.Show("Удалить запись?","Внимание!",
System.Windows.Forms.MessageBoxButtons.OKCancel).ToString();
if (s=="OK")
{
    try
    {
        r.Delete();
        MessageBox.Show("Запись удалена!");
    }
    catch(Exception ex)
    {
```

```
        MessageBox.Show("Запись не удалена! "+ex.Message);
    }
}
```

Изменение строки

Индексаторы класса *DataRow* позволяют установить новые значения столбцов строки, например:

```
r["фамилия"]=”Иванов”;
```

Однако при определении нового значения столбца объект *DataRow* сгенерирует исключение в том случае, если это значение будет конфликтовать со свойством *DataType* объекта *DataColumn*.

Существуют ситуации, в которых изменения в конкретную строку *DataRow* необходимо вносить параллельно. Обычно это делается тогда, когда одно изменение приводит к нарушению некоторого ограничения или когда необходимо иметь возможность отмены изменений перед их внесением в базу данных. В этом случае используются методы *BeginEdit(), EndEdit()* и *CancelEdit()* класса *DataRow*. Как только будет вызван метод *BeginEdit()*, изменения перестанут отражаться на объекте *DataRow* до тех пор, пока не будет вызван метод *EndEdit()*. Если внесенные изменения окажутся ошибочными, можно вызвать метод *CancelEdit()*, который вернет строку в первоначальное состояние (состояние, в котором она находилась до вызова метода *BeginEdit()*). Пример изменения строки показан в листинге 4.5.

Листинг 4.5.

```
//изменить запись
int n=10;
DataRow r=this.ds1.Tables["Таб_студ"].Rows[n];
r.BeginEdit();
r["фамилия"]=”Иванов”;
r["группа"]=”ИС-11”;
r["стипендия"]=2000;
r["дата"]=”1.01.05”;
r.EndEdit();
```

Определение схемы объекта DataSet

Предназначение схемы базы данных заключается в определении правил, которым должна соответствовать хранящаяся в базе данных информация. Создание схемы объекта *DataSet* возможно программным путем.

Пример создания первичных ключей приведен в листинге 4.6. Здесь же показано создание отношения между родительской таблицей *"Таб_гр"* и дочерней таблицей *"Таб_студ"* по полю *"группа"*.

Листинг 4.6.

```
DataTable gr=ds.Tables["Таб_гр"];
DataTable st=ds.Tables["Таб_студ"];
gr.PrimaryKey=new DataColumn[]{gr.Columns["группа"]};
st.PrimaryKey=new DataColumn[]{st.Columns["номер_студента"]};
ds.Relations.Add("gr_st",ds.Tables["Таб_гр"].Columns["группа"],
                         ds.Tables["Таб_студ"].Columns["группа"],true);
```

Первым параметром в методе *Relations.Add()* является имя отношения. Следующие два параметра представляют собой имена столбцов родительской и дочерней таблиц соответственно. Наконец, последний параметр является булевой переменной, определяющей необходимость создания ограничений для этого отношения. При создании ограничения объекту *DataSet* сообщается, что каждое значение уникального ключа родительской таблицы должно присутствовать в дочернем объекте *DataTable*.

Задание на лабораторную работу

Разработать приложение с объектом *DataSet* и с изменением содержимого программным способом.

Содержание отчета

В отчете указать цель работы, привести листинг разработанной программы и формы с результатами работы.

Контрольные вопросы

1. Что такое *DataSet*?
2. Какие методы для заполнения объекта *DataSet* данными существуют?
3. В чем особенность программного метода заполнения?

4. Изменение строки с помощью индексатора.

Лабораторная работа № 5

Объект DataAdapter

Цель работы: получить навык управления данными с помощью объекта DataAdapter. LINQ. Entity Framework 4.5.

Краткие сведения из теории

Объект *DataAdapter* – один из важнейших объектов ADO.NET. Этот объект является посредником между источником данных и набором данных *DataSet*. В приложениях *DataAdapter* обеспечивает считывание информации из базы данных и пересылку ее в *DataSet*, возврат изменений, сделанных пользователем, в исходную базу данных. Задача модификации данных решается через использование команд на основе SQL-запросов и хранимых процедур. В Visual Studio имеется несколько типов адаптеров данных:

- *System.Data.Odbc.OdbcDataAdapter*- используется для работы с ODBC – источниками данных;
- *System.Data.OleDb.OleDbDataAdapter* –используется для работы с любым источником данных, доступным через OLE DB –провайдера;
- *System.Data.SqlClient.SqlDataAdapter* - используется для работы с данными, хранящимися в SQL Server.

Каждый объект *DataAdapter* обеспечивает обмен данными между одной таблицей источника данных (базы данных) и одним объектом *DataTable* в наборе данных *DataSet.* Если *DataSet* содержит несколько таблиц (объектов *DataTable*), то необходимо иметь и несколько адаптеров данных.

Когда требуется заполнить данными таблицу в *DataSet,* вызывается соответствующий метод (*Fill*) объекта *DataAdapter*, который выполняет SQL-запрос или хранимую процедуру. Точно также, когда необходимо модифицировать базу данных, вызывается соответствующий метод (*Update*) объекта *DataAdapter*, который вызывает на исполнение соответствующий SQL-запрос или хранимую процедуру. В ре-

зультате этого все изменения, внесенные пользователем в таблицы набора данных, будут возвращены в соответствующие таблицы базы данных.

Заполнение объекта DataSet

Рассмотрим использование объекта *DataAdapter* для заполнения объекта *DataSet* данными.

Объект *DataAdapter* является связующим звеном между объектом DataSet и хранилищем данных. С помощью объекта *DataAdapter* можно заполнить объект *DataSet* информацией из хранилища данных, а также обновлять хранилище данных на основе объекта *DataSet.* Фактически объект *DataSet* представляет собой метакоманду. Так, объект *DataAdapter* состоит из четырех объектов *Command*, каждый из которых выполняет определенную задачу по модификации данных в базе данных: *SelectCommand, InsertCommand, UpdateCommand* и *DeleteCommand.* Объект *DataAdapter* используется каждый раз, когда объект *DataSet* нуждается в непосредственном взаимодействии с источником данных. Один из наиболее важных моментов заключается в том, что объект *DataAdapter* содержит объекты *Command* для каждой из основных операций, производимых над базами данных. Основное предназначение объекта *DataAdapter* заключается в формировании «моста» между объектом *DataSet* и базой данных.

Еще одной важной задачей объекта *DataAdapter* является минимизация времени, в течение которого соединение будет оставаться открытым. При использовании объекта *DataAdapter* явного открытия или закрытия соединения не происходит. *DataAdapter* знает, что соединение должно быть как можно более коротким, и самостоятельно управляет его открытием и закрытием. Если использовать объект *DataAdapter* совместно с уже открытым соединением, то состояние соединения будет сохранено.

Для заполнения *DataSet* информацией из базы данных необходимо:

- Создать экземпляр класса *DataAdapter,* который является специализированным классом, выполняющим функцию контейнера для команд, осуществляющих чтение и запись информации в базу данных;
- Создать экземпляр класса *DataSet*; только что созданный объект *DataSet* является пустым;

- Вызвать метод *Fill()* объекта *DataAdapter* для заполнения объекта *DataSet* данными из запроса, определенного в объекте *Command*.

Пример заполнения набора данных *ds1* показан в листинге 5.1.

Листинг 5.1.

```
System.Data.Odbc.OdbcDataAdapter da1;
System.Data.Odbc.OdbcDataAdapter da2;
System.Data.DataSet ds1;
...
da1=new System.Data.Odbc.OdbcDataAdapter("SELECT * FROM студент",con1);
da2=new System.Data.Odbc.OdbcDataAdapter("SELECT группа FROM группа",con1);
...
da1.Fill(ds1,"Таб_студ"); //заполнение данными набора данных
da2.Fill(ds1,"Таб_гр");
...
da1.UPdate(ds1,"Таб_студ"); //сохранение изменений в БД
da2.Update(ds1,"Таб_гр");
```

Используя объект *DataAdapter*, можно читать, добавлять, модифицировать и удалять записи в источнике данных. Чтобы определить, как каждая из этих операций должна выполняться, *DataAdapet* поддерживает следующие четыре свойства:

- *SelectCommand* – описание команды, которая обеспечивает выборку нужной информации из базы данных;
- *InsertCommand* – описание команды, которая обеспечивает добавление записей в базу данных;
- *UpdateCommand* – описание команды, которая обеспечивает обновление записей в базе данных;
- *DeleteCommand* – описание команды, которая обеспечивает удаление записей из базы данных.

Каждая из этих команд реализована в виде SQL-запроса или хранимой процедуры, как показано в листинге 5.2.

Листинг 5.2.

```
System.Data.Odbc.OdbcConnection con1;
```

```
System.Data.Odbc.OdbcDataAdapter da1;
System.Data.Odbc.OdbcCommand sellCmd;
System.Data.Odbc.OdbcCommand delCmd;
System.Data.Odbc.OdbcCommand insCmd;
System.Data.Odbc.OdbcCommand updCmd;
System.Data.DataSet ds1;
string selQry,delQry,insQry,updQry;
…
con1=new System.Data.Odbc.OdbcConnection();
da1=new System.Data.Odbc.OdbcDataAdapter();
ds1=new DataSet();
// команда выборки
selCmd=new System.Data.Odbc.OdbcCommand();
selQry="SELECT номер_студента, фамилия, группа, стипендия, дата FROM студент";
selCmd.Connection=con1;
selCmd.CommandText=selQry;
da1.SelectCommand=selCmd;
// команда выборки
selCmd=new System.Data.Odbc.OdbcCommand();
selQry="SELECT номер_студента, фамилия, группа, стипендия, дата FROM студент WHERE группа=?";
selCmd.Connection=con1;
selCmd.CommandText=selQry;
selCmd.Parameters.Add("p0",System.Data.Odbc.OdbcType.Char,50, "группа");
selCmd.Parameters["p0"].Value="ИС-11";
da1.SelectCommand=selCmd;
// команда удаления
delCmd=new System.Data.Odbc.OdbcCommand();
delQry="DELETE FROM студент WHERE номер_студента=?";
delCmd.Connection=con1;
```

```
delCmd.CommandText=delQry;
delCmd.Parameters.Add("p1",System.Data.Odbc.OdbcType.Int,4,
                        "номер_студента");
da1.DeleteCommand=delCmd;

// команда вставки
insCmd=new System.Data.Odbc.OdbcCommand();
insQry="INSERT INTO студент VALUES(?,?,?,?,?)";
insCmd.Connection=con1;
insCmd.CommandText=insQry;
insCmd.Parameters.Add("p2",System.Data.Odbc.OdbcType.Int,4,
                        "номер_студента");
insCmd.Parameters.Add("p3",System.Data.Odbc.OdbcType.VarChar,50, "фамилия");
insCmd.Parameters.Add("p4",System.Data.Odbc.OdbcType.VarChar,50, "группа");
insCmd.Parameters.Add("p5",System.Data.Odbc.OdbcType.Int,4, "стипендия");
insCmd.Parameters.Add("p6",System.Data.Odbc.OdbcType.DateTime,4, "дата");
da1.InsertCommand=insCmd;
// команда изменения
updCmd=new System.Data.Odbc.OdbcCommand();
updQry="UPDATE студент SET фамилия=?, группа=?, стипендия=?,
        дата=?  WHERE номер_студента=?";
updCmd.Connection=con1;
updCmd.CommandText=updQry;
updCmd.Parameters.Add("p7",System.Data.Odbc.OdbcType.VarChar,50, "фамилия");
updCmd.Parameters.Add("p8",System.Data.Odbc.OdbcType.VarChar,50, "группа");
updCmd.Parameters.Add("p9",System.Data.Odbc.OdbcType.Int,4, "стипендия");
updCmd.Parameters.Add("p10",System.Data.Odbc.OdbcType.DateTime,4, "дата");
updCmd.Parameters.Add("p11",System.Data.Odbc.OdbcType.Int,4,
                        "номер_студента");
da1.UpdateCommand=updCmd;
```

Самым сложным в использовании отсоединенных данных является синхронизация копии *DataSet* и базы данных. Несмотря на то, что в ADO.NET сохранение информации в базе данных осуществляется достаточно просто, проблемы начинаются тогда, когда нужно обеспечить поддержку параллелизма.

Ранее уровни доступа к данным возлагали поддержку параллелизма целиком и полностью на саму базу данных. Разработчику достаточно было указать тип параллелизма, который необходимо поддерживать, после чего база данных брала на себя основную работу – блокировку строк по мере обращения к ним пользователей. Свойство *IsolationLevel* интерфейса *Transaction* определяет уровень изоляции транзакции. Несмотря на то, что транзакции можно использовать для облегчения обработки параллельных обращений, они жестко связаны с соединениями, и их нельзя применять при работе с отсоединенными данными. В этом случае параллелизм должен обрабатываться за пределами базы данных, что, однако, не означает полный отказ от поддержки параллелизма со стороны базы данных.

Параллелизм в ADO.NET

ADO.NET поддерживает три типа параллелизма:

- Оптимистический параллелизм – все пользователи могут получить доступ к данным объекта *DataSet* до тех пор, пока кто-либо из них не начнет осуществлять запись информации в базу данных. Это самая распространенная модель параллелизма в ADO.NET;
- Пессимистический параллелизм – нельзя получить доступ к данным объекта *DataSet*, пока кто-либо из пользователей владеет копией данных, в этом случае данные блокируются;
- Деструктивный параллелизм – все пользователи имеют доступ к объекту *DataSet,* но в базе данных будут зафиксированы только самые последние изменения, на самом деле это означает фактическое отсутствие контроля параллелизма.

ADO.NET поддерживает оптимистический параллелизм с помощью класса *CommandBuilder*. При вызове метода *DataAdapter.Update()*, адаптер анализирует соответствующую таблицу, вставляет новые записи, удаляет запись, предназначенную для удаления, а также проводит обновление заданной строки. Команды *INSERT,*

DELETE, UPDATE при этом не определяются, так как объект *CommandBuilder* создает их сам по мере необходимости.

Объект CommandBuilder

Класс *CommandBuilder* отвечает за генерацию запросов по мере возникновения необходимости в них в объекте *DataAdapter*. Каждый управляемый поставщик содержит собственную реализацию класса *CommandBuilder (SqlCommandBuilder, OleDbCommandBuilder, OdbcCommandBuilder).* После создания объекта *CommandBuilder* его необходимо передать в конструктор объекта *DataAdapter*. Как только объект *CommandBuilder* узнает об объекте *DataAdapter,* он использует свойство *DataAdapter. SelectCommand,* чтобы получить информацию о столбцах объекта *DataTable* и иметь возможность создать команды вставки, обновления и удаления данных. Для того, чтобы гарантировать нормальное функционирование объекта *CommandBuilder*, необходимо учесть несколько моментов.

Свойство *DataAdapter.SelectCommand* должно содержать действительную команду, которая использовалась для заполнения обновляемого объекта *DataTable*. Объект *CommandBuilder* применяет SQL- оператор SELECT для того, чтобы иметь возможность создавать операторы вставки, обновления и удаления данных.

Таблица *DataTable,* которая будет обновляться, должна либо содержать столбец уникальных значений, либо для нее должен быть определен первичный ключ.

Пример использования объекта CommandBuilder приведен в листинге 5.3.

Листинг 5.3.

```
// объявление объектов
System.Data.Odbc.OdbcConnection con1;
System.Data.Odbc.OdbcDataAdapter da1;
System.Data.Odbc.OdbcDataAdapter da2;
System.Data.Odbc.OdbcCommand selCmd;
System.Data.Odbc.OdbcCommandBuilder b;
System.Data.DataSet ds1;
string selQry;
//создание объектов
con1=new System.Data.Odbc.OdbcConnection();
```

```
da1= new System.Data.Odbc.OdbcDataAdapter();
da2=new System.Data.Odbc.OdbcDataAdapter("SELECT группа FROM группа",con1);
ds1=new DataSet();
b=new System.Data.Odbc.OdbcCommandBuilder(da1);
// команда выборки
selCmd=new System.Data.Odbc.OdbcCommand();
selCmd.Connection=con1;
selQry="SELECT * FROM студент WHERE группа=?";
selCmd.CommandText=selQry;
selCmd.Parameters.Add("p0",System.Data.Odbc.OdbcType.Char,50, "группа");
selCmd.Parameters["p0"].Value="ИС-11";
da1.SelectCommand=selCmd;
con1.ConnectionString= "UID=stud;PWD=stud;DSN=stud";
// заполнение набора данных
da2.Fill(ds1,"Таб_гр");
da1.Fill(ds1,"Таб_студ");
//обновление базы данных
da1.Update(ds1.Tables["Таб_студ"]);
```

Задание на лабораторную работу

Создать объекты DataAdapter и CommandBuilder для вставки, удаления, изменения и выборки данных с параметрами. В форме разместить раскрывающийся список и таблицу для данных. Таблицу заполнять данными в соответствии с параметром, определенным раскрывающимся списком. Предусмотреть кнопки для сохранения изменений в БД и отмены внесенных изменений.

Содержание отчета

В отчете указать цель работы, привести листинг разработанной программы и формы с результатами работы.

Контрольные вопросы

1. Как типы параллелизма поддерживает ADO.NET?
2. Свойства DataAdapet?
3. Какие типы адаптеров есть в Visual Studio?

4. Для чего нужен DataAdapet?
5. Для чего нужен CommandBuilder?

Лабораторная работа № 6

Типизированные классы DataSet

Цель работы: получить навык использования типизированных классов DataSet.

Краткие сведения из теории

Типизированный класс *DataSet* представляет собой набор классов, порожденных непосредственно от классов семейства *DataSet*. Для того, чтобы указать тип данных, который будет храниться в столбцах таблицы *DataTable*, нужно создать объект *DataColumn* для каждого столбца. Это гарантирует проверку типов на этапе выполнения, но не гарантирует подобную проверку на этапе написания кода. Типизированный класс *DataSet* предоставляет разработчику именно такую функциональность.

При создании и использовании типизированных классов следует обратить внимание на следующие моменты. Во-первых, типизированный объект *DataSet* создается точно так же, как и обычный объект *DataSet.* Разница заключается лишь в том, что в типизированном объекте *DataSet* уже существует схема. Во-вторых, хотя это и типизированный объект *DataSet*, соответствующий класс непосредственно наследует класс *DataSet.* Таким образом, типизированный объект *DataSet* можно заполнять с помощью объектов *DataAdapter*. На самом деле типизированный класс *DataSet* является просто специализированным классом *DataSet*. Следует отметить, что синтаксис доступа к полям и таблицам при использовании типизированного класса *DataSet* существенно упрощается. Теперь обращение к каждой таблице возможно с помощью свойств класса. Каждое поле, в свою очередь, является свойством строки. Кроме упрощения синтаксиса, разработчик получает возможность проверять правильность написания имен элементов типизированного класса *DataSet* на этапе компиляции.

Типизированный класс *DataSet* наряду с обычными столбцами может содержать вычисляемые столбцы, что также позволяет обеспечить безопасность типов. Наконец, использование типизированных классов *DataSet* в качестве основы для

уровней объектов данных или бизнес-объектов является чрезвычайно мощным инструментом. Непосредственное наследование от типизированного класса *DataSet* позволяет избавиться от необходимости самостоятельно разрабатывать эти уровни и обеспечивать безопасность типов.

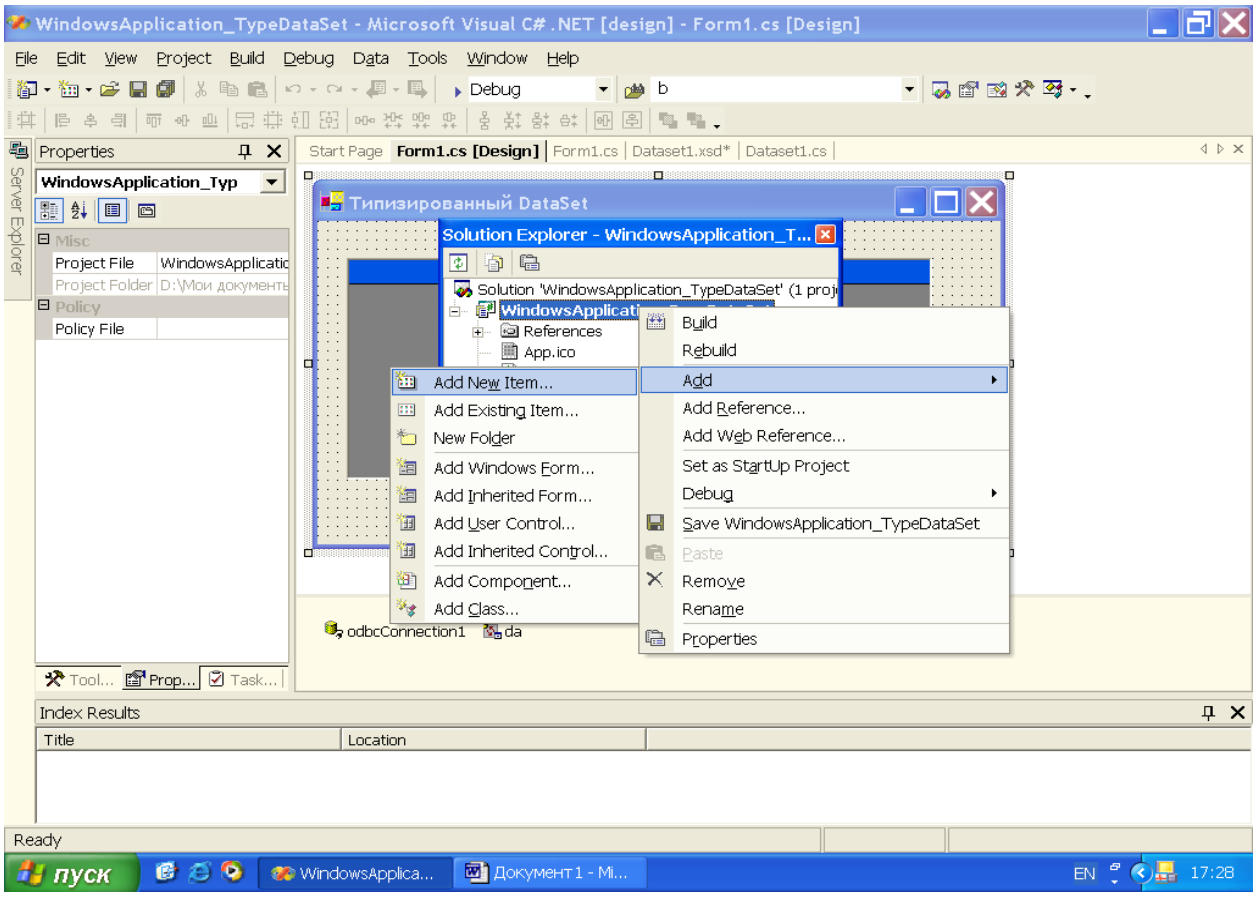

Рисунок 6.1 – Добавление нового элемента

Для создания типизированного класса необходимо выполнить следующие действия:

- В текущем проекте с помощью диалогового окна *Solution Explorer* (Обозреватель решения) добавить к проекту новый элемент (рисунок 6.1).
- Из вложенной в папку *Local Project Items* (Элементы локального проекта) папки *Data* (Данные) выбрать элемент *DataSet*. По умолчанию ему присваивается имя *Dataset1.xsd*. Типизированный класс имеет расширение *.xsd*. Это связано с тем, что его исходный код представляет собой XML-схему. Файл *.xsd* может включать в себя имена таблиц и столбцов, а также ключи, отношения и ограничения.

- После добавления к проекту типизированного класса *DataSet* можно с помощью диалогового окна *Server Explorer* (Обозреватель серверов) добавить к нему таблицы (рисунок 6.2), а с помощью панели *Toolbox* (Панель инструментов) – элементы схемы. После того как в окне *Server Explorer* будет выбрана существующая база данных, можно перетащить все необходимые таблицы в файл *.xsd*. На этом этапе новый типизированный класс *DataSet* включает в себя две таблицы, но между ними отсутствует отношение.

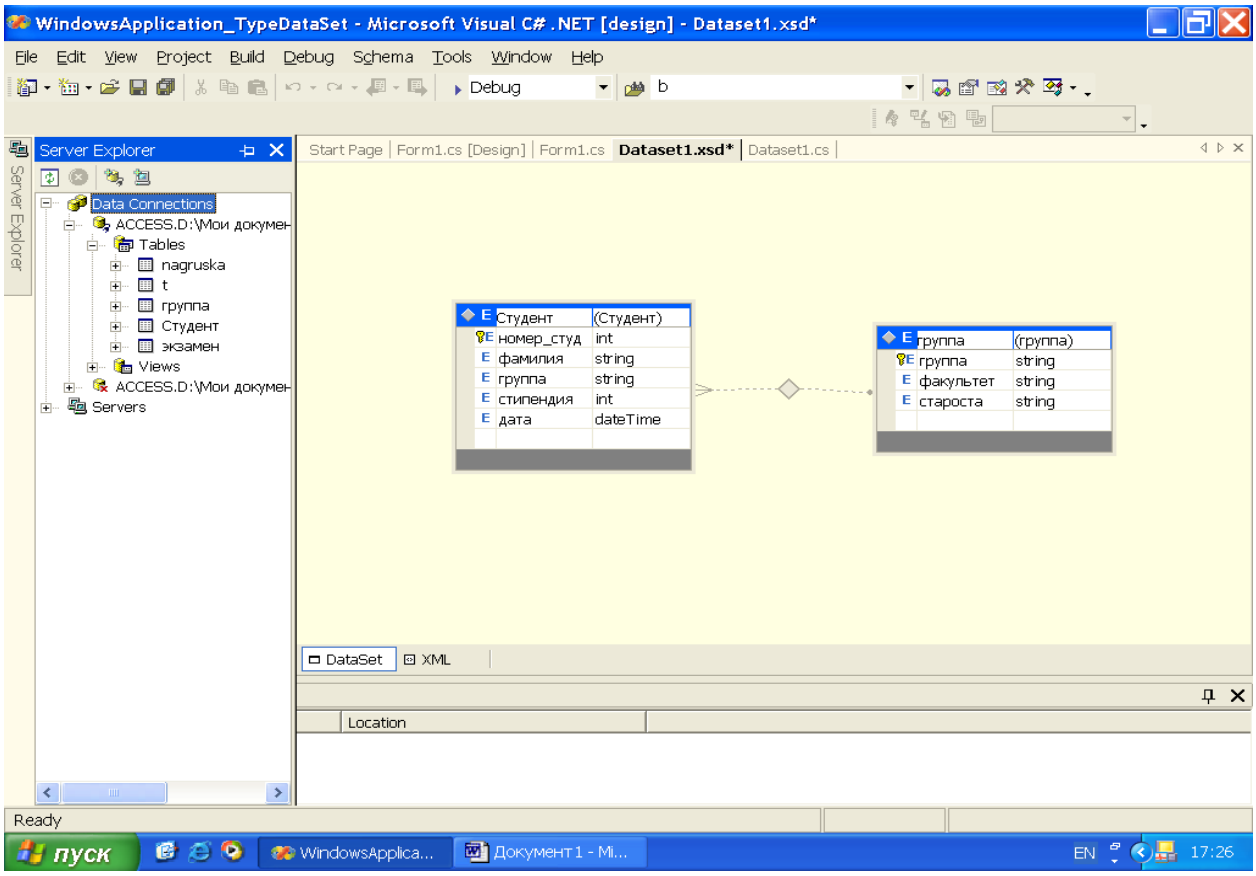

Рисунок 6.2 – Добавление таблицы

Для создания отношения в панели инструментов нужно выбрать элемент *Relation* и перетащить его в дочернюю таблицу типизированного набора. В диалоговом окне *Edit Relation* (рисунок 6.3) определяются параметры отношения: дочерняя и родительская таблица, поле внешнего ключа.

В листинге 6.1. приведен пример использования типизированного класса.

Листинг 6.1.

public class Form1 : System.Windows.Forms.Form

```
{
        System.Data.Odbc.OdbcConnection odbcConnection1;
        System.Data.Odbc.OdbcDataAdapter da;
        System.Data.Odbc.OdbcDataAdapter da1;
        System.Data.Odbc.OdbcCommandBuilder b;
        // объявление типизированного класса
        Dataset1 tds;
        private System.Windows.Forms.DataGrid dataGrid1;

private void InitializeComponent()
{
    this.odbcConnection1 = new System.Data.Odbc.OdbcConnection();
    this.da = new System.Data.Odbc.OdbcDataAdapter();
    // создание типизированного класса
    this.tds = new WindowsApplication_TypeDataSet.Dataset1();
    …
private void Form1_Load(object sender, System.EventArgs e)
{
da=new System.Data.Odbc.OdbcDataAdapter("SELECT * FROM студент",
this.odbcConnection1);
da1=new System.Data.Odbc.OdbcDataAdapter("SELECT * FROM группа",
this.odbcConnection1);
b=new System.Data.Odbc.OdbcCommandBuilder(da);
// заполнение типизированного класса
da1.Fill(tds,"группа");
da.Fill(tds,"Студент");
// использование  класса для заполнения элементов управления
this.dataGrid1.DataSource=tds.Студент;
this.label1.Text=tds.Студент[0].фамилия.ToString();
}
private void button1_Click(object sender, System.EventArgs e)
```

```
{
// использование типизированного класса в адаптере для сохранения
// изменений в источнике данных
        da.Update(tds,"Студент");
}}
```

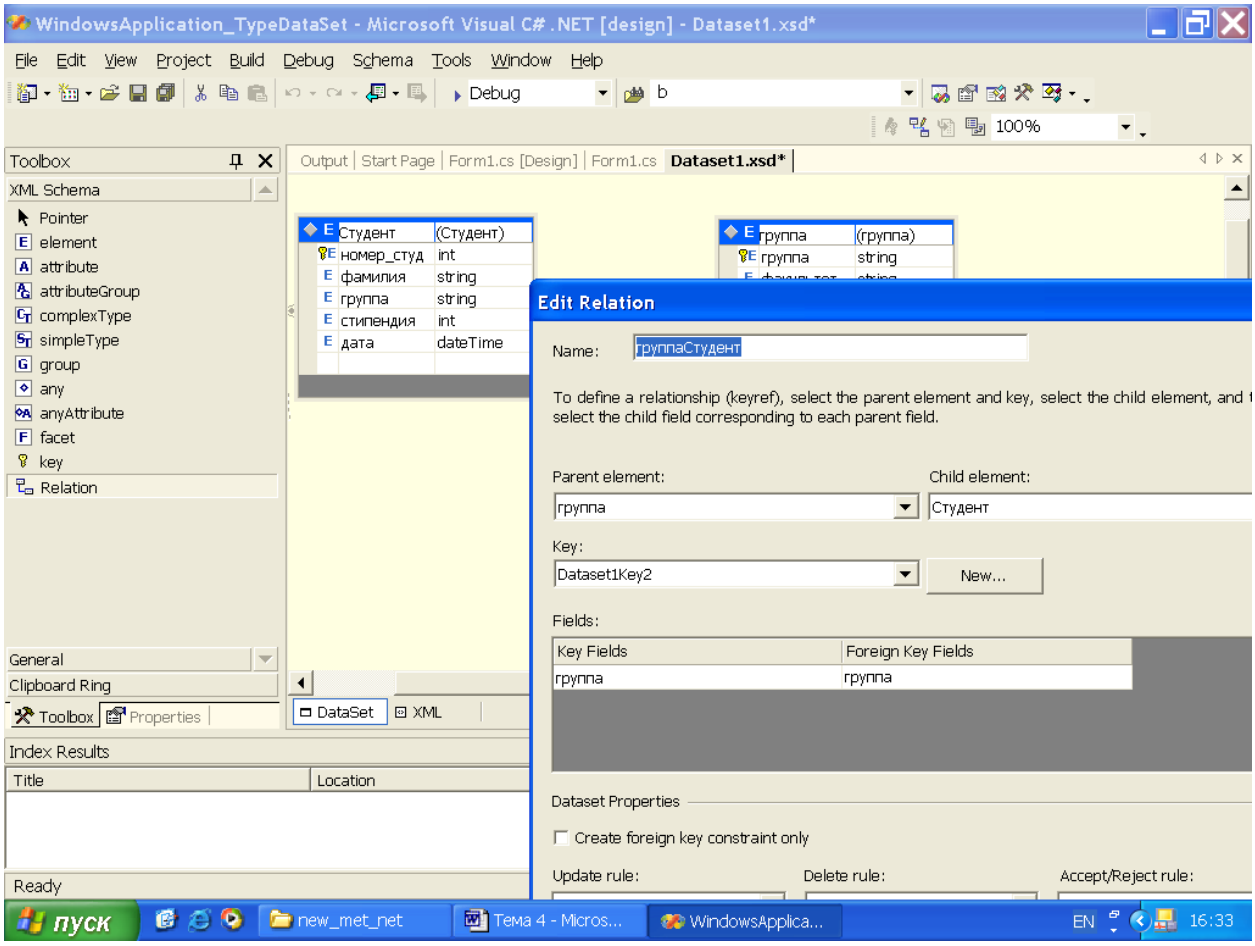

Рисунок 6.3 – Диалоговое окно *Edit Relation*

Задание на лабораторную работу

Создать типизированный набор данных, в Windows-форме разместить текстовые поля для отображения данных.

Содержание отчета

В отчете указать цель работы, привести листинг разработанной программы и формы с результатами работы.

Контрольные вопросы

1. Отличие типизированного от обычного объекта *DataSet*.
2. Для чего нужны вычисляемые столбцы?

3. Что такое «типизированный класс»?
4. В каких случаях используют типизированный класс?

Лабораторная работа № 7

Построение Windows-формы типа «родитель-потомок»

Цель работы: получить навык привязки данных типа «родитель-потомок»

Краткие сведения из теории

Привязка типа «родитель-потомок» предполагает привязку одного элемента управления к родительской таблице, а другого – к дочерней. Как показано на рисунке 7.1, верхняя сетка данных привязана к родительской таблице, а нижняя – к дочерней.

Пусть в наборе данных *ds* имеется две таблицы, родительская и дочерняя соответственно:

```
DataTable gr=ds.Tables["Таб_гр"];
DataTable st=ds.Tables["Таб_студ"];
```

Для того, чтобы сделать привязку данных, показанную на рисунке 7.1, необходимо связать родительскую таблицу с родительской сеткой данных, указав соответствующий объект *DataTable* в качестве значения свойства *DataSource*, но не указывая значение свойства *DataMember*:

```
this.dataGrid1.SetDataBinding(gr,"");
```

Затем нужно связать дочернюю таблицу с дочерней сеткой данных, указав родительский объект *DataTable* в качестве значения свойства *DataSource*, а имя отношения – в качестве значения свойства *DataMember:*

```
this.dataGrid2.SetDataBinding(gr,"gr_st");
```

Предварительно необходимо для каждой таблицы связи «родитель-потомок» в объекте *DataSe*t, если не используется типизированный набор данных, определить первичные ключи:

```
gr.PrimaryKey=new DataColumn[]{gr.Columns["группа"]};
st.PrimaryKey=new DataColumn[]{st.Columns["номер_студента"]};
```

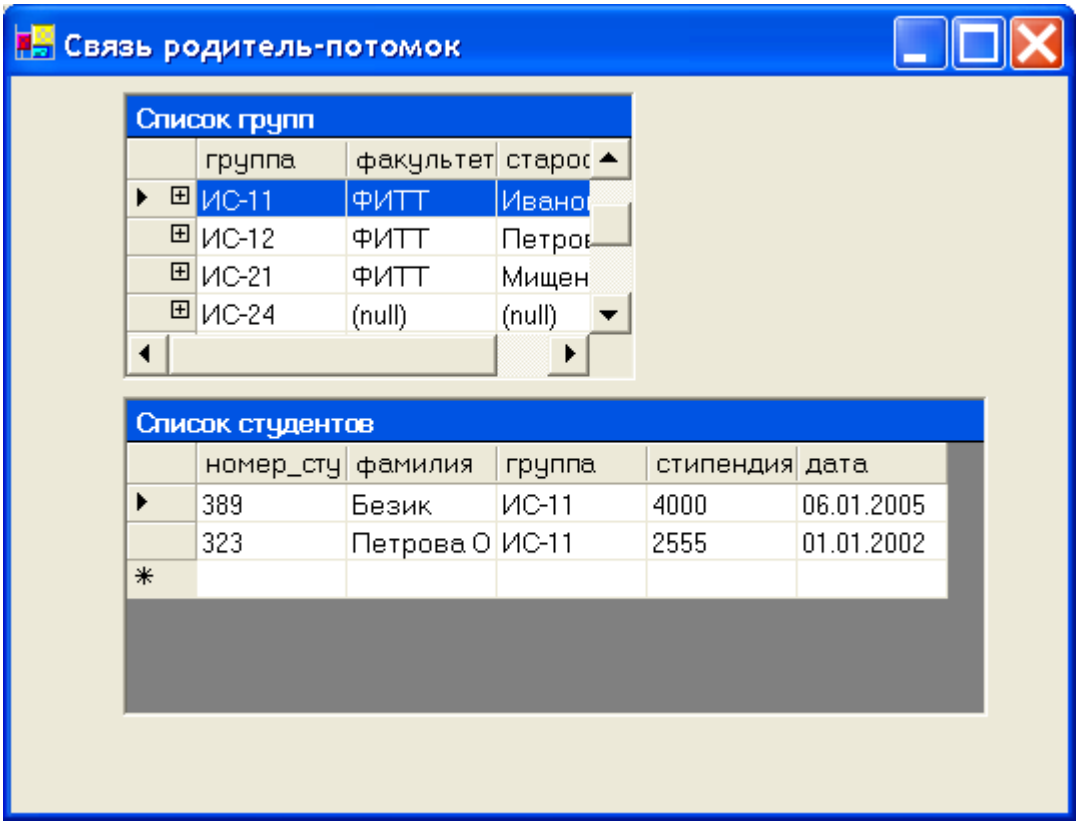

Рисунок 7.1 – Привязка к родительской и дочерней таблице

После определения первичных ключей необходимо сообщить объекту *DataSet* об отношениях, существующих между таблицами:

ds.Relations.Add("gr_st",ds.Tables["Таб_гр"].Columns["группа"],
ds.Tables["Таб_студ"].Columns["группа"],true);

Первым параметром в методе *Relations.Add()* является имя отношения. Следующие два параметра представляют собой имена столбцов родительской и дочерней таблиц соответственно. Наконец, последний параметр является булевой переменной, определяющей необходимость создания ограничений для этого отношения. При создании ограничения объекту *DataSet* сообщается, что каждое значение уникального ключа родительской таблицы должно присутствовать в дочернем объекте *DataTable*.

Пример привязки данных типа «родитель-потомок» в Windows-форме показан в листинге 7.1.

Листинг 7.1.

```
private System.Data.Odbc.OdbcConnection con;
private System.Data.Odbc.OdbcDataAdapter da1;
```

```
private System.Data.DataSet ds;
private System.Data.Odbc.OdbcDataAdapter da2;
private System.Windows.Forms.DataGrid dataGrid1;
private System.Windows.Forms.DataGrid dataGrid2;
…
this.con.ConnectionString = "PWD=stud;DSN=stud;UID=stud";
con.Open();
this.da1=new System.Data.Odbc.OdbcDataAdapter("SELECT * FROM группа
                                                ORDER BY группа", con);
this.ds=new DataSet();
da1.Fill(ds,"Таб_гр");
this.da2=new  System.Data.Odbc.OdbcDataAdapter("SELECT  номер_студента,  фами-
лия, группа,стипендия, дата FROM студент ORDER BY фамилия", con);
da2.Fill(ds,"Таб_студ");
DataTable gr=ds.Tables["Таб_гр"];
DataTable st=ds.Tables["Таб_студ"];
gr.PrimaryKey=new DataColumn[]{gr.Columns["группа"]};
st.PrimaryKey=new DataColumn[]{st.Columns["номер_студента"]};
ds.Relations.Add("gr_st",ds.Tables["Таб_гр"].Columns["группа"],
                        ds.Tables["Таб_студ"].Columns["группа"],true);
this.dataGrid1.SetDataBinding(gr,"");
this.dataGrid2.SetDataBinding(gr,"gr_st");
```

Пример Windows-формы с навигационной панелью показан на рисунке 7.2.

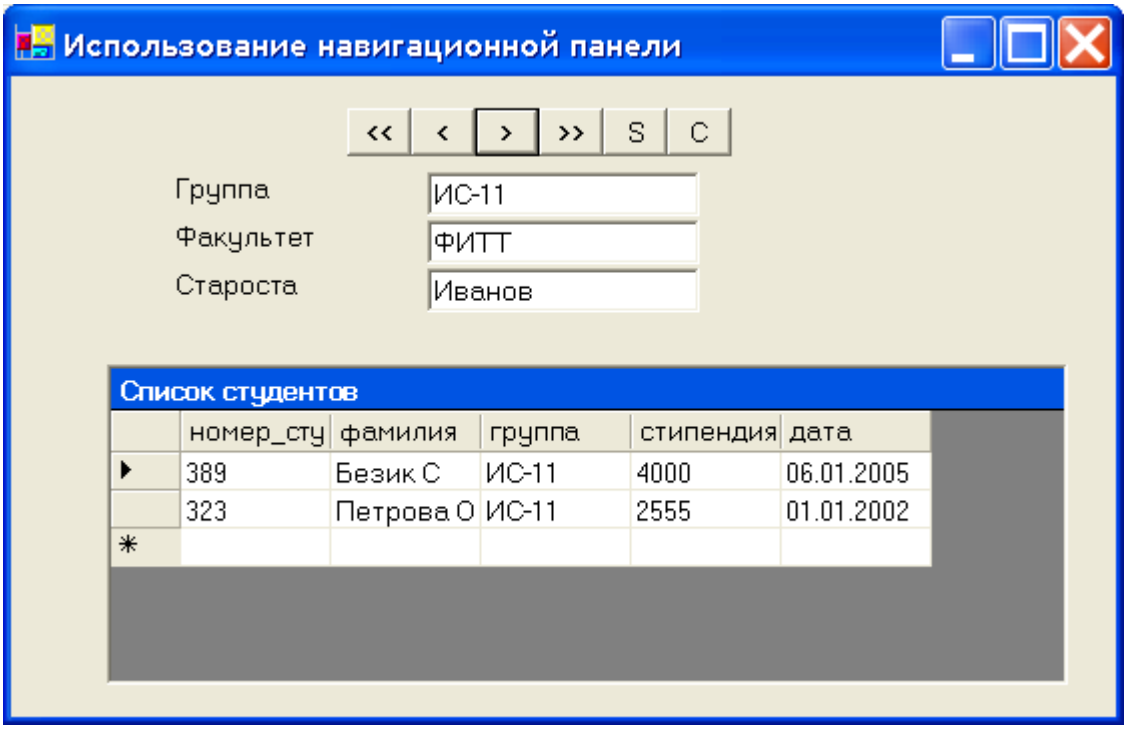

Рисунок 7.2 -

В этом приложении используется свойство формы *this.BindingContext*, связанное с определенным объектом данных и имеющее значение *Position*, которое позволяет изменить положение курсора в наборе данных. Например, установить курсор на первую запись в объекте *DataTable* с именем Таб_гр:

gr=ds.Tables["Таб_гр"];

…

this.BindingContext[gr].Position=0;

Пример привязки данных типа «родитель-потомок» в Windows-форме с использованием навигационной панели показан в листинге 7.2.

Листинг 7.2.

```
private System.Data.Odbc.OdbcConnection con;
private System.Data.Odbc.OdbcDataAdapter da1;
private System.Data.DataSet ds;
private System.Data.Odbc.OdbcDataAdapter da2;
private System.Data.Odbc.OdbcCommandBuilder bldr;
private System.Data.DataTable gr;
private System.Data.DataTable st;
…
private void Form1_Load(object sender, System.EventArgs e)
```

```
{
        this.con.ConnectionString = "PWD=stud;DSN=stud;UID=stud";
        this.da1=new System.Data.Odbc.OdbcDataAdapter"SELECT группа,
              факультет, староста FROM группа ORDER BY группа", con);
        this.ds=new DataSet();
        da1.Fill(ds,"Таб_гр");
        this.da2=new System.Data.Odbc.OdbcDataAdapter("SELECT
              номер_студента,фамилия, группа,стипендия, дата
              FROM студент ORDER BY фамилия", con);
        da2.Fill(ds,"Таб_студ");
        gr=ds.Tables["Таб_гр"];
        st=ds.Tables["Таб_студ"];
        gr.PrimaryKey=new DataColumn[]{gr.Columns["группа"]};
        ds.Relations.Add("gr_st",ds.Tables["Таб_гр"].Columns["группа"],
                    ds.Tables["Таб_студ"].Columns["группа"],true);
        bldr=new System.Data.Odbc.OdbcCommandBuilder(da2);
        this.textBox1.DataBindings.Add("Text",gr,"группа");
        this.textBox2.DataBindings.Add("Text",gr,"факультет");
        this.textBox3.DataBindings.Add("Text",gr,"староста");
        this.dataGrid1.SetDataBinding(gr,"gr_st");
}
private void button4_Click(object sender, System.EventArgs e)
{
        // на последнюю запись
        this.BindingContext[gr].Position=this.BindingContext[gr].Count-1;
}
private void button1_Click(object sender, System.EventArgs e)
{
        // на первую запись
        this.BindingContext[gr].Position=0;
}
```

```
private void button2_Click(object sender, System.EventArgs e)
{
        // на предыдущую запись
        if (this.BindingContext[gr].Position!=0)
                this.BindingContext[gr].Position-=1;
}
private void button3_Click(object sender, System.EventArgs e)
{
        // на следующую запись
        if (this.BindingContext[gr].Position<this.BindingContext[gr].Count-1)
                this.BindingContext[gr].Position+=1;
}
private void button5_Click(object sender, System.EventArgs e)
{
        // сохранить изменения в БД
        try
        {
                this.da2.Update(ds.Tables["Таб_студ"]);
                MessageBox.Show("Изменения успешно сохранены");
        }
        catch(Exception ex)
        {
                MessageBox.Show("Сохранение невозможно"+ex.Message);
                ds.Tables["Таб_студ"].Clear();
                da2.Fill(ds,"Таб_студ");
        }
}
private void button6_Click(object sender, System.EventArgs e)
{
        // отменить внесенные изменения
        ds.Tables["Таб_студ"].Clear();
```

```
        da2.Fill(ds,"Таб_студ");
}
}
```

Задание на лабораторную работу

Создать типизированный набор данных, в Windows-форме разместить две таблицы для отображения связанных данных. Перемещение по записям осуществлять с помощью навигационной панели.

Содержание отчета

В отчете указать цель работы, привести листинг разработанной программы и формы с результатами работы.

Контрольные вопросы

1. Для чего нужна привязка типа «родитель-потомок»?
2. Как организовать связь таблиц на форме Windiws?
3. Родительская и дочерняя таблицы.
4. Как использовать навигационную панель?

Лабораторная работа № 8

Построение Web-формы

Цель работы: получить навык привязки данных к ASP.NET и использования механизмов аутентификации

Краткие сведения из теории

При работе с Web- и ASP.NET-приложениями большое значение имеет отсоединенная природа протокола HTTP. Привязка данных к элементам управления Web-форм является отсоединенной и осуществляется в режиме только для чтения, что подчеркивает ее отсоединенную природу. Это не похоже на привязку данных к элементам управления Windows-форм, характеризующуюся непосредственной связью с источником данных. Элементы управления Windows-форм связываются с реальными данными (изменения в элементе управления влекут за собой изменения данных), в то время как элементы управления Web-форм – с их копией. Для того, чтобы осуществить привязку данных, необходимо вызвать метод *DataBind()* соответствующего элемента управления Web-формы. Вызов метода *DataBind()* элемента управления

Web-формы, размещенного на ASP.NET-странице, приведет к осуществлению фактической привязки данных к этому элементу управления. Для того, чтобы связать с данными все элементы управления ASP.NET-страницы, следует вызвать метод *DataBind()* для всей страницы.

Пример привязки данных объекта DataReader к списку показан в листинге 8.1.

Листинг 8.1.

```
protected System.Data.SqlClient.SqlCommand cmd;
protected System.Data.SqlClient.SqlDataReader rdr;
...
cmd.CommandText="SELECT номер_студента, фамилия FROM студент";
cmd.Connection=this.sqlConnection1;
sqlConnection1.Open();
try
{
    rdr=cmd.ExecuteReader();
}
catch(Exception ex)
{
    this.Label1.Text=ex.Message;
}
this.ListBox1.DataSource=rdr;
this.ListBox1.DataTextField="фамилия";
this.ListBox1.DataValueField="номер_студента";
this.ListBox1.DataBind();
```

Пример привязки полей первой строки таблицы *"Таб_студ"* объекта DataSet к текстовым полям показан в листинге 8.2.

Листинг 8.2.

```
string selQry="SELECT * FROM студент";
selCmd.Connection=this.sqlConnection1;
selCmd.CommandText=selQry;
da1.SelectCommand=selCmd;
```

```
da1.Fill(ds1,"Таб_студ");
...
this.TextBox1.Text=ds1.Tables["Таб_студ"].
                        Rows[0]["номер_студента"].ToString();
this.TextBox2.Text=ds1.Tables["Таб_студ"].
                        Rows[0]["фамилия"].ToString();
this.TextBox3.Text=ds1.Tables["Таб_студ"].Rows[0][2].ToString();
this.TextBox4.Text=ds1.Tables["Таб_студ"].Rows[0][3].ToString();
this.TextBox5.Text=ds1.Tables["Таб_студ"].Rows[0][4].ToString();
```

В листинге 8.3. приведен пример привязки данных объекта DataSet к таблице и раскрывающемуся списку.

Листинг 8.3.

```
this.DataGrid1.DataSource=ds1.Tables["Таб_студ"];
DataBind();
...
this.DropDownList1.DataSource=ds1.Tables["Таб_студ"];
this.DropDownList1.DataTextField="фамилия";
this.DropDownList1.DataValueField="номер_студента";
this.DropDownList1.DataBind();
```

Рассмотрим пример создания Web-приложения (рисунок 8.1), в котором реализованы следующие функции:

- Раскрывающийся список позволяет выбрать название группы;
- По значению раскрывающегося списка заполняется таблица;
- В таблице возможно редактирование и удаление строки;
- В режиме редактирования (рисунок 8.2) исключена возможность редактирования поля первичного ключа, в данном случае номера студента;
- Кнопка «Добавить» позволяет вставить в таблицу пустую строку, которую впоследствии можно отредактировать.

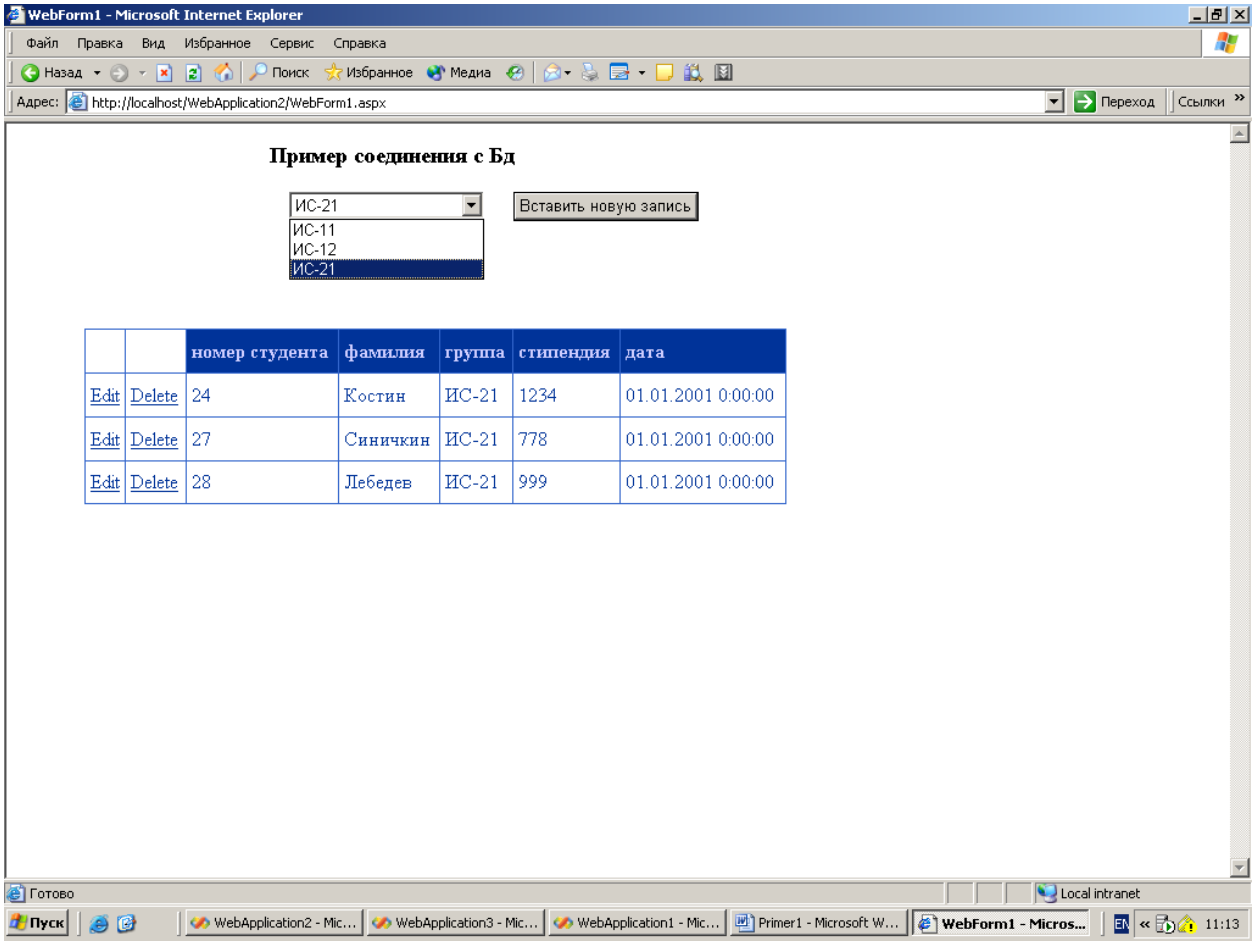

Рисунок 8.1 – Разрабатываемое Web-приложение

Для решения поставленной задачи необходимо определить ряд объектов для доступа к данным:

Объявить следующие объекты для доступа к данным:

protected System.Data.SqlClient.SqlConnection sqlConnection1;

protected System.Data.SqlClient.SqlDataAdapter da1;

protected System.Data.SqlClient.SqlDataAdapter da2;

protected System.Data.SqlClient.SqlCommandBuilder b1;

protected System.Data.DataSet ds1;

В метод *InitializeComponent()* добавить операторы создания объектов:

this.sqlConnection1 = new System.Data.SqlClient.SqlConnection();

this.sqlConnection1.ConnectionString = "user id=usera;

data source=\"ITS-SERVER\";initial catalog=basa_user; password =123";

this.ds1 = new System.Data.DataSet();

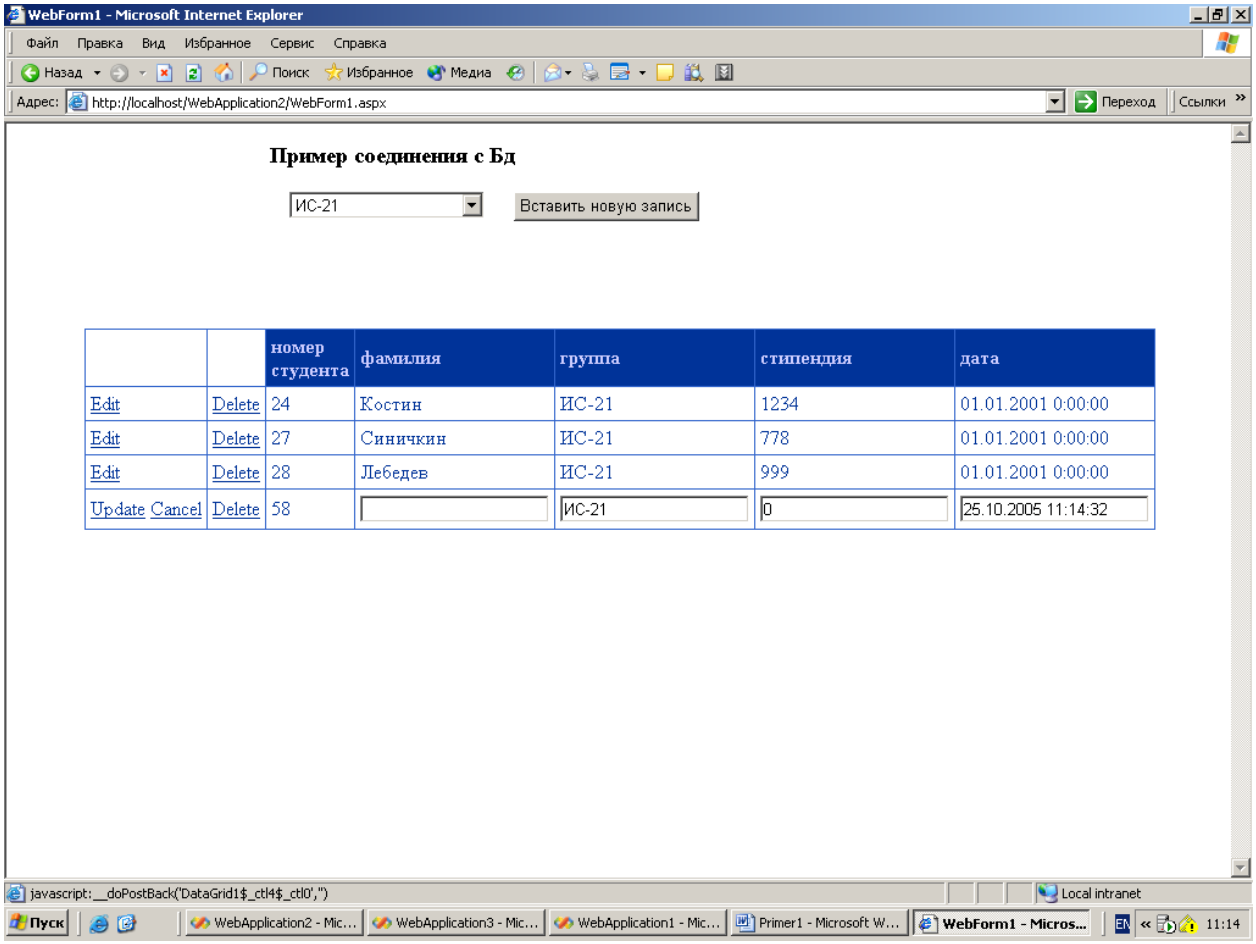

Рисунок 8.2 – Режим редактирования

В методе *Page_Load* создать адаптеры для заполнения данными раскрывающегося списка и элемента DataGrid, причем адаптер для заполнения таблицы содержит параметр, определяемый значением, выбранным в раскрывающемся списке. Объект *CommandBuilder* обеспечит обновление базы данных.

```
da1=new System.Data.SqlClient.SqlDataAdapter("SELECT DISTINCT
группа FROM студент ORDER BY группа",this.sqlConnection1);
da1.Fill(ds1,"tab1");
da2=new System.Data.SqlClient.SqlDataAdapter("SELECT * FROM
                студент where группа=@p", this.sqlConnection1);
da2.SelectCommand.Parameters.Add("@p",
                        System.Data.SqlDbType.VarChar,50);
b1=new System.Data.SqlClient.SqlCommandBuilder(da2);
```

Для того, чтобы при редактировании не выводились элементы управления TextBox для полей, представляющих собой первичный ключ таблицы, необходимо в

таблице DataGrid использовать столбцы типа *BoundColumn*. Столбцы данного типа позволяют установить свойство *ReadOnly* для предотвращения возможности редактирования, а также предоставляют различные возможности для форматирования данных в элементе управления DataGrid.

Для конфигурирования объекта DataGrid необходимо выполнить следующие действия:

- Щелкнуть правой кнопкой мыши на элементе DataGrid и выбрать *Property Builder*.
- В диалоговом окне *DataGrid Property* на вкладке *Columns* выполнить следующие действия:
- Снять флажок *Create columns automatically at runtime*;
- Выбрать пункт *Bound Column* и добавить столбцы в правое окно кнопкой “>”;
- Определить название столбцов, соответствующие им поля исходной таблицы данных и другие свойства столбцов, например, *ReadOnly*, как показано на рисунке 8.3.
- Изменить внешний вид элемента DataGrid можно путем использования ссылки *Auto Format* в нижней части окна свойств *Properties.*

Итак, создана форма для того, чтобы отобразить содержимое таблицы базы данных в элементе управления DataGrid. Чтобы обеспечить возможность редактирования данных, нужно добавить кнопки *Edit* и *Delete* к каждой строке сетки данных. Когда пользовать нажмет кнопку *Edit*, DataGrid перейдет в режим редактирования строки. При этом в редактируемой строке появятся текстовые поля для редактирования данных. При переводе строки в режим редактирования кнопка *Edit* будет заменена двумя другими кнопками (*Update* и *Cancel*, рисунок 8.2).

Для того, чтобы отобразить кнопки Edit, Update, Cancel и Delete, необходимо добавить их к элементу управления DataGrid. Для придания функциональности данным кнопкам необходимо добавить обработчики событий, в которых следует реализовать следующие функции:

- Кнопка *Edit* переводит строку в режим редактирования;
- Кнопка *Cancel* возвращает строку к режиму визуального отображения информации, не внося изменений ни в сетку, ни в базу данных;

- Кнопка *Update* выполняет обновление базы данных (на основе информации текущей строки).

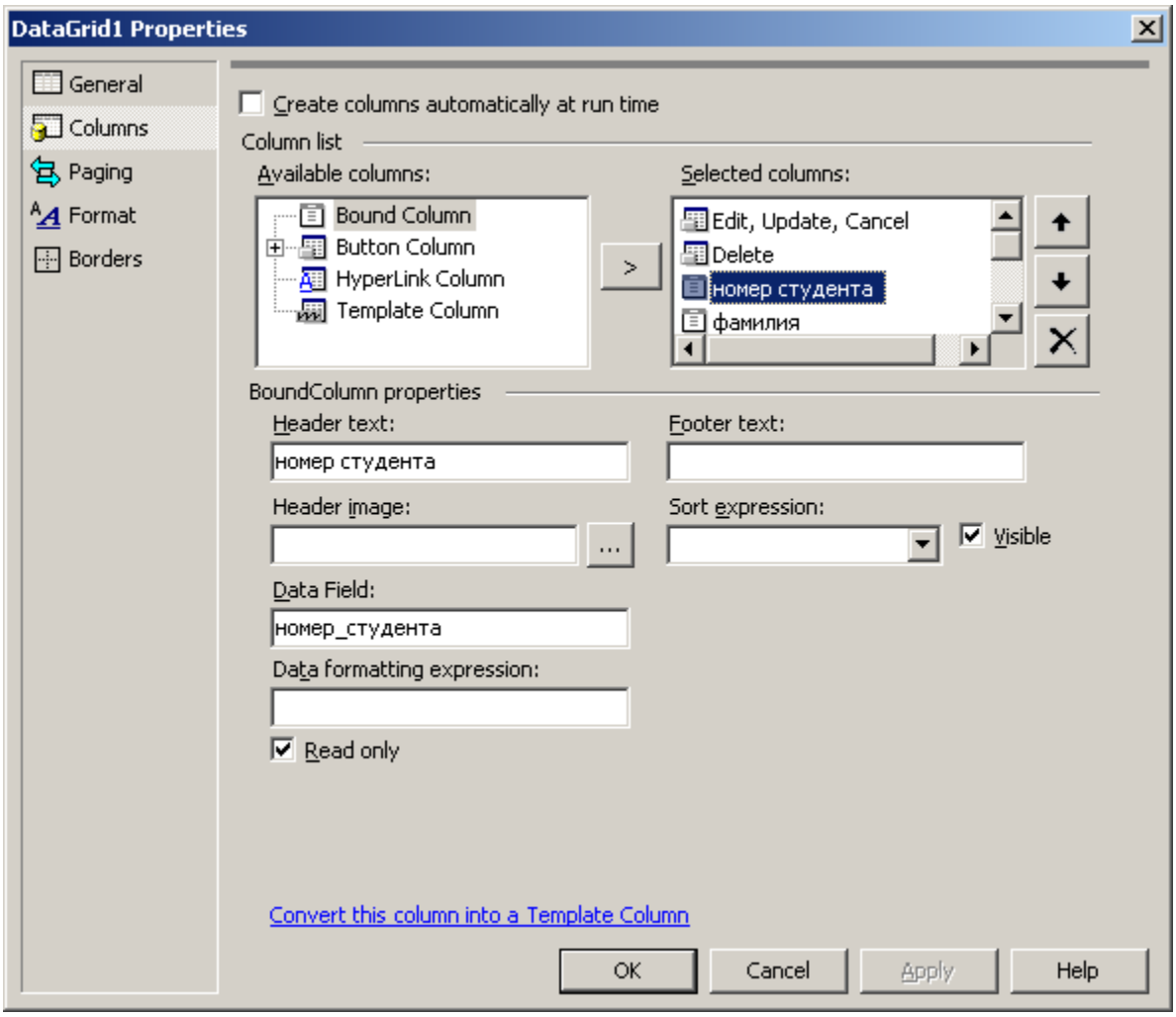

Рисунок 8.3 -

Для добавления возможности редактирования данных в DataGrid необходимо выполнить следующие действия:

- Выделить элемент управления DataGrid, нажать клавишу F4, чтобы отобразить окно свойств *Property*, затем нажать *Property Builder* в нижней части окна.
- В диалоговом окне *DataGrid Properties* перейти на вкладку *Columns.*
- Пролистать вниз список *Column list* до появления строки *Available columns* и открыть узел *Button Column.*
- Выбрать строку *Edit, Update, Cancel*, нажать кнопку >, чтобы добавить эти строки в список *Selected columns.*
- Сменить установленные по умолчанию надписи на кнопках *Edit, Update, Cancel* на Редактировать, Обновить, Отменить.

- В элементе управления DataGrid появится колонка с кнопками в виде ссылок. Первоначально видна только кнопка Редактировать (кнопки Обновить и отменить будут отсутствовать).

  Вид Web-формы в режиме конструктора представлен на рисунок 8.4.

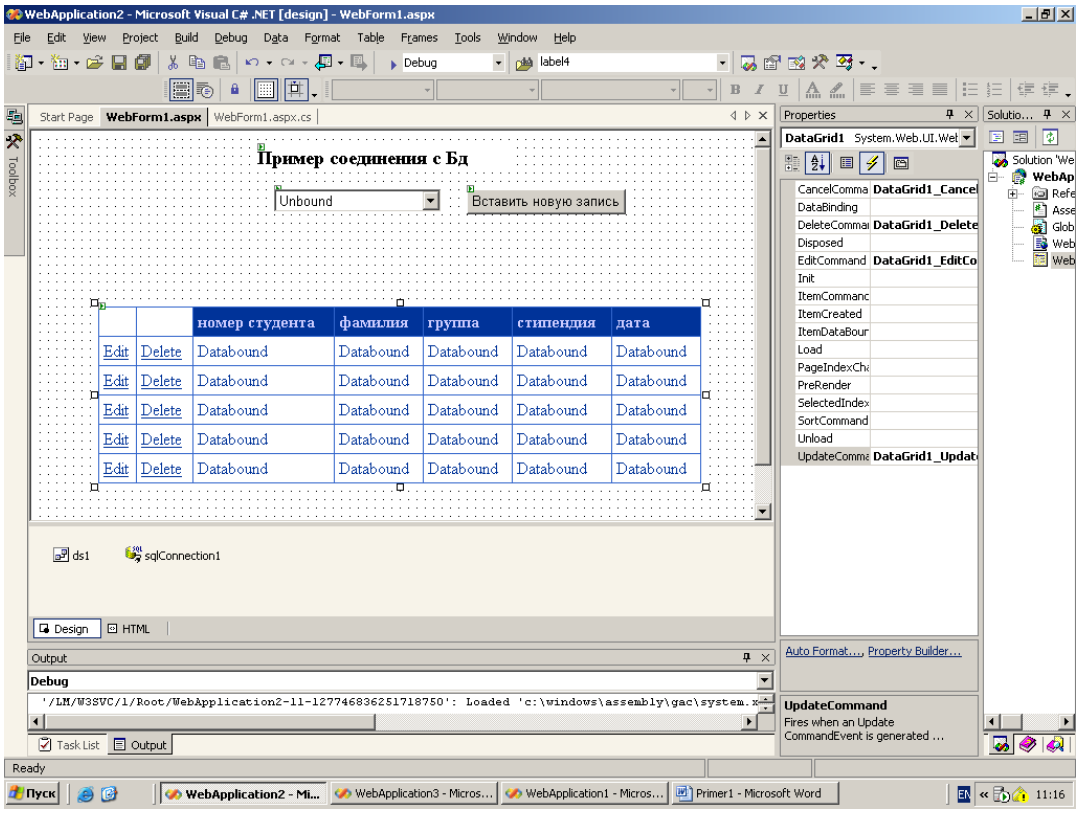

Рисунок 8.4 –

В приведенном примере раскрывающийся список отображает названия групп. Когда пользователь выбирает один из элементов списка, то есть когда происходит событие *SelectedIndexChanged,* необходимо отобразить записи таблицы Студент, соответствующие выбранной группе.

При создании раскрывающегося списка его свойству *AutoPostBack* присвоим значение *true*. Это означает, что страница будет возвращена на сервер, как только пользователь сделает выбор элемента из списка.

В обработчике события *Page_Load* нужно проверить значение свойства страницы *IsPostBack,* чтобы определить, загружается ли она первый раз.

Заполним раскрывающийся список:

```
if (!this.IsPostBack)
{
```

```
        this.DropDownList1.DataSource=ds1.Tables["tab1"];
        this.DropDownList1.DataTextField="группа";
        this.DropDownList1.DataBind();
        this.DropDownList1.AutoPostBack=true;
    }
```

Определим параметр и заполним элемент DataGrid:

```
    da2.SelectCommand.Parameters["@p"].Value=
                    this.DropDownList1. SelectedItem.ToString();
    da2.Fill(ds1,"tab2");
    this.DataGrid1.DataSource=ds1.Tables["tab2"];
    this.DataGrid1.DataKeyField="номер_студента";
    if (!this.IsPostBack)
    {
        this.DataGrid1.DataBind();
    }
```

Определить обработчик события для выбора элемента из раскрывающегося списка:

```
private void DropDownList1_SelectedIndexChanged(object sender, System.EventArgs e)
{
    da2.SelectCommand.Parameters["@p"].Value=
                    this.DropDownList1.SelectedItem.ToString();
    ds1.Tables["tab2"].RejectChanges();
    this.DataGrid1.DataBind();
}
```

Теперь, когда имеется кнопка *Edit,* необходимо написать обработчик события ее нажатия, чтобы перевести строку в режим редактирования. Для этого нужно свойству *EditItemIndex* элемента управления DataGrid присвоить индекс той строки, которую нужно редактировать (нумерация строк начинается с нуля). Чтобы вернуть строку в режим отображения данных свойству *EditItemIndex* присваивается значение -1. После смены режима редактирования необходимо повторно связать элемент управления DataGrid с источником данных. В режим редактирования нужно перевести ту строку, в которой пользователь нажал кнопку Edit (текущая строка). Номер

текущей строки можно получить из объекта обработчика события нажатия кнопки. В обработчике события имеется свойство *Item*, которое представляет всю строку DataGrid для модификации. Объект *Item*, в свою очередь, поддерживает набор различных свойств, включая и свойство *Item.ItemIndex*, которое содержит индекс той строки, с которой в данный момент работает пользователь.

Для перевода строки в режим редактирования необходимо создать обработчик события DataGrid1_EditCommand, для чего птребуется выполнить следующие действия:

- В режиме дизайнера формы выделить DataGrid, затем нажать клавишу F4, чтобы открыть окно Properties;
- Нажать кнопку Events в верхней части окна Properties;
- Дважды щелкнуть в строке EditCommand (или набрать текст DataGrid1_EditCommand).

Будет создан обработчик события DataGrid1_EditCommand. Добавим в обработчик события следующий код:

Листинг 8.4.

```
private void DataGrid1_EditCommand
(object source, System.Web.UI.WebControls.DataGridCommandEventArgs e)
        {
        //Обработчик события кнопки Редактировать
            this.DataGrid1.EditItemIndex=e.Item.ItemIndex;
            this.DataGrid1.DataBind();
        }
```

Аналогичным образом создается обработчик события DataGrid1_CancelCommand.

Листинг 8.5.

```
private void DataGrid1_CancelCommand
(object source, System.Web.UI.WebControls.DataGridCommandEventArgs e)
        {
        //Обработчик события кнопки Отменить
```

```
        this.DataGrid1.EditItemIndex=-1;
        this.DataGrid1.DataBind();
    }
```

Для того, чтобы из элемента управления DataGrid отправить в источник данных все изменения, сделанные пользователем, необходимо написать дополнительный программный код – обработку события нажатия кнопки Update. Обновление данных осуществляется в два этапа. На первом этапе изменения из элемента DataGrid переносятся в набор данных DataSet. На втором этапе изменения из набора данных передаются в базу данных. Когда строка DataGrid переходит в режим редактирования, в левом столбце появляется кнопка Update. Когда пользователь нажимает эту кнопку, генерируется событие UpdatetCommand. В обработчик этого события необходимо написать программный код, включающий следующие действия:

- Определить номер (индекс) той строки элемента управления DataGrid, в которую пользователь внес изменения.
- Получить измененные значения из идентифицированной строки DataGrid.
- Найти соответствующую строку в таблице набора данных и перенести в нее измененные значения из DataGrid. На этом шаге модифицируется набор данных, но эти изменения еще не переданы в базу данных.
- Передать все изменения из набора данных в базу данных.
- Обновить содержимое элемента управления DataGrid.

Для реализации первого шага нужно сначала получить номер измененной строки, для чего используется свойство *ItemIndex* объекта *Item*, сгенерированного событием *Edit (e.Item.ItemIndex):*

```
int index;
index=e.Item.ItemIndex;
DataRow r=ds1.Tables["tab2"].Rows[index];
```

Для реализации второго шага необходимо получить измененные значения из редактируемой строки DataGrid. Для этого нужно:

- В строке, которая была модифицирована (она передана в обработчик соответствующего события) перебрать все ячейки (их нумерация начинается с нуля).

Доступ к ячейке можно получить, задав ее индекс в коллекции *Cells*. Например, к крайней левой ячейке в строке можно обратиться *Cells(0);*

- Для каждой ячейки получить доступ к ее коллекции *Controls;*
- Для каждой ячейки получить значение из первого элемента коллекции *Controls* (в нашем случае этим элементом будет текстовое поле TextBox, в котором содержится отредактированное пользователем значение). Присвоить значение из элемента TextBox ячейки строки DataGrid любой переменной с типом TextBox;
- Запомнить полученные значения в переменных текстового типа.

```
TextBox tb;
tb=(TextBox)(e.Item.Cells[3].Controls[0]);
string v1=tb.Text;
```

Для реализации третьего шага необходимо найти соответствующую строку в таблице набора данных и обновить строку в наборе данных, изменяя значения столбцов в строке на полученные на шаге 2.

```
DataRow r=ds1.Tables["tab2"].Rows[index];
r.BeginEdit();
    r["фамилия"]=v1;
    r["группа"]=v2;
    r["стипендия"]=Convert.ToInt32(v3);
    r["дата"]=Convert.ToDateTime(v4);
r.EndEdit();
```

Для реализации четвертого шага необходимо передать изменения из набора данных в базу данных, вызвав метод Update адаптера данных.

```
da2.Update(ds1,"tab2");
```

Для реализации пятого шага необходимо переключить редактируемую строку в DataGrid в обычный режим отображения информации и обновить связь DataGrid с набором данных.

```
this.DataGrid1.EditItemIndex=-1;
this.DataGrid1.DataBind();
```

Полностью код обработчика события Обновить представлен в листинге 8.6.

Листинг 8.6.

```
private void DataGrid1_UpdatetCommand(
object source, System.Web.UI.WebControls.DataGridCommandEventArgs e)
        {
            // Обработчик события кнопки Обновить
            int index;
            index=e.Item.ItemIndex;
            DataRow r=ds4.Tables["tab4"].Rows[index];
            TextBox tb;
            tb=(TextBox)(e.Item.Cells[3].Controls[0]);
            string v1=tb.Text;
            tb=(TextBox)(e.Item.Cells[4].Controls[0]);
            string v2=tb.Text;
            tb=(TextBox)(e.Item.Cells[5].Controls[0]);
            string v3=tb.Text;
            tb=(TextBox)(e.Item.Cells[6].Controls[0]);
            string v4=tb.Text;
            r.BeginEdit();
                r["фамилия"]=v1;
                r["группа"]=v2;
                r["стипендия"]=Convert.ToInt32(v3);
                r["дата"]=Convert.ToDateTime(v4);
            r.EndEdit();

            da2.Update(ds1,"tab2");
            this.DataGrid1.EditItemIndex=-1;
            this.DataGrid1.DataBind();
        }
```

В программном коде обработчика события кнопки Удалить (листинг 8.7) определяется номер текущей строки. Строка с определенным номером удаляется из набора данных и из базы данных. Таблица в Web-форме обновляется.

Листинг 8.7.

```
private void DataGrid1_DeleteCommand(
object source, System.Web.UI.WebControls.DataGridCommandEventArgs e)
        {
        //Обработчик события кнопки Удалить
            int index;
            index=e.Item.ItemIndex;
            DataRow r=ds1.Tables["tab2"].Rows[index];
            r.Delete();
            this.DataGrid1.DataBind();
            da2.Update(ds1,"tab2");
        }
```

В программном коде обработчика события кнопки Добавить (листинг 8.8) к набору данных добавляется новая строка. Причем поля *"фамилия"* и *"стипендия"* остаются пустыми. Их можно заполнить в режиме редактирования. В поле *"группа"* помещается название группы, выбранной в раскрывающемся списке, а в поле *"дата"* записывается текущая дата. Таблица в Web-форме обновляется. Внесенные в набор данных изменения сохраняются в базе данных.

Листинг 8.8.

```
private void Button1_Click(object sender, System.EventArgs e)
    {
        //Обработчик события кнопки Добавить
        DataRow r=ds1.Tables["tab2"].NewRow();
        r["фамилия"]="";
        r["группа"]=this.DropDownList1.SelectedItem.ToString();
        r["стипендия"]=0;
        r["дата"]=System.DateTime.Now;
        ds1.Tables["tab2"].Rows.Add(r);
        da2.Update(ds1,"tab2");
        this.DataGrid1.DataBind();
    }
```

Следует отметить, что изменение набора данных и самой базы данных может быть выполнено с ошибками. Поэтому в программе следует предусмотреть перехват исключений с помощью блока *try….catch.*

Еще один пример Web-формы приведен на рисунке 8.5. В таблице представлена краткая информацию о студентах (номер и фамилия). Более полная информация о том или ином студенте (группа, стипендия, дата) появляется в надписях над таблицей при перемещении курсора по таблице.

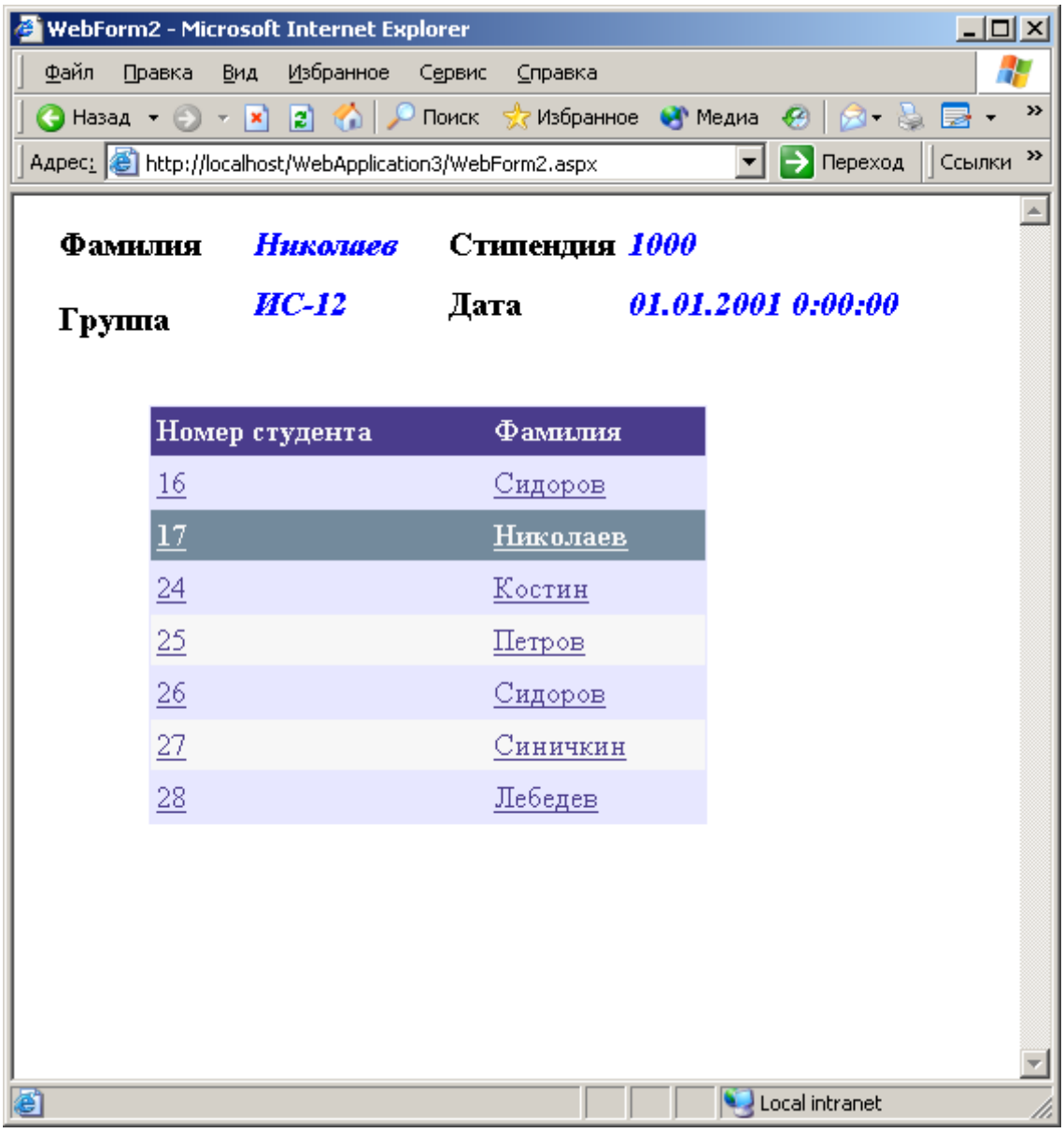

Рисунок 8.5 – Пример Web-формы с гиперссылками

Для создания подобного интерфейса сначала определим свойства объекта DataGrid. Для конфигурирования объекта DataGrid в режиме конструктора формы необходимо выполнить следующие действия:

- Щелкнуть правой кнопкой мыши на элементе DataGrid и выбрать *Property Builder.*
- В диалоговом окне *DataGrid Property* на вкладке *Columns* выполнить следующие действия:
- Снять флажок *Create columns automatically at runtime*;
- Пролистать вниз список *Column list* до появления строки *Available columns* и открыть узел *Button Column.*
- Выбрать строку *Select,* нажать кнопку «>», чтобы добавить эту строку в список *Selected columns.*
- Заполнить поля Header Text (Номер студента) и Text Field (номер_студента).

  Последние два шага повторить для поля «Фамилия».

Изменить внешний вид элемента DataGrid можно путем использования ссылки *Auto Format* в нижней части окна свойств *Properties.*

Далее добавим программный код, чтобы изменять текст надписей при изменении активной строки в DataGrid. Для этого необходимо дважды щелкнуть левой кнопки мыши на объекте DataGrid в режиме конструктора формы. При этом будет создан шаблон обработчика события смены строки. В обработчик данного события добавим код, представленный в листинге 8.9

Листинг 8.9.

```
private void DataGrid1_SelectedIndexChanged(object sender,
                                            System.EventArgs e)
{
    int index=DataGrid1.SelectedIndex;
    DataRow r=ds1.Tables["tab1"].Rows[index];
    this.Label2.Text=r["фамилия"].ToString();
    this.Label4.Text=r["группа"].ToString();
    this.Label6.Text=r["стипендия"].ToString();
    this.Label8.Text=r["дата"].ToString();
}
```

Аутентификация пользователя в Windows-приложении

Для ввода имени пользователя, пароля, имени источника данных (DSN) и других параметров соединения с базой данных в Windows-приложении создадим дополнительную форму в существующем проекте работы с БД. При запуске Windows-приложения сначала появится форма для ввода параметров соединения с БД, как показано на рисунке 8.6.

Для создания новой формы в текущем проекте достаточно поставить курсор на имя проекта в окне *Solution Explorer*, нажать правую кнопку мыши и выбрать пункт: *Add / Add Windows Form.* В новой форме разместим метки, текстовые поля и кнопку, как показано на рис.8.7.

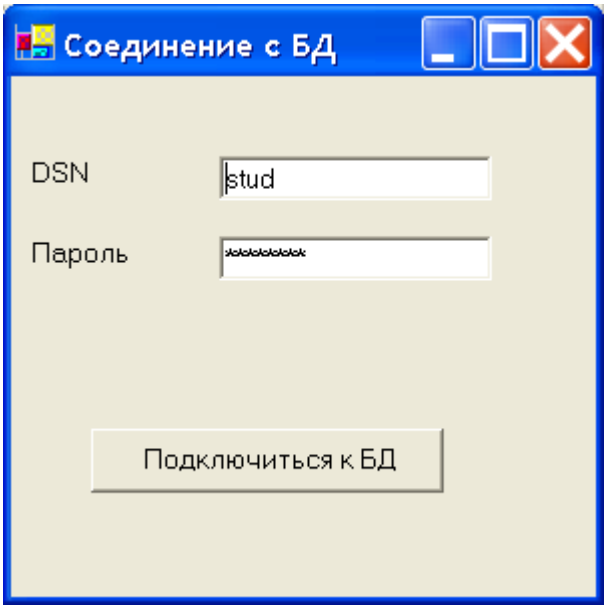

Рисунок 8.6 – Форма соединения с БД

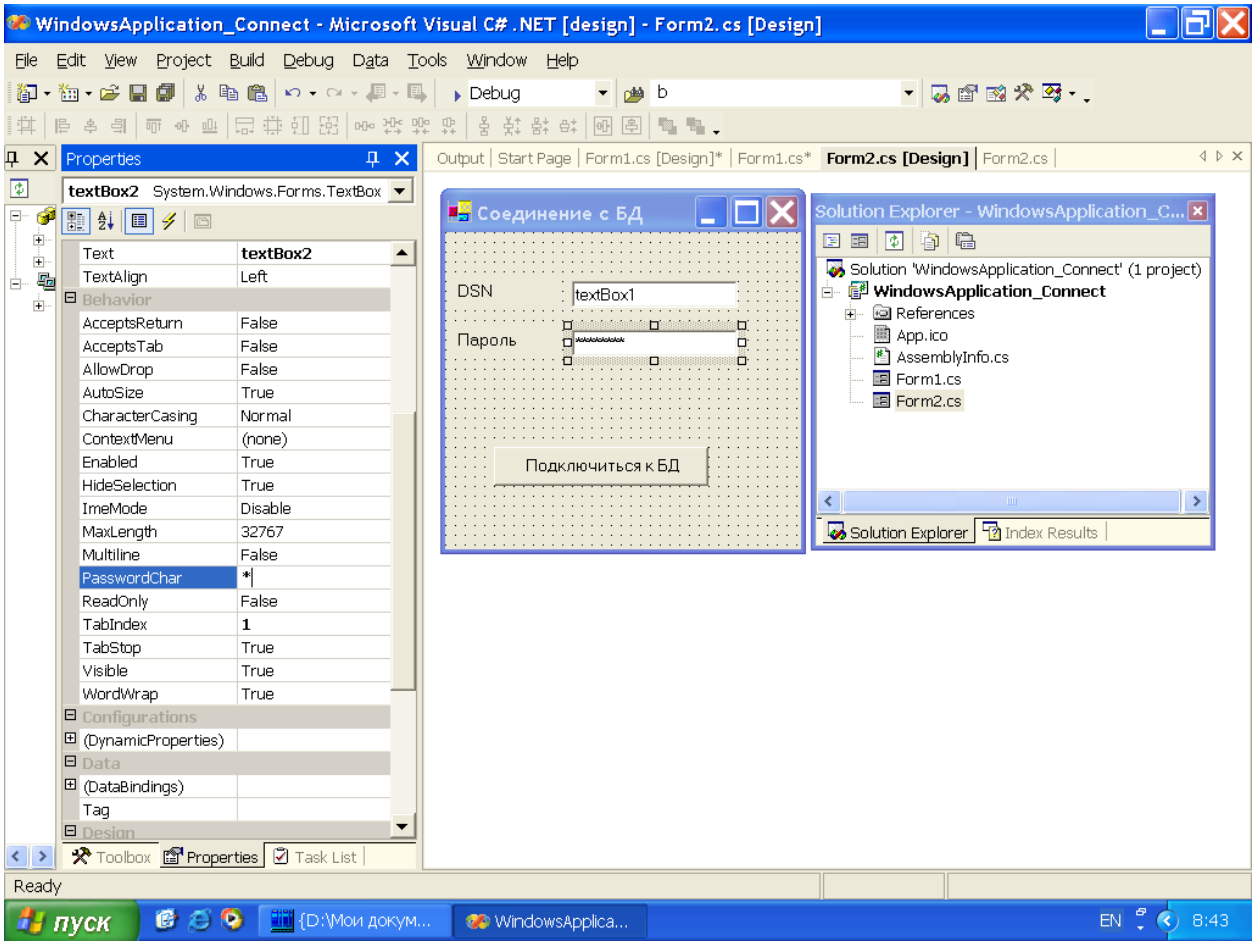

Рисунок 8.7 – Разработка формы аутентификации

В программный код внесем описание переменных для сохранения имени источника данных и пароля:

public string dsn,pw;

и напишем обработчик события нажатия кнопки (листинг 8.10):

Листинг 8.10.

```
private void button1_Click(object sender, System.EventArgs e)
        {
            dsn=this.textBox1.Text;
            pw=this.textBox2.Text;
            this.Dispose();
        }
```

Для передачи параметров соединения в существующую форму (Form1) внесем изменения:

Опишем переменные для аутентификации пользователя:

```
string v_dsn,v_pw;
```

В конструктор формы добавим параметры:

```
public Form1(string dsn,string pw)    {
                InitializeComponent(dsn,pw);
}
```

В блок инициализации также добавим параметры:

```
private void InitializeComponent(string dsn, string pw){
                v_dsn=dsn;
                v_pw=pw;
...
}
```

В методе Main сначала откроем форму для аутентификации пользователя, а затем введенные в эту форму параметры соединения с БД передадим в основную форму:

```
static void Main() {
                Form2 f2=new Form2();
                Application.Run(f2);
                Application.Run(new Form1(f2.dsn,f2.pw));

}
```

Параметры аутентификации используем при открытии формы для создания строки соединения с БД:

```
private void Form1_Load(object sender, System.EventArgs e){
            ...
            con1.ConnectionString= "PWD="+v_pw+";DSN="+v_dsn;
            ...
        }
```

Аутентификация пользователя в Web-приложении

Для ввода параметров соединения с БД в Web-приложении также создадим новую Web-форму в рамках существующего проекта и разместим метки, текстовые поля и кнопку. В обработчик события нажатия кнопку поместим программный код, представленный в листинге 8.11.

Листинг 8.11.

```
private void Button1_Click(object sender, System.EventArgs e)
    {
        Session["us"]=this.TextBox1.Text.ToString();
        Session["pw"]=this.TextBox2.Text.ToString();
        this.Response.Redirect("WebForm1.aspx");
    }
```

В Web-форме, в которой осуществляется работа с БД, прочитаем параметры, введенные пользователем, и используем их при соединении с БД:

Листинг 8.12.

```
private void Page_Load(object sender, System.EventArgs e)
    {
        string us=Session["us"].ToString();
        string pw=Session["pw"].ToString();
        try
        {
            this.sqlConnection1.ConnectionString = "user id="+us+
  ";data source=\"ITS-SERVER\";initial catalog=basa_user;password="+pw;
...
        }
    }
```

В листингах 8.11 и 8.12 используется объект Session. Данный объект реализует механизм поддержки сеансов работы пользователей. В частности, позволяет идентифицировать пользователя и отслеживать его работу в Web-страницей с момента входа. Часто информацию, введенную пользователем на одних Web-страницах, требуется использовать на других страницах. При этом заранее неизвестно, в какой последовательности пользователь будет открывать те или иные страницы. Информация, сохраненная в коллекции Session, постоянно сохраняется в элементах коллекции на протяжении всего сеанса работы пользователя с Web-страницей и доступна из программного кода любой страницы. Таким образом, элементы коллекции Session можно использовать как переменные, в которых сохраняется любая информа-

ция с одной Web-страницы, чтобы потом использовать ее на других страницах в течение всего сеанса работы пользователя. Данные, сохраненные в объекте Session, доступны только тому потоку, который и начал сеанс работы. То есть, при работе с объектом Session можно не бояться, что удаленные пользователи будут затирать данные друг друга.

После того, как параметры аутентификации пользователя сохранены в объекте Session, пользователя необходимо переадресовать на Web-страницу, которая предоставляет доступ к соответствующим информационным ресурсам. Для переадресации используется оператор:

this.Response.Redirect("WebForm1.aspx");

Посредством свойства *Response* объекта *Page* разработчик обращается к встроенному объекту *HttpResponse*, который выполняет пересылку информации браузеру удаленного пользователя. Метод *Redirect()* перенаправляет пользователя к другому ресурсу. В качестве параметра методу передается URL ресурса, который будет отправлен удаленному пользователю.

Задание на лабораторную работу

Создать Web-форму для аутентификации пользователя и Web-форму для вывода данных в таблицу по значению раскрывающегося списка. Реализовать функции модификации данных

Содержание отчета

В отчете указать цель работы, привести листинг разработанной программы и формы с результатами работы.

Контрольные вопросы

1. Как осуществляется привязка к данным в Web-приложениях?
2. Что такое ASP.NET?
3. Как реализуется форма для аутентификации?
4. Для чего нужен объект Session?

ЛАБОРАТОРНЫЙ ПРАКТИКУМ
по дисциплине
«КОРПОРАТИВНЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ»
Методические указания к выполнению лабораторных работ для студентов направления
230400.62 – «Информационные системы и технологии»

Составитель канд. техн. наук А.А. Евдокимов